An Investigation into the Viability of a Nonlinear Anisotropic Cloth Model for Simulation

Rachel Strohkorb
19 August 2019

MSc Computer Animation and Visual Effects

Bournemouth University

1 Introduction

Cloth simulation has been a long-standing problem in the field of computer graphics. Physically, fabric is a material defined by small interactions between tiny interwoven threads. Recreating the exact physical properties of cloth based on these interactions would be possible, given time, but for most applications the staggering amount of computations needed to reproduce a physically accurate fabric are not worth the time or effort.

Historically, the most popular way of approximating cloth for simulation has been the mass-spring method [12]. Instead of treating the cloth as a continuous object of interwoven threads, the cloth is divided into a series of finite elements or particles (typically the vertices of a mesh), each of which has a mass associated with it. These masses are then connected to each other by a network of springs, and the resulting mesh acts as a sheet of "cloth" that responds realistically to applied external forces.

However, while this method of modelling cloth is computationally efficient, it lacks the ability to recreate the actual physical properties of fabric and clothing, which can be easily detected by the human eye. Further, it is difficult to add user-end functionality to this model, thus making it difficult to have different samples of fabric created with the mass-spring method look different enough from each other to have a user or viewer believe that they are actually different samples of fabric.

In this paper, I investigate a different way of modelling cloth proposed by Volino and Magnenat-Thalmann [14] based on continuum mechanics. Typically, cloth simulation is divided into three areas – the cloth model itself, the integration method used for the physical simulation, and collision detection and response. I focus mostly on the applications of the cloth model, with some attention paid to integration and none given to collisions.

2 Previous Work

2.1 Mass-Spring

When it comes to modelling cloth, the mass-spring method is ubiquitous. As its main advantage is computational efficiency while retaining a physically accurate response to outside elements, there has been much effort put into making this model even more efficient. Most of the progress in this model's efficiency has taken place in the integration step, the most important being the work of Baraff and Witkin [1], who were the first to propose using the Backwards Euler method to integrate the cloth particles' velocity and positions implicitly. Explicit integrators estimate the velocity of each particle at the next time step based on the forces exerted on each particle by calculating the Taylor series of the velocity (typically up to 4 or 5 steps for the sake of better accuracy) [13]. Unfortunately, explicit integrators often don't remain stable unless the time step is very small, making them poor choices for real-time applications. Baraff and Witkin proposed an implicit method, wherein the partial derivatives of the forces on the mass points with respect to their positions are used to construct a linear system of equations, which than then be solved by iterative methods [1]. Iterating towards the solution has an associated cost, but because implicit integrators are able to use much larger time steps, this cost is outweighed.

Much effort has gone into making implicit integrators even more efficient. Kang and Cho [4] modified the implicit integration so that it could be completed piecewise, without each particle having to consider the velocity of every other particle at each time step. They also shortened the iterations to one iterative step, judging that this approximation was close enough to the actual solution, provided some damping forces were added. More recently, Liu et. al. [7] increased the speed of implicit solves for mass-spring systems by adding an additional local step that finds the optimal spring directions. By using this local step, they are able to separate their matrix used to solve the linear system of equations in the global step from being dependent on the current state, meaning that this matrix doesn't have to be computed every frame.

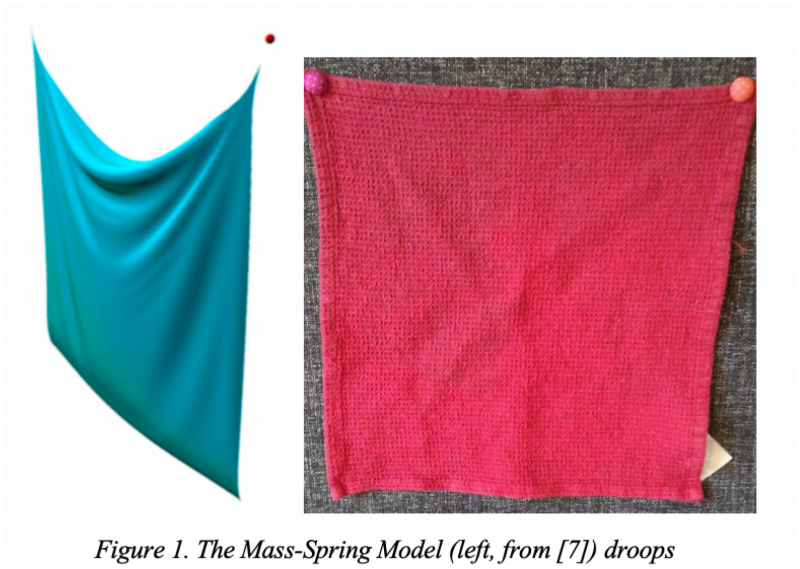
Figure 1. The Mass-Spring Model (left, from [7]) droops significantly under its own weight, while a cotton cloth swatch (right) barely stretches.

Despite being efficient, the mass-spring model still suffers from inaccuracies. One of these is that cloth in the real world does not droop as much under its own weight as cloth created by the mass-spring model. In fact, it often droops very little, to the point where the mass-spring model looks very unrealistic by comparison (*figure 1*). It would be possible to 'tighten up' a mass-spring system by increasing the spring constants of the structural springs, but increasing the spring constants drives up convergence time [2].

To combat this, Goldenthal et. al. [2] created a model for inextensible cloth, or cloth that does not stretch under its own weight. The cloth object is composed of mostly quads and some tris, and quad edges are constrained to be aligned to the weft and warp directions of the fabric. These edges are then constrained to maintain their original length. Shear forces are modeled with springs on both diagonals of each quad. Goldenthal et. al. then formulate an implicit integrator that's efficient at enforcing the edge length constraints using Constrained Lagrangian Mechanics. Their model utilizes the efficiency of the mass-spring model while making up for one of its greatest weaknesses.

2.2 Position-Based Dynamics

While the mass-spring model has proven to be quite efficient, this efficiency is often not efficient enough for virtual reality and games applications. One of the most common methods used to simulate cloth in modern gaming systems is position-based dynamics, a method proposed by Müller et. al. [9]. Most integrators will derive point positions and velocities from the forces exerted on the points. In position-based dynamics, this step is skipped, and the position of a point at the next time step is determined by a set of constraints. These constraints include maintaining physical realism, but also include collision detection, making handling collisions that much more efficient.

This model removes the time step issue of explicit solvers, meaning explicit integration methods, which are easier to comprehend and program, can be used without much associated cost. However, in the case of cloth, because the integration only works based on such as stretch and bending constraints it may be hard to represent particular types of cloth by controlling only these limited factors.

This model was adapted for use in APEX Clothing [10], Nvidia's cloth solver. It has been integrated into several game engines, such as Unreal 3 and 4, and has been used in several Batman Arkham games, as well as Bioshock Infinite.

2.3 Physically Accurate Model

On the other end of the spectrum from models optimized for efficiency, there are models designed to represent and simulate the actual physical behaviors and properties of cloth, where material behavior is defined by interactions between tiny threads. Sheet-based cloth models can work well when approximating fabric made by even interwoven threads running orthogonally to each other, but much of our clothing is knitted or woven. Knitted clothing, such as t-shirts, sweats, and much of our everyday casualwear, is especially difficult to simulate; as knitted fabrics are often composed of a single length of thread or yarn, there are thousands of self-collisions occurring at every time step.

Kaldor et. al. [3] propose a method to reduce yarn contact processing time by reusing data from previous contact steps, as locally, yarn-yarn contacts don't see much change over the course of a simulation. They further approximate the contact force between yarn strands as a corotational force, since yarn in knitted fabrics is typically looped around other strands. Their methods resulted in calculations taking place over four times faster than previous models, but per-frame calculations still took nearly a minute to complete. Impressively, their model exhibits behaviors unique to the particular stitch used to construct the item of clothing.

2.4 Continuum Mechanics

Volino et. al. [14] attempted to bridge the gap between the efficiency of the mass-spring method and the realism of physically accurate models by still treating the fabric object as a sheet, for the sake of efficiency, while modifying the way forces were calculated to more accurately emulate real-world fabric behavior. One of the main reasons the mass-spring method fails to accurately simulate how cloth behaves is because it assumes the forces acting internally are linear forces, and can thus be approximated with Hooke's Law. This is not true—for many fabrics, the internal forces are not at all proportional to the deformations that occur, and these deformations can vary based on the orientation of the fabric's weft and warp directions. Volino et. al. decided to base the mass point force calculations on force/displacement data taken from actual cloth samples. Their methods are the main subject of this paper and will be discussed in great detail in section three.

It is worth noting that their system of measurements used three tests from the

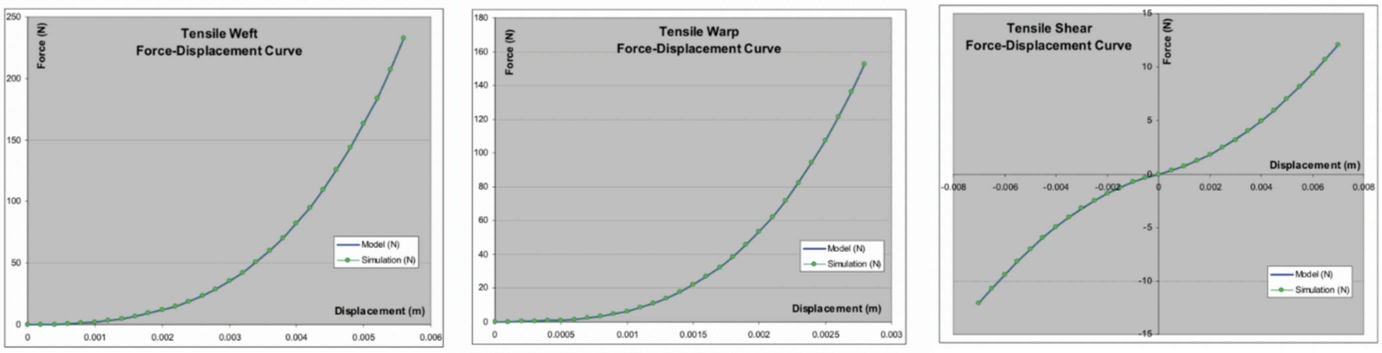

Figure 2. The data used in both [14] and my own project for the weft, warp, and shear modes of fabric. This force-displacement data was converted into strain-stress data using the appendix from [14]. Graphs also from [14].

Kawabata Evaluation System to measure their cloth: testing elongation in the weft, warp, and shear directions (shear being evaluated in the weft direction by pulling one end of the cloth in a direction parallel to its opposite end). It has been noted that this system of evaluation ignores many of the cross-dependencies that could exist between the three modes. Miguel et. al. [8] proposed an improved way of measuring these factors by capturing deformations in full 3D. Their tests do try to isolate the weft, warp, and shear factors, but these requirements are relaxed to get more realistic results. They also include bending tests that were left out of Volino et. al.'s model.

For my project, I was unable to produce hardware from either [14] or [8] in order to conduct my own fabric tests, so I used the data from [14] as the basis for my cloth object (*figure 2*).

3 Methods

3.1 Cloth model

The cloth model for this project assumes a mesh object composed of triangles for which no more than two triangles share an edge. The vertices of the triangles are treated as mass points and are given a mass equal to one-third of the object's total mass times the surface area of an attached triangle, summed over each triangle attached to the vertex. These triangles need not be arranged in a grid pattern, and although similarly sized triangles are preferred, it isn't necessary for the model to work correctly.

In the starting state, the cloth is assumed have its weft-warp directions orthogonally aligned; that is, the cloth is not experiencing any internal forces, so there is no strain on the fabric internally that would cause the weft-warp vectors to deform from their starting orthogonal state. From here on, the vector per triangle in the weft direction will be called ***u***, and in the warp direction ***v***.

Each of the three vertices of a triangle are mapped from 3D world coordinates to 2D parametric coordinates. These parametric coordinates must produce the same surface area as the 3D world coordinates in order to be considered valid. The easiest mapping from 3D to 2D has the cloth object lie flat on one of the x, y, or z-axes, but my implementation allows for that not to be the case. The goal of these 2D coordinates is to be able to calculate the deformation state of the triangle, or the change in the magnitude and direction of ***u*** and ***v***. By assuming ***u*** = [1, 0] and ***v*** = [0, 1] in the initial state, weights for each of the vertices of the triangle (a, b, c) can be calculated as follows:

$$r_{ua} = \frac{b_y - c_y}{d} \qquad r_{va} = \frac{c_x - b_x}{d}$$
$$r_{ub} = \frac{c_y - a_y}{d} \qquad r_{vb} = \frac{a_x - c_x}{d}$$
$$r_{uc} = \frac{a_y - b_y}{d} \qquad r_{vc} = \frac{b_x - a_x}{d}$$

Where *d* is two times the surface area of the triangle in the initial state. During the simulation, these weights can be used to calculate **u** and **v** as follows:

$$\boldsymbol{u} = \sum_{i \in (a,b,c)} r_{ui} P_i \qquad \boldsymbol{v} = \sum_{i \in (a,b,c)} r_{vi} P_i$$

Where P_i is the location of vertices (a, b, c) in 3D world coordinates.

This model is based on the idea that the strain state of the cloth object can be completely derived from the point positions of the triangle vertices. All of the internal forces produced by this model are derived from the magnitude of ***u*** and ***v***, as well as the angle between these vectors for the shear forces.

The material behavior of the cloth is defined by the relationship between the strain on the cloth and the stress (pressure) that results from the strain. This relationship is derived from the three force-displacement tests done for each of the weft, warp, and shear directions (also referenced as *uu*, *vv*, and *uv* directions). The force-displacement data is converted into strain-stress data using the appendix in [13]. A b-spline curve is then interpolated between the data points and used as a function defining the strain-stress relationship of the material.

The strain state of a given triangle can be derived from $\boldsymbol{u}$ and $\boldsymbol{v}$ as follows:

$$\varepsilon_{uu} = \frac{1}{2}(\boldsymbol{u}^t\boldsymbol{u} - 1)$$

$$\varepsilon_{vv} = \frac{1}{2}(\boldsymbol{v}^t\boldsymbol{v} - 1)$$

$$\varepsilon_{uv} = \frac{1}{2}(\boldsymbol{u}^t\boldsymbol{v} + \boldsymbol{v}^t\boldsymbol{u})$$

By plugging these strain values into their associated strain-stress curves, one acquires the stress state values σ_{uu}, σ_{vv}, and σ_{uv}. The stress is then used to calculate the internal forces created by the distortions in this triangle.

3.2 Force calculations

For each triangle, forces are computed for each of the three vertices (a, b, c). For any *j* in (a, b, c), the force on a point from this triangle is as follows:

$$\boldsymbol{F}_j = -s\left(\sigma_{uu}r_{uj}\boldsymbol{u} + \sigma_{vv}r_{vj}\boldsymbol{v} + \sigma_{uv}\left(r_{uj}\boldsymbol{v} + r_{vj}\boldsymbol{u}\right)\right)$$

Where *s* is the surface area of the triangle. Accumulating these forces for each triangle in the mesh accounts for all the internal forces in this cloth model. Forces can then be integrated with techniques from mass-spring systems, as those integrators also integrate over forces on individual mass points.

This force equation is enough information for explicit integration, but computing the Jacobian of the forces, or the partial derivatives of the forces on the mass points with respect to their positions, is necessary for implicit integration. For each triangle, and for any *i* and *j* among (a, b, c), the local Jacobian matrix can be computed as follows:

$$\begin{aligned}\boldsymbol{J}_{ji} = \frac{\partial \boldsymbol{F}_j}{\partial P_i} = -s\Bigg(&\frac{\partial\sigma_{uu}}{\partial\varepsilon_{uu}}r_{uj}r_{ui}\boldsymbol{u}\boldsymbol{u}^t \\ &+ \frac{\partial\sigma_{vv}}{\partial\varepsilon_{vv}}r_{vj}r_{vi}\boldsymbol{v}\boldsymbol{v}^t \\ &+ \frac{\partial\sigma_{uv}}{\partial\varepsilon_{uv}}\left(r_{uj}r_{vi}\boldsymbol{u}\boldsymbol{v}^t \right.\\ &\left.+ r_{vj}r_{ui}\boldsymbol{v}\boldsymbol{u}^t\right) \\ &+ \Big(\sigma_{uu}r_{uj}r_{ui} + \sigma_{vv}r_{vj}r_{vi} \\ &+ \sigma_{uv}\left(r_{uj}r_{vi} + r_{vj}r_{ui}\right)\Big)\boldsymbol{I}\Bigg)\end{aligned}$$

The Jacobian consists of a stiffness component, the component created by the terms multiplied by strain-stress derivates, and a geometric component, the component created by the terms multiplied by the stress σ. For my implementation, the partial derivatives of stress related to strain are computed as the derivatives of the associated curve (*uu, vv, uv*). They are included as partial derivatives because stress may depend on more than one factor of ε, the strain in the (*uu, vv, uv*) directions, or ε', the change in strain in those directions. However, based on the measurements I used in this project, no cross-dependencies were included in the model.

The local Jacobian is a 3x3 matrix composed of nine 3x3 block matrices and appears as follows:

$$\boldsymbol{J}_{local} = \begin{bmatrix}\boldsymbol{J}_{aa} & \boldsymbol{J}_{ab} & \boldsymbol{J}_{ac}\\ \boldsymbol{J}_{ba} & \boldsymbol{J}_{bb} & \boldsymbol{J}_{bc}\\ \boldsymbol{J}_{ca} & \boldsymbol{J}_{cb} & \boldsymbol{J}_{cc}\end{bmatrix}$$

The global J-matrix is an *n*x*n* matrix composed of 3x3 block matrices $\boldsymbol{J}_{kl}$, where *n* is the number of mass points in the cloth, *k* is the entry of the mass point being acted upon, and *l* is the entry of the mass point whose position is exerting a force on point *k*.

Because $\boldsymbol{J}_{local}$ is created for each triangle, there are potentially multiple contributions to each $\boldsymbol{J}_{kl}$. For instance, in the case where $k = l$, $\boldsymbol{J}_{kk}$ has a contribution from every triangle k is a part of. When k is not equal to l, $\boldsymbol{J}_{kl}$ has a contribution from every triangle where k and l share an edge, which in the case of this model is up to two triangles.

To construct the global J-matrix, the mass point values of the triangle vertices (a, b, c) are noted. Each local $\boldsymbol{J}_{aa}$ is added to the total value accumulated for the corresponding $\boldsymbol{J}_{kk}$, and each $\boldsymbol{J}_{ab}$ is added to the total value accumulated for the corresponding $\boldsymbol{J}_{kl}$. This is repeated for each of the matrices in $\boldsymbol{J}_{local}$, and accumulated over each triangle in the model, until the total global J-matrix has been calculated. It can then be used in implicit integration schemes.

Volino et. al. also provide the necessary math to compute the Jacobian matrix of the forces with respect to the point velocities. Unfortunately, in order to compute this Jacobian, I would need access to data that defined the relationship between σ and ε', or the relationship between stress and the change in strain. This relationship was difficult to measure with the Kawabata Evaluation System, so Volino et. al. did not use it in their implementation. With the measuring equipment in [8], it would likely be easier to track this relationship and the velocity Jacobian would then be able to be computed.

3.3 Integration

For this project, I implemented both RK4 explicit integration [5, 11] and Baraff and Witkin's modified Conjugate Gradient Method for implicit integration [1]. RK4 involves recalculating the internal triangle forces for each step in the Taylor series (in this case, 4 steps) based on the state created by the previous step. Because of the small time step sizes needed to keep RK4 stable and having to recalculate all internal forces four times per step, it performs quite slowly.

Baraff and Witkin [1] show that the equation to solve for the change in velocity for this time step can be rephrased in the context of a linear system $\boldsymbol{Ax} = \boldsymbol{b}$, where:

$$\boldsymbol{A} = \left(\boldsymbol{M} - h\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{v}} - h^2\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right)$$

$$\boldsymbol{b} = h\boldsymbol{f}_0 + h^2\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\boldsymbol{v}_0$$

$$\boldsymbol{x} = \Delta\boldsymbol{v}$$

Where h is the time step, $\boldsymbol{f}_0$ and $\boldsymbol{v}_0$ are the initial force and velocity vectors, and $\mathbf{M}$ is an nxn matrix composed of 3x3 block matrices, whose diagonal values $\mathbf{M}_{kk}$ are equivalent to the mass of mass point k. These values for $\boldsymbol{A}$ and $\boldsymbol{b}$ are able to be plugged into the conjugate gradient method as normal in order to solve for $\boldsymbol{x}$, the change in velocity at this time step for each mass point.

Baraff and Witkin further modify the conjugate gradient method to account for instances where the points in the cloth might become fixed in any given direction during the simulation, such as when a collision or self-collision is detected. For my implementation, I added this mass point filtering process in order to fix points in space to hang and drape the cloth, but it could easily be modified to help the cloth respond to collisions properly.

In order to filter the mass point forces and velocities, Baraff and Witkin construct a filtering matrix $\mathbf{S}$ of the same size of $\mathbf{M}$ and $\mathbf{J}$ such that the diagonal 3x3 block matrix $\mathbf{S}_{ii}$ is as in (*figure 3*).

$$\boldsymbol{S}_{ii} = \begin{cases} \boldsymbol{I} & \text{if point } i \text{ is free in all 3 directions} \\ (\boldsymbol{I} - \boldsymbol{p}_i\boldsymbol{p}_i^t) & \text{if point } i \text{ is free in 2 directions and not in } \boldsymbol{p} \\ (\boldsymbol{I} - \boldsymbol{p}_i\boldsymbol{p}_i^t - \boldsymbol{q}_i\boldsymbol{q}_i^t) & \text{if point } i \text{ is free in 1 direction and not in } \boldsymbol{p} \text{ or } \boldsymbol{q} \\ 0 & \text{if point } i \text{ has no degrees of freedom} \end{cases}$$

Figure 3. The S-Matrix used to filter for fixed points in the modified conjugate gradient method [1].

Applying this filtering matrix to certain steps in the conjugate gradient method allows the affected points to stay fixed without disrupting the forces of the connecting points. For my purposes, I was only concerned with cases where the point *i* had either 3 degrees of freedom or none. For the initial guess of $\Delta\boldsymbol{v}$, fixed points would acquire the change in velocity that was given in this opening guess.

Baraff and Witkin also employed a preconditioning matrix **P**, whose diagonal values are equal to the inverse of the diagonal values of **A**. The filtering operation involved multiplying the input vector by the filtering matrix **S**. With that in mind, this is the modified conjugate gradient method algorithm:

```
$\Delta\boldsymbol{v} \leftarrow$ initial guess, values for fixed points
$\delta_{test} \leftarrow \text{filter}(\mathbf{b})^T \mathbf{P}\ \text{filter}(\mathbf{b})$
$\mathbf{r} \leftarrow \text{filter}(\mathbf{b} - \mathbf{A}\Delta\boldsymbol{v})$
$\mathbf{q} \leftarrow \text{filter}(\mathbf{P}^{-1}\ \mathbf{r})$
$\delta_{new} \leftarrow \mathbf{r}^T\ \mathbf{c}$
$\gamma \leftarrow$ small test value, such as 1e-5
while $\delta_{new} > \gamma^2\ \delta_{test}$
        $\alpha \leftarrow \frac{\delta_{new}}{\mathbf{q}^t\ \text{filter}(\mathbf{Aq})}$
        $\Delta\boldsymbol{v} \leftarrow \Delta\boldsymbol{v} + \alpha\mathbf{q}$
        $\mathbf{r} \leftarrow \mathbf{r} - \alpha\ \text{filter}(\mathbf{Aq})$
        $\mathbf{z} \leftarrow \mathbf{P}^{-1}\ \mathbf{r}$
        $\delta_{old} \leftarrow \delta_{new}$
        $\delta_{new} \leftarrow \mathbf{r}^T\ \mathbf{z}$
        $\mathbf{q} \leftarrow \text{filter}(\mathbf{z} + \frac{\delta_{new}}{\delta_{old}}\ \mathbf{q})$
```

By comparing δ_{new} to the test value created by filtering **b** and inter-multiplying the preconditioning matrix, the algorithm can converge even when there are values in $\Delta\boldsymbol{v}$ that will not change from their starting values, due to being fixed points.

For my implementation, this was not enough to maintain stability of the simulation. I judged that because I'd shortened the original **A** by removing the $\frac{\partial f}{\partial v}$ term (the Jacobian matrix with respect to velocity) that there were not any viscous forces represented to damp the system. I decided to introduce the damping system used in [4], as it was simple to implement, and I'd had experience with it before. This involved the addition of a damping coefficient to both sides of the equation, such that:

$$\boldsymbol{A} = \left(\boldsymbol{M} - h^2\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right) + \ (n_i h\ c_d \boldsymbol{I})$$

$$\boldsymbol{b} = h\boldsymbol{f}_0 + \ h^2\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} + h\ c_d\boldsymbol{I}\right)\boldsymbol{v}_0$$

Where c_d is the damping coefficient, and n_i is the number of mass points connected to point *i*. This method worked to damp the system and make the simulation stable, but at the cost of increased convergence time in the conjugate gradient algorithm. I was unable to investigate different methods of damping, but for future work it would be good to either use the Jacobian matrix with respect to velocity, or use a more efficient means of damping the system for stability.

4 Results

4.1 Against Mass-Spring

The method presented by Volino et. al. certainly looks more realistic to the human eye than the mass spring method (*figure 4*). Without introducing constraints on the edge

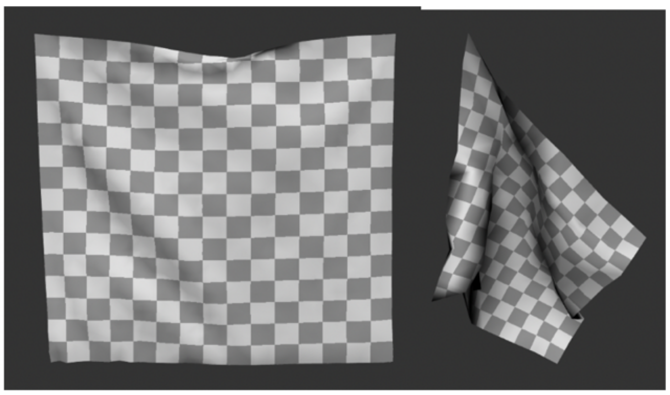
Figure 4. GNATV cloth hangs from top corners (left) and left corners, as a flag (right).

lengths as in [2], it manages not to sink under its own weight as the mass-spring model does, and it even folds over quite well, despite not having any bending forces included in the model.

Volino et. al. proved in their work that their model was able to correctly reproduce the data fed into it when the same tests were run in the simulation [14]. Miguel et al. showed similar results with their model [8], which incorporated more cross-dependencies between the weft, warp, and shear modes of fabric in their measurements. Because I was unable to measure data myself for this project, in either of the ways presented in [8, 14], I did not complete these tests myself, as the data for this model was the same as used in [14].

In my view, that is one of the main downsides of this model—measuring from real cloth samples does give this model a certain credibility not present in other sheet-based cloth models, but acquiring said equipment or building it yourself can be prohibitively difficult. If there were some sort of online database containing raw force-displacement graph data from a multitude of fabric samples, that would make using this model in commercial practice much more viable. Unfortunately, the only data readily available are figures that summarize the graph created, such as its linearity of the sample or the area under the created curve. This data, while useful for fashion designers, is not enough to recreate the actual data points necessary for this model.

FORCE	ADD	MULT	DIV	SQT
MASS SPRING	10	7	3	1
GNATV	42	51	0	0

Figure 5. Number of calculations needed to compute the force vector for each of MS and GNATV. The number of force calculations is equal to the number of springs in MS and 3 times the number of triangles in GNATV.

JACOB.	ADD	MULT	DIV	MTR (3X3)
MASS SPRING	10	18	11	1
GNATV	39	88	0	0

Figure 6. Number of calculations needed to compute the Jacobian matrix with respect to position for each of MS and GNATV. The number of force calculations is equal to the number of springs in plus the number of mass points MS and 9 times the number of triangles in GNATV.

Another downside of this model is that it is still not as efficient as the mass-spring method. It is difficult to track performance between the two for several reasons. Firstly, both examples that I have were created by myself, so they're subject to errors in efficiency. Further, the mass-spring model I made was based on [4], which makes several approximations in order to speed up the simulation, so it is not the most accurate comparison point.

I decided for the comparison to make a note of the number of mathematical operations necessary to calculate the forces and Jacobians with respect to position per-frame, assuming both need to be updated dynamically, and assuming since both can use similar methods, they will have similar performances when integrated. These costs are per-calculation, and it is pretty easy to see that the model put forward by Volino et. al. (hereafter called GNATV for 'General Nonlinear Anisotropic Tensile Viscoelasticity') is much more costly than the mass-spring method (*figures 5, 6*).

It is worth noting that these costs do not include the operations necessary to compute the stress (for the forces) or the stress partial derivative (for the Jacobians). Depending on the implementation, this may be computed any number of ways. If done by projecting a function onto the data, it may only add a few multiplications and additions on top of the load present. However, in my implementation as in Volino et. al.'s, a b-spline interpolator was used to connect the space between the data points, as cloth data is not always guaranteed to follow an easily expressible function. This adds a fair bit onto the associated costs of the model per-frame.

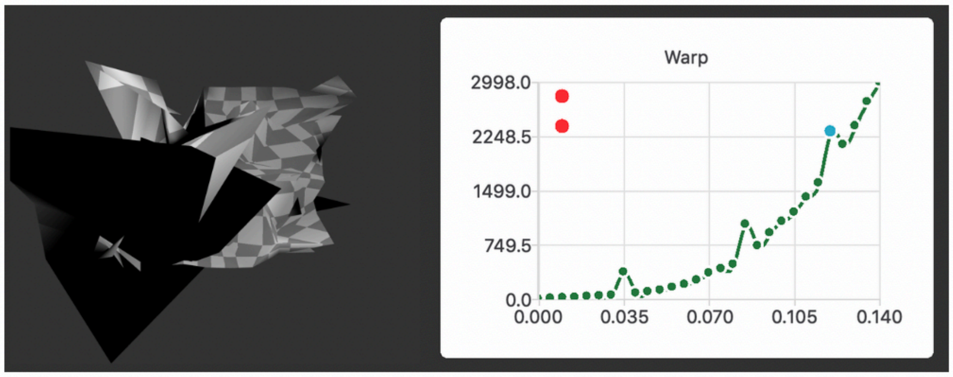

Figure 7. Introducing unrealistic irregularities in the b-spline curve relating stress to strain (right) will often result in the failure of the cloth object (left).

Other efficiency costs shown in my implementation may only be specific to my work; as previously noted, I needed to add damping forces to maintain stability in the model, which resulted in much higher convergence times. It would be worth investigating which damping forces, if any, are needed for this model to work and how efficient each method is.

4.2 Adjusting Data Points

Since I was unable to acquire data outside from that used in [14], I decided to expose the base b-spline data to the user and do some light experiments on changing the data the model is based on. I discovered that changing things willy-nilly will often lead to the cloth tearing itself apart (*figure 7*), so it seems some adherence to realism is necessary in order to maintain stability. I had the most success in manipulating the step in the strain direction and increasing the strain maximum, moving data points up to follow in as realistic of a manner as could be managed given the interface (*figures 8 - 11*).

Interestingly, increasing the step size for the warp direction by orders of ten greatly decreased convergence times. This suggests that this model might suffer from a similar problem as the mass spring – increasing stiffness too much may prohibitively increase convergence. For my purposes, it is difficult to separate my convergence times from other factors, so it is difficult to draw any conclusions from this result.

5 Conclusion

The model presented by Volino et. al. does produce more realistic results than the mass-spring method, but it is more expensive computationally. However, their setup, wherein all the forces are derived strictly from the positions of the mass points, would allow this algorithm to be fully evaluated on the GPU, which would greatly help speed and efficiency. In my opinion, for this model to be successful, better cloth measurements need to be taken, with cross-dependencies between the modes of cloth taken into consideration. For this model to be used generally, it may need a few speed-ups and approximations, as has been done with the mass-spring model. It would also require data from cloth testing experiments to be generally available so that each person wanting to implement this model wouldn't have to measure the data themselves.

Looking forward in cloth simulation, this will probably not be the model that people shift to when looking for a combination of efficiency and realism. As with most fields in computer science, research done with deep learning networks far outperforms any other work.

Lähner et. al. recently published a very impressive work in which neural networks were used to map a clothing item onto a model, and from there create a normal map while the model animated that realistically recreated highly detailed wrinkle patterns on the item of clothing [6]. The network does need to be retrained for different fabric materials, and as with all deep learning algorithms, the main issue with the model is acquiring and labelling data to train the network. However, once that work has been completed, a neural network can be used very efficiently in real time. If someone were to put in the work to make this into a viable product, I have no doubt it would be very successful and quickly become the most popular method of adding realism to cloth simulation.

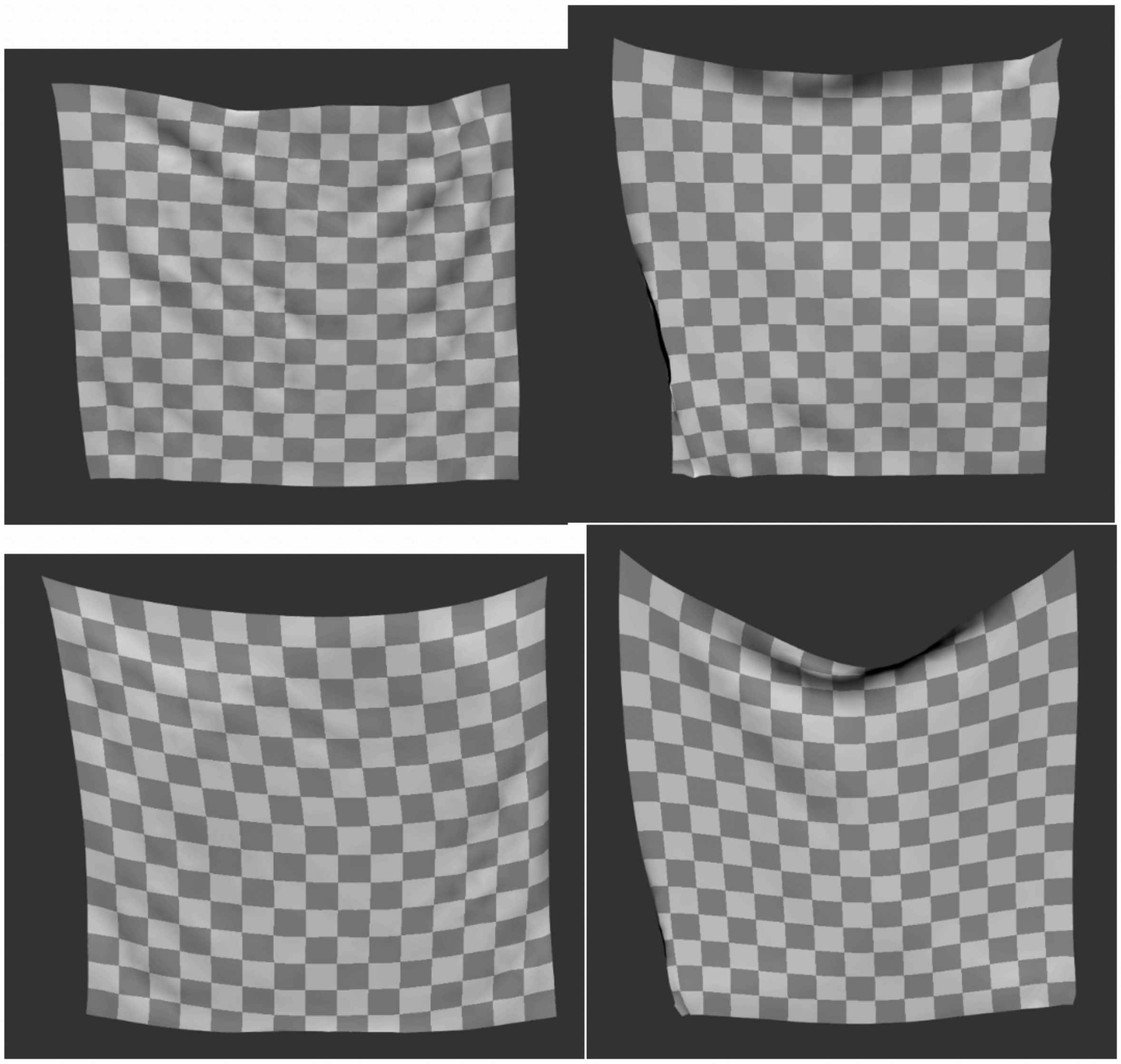

Figure 8. Increasing the step size in the strain direction leads to interesting results. Top left: weft step of 0.04 instead of 0.004. Top right: warp step of 0.1 instead of 0.0005. Bottom left: shear step of 0.1 instead of 0.01. Bottom right: all three previous steps applied.
Increasing the strain step size seems to correspond with 'loosening up' the cloth object, making it less stiff.

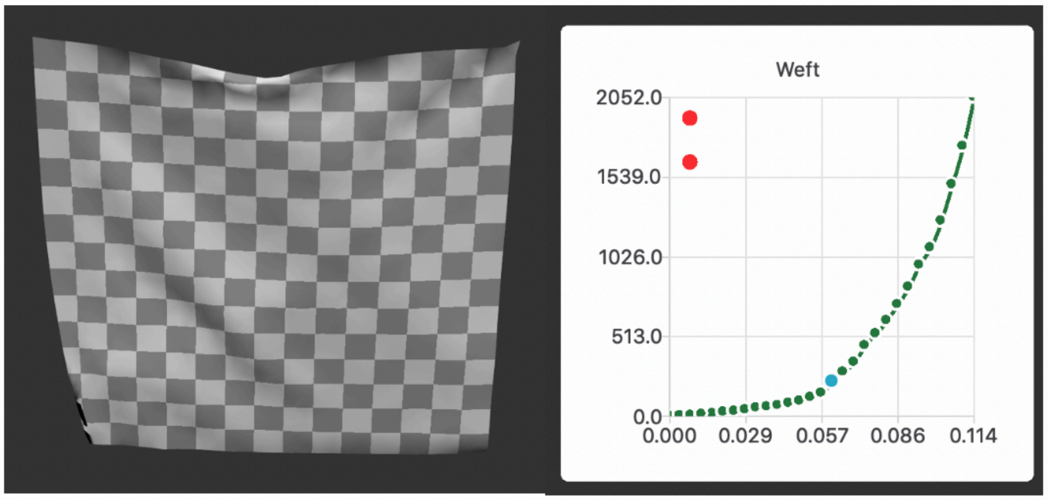

Figure 9. Increasing the maximum weft from 1052 to 2052 and adjusting the data points accordingly, with the addition of a linear trend between 0.057 and 0.1.

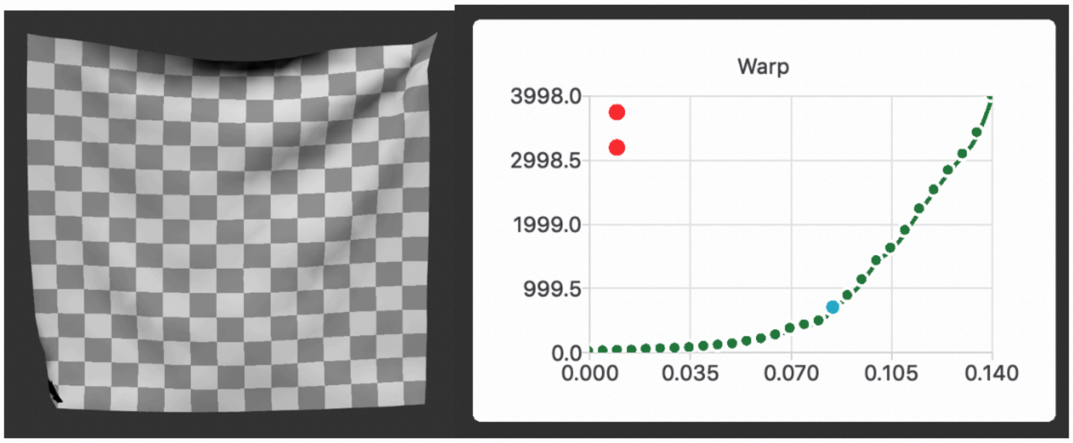

Figure 10. Increasing the maximum warp from 2998 to 3998 and adjusting the data points accordingly, with the addition of a linear trend between 0.08 and 0.13. Step size has been adjusted to 0.005 to speed convergence.

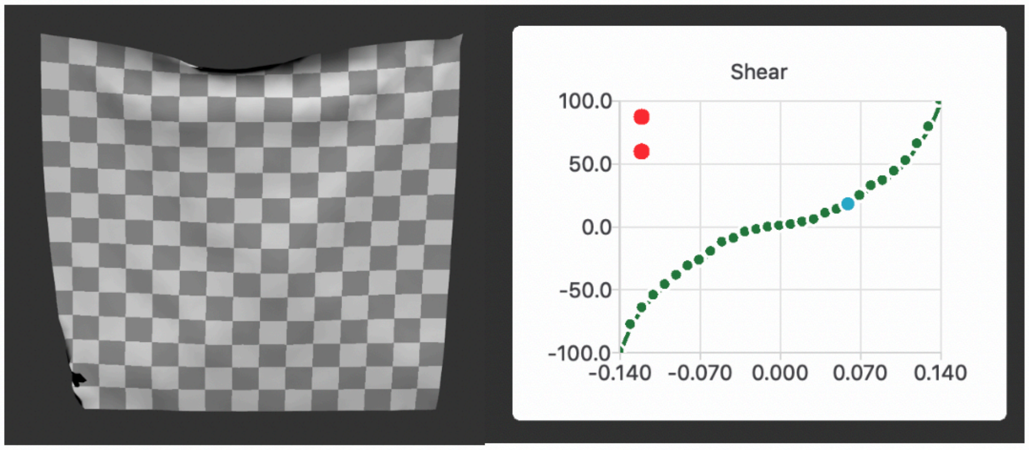

Figure 11. Increasing the maximum and minimum shear from 60 to 100 and adjusting the data points accordingly.

References

[1] D. Baraff, A.Witkin, 1998, Large Steps in Cloth Simulation. *Computer Graphics (SIGGRAPH'98 proceedings)*, ACM Press, pp 43-54.

[2] R. Goldenthal, D. Harmon, R. Fattal, M. Bercovier, E. Grinspun, 2007, Efficient Simulation of Inextensible Cloth. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3).

[3] J.M. Kaldor, D.L. James, S. Marschner, 2010, Efficient Yarn-Based Cloth with Adaptive Contact Linearization. *ACM Transactions on Graphics,* 29(4).

[4] Kang, Y. M., and Cho, H. G., 2002, Complex deformable objects in virtual reality. *VRST '02 Proceedings of the ACM symposium on Virtual reality software and technology*, ACM, 49-56.

[5] M. Kutta, 1901, *Beitrag zur näherungweisen Integration totaler Differentialgleichungen*.

[6] Z. Lähner, D. Cremers, T. Tung, 2018, Deep Wrinkles: Accurate and Realistic Clothing Modeling. *CoRR*, 1808.

[7] T. Liu, A.W. Bargteil, J.F. O'Brien, L. Kavan, 2013, Fast Simulation of Mass-Spring Systems. *ACM Trans. Graph.*, 32.

[8] E. Miguel, D. Bradley, B. Thomaszewski, B. Bickel, W. Matusik, M.A. Otaduy, S. Marschner, 2012, Data-Driven Estimation of Cloth Simulation Models. *Computer Graphics Forum,* 31(2, 2), 519 – 528.

[9] M. Müller, B. Heidelberger, M. Hennix, J. Ratcliff, 2007, Position Based Dynamics. *J. Vis. Comun. Image Represent,* 18(2), 109–118.

[10] PhysXInfo, 2012. APEX Clothing. *PhysXWiki,* last modified April 2016. http://physxinfo.com/wiki/APEX_Clothing

[11] C.D.T. Runge, 1895, Über die numerische Auflösung von Differenntialgleichungen. *Mathematische Annalen*, Springer, 46(2), 167-178.

[12] M. Seymour, 2018, Cloth Simulation, Opening the Kimono. *Fxguide.com*, last modified December 2018. https://www.fxguide.com/fxfeatured/cloth-simulation-opening-the-kimono/

[13] T. Stuyck, 2018, *Cloth Simulation for Computer Graphics*, Morgan & Claypool Publishers.

[14] P. Volino, N. Magnenat-Thalmann, F. Faure, 2009, A Simple Approach to Nonlinear Tensile Stiffness for Accurate Cloth Simulation. *ACM Transactions on Graphics, Association for Computing Machinery*, 28(4).

Implimentation can be found on my github: https://github.com/rstrohkorb/GNATV_cloth_sim