Chapter 3: Differentiable Simulation for Material Optimization

3.1 Introduction

Given the body estimation results in Chapter 2, the main task for building a virtual try-on system is to dress the given body with virtual garments. However, creating a cloth mesh that faithfully represents a real garment in a digital system is not easy. First, the geometry of the mesh needs to be designed well to fit the body. Second, the fabric materials of the designed garment are even more challenging to model correctly. While the geometry of the garment can still be intuitively hand crafted by experienced artists or designers, it is difficult to determine the fabric materials that exhibit mechanical properties closely resembling the real ones. There are two possible approaches to achieve this material cloning/estimation task. One way is to use supervised machine learning on manually labelled material data, which is labor-intensive and error-prone. The other method makes use of the geometric similarity instead of the direct material label, but needs a module that can compute the analytic relationship between the resulting geometry and the estimations or guesses of the fabric materials. Differentiable physics that can derive gradients regarding the simulation inputs can offer such a capability. With the differentiable simulation, one can either use an optimizer or train a network model to predict the materials from the input, and compute its geometric appearance analytically. After comparing with the target shapes, a loss can then be created and back-propagate to the network, guiding the learning in a more intuitive way.

In this chapter, I propose a differentiable cloth simulation algorithm. First, I use dynamic collision detection since the actual collision pairs are very sparse. The collision response is solved by quadratic optimization, for which I can use implicit differentiation to compute the gradient. I directly solve the equations derived from implicit differentiation by using the QR decomposition of the constraint matrix, which is much smaller than the original linear system and is often of low rank. This approach reduces the gradient computation to a linear system of a small upper triangular matrix (the R component of the decomposition), which enables fast simulation and backpropagation.

My experiments indicate that the presented method makes differentiable cloth simulation practical. Using my method, the largest size of the linear system is 10x-20x smaller than the original solver in the backpropagation of the collision response, and the solver is 60x-130x faster. I demonstrate the potential of differentiable cloth simulation in a number of application scenarios, such as physical parameter estimation and motion control of cloth. With only a few samples, the differentiable simulator can optimize its input variables to fit the data, thereby inferring physical parameters from observations and reaching desired control goals.

3.2 Related Work

Differentiable physics. With recent advances in deep learning, there has been increasing interest in differentiable physics simulation, which can be combined with other learning methods to provide physically consistent predictions. Belbute-Peres *et al*. [13] and Degrave *et al*. [14] proposed rigid body simulators using a static formulation of the linear complementarity problem (LCP) [82, 83]. Toussaint *et al*. [15] developed a robot reasoning system that can achieve user-defined tasks and is based on differentiable primitives. Hu *et al*. [16] implemented a differentiable simulator for soft robots based on the Material Point Method (MPM). They store the object data at every simulation step so that the gradient can be computed out of the box. Schenck and Fox [17] embedded particle-based fluid dynamics into convolutional neural networks, with precomputed signed distance functions for collision handling. They solved or avoided collisions by assuming special object shapes, transferring to an Eulerian grid, or solving the corresponding collision constraint equation.

None of these methods can be applied to cloth simulation. First, cloth is a 2D surface in a 3D world; thus methods that use an Eulerian grid to compute material density, such as MPM [16], are not applicable. Second, the collision constraints in cloth simulation are more dynamic and complex given the high number of degrees of freedom; thus constructing a static dense LCP for the entire system [13, 14] or constructing the overall state transition graph [15] is inefficient and usually impossible for cloth of common resolution, since contact can happen for every edge-edge or vertex-face pair. Lastly, the shape of cloth changes constantly so self-collision cannot be handled by precomputed signed distance functions [17].

In contrast, my method uses dynamic collision detection and computes the gradients of the collision response by performing implicit differentiation on the quadratic optimization used for computing the response. I utilize the low dimensionality and rank of the constraint matrix in the quadratic optimization and minimize the computation needed for the gradient propagation by giving an explicit solution to the linear system using QR decomposition of the constraint matrix.

Deep learning and physics. Supervised deep networks have been used to approximate physical dynamics. Mrowca *et al*. [84] and Li *et al*. [85] learned interaction networks to model particle systems. Ingraham *et al*. [86] trained a model to predict protein structures from sequences using a learnable simulator; the simulator predicts the deformation energy as an approximation to the physical process. Deep networks have also been used to support the simulation of fluid dynamics [87, 88, 89]. My method differs from many works that use deep networks to approximate physical systems in that I backpropagate through the true physical simulation. Thus my method conforms to physical law regardless of the scale of the problem. It can also naturally accept physical parameters as input, which enables learning from data.

Deep learning and cloth. Coupling cloth simulation with deep learning has become a popular way to solve problems such as detail refinement, garment retargeting, and material estimation. Yang *et al*. [21] proposed a recurrent model to estimate physical cloth parameters from video. Lähner *et al*. [23] trained a GAN to generate wrinkles on a coarse garment mesh which can then be automatically registered to a human body using PCA. Gundogdu *et al.* [8] trained a graph convolutional framework to generate drapes and wrinkles given a roughly registered mesh. Santesteban *et al.* [90] developed an end-to-end retargeting network using a parametric human body model with displacements to represent the cloth.

These applications may benefit from my method. For garment retargeting problems, the relationship between body pose and vertex displacement is made explicit via the computed gradient, which can then be applied in network regularization for better performance. For parameter estimation, the differentiable simulation provides an optimization-based solution rather than a learning-based one. Instead of learning statistics from a large amount of data, I can directly apply gradient-based optimization via the simulator, which does not require any training data.

3.3 Differentiable Cloth Simulation

In this section, I introduce the main algorithms for the gradient computation. In general, I follow the computation flow of the common approach to cloth simulation: discretization using the finite element method [91], integration using implicit Euler [27], and collision response on impact zones [74, 92]. I use implicit differentiation in the linear solve and the optimization in order to compute the gradient with respect to the input parameters. The discontinuity introduced by the collision response is negligible because the discontinuous states constitute a zero-measure set. During the backpropagation in the optimization, the gradient values can be directly computed after QR decomposition of the constraint matrix.

3.3.1 Cloth Simulation Basics

Generally, cloth simulation includes three steps: force computation, dynamic solve, and collision handling. Extra steps, such as plasticity handling and strain limiting, are omitted since they are not essential components of a basic cloth simulation.

3.3.1.1 Force Computation

For external forces, the most common ones are gravity and wind forces, which are both straightforward. I focus on internal, constraint and frictional forces here.

Clothes are usually modeled as a 2D manifold mesh in 3D space. I apply Finite Element Method (FEM) to compute internal forces. For each triangle face in the mesh, I compute the deformation gradient as a variable of the strain:

$$\mathbf{F} = \frac{\partial \mathbf{x}}{\partial \mathbf{X}} \tag{3.1}$$

Here, $\mathbf{x}$ is the current 3D position of the triangle, and $\mathbf{X}$ is their coordinate in the 2D material space. Then, the stress (or internal forces) is computed using the deformation gradient $\mathbf{F}$. Usually a strain energy $\mathbf{E}$ is defined and I use its negative gradient as the force. In my base simulator, the stress is defined as a piece-wise linear function regarding the Green-Lagrange Strain, defined by Wang *et al.* [93]:

$$\mathbf{E} = \frac{1}{2}(\mathbf{F}^\top \mathbf{F} - \mathbf{I}) \tag{3.2}$$

Note that due to the geometric modeling of the cloth, there is no force caused by the thickness of the cloth. Most simulators use an extra 'bending force' as a compensation, following Grinspun *et al.* [94]. The bending force is defined between two adjacent faces when their dihedral angle is not a resting one.

The other two categories are relatively simpler. Constraint forces are defined as the negative gradient of the constraint energy, while frictional forces are created when two objects are in close proximity and have relative motions.

3.3.1.2 Dynamic Solve

In the simplest case, I solve $\mathbf{Ma} = \mathbf{f}$ for the acceleration and update the position and velocity accordingly, as shown in Algorithm 1. This Forward Euler method suffers from the well-known stability issue and often limits the time step size for the simulation. In order to take larger step for faster simulation, Backward Euler is often used. More specifically, I want my acceleration to match the force computed in the next time step:

$$\mathbf{M}\frac{\Delta \mathbf{v}}{\Delta t} = \mathbf{f}(\mathbf{x} + \Delta \mathbf{x}) = \mathbf{f}(\mathbf{x} + \Delta t(\mathbf{v} + \Delta \mathbf{v})) \tag{3.3}$$

By using Taylor Expansion, I have:

$$(\mathbf{M} - \Delta t^2 \frac{\partial \mathbf{f}}{\partial \mathbf{x}})\Delta \mathbf{v} = \Delta t \mathbf{f}(\mathbf{x} + \Delta t \mathbf{v}) \tag{3.4}$$

So the matrix used in the linear solve (Sec. 3.3.3) is defined as:

$$\hat{\mathbf{M}} = \mathbf{M} - \Delta t^2 \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \tag{3.5}$$

As long as I have the Jacobian of the forces $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$, I can compute a more stable result of $\Delta \mathbf{v}$ and can apply larger Δt, as discussed by Baraff and Witkin [27].

3.3.1.3 Collision Handling

As introduced in Sec. 3.3.4, I used continuous collision detection between two simulation steps to detect all possible collisions. When two faces collide with each other, there are two different collision types: vertex-face collision and edge-edge collision. The common trait is that at time of collision, the four involved vertices are in the same plane. Based on this, I can develop and solve a cubic equation regarding the time of collision, t (Sec. 3.3.4).

When the collision is detected, I need to form the corresponding constraint at time t:

$$\left(\sum_{k=1}^{4} w_k \mathbf{x}_k(t) \right) \cdot \mathbf{n} \geq d \tag{3.6}$$

Here, w_k is the weight parameter, $\mathbf{x}_k(t)$ is the vertex position at time t, $\mathbf{n}$ is the normal of the plane, and d is the cloth thickness. The weight parameters are determined using barycentric coordinates of the intersection point in the face (in the vertex-face collision case) or on the edges (in the edge-edge collision case).

I consider w and $\mathbf{n}$ as constants during the optimization, and $\mathbf{x}_k(t)$ is linearly interpolated between two time steps. So it is a linear constraint regarding $\mathbf{x}$. Combining all constraints together, I have:

$$\mathbf{G}\mathbf{x} + \mathbf{h} \leq 0. \tag{3.7}$$

In the collision response phase, I want to introduce minimum energy to move the vertex away so that all constraints can be satisfied. Therefore, I form this optimization as a QP problem, as shown later in Sec. 3.3.5.

3.3.2 Overview

I begin by defining the problem formally and providing common notation. A triangular mesh $\mathcal{M} = \{\mathcal{V}, \mathcal{E}, \mathcal{F}\}$ consists of sets of vertex states, edges, and faces, where the state of the vertices includes both position $\mathbf{x}$ and velocity $\mathbf{v}$. Given a cloth mesh $\mathcal{M}_t$ together with obstacle meshes $\mathcal{M}_t^{obs}$ at step t, a cloth simulator can compute the mesh state $\mathcal{M}_{t+1}$ at the next step $t+1$ based on the computed internal and external forces and the collision response. A simple simulation pipeline is shown in Algorithm 1, where $\mathbf{M}$ is the mass matrix, $\mathbf{f}$ is the force, and $\mathbf{a}$ is the acceleration. For more detailed description of cloth simulation, please refer to Sec. 3.3.1. All gradients except the linear solve (Line 4 in Algorithm 1) and the collision response (Line 7) can be computed using automatic differentiation in PyTorch [95].

Algorithm 1: Cloth simulation

```
1: $\mathbf{v}_0 \leftarrow \mathbf{0}$
2: for $t = 1$ to $n$ do
3:     $\mathbf{M}, \mathbf{f} \leftarrow$ compute_forces$(\mathbf{x}, \mathbf{v})$
4:     $\mathbf{a}_t \leftarrow \mathbf{M}^{-1}\mathbf{f}$
5:     $\mathbf{v}_t \leftarrow \mathbf{v}_{t-1} + \mathbf{a}_t \Delta t$
6:     $\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} + \mathbf{v}_t \Delta t$
7:     $\mathbf{x}_t \leftarrow \mathbf{x}_t +$ collision_response$(\mathbf{x}_t, \mathbf{v}_t, \mathbf{x}_t^{obs}, \mathbf{v}_t^{obs})$
8:     $\mathbf{v}_t \leftarrow (\mathbf{x}_t - \mathbf{x}_{t-1})/\Delta t$
9: end for
```

3.3.3 Derivatives of the Physics Solve

In modern simulation algorithms, implicit Euler is often used for stable integration results. Thus the mass matrix $\mathbf{M}$ used in Algorithm 1 often includes the Jacobian of the forces. I denote it below as $\hat{\mathbf{M}}$ in order to mark the difference. A linear solve will be needed to compute the acceleration since it is time consuming to compute $\hat{\mathbf{M}}^{-1}$. I use implicit differentiation to compute the gradients of the linear solve. Given an equation $\hat{\mathbf{M}}\mathbf{a} = \mathbf{f}$ with a solution $\mathbf{z}$ and the propagated gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{a}}|_{\mathbf{a}=\mathbf{z}}$, where $\mathcal{L}$ is the task-specific loss function, I can use the implicit differentiation form

$$\hat{\mathbf{M}}\partial \mathbf{a} = \partial \mathbf{f} - \partial \hat{\mathbf{M}}\mathbf{a} \tag{3.8}$$

to derive the gradient as

$$\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{M}}} = -\mathbf{d_a}\mathbf{z}^\top \quad \frac{\partial \mathcal{L}}{\partial \mathbf{f}} = \mathbf{d}_{\mathbf{a}}^\top, \tag{3.9}$$

where $\mathbf{d_a}$ is obtained from the linear system

$$\hat{\mathbf{M}}^\top \mathbf{d_a} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}}^\top. \tag{3.10}$$

The proof is as follows. I take $\frac{\partial \mathcal{L}}{\partial \mathbf{f}}$ as an example here, the derivation of $\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{M}}}$ is similar:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{f}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}} \cdot \frac{\partial \mathbf{a}}{\partial \mathbf{f}} = \mathbf{d_a}^\top \hat{\mathbf{M}} \cdot \hat{\mathbf{M}}^\dagger \mathbf{I} = \mathbf{d_a}^\top. \tag{3.11}$$

The first equality is given by the chain rule, the second is given by Equations 3.8 and 3.10, and $\hat{\mathbf{M}}^\dagger$ is the pseudoinverse of matrix $\hat{\mathbf{M}}$.

3.3.4 Dynamic Collision Detection and Response

As mentioned in Sec. 3.1, a static collision solver is not suitable for cloth because the total number of possible collision pairs is very high: quadratic in the number of faces. A common approach in cloth simulation is to dynamically detect collision on the fly and compute the response. I use a bounding volume hierarchy for collision detection [96], and non-rigid impact zones [92] to compute the collision response.

Specifically, I solve a cubic equation to detect the collision time t of each vertex-face or edge-edge pair that is sufficiently close to contact:

$$(\mathbf{x}_1 + \mathbf{v}_1 t) \cdot (\mathbf{x}_2 + \mathbf{v}_2 t) \times (\mathbf{x}_3 + \mathbf{v}_3 t) = 0, \tag{3.12}$$

where $\mathbf{x}_k$ and $\mathbf{v}_k$ $(k = 1, 2, 3)$ are the relative position and velocity to the first vertex. A solution that lies in $[0, 1]$ means that a collision is possible before the next simulation step. After making sure that the pair indeed intersects at time t, I set up one constraint for this collision, forcing the signed distance of this collision pair at time t to be no less than the thickness of the cloth δ. The signed distance of the vertex-face or edge-edge pair is linear to the vertex position $\mathbf{x}$. The set of all constraints then makes up a quadratic optimization problem as discussed later in Sec. 3.3.5.

For backpropagation, I need to compute the derivatives of the solution t since it is related to the parameters of the constraints. I use implicit differentiation here to simplify the process. Generally, given a cubic equation $ax^3 + bx^2 + cx + d = 0$, its implicit differentiation is of the following form:

$$(3ax^2 + 2bx + c)\partial x = \partial a x^3 + \partial b x^2 + \partial c x + \partial d. \tag{3.13}$$

Therefore I have

$$\begin{bmatrix} \frac{\partial x}{\partial a} & \frac{\partial x}{\partial b} & \frac{\partial x}{\partial c} & \frac{\partial x}{\partial d} \end{bmatrix} = \frac{1}{3ax^2 + 2bx + c} \begin{bmatrix} x^3 & x^2 & x & 1 \end{bmatrix}. \tag{3.14}$$

3.3.5 Derivatives of the Collision Response

A general approach to integrating collision constraints into physics simulation has been proposed by Belbute-Peres *et al.* [13]. However, as mentioned in Sections 3.1 and 3.2, constructing a static LCP is often impractical in cloth simulation because of high dimensionality. Collisions that actually happen in each step are very sparse compared to the complete set. Therefore, I use a dynamic approach that incorporates collision detection and response.

Collision handling in my implementation is based on impact zone optimization [74]. It finds all colliding instances using continuous collision detection (Sec. 3.3.4) and sets up the constraints for all collisions. In order to introduce minimum change to the original mesh state, I develop a QP problem to solve for the constraints. Since the signed distance function is linear in $\mathbf{x}$, the optimization takes a quadratic form:

$$\underset{\mathbf{z}}{\text{minimize}} \quad \frac{1}{2}(\mathbf{z}-\mathbf{x})^\top \mathbf{W}(\mathbf{z}-\mathbf{x}) \tag{3.15}$$

$$\text{subject to} \quad \mathbf{G}\mathbf{z}+\mathbf{h} \leq \mathbf{0} \tag{3.16}$$

where $\mathbf{W}$ is a constant diagonal weight matrix related to the mass of each vertex, and $\mathbf{G}$ and $\mathbf{h}$ are constraint parameters. I further denote the number of variables and constraints by n and m, *i.e.* $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{h} \in \mathbb{R}^m$, and $\mathbf{G} \in \mathbb{R}^{m\times n}$. Note that this optimization is a function with inputs $\mathbf{x}$, $\mathbf{G}$, and $\mathbf{h}$, and output $\mathbf{z}$. My goal here is to derive $\frac{\partial \mathcal{L}}{\partial \mathbf{x}}$, $\frac{\partial \mathcal{L}}{\partial \mathbf{G}}$, and $\frac{\partial \mathcal{L}}{\partial \mathbf{h}}$ given $\frac{\partial \mathcal{L}}{\partial \mathbf{z}}$, where $\mathcal{L}$ refers to the loss function.

When computing the gradient using implicit differentiation [97], the dimensionality of the linear system (Equation 3.20) can be too high. My key observation here is that $n \gg m > \operatorname{rank}(\mathbf{G})$, since one contact often involves 4 vertices (thus 12 variables) and some contacts may be linearly dependent (*e.g.* multiple adjacent collision pairs). OptNet [97] solves a linear equation of size $m+n$, which is more than necessary. I introduce a simpler and more efficient algorithm below to minimize the size of the linear equation.

3.3.5.1 QR Decomposition

To make things simpler, I assume that $\mathbf{G}$ is of full rank in this section. At global minimum $\mathbf{z}^*$ and λ^* of the Lagrangian, the following holds for stationarity and complementary slackness conditions:

$$\mathbf{W}\mathbf{z}^* - \mathbf{W}\mathbf{x} + \mathbf{G}^\top \lambda^* = 0 \tag{3.17}$$

$$D(\lambda^*)(\mathbf{G}\mathbf{z}^* + \mathbf{h}) = 0, \tag{3.18}$$

with their implicit differentiation as

$$\begin{bmatrix} \mathbf{W} & \mathbf{G}^\top \\ D(\lambda^*)\mathbf{G} & D(\mathbf{G}\mathbf{z}^* + \mathbf{h}) \end{bmatrix} \begin{bmatrix} \partial \mathbf{z} \\ \partial \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{W}\partial \mathbf{x} - \partial \mathbf{G}^\top \lambda^* \\ -D(\lambda^*)(\partial \mathbf{G}\mathbf{z}^* + \partial \mathbf{h}) \end{bmatrix}, \tag{3.19}$$

where $D()$ transforms a vector to a diagonal matrix. Using similar derivation to Sec. 3.3.3, solving the equation

$$\begin{bmatrix} \mathbf{W} & \mathbf{G}^\top D(\lambda^*) \\ \mathbf{G} & D(\mathbf{G}\mathbf{z}^* + \mathbf{h}) \end{bmatrix} \begin{bmatrix} \mathbf{d}_\mathbf{z} \\ \mathbf{d}_\lambda \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{z}}^\top \\ \mathbf{0} \end{bmatrix} \tag{3.20}$$

can provide the desired gradient:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \mathbf{d}_{\mathbf{z}}^T \mathbf{W} \tag{3.21}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{G}} = -D(\lambda^*)\mathbf{d}_\lambda \mathbf{z}^{*\top} - \lambda^* \mathbf{d}_{\mathbf{z}}^\top \tag{3.22}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}} = -\mathbf{d}_\lambda^T D(\lambda^*). \tag{3.23}$$

(See Sec. 3.3.6.2 for the derivation.) However, as mentioned before, directly solving Equation 3.20 may be computationally expensive in my case. I show that by performing a QR decomposition, the solution can be derived without solving a large system.

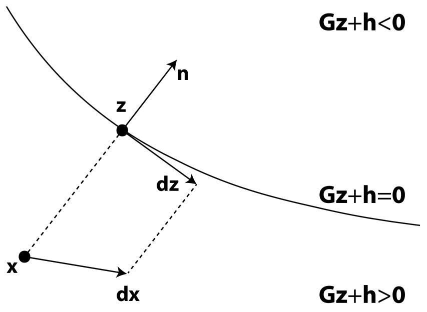

Figure 3.1: Impact of perturbation. A small perturbation of the target position will cause the final result to move along the constraint surface.

To further reduce computation, I assume that no constraint is ‘over-satisfied’, *i.e.* $\mathbf{G}\mathbf{z}^* + \mathbf{h} = \mathbf{0}$. I will remove these assumptions later in Sec. 3.3.5.2. I compute the QR decomposition of $\sqrt{\mathbf{W}}^{-1}\mathbf{G}^\top$:

$$\sqrt{\mathbf{W}}^{-1}\mathbf{G}^\top = \mathbf{Q}\mathbf{R}. \tag{3.24}$$

The solution of Equation 3.20 can be expressed as

$$\mathbf{d_z} = \sqrt{\mathbf{W}}^{-1}(\mathbf{I} - \mathbf{Q}\mathbf{Q}^\top)\sqrt{\mathbf{W}}^{-1}\frac{\partial \mathcal{L}}{\partial \mathbf{z}}^\top \tag{3.25}$$

$$\mathbf{d}_\lambda = D(\lambda^*)^{-1}\mathbf{R}^{-1}\mathbf{Q}^\top\sqrt{\mathbf{W}}^{-1}\frac{\partial \mathcal{L}}{\partial \mathbf{z}}^\top, \tag{3.26}$$

where $\sqrt{\mathbf{W}}^{-1}$ is the inverse of the square root of a diagonal matrix. The result above can be verified by substitution in Equation 3.20.

The intuition behind Equation 3.25 is as follows. When perturbing the original point $\mathbf{x}$ in an optimization, the resulting displacement of $\mathbf{z}$ will be moving along the surface of $\mathbf{Gx} + \mathbf{h} = \mathbf{0}$, which will become perpendicular to the normal when the perturbation is small. (Fig. 3.1 illustrates this idea in two dimensions.) This is where the term $\mathbf{I} - \mathbf{Q}\mathbf{Q}^\top$ comes from. Note that $\sqrt{\mathbf{W}}^{-1}\mathbf{G}^\top$ is an $n \times m$ matrix, where $n \gg m$ and the QR decomposition will only take $O(nm^2)$ time, compared to $O((n+m)^3)$ in the original dense linear solve. After that I will need to solve a linear system in Equation 3.26, but it is more efficient than solving Equation 3.20 since it is only of size m, and $\mathbf{R}$ is an upper-triangular matrix. In my collision response case, where $n \leq 12m$, my method can provide up to 183x acceleration in theory. The speed-up in my experiments (Sec. 3.4) ranges from 60x to 130x for large linear systems.

3.3.5.2 Low-rank Constraints

The algorithm above cannot be directly applied when $\mathbf{G}$ is low-rank, or when some constraint is not at boundary. This will cause $\mathbf{R}$ or $D(\lambda^*)$ to be singular. I now show that the singularity can be avoided via small modifications to the algorithm.

First, if $\lambda_k = 0$ for the kth constraint then $\mathbf{d}\lambda_k$ doesn't matter. This is because the final result contains only components of $D(\lambda^*)\mathbf{d}\lambda$ but not $\mathbf{d}\lambda$ alone, as shown in Equations 3.22 and 3.23. Intuitively, if the constraint is over-satisfied, then perturbing the parameters of that constraint will not have impact on $\mathbf{z}$. Based on this observation, I can remove the constraints in $\mathbf{G}$ when their corresponding λ is 0.

Next, if $\mathbf{G}$ is of rank k, where $k < m$, then I can rewrite Equation 3.24 as

$$\sqrt{\mathbf{W}}^{-1}\mathbf{G}^\top = \mathbf{Q}_1[\mathbf{R}_1 \quad \mathbf{R}_2], \tag{3.27}$$

where $\mathbf{Q}_1 \in \mathbb{R}^{n\times k}$, $\mathbf{R}_1 \in \mathbb{R}^{k\times k}$, and $\mathbf{R}_2 \in \mathbb{R}^{k\times(m-k)}$. Getting rid of $\mathbf{R}_2$ (*i.e.* removing those constraints from the beginning) does not affect the optimization result, but may change λ so that the computed gradients are incorrect. Therefore, I need to transfer the Lagrange multipliers to the linearly independent terms first:

$$\lambda_1 \leftarrow \lambda_1 + \mathbf{R}_1^{-1}\mathbf{R}_2\lambda_2, \tag{3.28}$$

where λ_1 and λ_2 are the Lagrange multipliers corresponding to the constraints on $\mathbf{R}_1$ and $\mathbf{R}_2$.

3.3.6 Derivations of the Gradient Computation

3.3.6.1 Proof of Equation 3.9

I now show that $\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{M}}} = -\mathbf{d}_{\mathbf{a}}\mathbf{z}^{\top}$. For convenience of expression, I split the matrix $\hat{\mathbf{M}}$ into elements $\{\hat{\mathbf{M}}_{i,j}\}$. Setting irrelevant variables to zero, I obtain from Equation 3.8 that:

$$\hat{\mathbf{M}}\partial\mathbf{a} = -\partial\hat{\mathbf{M}}\mathbf{z} = \begin{pmatrix} \mathbf{0} \\ -\partial\hat{\mathbf{M}}_{i,j}\mathbf{z}_j \\ \mathbf{0} \end{pmatrix} \tag{3.29}$$

Hence, I have:

$$\frac{\partial\mathbf{a}}{\partial\hat{\mathbf{M}}_{i,j}} = \hat{\mathbf{M}}^{\dagger}\begin{pmatrix} \mathbf{0} \\ -\mathbf{z}_j \\ \mathbf{0} \end{pmatrix} \tag{3.30}$$

Similar to Equation 3.11, I arrive at:

$$\frac{\partial\mathcal{L}}{\partial\hat{\mathbf{M}}_{i,j}} = \frac{\partial\mathcal{L}}{\partial\mathbf{a}} \cdot \frac{\partial\mathbf{a}}{\partial\hat{\mathbf{M}}_{i,j}} = \mathbf{d}_{\mathbf{a}}^{\top}\hat{\mathbf{M}} \cdot \hat{\mathbf{M}}^{\dagger}\begin{pmatrix} \mathbf{0} \\ -\mathbf{z}_j \\ \mathbf{0} \end{pmatrix} = -\mathbf{d}_{\mathbf{a}_i}\mathbf{z}_j \tag{3.31}$$

Combining all elements in $\hat{\mathbf{M}}$ together I have:

$$\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{M}}} = -\mathbf{d_a} \mathbf{z}^\top \tag{3.32}$$

3.3.6.2 Proof of Equation 3.20-3.23

Let $\hat{\mathbf{z}} = \begin{bmatrix} \mathbf{z} & \lambda \end{bmatrix}^\top$. Using Equation 3.19 I have:

$$\frac{\partial \hat{\mathbf{z}}}{\partial \mathbf{x}} = \begin{bmatrix} \mathbf{W} & \mathbf{G}^\top \\ D(\lambda^*)\mathbf{G} & D(\mathbf{G}\mathbf{z}^* + \mathbf{h}) \end{bmatrix}^\dagger \begin{bmatrix} \mathbf{W} \\ \mathbf{0} \end{bmatrix} \tag{3.33}$$

$$\frac{\partial \hat{\mathbf{z}}}{\partial \mathbf{h}} = \begin{bmatrix} \mathbf{W} & \mathbf{G}^\top \\ D(\lambda^*)\mathbf{G} & D(\mathbf{G}\mathbf{z}^* + \mathbf{h}) \end{bmatrix}^\dagger \begin{bmatrix} \mathbf{0} \\ -D(\lambda^*) \end{bmatrix} \tag{3.34}$$

Then, the chain rule can yield the results as:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{z}}} \cdot \frac{\partial \hat{\mathbf{z}}}{\partial \mathbf{x}} \tag{3.35}$$

$$= \begin{bmatrix} \mathbf{d}_\mathbf{z}^\top & \mathbf{d}_\lambda^\top \end{bmatrix} \begin{bmatrix} \mathbf{W} & \mathbf{G}^\top \\ D(\lambda^*)\mathbf{G} & D(\mathbf{G}\mathbf{z}^* + \mathbf{h}) \end{bmatrix} \cdot \begin{bmatrix} \mathbf{W} & \mathbf{G}^\top \\ D(\lambda^*)\mathbf{G} & D(\mathbf{G}\mathbf{z}^* + \mathbf{h}) \end{bmatrix}^\dagger \begin{bmatrix} \mathbf{W} \\ \mathbf{0} \end{bmatrix} \tag{3.36}$$

$$= \mathbf{d}_\mathbf{z}^\top \mathbf{W} \tag{3.37}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{z}}} \cdot \frac{\partial \hat{\mathbf{z}}}{\partial \mathbf{h}} \tag{3.38}$$

$$= \begin{bmatrix} \mathbf{d}_{\mathbf{z}}^\top & \mathbf{d}_\lambda^\top \end{bmatrix} \begin{bmatrix} \mathbf{W} & \mathbf{G}^\top \\ D(\lambda^*)\mathbf{G} & D(\mathbf{G}\mathbf{z}^* + \mathbf{h}) \end{bmatrix} \cdot \begin{bmatrix} \mathbf{W} & \mathbf{G}^\top \\ D(\lambda^*)\mathbf{G} & D(\mathbf{G}\mathbf{z}^* + \mathbf{h}) \end{bmatrix}^\dagger \begin{bmatrix} \mathbf{0} \\ -D(\lambda^*) \end{bmatrix} \tag{3.39}$$

$$= -\mathbf{d}_\lambda^\top D(\lambda^*) \tag{3.40}$$

Similarly as Sec. 3.3.6.1, I split the matrix $\mathbf{G}$ into elements $\{\mathbf{G}_{i,j}\}$. From Equation 3.19 I have:

$$\begin{bmatrix} \mathbf{W} & \mathbf{G}^\top \\ D(\lambda^*)\mathbf{G} & D(\mathbf{G}\mathbf{z}^* + \mathbf{h}) \end{bmatrix} \partial \hat{\mathbf{z}} = \begin{bmatrix} -\partial \mathbf{G}^\top \lambda^* \\ -D(\lambda^*)\partial \mathbf{G}\mathbf{z}^* \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ -\partial \mathbf{G}_{i,j}\lambda_i^* \\ \mathbf{0} \\ -\lambda_i^* \partial \mathbf{G}_{i,j}\mathbf{z}_j^* \\ \mathbf{0} \end{bmatrix} \tag{3.41}$$

which indicates that:

$$\frac{\partial \hat{\mathbf{z}}}{\partial \mathbf{G}_{i,j}} = \begin{bmatrix} \mathbf{W} & \mathbf{G}^\top \\ D(\lambda^*)\mathbf{G} & D(\mathbf{G}\mathbf{z}^* + \mathbf{h}) \end{bmatrix}^\dagger \begin{bmatrix} \mathbf{0} \\ -\lambda_i^* \\ \mathbf{0} \\ -\lambda_i^* \mathbf{z}_j^* \\ \mathbf{0} \end{bmatrix} \tag{3.42}$$

So the chain rule gives:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{G}_{i,j}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{z}}} \cdot \frac{\partial \hat{\mathbf{z}}}{\partial \mathbf{G}_{i,j}} \tag{3.43}$$

$$= \begin{bmatrix} \mathbf{d}_{\mathbf{z}}^{\top} & \mathbf{d}_{\lambda}^{\top} \end{bmatrix} \begin{bmatrix} \mathbf{W} & \mathbf{G}^{\top} \\ D(\lambda^*)\mathbf{G} & D(\mathbf{G}\mathbf{z}^* + \mathbf{h}) \end{bmatrix} \cdot \begin{bmatrix} \mathbf{W} & \mathbf{G}^{\top} \\ D(\lambda^*)\mathbf{G} & D(\mathbf{G}\mathbf{z}^* + \mathbf{h}) \end{bmatrix}^{\dagger} \begin{bmatrix} \mathbf{0} \\ -\lambda_i^* \\ \mathbf{0} \\ -\lambda_i^* \mathbf{z}_j^* \\ \mathbf{0} \end{bmatrix} \tag{3.44}$$

$$= -\mathbf{d}_{\mathbf{z}_j} \lambda_i^* - \mathbf{d}_{\lambda_i} \lambda_i^* \mathbf{z}_j^* \tag{3.45}$$

Combining all elements in $\mathbf{G}$ together, I have:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{G}} = -\lambda^* \mathbf{d}_{\mathbf{z}}^{\top} - D(\lambda^*) \mathbf{d}_{\lambda} \mathbf{z}^{*\top} \tag{3.46}$$

3.4 Experiments

I conduct three experiments to showcase the power of differentiable cloth simulation. First, I use an ablation study to quantify the performance gained by using my method to compute the gradient. Next, I use the computed gradient to optimize the physical parameters of cloth. Lastly, I demonstrate the ability to control cloth motion.

3.4.1 Ablation Study

As mentioned in Sec. 3.3.5.1, my method for computing the gradients of the optimization can achieve a speed-up of up to 183x in theory. I conduct an ablation study to verify this estimate in practice. In order to clearly measure the timing difference, I design a scenario with many collisions. I put a piece of cloth into an upside-down square pyramid, so that the cloth is forced to fold, come into frequent contact with the pyramid, and collide with itself, as shown in Fig. 3.2.

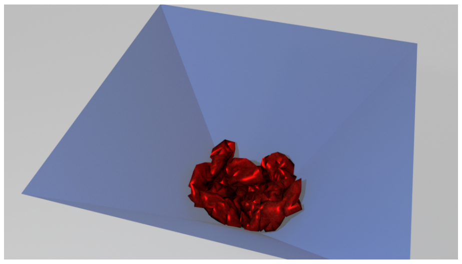
Figure 3.2: Example frame from the ablation study. A piece of cloth is crumpled inside a square pyramid, so as to generate a large number of collisions.

I measure the running time of backpropagation in each quadratic optimization and also the running time of the physics solve as a reference. With all other variables fixed, I compare to the baseline method where the gradients are computed by directly solving Equation 3.20. Timings are listed in Tab. 3.1. In this experiment, the backpropagation of the physics solve takes from 0.007s to 0.5s, which, together with the timings of the baseline, implies that the collision handling step is the critical bottleneck when there are many collisions in the scene. The results in Tab. 3.1 show that my proposed method can significantly decrease the matrix size required for computation and thus the actual running time, resolving the bottleneck in backpropagation.

The experimental results also match well with the theory in Sec. 3.3.5. Each collision involves a vertex-face or edge-edge pair, which both have 4 vertices and 12 variables. Therefore, the original matrix size ($n + m = 13m$) should be about 13 times bigger than in my method (m). In my experiment, the ratio of the matrix size is indeed close to 13. Possible reasons for the ratio not being exactly 13 include (a) multiple collision pairs that share the same vertex, making n smaller, and (b) the constraint matrix can be of low rank, as described in Sec. 3.3.5.2, making the effective m smaller in practice.

Mesh	Baseline		Mine		Speedup	
resolution	Matrix size	Runtime (s)	Matrix size	Runtime (s)	Matrix size	Runtime
16x16	599 ± 76	0.33 ± 0.13	$\mathbf{66 \pm 26}$	$\mathbf{0.013 \pm 0.0019}$	8.9	25
32x32	1326 ± 23	1.2 ± 0.10	$\mathbf{97 \pm 24}$	$\mathbf{0.011 \pm 0.0023}$	13	112
64x64	2024 ± 274	4.6 ± 0.33	$\mathbf{242 \pm 47}$	$\mathbf{0.072 \pm 0.011}$	8.3	64

Table 3.1: Statistics of the backward propagation with and without my method for various mesh resolutions. I report the average values in each cell with the corresponding standard deviations. By using my method, the runtime of gradient computation is reduced by up to two orders of magnitude.

3.4.2 Material Estimation

In this experiment, my aim is to learn the material parameters of cloth from observation. The scene features a piece of cloth hanging under gravity and subjected to a constant wind force, as shown in Fig. 3.3. I use the material model from Wang *et al*. [93]. It consists of three parts: density d, stretching stiffness $\mathbf{S}$, and bending stiffness $\mathbf{B}$. The stretching stiffness quantifies how large the reaction force will be when the cloth is stretched out; the bending stiffness models how easily the cloth can be bent and folded.

I used the real-world dataset from Wang *et al*. [93], which consists of 10 different cloth materials. There are in total 50 frames of simulated data. The first 25 frames are taken as input and all 50 frames are used to measure accuracy. This is a case-by-case optimization problem. My goal is to fit the observed data in each sequence as well as possible, with no "training set" used for training.

In my optimization setup, I use SGD with learning rate ranging from 0.01 to 0.1 and momentum from 0.9 to 0.99, depending on the convergence speed. The initial guess is the set of average values across all materials. I define the loss as the average MSE across all frames. In order to speed up optimization, I gradually increase the number of frames used. Specifically, I first optimize the parameters using only 1 simulated frame. I proceed to the second frame after the loss decreases to a certain threshold. This optimization scheme can help obtain a relatively good guess before additional frames are involved.

As a simple baseline, I measure the total external force and divide it by the observed acceleration to compute the density. For the stretching stiffness, I simplify the model to an isotropic one and record the maximum deformation magnitude along the vertical axis. Since the effect of the bending stiffness is too subtle to observe, I directly use the averaged value as my prior. I also compare my method with the L-BFGS optimization by Wang *et al*. [93] using finite difference. I used the PyTorch L-BFGS implementation and set the learning rate ranging from 0.1 to 0.2 depending on the convergence speed.

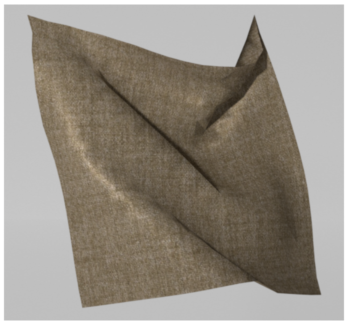
Figure 3.3: Example frame from the material estimation scene for cloth blowing in the wind.

For the performance measurement, I use the Frobenius norm normalized by the target as the metric for the material parameters:

$$\mathcal{E}(\mathbf{P}) = \frac{\|\mathbf{P} - \mathbf{P}_0\|_F}{\|\mathbf{P}_0\|_F}, \tag{3.47}$$

where $\mathbf{P}$ and $\mathbf{P}_0$ are the estimated and the target physics parameters, which stand for either density d, stretching stiffness $\mathbf{S}$, or bending stiffness $\mathbf{B}$. In order to show the final visual effect, I also measure the average distance of the vertices between the estimated one and the target normalized by the size of the cloth as another metric:

$$\mathcal{E}(\mathbf{X}) = \frac{1}{nTL} \sum_{1 \leq i \leq T, 1 \leq j \leq n} \|\mathbf{X}_{i,j} - \mathbf{Y}_{i,j}\|_2, \tag{3.48}$$

where L is the size of the cloth, and $\mathbf{X}$ and $\mathbf{Y}$ are $T \times n \times 3$ matrices denoting the n simulated vertex positions across T frames using the estimated parameter and the target, respectively.

Tab. 3.2 shows the estimation result. I achieve a much smaller error in most measurements in comparison to the baselines. The reason the stiffness matrices do not have low error is that (a) a large part of them describes the nonlinear stress behavior that needs a large deformation of the cloth and is not commonly observed in my environment, (b) different stiffness values can sometimes provide similar results, and (c) the bending force for common cloth materials is too small compared to gravity and the wind forces to make an impact. The table shows that the linear part of the stiffness matrix is optimized well. With the computed gradient using my model, one can effectively optimize the unknown parameters that dominate the cloth movement to fit the observed data.

Compared with regular simulators, my simulator is designed to be embedded in deep networks. When gradients are needed, my simulator shows significant improvement over finite-difference methods, as shown in Tab. 3.2. Regular simulators need to run one simulation for each input variable to compute the gradient, while my method only needs to run once for all gradients to be computed. Therefore, the more input variables there are during learning, the greater the performance gain that can be achieved by my method over finite-difference methods.

Method	Runtime (sec/step/iter)	Density error (%)	Non-ln streching stiffness error (%)	Ln streching stiffness error (%)	Bending stiffness error (%)	Simulation error (%)
Baseline	-	68 ± 46	74 ± 23	160 ± 119	**70 ± 42**	12 ± 3.0
L-BFGS [93]	2.89 ± 0.02	4.2 ± 5.6	64 ± 34	72 ± 90	**70 ± 43**	4.9 ± 3.3
Mine	**2.03 ± 0.06**	**1.8 ± 2.0**	**57 ± 29**	**45 ± 41**	77 ± 36	**1.6 ± 1.4**

Table 3.2: Results on the material parameter estimation task. Lower is better. 'Ln' stands for 'linear'. Values of the material parameters are the Frobenius norms of the difference normalized by the Frobenius norm of the target. Values of the simulated result are the average pairwise vertex distance normalized by the size of the cloth. My gradient-based method yields much smaller error than the baselines.

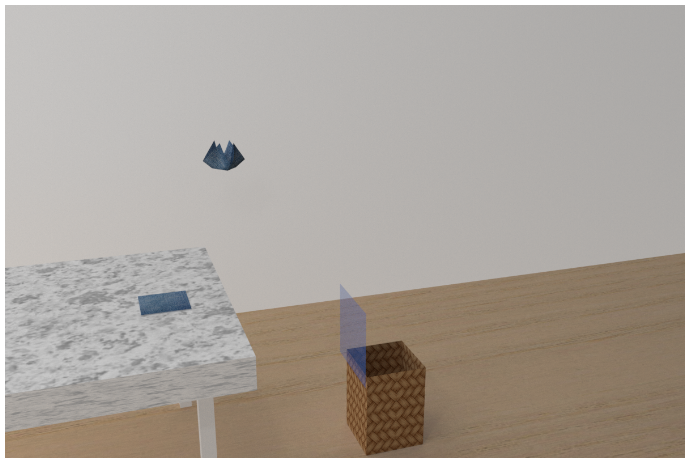

Figure 3.4: Example frame from the motion control experiment: dropping cloth into a basket.

3.4.3 Motion Control

I further demonstrate the power of my differentiable simulator by optimizing control parameters for motion control of cloth. The intended task is to drop a piece of cloth into a basket, as shown in Fig. 3.4. The cloth is originally placed on a table that is away from the basket. The system then applies external forces to the corners of the cloth to lift it and drop it into the basket. The external force is applied for 3 seconds and can be changed during this period. The basket is a box with an open top. A planar obstacle is placed between the cloth and the basket to increase the difficulty of the task.

The initial control force is set to zero. The control network consists of two FC layers, where the input (size $81 \times 2 \times 3$) is the position and velocity of each vertex, the hidden layer is of size 200, and the output is the control force (size 4×3). The learning rate is 10^{-4} and the momentum is 0.5. The reported result is the best among 10 trials.

I define the loss here as the squared distance between the center of mass of the cloth and the bottom of the basket. To demonstrate the ability to embed the simulator into neural networks, I also couple my simulator with a two-layer fully-connected (FC) network that takes the mesh states as input and outputs the control forces. My methods here are compared to two baselines. One of the baselines is a simple method that computes the momentum needed at every time step. The entire cloth is treated as a point mass and an external force is computed at each time step to control the point mass towards the goal. Obstacles are simply neglected in this method. The other baseline is the PPO algorithm, as implemented in Ray RLlib [98]. The reward function is defined as the negative of the distance of the center of mass of the cloth to the bottom of the basket.

Method	Error (%)	Samples
Point mass	111	–
PPO [98]	432	10,000
Mine	**17**	**53**
Mine+FC	39	108

Table 3.3: Motion control results. The table reports the smallest distance to the target position, normalized by the size of the cloth, and the number of samples used during training.

Tab. 3.3 shows the performance of the different methods and their sample complexity. The error shown in the table is the distance defined above normalized by the size of the cloth. My method achieves the best performance with a much smaller number of simulation steps. The bottom of the basket in my setting has the same size as the cloth, so a normalized error of less than 50%, as my methods achieve, implies that the cloth is successfully dropped into the basket.

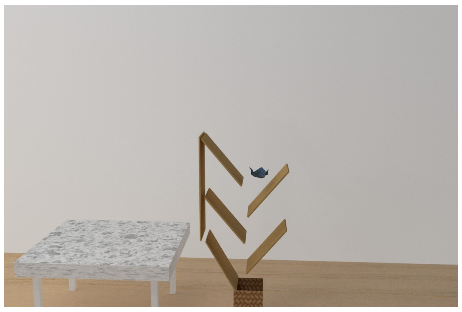

Figure 3.5: A motion control scene with more obstacles. The cloth needs to drop down and slide through the slopes to get to the target position.

3.4.4 Collision-rich Motion Control

I here demonstrate an example of motion control application with richer collisions. As shown in Figure 3.5, there is a series of obstacles above the basket that preclude the cloth from falling directly into it. The variable settings are the same as described in Sec. 3.4.3. My differentiable simulation provides the task with correct gradients so that the cloth is deposited into the basket.

3.5 Conclusion

I presented a differentiable cloth simulator that can compute the analytical gradient of the simulation function with respect to the input parameters. I used dynamic collision handling and explicitly derived its gradient. Implicit differentiation is used in computing gradients of the linear solver and collision response. Experiments have demonstrated that my method accelerates backpropagation by up to two orders of magnitude.

I have demonstrated the potential of differentiable cloth simulation in two application scenarios: material estimation and motion control. By making use of the gradients from the physically-aware simulation, my method can optimize the unknown parameters faster and more accurately than gradient-free baselines. Using differentiable simulation, I can learn the intrinsic properties of cloth from observation.

One limitation of my existing implementation is that the current simulation architecture is not optimized for large-scale vectorized operations, which introduces some overhead. This can be addressed by a specialized, optimized simulation system based solely on tensor operations.

This work has been published in the conference proceedings of Neural Information Processing Systems (NeurIPS) 2019.