SIMULATING YARN-BASED CLOTH

A Dissertation
Presented to the Faculty of the Graduate School
of Cornell University
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

by
Jonathan M. Kaldor
May 2011

SIMULATING YARN-BASED CLOTH

Jonathan M. Kaldor, Ph.D.

Cornell University 2011

Cloth is an important material to model and simulate correctly, both in computer graphics and other industrial applications. The commonly used models for cloth in computer graphics typically approximate the cloth as an elastic sheet with linear isotropic behavior inspired by the construction of woven fabrics. However, they do a poor job of predicting the behavior of knits, which are driven by the complex interactions of yarn loops pulled through each other. This thesis presents a yarn-based model for cloth where yarns in the fabric are explicitly modeled as inextensible but flexible spline curves. Yarn dynamics are dictated by both energy terms and hard constraints, while friction interactions, a critical component of correct yarn behavior, are approximated using a velocity filter that penalizes locally non-rigid motion. Qualitative comparison of the model to observed deformations of hand-knitted samples in the laboratory showed that the model predicts key mechanical properties of different knits. Since this model is slower than sheet-based approaches, further work looked at accelerating the model through both localized rigidification and adaptive contact linearization. In localized rigidification, regions of the cloth behaving almost rigidly are simulated using a cheaper model which avoids many of the expensive force computations. For adaptive contact linearization, yarn contacts are grouped into contact sets, with the associated contact force computed exactly at one timestep, and then approximated on subsequent steps via linearization in a rotated reference frame for nearby geometric configurations. Finally, additional work looked at the problem of creating initial yarn geometry for subsequent simulation as a yarn-based model.

BIOGRAPHICAL SKETCH

Jonathan Kaldor was born November 14th, 1980 and graduated from Miami Killian Senior High School in 1999 and Amherst College in 2003 with a Bachelor of Arts, double majoring in Mathematics and Computer Science. He started at Cornell University in 2004.

To my grandfather, Gilbert Tougas

ACKNOWLEDGMENTS

I would like to thank my family and friends, particularly my parents Kay and Michael, for all of the support and encouragement they have given me through the years. At Cornell, Robert Kleinberg and Charles Van Loan both served on my committee and gave me help and suggestions on various problems as they arose. My advisors Steve Marschner and Doug James were both an invaluable help, constantly providing not only advice, guidance, ideas, and suggestions, but also reading through drafts of this work and judging the appearance of an almost-literally uncountable number of videos of falling blue scarves. This research was supported in part by funding from the following: the National Science Foundation (CCF-0702490), the NSF CAREER program (CCF-0347303, CCF-0652597), two Alfred P. Sloan Research Fellowships, and additional support from Intel, The Boeing Company, Pixar, Autodesk, NVIDIA, and Unilever.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Computer simulation of various materials has become an integral part of the workflow in many industries, able to capture and reproduce characteristic behaviors and motions in cases where doing so in real life would be unnecessarily costly, time-consuming, or even impossible. Thus, simulation of prototypes allows designers to evaluate feasibility and make alterations virtually before ever physically creating the part, while film directors can add massive and realistic-looking special effects to movies which would otherwise be entirely impractical from both a physical and monetary point of view. Crucially, effective simulation depends on both the performance of modern hardware and the abstractions and approximations made by the simulation, and as the former has improved in speed so have the latter in accuracy and realism.

Among materials to simulate one of the most important is cloth, both for its versatility and ubiquity. Engineering applications have long focused on accurate cloth simulation for its importance in bulletproof armor [45, 122, 123, 85], composite materials [28, 12], and medical applications [121]. Similarly, computer graphics have emphasized cloth simulation due to the need to accurately reproduce clothing for human (and other) characters. However, most research on cloth mechanics has focused on woven cloth, both for its simplicity and because many fabrics used in engineering applications are woven. In clothing, though, knit fabrics are as commonly used as wovens, and many very common garments, such as T-shirts or leggings, owe their existence to knits and cannot be made from woven material.

The distinction between knits and wovens is important for simulation because their mechanical structures are entirely dissimilar, and as a result they behave differently at all scales. The yarns in woven fabric are nearly immobile, leading to an almost inextensible sheet with limited deformations in the yarn structure. In contrast, the interlocked loops in a knit material deform and slide readily, leading to a highly extensible sheet with dramatic changes in small-scale structure as the material stretches. This stretchiness is exploited as a key feature in garments, allowing for clothing to stretch and conform to the body in certain places like collars and sleeves.

Due to limited computational resources, cloth simulation in computer graphics thus far generally uses models that approximate the mechanics of linear elastic sheets. Because of the small in-plane deformations of woven materials, acceptable realism can often be achieved for woven fabric using these models. But linear-elastic sheet models inevitably look "rubbery" if they are allowed to stretch as much as a typical knit fabric does. This is unsurprising, since the mechanics of interlocking loops in a knit fabric bears little resemblance to the mechanics of a continuous elastic material—a fundamentally different kind of model is required. The small-scale behavior of knits is also important because many knits are made with large yarns, meaning that the yarn structure is clearly visible and must behave correctly for realistic results.

As available computational power has increased, though, the need to abstract away the internal yarn structure has diminished. This thesis meets the challenge of knits head-on, by directly solving for the motion of the yarn that makes up the fabric and the interactions between loops that determine its behavior. The physical model is concisely described by the behavior of a yarn: an inextensible curve with resistance to bending, a collision force that resists interpenetration, and damping for stability and to stand in for the effects of friction. From this model, though, significant yarn structures can be simulated, producing rich, complex deformations that are impossible to achieve using any kind of sheet-based simulation. Moreover, many of the properties of knit structures emerge naturally from the simulation as a result of the intra- and inter-yarn interactions, including the characteristic shapes and textures produced by different knitting patterns and the varying extensibility of the knit sheets. In addition, evaluation of repetitive yarn-level computations can naturally exploit multicore architectures, allowing for large knits to be simulated in practice.

A simulation at this level of detail is required for realistic results with coarse-knit garments like sweaters, scarves, or socks, because of their visible yarn structure. Furthermore, yarn-level simulation is a fundamental tool for studying the large-scale properties of finely knit fabric, in order to develop continuum models that can realistically describe knits under large deformations. The same approach can also lead to models for wovens that are able to capture the material-dependent subtleties missed by current models.

Despite the increases in computational power, though, yarn-based models are still significantly more costly than their sheet-based counterparts, and scaling up to large pieces of clothing presents problems. While sheet models can readily be coarsened for faster simulation, yarn models always need enough detail to describe the shape of the yarn, so reducing the number of degrees of freedom is not straightforward. In addition, sheet models only generate contact processing work when the sheet collides with objects or folds over and collides with itself, but yarn models derive their whole behavior from the thousands of self-collisions within the fabric's structure. These yarn self-contacts are what create the rich and interesting deformations, so they cannot simply be discarded. Ultimately, in order to make yarn-based simulation of character-sized garments practical to simulate, the model must be further approximated while still preserving the same overall quality of motion.

The remainder of this thesis is organized as follows: In Chapter 2, the basics of cloth structure are presented, while Chapter 3 discusses prior work on cloth simulation both in computer graphics and in other fields. Chapter 4 presents the yarn model and performs a qualitative validation to laboratory tests. Chapters 5 and 6 both look at different types of approximations that can be made in order to achieve further speedups to bring the simulation into the realm of practicality, with Chapter 5 exploring speedups through localized rigidification and Chapter 6 defining an approximate contact response which preserves quality while providing significant speedups. Finally, Chapter 7 discusses the challenges in constructing the initial yarn models, and Chapter 8 discusses some future avenues of research in yarn-based cloth simulation.

CHAPTER 2

CLOTH AT A GLANCE

Modeling the cloth at the yarn level requires, quite naturally, an understanding of the underlying geometry in real cloth. Section 2.1 discusses the geometry of cloth, which is formed from yarns and categorized based on how the yarns are interrelated. This geometry leads to distinct mechanical properties of different kinds of fabric, covered in overview in Section 2.2, and capturing these properties automatically and efficiently will be the goal of the yarn-based cloth simulators in Chapters 4, 5, and 6.

2.1 Cloth Structure

The discussion in this section is meant to be a general introduction to cloth structure, in particular knits, and it is drawn in general from several sources [103, 99, 94].

Cloth is composed of yarns[1], which are themselves composed of fibers, either long filaments like silk or shorter fibers like cotton twisted so that friction holds the yarns together. This friction plays an important role in the overall behavior of the yarn [17], as it tends to inhibit relative movement of the fibers at the core of the yarn, and as a result the yarn as a whole resists stretching. Yarns can also be made of multiple plies, where each ply is a single filament or group of twisted staple fibers.

In general, cloth can be divided into three broad categories: felt, woven, and knit fabrics. Felt, or nonwoven, fabrics are formed by individual fibers matted and compressed together, with a largely chaotic and random entanglement be-

[1] common usage usually refers to thin, $<$ 1mm diameter yarns in most fabrics as ‘threads’ and thicker yarns used for hand-knitting as actual ‘yarns’; in the textile community, both of these are considered to be just yarns

tween the fibers leading to structural stability. In contrast, both woven and knit fabrics rely on regular and repeated patterns of entanglement between yarns. Woven fabrics are composed of two sets of yarns, the warp and the weft, organized into two perpendicular directions on the cloth surface. At each crossing of a warp and weft yarn, either the warp or the weft yarn will be on top, and different types of woven fabric will have differing patterns of yarn crossings, with the vast majority being a repeated tiling of some small (on the order of 6×6) pattern. Yarns can be woven quite densely; for example, bedsheets can commonly contain upwards of 300 yarns per linear inch.

In contrast, the yarns in a knitted fabric are organized into a regular set of loops. There are two types of knitting, weft and warp, with almost all hand-knitted materials and the majority of machine knitted materials using weft-knitting [99]. In weft-knitting, the yarn runs horizontally, with the loops from each horizontal row of a knit pulled through the loops of the previous row, either in a “knit” stitch (up through the previous loop) or a “purl” stitch (down through the previous loop). The two primary directions in a knit are called the *course* and the *wale*, with the course traveling in the direction of a single row of loops and the wale traveling in the direction of the stack of loops. Typically in weft-knitting, when the yarn reaches the end of a row of a knit it then doubles backs and forms the next row as well. Alternatively, for cylindrical shapes like hats and socks, the yarn at the end of one row can be seamlessly knit into the start of the row, forming the next row. As a result, weft-knits can consist of only a few yarns, which is in contrast to woven fabrics and warp-knits which consist of many yarns. The first and last row of stitches are special stitches known as cast-ons and bind-offs, respectively, which serve to keep the knit from unraveling; although there are several styles, in general they are looped through

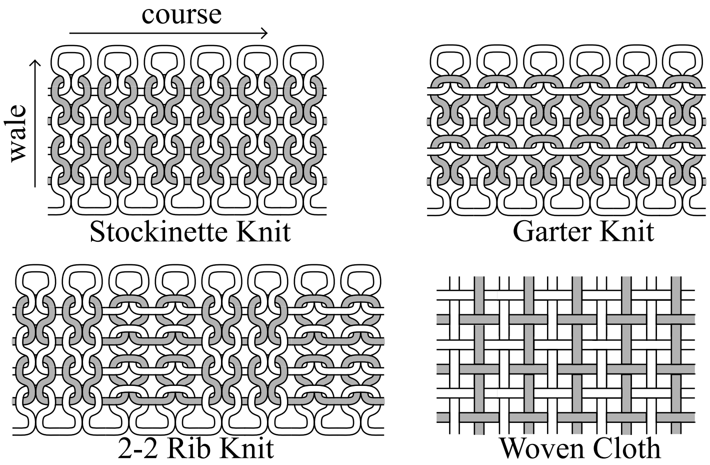


Figure 2.1: Interlocking loop structures of three knitting patterns and woven fabric.

their immediate neighbors in both the course and wale direction, as opposed to normal knit loops which are in general looped only through their immediate neighbors in the wale direction. The beginning and end of the yarn is either pulled back through the fabric several times and held in place by friction or simply knotted off.

2.1.1 Knit Geometry

Alternating between knit and purl stitches results in much of the variety in knitted fabrics, with three of the most common varieties being the stockinette (all "knit" stitches), the garter (alternating rows of "knits" and "purls"), and the 2-2 rib (each row consists of repetitions of 2 "knit" stitches followed by 2 "purl" stitches)[2]. Figure 2.1 shows samples of these three styles of knitting from above. The garter is the simplest of the three, and has the same overall pattern on both sides of the fabric. In comparison, the stockinette is different on the front and

[2] When hand knitting, the work is typically turned over after each row, which reverses the notation of stitches (i.e., a "purl" stitch when flipped looks like a "knit" stitch from the front). As a result, these definitions of a stockinette and a garter are reversed from standard hand-knitting definitions.

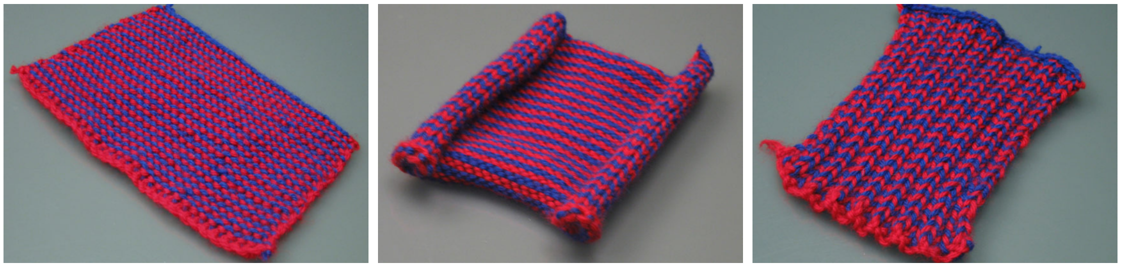
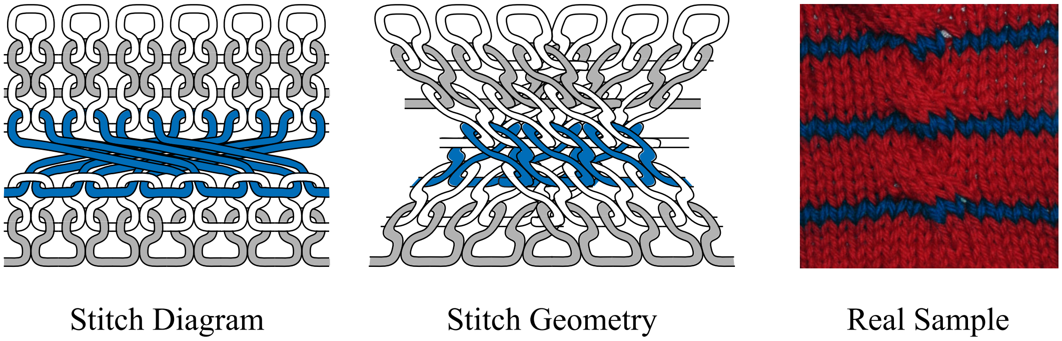


Figure 2.2: Views of three knitted samples.

the back, which leads to some dramatic curling behavior on the edges. The rib is much shorter in the course direction than either of the other two, again because of this same curling behavior. As can be seen from the pattern, it is essentially 2 columns of the front side of a stockinette followed by 2 columns of the back side of a stockinette. These columns curl like the regular stockinette, with adjacent columns curling in opposite directions, compressing the rib knit greatly and giving it a tremendous degree of stretchiness in the course direction; typically, the cuffs of shirts and sweaters and the ankles of socks are made out of ribbed stitching. Examples of these fabric are shown in overview in Figure 2.2.

There are a wide variety of stitches beyond knits and purls, however, limited only by the skill of the knitter or the capabilities of the machine creating the fabric [81, 71, 116]. For instance, increases / decreases change the number of loops per row of fabric, allowing the fabric to change size, in some instances in a hyperbolic fashion [106]. There are a wide variety of stitches which cause increases/decreases, some of which may add decorative holes or other interesting features in the final fabric. Stitches can be reordered in a given row as well, forming three dimensional features called cables to appear in the fabric. In a cable, certain rows contain a stitch interchange, where for instance the third, fourth, and fifth loops are pulled through the sixth, seventh, and eighth loops in the previous row and vice versa. This causes the fabric to curl over, forming the



Stitch Diagram

Stitch Geometry

Real Sample

Figure 2.3: Stitch diagram, geometry, and real sample of a cable stitch

appearance of two twisting columns of knit fabric. See Figure 2.3 for an example of a cable stitch, where the rows with loop interchanges are shown in blue. This complex assortment of possible stitches and topological structure means that knitting instructions can be quite complex, although an individual piece of clothing will typically only use a small subset of the overall space of stitches.

2.1.2 Constructing Knits

Knits can either be hand-made or machine-made. When constructed by hand, the knitter typically uses a small number of needles, with only two needles “active” at any given time (other needles hold currently unused stitches). For right handed knitting, the needle in the left hand holds the stitches from the previous row, while the right hand needle holds stitches in the current row. For the most basic type of knitting, knit and purl stitches, the right hand needle is slipped through the lastmost stitch on the left hand needle (from the previous row), the yarn is wrapped around the right hand needle, and then pulled through the old loop, producing a new loop through the previous stitch. The stitch on the left hand needle is then slipped off, leaving a new loop on the right hand needle. When the left hand needle has no more stitches, the two needles are exchanged and knitting proceeds to the next row, albeit from the opposite side of the work. More complicated knit stitches and patterns are created by extensions to this basic model, e.g. knitting through two stitches at once, slipping a stitch from left to right without knitting through it, etc.

In contrast, machine knitting typically consists of a large number of needles, with each needle holding a single stitch and designed such that a simple repeated motion will cause a new stitch to be created and the old stitch simultaneously to be slid off the needle. Machine knitting can reproduce most hand-knitting features in a significantly faster time and at sizes generally too fine for hand-needle work. See [99] for an in-depth discussion of knitting techniques, particularly machine knitting.

2.2 Cloth Dynamics

As a result of its construction, the deformations of cloth are multiphasic, particularly when being stretched. Duhovic and Bhattacharyya [28] contains a discussion on the various modes of yarn deformation and where in the force-displacement curve they are most prominent. When under tension, the cloth first begins unrolling from any compression caused by curling. This is particularly evident in ribbed knits, where the columns of the front-facing stockinette stitch are pulled apart, revealing the columns of back-facing stockinette stitch. After that, the cloth then begins deforming its woven or knit structure. In the case of a woven fabric, the warp-weft intersections become compressed, while in a knit fabric the loops are stretched in one dimension while being compressed in the other. Because the loops are typically free to undergo much larger deformations than the compression of the intersections in a woven fabric, knits tend to be much stretchier than their woven counterparts. At some point, however, the cloth is unable to deform much more in this fashion and so additional load causes the yarns themselves to stretch. As noted above, though, yarns are very resistant to stretching, which results in a sharp increase in the slope of the force-displacement curve at this point.

It is also important to note that stretching behavior in one dimension affects the characteristics of the other dimension as well. For instance, in a knit, as the loops are stretched in one dimension they compress in the other dimension, sometimes quite noticeably; as a result, capturing these couplings is important for visual accuracy. Although some current cloth simulators are capable of expressing these types of relationships, it can be difficult to tune the parameters correctly, and oftentimes they are ignored.

Because of the complex internal structure of cloth composed of yarns which are themselves composed of fibers, all of which is highly contact-mediated, approximations to the dynamics are typically required for tractability, with corresponding reductions in the accuracy of the result. The following chapter discusses various previous approaches to dealing with this complex material, and where and how they deviate from capturing the true mechanical properties of cloth.

CHAPTER 3

PRIOR WORK

Research in accurate cloth simulation cuts across at least three major disciplines — computer graphics and simulation, textile manufacturing, and engineering applications — each with their own notions of accuracy and speed required. This section presents an overview of related work in cloth modeling and simulation in all three of these domains. Section 3.1 discusses various models for cloth and yarn simulation at both the macro- and meso-scale, as well as other models and techniques which are used to accelerate these approaches. Section 3.2, meanwhile, discusses methods for resolving contact in simulation, as this will be the main cost of simulation for yarn-based models.

3.1 Models

3.1.1 Elastic Sheets

Cloth has been modeled in a variety of different ways in the literature. Perhaps the most straightforward approach, and the one primarily used in the computer graphics community, is to treat the cloth as an elastic sheet, typically one that is linearly elastic and isotropic. These models are either explicitly continuous [107] or a discrete approximation to some continuous surface [4], with the latter becoming the dominant approach over time for the relative simplicity in its implementation. The cloth is approximated as a mesh, with the degrees of freedom the vertices of the mesh and mass typically lumped at the vertices. Depending on the underlying model, this mesh may be enforced to be a rectilinear grid [23, 39] or a general triangular mesh [4, 43]. Forces are then defined at each vertex which locally resist stretching in the the two primary directions, along

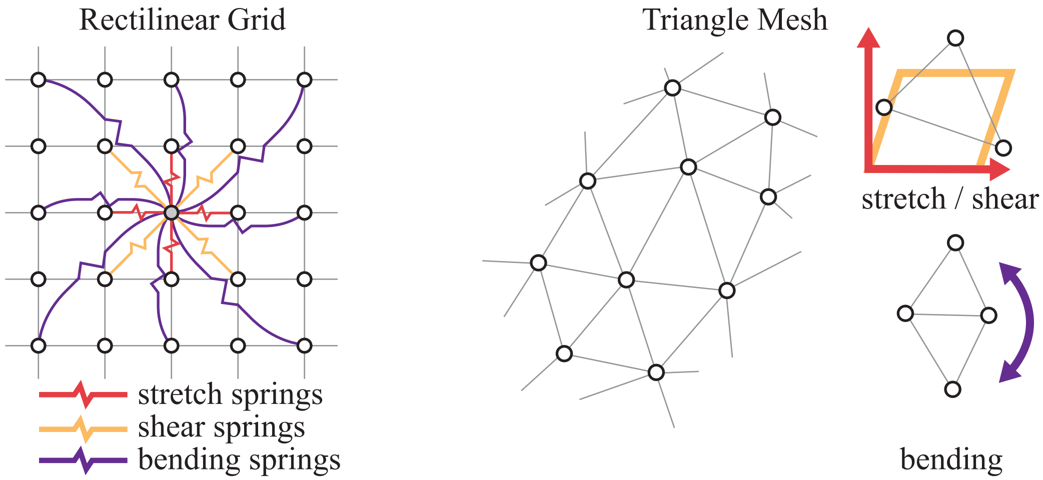


Figure 3.1: Rectilinear and triangular mesh cloth models.

with shearing and bending; these forces can either be generated from explicit springs between the vertices of the mesh or from forces defined as a result of the deformation of each triangle from its rest configuration. Figure 3.1 shows a diagram of the model for both rectilinear grids (using springs) and triangular meshes. Finally, damping is added, again typically as a linear damping of the velocity in the direction of each mode (stretching, shearing, bending) [4]. For the common case of linear elastic forces, this results in a single stiffness and damping parameter for each of the four penalized deformation modes.

Extensions to these models have focused on speeding up the computation time [114], simulating stable behavior under compression [23], revised models of bending [15, 43], or stable collision processing (see Section 3.2.1). Focus has also gone into limiting the amount of stretching fabric can undergo, either by a strain-limiting iterative process [90, 16, 118], a constraint satisfaction phase [39], or nonconforming elements [32]. While simulation speeds are relatively fast, there is in general a problem of mapping physical cloth properties to the parameter space of the elastic model. Jojic and Huang [59] used range scan data of static cloth configurations to estimate the elastic parameters. Bhat et al. [11]

used video data of moving cloth to estimate the elastic and damping parameters, with the experimentally determined parameters for the knit sample varying noticeably; this suggests that the elastic model may not be a good fit for knitted materials.

Due to the internal structure of cloth, however, these linear forces can be a poor match for the true behavior of cloth. More recent work has looked at accurately simulating the nonlinear forces that occur in fabric, derived from measured stress-strain relationships [113]. This can work well for capturing global deformations; however, it requires the force curve for each material to be known or measured in advance. In addition, if the deformation of threads is visually noticeable and nonlinear (such as for a ribbed knit being stretched) they must be modeled separately, since the model only deforms at the mesh element level. Yeoman et al. [121] adapted a nonlinear model used for soft-tissue to the case of knits and used a genetic algorithm to fit model parameters to measured fabrics.

3.1.2 Rods

Modeling yarns explicitly typically relies on some underlying model for thin flexible rods. These have been studied extensively in computer graphics in recent years [82, 10, 42, 109, 101]. The rod is typically modeled as a centerline plus material frame, which is needed to compute bending and twisting energies in the rod, particularly in non-straight rest configurations. Bergou et al. [9] defined an efficient discretized form of thin rods that supports an anisotropic bending response and non-straight reference configurations. This was followed up by additional work which further improved the efficiency of the underlying material frame representation [7], developed concurrently with the description in

Section 4.5.1. Contacts between rods can be resolved precisely using inequality constraints [102], but this does not allow for lateral compression of soft yarns as penalty-based models do, requiring explicit modeling via additional degrees of freedom. More recent work has looked at a unified representation of elastic materials in one-, two-, and three-dimensions using an element which allows for measurement of elastic deformations along any axis [69].

In the textile community, research has also looked at fiber-level models of yarns and how the interactions at the fiber level lead to yarn-level behavior, particularly bending rigidity [83, 84]. Notably, Choi and Tandon [22] develop a model of a multi-ply yarn, showing that their model reasonably approximates experimental results and predicts the strain energy of bending to be approximately quadratic with respect to the curvature. More recently, the torsional characteristics of yarns, particularly multi-ply yarns, was examined by Phillips et al. [88].

3.1.3 Yarn-Based Cloth

Several models have attempted to address the fact that cloth is comprised of a discrete set of yarns. Geometric modeling of yarns arguably began with Peirce [86], who derived a set of parameters and equations for modeling the crossing of yarns in a woven fabric as inextensible curves. Kawabata et al. [62] proposed a beam and truss model for yarn crossings in a woven fabric, as well as a system for measuring the physical force curves resulting from stretch, shear, and bend motions of cloth. Further work has extended this model to support analysis of non-plain-weave fabrics [89]. Variations of the beam-and-truss model have been used in the textile community to simulate the behavior of plain-weave fabrics like Kevlar [122]. The beam-and-truss model has also been adapted to model a unit cell of the fabric, which can then be used to drive the behavior of finite elements in a continuous sheet [66, 85]. Similar to the beam and truss model is the rigid bodies of Xiao and Geng [120], where the yarn crossings are treated as inextensible rigid lines and forces are defined to connect neighboring yarn crossings in a plain weave.

Other work has looked at approximating the behavior of woven fabric via specialized elements. Zhang and Fu [124, 125] considered a woven cell of fabric to be a hinged square truss and considered the buckling due to shear in both the warp-parallel and diagonal directions; further work considered shear in arbitrary directions [126]. Peng and Cao [87] convected a non-orthogonal coordinate system corresponding to the warp and weft directions of woven fabric and used it to derive the mechanics of thin shell elements; it relies on the decoupling between stretching and shear forces which is in general true for low shearing angles. Hamila and Boisse [50] used a semi-discrete triangular finite element that interpolates a discrete set of yarns in a defined material frame to compute stretching and in-plane shear response; it does not, however, consider the bending response or the effect of yarn crimping, although later work considers bending forces [49].

The system for measuring cloth was later formalized into the Kawabata Evaluation System (KES), and is commonly used to extract cloth properties from samples [61]. The measurements for a particular fabric can then be fed into a simulator, e.g. a mass-spring system [56], in order to replicate the behavior of the cloth. However, as noted earlier, this is limited to fabrics for which a physical sample exists and has been tested in the KES evaluation system. Bridging the gap between elastic sheets and textiles, models have also been developed for testing and predicting the Poisson ratio of both woven [105] and knit [58] fabrics.

In computer graphics, Breen et al. [13] and Eberhardt et al. [31] modeled woven fabric as a particle system, where a particle ideally represented a yarn crossing. Metaaphanon et al. [72] modeled cloth as a sheet until it is sufficiently stretched, at which point a yarn model is substituted in; however, this yarn model is meant to create physically realistic frayed edges when the cloth tears and not as a fundamental source of cloth behavior.

Woven yarn crossings have also been modeled as a pair of curves [119]. Nadler et al. [78] employed a two-scale model, treating cloth at the high level as a continuous sheet, and at the fine level as a collection of pairs of orthogonal curves in contact with each other, with feedback from the fine scale driving the simulation of the large scale. Yarns have also been modeled as splines, with Rémion [93] developing the basic equations for using splines in a knit; however, they used springs between the control points to preserve length. Jiang and Chen [57] used a spline-based yarn model to generate plausible static woven fabric configurations. Other simulations of woven materials for ballistic impacts include finite element modeling of the yarns as hexahedral elements [45, 123].

Some work has examined woven materials at extremely high levels of detail. Durville [29] modeled woven fabrics at fiber level, with models consisting of up to 400 fibers. Other work has modeled yarns as elastic materials using highly detailed finite elements, simulating a representative cell of plain-woven fabric [3, 67].

The work of Chu [24] is similar to the model presented in Chapter 4, in that both use B-splines to simulate fabric with similar terms for collisions, but the former is focused on woven fabrics and allows the yarns to stretch, which requires a much smaller timestep for stable results. In addition, the integral for the contact model is approximated by using closest points in a set of contacts predetermined by the cloth structure at initialization time.

Because of their relative complexity compared to woven fabrics, knits are not as well-studied, although they have been increasingly so in recent years. Wada et al. [115] used a geometric model for loop deformation, assuming uniform loops, and modeled contacts between loops using springs. Eberhardt et al. [30] modeled knits as a continuous sheet with force curves derived from Kawabata measurements. Nocent et al. [80] deformed a thin sheet and projected the changes to the sheet to an underlying spline-based geometry. Several works in the textile community have focused on generating knit geometry using spline curves, typically assuming incompressible yarns and specific geometric constraints [27, 38, 20, 21]; only Choi [20, 21] attempted to simulate the resulting geometry. Chen et al. [19] are primarily concerned with rendering knit geometry, and used as their base model a system of key points mapped to a mass-spring mesh.

Similar to the fiber-level simulations of woven materials, Duhovic and Bhattacharyya [28] performed simulation and analysis of small-scale knitted structures created out of fibers grouped into yarns. In addition, the overall energy contribution from deformation was broken up into individual contributions from bending, twisting, and stretching the fibers, which were then validated against real samples. Other work has performed detailed FEM analysis of the behavior of a representative cell of plain-knit geometry [47, 110].

3.1.4 Corotational

Corotational finite-element methods are commonly used in graphics for solid deformation. Müller et al. [75] popularized these techniques in graphics, initially via node-based “stiffness warping,” then, to overcome undesirable “ghost forces” due to element-level momentum imbalances, using rotated linear elements [74] (c.f. [35]). Corotational elements have also been used for sheet-based cloth simulation [33]. Shape-matching methods also use rotated frames to estimate “goal positions” for deformation forces [77, 95, 104].

3.1.5 Model Reduction

Model reduction techniques have been devised to accelerate sheet-based cloth and other deformable models in graphics by reducing the number of simulated degrees of freedom. Effective methods have been proposed for spatially adaptive mass-spring systems [54], sheet-based cloth models [111], and rod simulations that resolve challenging contact configurations such as knot tying [102]. For articulated rigid bodies, research has looked at adaptively selecting the joints to simulate in order to speed up the simulation [92]. Space-time adaptive simulation of deformable models can resolve localized contacts efficiently for real-time simulation [26]; and multi-scale basis refinement can reduce meshing issues for spatial adaptation [44]. Unfortunately, multi-resolution/adaptive approximations are difficult for knitted cloth given its complex geometric domain and topology, high number of degrees of freedom, and widespread self-contact.

Homogenenization techniques have been proposed to coarsen discrete simulation models while resolving inhomogeneous material response [64], and to support deformation of complex embedded geometry [79], but neither addresses fine-scale internal forces which are contact mediated. Dimensional model reduction techniques have been proposed to generate fast, low-rank simulation models for complex geometric domains [6, 1] and thin shells [18], but knitted cloth motions do not necessarily lie in low-dimensional subspaces, and specifying them *a priori* for precomputation purposes would be impractical. Recently, Kim and James [65] showed how to adapt subspace deformation models on the fly to avoid precomputation; however, they do not address contact forces.

Although they do not fall into the classical notion of model reduction, in recent years research has also looked at adding detail to a low-resolution simulation, usually as a post-processing step. TRACKS [8] takes as input a low resolution mesh, produced from either simulation or user animation, and simulates a high resolution deformable model that tracks the overall motion of the low resolution mesh while allowing fine-scale wrinkling. Data-driven approaches have also been used, synthesizing wrinkles generated from a database of high resolution meshes simulated using either independent character joint rotations [117] or similar character motion [36]. Rohmer et al. [96] generated a set of plausible wrinkle curves as a post-processing step, evolving the curves through time in a temporally coherent manner and deforming the high resolution mesh accordingly. Although these methods add additional detail to a coarse simulation, the details are in most cases physically plausible but not necessarily physically accurate. Moreover, the overall motion depends heavily on the motion of the low resolution mesh and, thus, on its accuracy, which as discussed may not be very good for knits.

3.2 Contact and Collision

3.2.1 Deformable Meshes

Resolving cloth self-contact is incredibly important for robustness, and as a result much research has gone to efficiently and correctly responding to self-intersection. Detection of interpenetration is commonly performed using overlap tests accelerated by spatial subdivisions, hash tables, or bounding volume hierarchies [108]. In addition, continuous contact detection is employed to detect contacts that occur during the timestep, preventing the mesh from interpenetrating [16]. Interpenetrations are usually corrected with impulse forces, with as a last resort contacts grouped into impact zones and their motion resolved either rigidly [91, 16] or via inelastic projection which allows for relative sliding [52]. Selle et al. [98] used a mass-spring system with strain limiting to simulate hair and a similar application of impulses to resolve contact. As they note, however, any hair state is innately intersection-free, and as a result they can ignore certain contacts, which is not possible in yarn-based cloth since it would lead to unravelling.

Further work has advanced the speed of cloth self-collision; however, many of these schemes are not applicable to yarn-based models since they are inherently sheet-based, e.g., curvature tests [112], normal cones [91], and chromatic decompositions [41]. As a different approach, Baraff et al. [5] considered interpenetration to be an inevitable result of interacting with user-animated characters (such as pinching in shoulders and elbows), and instead determined areas of the mesh which have interpenetrated and generated forces which smoothly resolve the interpentration when possible. Even in the case of sheet-based cloth, though, there are scenarios where it is difficult to determine the force needed to resolve the interpenetration. Moreover, it relies on the sheet-based nature of cloth to determine intersecting areas and so is not directly applicable to discrete yarn models.

3.2.2 Generalized Contact

Identifying and tracking persistent contacts is related to space-time scheduling and collision processing [73, 46], which have been widely considered for maintaining proximity information for moving objects [53, 37]. Mirtich's Timewarp method [73] investigated related strategies for collision detection and management of persistent contact groups for asychronous rigid-body simulation. For deformable models, Harmon et al. [51] used asynchronous contact mechanics and infinitely nested potentials to adaptively simulate penalty-based contact; however, their emphasis was on correctness for general scenarios, and not performance for hundreds of thousands of persistent yarn contacts.

CHAPTER 4

YARN MODEL

4.1 Overview

As opposed to sheet-based cloth simulation, which models cloth as a continuous surface, yarn-based cloth simulation explicitly models the individual yarns that comprise the cloth. The number of yarns can vary dramatically depending on the material and the garment constructed, with weft-knits typically containing 1–10 yarns and wovens and warp-knits containing hundreds to thousands. Without loss of generality, assume in the following sections that the cloth models are constructed using a single yarn.

Sections 4.2 and 4.3 discuss the original yarn model, while Section 4.4 validates and analyzes the simulation results. Section 4.5 details further improvements to this model that increase the speed and quality of results. Finally, Section 4.6 draws conclusions from this approach and discusses how further speed improvements can be achieved via the techniques in Chapter 5 and, more importantly, Chapter 6.

4.2 Rod Model

The yarn is an open cubic B-spline curve containing n segments with a constant radius of r, described by the control points $\mathbf{q} \in \mathbb{R}^{3(n+3)}$. In general, indices i, j range over spline segments, while indices k, l range over control points. The curve is described by $\mathbf{y}(s) = \sum b_k(s)\mathbf{q}_k$, $s \in [0, n]$, where $b_k(s)$ is the cubic B-spline basis function associated with control point k. Similarly, the velocity of the yarn at parametric point s is $\mathbf{v}(s) = \sum b_k(s)\dot{\mathbf{q}}_k$. For convenience, the curve restricted to a particular spline segment i is denoted $\mathbf{y}_i(s)$, $s \in [0,1]$ (and $\mathbf{v}_i(s)$ for the velocity). Each spline segment has a fixed arclength ℓ_i. The yarn has a mass per unit length of m_{unit}, and mass is spread along the curve according to the function $m(s) = m_{\text{unit}}\ell_{\lfloor s \rfloor}$, a piecewise constant function that assigns mass to segments according to their arclength and then spreads the mass uniformly in parameter space.

The yarn's time evolution is modeled using the equations of motion of constrained Lagrangian dynamics; Goldstein et al. [40] have a further description of Lagrangian mechanics, while Rémion et al. [93] apply it to spline curves. In addition, some of the stiffer properties of the cloth are enforced via holonomic constraints, which are algebraic conditions on the geometry that must be true at every timestep. The result is a differential algebraic equation (DAE) [14, 2, 48] of the form,

$$\begin{aligned} \mathbf{M}\,\ddot{\mathbf{q}} &= -\nabla_{\mathbf{q}} E(\mathbf{q}) - \nabla_{\dot{\mathbf{q}}} D(\dot{\mathbf{q}}) + \mathbf{f} && (4.1) \\ \mathbf{C}(\mathbf{q}) &= \mathbf{0}, && (4.2) \end{aligned}$$

where $\mathbf{M}$ is the mass matrix, $E(\mathbf{q})$ is the sum of all positional energy terms, $D(\dot{\mathbf{q}})$ is the sum of all damping energy terms, $\mathbf{f}$ are external forces, and $\mathbf{C}(\mathbf{q})$ is a vector of constraint functions. These terms are expanded in the following subsections; see Figure 4.1 for a high-level view of the terms involved.

4.2.1 Internal Forces

The kinetic energy of the yarn is:

$$T(\dot{\mathbf{q}}) = \sum_{i=0}^{n-1} m_{\text{unit}}\,\ell_i \int_0^1 \mathbf{v}_i(s)^T \mathbf{v}_i(s)\; ds \qquad (4.3)$$

In order to apply Lagrangian mechanics, $\frac{d}{dt}\left(\nabla_{\dot{\mathbf{q}}} T(\dot{\mathbf{q}})\right)$ must be computed. By expanding the integral on the right hand side of Equation 4.3, $\nabla_{\dot{\mathbf{q}}} T(\dot{\mathbf{q}})$ can be

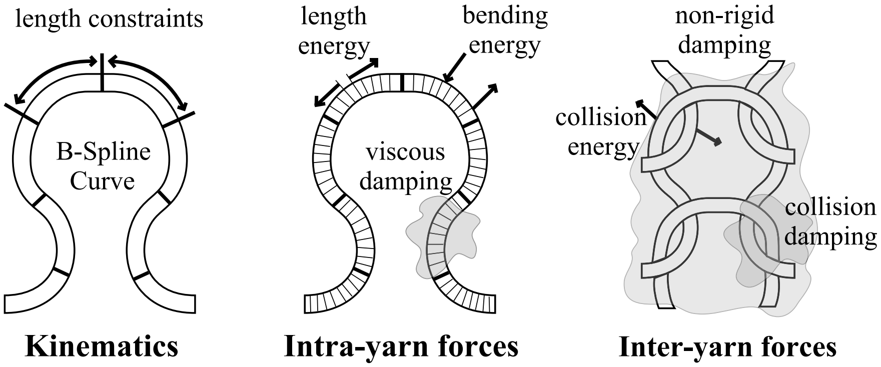


Figure 4.1: Summary of yarn-level model

rewritten as $\mathbf{M}\dot{\mathbf{q}}$, where

$$\mathbf{M}_{k,l} = \int_0^N m(s) b_k(s) b_l(s) ds.$$

Because this depends only on the arclengths, m_{unit}, and basis functions, all of which remain constant during simulation, this matrix can be precomputed, and because the cubic B-spline functions have local support, the matrix is sparse with upper and lower bandwidth of 12. Taking the derivative with respect to t yields $\mathbf{M}\ddot{\mathbf{q}}$, the left hand side of (4.1)

Bending resistance is modeled by a bend energy density functional which is quadratic in curvature:

$$E_i^{\text{bend}} = k_{\text{bend}}\, \ell_i \int_0^1 \kappa_i(s)^2 \; ds, \tag{4.4}$$

where $\kappa_i(s) = \frac{\|\mathbf{y}_i'(s) \times \mathbf{y}_i''(s)\|}{\|\mathbf{y}_i'(s)\|^3}$ is the curvature of spline segment i at s.

Because of their high resistance to stretching relative to the cloth, yarns are modeled as inextensible rods. Ideally this would be a constraint at the infinitesimal level; however, since there are only a finite number of degrees of freedom, applying the constraint at too fine a level would result in locking behavior. To avoid this, this model defines one length constraint for each spline segment

$$C_i^{\text{len}} = 1 - \frac{1}{\ell_i} \int_0^1 \|\mathbf{y}_i'(s)\| \; ds. \tag{4.5}$$

This constraint ensures that the total length of the segment remains constant, but it does not necessarily keep the mass of the spline from sliding around inside the curve as the parameterization speed changes; as long as the overall length is constant, the infinitesimal length can change without penalty. This can be prevented with an additional energy term which resists sliding at an infinitesimal level:

$$E_i^{\text{len}} = k_{\text{len}} \int_0^1 \left(1 - \frac{\|\mathbf{y}_i'(s)\|}{\ell_i}\right)^2 ds, \tag{4.6}$$

where k_{len} is a stiffness coefficient. It should be noted that this term does not have to be particularly stiff due to the use of length constraints, since it only needs to resist the stretching or compression of mass in a local area. For instance, in a piece of yarn hanging vertically, k_{len} only needs to be stiff enough to support the weight of a single spline segment, while without the constraint term k_{len} would need to be stiff enough so that the first segment could support the weight of the *entire* yarn.

4.2.2 Self-Contact Forces

Yarn-yarn collision forces are modeled with an energy term,

$$E_{(i,j)}^{\text{contact}} = k_{\text{contact}}\, \ell_i \ell_j \int_0^1 \int_0^1 f\left(\frac{\|\mathbf{y}_j(s') - \mathbf{y}_i(s)\|}{2r}\right) ds\, ds', \tag{4.7}$$

for i, j such that $i \neq j$, where $f(d)$ is defined such that $f(d) \to 0$ as $d \to 1$, $f'(d) \to 0$ as $d \to 1$, and $f(d) \to \infty$ as $d \to 0$. Ideally, this function should capture the behavior of the yarn compression forces, but these are a complicated function of yarn type and local fiber interactions and are in general difficult to measure exactly. However, acceptable results are achievable with simpler choices for this function, such as the one used in this model:

$$f(d) = \left\{ \begin{array}{ll} \frac{1}{d^2} + d^2 - 2, & d < 1 \\ 0, & \text{otherwise} \end{array} \right\}. \tag{4.8}$$

In practice, this collision model tends to be physically and computationally more robust than ones based on closest point distances while by definition also automatically handling arbitrary cloth self-collisions such as those seen in folding and bunching. As a result, the same force model is used to resolve both structural contacts, i.e. the persistent contacts that are due to the regular looping nature of the knit, as well as transient contacts, i.e. those arising temporarily due to the particular configuration of the cloth. In addition, yarns are typically relatively soft and can be laterally compressed, which this model allows for without any additional degrees of freedom, unlike other models based on hard interpentration constraints. Note also that it is symmetric, so if the contact force is computed between segments i and j, it is obviously the same as the force between segments j and i

4.2.3 Dissipative Forces

Damping and friction in knitted cloth structures are complex, with significant hysteresis effects. The interlooped structure of knits creates large numbers of interlinked contact regions, and for yarns made of short fibers the direct contact between yarns creates entanglements between the mass of intertwined stray fibers, or "fuzz," which resist relative motion between nearby yarns. The basic effect of these interactions is to rapidly damp out oscillations and deformations, one of the notable ways in which cloth differs from elastic sheets. In addition, these frictional forces are one of the prime sources of hysteresis in cloth. Although a full treatment of these interactions remains an interesting and difficult open question, in practice the following three damping models have proven themselves to be effective at achieving similar results (see also §4.5.2 for further improvements including hysteresis).

Mass-proportional damping

This is a classic way to dissipate any motion. Damping is applied uniformly to the yarn according to the following damping energy term:

$$D_i^{\text{global}} = k_{\text{global}} \int_0^1 \mathbf{v}_i(s)^T \mathbf{v}_i(s) ds. \tag{4.9}$$

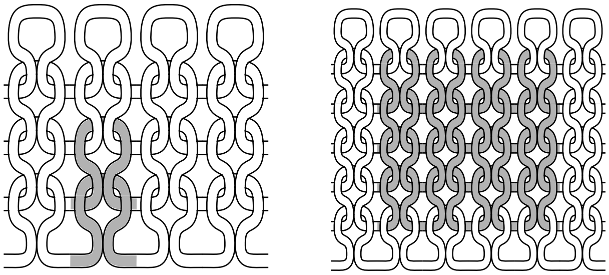
Because the density of the yarns is constant, the mass dependence is effectively pushed into k_{global}. This model is particularly effective during knit structure initialization (Chapter 7), as it allows the material to stably settle into its rest configuration. However, during actual simulation, excessive mass-proportional damping creates an overwhelming and undesirable sense of "underwater" cloth, and as a result k_{global} is typically turned off after initialization, with the following two damping models instead used to damp the motion of the cloth.

Yarn-Yarn Contact Damping

A yarn-yarn collision damping term $D_{(i,j)}^{\text{collision}}$ is used both to damp the stiff yarn-yarn contact forces and to approximate sliding friction, defined as:

$$D_{(i,j)}^{\text{collision}} = \ell_i \ell_j \int_0^1 \int_0^1 c_{ij}(s, s') \left(k_{dt} \|\Delta \mathbf{v}_{ij}\|^2 - (k_{dt} - k_{dn})(\hat{\mathbf{n}}_{ij}^T \Delta \mathbf{v}_{ij})^2\right) ds\, ds', \tag{4.10}$$

where $k_{dt} \geq 0$ controls damping in the tangential direction, and $k_{dn} \geq 0$ controls damping in the normal direction; $\Delta \mathbf{v}_{ij} = \Delta \mathbf{v}_{ij}(s, s') = \mathbf{v}_j(s') - \mathbf{v}_i(s)$ is the relative velocity; $c_{ij}(s, s') = 1$ if points s and s' are in contact according to Equation (4.8) and 0 otherwise; and $\hat{\mathbf{n}}_{ij} = \hat{\mathbf{n}}_{ij}(s, s')$ is the normalized value of the collision direction, $\mathbf{n}_{ij}(s, s') = \mathbf{y}_j(s') - \mathbf{y}_i(s)$.



Small rigid damping region (repeat every row / column)

Large rigid damping region (repeat every 2 rows / 2 columns)

Figure 4.2: Regions used for the non-rigid damping velocity filter

Non-rigid motion damping:

As noted earlier, accounting for the dissipative effects of "fuzz" properly is a rather difficult problem. Modelling it explicitly is challenging due to the large number of fiber interactions; however, ignoring it entirely is not an option either since it contributes to the high degree of damping in cloth while deforming. For the time being, this model uses a simple approach that works well in practice, but improvements in this area is a topic for further research.

In order to resist relative motion between nearby sections of yarn, non-rigid motions [76] within the cloth are damped. The cloth is broken up into fixed overlapping regions as in Rivers and James [95], and at each timestep the center of mass, angular momentum, and inertia tensor of each region are computed, which allows the computation of the expected rigid velocity $\mathbf{v}_{rigid}(s)$ of each point in the region. The yarn in each region is then damped according to

$$\mathbf{d}_{\text{rigid}}(s) = -\frac{\alpha m(s)}{hr(s)}(\mathbf{v}(s) - \mathbf{v}_{\text{rigid}}(s)), \tag{4.11}$$

where the damping response is more naturally expressed as an explicit force instead of the gradient of an associated damping function. The parameter $\alpha \in [0, 1]$ controls how strong the damping is, and $r(s)$ is the number of regions containing the point s. In practice, this is not treated as an energy term, but is instead integrated using a velocity filter (see Section 4.3.3).

There are several ways to break up the cloth into regions. In the current simulator, nonrigid damping is applied as a two-pass filter over parametrically static regions defined during yarn initialization, first heavily damping small regions of two yarn loops and then damping the motion of larger regions (see Figure 4.2). The first pass is designed to damp out motion locally where the yarns loop around each other, and the second pass damps stretching, shearing, and bending modes.

4.2.4 External Forces

Gravity is simple to treat as an energy potential:

$$E_i^{\text{grav}} = \int_0^1 m(s)\mathbf{g}^T\mathbf{y}_i(s)ds. \tag{4.12}$$

where g is the direction and magnitude of gravitational acceleration.

In order to prevent knitted cloths from unraveling in real knitted fabrics, the end of the yarn is typically either knotted off or pulled through several loops and held in place by friction. The same effect is accomplished in this model by “gluing” the end of the yarn to another piece of yarn via a constraint of the form $\mathbf{C}^{\text{glue}} = \mathbf{y}(s_1) - \mathbf{y}(s_2)$ for particular choices of s_1 and s_2. Similarly, when the cloth needs to be pinned in place, vector constraints of the form $\mathbf{C}_i^{\text{pin}} = \mathbf{y}(s_i) - \mathbf{p}_i$ are inserted. Like for the length constraints, it is important to avoid introducing too many hard constraints, which can lead to overconstrained or near-singular systems or degraded quality in yarn dynamics. This can be done by combining a hard pin constraint applied every few spline segments with energy terms of the form $k_{\text{pin}}(\mathbf{C}_i^{\text{pin}})^T(\mathbf{C}_i^{\text{pin}})$ applied at yarn points between the hard constraints.

Object contacts are handled in two different ways, depending on the way the object is defined. For objects defined over an implicit surface (for example, the leg in Figure 4.9), contact is handled as a penalty force modeled via an energy term:

$$\mathbf{E}_i^{\text{obj}} = k_{\text{obj}} \int_0^1 \begin{Bmatrix} (f(\mathbf{y}_i(s)) - f_0)^2, & f(\mathbf{y}_i(s)) < f_0 \\ 0, & \text{otherwise} \end{Bmatrix} ds, \tag{4.13}$$

where $f(\mathbf{y})$ is the function for the implicit surface and f_0 is the desired isolevel. Corresponding damping forces are also included in the contact evaluation (analogous to yarn-yarn collision damping (4.10)). Alternately, for objects with distance fields (such as the scarf falling on a plane), a force can be calculated which will bring the yarn to the surface of the object as well as applying an approximate friction response. In practice, instead of being computed as an explicit force it is instead computed as a velocity filter (Section 4.3.3, Bridson et al. [16]).

4.3 ODE Integration

Implementing this yarn-level model requires careful choice of simulation methods and attention to several crucial details in evaluating the terms presented in Section 4.2. Figure 4.3 contains an overview of the steps in the simulator. See also Section 4.5 for further improvements to the model and integrator.

4.3.1 Integration Method

The DAE is stepped forward in time using the Fast Projection method [39], using Algorithm 1 in that paper. An explicit midpoint step is used as the unconstrained step. The algorithm iterates until convergence, with each iteration requiring a sparse linear system solve of a matrix which depends on the inverse mass matrix and the Jacobian of the constraint function. To speed up the simulation and simplify inverse mass computation, the integrator lumps the mass

```
For each timestep, $h$
    $[\mathbf{q}, \mathbf{v}]$ = unconstrained_step($\mathbf{q}, \mathbf{v}, t$)
    $[\mathbf{q}, \mathbf{v}]$ = satisfy_constraints($\mathbf{q}, \mathbf{v}$)
    $\mathbf{v}$      = filter_velocity($\mathbf{q}, \mathbf{v}$)
    $t$      = $t + h$
end
```

Figure 4.3: Overview of initial time-stepping scheme

matrix along the diagonal. The matrix system itself is solved using Preconditioned Conjugate Gradients (PCG) with a diagonal preconditioner. Due to the small timesteps, at most 4-5 iterations were usually needed for acceptable convergence, and oftentimes only 1 or 2.

Although the integrals for the mass matrix, gravity, and global damping are readily solved, the others lack efficient closed forms. As a result, the integrals were discretized using Simpson's quadrature at fixed positions in parameter space, with (4.4), (4.5), (4.6), (4.7), (4.10), and (4.13) all computed typically using 11 quadrature points per spline segment.

4.3.2 Yarn Collisions

In practice, expanding the integrals in (4.7) and (4.10) is the bottleneck for the simulation, so they must be computed efficiently. Naive evaluation is exceedingly slow, because it involves a double integral over the entire yarn. However, it should be noted that the integrand is zero over the vast majority of the integration domain, since yarn segments typically contact only a few neighbors. To compute this integral effectively, the simulator uses spatial culling: bounding spheres are generated at fixed quadrature points in parameter space with a radius equal to the radius of the yarn and then inserted into an AABB hierarchy generated at the beginning of the simulation. The integral is then evaluated by first updating the spatial bounds of all nodes in the hierarchy, and then intersecting the hierarchy with itself to determine the quadrature point-pairs requiring evaluation. For additional speed, all of the force computations can be computed in parallel, as well as being parallelized themselves; this has the most practical importance for the AABB tree traversal to find self-contacts and computing object contacts.

Further speeding up the evaluation of yarn-yarn collisions is the focus of Chapter 6.

4.3.3 Velocity Filters

The simulator also allows for velocity filters to update control point velocities directly. Most previous velocity filters are used on discrete particle systems; however, here the yarns are instead modeled as a continuous curve. In order to filter the spline's control points, the curve is sampled, typically using 6–10 sample points per spline segment. A desired impulse $\Delta\mathbf{v}(s)$ is computed for each sample point, with $b_k(s)\Delta\mathbf{v}(s)$ the resultant impulse applied to the k^{th} control point. All of the control point impulses are accumulated and then multiplied by the lumped $\mathbf{M}^{-1}$ to produce the actual change in velocity $\Delta\dot{\mathbf{q}}$ for the control points. Finally, $\dot{\mathbf{q}}_{\text{new}} = \dot{\mathbf{q}} + \Delta\dot{\mathbf{q}}$. To prevent impulses from affecting each other, all impulses for a particular velocity filter are computed first, then applied together. The non-rigid damping and non-penalty-based object collisions (for objects with distance fields) are both handled with a velocity filter.

	Relaxation	Simulation
r	0.125 cm	0.125 cm
m	0.006 $\frac{\mathrm{g}}{\mathrm{cm}}$	0.006 $\frac{\mathrm{g}}{\mathrm{cm}}$
k_{len}	10000 $\frac{\mathrm{g\,cm}^2}{\mathrm{s}^2}$	2000 $\frac{\mathrm{g\,cm}^2}{\mathrm{s}^2}$
k_{bend}	0.005 $\frac{\mathrm{g\,cm}^2}{\mathrm{s}^2}$	5 $\frac{\mathrm{g\,cm}^2}{\mathrm{s}^2}$
k_{global}	1.5 $\frac{\mathrm{g}}{\mathrm{s}}$	0 $\frac{\mathrm{g}}{\mathrm{s}}$
k_{contact}	3250 $\frac{\mathrm{g}}{\mathrm{s}^2}$	3250 $\frac{\mathrm{g}}{\mathrm{s}^2}$
k_{dt}	0.001–0.005 $\frac{\mathrm{g}}{\mathrm{cm}^2\mathrm{s}}$	0.001–0.005 $\frac{\mathrm{g}}{\mathrm{cm}^2\mathrm{s}}$
k_{dn}	0.01–0.05 $\frac{\mathrm{g}}{\mathrm{cm}^2\mathrm{s}}$	0.01–0.05 $\frac{\mathrm{g}}{\mathrm{cm}^2\mathrm{s}}$
α_{small}	0.3	0.6–0.75
α_{large}	0.3	0.2–0.4

Table 4.1: Parameters used during relaxation and simulation.

4.4 Validation

The simulator is implemented in Java and, for the results in this section, was run on machines with two 4-core Intel Xeon X5355 processors clocked at 2.66GHz. Simulation parameters common to all scenes are in Table 4.1, while scene-specific parameters and details are available in Table 4.2. The renderings were made using a software implementation of the Lumislice method [19] in a ray tracer which follows the original except for the following differences: it uses volume ray tracing rather than alpha blending to accumulate light through the volume; it uses first-order spherical harmonics, rather than a directional table, to store the Volume Reflectance Function; and it uses distribution ray tracing, rather than a shadow map, to compute shadows from area sources. The expensive shadow computations are performed at regularly spaced points throughout the yarn volume, then interpolated as the yarn volume is traversed. Rendering times range from 4 to 15 minutes per frame, on the same hardware as used for simulation.

As a first comparison, the outputs of the simulator for a set of three simple knit patterns—garter, stockinette, and 2×2 rib—were compared to real knitted

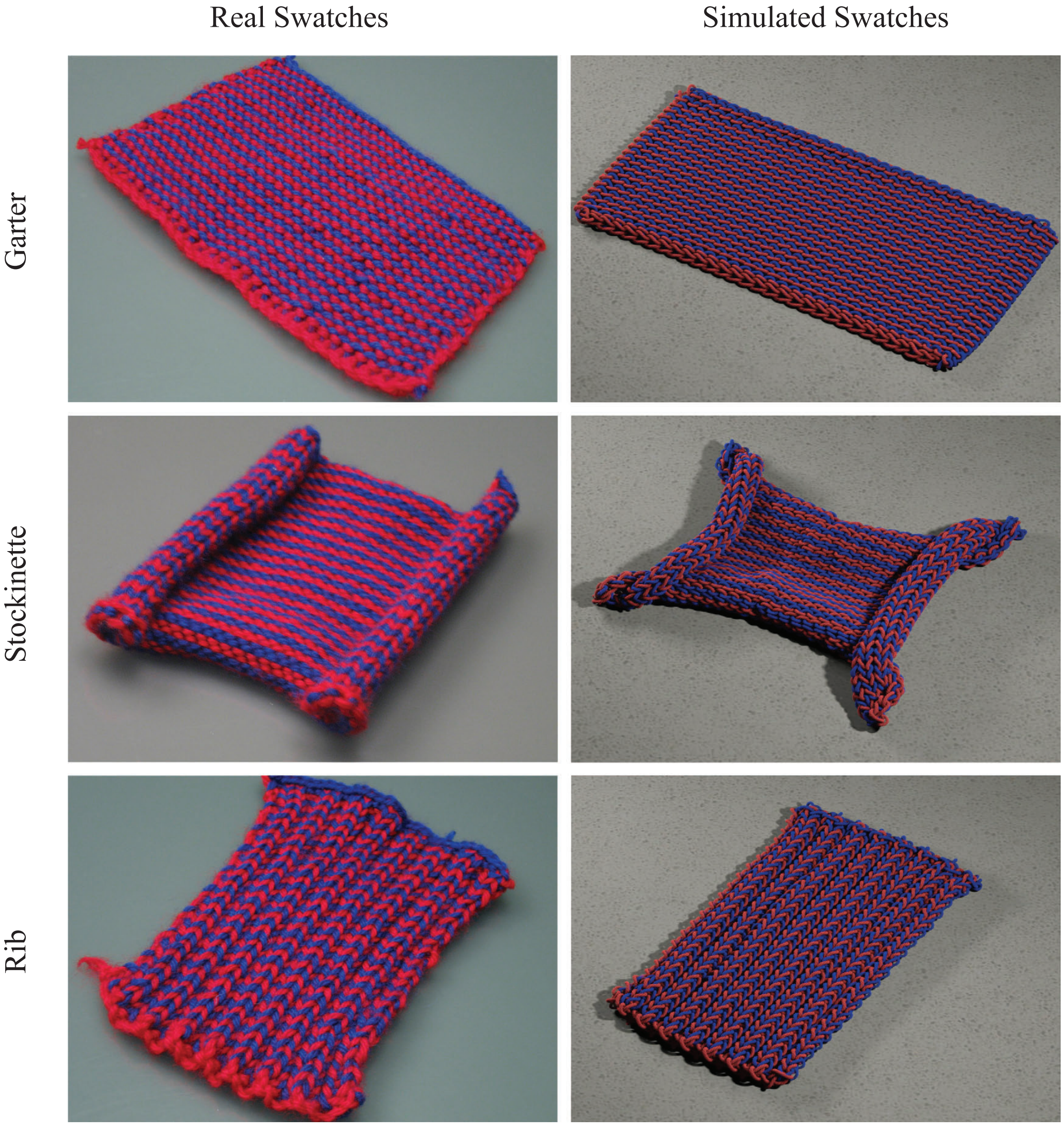


Figure 4.4: Validation comparison

	Stretch	Scarf	Legwarmer
h	1/11800 s	1/22500 s	1/12000 s
Avg # segs/knit loop	8	8	8
# of spline segs	11264	26240	35200
# collision quadrature pts per seg	20	10	10
Avg time per frame	6.8 min	10.7 min	10.8 min
Yarn Collisions	58%	57%	52%
Other energy	7%	12%	23%
Constraints	7%	9 %	19%
Velocity filters	28%	22 %	6%

Table 4.2: Scene statistics.

samples. All of the real samples were knitted using wool worsted size 8 yarn, with each row knitted using alternating colors so that the knit structure is more readily apparent. Each sample consists of 42 rows, with each row containing 32 stitches. The weight and diameter of this yarn was used as input parameters to the simulated models, which consist of the same number of stitches in each row and column. Figure 4.4 illustrates the results of relaxing an initial configuration into a default rest state for the three samples compared to their real-life equivalents. Other than the placement of knit and purl stitches according to the model's knit pattern, all of the other parameters for the three models were identical, so all differences in observed shape are due solely to the differences in input knit pattern. Note that the yarn-based model accurately predicts the curling on the edges of the stockinette, as well as the compression of the rib knit in the course direction and the garter knit in the wale direction. These properties arose naturally from the interactions of the yarn in the yarn-based cloth model; in comparison, to achieve the same effect in an elastic sheet model would require careful manual tweaking of rest angles customized for each particular knit.

As further validation, each of the real and simulated samples were stretched in three directions—along the course, the wale, and the diagonal between the two—and the observed results compared with the simulated results, shown in Figures 4.5, 4.6, and 4.7. For some of the directions, an additional comparison is made with a piece of cloth simulated using an elastic sheet model [4]. For all tests, one end of the cloth was held fixed while the other end was clamped and moved. For the elastic sheet mesh, no yarn geometry is present, so the control points of the knitted cloth are projected onto the mesh while both are at rest, determining for each point its barycentric coordinates with respect to the nearest triangle. Those barycentric coordinates are then used to deform the control points when the mesh is deformed.

The yarn-based model predicts the characteristic shape of the knit while being stretched, in particular the tightening of the yarn loops in the garter, the separation of the ridges in the rib when stretched in the course direction, and the rapid curling of the ends of the stockinette. The elastic model, due to its assumptions of infinitesimal continuity, predicts an unrealistic and inaccurate shape for the garter and the rib as the entire cloth stretches instead of the yarns deforming. For the stockinette, the elastic model does a reasonable job deforming the yarn structure; however, it fails to curl at the ends, which happens in both the real sample and the yarn-model simulation. The yarn model is in fact overeager to curl compared to the sample, although this is probably due to the lack of a complete model for yarn friction. Note that this is a rather strenuous test resulting in stiff but stable contacts, with some of the final states depending largely on frictional forces, which are not being accurately modeled; as a result, some of the configurations reached are unstable and tend to rapidly shift to lower energy ones. However, despite this, the overall deformations of the yarns in the cloth are still captured.

Figure 4.8 shows the robustness of the yarn collision model when applied to

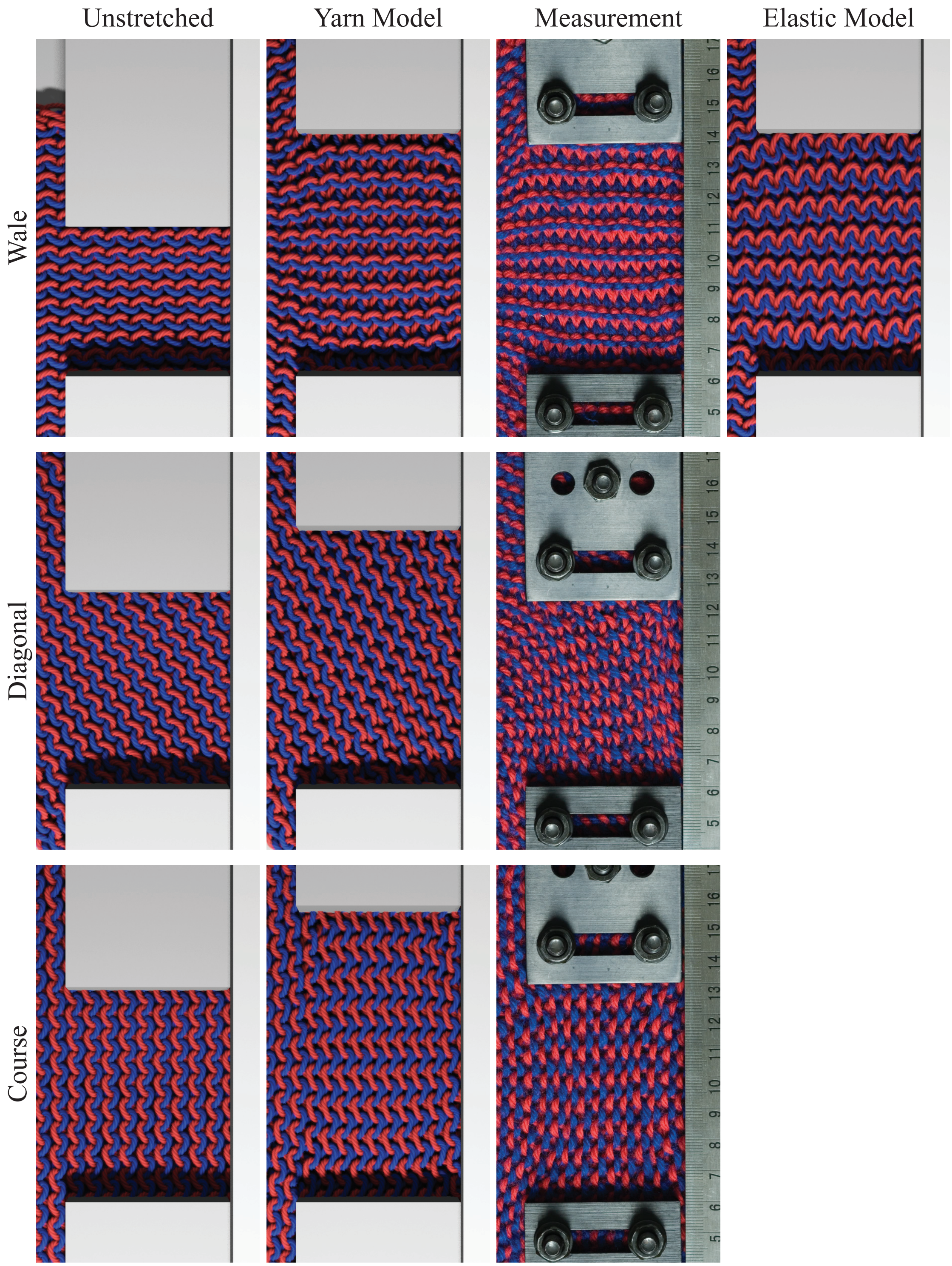


Figure 4.5: Stretch-test comparisons for garter knits.

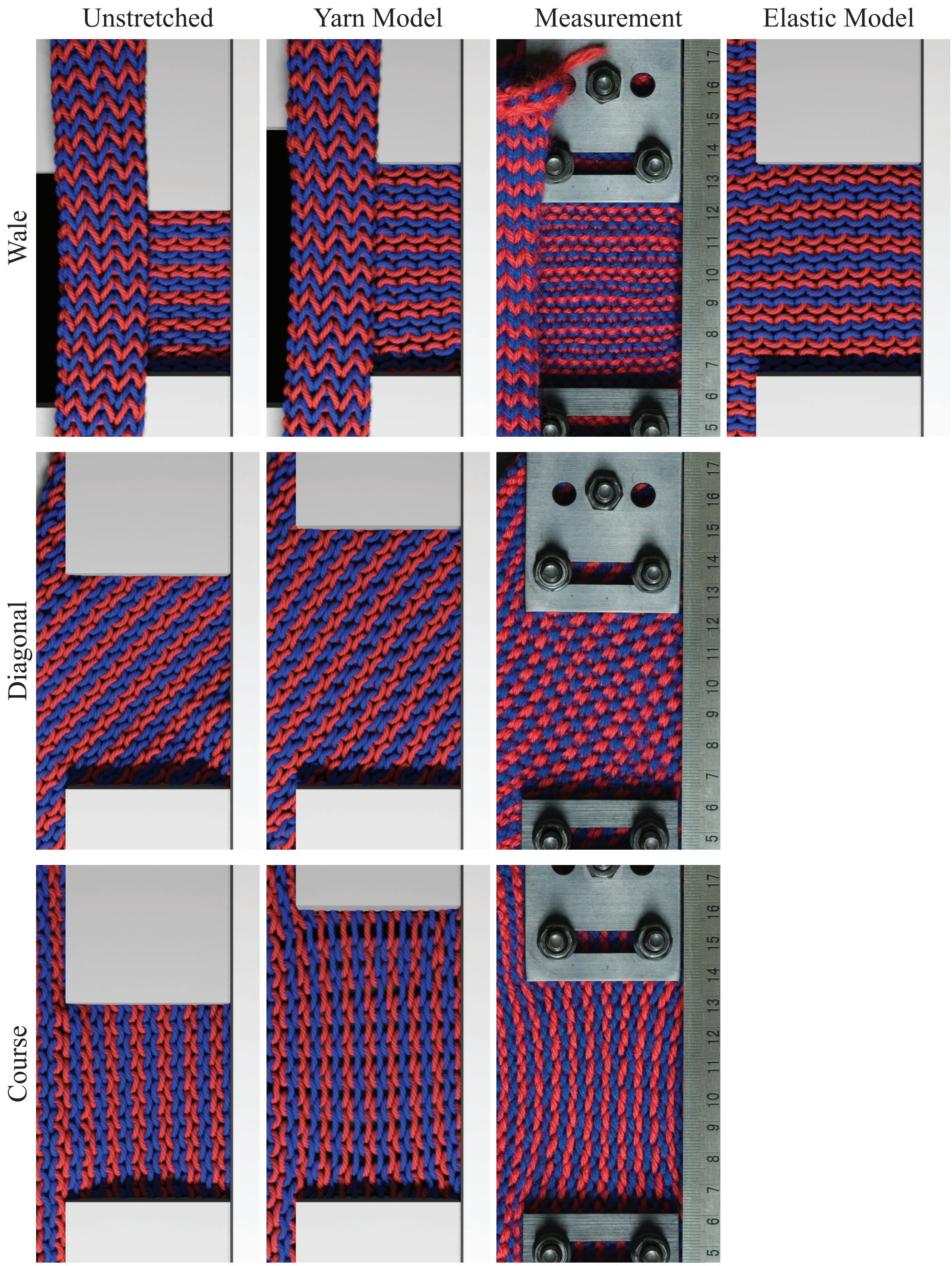


Figure 4.6: Stretch-test comparisons for stockinette knits.

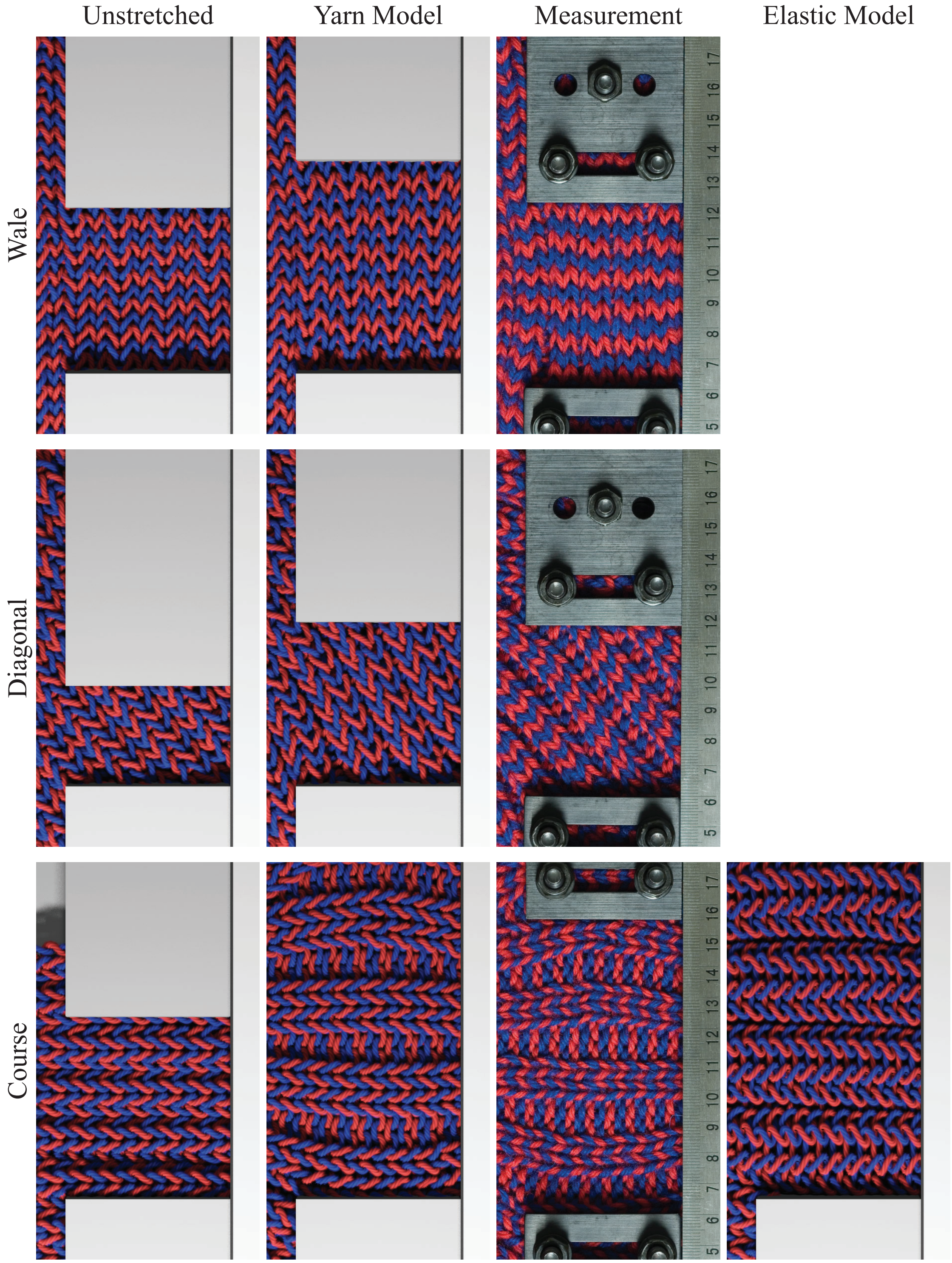


Figure 4.7: Stretch-test comparisons for rib knits.

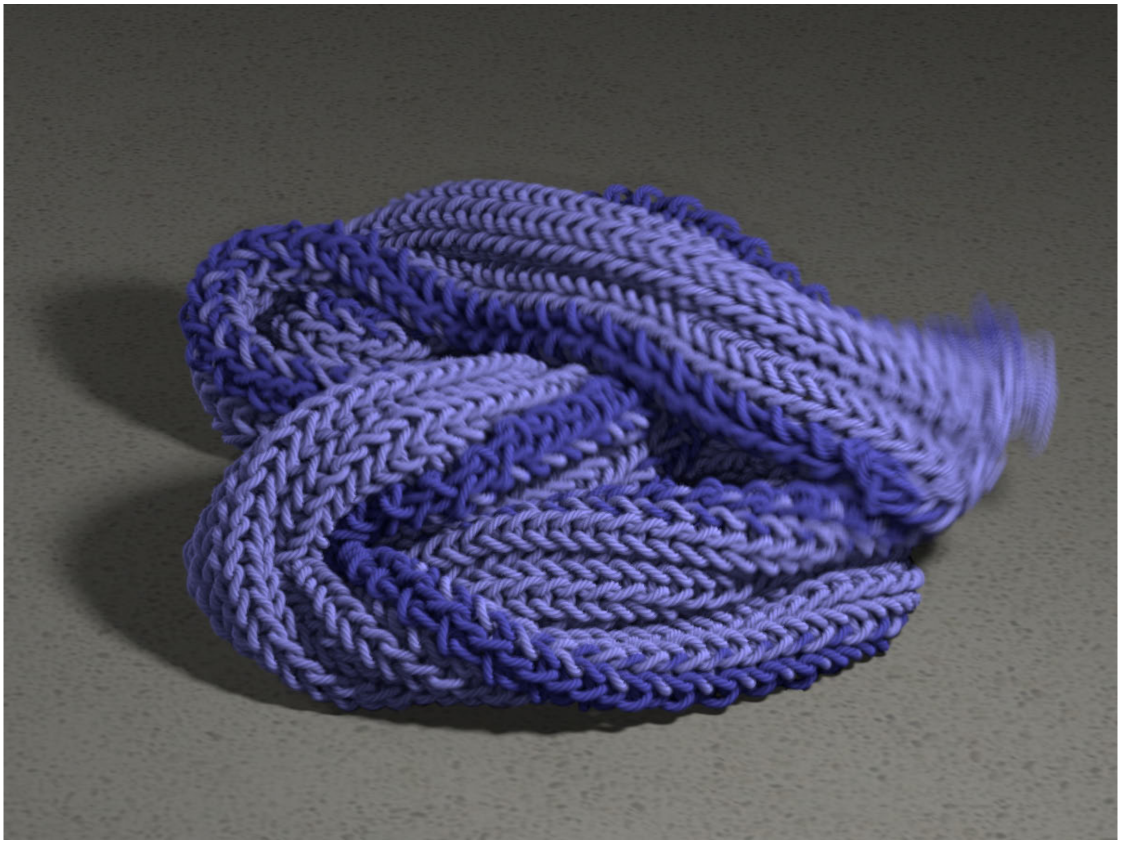


Figure 4.8: Falling scarf

a 20×160 knitted scarf falling onto a plane. The contact model is able to resolve the collisions resulting from contact with both the plane and itself. The average time per frame was about 10.7 minutes due to the small timestep used, which is comparable to the rendering time of about 9 minutes per frame for video-quality renderings.

Figure 4.9 shows a 44×96 knitted leg warmer being pulled over a foot. Because the yarn contacts are simulated directly, the simulator is able to resolve the complicated stretching pattern as it slides over the heel. Due to the size of the model, there are over 100 billion pairs of quadrature points that potentially need to be evaluated for the collision integral at each step. However, the bounding box hierarchy is able to quickly find the 3.7 million pairs on average

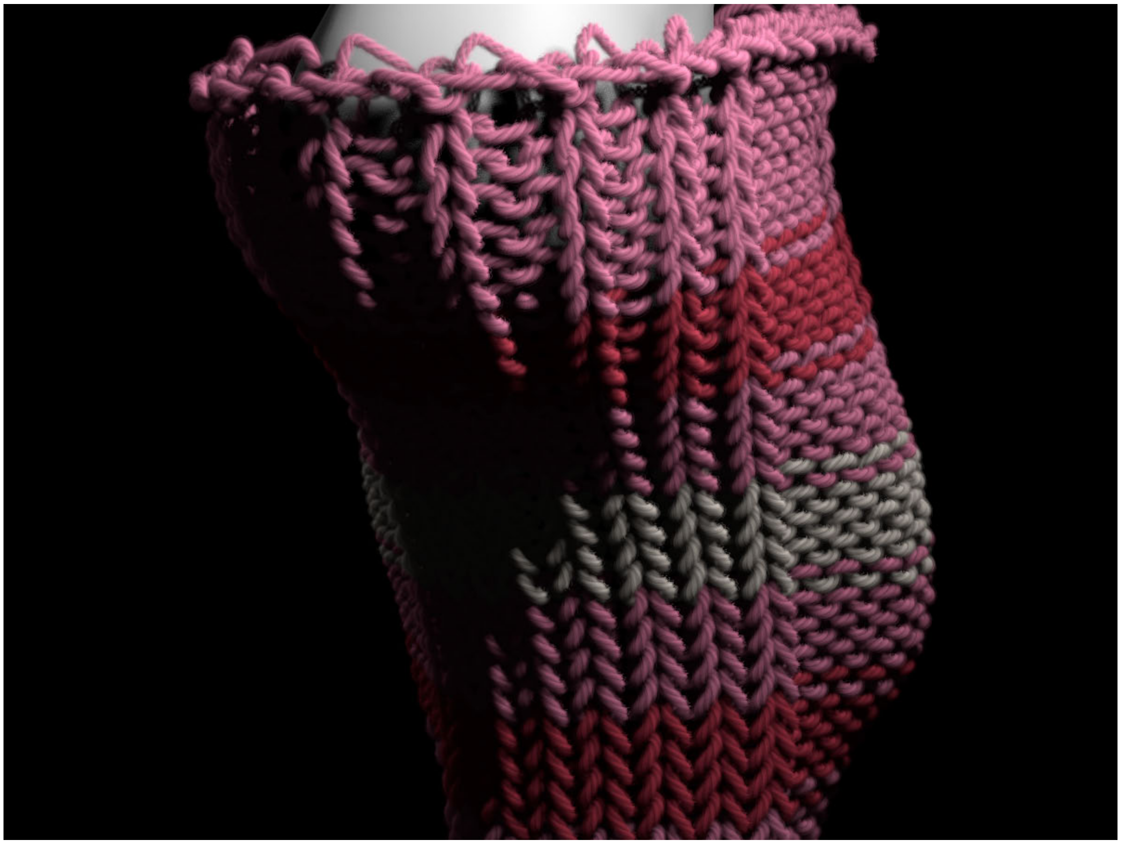


Figure 4.9: Leg warmer

that are in contact, using only 12 million bounding box traversals and 12 million sphere-sphere evaluations on average.

Finally, Figure 4.10 shows a 20×400 scarf composed of 64,690 spline segments falling on an inclined plane. The full movie of the sequence appears in the SIGGRAPH 2008 Computer Animation Festival [60].

4.5 Further Model Improvements

Since this validation study, further improvements to the model and simulator for both quality and performance reasons have been incorporated. These break down into changes to the rod model, internal friction approximation, and integrator.

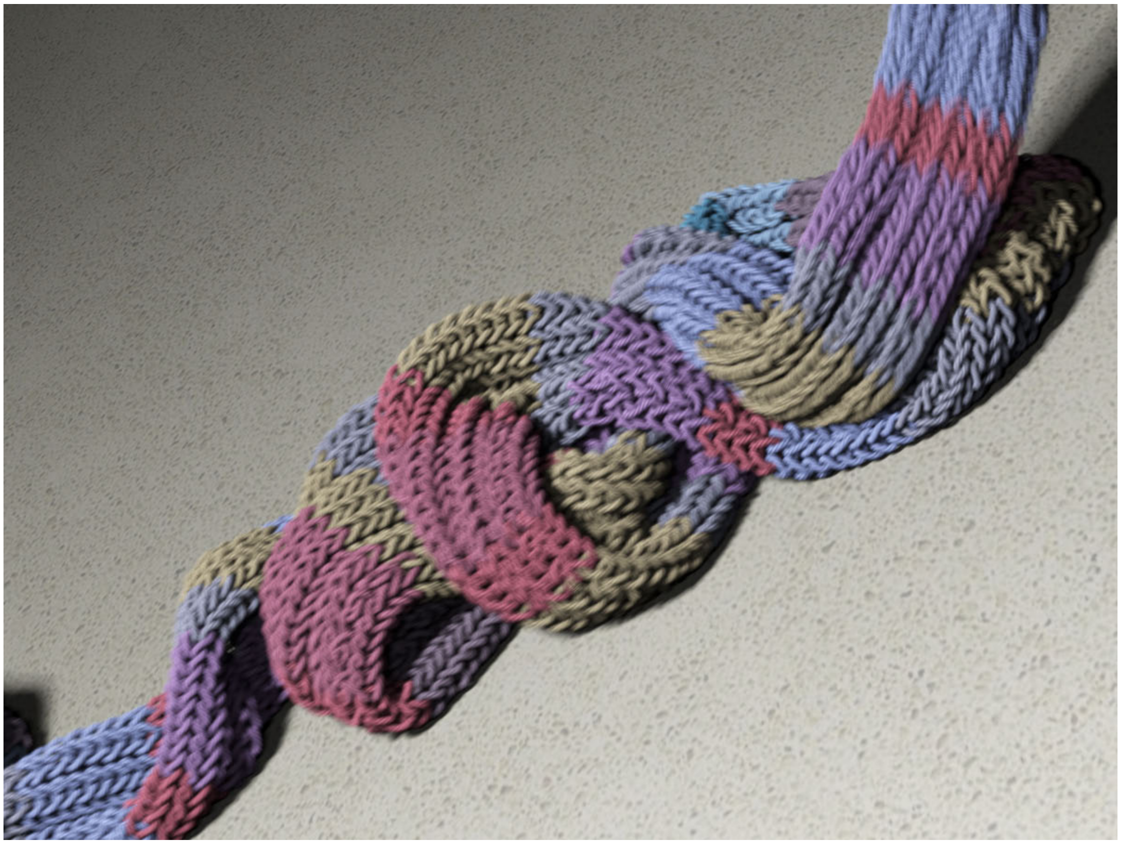


Figure 4.10: A longer scarf on an inclined plane

4.5.1 Discrete Rods

Representing the rod as a higher order continuous surface is convenient for yarn-yarn contacts, since it allows the yarns to slide over each other without snagging. However, this formulation can be overkill for internal yarn forces like stretching and bending, which can be computed in a significantly coarser fashion without losing much accuracy. In addition, the B-spline model proposed in Section 4.2 only models isotropic rods with a straight rest configuration and does not account for internal twisting of the rod, while real yarns can and do have non-zero bending and twisting angles at rest due to both viscoelastic response and the internal friction between fibers [63]. In particular, yarns in knits have very non-straight rest states as a result of the knitting process, which can

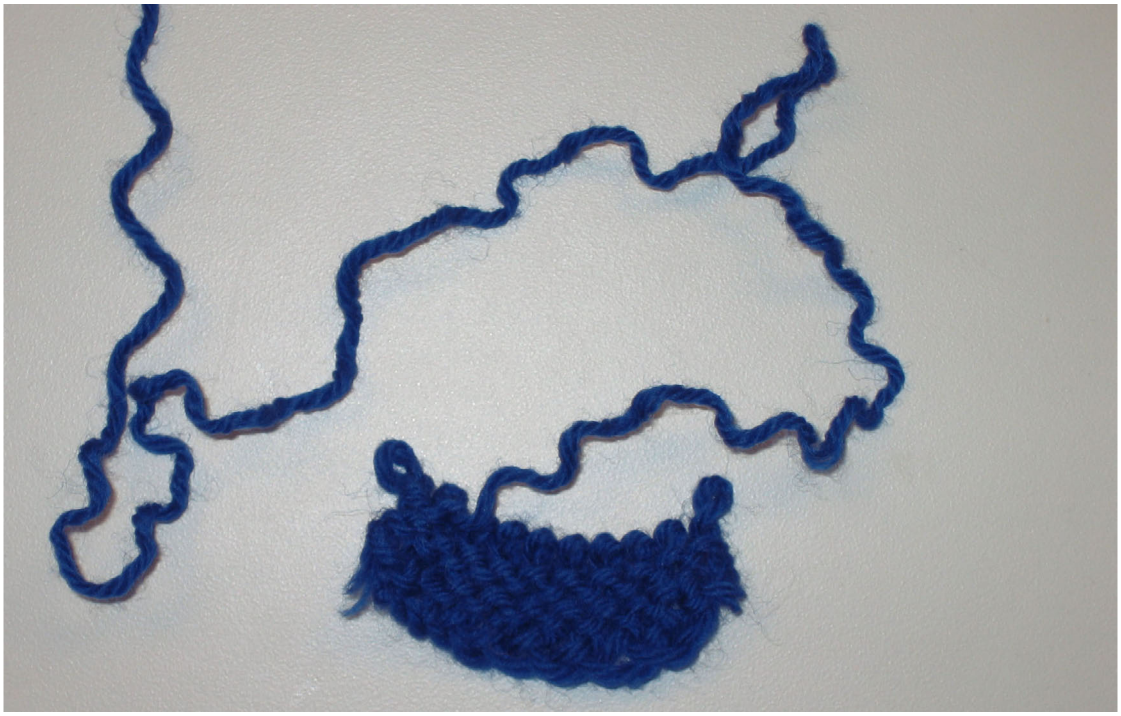


Figure 4.11: Partially unravelled yarn showing non-straight rest configuration

be observed by unravelling a knit [68]; this can result in hysteresis in the behavior of knitted fabrics [70]. Figure 4.11 shows a partially unravelled knit, where the unraveled yarn is still in the characteristically bent shape from when it was knitted. This non-straight rest state can be very important for the overall motion of the cloth, as yarns will in general be in a lower-energy state than that predicted by a model which treats them as straight rods that are bent. Allowing for these rest configurations in the model requires a notion of an internal material frame, adapted to the yarn centerline, with bending and twisting computed from the deformations of these frames over time.

Both of these issues can be addressed by changing the underlying rod model to the Discrete Elastic Rod model proposed in Bergou et al. [9], while preserving the same contact model discussed in §4.2.2. In Discrete Elastic Rods, rods are modeled as piecewise linear centerlines consisting of n segments defined over a set of control points $\mathbf{q} \in I\!R^{3n+3}$, with an adapted twist-free reference frame $\mathbf{u}, \mathbf{v} \in I\!R^{3n}$ and a set of rotations $\theta \in I\!R^{n}$ defined on each segment $\mathbf{e}^i = \mathbf{q}_{i+1} - \mathbf{q}_i$ that measures the angle between the twist free reference frame and the actual material frame. For convenience, $\mathbf{u}^i$ and $\mathbf{v}^i$ will denote the reference frame, θ^i the rotation of the material frame, and $\mathbf{m}_1^j$ and $\mathbf{m}_2^j$ the material frame of the i-th segment. The energy in the yarn is composed of two terms: E_{twist} and E_{bending}, which account for the twisting and bending energy in the yarn, respectively, and which will be defined below.

Because the material is assumed to be thin in two dimensions, twisting waves propagate very quickly, and so the material frames are solved for quasistatically at each timestep. Bending and twisting energies are computed as closed form discrete integrals at each vertex, while yarn inextensibility can be enforced on the piecewise linear segments, avoiding costly quadrature summations. In addition, the nonrigid damping and object contact can also be computed in a significantly coarser fashion directly over the control points without an observed loss of fidelity. Moreover, the model no longer has extra degrees of freedom that allow mass to slide along the yarn, so the stretching energy proposed in §4.2.1 is no longer required. Because the piecewise linear centerline could allow yarns to snag on each other during self-contact, the contact response is computed over the Catmull-Rom interpolating spline through the control points of the linear discretization, which provides a smoother contact surface without adding additional degrees of freedom.

As noted above, in Bergou et al. [9] the material frames on each rod segment were described as a scalar rotation from a zero-twist reference frame, the Bishop frame, defined on each segment. The time evolution of the frame depends on the notion of parallel transport, which is a method of transporting vectors along curves such that the vector does not twist as it travels along the curve. In the discrete piecewise linear case, parallel transport $P_{\mathbf{a}}^{\mathbf{b}}(\mathbf{w})$ of the vector $\mathbf{w}$ from $\mathbf{a}$ to $\mathbf{b}$ reduces to a rotation around the cross product of $\mathbf{a}$ and $\mathbf{b}$ (or binormal of the curve). More simply, if $\mathbf{w}^T\mathbf{a} = 0$, then $P_{\mathbf{a}}^{\mathbf{b}}(\mathbf{w})$ is the vector which is the smallest rotation of $\mathbf{w}$ around $\mathbf{a} \times \mathbf{b}$ such that $P_{\mathbf{a}}^{\mathbf{b}}(\mathbf{w})^T\mathbf{b} = 0$.

In Bergou et al. [9], the Bishop frame at the beginning of the rod is parallel transported through time at each step (i.e. from $\mathbf{e}^0(t)$ to $\mathbf{e}^0(t+h)$), and the remaining frames are generated by parallel-transporting that first frame through space along the rod. However, for nonisotropic or naturally curved rods this zero-twist frame inherently depends on the global state and so the derivative of the bending energy at any point in the yarn depends on the position of every prior point. Despite this dependence, the derivative of bending energy can still be computed in $O(n)$ time instead of $O(n^2)$. Unfortunately, these recursive bending energy computations are difficult to parallelize, and very long yarns (such as in garments) could produce large end-to-end rotations in the reference frame per timestep which can complicate endpoint orientation constraints.

However, twist-free reference frames were only used to simplify twist-energy computation—since then the only twisting is due to the material frame. This restriction can be removed, and instead the reference frame now starts as a twist-free frame, but *every segment's frame* is parallel transported through time (instead of space) from its previous position. Because there is no more spatial parallel transport, rod energy computations have local support and are easily parallelized. The reference frame does accumulate twist, but it can be accounted for and corrected in the twisting energy computation. This was discovered concurrently by Bergou et al. [7], which contains an alternative method of derivation, as well as noting that this local dependence allows for efficient implicit integration.

Each $\mathbf{u}^i$ at time $t+h$ is now updated by parallel transporting the previous $\mathbf{u}^i(t)$ through time (i.e. by parallel transporting $\mathbf{u}^i$ from the vector $\mathbf{e}^i(t)$ to $\mathbf{e}^i(t+h)$). The twist from $(\mathbf{u}^i, \mathbf{v}^i)$ to $(\mathbf{u}^{i+1}, \mathbf{v}^{i+1})$ is denoted as $\hat{\theta}^{i+1}$. This twist can be computed on each timestep by computing the angle between $P_{\mathbf{e}^i(t+h)}^{\mathbf{e}^{i+1}(t+h)}(\mathbf{u}^i)$ and $\mathbf{u}^{i+1}$, i.e., the twist between the space-parallel transported $\mathbf{u}^i$ (which will have zero twist relative to $\mathbf{u}^i$) and $\mathbf{u}^{i+1}$, taking care to handle relative twists greater than 2π properly.

Given this change, the twisting and bending energies must then be redefined to include the relative twist in the reference frame. In addition, derivatives for the energies are needed with respect to both $\mathbf{q}$ (for force evaluation) and θ (to solve for the quasistatic material frame). The modified twisting energy must simply take into account the twist in the reference frame, and becomes:

$$E_{\text{twist}} = \sum_{i=1}^{n} k_{\text{twist}} \frac{(\theta^i - \theta^{i-1} - \hat{\theta}^i)^2}{\bar{\ell}_i}$$

$\frac{\partial E_{\text{twist}}}{\partial \theta^i}$ is straightforward to compute because the twist in the reference frame does not depend at all on the twist in the material frame, so $\frac{\partial \hat{\theta}^j}{\partial \theta^i} = 0$ for all i, j. However, the twist in the reference frame does depend on the position of the centerline, and so $\frac{\partial E_{\text{twist}}}{\partial \mathbf{q}_i}$ becomes:

$$\frac{\partial E_{\text{twist}}}{\partial \mathbf{q}_i} = \sum_{j=0}^{n} \frac{\partial E_{\text{twist}}}{\partial \theta^j} \frac{\partial \theta^j}{\partial \mathbf{q}_i} + \frac{\partial E_{\text{twist}}}{\partial \hat{\theta}^j} \frac{\partial \hat{\theta}^j}{\partial \mathbf{q}_i}$$

where $\frac{\partial \theta^j}{\partial \mathbf{q}_i} = 0$ for all i, j because each frame is independently parallel transported through time in a twist-free manner; $\frac{\partial \hat{\theta}^j}{\partial \mathbf{q}_i}$ is not necessarily zero since $\hat{\theta}^j$ measures the twist in the reference frame and so changes with $P_{\mathbf{e}^i(t+h)}^{\mathbf{e}^{i+1}(t+h)}(\mathbf{u}^i)$ and $\mathbf{u}^{i+1}$. However, because $\mathbf{u}^i$ and $\mathbf{u}^{i+1}$ are updated via parallel transport through time (i.e. with zero twist), this corresponds precisely to the gradient of holonomy of $P_{\mathbf{e}^i(t+h)}^{\mathbf{e}^{i+1}(t+h)}(\mathbf{u}^i)$ relative to $\mathbf{u}^i$ (§6 of [9]), and so:

$$
\begin{aligned}
\frac{\partial \hat{\theta}^i}{\partial \mathbf{q}_{i-1}} &= \frac{(\kappa \mathbf{b})_i}{2|\bar{e}^{i-1}|} \\
\frac{\partial \hat{\theta}^i}{\partial \mathbf{q}_{i+1}} &= -\frac{(\kappa \mathbf{b})_i}{2|\bar{e}^{i}|} \\
\frac{\partial \hat{\theta}^j}{\partial \mathbf{q}_{i}} &= \begin{cases} -\left(\frac{\partial \hat{\theta}^i}{\partial \mathbf{q}_{i-1}} + \frac{\partial \hat{\theta}^i}{\partial \mathbf{q}_{i+1}}\right), & i = j \\ 0, & \text{otherwise.} \end{cases}
\end{aligned}
$$

where $(\kappa \mathbf{b})_i$ is the discrete curvature binormal

$$
(\kappa \mathbf{b})_i = \frac{2\mathbf{e}^{i-1} \times \mathbf{e}^i}{|\mathbf{e}^{i-1}||\mathbf{e}^i| + (\mathbf{e}^{i-1})^T \mathbf{e}^i}
$$

As for the bending energy, the definition itself does not change from [9], but its derivatives do. The bending energy is defined as

$$
E_{\text{bend}} = \sum_{i=1}^{n} \frac{1}{2\ell_i} \sum_{j=i-1}^{i} (\omega_i^j - \bar{\omega}_i^j)^T \mathbf{B}^j (\omega_i^j - \bar{\omega}_i^j), \tag{4.14}
$$

where $\ell_i = |\mathbf{e}^{i-1}| + |\mathbf{e}^i|$, $\mathbf{B}^j$ is the 2×2 symmetric positive definite bending matrix, $\omega_i^j = \left((\kappa \mathbf{b})_i^T \mathbf{m}_2^j, -(\kappa \mathbf{b})_i^T \mathbf{m}_1^j\right)^T$ is the material curvature at vertex i with respect to material frame j, and $\bar{\omega}_i^j$ is the material curvature at rest. $\frac{\partial E_{\text{bend}}}{\partial \theta_i}$ does not change from its definition in [9], but $\frac{\partial E_{\text{bend}}}{\partial \mathbf{q}_i}$ becomes much simpler to compute.

Because each $\mathbf{u}^i$ is parallel transported through time, there is no need to account for the variation in the Bishop frame. As a result, the gradient of the material-frame curvature is now

$$
\nabla_i \omega_k^j = \begin{pmatrix} (\mathbf{m}_2^j)^T \\ -(\mathbf{m}_1^j)^T \end{pmatrix} \nabla_i (\kappa \mathbf{b})_k.
$$

Because $\nabla_i (\kappa \mathbf{b})_k$ is nonzero only for $k - 1 \leq i \leq k + 1$, the gradient of the material-frame curvature is nonzero only for $k - 1 \leq i \leq k + 1$. As a result, summation when computing $\frac{\partial E_{\text{bend}}}{\partial \mathbf{q}_i}$ merely needs to occur over the three-vertex stencil for each bending element:

$$
\frac{\partial E_{\text{bend}}}{\partial \mathbf{q}_i} = \sum_{k=i-1}^{i+1} \frac{1}{\ell_k} \sum_{j=k-1}^{k} \left(\nabla_i \omega_k^j\right)^T \mathbf{B}^j \left(\omega_k^j - \bar{\omega}_k^j\right) \tag{4.15}
$$

4.5.2 Improved Internal Friction Model

Internal Yarn Plasticity

Yarns are not simple elastic materials; rather, they display a complex set of dynamics driven by fiber interactions, which include significant plastic behavior under deformation. This can be approximated using a simple plasticity model on the rest state of the rod in angular space. In Discrete Elastic Rods, this rest state $\bar{\omega}_j^i$ is the product of the curvature binormal at a given bending element $\mathbf{q}_j$ (vertex) with the material frame of a neighboring segment, so $\bar{\omega}_j^i \in I\!R^2$ and can be represented as a 2D point. If this point lies outside the circle of radius p_{plastic} centered at the current state ω_j^i of that pair, the rest state is projected onto the boundary of the circle. Similarly, if the rest state falls outside the circle of radius $p_{\text{plastic}}^{\text{max}}$ centered at the origin, it is projected onto the boundary of that circle; this approximates the fact that there is some maximum angle at which the compression of the material overwhelms the frictional forces. This model is easy to evaluate while still allowing for permament deformations of the rest state within some allowable angular range at each bending element.

Nonrigid Damping

As discussed in §4.2.3, fiber interactions and entanglements are important for capturing the overall motion of cloth, but they are difficult to model explicitly. The nonrigid damping in §4.2.3 works well at plausibly damping out the overall motion, but it has some important limitations. Most critically, fiber entanglements in real cloth can lead to plastic deformations—for instance, creases or persistent folds—but the nonrigid damping only subtracts a fixed proportion of the overall nonrigid motion on each timestep. This means that there is always some remaining nonrigid motion, which shows up as an undesirable creeping

```
$\mathbf{f}^{(t+1)}$ = evaluate_forces()
$\dot{\mathbf{q}}_{\text{uncons}} = \dot{\mathbf{q}}^{(t)} + h\mathbf{M}^{-1}\mathbf{f}^{(t+1)}$
$\dot{\mathbf{q}}_{\text{glue}} =$ satisfy_glue_constr($\mathbf{q}^{(t)} + h\dot{\mathbf{q}}_{\text{uncons}}$)
$\dot{\mathbf{q}}_{\text{length}} =$ satisfy_length_constr($\mathbf{q}^{(t)} + h\dot{\mathbf{q}}_{\text{glue}}$)
$\dot{\mathbf{q}}_{\text{damp}} =$ nonrigid_damping($\dot{\mathbf{q}}_{\text{length}}$)
$\dot{\mathbf{q}}^{(t+1)} =$ object_contact($\mathbf{q}^{(t)} + h\dot{\mathbf{q}}_{\text{damp}}$)
$\mathbf{q}^{(t+1)} = \mathbf{q}^{(t)} + h\dot{\mathbf{q}}^{(t+1)}$
quasistatic_frames()
```

Figure 4.12: Overview of revised algorithm

behavior towards the energy minimum.

This can be accounted for by allowing the nonrigid damping to remove a constant (as opposed to proportional) amount of nonrigid velocity at low speeds. For a given damping region, let k be the number of control points in the region, and $\mathbf{v}_{\text{nonrigid}} \in \mathbb{R}^{3k}$ be the nonrigid velocity of that region. The applied change to the velocity of all control points in the region is then:

$$\Delta\, \mathbf{v} = -\mathbf{v}_{\text{nonrigid}} \min\left(1, \max\left(h\mu_{\text{prop}}, \frac{h\mu_{\text{const}}\sqrt{k}}{\|\mathbf{v}_{\text{nonrigid}}\|_2}\right)\right).$$

Note that this computation is over the $3k$-dimensional velocity and not at each point, to ensure conservation of momentum inside the region[1]. At low speeds, rather than removing a proportional amount of nonrigid velocity at each step, a constant amount is removed, up to the entire nonrigid velocity; this allows the cloth to come to rest in finite time, which is an approximation of the complex hysteresis seen in real cloth. Moreover, the simulator now only applies one phase of nonrigid damping on each timestep, over blocks of 4×4 knit loops.

4.5.3 Reordered Integration

For the explicit integration of the force terms, symplectic Euler is a better choice than explicit midpoint since it preserves momentums while only requiring one force evaluation; the additional accuracy of midpoint is typically not of practical significance given the small timesteps involved in simulation.

Due to the length constraints now being solved over the piecewise linear discretization, with only length constraints the linear system that needs to be solved with Fast Projection [39] is tridiagonal and thus easy to solve directly. However, including glue (or other) constraints as well destroys the tridiagonality of the system, and even with the tridiagonal part of the matrix used as a preconditioner the system requires several PCG iterations to solve. Therefore, to accelerate the constraint satisfaction process the glue and length constraints are solved separately, producing simple systems that can be solved directly (diagonal and tridiagonal, respectively). Moreover, both damping and object contact are applied after glue and length constraints to allow the cloth to come to rest; while this means that the constraints are in principle no longer satisfied exactly at the end of each timestep, this has not resulted in any observed problems in practice. Because the glue constraints are solved for first, and are thus most likely to be violated, additional springs are inserted at the glue points to ensure the points stay close enough together to remain glued, but this represents a negligible overall cost to the simulation. Figure 4.12 lists the complete revised algorithm, while Table 4.3 lists the common values for the new simulation parameters.

[1] If the regions overlap, however, it will not necessarily be momentum conserving. This can be corrected by computing the total change in momentum of the cloth after applying the filter, and then correcting for it

parameter	description	value
h	timestep	1/24000s
p_{plastic} ; $p_{\text{plastic}}^{\max}$	yarn plasticity	0.01 ; 2.5
μ_{const} ; μ_{prop}	nonrigid damping	500 ; 1500 – 4500

Table 4.3: Parameters used for revised simulator.

4.5.4 Revised Results

Figure 4.13 shows six frames from the falling scarf simulated using the model improvements discussed in this section. Note that the scarf is still lively and readily buckles upon contacting with the floor. However, it still quickly comes to rest on the ground without any oscillations or further sliding. Comparing the timings for this example with the timings from Section 4.4 on the same hardware, this simulation takes 6m 52s per frame.

4.6 Conclusions

This chapter demonstrated a robust and scalable technique for simulating knitted cloth at the yarn level that can exploit the parallelism in current multi-core architectures. The approach allows for significant increases in yarn-level knitted cloth complexity over previous research, while achieving practical offline simulation rates. Qualitative validation shows that the yarn-based simulation closely matches observed behavior in actual knit samples and automatically captures these visually noticeable nonlinear effects that are not in the elastic sheet approximation. In particular, the model is able to capture the salient mechanical features of garter, stockinette, and rib knits at rest without any parameter tuning or special cases—it follows directly from yarn interactions.

Although simulations of moderate complexity proved to be tractable in a reasonable amount of time, in order to be truly useful the simulator must be

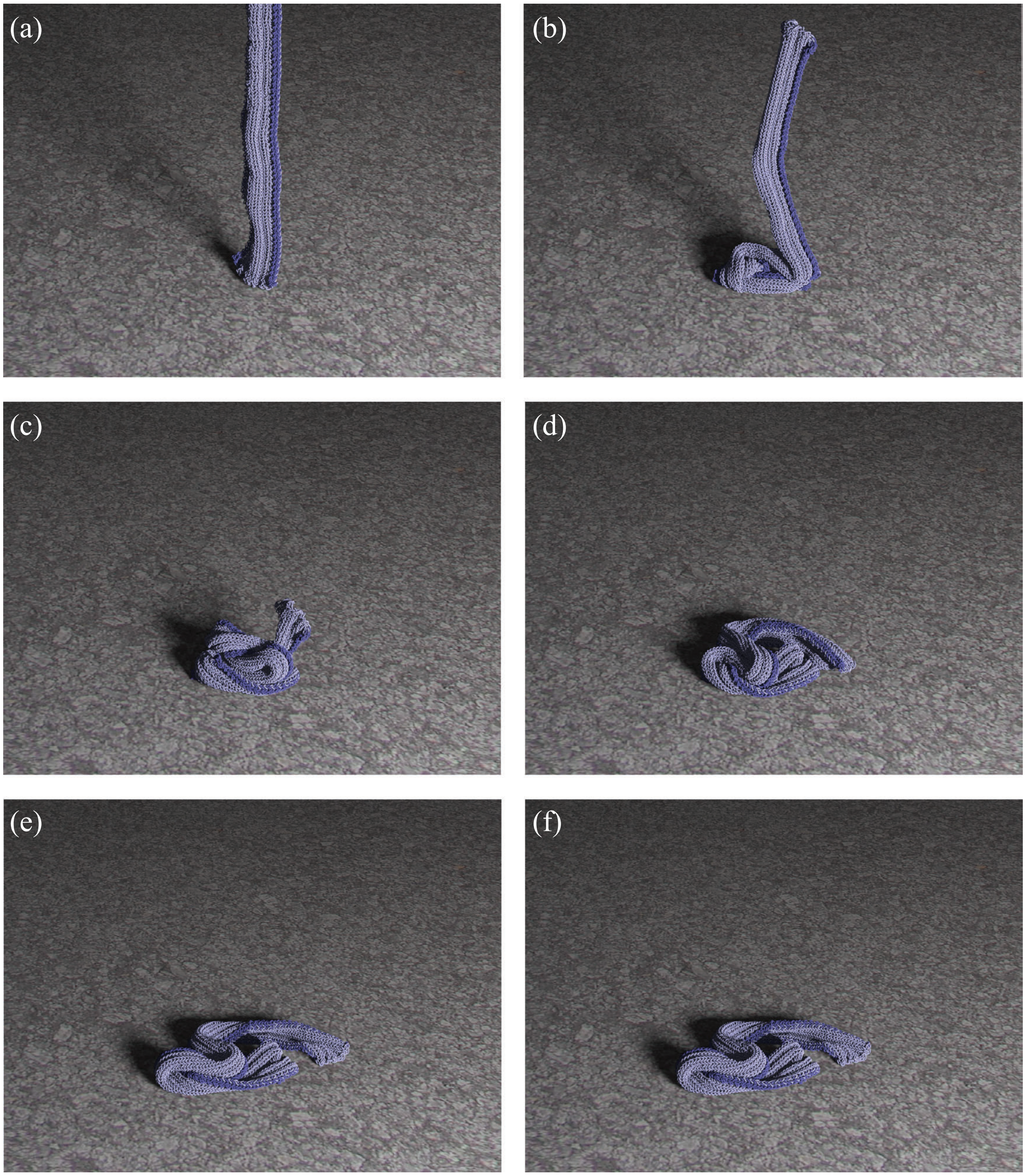


Figure 4.13: Six frames from the falling scarf after model improvements of Section 4.5

able to scale up to character-sized garments, which can contain well over 50,000 knit loops. This order-of-magnitude jump in model size requires additional speedups in the model simulation in order to stay in the realm of practicality. However, although there is a high degree of *potential* complexity in the motion of knit models, in practice there appears to be a relatively low amount of *actual* complexity at any given time. Rather, the geometry evolves relatively slowly, with many regions of the cloth undergoing little or no deformations at any given time. This observation will be explored in more detail in the following two chapters.

CHAPTER 5

RIGIDIFICATION

The high degree of complexity in yarn-based cloth models is a limit on the overall performance, with the bulk of the time spent resolving close contacts with many degrees of freedom. However, for many simulations some number of these degrees of freedom may be inactive at any given time. For instance, cloth laying on the ground at rest is obviously not moving at all, or a skintight garment may move along with the character's motion but in some lower dimensional space; here, only a limited range of stretching modes may be exercised as the material is held in place locally by friction. Unfortunately, the simulator described in Chapter 4 cannot recognize these regions, and so it is forced to evaluate the complex yarn behavior—in particular the contact response—at every point and on every timestep. As a result, presumably easy cases like cloth laying at rest require just as much simulation effort as when the cloth is rapidly deforming, all to determine that it is undergoing some simpler motion instead.

As discussed in Section 3.1.5 there is a wide range of prior work on reduced models for deformable bodies, which allow for efficient simulation within some reduced space of allowable motions. Most of these techniques are globally applied, though, which is challenging for yarn-based cloth since the global space of current deformation modes may not be small. However, as observed above, in local regions the space of deformations may be much smaller, and so the goal is to apply these same ideas of model reduction but at a local level instead. These local models should ideally allow some range of motion in some lower-dimensional space, but they should not be expected to provide the same quality of motion as the full yarn model. Rather, the simulator will decide when and where it is appropriate to substitute these models in and, more importantly, when the simple models represent an unacceptable degradation in quality and the full yarn model should be used instead. The result is a hybrid simulation, with regions of high motion complexity simulated expensively using the model in Chapter 4 and regions of low motion complexity simulated using some cheap-to-evaluate model, and the simulator automatically switching regions back and forth between the two models as needed to maintain a specified quality of motion while minimizing computation time.

There are several questions and challenges that need to be addressed, however. First, what is an adequate low-dimensional and cheap to evaluate model? This chapter discusses an approach that uses rigid bodies as its simplified model, discussed in detail in Section 5.1, where local regions of the yarn-based model are approximated as being rigid. The key benefit of using rigid bodies as the simplified model is that it prevents any changes in proximity inside the region and, as a result, the computation of the self-contact force inside the reduced model can be avoided entirely. Because neighboring sections of the same piece of cloth may be rigid or not, the integrator must be adapted to properly handle the interfacing between the two models, as addressed in Section 5.2. Finally, because approximating the cloth as rigid will obviously greatly reduce the quality of deformations, how does the simulator decide when it is acceptable to rigidify a region and, most crucially, determine when to derigidify a given region to maintain the overall quality of motion? Section 5.3 discusses the most successful attempts at solving the difficult problem of when to switch back and forth between the two models. Section 5.4 shows some results for the new simulator, in particular cases where the model fails to adequately respond to changes in applied force in time to avoid significant degradation in motion. Finally, Section 5.5 discusses the underlying problems and lessons learned from this approach that make it so challenging to adequately solve and lays the groundwork for the ultimately more successful methods of Chapter 6.

5.1 Rigidification Model

The cloth is broken up into a hierarchy of rigid zones, each of which may or may not be rigidified at any moment in time, with each zone containing some set of control points. When a given zone Z_k is rigidified, all control points in that zone now evolve through time rigidly, using the rigid body equations of motion. Note that since the zones are arranged in a hierarchy, Z_k has a parent $\textbf{parent}(Z_k)$, and may also have some number of child zones $\textbf{child}(Z_k)$ as well; each child zone $Z_c \in \textbf{child}(Z_k)$ must be a strict subset of Z_k, so $Z_c \subset Z_k$, and Z_k must itself be a strict subset of $\textbf{parent}(Z_k)$ if it exists. Because some control points in zone Z_k may also be in child zones of Z_k while others are not, it will be helpful to distinguish the control points Z_k^{local} which are rooted at Z_k (i.e. are in Z_k but not any child zones of Z_k). It is a necessary but not sufficient condition for a zone to be rigidified if all of its children zones are already rigid; further restrictions on the ability of a zone to be rigidified will be discussed in Section 5.3.

Rigidity represents a severe reduction in the number of degrees of freedom in the simulation at a local level. Because of this, if two rigidified zones are too topologically close to each other, there may not be enough remaining degrees of freedom between the two zones to allow them to move independently from each other, and the two zones will be effectively rigidly coupled. This has the effect of introducing additional, unintended locking behavior in the simulation, which should be avoided. As a result, a buffer of free control points is left around each Z_k and not included in any other rigid zone at that level in the rigid zone hierarchy; they may be included in ancestor zones of Z_k in the hierarchy, however.

Section 5.1.2 contains a more detailed discussion on how rigid zones are formed and grouped. See Figure 5.1 for an example of a hierarchy of rigid zones along with buffer regions.

5.1.1 Rigid Zone Dynamics

In order to simulate a zone rigidly, the dynamics of a rigid body must be specified. This is a well-understood problem, and there are many additional sources of information which can be consulted in addition to this section [40, 34].

When a zone is rigidified, the necessary rigid body properties are computed from the control points. This includes the total mass M_{Z_k}, center of mass $\mathbf{x}_{Z_k}$, velocity $\mathbf{v}_{Z_k}$, angular momentum $\mathbf{L}_{Z_k}$, and inertia tensor $\mathbf{I}_{Z_k}$:

$$\begin{aligned}
M_{Z_k} &= \sum_{i \in Z_k} m_i \\
\mathbf{x}_{Z_k} &= \frac{1}{M_{Z_k}} \sum_{i \in Z_k} m_i \mathbf{q}_i \\
\mathbf{v}_{Z_k} &= \frac{1}{M_{Z_k}} \sum_{i \in Z_k} m_i \dot{\mathbf{q}}_i \\
\mathbf{L}_{Z_k} &= \sum_{i \in Z_k} m_i \mathbf{d}_i \times \dot{\mathbf{q}}_i \\
\mathbf{I}_{Z_k} &= \sum_{i \in Z_k} m_i \left(\mathbf{d}_i^2 \mathbf{E}_{3\times 3} - \mathbf{d}_i \mathbf{d}_i^T \right)
\end{aligned}$$

where m_i is the mass of control point i, $\mathbf{q}_i$ and $\dot{\mathbf{q}}_i$ are its position and velocity, respectively, and $\mathbf{d}_i = \mathbf{q}_i - \mathbf{x}_{Z_k}$. To distinguish between the identity matrix and the inertia tensor, $\mathbf{E}$ will be used in the remainder of this chapter to denote the identity matrix. The initial rotation $\mathbf{R}_{Z_k}$ of the zone is set to the identity matrix.

For numerical robustness it may be desirable to represent the rotation instead as a unit quaternion, denoted as $\mathbf{r}_{Z_k}$; the remainder of this chapter will use either $\mathbf{R}_{Z_k}$ or $\mathbf{r}_{Z_k}$ depending on convenience. Each of these quantities is defined as a loop over all control points, but if they are already known for the child zones of Z_k then they can be computed much faster as a loop over the child zones as well as the control points in Z_k^{local}:

$$M_{Z_k} = \sum_{i \in Z_k^{\text{local}}} m_i + \sum_{Z_c \in \textbf{child}(Z_k)} M_{Z_c} \tag{5.1}$$

$$\mathbf{x}_{Z_k} = \frac{1}{M_{Z_k}} \left(\sum_{i \in Z_k^{\text{local}}} m_i \mathbf{q}_i + \sum_{Z_c \in \textbf{child}(Z_k)} M_{Z_c} \mathbf{x}_{Z_c} \right) \tag{5.2}$$

$$\mathbf{v}_{Z_k} = \frac{1}{M_{Z_k}} \left(\sum_{i \in Z_k^{\text{local}}} m_i \dot{\mathbf{q}}_i + \sum_{Z_c \in \textbf{child}(Z_k)} M_{Z_c} \mathbf{v}_{Z_c} \right) \tag{5.3}$$

$$\mathbf{L}_{Z_k} = \sum_{i \in Z_k^{\text{local}}} m_i \mathbf{d}_i \times \dot{\mathbf{q}}_i + \sum_{Z_c \in \textbf{child}(Z_k)} \left(\mathbf{L}_{Z_c} + M_{Z_c} \mathbf{d}_{Z_c} \times \mathbf{v}_{Z_c} \right) \tag{5.4}$$

$$\mathbf{I}_{Z_k} = \sum_{i \in Z_k^{\text{local}}} m_i \left(\mathbf{d}_i^2 \mathbf{E}_{3\times3} - \mathbf{d}_i \mathbf{d}_i^T \right) + \tag{5.5}$$

$$\sum_{Z_c \in \textbf{child}(Z_k)} \left(\mathbf{R}_{Z_c} \mathbf{I}_{Z_c} \mathbf{R}_{Z_c}^T + M_{Z_c} \left(\mathbf{d}_{Z_c}^2 \mathbf{E}_{3\times3} - \mathbf{d}_{Z_c} \mathbf{d}_{Z_c}^T \right) \right) \tag{5.6}$$

where $\mathbf{d}_{Z_c} = \mathbf{x}_{Z_c} - \mathbf{x}_{Z_k}$. For zones where $|Z_k^{\text{local}}| \ll |Z_k|$ this represents a substantial savings; because zones are only rigidified when all of their children are already rigid, these quantities are always guaranteed to be known.

Given these quantities, the equations of motion for a rigid zone follow directly from the dynamics of a rigid body:

$$\ddot{\mathbf{x}}_{Z_k} = \dot{\mathbf{v}}_{Z_k} = \frac{1}{M_{Z_k}} \sum_{i \in Z_k} \mathbf{f}_i \tag{5.7}$$

$$\dot{\mathbf{L}}_{Z_k} = \sum_{i \in Z_k} \mathbf{d}_i \times \mathbf{f}_i \tag{5.8}$$

$$\dot{\mathbf{r}}_{Z_k} = \frac{1}{2}[0; \mathbf{R}\mathbf{I}_{Z_k}^{-1}\mathbf{R}^T\mathbf{L}_{Z_k}]\mathbf{r}_{Z_k} = \frac{1}{2}[0; \omega_{Z_k}]\mathbf{r}_{Z_k} \tag{5.9}$$

where $\mathbf{f}_i$ is the applied force to control point i Note that the rotation is integrated using quaternions, allowing for easy renormalization after each timestep, and that ω_{Z_k} is treated as a purely imaginary quaternion during multiplication.

Once zone Z_k is rigidified, the rigid body variables become the active simulation variables, and the position and velocity of the now-rigidified control points in Z_k become dependent on these variables. The positions of rigidified control points are typically computed on an as-needed basis by the simulator; however, due to the design of the yarn model there are some which are always needed, which are grouped into the set $Z_k^{\text{boundary}} \subset Z_k$ and which are always kept up-to-date. The contents and purpose of this set will be discussed in Section 5.2.1.

5.1.2 Defining Rigid Zones

There are obviously many issues to consider when defining the sets of rigid zones and their hierarchical grouping. For starters, the zones can either be defined statically, i.e. fixed at the beginning of simulation, or dynamically, i.e. time-varying during the course of simulation. Static zone generation is simpler to implement, but it can fail to achieve maximum performance if the motion of the cloth contains a rigid region in the cloth that crosses through but does not entirely contain several statically defined rigid zones. In contrast, dynamically defined zones can in theory capture arbitrary sets and subsets of rigidity

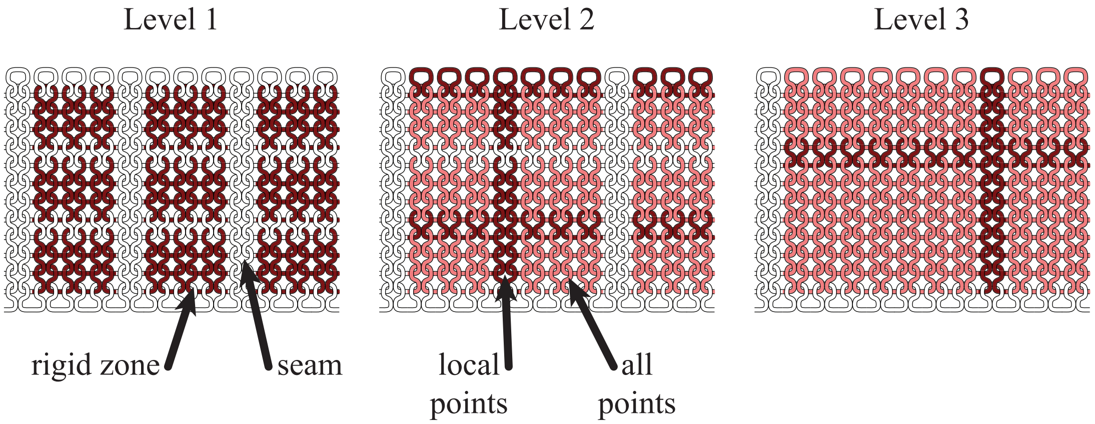


Figure 5.1: Hierarchy of zones in a simple knit

in the cloth, by defining and redefining zones which precisely encompass only the rigidly deforming regions; however, this introduces additional complexity as the simulator must now also create and refine these zones over time, presumably using some clustering or matching criteria. Because of this complexity, the rest of this chapter assumes a static set of rigid zones.

Given these static zones, there are several additional issues that need to be addressed—the size of the smallest level of rigid zone, how they are grouped to form larger zones, and how to prevent locking behavior when neighboring zones are rigidified. The simulator presented in this chapter creates zones out of 3×3 blocks of knit yarn loops, and creates larger rigid zones out of 2×2 blocks of neighboring rigid zones (if there are an odd number of zones at any given level, some zones created on the next level will have either/both 3 rows and 3 columns). Finally, as noted earlier, rigid zones have severely reduced dynamics, and it is important to reduce the chance of locking behavior by removing too many degrees of freedom. As a result, a buffer of one yarn loop is kept between the rigid zones at any given level; as the hierarchy of zones is traversed upward, buffer loops which are entirely internal to a rigid zone are rigidified along with the zone. See Figure 5.1 for a diagram of the zone generation procedure.

```
For each timestep, $h$
    update_rigidity_state()
    [$\mathbf{q}_n, \dot{\mathbf{q}}_n$] = integrate_dynamics($\mathbf{q}_0$, $\dot{\mathbf{q}}_0$)
    Evaluate impulses, determine which zones
              should be broken on next timestep
end
```

Figure 5.2: Overview of rigid-cloth simulator

5.2 Modifications to Integrator

As seen in Figure 5.2, the algorithm can be broken down into three steps. Before the step is taken, the current rigidity state is updated, with zones rigidified or derigidified according to the oracle in Section 5.3. The dynamics are then integrated, producing a new position and velocity for the system. Given this new state, the zones are further evaluated to determine whether any impulses over the timestep should result in zones being derigidified, before the step is finished and the next one is started; this is discussed in Section 5.3.3. The remainder of this section discusses the necessary modifications to the `integrate_dynamics()` step.

Although both the yarn and the rigidified zones have well-defined equations of motion and methods of integration, because they are interrelated it is necessary to integrate both models forward in time simultaneously. This leads to several modifications to the basic integrator in order to efficiently support simultaneous integration and ensure that rigid zones represent a sufficient speedup. The goal is to take advantage of the hierarchy of rigid zones in order to avoid looping over all points in all rigid zones. Rather, the integrator should avoid computing unnecessary quantities and instead try and use aggregate quantities to make computation proportional to $|Z_k^{\mathrm{local}}|$ and $|\mathbf{child}(Z_k)|$, instead of $|Z_k|$.

5.2.1 Force Computation

In principle, since the rigid body equations of motion (5.7), (5.8), and (5.9) are expressed as summations over the forces on control points, integrating the rigidified zones requires merely computing the force on all control points as before and then summing accordingly. However, many of these force computations are either no longer necessary or can be simplified inside the rigid zones. Note that all of these represent exact aggregate values for the zones; only computations which can have no effect on the motion of the rigid zone are discarded. Depending on the type of force, this is accomplished in one of several ways; in the following discussion, assume zone Z_k has been previously rigidified, and without loss of generality assume the yarn model consists of a single yarn.

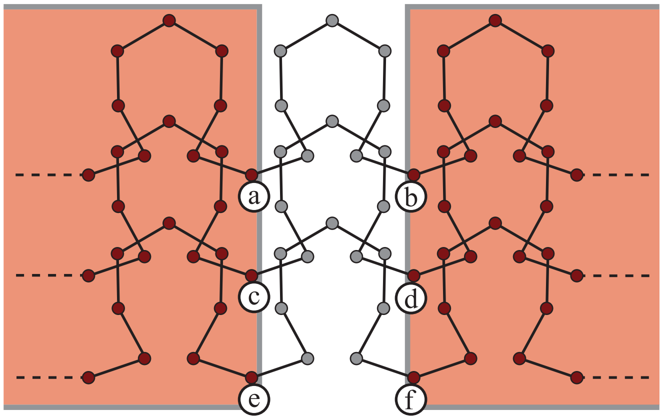
Gravity and damping

Gravity of acceleration g on control points in Z_k can be applied as a single force of magnitude M_{Z_k}g directly at the center of mass of the rigidified zone, while mass-proportional damping over all control points can be computed as a force $-k_d\mathbf{v}_{Z_k}$ applied at the center of mass and a change of angular momentum $-k_d\mathbf{L}_{Z_k}$.

Bending / Twisting

For any internal rod dynamics like bending, twisting, or self-contact, if all of the affected control points lie in the same rigidified zone then they cannot influence the final motion of the zone; because all of them are pairwise equal-and-opposite forces, they are conservative forces and so produce net zero torque and force. As a result, forces entirely internal to a rigidified zone can be safely skipped; the problem is efficiently determining when this is the case.

The set of zones which have been rigidified induce a natural partitioning of the yarn into a set of intervals, where the intervals alternate between being rigid and being nonrigid. Moreover, because rigid zones are either strict supersets or do not intersect, and there are buffers between zones, any points in a given rigid interval must be in the same rigid zone. Thus, for bending and twisting elements, any force computation entirely inside a rigid interval can be avoided. Define the set $Y = \{(i, j)\}$ to be the set of nonrigid intervals, where for each interval (i, j) control points i and j are rigid (or the beginning/end of the yarn), and all control points between i and j are not rigid. Figure 5.3 contains an example of several rigidified zones and the corresponding set Y. Given this set, computing only the necessary bending / twisting / gravity / damping forces on nonrigid control points becomes a simple loop over the intervals in Y. Note that each force must loop over the elements slightly differently; given an interval (i, j) where $i \in Z_k$, the gravitational/damping force needs to be computed on all control points excluding i and j, since the force on those points is generated by the rigid body forces from above. However, the bending/twisting forces need to include the force generated by the three-control-point bending elements centered at both i and j. Because at i this three-element stencil includes two rigid control points ($i-1$ and i), the positions $\mathbf{q}_{i-1}$ and $\mathbf{q}_i$ are always needed to compute the bending/twisting force at $\mathbf{q}_i$; as a result, both $i-1$ and i are included in the set Z_k^{boundary}. A similar argument holds when $j \in Z_k$. The set Y can be quickly computed, and only needs to be updated when the set of rigidified zones changes.



Y = {..., (a,b), (c,d), (e,f), ...}

Figure 5.3: Example of intervals between rigidified zones

Yarn Collisions

Unfortunately, the interval set Y is not directly useful for accelerating the yarn-yarn contact computation, since there may be arbitrary collision complexity involving any yarn point (rigid or non-rigid) and any other yarn point (rigid or non-rigid) or object, regardless of whether the point is within Z_k^{boundary} or not. Moreover, only the collisions entirely internal to Z_k can be ignored; contacts between rigid points in Z_k and nonrigid points, or Z_k and another rigidified zone $Z_{k'}$ must still be processed. Despite these restrictions, these entirely internal self-contacts represent a significant performance cost, and efficiently culling them out represents the bulk of the possible speedups due to rigidification.

As in Chapter 4, a bounding volume hierarchy (BVH) is used to detect yarn-yarn collisions, with spheres instead of axis-aligned bounding boxes. However, the hierarchy is constructed in a specific way, based upon the construction of the rigid zones in Section 5.1.2. Since the curve is a Catmull-Rom spline, every quadrature point in the BVH depends on the positions of at most four control points. For zone Z_k, if a quadrature point depends only on control points in Z_k, it will transform rigidly with the zone when Z_k is rigidified. Because of this, the

BVH is constructed such that all of the quadrature points depending only on Z_K are rooted in a subtree of the BVH, the root of which is denoted as node n_{Z_k}. When Z_k is rigidified, n_{Z_k} is flagged as not needing internal collision processing. Subsequent traversals of the hierarchy to find collisions then can safely skip processing collisions internal to n_{Z_k}, which is detected as checking for collisions between n_{Z_k} and itself. Moreover, because spheres are used instead of bounding boxes, subtrees consisting only of rigidified quadrature points in Z_k will also transform rigidly. As a result, the subtree at n_{Z_k} does not need rebounding before collision processing, and the current positions of any node below n_{Z_k} in the tree are not computed in advance but are instead transformed on-demand by the same rigid transformations being applied to Z_k whenever they are needed to find cross-zone collisions.

Due to this efficiency, all object collisions are handled in a similar way, using the same collision hierarchy as for yarn-yarn collisions and testing each sphere for intersection with all objects in the scene. Because the collision hierarchy is also used for yarn-yarn collisions, it contains several quadrature points per segment, but cloth-object collisions can be much coarser. As a result, for object collision only the spheres corresponding to the vertices of the discrete yarn need to be checked for collisions. In addition, in order to avoid updating the tree multiple times per timestep, all object collisions are treated using a penalty force model, which allows the object contact response to be computed at the same time as yarn self-collisions (and, in fact, share the work of updating the collision tree as needed).

5.2.2 Non-Rigid Damping

Non-rigid damping is treated much as in Section 4.2.3, with the caveat that these damping regions are distinct from the set of rigid zones, and so a damping region may contain parts of one or more rigid zones, each of which may or may not be rigid; this adds complications to the overall computation, but it can still be computed as an applied change to the velocity and angular momentum of a rigidified zone Z_k. There are two speedups to be observed here, though. One is that a damping region entirely internal to a single rigid zone can be safely skipped, since by definition there is no non-rigid motion inside a rigidified zone. The second is that computing the damping requires the same rigid body quantities as the rigid zone, but *only* for the set of control points in the rigid zone that overlap the damping region.

The simplest approach for computing these quantities explicitly computes them by looping over each control point and generating its position. This is inefficient, though, since it may require looping over a significant fraction of control points deep in the rigid zone hierarchy, which each require multiple transformations in order to correctly compute their position in world space. However, the necessary quantities can be transformed and aggregated with the rigid zone hierarchy as zones are rigidified, since both the damping sets and the rigid sets are static so every possible overlap is known in advance. The end result is that each rigid zone maintains the necessary rigid body quantities for all subsets of overlapping damping regions, which are computed upon rigidification from the same quantities stored in the damping region subsets of child zones. While doable, this represents a significant bookkeeping challenge, and since the non-rigid damping is itself an approximation of cloth behavior, it is unclear that it needs to be computed exactly. Rather, a compromise approach applies damping to each region as usual, but if a damping region overlaps a rigidified zone then the entire rigid zone is included in the damping region rather than just the overlapping part. This means that the applied damping will change depending on the current state of rigidification; however, it also makes its computation significantly more efficient and easier since the rigid body variables of the entire zone can be used directly and no bookkeeping is required to keep track of the variables for just the overlapping pieces. In practice, no significant difference in the overall motion was observed with this simplification.

5.2.3 Constraints

Much like internal forces, length constraints entirely inside a rigidified zone Z_k do not need to be enforced; in fact, including them in the constraint solve of Fast Projection produces a rank-deficient matrix since there are no available degrees of freedom along the constraint direction. These constraints can be filtered out efficiently using the same non-rigid interval set Y from Section 5.2.1.

The main complexity lies in handling the coupled constraints, where one control point of a length constraint is in a rigidified zone while the other is not. The position of the control point which is rigidified is dependent on the rigid body variables for the zone, and so the constraint solver must ultimately find acceptable values for the rigid body variables which satisfy all length constraints connected to the zone. This problem was addressed in the rigid body and rod coupling described in Bergou et al. [9], which like the constraint solver discussed in Section 4.3.1 uses the Fast Projection method introduced by Goldenthal et al. [39]. The rest of this section presents a rederivation of this approach that lends itself well to a simulator which allows rapid and simple rigidification / derigification of zones.

Without loss of generality, assume there is a single rigidified zone Z_0, and drop the subscripts associated with the rigid zones. This single rigid zone has a proposed position $\mathbf{x}$ and rotation $\mathbf{r}$, total mass M, and reference frame inertia tensor $\mathbf{I}$. Let there be n control points in the yarn, relabeled such that control points $0 \dots b-1$ are in Z_0, and $b \dots n-1$ are not held rigid. Define $\mathbf{q}_{\text{nonrigid}} = [\mathbf{q}_b \dots \mathbf{q}_{n-1}]$, $\mathbf{M}_{\text{nonrigid}} = \text{diag}(m_b \dots m_{n-1})$, and $\bar{\mathbf{q}} = [\mathbf{r}; \mathbf{x}; \mathbf{q}_{\text{nonrigid}}]$. The problem of constraint satisfaction is then to compute the minimal update (with respect to the metric derived from the kinetic energy of the system) to the positions of all nonrigid control points and the position and rotation of the rigid zone Z_0 such that all constraints are satisfied within a specified tolerance.

The kinetic energy of the system is:

$$(\mathbf{r}^{-1}\omega\mathbf{r})^T\mathbf{I}(\mathbf{r}^{-1}\omega\mathbf{r}) + M\dot{\mathbf{x}}^T\dot{\mathbf{x}} + \dot{\mathbf{q}}_{\text{nonrigid}}^T\mathbf{M}_{\text{nonrigid}}\dot{\mathbf{q}}_{\text{nonrigid}}. \tag{5.10}$$

Note that the angular velocity ω is transformed by the inverse of the current rotation $\mathbf{r}$ to move it into the reference space of the inertia tensor. Since $\dot{\mathbf{r}} = 0.5\omega\mathbf{r}$, this can be rewriten as $\mathbf{r}^{-1}\omega\mathbf{r} = 2\mathbf{r}^{-1}\dot{\mathbf{r}}$, producing the same 'generalized velocity' $\mathbf{y} = [\mathbf{r}^{-1}\dot{\mathbf{r}}, \dot{\mathbf{x}}, \dot{\mathbf{q}}_{\text{nonrigid}}]$ from Bergou et al. [9], allowing the kinetic energy to be expressed in matrix form as

$$\begin{bmatrix} (\mathbf{r}^{-1}\dot{\mathbf{r}})^T & \dot{\mathbf{x}}^T & \dot{\mathbf{q}}_{\text{nonrigid}}^T \end{bmatrix} \begin{bmatrix} 4\mathbf{I} & 0 & 0 \\ 0 & M\mathbf{E}_{3\times3} & 0 \\ 0 & 0 & \mathbf{M}_{\text{nonrigid}} \end{bmatrix} \begin{bmatrix} \mathbf{r}^{-1}\dot{\mathbf{r}} \\ \dot{\mathbf{x}} \\ \dot{\mathbf{q}}_{\text{nonrigid}} \end{bmatrix} \equiv \mathbf{y}^T\mathbf{M}\mathbf{y} \tag{5.11}$$

Given this, it is a matter of simply re-applying the derivation of Fast Projection from Goldenthal et al. [39], which results in a pair of equations:

$$0 = \mathbf{M}\mathbf{y}^{j+1} + h\mathbf{A}^T\nabla\mathbf{C}(\bar{\mathbf{q}}^j)^T\lambda^{j+1}$$

$$0 = \mathbf{C}(\bar{\mathbf{q}}^j) + h\nabla\mathbf{C}(\bar{\mathbf{q}}^j)\mathbf{A}\mathbf{y}^{j+1}$$

where superscripts denote the iteration of the Fast Projection algorithm, $\mathbf{A} = \begin{bmatrix} \mathbf{Q} & 0 \\ 0 & \mathbf{E}_{n-b+3 \times n-b+3} \end{bmatrix}$, and $\mathbf{Q}$ is the 4×3 matrix that multiplies a purely imaginary quaternion by the quaternion $\mathbf{r}^j$. Solving for $\mathbf{y}^{j+1}$ in the first equation and substituting into the second gives the following three equations:

$$\mathbf{C}(\bar{\mathbf{q}}^j) = h^2 \left(\nabla \mathbf{C}(\bar{\mathbf{q}}^j) \mathbf{A} \mathbf{M}^{-1} \mathbf{A}^T \nabla \mathbf{C}(\bar{\mathbf{q}}^j)^T \right) \lambda^{j+1} \tag{5.12}$$

$$\mathbf{y}^{j+1} = -h \mathbf{M}^{-1} \mathbf{A}^T \nabla \mathbf{C}(\bar{\mathbf{q}}^j)^T \lambda^{j+1} \tag{5.13}$$

$$\bar{\mathbf{q}}^{j+1} = \bar{\mathbf{q}}^j + h \mathbf{A} \mathbf{y}^{j+1} \tag{5.14}$$

To solve for the new projected positions, the first equation is solved for λ^{j+1}, which involves a linear solve of an SPD system, with that solution substituted into the second equation to solve for $\mathbf{y}^{j+1}$, which is substituted into the third equation to obtain the updated $\bar{\mathbf{q}}^{j+1}$. Also, note that the constraint gradient $\nabla \mathbf{C}(\bar{\mathbf{q}}^j)$ is with respect to $\bar{\mathbf{q}}^j$. Via the chain rule, this can be broken up into two gradients so that $\nabla \mathbf{C}(\bar{\mathbf{q}}^j) = \nabla \mathbf{C}_{\mathbf{q}} \nabla \mathbf{P}_{(\mathbf{r}, \mathbf{x}, \mathbf{q}_{\text{nonrigid}})}$, where the function $\mathbf{P}()$ describes the position of each yarn point with respect to the variables $\mathbf{r}, \mathbf{x}, \mathbf{q}_{\text{nonrigid}}$:

$$\mathbf{p}_i = \mathbf{x} + \mathbf{r} \mathbf{d}_i \mathbf{r}^{-1} \qquad \text{for } i = 0 \ldots b-1$$

$$\mathbf{p}_i = \mathbf{q}_i \qquad \text{for } i = b \ldots n$$

Define

$$\mathbf{F} = \begin{bmatrix} 0.5\mathbf{R}^j[\mathbf{d}_1] & \mathbf{E}_{3\times3} & 0 & 0 & \ldots & 0 \\ 0.5\mathbf{R}^j[\mathbf{d}_2] & \mathbf{E}_{3\times3} & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.5\mathbf{R}^j[\mathbf{d}_m] & \mathbf{E}_{3\times3} & 0 & 0 & \ldots & 0 \\ 0 & 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & 1 \end{bmatrix}$$

where $[\mathbf{d}_i]$ is the 3×3 matrix such that $[\mathbf{d}_i]\mathbf{b} = \mathbf{d}_i \times \mathbf{b}$ for all vectors $\mathbf{b}$. Define $\tilde{\mathbf{M}}^{-1} = \mathbf{F}\mathbf{M}^{-1}\mathbf{F}^T$. It can be shown symbolically that $\nabla\mathbf{P}\mathbf{A}\mathbf{M}^{-1}\mathbf{A}^T\nabla\mathbf{P}^T = \tilde{\mathbf{M}}^{-1}$, and so 5.12 can be rewritten as:

$$\mathbf{C}(\mathbf{q}^j) = h^2 \left(\nabla\mathbf{C}(\mathbf{q}^j)\tilde{\mathbf{M}}^{-1}\nabla\mathbf{C}(\mathbf{q}^j)^T\right) \lambda^{j+1}. \tag{5.15}$$

Note that this is expressed over the set of all control points $\mathbf{q}$, and not the reduced rigid state $\bar{\mathbf{q}}$. As a result, solving for the constraints on the yarn involves two simple steps. The solver first ignores any constraint entirely within a rigid zone. For the remaining constraints, the solver merely needs to compute the derivative of each constraint with respect to each control point, regardless of whether the control point is held rigid or not. If that point is moving rigidly in some zone, it will then be further filtered by $\tilde{\mathbf{M}}^{-1}$ to account for the rigidity. Thus, the simulator can quickly enable and disable rigidity in zones by changing the filtering properties of $\tilde{\mathbf{M}}^{-1}$ which is trivial since the filtering is controlled by the matrix $\mathbf{F}$.

This method does have one notable limitation as currently derived: there is no penalizing of modes which cause the rotation quaternion to stretch. As a result, the update may cause the quaternion to deviate from unit-norm over time. However, the update to $\mathbf{r}$ will always be orthogonal to $\mathbf{r}$, which is the first order approximation of maintaining unit-norm. As a result, while $\mathbf{r}$ will in general not be unit length after the update, because the simulator timesteps are so small in practice the deviation from unit-norm is small per iteration, and the quaternion can simply be normalized after each iteration. This is not entirely physically accurate, though, and a more complete implementation with larger timesteps might need to correct for this. For instance, the actual implementation of [9] uses a 4×4 matrix for the rotational component of the mass matrix, $\left[\begin{smallmatrix} 1 & 0 \\ 0 & \mathbf{I} \end{smallmatrix}\right]$, and $\mathbf{Q}$ in $\mathbf{A}$ is the 4×4 matrix that multiplies a (not necessarily purely imaginary) quaternion by the quaternion $\mathbf{r}^{-1}$. Furthermore, an additional constraint is added to the system which ensures that $\mathbf{r}$ remains unit length. This necessitates adding in an additional constraint per-rigid zone to the solve, and also eliminates the simple re-expression of the rigid body constraints as a filter on the mass matrix since this constraint cannot be expressed as being applied to $\mathbf{q}$. Moreover, in practice using just the mass-matrix filter without this additional constraint sees convergence of the constraints to a tolerance of 10^{-5} in 1 - 2 iterations per timestep.

5.3 Rigidification Oracles

So far, it has been assumed that the set of rigidified zones is already known to the simulator. In practice, however, the simulator must update this set of rigid zones over time, a task assigned to an oracle which, given the current configuration of the simulator, rigidifies or derigidifies zones as necessary to maintain quality as specified by some user-supplied parameters. In addition, this oracle should have a number of additional properties. Because the simulation does not rewind, the oracle should quickly respond to changes in applied loads. Moreover, due to the large number of timesteps it should be relatively inexpensive to run. Finally, it should also take advantage of the rigid hierarchy and not require traversing all the way down to the leaves to verify rigidity for an ancestor.

All of the tested oracles define the set of currently rigid zones as a cut through the hierarchy of zones, with the zones on the cut predicted to be rigid and all zones below the cut rigidifed and all above non-rigid. The zones on the cut are zones that are predicted to be rigid, but they will not be simulated as actually rigid. Rather, simulating them as nonrigid gives the oracle valuable information about its prediction and allows it to detect when the prediction is no longer valid. In addition, the oracle enforces a maximum one-level difference in the current set of rigid nodes in the hierarchy; this allows changes detected at one node to rapidly propagate in a local region around the change.

5.3.1 Rigidification

Detecting that a zone can be rigidified is significantly easier than detecting when a rigidified zone needs to be derigidified. This is done by computing a simple rigidity metric on zones that measures deviations from rigidity with respect to a reference configuration, which is updated over time. Zone Z_k is checked for rigidity if all of its children are predicted to be rigid and it can rigidify based on the maximum one-level difference. Checking for rigidity involves storing a reference configuration for Z_k, which consists of the positions of all points in Z_k^{local} and the positions and rotations of all zones in $\textbf{child}(Z_k)$, and then evaluating the rigidity metric h_{rigid} simulation seconds later; if it is less than a user-defined tolerance τ_{rigid}, Z_k is rigidified and the cut is collapsed to the parent zone. If Z_k is not moving rigidly, the reference configuration is set to be the current configuration, and the rigidity is again estimated after h_{rigid} seconds.

In order to compute the rigidity of Z_k, shape matching is used to estimate the best rigid transformation of the zone [77, 95]. For each point $i \in Z_k$, there is a reference position $\mathbf{q}_i^0$ and current position $\mathbf{q}_i$. In order to efficiently compute this transformation it should take advantage of the rigid hierarchy—if Z_k contains a child rigid zone Z_c, the best rigid transformation should be computed without directly examining the points inside Z_c. Rather, it should be computed using easily precomputed aggregate quantities over Z_c as well as the points in Z_k^{local}. As shown in the remainder of this section, this is done by substituting in the rigid position for all points in Z_c and reordering summations to compute over the rigid body variables of Z_c.

The two best translations are just the center of mass of the set of points in both the initial and final positions, which can be easily computed using the center of mass equation (5.2) for both the reference configuration and the current configuration, giving $\mathbf{x}^0_{Z_k}$ and $\mathbf{x}_{Z_k}$ respectively.

The best rotation is found by computing the polar decomposition of the matrix [77]:

$$\mathbf{A}_{Z_k} = \sum_{i \in Z_k} m_i(\mathbf{q}_i - \mathbf{x}_{Z_k})(\mathbf{q}^0_i - \mathbf{x}^0_{Z_k})^T. \tag{5.16}$$

Expanding this summation to separate points in Z^{local}_k from points in child zones of Z_k gives:

$$\begin{aligned}\mathbf{A}_{Z_k} = & \sum_{i \in Z^{\text{local}}_k} m_i(\mathbf{q}_i - \mathbf{x}_{Z_k})(\mathbf{q}^0_i - \mathbf{x}^0_{Z_k})^T + \\ & \sum_{Z_c \in \textbf{child}(Z_k)} \sum_{j \in Z_c} m_j(\mathbf{q}_j - \mathbf{x}_{Z_k})(\mathbf{q}^0_j - \mathbf{x}^0_{Z_k})^T \\ = & \sum_{i \in Z^{\text{local}}_k} m_i(\mathbf{q}_i - \mathbf{x}_{Z_k})(\mathbf{q}^0_i - \mathbf{x}^0_{Z_k})^T + \sum_{Z_c \in \textbf{child}(Z_k)} \mathbf{A}'_{Z_c},\end{aligned} \tag{5.17}$$

Let Z_c be a child rigid zone of Z_k (which must be moving rigidly if the rigidity of Z_k is being checked). Then the positions of all control points in Z_c can be expressed in terms of the rigid body variables instead. Thus, for $j \in Z_c$, if $\mathbf{d}_j$ is the relative position of control point j in Z_c, then $\mathbf{q}^0_j = \mathbf{R}^0_{Z_c}\mathbf{d}_j + \mathbf{x}^0_{Z_c}$ and $\mathbf{q}_j = \mathbf{R}_{Z_c}\mathbf{d}_j + \mathbf{x}_{C_k}$, which, when substituted into $\mathbf{A}'_{Z_c}$ from Equation (5.17), allows $\mathbf{A}'_{Z_c}$ to be re-expressed as:

$$\begin{aligned}\mathbf{A}'_{Z_c} &= \sum_{j\in Z_c} \mathbf{R}_{Z_c}(m_j \mathbf{d}_j \mathbf{d}_j^T)(\mathbf{R}^0_{Z_c})^T \\ &+ \sum_{j\in Z_c} \mathbf{R}_{Z_c} m_j \mathbf{d}_j (\mathbf{x}^0_{Z_c} - \mathbf{x}^0_{Z_k})^T \\ &+ \sum_{j\in Z_c} (\mathbf{x}_{Z_c} - \mathbf{x}_{Z_k}) m_j \mathbf{d}_j^T (\mathbf{R}^0_{C_k})^T \\ &+ \sum_{j\in Z_c} m_j (\mathbf{x}_{Z_c} - \mathbf{x}_{Z_k})(\mathbf{x}^0_{Z_c} - \mathbf{x}^0_{Z_k})^T \\ \mathbf{A}'_{Z_c} &= \mathbf{R}_{Z_c} \sum_{j\in Z_c} (m_j \mathbf{d}_j \mathbf{d}_j^T)(\mathbf{R}^0_{Z_c})^T + M_{Z_c}(\mathbf{x}_{Z_c} - \mathbf{x}_{Z_k})(\mathbf{x}^0_{Z_c} - \mathbf{x}^0_{Z_k})^T\end{aligned} \tag{5.18}$$

Note that the middle two terms in the expansion are zero because by definition $\sum m_j \mathbf{d}_j = \mathbf{0}$. Note also that this requires storing the mass-weighted sum of the outer product of the relative position of all points in the zone relative to the center of mass of the zone $\sum_{j\in Z_c} (m_j \mathbf{d}_j \mathbf{d}_j^T)$; fortunately, this quantity can also be computed and stored recursively per-zone. Given this, the matrix $\mathbf{A}_{Z_k}$ can then be formed as the sum of all $\mathbf{A}'_{Z_c}$ for child zones as well as additional non-child points in Z_k^{local}, without descending further down the hierarchy of zones.

Given the best translations $\mathbf{x}^0_{Z_k}$ and $\mathbf{x}_{Z_k}$, and the best rotation $\mathbf{R}_{Z_k}$, it is then necessary to compute the deviation from rigidity over the specified time period. An efficient metric for doing so is:

$$\textbf{rigidity}(Z_k) = \frac{1}{W_{Z_k} h_{\text{rigid}}} \sqrt{\frac{\sum m_i \|\mathbf{R}_{Z_k}(\mathbf{q}^0_i - \mathbf{x}^0_{Z_k}) + \mathbf{x}_{Z_k} - \mathbf{q}_i\|_2^2}{M_{Z_k}}}, \tag{5.19}$$

where W_{Z_k} is a weight, with units of distance, measuring the size of zone Z_k; in the implementation, this is taken to be the radius of the collision hierarchy bounding sphere n_{Z_k}. This can be interpreted as measuring the average non-rigid velocity of each point inside Z_k over time time h_{rigid}, but it is particularly efficient because much like $\mathbf{A}_{Z_k}$ it can be computed over aggregate quantities on the child zones and Z_k^{local} instead of requiring a full loop over Z_k. The derivation of the expansion is computed in the same way as $\mathbf{A}_{Z_k}$, via substitution of the rigid body positions of $\mathbf{q}_i$ for all points inside child zones Z_c, but it is not as simple as $\mathbf{A}_{Z_k}$. Code for computing it efficiently can be generated using a symbolic toolkit, e.g. Maple or Mathematica, and it only requires storing the same aggregate quantities (outer product sum) as $\mathbf{A}_{Z_k}$. After evaluating this metric, if **rigidity**$(Z_k) < \tau_{\text{rigid}}$, then the zone is marked as rigid and the cut through the hierarchy of rigid zones is updated to reflect the change.

This gives us a simple to evaluate metric for determining the rigidity of a given zone as measured from some reference configuration. It meets the goals stated at the beginning of the section, in that it depends only on control points in Z_k^{local}, rigid body variables of zones in **child**(Z_k), and easily computable aggregate quantities over zones in **child**(Z_k) that can be computed and stored when the zone is rigidified. Moreover, as the next section shows, it also provides a method for detecting when to derigidify a zone.

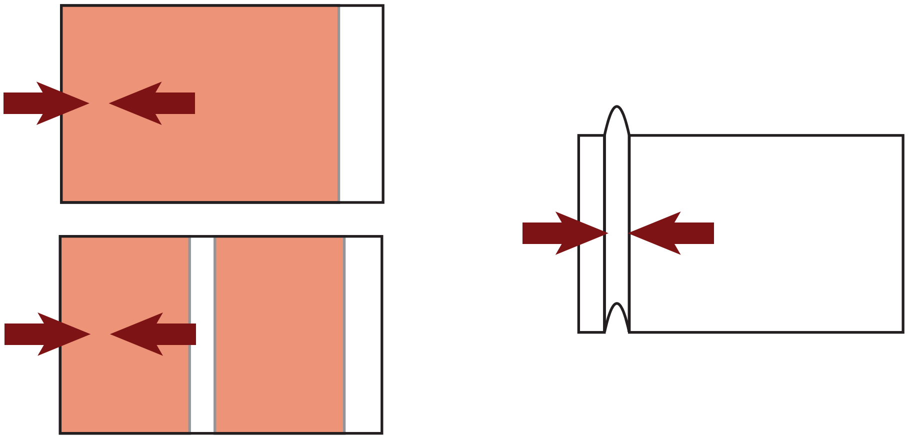
5.3.2 Basic Derigidifier

The metric in equation (5.19) can also be used for derigidification. The key idea is that the rigid zones on the cut are *predicted* to be rigid, but they do not necessarily have to be *simulated* rigidly, and in fact simulating them nonrigidly can give the simulator valuable information on how accurate its predictions are. Suppose zone Z_k has been determined to be rigid and is now on the current cut of active rigid zones. Z_k will not be simulated rigidly, however. Instead, when Z_k is determined to be transforming rigidly, the current configuration of Z_k is stored as the reference configuration, and on every timestep the metric in equation (5.19) is evaluated. Because the current configuration is never updated, there is no need to scale for time, and so the scale factor of $\frac{1}{h_{\text{rigid}}}$ is removed. As a result, the metric is measuring the total deviation from rigidity since Z_k was rigidified, and not the average rate of rigidity change, as was being measured in Section 5.3.1. Once this metric goes above τ_{nonrigid}, Z_k is no longer predicted to be rigid, the cut of active rigid zones is expanded to include the children of Z_k, and the children of Z_k are now derigidified (since they are now on the active cut, they are now only predicted to be rigid).

It is worth expanding on the reasons for why the reference configuration is fixed at the time of rigidification. If instead the reference configuration for zone Z_k is regularly updated while it is predicted to be rigid, then this measures the rate of change of rigidity over time; once the zone begins deforming at a rate faster than τ_{nonrigid}, the zone is no longer treated as rigid. Because this will not break rigid zones under slow but constant nonrigid deformation, it is better to instead fix the reference configuration at the time Z_k is determined to be moving rigidly, and remove the scale factor of time from the metric. This will then cause zones to break once they undergo some total amount of nonrigid deformation, regardless of the period of time over which it occurs, and in general functions better for preserving the quality of the final results.

5.3.3 Impulse Derigidifier

The above rigidifier/derigidifier works well for detecting slow changes in the rigid state that are occurring over a length of time. However, it does not do as well for rapid and unpredictable changes in the rigid state, such as those caused by object collisions. The predictor works by analyzing the difference in the motion of two levels of the hierarchy, but if those two levels are high up in the hierarchy then they both may be equally bad at capturing the true



Two levels in rigid simulation

Unreduced simulation result

Figure 5.4: Failure case for rigid predictor

nonrigid behavior. In particular, if the bulk of the cloth is rigidified, some object impulses may not cause sufficient nonrigid deformation in time to derigidify before the quality of the simulation degrades significantly. One notable and common case is cloth rigidified into a planar sheet and an in-plane impulse force, where the correct buckling behavior occurs entirely inside a rigidified zone and is not detected. Figure 5.4 illustrates how this is missed by the two-level predictor, while Figure 5.5 contains six frames of a falling scarf where this behavior occurs. The scarf quickly rigidifies (shown in red) to the top hierarchy level, while the object contact generates a force in the plane of the scarf. While the scarf would normally buckle in the absence of rigidification, here the scarf does not undergo sufficient nonrigid movement to derigidify, and as a result the scarf instead topples sideways like a solid beam rather than a soft piece of fabric.

Ultimately, detecting these failure cases is extremely difficult, in particular since they arise largely as a result of certain contacts and collisions which are difficult to anticipate. Many different approaches were tested, with none of them completely addressing the problem. The most successful of these approaches

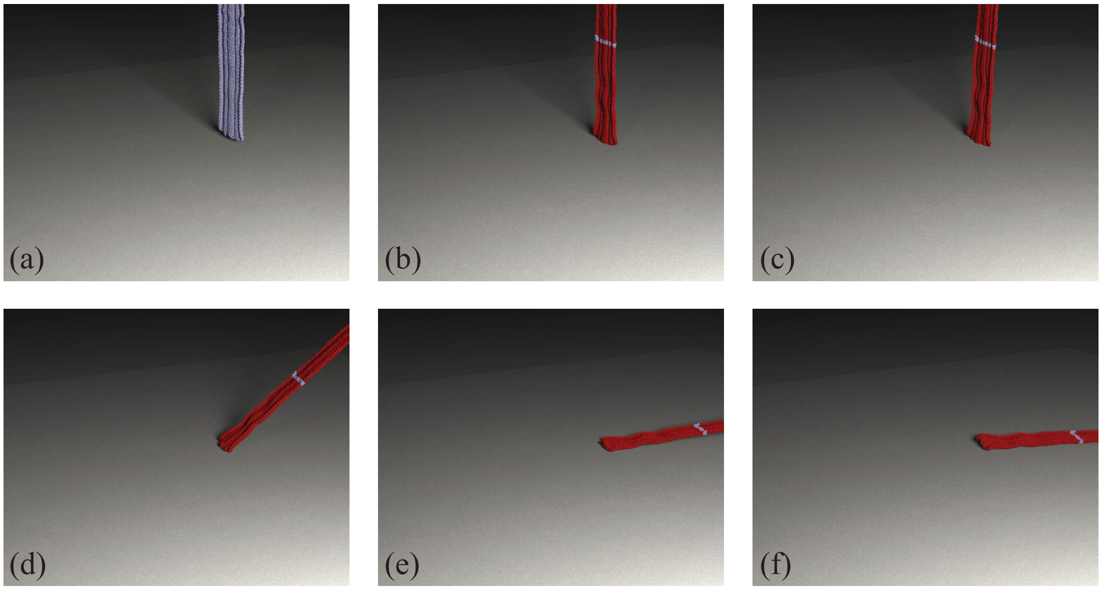


Figure 5.5: Falling scarf without impulse derigidification

was the impulse derigidifier. At a high level, for each zone Z_k on the cut of active rigid zones the impulse derigidifier measures the deviation of each point $j \in Z_k$ from the expected position of the point if the model had been simulated as if *no zones* were rigid in the simulator.

More formally, let Z_k be some zone in the active rigid cut, and $j \in Z_k$ be arbitrary. At the start of the timestep, control point j has starting position $\mathbf{q}_j^0$ and velocity $\dot{\mathbf{q}}_j^0 = \mathbf{v}_{Z_k}^0 + \omega_{Z_k}^0 \times \mathbf{R}_{Z_k}^0 \mathbf{d}_j$, and at the end of the timestep new position $\mathbf{q}_j^n = \mathbf{q}_j^0 + h\dot{\mathbf{q}}_j^n$ and velocity $\dot{\mathbf{q}}_j^n = \mathbf{v}_{Z_k}^n + \omega_{Z_k}^n \times \mathbf{R}_{Z_k}^n \mathbf{d}_j$. Over the timestep, there have also been forces and impulses, both internally and externally generated, applied at control point j which induced acceleration or torque on zone Z_k; these are accumulated into the total change in velocity $\Delta\dot{\mathbf{q}}_j$. Thus, if control point j were not held rigid, the expected new position would have been $\mathbf{q}_j^f = \mathbf{q}_j^0 + h(\dot{\mathbf{q}}_j^0 + \Delta\dot{\mathbf{q}}_j)$.

The deviation from rigidity over the timestep for control point j is then:

$$\mathbf{n}_j = \frac{1}{h}\left(\mathbf{q}_j^f - \mathbf{q}_j^n\right) \tag{5.20}$$

$$= \mathbf{v}_{Z_k}^0 + \omega_{Z_k}^0 \times \mathbf{R}_{Z_k}^0 \mathbf{d}_j + \Delta\dot{\mathbf{q}}_j - \left(\mathbf{v}_{Z_k}^n + \omega_{Z_k}^n \times \mathbf{R}_{Z_k}^n \mathbf{d}_j\right) \tag{5.21}$$

and the associated metric over the entire zone Z_k is

$$\textbf{vel\_rigidity}(Z_k) = \frac{1}{W_{Z_k}}\sqrt{\frac{\sum\limits_{j \in Z_k} m_j \|\mathbf{n}_j\|_2^2}{M_{Z_k}}} \tag{5.22}$$

Essentially, this is measuring the difference in motion caused by the rigidity. This difference can be manifested in several ways: for instance, how much of the impulse at point j got thrown away because e.g. it cancelled out with another impulse at another point, as well as how much final velocity is unaccounted for because e.g. it was produced by an impulse at another point and got distributed through the rigid body to point j. If this value crosses a tolerance τ_{vel}, the zone is immediately derigidified at the end of the timestep. Note the similarities to the rigidity metric (5.19); both measure the difference between expected positions and actual positions, mass-weighted and scaled over the period of time. In addition, much like the rigidity metric, the equation can be expanded using a symbolic toolkit and computed in terms of aggregate quantities, in this case $\sum_j m_j \Delta\dot{\mathbf{q}}_j \mathbf{R}_{Z_k} \mathbf{d}_j^T$, $\sum_j m_j \Delta\dot{\mathbf{q}}_j$, and $\sum_j m_j \Delta\dot{\mathbf{q}}_j^T \Delta\dot{\mathbf{q}}_j$. The results of this derigidifier can be seen in Figure 5.6. Note that as the scarf collides with the ground plane, the zones in the area are immediately derigidified and the cloth begins to buckle as expected, while zones higher up the cloth remain rigidified.

However, unlike with the rigidity metric, there are important limitations and drawbacks with (5.22). The first is simply the difficulty of correctly implementing it: carefully accounting for all impulses on all control points in the simulation is challenging, and any missed impulses induce significant errors in the final computation. Note also that the impulse must be as if the zone were not

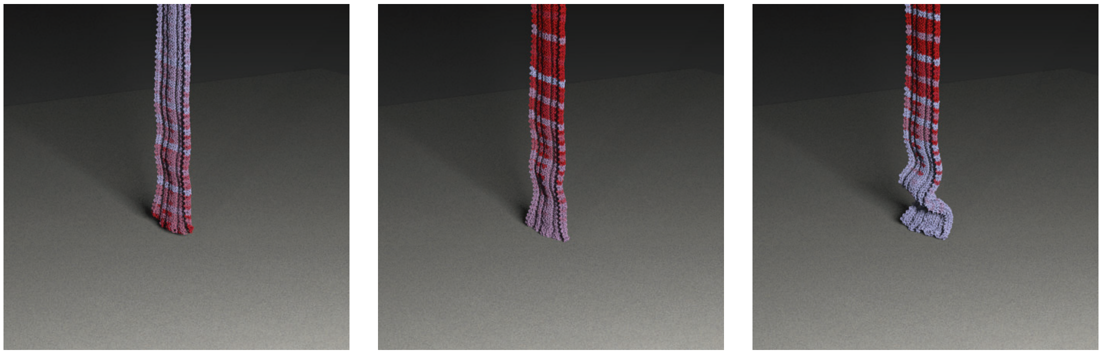


Figure 5.6: Falling scarf with impulse derigidification

being simulated rigidly, which means it must include those forces discussed in Section 5.2 which did not need to be explicitly computed for the rigid zone because they contributed net zero force to the rigid zone. Because these forces rigidly transform with the zone, they can be computed when the zone is rigidified, but there is still the problem of correctly accounting for impulses (like object contact friction) which depend on other impulses which may or may not be normally included depending on rigidity. All in all, it represents a significant and difficult engineering challenge to simply get near to a correct computation.

More importantly, this estimator is evaluated on every timestep, since it needs to quickly respond to changes in applied loads. While this is good for rapid response, it also means that this is a rather noisy estimator, and tuning parameters such that it adequately responds while not overreacting and aggressively derigidifying zones is difficult. Finally, although the estimator can be computed using aggregate quantities, in a typically structured integrator it is difficult to compute $\sum_j m_j \Delta\dot{\mathbf{q}}_j^T \Delta\dot{\mathbf{q}}_j$ as a running sum, since the applied impulse to a given control point is usually computed as a series of impulses, and computing the mass-weighted sum of squared impulses without looping over all control points in Z_k is problematic. In the current implementation, this loop

parameter	description	value
τ_{rigid}	rigidification tolerance	1.5 – 10 $\frac{1}{s}$
τ_{nonrigid}	derigidification tolerance	$0.002 – 0.013$
τ_{vel}	velocity impulse tolerance	10 $\frac{1}{s}$
h_{rigid}	rigid test frequency	$10h$
h	timestep	1/24000s
b	quadrature points / seg.	11
k_{contact}	contact stiffness	3000
r	yarn radius	0.125 cm

Table 5.1: Parameters used during rigid simulation.

over all control points is done, and it represents the only computation which does not take advantage of the rigid hierarchy. However, this loop is extremely fast since it merely needs to square the applied impulse at each control point, without applying any other transformations, and so in practice it does not appear to represent a performance bottleneck.

5.4 Results

The rigidification code was implemented on top of the simulator discussed in Chapter 4, using the Discrete Elastic Rods implementation of Section 4.5 but not the further model improvements, and is written in Java and multithreaded. Results were generated on the same machines as in Chapter 4. Parameter settings are specified in Table 5.1. In all images, red corresponds to the frequency with which rigidification allowed the self-contact force computation to be skipped for that piece of yarn; thus, the brighter and more frequent the red, the more the expected speedup in the final result.

Figure 5.7 shows an animation of a scarf falling on a ground plane and then being picked up again. There are many opportunities for rigidification to provide acceleration; however, the derigidifier must also be proactive in order to maintain quality. Unfortunately, the noisiness in the impulse estimator both prevents possible speedups and causes degraded quality in the final motion. The sharp impulse with the ground plane is enough to correctly trigger the derigidifier, but when laying at rest on the ground the impulse estimator is overeager and continually derigidifies, not allowing the hierarchy to rigidify to the highest levels. At the same time, because of this noisiness τ_{vel} must be set relatively high in order to allow even some rigidification, which means the estimator does not respond entirely to the low speed unfolding behavior as the scarf is lifted. Zones are derigidified as it is lifted, but the boundaries between rigidified zones are still sharply visible as the hierarchy is never completely refined to the finest level, which results in a significant reduction in the quality of the simulation. Turning τ_{vel} down causes the velocity impulse estimator to be even more overeager about derigidifying zones otherwise at rest, while turning it up causes the simulator to fail to derigidify at all when it the scarf is being lifted.

Even excluding the concerns over quality, the expected performance gains do not entirely materialize. Figure 5.8 shows the relative performance gain for a blanket being simulated at various levels of rigidity in the hierarchy, normalized to the cost of the unreduced model. Note that rigidifying at the leaves of the hierarchy represents a small overall performance boost; this is due to the fact that only a small fraction of the self-contact force computation can be skipped. As it proceeds to higher rigidity levels the performance increases, but remains less than 10 times faster than the unreduced model; this is most likely due to the presence of the seams, where computing the self-contact force requires traversing relatively deep in the bounding tree of quadrature points even if it rarely gets to the leaves (the actual quadrature points). A 10x performance boost would still be notable, if it did not require almost the entire cloth to be rigidified; such situations would only be likely for cloth at rest. Looking at more likely rigidification scenarios, in the level-2 to level-3 range, results in a significantly more modest 2x to 3x speedup. This is in general borne out by the larger-scale examples in Figure 5.9, where the cloth generally fails to rigidify past the first few levels, and the overall speedup is modest.

5.5 Lessons Learned

Rigidification is an interesting idea, but this analysis seems to indicate there are too many drawbacks to be an effective acceleration technique for cloth simulation. Detecting when and where to derigidify is a challenging problem, made more so by the fact that many interesting cloth behaviors arise as a result of contact forces, which are difficult to account for in reduced models. Although additional solutions for this could be pursued, it is unlikely that most simulations would have enough rigid regions to achieve a significant performance boost.

One possible conclusion to be drawn from this approach is that reduced models are still a perfectly reasonable choice for accelerated cloth simulation, but that a reduced rigid model is unsuitable. However, the goal is accelerating the simulation, and evaluating the self-contact force was the most expensive part by far. Rigidity was a useful reduced model precisely because it could easily eliminate the need to evaluate portions of the self-contact force. More complicated reduced models might require estimation of the contact force or material stiffness parameters, reducing the overall speedup possible. Although optimized cubature algorithms [1] could be used in conjunction with a tailored reduced model that updates on the fly [65], it is unlikely that such an approach could be trained sufficiently to adequately respond to the wide spectrum of contact behaviors that may arise during simulation. As a result, the next chapter will approach this problem from a different tack by looking to directly accelerate the contact computation via approximations.

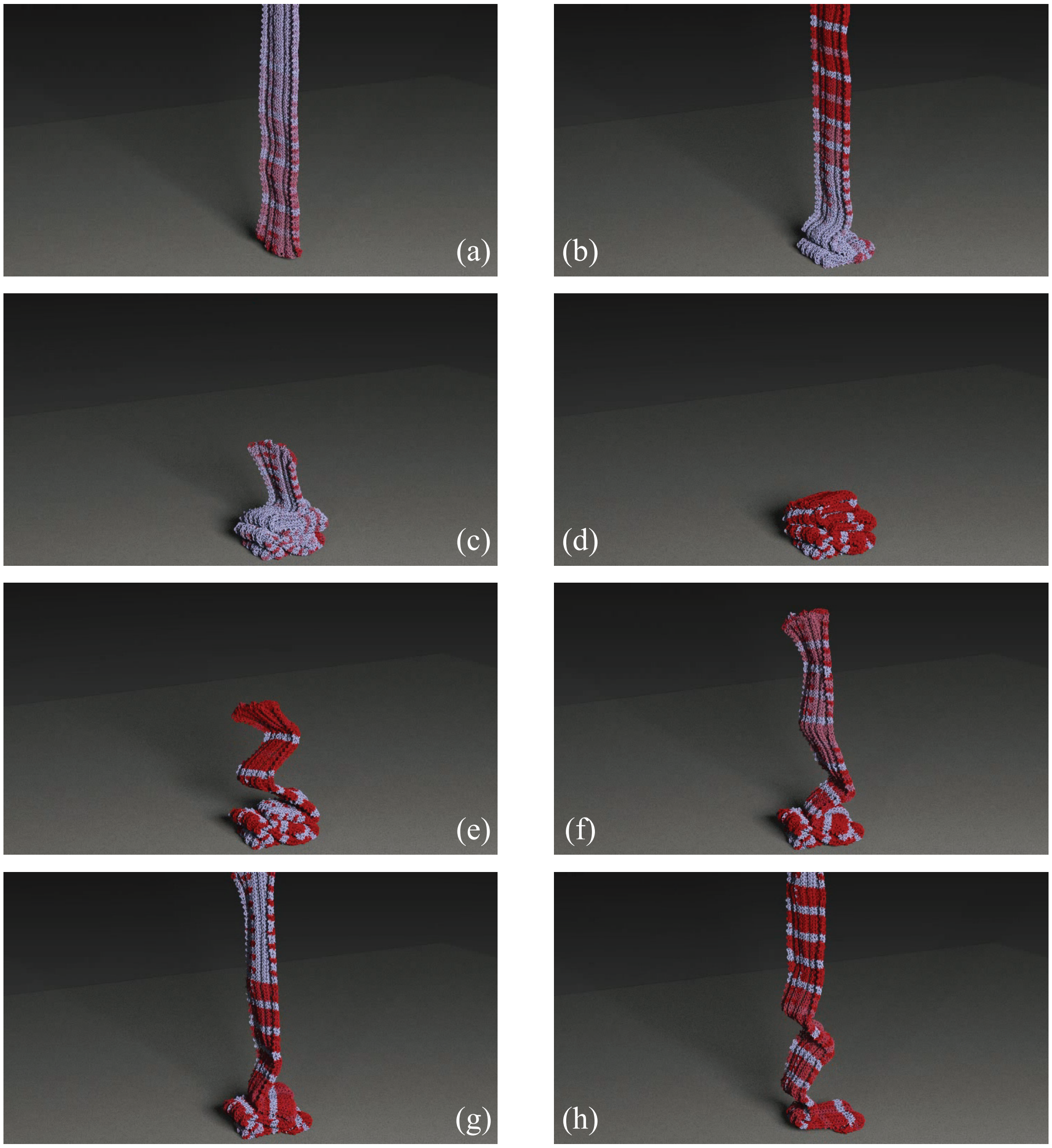


Figure 5.7: Scarf falling on a plane and then being picked up again

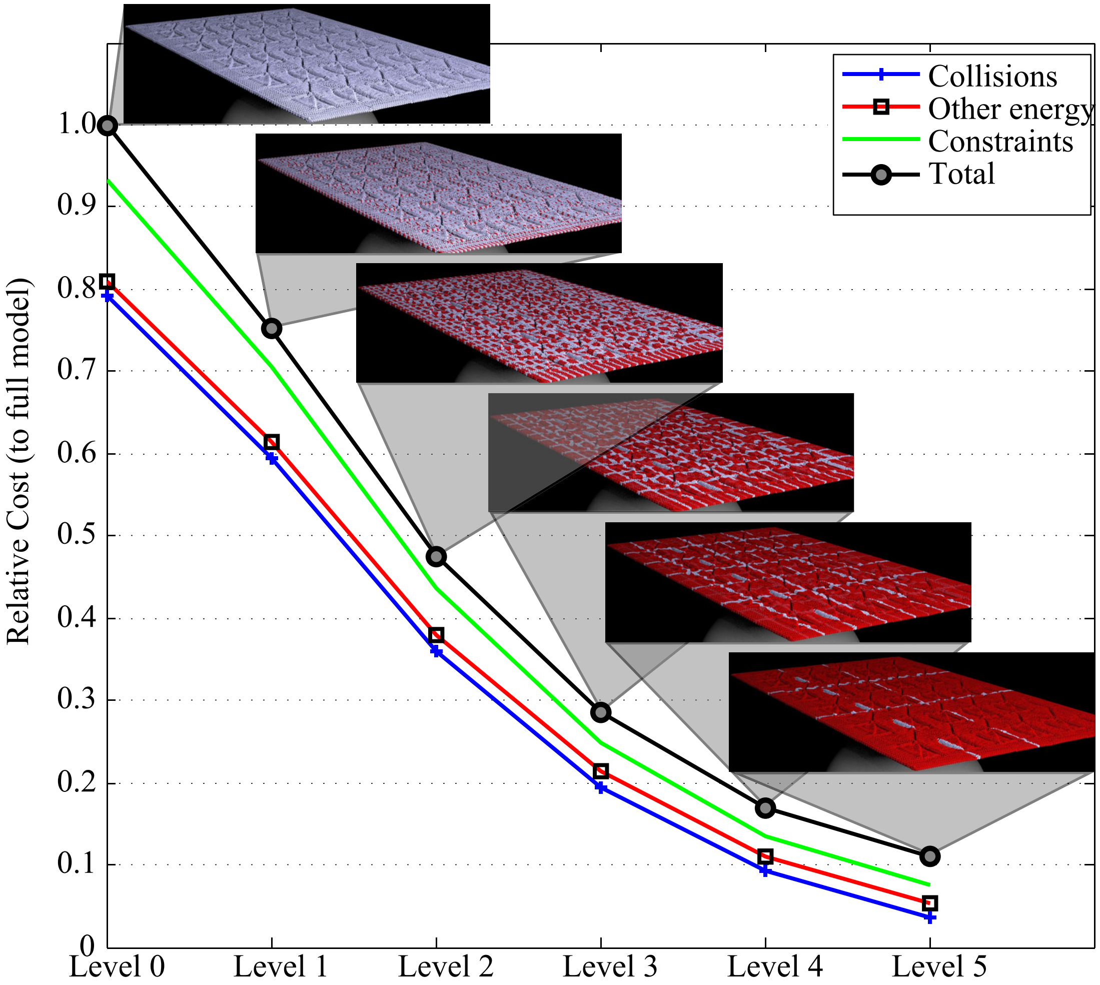


Figure 5.8: Performance scaling for blanket at various levels in the rigid hierarchy

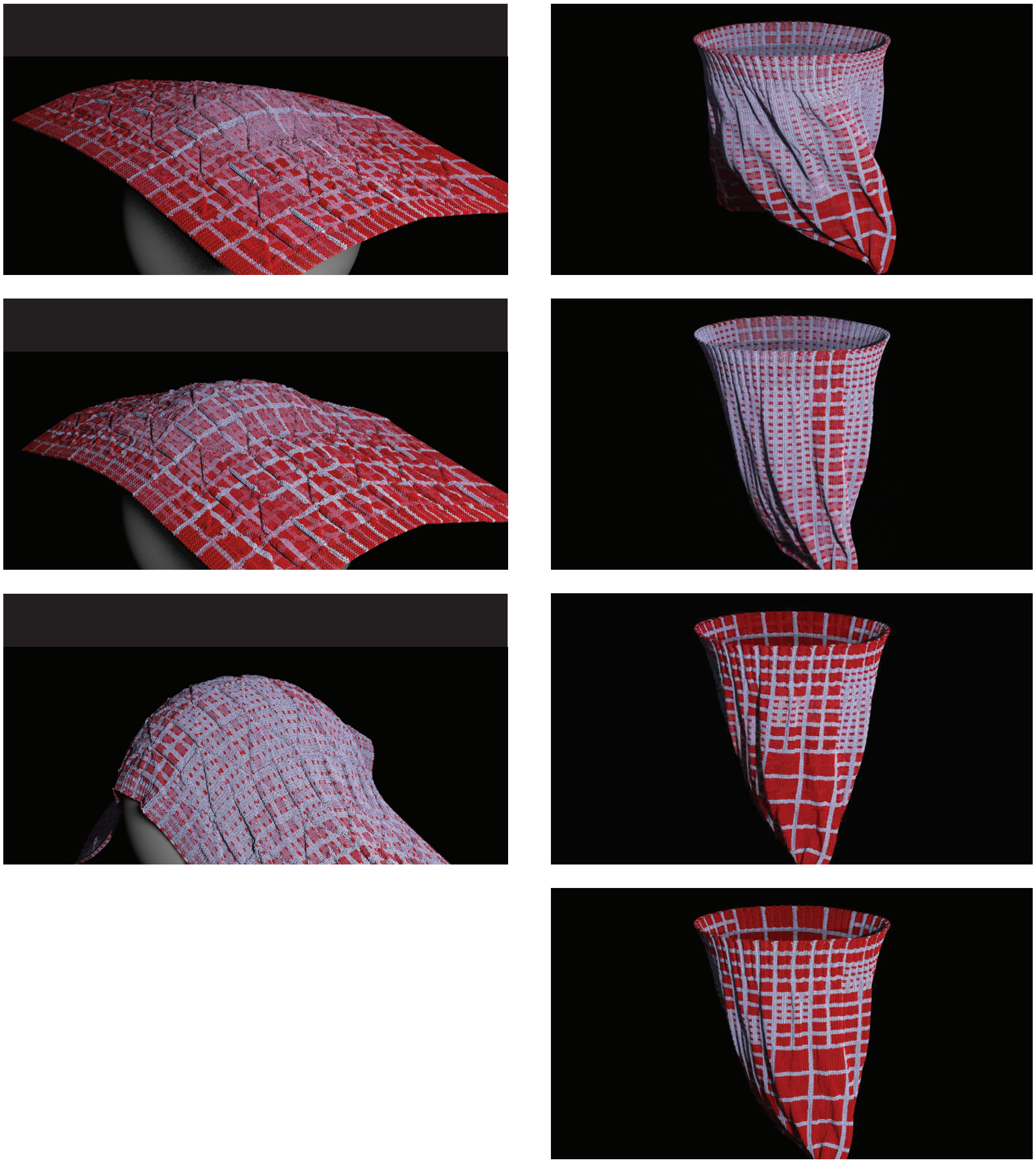


Figure 5.9: Frames from falling blanket and sack examples

CHAPTER 6

ADAPTIVE CONTACT LINEARIZATION

As discussed in Chapter 5, evaluating contact response in yarn models is the simulation bottleneck, but oftentimes that costly computation is expended only to determine that some simpler behavior is occurring, e.g. the cloth isn't moving. Model reduction techniques such as the one explored in the previous chapter look at this in the context of motion in a reduced space, for instance the space of local rigid transformations, and then computing an exact force response within this reduced configuration space while determining when the reduced model is no longer valid. Unfortunately, this space of rigid transformations is too restrictive to effectively exploit, and it seems likely that other types of reduced models would similarly struggle. The contact mediated structure and widespread self-contact make it difficult to determine when the system is straying outside of the space of motions available in the reduced space without degraded mechanics or rewinding of the simulation.

Instead of computing an exact force response in a reduced space, though, consider computing an *approximate* force response in the full configuration space. Because no degrees of freedom are removed in the simulation, the model can freely move, while the burden shifts to computing a reasonably accurate approximation. However, this is simpler since the contact force is a summation of local contact responses, and it is easier to determine at the force level which pairwise contacts need better approximation instead of figuring out which degrees of freedom are currently unimportant (and, more critically, when they are needed again). Moreover, errors in the force approximation can be quite large before they result in differing motion, and even larger before they result in qualitatively different material behavior, allowing for extremely cheap but not necessarily very accurate force approximation.

While there is obviously a wide set of possibilities for force approximation, there are additional features of yarn-based cloth to consider in the choice. Locally, internal contacts are coherent: they tend to persist throughout the simulation, and the local yarn shape changes slowly (in a local frame of reference), with individual contacts often exhibiting near-rigid motion. Consequently, the contact force, although stiff, is temporally coherent, suggesting that the approximation should take advantage of contact information over the course of many simulation steps.

This chapter discusses a corotational force approximation to the yarn's penalty-based contact force. The exact contact response is computed for a particular configuration, and then a rotated linear force model is used to approximate the force under small deformations. Once the shape changes too much, a new model is built centered at the new configuration. Temporal adaptivity is controlled by a single quality parameter that determines how frequently the contact models are rebuilt. Section 6.1 shows how the contact evaluation is split into a set of individual contact sets, each of which is individually approximated to compute the force response by the linearized contact force in Section 6.2. New contacts are discovered by the space-time scheduler of Section 6.3, which efficiently finds new contacting regions while potentially managing tens of millions of spline segment pairs. Results are shown in Section 6.4, and Section 6.5 ends with some conclusions to be drawn about this approach.

6.1 Contact Sets

Although the space for potential contacts is huge in a yarn-based cloth model, at any given point in time only a very small fraction of it is actually in contact.

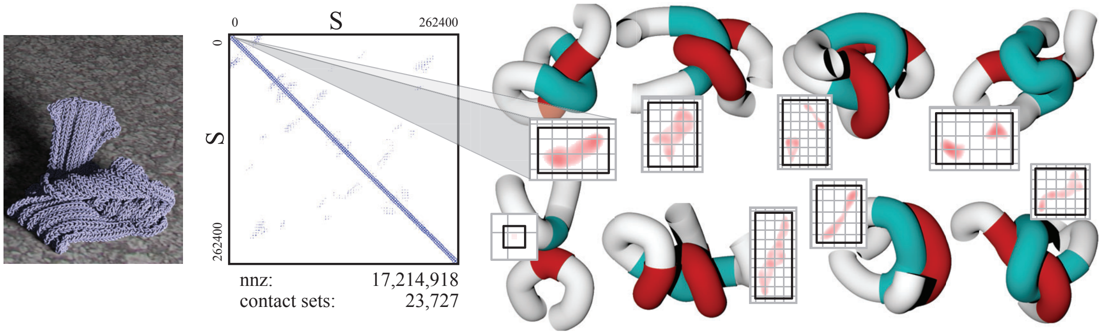


Figure 6.1: Contact structure in single frame of falling scarf

Figure 6.1 shows a single frame of a scarf falling on a ground plane, and the corresponding contact matrix, which is the matrix showing which parts of the yarn are in contact. The contacts clustered around the diagonal correspond to the persistent contacts due to the knit structure, while the scattered clumps further off the diagonal result from the folding of the scarf as it crumples. Due to the looping nature of knits, however, these contacts can take on a variety of unique shapes, shown on the right, with the actual pairs of points in contact in the contact matrix shown in red and boundaries between spline segments marked in gray.

Let S be the set of quadrature points used to compute the collision integral in Equation 4.7; if there are n segments with b quadrature points per segment, then $S = [0, bn]$. Individual contacts are then defined by partitioning $S \times S$ into disjoint sets $E, C_0, C_1, \dots, C_m$, where E is empty space, and C_k is contact set k, with the invariant that every pair of points (i, j) in contact must be in a contact set: if $i, j \in S$ and $f(\mathbf{y}_i, \mathbf{y}_j) \neq 0$, then there is some k such that $(i, j) \in C_k$. Because the contact matrix is symmetric, only the upper triangular part of $S \times S$ needs to be partitioned. Contact sets are allowed to be time-varying, but for the moment simply assume that the sets are given; their construction and maintenance are addressed in Section 6.3. Contact sets can theoretically be of any shape, but in

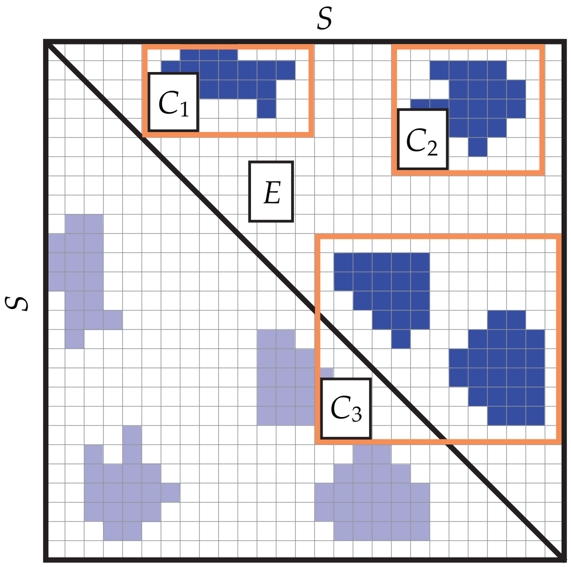


Figure 6.2: Examples of contact sets

practice in this simulator they will be padded bounding boxes around small contiguous contacts in $S \times S$; this is due to performance reasons discussed in Section 6.3.1. In the example contacts shown in Figure 6.1, the contact sets for each contact are shown as a black box, while Figure 6.2 shows several contact sets in context inside a contact matrix.

Given the partition into contact sets, the collision energy can be reformulated as a sum over contact features,

$$E(\mathbf{q}) = \sum_{k=0}^{m} E_k(\mathbf{q}) = \sum_{k=0}^{m} \left[\sum_{(i,j)\in C_k} w_{ij} f(\mathbf{y}_i, \mathbf{y}_j) \right] \tag{6.1}$$

where w_{ij} consists of all of the associated quadrature weights, stiffness constants, and scaling by segment lengths. It follows that the total contact force is a sum over all contacts,

$$\mathbf{f}(\mathbf{q}) = -\nabla_{\mathbf{q}} E(\mathbf{q}) = \sum_{k=0}^{m} \mathbf{f}_k(\mathbf{q}), \tag{6.2}$$

with each contact's force a sum over contacting quadrature points,

$$\mathbf{f}_k(\mathbf{q}) = -\nabla_{\mathbf{q}} E_k(\mathbf{q}) = \sum_{(i,j)\in C_k} w_{ij} \nabla_{\mathbf{q}} f(\mathbf{y}_i, \mathbf{y}_j). \tag{6.3}$$

6.2 Linearized Contact Approximation

Ultimately, it is this sum over quadrature points which is the expensive operation. Looking back at Figure 6.1, individual contact sets can consist of hundreds of sphere pairs which need to be processed and accumulated to generate the full contact response. Instead of looping over all these sphere pairs, though, a significant speedup could be achieved if there were a sufficiently reasonable approximation that could be computed via a loop over segments (the gray lines) instead of the sphere pairs (red boxes), avoiding the full evaluation of the penalty-based contact force computation (6.3) at each timestep.

Taking advantage of this observation, each contact set maintains its own local force approximation using a simple linearized model. These simplified models are used as a cheap approximation of the true contact force for a set of configurations near some reference configuration. When the current contact configuration strays too far from the reference, the contact model is discarded and a new one is constructed on the fly from the current configuration, which becomes the new reference configuration.

For any given contact the linearization of the force around a reference configuration $\bar{\mathbf{q}}_k = \mathbf{q}(t_k)$ is

$$\tilde{\mathbf{f}}_k(\mathbf{q}) = \mathbf{f}_k(\bar{\mathbf{q}}_k) + \bar{\mathbf{K}}_k\,(\mathbf{q} - \bar{\mathbf{q}}_k) \equiv \bar{\mathbf{f}}_k + \bar{\mathbf{K}}_k\mathbf{q}, \tag{6.4}$$

where $\bar{\mathbf{K}}_k$ is the stiffness matrix of $\mathbf{f}_k$ at $\bar{\mathbf{q}}_k$, and the force offset is $\bar{\mathbf{f}}_k = \mathbf{f}_k(\bar{\mathbf{q}}_k) - \bar{\mathbf{K}}_k\bar{\mathbf{q}}_k$. If contact C_k involves only c contacting quadrature points, then $\bar{\mathbf{K}}_k$ has at most $O(c^2)$ nonzero entries. $\bar{\mathbf{K}}$ is also obviously symmetric, and each 3×3 block corresponding to a control point is also symmetric. Thus, $\bar{\mathbf{K}}$ can be efficiently stored in a dense triangular matrix format with 6 entries per 3×3 block, along with $\bar{\mathbf{f}}_k$. On subsequent timesteps $t > t_k$, the linearized approximation of the force (6.4) can be used to quickly approximate the true value of $\mathbf{f}_k(\mathbf{q}(t))$. Note that evaluating this approximation involves a loop over the control points which are in the support of the quadrature points involved in the contact set, and not over the quadrature points themselves; in the context of Figure 6.1, evaluating the approximation is a loop over the intersection of the grey lines instead of the red squares. Computing the initial linearization, however, still requires a loop over the quadrature points.

6.2.1 Corotational Model

This linear approximation works well for configurations near the reference configuration, but quickly diminishes in quality as the contact changes shape. However, many of these shape changes are actually large near-rigid deformations, and because the contact force is a sum of pairwise interactions it transforms rigidly in the expected manner. As a result, the linear approximation can be improved by using a corotational force approach analogous to Müller et al. [74]. For each contact C_k, the nearest rigid transformation of the reference control points $\bar{\mathbf{q}}_k$ associated with C_k to the deformed configuration $\mathbf{q}$ is estimated by first matching the contact's center of mass and then finding its rotation R_k using the polar decomposition [77, 95]. The linear force model $\tilde{\mathbf{f}}_k(\mathbf{q})$ in (6.4) is then replaced by its corotational generalization,

$$\mathbf{f}_k(\mathbf{q}) \approx \mathbf{R}_k \tilde{\mathbf{f}}_k(\mathbf{R}_k^T \mathbf{q}) = \mathbf{R}_k \bar{\mathbf{f}}_k + \mathbf{R}_k \bar{\mathbf{K}}_k \mathbf{R}_k^T \mathbf{q}, \tag{6.5}$$

where $\mathbf{R}_k \in I\!R^{(3n+3)\times(3n+3)}$ is the matrix with the 3×3 matrix R_k repeated on the diagonal, and $\mathbf{q}$-translations omitted since $\bar{\mathbf{K}}_k$ annihilates them. Due to sparsity, only C_k-related control points and forces are evaluated in practice. Because R_k is updated on each timestep, the polar decomposition can be further accelerated via warm-starting the Jacobi iteration with the eigenvalues/vectors from the previous timestep [95]. By tracking near-rigid motion, the corotational contact force model (6.5) can be used longer than (6.4) before recomputation is necessary.

6.2.2 Model Invalidation

Even with a corotational force model, at some point after sufficient non-rigid deformation the linearized approximation (6.5) will be insufficiently accurate, and so the the contact model must be invalidated and rebuilt. If the current configuration, rigidly transformed back to the reference frame and denoted $\tilde{\mathbf{q}}_k$, has strayed too far from the reference configuration $\bar{\mathbf{q}}_k$, the linearized model is discarded, the current configuration $\mathbf{q}$ becomes the new reference configuration, and a new linear model is built. The shape estimate, $\texttt{metric}(C_k)$, used to indicate contact invalidation is

$$\texttt{metric}(C_k) = \max_i M_{ik} 2r \|(\tilde{\mathbf{q}}_k)_i - (\bar{\mathbf{q}}_k)_i\|_2 / \epsilon_k^2 \tag{6.6}$$

where if $C_k = [u_{\text{min}}, u_{\text{max}}] \times [v_{\text{min}}, v_{\text{max}}]$, then M_{ik} is the weight for control point i in contact k, taken to be

$$M_{ik} = \int_{u_{\text{min}}}^{u_{\text{max}}} |b_i(s)| ds + \int_{v_{\text{min}}}^{v_{\text{max}}} |b_i(s)| ds,$$

and ϵ_k is the minimum distance between pairs of interacting quadrature points in C_k computed when the model was last rebuilt. Again, due to sparsity in the contact sets, this metric only needs to be evaluated for control points related to C_k. When this metric is larger than some user-supplied tolerance τ, the model is rebuilt; in practice, $\tau = 0.004$–0.3, but sufficient speed and quality can be achieved with $\tau = 0.04$ and in general this parameter does not require much, if any, tuning. This metric is cheap to evaluate, allows greater movement for control points that weakly (or don't) influence contact C_k, and causes more frequent invalidations for close-proximity contacts.

6.2.3 Approximation Errors

Obviously, with a highly nonlinear force term and a linearized approximation, errors will be introduced in the force computation. One important feature of the metric (6.6) is that the introduced error will be zero when the tolerance τ is set to zero; on every timestep, all models will be rebuilt around the current configuration, which produces the correct force. As the value of τ is increased, however, the linearized model will be used more often before invalidation. If the value of τ is sufficiently large and the current configuration is sufficiently far from the reference configuration, the force generated by the linearization may actually reverse direction; the approximate contact force operates as an attractive force rather than a repulsive force. Figure 6.3 shows a simplified example of this occurring in a one-dimensional contact response.

While on the face of it this appears to be a significant drawback, there are several issues to consider. The first is that the approximate force will always be repulsive at the reference configuration, since it is always correct, and so configurations where the force is attractive will always be further apart than the geometry at the reference configuration; the linearization will not result in contacts wanting to be closer than the reference configuration. Secondly, this is an issue of parameter tuning, and in practice there seems to be a reasonably large space of parameter values for τ where this is not an issue. Finally, since the force becomes attractive, continuing to separate the contact requires additional energy influx at the contact in order for it to overcome the contact force. At some point, the metric will be invalidated and the contact will be rebuilt, and

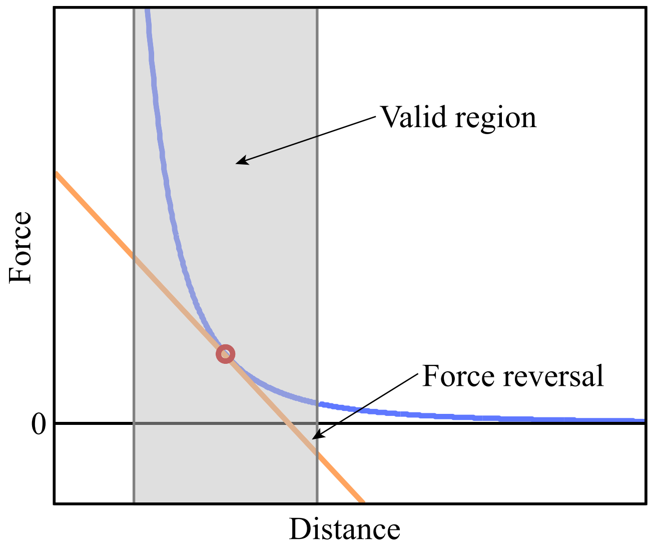


Figure 6.3: Force response curve and linearization in one dimension, with a valid region that includes force reversal

the force will once again become repulsive, but the energy needed to overcome the previously attractive force is lost. Thus, overly aggressive linearization acts as a form of damping in the system, and setting the parameter τ too large results in excessively damped cloth. This behavior can be observed in the validation study performed in Section 6.4.

6.3 Contact Adaptation

The contact algorithm is broken up into several phases, which can be broadly categorized into "contact detection" and "contact evaluation." The algorithm is summarized in Figure 6.4.

```
$\mathbf{f}^{(t+1)}$ = evaluate_other_forces()
for each segment $i$
    rasterize_to_grid($i$)
for each new segment/cell pair $i, j$
    create_schedule($i$, grid($j$))
for each schedule entry $sched$
    process_schedule($sched$)
coalesce_contacts()
for each contact $C_k$
    if (metric($C_k$) $> \tau$) rebuild_model($C_k$)
    $\mathbf{f}^{(t+1)} = \mathbf{f}^{(t+1)}$ + compute_force($C_k$)
$\dot{\mathbf{q}}_{\text{uncons}} = \dot{\mathbf{q}}^{(t)} + h\mathbf{M}^{-1}\mathbf{f}^{(t+1)}$
$\dot{\mathbf{q}}_{\text{glue}} =$ satisfy_glue_constr($\mathbf{q}^{(t)} + h\dot{\mathbf{q}}_{\text{uncons}}$)
$\dot{\mathbf{q}}_{\text{length}} =$ satisfy_length_constr($\mathbf{q}^{(t)} + h\dot{\mathbf{q}}_{\text{glue}}$)
$\dot{\mathbf{q}}_{\text{damp}} =$ nonrigid_damping($\dot{\mathbf{q}}_{\text{length}}$)
$\dot{\mathbf{q}}^{(t+1)} =$ object_contact($\mathbf{q}^{(t)} + h\dot{\mathbf{q}}_{\text{damp}}$)
$\mathbf{q}^{(t+1)} = \mathbf{q}^{(t)} + h\dot{\mathbf{q}}^{(t+1)}$
quasistatic_frames()
```

Figure 6.4: Algorithm Overview

6.3.1 Representation

From the description in Section 6.2, there is an obvious performance tradeoff in the construction of the contact sets C_k. Choosing to make smaller sets means that they are cheaper to update but produces more sets to process and increases the cost of set maintenance, for instance to determine when sets are overlapping.

To strike a balance between these competing concerns, contact sets are represented by bounding boxes in $S \times S$, and contain an estimate of a single contiguous contact between two parts of the yarn along with a specified amount of padding α to aid in detecting contact sliding (see Figure 6.6). Overlap tests are thus trivial to implement, and it provides a simple rule for when and how to merge contact sets. At the same time, in testing it seems to reasonably balance the number and size of contact sets while only including a relatively small number of non-contacting points.

6.3.2 Detection

At each timestep, contact detection explores the empty contact space, E, to find newly colliding points and either (a) incorporates them into an existing contact if they overlap, or (b) creates a new contact. Given the sheer size of E, efficiency is of paramount importance. Moreover, the structure of the cloth leads to a large number of non-convex objects in close proximity, which often presents a problem for many collision detection algorithms. Thus, in order to take advantage of the temporal coherence of the cloth, the simulator uses a combination a broad-phase spatial hashing of spline segments to find potential new contacts and a space-time collision scheduler that can efficiently track millions of close-proximity spline segment-segment pairs.

Due to the sheer number of quadrature points and potential pairs, the scheduler tracks potential collisions between pairs of spline segments, each of which contains b quadrature points. However, contact sets are defined at the quadrature point level, which means that between any two segments some pairs of quadrature points may already belong to a contact set while others do not. Thus, in order to completely explore E while not spending time rediscovering already-known contacts, each schedule entry also stores a bitmap which allows finer control over which pairs of quadrature points are already part of some existing contact and which still need to be checked. For memory reasons, this bitmap is implemented as a single 64-bit integer, allowing for a maximum sub-segment resolution of 8×8. Finally, each schedule entry i stores a conservative minimum distance bound d_i^t which represents a best (but conservative) estimate for the closest possible distance between the two segments at time t. The estimator also makes use of the maximum movement and maximum change in movement of each segment per timestep, denoted as $\Delta\mathbf{y}^{\min}, \Delta\mathbf{y}^{\max}$

and $\Delta^2\mathbf{y}^{\min}, \Delta^2\mathbf{y}^{\max}$, which are defined respectively in units of distance per timestep and distance per timestep squared.

Contact detection is divided into four phases: grid rasterization, newcomer scheduling, schedule processing, and contact coalescing

Grid Rasterization

The first step rasterizes each spline segment into a hashed uniform grid centered on the cloth's center of mass, with cell width γ. The AABB of each spline segment is computed in parallel by solving three quadratic equations to obtain its maximum and minimum extent in each of the three dimensions. Grid cells overlapping the AABB are marked as occupied by the segment. A segment not already belonging to a grid cell is flagged as a *newcomer* to the cell. At this time, the minimum and maximum movement and change in movement of the spline segment over the previous step are also computed (by solving six more quadratics) for later use by the scheduler.

Note that if two segments occupy disjoint sets of grid cells, this serves as a certificate that the segments cannot be in contact. Since this is performed for all segments on every timestep, this serves to filter out any possible contacts between segments for which this is true. Figure 6.5 shows a histogram of maximum AABB edge length for segments, normalized to the grid size of 0.6cm, for a single timestep of the sweater; nearly all segment bounding boxes are smaller than a single grid cell.

Newcomer Scheduling

After grid rasterization, each cell newcomer is checked in parallel against the cell's other segments to see if there is already a collision schedule entry for that

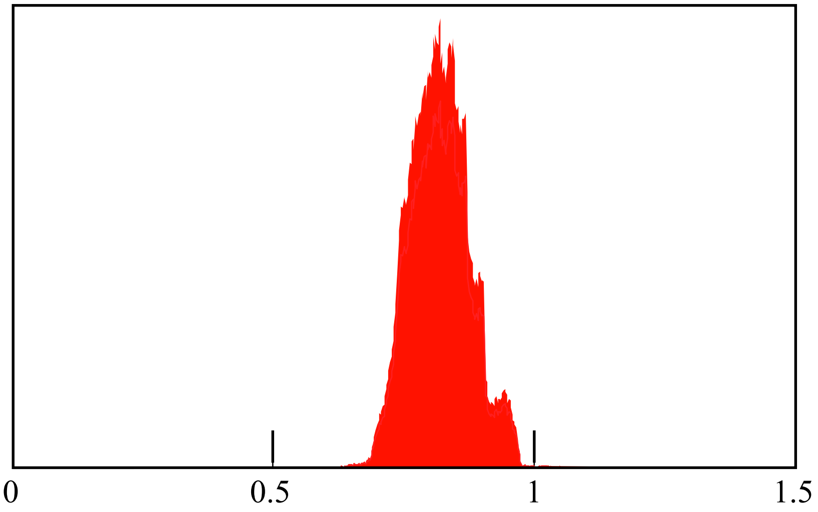


Figure 6.5: Histogram of segment size, normalized to grid size

newcomer-segment pair; if not, one is created and marked for immediate processing (by marking its minimum distance bound as negative).

Space-Time Schedule Processing

Once all possible schedule entries are created, the schedules are looped over in parallel and executed. When an entry is determined to need full processing, the scheduler examines all active pairs of quadrature points on the two segments, computes the true minimum distance, and sets d_i^t to the true minimum distance between the segments.

For schedule entry i, the minimum distance estimate d_i^t between the pair of segments is used to determine when full processing is needed. As long as the minimum distance is positive, then they are known to not be in contact. The scheduler updates this minimum distance estimate and only fully processes the entry again when the estimate becomes negative. This minimum distance estimate is updated using the minimum and maximum movement computed during grid rasterization to determine the maximum relative movement between the two segments over the previous timestep

$$\Delta\mathbf{d}_i = \begin{bmatrix} \max(& |\Delta\mathbf{y}^{\text{max}}_{\text{seg1}}.x - \Delta\mathbf{y}^{\text{max}}_{\text{seg2}}.x|, |\Delta\mathbf{y}^{\text{min}}_{\text{seg1}}.x - \Delta\mathbf{y}^{\text{max}}_{\text{seg2}}.x|, \\ & |\Delta\mathbf{y}^{\text{max}}_{\text{seg1}}.x - \Delta\mathbf{y}^{\text{min}}_{\text{seg2}}.x|, |\Delta\mathbf{y}^{\text{min}}_{\text{seg1}}.x - \Delta\mathbf{y}^{\text{min}}_{\text{seg2}}.x|), \\ \max(& |\Delta\mathbf{y}^{\text{max}}_{\text{seg1}}.y - \Delta\mathbf{y}^{\text{max}}_{\text{seg2}}.y|, |\Delta\mathbf{y}^{\text{min}}_{\text{seg1}}.y - \Delta\mathbf{y}^{\text{max}}_{\text{seg2}}.y|, \\ & |\Delta\mathbf{y}^{\text{max}}_{\text{seg1}}.y - \Delta\mathbf{y}^{\text{min}}_{\text{seg2}}.y|, |\Delta\mathbf{y}^{\text{min}}_{\text{seg1}}.y - \Delta\mathbf{y}^{\text{min}}_{\text{seg2}}.y|), \\ \max(& |\Delta\mathbf{y}^{\text{max}}_{\text{seg1}}.z - \Delta\mathbf{y}^{\text{max}}_{\text{seg2}}.z|, |\Delta\mathbf{y}^{\text{min}}_{\text{seg1}}.z - \Delta\mathbf{y}^{\text{max}}_{\text{seg2}}.z|, \\ & |\Delta\mathbf{y}^{\text{max}}_{\text{seg1}}.z - \Delta\mathbf{y}^{\text{min}}_{\text{seg2}}.z|, |\Delta\mathbf{y}^{\text{min}}_{\text{seg1}}.z - \Delta\mathbf{y}^{\text{min}}_{\text{seg2}}.z|) \end{bmatrix}. \tag{6.7}$$

One simple approach updates the minimum distance estimate each timestep according to the rule $d_i^t = d_i^{t-1} - \|\Delta\mathbf{d}_i\|_2$. This distance bound is guaranteed to always be less than or equal to the actual minimum distance between the two segments, and so entries are guaranteed to be processed on time. Experimentally, this also succeeds in filtering out the vast majority of possible all-pairs checks per timestep, but it does incur the cost of examining and updating d_i^t for each schedule entry every timestep to determine if it needs to be processed further.

This cost of simply examining each schedule entry can begin to dominate the overall cost of schedule execution, however, due to the vast number of schedule entries. To compensate for this, the schedule is further divided into a set of bins, where each bin $\lambda \in [0, \lambda_{\text{max}}]$ will now only be examined every 2^λ timesteps. Each segment now also stores the minimum and maximum movement $\Delta\mathbf{y}^{\text{min}(\lambda)}$ and $\Delta\mathbf{y}^{\text{max}(\lambda)}$ over the previous 2^λ timesteps. When bin λ is examined, each schedule i in the bin updates its minimum distance bound $d_i^t = d_i^{t-2^\lambda} - \|\Delta\mathbf{d}_i^\lambda\|_2$, i.e., using the maximum relative movement over the time period in which it was not being examined. After either examining or processing a schedule, it is then assigned to an appropriate bin based on the current minimum distance, maximum relative movement $\Delta\mathbf{d}_i^\lambda$, and maximum relative change in movement $\Delta^2\mathbf{d}_i$ by solving the quadratic equation:

$$-d_i + \|\Delta \mathbf{d}_i^\lambda\| \frac{t}{h} + \frac{\left(\frac{t}{h}\right)^2 + \frac{t}{h}}{2} \left(\|\Delta^2 \mathbf{d}_i\| + \omega\right) = 0, \tag{6.8}$$

to find the minimum nonnegative root, then assigning it to bin $\max(0, \lfloor \log_2 \frac{t}{h} \rfloor)$. Because the relative velocity could change arbitrarily between checks, as in Hubbard [53] Equation (6.8) allows for some bounded deviation in relative change in movement ω. When the relative change in movement between the two segments is larger than $\|\Delta^2 \mathbf{d}_i\| + \omega$, the predicted bin is no longer valid and d_i^t needs to be immediately updated and a new bin reselected. However, the original goal was to avoid examining every schedule pair every timestep. Thus, the scheduler instead monitors the total change in movement of each segment, and when it is more than $\frac{\omega}{2}$ from some reference change in movement, all schedules associated with that segment are immediately scheduled to have their minimum distance estimates updated, and the reference change in movement is updated to be the current change in movement. This conservatively reschedules contacts before their bin assignment becomes incorrect, while avoiding any loops over all schedules every timestep, at the cost of invalidating some schedule entries before their bin assignment is actually invalid.

Contact Coalescing

The output of the schedule processor is a list of quadrature points $(i, j) \in S \times S$ that are currently in contact. The coalescer then takes these pairs (i, j) and groups them together into new contiguous contacts by floodfilling in $S \times S$ the axis aligned bounding box $(i - \alpha, j - \alpha)$ through $(i + \alpha, j + \alpha)$. Any overlapping contact sets (either new or pre-existing contacts) are merged into new AABBs until no more merges are required. This results in contact sets with a buffer of

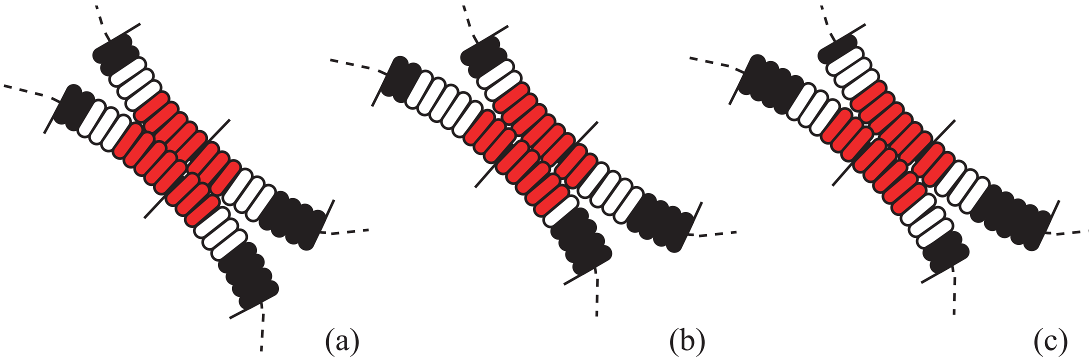


Figure 6.6: Contact Resizing

non-contacting pairs α around the detected contact, which allows the contact processor to detect when to resize a particular contact.

6.3.3 Evaluation and Evolution

After detecting new contacts, all contacts must be evaluated. This involves looping over the contact sets C_k in parallel, computing the best rigid transformation and evaluating the contact-invalidation metric. If the metric is not violated, the contact set evaluates the current approximate model (6.5) via matrix-vector multiplication and vector addition. Because of the semi-regular looping structure of cloth, contiguous contacting regions tend to be small and localized, leading to small contact sets (which are designed to cover a single contact each); in practice, contact matrices $\bar{\mathbf{K}}_k$ typically have between 24 and 63 rows/columns.

If the metric (6.6) is violated for contact k, the linearized contact model $(\bar{\mathbf{f}}_k, \bar{\mathbf{K}}_k)$ is rebuilt for $\bar{\mathbf{q}}_k = \mathbf{q}$, which involves looping over all $(i, j) \in C_k$ to accumulate first and second derivatives of $f(\mathbf{y}_i, \mathbf{y}_j)$. Once this is done, corotational forces are evaluated using (6.5). While looping over the set, the model rebuilder also computes the minimum and maximum points $i_{\text{min}}, i_{\text{max}}, j_{\text{min}}, j_{\text{max}}$ that are in

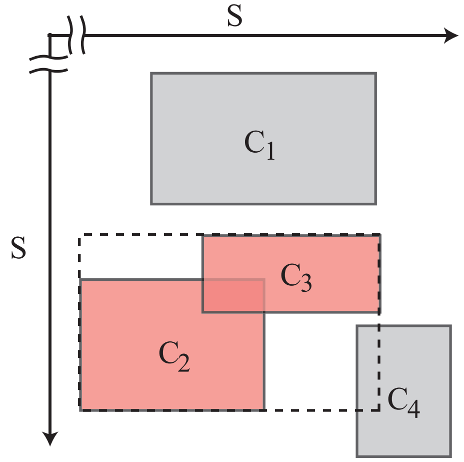


Figure 6.7: Contact Merging

contact, and afterwards updates C_k to cover $[i_{\text{min}}-\alpha, i_{\text{max}}+\alpha] \times [j_{\text{min}}-\alpha, j_{\text{max}}+\alpha]$. Figure 6.6 shows this process in action; the contact (a) slides between updates to (b), shifting the set of points in contact, but the buffer allows this to be detected during update and the contact resizes itself to the new bounds (c). If there are no contacting points in the set it is marked as contributing zero force, and once the minimum inter-point distance becomes larger than some threshold (in practice, $2.1r$) then C_k is deleted and the pairs of points in C_k become part of the empty region, E.

The contact set structures can also be used to efficiently model additional yarn behaviors at negligible cost. As a simple example, each contact set also maintains a small number (4) of stiction springs which model the interactions due to entangled fibers. These springs are inserted at fixed locations within the contact set, and are broken and rebuilt once their energy exceeds a specified value.

Following this contact evaluation, the simulator must determine whether the contacts are now overlapping in order to avoid double-counting of contact force contributions. To efficiently accomplish this, S is divided into a set of bins

parameter	description	value
τ	approximation tolerance	0.004 – 0.3
α	flood-fill size	3
γ	grid size	0.6 cm
λ_{max}	num. schedule bins	8
ω	movement change bound	0.0006 cm/timestep^2
h	timestep	1/16200s – 1/24000s
b	quadrature points / seg.	11
k_{contact}	contact stiffness	3000 – 4500

Table 6.1: Parameters used during ACL simulation.

(with each bin of size 8) and each C_k is placed into each bin for which its first dimension overlaps. The bins are then looped over, scanning for overlaps only among sets in that bin, and merging any that are found. This process is illustrated in Figure 6.7. Note that C_2 and C_3 are overlapping, which causes them to be merged together, which will then cause a cascading merge with the resultant merged contact and C_4. In principle, contacts should also be checked to determine if they have in fact split–that is, one contact set is now representing multiple contiguous contacts–since the cached dense stiffness matrix might become large with many zero entries. Because cloth has such regular structure, however, in practice it does not appear to be necessary to split contact sets into subpieces since contact splitting is a rare occurrence, and when it does happen the contacts tend to stay close to each other (see Figure 6.1 for examples of contact splits), and so the matrices stay reasonable in size. Note however, that this appears to be a result of a cloth-specific property; if this method were adapted for e.g. hair contact, it is much more likely that explicit contact splitting would be necessary.

model	scarf	scarf	scarf	scarf	sack	afghan	sweater
loops	3,240	-	-	-	41,272	54,340	45,960
segments	26,240	-	-	-	334,464	438,485	370,650
tolerance	0.004	0.04	0.1	0.3	0.04	0.04	0.04
contacts	23,186	23,475	24,296	29,037	262,062	365,305	295,702
updates	3.7%	0.46%	0.19%	0.04%	0.30%	0.47%	0.21%
contact eval							
old	254ms	-	-	-	3,063ms	3,995ms	3,665ms
new	41.7ms	30.9ms	30.5ms	32ms	366ms	600ms	405ms
speedup	6.1x	8.2x	8.3x	7.9x	8.4x	6.7x	9.1x
overall (per 1/30s frame)							
old	4m10s	-	-	-	33m55s	44m8s	40m16s
new	58.4s	50.2s	50.6s	50.8s	7m27s	10m34s	8m6s
speedup	4.3x	5.0x	4.9x	4.9x	4.6x	4.2x	5.0x

Table 6.2: Model and scene statistics and timings (All numbers and timings are an average number over 2s of simulation)

6.4 Results

The adaptive contact linearization method was implemented on top of the simulator discussed in Chapter 4, including the optimizations discussed in Section 4.5, and is written in Java and multithreaded. Results were generated on a Mac Pro machine with two 4-core 2.93GHz Intel Xeon processors with 16GB of RAM. The adaptive contact linearization force is compared against the parallelized bounding hierarchy descent method discussed in Section 4.3.2. In order to estimate scheduler performance, the adaptive contact linearization force is also compared to simulations where the tolerance was zero (so models are rebuilt on every timestep) and stiffness matrix computation was disabled, providing a good approximation of the best possible speed obtainable when solving the contact forces exactly on each timestep. Timings and performance breakdowns are in Table 6.2 and Figure 6.10. Note that the average cost of contact coalescing was negligible ($<$ 1ms) in all simulations and is thus omitted from timings.

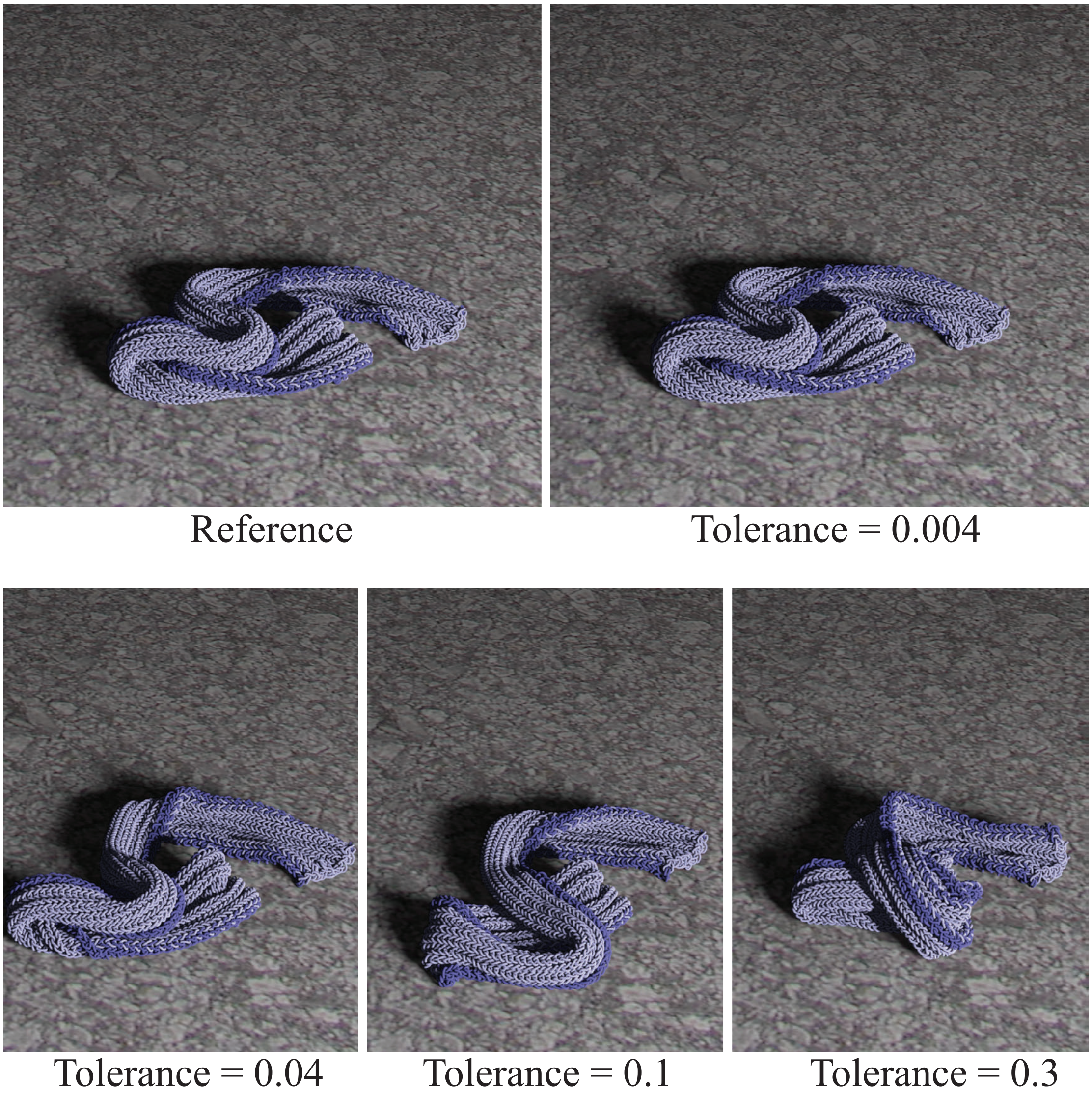


Figure 6.8: A scarf falling on a flat plane for a variety of tolerances.

Figure 6.8 shows the results of a validation comparison to measure the qualitative accuracy of adaptive contact linearization with various tolerances, as well as the reference implementation. In this experimental setup, a scarf falls on a ground plane. Because for this example trivial deviations in force can result in drastically different behavior, variations in the final configuration are expected. For small tolerances ($\tau = 0.004$), the simulations are qualitatively identical, with indistinguishable behavior for the entire simulation. At higher

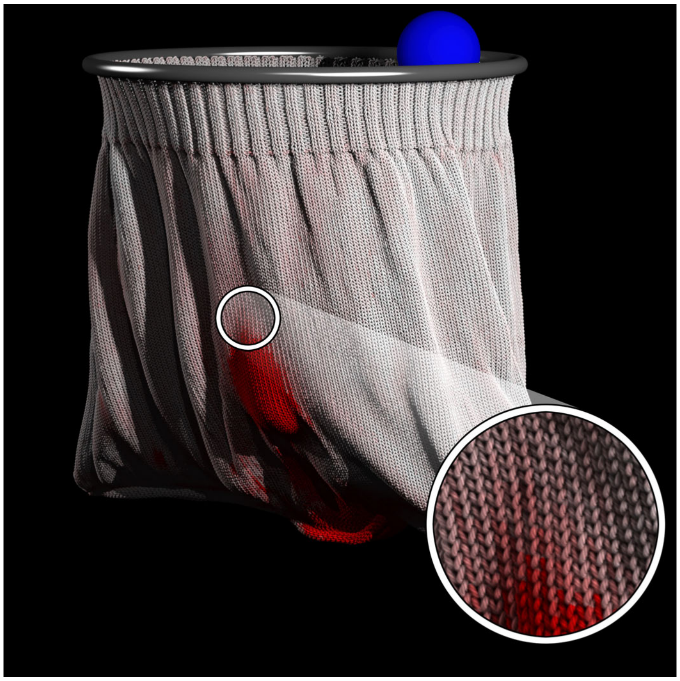


Figure 6.9: Sack, with model updates visualized in red. Pure red is ≥ 13 updates over the 1/30s frame (540 timesteps)

tolerances ($\tau = 0.04 - -0.1$), the small deviations introduced by the approximate force result in the cloth ending up in a different final state, although it still has the same qualitative material properties as it moves. Finally, at the highest tested tolerance ($\tau = 0.3$), the approximation becomes evident resulting in different material behavior, although it still is plausible; the material looks overly damped, a result of the issues discussed in Section 6.2.3. Note also that the overall speedup levels off, indicating that the vast majority of the time is spent on scheduling and force evaluation, and not model regeneration.

However, the main benefit of ACL forces is the simulation of significantly larger yarn models, consisting of 41,000 to 54,000 knit loops (compare to the 3,240 loops in the scarf). As the sack (Figure 6.9) is filled with 90 rigid balls, model updates (exaggerated in red, with pure red corresponding to ≥ 13 updates per frame) are localized primarily around where the spheres contact the cloth resulting in contact computation speedups from 7.5x to 9.4x (averaged over 5 frame intervals), and 4.3x to 4.8x overall. A blanket (Figure 6.11) falls onto a sphere, causing high speed self-contacts across large portions of the cloth and large scale global deformations; this is the most challenging scenario for both the model and scheduler due to the high speed contacts and rapid deformations, but it still results in speedups from 4.5x to 8.4x in contact computation and 3.3x to 4.7x overall. Finally, a wooden mannequin wearing a sweater walks forward (Figure 6.12), showing efficient simulation of character-sized garments in complex contact configurations; this achieves speedups from 7.5x to 10.5x in contact computation and 4.5x to 5.3x overall. Note also that because the yarns were simulated directly, the model automatically captures the curling behavior of stockinette, visible at the ends of the sleeves and body. The size of these models challenges the scalability of exploring empty space, but for instance on the sweater the scheduler limits the number of overall schedule entries being tracked to an average of 14 million, with on average only 1.2 million examined and 77,000 processed per timestep.

In all scenes the percentage of model updates per timestep is extremely low, on the order of 0.5% per timestep, corresponding to more than 200 timesteps between invalidation for the mean contact set, or around three times per 1/30s frame. Thus, even though the simulator takes small timesteps, the temporal coherence is such that contact linearization and invalidation would still be effective even with a timestep 10x larger. In addition, even with relatively conservative tolerances the cost of model updates in an amortized sense was negligible, as seen in the cost for the various scarves; if larger timesteps were used and model updates became a performance bottleneck, the tolerance could be increased while still producing reasonable results.

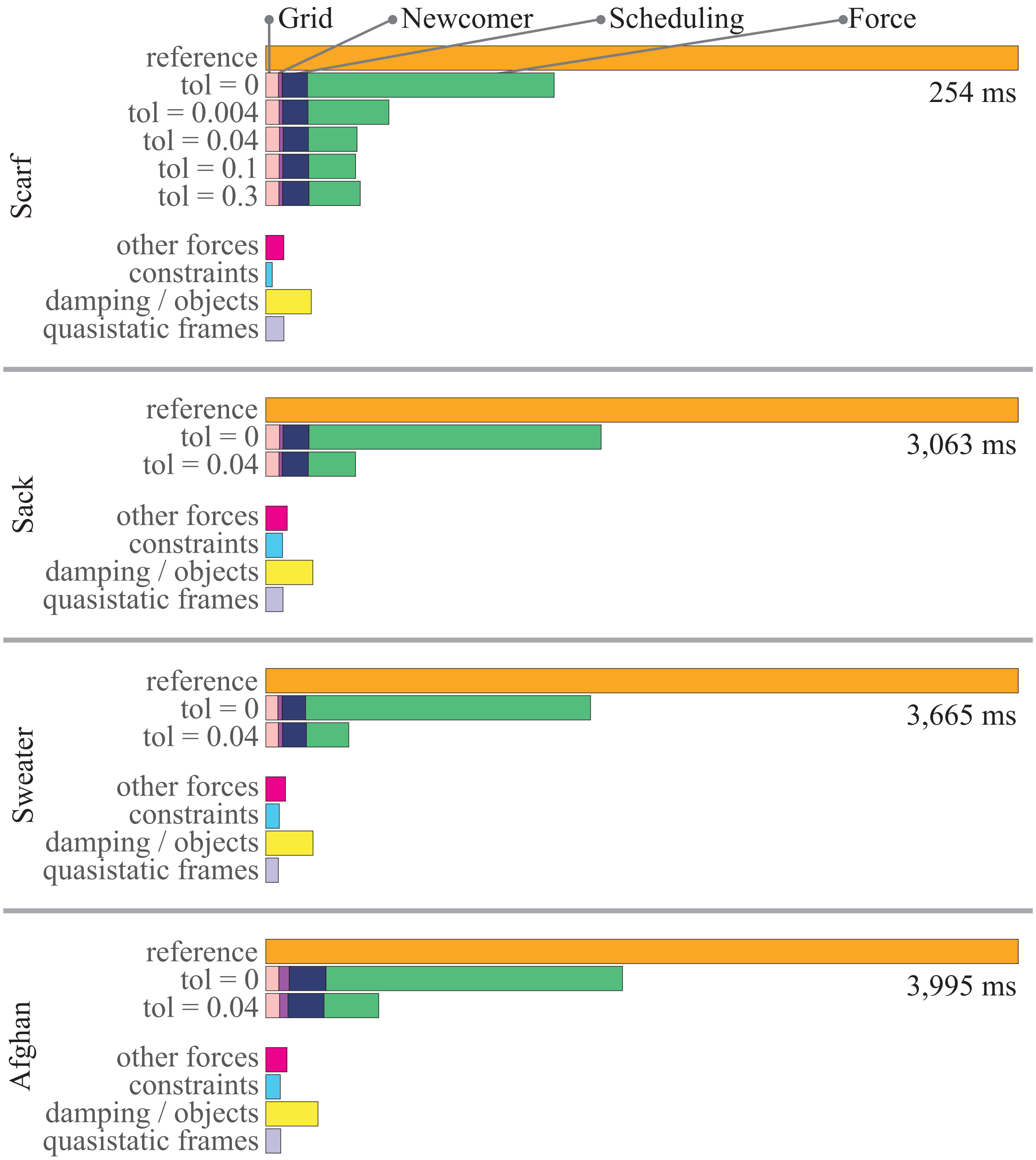


Figure 6.10: Performance comparison of adaptive contact linearization

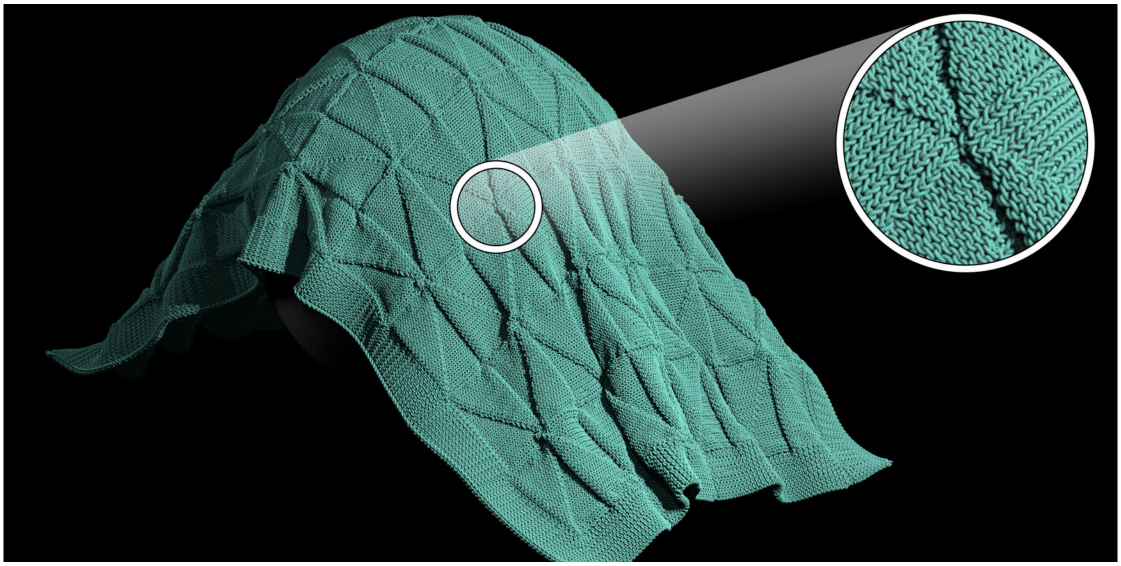


Figure 6.11: Blanket falling on sphere

6.5 Conclusion

This chapter has discussed a method for speeding up contact force evaluations for yarn-based cloth models by breaking up the contact problem into a set of disjoint regions and adaptively constructing local models for each region to approximate the true force response. This results in typical speedups of 7x-10x over naïve force evaluation, which brings the cost of force evaluation in line with other phases of the simulation, while still maintaining similar (and in many cases, visually identical) motion.

This contact-level approach may lead to many interesting future possibilities for both quality and performance improvements that are difficult to solve using naïve contact evaluation. In particular, further additions may help address two of the most pressing problems in yarn-based simulation: modeling hysteresis and taking larger timesteps. As discussed previously, hysteresis and damping is a critical component to get right in order to achieve accurate cloth simulation. Adding plasticity in at the contact level is more physically plausible (due to fiber

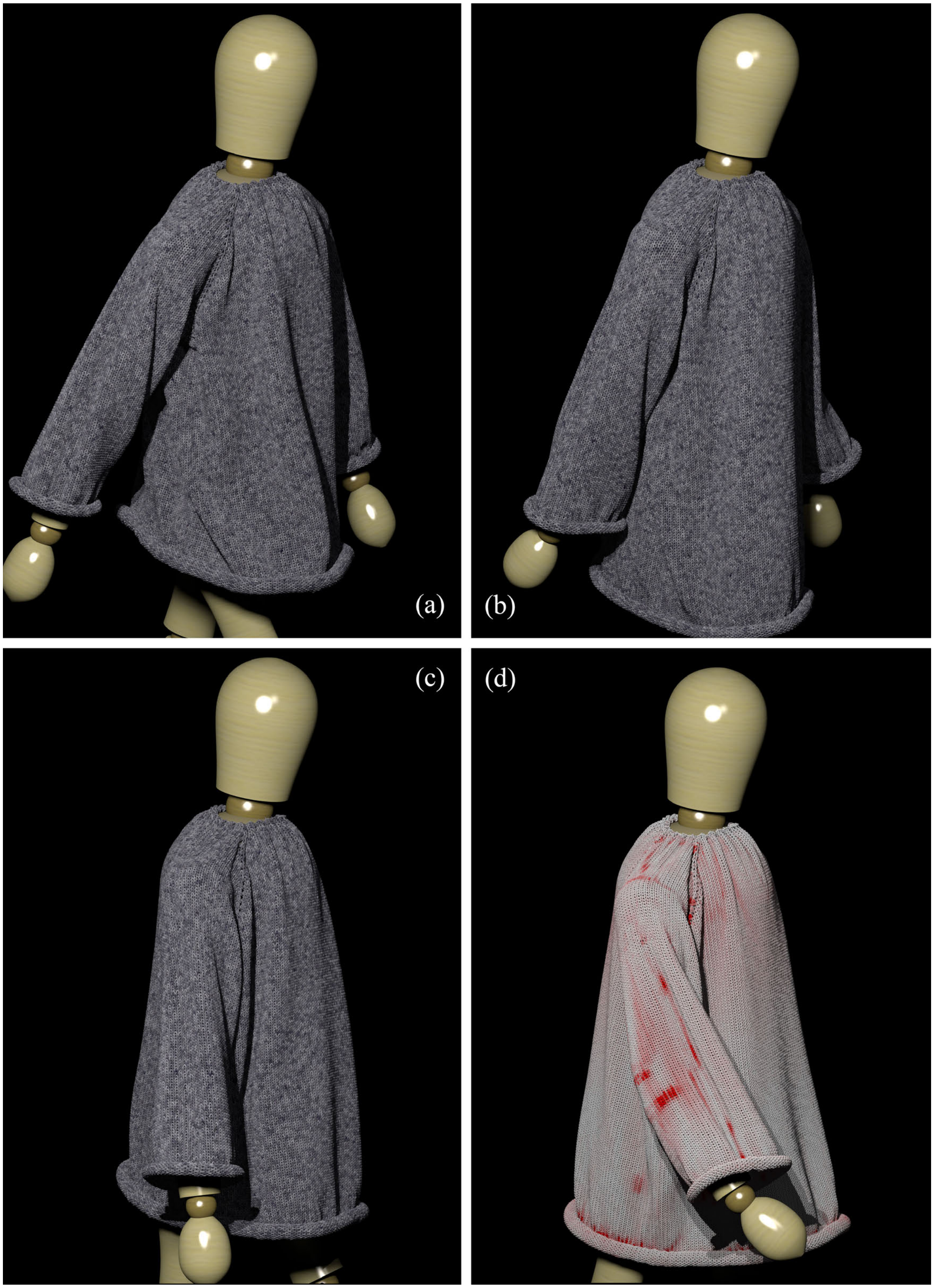


Figure 6.12: Four frames from the sweater animation

entanglement), and the contact set formulation provides a primitive on which to model these effects, as in the simple approach of adding stiction springs.

Current timestep restrictions are due to two factors: the stiffness in the collision response, and, more crucially, the inability to detect and respond when the yarns slip through each other. This is a catastrophic simulator failure, since it can lead to the entire structure unravelling, and is currently addressed through small timestepping to ensure sufficient time for contact force response. Moving contact evaluation to a higher level (contact sets instead of collision points) should allow the simulator to more easily detect when this occurs. Ultimately, the hope is that this will lead to the ability to adaptively timestep yarn-based models and automatically step down when the timestep becomes too aggressive and results in slip-through.

CHAPTER 7

CONSTRUCTING YARN GEOMETRY

All physical simulations which rely on the solution of initial value problems naturally require some way to specify or generate the initial conditions or configurations which will be simulated. For some simulations this can be easy or even trivial; consider for instance the case of elastic sheet simulation, where the initial geometry can simply be specified as a mesh created using standard modeling tools. Unfortunately, for yarn-based cloth simulation the initial configuration involves specifying the rich yarn geometry; these yarns must be properly interlooped or interlaced, since any errors in their construction propagate to the final result. While this geometry can also be constructed by hand using standard modeling tools, large models can involve tens of thousands of local yarn relationships and are thus are impractical to model without automated tools.

Woven fabrics, due to their simpler local geometry, are relatively straightforward to generate automatically, but the complexity of the loop interactions in knits make them particularly difficult to recreate. However, although in theory the geometry for each loop in a piece of knitted fabric might be different, in practice there is a significant amount of repetition in almost all fabrics. As a result, automated systems to construct geometry given initial specifications (e.g. knitting instructions) should in principle be possible. For instance, systems have been proposed for generating knit geometry by simulating the knitting process itself [30, 28]. A slight twist on the problem was discussed by Igarashi et al. [55], which takes as input mesh geometry and additional user input and generates a possible yarn geometry and knitting instructions corresponding to the mesh. Section 7.1 discusses the implementation of a basic automatic knit yarn generator which uses the simulator itself to enable the creation of models for the simulator; this is the tool that was used to generate all of the models in this thesis. Following that, Section 7.2 discusses some possibilities for more elaborate knit yarn generators in the future.

7.1 An Automatic Knit Yarn Generator

Although there is a wide variety of knit stitches available, in practice a typical piece of fabric only uses a small number of them. These stitches will each take on some shape in the final fabric, but that shape is determined by the types and shapes of other stitches in the local neighborhood around each stitch. In addition, because of the regular repeating nature of most cloth, there are likely to be a limited number of distinct neighborhoods as well. Thus, although there may be a large number of stitches overall, there are likely to be a small number of unique stitch shapes (expressed in some reference frame). A piece of knit fabric can then be generated using models of all the types of stitches that might be seen, with the generator given as input a list of stitches and outputting the correct geometry at each stitch location based on the stitch type as well as the local neighborhood.

This can still require a significant number of stitch models. For instance, a single knit stitch may require separate models if it is next to a purl stitch in each of the four fabric directions, on the edge of the fabric, or next to a bind-on / bind-off. However, the number of stitch models can be further limited via the observation that the generator does not need to generate final geometry for the model; rather, it simply needs to generate 'good enough' geometry with the proper interlooping of yarns. This geometry can then be refined in the simulator, further relaxing the geometry to a final rest state which corresponds to the actual knit model to be simulated. As a result, stitch models merely need to

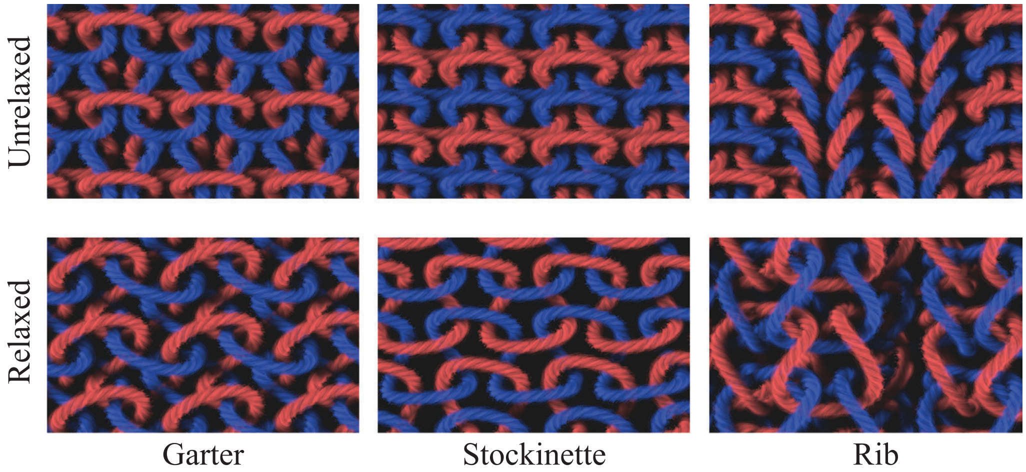


Figure 7.1: Unrelaxed and relaxed geometry for each of the three basic knit types (yarn radius shrunk for clarity).

guarantee proper interlooping, with other errors in shape corrected afterwards, and so many fewer potential models are needed. Figure 7.1 shows both the initial geometry and the geometry after further refinement in the simulator for the three basic knit types.

Based on this, a simple knit yarn generator for rectangular / cylindrical fabrics can be created using only 7 stitch models (see figure 7.2). The stitch models are designed to fit together seamlessly, with a single model of a general loop forming the bulk of the fabric and the other six models forming the boundary. This general loop model can be furthermore flipped along the z-axis to create either a knit or a purl stitch. Given a $m \times n$ bit array P, where $P_{i,j} = 1$ means that stitch (i, j) is purled, and the number of spline segments k to generate per stitch, the generator forms a single spline curve to describe the fabric by laying down one stitch at a time and finding the best least-squares spline approximation of that stitch using k segments and given the control points already added by the previous stitch. Since control points already added are fixed, this results in a small least-squares problem of size $O(k)$ to be solved for each stitch in the

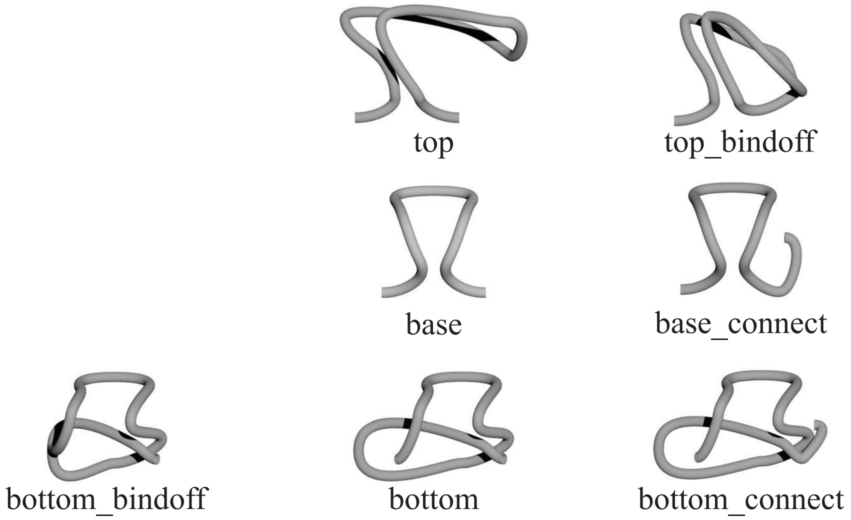


Figure 7.2: Stitch models for basic grid / cylindrical construction.

fabric. Cylindrical fabrics can be generated by simply laying down stitches in a helix, with the height of the helix equal to one row of stitches; the next row will then logically start immediately from the end of the previous row.

The goal of the model generation is to obtain a configuration where all of the loops are properly interconnected according to the specified pattern, but it is not necessarily the rest state. The rest state is determined by then simulating the pattern using the simulator with a few slight changes. The yarn is no longer treated as inextensible but is instead treated as being linearly elastic with high stiffness; this allows the yarn to stretch and compress as needed to fit the final model. In addition, high amounts of viscous damping are used, since although the model is in a feasible configuration it may be in a relatively high-energy state, and damping helps to limit energy growth and bleed energy out of the system. Finally, the yarn is shrunk by setting the desired arclength ℓ_i of each spline segment to $c\ell_i^0$, where ℓ_i^0 is the starting arclength and $c<1$ is a shrinking factor. This shrinking factor is obviously dependent on the model stitches used in the generator; particularly loose stitches will require a larger shrinking factor.

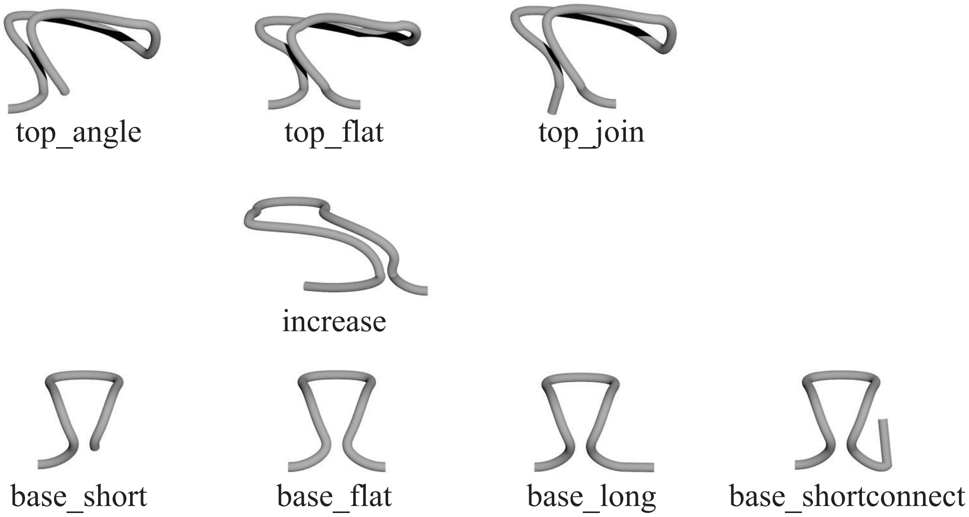


Figure 7.3: Additional stitch models for generating the sweater model.

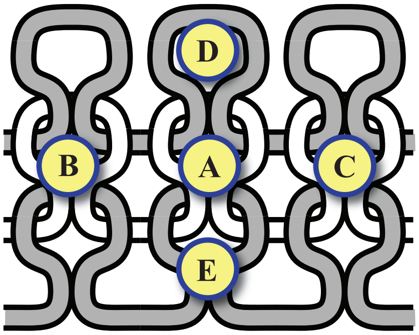
For the stitch models in figure 7.2, $c = 0.935$. This causes the entire cloth to compress and settle into a general rest state, which can then be used as a cloth sample in simulations.

This process works well for simple cloth models that can be expressed as combinations of cylindrical and rectangular patches, but moving to more complex topological knit structures significantly increases the complexity of the generator. For the sweater shown in Figure 6.12, for instance, each row of the upper part of the chest consists of a set of eight increases (two each on each side and front/back) until the sleeves are attached; afterwards, the yarn shifts to a cylindrical pattern to form the bottom half of the chest. The same generic process of laying down stitches in succession was used to form the initial geometry for the sweater, but placing stitches correctly required much more tweaking to ensure proper looping behavior (and in some cases, such as around the neck, it is likely that errors in the structure were introduced during this initial laying of stitches). Moreover, although there was only one new type of stitch introduced—an increase—eight additional stitch models were used in order to adequately reconstruct the geometry given the sharp turns in the construction. Figure 7.3 shows the additional eight stitch models, many of which simply represent a small variation on the original stitch necessary to achieve correct interlooping.

7.2 Future Directions

Obviously, while the generator described in Section 7.1 is capable of easily creating topologically simple models and, with modification, specific instances of more complicated geometry, it does not seem to fulfill the goal of a general, all-purpose cloth generator. It is worth taking a step back and examining what properties are desirable and what should be the final result. In order to be of practical use, cloth should be able to be generated by non-programmers, while the semi-scripted method of laying stitches in Section 7.1 requires programmatic effort to support each new type of clothing. As a result, it would be ideal if the generator could automatically determine the final shape, both local and global, of the piece of clothing without any additional input.

Out of several general ideas which were explored, one in particular seemed to present some promise, built on the idea of breaking the problem up into multiple pieces. The generator above moves from "instructions" to final geometry in essentially one step, when in fact there are multiple smaller steps that can be performed and verified in sequence. Ultimately, knit geometry is described by a set of loops, and relationships defined between those loops. One way of expressing these loop relationships is as a graph: given a loop L, there is a predecessor and successor loop in the yarn direction (the loop formed by the yarn immediately before and after L), as well as the loop L is pulled through, and some number of loops (possibly zero) which are themselves pulled through L



Loop A:

is the successor of Loop B
is the predecessor of Loop C
is knit through Loop E
has loop D knit through itself

Figure 7.4: A simple knit structure and associated loop relationships

(See Figure 7.4 for a diagram of one set of relationships for a loop in a standard knit). Then, one way of decomposing the problem of model generation is as follows:

1. Convert knitting instructions or model into graph of knit loops and relationships between loops
2. Given the graph of relationships, find positions and orientations for all loops which locate related loops in their proper relative positions, which reconstructs the global shape of the fabric
3. Generate geometry for each loop which precisely satisfies the relationships
4. Use the initial geometry and the simulator to relax the geometry to the final rest position

With clear separation between the steps, solutions to each phase can now be pursued individually.

7.2.1 Finding Loop Positions

Given that the loop relationships are expressed as a graph, it suggests applying graph algorithms to find stitch locations. To do so, the graph of relations is converted into local relative distances between yarn loops and then global positions are found for all loops which best satisfy the relative distances with respect to some metric. It is important to note that because the relationship graph is incomplete, the relative distances between yarn loops may also be incomplete; the idea is that each yarn loop should only specify distances to other yarn loops in some small local region around itself (based on the set of relationships defined by the graph), with global reconstruction following from these local distances. Two potential (and related) methods for achieving this local-to-global reconstruction were explored: spectral embedding and locally linear embedding.

Spectral Embedding

In spectral embedding [25, 100], given a nonnegative weighted graph in matrix form $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\mathbf{A}(i, j)$ being the weight between vertices i and j, each vertex i is assigned a k-dimensional position $\mathbf{x}_i$ which minimizes

$$\sum_{0 \leq i,j < n} \mathbf{A}(i, j) \|\mathbf{x}_i - \mathbf{x}_j\|_2^2, \tag{7.1}$$

that is, minimizes the energy function which penalizes the $2-$norm distance between points based on the weight between those points in the graph. This is done by computing the eigenvectors of the matrix $\mathbf{L} = \mathbf{D} - \mathbf{A}$ where $\mathbf{D}$ is the diagonal matrix consisting of the sums of each row/column of $\mathbf{A}$. For nonnegative weights $\mathbf{L}$ will be positive semi-definite, with one zero eigenvector (the vector of all ones). The eigenvectors $\mathbf{v}_2 \ldots \mathbf{v}_{k+1}$ corresponding to the k smallest non-zero eigenvalues are the minimizers of Equation 7.1; each vertex is assigned position $\mathbf{x}_i = [\mathbf{v}_2(i), \mathbf{v}_3(i), \ldots \mathbf{v}_{k+1}(i)]^T$.

Given a set of desired distances between some sets of yarn loops, these must be converted into weights on a graph such that greater distances between loops correspond to smaller weights: for instance, $\mathbf{A}(i,j) = 1/d(i,j)$ if the distance between loops i and j is specified, and $\mathbf{A}(i,j) = 0$ otherwise. Alternatively, the distances can be linearly mapped to the weights by setting $\mathbf{A}(i,j) = \alpha + \beta - d(i,j)$, where $\alpha = \max_{i,j} d(i,j)$ and $\beta = \min_{i,j} d(i,j)$; in practice there seems to be minimal differences between the two mappings in the results.

Locally Linear Embedding

In contrast to spectral embedding, locally linear embedding (LLE) directly seeks to recover low-dimensional data from some high-dimensional input [97]. For instance, measurements of data may be taken in a high dimensional space, but they are actually well-described by a lower-dimensional surface embedded into the high dimensional space. This embedding may be nonlinear, however, and so LLE attempts to find it through local linear approximations; if data points are dense enough, then the local linear approximation is expected to be a good one. Given the n data points $\bar{\mathbf{x}}_i$ in $p-$dimensional space, the algorithm computes reconstruction weights w_{ij} and positions $\mathbf{x}_i$ in $k-$dimensional space which minimize the two equations:

$$\sum_i |\bar{\mathbf{x}}_i - \sum_{(i,j) \text{ related}} w_{ij}\bar{\mathbf{x}}_j|^2 \tag{7.2}$$

$$\sum_i |\mathbf{x}_i - \sum_{(i,j) \text{ related}} w_{ij}\mathbf{x}_j|^2 \tag{7.3}$$

This is done in two stages—the first equation is minimized over the weights w_{ij}, and then the second equation is minimized over the positions $\mathbf{x}_i$ given the fixed weights. Much like spectral embedding, this second phase involves computing the eigenvectors corresponding to the smallest eigenvalues of a matrix whose sparsity depends on the number of neighbors of each data point.

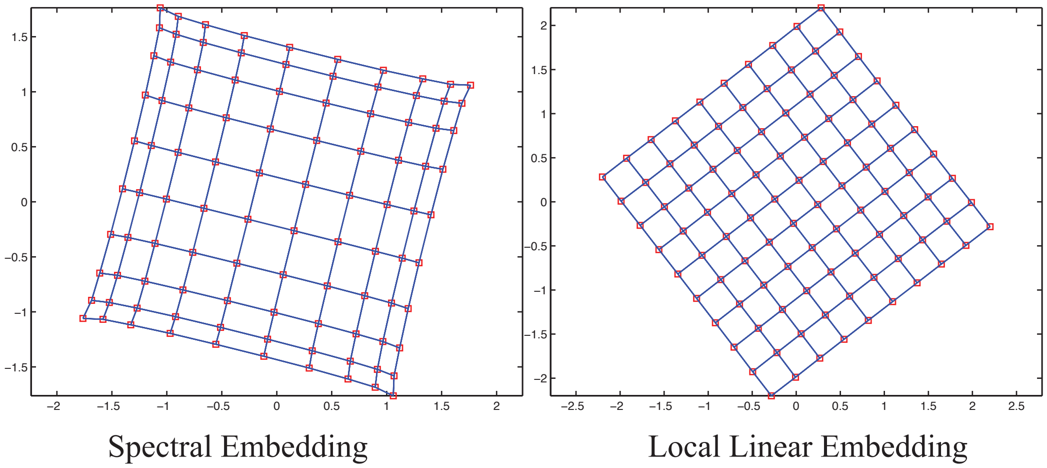
Because it assumes data points already exist in high dimensional space, LLE requires a bit of modification to fit the knit model generation problem. The most straightforward way of adapting it involves recognizing that the LLE algorithm does not care what the high-dimensional positions of each data point are, only the relative positions to all the data points in its local neighborhood. Thus, rather than converting the relationship graph to relative distances, it is converted to relative positions; for instance, loop L is assigned to $(0, 0, 0)$, and the successor loop to loop L might be assigned a relative position of $(1, 0, 0)$, while a loop pulled through L would be assigned relative position $(0, 1, 0)$. Given this, LLE can then be applied directly to produce global positions for all loops.

Yarn Loop Neighborhoods

In general, it is not entirely sufficient to specify distances to just the set of loops directly related to each loop, as this can result in the collapse of the geometry to a few points. Instead, relative distances (or positions) are computed at each yarn loop L for all loops related to L, and all loops related to loops related to L (the 2-neighborhood around L)

Results

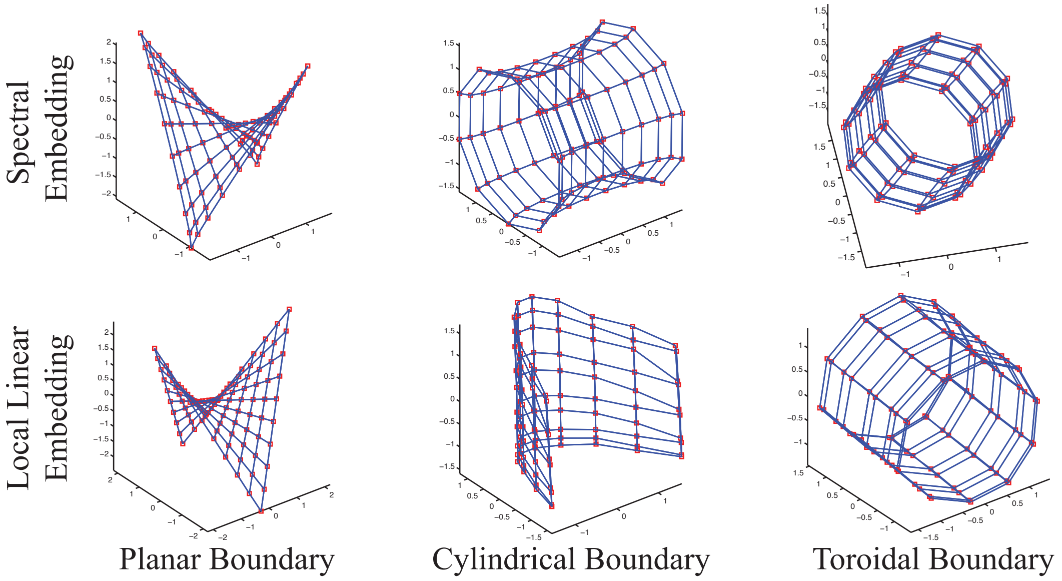
As a first test, both spectral embedding and locally linear embedding were compared when reconstructing a 10×10 grid of simple knit stitches, where each point represents a yarn loop. Figure 7.5 has the results of a square of fabric, embedded into $\mathbb{R}^2$. Both methods produce acceptable embeddings, although in spectral embedding the edges tend to be a little more compressed as compared to LLE, which produces a uniform plane. Because the relationships are entirely



Spectral Embedding

Local Linear Embedding

Figure 7.5: Comparison of 2D embeddings of 10×10 grid for both spectral embedding and Local Linear Embedding



Planar Boundary

Cylindrical Boundary

Toroidal Boundary

Figure 7.6: Comparison of 3D embeddings of 10×10 grid for both spectral embedding and Local Linear Embedding, with varying boundary conditions

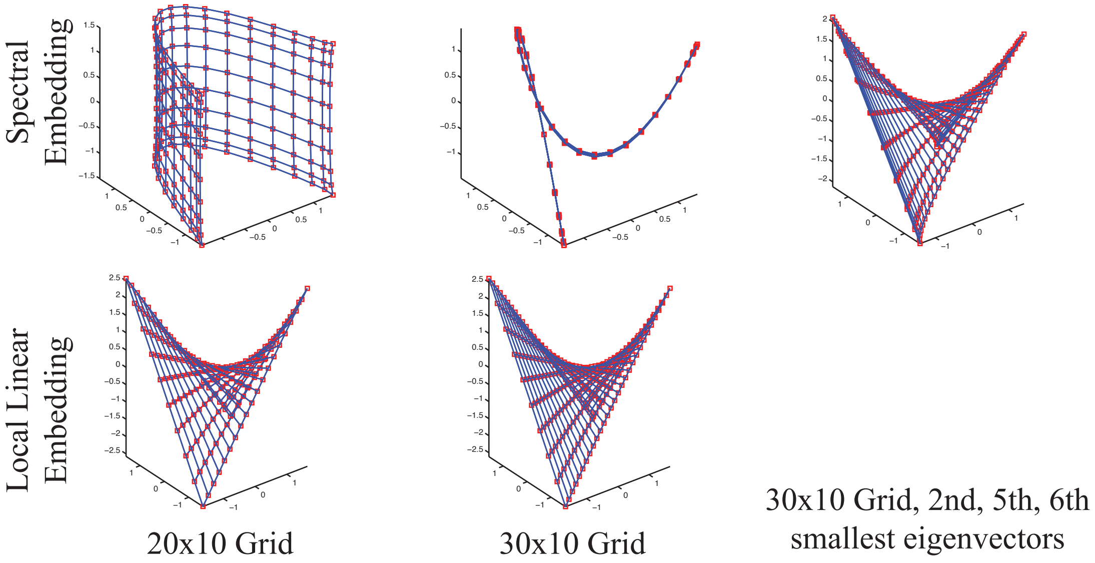
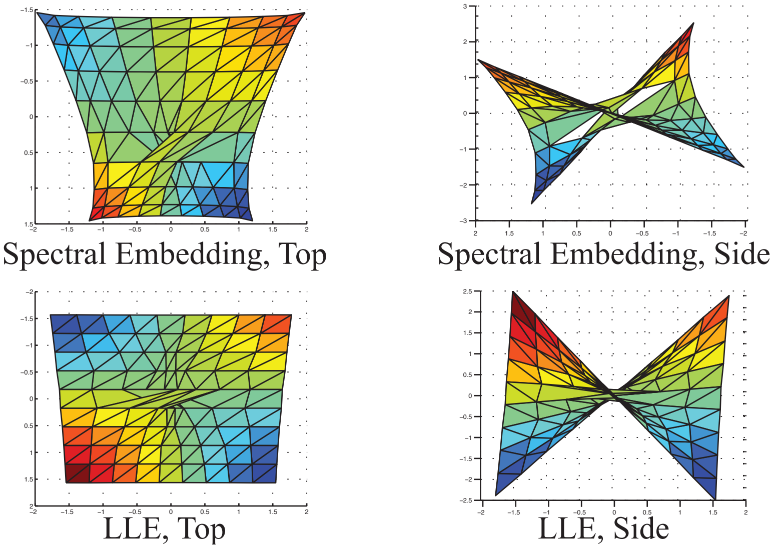


Figure 7.7: Comparison of 3D embeddings of 20×10 grid and 30×10 grid for both spectral embedding and locally linear embedding

local, a cylindrical piece of fabric can be generated by simply relating the rightmost column to the leftmost column of stitches, while a toroidal piece of fabric can be generated by also relating the topmost row to the bottommost row. The results of both methods and all three boundary types are shown in Figure 7.6. Note that in 3D, the plane is embedded in a saddle point configuration. For the cylindrical and toroidal boundary conditions, spectral embedding properly predicts that the sheet will roll up into a cylinder and torus, respectively. The implementation of LLE, however, has issues with these conditions and produces sub-optimal shapes.

As a second test, both algorithms were also run on 20×10 and 30×10 grids, with planar boundary conditions. Results are shown in Figure 7.7. Note that as the grid becomes more and more rectangular, the spectral embedded version begins taking on alternate shapes, finally collapsing entirely in one direction forming a 1D chain. However, the original saddle point embedding is still present, but is no longer the smallest eigenvectors; instead it is present in the



Spectral Embedding, Top

Spectral Embedding, Side

LLE, Top

LLE, Side

Figure 7.8: Comparison of 3D embeddings of a cable knit pattern

embedding derived from the second, fifth, and sixth smallest eigenvectors. This hints at phase transitions as the eigenvalues for certain configurations grow and shrink over time, causing different configurations to be found as the "best" embedding, even though these different configurations may be very undesirable from the perspective of yarn geometry generation. In comparison, LLE continues to find the saddle point configuration even as the sheet becomes more and more rectangular.

Ultimately, it seems as if unfortunately neither of these methods is ideally suited to the task as-is. Both methods seem to need human intervention in certain cases (for spectral embedding, finding the particular eigenvectors corresponding to the embedding desired, and for LLE handling cylindrical boundary conditions, which occur for the case of knitting in the round). A further sticking point arises in LLE, which produces weights that are invariant to translation, rotation, and scaling. While the first two are desirable properties to have, being invariant to scaling means that distances may be locally distorted. This has a particular impact on more complicated knit patterns like the cable knit shown in Figure 7.8. Recall that in a cable knit, certain rows consist of interchanges, where loops 3, 4, and 5 are pulled through loops 6, 7, and 8 in the previous row, and vice versa (see Figure 2.3). This creates a 3D structure of cabling if repeated. For the spectral embedded version, this 3D structure can be observed as the interchanges cause a separation between the loops going on top and the loops going on the bottom. In the LLE version, however, because the algorithm generates scale-invariant weights, the distances in the interchange row are longer than they should be, and as a result there is not nearly as much separation between the top and bottom sets of loops. Although these seem like promising ideas for generating cloth geometry automatically, future work needs to be done to improve these issues of robustness and quality.

CHAPTER 8

CONCLUSION

The methods and techniques in this thesis have led to a robust and scalable technique for simulating knitted cloth at the yarn level. The complex geometry of knitted cloth can now be simulated directly with garment sizes significantly larger than previous research. Validation shows that the yarn-based model reproduces characteristics of different knits automatically, as a result of the interactions and contacts between yarns, and that these features are difficult to replicate using the common elastic sheet model. This yarn-based model can then be accelerated by taking advantage of local temporal coherence in the contacts. While an approximation, it is capable of creating visually indistinguishable results in a fraction of the time, allowing for character-sized garments to be simulated in a practical amount of time.

It is hoped that this work will be especially valuable in developing new applications for the textile community, particularly for rapid design of clothing, by allowing designers to see how knitted materials will drape and react without having to actually create the material. It is also applicable anywhere visual accuracy is of the utmost importance, such as for large, loose knits in computer animation, where individual yarns are visible in the frame and incorrect motion may be visually distracting. Finally, the methods used to accelerate the model may also be useful in other scenarios with similar properties—namely, with dense and coherent contact structure. For instance, hair is another material with a rich set of contacts which might be temporally coherent in many common circumstances (shorter hair, no wind, etc) and which could be accelerated using similar techniques.

From this, there is a wide range of possibilities for future avenues of research. There is nothing that specifically limits yarn-based models to knits versus woven materials; rather, knits were considered first because of their more obviously nonlinear dynamics, so simulating yarn-based knit models provides a more dramatic variance from previous sheet-based methods. Other than the nonrigid damping of Section 4.2.3, though, there is nothing in the model which is explicitly aware of the knitted relationships in the yarn, and even that can be easily adapted to woven fabric structure. Although the yarn interactions of woven materials produce more subtle differences from linear elastic sheet models than knits, they are still present, and there may be cases where simulation of woven materials requires yarn-based modeling. For example, there has already been work on engineering applications where individual yarn behavior is important to capture for correctness, i.e. modeling bulletproof armor [45, 122, 123].

This ties into an important fact: even as computational power continues to increase, there will still be scenarios where elastic sheet-based cloth simulation makes more sense because the extra detail of yarn-based models are unimportant for that application and the associated extra cost excessive. For example, elastic sheet-based cloth may be the preferred method when the clothing is far away from the camera, or the clothing is on a character in a dense crowd, or the application needs to be real-time. Instead, the two models can and should coexist, and in fact interesting future work may focus on hybridizations of the two. For instance, sheet-based and yarn-based models may be able to be used together depending on the level-of-detail needed, where the cloth is simulated as a sheet until the yarn-based motion is apparent, at which point yarns are simulated directly. While this seems similar to the model reduction methods proposed in Chapter 5, that was concerned with maintaining perceived quality of motion at all scales, while for certain applications it may instead be suitable to look at yarn-based cloth models as a tool to add detail to existing sheet-based simulations where desired. Yarn-based simulations could also be used as a source of data or validation for sheet-based models; for instance, they could allow for rapid computation of force response curves of various fabrics which could then be plugged into a nonlinear sheet-based simulator such as the one proposed by Volino et al. [113].

There is also additional work that can be done to improve the yarn model itself. The absence of a complete treatment of inter-yarn friction is a notable limitation, although the problem is exceedingly difficult due to the large numbers of interrelated and distributed contacts. Friction is particularly important because it is one of the main factors in cloth hysteresis, and while the revised non-rigid damping and internal plasticity of Section 4.5.2 helps to allow simulated cloth to settle in path-dependent rest states, they are just heuristics, and a full friction model would most likely help to further improve realism and accuracy.

Although the simulator presented is coded for multicore architectures, more and more the future of computer graphics, and computing in general, is obviously moving to manycore systems like GPUs and other highly data-parallel vector architectures. Because of the immense size and highly parallel nature of yarn-based simulations, the GPU seems like a perfect match for achieving additional performance gains. However, although much of the model can be moved to a GPU implementation in a straightforward manner, other pieces require engineering and simulation challenges to be overcome before yarn-based models can be simulated easily on the GPU.

Finally, as noted in Chapter 7, additional advances need to be made in the construction of yarn-based models. While current approaches work for simple models, the complexity of knit construction combined with the need for sophisticated artistic/stylistic direction makes for a challenging modeling problem. Ultimately, this may result in multiple approaches for addressing the problem depending on requirements, with textile designers generating exact models from knitting instructions while artists generate plausible models from stylistic direction. Hopefully, though, in the future there will be a wide variety of yarn geometries to which the techniques of yarn-based cloth simulation can be applied.

BIBLIOGRAPHY

[1] Steven S. An, Theodore Kim, and Doug L. James. Optimizing cubature for efficient integration of subspace deformations. *ACM SIGGRAPH Asia*, 27(5):164:1–11, 2008.

[2] Uri M. Ascher and Linda R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, Philadelphia, 1998.

[3] Pierre Badel, Emmanuelle Vidal-Sallé, and Phillipe Boisse. Computational determination of in-plane shear mechanical behaviour of textile composite reinforcements. *Computational Materials Science*, 40:439–448, 2007.

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

[5] David Baraff, Andrew Witkin, and Michael Kass. Untangling cloth. *ACM Transactions on Graphics*, 22(3):862–870, 2003.

[6] Jernej Barbič and Doug James. Real-Time subspace integration for St. Venant-Kirchhoff deformable models. *ACM Transactions on Graphics*, 24(3):982–990, August 2005.

[7] Miklós Bergou, Basile Audoly, Etienne Vouga, Max Wardetzky, and Eitan Grinspun. Discrete viscous threads. *ACM Transactions on Graphics*, 29(4):116:1–116:10, 2010.

[8] Miklós Bergou, Saurabh Mathur, Max Wardetzky, and Eitan Grinspun. TRACKS: Toward Directable Thin Shells. *ACM Transactions on Graphics*, 26(3):50:1–50:10, 2007.

[9] Miklós Bergou, Max Wardetzky, Stephen Robinson, Basile Audoly, and Eitan Grinspun. Discrete elastic rods. *ACM Transactions on Graphics*, 27(3):63:1–63:12, 2008.

[10] Florence Bertails, Basile Audoly, Marie-Paule Cani, Bernard Querleux, Frédéric Leroy, and Jean-Luc Lévêque. Super-helices for predicting the dynamics of natural hair. *ACM Transactions on Graphics*, 25(3):1180–1187, August 2006.

[11] Kiran Bhat, Christopher Twigg, Jessica Hodgins, Pradeep Khosla, Zoran Popovic, and Steven Seitz. Estimating cloth simulation parameters from video. In *Proc. SCA '03*, pages 37–51. Eurographics Association, 2003.

[12] P. Boisse, N. Hamila, F. Helenon, B. Hagege, and J. Cao. Different approaches for woven composite reinforcement forming simulation. *International Journal of Material Forming*, 1(1):21–29, 2008.

[13] David Breen, Donald House, and Michael Wozn. A particle-based model for simulating the draping behavior of woven cloth. *Textile Research Journal*, 64(11):663–685, November 1994.

[14] Kathryn Eleda Brenan, S L Campbell, and Linda Ruth Petzold. *Numerical solution of initial-value problems in differential-algebraic equations*. Elsevier, New York, 1989.

[15] R. Bridson, S. Marino, and R. Fedkiw. Simulation of clothing with folds and wrinkles. *Symposium on Computer Animation*, 32:28–36, 2003.

[16] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. *ACM Transactions on Graphics*, 21(3):594–603, 2002.

[17] Rui Campos, Thomas Bechtold, and Christian Rohrer. Fiber friction in yarn — a fundamental property of fibers. *Textile Research Journal*, 73:721–726, 2003.

[18] Jeffrey N. Chadwick, Steven S. An, and Doug L. James. Harmonic shells: A practical nonlinear sound model for near-rigid thin shells. *ACM Transactions on Graphics*, 28(5):119:1–119:10, December 2009.

[19] Yanyun Chen, Stephen Lin, Hua Zhong, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Realistic rendering and animation of knitwear. *IEEE Transactions on Visualizations and Computer Graphics*, 9:43–55, 2003.

[20] K.F. Choi and T.Y. Lo. An energy model of plain knitted fabric. *Textile Research Journal*, 73:739–748, 2003.

[21] K.F. Choi and T.Y. Lo. The shape and dimensions of plain knitted fabric: A fabric mechanical model. *Textile Research Journal*, 76(10):777–786, 2006.

[22] K.F. Choi and S.K. Tandon. An energy model of yarn bending. *Journal of the Textile Institute*, 97:49–56, 2006.

[23] Kwang-Jin Choi and Hyeong-Seok Ko. Stable but responsive cloth. *ACM Transactions on Graphics*, 21(3):604–611, 2002.

[24] Lillian Chu. *A Framework for Extracting Cloth Descriptors from the Underlying yarn Structure*. PhD thesis, University of California, Berkeley, 2005.

[25] Fan Chung. *Spectral Graph Theory (revised online edition)*. AMS, http://www.math.ucsd.edu/ fan/research/revised.html, 1997.

[26] Gilles Debunne, Mathier Desbrun, Marie-Paule Cani, and Alan H. Barr.

Dynamic real-time deformations using space and time adaptive sampling. *ACM SIGGRAPH*, (31–36), 2001.

[27] A. Demiroz and T. Dias. A study of the graphical representation of plain-knitted structures part I: Stitch model for the graphical representation of plain-knitted structures. *Journal of the Textile Institute*, 91:463–480, 2000.

[28] M. Duhovic and D. Bhattacharyya. Simulating the deformation mechanisms of knitted fabric composites. *Composites Part A: Applied Science and Manufacturing*, 37(11):1897–1915, 2006.

[29] Damien Durville. Simulation of the mechanical behavior of woven fabrics at the scale of fibers. *International Journal of Material Forming*, 3(Suppl 2):1241–1251, 2010.

[30] Bernhard Eberhardt, Michael Meissner, and Wolfgang Strasser. Knit fabrics. In Donald House and David Breen, editors, *Cloth Modeling and Animation*, chapter 5, pages 123–144. A K Peters, 2000.

[31] Bernhard Eberhardt, Andreas Weber, and Wolfgang Strasser. A fast, flexible, particle-system model for cloth draping. *IEEE Computer Graphics and Applications*, 16(5):52–59, 1996.

[32] Elliot English and Robert Bridson. Animating developable surfaces using nonconforming elements. *ACM Transactions on Graphics*, 27(3):66:1–66:5, 2008.

[33] O. Etzmuss, M. Keckeisen, and W. Straßer. A fast finite element solution for cloth modelling. In *Computer Graphics and Applications, 2003. Proceedings. 11th Pacific Conference on*, pages 244–251, 2003.

[34] Roy Featherstone. *Rigid Body Dynamics Algorithms*. Springer, 2007.

[35] C.A. Felippa. A systematic approach to the element-independent corotational dynamics of finite elements. *Center for Aerospace Structures Document Number CU-CAS-00-03, College of Engineering, University of Colorado*, 2000.

[36] Wei-Wen Feng, Yizhou Yu, and Byung-Uck Kim. A deformation transformer for real-time cloth animation. *ACM SIGGRAPH*, 29(4):108:1–108:9, 2010.

[37] Jie Gao, Leonidas J. Guibas, and An Nguyen. Deformable spanners and applications. *Proc. 20th ACM Symp. on Comp. Geom.*, pages 179–199, 2004.

[38] O. Göktepe and S. C. Harlock. Three-dimensional computer modeling of warp knitted structures. *Textile Research Journal*, 72:266–272, 2002.

[39] Rony Goldenthal, David Harmon, Raanan Fattal, Michel Bercovier, and Eitan Grinspun. Efficient simulation of inextensible cloth. *ACM SIGGRAPH*, 26(3), 2007.

[40] Herbert Goldstein, Charles Poole, and John Safko. *Classical Mechanics*. Addison Wesley, 3rd edition, 2002.

[41] Naga K. Govindaraju, David Knott, Nitin Jain, Ilknur Kabul, Rasmus Tamstorf, Russell Gayle, Ming C. Lin, and Dinesh Manocha. Interactive collision detection between deformable models using chromatic decomposition. *ACM Transactions on Graphics*, 24(3):991–999, 2005.

[42] Mireille Grégoire and Elmar Schömer. Interactive simulation of one-dimensional flexible parts. *Computer-Aided Design*, 39(8):694–707, 2007.

[43] Eitan Grinspun, Anil Hirani, Mathieu Desbrun, and Peter Schröder. Discrete shells. *Symposium on Computer Animation*, pages 62–67, 2003.

[44] Eitan Grinspun, Petr Krysl, and Peter Schröder. CHARMS: A simple framework for adaptive simulation. *ACM Transactions on Graphics*, 21:281–290, 2002.

[45] Bohong Gu. Ballistic penetration of conically cylindrical steel projectile into plain-woven fabric target - a finite element simulation. *Journal of Composite Materials*, 38(22):2049–2074, 2004.

[46] L.J. Guibas. Kinetic Data Structures. In D. Mehta and S. Sahni, editors, *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, 2004.

[47] Benjamin Hagège, Phillipe Boisse, and Jean-Louis Billoët. Finite element analyses of knitted composite reinforcement at large strain. *European Journal of Computational Mechanics*, 14(6–7):767–776, 2005.

[48] Ernst Hairer and Gerhard Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer, New York, 2nd edition, 2002.

[49] N. Hamila, P. Boisse, F. Sabourin, and M. Brunet. A semi-discrete shell finite element for textile composite reinforcement forming simulation. *International Journal for Numerical Methods in Engineering*, 79:1443–1466, 2009.

[50] Nahiène Hamila and Phillipe Boisse. Simulations of textile composite reinforcement draping using a new semi-discrete three node finite element. *Composites Part B: Engineering*, 39:999–1010, 2008.

[51] David Harmon, Etienne Vouga, Breannan Smith, Rasmus Tamstorf, and Eitan Grinspun. Asynchronous contact mechanics. *ACM Transactions on Graphics*, 28(3):87:1–87:12, 2009.

[52] David Harmon, Etienne Vouga, Rasmus Tamstorf, and Eitan Grinspun. Robust treatment of simultaneous collisions. *ACM Transactions on Graphics*, 27(3):23:1–23:4, 2008.

[53] PM Hubbard. Collision detection for interactive graphics applications. *IEEE Trans. Visualization and Computer Graphics*, 1(3):218–230, 1995.

[54] D. Hutchinson, M. Preston, and T. Hewitt. Adaptive refinement for mass/spring simulations. In *Proceedings of the Eurographics workshop on Computer animation and simulation'96*, page 45. Springer-Verlag New York, Inc., 1996.

[55] Yuki Igarashi, Takeo Igarashi, and Hiromasa Suzuki. Knitting a 3d model. *Computer Graphics Forum*, 27(7):1737–1743, 2008.

[56] Feng Ji, Ruqin Li, and Yiping Qiu. Simulate the dynamic draping behavior of woven and knitted fabrics. *Journal of Industrial Textiles*, 35(3):201–215, 2006.

[57] Y. Jiang and X. Chen. Geometric and algebraic algorithms for modelling yarn in woven fabrics. *Journal of the Textile Institute*, 96:237–245, 2005.

[58] Zhou Jinyun, Li Yi, Jimmy Lam, and Cao Xuyong. The poisson ratio and modulus of elastic knitted fabrics. *Textile Research Journal*, 80(18):1965–1969, 2010.

[59] Nebojsa Jojic and Thomas Huang. Estimating cloth draping parameters from range data. In *Proc. Intl Workshop on Synthetic-Natural Hybrid Coding and Three Dimensional Imaging*, pages 73–76, 1997.

[60] Jonathan Kaldor, Doug James, and Steve Marschner. Simulating cloth at

the yarn level. SIGGRAPH 2008 Computer Animation Festival, August 2008.

[61] S. Kawabata. *The standardization and analysis of hand evaluation*. The Textile Machinery Society of Japan, Osaka, 2nd edition edition, 1980.

[62] S. Kawabata, Masako Niwa, and H. Kawai. The finite deformation theory of plain-weave fabrics part I: The biaxial-deformation theory. *Journal of the Textile Institute*, 64:21–46, 1973.

[63] P. A. Kelly. A viscoelastic model for the compaction of fibrous materials. *Journal of the Textile Institute*, 2011.

[64] Lily Kharevych, Patrick Mullen, Houman Owhadi, and Mathieu Desbrun. Numerical coarsening of inhomogeneous elastic materials. *ACM Transactions on Graphics*, 28(3):1–8, 2009.

[65] Theodore Kim and Doug L. James. Skipping steps in deformable simulation with online model reduction. *ACM Transactions on Graphics*, 28(5):1–9, 2009.

[66] M.J. King, P. Jearanaisilawong, and S. Scorate. A continuum constitutive model for the mechanical behavior of woven fabrics. *International Journal of Solids and Structures*, 42:3867–3896, 2005.

[67] Hua Lin, Martin Sherburn, Jonathan Crookston, Andrew C. Long, Mike J. Clifford, and I. Arthur Jones. Finite element modeling of fabric compression. *Modelling and Simulation in Materials Science and Engineering*, 16(3):035010, 2008.

[68] Wai-Sze Lo, Ka-Fai Choi, and T.Y. Lo. Measurement of yarn bending and

torsion rigidities of naturally curved yarns part 1: Monofilament yarn in helical shape. *Textile Research Journal*, 80(18):1875–1886, 2010.

[69] Sebastian Martin, Peter Kaufmann, Mario Botsch, Eitan Grinspun, and Markus Gross. Unified simulation of elastic rods, shells, and solids. *ACM SIGGRAPH*, 29(4):39:1–39:10, 2010.

[70] Masaru Matsuo and Tomoko Yamada. Hysteresis of tensile load – strain route of knitted fabrics under extension and recovery processes estimated by strain history. *Textile Research Journal*, 79(3):275–284, 2009.

[71] Anne Matthews. *Vogue Dictionary of Knitting Stitches*. The Condé Nast Publications, Ltd., New York, NY, 1984.

[72] Napaporn Metaaphanon, Yosuke Bando, Bing-Yu Chen, and Tomoyuki Nishita. Simulation of tearing cloth with frayed edges. *Computer Graphics Forum*, 28:1837–1844, 2009.

[73] Brian Mirtich. Timewarp rigid body simulation. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 193–200, July 2000.

[74] M. Müller and M. Gross. Interactive virtual materials. In *Proceedings of Graphics Interface 2004*, pages 239–246, 2004.

[75] Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow, and Barbara Cutler. Stable real-time deformations. In *ACM SIGGRAPH Symposium on Computer Animation*, pages 49–54, July 2002.

[76] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. In *Proc. Virtual Reality Interactions and Physical Simulations (VRIPhys)*, pages 71–80. Eurographics, 2006.

[77] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. *ACM SIGGRAPH*, 24(3):471–478, 2005.

[78] Ben Nadler, Panayiotis Papadopoulos, and David J. Steigmann. Multiscale constitutive modeling and numerical simulation of fabric material. *International Journal of Solids and Structures*, 43:206–221, 2006.

[79] Matthieu Nesme, Paul G. Kry, Lenka Jeřábková, and François Faure. Preserving topology and elasticity for embedded deformable models. *ACM Transactions on Graphics*, 28(3):1–9, 2009.

[80] Oliver Nocent, Jean-Michel Nourrit, and Yannick Remion. Towards mechanical level of detail for knitwear simulation. In V. Skala, editor, *WSCG 2001 Conference Proceedings*, 2001.

[81] James Norbury. *Traditional Knitting Patterns from Scandinavia, the British Isles, France, Italy, and other European Countries*. Dover Publications, Inc., New York, NY, 1973.

[82] Dinesh Pai. STRANDS: Interactive simulation of thin solids using Cosserat models. *Eurographics*, 21:347–352, 2002.

[83] Jung-Whan Park and Ae-Gyeong Oh. Bending mechanics of ply yarns. *Textile Research Journal*, 73:473–479, 2003.

[84] Jung-Whan Park and Ae-Gyeong Oh. Bending rigidity of yarns. *Textile Research Journal*, 76:478–485, 2006.

[85] Ethan M. Parsons, Tusit Weerasooriya, and Simona Socrate. Impact of woven fabric: Experiments and mesostructure-based continuum-level sim-

ulations. *Journal of the Mechanics and Physics of Solids*, 58(11):1995–2021, 2010.

[86] F.T. Peirce. The geometry of cloth structure. *Journal of the Textile Institute*, 28:T45–T97, 1937.

[87] X. Q. Peng and J. Cao. A continuum mechanics-based non-orthogonal constitutive model for woven composite fabrics. *Composites Part A: Applied Science and Manufacturing*, 36:859–874, 2005.

[88] D. G. Phillips, Canh-Dung Tran, W. B. Fraser, and G. H. M. van der Heijden. Torsional properties of staple fibre plied yarns. *Journal of the Textile Institute*, 101(7):595–612, 2010.

[89] P. Potluri, S.A. Ariadurai, and I. L. Whyte. A general theory for the deformation behavior of non-plain-weave fabrics under biaxial loading. *Journal of the Textile Institute*, 91:493–508, 2000.

[90] Xavier Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. *Proc. Graphics Interface '95*, pages 147–154, 1995.

[91] Xavier Provot. Collision and self-collision handling in cloth model dedicated to design garments. In *Computer Animation and Simulation '97: Proceedings of the Eurographics Workshop in Budapest, Hungary, September 2-3, 1997*, page 177. Springer, 1997.

[92] Stephane Redon, Nico Galoppo, and Ming C. Lin. Adaptive dynamics of articulated rigid bodies. *ACM Transactions on Graphics*, 24(3):936–945, 2005.

[93] Yannick Rémion, Jean-Michel Nourrit, and Didier Gillard. Dynamic animation of spline like objects. In V. Skala, editor, *Proc. WSCG'99*, 1999.

[94] Maggie Righetti. *Knitting in Plain English*. St. Martin's Press, New York, NY, second edition, 2007.

[95] Alec R. Rivers and Doug L. James. FastLSM: Fast lattice shape matching for robust real-time deformation. *ACM Transactions on Graphics*, 26(3):82, 2007.

[96] Damien Rohmer, Tiberiu Popa, Marie-Paule Cani, Stefanie Hahmann, and Alla Sheffer. Animation wrinkling: Augmenting coarse cloth simulations with realistic-looking wrinkles. *ACM SIGGRAPH Asia*, 2010.

[97] Sam Roweis and Lawrence Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, December 2000.

[98] Andrew Selle, Michael Lentine, and Ronald Fedkiw. A mass spring model for hair simulation. *ACM Transactions on Graphics*, 27(3):64:1–64:11, 2008.

[99] David J. Spencer. *Knitting Technology*. Woodhead Publishing Limited, third edition, 2001.

[100] Dan Spielman. Spectral graph theory and its applications, a tutorial at FOCS 2007 (http://www.cs.yale.edu/homes/spielman/sgta/), Retrieved Dec 2008.

[101] Jonas Spillmann and Matthias Teschner. CoRdE: Cosserat rod elements for the dynamic simulation of one-dimensional elastic objects. *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 63–72, 2007.

[102] Jonas Spillmann and Matthias Teschner. An adaptive contact model for the robust simulation of knots. *Eurographics*, 27:497–506, 2008.

[103] Montse Stanley. *Reader's Digest Knitter's Handbook*. Reader's Digest, 1999.

[104] Thomas Stumpp, Jonas Spillmann, Markus Becker, and Matthias Teschner. A geometric deformation model for stable cloth simulation. In *Proc. VRIPHYS*, Nov 2008.

[105] Huiyu Sun, Ning Pan, and Ron Postle. On the poisson's ratios of a woven fabric. *Composite Structures*, 68:505–510, 2005.

[106] Daina Taimina. *Crocheting Adventures with Hyperbolic Planes*. A K Peters, 2009.

[107] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. *Computer Graphics*, 21:205–214, 1987.

[108] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino. Collision detection for deformable objects. In *Computer Graphics Forum*, volume 24, pages 61–81. Blackwell Publishing, 2005.

[109] Adrien Theetten, Laurent Grisoni, Christian Duriez, and Xavier Merlhiot. Quasi-dynamic splines. In *Proc. ACM Symposium on Solid and Physical Modeling '07*, 2007.

[110] Savvas G. Vassiliadis, Argyro E. Kallivretaki, and Christopher G. Provatidis. Mechanical simulation of the plain weft knitted fabrics. *International Journal of Clothing Science and Technology*, 19(2):109–130, 2007.

[111] J. Villard and H. Borouchaki. Adaptive meshing for cloth animation. *Engineering with Computers*, 20(4):333–341, 2005.

[112] Pascal Volino, Martin Courchesne, and Nadia Magnenat-Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. In *ACM SIGGRAPH*, pages 137–144, 1995.

[113] Pascal Volino, Nadia Magnenat-Thalmann, and Francois Faure. A simple approach to nonlinear tensile stiffness for accurate cloth simulation. *ACM Transactions on Graphics*, 28, 2009.

[114] Pascal Volino and Nadia Magnenat Thalmann. Implementing fast cloth simulation with collision response. In *Proc. Computer Graphics International*, pages 257–266, 2000.

[115] T. Wada, S. Hirai, T. Hirano, and S. Kawamura. Modeling of plain knitted fabrics for their deformation control. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 3, pages 1960–1965, 1997.

[116] Barbara G. Walker. *A Fourth Treasury of Knitting Patterns*. Schoolhouse Press, Pittsville, WI, 2001.

[117] Huamin Wang, Florian Hecht Florian, Ravi Ramamoorthi, and James O'Brien. Example-based wrinkle synthesis for clothing animation. *ACM Transactions on Graphics*, 29(4):107:1–107:8, 2010.

[118] Huamin Wang, James O'Brien, and Ravi Ramamoorthi. Multi-resolution isotropic strain limiting. *ACM Transactions on Graphics*, 29(6):156:1–156:10, 2010.

[119] William Warren. The elastic properties of woven polymeric fabric. *Polymer Engineering and Science*, 30:1309–1313, 1990.

[120] Mingxing Xiao and Zhaofeng Geng. A model of rigid bodies for plain-weave fabrics based on the dynamics of multibody systems. *Textile Research Journal*, 80(19):1995–2006, 2010.

[121] Mark S. Yeoman, Daya Reddy, Hellmut C. Bowles, Deon Bezuidenhout, Peter Zilla, and Thomas Franz. A constitutive model for the warp-weft coupled nonlinear behavior of knitted biomedical textiles. *Biomaterials*, 31:8484–8493, 2010.

[122] X.S. Zeng, V. B. C. Tan, and V. P. W. Shin. Modelling inter-yarn friction in woven fabric armor. *International Journal for Numerical Methods in Engineering*, 66:1309–1330, 2006.

[123] G. M. Zhang, R. C. Batra, and J. Zheng. Effect of frame size, frame type, and clamping pressure on the ballistic performance of soft body armor. *Composites Part B: Engineering*, 39(3):476–489, 2008.

[124] Y. T. Zhang and Y. B. Fu. A micromechanical model of woven fabric and its application to the analysis of buckling under uniaxial tension: Part 1: The micromechanical model. *International Journal of Engineering Science*, 38(17):1895–1906, 2000.

[125] Y. T. Zhang and Y. B. Fu. A micro-mechanical model of woven fabric and its application to the analysis of buckling under uniaxial tension. part 2: buckling analysis. *International Journal of Engineering Science*, 39:1–13, 2001.

[126] Y. T. Zhang and J. F. Hu. Buckling analysis of woven fabric under simple shear along arbitrary directions. *Textile Research Journal*, 72:147–152, 2002.