Real-Time Dynamic Wrinkling of Coarse Animated Cloth

Russell Gillette and Craig Peters,* Nicholas Vining, Essex Edwards and Alla Sheffer
University of British Columbia

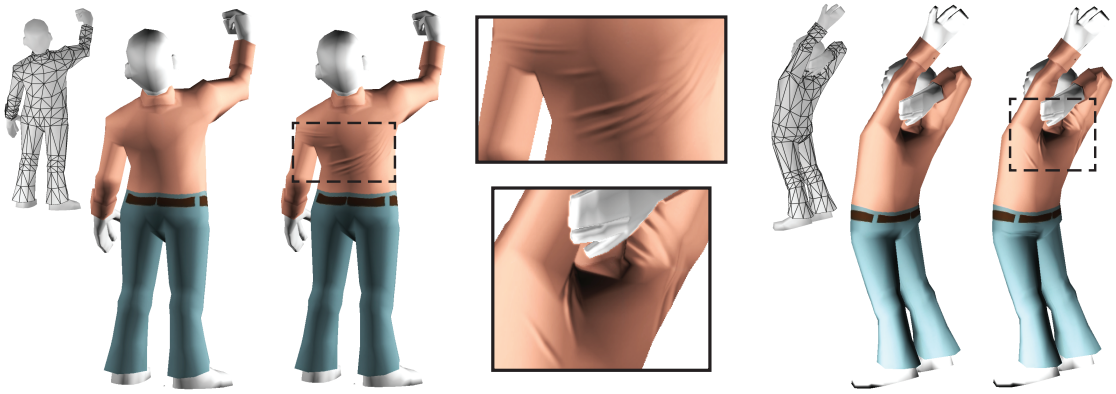


Figure 1: *Coarse game-level cloth animation augmented with realistic looking wrinkles in real-time.*

Abstract

Dynamic folds and wrinkles are an important visual cue for creating believably dressed characters in virtual environments. Adding these fine details to real-time cloth visualization is challenging, as the low-quality cloth used for real-time applications often has no reference shape, an extremely low triangle count, and poor temporal and spatial coherence. We introduce a novel real-time method for adding dynamic, believable wrinkles to such coarse cloth animation. We trace spatially and temporally coherent wrinkle paths, overcoming the inaccuracies and noise in low-end cloth animation, by employing a two stage stretch tensor estimation process. We first employ a graph-cut segmentation technique to extract spatially and temporally reliable surface motion patterns, detecting consistent compressing, stable, and stretching patches. We then use the detected motion patterns to compute a per-triangle temporally adaptive reference shape and a stretch tensor based on it. We use this tensor to dynamically generate new wrinkle geometry on the coarse cloth mesh by taking advantage of the GPU tessellation unit. Our algorithm produces plausible fine wrinkles on real-world data sets at real-time frame rates, and is suitable for the current generation of consoles and PC graphics cards.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling; I.3.8 [Computer Graphics]: Applications—;

Keywords: cloth animation, wrinkle augmentation, computer games, real time cloth

1 Introduction

Plausible cloth deformation and wrinkling provides important visual cues necessary for believable characters in virtual environments. Recent advances in GPU technology and parallelism have now made coarse cloth animation possible for real-time applications; however, these low-end cloth animations operate on meshes with low triangle counts and lack the fine wrinkling patterns exhibited by real-world cloth. Running a sufficiently accurate cloth simulation to capture fine wrinkling behaviours is both too computationally expensive for real-time and challenging to solve numerically even in offline production rendering. We introduce a new method for achieving plausible wrinkling on low end, coarse cloth for games and virtual environments with real-time performance (Figure 1).

Cloth animation for games is not typically generated by a pure physics simulation; the shape of a garment on a character reacts to player input and is driven by a combination of coarse simulation, inverse kinematics, and animation blending from multiple sources [Lander 1999]. Rather than trying to capture fine wrinkling in the simulation, we are motivated by offline approaches for adding wrinkling and folds to high-quality cloth simulation as a post-process [Müller and Chentanez 2010; Kavan et al. 2011; Rohmer et al. 2010b], Figure 2, top. These post-processing methods are motivated by the key observation that cloth is incompressible [Hu 2004]. We can measure whether or not cloth is compressed by comparing its shape to a known *reference shape*, which is free of compression and stretch. Since cloth is effectively incompressible, any compression observed in the animated cloth with respect to the reference shape is thus viewed as due to insufficient animation resolution. The post-process therefore resolves the compression and accounts for the excess material by forming wrinkles perpendicular to the direction of the compression.

Cloth meshes for video games are generated without a meaningful reference shape [Enqvist 2010], thus we have no ground truth to measure compression with respect to. We therefore construct an approximate frame of reference, in the form of a local *per-triangle* reference shape, which is accurate enough with respect to recently observed frames of animation to create temporally plausible wrinkles. This reference shape is allowed to change and evolve over time as animation is seen by the viewer, and is progressively updated as new frames are processed in response to user input and intrinsic changes in triangle shape are measured. In each frame, we then compute the compression of the current triangle with respect to the per-triangle reference shape, and use this information to seed and modify wrinkles.

*Joint First Authors

Accurately determining a plausible per-triangle reference frame on the fly from newly generated frames of animation requires us to overcome the inaccuracies and noise often present in typical targeted real-time animation; specifically, triangle compression measured with respect to the previous input frame is noisy. To update the local per-triangle reference shape in a temporally coherent fashion, we first employ a graph-cut segmentation technique to reliably detect intrinsic deformation patterns in the animation, then use the detected deformation patterns to compute and update the individual triangle reference shapes. Our motion analysis ensures that the compression and stretch tensors with respect to the per-triangle rest pose, as well as the generated wrinkles, are spatially consistent between adjacent triangles and temporally consistent across consecutive animation frames. We then use the current reference shape to compute a surface compression field in the current frame, modifying it as new animation becomes available. This technique for compression measurement runs in real-time, handles low triangle count meshes, and requires no pre-processing.

We use the per-triangle compression field to trace wrinkle paths across areas of high compression. We directly operate on the piecewise constant tensor data, placing wrinkles in areas of high compression and solving for spatially and temporally smooth wrinkle paths that are well aligned with the desired fold directions. As our input meshes are coarse and not capable of capturing wrinkle detail, we use programmable GPU tessellation to adaptively generate wrinkle geometry and normals on the GPU. Taking advantage of programmable hardware in this fashion enables us to generate cloth wrinkle animations on low-end animations at a sustained framerate of 60 frames per second, making it ideally suited for game environments.

Our key technical contributions are three-fold. First, we introduce a method for generating a temporally adaptive reference shape for typical video game cloth animation sequences, consisting of low-triangle count, hand-animated meshes. Our approach makes no assumptions about cloth developability, does not rely on a parameterization of the underlying mesh, has no specific authoring requirements, and does not rely on training data sets that require updating for every new garment. Our second contribution is a method for dynamically seeding and evolving wrinkle paths on a coarse cloth mesh following a piecewise constant per-triangle compression field. Finally, we show how to generate and render wrinkle geometry on cloth in real-time, taking full advantage of GPU tessellation and shading capabilities for parallelism.

We demonstrate our method on a variety of animated garments taken from real-world video game titles. We validate our approach by comparing it to alternative approaches and artist drawn static texture-level folds.

2 Related Work

Realistic cloth simulation has long been an area of interest in the graphics research community, starting with the foundational work of [Baraff and Witkin 1998], and is of key importance in film, animation, and virtual environments [Choi and Ko 2005]. Many of the issues addressed by cloth simulation, such as collision handling, are outside the scope of this paper. Our work falls in the category of "wrinkle augmentation" research, which attempts to refine a coarse cloth simulation with fine details.

In mechanics, the wrinkling effect of a physical compressive force applied to a thin sheet of textile material is known as *buckling* and is particularly apparent in sheets that strongly resists stretch and compression, where out-of-plane buckling is energetically favorable to almost any in-plane deformation. Typical formulations of cloth simulation with buckling produce a stiff set of equations due

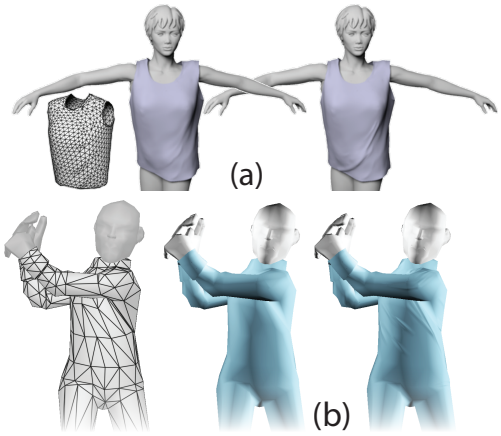


Figure 2: *(a) Off-line wrinkling of simulated clothing using a user-provided reference shape [Rohmer et al. 2010b]. (b) Our method adds wrinkles to a coarse cloth mesh typical of real-time applications, where no reference shape exists and where the animation is too coarse to capture fine intrinsic motion.*

to the strong penalty on stretch and compression that are both unstable and highly non-linear [Choi and Ko 2002]. To address the problem of small time steps, Baraff and Witkin [1998] use an implicit integration approach. Choi and Ko [2002] build upon this work using a non-linear formulation of bending energy which overcomes the "post-buckling instability" of previous work. The problem of buckling has led to novel approaches such as using non-conformal elements to eliminate stretch and compression during simulation [English and Bridson 2008] and building strain limits directly into a continuum-based deformation model [Thomaszewski et al. 2009]. Wrinkle augmentation research has attempted to sidestep these problems by adding wrinkling and fine detail to cloth after the initial simulation is complete.

Offline Wrinkle Augmentation Bergou et al. [2007] use constrained Lagrangian mechanics to add physically-based details such as wrinkles to an artist animated thin shell, such as cloth; this idea was later refined by Rémillard and Kry [2013] to simulate the detailed deformation of a thin shell on the surface of a deformable solid. Rohmer et al. [2010b; 2010a] compute a smooth stretch tensor field between the animation frames and a reference garment shape and use model wrinkles to follow paths traced on this field in areas of high compression. They assume the underlying animation to be sufficiently smooth in time and space to result in visually coherent wrinkles. We are inspired by this work in our use of a stretch tensor field on the mesh to measure deformation; however we do not assume the existence of a valid reference frame nor expect the garment motion to be smooth. In contrast to these offline approaches our method is fast enough to provide real-time performance for interactive applications.

Data-Driven Approaches A range of data-driven approaches wrinkle cloth meshes by utilizing information gained from offline physical simulations. Guan et al. [2012] combine a database of simulated fine garments with a parametrization of the underlying physical body to generate visually appealing results. Kim et al. [2013] use a novel compression scheme to make the traversal of over 33 gigabytes of cloth training data tractable in real time. Zurdo et al. [2013] compute a physical simulation on high-resolution meshes offline and simplify it to a coarser mesh in real time. Kavan et al. [2011] use a learning method to "upsample" coarse cloth simulations in real-time, using harmonic basis functions learned offline on sample animations. Wang et al. [2010] pro-

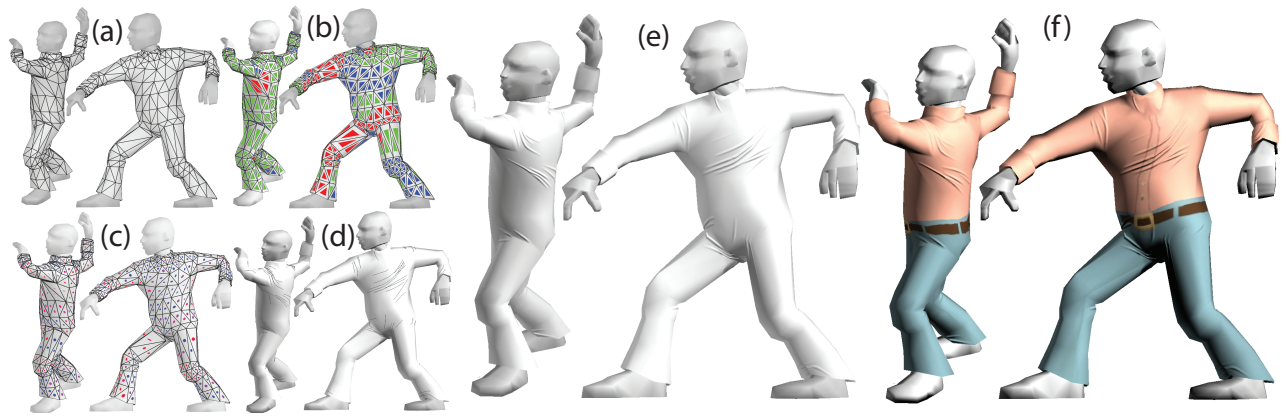


Figure 3: *Algorithm components: (a) input animation frames; (b) compression(blue)/stretch(red)/neutral(green) labeling; (c) local stretch tensors shown by oriented ellipses; (d) temporally coherent wrinkle paths; (e) final wrinkled cloth rendered at real-time without and (f) with texture.*

duce impressive results using an example-based approach, merging data from a wrinkle database indexed by joints on the human body. This work is only suitable for tight-fitting clothing, and does not support loose clothing such as dresses or skirts. Hahn et al. [2014] combine machine learning with a dynamically updated subspace basis to produce impressive wrinkling and torsional folds, but this approach is not fast enough for real-time applications; it requires close-fitting clothing rigged to a skeleton, a reference shape, and a set of training simulations. The data-driven methods are most suited for wrinkling garment animations with similar design and motion to the training data. Adding a new piece of a garment typically requires the construction of new training sets. Our framework does not require training data or a reference shape.

Real-Time Cloth Animation and Wrinkling Real-time cloth simulation for games typically uses a mass and spring system on a coarse mesh [Lander 1999], connecting vertices with simple springs designed to apply shearing and bending forces while maintaining garment structure. These mass and spring systems form a series of differential equations that are typically integrated using a stable integration method, e.g. [Verlet 1967]. A detailed example of a cloth simulation in a commercially available video game is discussed by Enqvist [2010]. Typical real-time cloth simulation is very coarse and crude (e.g. 600 to 800 vertices [Wang et al. 2010]), and does not address wrinkling or buckling in any part of the simulation.

Adding dynamic wrinkles in video games typically favours simple strategies. Oat [2007] introduces artist-painted wrinkle maps designed to fade-in along key areas of the mesh as cloth stretches and compresses. Wrinkle maps are easy to implement, but are static and cannot simulate dynamic wrinkle evolution or react to movement not anticipated by the artist. A similar idea was presented by Hadap et al. [1999], who deform a user-defined wrinkle pattern in response to animation. Müller and Chentanez [2010] attach a high-resolution "wrinkle mesh" to a coarse cloth simulation and run a static solver in parallel with the motion of the base mesh to animate the fine-grained wrinkles. This approach requires a reliable reference shape and assumes the coarse cloth motion to be spatially and temporally smooth, which is rarely the case for game-type cloth animations.

3 Overview

Our method addresses the creation of believable, real-time wrinkles on top of low-quality animations of coarse meshes. In this setup we can no longer rely on the surface animation to correctly capture

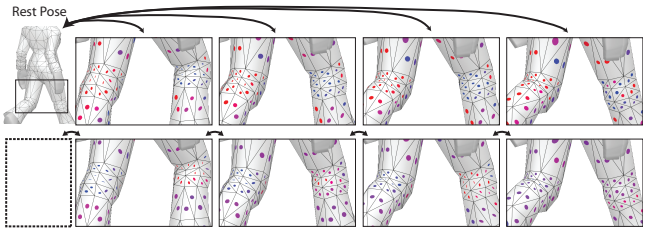


Figure 4: *Typical game animation frames with intrinsic deformation (stretch) tensor computed with respect to the first (top) and previous (bottom) frame. The shape of the tensor shows compression stretch magnitude ratio and color reflects the larger between the eigenvalues (blue for compression, red for stretch). Using compression on the top tensor as cue for wrinkling, will generate no wrinkles on the left knee (an inverse reference pose will generate no wrinkles on the right). The local tensor (bottom) provides better, but noisy cues.*

even the coarse intrinsic surface motion, and have no expectation that the mesh will exhibit coherent, temporally and spatially smooth deformation with respect to some static reference frame [Rohmer et al. 2010a; Müller and Chentanez 2010] (see Figure 4, top). Since we focus on believability rather than correctness, we assume that humans largely rely on local movement when predicting wrinkle appearance on surfaces. Surfaces lacking wrinkles appear plausible as long as the intrinsic geometry is largely unchanged, and human observers expect wrinkles to show up when the surface visibly contracts and expect these to dissolve when a surface stretches following contraction. Real-life wrinkles are also persistent - appearing, moving and disappearing gradually. We therefore can expect that humans will anticipate similar behavior from virtual wrinkles - expecting them to evolve gradually, and to stay in place absent underlying surface motion.

Following these observations, we are motivated to analyze *local* intrinsic surface deformation to predicting wrinkle appearance. While local deformation across the model shows clear contraction/stretch patterns (Figure 4, bottom), this deformation is both spatially and temporally noisy when evaluated on a per-triangle basis. We denoise this data by using a two step stretch tensor extraction process (Section 4). First we use a graph-cut based approach to detect coherent contraction and stretch patterns (Section 4.1, Figure 3, b); then, to maintain wrinkle persistence across time we construct a temporally local per-triangle reference shape (Section 4.2). As new frames are rendered, we smoothly change the reference triangles over time to ensure that the tensors – and the wrinkles subse-

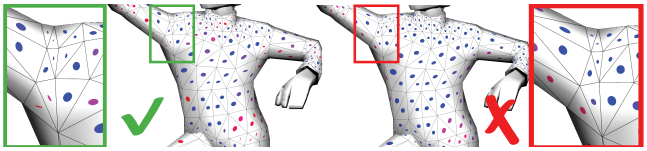


Figure 5: *On coarse meshes smoothing the piece-wise constant tensor field (left) to generate a piece-wise liner one (right) leads to loss of details, such as the compression on the shoulder and across the chest (left) which are no longer distinguishable on the right.*

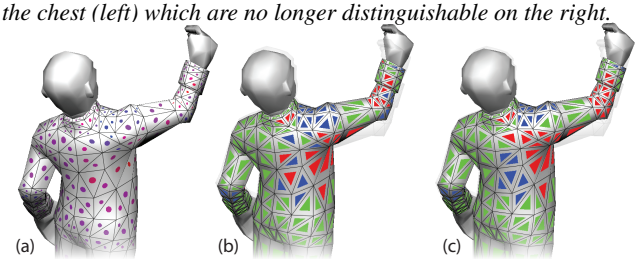


Figure 6: *Left to right: per-triangle stretch tensor with respect to previous frame (red to blue shows stretch to compression ratio); raw deformation classification (blue - compression, red - stretch, green - rest); spatially and temporally coherent classification.*

quently computed from them – exhibit smooth temporal behaviour.

We use the compression component of the stretch tensor field to trace wrinkle paths on the input garment surfaces (Section 5, Figure 3, d) generating both more and deeper wrinkles in areas where the compression is larger. The computed stretch tensor is constant per-triangle, but due to the coarseness of our meshes converting it into a piece-wise linear tensor field by first averaging adjacent tensors at mesh vertices and then using barycentric interpolation (as suggested by [Rohmer et al. 2010b], for instance) results in a loss of information (Figure 5, b). We therefore cannot leverage field smoothness to obtain temporally persistent and spatially smooth wrinkle paths. Instead, we directly optimize the shape of the individual wrinkle paths, balancing alignment with compression directions against spatial and temporal smoothness (Figure 9). Our optimization generates spatially smooth wrinkle paths that are both well-aligned with the compression field and persistent over time. Finally, we smoothly wrinkle the garment surface along the traced paths in real-time (Section 7, Figure 3, e). Existing wrinkling methods directly modify the animated mesh; to capture fine wrinkles they either require the input mesh to be sufficiently fine to represent wrinkle geometry, or perform on-the-fly mesh refinement to adequately capture the wrinkles [Rohmer et al. 2010b; Müller and Chentanez 2010]. Both approaches require significant memory and time overhead. Instead, we employ the tessellation shader to generate high-resolution wrinkle geometry on the GPU. We further improve the believability of the wrinkles by computing normals in the fragment shader to correctly shade the wrinkled material at an even higher resolution than the tessellated geometry. The combined method generates believable wrinkles (Figure 3, f) at a real-time speed of over 60 frames per second on typical inputs (Section 8).

4 Compression Field Construction

4.1 Compression Pattern Extraction

As discussed earlier, the first step in computing a reliable stretch tensor field suitable for wrinkle tracing is to locate the regions on the surface which undergo noticeable compression or stretch (Figure 6). The first indication of such changes occurring is the purely local deformation of an individual triangle within any given frame with respect to the previous frame; we may classify this behaviour as compressing, stretching, or resting. The strongest cue for this classification is given by the stretch tensor of the affine transformation between the two triangles (Section 4.1.1); the indicated amounts of compression and stretch provide a local measurement of surface behavior. However, while real-life cloth deformation is typically both spatially and temporally smooth, meshes for video games are typically coarse and the animation may be noisy, with artifacts such as inter-surface penetration and jitter. On such inputs, these raw measurements are an unreliable indicators of global surface behavior (Figure 6, b). Therefore rather than using the stretch and compression magnitudes directly to classify the current state of the mesh triangles, we use this data as input to a more sophisticated labeling process which balances these measurements against a preference for spatially and temporally continuous labels (Figure 6, c).

4.1.1 Triangle Stretch Tensor

Given a pair of current and reference triangles we measure the stretch and compression of the transformation between them using the stretch tensor from continuum mechanics [Bonet 1997]. Given a current triangle with edge vectors $(\mathbf{u}_1 = (v_1 - v_0), \mathbf{u}_2 = (v_2 - v_0))$, and reference edge vectors $(\bar{\mathbf{u}}_1, \bar{\mathbf{u}}_2)$, expressed as 2D edge vectors with respect to the local frames of the current and reference triangles, we define the *Deformation Gradient* as,

$$F = [\mathbf{u}_1, \mathbf{u}_2][\bar{\mathbf{u}}_1, \bar{\mathbf{u}}_2]^{-1}. \quad (1)$$

The *stretch tensor* is then defined as

$$U = \sqrt{F^T F}. \quad (2)$$

The 2x2 matrix U is symmetric positive definite, has eigenvectors pointing in the directions of maximal stretch and compression, and has eigenvalues λ_{max} and λ_{min} indicating the ratio of current length to rest length for stretch and compression respectively. We define and employ $\tilde{\lambda}_{min} = {}^1/_{\lambda_{min}}$ through the rest of the paper, as we find it more natural to work with. $\tilde{\lambda}_{min}$ is 1 when there is no compression, and approaches grows as the triangle compresses. Similarly, λ_{max} is 1 when there is no stretch, and grows as the triangle stretches.

4.1.2 Graph Cut Formulation

We formulate the problem of triangle labeling using a graph-cut framework. We solve for a label $l \in (C, S, R)$ per triangle where C indicates compression, S indicates stretch, and R indicates a neutral rest state. We construct a graph $G = (N, E)$ in which each node $i \in N$ is a tuple (f, t) where f is a face and t is a time step. Each node has three spatial neighbors (the three adjacent triangles in the mesh at time t) and two temporal ones (f in frames $t-1$ and $t+1$). Triangles along mesh boundaries as well as those in the first or last frames have fewer adjacencies.

For each node n we compute a unary cost for assigning it a particular label l_n, and for each pair of adjacent nodes we define a label-compatibility cost for each pair of label combinations assigned to them. We then find a labeling that minimizes the following discrete functional:

$$\sum_{i \in N} A(i, l_i) + \sum_{(i,j) \in E} B(l_i, l_j) \quad (3)$$

where $A(i, l_i)$ is the cost of assigning the label l_i to node i, E is the set of edges in G, and $B(l_i, l_j)$ is the binary cost of assigning labels l_i and l_j to nodes i and j.

Our unary costs are functions of the stretch tensor magnitudes $\tilde{\lambda}^i_{min}$ and λ^i_{max} over each triangle i (Figure 7). The rest label's cost is designed to be low when $\tilde{\lambda}_{min}$ and λ_{max} are both near 1, and grow

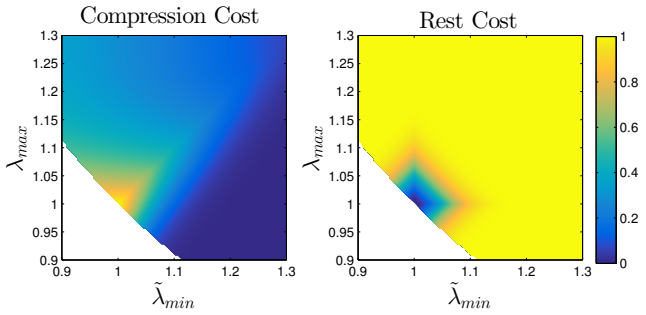


Figure 7: *The unary cost functions for label assignment depend on $\tilde{\lambda}^i_{min}$ and λ^i_{max}. Left: $A(i, C)$, Right: $A(i, R)$. These eigenvalues measure deformation between two frames, not to the rest frame.*

as they move away from 1 and is set using a symmetric Gaussian function,

$$A(i, R) = 1 - e^{-\|(\tilde{\lambda}^i_{min}-1, \lambda^i_{max}-1)\|_1^2/2\sigma^2}. \quad (4)$$

The stretching and compressing costs are defined symmetrically using the auxiliary function $c(a, b)$:

$$c(a,b) = \begin{cases} (H-h)e^{-(\frac{1}{2}(a-b)^2)/2\sigma^2} + h, & a > b \\ \frac{b}{a}(H-1)+1, & a < b \end{cases} \quad (5)$$

$$H = e^{-\alpha(a-1)}, \qquad h = e^{-\beta a}$$

$$A(i, C) = c(\tilde{\lambda}^i_{min}, \lambda^i_{max}) \quad (6)$$

$$A(i, S) = c(\lambda^i_{max}, \tilde{\lambda}^i_{min}) \quad (7)$$

We empirically set $\sigma = 0.05$, $\alpha = 9$, $\beta = 90$. The motivation for this design of the cost function is as follows. We want the cost of the compression label to be large when $\tilde{\lambda}^i_{min} \leq 1$, and want it to decrease both when $\tilde{\lambda}^i_{min}$ increases and when it increasingly dominates λ^i_{max} (i.e. when $\tilde{\lambda}^i_{min} - \lambda^i_{max}$ grows). We want a symmetric behavior for the cost of the stretch label.

The goal of the binary term $B(l_i, l_j)$ is to penalize label changes between adjacent faces. It depends only on the labels and is zero when l_i is equal to l_j, and is positive otherwise. As we expect the animation to be gradual, triangles should not immediately transition from compression to stretch without at some point resting; we therefore assign a higher cost of 0.4 for assigning C and S labels to adjacent triangles and a lower cost of 0.2 for assigning them the R and either C or S labels.We use the same costs for both temporal and spatial adjacencies. We solve the labeling problem using the solver of Boykov, Komolgorov, et al. [2001; 2004; 2004], which efficiently minimizes Equation 3.

As designed, the framework can be applied to a frame sequence of any length. In a scenario where an entire coarse animation sequence is available in advance, we can therefore label it all at once. In a real-time application where the animation is on the fly, we apply this framework using a one-lookahead window: given each new animation frame we fix the labels for the previously displayed frame and solve the optimization problem for the current frame, while taking the binary cost across the temporal edges linking the current frame to the fixed, previous, one into account. For the first frame in the animation this binary cost is treated as zero. This strategy allows for real-time labeling update and is sufficient to overcome temporal noise in all the animations we tested our method on.

4.2 Local Reference Triangles

To generate a stretch tensor that guides our wrinkle formation we require a local reference shape. Since no such shape is provided a priori we compute one on the fly as the animation progresses. Intuitively, for a perfect incompressible cloth animation, for each individual triangle in the mesh its most stretched, or largest instance in the animation sequence provides the ideal reference shape. However, in real-life data triangles can and do stretch due to noise and animation inaccuracy. Our on the fly-rest triangle estimation (Figure 8) is designed around these observations.

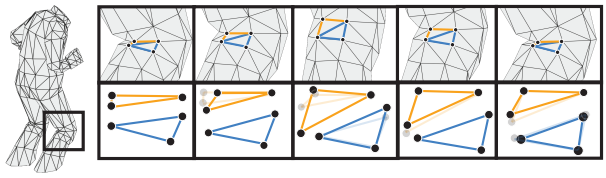


Figure 8: *Evolution of reference triangles (bottom) throughout the animation process.*

In the first frame of the animation, we set the reference triangles to be identical to the current one, as we have no other sources of information as to the plausible reference shape of the garment. The reference triangles are then smoothly updated as more information becomes available, with the update strategy reflecting the triangle's intrinsic motion as reflected by our labeling:

Compression. If a triangle is labeled as compressed we theoretically could leave its reference triangle unchanged. However, we anticipate some noise in our reference triangle estimation, and in particular want to avoid it reflecting outlier stretched triangles. Thus we choose to dampen the reference triangle size, relaxing it toward the current mesh triangles as these undergo compression. In order for our tensor field to remain smooth and consistent with previous frames, this relaxation must preserve the tensor eigenvectors, while smoothly changing the eigenvalues. We therefore directly modify the eigenvalues of the stretch tensor, and then solve for reference triangles that generate the updated stretch tensor, as follows.

Let $F = A\Sigma B^T$ be the singular value decomposition of the deformation gradient (Equation 1) of the transformation from the reference to the current triangle. Then, the stretch tensor U can be written as $B\Sigma B^T$, with the eigenvectors encoded by B and the eigenvalues on the diagonal Σ. We compute new eigenvalues, that lie closer to 1 by setting $\Sigma' = 0.95\Sigma + 0.05I$, here I is the identity matrix. We construct a new Deformation Gradient $F' = A\Sigma' B^T$ and compute the new reference triangles as

$$[\bar{\mathbf{u}}'_1, \bar{\mathbf{u}}'_2] = F'^{-1}[\mathbf{u}_1, \mathbf{u}_2] = B\Sigma'^{-1}A^T[\mathbf{u}_1, \mathbf{u}_2].$$

Stretching. We expect a stretching label to reflect a dissolving compressed, or wrinkled, triangle. In this configuration if both eigenvalues of the stretch tensor from the reference to the current triangle are larger than one they indicate a stretch beyond the current reference triangle. We interpret this configuration as dissolving of previously undetected compression wrinkles. We consequently replace the previous reference triangle with this, new, less compressed one. Since we aim to enrich the rendered garments, we prefer to err on the side of overestimating the reference triangle size, thus we use no smoothing or averaging in this scenario.

Resting. If a triangle is currently labeled as being in a rest state, we leave its reference triangle as is with no adjustment. This choice results in wrinkles that persist unchanged through periods of the animation with little deformation, such as a character holding a fixed pose.

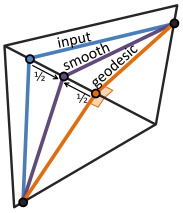
5 Wrinkle Path Tracing

The first step in creating wrinkles is tracing their paths on the mesh surface based on the computed stretch tensor computed with respect to the local reference shape. Our computation builds on many of the ideas described by Rohmer et al. [2010b]; however, in contrast to their formulation that assumes that the tensor field is piecewise linear and smooth, we modify the framework to operate directly on a piecewise constant field. The reason for the change is illustrated in Figure 5: since the meshes we operate on are exceedingly coarse, converting the per-triangle tensors into a piece-wise linear field (averaging the tensors at the vertices using tensor arithmetic [Rohmer et al. 2010b] and then using barycentric coordinates to define values across triangles) smooths out critical details. We refer the reader to [Rohmer et al. 2010b] for fine details on the wrinkle path tracing and here focus on the overall process and the main points of difference between the two algorithms.

5.1 Wrinkle Initialization

Wrinkles are traced in areas of high compression and are placed orthogonally to the direction of maximal compression. At each time step in the animation, we seed new wrinkles at random locations within highly compressed triangles (where $\tilde{\lambda}^i_{min}$ is greater than compression threshold τ), keeping the seeds at least a wrinkle width away from previously generated paths. (The computation of wrinkle width is discussed in Section 6). We then initialize the path by propagating a polyline across the mesh surface orthogonal to the maximal compression direction within each triangle. The propagation terminates once the compression magnitude drops below the compression threshold τ. It also terminates if the propagated path intersects another wrinkle path. The parameter τ is set to reflect the desired fabric stiffness [Rohmer et al. 2010b]; the expectation is that thinner fabrics, e.g. silk, will be more sensitive to compression and will also exhibit more and longer folds than thicker and stiffer materials such as wool or leather [Hu 2004].

A theoretical possibility is that the stretch tensor may have equal, high compression along both directions; this could occur when a surface region contracts simultaneously in all dimensions, in which case the choice of tracing direction is ill-posed. We have not encountered such situations in practice and expect them to be exceedingly rare. We believe that the best solution in this scenario would be to avoid seeding wrinkles in such triangles, and maintaining the previous wrinkle direction if a wrinkle reaches such a triangle during tracing.



While real wrinkles lie on the garment surface and hence follow the local curvature of this surface, they typically have low curvature in the tangential space of the garment. To mimic this behavior and eliminate undesirably sharp changes in wrinkle direction, we apply one iteration of tangential Laplacian smoothing to each path after tracing it, moving each intersection of a path with the mesh edges along this edge toward the shortest geodesic between the adjacent intersections (see inset).

5.2 Temporal Persistence

Real-life wrinkles are persistent, changing their shape and position gradually over time. To mimic such persistence instead of regenerating wrinkle paths in each frame from scratch, we first adjust existing wrinkles accounting for both the extrinsic and intrinsic deformation the garment undergoes between frames (Figure 9), and only then add new wrinkles away from the evolved ones.

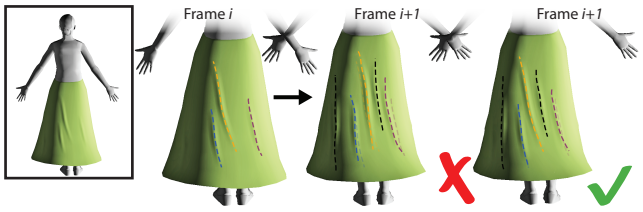
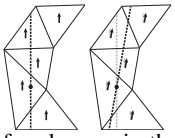


Figure 9: *Wrinkle paths generated independently per frame vary significantly in both direction and length as highlighted when rendering both current and previous paths (center); our temporally coherent wrinkle paths change gradually across all modalities (right).*

A light-weight approach used by Rohmer et al. [2010b] is to move wrinkle seed points based on changes in the stretch tensor field and then repropagate the wrinkle paths from scratch. However on coarse meshes using this approach can drastically move the traced wrinkles following even minor directional change in the field (see inset). To achieve the desired temporal persistence while accounting for changes in the stretch field directions and magnitudes we employ a global optimization framework.



We represent each wrinkle path as a polyline whose vertices lie on the edges of the garment mesh, and encode them as linear combinations of the edge end vertices:

$$p_i = v_i^1 t_i + v_i^2 (1 - t_i). \tag{8}$$

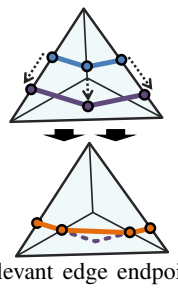
We optimize the shape of the path balancing three terms: orthogonality to compression direction, preservation of relative path vertex positions on the mesh, and wrinkle shape preservation:

$$E = \alpha \sum_{i=1}^{n-1} ((p_{i+1} - p_i) \cdot e_i)^2 + \sum_{i=1}^{n} \|t_i - \bar{t}_i\|^2 +$$

$$\sum_{i=0}^{n} \|p_i - (p_{i-1} + p_{i+1})/2 - (\bar{p}_i - (\bar{p}_{i-1} + \bar{p}_{i+1})/2)\|^2 \tag{9}$$

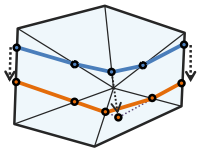
$$p_{-1} \equiv p_1,\ p_{n+1} \equiv p_{n-1} \text{at endpoints}$$

where $\bar{p}_i$ and $\bar{t}_i$ encode the absolute and relative positions of the wrinkle control point from the previous frame, and e_i is the direction of maximal compression in the triangle shared by p_i and p_{i+1}. The optimization is over just the small number of t_i variables, as we represent p_i as a function of t_i. Since our computation is dominated by persistence we use a relatively small $\alpha = 0.4$. Persistence requires both position and shape preservation: preserving shape alone allows wrinkles to slide uncontrollably, while preserving positions alone leads to artifacts when the underlying mesh triangles undergo significant deformation.



While the solve constrains the vertices to lie on the straight line defined by their currently associated edges, we avoid explicitly constraining t_i to the $[0, 1]$ interval as we want to allow wrinkle paths to slide along the mesh. If and when a computed t_i lands outside the $[0, 1]$ interval, we locate the best position for the vertex on the mesh as follows, and then update the the polyline accordingly. To compute the vertex position we parameterize the umbrella around the relevant edge endpoint (v_i^1 if $t_i < 0$ and v_i^2 if $t_i > 1$) and place p_i using the continuation of the edge projection using Equation 8 but constraining it to

the umbrella triangles. We then compute the geodesic paths from the previous/next vertices (v_{i-1}, v_{i+1}) to the new location and use the path's intersections with the mesh edges as new wrinkle path vertices (see inset).

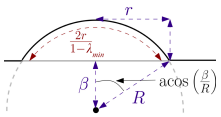


Lastly, if and when more than one wrinkle path vertex lies in the immediate vicinity of a single mesh vertex ($t_i < 0.1$ or $t_i > 0.9$) we use only one of these vertices in the optimization above and then use mesh geodesics to update the new set of intersection (see inset). Without this modification, the locations of the vertices become over constrained as a movement in their individually preferred directions would lead to path self-intersection.

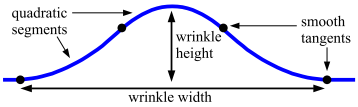
Length Update As a cloth garment deforms, wrinkles can grow and shrink in length. To replicate this process for each frame in the animation, we extend or trim all existing wrinkles based on the magnitude of the underlying stretch field, using the same compression threshold τ. To maintain temporal continuity, we do not change the length of a wrinkle by more than 15% per frame.

6 Wrinkle Shape Parameters

To generate actual wrinkle geometry from the traced wrinkle paths, we need to assign a target wrinkle height and width to each point along a path and generate appropriate cross-sectional geometry across it. We follow Rohmer et al. [2010b] in our computation of wrinkle height and width, using a formula that accounts for the material properties and the amount of compression along the wrinkle. Intuitively, the more flexible the material the higher the wrinkles are expected to be, and the higher the amount of compression the larger the ratio between the cross-section length and the cross-section width should be. Given a user-specified minimal wrinkle radius R_{min} the radius of a wrinkle as a function of compression magnitude is set to $(1-2/\pi)/\tilde{\lambda}_{min}R_{min}$. The wrinkle local width and height are then expressed via the radius as described by Rohmer et al. [2010b].



Our framework differs from Rohmer et al.'s, in that the compression field we operate on is piece-wise constant instead of piece-wise linear. Using pointwise magnitudes on this input results in discontinuous wrinkle dimensions. Instead we use the maximal compression along a wrinkle path to obtain the maximal wrinkle perimeter and height, and the corresponding width (see inset; here the width is $2r$ and the perimeter is $2r/(1-\lambda_{min})$). We then use these values as wrinkle parameters at its mid-point. At the wrinkle end points we set the width to be equal to the perimeter, and smoothly interpolate the width along the rest of the wrinkle path. We then use these width values to compute pointwise wrinkle height. Note that when the width is equal to the perimeter (at the end points) the height is, by construction, zero. This computation leads to smooth naturally dissolving wrinkle shapes along each path.



While the use of circular arcs to represent wrinkles is well suited for wrinkle height and width estimation, real-life wrinkle profiles, or cross-sections, deviate from this shape and smoothly blend with the surrounding surface. To achieve this effect we define wrinkle cross-sections using a quadratic B-spline (see inset). We also smooth out the path of each wrinkle, replacing the linear segments within each mesh triangle with Bezier

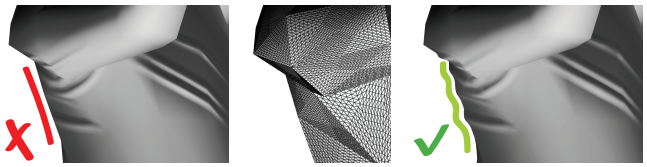


Figure 10: *Left to right: wrinkles rendered using only normal maps; tessellated wrinkle region; wrinkles rendered using displacement (tesselation) and normal maps.*

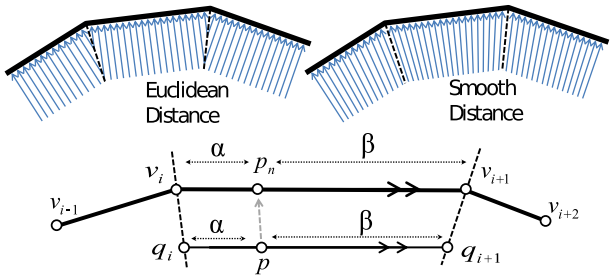


Figure 11: *Projection of a point p to the winkle segment v_iv_{i+1}.*

paths whose tangents across triangle edges are set to the average of the line segment tangents in the adjacent triangles. For efficiency this computation is done on the GPU in parallel per triangle, and each Bezier segment is discretized using a polyline.

7 Fast GPU-Based Wrinkle Modeling

Embedding fine wrinkles in the actual garment mesh requires a very high mesh resolution. Storing and rendering such a mesh would significantly slow the animation display and gameplay. We avoid modifying the underlying surface mesh or animating a finer mesh in parallel by modeling wrinkles, at render-time, directly on the GPU; specifically, we use the OpenGL tessellation shader to create coarse wrinkle geometry in real time and use the fragment shader to compute per-pixel wrinkle normal maps to increase rendered wrinkle believability. The combined method creates realistic looking wrinkles and is very efficient, allowing on-the-fly wrinkle modeling at 60 frames per second, when the rendered characters occupy the majority of the screen on a standard computer monitor. While using normal maps alone is clearly even faster, the results do not appear as realistic as the rendered frames lack inter-wrinkle occlusions and characteristic changes in character contours (Figure 10). Our approach which leaves the original mesh unchanged allows for wrinkle rendering to be enabled or disabled as the character moves further away from the viewer.

Wrinkle Geometry We use the tessellation shader to subdivide each mesh triangle that falls within the radius of influence of a wrinkle path (Figure 10, b). We upload a list of wrinkles affecting each triangle in each frame, and a triangle is tessellated only if it has a non-empty wrinkle list. We use a uniform tessellation scheme, and a target edge length that is half of r_{min}. We then offset each vertex in the direction of the triangle normal according to the cross-section height at that point. To compute the height at a point p we project that point on to the wrinkle path, evaluate the wrinkle height and width at that point, then use the B-spline shape function to determine the offset at p.

Projecting to the wrinkle path using the Euclidean closest-point is a discontinuous operation in the vicinity of the medial axis of the path (Figure 11 top), leading to discontinuities in the offsets. We avoid such discontinuities by using a modified projection (Figure 11). To find the projection of a point p onto a wrinkle path, we compute an

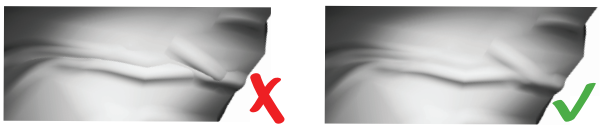


Figure 12: *Result without (left) and (with) wrinkle blending.*

approximate geodesic Voronoi diagram of the path edges. Within each Voronoi cell, p is projected parallel to the wrinkle edge $v_i v_{i+1}$ onto the Voronoi facets to find points q_i and q_{i+1}. We find the barycentric coordinates α and β of p with respect to these projected points, and then construct the final projection p_n on to the wrinkle path using the same barycentric coordinates along the wrinkle edge.

Wrinkle Normals In the fragment shader, we use the method above to obtain the projected path points and corresponding widths on adjacent wrinkle paths, and use the B-spline formula to obtain an analytic normal. We shade the wrinkles using Phong shading, using this computed normal in the fragment shader and directly evaluating the lighting equation per-pixel. Other shading methods, such as Minneart shading and physical based rendering models, could be computed on the same normal data.

Blending One of the main challenges when modeling wrinkles is to believably handle cases where wrinkles come close by or overlap. We found that the following simple heuristic generates visibly believable results while avoiding time consuming explicit geometry blending (Figure 12). For each tessellation vertex which is within the radius of influence of multiple wrinkle paths, we compute the offset independently for each path and then use the maximum of the offsets as the new offset. For each pixel which is within the radius of influence of multiple wrinkle paths, we compute both offsets and normals independently per path. We generate the final normal by averaging these per-path normals using the computed offsets as weights.

8 Results

We demonstrate our method on a range of meshes and animation sequences, most of which are taken from currently shipping video game titles. Our method plausibly adds believable, temporally coherent wrinkles to low-resolution character meshes and geometry (Fig. 15 and elsewhere in the paper.). The skirt example (Fig. 15) was provided courtesy of Kavan et al. [2011] and is created using coarse physics-based simulation.

Parameters The two tunable parameters in our system are compression sensitivity τ and minimal wrinkle width R_{min}. Both are linked to the expected material properties – how easily the modeled fabric bends and how wide the wrinkles are expected to be. Compression sensitivity may also depend on the quality of the animation – one may choose to use a higher sensitivity threshold if the animation is more noisy and exhibits more spurious tangential motion. Figure 13 shows the impact of changing the parameter values. Table 1 list the numbers used for the animation sequences shown in the paper.

Performance Table 1 shows the run-time performance of our algorithm on various meshes, as well as the mesh triangle counts and wrinkle parameters used. All times are computed on an Intel Core i7-3930K CPU running at 3.2 GHz, with 32 GB of RAM and an NVIDIA GeForce GTX 670 graphics card. Frames were rendered at a resolution of 1920x1080.

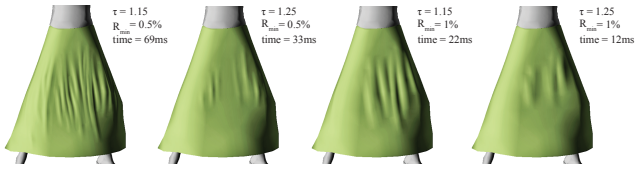


Figure 13: *Impact of different choices of parameter values τ and R_{min}.*

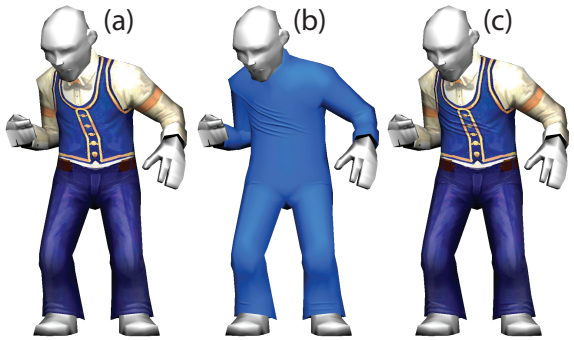


Figure 14: *Left: Artist drawn static texture and our wrinkles without (center) and with same texture (right).*

Figure 13 reports the impact of the choice of parameters on performance. As expected, performance decreases as the number of wrinkles increases (τ decreases) and their width increase; however, we maintain interactive performance throughout. The majority of our time per-frame is spent in the fragment shader, computing wrinkle normals in real time; in a real world scenario such as a first-person action game where models occupy smaller portions of the screen, performance increases accordingly.

Comparison to Alternative Strategies We validate our algorithmic choices against those used by previous work throughout the paper. Figure 4, (top) highlights the infeasibility of employing one of the frames in an animation sequence as a reference shape. Methods such as [Rohmer et al. 2010b; Müller and Chentanez 2010] heavily rely on the existence of such reference shape. Figure 5 motivates our use of a path tracing strategy designed to operate on piecewise-constant instead of smoothed piecewise linear stretch tensor fields [Rohmer et al. 2010b]. Our combined method is dramatically faster than that of Rohmer et al. who report speeds of over a second per frame.

We also compare our method against the use of artist drawn wrinkles (Figure 14). We generate temporal motion-driven wrinkles, such as those on the chest, where static wrinkles would be meaningless, and place wrinkles at many concave joints (elbow, ankle, pelvis) where artist placed similarly shaped wrinkles. Since our method is motion based, if part of the anatomy remains fixed we will not generate wrinkles in that area; thus our method lacks wrinkles under the arms where the artist chose to place some.

9 Conclusion

We present a practical method for adding real-time wrinkling to cloth surfaces. Our method operates in real-time, on arbitrary cloth meshes and animations, and is suitable for interactive wrinkling of cloth and fabrics for video games on current-generation and next-generation consoles and hardware.

Future work involves improvements to both performance and rendering quality, as well as applying the technique on other wrinkling surfaces, e.g. human skin. Our method has a few limitations that

Figure 15: *Some wrinkling results: input frames on top, wrinkled below.*

Animation	τ	R_{min}	# tri.	avg # wrinkles	avg. ref. shape computation time	avg. path tracing time	avg. modeling time	total frame time
Fig. 15 (a)	1.4	0.5%	734	22.75	1.69ms	5.33ms	8.30ms	12.45ms
Fig. 15 (b)	1.3	0.5%	602	5.60	1.69ms	3.92ms	5.53ms	11.69ms
Fig. 15 (c)	1.4	0.5%	534	5.89	1.74ms	2.46ms	4.68ms	7.39ms
Fig. 15 (d)	1.4	0.5%	628	11.80	1.57ms	3.77ms	6.55ms	10.67ms
Fig. 15 (e)	1.2	1%	356	10.57	0.92ms	4.84ms	8.95ms	13.19ms
Fig. 15 (f)	1.4	0.5%	602	19.03	1.72ms	5.69ms	9.78ms	14.66ms

Table 1: *Performance statistics for various models processed with our algorithm. R_{min} computed as percent of character height. All times in milliseconds.*

merit further research. Our work addresses dynamic wrinkles - i.e. those whose presence is hinted at by the cloth motion, we do not address static wrinkles which humans often expect to be present in areas of negative Gaussian curvature. Our method, like most previous work, generates outward bulging wrinkles - well suited for tight garments. Future work should explore automatic selection of wrinkle direction consistent with human expectations. Additionally, our method does not currently handle stretch wrinkles, and incorporating this wrinkling is a source for future investigation.

Acknowledgements. This work was supported by NSERC and GRAND NCE. The skirt model was provided courtesy of Kavan et al. [2011]. Other models shown in the paper are taken from the video game Clockwork Empires, and are used with permission of Gaslamp Games.

References

BARAFF, D., AND WITKIN, A. 1998. Large steps in cloth simulation. In *Proc. Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, 43–54.

BERGOU, M., MATHUR, S., WARDETZKY, M., AND GRINSPUN, E. 2007. TRACKS: Toward Directable Thin Shells. *ACM Trans. Graph. (SIGGRAPH) 26*, 3, 50:1–50:10.

BONET, J. 1997. *Nonlinear continuum mechanics for finite element analysis*. Cambridge university press.

BOYKOV, Y., AND KOLMOGOROV, V. 2004. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 1124–1137.

BOYKOV, Y., VEKSLER, O., AND ZABIH, R. 2001. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence 23*, 1222–1239.

CHOI, K.-J., AND KO, H.-S. 2002. Stable but responsive cloth. *ACM Trans. Graph. 21*, 3, 604–611.

CHOI, K.-J., AND KO, H.-S. 2005. Research problems in clothing simulation. *Computer Aided Design 37*, 6, 585–592.

ENGLISH, E., AND BRIDSON, R. 2008. Animating developable surfaces using nonconforming elements. *ACM Trans. Graph. 27*, 3, 66:1–66:5.

ENQVIST, H. 2010. The secrets of cloth simulation in Alan Wake. *Gamasutra* (Apr.).

GUAN, P., REISS, L., HIRSHBERG, D., WEISS, A., AND BLACK, M. J. 2012. Drape: Dressing any person. *ACM Trans. on Graphics (Proc. SIGGRAPH) 31*, 4 (July), 35:1–35:10.

HADAP, S., BANGERTER, E., VOLINO, P., AND MAGNENAT-THALMANN, N. 1999. Animating wrinkles on clothes. In *Proceedings of the Conference on Visualization '99: Celebrating Ten Years*, IEEE Computer Society Press, VIS '99, 175–182.

HAHN, F., THOMASZEWSKI, B., COROS, S., SUMNER, R. W., COLE, F., MEYER, M., DEROSE, T., AND GROSS, M. 2014. Subspace clothing simulation using adaptive bases. *ACM Trans. Graph. 33*, 4, 105:1–105:9.

HU, J. 2004. *Structure and Mechanics of Woven Fabrics*. Woodhead Publishing Series in Textiles. Elsevier Science.

KAVAN, L., GERSZEWSKI, D., BARGTEIL, A., AND SLOAN, P.-P. 2011. Physics-inspired upsampling for cloth simulation in games. *ACM Trans. Graph. (SIGGRAPH) 30*, 4, 93:1–93:9.

KIM, D., KOH, W., NARAIN, R., FATAHALIAN, K., TREUILLE, A., AND O'BRIEN, J. F. 2013. Near-exhaustive precomputation of secondary cloth effects. *ACM Transactions on Graphics 32*, 4, 87:1–7. Proc. SIGGRAPH.

KOLMOGOROV, V., AND ZABIH, R. 2004. What energy functions can be minimized via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 65–81.

LANDER, J. 1999. Devil in the blue-faceted dress: Real-time cloth animation. *Game Developer Magazine* (May).

MÜLLER, M., AND CHENTANEZ, N. 2010. Wrinkle meshes. In *Proc. Symposium Computer Animation*, SCA '10, 85–92.

OAT, C. 2007. Animated wrinkle maps. In *SIGGRAPH Courses*, SIGGRAPH, 33–37.

RÉMILLARD, O., AND KRY, P. G. 2013. Embedded thin shells for wrinkle simulation. *ACM Trans. Graph. 32*, 4, 50:1–50:8.

ROHMER, D., HAHMANN, S., AND CANI, M.-P. 2010. Active geometry for game characters. In *Motion in Games*, vol. 6459 of *Lecture Notes in Computer Science*. 170–181.

ROHMER, D., POPA, T., CANI, M.-P., HAHMANN, S., AND SHEFFER, A. 2010. Animation Wrinkling: Augmenting Coarse Cloth Simulations with Realistic-Looking Wrinkles. *ACM Trans. Graph. (Proc. SIGGRAPH ASIA) 29*, 5.

THOMASZEWSKI, B., PABST, S., AND STRAER, W. 2009. Continuum-based strain limiting. *Computer Graphics Forum 28*, 2, 569–576.

VERLET, L. 1967. Computer experiments on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Physical Review 159*, 1, 98.

WANG, H., HECHT, F., RAMAMOORTHI, R., AND O'BRIEN, J. F. 2010. Example-based wrinkle synthesis for clothing animation. *ACM Trans. Graph 29*, 4, 107:1–8.

ZURDO, J. S., BRITO, J. P., AND OTADUY, M. A. 2013. Animating wrinkles by example on non-skinned cloth. *IEEE Transactions on Visualization and Computer Graphics 19*, 1, 149–158.