Abstract

Regularization is used to avoid overfitting when training a neural network; unfortunately, this reduces the attainable level of detail hindering the ability to capture high-frequency information present in the training data. Even though various approaches may be used to re-introduce high-frequency detail, it typically does not match the training data and is often not time coherent. In the case of network inferred cloth, these sentiments manifest themselves via either a lack of detailed wrinkles or unnaturally appearing and/or time incoherent surrogate wrinkles. Thus, we propose a general strategy whereby high-frequency information is procedurally embedded into low-frequency data so that when the latter is smeared out by the network the former still retains its high-frequency detail. We illustrate this approach by learning texture coordinates which when smeared do not in turn smear out the high-frequency detail in the texture itself but merely smoothly distort it. Notably, we prescribe perturbed texture coordinates that are subsequently used to correct the over-smoothed appearance of inferred cloth, and correcting the appearance from multiple camera views naturally recovers lost geometric information.

1. Introduction

Since neural networks are trained to generalize to unseen data, regularization is important for reducing overfitting, see e.g. [26, 61]. However, regularization also removes some of the high variance characteristic of much of the physical world. Even though high-quality ground truth data can be collected or generated to reflect the desired complexity of the outputs, regularization will inevitably smooth network predictions. Rather than attempting to directly infer high-frequency features, we alternatively propose to learn a low-frequency space in which such features can be embedded.

We focus on the specific task of adding high-frequency wrinkles to virtual clothing, noting that the idea of learning a low-frequency embedding may be generalized to other tasks. Because cloth wrinkles/folds are high-frequency features, existing deep neural networks (DNNs) trained to infer cloth shape tend to predict overly smooth meshes [1, 16, 27, 28, 35, 40, 48, 59, 68]. Rather than attempting to amend such errors directly, we perturb texture so that the rendered cloth mesh appears to more closely match the ground truth. See Figure 1. Then given texture perturbations from at least two unique camera views, 3D geometry can be accurately reconstructed [32] to recover high-frequency wrinkles. Similarly, for AR/VR applications, correcting visual appearance from two views (one for each eye) is enough to allow the viewer to accurately discern 3D geometry.

Our proposed texture coordinate perturbations are highly dependent on the camera view. Thus, we demonstrate that one can train a separate texture sliding neural network (TSNN) for each of a finite number of cameras laid out into an array and use nearby networks to interpolate results valid for any view enveloped by the array. Although an approach similar in spirit might be pursued for various lighting conditions, this limitation is left as future work.
since there are a great deal of applications where the light is ambient/diffuse/non-directional/etc. In such situations, this further complication may be ignored without significant repercussion.

2. Related Work

Cloth: While physically-based cloth simulation has matured as a field over the last few decades [7, 8, 12, 13, 63], data-driven methods are attractive for many applications. There is a rich body of work in reconstructing cloth from multiple views or 3D scans, see e.g. [11, 22, 66]. More recently, optimization-based methods have been used to generate higher resolution reconstructions [33, 53, 69, 71]. Some of the most interesting work focuses on reconstructing the body and cloth separately [6, 49, 72, 74].

Cloth and Learning: With advances in deep learning, one can aim to reconstruct 3D cloth meshes from single views. A number of approaches reconstruct a joint cloth/body mesh from a single RGB image [1, 4, 48, 57], RGB-D image [73], or video [2, 3, 30, 70]. To reduce the dimensionality of the output space, DNNs are often trained to predict the pose/shape parameters of human body models such as SCAPE [5] or SMPL [41] (see also [52]). [1, 2, 3] predict SMPL model parameters along with per-vertex offsets to add details, and [4] refines the mesh using the network proposed in [34]. [30, 48, 65] leverage predicted pose information to infer shape. Estimating shape from silhouettes given an RGB image has also been explored [18, 19, 48]. When only the garment shape is predicted, a number of recent works output predictions in UV space to represent geometric information as pixels [16, 35, 40], although others [28, 59] define loss functions directly in terms of the 3D cloth vertices.

Wrinkles and Folds: Cloth realism can be improved by introducing wrinkles and folds. In the graphics community, researchers have explored both procedural and data-driven methods for generating wrinkles [17, 27, 31, 47, 56, 67]. Other works add real-world wrinkles as a postprocessing step to improve smooth captured cloth: [54] extracts the edges of cloth folds and then applies space-time deformations, [55] solves for shape deformations directly by optimizing over all frames of a video sequence. Recently, [40] used a conditional Generative Adversarial Network [45] to generate normal maps as proxies for wrinkles on captured cloth.

Geometry: More broadly, deep learning on 3D meshes falls under the umbrella of geometric deep learning, which was coined by [14] to characterize learning in non-Euclidean domains. [60] was one of the earliest works in this area and introduced the notion of a Graph Neural Network (GNN) in relation to CNNs. Subsequent works similarly extend the CNN architecture to graphs and manifolds [9, 42, 44, 46]. [39] introduces a latent representation that explicitly incorporates the Dirac operator to detect principal curvature directions. [64] trains a mesh generative model to generate novel meshes outside an original dataset. Returning to the specific application of virtual cloth, [35] embeds a non-Euclidean cloth mesh into a Euclidean pixel space, making it possible to directly use CNNs to make non-Euclidean predictions.

Texture: In the computer graphics community, textures have historically been used to capture both geometric and material details lost by using simplified models [21, 43], which is similar in spirit to our approach. Though, to the best of our knowledge, we are the first to propose learning texture coordinate perturbations to facilitate the accurate reconstruction of lost geometric details. For completeness, we briefly note a few works that use learning for texture synthesis and/or style transfer [20, 23, 24, 29, 36, 58].

3. Methods

We define texture sliding as the changing of texture coordinates on a per-camera basis such that any point which is visible from some stereo pair of cameras can be triangulated back to its ground truth position. Other stereo reconstruction techniques can also be used in place of triangulation because the images we generate are consistent with the ground truth geometry. See e.g. [10, 32, 62].

3.1. Per-Vertex Discretization

Since the cloth mesh is discretized into vertices and triangles, we take a per-vertex, not a per-point, approach to texture sliding. Our proposed method (see Section 4.1) computes per-vertex texture coordinates on the inferred cloth that match those of the ground truth as seen by the camera under consideration. Then during 3D reconstruction, barycentric interpolation is used to find the subtriangle locations of the texture coordinates corresponding to ground truth cloth vertices. This assumes linearity, which is only valid when the triangles are small enough to capture the inherent nonlinearities in a piecewise linear sense; moreover, folds and wrinkles can create significant nonlinearity. See Figure 2.

3.2. Occlusion Boundaries

Accurate 3D reconstruction requires that a vertex of the ground truth mesh be visible from at least two cameras and that camera projections of the vertex to the inferred cloth exist and are valid. However, occlusions can derail these assumptions.

First, consider things from the standpoint of the inferred cloth. For a given camera view, some inferred cloth triangles will not contain any visible pixels, and we denote a vertex as occluded when none of its incident triangles contain any visible pixels. Although we do not assign perturbed texture coordinates to occluded vertices (i.e. they keep their
original texture coordinates, or a perturbation of zero), we do aim to keep the texture coordinate perturbation function smooth (see Section 4.2). In addition, there will be so called non-occluded vertices in the inferred cloth that do not project to visible pixels of the ground truth cloth. This often occurs near silhouette boundaries where the inferred cloth silhouette is sometimes wider than the ground truth cloth silhouette. These vertices are also treated as occluded, similar to those around the back side of the cloth behind the silhouette, essentially treating some extra vertices near occlusion boundaries as also being occluded. See Figure 3a.

Next, consider things from the standpoint of the ground truth cloth. For example, consider the case where all the cameras are in the front, and vertices on the back side of the ground truth cloth are not visible from any camera. The best one can do in reconstructing these occluded vertices is to use the inferred cloth vertex positions; however, care should be taken near occlusion boundaries to smoothly taper between our texture sliding 3D reconstruction and the inferred cloth prediction. A simple approach is to extrapolate/smooth the geometric difference between our texture sliding 3D reconstruction and the inferred cloth prediction to occluded regions of the mesh. Once again, the definition of occluded vertices needs to be broadened for silhouette consideration. Not only will vertices not visible from at least two cameras have to be considered occluded, but vertices that don’t project to the interior of an inferred cloth triangle with valid texture coordinate perturbations will also have to be considered occluded. See Figure 3b.

4. Dataset Generation

Let \( C = \{X, T\} \) be a cloth triangulated surface with \( n \) vertices \( X \in \mathbb{R}^{3n} \) and texture coordinates \( T \in \mathbb{R}^{2n} \). We assume that mesh connectivity remains fixed throughout. The ground truth cloth mesh \( C_G(\theta) = \{X_G(\theta), T_G\} \) depends on the pose \( \theta \). Given a pre-trained DNN (we use the network from [35]), the inferred cloth \( C_N(\theta) = \{X_N(\theta), T_N\} \) is also a function of the pose \( \theta \). Our objective is to replace the ground truth texture coordinates \( T_G \) with perturbed texture coordinates \( T_N(\theta, v) \), i.e. to compute \( C_N'(\theta, v) = \{X_N(\theta), T_N(\theta, v)\} \) where \( T_N(\theta, v) \) depends on both the pose \( \theta \) and the view \( v \). Even though \( T_N(\theta, v) \) is in principle valid for all \( v \) using interpolation (see Section 6.3), training data \( T_N(\theta, v_p) \) is only required for a finite number of camera views \( v_p \). For each camera \( p \), we also only require training data for finite number of poses \( \theta_k \), i.e. we require \( T_N(\theta_k, v_p) \), which is computed from \( T_G \) using \( X_G(\theta_k), X_N(\theta_k) \), and \( v_p \).

4.1. Texture Coordinate Projection

We project texture coordinates to the inferred cloth vertices \( X_N(\theta_k) \) from the ground truth cloth mesh \( C_G(\theta_k) \) using ray intersection. For each inferred cloth vertex in \( X_N(\theta_k) \), we cast a ray from camera \( p \)’s aperture through the vertex and find the first intersection with the ground truth mesh \( C_G(\theta_k) \); subsequently, \( T_G \) is barycentrically interpolated to the point of intersection and assigned to the inferred cloth vertex as its \( T_N(\theta_k, v_p) \) value. See Figure 4. Rays are only cast for inferred cloth vertices that have at least one incident triangle with a nonzero area subregion visible to camera \( p \). Also, a ground truth texture coordinate value is
only assigned to an inferred cloth vertex when the point of intersection with the ground truth mesh is visible to camera \( p \). We store and learn texture coordinate displacements \( d_{v_p}(\theta_k) = T_N(\theta_k, v_p) - T_G \). After this procedure, any remaining vertices of the inferred cloth that have not been assigned \( d_{v_p}(\theta_k) \) values are treated as occluded and handled via smoothness considerations as discussed in Section 4.2.

![Image](image.png)

Figure 4: Illustration of the ray intersection method for transferring texture coordinates to the inferred cloth from the ground truth cloth. Texture coordinates for the inferred cloth vertex (red cross) are interpolated from the ground truth mesh to the point of ray intersection (red circle).

### 4.2. Occlusion Handling

Some vertices of the inferred cloth mesh remain unassigned with \( d_{v_p}(\theta_k) = 0 \) after executing the algorithm outlined in Section 4.1. This creates a discontinuity in \( d_{v_p}(\theta_k) \) which excites high frequencies that require a more complex network architecture to capture. In order to alleviate demands on the network, we smooth \( d_{v_p}(\theta_k) \) as follows. First, we use the Fast Marching Method on triangulated surfaces \([37]\) to generate a signed distance field. Then, we extrapolate \( d_{v_p}(\theta_k) \) normal to the distance field into the unassigned region, see \( e.g. \ [50]\). Finally, a bit of averaging is used to provide smoothness, while keeping the assigned values of \( d_{v_p}(\theta_k) \) unchanged. Alternatively, one could solve a Poisson equation as in \([15]\) while using the assigned \( d_{v_p}(\theta_k) \) as Dirichlet boundary conditions.

### 5. Network Architecture

A separate texture sliding neural network (TSNN) is trained for each camera \( p \); thus, we drop the \( v_p \) notation in this section. The loss is defined over all poses \( \theta_k \) in the training set

\[
\mathcal{L} = \sum_{\theta_k} \| d(\theta_k) - \hat{d}(\theta_k) \|_2
\]  

(1)

to minimize the difference between the desired displacements \( d(\theta_k) \) and predicted displacements \( \hat{d}(\theta_k) \). The inferred cloth data we chose to correct are predictions of the T-shirt meshes from \([35]\), each of which contains about 3,000 vertices. The dataset spans about 10,000 different poses generated from a scanned garment using physically-based simulation. To improve the resolution, we up-sampled each cloth mesh by subdividing each triangle into four subtriangles. Notably, our texture sliding approach can be used to augment the results of any dataset for which ground truth and inferred training examples are available. Moreover, it is trivial to increase the resolution of any such dataset simply by subdividing triangles. Note that perturbations of the subdivided geometry are unnecessary, as we merely desire more sample points (to address Figure 2). Finally, we applied an 80-10-10 training-validation-test set split.

Similar to \([35]\), the displacements \( d(\theta_k) \) are stored as pixel-based cloth images for the front and back sides of the T-shirt, though we still output per-vertex texture coordinate displacements in UV space. See Figure 5 for an overview of the network architecture. Given input joint transformation matrices of shape \( 1 \times 1 \times 90 \), TSNN applies a series of transpose convolution, batch normalization, and ReLU activation layers to upsample the input to \( 512 \times 512 \times 4 \). The first two dimensions of the output tensor represent the predicted displacements for the front side of the T-shirt, and the remaining two dimensions represent those for the back side.

![Image](image.png)

Figure 5: Texture sliding neural network (TSNN) architecture.

### 6. Experiments

In Section 6.1, we quantify the data generation approach of Section 4 and highlight the advantages of mesh subdivision for up-sampling. In Section 6.2, we evaluate the predictions made by our trained texture sliding neural network
(TSNN). In Section 6.3, we demonstrate the interpolation of texture sliding results to novel views between a finite number of cameras. Finally, in Section 6.4, we use multi-view texture sliding to reconstruct 3D geometry.

6.1. Dataset Generation and Evaluation

We aim to have the material coordinates of the cloth be in the correct locations as viewed by multiple cameras, so that the material can be accurately 3D reconstructed with point-wise accuracy. As such, our error metric is a bit more stringent than that commonly used because our aim is to reproduce the actual material behavior, not merely to mimic its look (e.g., by perturbing normal vectors to create shading consistent with wrinkles in spite of the cloth being smooth, as in [40]). In order to elucidate this, consider a two-step approach where one first approximates a smooth cloth mesh and then perturbs that mesh to add wrinkles (similar to [59]). In order to preserve area and achieve the correct material behavior, material in the vicinity of a newly forming wrinkle should slide laterally towards that wrinkle as it is formed. Merely non-physically stretching the material in order to create a wrinkle may look plausible, but does not admit the correct material behavior. In fact, the texture would be unrealistically stretched as well, although this is less apparent visually when using simple textures.

Since texture coordinates provide a proxy surface parameterization for material coordinates, we measure texture coordinate errors in a per-pixel fashion comparing between the ground truth and inferred cloth at the center of each pixel. Figure 6a shows results typical for cloth inferred using the network from [35], and Figure 6b shows the highly improved results obtained on the same inferred geometry using our texture sliding approach (with 1 level of subdivision). Note that the vast majority of the errors in Figure 6b occur near the wrinkles where the nonlinearities illustrated in Figure 2 are most prevalent. In Figure 6c, we deform the vertices of the inferred cloth mesh so that they lie exactly on the ground truth mesh in order to mimic a two-step approach (as discussed above). Note how our error metric captures the still rather large errors in the material coordinates (and thus cloth vertex positions) in spite of the mesh in Figure 6c appearing to have the same wrinkles and folds as the ground truth mesh. Figure 7 compares the local compression and extension energies for a sample pose, comparing the ground truth cloth (a), the inferred cloth (b), and the result of a two-step process (c). In spite of the cloth mesh in (c) bearing visual resemblance to the ground truth in (a), it still has quite erroneous deformation energies.

The texture sliding result shown in Figure 8b better represents the ground truth cloth. Figures 8c and 8d show how subdividing the inferred cloth mesh one and two times (respectively) progressively alleviates errors emanating from the linearity assumption illustrated in Figure 2. Table 1 shows quantitative results comparing the inferred cloth to texture sliding with and without subdivision.

<table>
<thead>
<tr>
<th>Method</th>
<th>SqrtMSE ($\times 10^{-3}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jin et al. [35]</td>
<td>24.496 ± 6.9536</td>
</tr>
<tr>
<td>TS</td>
<td>5.2662 ± 2.2320</td>
</tr>
<tr>
<td>TS + subdivision</td>
<td>3.5645 ± 1.6617</td>
</tr>
</tbody>
</table>

Table 1: Per-pixel square root of mean squared error (SqrtMSE) for the entire dataset.
6.2. Network Training and Inference

The network was trained using the Adam optimizer [38] with a $10^{-3}$ learning rate in PyTorch [51]. As mentioned earlier, we subdivided the mesh triangles once. Figure 9 shows a typical prediction on a test set example, including the per-pixel errors in predicted texture coordinates. While the TSNN is able to recover the majority of the shirt, it struggles near wrinkles. Figure 10 highlights a particular wrinkle comparing the inferred cloth (Figure 10a) and the results of the TSNN before (Figure 10b) and after (Figure 10c) subdivision to the ground truth (Figure 10d). Table 2 shows quantitative results comparing the inferred cloth to TSNN results with and without subdivision.

![Image](a) (b) (c) (d) (e)

Figure 8: As the inferred cloth mesh (a) is subdivided, texture sliding (b-d) moves the appearance of the inferred mesh closer to the ground truth (e).

Table 2: Per-pixel SqrtMSE for the test set. Inspite of Table 1 demonstrating that subdivision improves the ground truth TS data, the improvements are not uniformly realized by the TSNN (which we discuss in the appendix).

<table>
<thead>
<tr>
<th>Network</th>
<th>SqrtMSE ($\times 10^{-4}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jin et al. [35]</td>
<td>24.871 ± 7.0613</td>
</tr>
<tr>
<td>TSNN</td>
<td>13.335 ± 4.2924</td>
</tr>
<tr>
<td>TSNN + subdivision</td>
<td>13.591 ± 4.5194</td>
</tr>
</tbody>
</table>

6.3. Interpolating to Novel Views

Given a finite number of camera views $v_p$, one can specify a new view enveloped by the array using a variety of interpolation methods. For the sake of demonstration, we take a simple approach assuming that one can interpolate via $v = \sum_p w_p v_p$, and then use these same weights to compute

$$T_N(\theta_k, v) = \sum_p w_p T_N(\theta_k, v_p)$$ (2)

This same equation is also used for $\hat{T}_N(\theta_k, v)$. Figure 11 shows the results obtained by linearly interpolating between two camera views. Note how the largest errors appear near areas occluded by wrinkles, where one (or both) of the cameras has no valid texture sliding results and instead uses the inferred cloth textures. This can be alleviated by using more cameras placed closer together. Figure 12 quantifies these results for the inferred cloth $C_N(\theta_k)$, texture sliding $C'_N(\theta_k, v)$, and the results of the TSNN $C''_N(\theta_k, v)$. In Figure 13, we repeat these comparisons, except using bilinear interpolation between four camera views.

6.4. 3D Reconstruction

In order to reconstruct the 3D position of a vertex of the ground truth mesh, we take the usual approach of finding rays that pass through that vertex and the camera aperture for a number of cameras. Then given at least two rays, one can triangulate a 3D point that is minimal distance from all the rays. We can do this without solving the typical image to image correspondence problem because we know the ground truth texture coordinates for any given vertex. Thus, we merely have to find the ray that passes through the camera aperture and the ground truth texture coordinate for the vertex under consideration.

To find a ground truth texture coordinate on a texture corrected inferred cloth mesh $C'_N(\theta_k, v)$, or $C''_N(\theta_k, v)$, we first find the triangle containing that texture coordinate. This can be done quickly by using a hierarchical bounding box structure where the base level boxes around each triangle are defined using the min/max texture coordinates at the three vertices. Then one can write the barycentric interpolation formula that interpolates the triangle vertex texture coordinates to obtain the given ground truth texture coordinate, and subsequently invert the matrix to solve for the weights. These
weights determine the sub-triangle position of the vertex under consideration (taking care to note that different answers are obtained in 3D space versus screen space, since the camera projection is nonlinear). Figure 14 shows the 3D reconstruction of a test set example using texture sliding (Figure 14c) and the TSNN (Figure 14d). Figure 15 compares the per-pixel errors and local compression/extension energies of Figures 14c and 14d.

7. Discussion and Future Work

There are many disparate applications for clothing including for example video games, AR/VR, Hollywood special effects, virtual try-on and shopping, scene acquisition and understanding, and even bullet proof vests and soft armor. Various scenarios define accuracy or fidelity in vastly different ways. So while it is typical to state that one cares about more than just the visual appearance (or “graphics”), often those aiming for predictive capability still make concessions. For example, wherein [59] proposes a network that well predicts wrinkles mapped to new body types, the discussion in [40] implies that the horizontal wrinkles predicted by [59] are more characteristic of inaccurate physical simulation than real-world behavior. Instead, [40] strives for more vertical wrinkles to better match their data, but they accomplish this by predicting lighting to match an image while accepting overly smooth geometry. And as we have shown in Figure 7c, predicting the correct geometry still allows for rather large errors in the deformation (see [25]).
Figure 14: Comparison of the ground truth cloth (a) and inferred cloth (b) to the 3D reconstructions obtained using texture sliding (c) and the TSNN (d). To remove reconstruction noise generated by network inference errors in (d), we used the postprocess from [25]; although, there are many other smoothing options in the literature that one might also consider.

In light of this, we state the problem of most interest to us: Our aim is to study the efficacy of using deep neural networks to aid in the modeling of material behavior, especially for those materials for which predictive methods do not currently exist because of various unknowns including friction, material parameters (for cloth and body), etc. Given this goal, we focus on the accurate prediction of material coordinates, which are a super set of deformation, geometry, lighting, visual plausibility, etc.

As demonstrated by the remarkably accurate 3D reconstruction in Figure 14c (see 15a), our approach to encoding high frequency wrinkles into lower frequency texture coordinates (i.e. texture sliding) works quite well. It can be used as a post-process to any existing neural network to capture lost details (as long as ground truth and inferred training examples are available); moreover, we showed that trivial subdivision could be used to increase the sampling resolution to limit linearization artifacts. The main drawback of our approach is that it relies on triangulation or multi-view stereo in order to construct the final 3D geometry, although this step is not required for AR/VR applications. This means that one needs to take care when training the texture sliding neural network (TSNN) since inference errors can cause reconstruction noise. Thus, as future work, we plan on experimenting with the network architecture, the size of the image used in the CNN, the smoothing methods near occlusion boundaries, the amount of subdivision, etc. In addition, it would be interesting to consider more savvy multiview 3D reconstruction methods (particularly ones that employ DNNs; then, one might train the whole process end-to-end). Our current solution to addressing multiview reconstruction noise is to simply use the method from [25] as a postprocess to the triangulation of the TSNN results. As can be seen in Figure 14d, this leads to a high quality reconstruction with many high frequency wrinkles faithful to the ground truth; however, an improved TSNN would lead to more accurate per-pixel texture coordinates than those in Figure 15b (top).

Acknowledgements

Research supported in part by ONR N00014-13-1-0346, ONR N00014-17-1-2174, and JD.com. We would like to thank Reza and Behzad at ONR for supporting our efforts into machine learning, as well as Rev Lebaredian and Michael Kass at NVIDIA for graciously loaning us a GeForce RTX 2080Ti to use for running experiments. We would also like to thank Matthew Cong and Yilin Zhu for...
their insightful discussions, and Congyue Deng for generating realistic cloth textures.

Appendix

A. Dataset Generation

A.1. Topological Considerations

There are some edge cases that require additional topological consideration. In particular, the collar, sleeves, and waist are areas where a ray cast to an inferred cloth vertex can intersect with a back-facing triangle on the inside of the ground truth shirt. We aim to define texture coordinates on inferred cloth vertices so that barycentric interpolation can be used to find the texture coordinates of a ground truth vertex for 3D reconstruction. However, mixing texture coordinates from the inside and outside of the shirt in a single triangle causes dramatic interpolation error. In fact, as shown in Figure 2, large errors may occur for any triangle that mixes texture coordinates from geodesically far-away regions. Thus, we omit such triangles from consideration by omitting a vertex from any edge that connects two geodesically far-away regions.

As a further improvement to our method, one can treat the inside and outside of the shirt as separate meshes, applying texture sliding twice and training two separate networks; moreover, one may take a patch-based approach, applying TS and training a TSNN for each (slightly overlapping) patch of the shirt.

A.2. Smoothness Considerations

When training a neural network, more predictable results are obtained when the inferred cloth vertex data is smoother. Thus, there exists tradeoffs between smoothness and accuracy when assigning texture coordinates. An edge that connects two geodesically far-away regions introduces a jump discontinuity in the texture coordinates leading to high frequencies in the ground truth data that place increased demands on the network. Although subdivision adds degrees of freedom along such edges to better sample the high frequency, it is often better to delete such edges entirely by removing one of the edge’s vertices. Recall that any vertex not assigned a ground truth texture coordinate is instead defined via smoothness considerations (see Section 4.2) reducing demands on the network.

B. 3D Reconstruction

There are a couple of issues with finding the texture coordinates of the ground truth vertices on an inferred cloth mesh whether it be TS or TSNN data. Firstly, there could be seams in the texture in which case smoothing would be needed near the seam as discussed above in order to avoid degrading the data. A patch-based approach can be used to alleviate any such seams. Secondly, seams, smoothing, and non-linearity along the lines of Figure 2 may all contribute to more than one inferred cloth triangle containing the texture coordinates of a ground truth vertex. This ambiguity can be treated similarly to how correspondence uncertainties are addressed in standard multi-view stereo algorithms. The straightforward approach is to consider each distinct possibility for each camera in all possible combinations and choose the set of rays that have the least disagreement for triangulation; furthermore, one may also consider the 3D reconstruction of neighboring vertices, material deformation, etc. Overall, reliance on multi-view stereo does require careful attention when utilizing our method. As such, we provide a few more examples of 3D reconstruction for examples from the test set in order to demonstrate the efficacy of our approach. See Figure 16.

Instead of applying a standard smoothing algorithm to the somewhat noisy results of the 3D reconstructions of the TSNN data, we used the postprocess from [25]. This choice was made because of our desire to use neural networks to branch the gap between physical simulations and real-world material behavior. In order to quantify the impact of the

Figure 16: Comparisons of the ground truth cloth (a) and inferred cloth (b) to the 3D reconstructions obtained using texture sliding (c) and the TSNN (d) for three test set examples. Note that the postprocess in [25] is only applied to (d).
postprocess from [25] on the final results, Figure 17 shows the results obtained when applying the postprocess directly to the inferred cloth as compared to applying it to TS and TSNN data.

C. Novel View Interpolation

Interpolating between two cameras, each with TS or TSNN data, has the effect of following a straight-line path. However, by choosing the camera array and subsequent interpolation carefully one can interpolate along curved paths. For example in Figure 18, one can interpolate between the 12 cameras (represented by blue dots) in order to follow the curved camera path.

D. Error Analysis (for Patches)

In this section, we consider each step of the ray intersection algorithm, carefully illustrating the sources of error. This is done for a single patch consisting of the entire front half of the shirt in order to ensure continuous and unique texture coordinates. Additionally, this section highlights our patch-based approach, noting that we would utilize this approach on a number of overlapping patches and blend the final results together. In fact, when considering only a single patch, we modify our nodes from the inferred cloth to only include that patch, ignoring the rest of the vertices and triangles in the mesh. Similarly, the ground truth cloth is assumed to only consider the data for that patch. Note that any existing network that predicts cloth vertex positions can be adapted to this patch-based approach as a postprocess applied to their training examples, and that one may readily apply the predicted texture separately to each patch.

Along the lines of Section 4.1 and Appendix A, a ray between the camera aperture and each inferred cloth vertex of the patch is intersected with the ground truth cloth, in order to find the ground truth texture coordinates to assign to the inferred cloth vertex. Recall that the inferred cloth vertex remains unassigned when occluded; however, we modify our definition of occlusion to only consider the inferred cloth patch under consideration. This allows, for example, one to reconstruct the back half of the shirt with cameras from the front, since the front half of the shirt would not be considered and not be occluded by the back half. Since we only consider the front half of both the inferred and ground truth cloth, one also does not compute ground truth texture coordinates to be assigned to the inferred vertex when the ray does not intersect the front half patch of the ground truth cloth. Separating the front and back of the shirt guarantees the inferred cloth patch is assigned texture coordinates from a continuous texture. This leads to a sub-mesh of assigned texture coordinates $T_N$. As usual, we remove any edge (by deleting an inferred vertex) which connects geodesically far away regions as indicated by differing texture coordinate values. See Figure 19.

D.1. Texture Coordinates

To quantify the worst case texture sliding scenarios, we first consider $T_N$ for every pose $\theta_k$ and camera view $v_p$ used in training. The edge with the largest difference in texture coordinates (as a proxy for geodesic distances) is shown in Figure 20. We do the same for Euclidean distances along every edge to connected vertices in Figure 21.

D.2. Texture Coordinate Displacements

In order to fill in unassigned vertices for the patch under consideration, we show the most extreme behavior of texture sliding over all $(\theta_k, v_p)$. First, we compute the maximal value of $||d_{v_p}(\theta_k)||$ among all $(\theta_k, v_p)$ pairs, i.e. where
maximal texture sliding occurs in our training set. See Figure 19b. We also compute $\Delta d_{v_p}(\theta_k)$ along each assigned edge in order to ascertain the biggest jump (indicating high frequency) that would be seen by the TSNN. The edge with the maximal $||\Delta d_{v_p}(\theta_k)||$ over all $(\theta_k, v_p)$ pairs is shown in Figure 22. Note that one may obtain better smoothness when training the TSNN by not assigning vertices where $d_{v_p}(\theta_k)$ is too large or one of the vertices of an edge where $\Delta d_{v_p}(\theta_k)$ is too large.

D.3. Smoothness Considerations

As long as extrapolation is done smoothly to assign texture coordinates to the remaining vertices, there should be no new extrema in $\Delta T_N(\theta_k, v_p)$ and $\Delta d_{v_p}(\theta_k)$. After applying smoothing, we verify that the largest $\Delta T_N(\theta_k, v_p)$ and $\Delta d_{v_p}(\theta_k)$ are the same as before.

E. TSNN – Additional Experiments

Table 3 shows additional TSNN results after applying a displacement threshold to the TS dataset. In addition, in Table 4 we decompose the TSNN errors based on whether vertices were assigned via our ray intersection method or extrapolation. Results indicate that training separate networks for smooth and wrinkled regions of the cloth may be a promising avenue for future work.

<table>
<thead>
<tr>
<th>Network</th>
<th>SqrtMSE ($\times 10^{-3}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSNN</td>
<td>15.058 ± 6.5256</td>
</tr>
<tr>
<td>TSNN + subdivision</td>
<td>14.926 ± 6.5918</td>
</tr>
</tbody>
</table>

Table 3: Comparisons of per-pixel SqrtMSE for the test set after applying a threshold to the ground truth TS displacements.

<table>
<thead>
<tr>
<th>Method</th>
<th>TSNN (original)</th>
<th>TSNN (threshold)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ray Intersection</td>
<td>11.670 ± 3.2160</td>
<td>10.958 ± 2.9056</td>
</tr>
<tr>
<td>Extrapolation</td>
<td>39.564 ± 27.087</td>
<td>95.200 ± 48.406</td>
</tr>
<tr>
<td>Combination</td>
<td>14.279 ± 4.5970</td>
<td>15.405 ± 5.5923</td>
</tr>
</tbody>
</table>

Table 4: Breakdown of the TSNN errors ($\times 10^{-3}$) in Tables 2 and 3 based on whether each pixel contains vertices assigned via ray intersection, diffusion, or a combination of both.
References


[57] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. Pifu: Pixel-aligned implicit function for high-resolution clothed human digit-


14