DOI: 10.1111/cgf.14107
ACM SIGGRAPH / Eurographics Symposium on Computer Animation 2020
J. Bender and T. Popa
(Guest Editors)
Volume 39 (2020), Number 8

Cloth and Skin Deformation with a Triangle Mesh Based Convolutional Neural Network

Nuttapong Chentanez[1,2], Miles Macklin[1,3], Matthias Müller[1], Stefan Jeschke[1] and Tae-Yong Kim[1]

[1]NVIDIA
[2]Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University
[3]University of Copenhagen

Figure 1: *Results of our upsampling networks for blouse-symmetric, tshirt, vest, and dress. For each case, left is low resolution simulation input, middle is the ground truth high resolution simulation and right is the output of our networks.*

Abstract

We introduce a triangle mesh based convolutional neural network. The proposed network structure can be used for problems where input and/or output are defined on a manifold triangle mesh with or without boundary. We demonstrate its applications in cloth upsampling, adding back details to Principal Component Analysis (PCA) compressed cloth, regressing clothing deformation from character poses, and regressing hand skin deformation from bones' joint angles. The data used for training in this work are generated from high resolution extended position based dynamics (XPBD) physics simulations with small time steps and high iteration counts and from an offline FEM simulator, but it can come from other sources. The inference time of our prototype implementation, depending on the mesh resolution and the network size, can provide between 4 to 134 times faster than a GPU based simulator. The inference also only needs to be done for meshes currently visible by the camera.

CCS Concepts

• ***Computing methodologies*** → ***Physical simulation; Neural networks;***

1. Introduction

Cloth and deformable body simulation has been used successfully in many games and movies. However, the cost of simulation goes up as the resolution and the number of objects increase. Simulation also needs to be done regardless of whether the objects are currently visible from the camera or not. One may also want to make the simulation details scale with the computation resource available, while maintaining overall long-term consistency across all level of details. A way to achieve this is by running simulations at a low resolution and upsampling the result to a desired level of detail. Another possibility, when cloth or deformable objects are driven mostly by low dimensional inputs, is to avoid the simulation altogether and infer the deformation from the inputs directly. An example is clothing on a character that mostly depends on the characters body parts' poses and velocities. Another example is cloth simulation that is compressed with PCA, where the current

cloth shape is determined by a handful of PCA coefficients. Yet another example is skin, which is driven by flesh deformation which in turn is mainly driven by the underlying bones. In this work, we are mainly interested in these problems, namely, cloth/deformable object up-resolution and cloth/deformable object regression from low dimensional inputs.

Recently, Deep Learning (DL) has become an invaluable tool in many fields. Its use in physics simulation has increased significantly in recent years. However, one main issue that limits its usage is that the most common types of DL are either fully connected (FCN) or convolutional on a regular grid (CNN). FCN requires at least $O(NM)$ weights, where N is the size of input and M is the size of output which limits the scalability of the approach. Most CNNs operate on grids, whereas most cloth and deformable bodies are represented by manifold triangle meshes. To allow image based CNN to operate on a triangle mesh, one will need a parameterization, which can be cumbersome to obtain and can have problems with distortion. Recently, graph based CNNs have started to find applications in this area, but they are designed to work on general graphs and not specifically on triangle meshes.

In this work, we propose a novel triangle mesh based CNN that can be used for cloth and deformable body upsampling and regression applications. Our network is designed specifically for manifold triangle meshes, potentially with an open boundary. It takes advantage of the fact that the topology of the mesh does not change during training and at inferencing time.

The main contributions of our work are as followed:

- A novel convolutional network that operates on manifold triangle meshes, where the same learned weights can be used for meshes with different topologies
- Novel convolution operators for triangle meshes that operate on vertices
- Novel triangle mesh based average and max pooling and unpooling operator construction
- Applications of the network on several problems including cloth upsampling, pose to cloth regressions, PCA coefficients to cloth deformation and joint angles to hand skin deformation

2. Related Work

There have been a number of works that consider the problem of enriching cloth simulation details. A first type of approach are procedural methods. They typically use an underlying buckling model and solve it on the fine mesh [HBVMT99, KCCL01, LC04, KL07, RPC*10]. A simplified quasi-static GPU based solver was used to add wrinkles in [MC10].

Enhancing cloth detail by using data generated with high resolution simulation has been considered in a number of works. Feng et al. [FYK10] used a number of techniques to match low and high resolution simulations for creating a non-linear upsampling operator. Kavan et al. [KGBS11] constrained the low resolution simulation to match the high resolution at low frequencies and trained a linear upsampling operator, optionally with oscillatory modes. During run-time Seiler et al. [SSH12] computed non-linear weights to blend the differences between pre-computed examples of low and high resolution. In computer vision, Lähner et al. [LCT18] uses a Pix2Pix [IZZE17] network to learn to upsample the normal map image of a coarse simulation to real-world captured cloth wrinkles. Chen et al. [CYJ*18] trained a CNN akin to image super resolution to upsample low resolution cloth to high resolution in texture space. Oh et al. [OLL18] train fully connected networks (FCNs) for subdividing a triangle into 4 triangles and upsampling the cloth simulation. They average the outputs of the edge vertices predicted by the network on adjacent triangles. The authors later improved the result by using a grid instead of triangles and take input from past frames into account as well in [LOL19].

A number of works attempt to generate clothing deformation directly from character poses without simulation. Kim et al. [KKN*13] generate a large motion graph by simulating clothing resulting from a large number of character poses and compress the data to a small size, which can be queried quickly during runtime. Bailey et al. [BODO18] use multiple FCNs to learn the difference between the deformation resulting from complex character rigs and linear skinned blending. Jin et al. [JZGF18] uses CNNs to learn regressing from joint rotation matrices to cloth displacements in patches which are then merged together by projecting to PCA coefficients and reconstruct the final mesh. Santesteban et al. [SOC19] uses FCNs to fit garment onto human shapes and poses represented by SMPL [LMR*15].

A number of works use machine learning to try to replace physics simulation. Fulton et al. [FMD*19] use PCA to compute a low dimensional embedding of a deformable mesh and train an auto-encoder FCN to reduce dimension even further and use L-BFGS [BNS94] to solve for physics in the latent space. Holden et al. [HDDN19] also uses PCA and train FCN to learn the temporal evolution of the PCA coefficients. Tan et al. [TPGM19] uses Graph based CNN to train an auto-encoder for cloth simulation data. They train FCN to learn temporal evolution of the latent space value and the decoder to obtain the output mesh.

All the deep learning works for cloth deformation thus far use either FCN or image based CNN for function approximation, with the exception of [TPGM19], where only simple rectangular cloths are demonstrated. FCN limit the number of inputs and outputs that can be used efficiently. Using image based CNN for these applications requires a reasonable mesh parameterization [HPS08] and, for practical garment, where seams cannot be avoided, handling the jump in parameterization will be required. Moreover, computation will also need to be spent on pixels that do not correspond to vertices. Our work address these problems by utilizing a triangle mesh based CNN.

A number of works perform CNN on triangle meshes. GCNN [MBBV15] define convolutions on patches based on geodesic polar coordinates around each vertex for a shape retrieval task. ACNN [BMRB16] proposed an anisotropic convolution operator based on the heat kernel. Poulenard and Ovsjanikov [PO18] proposes multi-direction convolution on triangle mesh and demonstrate it on classification, shape segmentation and shape matching problems. MoNet [MBM*16] uses a Gaussian mixture model with learnable kernel weights around vertices to define convolution. Lim et al. [LDCK19] propose a spiral convolution operator on LSTM, where the first vertex of the spiral are chosen randomly. Recently, Bouritsas et al. [BBP*19] propose to consistently choose the first vertex of the spiral that is closest geodesically to a vertex and also use mesh simplification to generate lower resolution meshes. They then use barycentric interpolation to transfer data between mesh resolutions and define an auto encoder network for a fixed triangle mesh. They applied it to shape reconstruction problems. Gong et al. [GCBZ19] used a similar spiral convolution but pooling is done by a mesh simplification matrix and unpooling is done with barycentric inter-

polation from the closest triangle. They demonstrated the network on dense shape correspondences, 3D facial expression classification and 3D shape reconstruction. In contrast to some networks in this work, each network in [BBP*19] and [GCBZ19] are for one specific triangle mesh only and they also do not consider physics related problems. Hanocka et al. [HHF*19] define edge based convolution on a triangle mesh and perform pooling/unpooling during training and inferencing via edge collapses based on the activation on edges. While this allows for the edge collapse to be task dependent, the training and inferencing cost is high as the edge collapse needs to be performed during runtime. Moreover, the input and outputs are specified on edges, which are not directly applicable for problems where input or outputs naturally lie on vertices.

A number of works map the triangle mesh onto a 2D image before performing convolution. Maron et al. [MGA*17] construct a torus from four copies of the mesh and map it to a flat torus. Haim et al. [HSBH*19] construct a branch covering map and flatten it to a toric surface. Ezuz et al. [ESKBC17] map the mesh onto an image by minimizing a differentiable distortion metric, which allows the mapping to be done in a DL framework. While these methods try to reduce the amount distortion introduced by the mapping, the distortion is inherently not always avoidable.

3. Method

Our networks are built using three basic building blocks, namely Convolution, Pooling and Unpooling operators that operate directly on a manifold triangle mesh. They are connected together to create DownConv and UpConv blocks, shown in Figure 5, which are then connected together to form our encoder-decoder network with skip connection and the decoder network shown in Figure 6 and 7 respectively. For the ease of reading, abbreviations used in this section are summarized in Table 1.

Abbrev	Description
L	Length of convolution filter
SP	Spiral convolution without dilation of [BBP*19]
SPD	Spiral convolution with dilation of [BBP*19]
R_s	List of sampled one-ring curve
E_s	List of sampled best fit ellipse
RC	Convolution based on the one-ring curve (our method)
EC	Convolution based on the best fit ellipse (our method)

Table 1: *Abbreviations used throughout the method section.*

3.1. Convolution

Unlike images, where the connectivity between pixels is regular, manifold triangle mesh vertices are connected irregularly. Hence, it is not straightforward to define a convolution operator for them.

Recently, impressive work that defines CNNs on triangle meshes was published [BBP*19,GCBZ19]. They utilize a vertex based spiral convolution. The convolution is defined as weighted sums of vertices in the spiral sequence starting from the center vertex. The spiral sequence spirals out in a counter-clockwise direction, where the first vertex is chosen to be the one that is closest geodesically to a fixed vertex. The spiral sequence can be dilated by a factor of k by including a vertex and skip the next $k-1$ vertices in the sequence, where $k \in 1,2$ were used in the paper. The length of the sequence is then truncated to a fixed number L. Inspired by the elegance of the method, we came up with a novel family of triangle mesh convolution operators that result in lower errors in many cases compared to the spiral convolution.

We first made several observations about the spiral convolution operator. The spiral sequences of different vertices do not have a perfect spatial or semantic relationship with each other, as each vertex can have a different number of neighbors. For example, the j^{th} vertex in the spiral sequence of a vertex a can be from a one-ring neighbor, but for vertex b, it can be from a two-ring neighbor and they can be in different relative position in material space with respect to vertex a and b. From a physics simulation point of view, they likely have quite different influences on vertex a and b. However, in the spiral convolution operator, they use the same weight. Nevertheless, based on our tests, the spiral convolution can still produce good deep learning models. This information gives us insight that vertices involved in the convolution that share the weights do not have to have a perfect correspondence.

Another observation we made is that the spiral convolution (SP) without dilation is biased toward including only the first few vertices in the two-ring neighbors that just happen to be in the spiral sequence, as typically L is not large enough to include the whole two-ring neighbors. Therefore, the information from the two ring neighbors tend to concentrate on a small fraction of directions as shown in Figure 2, SP. For the spiral convolution with dilation (SPD), while its receptive field is large, it skips some vertices near the center vertex which likely contain useful information, as seen in Figure 2, SPD.

Based on these observations, we propose novel ways to define mesh convolution operators that yield lower error than the spiral convolutions in most cases. For each vertex, we enumerate its one-ring neighbors in a counter-clockwise order. As the mesh is manifold, the one-ring neighbors are well-defined. We then treat the one-ring as a piecewise linear curve and sample it uniformly, based on length, with $L-1$ samples and store them in a list R_s. The first sample of the list is placed at the neighbor closest geodesically to a fixed vertex. Let's say the center vertex is v and the one-ring neighbors are $v_0, v_1, ..., v_{n-1}$. We first compute

$$l = \sum_{i=0}^{n-1} ||v_i - v_{(i+1) \bmod n}||_2, \tag{1}$$

we then create uniform sampled points along the piecewise linear curve with spacing $l/(L-1)$. We will refer to the convolution constructed with these sampled points as Ring-based Convolution (RC). An example of the sampled points are shown in Figure 2, RC, as hollow blue circles.

For a boundary vertex of a manifold mesh, where the one-ring neighborhood is topologically equivalent to a half disk, we pad a dummy vertex to complete the ring before sampling. The dummy vertex position is placed in the mid way, angular-wise and distance-wise, to the two ends of the one-ring, when oriented consistently with the normal of the center vertex. Figure 3 left illustrates the placement of this vertex. The dummy vertex is included in the curve for generating the $L-1$ samples. The dummy vertex allows us to have a closed curve for sampling, and later on, it will indicate the boundary for zero-padding the convolution operator.

Another way of constructing the samples we considered is by least squares fitting an ellipse [OZM04] of the projected one-ring neighbors, along with the padded dummy vertex, if any, onto the plane defined by the center vertex position and its normal. We then

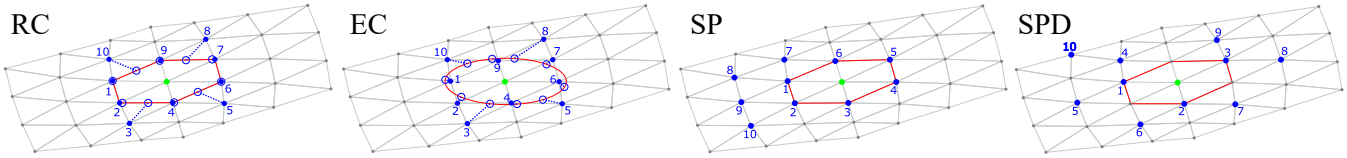

Figure 2: *RC, EC show our one-ring based and ellipse based convolutions respectively. SP and SPD show spiral convolution of [BBP* 19] and [GCBZ19] without and with dilation respectively. The center vertex is shown as green dot. Blue dots indicate vertices in the convolution filter, where the numbers specify the indices. The red polygon/ellipse shows the curve we used for sampling. The hollow circles indicate the RC and EC uniform samples, and the dashed lines indicate the minimum sum distance assignment. In all cases, vertex 1 is the geodesically closest vertex to the fixed vertex.*

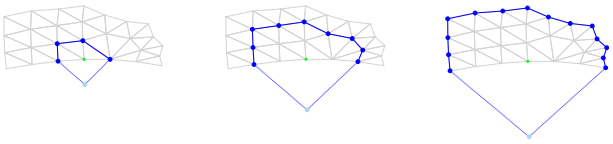

Figure 3: *Left, Middle, Right are one-ring, two-ring and three-ring neighbors and their dummy vertices.*

use the ellipse to generate the uniform length samples, starting from the point on the ellipse closest to the neighbor that is geodesically closest to the fixed vertex. We will refer to the convolution constructed with the sampled ellipse as Ellipse-based Convolution (EC). An example of the ellipse is shown in Figure 2, EC, where the red ellipse is the best fit ellipse and the hollow blue circles indicate the samples. The rationale behind this choice is that, while we prefer the convolution to respect the shape of the local neighbors, it may be irregular. The least square fit ellipse provides anisotropicity while being somewhat regular.

We then enumerate the two-ring and three-ring neighbors in a counter-clockwise ordering. We also add a dummy vertex in the same manner as we did for the one-ring if the two-ring or three-ring neighbors cross a boundary, as shown in Figure 3 middle and right. We then collect all the one-ring, two-ring and three-ring neighbors along with the dummy vertices, ie. all dark and light blue vertices in Figure 3, and add them into a list, L_c. We now compute a $|R_s| \times |L_c|$ Euclidean distance matrix. We then solve a rectangular assignment problem [Kuh55], where for each sample in R_s, we pick a vertex in L_c, such that no vertex in L_c is chosen more than once and the sum of the distance is minimized. Examples of such assignments are shown in Figure 2 RC and EC as dashed lines. We use an $O(|L_c|^3)$ algorithm [JV88]. Note though that this needs only be done once at pre-processing time and it does not add significant overhead to our pooling and unpooling operators constructions to be discussed in the next sections. Now, each sample in this list R_s has a corresponding unique vertex in the mesh, some of which may be the dummy vertices, which will indicate zero-padding for the boundary. We use these $|R_s| = L - 1$ vertices in the weighted sum for the convolution, where the dummy vertices are replaced with 0 index which will always have zero value. Our convolution operators tend to include all the one-ring neighbors and some two or three ring neighbors sampled from directions where it was under sampled by the one-ring neighbors.

For each mesh, we consistently choose the fixed vertex for geodesic distance to be the one where its position in the material

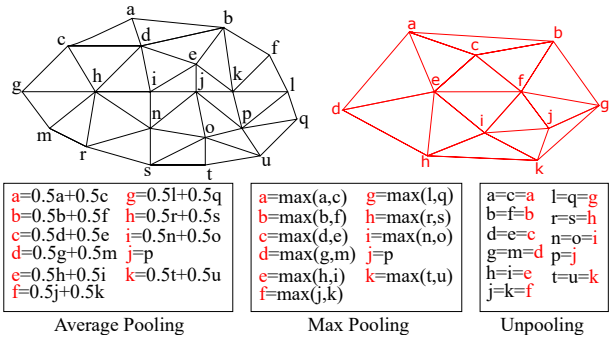

Figure 4: *Meshes before (black) and after (red) a step of independent edge collapses. We use the edge collapses to define our novel pooling and unpooling operators. Average pooling, max pooling and unpooling operations are shown in the boxes.*

space is closest to

$$(\frac{\max_x + \min_x}{2}, \max_y, \frac{\max_z + \min_z}{2}), \tag{2}$$

where $(\min_x, \min_y, \min_z), (\max_x, \max_y, \max_z)$ is the bounding box of the mesh in material space. The y-axis is the vertical axis. The rationale is that the fixed vertex should be chosen so that the convolution filter orients somewhat consistently in material space. In this case, it orients such that the first vertex tends to point upward. This choice of the fixed vertex also makes the convolution operator more consistent across different meshes.

3.2. Pooling and Unpooling

In addition to the previously defined convolutional operators, our novel pooling and unpooling operators are defined using parallel independent edge collapses and their reverse, as demonstrated in Figure 4. To decide which edges to collapse, we use a variant of [GH97] which maintains a priority queue of edges based on a quadric error. The edge that has the lowest quadric error, when the two endpoint vertices are replaced by the mid-point, is chosen to be collapsed. We then mark the edges sharing a vertex with the collapsed edge not be collapsible in the current pooling pass. We also disallow collapsing edges that will yield a non-manifold mesh. We continue to collapse edges until no more edge can be collapsed, or until the quadric error is higher than a threshold and the number of vertices is less than 60% the number of vertices at the start. The edges chosen in this manner can be collapsed independently in parallel without data dependency and we will end up

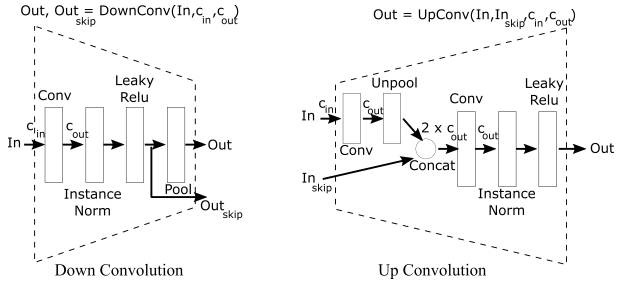

Figure 5: *DownConv block (left) performs a convolution on In followed by instance normalization, leaky ReLU and Pooling. It outputs features on the original resolution and the lower resolution meshes as Out and Out_{skip} respectively. UpConv (right) does convolution on In followed by Unpooling and then concatenate with In_{skip}, which is Out_{skip} from the corresponding DownConv block, followed by another convolution, instance normalization and leaky ReLU.*

with between 50% to 60% the number of vertices, unless there are too many non-collapsible edges, which only can happen when the mesh is very coarse. The edges that are collapsed define our pooling operator. The vertices that remain either come from one or two original vertices. For a vertex that comes from one original vertex, we simply copy the features from the original vertex. For a vertex that comes from two original vertices, we either use the componentwise average or max of the features of the two original vertices. Our pooling operator guarantees that the features at all fine vertices are used in computing the features of coarse vertices. This is in contrast to the pooling operators based on barycentric interpolation used in [GCBZ19, BBP*19] where some fine vertices may never be used in computing any coarse vertex. MeshCNN [HHF*19] also does pooling by averaging features, but they operate on edges and their pooling cannot be done in parallel as the edges to be collapsed are chosen sequentially during training and inferencing.

Our unpooling operator is defined using the same set of edge collapses used for pooling, albeit in a reverse way. We simply copy the value of an input vertex to either one or two output vertices, depending on whether it comes from edge collapse or not. Our unpooling operator is hence akin to nearest neighbor upsampling.

3.3. DownConv and UpConv Blocks

We define our down convolution block, shown in Figure 5 left, in a similar manner to [HHF*19], ie. convolution followed by instance normalization [UVL16] followed by a leaky-ReLU and then pooling. We store the features before pooling for later use in UpConv.

Our up convolution block, shown in Figure 5 right is defined in a similar manner to [HHF*19], ie. convolution followed by unpooling and optionally concatenated with features from the DownConv block followed by convolution then instance normalization and leaky ReLU.

3.4. Encoder-Decoder Network

For regression problems in which the input is specified on the mesh vertices, we use an encoder-decoder architecture with skip connection in the similar manner to U-net architecture [RPB15]. The skip connection does not suffer from information bottleneck unlike the encoder-decoder without skip connection. The network consists of k DownConv blocks followed by k UpConv blocks followed by two Convs, as shown in Figure 6.

For our choice of edge collapse, we will have between 0.5^k to 0.6^k the number of vertices in the innermost level of our network. For $k = 10$, which we used in all of our experiments, it corresponds to between $1/1024$ and $1/165$. Note that when the number of vertices is very small, the condition that we disallow edge collapse causing non-manifold meshes will prevent the number of vertices to reduce further, i.e. the pooling and unpooling will simply be an identity operator. Note that this network is fully convolutional and hence can be applied to arbitrary manifold triangle meshes regardless of the number of vertices and topology.

We also found that in most cases, for the same number of learnable parameters, it is beneficial to make the network wider but shallower. In this case, some DownConv blocks are replaced with pooling operators and some UpConv blocks are replaced with unpooling operators. We will discuss this in the results section.

3.5. Decoder Network

For the regression problems in which the input is not naturally on the mesh vertices, we use a fully connected network that outputs the features of the coarsest vertices followed by the decoder without skip connection. An example of this is shown in Figure 7.

3.6. Loss Functions

Having the network directly produce the output on vertices makes it easy for us to employ various loss terms. The loss terms we have experimented with are L1, L2, and face normal difference. Throughout this section, let the superscript g denote the ground truth quantity and the superscript $*$ denotes the output produced by the network. Let x_i be the vertex position of the i^{th} vertex, n_j be the normal of the j^{th} face. Suppose the j^{th} face consists of vertices x_0, x_1, x_2 in counter-clockwise order, then

$$n_j = normalize((x_1 - x_0) \times (x_2 - x_0)). \tag{3}$$

We define our L1 vertex position error as

$$L_1 = mean\left(\frac{||x_i^g - x_i^*||_1}{3}\right), \tag{4}$$

L2 vertex position error as

$$L_2 = \sqrt{mean\left(||x_i^g - x_i^*||_2^2\right)} \tag{5}$$

and the L1 normal error as

$$L^n = mean\left(\frac{||n_i^g - n_i^*||_1}{3}\right). \tag{6}$$

Our total loss function is hence

$$L_{total} = \alpha L_1 + \beta L_2 + \gamma L^n. \tag{7}$$

For some meshes, simply minimizing L_1 yields virtually indistinguishable results from the ground truth already, nonetheless, we found that for meshes with large low curvature area, having the L^n term improves visual quality. Also, for the decoder network, we observe that sometimes error is concentrated at certain vertices, instead of distributing throughout. Having the L_2 term helps in this case. We have also experimented with edge dihedral angle loss, but this did not improve visual quality noticeably. Temporal coherence

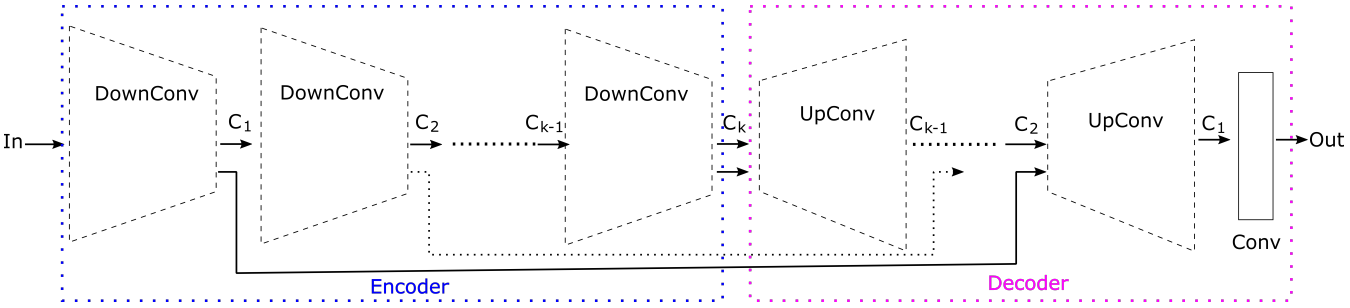

Figure 6: *The encoder-decoder Network is used when both the input and the output are defined on the vertices of a triangle mesh. The input is passed through a series of DownConv blocks to aggregate features on ever coarser meshes until reaching the coarsest mesh. The features are then passed through UpConv blocks to scatter information on ever finer meshes until reaching the original resolution. The DownConv blocks' Out_{skip} are connected to the corresponding UpConv blocks' In_{skip} for skip connection. In practice, some of the DownConv and UpConv blocks are replaced with Pooling and Unpooling to reduce the number of learnable parameters.*

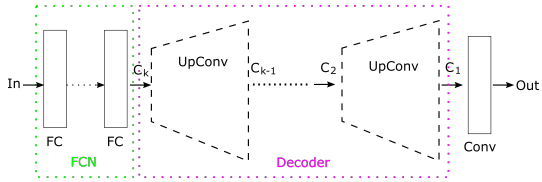

Figure 7: *The decoder network is used when the input is a vector of real numbers but the output is defined on the vertices of the mesh. The input is passed through fully connected layers to transform it to the per-vertex features of the coarsest mesh. It is then passed through a series of UpConv blocks, where the In_{skip}'s are not used, so no concatenation is done. In practice, some of the UpConv blocks are replaced with Unpooling to reduce the number of learnable parameters.*

loss as in [LCT18] could also be used, but we found that the temporal incoherency is small enough to not be visually noticeable in our examples.

4. Results

We implement our network in PyTorch and use Adam [KB14] as the optimizer with the default parameters ie. $lr = 10^{-3}, betas = [0.9, 0.999], eps = 10^{-8}$, leaky-ReLU with a negative slope of 0.01, instance normalization with $eps = 10^{-5}$ and batch size of 3 in all experiments. Videos, sample codes and data will be available at https://github.com/nchentanez/TriMeshCNN to aid future research. The bounding box of the meshes used in our experiment has a length between 0.7 and 2.0 units. All timing experiments are done on a PC with NVIDIA RTX2080 Ti and 3.4 GHz Intel Core i7-6700 with 64GB of RAM. We demonstrate the application of our networks in several regression problems in this work as follows:

1. Upsampling low-res cloth simulation to high-res cloth simulation (Cloth Upsampling)
2. Regress the poses and velocities of the body parts of a mannequin to cloth vertex positions (Pose to Cloth)
3. Regress cloth simulation reconstructed with only a few principal component analysis (PCA) bases to the original full space (PCA Details Recovery)

Problem	Input	Predictor	Target Output
Cloth Upsampling	Loop-subdivided low resolution cloth vertices position relative to torso	Input	High res cloth vertices pos relative to torso
Pose to Cloth	Position, quaternion, linear vel and angular vel of each body parts (208 scalars)	LBS of rest cloth relative to torso	High res cloth vertices pos relative to torso
PCA Details Recovery	PCA coefficients (16 or 32 or 64 scalars)	LBS of rest cloth relative to torso	High res cloth vertices pos relative to torso
Joint Angles to Hand	Joint angles of the bones in the hand (18 scalars)	LBS of hand in world space	High res hand vetices pos in world space

Table 2: *Summary of the input, the predictor and the output of the networks in our examples. LBS stands for linear blend skinning.*

4. Regress hand and finger joint angles to high resolution wrinkles of the hand skin resulting from a high resolution 3D flesh simulation (Joint Angles to Hand)

In all of the problems we consider, the networks do not directly predict the world space position of all vertices. They only need to predict the displacement of vertices from easy-to-compute predictors that already contain reasonable low frequency components of the output meshes. The input, the predictor and the target output for each problem are summarized in Table 2.

For the Cloth Upsampling, PCA Details Recovery, Pose to Cloth problems, we generate ground truth simulation data using a GPU accelerated XPBD Solver [MMC16] with dynamic triangle collision detection and handling. The simulations are run with a time step of $1/60s$ using 8 sub-steps each with 75 iterations. The clothing and mannequin animations are from UC Berkeley Garment Library [dJNO*12]. We subdivide the coarse meshes to have all edge lengths under 0.08 units and use this as our low resolution mesh. The high resolution mesh comes from subdividing the low resolution mesh with Loop subdivision twice, resulting in roughly 16 times the number of vertices. For the Cloth Upsampling problem, we use a spring that allows compression up to 75% to facilitate wrinkle formation as suggested in [JLGF17]. We add a hard XPBD constraint (compliance zero), to make the low frequency components of the high resolution to match with those of low resolution using the harmonic test functions described in [KGBS11]. For a mesh with V vertices, we use $V/40$ first lowest frequency test functions.

For the Joint Angles to Hand problem, we use a quasi-static CPU based FEM solver with Neo-Hookean material model. The hand skin is embedded in a volumetric tetrahedral mesh, which in turn

is driven by the underlying bones. Wrinkle Meshes [MC10] is used to add additional wrinkles to the hand.

5. Comparison and ablation study

Throughout this section, we use the following notation. Let D_k^h and U_k^h denote a DownConv block and a UpConv block respectively, where the output has h channels, where if h is absent, it indicates simple Pooling and Unpooling and k refers to how many times the same block is repeated (k is omitted if it is 1). For example D_3^{128} is the same as $D^{128}D^{128}D^{128}$. If the network is an encoder-decoder, we only specify the DownConv layers and it is understood that the symmetric UpConv layers are added with appropriate skip connections. We abbreviate low resolution vertex count (lv) and high resolution vertex count (hv).

5.1. Upsampling

We first compare our proposed convolution operators and the non-dilated and dilated spiral convolution. We use $\alpha = 1, \beta = 0, \gamma = 0$ for these experiments, so that objective is to minimize L1 positional error, which is easy to interpret. Based on some initial tests, an encoder-decoder architecture of the form $D^xD^{2x}D^{4x}D_5D^{4x}$ produces a good quality result, so we use it for testing the various convolutions for a medium mesh, pants (265 lv, 4063 hv), and a large mesh, tank-female (969 lv, 14634 hv). We use frames from the dance animation (4233 frames) for these experiments. Throughout this section, we use the following abbreviations: RC_s and EC_s denotes our assignment problem based convolution with the filter size of s using uniform sampling of the one-ring neighbors and the best fit ellipse respectively, MeshCNN refers to using the network of [HHF*19], Spiral++ refers to using the network of [GCBZ19]. The details about MeshCNN and Spiral++ are given in the supplementary material. We also experimented with using our encoder-decoder network architecture with our proposed pooling and unpooling operators, but replacing our proposed convolution operators with several alternatives as follows: $V4$ denotes a baseline vertex based convolution adapted from [HHF*19] as explained in the Appendix, SP_s and SPD_s are the spiral convolution operators [BBP*19,GCBZ19] without and with dilation of 2 respectively. We split training and testing as 90% and 10% respectively and report the error after training for 200 episodes as shown in our supplementary material, Table 1. Note that we cannot run MeshCNN on the tank-female mesh as it runs out of memory on our 16GB GPUs.

From the table, we can see that our proposed networks (RC, EC) yield significantly lower errors compared to MeshCNN and Spiral++ in all cases. RC and EC also outperform V4 in all cases and outperform SP and SPD in most cases. This suggests that our proposed network architecture, pooling and unpooling operators and convolution operators all contribute to lowering the errors. RC/EC yield lowest training/testing errors in 34 cases, while SP/SPD yield lowest errors in 14 cases. One can notice that despite the fact that the tank-female mesh has roughly 4 times the number of vertices compared to the pants mesh, the error for tank-female mesh is not larger than the pants mesh, for a network with the same number of learnable weights. As V4, MeshCNN, Spiral++ do not yield the lowest error in any of these cases, we exclude them from further consideration.

We then proceed to experiment with $RC_{13}, EC_{13}, SP_{13}, SPD_{13}$ for various architectures as well as experimenting with using a max operator for pooling as shown in our supplementary material, Table 2. For each architecture, we find x such that it has the number of learnable weights as close as possible to that of $D^{80}D^{160}D^{320}D_5D^{320}$, which is 7.8M. RC_{13}/EC_{13} produce the lowest error in 37 cases, while SP_{13}/SPD_{13} produces the lowest error in only 1 case. Max produces the lowest error in 28 cases, while average produces the lowest error in 18 cases. Hence in practice, max pooling is worth considering. The architecture that yields the lowest training and test error is $D^{62}D^{124}D^{248}DD^{248}D_3D^{248}$ with RC_{13}^{max} and hence we use it for all remaining experiments that use encoder-decoder architecture, unless otherwise specified.

We next experiment with training a network using multiple meshes with different topology but share the same weights. This case was not considered in [BBP*19] and [GCBZ19]. We sort the meshes from the Garment Library by the number of vertices in ascending order and choose the first 10 meshes (lv, hv) as follows: blouse-asymmetric (158, 2198), blouse-symmetric (161, 2210), blouse-symmetric-loose(174,2382), skirt(193,2794), dress-asymmetric(204,2970), shorts(235,3559), tank-male(249,3666), pants(265,4063), dress-victor(327,4827) and dress2(329,4922). The result is shown in our supplementary material, Table 3. As expected, the error tends to increase as the number of meshes used for training grows, but they still remain relatively low. The result shows that our network architecture allows for the possibility of upsampling multiple meshes with a single network, when needed.

5.2. Pose to Cloth

We now consider the decoder network for regressing from pose to cloth displacement. We consider various architectures and choose x so that the number of learnable weights is as close as possible to that of $D^{80}D^{160}D^{320}D_5D^{320}$, which is 7.2M. We choose the fully connected part to simply be a linear layer followed with instance normalization and a leaky-ReLU. The result for the dress2 mesh is shown in our supplementary material, Table 4. In this case, our RC_{13}/EC_{13} produces lowest errors in 23 cases, while SP_{13}/SPD_{13} yield lowest error in 7 cases. We note though that RC, EC, SP and SPD can be interchanged easily. The inference phase need not be changed at all, and in practice one can test to see which one yields the lowest error for a given problem and use it. As RC_{13} with $U^{208}UU^{208}UU^{208}UU^{208}U^{104}U^{52}U^3$ provides the lowest training error and also a small test error, we use it in the rest of our decoder network experiments, unless otherwise specified.

We now turn to the visual quality of various choices of loss functions. For most experiments, having L_1 error alone produces a visually smooth surface already, nonetheless, in some cases, the surface can be slightly bumpy. Therefore, we include the surface normal loss, L^n, which improves the smoothness of the surface. Moreover, for some decoder experiments, we also found that with just L_1 error alone, in some rare cases the network produces a small error in most vertices but a large error in a few vertices manifesting as small spikes. Adding L_2 loss removes these artifacts. We include L_2 loss with weights of 1 for all the visual results with the decoder network. This is demonstrated in Figure 8, where one can notice that the L_1 alone produces a bumpy surface in some places, while having the L^n loss with $\gamma = 0.02$ helps improving the visual quality and adding L_2 loss with $\beta = 1$ does not reduce the visual quality.

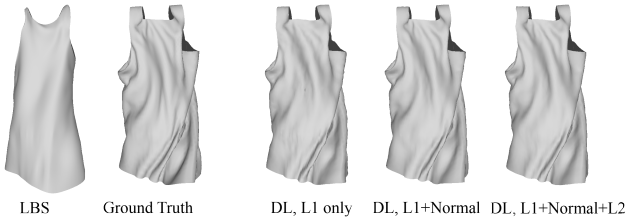

Figure 8: *From left to right, LBS, ground truth, L_1 only, $L_1+0.02L^n$,$L_1+0.02L^n+L_2$. One can see that L1 only results in a bumpy surface in some places, while $L1+0.02L^n$ and $L_1+0.02L^n+L_2$ are visually comparable.*

5.3. Visual Results

For visual results, we let each network train for 200 episodes using randomly selected 90% of the frames from both the dance (4233 frames) the karate-smooth (4155 frames) animations before saving the network out for inference. The network training takes between 4 to 20 hrs for training depending on the mesh size and the network size. We also experimented with reduced network sizes simply by changing the filter size from 13 to 9, reduce x to a half and a quarter for some selected cases and report the timing and the errors for them below. From all experiments we run, our networks with between 5M to 8M learnable parameters are able to produce training output that are largely visually indistinguishable from the ground truth. For inference timing, we save out the network and run the inference in C++, using cuBLAS for dense matrix multiplications, cuDNN for 2D convolutions and our own unoptimized CUDA kernels for sparse matrix operations, pooling, unpooling, leaky-ReLU and instance normalization.

Figure 1 shows still frames from animations of various meshes for the encoder-decoder network for cloth upsampling. We report the timing information and speed up gain of this use case for several meshes in Table 3. For simulation, we report the time required for the low resolution simulation alone and the high resolution simulation alone, without the constraints to match the low resolution. For the upsampling problem, we report the speed up gain over GPU physics simulation, if we were to run low res simulation with 8, 2 and 1 substeps and up-resolution it with our network for rendering at 60Hz compared to running high resolution physics simulation at 8 substeps. Using fewer substeps for the low resolution simulation is possible because the low resolution simulation alone does not need as many substeps as high resolution. Also, our network only needs to be run for the frame we render and only for the mesh visible by the camera. We believe the most common use case would be to run the low res simulation with 1 substep and use DL to up-resolution to high res result as if it were simulated with 8 substeps, in which case, the speedup expected would be between 4.4X to 10.3X over the high res sim.

Figure 9 shows frames from animations of the decoder network used for the Pose to Cloth problem. In this case, the network can operate without the use of a physics simulator, when the cloth is driven entirely by the character. The timing, errors and the speedup compared to high resolution GPU physics simulation for several meshes is reported in Table 4. Depending on the mesh and the size of the network used, the speedup is between 10X to 65X with varying degrees of visual quality / speed tradeoff.

Figure 9: *Result of our Pose to Cloth deformation networks for dress2, skirt, blouse-symmetric, blouse-asymmetric, shorts, tankmale respectively. For each case, left is linear blend skinning, middle is the ground truth high resolution simulation and right is the output of our networks. Clothing meshes come from UC Berkeley Garment Library.*

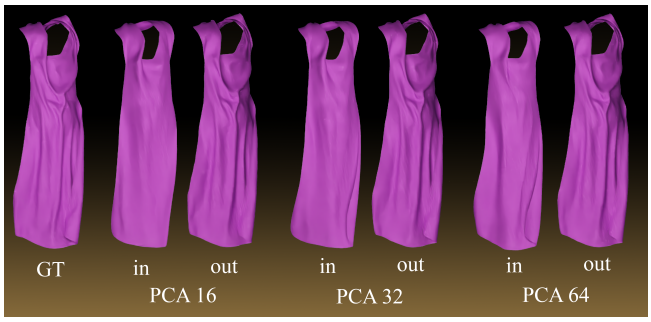

Figure 10: *Ground truth, PCA compressed cape and output of our network whose inputs are the PCA cofficients and output are the displacement from PCA to ground truth for 16, 32 and 64 PCA bases respectively. Notice how our network is able to closely match the ground truth. Clothing meshes come from UC Berkeley Garment Library.*

Figure 10 shows the result of our decoder network for adding back details of PCA reduced order simulation for the cape (369 lv, 5746 hv) mesh. These networks receive 16, 32 or 64 PCA coefficients as input and produce displacements from the PCA reconstruction to the high resolution simulation. The timing and the error is reported in Table 4. This gives a possibility that our network can be used for enhancing the quality of the reduced order simulation or DL based simulation methods such as [FMD*19] and [HDDN19], when the cloth or the deformable objects are in a close-up view.

Figure 11 shows frames from the Joint Angles to Hand experiment. This hand mesh consists of a large number of vertices (33k) and is simulated with a volumetric FEM simulation and a pose processing, the data set consists of 5000 frames. Our network is able to regress from 18 joint angles to the hand deformation with good visual acuity. The timing information and the errors can be found in Table 4. The speedup over CPU simulation is between 25X to 134X. This demonstrates a possibility of using our network for regressing a complex non-linear system that results in a high resolution triangle mesh deformation with a large potential speed-up gain.

5.4. Large Dataset

We also experimented with training our network on a large dataset consisting of animations from the CMU mocap database [cmu]. We

Figure 11: *First row: LBS of the hand mesh skinned to the current bone transform. Second row) Ground truth of hand skin driven by an offline FEM Neo-Hookean Material Model Simulator. Third row) Output of our network. Fourth and fifth rows) Image differences of the ground truth and LBS and DL respectively, with brightness amplified by* 150%. *This shows that FEM generates substantial deformation at both small and large scales, which DL is able to capture well. Please zoom in to see more details.*

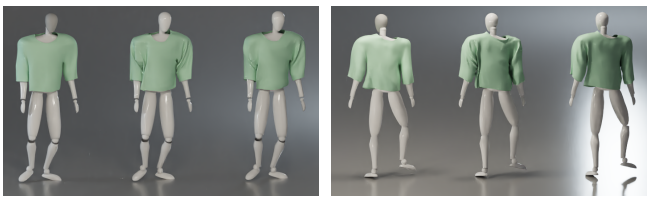

Figure 12: *A small network (RC_9, $x/4$) is trained with frames from animations from CMU mocap database. The figure shows frames from an animation not used for training. Left is input, middle is ground truth, right is the DL upsampled result. It demonstrates the ability for our network to generalize to unseen data.*

rescale the length of the limbs to match with that of Berkeley's garment database manequin. We then put the garment onto the start frame of each animation by gradually blending in the joint angles of the poses at the first frame of the dance animation to the poses at the first frame of each CMU mocap, all while the cloth simulation is running. We then pause the animation for 100 frames to let the clothing settle before starting the animation. We discard animations that result in limbs' intersection or tangled simulation. We also omit 10 randomly selected animations for testing and use the remaining animation as our dataset. We ended up with the remaining 2340 animations consisting of 486720 frames of animation. We then train a small Cloth Upsampling network (RC_9, $x/4$) using tshirt2 mesh for 200 epochs, where we sample 10% of the frames of the training set for training in each epoch. The inference time of the network is 3.4ms. Results of our networks running on an animation not trained are shown in Figure 12 and the accompanying video, demonstrating the ability of our network to generalize to unseen data, when a sufficiently large dataset is used for training.

6. Discussion

We propose a triangle mesh convolutional neural network that utilizes novel convolutions, pooling and unpooling operators as building blocks. We demonstrate its applications on cloth upsampling, cloth regression from character poses, cloth regression from PCA coefficients, and hand skin deformation from bone joint angles. The network can produce high visual accuracy at small inference time and its size can be chosen as needed to provide various quality vs. performance trade-offs.

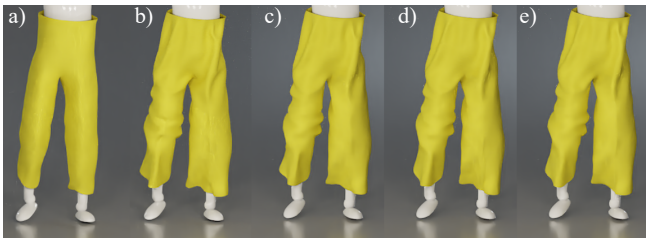

Figure 13: *A frame of pants animation not used for training. a) Low-res input. b) High-res ground truth. c) DL trained with* 90% *of random frames. d) DL trained with one every fifth frame. e) DL trained with one every tenth frame. The DL outputs are almost identical to the ground truth.*

The method however is not without drawbacks. As with most neural network problems, the quality of the result greatly depends on the amount and the quality of data used for training. The time required for generating large amount training data is significant.The network also can only be expected to provide good quality inference for data that is similar to the training set. Figure 13d,e shows a frame from the result when we train the network with one of every 5 frames and one of every 10 frames respectively. In these cases, it still manages to produce results that match visually well with the ground truth. However, if the input data lies completely outside of the training data, the network may not be able to produce as good quality results. The output of our network also does not guarantee collision free deformations. If desired, a collision post processing such as those employed in [SOC19] could be applied. We currently do not use it in our experiments. When training on the CMU mocap dabase, our network sometimes does not produce as much high frequency details in some area as in the ground truth. We speculate that details are somewhat smoothed out due to the minimization of L1 and L2 on large data set. Using generative adversarial network (GANs) and its variants [GPAM*14, GAA*17, WSW19] may improve the quality as reported in [LCT18]. We leave this as a future work.

Figure 14: *A frame of tshirt2, pants and tank-female mesh. Left is the input low res simulated with 1 substep, right is the output of our RC_9 max networks. The low res sim at 1 substep differs greatly from the 8 substeps used for training the network. Hence, the network has to extrapolate far outside the training set with varying degrees of quality.*

For the upsampling problem, we currently use the data generated by a low resolution simulation using 8 substeps. In this case, the networks have never seen the input of the low res simulation using 1 substep during training and hence the visual quality is not optimal, as shown in Figure 14. One would likely get a much higher quality result if the training data was generated such that the input was a low res simulation with 1 substep and the output is the high res simulation with 8 substeps, constrained to match the interpolated low res sim at low frequencies.

We also have not attempted a parameter search to find the best network given a preferred number of learnable weights for the smaller networks. Table 2 and 4 in the supplementary material suggest that for a given problem, the error can vary greatly with the choice of architectures. We believe the error could have been significantly lowered if this had been searched. As our network assumes constant mesh topology, it is not suitable for applications that require frequent mesh topology changes such as interactive mesh design and editing.

We believe our network is useful for the applications we consider in this paper and we think it can be used for other applications where the input and/or output are triangle meshes as well. Potentially interesting future applications of this network include regression from rig control parameters to mesh deformation [BODO18], generating triangle meshes with texture from images by combining it with image network and differentiable rendering [CLG*19], and predicting the results of geometric mesh optimizations such as curvature flows.

7. Acknowledgements

We thanks reviewers of Symposium on Computer Animation 2020 for their helpful comments and suggestions. We thanks members of NVIDIA GameWorks team for their valuable feedback. Some data used in this project was obtained from mocap.cs.cmu.edu which was created with funding from NSF EIA-0196217. Mannequin, clothing meshes, dance and karate-smooth animations are from Berkeley Garment Database.

References

[BBP*19] BOURITSAS G., BOKHNYAK S., PLOUMPIS S., BRONSTEIN M., ZAFEIRIOU S.: Neural 3d morphable models: Spiral convolutional networks for 3d shape representation learning and generation. In *The IEEE International Conference on Computer Vision (ICCV)* (2019). 2, 3, 4, 5, 7

[BMRB16] BOSCAINI D., MASCI J., RODOIÀ E., BRONSTEIN M.: Learning shape correspondence with anisotropic convolutional neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems* (USA, 2016), NIPS'16, Curran Associates Inc., pp. 3197–3205. 2

[BNS94] BYRD R. H., NOCEDAL J., SCHNABEL R. B.: Representations of quasi-newton matrices and their use in limited memory methods. *Mathematical Programming 63*, 1 (Jan 1994), 129–156. 2

[BODO18] BAILEY S. W., OTTE D., DILORENZO P., O'BRIEN J. F.: Fast and deep deformation approximations. *ACM Transactions on Graphics 37*, 4 (Aug. 2018), 119:1–12. Presented at SIGGRAPH 2018, Los Angeles. 2, 10

[CLG*19] CHEN W., LING H., GAO J., SMITH E., LEHTINEN J., JACOBSON A., FIDLER S.: Learning to predict 3d objects with an interpolation-based differentiable renderer. In *Advances in Neural Information Processing Systems 32*, Wallach H., Larochelle H., Beygelzimer A., dAlché-Buc F., Fox E., Garnett R., (Eds.). Curran Associates, Inc., 2019, pp. 9605–9616. 10

[cmu] Cmu motion capture database. http://mocap.cs.cmu.edu/. 8

[CYJ*18] CHEN L., YE J., JIANG L., MA C., CHENG Z., ZHANG X.: Synthesizing cloth wrinkles by cnn-based geometry image superresolution. *Computer Animation and Virtual Worlds 29*, 3-4 (2018), e1810. e1810 cav.1810. 2

[dJNO*12] DE JOYA M. J., NARAIN R., O'BRIEN J. F., SAMII A., ZORDAN V.: Berkeley garment library. http://graphics.berkeley.edu/resources/GarmentLibrary/, 2012. 6

[ESKBC17] EZUZ D., SOLOMON J., KIM V. G., BEN-CHEN M.: Gwcnn: A metric alignment layer for deep shape analysis. *Computer Graphics Forum 36*, 5 (2017), 49–57. 3

[FMD*19] FULTON L., MODI V., DUVENAUD D., LEVIN D. I. W., JACOBSON A.: Latent-space dynamics for reduced deformable simulation. *Computer Graphics Forum* (2019). 2, 8

[FYK10] FENG W.-W., YU Y., KIM B.-U.: A deformation transformer for real-time cloth animation. *ACM Trans. Graph. 29*, 4 (July 2010), 108:1–108:9. 2

[GAA*17] GULRAJANI I., AHMED F., ARJOVSKY M., DUMOULIN V., COURVILLE A. C.: Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems 30*, Guyon I., Luxburg U. V., Bengio S., Wallach H., Fergus R., Vishwanathan S., Garnett R., (Eds.). Curran Associates, Inc., 2017, pp. 5767–5777. 9

[GCBZ19] GONG S., CHEN L., BRONSTEIN M. M., ZAFEIRIOU S.: Spiralnet++: A fast and highly efficient mesh convolution operator. In *The IEEE International Conference on Computer Vision (ICCV)* (2019). 2, 3, 4, 5, 7

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 209–216. 4

[GPAM*14] GOODFELLOW I., POUGET-ABADIE J., MIRZA M., XU B., WARDE-FARLEY D., OZAIR S., COURVILLE A., BENGIO Y.: Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, Ghahramani Z., Welling M., Cortes C., Lawrence N. D., Weinberger K. Q., (Eds.). Curran Associates, Inc., 2014, pp. 2672–2680. 9

[HBVMT99] HADAP S., BANGERTER E., VOLINO P., MAGNENAT-THALMANN N.: Animating wrinkles on clothes. In *Proceedings of the Conference on Visualization '99: Celebrating Ten Years* (Los Alamitos, CA, USA, 1999), VIS '99, IEEE Computer Society Press, pp. 175–182. 2

[HDDN19] HOLDEN D., DUONG B. C., DATTA S., NOWROUZEZAHRAI D.: Subspace neural physics: Fast data-driven interactive simulation. In *Proceedings of the 18th Annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2019), SCA '19, ACM, pp. 6:1–6:12. 2, 8, 11

[HHF*19] HANOCKA R., HERTZ A., FISH N., GIRYES R., FLEISHMAN S., COHEN-OR D.: Meshcnn: A network with an edge. *ACM Trans. Graph. 38*, 4 (July 2019), 90:1–90:12. 3, 5, 7

[HPS08] HORMANN K., POLTHIER K., SHEFFER A.: Mesh parameterization: Theory and practice. In *ACM SIGGRAPH ASIA 2008 Courses* (New York, NY, USA, 2008), SIGGRAPH Asia '08, ACM, pp. 12:1–12:87. 2

[HSBH*19] HAIM N., SEGOL N., BEN-HAMU H., MARON H., LIPMAN Y.: Surface networks via general covers. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (Oct 2019). 3

[IZZE17] ISOLA P., ZHU J., ZHOU T., EFROS A. A.: Image-to-image translation with conditional adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (July 2017), pp. 5967–5976. 2

[JLGF17] JIN N., LU W., GENG Z., FEDKIW R. P.: Inequality cloth. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (New York, NY, USA, 2017), SCA '17, ACM, pp. 16:1–16:10. 6

	LV	HV	params	DL+ SubD	Hsim 8 sub	Lsim 8 sub		Lsim 8 sub	Lsim 2 sub		Lsim 1 sub	
						time	X	Errors	time	X	time	X
pants - RC13	265	4063	7902829	9.4	104.2	84.5	1.1	0.842 / 0.794	20.3	3.5	10.6	5.2
pants - RC9			5471897	5.7			1.2	1.047 / 0.972		4.0		6.4
pants - RC9, x/2			1371757	3.7			1.2	1.583 / 1.471		4.3		7.3
pants - RC9, x/4			346265	2.2			1.2	2.677 / 2.342		4.6		8.1
tshirt2 - RC13	563	8654	7902829	17.6	156.7	93.0	1.4	1.458 / 1.717	23.3	3.8	11.6	5.4
tshirt2 - RC9			5471897	10.2			1.5	1.703 / 1.791		4.7		7.2
tshirt2 - RC9, x/2			1371757	5.9			1.6	2.766 / 2.914		5.4		8.9
tshirt2 - RC9, x/4			346265	3.4			1.6	4.344 / 4.443		5.9		10.4
tank-female - RC13	969	14634	7902829	26.6	164.6	81.9	1.5	1.054 / 0.895	21.0	3.5	10.9	4.4
tank-female - RC9			5471897	15.2			1.7	1.117 / 0.972		4.5		6.3
tank-female - RC9, x/2			1371757	9.1			1.8	1.797 / 1.525		5.5		8.3
tank-female - RC9, x/4			346265	5.1			1.9	2.755 / 2.168		6.3		10.3

Table 3: *Timing and errors for our encoder-decoder networks for up-resolution. All timings are in ms, error should be multiplied by* 10^{-3}. *Time for up-res include GPU Loop subdivision and DL inference. Lsim K sub refer to low res simulation using K sub steps. Columns with X refer to the speedup factor when using DL compared to the high res simulation.*

Mesh	verts	Hsim	HC13				HC9				HC9, x/2				HC9, x/4			
			params	DL	errors	X	params	DL	errors	X	params	DL	errors	X	params	DL	errors	X
dress2	4922	113.7	7117519	10.5	1.367 / 1.593	10.8	5691453	7.0	1.558 / 1.594	16.2	2005611	3.1	2.180 / 1.780	36.7	793087	1.7	3.824 / 2.915	66.9
tank-male	3666	103.0	6595855	7.3	2.204 / 1.562	14.1	5109597	5.6	1.463 / 1.308	18.4	1714683	2.8	2.419 / 1.547	36.8	647623	1.6	2.808 / 2.396	64.4
hand	33641	800.0	5746175	31.8	0.216 / 0.215	25.1	4161877	30.1	0.327 / 0.353	26.6	1240823	12.1	0.349 / 0.373	66.2	410693	6.0	0.355 / 0.385	133.8
cape PCA 32	5430	111.7	5394655	9.3	0.898 / 0.952	12.0*	3847307	7.1	0.847 / 1.076	15.7*	1053685	3.1	1.632 / 1.699	35.6*	309383	1.6	2.970 / 3.146	69.8*
cape PCA 64			5747423	9.3	0.793 / 0.905	12.0*	4200075	7.0	0.850 / 0.976	16.0*	1230069	3.2	1.639 / 1.683	36.0*	397575	2.1	2.488 / 2.556	53.9*

Table 4: *Timing and errors for our decoder networks for regressing character poses to cloth, PCA coefficients to cloth, bone joint angles to hand skin. All times are in ms, error should be multiplied by* 10^{-3}. *Columns with X refer to the speedup factor when using DL compared to the high res simulation. For PCA, this is a speculative speedup assuming that a reduced space PCA simulation time of* 83μs *as reported for the cape mesh with 64 bases in [HDDN19]*

[JV88] JONKER R., VOLGENANT T.: A shortest augmenting path algorithm for dense and sparse linear assignment problems. In *DGOR/NSOR* (Berlin, Heidelberg, 1988), Schellhaas H., van Beek P., Isermann H., Schmidt R., Zijlstra M., (Eds.), Springer Berlin Heidelberg, pp. 622–622. 4

[JZGF18] JIN N., ZHU Y., GENG Z., FEDKIW R.: A pixel-based framework for data-driven clothing. *ArXiv abs/1812.01677* (2018). 2

[KB14] KINGMA D., BA J.: Adam: A method for stochastic optimization. *International Conference on Learning Representations* (12 2014). 6

[KCCL01] KANG Y.-M., CHOI J.-H., CHO H.-G., LEE D.: An efficient animation of wrinkled cloth with approximate implicit integration. *The Visual Computer 17* (05 2001), 147–157. 2

[KGBS11] KAVAN L., GERSZEWSKI D., BARGTEIL A., SLOAN P.-P.: Physics-inspired upsampling for cloth simulation in games. *ACM Trans. Graph. 30*, 4 (2011), 93:1–93:9. 2, 6

[KKN*13] KIM D., KOH W., NARAIN R., FATAHALIAN K., TREUILLE A., O'BRIEN J. F.: Near-exhaustive precomputation of secondary cloth effects. *ACM Trans. Graph. 32*, 4 (July 2013), 87:1–87:8. 2

[KL07] KANG M. K., LEE J.: A real-time cloth draping simulation algorithm using conjugate harmonic functions. *Computers and Graphics 31* (2007), 271–279. 2

[Kuh55] KUHN H. W.: The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly 2*, 1–2 (March 1955), 83–97. 4

[LC04] LARBOULETTE C., CANI M.-P.: Real-time dynamic wrinkles. In *Proceedings of the Computer Graphics International* (Washington, DC, USA, 2004), CGI '04, IEEE Computer Society, pp. 522–525. 2

[LCT18] LÄHNER Z., CREMERS D., TUNG T.: Deepwrinkles: Accurate and realistic clothing modeling. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part IV* (2018), pp. 698–715. 2, 6, 9

[LDCK19] LIM I., DIELEN A., CAMPEN M., KOBBELT L.: A simple approach to intrinsic correspondence learning on unstructured 3d meshes. In *Computer Vision – ECCV 2018 Workshops* (Cham, 2019), Leal-Taixé L., Roth S., (Eds.), Springer International Publishing, pp. 349–362. 2

[LMR*15] LOPER M., MAHMOOD N., ROMERO J., PONS-MOLL G., BLACK M. J.: Smpl: A skinned multi-person linear model. *ACM Trans. Graph. 34*, 6 (Oct. 2015), 248:1–248:16. 2

[LOL19] LEE T. M., OH Y. J., LEE I.-K.: Efficient cloth simulation using miniature cloth and upscaling deep neural networks. *ArXiv abs/1907.03953* (2019). 2

[MBBV15] MASCI J., BOSCAINI D., BRONSTEIN M. M., VANDERGHEYNST P.: Geodesic convolutional neural networks on riemannian manifolds. In *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)* (Dec 2015), pp. 832–840. 2

[MBM*16] MONTI F., BOSCAINI D., MASCI J., RODOLÀ E., SVOBODA J., BRONSTEIN M. M.: Geometric deep learning on graphs and manifolds using mixture model cnns. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), 5425–5434. 2

[MC10] MÜLLER M., CHENTANEZ N.: Wrinkle meshes. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Goslar Germany, Germany, 2010), SCA '10, Eurographics Association, pp. 85–92. 2, 7

[MGA*17] MARON H., GALUN M., AIGERMAN N., TROPE M., DYM N., YUMER E., KIM V. G., LIPMAN Y.: Convolutional neural networks

on surfaces via seamless toric covers. *ACM Trans. Graph. 36*, 4 (July 2017). 3

[MMC16] MACKLIN M., MÜLLER M., CHENTANEZ N.: Xpbd: Position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games* (New York, NY, USA, 2016), MIG '16, ACM, pp. 49–54. 6

[OLL18] OH Y. J., LEE T. M., LEE I.-K.: Hierarchical cloth simulation using deep neural networks. In *Proceedings of Computer Graphics International 2018* (New York, NY, USA, 2018), CGI 2018, ACM, pp. 139–146. 2

[OZM04] O'LEARY P. L., ZSOMBOR-MURRAY P.: Direct and specific least-square fitting of hyperbola and ellipses. *Journal of Electronic Imaging 13*, 3 (2004), 492 – 503. 3

[PO18] POULENARD A., OVSJANIKOV M.: Multi-directional geodesic neural networks via equivariant convolution. *ACM Trans. Graph. 37*, 6 (Dec. 2018). 2

[RPB15] RONNEBERGER O., P.FISCHER, BROX T.: U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)* (2015), vol. 9351 of *LNCS*, Springer, pp. 234–241. (available on arXiv:1505.04597 [cs.CV]). 5

[RPC*10] ROHMER D., POPA T., CANI M.-P., HAHMANN S., ALLA S.: Animation Wrinkling: Augmenting Coarse Cloth Simulations with Realistic-Looking Wrinkles. *ACM Transactions on Graphics 29*, 5 (Dec. 2010), 157. Special Issue: SIGGRAPH Asia 2010. 2

[SOC19] SANTESTEBAN I., OTADUY M. A., CASAS D.: Learning-based animation of clothing for virtual try-on. *Comput. Graph. Forum 38* (2019), 355–366. 2, 9

[SSH12] SEILER M., SPILLMANN J., HARDERS M.: Enriching coarse interactive elastic objects with high-resolution data-driven deformations. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Goslar, DEU, 2012), SCA '12, Eurographics Association, p. 9–17. 2

[TPGM19] TAN Q., PAN Z., GAO L., MANOCHA D.: Realtime simulation of thin-shell deformable materials using cnn-based mesh embedding. *ArXiv abs/1909.12354* (2019). 2

[UVL16] ULYANOV D., VEDALDI A., LEMPITSKY V. S.: Instance normalization: The missing ingredient for fast stylization. *CoRR abs/1607.08022* (2016). 5

[WSW19] WANG Z., SHE Q., WARD T. E.: Generative adversarial networks: A survey and taxonomy. *CoRR abs/1906.01529* (2019). 9