Subspace Neural Physics: Fast Data-Driven Interactive Simulation

Daniel Holden
Ubisoft La Forge, Ubisoft
Montreal, QC, Canada
daniel.holden@ubisoft.com

Bang Chi Duong
Ubisoft La Forge, Ubisoft
Montreal, QC, Canada
bangchi.duong.20193@outlook.com

Sayantan Datta
McGill University
Montreal, QC, Canada
sayantan.datta@mail.mcgill.ca

Derek Nowrouzezahrai
McGill University
Montreal, QC, Canada
derek@cim.mcgill.ca

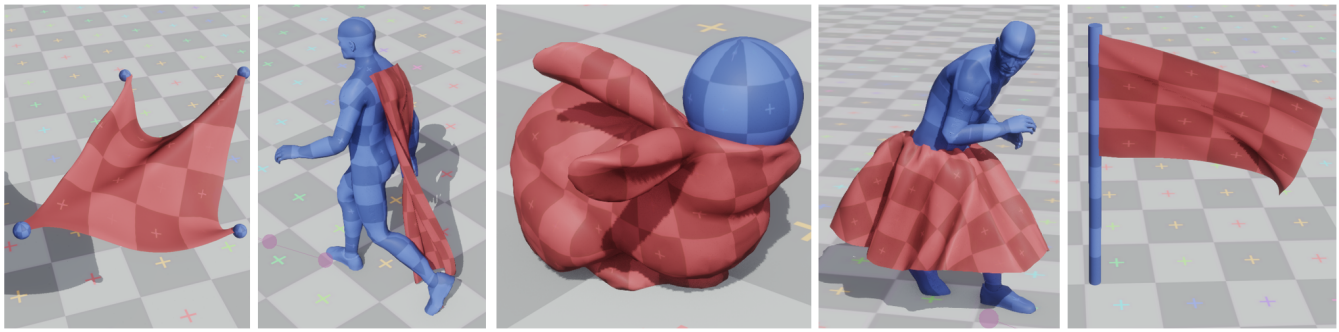

Figure 1: Our method simulates deformation effects, including external forces and collisions, 300× to 5000× faster than standard offline simulation.

ABSTRACT

Data-driven methods for physical simulation are an attractive option for interactive applications due to their ability to trade precomputation and memory footprint in exchange for improved runtime performance. Yet, existing data-driven methods fall short of the extreme memory and performance constraints imposed by modern interactive applications like AAA games and virtual reality. Here, performance budgets for physics simulation range from tens to hundreds of micro-seconds per frame, per object. We present a data-driven physical simulation method that meets these constraints. Our method combines subspace simulation techniques with machine learning which, when coupled, enables a very efficient subspace-only physics simulation that supports interactions with external objects – a longstanding challenge for existing subspace techniques. We also present an interpretation of our method as a special case of subspace Verlet integration, where we apply machine learning to efficiently approximate the physical forces of the system directly in the subspace. We propose several practical solutions required to make effective use of such a model, including a novel training methodology required for prediction stability, and a GPU-friendly subspace decompression algorithm to accelerate rendering.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Physical simulation**; *Collision detection.*

KEYWORDS

cloth simulation, collision detection, neural networks, machine learning, model reduction, data-driven simulation

ACM Reference Format:
Daniel Holden, Bang Chi Duong, Sayantan Datta, and Derek Nowrouzezahrai. 2019. Subspace Neural Physics: Fast Data-Driven Interactive Simulation. In *SCA '19:The ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA '19), July 26–28, 2019, Los Angeles, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3309486.3340245

SCA '19, July 26–28, 2019, Los Angeles, CA, USA

ACM ISBN 978-1-4503-6677-9/19/07…$15.00
https://doi.org/10.1145/3309486.3340245

1 INTRODUCTION

Many visual effects in feature films rely on realistic simulations of the interaction and motion of deformable objects. Reproducing these convincing and costly numerical simulations in the context of interactive graphics applications remains an open challenge.

Progress in this area has relied on both the increasing processing power of commodity hardware, and fast and stable simulation methods such as Position Based Dynamics (PBD) [Bender et al. 2017;

Macklin et al. 2016; Müller et al. 2007]. The performance of these methods, while significantly faster than offline methods employed in films, still often exceeds the extreme constraints of modern AAA video games and virtual reality applications: here, only a fraction of each frame's time budget is available for simulation, most often in the range of 10s to 100s of microseconds. For simulations with effects like self-collision (e.g., cloth), collisions between arbitrary geometries, or volume preserving softbody deformation, PBD methods often either require GPU acceleration or a large percentage of the CPU's capacity to maintain interactive performance. Given this gap, the latest AAA games rely on simpler, coarse-scale physical simulation with heavily simplified collisions [Vaisse 2016].

Promising avenues to improve performance in this domain include subspace simulation and data-driven methods. Subspace methods aim to perform simulation in a reduced or compressed subspace where only the relevant deformation modes are accounted for. This can lead to massive performance gains, however no current subspace approaches support interaction with external objects without resorting to costly partial deprojection into the full space. This precludes their use under such tight compute budgets. Data-driven methods trade runtime memory usage (and potentially costly preprocessing, including data acquisition) for runtime performance. Despite their promise, so far such methods have not lead to the several orders-of-magnitude performance increases (over full simulation) needed to bridge the required performance gap.

We combine subspace simulation with data-driven methods to efficiently simulate the motion and interaction of deformable objects – as such we benefit from the strengths of both of these approaches (while inheriting some of their weaknesses). One can therefore view our method in two lights: either as a subspace simulation method that efficiently resolves interactions with external objects using a data-driven function approximator (i.e., parameterized using a neural network), *or* as a data-driven method that relies on subspace simulation to build a compressed representation of the simulation state. For consistency, in the rest of this paper we adopt the second of these views, and provide further discussion on this alternative perspective in Section 8.

We present several contributions to effectively combine these two paradigms into a practical, end-to-end solution: first, we develop a novel neural network training procedure that back-propagates errors through the entire simulation integration step, leading to stable long term predictions; second, we devise several important runtime algorithms and optimizations, including efficient GPU-based decompression, an efficient method for computing vertex normals for shading, and an approach for avoiding visual interpenetration artifacts. We additionally present an interpretation of our method as a special case of subspace Verlet integration, with a machine-learned approximation of the subspace forces, which we believe is useful for motivating future work in this area.

2 RELATED WORK

We discuss work in three areas most related to our approach: fast and stable simulation methods appropriate for interactive applications, subspace techniques, and data-driven physics simulation.

Fast and Stable Simulation. One of the first fast, stable simulation methods used in a AAA game combines a simple Verlet integration scheme with a constraint projection step to simulate realistic character rag-dolls [Jakobsen 2001]. Position Based Dynamics [Müller et al. 2007] methods are unconditionally stable, using a constraint solver that limits extrapolation with an explicit/semi-implicit time integration scheme that constrains the system to only physically-valid states. Many frameworks tailored for industrial-caliber applications, such as Bullet, PhysX, Havok Cloth and Maya nCloth [Stam 2009], leverage the many extensions and improvements built atop of PBD [Kim et al. 2012; Macklin and Müller 2013; Macklin et al. 2016; Müller and Chentanez 2011; Müller et al. 2014]. Interested readers may refer to a more comprehensive review by Bender et al. [2017].

More recently Projective Dynamics [Bouaziz et al. 2014; Liu et al. 2013] has appeared as a competing method for fast and stable physics simulation. In Projective Dynamics the constraint projection step is decoupled into local and global steps which are solved separately. The local step is performed independently for each constraint and hence can be more easily parallelized. Due to this it has quickly been applied to many domains such as cloth, elastic body simulation, and fluids [Narain et al. 2016; Weiler et al. 2016]. However, in both Position Based Dynamics and Projective Dynamics the computational cost usually scales with the number of vertices and constraints, which can grow large for high resolution cloth. This can make it inaccessible within the time constraints imposed by games, in particular when these constraints become expensive to compute such as self-collisions and collisions on arbitrary meshes.

Subspace Simulation. Subspace, or model reduction methods project the equations of motion into a reduced subspace with the hope of solving them more efficiently. This can work particularly well in constrained systems, where one could expect the effective degrees of freedom of a system to be far smaller than the dimensionality of the full state space. Many works explore the construction of efficient and effective subspace bases [Harmon and Zorin 2013; Huang et al. 2011; Sifakis and Barbic 2012; von Radziewsky et al. 2016; von Tycowicz et al. 2013; Wang et al. 2015; Yang et al. 2015, 2013], however a standard PCA basis remains an effective choice (for a linear basis) in the presence of a representative set of deformation examples [Barbič and James 2005; Treuille et al. 2006]. One interesting extension to this is presented by Fulton et al. [2019] who build a more compressed non-linear subspace on top of the PCA subspace using an auto-encoder to accelerate the solution of the implicit integration step of the simulation.

Given a basis, projecting the equations of motion is simple, but incorporating other operations can be challenging. Previous work has focused on applying these methods to elastic deformation and FEM solvers [An et al. 2008; Barbič and James 2005; Kim and James 2009; Pan et al. 2015], cloth simulation [Hahn et al. 2014], articulated characters [Galoppo et al. 2007, 2009; Kry et al. 2002; Xu and Barbič 2016], and collisions [Teng et al. 2015, 2014].

These approaches are difficult to incorporate with PBD-based methods due to the difficulty in formulating the PBD constraint projection as a subspace operator and so implicit Euler integration remains a popular subspace integration scheme. However, Brandt et al. [2018] apply subspace methods to Projective Dynamics by leveraging a separate subspace for the constraints and sampling

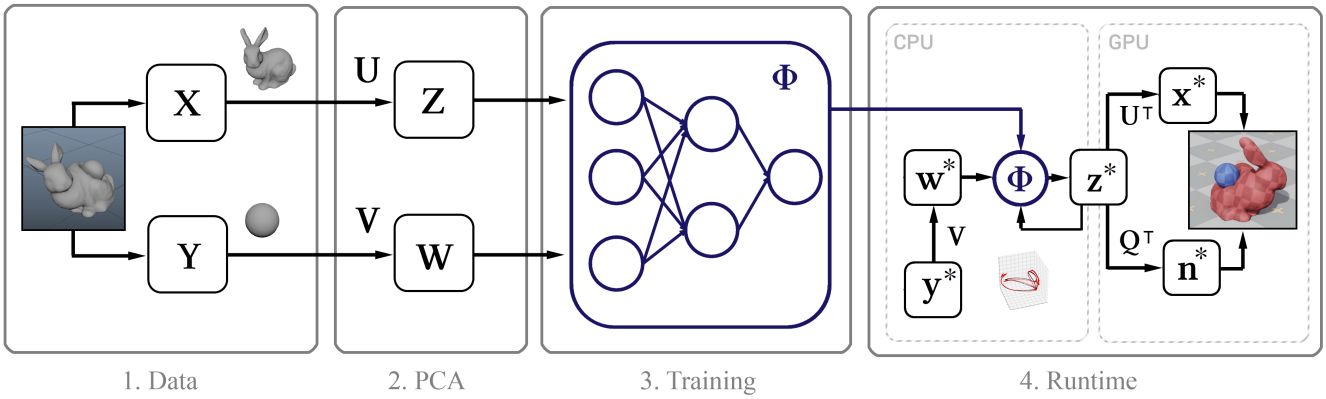

Figure 2: Overview – we acquire training data X and Y offline using Maya's nCloth, and perform PCA to obtain compressed representations Z and W, before training a neural network Φ to recurrently predict the compressed state of the object z^*, given the previous state of object z^*_{t-1}, and the compressed state of external objects w^*. Simulation object positions x^* and normals n^* are computed directly from the models output, for rendering.

constraint projections to approximate their subspace representation with a least squares fit.

Handling internal and external collisions efficiently is particularly challenging for subspace methods due to the geometric operations involved. Teng et al. [2015; 2014] manage self-collisions by applying forces on a sparse set of deprojected simulation points. They support external collisions by allowing partial, albeit costly, full-space simulation in areas of the mesh involving collisions. We propose an alternative solution to efficiently compute the effects of such external objects *directly and entirely* in the subspace (i.e., without any deprojection during simulation.)

Data-Driven Simulation. Data-Driven methods use data, typically precomputed offline using accurate simulation techniques, to ideally enhance, accelerate or approximate physical simulations at runtime. This is popular in character clothing due to the cost of resolving collisions with the cloth and character. Kim et al. [2013] precompute and compress all possible secondary cloth motions on a character and efficiently query this database at runtime. Xu et al. [2014] use example cloth shapes to deform a character's cloth based on its pose. Wang et al. [2010] similarly layer wrinkles on top of a coarse simulation mesh and while these methods produce high quality results, they tend to have large memory footprints as much of the training data must be kept resident. Luo et al. [2018] propose a neural network specialized in modelling non-linear deformation for full-space simulated objects while Edilson de Aguiar et al. [2010] instead rely on PCA, as we do, to compute a cloth subspace and train a simple linear model of the cloth dynamics within this subspace. Although their performance and memory usage is similar to ours, their approach does not scale to more complex deformations as their linear model is too simple.

Data-driven methods have also been applied to fluid simulation [Kim et al. 2018; Wiewel et al. 2018]. Regression forests can predict the movement of fluid particles based on global information, such as spatially-varying pressure [Ladicky et al. 2017; Ladický et al. 2015]. Since these models predict individual particle velocities, an efficient GPU implementation is required to achieve the best performance. Data-driven methods can also be employed selectively to accelerate components of the physics pipeline: Thompson et al. [2016] use a highly specialized deep neural network to solve the incompressible Euler equations of a fluid simulation.

Alternatively, one can use data-driven methods to add fine details to coarse-scale simulations. Chu et al. [2017] parameterize a feature descriptor using a Convolutional Neural Network (CNN) to match simulation datasets, whereas Xie et al. [2018] use Generative Adversarial Networks for fluid flow super resolution. Jin et al. [2018] use texture coordinates and a pixel based representation for producing the cloth's wrinkles via a CNN. More recently, Lahner et al. [2018] combine a coarse simulation with a Generative Adversarial Network which adds high frequency details such as wrinkles to cloth. Unfortunately, evaluating CNNs is expensive and memory intensive, and so it is difficult for these techniques to reach the multiple orders-of-magnitude gains needed for incorporation in modern game engines (even with GPU acceleration).

The inherent challenge of data-driven methods typically lies in striking a balance between runtime performance, memory usage, accuracy, and model capacity/expressiveness. Ours is the first data-driven method for deformable objects that performs well in all of these domains.

3 OVERVIEW

Fig 2 outlines our method: we first collect high-quality simulation data using Maya's nCloth before computing a linear subspace using PCA. We then devise a machine learning approach, including a neural network model and a novel training methodology. We integrate this model into an interactive runtime algorithm that includes several optimizations, such as an efficient GPU decompression algorithm and a vertex normal approximation method.

4 TRAINING DATA

Generally speaking, almost any simulation method is suitable for acquiring the data for our method, as the only input to our training procedure is a raw time-series of frame-by-frame vertex positions. We detail the exact data acquisition process we use for our results.

Table 1: Training data acquisition parameters and timings.

Scene	Material	#Verts	#Frames	FPS	Time
Ball & Sheet	T-Shirt	2601	1,000,000	7.6	36h
Four Pins	T-Shirt	2601	1,000,000	15.5	18h
Flag	T-Shirt	2601	1,000,000	10.9	25h
Skirt	Denim	3000	650,000	3.1	60h
Cape	T-Shirt	2601	650,000	1.9	95h
Bunny	Rubber	2503	200,000	0.4	129h
Dragon	Rubber	3000	500,000	1.0	138h

We perform all simulation using Maya's nCloth, capturing data at 60 frames per second, with between 5 and 20 substeps and 10 and 25 constraint iterations, depending on simulation stability. For cloth-like objects we primarily use the T-Shirt material preset with small increases in weight and stretch resistance. For deformable objects we use the solid rubber material preset with reduced friction to allow objects to slide easily over the surface. We perform external collisions against the triangles of the external geometry, while self-collisions use vertex-on-vertex collisions for cloth, and triangle-on-triangle collisions for deformable objects. In both cases we use a fairly large collision thickness of $\sim$ 5 cm to ensure stability and prevent the cloth snagging and breaking during simulation. This additionally allows for some leniency during prediction without immediate visual intersection artifacts appearing (see Section 6.4).

For simple interaction objects (e.g., pins, spheres) we generate their movement in the training data randomly by keyframing random positions at random times to produce different kinds of interaction. For cloth-character interactions we use a large motion capture database of $\sim 6.5 \times 10^5$ animation frames, stitched together to form one large animation. We then simulate the entire series. After simulation we check the data and exclude any frames where unstable or bad behavior was seen. For the skirt scene we remove the character's arms because they intersect frequently with the leg mesh geometry, causing the cloth to break.

Generally, we attempt to acquire between 10^5 and 10^6 frames of training data. We found in most cases $\sim 10^5$ was sufficient for testing, while the best results were achieved with closer to $\sim 10^6$ frames. For further details on the data acquisition please see Table 1.

5 TRAINING

We discuss how we train our model, including its parameterization, the network architecture, and our training methodology.

5.1 Parameterization

Given the simulation data gathered in the previous section, we construct our training set by first flattening the vertex positions at each frame t into a single, large vector $\mathbf{x}_t \in \mathbb{R}^{3c}$, where c is the number of vertices. We then concatenate these vectors into a single large matrix $\mathbf{X} = [\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_{n-1}] \in \mathbb{R}^{3c \times n}$. This matrix represents the states of the simulated object, after which we must build a representation of the states of the external objects at each frame. For simple objects, such as balls, we can use their 3D position; while, for complex objects like full characters, we use the positions of every joint relative to a reference fame (in the case of the skirt we use the hip joint as the reference frame, and in the case of cape we use the neck joint as the reference frame) flattened into a large vector, ignoring the joint rotations. For objects with a moving reference frame, we also include the position of the ground relative to this frame so our system knows the gravity direction and floor location as well as the velocity, acceleration, rotational velocity, and rotational acceleration of this frame. For the flag, we include wind speed and direction. After building this parameterization, we have a single large vector representing the state of the external objects for each frame $\mathbf{y}_t \in \mathbb{R}^e$, where e is the number of degrees of freedom of the external objects, which we also concatenate into a single large matrix $\mathbf{Y} = [\mathbf{y}_0, \mathbf{y}_1, ..., \mathbf{y}_{n-1}] \in \mathbb{R}^{e \times n}$.

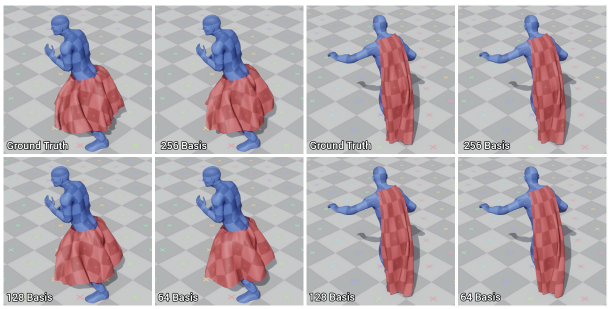

Figure 3: Basis size impact on complex geometry and dynamics. There is a subtle loss of finer wrinkles as we reduce the size of the basis. The skirt is the most challenging to compress, suffering from more substantial degradation.

We now apply PCA to both $\mathbf{X}$ and $\mathbf{Y}$ and use the computed transformation matrices to construct subspace representations $\mathbf{Z} = \mathbf{U}(\mathbf{X} - \mathbf{x}_\mu)$, $\mathbf{W} = \mathbf{V}(\mathbf{Y} - \mathbf{y}_\mu)$, where $\mathbf{U} \in \mathbb{R}^{u \times 3c}$, $\mathbf{V} \in \mathbb{R}^{v \times e}$, u is the number of subspace bases (in our results we use 64, 128 and 256), v is the number of bases used to compress the external object representation. Here, $\mathbf{x}_\mu$ is the mean value of all $\mathbf{x}$'s, and $\mathbf{y}_\mu$ is the mean value of all $\mathbf{y}$'s. As we require no compression of the parameterization for the external objects in our examples, we typically set $v = e$. If the memory usage is too large to perform PCA we subsample the data before applying it.

PCA compression inevitably causes a loss in detail, particularly for objects with many potential states, such as fine folds; however, we found that 256 bases generally preserved the majority of details. For a visual comparison, see Fig 3.

5.2 Initial Model

Given subspace data $\mathbf{Z}$ and $\mathbf{W}$, our goal is to devise a model capable of predicting $\mathbf{z}_t$ from $\mathbf{z}_{t-1}$, $\mathbf{z}_{t-2}$ and $\mathbf{w}_t$. Since simulated objects generally express inertia, with a tendency toward some average rest state (represented by zero after PCA), a good initial model for $\mathbf{z}_t$ (denoted here as $\bar{\mathbf{z}}_t$) is:

$$\bar{\mathbf{z}}_t = \boldsymbol{\alpha} \odot \mathbf{z}_{t-1} + \boldsymbol{\beta} \odot (\mathbf{z}_{t-1} - \mathbf{z}_{t-2}), \tag{1}$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are parameters of the model and $\odot$ is component-wise multiplication. We obtain the values of these parameters by solving a linear least squares equation individually for each dimension of α and β, denoted by m:

$$\begin{bmatrix} \alpha_m & \beta_m \end{bmatrix} = \begin{bmatrix} \mathbf{z}_{t,m} \end{bmatrix} \begin{bmatrix} \mathbf{z}_{t-1,m} \\ \mathbf{z}_{t-1,m} - \mathbf{z}_{t-2,m} \end{bmatrix}^\dagger, \tag{2}$$

with $t \in [2, n)$ and where $\dagger$ denotes the matrix pseudo-inverse.

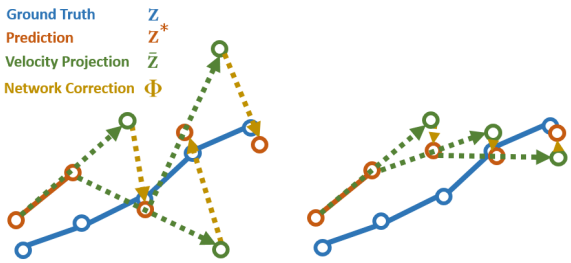

Figure 4: Visual illustration of our training method – standard procedures target the most accurate result on a per-frame basis (left). These predictions are unstable due to high velocities produced by aggressive over-correction. Our method targets an accurate prediction across an window of frames (right), resulting in less aggressive and more stable corrections.

5.3 Extended Model

Since $\bar{\mathbf{z}}_t$ is only a very rough approximation of $\mathbf{z}_t$, and does not take into account the effects of external objects $\mathbf{w}$, we know it will not be capable to accurately model our training data. We therefore train a neural network Φ to approximate the residual effects of the model, such that:

$$\mathbf{z}_t = \bar{\mathbf{z}}_t + \Phi\left(\begin{bmatrix}\bar{\mathbf{z}}_t & \mathbf{z}_{t-1} & \mathbf{w}_t\end{bmatrix}^{\mathrm{T}}\right) \tag{3}$$

Here, we parameterize Φ by a standard feed-forward neural network with 10 layers, each layer (except the output layer) using the ReLU activation [Nair and Hinton 2010]. Excluding the input and output layers we set the number of hidden units at each layer to 1.5× the PCA basis size, which we found struck a good trade-off between capacity and performance.

5.4 Network Training

The standard way of training Φ would be to iterate over the dataset in mini-batches and to train the network to predict the value $\mathbf{z}_t - \bar{\mathbf{z}}_t$ for all t. While this approach will produce a low training error, the auto-recurrent nature of Φ, and the coupled velocity step of Equation (1) results in unstable behaviour when predictions are fed back into the network at the next time step. Due to this, previous work has proposed to feed back the prediction into the neural network at the next time-step for correction [Dollár et al. 2010; Kanazawa et al. 2017; Oberweger et al. 2016]. Inspired by this, we present a training algorithm which predicts motion over a window of frames, and back-propagates errors through the complete integration procedure described by Equation (3) to ensure stable long term prediction. For a detailed algorithmic explanation please see Algorithm 1.

At a high level our training procedure is as follows: given a small window of values for $\mathbf{z}$ and $\mathbf{w}$ from the training set, we take the first two frames $\mathbf{z}_0$, $\mathbf{z}_1$ and add some small noise $\mathbf{r}_0$, $\mathbf{r}_1$ to perturb them slightly off the training trajectory. From these initial states, we repeatedly use Equation (1) and Equation (3), to predict the following frames, feeding back in previous predictions at each new time step. Once the full trajectory is predicted, we compute the average positional error and velocity error across the window of motion. We pass this error to the AmsGrad optimizer [Reddi et al. 2018], using automatic derivatives computed from TensorFlow.

Algorithm 1: Our training algorithm for Φ. Given a short window of s frames, we predict the subspace state of the physical object, and use the error to update the network parameters $\boldsymbol{\theta}$. While we present this procedure for a single training sample here, we apply it to each element in the mini-batch individually and average the result when updating $\boldsymbol{\theta}$.

Function Train($\mathbf{z}$, $\mathbf{w}$, s, $\boldsymbol{\theta}$):
- /* Sample two noise vectors $\mathbf{r}_0$, $\mathbf{r}_1$ */
- $\mathbf{r}_0,\ \mathbf{r}_1 \sim \mathcal{N}(0, r^\sigma)$
- /* Add noise to initial states $\mathbf{z}_0$, $\mathbf{z}_1$ */
- $\mathbf{z}^*_0,\ \mathbf{z}^*_1 \leftarrow \mathbf{z}_0 + \mathbf{r}_0,\ \mathbf{z}_1 + \mathbf{r}_1$
- /* Predict values of $\mathbf{z}^*$ over a short window s */
- **for** $i \leftarrow 2$ **to** s **do**
  - /* Predict $\bar{\mathbf{z}}^*_i$ using $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ */
  - $\bar{\mathbf{z}}^*_i \leftarrow \boldsymbol{\alpha} \odot \mathbf{z}^*_{i-1} + \boldsymbol{\beta} \odot (\mathbf{z}^*_{i-1} - \mathbf{z}^*_{i-2})$
  - /* Predict $\mathbf{z}^*_i$ using Φ */
  - $\mathbf{z}^*_i \leftarrow \bar{\mathbf{z}}^*_i + \Phi\left(\begin{bmatrix}\bar{\mathbf{z}}^*_i & \mathbf{z}^*_{i-1} & \mathbf{w}_i\end{bmatrix}^{\mathrm{T}}; \boldsymbol{\theta}\right)$
- **end**
- /* Compute Loss $\mathcal{L}$ using Mean Absolute Error */
- $\mathcal{L}_{pos} \leftarrow \left\| \mathbf{z}^*_{2\to s} - \mathbf{z}_{2\to s} \right\|_1$
- $\mathcal{L}_{vel} \leftarrow \left\| \frac{\mathbf{z}^*_{2\to s} - \mathbf{z}^*_{1\to(s-1)}}{dt} - \frac{\mathbf{z}_{2\to s} - \mathbf{z}_{1\to(s-1)}}{dt} \right\|_1$
- $\mathcal{L} \leftarrow \mathcal{L}_{pos} + \mathcal{L}_{vel}$
- /* Update network parameters $\boldsymbol{\theta}$ */
- $\boldsymbol{\theta} \leftarrow \mathrm{AmsGrad}(\boldsymbol{\theta}, \nabla\mathcal{L})$

end

To motivate this training procedure, consider the case where the network predicts a large change in position from one time step to the next. This difference will be used in the next time step by Equation (1) to produce a large initial guess which, itself, may require aggressive correction from the network back toward the training data. As this is repeated, the correction sizes increase and the prediction becomes unstable. If we, however, train over a window of frames, the network will only produce corrections that improve the result over the entire window, encouraging smaller, more stable corrections. Another way to view our training procedure is as a variant of Back-Propagation Through Time [Rumelhart et al. 1986], where errors are not only back-propagated through the network but also through the integration step of Equation (1). Fig 4 provides visual intuition.

We repeat this training procedure on mini-batches of size 16, using overlapping windows of size 32, for around 100 epochs or until training converges. We use a learning rate of 0.0001, and a learning rate decay ratio of 0.999. We use a noise standard deviation r^σ of 0.01, which we found by visualizing the result of this perturbation in the first 3 components of the PCA space. Training takes between 10 and 48 hours depending on the complexity of the setup and the number of PCA bases used. We include more discussions on the importance of this training method in Section 7.2.

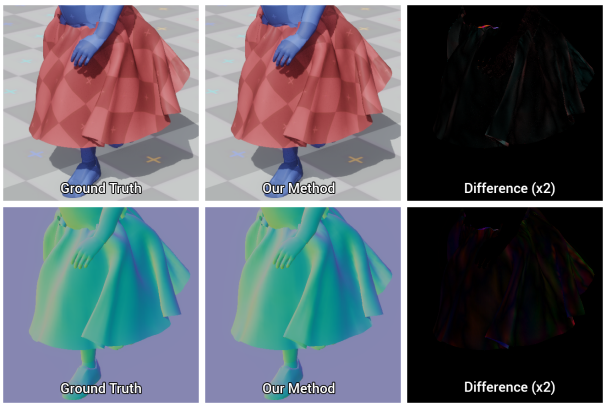

Figure 5: Comparison of our normal computation method against the ground truth. In most cases our result produces extremely small differences.

6 RUNTIME IMPLEMENTATION

We detail the runtime implementation of our method in an interactive environment, including the evaluation of the neural network, how we compute the normals of the object surfaces for rendering, and the technique we use to prevent visible intersections.

6.1 Interactive Application

We render the results of our method in a simple interactive 3D application written in C++ and DirectX. We re-implement the preprocessing and neural network operations in single-threaded C++ code and load the binary network weights obtained during our training procedure. We perform some simple optimisations to the network evaluation, including the re-use of memory buffers and sparse vector-matrix products that are possible due to the zero-valued hidden units produced as a consequence of ReLU activations.

For the cape and skirt results, we implement a basic character controller using Motion Matching [Clavet 2016] and allow the user to dynamically control the character with a gamepad. The data we use in the motion matching includes motion clips that were present in the training data. Most other user interaction is simply enabled by allowing the user to manipulate the interaction object(s) with a mouse, or sliders on the user interface.

6.2 GPU Decompression

Since Equation (3) expects the compressed state $\mathbf{z}$ as input, the only place the full object state $\mathbf{x}$ is required is for rendering. We can therefore send the compressed cloth state $\mathbf{z}$ to the GPU, and perform decompression only at render time. To do so, we use a simple GPU compute shader which, for each object vertex, computes the dot product of $\mathbf{z}$ and the three rows of matrix $\mathbf{U}^{\mathrm{T}}$, corresponding to the x, y and z components of that vertex's position, before finally adding the mean value $\mathbf{x}_\mu$. This approach has two advantages over a naïve CPU decompression method: first, GPU parallelism greatly accelerates the computation of $\mathbf{x}$, which we found could take up to 1 ms on the CPU; second, it reduces the GPU-CPU memory transfer by an order-of-magnitude, which is important on platforms where transferring the entire object state becomes prohibitively slow.

Figure 6: We fit capsules to each joint of the character and use them as collision geometry to adjust for visible intersections in the final render. A typically example of this fix is shown above.

6.3 Vertex Normal Prediction

At render time it is not sufficient to only have access to vertex positions, as the deformed vertex normals are also needed for rendering. Previous subspace simulation methods either omit this computation or perform a naïve re-calculation of per-face normals (each frame) followed by a distribution to neighboring vertices. This can be inefficient: we found that a basic CPU implementation requires ~ 150 μs (in addition to the CPU decompression and memory transfer costs). While it is possible to perform this computation on the GPU, it can be difficult to implement efficiently as it requires performing parallel random access writes.

Instead, similar to the method presented in James et al. [2002], we propose *learning* a linear regression from the subspace state to the full state's normal vectors, and we perform this regression on the GPU compute shader alongside the vertex position computation. Given the vertex normals at each frame, flattened into one large vector and concatenated together $\mathbf{N} = [\mathbf{n}_0, \mathbf{n}_1, ..., \mathbf{n}_n] \in \mathbb{R}^{3c \times n}$, we can find the matrix $\mathbf{Q}$ that best maps from the subspace representation $\mathbf{Z}$ to the vertex normals using the following equation: $\mathbf{Q} = (\mathbf{N} - \mathbf{n}_\mu)\mathbf{Z}^\dagger$, where $\mathbf{n}_\mu$ is the mean of $\mathbf{n}$ for all t. Once computed, $\mathbf{Q}^{\mathrm{T}}$ can be used in the same way as $\mathbf{U}^{\mathrm{T}}$ to predict the vertex normals of each vertex, by taking the dot product of the subspace state with the three columns of $\mathbf{Q}^{\mathrm{T}}$ that correspond to the vertex normal, adding the mean $\mathbf{n}_\mu$ and re-normalizing.

Since the subspace representation was not constructed with normal prediction in mind, there is no guarantee that this method of normal prediction will be accurate, however we found in practice it yielded accurate enough results. For a visual comparison, please see Fig 5.

One limitation to our method is that the computational cost of this technique grows with the number of bases we use, and so we expect that a GPU-accelerated implementation of the standard per-face distribution method could be more efficient when large enough basis terms are employed.

6.4 Avoiding Visible Intersections

Our method learns to efficiently perform collisions, however, due to errors induced by the subspace compression and inaccuracies in the prediction result, visible intersection artifacts between the external objects and the simulation objects may still occur. Moreover, since we defer the computation of the full state $\mathbf{x}$ until just prior to rendering, we leave no opportunity to efficiently address these artifiacts (e.g., geometrically, on the CPU). In the spirit of maintaining high performance, we must resolve these intersections at render time.

Previous work, such as Edilson de Aguiar et al. [2010], solve this problem using a depth bias when writing fragments, even when they intersect with geometry of labelled character components. While this would work in our case, we propose an alternative, simple and efficient solution that instead projects intersecting vertices onto the surface of simple proxy collision objects representing the character. This projection is simple to evaluate on the GPU with our existing compute shader that decompresses the cloth state and computes shading normals.

First, we build a proxy collision object for the articulated character by fitting capsules (with varying start and end radii) to all the vertices associated to each character joint (see Fig 6). Once fit, we pass the capsule start/end positions and radii to the GPU decompression compute shader. Here, we additionally test the (decompresed) vertex positions for intersection against every relevant capsule and, if an intersection is found, we project the vertex back onto the capsule surface. We only adjust the position of the vertex, leaving the computed normal unmodified to not affect the shading.

Providing vertex displacement errors generated during simulation are not significant enough to result in projection onto the "backside" of a capsule, our capsule projection pass removes small visible intersections from our final results. For a visual example please see Fig 6.

7 RESULTS AND EVALUATION

We test our method on a variety of scenes with different deformations and interactions with external forces and objects some of which are shown in Fig 1. For visualizations of all scenes tested please see supplementary video. The scenes in which we apply our method include a hanging sheet interacting with a user-controllable ball, a user manipulating the pinned corners of a deformable sheet, a flag on a pole where the user can move the pole or adjust the wind speed and direction, a cape and a skirt attached to an animating character controlled dynamically by the user, demonstrating generalization as well as self and character-collisions, a deformable bunny with a user-controlled ball that can squash and push the object, and a deformable dragon perturbed by a moving teapot. In all examples we produce natural deformation behavior. In Fig 11 we stress tests our method on scenes with hundreds of bunnies (left) and 16 characters (right), each simulated independently at framerates of 120 FPS and 240 FPS.

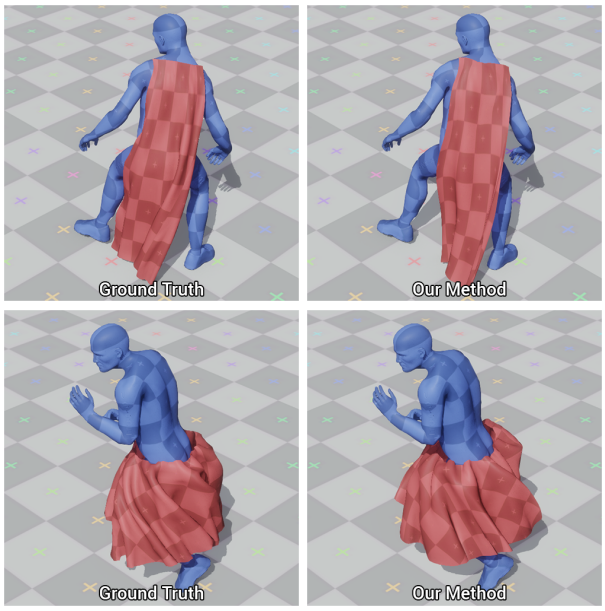

Figure 7: Generalization: our method versus ground truth from a test set.

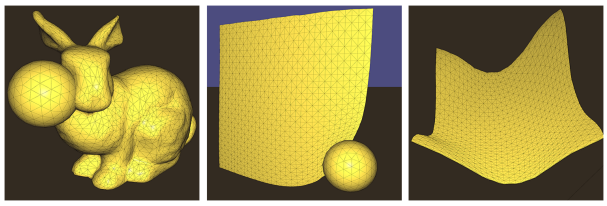

Figure 8: Applying [Brandt et al. 2018] to a selection of our test scenes. Although it produces compelling elastic deformation, this method struggles to produce accurate deformations resulting from collisions.

7.1 Evaluation

We evaluate the accuracy and performance of our method, including comparisons to ground truth data (from a held-out test set), the state-of-the-art in subspace methods [Brandt et al. 2018], and some alternative machine learning models we implement as baselines (including common recurrent models such as LSTM and GRU networks). We also present the impact of PCA basis size on our results. Please refer to the supplemental video for full visual comparisons.

Comparisons. Fig 7 compares our method to ground truth from a held-out test set. Although we lose some detail by subspace compression and network approximation, our method generalizes well and produces realistic results visually similar to ground truth.

Fig 9 illustrates how our method performs when we adjust the number of PCA bases. Users can easily trade quality for performance by adjusting the number of bases and network size. See Table 2 for more detailed performance metrics.

Fig 8 applies the method of Brandt et al. [2018] to some of our scenes: while producing natural deformation, it struggles to accurately capture deformations resulting from collisions, particularly when such deformations are not present in the subspace basis.

In Fig 10 we visualize our method's prediction in the subspace of the first three PCA bases. We see that, even when our method is not entirely accurate, it still often produces motions with similar shape and timing profiles to that of the ground truth.

Performance. One of the key strengths of our method is its performance - both in runtime speed and memory usage. In Table 2 we compare our method numerically against other methods and ground truth data taken from a test set. Our method achieves speedups ranging from $\sim 300\times$ to $\sim 5000\times$ over the raw simulation used to gather the training data. It also has good performance when compared to other state-of-the-art methods such as HRPD [Brandt et al. 2018]. Compared neural network structures are designed to be as similar as possible in size and memory use. All performance

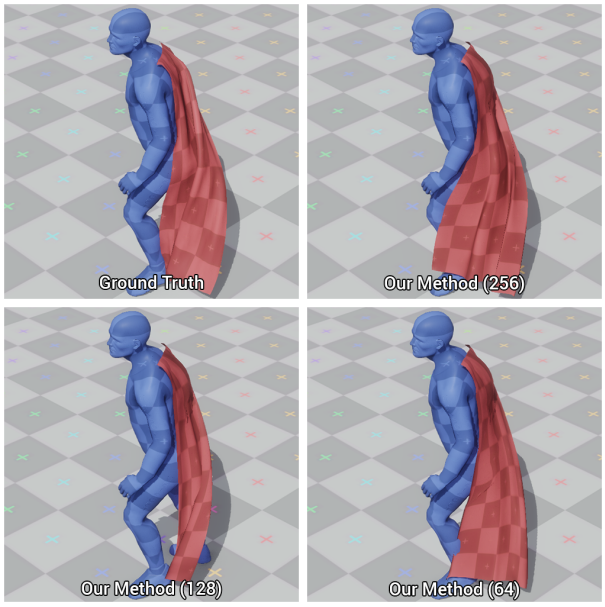

Figure 9: Our method when using different sizes of PCA basis.

measurements are made on an Intel Xeon E5-1650 3.5 GHz CPU single threaded, and a GeForce GTX 1080 Titan GPU.

7.2 Ablation Study

In this section we perform an ablation study to observe the effects of removing various components of our system. We test these ablations across several models including Linear Regression [de Aguiar et al. 2010]), Radial Basis Functions (RBF), Feed-forward Neural Networks (Our Method), and recurrent models (RNN, GRU, and LSTM Networks).

State Difference Prediction. We propose to train a model which in essence predicts the delta between two time steps. If instead we predict the absolute value of the next time step we observe stable prediction behaviour across all models (with the exception of Linear Regression), however, the predictions are very inaccurate, and the observed movement is stiff with many inter-penetrations visible. We found this issue to be particularly bad for RBFs, which also suffer from poor runtime performance and memory usage when using large amounts of training data.

Training Procedure. If we predict the delta between time steps we find the system is more accurate and less stiff, however, without our training procedure (instead using the mean squared error computed at each frame independently) we find all models to be extremely unstable during runtime prediction (see Fig 12).

Initial State Noise. We add some noise to the initial states during our training procedure. We observe that this encourages our network to learn to correct small errors, driving the prediction towards the training data when it diverges – increasing stability. Although we found dropout [Gal and Ghahramani 2016; Srivastava et al. 2014] also able to produce a similar regularizing effect, we found our solution was more effective in improving the stability of predictions.

Initial Model. In Section 5.2 we present a simple linear model used as an initial prediction of the next time step. We found omitting this model gave less accurate results and less stability. One reason for this increased stability is that the $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ parameters essentially provide a way of damping the velocity and pulling the system toward the rest state when it is far from the training data.

Recurrent Model. When using recurrent models such as RNNs, GRUs, and LSTMs trained with our loss function we observed no large differences in quality to our proposed Feed-forward Neural Network structure – however for these models the training time is often long, and the hidden state initialization needs to be handled carefully. Given that physical systems can be fully described by their position and velocity (for plastic materials, the rest state and rest state velocity are also required), we also expect that the "memory" provided by recurrent models is not required for our problem. We therefore opt for the simpler Feed-forward model in our proposed method.

8 SUBSPACE VERLET INTEGRATION

One way to interpret our method is as a special case of elastic subspace Verlet integration where an efficient approximation of the non-linear subspace forces is used. To derive this interpretation we start with a standard Verlet forward integration method described as follows:

$$\mathbf{x}_t = \mathbf{x}_{t-1} + (\mathbf{x}_{t-1} - \mathbf{x}_{t-2}) + dt^2\mathbf{M}^{-1}f_{t-1}, \tag{4}$$

where $\mathbf{M}^{-1}$ is the inverse mass matrix, f are the forces, and dt is the time-step. Depending on the model used, typically the forces f are split into separate terms such as internal forces $f_{int}(\mathbf{x})$, external forces $f_{ext}(\mathbf{x}, \mathbf{y})$, constant forces such as gravity f_{grav}, and other forces such as Coriolis forces for objects in moving reference frames. If we assume an extremely simple linear elastic system centered around the rest state represented by the zero vector (this can be constructed by subtracting the actual rest state from all other states), we can introduce constant *stiffness* and *damping* matrices $\mathbf{K}$ and $\mathbf{V}$ and split the force term into linear elastic forces and other non-linear forces $\tilde{f}$ as follows:

$$f_t = -\mathbf{K}\mathbf{x}_t - \mathbf{V}\frac{\mathbf{x}_t - \mathbf{x}_{t-1}}{dt} + \tilde{f}_t \tag{5}$$

which can then be inserted into Equation (4) and factorized to obtain

$$\mathbf{x}_t = (\mathbf{I} - dt^2\mathbf{M}^{-1}\mathbf{K})\mathbf{x}_{t-1} + (\mathbf{I} - dt\mathbf{M}^{-1}\mathbf{V})(\mathbf{x}_{t-1} - \mathbf{x}_{t-2}) + dt^2\mathbf{M}^{-1}\tilde{f}_{t-1}, \tag{6}$$

which can be simplified by letting $\mathbf{A} = \mathbf{I} - dt^2\mathbf{M}^{-1}\mathbf{K}$, $\mathbf{B} = \mathbf{I} - dt\mathbf{M}^{-1}\mathbf{V}$, and $\mathbf{C} = dt^2\mathbf{M}^{-1}$:

$$\mathbf{x}_t = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{B}(\mathbf{x}_{t-1} - \mathbf{x}_{t-2}) + \mathbf{C}\tilde{f}_{t-1}. \tag{7}$$

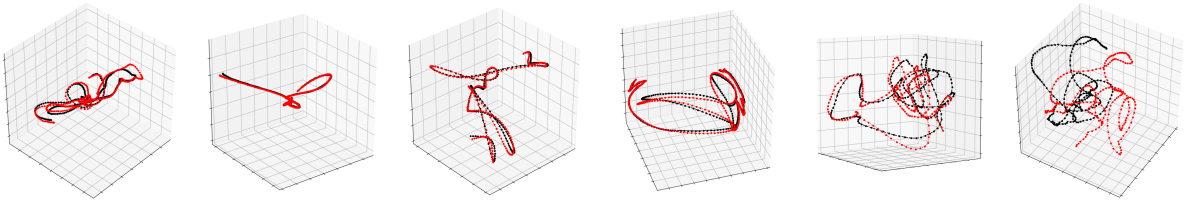

Figure 10: Visualizing our method's first three PCA bases predictions (black) versus ground truth (red), on an unseen test set. Left to right: Bunny, Ball & Sheet, Four Pins, Flag, Skirt and Cape.

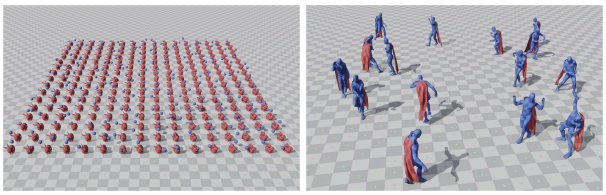

Figure 11: Left: simulation of 256 deformable bunnies using 64 bases at ~120 FPS. Right: simulation of 16 dancers using 256 bases at ~240 FPS.

Given a orthogonal subspace matrix $\mathbf{U}$ such as that constructed by PCA, we can produce a subspace version of this equation by multiplying all terms by $\mathbf{U}$,

$$\mathbf{U}\mathbf{x}_t = \mathbf{U}\mathbf{A}\mathbf{x}_{t-1} + \mathbf{U}\mathbf{B}(\mathbf{x}_{t-1} - \mathbf{x}_{t-2}) + \mathbf{U}\mathbf{C}\tilde{f}_{t-1}, \tag{8}$$

and by letting $\mathbf{z} = \mathbf{U}\mathbf{x}$, $\hat{\mathbf{A}} = \mathbf{U}\mathbf{A}\mathbf{U}^{\mathrm{T}}$, $\hat{\mathbf{B}} = \mathbf{U}\mathbf{B}\mathbf{U}^{\mathrm{T}}$, and $\hat{\mathbf{C}} = \mathbf{U}\mathbf{C}$ we get the following:

$$\mathbf{z}_t = \hat{\mathbf{A}}\mathbf{z}_{t-1} + \hat{\mathbf{B}}(\mathbf{z}_{t-1} - \mathbf{z}_{t-2}) + \hat{\mathbf{C}}\tilde{f}_{t-1}. \tag{9}$$

The result is an equation where the first two terms are much cheaper to compute than before as the dimension of $\mathbf{z}$ is far smaller than that of $\mathbf{x}$, but with a force calculation term $\tilde{f}_{t-1}$ which is potentially still expensive as it is a function of $\mathbf{x}$ and $\mathbf{y}$. Previous work has therefore often been focused on finding methods for computing subspace forces in a more efficient way, and many solutions have been found for particular force terms such as internal forces [Barbič and James 2005], Coriolis forces [Kim and James 2011], and external and internal forces resulting from collisions [Teng et al. 2015, 2014].

We can observe that because each basis in $\mathbf{U}$ is constructed to be statistically independent we expect $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ in this case to be close to diagonal matrices (see Appendix A). Then, if we discard the non-diagonal entries the multiplication can be written as a component-wise vector multiplication with diagonal entries given as $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$:

$$\mathbf{z}_t = \boldsymbol{\alpha} \odot \mathbf{z}_{t-1} + \boldsymbol{\beta} \odot (\mathbf{z}_{t-1} - \mathbf{z}_{t-2}) + \hat{\mathbf{C}}\tilde{f}_{t-1}, \tag{10}$$

and finally by replacing $\hat{\mathbf{C}}\tilde{f}_{t-1}$ with Φ we can recover Equation (3):

$$\mathbf{z}_t = \boldsymbol{\alpha} \odot \mathbf{z}_{t-1} + \boldsymbol{\beta} \odot (\mathbf{z}_{t-1} - \mathbf{z}_{t-2}) + \Phi\left(\begin{bmatrix}\bar{\mathbf{z}}_t & \mathbf{z}_{t-1} & \mathbf{w}_t\end{bmatrix}^{\mathrm{T}}\right). \tag{11}$$

Following this derivation we see that our method reduces to a special case of linear elastic subspace Verlet integration, where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are found from data, and a neural network Φ approximates all the other forces directly in the subspace. And, since our network Φ takes $\bar{\mathbf{z}}$ as input rather than $\bar{\mathbf{x}}$, it can do so efficiently and with a cost proportional to the basis size (rather than the number of vertices).

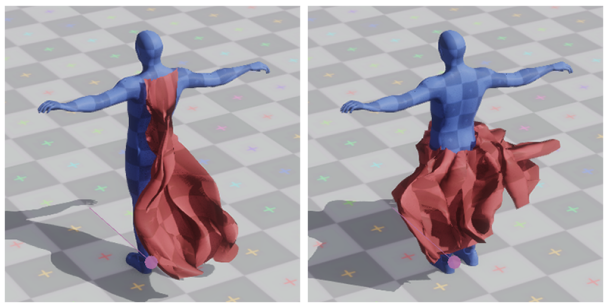

Figure 12: Without our training method the prediction is unstable.

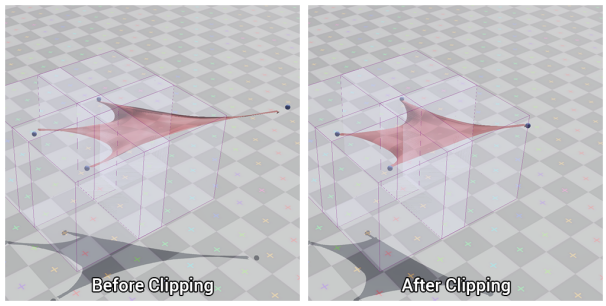

Figure 13: Figure showing the extrapolation behaviour of the *Four Pins* scene. Bad extrapolation can be prevented by simply clipping the inputs and outputs of the system within the range given in the training data.

This interpretation is important because, although it assumes an over-simplified inaccurate physical model, it opens up potential for future hybrid data-driven subspace methods using different integration schemes and/or more complex and accurate physical models which can potentially target more specific components of the simulation such as the expensive to calculate forces resulting from external collisions.

8.1 Limitations & Future Work

Like all data-driven methods our technique has a number of limitations. As expected, there is no guarantee that it will generalize well beyond the training distribution. However, we found that clipping the inputs and outputs of the system to the minimum and maximum found in the training data proved effective to combat this. See Fig 13 and supplementary video for visual examples. Similarly, all external object must be parameterized which in some cases can be difficult for example with varying numbers of external objects.

Table 2: Performance & Memory Comparison – we compare to Hyper-Reduced Projective Dynamics (HRPD) [Brandt et al. 2018], LSTMs [Hochreiter and Schmidhuber 1997] and GRUs [Cho et al. 2014]. We demonstrate performance gains of ~ 5× compared to HRPD and between ~ 300× to ~ 5000× compared to ground truth. Our architectural choice fairs well compared to the alternative LSTM and GRU baselines we explored.

Scene	Method	Time (μs)		Memory (MB)	
		CPU	GPU	CPU	GPU
Bunny	Reference	2.5×10^6	—	—	—
	HRPD	2834	—	26.0	—
	LSTM 256	448	128	5.4	7.5
	GRU 256	389	128	5.9	7.5
	Ours 256	360	128	5.7	7.5
	Ours 128 / 64	113 / 34	68 / 30	1.4 / 0.3	3.7 / 1.8
Ball & Sheet	Reference	1.32×10^5	—	—	—
	HRPD	2623	—	20.8	—
	LSTM 256	447	157	5.4	7.8
	GRU 256	377	157	5.9	7.8
	Ours 256	446	157	5.7	7.8
	Ours 128 / 64	103 / 38	77 / 42	1.4 / 0.3	3.9 / 1.9
Four Pins	Reference	6.45×10^4	—	—	—
	HRPD	2711	—	20.8	—
	LSTM 256	433	157	5.4	7.8
	GRU 256	403	157	5.9	7.8
	Ours 256	447	157	5.7	7.8
	Ours 128 / 64	121 / 38	77 / 42	1.4 / 0.3	3.9 / 1.9
Flag	Reference	9.17×10^4	—	—	—
	LSTM 256	434	157	5.4	7.8
	GRU 256	384	157	5.9	7.8
	Ours 256	344	157	5.7	7.8
	Ours 128 / 64	103 / 39	77 / 42	1.4 / 0.3	3.9 / 1.9
Skirt	Reference	3.22×10^5	—	—	—
	LSTM 256	447	183	5.4	9.0
	GRU 256	425	183	5.9	9.0
	Ours 256	342	183	5.7	9.0
	Ours 128 / 64	115 / 38	79 / 60	1.4 / 0.3	4.5 / 2.2
Cape	Reference	5.26×10^5	—	—	—
	LSTM 256	491	157	5.5	7.8
	GRU 256	393	157	6.0	7.8
	Ours 256	259	157	5.9	7.8
	Ours 128 / 64	124 / 41	77 / 42	1.5 / 0.3	3.9 / 1.9
Dragon	Reference	1.0×10^6	—	—	—
	LSTM 256	459	164	5.7	9.0
	GRU 256	363	164	5.7	9.0
	Ours 256	287	164	5.7	9.0
	Ours 128 / 64	84 / 37	82 / 37	1.4 / 0.3	4.5 / 2.2

Like all subspace methods we are limited by the expressiveness of the basis, and if additional fine details cannot be captured accurately with the addition of more modes, the computational cost can increase quickly.

Our method requires a time consuming training data acquisition, with up to several days of simulation time needed. As with any data-driven approach, this process requires proper management, e.g., removing any erroneous simulation data from the training set. We did not take any special measures here, and we expect this capture and preprocessing can be accelerated by simply running multiple simulations in parallel. Converged training time for our method is long and, although initial results for testing can be ready in an hour or two, our final models took roughly a day to train.

On its own, our method cannot guarantee that no intersections will occur with external geometry without relying on a separate solution like the capsule projection method (Section 6.4). Similarly, we cannot provide hard guarantees that self-collisions will not occur, however it seems challenging to provide such guarantees while also maintaining a simulation that runs entirely in the subspace (and with decompression only deferred until render time).

Currently we only show our results on elastic objects but we believe our method can be extended to plastic objects. One option is to explicitly track the rest state, predicting the change in rest state with the neural network. In plastic setups, care would likely need to be taken in how training data is acquired, as objects cannot be expected to return to their rest state when there are no interactions. An approach using adaptive bases may be required to combat the increased number of potential states [Hahn et al. 2014; Kim and James 2009].

Finally, our examples only show interactions where the external object is fixed in place, and only show simulated objects where the reference frame for the object is controlled externally. We believe it would be straightforward to extend this to having the neural network predict the forces applied to the reference frame of external objects, to allow them to move freely. This would allow for unpinned interactions, such as throwing boxes at a deformable object and having it react by rolling around on the floor. Here, we expect similar care would need to be taken in setting up the training data simulation and collection process.

9 CONCLUSION

We presented a data-driven method for subspace physical simulation of deformation, including self-collisions, and interactions with external objects and forces, combining subspace simulation with machine learning. Our method generates high-quality results several orders-of-magnitude faster than the reference simulation, significantly outperforming the state-of-the-art. The applicability to a broad set of deformation behaviors, its performance and its modest memory footprint make our method practical for use in modern AAA game and virtual reality engines.

A DIAGONALITY OF $\hat{\mathbf{A}}$ AND $\hat{\mathbf{B}}$

If we construct our subspace projection matrix $\mathbf{U}$ using PCA we expect each basis to be orthogonal and statistically independent. Given this, we should also expect each dimension of $\mathbf{z}$ to *change* largely independently. As such, we would expect that in a relation such as $\mathbf{z}_t = \hat{\mathbf{A}}\mathbf{z}_{t-1} + \hat{\mathbf{B}}(\mathbf{z}_{t-1} - \mathbf{z}_{t-2})$, we would find the matrices $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ to be largely diagonal as non-diagonal entries would imply statistical coupling between the dimensions of $\mathbf{z}$. To test this hypothesis we compute matrices $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ using linear least squares fitting of the above equation on our training data $\mathbf{Z}$. As expected, we find that $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ are primarily diagonal matrices, as visualized in Fig 14.

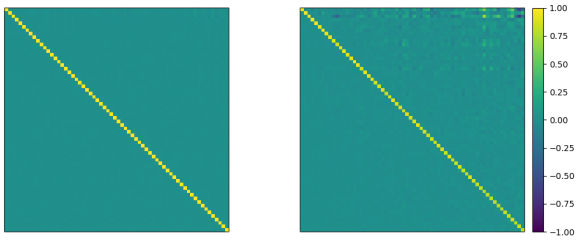

Figure 14: Visualisation of matrices $\hat{A}$ and $\hat{B}$ using the 64 bases of the cloth from the *Cape* scene. Typical values of the diagonal of $\hat{A}$ range from 0.995 to 1.0 while typical values on the diagonal of $\hat{B}$ range from 0.75 to 1.0.

ACKNOWLEDGMENTS

This research was funded in part through the Natural Sciences and Engineering Research Council of Canada's NSERC/Ubisoft Industrial Research Chair on Believable Virtual Character Experience (NSERC IRCPJ 522419-17).

REFERENCES

Steven S. An, Theodore Kim, and Doug L. James. 2008. Optimizing Cubature for Efficient Integration of Subspace Deformations. In *ACM SIGGRAPH Asia 2008 Papers (SIGGRAPH Asia '08)*. ACM, New York, NY, USA, Article 165, 10 pages. https://doi.org/10.1145/1457515.1409118

Jernej Barbič and Doug L. James. 2005. Real-Time Subspace Integration for St. Venant-Kirchhoff Deformable Models. In *ACM SIGGRAPH 2005 Papers (SIGGRAPH '05)*. ACM, New York, NY, USA, 982–990. https://doi.org/10.1145/1186822.1073300

Jan Bender, Matthias Müller, and Miles Macklin. 2017. Position-Based Simulation Methods in Computer Graphics. In *EUROGRAPHICS 2017 Tutorials*. Eurographics Association. https://doi.org/10.2312/egt.20171034

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective Dynamics: Fusing Constraint Projections for Fast Simulation. *ACM Trans. Graph.* 33, 4, Article 154 (July 2014), 11 pages. https://doi.org/10.1145/2601097.2601116

Christopher Brandt, Elmar Eisemann, and Klaus Hildebrandt. 2018. Hyper-reduced Projective Dynamics. *ACM Trans. Graph.* 37, 4, Article 80 (July 2018), 13 pages. https://doi.org/10.1145/3197517.3201387

KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *CoRR* abs/1409.1259 (2014). arXiv:1409.1259 http://arxiv.org/abs/1409.1259

Mengyu Chu and Nils Thuerey. 2017. Data-Driven Synthesis of Smoke Flows with CNN-based Feature Descriptors. *CoRR* abs/1705.01425 (2017). arXiv:1705.01425 http://arxiv.org/abs/1705.01425

Simon Clavet. 2016. Motion Matching and The Road to Next-Gen Animation. In *Proc. of GDC 2016*.

Edilson de Aguiar, Leonid Sigal, Adrien Treuille, and Jessica K. Hodgins. 2010. Stable Spaces for Real-time Clothing. In *ACM SIGGRAPH 2010 Papers (SIGGRAPH '10)*. ACM, New York, NY, USA, Article 106, 9 pages. https://doi.org/10.1145/1833349.1778843

P. Dollár, P. Welinder, and P. Perona. 2010. Cascaded pose regression. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 1078–1085. https://doi.org/10.1109/CVPR.2010.5540094

Lawson Fulton, Vismay Modi, David Duvenaud, David I. W. Levin, and Alec Jacobson. 2019. Latent-space Dynamics for Reduced Deformable Simulation. *Computer Graphics Forum* (2019).

Yarin Gal and Zoubin Ghahramani. 2016. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'16)*. Curran Associates Inc., USA, 1027–1035. http://dl.acm.org/citation.cfm?id=3157096.3157211

Nico Galoppo, Miguel Otaduy, Serhat Tekin, Markus Gross, and Ming Lin. 2007. Soft Articulated Characters with Fast Contact Handling. *Comput. Graph. Forum* 26 (09 2007), 243–253. https://doi.org/10.1111/j.1467-8659.2007.01046.x

Nico Galoppo, Miguel A. Otaduy, William Moss, Jason Sewall, Sean Curtis, and Ming C. Lin. 2009. Controlling Deformable Material with Dynamic Morph Targets. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games (I3D '09)*. ACM, New York, NY, USA, 39–47. https://doi.org/10.1145/1507149.1507156

Fabian Hahn, Bernhard Thomaszewski, Stelian Coros, Robert W. Sumner, Forrester Cole, Mark Meyer, Tony DeRose, and Markus Gross. 2014. Subspace Clothing Simulation Using Adaptive Bases. *ACM Trans. Graph.* 33, 4, Article 105 (July 2014), 9 pages. https://doi.org/10.1145/2601097.2601160

David Harmon and Denis Zorin. 2013. Subspace Integration with Local Deformations. *ACM Trans. Graph.* 32, 4, Article 107 (July 2013), 10 pages. https://doi.org/10.1145/2461912.2461922

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (Nov. 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

J. Huang, Y. Tong, K. Zhou, H. Bao, and M. Desbrun. 2011. Interactive Shape Interpolation through Controllable Dynamic Deformation. *IEEE Transactions on Visualization and Computer Graphics* 17, 7 (July 2011), 983–992. https://doi.org/10.1109/TVCG.2010.109

Thomas Jakobsen. 2001. Advanced character physics. In *Game Developers Conference*.

Doug L. James and Dinesh K. Pai. 2002. DyRT: Dynamic Response Textures for Real Time Deformation Simulation with Graphics Hardware. *ACM Trans. Graph.* 21, 3 (July 2002), 582–585. https://doi.org/10.1145/566654.566621

Ning Jin, Yilin Zhu, Zhenglin Geng, and Ronald Fedkiw. 2018. A Pixel-Based Framework for Data-Driven Clothing. *CoRR* abs/1812.01677 (2018). arXiv:1812.01677 http://arxiv.org/abs/1812.01677

Angjoo Kanazawa, Michael J. Black, David W. Jacobs, and Jitendra Malik. 2017. End-to-end Recovery of Human Shape and Pose. *CoRR* abs/1712.06584 (2017). arXiv:1712.06584 http://arxiv.org/abs/1712.06584

Byungsoo Kim, Vinicius C. Azevedo, Nils Thuerey, Theodore Kim, Markus H. Gross, and Barbara Solenthaler. 2018. Deep Fluids: A Generative Network for Parameterized Fluid Simulations. *CoRR* abs/1806.02071 (2018). arXiv:1806.02071 http://arxiv.org/abs/1806.02071

Doyub Kim, Woojong Koh, Rahul Narain, Kayvon Fatahalian, Adrien Treuille, and James F. O'Brien. 2013. Near-exhaustive Precomputation of Secondary Cloth Effects. *ACM Trans. Graph.* 32, 4, Article 87 (July 2013), 8 pages. https://doi.org/10.1145/2461912.2462020

Theodore Kim and Doug L. James. 2009. Skipping Steps in Deformable Simulation with Online Model Reduction. In *ACM SIGGRAPH Asia 2009 Papers (SIGGRAPH Asia '09)*. ACM, New York, NY, USA, Article 123, 9 pages. https://doi.org/10.1145/1661412.1618469

Theodore Kim and Doug L. James. 2011. Physics-based Character Skinning Using Multi-domain Subspace Deformations. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '11)*. ACM, New York, NY, USA, 63–72. https://doi.org/10.1145/2019406.2019415

Tae-Yong Kim, Nuttapong Chentanez, and Matthias Müller-Fischer. 2012. Long Range Attachments - a Method to Simulate Inextensible Clothing in Computer Games. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '12)*. Eurographics Association, Goslar Germany, Germany, 305–310. http://dl.acm.org/citation.cfm?id=2422356.2422399

Paul G. Kry, Doug L. James, and Dinesh K. Pai. 2002. EigenSkin: Real Time Large Deformation Character Skinning in Hardware. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '02)*. ACM, New York, NY, USA, 153–159. https://doi.org/10.1145/545261.545286

Lubor Ladicky, SoHyeon Jeong, Nemanja Bartolovic, Marc Pollefeys, and Markus Gross. 2017. Physicsforests: Real-time Fluid Simulation Using Machine Learning. In *ACM SIGGRAPH 2017 Real Time Live! (SIGGRAPH '17)*. ACM, New York, NY, USA, 22–22. https://doi.org/10.1145/3098333.3098337

L'ubor Ladický, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. 2015. Data-driven Fluid Simulations Using Regression Forests. *ACM Trans. Graph.* 34, 6, Article 199 (Oct. 2015), 9 pages. https://doi.org/10.1145/2816795.2818129

Zorah Lähner, Daniel Cremers, and Tony Tung. 2018. DeepWrinkles: Accurate and Realistic Clothing Modeling. *CoRR* abs/1808.03417 (2018). arXiv:1808.03417 http://arxiv.org/abs/1808.03417

Tiantian Liu, Adam W. Bargteil, James F. O'Brien, and Ladislav Kavan. 2013. Fast Simulation of Mass-spring Systems. *ACM Trans. Graph.* 32, 6, Article 214 (Nov. 2013), 7 pages. https://doi.org/10.1145/2508363.2508406

Ran Luo, Tianjia Shao, Huamin Wang, Weiwei Xu, Xiang Chen, Kun Zhou, and Yin Yang. 2018. NNWarp: Neural Network-based Nonlinear Deformation. *IEEE Transactions on Visualization and Computer Graphics* PP (11 2018), 1–1. https://doi.org/10.1109/TVCG.2018.2881451

Miles Macklin and Matthias Müller. 2013. Position Based Fluids. *ACM Trans. Graph.* 32, 4, Article 104 (July 2013), 12 pages. https://doi.org/10.1145/2461912.2461984

Miles Macklin, Matthias Müller, and Nuttapong Chentanez. 2016. XPBD: Position-based Simulation of Compliant Constrained Dynamics. In *Proceedings of the 9th International Conference on Motion in Games (MIG '16)*. ACM, New York, NY, USA, 49–54. https://doi.org/10.1145/2994258.2994272

Matthias Müller and Nuttapong Chentanez. 2011. Solid Simulation with Oriented Particles. In *ACM SIGGRAPH 2011 Papers (SIGGRAPH '11)*. ACM, New York, NY, USA, Article 92, 10 pages. https://doi.org/10.1145/1964921.1964987

Matthias Müller, Nuttapong Chentanez, Tae-Yong Kim, and Miles Macklin. 2014. Strain Based Dynamics. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '14)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 149–157. http://dl.acm.org/citation.cfm?id=2849517.2849542

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position based dynamics. *Journal of Visual Communication and Image Representation* 18, 2 (2007), 109 – 118. https://doi.org/10.1016/j.jvcir.2007.01.005

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning (ICML'10)*. Omnipress, USA, 807–814. http://dl.acm.org/citation.cfm?id=3104322.3104425

Rahul Narain, Matthew Overby, and George E. Brown. 2016. ADMM ⊇ Projective Dynamics: Fast Simulation of General Constitutive Models. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '16)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 21–28. http://dl.acm.org/citation.cfm?id=2982818.2982822

Markus Oberweger, Paul Wohlhart, and Vincent Lepetit. 2016. Training a Feedback Loop for Hand Pose Estimation. *CoRR* abs/1609.09698 (2016). arXiv:1609.09698 http://arxiv.org/abs/1609.09698

Zherong Pan, Hujun Bao, and Jin Huang. 2015. Subspace Dynamic Simulation Using Rotation-strain Coordinates. *ACM Trans. Graph.* 34, 6, Article 242 (Oct. 2015), 12 pages. https://doi.org/10.1145/2816795.2818090

Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. 2018. On the Convergence of Adam and Beyond. In *International Conference on Learning Representations*. https://openreview.net/forum?id=ryQu7f-RZ

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1. MIT Press, Cambridge, MA, USA, Chapter Learning Internal Representations by Error Propagation, 318–362. http://dl.acm.org/citation.cfm?id=104279.104293

Eftychios Sifakis and Jernej Barbic. 2012. FEM Simulation of 3D Deformable Solids: A Practitioner's Guide to Theory, Discretization and Model Reduction. In *ACM SIGGRAPH 2012 Courses (SIGGRAPH '12)*. ACM, New York, NY, USA, Article 20, 50 pages. https://doi.org/10.1145/2343483.2343501

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15 (2014), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html

J. Stam. 2009. Nucleus: Towards a unified dynamics solver for computer graphics. In *2009 11th IEEE International Conference on Computer-Aided Design and Computer Graphics*. 1–11. https://doi.org/10.1109/CADCG.2009.5246818

Yun Teng, Mark Meyer, Tony DeRose, and Theodore Kim. 2015. Subspace Condensation: Full Space Adaptivity for Subspace Deformations. *ACM Trans. Graph.* 34, 4, Article 76 (July 2015), 9 pages. https://doi.org/10.1145/2766904

Yun Teng, Miguel A. Otaduy, and Theodore Kim. 2014. Simulating Articulated Subspace Self-contact. *ACM Trans. Graph.* 33, 4, Article 106 (July 2014), 9 pages. https://doi.org/10.1145/2601097.2601181

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. 2016. Accelerating Eulerian Fluid Simulation With Convolutional Networks. *CoRR* abs/1607.03597 (2016). arXiv:1607.03597 http://arxiv.org/abs/1607.03597

Adrien Treuille, Andrew Lewis, and Zoran Popović. 2006. Model Reduction for Real-time Fluids. In *ACM SIGGRAPH 2006 Papers (SIGGRAPH '06)*. ACM, New York, NY, USA, 826–834. https://doi.org/10.1145/1179352.1141962

Alexis Vaisse. 2016. Ubisoft Cloth Simulation: Performance Postmortem and Journey from C++ to Compute Shaders. In *GDC 2016 Talks*.

Philipp von Radziewsky, Elmar Eisemann, Hans-Peter Seidel, and Klaus Hildebrandt. 2016. Optimized Subspaces for Deformation-based Modeling and Shape Interpolation. *Comput. Graph.* 58, C (Aug. 2016), 128–138. https://doi.org/10.1016/j.cag.2016.05.016

Christoph von Tycowicz, Christian Schulz, Hans-Peter Seidel, and Klaus Hildebrandt. 2013. An Efficient Construction of Reduced Deformable Objects. *ACM Trans. Graph.* 32, 6, Article 213 (Nov. 2013), 10 pages. https://doi.org/10.1145/2508363.2508392

Huamin Wang, Florian Hecht, Ravi Ramamoorthi, and James F. O'Brien. 2010. Example-based Wrinkle Synthesis for Clothing Animation. In *ACM SIGGRAPH 2010 Papers (SIGGRAPH '10)*. ACM, New York, NY, USA, Article 107, 8 pages. https://doi.org/10.1145/1833349.1778844

Yu Wang, Alec Jacobson, Jernej Barbič, and Ladislav Kavan. 2015. Linear Subspace Design for Real-time Shape Deformation. *ACM Trans. Graph.* 34, 4, Article 57 (July 2015), 11 pages. https://doi.org/10.1145/2766952

Marcel Weiler, Dan Koschier, and Jan Bender. 2016. Projective Fluids. In *Proceedings of the 9th International Conference on Motion in Games (MIG '16)*. ACM, New York, NY, USA, 79–84. https://doi.org/10.1145/2994258.2994282

Steffen Wiewel, Moritz Becher, and Nils Thuerey. 2018. Latent-space Physics: Towards Learning the Temporal Evolution of Fluid Flow. *CoRR* abs/1802.10123 (2018). arXiv:1802.10123 http://arxiv.org/abs/1802.10123

You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. 2018. tempoGAN: A Temporally Coherent, Volumetric GAN for Super-resolution Fluid Flow. *CoRR* abs/1801.09710 (2018). arXiv:1801.09710 http://arxiv.org/abs/1801.09710

Hongyi Xu and Jernej Barbič. 2016. Pose-space Subspace Dynamics. *ACM Trans. Graph.* 35, 4, Article 35 (July 2016), 14 pages. https://doi.org/10.1145/2897824.2925916

Weiwei Xu, Nobuyuki Umentani, Qianwen Chao, Jie Mao, Xiaogang Jin, and Xin Tong. 2014. Sensitivity-optimized Rigging for Example-based Real-time Clothing Synthesis. *ACM Trans. Graph.* 33, 4, Article 107 (July 2014), 11 pages. https://doi.org/10.1145/2601097.2601136

Yin Yang, Dingzeyu Li, Weiwei Xu, Yuan Tian, and Changxi Zheng. 2015. Expediting Precomputation for Reduced Deformable Simulation. *ACM Trans. Graph.* 34, 6, Article 243 (Oct. 2015), 13 pages. https://doi.org/10.1145/2816795.2818089

Y. Yang, W. Xu, X. Guo, K. Zhou, and B. Guo. 2013. Boundary-Aware Multidomain Subspace Deformation. *IEEE Transactions on Visualization and Computer Graphics* 19, 10 (Oct 2013), 1633–1645. https://doi.org/10.1109/TVCG.2013.12