Swish: Neural Network Cloth Simulation on Madden NFL 21

Chris Lewin
chlewin@ea.com
Electronic Arts / SEED
UK

James Power
jpower@ea.com
Electronic Arts / Tiburon
USA

James Cobb
jcobb@ea.com
Electronic Arts / Tiburon
USA

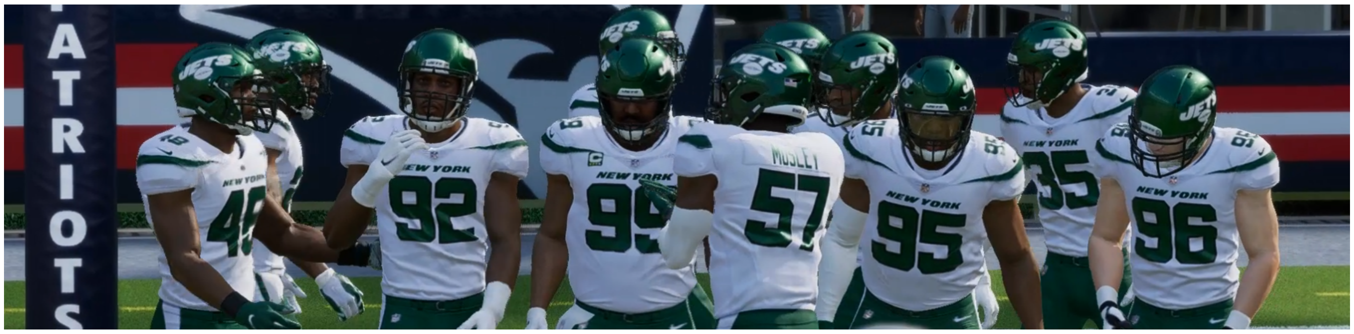

Figure 1: Player jerseys in Madden NFL 21 simulated using our system.

ABSTRACT

This work presents Swish, a real-time machine-learning based cloth simulation technique for games. Swish was used to generate realistic cloth deformation and wrinkles for NFL player jerseys in Madden NFL 21. To our knowledge, this is the first neural cloth simulation featured in a shipped game. This technique allows accurate high-resolution simulation for tight clothing, which is a case where traditional real-time cloth simulations often achieve poor results. We represent cloth detail using both mesh deformations and a database of normal maps, and train a simple neural network to predict cloth shape from the pose of a character's skeleton. We share implementation and performance details that will be useful to other practitioners seeking to introduce machine learning into their real-time character pipelines.

ACM Reference Format:
Chris Lewin, James Power, and James Cobb. 2021. Swish: Neural Network Cloth Simulation on Madden NFL 21. In *Siggraph '21 Talks, Aug 01–05, Los Angeles, CA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The clothing worn by characters in modern video-games is often simulated using techniques such as Position-Based Dynamics [Müller et al. 2007]. These techniques broadly work well for clothing that is loosely fitting and relatively low in resolution. However, real clothing exhibits high frequency folds and wrinkles that cannot be simulated with a low-resolution representation.

Recently, advances in machine learning have stimulated interest in learning-based cloth solutions. Although promising results have been achieved using convolutional and graph neural networks, subspace methods offer the greatest potential efficiency gains over standard full-space simulations. Subspace methods marry well with tight clothing, which is naturally more heavily constrained closely to the body.

2 IMPLEMENTATION

Swish is a quasi-static pose-based deformer system for tight clothing. We follow [Holden et al. 2019] in using a PCA-based representation of cloth shape, but we do not seek to represent dynamics; instead we train a stateless pose-to-cloth-shape regressor. We learn a single model per combination of cloth and underlying body, with no attempt to parameterize body shape. To represent high resolution detail, we build a database of normal maps which can be heavily compressed and then indexed into efficiently at run time.

2.1 Shape and Pose Representation.

A typical practice in real-time rendering is to use mesh geometry that is as simple as possible, and to suggest the appearance of high frequency details using normal maps. We adopt this practice by breaking high-resolution source simulations into paired deformations of a low-resolution in-game mesh, and a tangent-space normal map texture. Swish mesh deformations are represented as vertex-based perturbations of a skinned mesh, in the style of [Kry et al. 2002]: we transform cloth meshes to a pre-skinning space and then use PCA to decompose different deformation samples into a number of basis vectors, and a series of corresponding weights for each frame.

We also apply PCA to our normal textures, but reconstructing images from a PCA representation is both expensive and gives muddy results. To avoid this, we use a nearest neighbour database of textures as our rendered representation, with the texture's PCA projection used as the database key. This allows us to use a relatively

Siggraph '21 Talks, Aug 01–05, 2021, Los Angeles, CA
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

small number of PCA dimensions while not compromising on the quality of our normal maps at all.

Stacking the PCA weights for the mesh shape together with the PCA weights for the normal maps gives the total shape vector Λ that we wish to infer each frame from the character's pose Θ. This is the concatenation of the 6D rotation representations [Zhou et al. 2020] of the *relevant joints* $\mathbf{J}_i$. This set of skeleton joints is selected by the user, and in this work we uniformly use the character's spine joints only.

2.2 Training Process

To generate training data pairs (Θ, Λ) for our simulation, we must first generate samples Θ_i that span the different poses the host character can enter during gameplay. To do this, we record animation from all characters in the game for a short period of gameplay, generating several hundred thousand logically-unique pose samples. We thin this dense cloud of samples out to a reduced set using a greedy algorithm that simply rejects any samples that are too close to one another, arriving at a list of poses that are guaranteed to be separated by some tolerance. In our shipped models, pose data was gathered from ten minutes of varied gameplay and was thinned down to around 700 pose samples Θ_i.

Cloth Simulation. For each pose sample, we need to generate a corresponding cloth shape. To do this, we use a Marvelous Designer simulation for each pose sample. To remove the dynamics from the cloth simulation, we generate a slow (1 second) transition animation where the character moves smoothly from a neutral A-pose to the target pose Θ_i. This pose is then associated with the cloth shape at the final frame of the simulation. This method ensures that cloth shape is explainable purely as a function of the target pose, and that the deformation history is consistent between similar poses.

Postprocessing. Cloth simulation results in a high-resolution triangle mesh that is not suitable for usage directly in a game. To generate the low-resolution mesh deformations and normal maps required by our method, we use a postprocessing stack implemented in Autodesk Maya. We generate low-resolution mesh deformations by binding an artist-authored in-game mesh to follow corresponding locations on the high-resolution cloth simulation mesh, and then use Maya functionality to generate normal map textures that represent the difference in detail between the two meshes. Trying to represent the deformations of a high-resolution mesh with a low-resolution proxy has obvious sampling frequency problems that present as jagged triangle quality in the in-game mesh. To tame this artifact we apply low-pass filtering using Laplacian mesh smoothing.

Pose Based Regressor. Given corresponding pairs of (Θ, Λ), we train a regressor $\Lambda = R(\Theta)$. We use a small Multi-Layer Perceptron (MLP) to represent R. Specifically, in our final shipped version our MLP used 5 hidden layers of 32 neurons, using SELU activations, with an input size of 32 and an output size of between 20 and 30 depending on the individual cloth asset. This model could be trained in 5 minutes, which was very small compared to the preprocessing times of between 6 and 48 hours per model required to generate the cloth simulation data. We found that shrinking the network below this level of complexity rapidly degraded the quality of our results.

Normal Map Database. To maintain high quality in our shading details, we use a nearest-neighbour database of normal maps. We use k-means to cluster the PCA codes corresponding to source normal maps, and choose the PCA point closest to each cluster center to become a data point in the database. The database key is the PCA code, and for lookup we simply use a brute force search for the nearest code. In our shipped version, we use databases of 128 textures. Each texture is 360x360 pixels. To compress this database, we use Vector Quantization (VQ) applied to 4x4 blocks. With 16-bit block indices, this leads to a texture memory cost of 3MB per cloth asset, a saving of over 5x compared to using BC5 compression alone.

Table 1: Summary of processing time, per model instance, on a console platform.

Stage	Device	Cost (μs)
Neural Network Inference	CPU	20
Mesh Deform	GPU	115
Normal Maps	GPU	5

We perform no blending at all between different samples in the database; without additional measures this means that users can see a pop when the neural network changes its choice of normal map. To counteract this, we use a simple temporal filter that blends the new selection into the last few frames' normal maps. This works well to smooth out transitions between frames as long as the character animates rapidly enough.

2.3 Runtime

Each frame, the Swish simulation process for a cloth model is simple:

(1) Extract joint rotations for this frame to form Θ
(2) Run neural network inference $\Lambda = R(\Theta)$
(3) Reconstruct cloth shape using PCA vectors and mesh weights
(4) Look up normal map in database using normal map weights
(5) Render character using mesh deformations and normal map

Relevant performance data for our system is listed in Table 1.

3 DISCUSSION

Overall, our method is several orders of magnitude cheaper than the source simulations, and also cheaper than a standard real-time cloth simulation. Our method does not handle dynamics, which is a significant omission even in the case of relatively tight clothing. Our main priority for future work is to support dynamic deformation, as well as to generalise further to different body types, which will allow us to avoid training multiple models that only differ in the underlying character shape. Further reducing memory costs is also a priority.

REFERENCES

Daniel Holden, Bang Chi Duong, Sayantan Datta, and Derek Nowrouzezahrai. 2019. Subspace Neural Physics: Fast Data-Driven Interactive Simulation. In *Proceedings of the 18th Annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Los Angeles, California) *(SCA '19)*.

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* (San Antonio, Texas) *(SCA '02)*. Association for Computing Machinery, New York, NY, USA, 153–159. https://doi.org/10.1145/545261.545286

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position Based Dynamics. *J. Vis. Comun. Image Represent.* 18, 2 (April 2007), 109–118.

Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. 2020. On the Continuity of Rotation Representations in Neural Networks. arXiv:1812.07035 [cs.LG]