Generating Motion Capture Data for Arbitrary Rigs

Marianna Neubauer, Shannon Kao

I. Introduction

As motion capture techniques become more advanced and accessible, motion capture data has become an increasingly popular source of efficient, realistic animation. The process of motion capture involves placing markers on an actor's body and recording movement data for each marker. This data can then be mapped onto a digital human skeleton, called a rig, which consists of joints (e.g. hips, elbows, wrists). Motion capture is a practical alternative to the more traditional method of hand-animating each joint in a rig. However, the data obtained by motion capture is highly specific to the rig used to record it, making it difficult to transfer recorded animation from one rig setup to another.

There is a plethora of motion capture data available on-line, however 3D artists who already have an existing rig that is well suited for a unique character cannot use that animation. This project aims to break the dependence of motion capture on a specific rig by generalizing animation for arbitrary rigs using machine learning algorithms.

A. System Overview

In this paper we introduce a system that generates motion capture-based animation for an arbitrary input rig. We split our training data–rigs and associated motion capture animation from various databases–into individual joints, assigning each joint a time-varying series of translations and rotations. Our system then applies a k-means clustering algorithm to classify the joints. Finally, it uses linear regression with gradient descent on the calculated clusters to output an appropriate animation sequence for each joint in the input test rig.

II. Related Work

Motion synthesis is the problem of generating animated motions that meet a set of requirements. Recently, there has been a surge of algorithms addressing the creation of new animation sequences from existing motion segments. In general, motion synthesis methods can be classified as physically-based [1], [2] or example-based, like our algorithm.

In the realm of example-based motion synthesis, the concept of motion graphs, introduced by Kovar [3], Lee [4], and Thuraisingham [5] in 2002, has gained significant traction. Motion graphs apply the technique of texture synthesis to the problem of motion synthesis. In general, though, these algorithms require a library of motion clips specific to one skeleton. These clips are cut and interpolated to create unique, original motions [6], [7].

Image-based motion synthesis uses video clips as input data. There are a variety of techniques used to infer 3D structure from video [8], [9], [10] and extract motion data from these structures [9] to output a three-dimensional animation. However, when applied to rigged characters, these methods largely require the user to have foreknowledge of skeletal positioning and hierarchy.

Motion synthesis methods up until this point have focused largely on generating original motion for an existing character rig. They assume knowledge of the rig structure, as well as an existing set of animations specific to that rig. We approach this problem from the other direction: assuming a large dataset of one class of motion, can we generate similar data for an arbitrary rig?

To analyze such a dataset, we turn to machine learning techniques. There is a variety of research in the area of motion classification and analysis [11], ranging from analysis on level of motion clips as a whole [12], [13], down to a neurological interpretation of joints as primitives [11]. While these algorithms are largely used to split and classify a range of different motions, we found the common method of clustering over animation groups [4] applicable to our problem.

The system we propose makes no assumptions about rig structure or layout. By treating joints as individual data points, we are able to apply these methods agnostic of underlying skeletal hierarchy. Further, this system makes no requirements on input animation. Drastically different rig structures can be used to provide useful motion data, despite variation in scale, positioning, and animation phase. We utilize clustering and linear regression to learn a predefined motion from a large database of example animation, then generate believable, realistic motion data for the input rig.

III. Data Set and Features

We used Biovision Hierarchy (BVH) files for our training data. Each BVH file contains a rig and a single motion defined as a time series of the root translation and joint rotations. Our data set consists of 97 files from Carnegie Mellons motion capture database [14]. For the scope of the project, we trained only on walk cycles, a common motion that has a well defined structure. However, our techniques could easily be generalized to commonly-encountered motions like running, sitting, or throwing.

BVH file format: A BVH file has two sections. The first section defines a rig in a tree-like hierarchy of joint structures, with a root that represents the pelvis of a human skeleton. Each BVH file has a unique skeleton, where the number of joints, parenting structure, motion channels, and joint offset from parent may vary. The second section defines the animation. Each joint in the skeleton has a time series of transformations: translation for the root joint and rotations for the other joints.

A. Data Preprocessing

1) Loading the BVH files: We utilized a BVH loader [15] to load and preprocess the training data. A BVH file contains the

absolute translation and rotation of the root joint and relative rotations of all other joints. This script computed the absolute translations from that information and the relative offsets of the joints from their parent. We use these absolute translations in our error metric and curve visualization.

2) Normalise animation frames: The input data has animations recorded at anywhere from 30 to 120 frames per second. We normalise all data to 30 fps by omitting extraneous frames of animation from higher-rate samples. A minimum frame rate of 30 fps was chosen to reflect general industry standard for real-time animation.

3) Joint absolute positions: Our system is designed to works on varying rigs, so the number of joints and their placement will vary between different samples. To handle this, we explode each rig into individual joints, and train our data across all joints. A joint in the BVH file is defined with a transformation relative to its parent in the hierarchy, so we perform a recursive depth-first traversal of the BVH to obtain the absolute positioning of each joint.

4) Scale and center rigs: The input skeletons have a wide range of positions and sizes. To address this, we scale all rigs to be the same height, using the y-axis as a metric in order to avoid size errors due to hand or arm positioning along the x- and z-axes. We also center each rig at (0, 0, 0) to normalize for offsets in geometry placement.

B. Features

1) Input Features: The input features for each joint includes the absolute position in xyz coordinates, the offset of the joints from its parent, the depth of the joint in the hierarchy tree, and the number of children of the joint. This feature space encapsulates both position and hierarchy information for the joints.

2) Target Features: Our goal is to generate an animation sequence defining the movement of a particular joint over a time sequence. This movement can be recorded as a series of rotations or translations, each associated with a particular time t. Each spatial and rotational coordinate is called a "channel". Both rotations and translations have 3 channels – x, y, and z – for a total of 6 channels for an animation curve (Figure 1).

Rotation: a $3 \times T$ matrix, where T is the total number of animation frames in our training data. At each frame, there is an xyz vector defining the Euler rotation about each axis, relative to the parent joint.

Translation: a $3 \times T$ matrix. The xyz coordinates define the absolute translation of that joint in word space. This translation is computed from the relative rotations and the offset of the joint from its parent.

IV. Methods

In order to assign animation sequences to an arbitrary skeleton rig, we break down the skeleton into its individual joints and assign an animation sequence to each joint. Our training algorithm has two steps: cluster the joints, then run a linear regression within each cluster. Using the output of the linear regression on each cluster, we hope to produce animation curves like the ones seen in Figure 1 from the position and joint hierarchy information of a test skeleton rig.

A. K-means Clustering

We cluster the joints according to the input features described in section *III*. We set k to be the maximum number of joints a single skeleton has in the training data set. If a skeleton has more joints than the number of clusters, we do not want neighboring joints to collapse into the same cluster because they may have dramatically different animation curves. Consider a collarbone joint. Many skeletons may not this particular joint, but it should not be assigned the animation curve of the neighboring shoulder joint. Using our feature set and a fixed k, there are many optimal solutions to k-means clustering if the centroids are randomly initiated. This presents a problem because we require that the joints in the training set are consistently clustered so linear regression runs on animation curves with similar structures. Our current solution to this problem is to set the initial centroids to the joints of the skeleton with the most joints.

B. Linear Regression

We perform a linear regression within each joint cluster to compute time-varying coefficients for each channel of the animation curve. Using the aforementioned input features, we perform gradient descent [16] on each channel and time point individually. This results in c $n \times T$ matrices of coefficients (θ), where c is the number of channels, n is the number of features, and T is the number of frames.

Gradient descent attempts to calculate $\theta \in R^{n \times T}$ for each animation channel: θ_{tx}, θ_{ty}, θ_{tx}, θ_{rx}, θ_{ry}, and θ_{rz}. We iteratively assign the elements of θ according to the cost function $J(\theta)$ and learning rate α:

$$\theta_{jt} := \theta_{jt} - \frac{\partial}{\partial \theta} J(\theta) \tag{1}$$

j is an input feature and t is the time point in the animation curve. The cost function, $J(\theta)$ is the least squares cost function:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} (\theta_t x^{(i)} - y_t^{(i)})^2 \tag{2}$$

i is the current joint in the cluster and $y_t^{(i)}$ is the animation curve at time t for joint i.

C. Computing Animation for a Test Rig

Given a test rig with no associated animation, we compute the rotations and translations of each joint at each time point by premultiplying the input features of each joint by the corresponding θ.

D. Result Visualization

Using the xyz translation coefficients found in linear regression, we can compute the absolute translations of the joints over time and visualize it in 3D animation software such as Autodesk Maya, as in Figure 4.

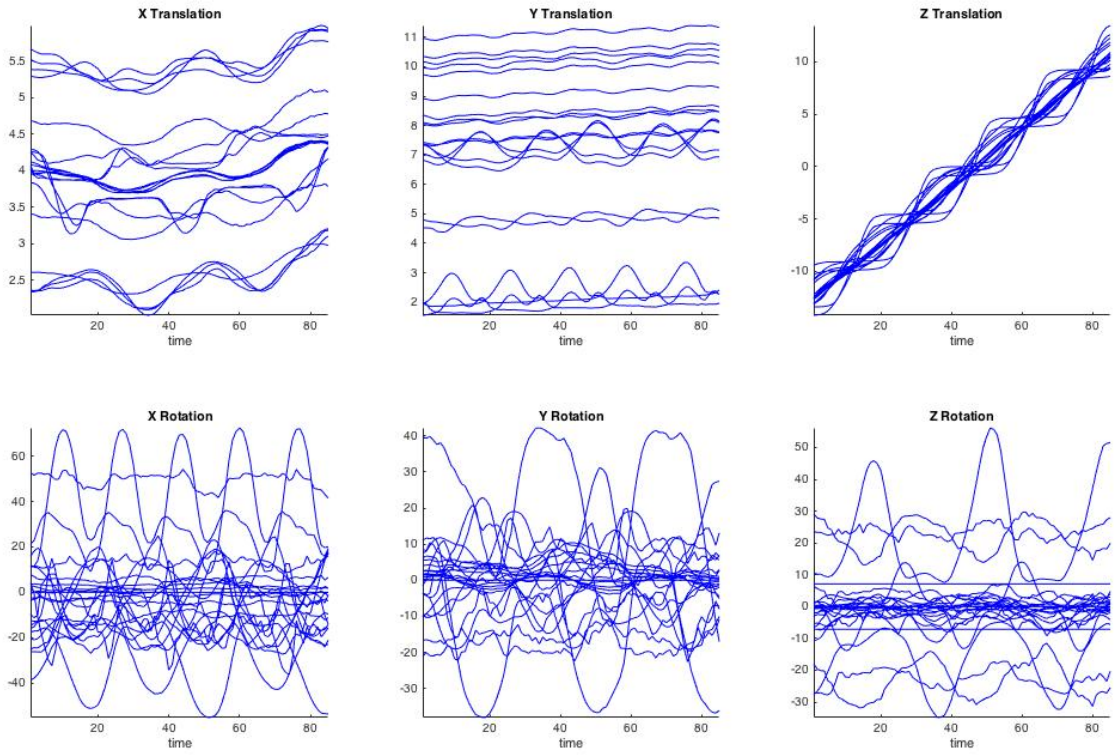

Fig. 1. The animation curves of one skeleton from the CMU dataset. Each curve represents the translation or rotation of a single joint over time. This skeleton has 31 joints and 85 frames of animation.

E. Error Computation of Animation Curves

In order to compute the error between animation curves, which are time varying series, we compute the distance between two $cx \times T$ vectors (Figure 2), where c is the number of channels [17]. Each channel of an animation curve has periodicity, so it is appropriate to perform DTW on the animation curve. DTW is a dynamic programming algorithm

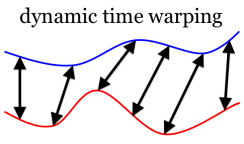

Fig. 2. Conceptual drawing of the distances calculated in DTW [17]

that computes, in order, the difference between each pair of time points in two curves, adds it to the minimum distance of its three neighboring pairs, and caches the result. The final distance observed after all pairs of time points are compared is the reported distance of the curve.

V. Results

A. Clustering

For our clustering algorithm, we found that using absolute position, offset from parent, depth in the joint hierarchy, and number of children produced the best results. Figure 3 shows the joint clusters found using these features. Other feature sets we tried include absolute position alone, animation curve alone, and the set of position, hierarchy, and animation data. See section *V-C-2* for the performance details of these different methods.

B. Linear Regression

By iteratively tuning parameters in the linear regression, we found $\alpha = 0.01$ and 1000 iterations to produce the best results. If alpha was smaller or the number of iterations were smaller, gradient descent did not converge. If they were larger, the cost function J diverged. Since our dataset was small, iterative tuning was reasonable. Figure 5 shows the animation curves derived from the θ's computed from linear regression.

C. Performance

Distance Metric: In order to analyze performance, we needed a distance metric to calculate the similarity between

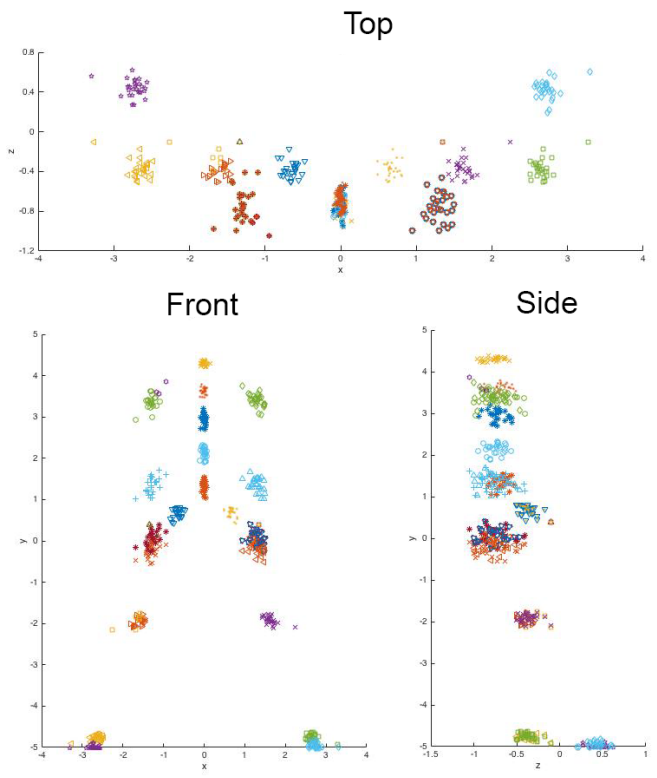

Fig. 3. K-means clustering of joints from the CMU data set. $k = 31$. Features include position and joint hierarchy

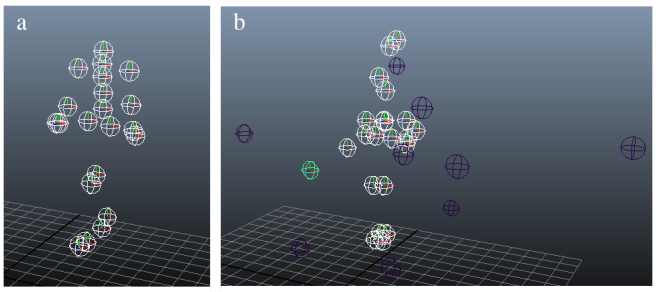

Fig. 4. Comparison of true skeleton pose during an animation and the corresponding pose computed from our learning algorithm. Visualized in Maya. a) Frame 33 of an animation from a bvh file in the CMU dataset. b) Frame 33 of a the learned animation computed for the same skeleton. 22 of the joints, highlighted in white, move along the z direction in the shape of a human. The 4 feet joints (2 on each foot) alternate back and forth, but with a smaller stride than in the ground truth. There appears to be one long arm swinging back and forth.

two motion curves. Our initial attempt was simply the sum of the Euclidean distance between corresponding frames along the curve. We found, however, that a frame-by-frame comparison of this time-series data was extremely susceptible to slight variation in animation phase and period. As our data was highly periodic, we modified our distance metric to use dynamic time warping (DTW) as described in section *IV-E*.

1) Linear Regression: We compare linear regression using the average error: that is, the distance between the animation output of our theta function and the ground truth animation curve. This distance is calculated using dynamic time warping.

Data Type	Average Error
Clusters (without curve)	9.8082
Cluster (with curve data)	48.2566
All points	92.1183

We ran a simple linear regression on every joint in the data-set and found it fairly ineffective. Linear regression on the clustered points (calculated using all features including the animation curve) was significantly better, while clusters (without the animation curve feature) result in by far the most accurate linear regression models.

2) Clustering: The performance of the clustering was measured using the ratio of cluster diameter to cluster spread, where cluster diameter was the average distance between each pair of points in a cluster, and cluster spread was the average distance between each pair of cluster centroids.

Data Type	Range/Spread
All features	0.1612
Absolute position	0.2739
All features + animation curve	0.8913
Animation curve	2.1061

Our initial method used only the absolute position of the joints to compute clusters. We found, however, that with the full set of input features described in section *III-B*, our clusters were much more precise. Additionally, we attempted clustering by animation curve alone, but found that these clusters were not accurate or useful.

3) Cross Validation: To analyze our system as a whole we implemented 2-fold cross validation. We split the data 80/20 for training/testing sets respectively, using the difference between output animation curve (calculated using linear regression) and the known animation sequence as the distance metric. We again found that using the feature set described but excluding animation curves was the most effective for our algorithm.

Feature Set	Average Error
All features	11.3739
All features + animation curve	51.5409
Animation curve	78.9163

VI. Future Work

There is much that can be done to improve and test our learning algorithms. We first hope to train our system on a more diverse data set. The skeletons we found currently share

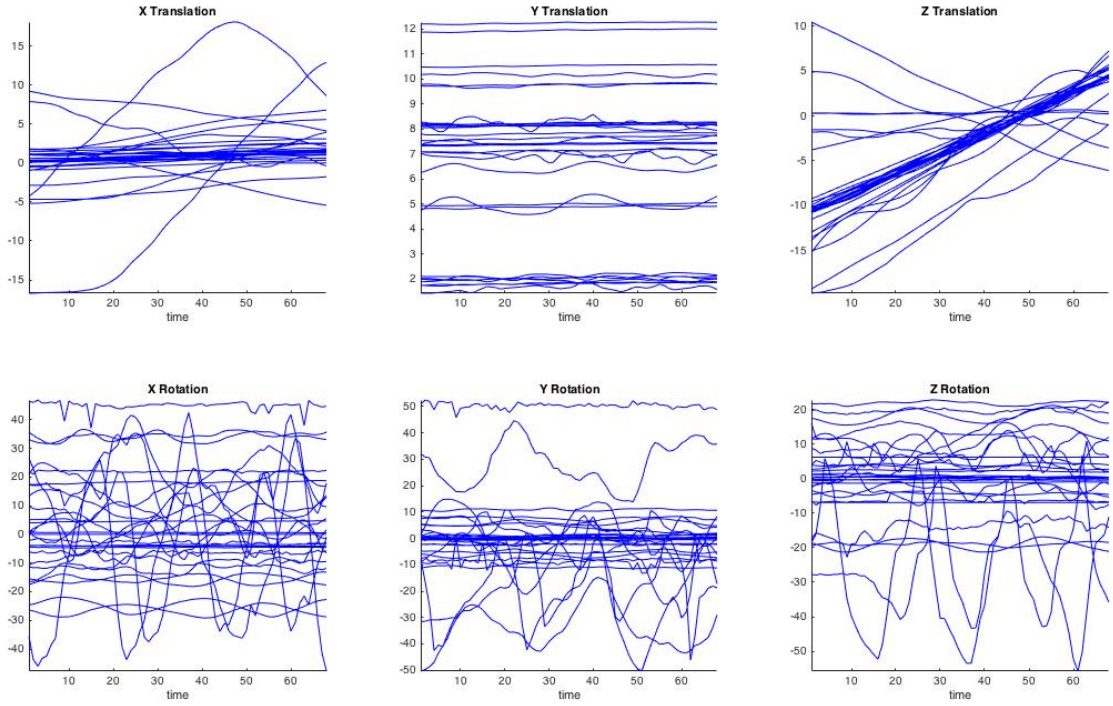

Fig. 5. Animation curves computed from linear regression on a test skeleton taken from the CMU data set. This test skeleton is the same skeleton whose true animation curves are Figure 1

similar joint and parenting structure, but we would like our system to be robust to widely varying rig structures.

Both k-means clustering and linear regression do not converge reliably. K-means clustering must be initialized with the same set of joints in order to converge on the same clusters. Since the animation data is periodic, we could add features from the frequency domain to cluster over. This may improve convergence of the clustering with random initialization. Linear regression requires hand tuning parameters in order to get the most reasonable result.

We should use cross correlation to determine the time lag between the animation curves and then offset the curves appropriately before running linear regression. We will likely get much stronger signals in our result if we include that step in the preprocessing.

VII. Conclusion

In this paper we proposed an example-based motion synthesis technique for arbitrary rigs. We utilize k-means clustering to categorize the joints, then perform linear regression within each cluster to generate an appropriate animation curve for an input joint. By breaking hierarchical rigs into individual joints, this algorithm is able to generate animation data for any rig configuration. Our approach allows users without access to expensive motion capture hardware to utilize free on-line motion capture data in order to animate an input rig of their choice. There are several interesting refinements to this system, from more precise clustering to cross-correlation, that we hope to address in future iterations of this project.

Acknowledgment

The authors would like to thank Sam Corbett-Davies, Nikhil Parthasarthy, the CS 229 Staff, and Professor Andrew Ng.

References

[1] C. K. Liu and Z. Popović, "Synthesis of complex dynamic character motion from simple animations," in *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '02. New York, NY, USA: ACM, 2002, pp. 408–416.

[2] P. Hämäläinen, S. Eriksson, E. Tanskanen, V. Kyrki, and J. Lehtinen, "Online motion synthesis using sequential monte carlo," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 51:1–51:12, Jul. 2014.

[3] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," in *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '02. New York, NY, USA: ACM, 2002, pp. 473–482.

[4] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard, "Interactive control of avatars animated with human motion data," in *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '02. New York, NY, USA: ACM, 2002, pp. 491–500.

[5] B. Thuraisingham, B. Prabhakaran, L. Khan, and K. W. Hamlen, "A database inference controller for 3D motion capture databases," *International Journal of Information Security and Privacy (IJISP)*, 2012, forthcoming.

[6] O. Arikan and D. A. Forsyth, "Interactive motion generation from examples," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 483–490, Jul. 2002.

[7] O. Arikan, D. A. Forsyth, and J. F. O'Brien, "Motion synthesis from annotations," in *ACM SIGGRAPH 2003 Papers*, ser. SIGGRAPH '03. New York, NY, USA: ACM, 2003, pp. 402–408.

[8] P. F. Felzenszwalb and D. P. Huttenlocher, "Pictorial structures for object recognition," *Int. J. Comput. Vision*, vol. 61, no. 1, pp. 55–79, Jan. 2005.

[9] M. Pawan Kumar, P. H. Torr, and A. Zisserman, "Learning layered motion segmentations of video," *Int. J. Comput. Vision*, vol. 76, no. 3, pp. 301–319, Mar. 2008.

[10] M. C. Burl, M. Weber, and P. Perona, "A probabilistic approach to object recognition using local photometry and global geometry," in *Proceedings of the 5th European Conference on Computer Vision-Volume II - Volume II*, ser. ECCV '98. London, UK, UK: Springer-Verlag, 1998, pp. 628–641.

[11] A. Fod, M. J. Mataricacute, and O. C. Jenkins, "Automated derivation of primitives for movement classification," *Autonomous Robots*, vol. 12, no. 1, pp. 39–54, 2004.

[12] J. Barbič, A. Safonova, J.-Y. Pan, C. Faloutsos, J. K. Hodgins, and N. S. Pollard, "Segmenting motion capture data into distinct behaviors," in *Proceedings of Graphics Interface 2004*, ser. GI '04. School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada: Canadian Human-Computer Communications Society, 2004, pp. 185–194.

[13] T. Kwon and S. Y. Shin, "Motion modeling for on-line locomotion synthesis," in *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '05. New York, NY, USA: ACM, 2005, pp. 29–38.

[14] Cmu graphics lab motion capture database. [Online]. Available: http://http://mocap.cs.cmu.edu/

[15] A. Wetzler. Linear blend skinning. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/43039-linear-blend-skinning/content/loadbvh.m

[16] J. Rebello. Linear regression with multiple variables without regularization. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/42768-linear-regression-with-multiple-variables-without-regularization/

[17] Q. Wang. Dynamic time warping (dtw). [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/43156-dynamic-time-warping–dtw-

[18] K. H. Lee, M. G. Choi, and J. Lee, "Motion patches: Building blocks for virtual environments annotated with motion data," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 898–906, Jul. 2006.