# Animation Generation with a Low-Dimensional Simplicial Complex

1st Brian Gauch
*EECS*
*Vanderbilt University*
Nashville, TN
brian.gauch@vanderbilt.edu

2nd Alan Peters
*EECS*
*Vanderbilt University*
Nashville, TN
alan.peters@vanderbilt.edu

***Abstract*—Given motion capture training data, we control an animated character by reducing the dimensionality of the pose space, finding a simplicial complex using Delaunay triangulation, creating a motion graph using the simplicial complex, and then using the motion graph and simplicial complex for distance estimates and interpolation respectively. Using a simple pathfinding algorithm, we compare the above model to a simpler model where k-NN is use to induce a motion graph and interpolate.**

***Index Terms*—animation, motion graph, manifold, Delaunay triangulation, simplicial complex**

## I. Introduction

Realistic animation of human motion is of crucial importance to the film and video game industries, and has applications elsewhere such as robotics and biomechanics. Animation of human motion is particularly challenging because the models have a fairly large number of joints and because viewers can easily detect unrealistic motions. Due to these difficulties, this area has received much research attention within Computer Graphics [1] [2].

When creating animations, a large portion of artist effort is put into so-called “inbetweening”, where start and end poses are known and the frames in between need to be generated. This has been an important area of Computer Graphics for some time [3]. At its most basic, inbetweening can be done with a linear interpolation, but these kinematic models do not lead to very realistic motions. More recently, animators incorporate a dynamics model that may be learned from motion capture [4].

Our goal is to generate realistic human motion animations given a small number of artist-generated or motion captured training motions. To do this we synthesize the intermediate skeletal positions by blending neighboring motions. Unlike the distance metric approaches used by some algorithms to find neighboring points in the state space, e.g. k-NN, we consider two poses to be neighbors if they are both vertices of the same simplex of the Delaunay triangulation.

## II. Related Work

### A. Dimensionality Reduction

*1) Dimensionality Reduction Motivation:* There has been significant prior work in modeling and generating motions, on which our work builds. Many motion modeling algorithms' speeds depend not only on the number of data points $n$, but also on the dimensionality of the state space. Examples of areas in which dimensionality is critical include clustering, modeling and controlling a dynamical system, and machine learning. While adding dimensions to the input data ideally provides useful information, the reduction in speed associated with higher dimensionality is typically referred to as the “curse of dimensionality”. This creates a temptation to hand pick the input features, limiting them to those that seem likely to be important in some sense. Dimensionality reduction techniques aim to eliminate the need for hand-picking features, by preprocessing input data and selecting those dimensions which are deemed important programmatically. These new dimensions need not be (and indeed, rarely are) a subset of the original dimensions. Classical techniques like Principle Component Analysis limit new features to be linear combinations of input features, but many newer techniques do not.

*2) Dimensionality Reduction Methods:* Perhaps the best known dimensionality reduction technique is Principle Component Analysis [5] (PCA), because it was among the first dimensionality reduction methods, and is still one of the fastest. While naïve PCA is $O(n^3 + D^3)$, it may be as fast as $O(ndD)$ where $D$ is the dimensionality of the data space and $d$ is the dimensionality of the latent space [6]. Most other common dimensionality reduction methods like Multi-Dimensional Scaling [7] (MDS), ISOMAP [8], local linear embedding [9] (LLE), and Laplacian eigenmaps [10] are $O(n^3)$ because they involve an eigendecomposition of an $n$-by-$n$ matrix [11] [12]. Other methods include local tangent space alignment [13], Gaussian Process Latent Variable Models (GPLVM) [14], and Gaussian Process Dynamical Models (GPDM) [15].

### B. Motion Models

Regardless of how we reduce dimensionality, the key to animation generation is the motion model. In general we may wish to allow exaggerated or cartoony physics [16]. Some human motion tracking algorithms compute a maximum likelihood estimator (MLE) of the next pose given a sequence of prior poses. Because our work focuses on animation generation, we are interested in the probabilities of pose sequences. Some animation generation algorithms build motion models from a discrete graphical representation of training data. Arikan et al. [17] give a good introduction to motion graph models and how to generate animations from them. The model is then augmented by adding edges to the motion graph, $G$, interpolating between nodes to create new nodes, or both. Ren et al. [18] add edges to $G$ to minimize the lengths of shortest paths, and ask the user to verify the quality of resulting animations periodically. Zhao et al. [19] interpolate between whole sequences at a time, to produce a set of candidate nodes and edges. They then use a threshold based on dynamical probability to determine which to add to $G$. Levine et al. [20] also add nodes and edges to $G$. They simulate many user inputs when creating their controller, and create new nodes in regions that are frequently traveled. They add edges such that each node is connected to its k-NN in the latent space.

*1) Motion interpolation and generation:* Motion interpolation can be done while each training sequence is considered separately, by e.g., finding a smooth function that passes through each of the training data points. Motion blending means interpolating between two or more training sequences, e.g., creating a sequence that is a piecewise linear combination of two training sequences. Shin et al. [21] do motion blending in the latent spaces derived by various dimensionality reduction methods.

*a) :* Motion interpolation and motion blending are both relevant to animation in the film industry, but in the video game industry there is more interest in animations as a function of pose and user control signal. Such a policy is called a character controller. A controller should cause the character to respond quickly to a change of control signal. However, searching the motion graph for candidate responses to the control signal [17] and evaluating both the motion quality and control quality [22] of each candidate motion is generally too slow. Therefore, significant precomputation is usually needed to achieve the desired runtime speed, so that at runtime the controller does little more than a table lookup. Note that the size of the space of control signals must be small for the creation of such a table to be feasible. Typically the control signal is one direction (e.g., walking direction) or the location of the "end effector", i.e. one joint of interest, e.g., a hand. Control of an end effector is the well-studied problem of inverse kinematics, used in robotics. Agrawal and van de Panne [23] presented a method to get foot positions from higher-level task-specific plans, then use those foot positions to generate a complete animation.

*b) :* There is a tradeoff in the creation of a motion graph for an animation or character controller between motion quality and control quality [22]. If control quality is to be emphasized, as Mccann et al. suggest for the case of video game character control [22], then many edges should be added to the motion graph. If motion quality is to be emphasized, then the motion graph should only have a few, high-quality edges beyond the edges in the training data. The tradeoff typically takes the form of a threshold for edge motion quality.

*c) :* Given a motion graph $G$, creating a controller can be relatively simple. A separate Markov Decision Process (MDP) is applied to $G$ for every possible control signal, or for a dense sampling of possible control signals if there is a distance metric in the control space. Mccann et al. [22] compare a MDP controller to other simpler, faster controllers.

*d) :* Levine et al. [20] also use an MDP-based controller. After the expected rewards of the MDP have finished propagating on the finite $G$, Their algorithm interpolates reward between nodes, allowing their controller to go to points in the latent space which are not on $G$.

### C. Manifold reconstruction

*1) n-Dimensional Delaunay Triangulation:* In the motion models above, candidate edges for $G$ are typically found using a distance metric, whether in the original space or in the latent space. Similarly, neighbors for interpolation can be found via k-NN. We instead use a simplicial complex [24], where each simplex can be thought of as a neighborhood. We find a simplicial complex using Delaunay triangulation.

Delaunay triangulations were originally defined in [25]. We first consider the Delaunay triangulation of a set of points in a two-dimensional space. We are interested in the following properties:

1) Given a set of two-dimensional points, a Delaunay triangulation is a set of triangles that covers the region in the plane bounded by the convex hull of the points.
2) The set of triangle vertices and the set of input points are equal.
3) Among the coverings that satisfy the two above conditions, a Delaunay triangulation is a covering that maximizes the minimum interior angle of its triangles.

This last property makes the Delaunay triangulation unique (i.e., it does not depend on the order of input points) under certain assumptions. Delaunay "triangulation" is a bit of a misnomer when it is generalized to higher dimensions, in which case a Delaunay triangulation produces a simplicial complex [26]. Variations of Delaunay triangulations have been used in Computer Graphics to generate triangular meshes [27] [28], from sets of points [29]. More recently, Delaunay triangulations have been used in manifold learning [30] [31].

## III. Approach

Our approach to generating animations that start at an arbitrary point $a$ in the pose space comprises the following steps.

(a) Reduce the dimensionality of the pose space to produce a lower-dimensional latent space. Project point $a$ from the latent space to point $a'$ in the latent space.
(b) Create a graph of the training data in the latent space using either k-NN or Delaunay triangulation.
(c) Use the graph to define a neighborhood function over the latent space.
(d) Find the vectors in the latent space which are projected directional derivatives of the training data.
(e) Interpolate the latent vectors to define a vector field.
(f) Find a path as follows: Sample the vector field at $a'$, move with the same direction and magnitude as that vector, and

sample the vector field at that new location. Continue until the stopping condition is reached.

(g) Reproject the path from the latent space back into the pose space, where it the result should resemble the original training animations.

## A. Dimensionality Reduction

We use PCA for dimensionality reduction because it is fast, and because it is straightforward to reproject back into to pose space where we eventually want to see our animations.

## B. Motion Graph

We use a collection of training sequences to define our motion graph $G$. The first step is to add nodes and directed edges corresponding to each of the "clips" in our training data. We then extend this graph by finding the Delaunay triangulation of these nodes, or by adding edges using k-NN.

We have implemented n-dimensional Delaunay triangulation using the algorithm described by Watson [26]. One step of Watson's algorithm (step B) is to find the circumspheres of new simplices; However Watson does not go into detail about how this done. We solved this subproblem using an algorithm described by Daly [32]. We generalize their "Center of Circumscribed Sphere" algorithm to $d$ dimensions. However, for the results presented in this paper, we always reduce dimension to 2, so a simpler implementation of Delaunay triangulation which only works for 2 dimensions would have sufficed.

Using k-NN, we add an undirected edge between points $p_1$ and $p_2$ if $p_2$ is one of $p_1$'s k nearest neighbors or vice versa. Using the Delaunay triangulation, there is an undirected edge between points $p_1$ and $p_2$ if there is at least one simplex of the Delaunay triangulation such that $p_1$ and $p_2$ are vertices of that simplex.

The next step is to direct the edges added by Delaunay or k-NN so the edge points in roughly the same direction as the training data. To do this, consider an undirected edge between training data points $p_1$ and $p_2$, with associated velocities $v_1$ and $v_2$. Let $v_{target} = \frac{(v_1+v_2)}{2}$ represent a vector that is in the desired direction. Take the dot product of $v_{target}$ with the vectors ($p_2$-$p_1$) and ($p_1$-$p_2$). Keep whichever edge has a positive dot product with $v_{target}$. We call the resulting directed graphs $G'_{Delaunay}$ and $G'_{k-NN}$, or simply $G'$ when the method of adding edges need not be specified. Examples of $G'_{k-NN}$ and $G'_{Delaunay}$ can be seen in Figure 1 and Figure 2 respectively.

Ideally, we would like to be able to find a transition from any animation frame in the training data to any other. Since such a transition amounts to a path on $G'$, all pairwise graphical distances need to be calculated, which requires that $G'$ be a single connected component.

If $G'$ instead has undirected edges, The graph $G'_{Delaunay}$ induced by the Delaunay triangulation would be a single connected component, because the simplices of a Delaunay triangulation cover the convex hull of the training data. $G'_{Delaunay}$ has $O(n \cdot d)$ edges where $d$ is the dimensionality of the latent space. The graph $G'_{k-NN}$ induced by k-NN is

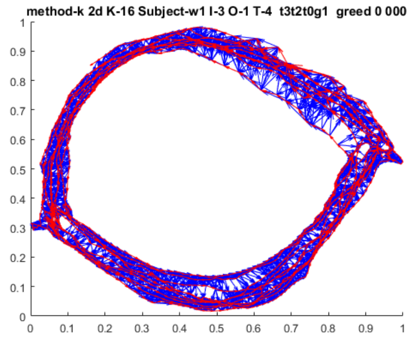

Fig. 1. Three folds of training data (red) and resulting edges of $G'_{k-NN}$ (blue).

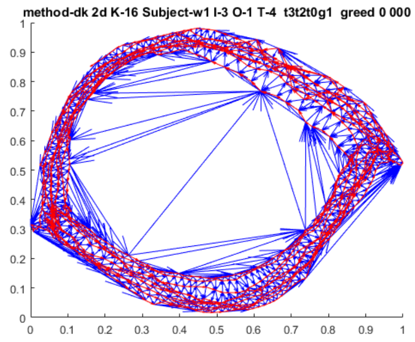

Fig. 2. Three folds of training data (red) and resulting edges of $G'_{Delaunay}$ (blue).

not necessarily a single connected component for $k \leq \frac{n-1}{2}$. To see this, consider the case where our data is divided into two distinct clusters, where the distance between the clusters greater than the distance between any two members of the same cluster. Some algorithms using k-NN get around this issue by adding edges to connect the graph [33].

If $G'$ has directed edges then it has a single component only if its undirected version is likewise one component. For the directed graph to be connected, we also need the data to be cyclical. More precisely, we need the following: Consider the set of time series $T_1, T_2, ..., T_\tau$ that make up $X$. Consider the directed bipartite graph $H$ such that for each $T_i$ there is a node in the left set of $H$ corresponding to the first element of $T_i$ and a node in the right set of $H$ corresponding to the last element of $T_i$. Let there be an edge from $a$ to $b$ in $H$ iff there is a path from $a$ to $b$ in $G'$. Note that this implies an edge from the beginning of each $T_i$ to its end. Then we claim that $G'$ has a single connected component iff $H$ has a single connected component.

Proof: Suppose $H$ is connected (i.e., has a single connected component) and $G'$ has at least two connected components. Then there is some pair of nodes $a, b \in G'$ such that there is no path from $a$ to $b$. Let $a \in T_a, b \in T_b$. Because $G'$ includes a path graph for each $T_i$, $\forall t \in T_i$ there is a path in $G'$ from $t$ to the $t_F$, the last element of $T_i$. $\forall t \in T_i$ there is a path in $G'$ from $t_0$, the first element of $T_i$, to $t$. Therefore, there is a path in $G'$ from $a$ to the $a_F$, last element of $T_a$, and there is a path in $G'$ from $b_0$, the first element of $T_b$, to $t$. Because $H$ is connected, there is a path in $H$ from $a_F$ to $b_0$. Because the edges of $H$ are a subset of the transitive edges of $G'$, there is a path between nodes in $H$ only if there is a path between those nodes in $G'$. Therefore, there is a path in $G'$ from $a_F$ to $b_0$. Finally, there is a path from $a$ to $b$: $a \to a_F \to b_0 \to b$. This is a contradiction to our assumption that $G'$ has at least two connected components. Therefore, $H$ connected implies $G'$ is also connected.

To make the data more cyclical, we remove the root coordinates from the motion capture data, so that, e.g., a walk cycle stays in place. That is, we expect $H$ to have more edges if each time series ends close to where it starts. However, removing the root coordinates does not guarantee that $H$ is connected.

### C. Neighborhoods

After constructing $G'$, the next step is to use it to define a function which, given a latent point, returns a set of neighbors which are elements of $G'$. Within each neighborhood we treat a point as a weighted combination of its neighbors.

For $G'_{k-NN}$, the neighborhood of a point is the k-NN of that point. For $G'_{Delaunay}$, the neighborhood of a point is the set of vertices of its containing simplex. In the case of $G'_{Delaunay}$, the neighborhoods returned by the neighborhood function form a topology. Specifically, the simplicial complex returned by Delaunay triangulation is the "geometric realization" [34] of an abstract simplicial complex, which has a topology. oNote that for $G'_{Delaunay}$, this function is only defined over the convex hull of the training data. To calculate weights within a neighborhood, we use inverse distance weighting. Specifically, we calculate the Euclidean distances from an arbitrary point $p$ to the $N$ points in the neighborhood, and the weight for each neighbor is given by $1/D^2$ normalized such that weights sum to 1.

If we use $G'_{Delaunay}$, our neighborhood function has discontinuities along the boundaries between simplices. If we use $G'_{k-NN}$, our neighborhood function has discontinuities along the boundaries between the cells of the corresponding $k_{th}$-order Voronoi diagram [35]. In general, we expect the neighborhood function induced by $G'_{k-NN}$ to have more discontinuities, but we expect the discontinuities in the neighborhood function induced by $G'_{Delaunay}$ to be more pronounced, because with fewer neighbors, the weight of the neighbor being dropped in the distance-weighted interpolation is usually higher.

### D. Training Data Vectors

The next step is to use the training data to generate a vector field so new motions can be estimated and generated, We start by associating a vector $v_{x_i}$ with each training data point $x_i$ in a time series $X$, which is a discrete approximation of the derivative of $X$ at that point. We use the following: $v_{x_i} = \frac{x_{i+1}-x_i}{2}$ where $x_{i+1}$ exists and $v_{x_i} = \frac{x_i - x_{i-1}}{2}$ where it does not (at the end of a time series).

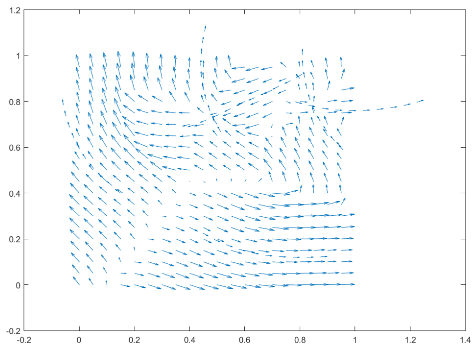

Fig. 3. Vector field generated by creating vectors from consecutive pairs in the training data and interpolating using k-NN.

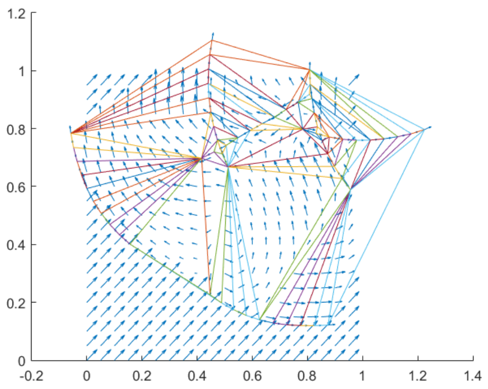

Fig. 4. Vector field generated by creating vectors from consecutive pairs in the training data and interpolating using the Delaunay triangulation.

### E. Vector Field

We then use the neighborhood function and local weights to interpolate among these vectors, creating a vector field which is defined wherever the neighborhood function is defined. See Figure 3 and Figure 4 for visualizations of the vector fields induced by k-NN and the Delaunay triangulation respectively, applied to toy data.

### F. Finding Paths

Following our vector field should produce plausible animations, as long as we stay near our training data. That is, we sample the vector field, move according to that vector, and sample again at the new location until we are satisfied with the length of the path. See Figure 5 and Figure 6 for paths/animations found in the low-dimensional latent space using k-NN and the Delaunay triangulation respectively.

### G. Reprojecting

Our last step is to move from the latent space back to the pose space. To do this, we essentially undo PCA, albeit with some amount of data loss which depends on how many dimensions we reduced to and whether the original dimensions were linear combinations of one another. Because PCA works by applying a matrix transform, we can simply take the inverse of that transform and apply it. See Figure 7 for a generated

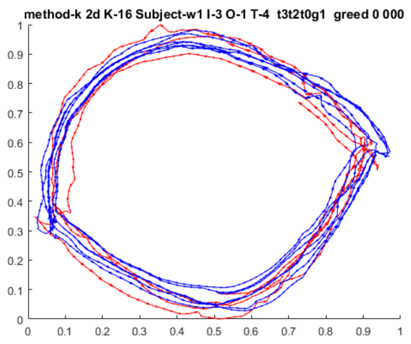

Fig. 5. The original path, which was not used for training (red) and the path found using $G'_{k-NN}$ (blue).

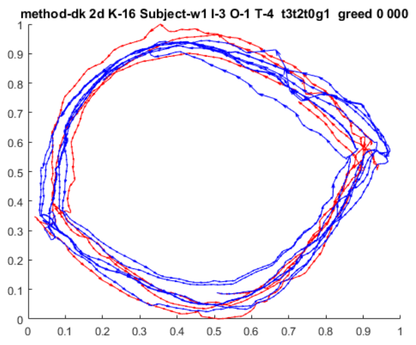

Fig. 6. The original path, which was not used for training (red) and the path found using $G'_{Delaunay}$ (blue).

walking animation reprojected back into the original data space, where animations are much prettier.

## IV. Evaluation

In order to compare the abilities of the $G'_{k-NN}$ and $G'_{Delaunay}$ models at finding new, plausible animations, we perform a cross-fold validation of sorts.

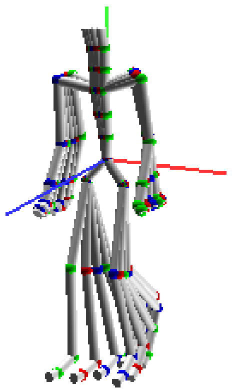

Fig. 7. Walking animation generated by reprojecting a motion found by following the vector field induced by $G'_{Delaunay}$.

TABLE I
Details of the test sets.

| Dataset | # Dim | # Seq | Sequence Lengths | | | | Total # Nodes |
|---|---|---|---|---|---|---|---|
| w1 | 56 | 4 | 443 | 360 | 455 | 454 | 1712 |
| w2 | 56 | 4 | 347 | 520 | 269 | 282 | 1418 |
| r1 | 56 | 4 | 167 | 167 | 175 | 160 | 669 |
| r2 | 56 | 4 | 258 | 139 | 200 | 163 | 760 |

### A. Data sets and Metrics

We use data from CMU Graphics Lab Motion Capture Database, a collection of motion capture data. In particular, we collected 16 time series and broke them into four subsets, each containing four related time series. Two were labeled walking (w1 and w2), and two were labeled running (r1 and r2); w2 and r2 include turning behavior. More details about the sequences used appear in Table I.

We define two metrics to evaluate the similarity of a generated animation to a "true" animation, assuming that both animations share the same starting point. Error F is the distance from the generated animation to the final point of the true animation during the closest approach. Error A is the average distance from the true animation to the (closest point on the) generated animation, and vice versa, i.e., average distance from the generated animation to the (closest point on the) true animation. Error A is intuitively similar to the area between the two curves.

### B. Experimental design

Our cross-fold validation of animations works as follows: For each of our four subsets, we train on all but one of the sequences (three sequences) to learn $G'_{k-NN}$ and $G'_{Delaunay}$ and the corresponding and vector fields. We then generate a path in the latent space by starting at the same point as the fold that was left out of training and following the vector field. On the rare occasion that our path takes us outside of the convex hull of the training data, and the vector field induced by Delaunay triangulation is therefore undefined, we use k-NN as a fallback. Finally, for each subset, for each method (Delaunay or k-NN), for each of the four folds, we measure Error A and Error F of the generated path where the left out fold is consider the true animation.

### C. Results

Table II shows the results of our experiments. It reports the medians of both metrics from each of the 4-fold cross-validations. You can see that the $G'_{Delaunay}$ model gave better results than $G'_{k-NN}$ overall, with medians of 0.068 vs 0.079 (Error A) and 0.041 vs 0.042 (Error F) across all subsets. $G'_{Delaunay}$ also performed better than $G'_{k-NN}$ within five of the eight combinations of subset and metric.

## V. Discussion

We present a new method of creating a motion graph from motion capture data, based on Delaunay triangulation. Our method first finds the Delaunay triangulation of the training data points, then defines a vector field over the convex hull

TABLE II
MEDIAN ACCURACY OF DELAUNAY VS K-NN ON 4 DATASETS USING TWO DIFFERENT METRICS, ERROR A AND ERROR F.

| | | w1 | w2 | r1 | r2 | all data |
|---|---|---|---|---|---|---|
| Error A | Delaunay | 0.023 | 0.073 | 0.749 | 0.068 | 0.068 |
| | k-NN (k=16) | 0.022 | 0.105 | 0.099 | 0.118 | 0.079 |
| Error F | Delaunay | 0.02 | 0.043 | 0.038 | 0.067 | 0.041 |
| | k-NN (k=16) | 0.057 | 0.114 | 0.044 | 0.024 | 0.042 |

of the training data by interpolating among vertices of the containing simplex.

Finally, given a start point, it generates an animation by following the vector field. Our experiments showed that our approach outperformed k-NN, as measured by Error A and Error F.

### A. Future Work

In future, we plan to extend our Delaunay triangulation method to higher dimensions, and to investigate better pathfinding methods. In particular, we are interested in the use of all-pairs shortest paths to find paths towards some target point, such as the final point of a fold in cross-fold validation. We also plan to consider combining the k-NN and Delaunay models, perhaps by defining the neighborhood of k-Delaunay to be the union of the neighborhoods of Delaunay and k-NN.

## ACKNOWLEDGMENT

The data used in this project was obtained from mocap.cs.cmu.edu. The database was created with funding from NSF EIA-0196217.

## REFERENCES

[1] Luis Molina Tanco and Adrian Hilton. Realistic synthesis of novel human movements from a database of motion capture examples. In *Human Motion, 2000. Proceedings. Workshop on*, pages 137–142. IEEE, 2000.

[2] Xiaolei Lv, Jinxiang Chai, and Shihong Xia. Data-driven inverse dynamics for human motion. *ACM Transactions on Graphics (TOG)*, 35(6):163, 2016.

[3] William T Reeves. Inbetweening for computer animation utilizing moving point constraints. *ACM SIGGRAPH Computer Graphics*, 15(3):263–269, 1981.

[4] Layale Saab, Oscar E Ramos, François Keith, Nicolas Mansard, Philippe Soueres, and Jean-Yves Fourquet. Dynamic whole-body motion generation under rigid contacts and other unilateral constraints. *IEEE Transactions on Robotics*, 29(2):346–362, 2013.

[5] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[6] Sam Roweis. Em algorithms for pca and spca. *Advances in neural information processing systems*, pages 626–632, 1998.

[7] Joseph B Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[8] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[9] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[10] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, volume 14, pages 585–591, 2001.

[11] Xiaoming Huo, Xuelei Sherry Ni, and Andrew K Smith. A survey of manifold-based learning methods. *Recent advances in data mining of enterprise data*, pages 691–745, 2007.

[12] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[13] Zhen-yue Zhang and Hong-yuan Zha. Principal manifolds and nonlinear dimensionality reduction via tangent space alignment. *Journal of Shanghai University (English Edition)*, 8(4):406–424, 2004.

[14] Neil Lawrence, Matthias Seeger, and Ralf Herbrich. Fast sparse gaussian process methods: The informative vector machine. In *Proceedings of the 16th Annual Conference on Neural Information Processing Systems*, number EPFL-CONF-161319, pages 609–616, 2003.

[15] Jack M Wang, David J Fleet, and Aaron Hertzmann. Gaussian process dynamical models for human motion. *IEEE transactions on pattern analysis and machine intelligence*, 30(2):283–298, 2008.

[16] Yunfei Bai, Danny M Kaufman, C Karen Liu, and Jovan Popović. Artist-directed dynamics for 2d animation. *ACM Transactions on Graphics (TOG)*, 35(4):145, 2016.

[17] Okan Arikan, David A Forsyth, and James F O'Brien. Motion synthesis from annotations. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 402–408. ACM, 2003.

[18] Cheng Ren, Liming Zhao, and Alla Safonova. Human motion synthesis with optimization-based graphs. In *Computer Graphics Forum*, volume 29, pages 545–554. Wiley Online Library, 2010.

[19] Liming Zhao and Alla Safonova. Achieving good connectivity in motion graphs. *Graphical Models*, 71(4):139–152, 2009.

[20] Sergey Levine, Jack M Wang, Alexis Haraux, Zoran Popović, and Vladlen Koltun. Continuous character control with low-dimensional embeddings. *ACM Transactions on Graphics (TOG)*, 31(4):28, 2012.

[21] Hyun Joon Shin and Jehee Lee. Motion synthesis and editing in low-dimensional spaces. *Computer Animation and Virtual Worlds*, 17(3-4):219–227, 2006.

[22] James McCann and Nancy Pollard. Responsive characters from motion fragments. In *ACM Transactions on Graphics (TOG)*, volume 26, page 6. ACM, 2007.

[23] Shailen Agrawal and Michiel van de Panne. Task-based locomotion. *ACM Transactions on Graphics (TOG)*, 35(4):82, 2016.

[24] Edwin H Spanier. *Algebraic topology*, volume 55. Springer Science & Business Media, 1994.

[25] Boris Delaunay. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7(793-800):1–2, 1934.

[26] David F Watson. Computing the n-dimensional delaunay tessellation with application to voronoi polytopes. *The computer journal*, 24(2):167–172, 1981.

[27] Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 415–421. ACM, 1998.

[28] Nina Amenta and Marshall Bern. Surface reconstruction by voronoi filtering. In *Proceedings of the fourteenth annual symposium on Computational geometry*, pages 39–48. ACM, 1998.

[29] Jules Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5(4):341–355, 1988.

[30] Jean-Daniel Boissonnat and Arijit Ghosh. Manifold reconstruction using tangential delaunay complexes. *Discrete & Computational Geometry*, 51(1):221–267, 2014.

[31] Siu-Wing Cheng, Tamal K Dey, and Edgar A Ramos. Manifold reconstruction from point samples. In *SODA*, volume 5, pages 1018–1027, 2005.

[32] Patrick W Daly. The tetrahedron quality factors of csds. *Max-Planck-Institut für Aeronomie*, 1994.

[33] Martin HC Law and Anil K Jain. Incremental nonlinear dimensionality reduction by manifold learning. *IEEE transactions on pattern analysis and machine intelligence*, 28(3):377–391, 2006.

[34] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 19–26. ACM, 1993.

[35] Atsuyuki Okabe. *Spatial tessellations*. Wiley Online Library, 1992.