An Analysis of Motion Blending Techniques

Andrew Feng[1], Yazhou Huang[2], Marcelo Kallmann[2], and Ari Shapiro[1]

[1] Institute for Creative Technologies, University of Southern California
[2] University of California, Merced

Abstract. Motion blending is a widely used technique for character animation. The main idea is to blend similar motion examples according to blending weights, in order to synthesize new motions parameterizing high level characteristics of interest. We present in this paper an in-depth analysis and comparison of four motion blending techniques: Barycentric interpolation, Radial Basis Function, K-Nearest Neighbors and Inverse Blending optimization. Comparison metrics were designed to measure the performance across different motion categories on criteria including smoothness, parametric error and computation time. We have implemented each method in our character animation platform SmartBody and we present several visualization renderings that provide a window for gleaning insights into the underlying pros and cons of each method in an intuitive way.

1 Introduction

Motion blending, also known as motion interpolation, is widely used in interactive applications such as in 3D computer games and virtual reality systems. It relies on a set of example motions built either by key-framing animation or motion capture, and represents a popular approach for modifying and controlling high level characteristics in the motions. In essence, similar motion examples are blended according to blending weights, achieving parameterizations able to generate new motions similar to the existing examples and providing control of high level characteristics of interest to the user.

Although several different methods have been proposed in previous works, there is yet to be a one-solves-all method that works best in all scenario. Each method has its own advantages and disadvantages. The preferred motion parameterization for an application highly depends on the application type and required constraints. For locomotion animation, it is more important to ensure the weights vary smoothly as the parameterization generates them, in order to ensure visually pleasing results. On the other hand, a reaching motion forms a non-linear parameter space and requires precise goal-attainment for grasping. Therefore methods with less parametric error are preferred for accuracy. The goal of this work is to throughly analyze several existing methods for motion parameterization and to provide both visual and numerical metrics for discriminating different methods.

We present in this paper an detailed analysis among 4 popular motion blending methods implemented on our character animation platform SmartBody, including Barycentric interpolation, Radial Basis Function (RBF) interpolation, K-Nearest Neighbors (KNN) interpolation, and Inverse Blending (InvBld) optimization [7]. A motion capture dataset was carefully built containing 5 different type of motions, and comparison metrics were designed to measure the performance on different motion categories using criteria including parametrization accuracy of constraint enforcement, computation time and smoothness of the final synthesis. Comparison results are intuitively visualized through dense sampling inside the parametrization (blending) space, which is formulated using the corresponding motion parameters. We believe our results provide valuable insights into the underlying advantages and disadvantages of each blending method.

2 Related Work

Several blending methods have been proposed in the literature to produce flexible character animation from motion examples. Different approaches have been explored, such as: parameterization using Fourier coefficients [20], hierarchical filtering [2] and stochastic sampling [21]. The use of Radial Basis Functions (RBFs) for building parameterized motions was pioneered by Rose et al. [16] in the Verbs and Adverbs system, and follow-up improvements have been proposed [17,15]. RBFs can smoothly interpolate given motion examples, and the types and shapes of the basis functions can be fine tuned to better satisfy constraints for different dataset. The method assigns a function for each blending weight that is the sum of a linear polynomial and a set of radial basis functions, with one radial basis function for each example. If the requested parameters are far from the examples, blending weights will largely rely on the linear polynomial term. We have selected the Verbs and Adverbs formulation as the RBF interpolation method used in the presented comparisons.

Another method for interpolating poses and motions is KNN interpolation. It relies on interpolating the k nearest examples measured in the parametrization space. The motion synthesis quality can be further improved by adaptively adding pseudo-examples in order to better cover the continuous space of the constraint [10]. This random sampling approach however requires significant computation and storage in order to well meet constraints in terms of accuracy, and can require too many examples to well handle problems with multiple constraints.

In all these blending methods spatial constraints are only handled as part of the employed motion blending scheme. One way to solve the blending problem with the objective to well satisfy spatial constraints is to use an Inverse Blending optimization procedure [7], which directly optimizes the blending weights until the constraints are best satisfied. However, a series of smooth variation inside the parametrization space may generate non-smooth variations in the weight space, which lead to undesired jitter artifacts in the final synthesis. Another possible

technique sometimes employed is to apply Inverse Kinematics solvers in addition to blending [4], however risking to penalize the obtained realism.

Spatial properties such as hand placements or feet sliding still pose challenges to the previously mentioned blending methods. Such problems are better addressed by the geo-statistical interpolation method [14], which computes optimal interpolation kernels in accordance with statistical observations correlating the control parameters and the motion samples. When applied to locomotion systems, this method reduces feet sliding problems but still cannot guarantee to eliminate them. The scaled Gaussian Process Latent Variable Model (sGPLVM)[5] provides a more specific framework targeting similar problems with optimization of interpolation kernels specifically for generating plausible poses from constrained curves such as hand trajectories. The approach however focuses on maintaining constraints described by the optimized latent spaces. Although good results were demonstrated with both methods, they remain less seen in animation systems partly because they are less straightforward to implement.

We also analyze the performance of selected blending methods for locomotion parametrization, which is a key problem in character animation. Many blending-based methods have been proposed in the literature for [12,11,8], for interactive navigations with user input [12,1], for reaching specified targets or dodging obstacles during locomotion [18,6], and also for space-time constraints [9,19,13]. Here in this paper we investigate in comparing how well each selected popular blending method works for locomotion sequence synthesis.

In conclusion, diverse techniques based on motion blending are available and several of these methods are already extensively used in commercial animation pipelines for different purposes. In this work we present valuable experimental results uncovering the advantages and disadvantages of four motion blending techniques. The rest of this paper is organized as follows: In Section 3 and 4 we briefly review each blending method and then describe the setup of our experiments and selected metrics for analysis. Section 5 present detailed comparison results, and Section 6 concludes this paper.

3 Motion Parameterization Methods

In general, a motion parameterization method works as a black box that maps desired motion parameters to blending weights for motion interpolation. We have selected four methods (Barycentric, RBF, KNN, and InvBld) for our analysis. This work focuses on comparing the methods that are most widely used in practice due to their simplicity in implementation, but we plan as future work to also include other methods like [5] and [14]. Below is a brief review of the 4 methods selected. Each motion M_i being blended is represented as a sequence of poses with a discrete time (or frame) parameterization t. A pose is a vector which encodes the root joint position and the rotations of all the joints of the character. Rotations are encoded as quaternions but other representations for rotations can also be used. Our interpolation scheme computes the final blended

motion $M(\boldsymbol{w}) = \sum_{i=1}^{k} w_i M_i$, with $\boldsymbol{w} = \{w_1, \ldots, w_k\}$ being the blending weights generated by each of the methods.

Barycentric interpolation is the most basic form for motion parameterization. It assumes that motion parametrization can be linearly mapped to blending weights. While the assumption may not hold for all cases, in many situations this simple approximation does achieve good results. Without loss of generality, we assume a 3D motion parameter space with a set of example motions M_i $(i = 1 \ldots n)$. As the dimension of parametric space goes from 1D to n-D, linear interpolation becomes Barycentric interpolation. Specifically for a 3D parametric space, we construct the tetrahedralization $V = \{T_1, T_2, \ldots T_v\}$ to connect these motion examples M_i in space, which can be done either manually or automatically using Delaunay triangulation. Therefore, given a new motion parameter $\boldsymbol{p'}$ in 3D space, we can search for a tetrahedron T_j that encloses $\boldsymbol{p'}$. The motion blending weights $w = \{w_1, w_2, w_3, w_4\}$ are given as the barycentric coordinates of $\boldsymbol{p'}$ inside T_j. Similar formulations can be derived for 2-D and n-D cases by replacing a tetrahedron with a triangle or a n-D simplex respectively. The motion parameterization $\boldsymbol{p'}$ is not limited to a 3D workspace but can also be defined in other abstract parameterization spaces, with limited ability for extrapolation outside the convex hull.

Radial Basis Function (RBF) is widely used for data interpolation since first introduced in [16]. In this paper we implement the method by placing a set of basis functions in parameter space to approximate the desired parameter function $f(p) = w$ for generating the blend weights. Specifically, given a set of parameter examples $p_1, p_2, \ldots p_n$, $f(p) = w_1, w_2, \ldots w_n$ is defined as sum of a linear approximation $g(p) = \sum_{l=0}^{d} a_l A_l(p)$ and a weighted combination of radial basis function $R(p)$. The function for generating the weight w_i is defined as :

$$w_i(p) = \sum_{j=1}^{n} r_{i,j} R_j(p) + \sum_{l=0}^{d} a_{i,l} A_l(p)$$

where $R_j(p) = \phi(\|p - p_j\|)$ is the radial basis function, and $A_l(p)$ is the linear basis. Here the linear coefficients $a_{i,l}$ are obtained by fitting a hyperplane in parameter space where $g_i(p_j) = \delta_{i,j}$ is one at i-th parameter point and zero at other parameter points. The RBF coefficients $r_{i,j}$ are then obtained by solving the following linear equation from linear approximation to fit the residue error $e_{i,j} = \delta_{i,j} - g_i(p_j)$.

$$\begin{pmatrix} R_1(p_1) & R_1(p_2) & \ldots \\ R_2(p_1) & \ldots & \ldots \\ \vdots & \ldots & \ldots \end{pmatrix} r = e$$

RBF can generally interpolate the example data smoothly, though it usually requires some tuning in the types and shapes of basis function to achieve good results for a specific data set. The parameterization space could also be defined on an abstract space like motion styles [16], with the ability to extrapolate outside the convex hull of example dataset. However the motion quality from such extrapolation may not be guaranteed, especially when p travels further outside the convex hull.

K-Nearest Neighbors (KNN) interpolation finds the k-closest examples from an input parameter point and compute the blending weights based on the distance between the parameter point and nearby examples. Specifically, given a set of parameter examples $p_1, p_2, \ldots p_n$ and a parameter point p', the method first find example

points $p_{n_1}, p_{n_2}, \ldots p_{n_k}$ that are closest to p'. Then the i-th blending weight for p_{n_i} are computed as :

$$w_i = \frac{1}{\|p - p_{n_i}\|} - \frac{1}{\|p - p_{n_k}\|}$$

The blending weights are then normalized so that $w_1 + w_2 + \ldots + w_k = 1$. The KNN method is easy to implement and works well with a dense set of examples in parameter space. However, the result may be inaccurate when the example points are sparse. To alleviate this problem, pseudo-examples are usually generated to fill up the gap in parameter space [10]. A pseudo-example is basically a weighted combination of existing examples and can be generated by randomly sampling the blend weights space. Once a dense set of pseudo-examples are generated, a k-D tree can be constructed for fast proximity query at run-time.

Inverse Blending (InvBld) was designed for precise enforcement of user-specified constraints in the workspace [7]. Each constraint C is modeled with function $e = f^C(\boldsymbol{M})$, which returns the error evaluation e quantifying how far away the given motion is from satisfying constraint C under the given motion parametrization p'. First, the k motions $\{M_1, \ldots, M_k\}$ best satisfying the constraints being solved are selected from the dataset, for example, in a typical reaching task, the k motion examples having the hand joint closest to the target will be selected. An initial set of blending weights w_j $(j = \{1, \ldots, k\})$ are then initialized with a radial basis kernel output of the input $e_j = f^C(M_j)$. Any kernel function that guarantee smoothness can be used, as for example kernels in the form of $exp^{-\|\boldsymbol{e}\|^2/\sigma^2}$. Weights are constrained inside $[0, 1]$ in order to stay in a meaningful interpolation range, they are also normalized to sum to 1. The initial weights $\boldsymbol{w}$ are then optimized with the goal to minimize the error associated with constraint C:

$$e^* = min_{w_j \in [0,1]} f^C \left(\sum_{j=1}^{k} w_j M_j \right).$$

Multiple constraints can be accounted by introducing two coefficients n_i and c_i for each constraint C_i, $i = \{1, \ldots, n\}$, and then solve the multi-objective optimization problem that minimizes a new error metric composed of the weighted sum of all constraints' errors: $f(\boldsymbol{M}(\boldsymbol{w})) = \sum_{i=1}^{n} \left(c_i \; n_i \; f^{C_i} \left(\boldsymbol{M}(\boldsymbol{w}) \right) \right)$, where c_i is used to prioritize C_i, and n_i to balance the magnitude of the different errors.

4 Experiments Setup

We have captured five different categories of motion examples of the following actions: reach, jump, punch kick and locomotion. Corresponding feature is formulated for each motion category to define a parameterization space. We further defined three metrics to be used in our evaluation: computation time, parameterization accuracy (or parametric error) and smoothness. The table and figures in Fig 1 gives an overview of the datasets.

4.1 Performance Metrics

The main application of motion parameterization is to synthesize new motions interactively based on input parameters. In order to numerically compare the methods, we defined three metrics: computation time, parametrization accuracy and smoothness.

category	number of examples	parametrization	joint paramtrized	note
Reach	24	$p=(x,y,z)$	wrist	full body reaching with bending down and turning around
Punch	20	$p=(x,y,z)$	wrist	start and end with fighting stance; targets are mostly in front
Kick	20	$p=(x,y,z)$	ankle	start and end with fighting stance; p is ankle position at kick apex
Jump	20	$p=(d,\theta,h)$	base	d: jump distance; θ: jump direction; h: max height during jump
Locomotion	20	$p=(v_f,\omega,v_s)$	base	v_f: walk forward speed; ω: turning rate; v_s: walk sideways speed

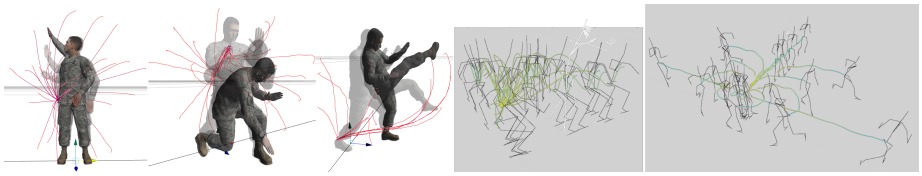

Fig. 1. An overview of the motion capture dataset used for our analysis, from left to right: reach, punch, kick, jump and locomotion.

Parametrization Accuracy: While each motion parameterization method can generate a unique set of blending weights given some input parameters, there is no guarantee that the blended motion will satisfy the input parameters. We define the parametric error as the squared difference between the desired input parameter and the actual motion parameter derived from the blended motion. Depending on the type of applications, this error may be of less importance: for application that requires precise end-effector control such as reaching, punching and kicking a given target, the parameter error would directly determine whether the result is valid for a given task; for abstract motion parameterization space such as emotion (happy walk v.s. sad walk) or style (walk forward v.s. walk sideways) control, only qualitative aspects of motion are of interest.

Computation Time is divided into pre-computation phase and run-time computation phase. Pre-computation time is the amount of time taken for a method to build the necessary structures and required information to be used in the run-time phase. While this may usually be negligible for Inverse Blending or Barycentric, it may require significant amount of time for KNN and RBF depending on the number of pseudo examples or size of the dataset. A method require little to no pre-computation is more flexible in changing the example data on-the-fly, which can be beneficial for applications that require on-line building and adjusting motion examples [3]. Run-time computation phase is the time required for the method to compute the blending weights based on given input parameters, which reflects the real-time performance.

Smoothness determines whether the blending weights would change smoothly when motion parametrization varies. This metric is of more importance when parameters are changed frequently during motion synthesis. Specifically speaking, smoothness may be less required for motions like reach, kick and punch where parametrization usually stays constant during each action execution. However it is critical for other motion parametrization such as locomotion where parameters may need to be changed continuously even within each locomotion gait. And for such applications, jitter artifacts would occur and degrade the quality of synthesized motions if smoothness can not be guaranteed. We numerically define the smoothness of blending weights as curvature of blending weights $w_{x,y}$ over a $m \times m$ surface grid $G = \{p = (x, y)|(0 \leq x \leq m, 0 \leq y \leq m)\}$. For

given grid G, we compute the curvature $\kappa_{x,y}$ at each grid vertex $p = (x, y)$ as :

$$\kappa_{x,y} = \frac{1}{8} \| \sum_{a=-1}^{1} \sum_{b=-1}^{1} (w_{x,y} - w_{x+a,y+b}) \|$$

This curvature is computed over several grids to uniformly sample the volume within the 3D parametric space and use the average curvature $\bar{\kappa}$ as the smoothness metric.

We also propose visualizing the smoothness (visual quality) of the final synthesis with **motion vector flows**: each vector denotes the absolute movement of a particular skeleton joint as it traverses the 3D workspace between two consecutive motion frames. Distinguishable colors are assigned to the vectors representing sudden change in vector length compared against local average of the length computed with a sliding window, thus highlighting the abnormal speed-ups (warm color) and slowdowns (cool color) caused by jitters and such. Fig 6 shows the motion vector flow from 150-frame locomotion sequences generated by 4 blending methods, each showing a character transitioning from slow walk to jogging over the same course of variations inside parametrization space. Motion frames are selectively plotted with stick figures on top of the vector flow.

5 Results and Discussions

We uniformly sampled inside the parametric space of each method and measured the obtained errors. Since the parametric space for reach, punch and kick naturally coincides with the 3D workspace, we sample the parameter point $p = (x, y, z)$ over a spherical surface and compute the error as the euclidean distance between the target p and where the wrist/ankle actually reaches, see Fig 2. For jump and locomotion where the parametric space represents abstract values such as turning angle and speed, we sample the parameter point on a rectangular grid.

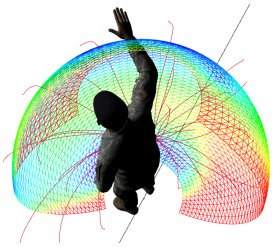

Fig. 2. Parametric space for reaching dataset.

Parametric Error Comparison: The parametrization accuracy visualizations for each method are shown in Fig 3. The first 3 rows showing the result for reach, punch and kick respectively, and the surface we used to sample p is to fix parameter z (distance from the character) in mid-range of the dataset coverage. Similarly for jump and locomotion (row 4 and 5), jump height h and sideways speed v_s (see Section 4) are chosen respectively in mid-range. InvBld by comparison tends to be the most accurate as it relies on numerical optimization to find blend weights that yield minimal errors. KNN also performs relatively well as it populates the gap in parametric space with pseudo examples to effectively reduce the error. Thus for applications that require high parametrization accuracy such as reaching synthesis, it is preferred to apply either InvBld or KNN with dense data. On the other hand Barycentric and RBF numerically tend to generate less accurate results, however this does not necessarily mean the motions generated are of poor quality. In fact, as human eyes are more sensitive to high frequency changes than to low frequency errors, Barycentric and RBF are able to produce reasonable motions for locomotion and jumping, which are parameterized in the abstract space. The table and chart in Fig 5 (left side) lists the average parametric error using results from a more densely sampled parametrization space ($60 \times 60 \times 5$ samples on average).

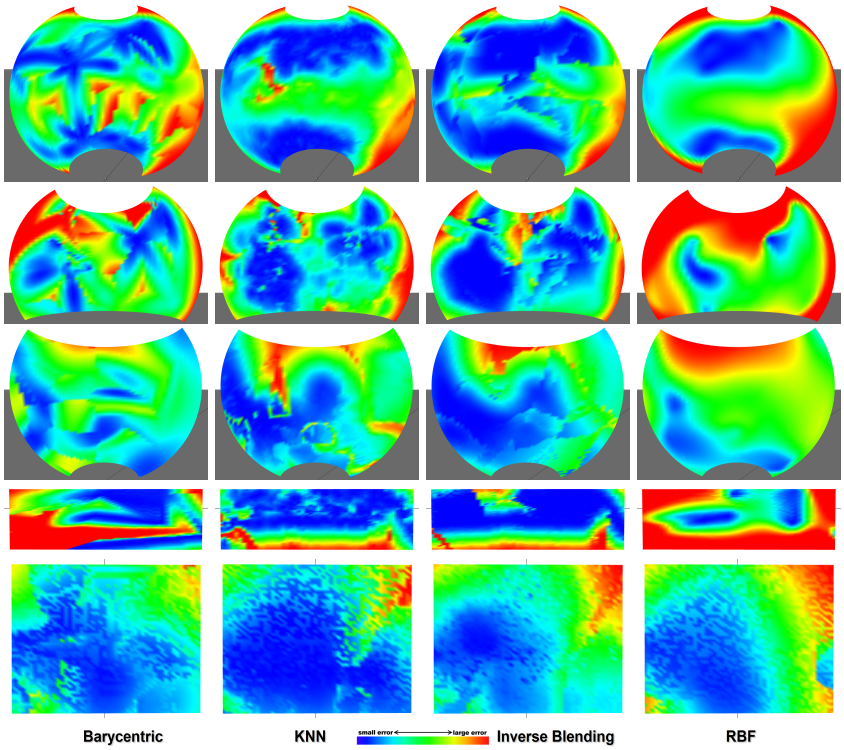

Fig. 3. Parametrization accuracy visualizations for 4 blending methods on different motion dataset. From top row to bottom are reach, punch, kick, jump and locomotion; from left column to right are: Barycentric, KNN, InvBld and RBF.

Smoothness Comparison: Although InvBld outperforms in parametrization accuracy, it falls behind in terms of smoothness, which can be observed both visually (Fig 4) and numerically (Fig 5 right side). By comparing the error and smoothness maps with other methods, we observe that there are several semi-structural regions with both high errors and discontinuity in smoothness. Depending on the initial condition, InvBld optimization procedure may get trapped in local minimal at certain regions in parametric space, which results in high error and discontinuous regions shown in column 3 of Fig 4 and 3. KNN also suffers from similar smoothness problems (Fig 4 column 2), and since KNN requires a dense set of pseudo examples to reduce parametric errors, the resulting parametric space tends to be noisier than others. Moreover, for KNN and InvBld, there can be sudden jumps in blending weights due to changes in the nearest neighbors as the parametrization changes, leading to the irregular patterns in the smoothness visualizations (Fig 4).

Barycentric produces a smoother parameterization as the blending weights only change linearly within one tetrahedron at any given time. However obvious discontinuities occur when moving across the boundaries between adjacent tetrahedra. Note that although both KNN and Barycentric interpolation have similar numerical smoothness in certain cases, the resulting motions from Barycentric usually look more visually

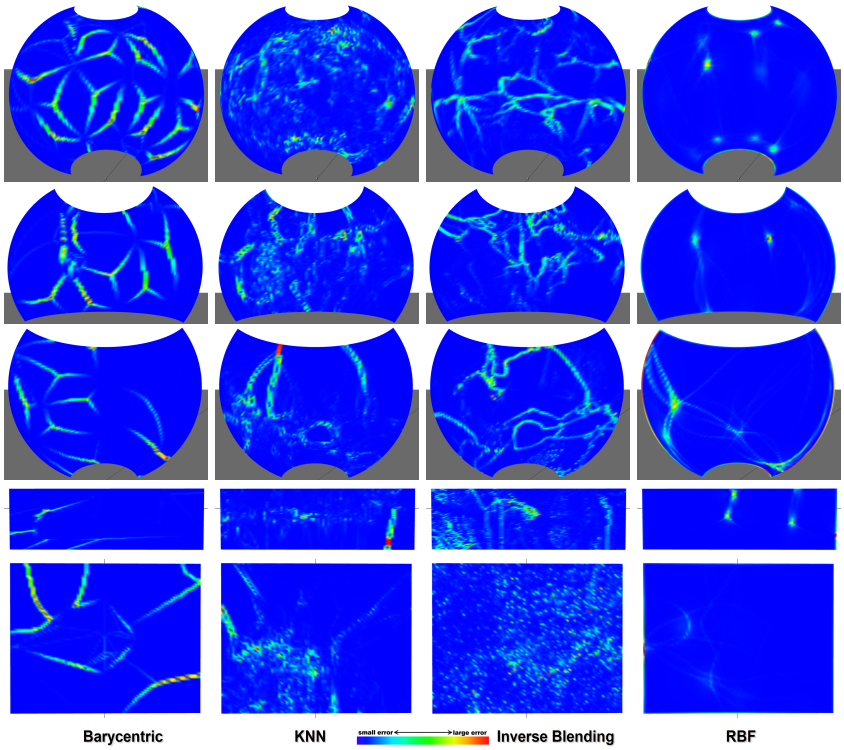

Fig. 4. Smoothness visualizations for 4 blending methods on different motion dataset. From top row to bottom are reach, punch, kick, jump and locomotion; from left column to right are: Barycentric, KNN, InvBld and RBF.

pleasing. This is because the weight discontinuity is only visible when moving between different tetrahedra for Barycentric, while for KNN the irregular blending weights could cause constant jitters in the resulting motions. Finally, RBF tends to generate the smoothest result visually and numerically, which may be a desirable trade-off for its high parametric error in certain applications. Low performance in numerical smoothness corresponds to low visual quality of the final synthesis, as shown in Fig 6 and also the accompanied video where more jitters and discontinuities can be observed.

Computation Time Comparison: KNN requires more pre-computation time than other methods for populating the parametric space with pseudo-examples as well as constructing a k-D tree to accelerate run-time efficiency. Moreover, whenever a new motion example is added, it needs to re-build both pseudo examples and k-D tree since it is difficult to incrementally update the structures. This makes KNN less desirable for applications that require on-line reconstruction of new parametric space when new motion examples are added. RBF on the other hand can usually be efficient in dealing with a small number of examples, however the cost of solving linear equations increases as dataset gets larger. Barycentric requires the tetrahedra to be either manually prespecified or automatically computed, and may become less flexible for high dimension

parametric space. InvBld by comparison is more flexible since it requires very little pre-computation by moving the computational cost to run-time.

For run-time performance, all methods can perform at interactive rate, see last column of the table in Fig 5. However, InvBld is significantly more expensive than other methods as it requires many numerical iterations with kinematic chain updates to obtain optimal results. Also, the computation time greatly depends on the initial estimation of the blending weight and therefore may have large variations across different optimization sessions, posing big challenges on its real-time performance for multi-characters simulations. The other methods require only a fixed number of operations (well under 1 millisecond) and are much more efficient for real-time applications.

evaluation / methods	average parametrization accuracy					average smoothness					average compute time
	Reach	Punch	Kick	Jump	Locomotion	Reach	Punch	Kick	Jump	Locomotion	
Barycentric	0.2060188	0.2351264	0.4723036	0.3499212	1.3287728	0.0377356	0.0248206	0.0224772	0.0157104	0.0212154	0.625 ms
KNN	0.1676776	0.177986	0.3386378	0.1629752	1.0714436	0.0376208	0.0318934	0.0291025	0.0241884	0.0182888	0.221 ms
InvBld	0.1511574	0.17205	0.3217224	0.0756766	1.2572206	0.0698706	0.0785622	0.0801812	0.0609856	0.0602558	4.394 ms*
RBF	0.241626	0.325049	0.5883928	0.3816458	1.7279502	0.0035818	0.0023684	0.0021972	0.001998	0.0012648	0.228 ms

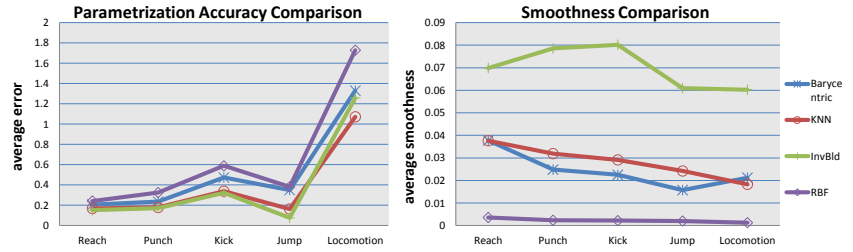

Fig. 5. Parametrization accuracy and smoothness comparison chart across four blending methods on different motion sets. * Computation time measured with Quad Core 3.2GHz running on single core. InvBld can expect $2 \sim 3X$ speed-up with optimized code on kinematic chain updates [7].

The overall performance for each blending method is summarized in Figure 7. In terms of parametric error and smoothness, InvBld has the most precision results but is poor in smoothness. On the opposite end, RBF produces the smoothest parametric space but is the least accurate method as a trade-off. KNN and Barycentric fall in between with KNN slightly more accurate and less smooth than Barycentric. In terms of computation time, KNN requires most pre-computation while InvBld requires none. RBF and Barycentric require some pre-computation and may also require user input to setup tetrahedra connectivity or fine tune the radial basis kernel. Therefore InvBld is most suitable for on-line update of motion examples, with the trade-off being most expensive for run-time computation while the other methods are all very efficient at run-time.

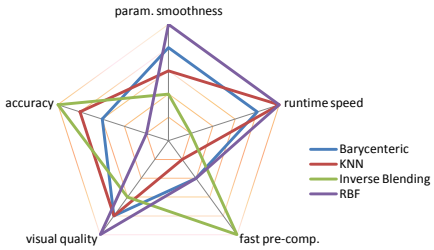

Fig. 7. Performance overview across four blending methods. Measurements are not to scale.

These performance results suggest that there is no method that works well for all metrics. To gain advantages in some metrics, a method would need to compromise in

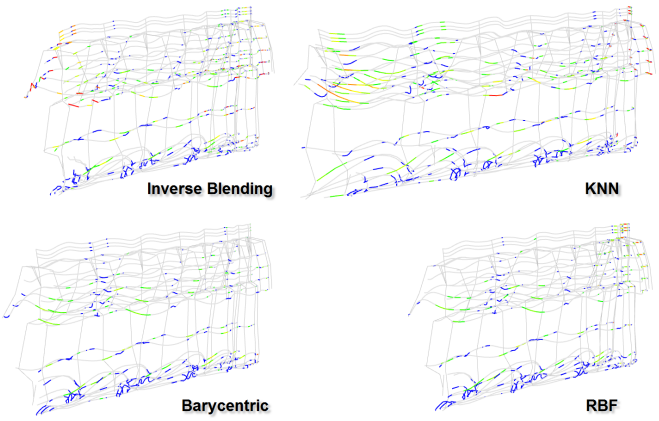

Fig. 6. Motion vector flow visualization of a 150-frame locomotion sequence transitioning from slow walk to jogging. Color segments indicates jitters and unnatural movements during the sequence. By comparison results from InvBld and KNN (top row) contain more jitters than Barycentric and RBF (bottom row).

other metrics. InvBld and RBF show a good example of such compromise that are in the opposite ends of the spectrum. Overall, for applications that do not require high accuracy in parametric errors, RBF is usually a good choice since it is mostly smooth, easy to implement, and relatively efficient both at pre-computation and run-time. On the other hand, if parametric accuracy is very important for the application, InvBld provides the best accuracy at the cost of smoothness in parametric space. KNN and Barycentric fall in-between the two ends, with Barycentric being smoother and KNN being more accurate. Note that KNN may require much more pre-computation time than other methods depending on how dense the pseudo-examples are generated, which may hurt its applicability in certain interactive applications.

6 Conclusion

We present in this paper an in-depth analysis among four different motion blending methods. The results show that there is no one-solves-all method for all the applications and compromises need to be made between accuracy and smoothness. This analysis provides a high level guidance for developers and researchers in choosing suitable methods for character animation applications. The metrics defined in this paper would also be useful for testing and validating new blending methods. As future work we plan to bring in more motion blending schemes for comparison analysis. A new motion blending method that satisfy or make better compromise at both parametric error and smoothness would be desirable for a wide range of applications.

Please see our accompanying video at http://people.ict.usc.edu/∼shapiro/mig12/paper10/. **LINK**

References

1. Abe, Y., Liu, C.K., Popović, Z.: Momentum-based parameterization of dynamic character motion. In: SCA '04: 2004 ACM SIGGRAPH/Eurographics symposium

on Computer animation. pp. 173–182. Eurographics Association, Aire-la-Ville, Switzerland (2004)
2. Bruderlin, A., Williams, L.: Motion signal processing. In: SIGGRAPH '95. pp. 97–104. ACM, New York, NY, USA (1995)
3. Camporesi, C., Huang, Y., Kallmann, M.: Interactive motion modeling and parameterization by direct demonstration. In: Proceedings of the 10th International Conference on Intelligent Virtual Agents (IVA) (2010)
4. Cooper, S., Hertzmann, A., Popović, Z.: Active learning for real-time motion controllers. ACM Transactions on Graphics (SIGGRAPH 2007) (Aug 2007)
5. Grochow, K., Martin, S., Hertzmann, A., Popović, Z.: Style-based inverse kinematics. ACM Transactions on Graphics (Proceedings of SIGGRAPH) 23(3), 522–531 (2004)
6. Heck, R., Gleicher, M.: Parametric motion graphs. In: I3D '07: Proc. of the 2007 symposium on Interactive 3D graphics and games. pp. 129–136. ACM, New York, NY, USA (2007)
7. Huang, Y., Kallmann, M.: Motion parameterization with inverse blending. In: Proceedings of the Third International Conference on Motion In Games. Springer, Berlin (2010)
8. Johansen, R.S.: Automated Semi-Procedural Animation for Character Locomotion. Master's thesis, Aarhus University, the Netherlands (2009)
9. Kim, M., Hyun, K., Kim, J., Lee, J.: Synchronized multi-character motion editing. ACM Trans. Graph. 28(3), 79:1–79:9 (Jul 2009)
10. Kovar, L., Gleicher, M.: Automated extraction and parameterization of motions in large data sets. ACM Transaction on Graphics (Proceedings of SIGGRAPH) 23(3), 559–568 (2004)
11. Kovar, L., Gleicher, M., Pighin, F.H.: Motion graphs. Proceedings of SIGGRAPH 21(3), 473–482 (2002)
12. Kwon, T., Shin, S.Y.: Motion modeling for on-line locomotion synthesis. In: SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation. pp. 29–38. ACM, New York, NY, USA (2005)
13. Levine, S., Lee, Y., Koltun, V., Popović, Z.: Space-time planning with parameterized locomotion controllers. ACM Trans. Graph. 30(3), 23:1–23:11 (May 2011)
14. Mukai, T., Kuriyama, S.: Geostatistical motion interpolation. In: ACM SIGGRAPH. pp. 1062–1070. ACM, New York, NY, USA (2005)
15. Park, S.I., Shin, H.J., Shin, S.Y.: On-line locomotion generation based on motion blending. In: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation. pp. 105–111. SCA '02, ACM, New York, NY, USA (2002)
16. Rose, C., Bodenheimer, B., Cohen, M.F.: Verbs and adverbs: Multidimensional motion interpolation. IEEE Computer Graphics and Applications 18, 32–40 (1998)
17. RoseIII, C.F., Sloan, P.P.J., Cohen, M.F.: Artist-directed inverse-kinematics using radial basis function interpolation. Computer Graphics Forum (Proceedings of Eurographics) 20(3), 239–250 (September 2001)
18. Safonova, A., Hodgins, J.K.: Construction and optimal search of interpolated motion graphs. In: ACM SIGGRAPH '07. p. 106. ACM, New York, NY, USA (2007)
19. Shapiro, A., Kallmann, M., Faloutsos, P.: Interactive motion correction and object manipulation. In: ACM SIGGRAPH Symposium on Interactive 3D graphics and Games (I3D'07). Seattle (April 30 - May 2 2007)
20. Unuma, M., Anjyo, K., Takeuchi, R.: Fourier principles for emotion-based human figure animation. In: SIGGRAPH '95. pp. 91–96. ACM, New York, NY, USA (1995)
21. Wiley, D.J., Hahn, J.K.: Interpolation synthesis of articulated figure motion. IEEE Computer Graphics and Applications 17(6), 39–45 (1997)