The Visual Computer (2019) 35:1131–1142
https://doi.org/10.1007/s00371-019-01693-8

ORIGINAL ARTICLE

Precomputed optimal one-hop motion transition for responsive character animation

Yuki Koyama[1] · **Masataka Goto**[1]

Published online: 9 May 2019

Abstract
Characters in interactive 3D applications are often animated by creating transitions from one motion clip to another in response to user input. It is not trivial, however, to achieve quick, natural-looking transitions between two arbitrary motion clips, especially when the two motions are dissimilar. To tackle this problem, we present a simple framework called *optimal one-hop motion transition*, which creates quick, natural-looking transitions on the fly without requiring careful manual specifications. The key ideas are (1) to insert a short intermediate motion clip, called a *hop*, between the source and destination motion clips, and (2) to select such a hop motion clip and its temporal alignment in an optimal way by solving a search problem. In the search problem, our framework tries to balance the naturalness of the resulting transitions and the responsiveness to user input. This search can be precomputed and the results can be stored in a lookup table, making the runtime cost to play an optimal transition negligible. We demonstrate that our framework is easily integrated into a widely used game engine, and that it greatly improves the quality of transitions in practical scenarios.

Keywords Motion transition · Character animation · Motion graphs

1 Introduction

Interactive applications that control 3D characters in real time have become increasingly popular in the contexts of computer games, virtual environments, and even stage performances by virtual singers. For such applications, a popular current practice is to organize a character animation by using a *state machine* [7,36], in which each state is typically associated with a certain motion clip that should be played when the character is in that state. When a state change occurs at runtime (usually triggered by user input), a *motion transition* (i.e., crossfading from one motion clip to another, often in two seconds or less) is applied to the character. For this, a simple pose-space linear blending technique is typically used because of its runtime efficiency. This approach only works well, however, when the source and destination motion clips are sufficiently similar; otherwise, it produces severe visual artifacts, such as footskate from "standing with legs open" to "standing with legs closed" and sudden, unexpected change of momentum.

To avoid such unnatural transitions, current game engines allow game designers to manually specify several parameters, such as the *blending duration* (i.e., how many frames to use for a transition) and the *transition point* (i.e., which frame to use for the beginning of a transition), for every possible transition in the state machine [35]. There are several problems, however, in this approach:

- When two motion clips are dissimilar, adjusting these parameters is not very helpful; it is necessary to manually insert a state between the two states to play a short motion clip (e.g., a leg-closing motion). This greatly complicates management of the state machine.
- As the frame at which the transition is triggered is not predictable, it is not sufficient to define a single fixed transition point for each motion clip. Ideally, the tran-

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s00371-019-01693-8) contains supplementary material, which is available to authorized users.

✉ Yuki Koyama
koyama.y@aist.go.jp

Masataka Goto
m.goto@aist.go.jp

[1] National Institute of Advanced Industrial Science and Technology (AIST), Central 2, 1-1-1 Umezono, Tsukuba, Ibaraki 305-8568, Japan

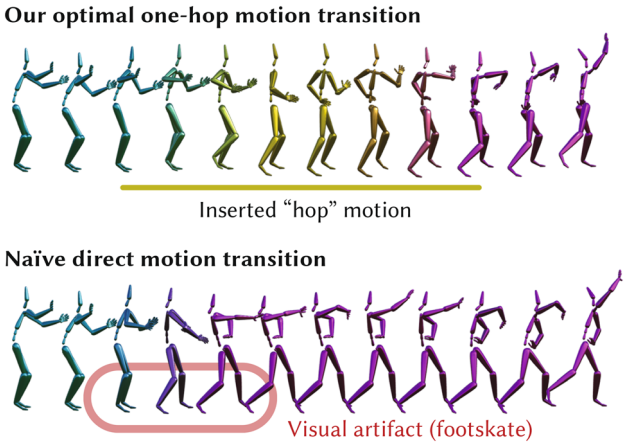

Fig. 1 Example of our *optimal one-hop motion transition* (Top). Our framework creates natural-looking, responsive motion transitions by inserting an intermediate motion called a *hop* (yellow) between the *source* (cyan) and *destination* (magenta) motions. It can produce a natural-looking motion transition even for a case in which a naïve direct transition exhibits severe visual artifacts (Bottom)

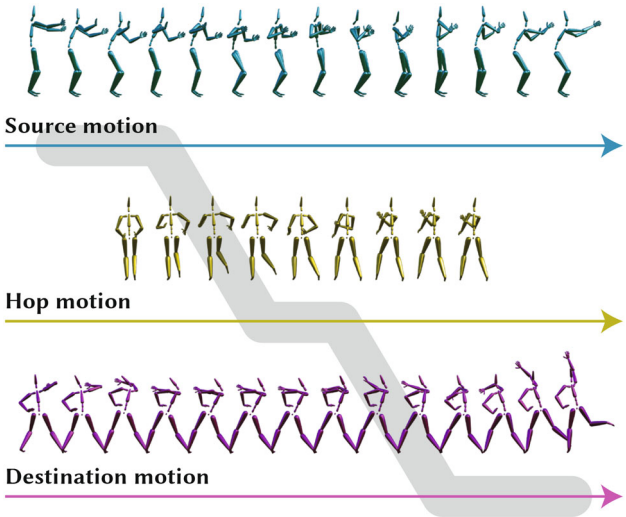

Fig. 2 Our *one-hop* motion transition design. Rather than directly visiting the destination, our framework first visits a *hop* and then visits the destination. Between the source and destination motion clips, it automatically inserts an optimal hop selected by solving a search problem in precomputation

sition point should be adaptively set for every possible triggered frame.

To tackle these problems, we present a practical framework called *optimal one-hop motion transition*. This framework greatly improves the quality of transitions even when the source and destination motions are dissimilar, as seen in Fig. 1. It also optimizes the responsiveness for every possible triggered frame and still maintains the simplicity of the state machine management. Figure 2 illustrates our framework. The key idea is twofold:

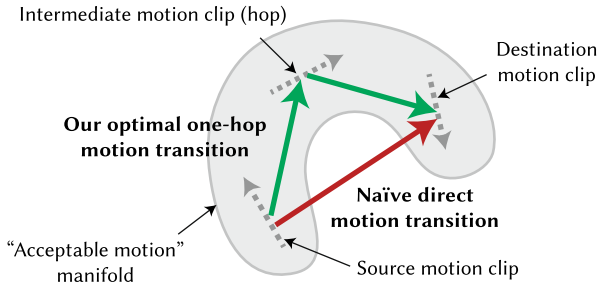

Fig. 3 Conceptual explanation of our *one-hop* approach. A naïve direct motion transition often produces unacceptable motions during the transition. In contrast, our approach visits a hop, which is chosen so that both the source-to-hop and hop-to-destination transitions are as acceptable as possible

1. First, our framework automatically inserts a short, intermediate motion clip (called a *hop*) between the source and destination motion clips. That is, it converts a transition into two individual transitions (i.e., source to hop and hop to destination). This *one-hop* approach has great potential to improve the acceptability of the resulting motion transition as compared to the *direct* approach. Figure 3 provides an intuition of the reason for this: if we can arrange a hop such that the two individual transitions are both sufficiently natural, then the "path" of the overall transition is likely to be within a manifold of "acceptable motion".[1]
2. Second, our framework selects the hop motion clip and its *temporal alignment offset* in an optimal way by solving a search problem. In the search, the framework considers both the *naturalness* (i.e., how natural the individual transitions look) and the *responsiveness* (i.e., how quickly the overall transition finishes) of the resulting transitions and balances the trade-off between these two objectives.

Our framework has several desirable features. First, the search is precomputable; in precomputation, the system searches for optimal transitions for every possible combination of a pair of motion clips and a triggered frame. At runtime, the system dynamically retrieves an optimal transition for a triggered frame. As this requires only negligible computation cost, it is feasible to use the framework in computational resource-restricted applications. The framework is also easily integrated into widely used game engines; we demonstrate this by using Unity. In addition, the framework does not assume either any special annotations to motion clips or any special skeletal structures (e.g., a humanoid).

Our contributions are summarized as follows.

– Design of a framework for optimal one-hop motion transition, which improves the naturalness of transitions

[1] Note that we use these quoted terms in this sentence informally.

while maintaining the ease of control and necessary responsiveness.
– Demonstration of the framework's practicality: the search is precomputable, efficient at runtime, and easy to integrate into a popular game engine.

2 Related work

Motion transition is a primitive problem in interactive character animation, and researchers have worked on this and related topics for decades (e.g., [1,6,9,13,16,23,29,33,37]). Our proposed framework can be seen as a variant of the *motion-graph-based* approach [1,16,23], in that we consider graphs with a specific structure (i.e., one-hop graphs) and use techniques developed in that context. This section reviews previous approaches from various points of view and clarifies the position of this work.

Motion graphs Motion-graph-based methods have been intensively studied for years [1,9,16,23,34]. One common problem of this approach is that it does not ensure sufficient responsiveness: a sufficiently short path from the source state to the destination state might not exist, and thus, the duration for a transition might be unacceptably long (this problem was clearly described in [13]). In addition, it is difficult to explicitly specify the temporal alignment between the source and destination motion clips. These problems make it difficult to use motion graphs for interactive, precise character control. Our framework, on the other hand, uses a designer-defined state machine for intuitive control logic, and it incorporates responsiveness into the search objective.

High-level motion synthesis Many offline motion synthesizers or online motion controllers have been proposed to animate characters by using high-level constraint specifications (such as semantic tasks to engage, directions or paths to follow, and goal positions to reach) [10,16,20,21,24,27,34]. Even though many such methods have been proposed so far, the state machine approach is still highly popular in interactive application development [7,36], probably because of its simplicity and ease of control. We focus here on a primitive problem in state-machine-based control: the transition between two motion clips.

Motion matching In the game industry, a method called *motion matching* has recently been developed [5,38]. That method is similar to ours in that it searches for an optimal motion clip and its time alignment in every necessary frame to achieve high responsiveness. It is not designed, however, for creating transitions between two motion clips.

Statistical models Learning a statistical motion model from a captured dataset has been a recent trend [3,10,11,25,28]. While this approach is quite promising, it is not feasible for individual game designers to construct a good statistical model, understand its capability correctly, design game logic based on it, write a script to control it at runtime, and fine tune it if necessary. Thus, the state-machine-based approach still seems more popular in practice than the statistical approach, indicating a large demand for our framework.

Motion blending Motion blending is a technique to create an interpolated motion from multiple motion clips and associated blending weights. In the case of motion transition, motion blending involves two motion clips (i.e., source and destination) and a time-varying weight. Pose-space linear blending has been widely used because of its simplicity and runtime efficiency. To achieve higher quality, researchers have investigated more sophisticated methods [15,30]. Our focus is on how to choose motion clips to blend and align them, and thus, arbitrary motion blending methods can be used.

Motion naturalness Physical criteria are often applied to enforce naturalness on motions, as in space-time optimization [33] and in keyframe motion editing [18]. Another popular approach is to use statistical models (e.g., [3]). In our framework, we need to evaluate the naturalness of a blended motion during a transition. As with previous motion-graph-based methods (e.g., [9,16,37]), we assume that a transition becomes smoother and thus more natural if its source and destination motions are more similar.

Interactive motion transitions Several previous methods specifically focused on interactive motion transitions (e.g., [2,6,13,29]). Among them, the method by Ikemoto et al. [13] is probably the most similar to ours. Their method searches for one- or multi-hop transitions to ensure the naturalness of the resulting motions. One of the most notable differences is in the handling of the duration of the overall transition; while the method of Ikemoto et al. uses a fixed duration for the overall transition (i.e., it always takes the same duration to finish a transition, regardless of whether it is an "easy" transition or not), ours automatically adapts the duration of the overall transition according to its "difficulty". That is, our framework considers not only naturalness but also responsiveness, unlike the previous one. In addition, the framework incorporates the *stay duration* (i.e., the duration for which a motion clip is played alone before beginning an individual transition) into search variables, which enables it to find even better (i.e., lower-cost) transitions. The way of storing precomputed results also differs between our method and that of Ikemoto et al.; we store *per-frame* optimal solutions, which is necessary for playing transitions with optimal responsiveness for every possible triggered frame. Note that their work also presented some additional techniques such as clustering of motion segments for performance, time warping of motions, and learning-based evaluation of motion natural-

ness. We emphasize here that, although our framework is extensible and able to incorporate most of their additional techniques, for now we have not adopted those techniques to keep our framework lightweight, easy to implement, and intuitive for game developers and designers.

3 Optimal one-hop motion transition

3.1 Problem setting and search variables

Our goal is to create a natural-looking transition from a source motion $\mathbf{M}_{\text{src}}$ to a destination motion $\mathbf{M}_{\text{dst}}$. This transition should be responsive: the user triggers it at an arbitrary frame t_{trigger} (e.g., via a gamepad), and the destination motion should then be played as quickly as possible while maintaining naturalness. We assume that a set of n motion clips $\mathcal{M} = \{\mathbf{M}_1, \ldots, \mathbf{M}_n\}$ (including $\mathbf{M}_{\text{src}}$ and $\mathbf{M}_{\text{dst}}$) is in memory at runtime, so that the clips are available as candidates for hop motion clips.

We consider the case in which the temporal alignment between the source and destination motion clips is hard-constrained and cannot be altered by the transition calculation process. Note that satisfying this constraint is necessary for many practical scenarios. For example, in a fighting action game, a character might need to start playing a jump motion clip in response to player input to avoid an opponent's kick; it is not acceptable in a typical game design if the temporal alignment offset of the jump motion clip relative to the triggered frame is altered by the transition algorithm every time. Another example is the case in which a character's movement has to be synchronized with a certain factor (e.g., background music for a dance character) in both the source and destination states.

In precomputation, we search for optimal one-hop transitions for all possible triggered frames. Figure 4 illustrates the problem setting of the search for a certain triggered frame t_{trigger}, which involves the following unknowns:

- The *index* of the hop motion clip, $i \in \{1, \ldots, n\}$.
- The *temporal alignment offset* of the hop motion clip relative to the source motion clip, $o_{\text{s}\to\text{h}} \in \mathbb{Z}$.
- The first *stay duration* (i.e., the number of frames to wait for before starting the transition from the source motion clip to the hop motion clip), $s_{\text{s}} \in \mathbb{Z}_{\geq 0}$.
- The second stay duration (i.e., the number of frames to wait for before starting the transition from the hop motion clip to the destination motion clip), $s_{\text{h}} \in \mathbb{Z}_{\geq 0}$.
- The *blend duration* for the first transition from the source motion clip to the hop motion clip, $b_{\text{s}\to\text{h}} \in \mathbb{Z}_{>0}$.
- The blend duration for the second transition from the hop motion clip to the destination motion clip, $b_{\text{h}\to\text{d}} \in \mathbb{Z}_{>0}$.

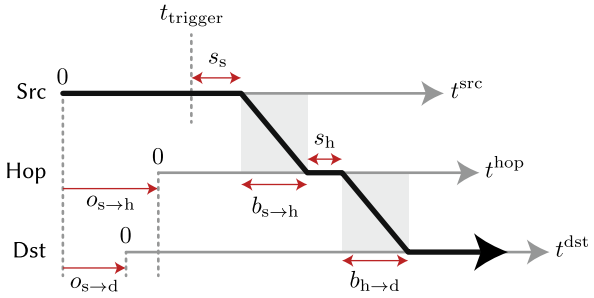

Fig. 4 Illustration of our problem setting for each search task. The frame of receiving the user input, t_{trigger}, and the *temporal alignment offset* between the source and destination motion clips, $o_{\text{s}\to\text{d}}$, are given. In addition to the hop motion clip chosen from a candidate set, the *stay durations*, s_{s} and s_{h}, the *blend durations*, $b_{\text{s}\to\text{h}}$ and $b_{\text{h}\to\text{d}}$, and the temporal alignment offset between the source and hop motion clips, $o_{\text{s}\to\text{h}}$, are nontrivial variables

Among these variables, for simplicity we assume that the blend durations $b_{\text{s}\to\text{h}}$ and $b_{\text{h}\to\text{d}}$ are given; we thus omit them from the search variables. This is because optimizing them is a nontrivial problem. Some advanced techniques can find optimal blend durations (e.g., [37]), but for now we assign them a fixed value of 30 frames (0.5 sec in our implementation, which was selected following [31,33]), for simplicity.

As a result, the search variables in our framework are expressed as follows. Let $\mathcal{T}_t$ be a set of all the possible one-hop transitions for a triggered frame t. We represent a one-hop transition, $T_t \in \mathcal{T}_t$, by a four-element tuple:

$$T_t = (i, o_{\text{s}\to\text{h}}, s_{\text{s}}, s_{\text{h}})\,. \tag{1}$$

Given a target triggered frame t, our framework seeks to find an optimal transition $T_t^* \in \mathcal{T}_t$ (i.e., the best possible combination of i, $o_{\text{s}\to\text{h}}$, s_{s}, and s_{h}) that minimizes the duration necessary for the overall transition while maximizing the overall naturalness; this constitutes a trade-off.

3.2 Search for optimal one-hop motion transition

Search objective We solve the following discrete search problem for each possible triggered frame t:

$$T_t^* = \underset{T_t \in \mathcal{T}_t}{\arg\min}\{S(T_t) + wR(T_t)\}, \tag{2}$$

where $S : \mathcal{T} \to \mathbb{R}_{\geq 0}$ is a *smoothness* cost function, $R : \mathcal{T} \to \mathbb{R}_{\geq 0}$ is a *responsiveness* cost function, and $w \in \mathbb{R}_{>0}$ is a weight parameter for the user to control the trade-off between the transition smoothness and responsiveness, depending on the usage scenario. The smoothness cost function is defined as

$$S(T_t) = D_{\text{s}\to\text{h}}(T_t) + D_{\text{h}\to\text{d}}(T_t), \tag{3}$$

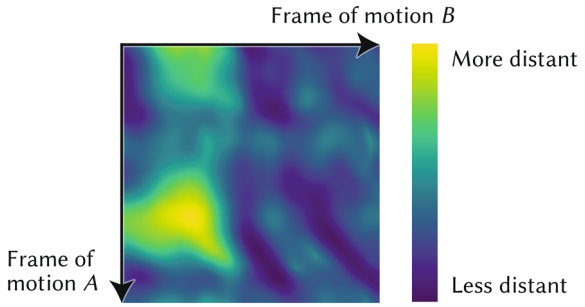

Fig. 5 Colormap visualization of distance function values between a pair of motion clips (say, A and B). A lower value means that the two motions are less distant (i.e., more similar) at the indicated frames, and thus, a transition created under that condition is expected to become smoother. The details of the distance function are described in [16]. To evaluate the smoothness of a one-hop transition, our framework calculates distance values for both (1) the pair of source and hop motion clips and (2) the pair of hop and destination motion clips, and then it evaluates the overall smoothness of the one-hop transition (Eq. 3)

where $D_{\mathrm{s}\to\mathrm{h}}$ and $D_{\mathrm{h}\to\mathrm{d}}$ are distance (or dissimilarity) functions that measure the distances between the source and hop motion clips and the hop and destination motion clips, respectively, during individual transitions. The responsiveness cost function should be a monotonically increasing function with respect to the duration of the overall transition. Our current implementation simply defines it as the sum of the stay durations:

$$R(T_t) = s_{\mathrm{s}} + s_{\mathrm{h}}. \tag{4}$$

Choice of distance function Our current implementation uses a distance function proposed by Kovar et al. [16] which has been used in many follow-up works [6,15,17]. It spatially aligns the two motion fragments to be blended by finding an optimal 2D rigid transformation (i.e., horizontal translation and the yaw rotation) in the least-squares sense, and then it calculates the sum of the distances between corresponding points on the character's body for every pair of frames. This metric is coordinate invariant and implicitly incorporates derivative information by considering multiple frames rather than a single frame. We do not repeat the specific equations here; refer to the original paper for the details. Figure 5 shows an example of computed distance function values for a pair of motion clips. We emphasize that the choice of the distance function is not our focus, and that our framework is orthogonal to that choice. It can use any other distance function, such as those considering joint orientations and velocities [23,37], joint accelerations [1], and so on.

Search strategy We solve the search problem by using a simple exhaustive strategy with pruning. If at least one of three values, $D_{\mathrm{s}\to\mathrm{h}}(T_t)$, $D_{\mathrm{h}\to\mathrm{d}}(T_t)$, and $wR(T_t)$, is larger than the objective value of the "best current" solution, then that T_t cannot be the optimal solution (because these three values are all nonnegative), and thus, we can prune that condition without calculating the other values. We calculate $wR(T_t)$ first and then the other two, as $wR(T_t)$ can be more efficiently calculated than the others can. Also, we visit smaller s_{s} and s_{h} first and then increment these values; because R is a monotonically increasing function with respect to both s_{s} and s_{h}, doing so is likely to find lower-cost solutions at earlier stages of the search, which is helpful for effective pruning.

Storage and use of precomputed results Precomputed optimal transitions can be stored in a lookup table in which triggered frames are the keys for retrieval. By using a random-access array to implement the lookup table, the runtime cost of retrieval has only constant time complexity $O(1)$ and is sufficiently fast for computational-resource-restricted scenarios such as video games. Furthermore, as a one-hop transition can be represented by using four integers (Eq. 1), the runtime module is not memory intensive. For example, when the state machine has 50 possible state transitions and every motion clip has 1k frames, the runtime module uses 800 kB (= 50 [state transitions]×1k [frames]×4 [integers]×4 [bytes]), which is sufficiently small compared to the memory used by typical assets such as textures.

Discussion By selecting the destination motion clip as the hop motion clip and setting $o_{\mathrm{s}\to\mathrm{h}} = o_{\mathrm{s}\to\mathrm{d}}$ and $s_{\mathrm{s}} = s_{\mathrm{h}} = 0$, which is included in the search space, our one-hop transition can express the same transition as by the direct approach. Therefore, our framework is always guaranteed to find a transition that is equal to or better than that by the direct approach in terms of the search objective.

3.3 Default weight

Though it is important to let users adjust the weight parameter w in Eq. 2, it is helpful to provide a "default" value. We present a simple algorithm to find such a value as follows. First, from the set of target motion clips, we randomly sample N transition conditions, each consisting of source and destination motion clips, their temporal alignment, and a trigger frame. Then, we search for optimal transitions for these conditions without using stay durations (i.e., we set $s_{\mathrm{s}} = s_{\mathrm{h}} = 0$) and record the average smoothness cost value $\bar{S}_0$. Next, we perform the search again but incorporate the stay durations as search variables, with an upper bound $s_{\max} \in \mathbb{Z}_{>0}$ and with $w = 0$, and we record the average smoothness cost value $\bar{S}_{s_{\max}}$. Finally, we obtain a default weight as

$$w_{\mathrm{default}} = \frac{\bar{S}_0 - \bar{S}_{s_{\max}}}{2s_{\max}}. \tag{5}$$

Intuitively, this algorithm measures how much the smoothness cost decreases when using stay durations as compared

to when not using them, and then it normalizes the value to find a reasonable relative scale between the smoothness and responsiveness costs. We use $2s_{\max}$ as the denominator, because it is the upper bound of the responsiveness cost when not using stay durations. Another option is to use the average of the responsiveness costs. Either way is valid, because the smoothness and responsiveness costs become comparable through multiplication by the weight. Unless otherwise stated, the results presented in this paper were generated with this default weight automatically set ($N = 100$, $s_{\max} = 60$).

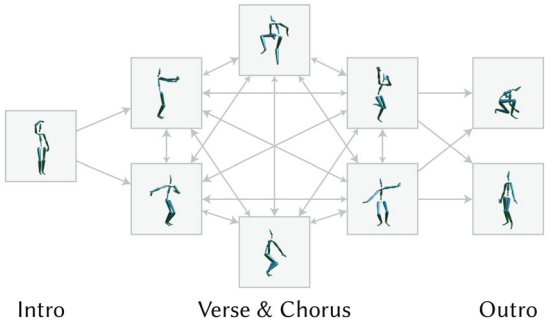

Fig. 6 Conceptual example of a state machine configuration for an interactive dance application. The choreography is assumed to be created in an improvised manner

4 Results

4.1 Implementation details

To demonstrate that our framework is compatible with a popular game engine, we implemented the runtime component in Unity. In particular, we used the *Playables API* in Unity to blend motion clips. The precomputation module was implemented as an independent executable, which can be run via either a command line or the Unity Editor.

As a baseline for comparison, we also implemented the direct approach, which begins a transition immediately from the triggered frame and uses 30 frames to make the transition directly to the destination motion clip. This mimics the common practice.

We tested our framework by using a dance motion dataset that we created using a motion capture system. The dataset consists of various motions from three dance genres: hip-hop, jazz, and pop. Each genre has around 20 motion clips, and each clip lasts around 20 seconds. The frame rate was reduced to 60 frames per second in postprocessing. Unless otherwise stated, each result was generated using around 20 motion clips from this dataset.

Note that we did not apply any postprocessing techniques to clean up footskate artifacts (e.g., [12]) for demonstration purposes. Thus, small footskate artifacts that appear in our results can be fixed to improve the naturalness.

4.2 Use case: interactive choreography

Assumed scenario As a use case for our framework, we considered a scenario of "interactive character dancing," in which the choreography of a virtual dancer is assembled in real time by an operator who improvises the selection of a prerecorded motion clip to play in the next moment. Figure 6 shows a conceptual example state machine for this application. It requires making the played dance sequence as natural-looking as possible, and at the same time, as responsive to the requests to change the choreography as possible.

Visual comparison Refer to the accompanying video, in which we show many transition results produced by our framework and also the baseline approach. Here, we include some of the representative results. Figure 7 shows two cases in which the framework greatly improved the visual quality by choosing appropriate hop motions. Figures 1 and 8 show cases in which the framework successfully cleaned up the footskate artifacts that appeared in the direct transitions.

Robustness test To check whether our framework can robustly generate more plausible transitions than the baseline approach can for various inputs, we procedurally generated an extra set of 15 transition results by an objective rule (i.e., sequentially selecting triggered frames with a fixed interval, rather than applying subjective selection). The accompanying video includes these 15 transitions. According to our observation, the transitions produced by the framework are better than or at least equivalent to the baseline transitions in most cases.

Mostly unchanged cases When the direct transition is already very smooth (i.e., the smoothness cost of the original direct transition is sufficiently small that there is only a little room for improvement), the search module often selects the destination motion clip as the hop and sets a similar temporal alignment. Figure 9 shows such an example, in which only a slight difference in $o_{\mathrm{s}\to\mathrm{h}}$ and $o_{\mathrm{s}\to\mathrm{d}}$ is introduced, thus giving mostly the same results. For another example, see Result 07 of the robustness test in the accompanying video. Note that, in this situation, the resulting transition looks extremely responsive as if it were a direct (zero-hop) transition, but it can appear more natural than a transition by the naïve direct approach because our framework selects s_s and $o_{\mathrm{s}\to\mathrm{h}}$ in an optimal way.

Cost values To better understand the behavior of our framework from a quantitative perspective, we calculated the cost values of both our optimal one-hop transitions (i.e., the summation of $D_{\mathrm{s}\to\mathrm{h}}$, $D_{\mathrm{h}\to\mathrm{d}}$, and wR) and the corresponding

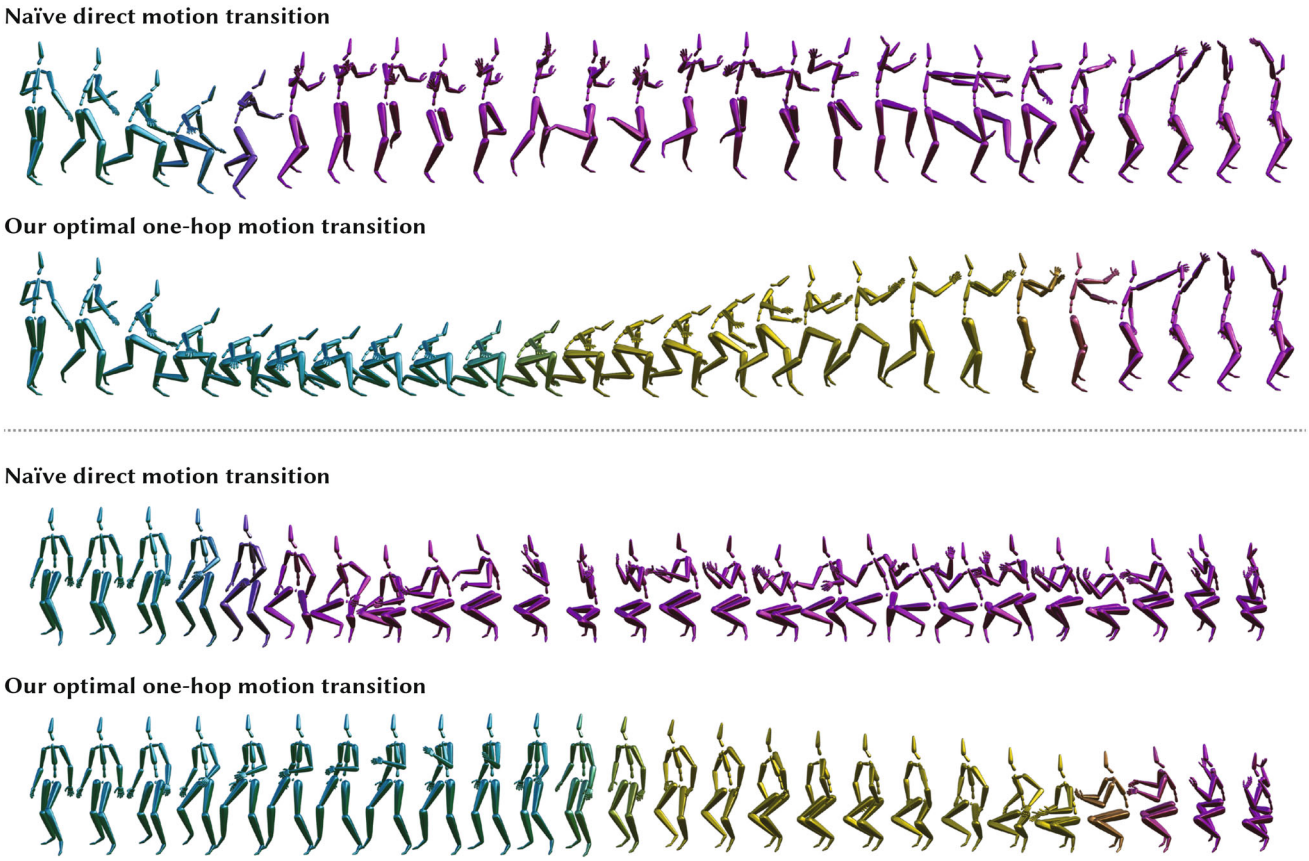

Fig. 7 Results of direct motion transitions and our optimal one-hop motion transitions for two challenging scenarios. (Top) The transition is triggered when the source motion involves crouching. The direct transition exhibits a severe visual artifact of sudden momentum change. In contrast, our transition inserts a standing-up motion after some stay duration, which makes the overall transition look much more natural. (Bottom) The source and destination motions are dances that involve standing and sitting, respectively. The direct transition creates a sudden sitting-down motion. In contrast, our transition lets the character sit down while dancing by inserting a dance segment with a sitting-down motion, which is much more natural in terms of choreography

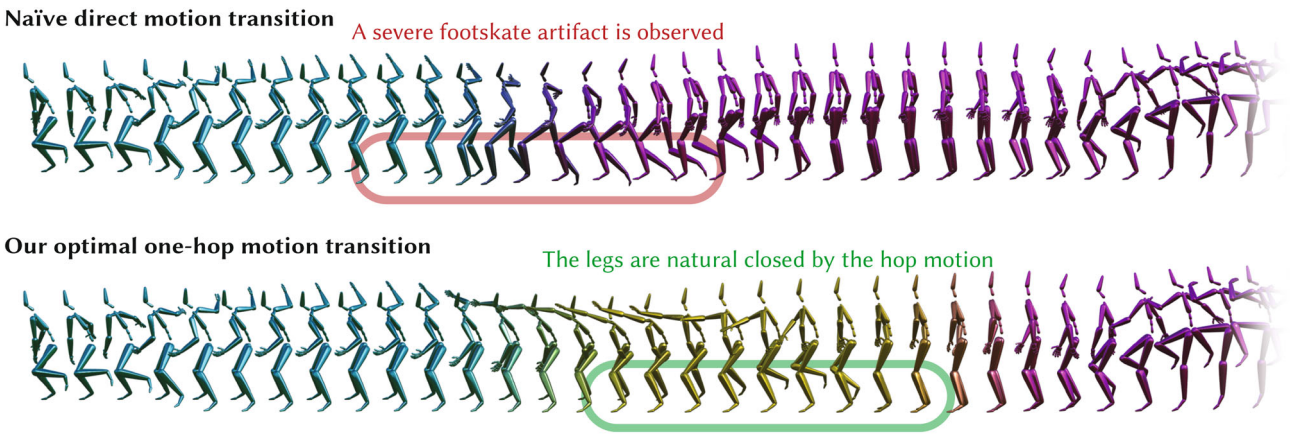

Fig. 8 Example of resolving a footskate problem. (Top) A direct motion transition causes a severe footskate artifact that is so large that it is difficult for inverse-kinematics-based postprocessing techniques to fix it. (Bottom) Our optimal one-hop motion transition inserts an appropriate hop motion of closing the legs, and the resulting footskate artifact is almost negligible

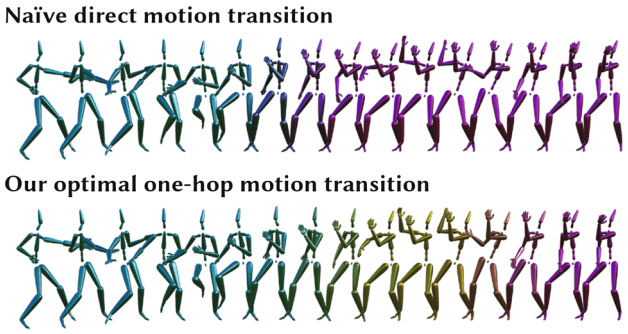

Fig. 9 Example of a case in which the direct (Top) and optimal one-hop (Bottom) approaches produce similar results. In this case, the destination motion clip was selected as the optimal hop motion. This is likely to happen when the direct motion transition is already very smooth

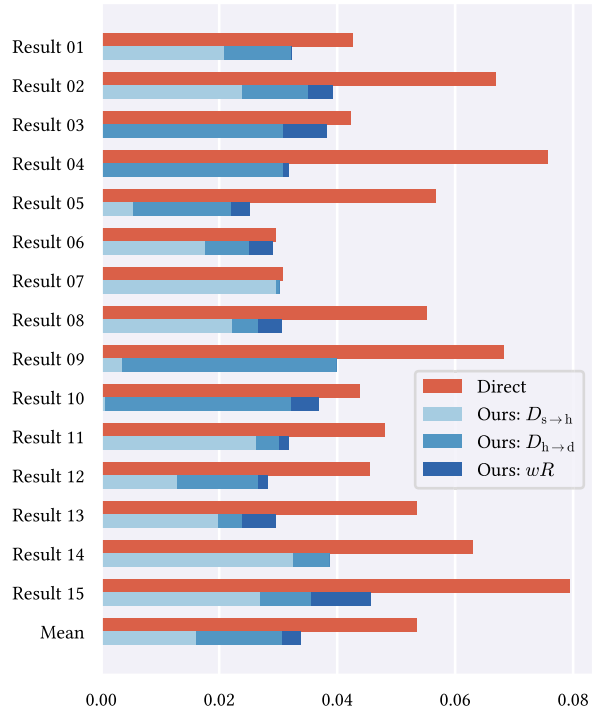

Fig. 10 Cost value comparison of our optimal one-hop motion transitions and the corresponding naïve direct transitions

direct transitions (i.e., the distance between the source and destination motion clips during a direct transition, in other words, $D_{s\to d}$). Figure 10 displays the results in a side-by-side manner. The transitions generated for the robustness test were used here; see the accompanying video for the corresponding rendered motions. The figure shows that our framework could always find lower-cost transitions than those produced by the direct approach, and that our results have various proportions of individual cost values to total cost values.

4.3 Effect of changing weight

To demonstrate the effect of the weight parameter w, we generated transition results by using different weights: $0.1 \cdot w_{\text{default}}$, w_{default}, and $10 \cdot w_{\text{default}}$. Figure 11 shows time-lapse visualizations of these results. Although the source and destination motion clips, their temporal alignment, and the triggered frame were identical, different optimal one-hop transitions were obtained depending on the weights. This demonstrates that the weight controls a trade-off between smoothness and responsiveness: our transition becomes more responsive when using a larger weight, while it becomes smoother when using a smaller weight. Note that even when the largest weight (i.e., $10 \cdot w_{\text{default}}$) was used, it provided a much smaller smoothness cost (0.0467) than the direct transition did (0.0984), owing to the insertion of a hop motion.

4.4 Extensibility

One of the advantages of our framework is its simplicity, which makes it easy to investigate various extensions. To demonstrate its extensibility, we created a *beat-aware* version of the one-hop transition search module for interactive dancing applications. In this extension, hop dance motions are selected and aligned by taking their beats into consideration. This is easily achieved by restricting the search space of the temporal alignment offset $o_{s\to h}$ to only *beat* frames. Figure 12 shows a result generated by this extension. As the source, hop, and destination motions share the same beats in this case, the resulting motions appear well synchronized to the underlying music.

5 Discussion

Multiple hops The number of inserted hops is always one in our framework. We adopted this design because it is a minimal insertion but greatly improves the quality. Being minimal is important to provide better transparency for designers as well as better responsiveness of transitions. Nevertheless, it will be an important future work to investigate the approach of inserting multiple motion clips (e.g., [13]) to obtain even smoother transitions. Note that it is possible to search multiple hops in our framework with a straightforward extension, but this prohibitively increases the computational cost of the search; therefore, we did not include that approach as a baseline in the evaluation, and some techniques need to be developed to make it tractable.

Higher responsiveness As shown in Fig. 9, when the destination motion clip is selected as the hop motion clip and the temporal alignments $o_{s\to h}$ and $o_{s\to d}$ are similar, the resulting transition can look very responsive. For users who desire

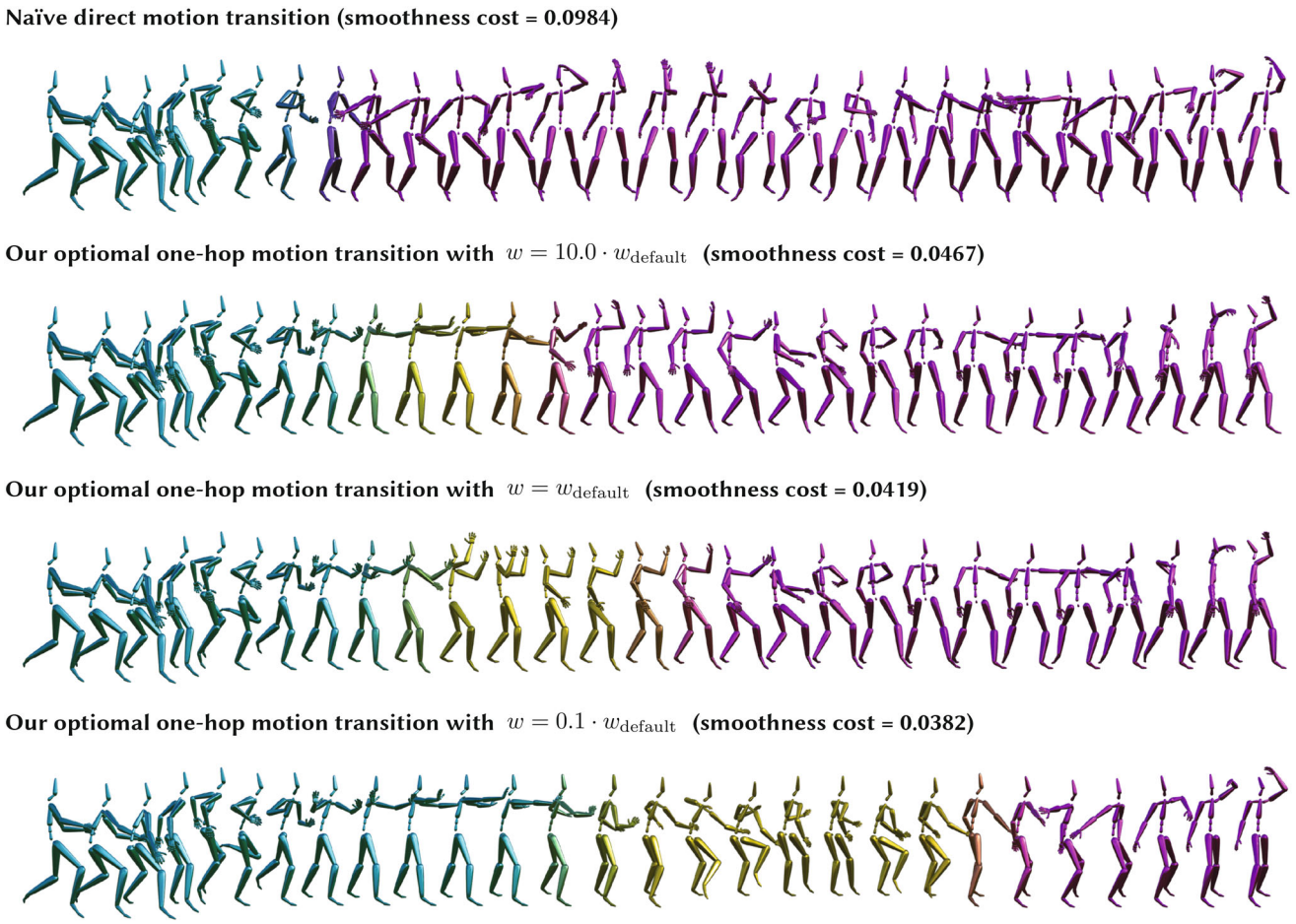

Fig. 11 Visualizations of a direct motion transition (first row) and our optimal one-hop motion transitions with various weights (the other rows) under the same conditions. The weight controls a trade-off between smoothness and responsiveness: our transition becomes more responsive when using a larger weight, while it becomes smoother when using a smaller weight

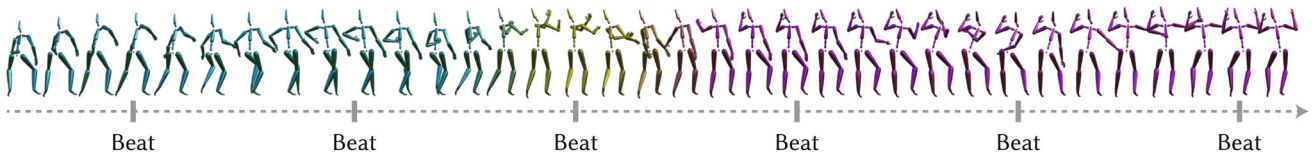

Fig. 12 Result of a *beat-aware* optimal one-hop dance motion transition. Our extended framework inserted a hop motion so that it shared the same beats as the source and destination dance motion clips. Thus, the resulting transition was well synchronized to the underlying music

very quick transitions, it would be useful to provide an option to enforce (or favor) this condition in the search.

Blending duration We used a fixed blending duration to maintain the simplicity of our framework, but making this variable adaptive [37] would be beneficial to improving both the naturalness and the responsiveness. Thus, we suggest that developers of motion control systems pursue that direction if they require increased quality.

Separate handling of body parts It would be an interesting future work to investigate how to apply one-hop transitions in a situation with some body parts (e.g., the upper and lower bodies) separately controlled. It is straightforward to independently precompute optimal one-hop transitions for each body part and play them at runtime; however, it is not trivial to determine how that approach would affect the naturalness of transitions.

Efficient search strategy Our current implementation uses a simple exhaustive search to obtain optimal solutions in precomputation. Our (unoptimized) code typically takes a 36-core server about one hour to perform the search for around 1000 possible triggered frames (depending on the total length of the motion clips to be used). If necessary, using a heuristic search algorithm such as beam search could reduce the precomputation cost. It would also be interesting to seek efficient algorithms that use the found solution for a triggered frame to obtain the solution of the next frame, because optimal solutions are often similar to those of their neighboring frames. This idea could also be used for compressing the lookup table to reduce the runtime memory usage.

Positional and orientational constraints We do not yet incorporate positional or orientational constraints in the search (i.e., searching for an optimal one-hop transition that results in a certain amount of translation and rotation), which are necessary for certain use cases. This would require some extensions to the objective of the search, as well as a way of managing the lookup table.

Interpolated motions We assumed a scenario in which every state is associated with a single motion clip to play. It is also a common practice, however, to associate some states with parametric motions defined as continuous interpolations of multiple motion clips. This approach is used for dynamically generating motion variations; for example, interpolating running and walking motion clips generates intermediate-speed walking motions. Also, time warping techniques can be combined to create further variations. We have not yet considered this scenario. It would thus be a good future work to further improve the practicality of our framework by handling dynamically interpolated motions.

Creation of dedicated motion datasets To demonstrate our framework, we used an existing dataset of dance motions for hop motion candidates. It would be an interesting future work to create motion datasets dedicated to such use for hop motions. An ideal dataset could improve our framework in providing more natural transitions (even for transitions between totally dissimilar motions such as from sleeping to running), providing quicker transitions, and reducing the runtime memory usage.

Connection of nonoverlapping motions Our framework can also be used for connecting two nonoverlapping motion clips (i.e., connecting the last frame of one clip and the first frame of another). This situation often arises (e.g., [1,32]), and is typically solved by smoothing the naïvely connected motion with a certain window duration. Rather than smoothing, our framework connects the two nonoverlapping motions by inserting an appropriate hop motion.

Reason for precomputing every frame Motion-graph-based methods (e.g., [1,16,23]) first create a graph structure only once and then search "paths" in the graph by considering user-specified constraints to synthesize motions. In contrast, we compute optimal one-hop transitions (one-hop graphs) for every possible triggered frame. This is necessary because the strategy of being as responsive as possible means that every frame can produce a different optimal one-hop transition.

More sophisticated metrics for motion transition Our current implementation uses the distance metric used by Kovar et al. [16] to measure and ensure the smoothness of transitions. This metric, however, only considers the pose-space geometric distances between motion segments and does not consider either the semantics or styles of motions. Thus, it may potentially cause insertion of a perceptually awkward hop motion. Measurement of such motion compatibility related to visual perception could be enabled by a data-driven (or machine-learning) approach with crowdsourced data generation, as researchers have recently demonstrated the effectiveness of that approach in various domains [4,8,19,22,26].

Ease of extension Our framework is simple enough to easily be extended according to specific usage scenarios. We demonstrated an extension of our framework in the previous section (see Fig. 12), in which we could make the search module aware of musical beats by simply modifying the search space. Similarly, we could offer another extension of the search module as follows. As our dataset has annotations of dance genres (i.e., hip-hop, jazz, and pop), we could create a *genre-aware* version of the search module for interactive dancing applications. This could be achieved by, for example, adding a *genre* cost term to the search objective in Eq. 2:

$$G(i) = \begin{cases} 0, & \text{if } \mathbf{M}_i \text{ is from the target genre} \\ c, & \text{otherwise} \end{cases}, \tag{6}$$

where $c \in \mathbb{R}_{>0}$ is a constant value to control the genre term's strength. In this way, the search module could favor hop motions from a target genre (e.g., the genre of the source and destination motions), but it could also choose a hop motion from different genres when necessary to ensure sufficient smoothness. Hence, we can extend the search to favor solutions that are more context appropriate.

Applications to other domains We believe that the design of our optimal one-hop transition framework is applicable not only to character motion but also to other domains. For example, "interactive music" [14] is a technique to make the transitions of background music in games naturally responsive to game contexts. Inserting an optimal hop (or *fill*) would potentially create quicker, more natural transitions for that technique.

Funding This work was supported in part by JST ACCEL Grant Number JPMJAC1602, Japan.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Arikan, O., Forsyth, D.A.: Interactive motion generation from examples. ACM Trans. Graph. **21**(3), 483–490 (2002). https://doi.org/10.1145/566654.566606
2. Arikan, O., Forsyth, D.A., O'Brien, J.F.: Pushing people around. In: Proceedings of the SCA '05, pp. 59–66 (2005). https://doi.org/10.1145/1073368.1073376
3. Chai, J., Hodgins, J.K.: Constraint-based motion optimization using a statistical dynamic model. ACM Trans. Graph. (2007). https://doi.org/10.1145/1276377.1276387
4. Chaudhuri, S., Kalogerakis, E., Giguere, S., Funkhouser, T.: Attribit: content creation with semantic attributes. In: Proceedings of the UIST '13, pp. 193–202 (2013). https://doi.org/10.1145/2501988.2502008
5. Clavet, S.: Motion matching: road to next-gen animation. Presented at GDC '16 (2016)
6. Egbert, C., Egbert, P.K., Morse, B.S.: Real-time motion transition by example. In: Articulated Motion and Deformable Objects, pp. 138–147 (2010)
7. Epic Games, Inc.: State machines. https://docs.unrealengine.com/en-US/Engine/Animation/StateMachines (2019). Retrieved 31 Jan 2019
8. Garces, E., Agarwala, A., Gutierrez, D., Hertzmann, A.: A similarity measure for illustration style. ACM Trans. Graph. **33**(4), 93:1–93:9 (2014). https://doi.org/10.1145/2601097.2601131
9. Gleicher, M., Shin, H.J., Kovar, L., Jepsen, A.: Snap-together motion: assembling run-time animations. In: Proceedings of the I3D '03, pp. 181–188 (2003). https://doi.org/10.1145/641480.641515
10. Holden, D., Komura, T., Saito, J.: Phase-functioned neural networks for character control. ACM Trans. Graph. **36**(4), 42:1–42:13 (2017). https://doi.org/10.1145/3072959.3073663
11. Holden, D., Saito, J., Komura, T.: A deep learning framework for character motion synthesis and editing. ACM Trans. Graph. **35**(4), 138:1–138:11 (2016). https://doi.org/10.1145/2897824.2925975
12. Ikemoto, L., Arikan, O., Forsyth, D.: Knowing when to put your foot down. In: Proceedings of the I3D '06, pp. 49–53 (2006). https://doi.org/10.1145/1111411.1111420
13. Ikemoto, L., Arikan, O., Forsyth, D.: Quick transitions with cached multi-way blends. In: Proceedings of the I3D '07, pp. 145–151 (2007). https://doi.org/10.1145/1230100.1230125
14. Iwamoto, S.: Epic and interactive music in 'final fantasy xv'. Presented at GDC '17 (2017)
15. Kovar, L., Gleicher, M.: Flexible automatic motion blending with registration curves. In: Proceedings of the SCA '03, pp. 214–224 (2003)
16. Kovar, L., Gleicher, M., Pighin, F.: Motion graphs. ACM Trans. Graph. **21**(3), 473–482 (2002). https://doi.org/10.1145/566654.566605
17. Kovar, L., Schreiner, J., Gleicher, M.: Footskate cleanup for motion capture editing. In: Proceedings of the SCA '02, pp. 97–104 (2002). https://doi.org/10.1145/545261.545277
18. Koyama, Y., Goto, M.: Optimo: Optimization-guided motion editing for keyframe character animation. In: Proceedings of the CHI '18, pp. 161:1–161:12 (2018). https://doi.org/10.1145/3173574.3173735
19. Koyama, Y., Sakamoto, D., Igarashi, T.: Crowd-powered parameter analysis for visual design exploration. In: Proceedings of the UIST '14, pp. 65–74 (2014). https://doi.org/10.1145/2642918.2647386
20. Lau, M., Kuffner, J.J.: Behavior planning for character animation. In: Proceedings of the SCA '05, pp. 271–280 (2005). https://doi.org/10.1145/1073368.1073408
21. Lau, M., Kuffner, J.J.: Precomputed search trees: Planning for interactive goal-driven animation. In: Proceedings of the SCA '06, pp. 299–308 (2006)
22. Laursen, L.F., Koyama, Y., Chen, H.T., Garces, E., Gutierrez, D., Harper, R., Igarashi, T.: Icon set selection via human computation. In: Proceedings of the Pacific Graphics '16 – Short Papers, pp. 1–6 (2016). https://doi.org/10.2312/pg.20161326
23. Lee, J., Chai, J., Reitsma, P.S.A., Hodgins, J.K., Pollard, N.S.: Interactive control of avatars animated with human motion data. ACM Trans. Graph. **21**(3), 491–500 (2002). https://doi.org/10.1145/566654.566607
24. Lee, Y., Wampler, K., Bernstein, G., Popović, J., Popović, Z.: Motion fields for interactive character locomotion. ACM Trans. Graph. **29**(6), 138:1–138:8 (2010). https://doi.org/10.1145/1882261.1866160
25. Levine, S., Wang, J.M., Haraux, A., Popović, Z., Koltun, V.: Continuous character control with low-dimensional embeddings. ACM Trans. Graph. **31**(4), 28:1–28:10 (2012). https://doi.org/10.1145/2185520.2185524
26. Liu, T., Hertzmann, A., Li, W., Funkhouser, T.: Style compatibility for 3d furniture models. ACM Trans. Graph. **34**(4), 85:1–85:9 (2015). https://doi.org/10.1145/2766898
27. Min, J., Chai, J.: Motion graphs++: a compact generative model for semantic motion analysis and synthesis. ACM Trans. Graph. **31**(6), 153:1–153:12 (2012). https://doi.org/10.1145/2366145.2366172
28. Min, J., Chen, Y.L., Chai, J.: Interactive generation of human animation with deformable motion models. ACM Trans. Graph. **29**(1), 9:1–9:12 (2009). https://doi.org/10.1145/1640443.1640452
29. Mizuguchi, M., Bochanan, J., Calvert, T.: Data driven motion transitions for interactive games. In: Eurographics 2001—Short Presentations (2001). https://doi.org/10.2312/egs.20011039
30. Mukai, T., Kuriyama, S.: Geostatistical motion interpolation. ACM Trans. Graph. **24**(3), 1062–1070 (2005). https://doi.org/10.1145/1073204.1073313
31. Pullen, K., Bregler, C.: Motion capture assisted animation: texturing and synthesis. ACM Trans. Graph. **21**(3), 501–508 (2002). https://doi.org/10.1145/566654.566608
32. Reitsma, P.S.A., Pollard, N.S.: Evaluating motion graphs for character animation. ACM Trans. Graph. **26**(4), 18:1–18:24 (2007). https://doi.org/10.1145/1289603.1289609
33. Rose, C., Guenter, B., Bodenheimer, B., Cohen, M.F.: Efficient generation of motion transitions using spacetime constraints. In: Proceedings of the SIGGRAPH '96, pp. 147–154 (1996). https://doi.org/10.1145/237170.237229
34. Safonova, A., Hodgins, J.K.: Construction and optimal search of interpolated motion graphs. ACM Trans. Graph. **26**(3), 106:1–106:11 (2007). https://doi.org/10.1145/1276377.1276510

35. Unity Technologies: Unity—manual: Animation transitions. https://docs.unity3d.com/Manual/class-Transition.html (2018). Publication: 2018.3-002N. Built: 2018-12-04
36. Unity Technologies: Unity—manual: State machine basics. https://docs.unity3d.com/Manual/StateMachineBasics.html (2018). Publication: 2018.3-002P. Built: 2019-01-17
37. Wang, J., Bodenheimer, B.: Synthesis and evaluation of linear motion transitions. ACM Trans. Graph. **27**(1), 1:1–1:15 (2008). https://doi.org/10.1145/1330511.1330512
38. Zadziuk, K.: Motion matching: The future of gameplay animation... today. Presented at GDC '16 (2016)

Yuki Koyama is a Researcher at the National Institute of Advanced Industrial Science and Technology (AIST), Japan. He received his Ph.D. from the University of Tokyo in 2017. His research interests include computer graphics and human-computer interaction. In particular, he is interested in developing computational techniques for enabling new interactions, producing creative artifacts, and enhancing design processes.

Masataka Goto received the Doctor of Engineering degree from Waseda University in 1998. He is currently a Prime Senior Researcher at the National Institute of Advanced Industrial Science and Technology (AIST), Japan. Over the past 27 years he has published more than 250 papers in refereed journals and international conferences and has received 47 awards, including several best paper awards, best presentation awards, the Tenth Japan Academy Medal, and the Tenth JSPS PRIZE. He has served as a committee member of over 110 scientific societies and conferences, including the General Chair of ISMIR 2009 and 2014.