Computational Visual Media
https://doi.org/10.1007/s41095-019-0138-z Vol. 5, No. 2, June 2019, 171–191
Research Article

Optimal and interactive keyframe selection for motion capture

Richard Roberts[1] (✉), **J. P. Lewis**[2], **Ken Anjyo**[1,3], **Jaewoo Seo**[4], **and Yeongho Seol**[5]

Abstract Motion capture is increasingly used in games and movies, but often requires editing before it can be used, for many reasons. The motion may need to be adjusted to correctly interact with virtual objects or to fix problems that result from mapping the motion to a character of a different size or, beyond such technical requirements, directors can request stylistic changes. Unfortunately, editing is laborious because of the low-level representation of the data. While existing motion editing methods accomplish modest changes, larger edits can require the artist to "re-animate" the motion by manually selecting a subset of the frames as keyframes. In this paper, we *automatically* find sets of frames to serve as keyframes for editing the motion. We formulate the problem of selecting an *optimal* set of keyframes as a shortest-path problem, and solve it efficiently using dynamic programming. We create a new simplified animation by interpolating the found keyframes using a naive curve fitting technique. Our algorithm can simplify motion capture to around 10% of the original number of frames while retaining most of its detail. By simplifying animation with our algorithm, we realize a new approach to motion editing and stylization founded on the time-tested keyframe interface. We present results that show our algorithm outperforms both research algorithms and a leading commercial tool.

Keywords motion capture; motion editing; keyframe animation; dynamic programming

1 Victoria University of Wellington, Wellington, New Zealand. E-mail: richard.andrew.roberts@gmail.com (✉).
2 SEED, Electronic Arts, Los Angeles, United States. E-mail: noisebrain@gmail.com.
3 OLM Digital, Tokyo, Japan. E-mail: anjyo@acm.org.
4 Pinscreen, Los Angeles, United States. E-mail: goongsang@gmail.com.
5 Weta Digital, Wellington, New Zealand. E-mail: seolyeongho@gmail.com.

Manuscript received: 2018-12-13; accepted: 2019-02-15

1 Introduction

Motion capture ("mocap") is widely used in games and movies. However, the raw motion capture data is rarely usable for final production. It is often necessary to edit the mocap, for several reasons:

- Complex interactions between characters, such as hugging or wrestling, can introduce tracking and solving errors due to extensive occlusion and close proximity of the characters.
- Motion is frequently "retargeted" to virtual fantasy characters with different proportions. Such retargeted motion may require editing to look realistic.
- It is often necessary to adjust the character's motion to fit the virtual environment. This can happen if the proportions of the mocap studio do not match the virtual environment: for example, the virtual terrain may be uneven while the studio has a flat floor. Beyond matching problems, changes to the design of a scene or object can also occur during post-production after motion is captured. For example, the heights of doors or controls in a spaceship might change, with consequent changes required to the motion.
- In games, it is often necessary to alter player-triggered moves, such as punches, to be more rapid, to provide a more responsive feel to the game.
- The director may request adjustments to the performance for a variety of reasons.

Unfortunately it is not practical to directly edit mocap, for much the same reason that editing a picture by changing individual pixels is ineffective. Mocap is typically recorded at 30 frames per second (fps) or higher (60 and 120 fps are common), and the body model may have 50 degrees of freedom or more, so even a short motion may be represented by tens of thousands of numbers.

A standard approach provided by motion editing and animation software is to allow the motion editor to blend changes to a given pose smoothly across surrounding frames using a spline falloff[①]. This approach is suitable for relatively small and smooth edits, but it raises the question of where to place the control vertices (CVs) or keyframes on such a spline. Simply placing keys at regular intervals is adequate for mild edits, but does not provide precise control, since the range of influence of each CV is not related to the motion. For example, adjusting the motion before a footstep, but not afterwards, generally requires having a CV located at the time of the step.

In practice, when extensive editing is required, motion editors sometimes resort to manually deleting ranges of frames to create an editable representation with a small number of frames that can serve as keyframes [1, 2].

1.1 Terminology

In this paper we will use the word *keyframe* (KF) to denote a frame where a key exists for every degree of freedom (i.e., on every motion curve), whereas *keypoint* will denote a key on a single curve at a frame in which other curves may not have keys.

1.2 Approach

While a number of pioneering techniques, such as space–time optimization and sketching interfaces provide powerful and novel interfaces for motion editing, they are yet to be widely adopted in commercial motion editing practice.

Keyframe animation is both widely supported and a central part of training for both motion editors and animators. Based on the observation that animation practice generally recommends working with keyframes first and keypoints later, we suggest that introducing a motion editing technique based primarily on the manipulation of keyframes will help with adoption[②].

In this paper we introduce an interactive and optimal way to identify keyframes from a motion that are important for editing. With a set of keyframes identified, we build a new animation in which the selected keyframes serve as a minimal set of poses that can be interpolated to closely approximate the original motion of the character or object. By recreating the motion in this way we can overcome the problem that motion capture is difficult to edit due to the low-level representation of its data.

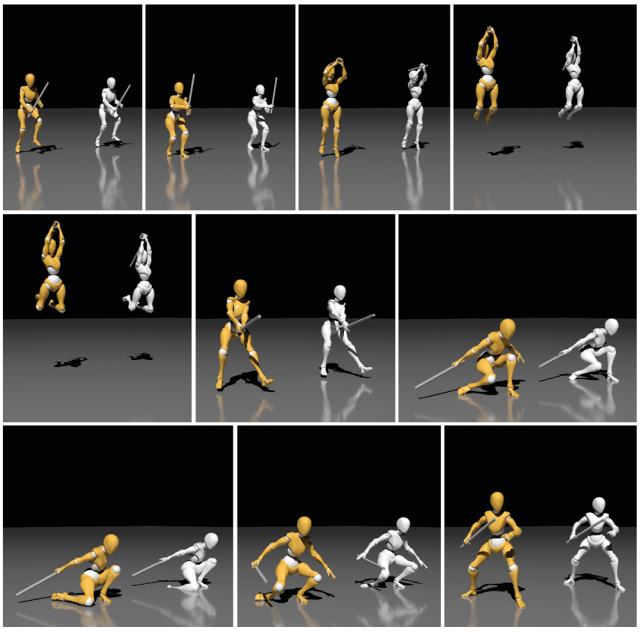


Fig. 1 Frames from an animation, before (character on right in grey) and after (character on left in yellow) editing using our technique.

In abstract terms, a sparse set of keyframes provide an economical set of parameters that provide complete control over an animation. From this perspective, our approach converts mocap into a simplified keyframe-based animation that can be edited using the extensive and fluid support for keyframe-based editing that commercial software provides. A number of approaches for keyframe selection have already been explored as an extension to keypoint selection methods; these are summarized in Section 2. Conceptually, keypoint selection is the process of identifying a sparse set of points that summarize a curve effectively. Extending to the case of animation, keyframe selection is the process of identifying a sparse set of frames that summarize the motion effectively.

One aspect that demands careful consideration is the criterion that determines the importance of each frame. Some keypoint approaches assume that points of high curvature or curvature extrema provide good keypoints for approximation. While these approaches

① Tools for falloff editing are provided by most feature-complete digital content creation tools such as Autodesk's Maya and Blender Foundation's Blender.

② Animation practice generally recommends working in terms of poses (keyframes) when making initial large-scale edits, followed by editing individual curves in a subsequent refinement stage. This is termed *pose-to-pose* animation [3, 4].

are simple to implement and inexpensive to evaluate, it is easy to find counterexamples where these simple heuristics lead to a set of keypoints that provide a poor summary (see Fig. 2). In contrast, other approaches take the approach of fitting a geometric approximation and choosing points that correspond to the approximation's vertices; the well-known Ramer–Douglas–Peucker algorithm [5] and its variations often perform well in contrast to the extrema-based approaches. While still fast, a primary limitation of greedy approximation approaches is that they tend to produce suboptimal solutions. Recent research has explored how genetic algorithms and particle swarm optimization can be used to choose keyframes that minimize a combination of compression and approximation error. However, the problem of choosing keyframes to minimize such a function leads to a non-convex problem space. Converging to the optimal solution in this space requires heavier computation and is not practical for interactive use (see Sections 2.1 and 4.4).

As our primary contribution, we provide the first interactive and optimal approach for keyframe selection. To realize a solution that is both interactive and optimal, we formulate the problem as a search for a minimal set of frames that, when interpolated, provide the closest approximation to the entire motion. This is a combinatorial problem as there are $\binom{N}{k}$ potential choices of k keyframes to summarize N frames. While this is intractable, our formulation of the problem is related to a particular form of the classic shortest path problem [6] and, consequently, we can use an efficient dynamic programming algorithm to obtain the optimal solution. In practice

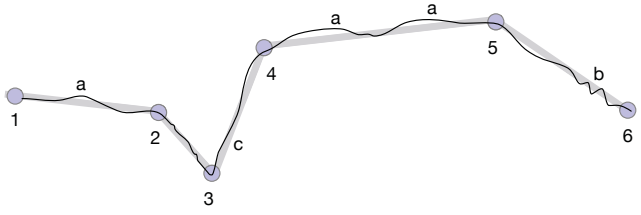


Fig. 2 Picking important points on a motion using simple intuitive principles such as extrema (**a**) or points of high curvature (**b**) can easily fail. In this schematic diagram, the freeform curve represents the motion, and points represent keyframes. Points **(1)**–**(6)** nicely summarize this one-dimensional curve, although points **(2), (4), (5)** are neither points of high curvature nor extrema. Instead of focusing on points individually, our approach considers the error in approximating the intervening motion by pairs of points (visualized as the grey line in this one-dimensional case). For any number of points, we find the *optimal* set of points in terms of reducing this approximation error.

an optimal set of keyframes can be found in a reasonable time (e.g., less than a second for clips of typical length) when executed on a personal computer.

Our algorithm targets professional motion editors. While these are a fraction[①] of the potential novice users of any computer graphics technique, they have both disproportionate need and impact: motion editors perform editing many hours per day year round whereas a casual user might only occasionally launch a graphics tool. Furthermore, much of the world's population has experienced movies or games that include motion-edited characters. An important aspect of our algorithm, with regards to professional motion editing, is that in a single execution it provides the optimal solution for *all* numbers of keyframes less than or equal to a number requested by the artist. After execution all solutions are encoded in a lookup table and, therefore, the entire range of solutions can be browsed interactively by specifying differing numbers of keyframes. Browsing the solutions in this way is important to help artists find the particular solution that is best suited to their editing task.

In terms of quality, the resulting keyframe animation closely resembles the original motion, but is now editable using standard approaches and polished industrial tools. Additionally, fine-scale detail lost through keyframe approximation can be saved by subtracting the approximation from the original. If it is important to preserve such lost fine-scale detail this residual can be added back to the simplified animation after editing[②]. Adding the residual re-establishes the fine-scale detail so that unedited areas are recreated exactly as in the original. While adding the residual can be useful to preserve such detail, it can result in artefacts after editing. Consider the case of the fine-scale finger motion of someone playing the guitar. If the artist changes the motion such that the guitar playing pauses momentarily, adding the residual vibrations of the fingers during that pause may appear as an artefact. Consequently, the extent to which the residual layer is employed must be directed by the artist.

① For example the U.S. game industry is estimated at 220,000 jobs [7], while the global games audience has been estimated at more than 2.2 billion people [8].
② Leading animation software typically provides a mechanism to compose animation by summing multiple channels, such as Autodesk's Maya's *animation layers*.

In summary, we contribute the first interactive and optimal solution to the important problem of editing motion capture. Importantly, our solution respects the keyframe interface typically preferred by artists. As we demonstrate in Section 4, our solution outperforms both existing methods and a leading commercial tool.

2 Related work

In this section, we examine the closely related work on keyframe selection for mocap (Section 2.1), briefly summarize research surrounding motion editing (Section 2.2), and touch upon tangential work (Section 2.3). We conclude the section by outlining a set of limitations that are addressed by our algorithm (Section 2.4). See Ref. [9] for a recent survey of processing and editing techniques for mocap.

2.1 Keyframe selection

The problem of identifying KFs resembles the problem of picking a perceptually important subset of vertices on a curve so as to simplify it. In contrast to that problem, the KF problem is high-dimensional.

To provide context for our algorithm, we present a brief review of the closely related literature on keyframe selection, organizing the reviewed work into three categories:

- **detection**, in which local properties of the motion's data are used to identify keyframes;
- **approximation**, in which geometry is fitted to the motion and keyframes are selected as those frames that correspond to the fitted geometry's vertices, and
- **optimization**, in which a function providing an error-based criterion for keyframe selection is minimized.

2.1.1 Detection

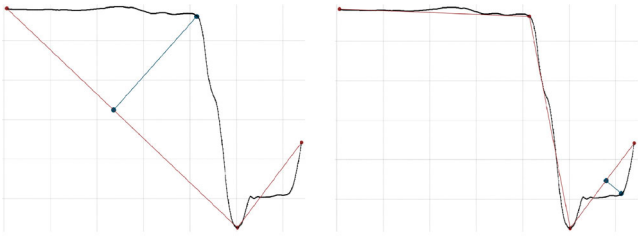
Detection approaches identify points on a curve that should be retained as keypoints using local properties without considering the remainder of the curve. A common criterion is to pick keypoints based on derivative features, such as points of high curvature or extrema (derivative zero crossings). Techniques have been proposed that use extrema of the first principle component of the animation data [10], that pick keypoints using curvature-related finite differences operating at a coarse scale [11], that consider changes between neighbouring poses [12–14], and others that use a saliency measure based on differences between Gaussian-weighted averages at different scales [15, 16]—this measure can also be interpreted as a derivative, recalling a classic vision result that the difference of Gaussians closely approximates the Laplacian of a Gaussian-smoothed version of the signal [17].

While detection methods can be efficient in terms of computational effort, it is difficult to directly control the number of keypoints obtained. Furthermore some postprocessing is generally needed to remove near-duplicate detections and those due to noise [15, 16].

These methods are poorly suited to our problem: they use a purely local objective and, consequently, can fail to find points that summarize the entire curve (or, in our case, the motion). In addition, most of these methods are only formulated for the one-dimensional case, although analogous high-dimensional methods may be possible. In any case, they are rarely able to offer explicit control over the number of keyframes provided, which limits the number of editing tasks to which they can be applied.

2.1.2 Approximation

Rather than examining points in isolation, curve approximation techniques consider the accuracy of approximating the entire curve when picking keypoints, and thus produce generally better results. A number of methods are variants of the Ramer–Douglas–Peucker (RDP) algorithm [5, 18, 19]. In this greedy algorithm, the curve is approximated by a chord, and the point on the curve that is furthest (in a perpendicular direction) to the chord is selected as a new keypoint. This keypoint divides the curve into two, and the algorithm recurses on each sub-curve, terminating when some error tolerance is reached. Figure 3 illustrates two iterations.



(a) Selecting keypoint #4 (b) Selecting keypoint #5

Fig. 3 Iterations performed by the Ramer–Douglas–Peucker (RDP) algorithm for identifying keypoints that summarize a curve [5, 18, 19]. In each iteration, the algorithm adds the keypoint furthest from the piecewise linear interpolation of the current set of keypoints.

In the context of motion capture, Lim and Thalmann [20] introduced the idea of applying an RDP-like approach to compressing mocap by simplifying the high-dimensional curve representing the motion. A reverse variation has also been proposed, in which the algorithm starts with the full curve and eliminates unimportant points. While both techniques employ the same measure, the reverse algorithm generally performs less well [21].

With the high-dimensional representation, approximation techniques have the advantage of identifying keyframes based on poses in the motion rather than keypoints on individual motion curves. Furthermore, they also are relatively fast as they need only scan the motion once in each iteration. Despite these advantages, their greedy design means that the resulting keyframes are a suboptimal representation of the motion.

2.1.3 Optimization

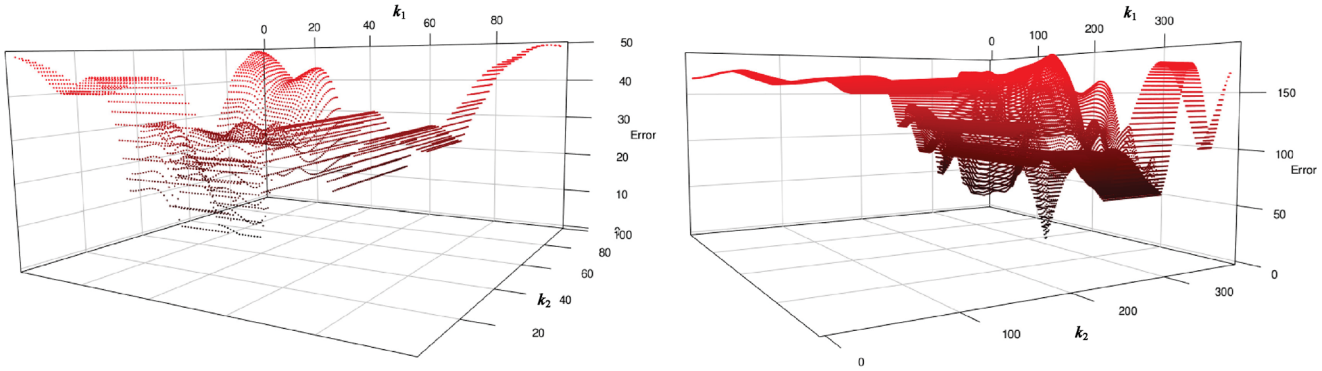
The use of global optimization to approximate mocap has also been explored, with the benefit that the resulting solutions offer the best trade-off between compression and error. In contrast to approximation techniques, the advantage of finding salient poses is that the latter can offer higher levels of sparsity while preserving the same level of detail.

Since the problem of optimal approximation is combinatorial, researchers have employed evolutionary algorithms such as particle-swarm optimization and genetic algorithms [22–25]. While these approaches can produce optimal results they unfortunately must traverse a highly non-linear problem space (see Fig. 4) and generally cannot provide a bound on the time required to find the global optimum. In practice, considerable computation may be required.

Beyond evolutionary algorithms, another approach to optimization is dynamic programming. In the case of one-dimensional curves, efficient and optimal dynamic programming algorithms have been used for a number of purposes, including incremental spline fitting [26], curve simplification [21], and finding perceptually important points [27]. As described later in Section 3, we extend such dynamic programming techniques to the higher-dimensional problem of motion capture to realize our approach.

2.2 Motion editing research

Researchers in computer graphics have developed a number of novel, powerful, and efficient motion editing techniques that bypass the use of keyframes. These include methods that optimize a motion to simultaneously satisfy editing constraints and (typically derivative) similarity to the original motion, using a space–time approach [28–31]. In fact, some of these approaches resemble KF interpolation in an abstract form, in that minimizing a derivative energy is a foundational principle for splines. However, they differ in that edit constraints may be applied temporarily as needed, at arbitrary frames, and



(a) Taunt (99 frames) (b) Opening door inwards (383 frames)

Fig. 4 Finding the optimal set of keyframes to approximate a motion is a difficult problem. Here we show a problem of choosing just two additional keyframes, k_1 and k_2, to approximate a motion (resulting in the set of keyframes $\{1, k_1, k_2, n\}$, where 1 and n are the first and last frames of the motion). Horizontal axes correspond to k_1 and k_2, while the vertical axes gives the resulting error. Importantly, note that the problem space is not convex. Instead, it contains multiple extrema and regions that break geometric continuity. When choosing more than two keyframes the dimensionality of the problem space increases and, consequently, both the number of extrema and the number of regions breaking geometric continuity increase. These attributes make optimizing keyframe selection computationally expensive for numerical solvers and genetic algorithms. Note: error values here have been adapted to clarify the visualization: the error values are first scaled so that they span $[0, 1]$ and then scaled again by half the number of frames in the motion.

at isolated (incomplete) degrees of freedom. On the other hand, in the case of KF animation, such "constraints" are usually temporally aligned and form the fundamental and relatively permanent representation of the data. Consequently, there remains a disconnect between these techniques and keyframe animation practice. Other work has introduced motion editing interfaces that discard the idea of KF animation by instead allowing the artist to fluidly sketch aspects of the motion or pose [32, 33]. We will not further survey these sketching and optimization-based editing approaches since they are not directly related to our goal of identifying KFs to support a traditional animation approach.

2.3 Compression and other techniques

Other topics that use keyframe-like representations, but are otherwise tangential to our purpose of keyframe selection for editing include motion remixing [34–36], retargeting [37], synthesis [38], and visualization and summarization [39–41].

While keyframe identification produces a compressed approximation of the motion in terms of the interpolated KFs, other methods that focus specifically on compression of mocap [42–45] generally do not satisfy our purpose of producing an editable representation. This is because compression methods typically represent information at a particular frame as a weighted linear sum of multiple basis vectors that have no temporal identity. This differs considerably from the interface provided by KF animation, where the pose of a frame can be determined by nonlinear spline interpolation of the closest two KFs (with corresponding tangents) that temporally bound the particular frame.

2.4 Limitations of previous work

From this survey of related work we see that research methods surrounding motion editing and compression do not offer the ability to edit mocap using the same interface as keyframes.

In contrast, keyframe selection techniques can be applied to convert mocap into an editable keyframe animation, but suffer from one or more limitations with respect to our goal of identifying keyframes suitable for editing:

- simplification of individual curves rather than addressing the high-dimensional problem of identifying poses;
- the use of local objectives (detection) or greedy algorithms that do not always produce the best result, and
- the use of stochastic optimization, which is needlessly inefficient (see Section 4.4) and does not provide a bound on the time required to obtain an optimal result.

In the next section we introduce a new algorithm, named *Salient Poses*, that does not have any of these limitations.

Section 4 demonstrates the advantages of this algorithm through comparison to alternative approaches and a commercial tool.

3 Fast and optimal keyframe selection

3.1 Considerations and objectives

The qualities of "good" keyframes are a matter of artistic judgement rather than mathematical definition. Individual artists do not always agree on some aspects of what constitutes a "good" keyframe.

We have discussed our research with seven professional motion editors and animators from three major companies and found the following disagreements:

- *Flat versus free tangents.* Animation practice [4] often recommends that the tangents of individual curves at keyframes should be flat (have zero derivative) wherever possible, ensuring that keyframes are extrema of the motion. While one of our experts agreed with this practice, the majority advocated allowing the tangents to be "free" so as to best fit the motion. One artist presented the other extreme, preferring to displace flat tangents so as to avoid a "pose-to-pose" or cartoon animation feel.
- *Whole pose versus individual curves.* In general artists prefer to work in terms of KFs early in the editing process to the extent possible, and switch to editing of individual curves when necessary to add details [3, 4]. However one artist felt that using KFs biases the result, and thus he prefers to work at the level of individual curves.

Nevertheless, there are several criteria that are both self-evident and supported in animation literature:

1. Keyframes may occur at the extremes of the motion, and more generally at locations where there are large-scale changes. For example, imagining that

the keypoint **(3)** in Fig. 2 depicts a point of impact, this keypoint permits both sides of the impact to be edited, whereas a keypoint at **(c)** would not. The maxima at points **(a)**, **(b)** are not important, but **(a)** will become important as more keypoints are added. Finally, point **(4)** is important despite not being an extremum.

2. The spacing of keyframes should be proportional to the amount of change. All other things being equal, a representation in which the keyframes are spread more evenly is to be preferred.

While the desirable points **(1)**–**(6)** in Fig. 2 have little in common, it is clear that they are vertices of a polyline that closely approximates the curve. From this observation, we find the following approximation principle is evident: *keyframes are those frames that best allow the remainder of the motion to be interpolated.*

3.2 Algorithm

To solve our problem, we express it as a particular shortest path problem [6], having a dynamic programming algorithm for solution. Each of N frames in the original mocap clip, in which every frame is a keyframe, becomes a node in a graph with directed edges to all temporally subsequent nodes (see Fig. 5). The weight of an edge $e_{i,j}$ is the cost of approximating the high-dimensional motion between frames i, j by an approximation that uses nodes i and j as keyframes. The approximation and error measure (least squares, absolute value, infinity norm, etc.) are both design choices; we describe our choice in Section 3.3.

Our problem differs only slightly from the single-source-all-destinations problem used to motivate Dijkstra's well known algorithm [46]. Similar to Dijkstra's formulation of the problem, we seek a single minimum-cost path from the start node (here, the first frame of the animation) to the end node (the last frame). Unique to our problem, the keyframes have a total (temporal) ordering and we seek a solution that passes through a number (denoted K, and specified by the artist) of intermediate keyframes while skipping the remaining keyframes.

3.2.1 All pairs table

As a preprocessing step, a table

$$\{e_{i,j}; \quad 1 \leqslant i < j \leqslant N\}$$

of all edge costs is computed. The table has $\binom{N}{2}$ unique entries and hence quadratic cost. The entries can be computed independently however, and are computed in parallel in our implementation (taking advantage of parallel computation, typically via the GPU, is essential for good performance).

In practice the cost of this step is minor and generally not noticeable compared to the time required for common interface interactions such as loading files and opening user interface windows.

3.2.2 Successive keyframe selections

Dynamic programming relies on the problem having an *optimal substructure*, which means that the optimal solution for the complete problem is composed of *optimal sub-solutions* to smaller parts of the problem.

Our shortest path problem exhibits an optimal substructure in that the optimal solution using m keyframes is necessarily equal to the combination of: (1) the solution for $m-1$ keyframes between the start

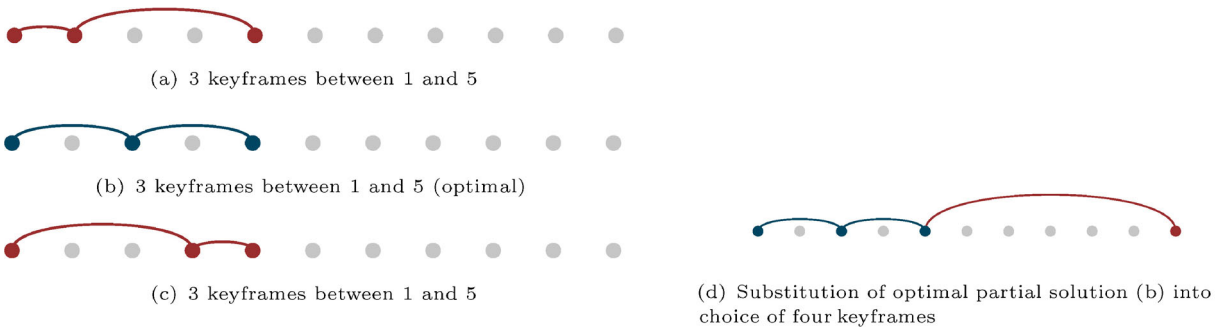


Fig. 5 Salient Poses enumerates selections of keyframes to find those that are optimal. While the computational cost of such a process would otherwise be combinatorial, the solution for a subset of keyframes can be reused during the search for a larger set. This figure illustrates a step in this computation. (a)–(c) schematically show three choices of keyframes for a motion beginning at frame 1 and ending at frame 5. The arcs encode the cost of approximating that span of the motion using the keyframes that bound it. Choice (b) is selected as optimal here. A later step of the search (d) considers the best set of four keyframes lying between frames 1 and 11. In cases where there are no keyframes between 5 and 11, the previously found optimal solution for frames $1 \ldots 5$ is *substituted* into this search, thereby limiting the combinatorial complexity.

node i and some node k, and (2) the (trivial) solution for the remaining frames $k \ldots j$ that contains only the two keyframes at k and j. This structural division can be applied to decompose the solution for $m-1$ keyframes into two parts, which can continue until the base case of $m = 2$. Note that the condition $i < k < j$ must hold.

Using the notion of optimal substructure, our dynamic programming algorithm can be concisely described as

$$E_{i,j}^{m} = \min_{k} \left(E_{i,k}^{m-1} + e_{k,j} \right)$$

with

$$E_{i,j}^{1} \equiv e_{i,j}$$

The formula above states that the optimal approximation of m keyframes for the entire motion, denoted $E_{i,j}^{m}$, can be computed from the optimal sub-solution of $m-1$ keyframes, denoted $E_{i,k}^{m-1}$.

With the observation that each optimal solution of m keyframes can be derived from an optimal sub-solution of $m-1$ keyframes, we can formulate an iterative algorithm that computes the optimal solution for every m as

$$\arg\min_{k} E_{i,k}^{m-1} + e_{k,j}$$

Importantly, the computation for step m is reused in step $m+1$ (see Fig. 5), thereby resulting in a dynamic programming optimization. This is critical as it enables solutions from each $m-1$ to be recycled as sub-solutions in the successor m.

Also note that, unlike the greedy RDP algorithms, the set of keyframes chosen for m keyframes is *not* a superset of those chosen for the solution of its predecessor ($m-1$ keyframes). Nevertheless, our algorithm produces keyframe selections for *all* $m \leqslant K$ (the solution for each m is stored in a table). Preserving each selection of m keyframes is useful in that it allows the artist to interactively browse solutions with various numbers of keyframes and pick one that provides the best trade-off between fidelity and editability with respect to their editing task.

The cost of each step is approximately quadratic, resulting from the search over k, j, and the overall cost is approximately cubic. In fact, the algorithm only needs to proceed as far as K keyframes. However the expected number K generally grows with the length of the mocap sequence, so it is reasonable to summarize the algorithm cost as $O(N^3)$. Figure 16 of Section 4 provides an illustration.

3.2.3 Editable animation from a set of keyframes

Given the range of solutions produced by Salient Poses, we need only create a new animation from a particular solution. This process involves first choosing a particular solution and then building the new animation in a way that is faithful to the original.

With the realization that different editing tasks call for different amounts of control, we entrust the choice of best solution to the artist. In practice, artists can choose to identify the most desirable number of keyframes using a graph that displays the error corresponding to each solution of m keyframes or by examining the poses in the selected keyframes.

With a particular solution, a curve fitting scheme should be applied to recover the new animation. We apply a variation on the iterative curve fitting step from Schneider's well-known algorithm [47], which first prescribes tangents using a finite-difference approximation and then solves for CVs along tangent vectors. We model the problem as the need to minimize the distance between the interpolating spline and the curve, and apply a least squares solution. Note that our variation does not ensure geometric continuity through keyframes, which is desirable for fitting the original animation more accurately. Despite not enforcing geometric continuity, the fitted curves tend to pass through keyframes smoothly. Note that the topic of curve fitting is extensive and we leave the task of identifying the best choice for reconstruction of motion to future work.

3.3 Discussion of choice of approximation

Each node in the graph corresponds to a keyframe. Each keyframe corresponds to a pose for the animated character, containing multiple degrees of freedom (50 degrees or more are common in a standard mocap performance rig).

Geometrically, each node is a point in $\mathbb{R}^D$ with the dimension $D \approx 50$ or more. From this geometric perspective the original mocap can be regarded as a *curve* in this $\mathbb{R}^D$ space. The edge cost e_{ij} measures the approximation error between the portion of this curve lying between keyframes i and j, and the approximated motion. In our implementation, we approximate the motion using a piecewise linear interpolation of the high-dimensional points corresponding to the keyframes of i and j.

Given the approximation primitive, we define the approximation error as the high-dimensional *perpendicular* distance (i.e., shortest distance) from the motion to its approximation, aggregated across frames of the original motion. For the distance we use $\|\cdot\|_\infty$. We also experimented with minimizing the squared error between the high-dimensional curve and its approximation. That did not give any clear advantage, and so we selected the infinity norm for its simplicity and also because it has the interpretation of minimizing the maximum error.

Our algorithm can straightforwardly employ splines rather than piecewise linear approximation. While a high-dimensional spline could also be used as the approximation primitive, we found that the KFs selected using a piecewise linear approximation to the motion are better for editing than those obtained using spline approximation[①]. An example is shown in Fig. 6. Although this observation is necessarily subjective (recalling the differences of expert opinion noted in Section 3.1), we came to the following conclusion after discussion with a number of artists: keyframes obtained using piecewise linear approximation tend to correspond with extremes and inflection points that are important for editing, whereas those from the spline tend to correspond less cohesively to distinctive poses.

4 Results

In the following results, except for those presented in Section 4.1, each frame is represented by a high dimensional point. The first dimension corresponds to time, while the other dimensions correspond to joint positions. We favor positions over rotations since they enable simple and tractable distance measurements. Specifically, each frame corresponds to a single point in a space of 55 dimensions: time in one dimension, and then three dimensions for each further joint in the skeleton (a total of 18 joints, with six joints along the spine and three along each limb).

The mocap clips that we used to create these results were obtained from Adobe's Mixamo [48]. The complete set of results, including playable animation

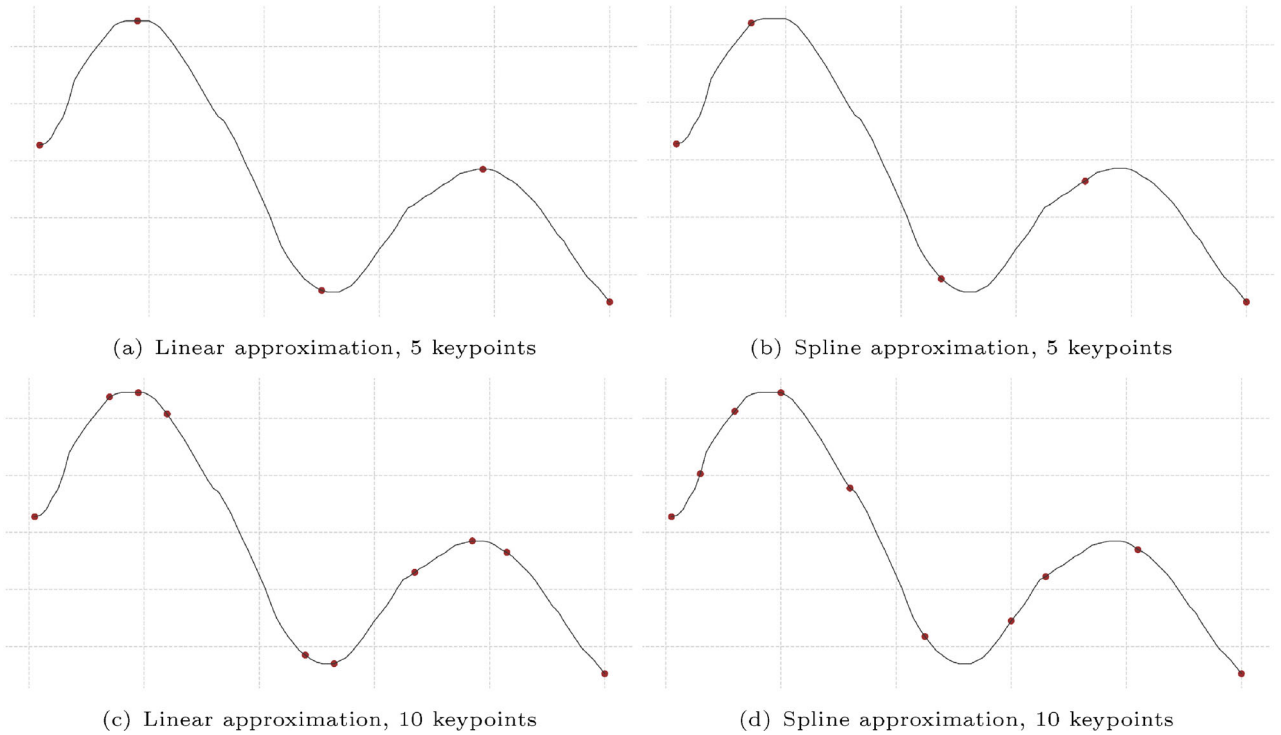


Fig. 6 An example of how keypoints selected using a spline approximation function do not always provide useful controls for editing. Notice how, when using the linear approximation (a, c), the keypoints tend to be located near extrema and inflection points of the curve, at least more so than in the case of spline approximation (b, d). Although harder to quantify, through informal discussion with artists we theorize that this effect extends to the problem of selecting keyframes for higher dimensional curves. See Section 3.3 for further discussion.

① We used Schneider's technique [47], which fits piecewise cubic Bézier curves, to conduct informal pilot experiments that approximated sets of 2D points (time and value) derived from one degree of freedom of the motion capture data.

clips and further auxiliary results can be viewed through our online portal[①]. Please also refer to the video in the Electronic Supplementary Material (ESM).

4.1 1D comparisons

Our algorithm easily generalizes to different dimensions simply by adapting the distance function (see Section 3.3). To visualize the performance of the optimal approximation in a simple setting, we have prepared a version of Salient Poses that operates on digitized one dimensional functions. (Note that our algorithm normally operates on high-dimensional curves representing the character motion.)

Figure 7 shows the performance of our Salient Poses, serving here as an optimal keypoint selection algorithm, versus a greedy approximation algorithm (RDP) and also an algorithm that selects key points based on a form of finite difference [11]. While each algorithm tends to pick distinctive points, Salient Poses is able to do so in a way that minimizes error.

4.2 Comparison to greedy algorithm

As a further comparison with existing greedy algorithms, we apply both the greedy approximation algorithm [20] and our optimal algorithm to select 5, 10, 15, and 30 KFs from a mocap sequence of a jumping action spanning 140 frames; Fig. 8 illustrates the result of each selection as a series of renders presented in a time-lapse image. The better distribution of KFs provided by our algorithm is evident in this figure. Figure 9 shows another example comparing both algorithms in the case of 14 KFs selected from a mocap sequence of an acrobatic running action spanning 56 frames. The better distribution of KFs in the optimal case is again evident in the figure.

Figure 10 presents examples showing the performance of our algorithm versus the greedy algorithm [20]. The performance of the greedy algorithm is sometimes quite good; however, our algorithm is always at least as good, and often better when choosing selections with a level of compression between 85% and 95% (refer to Fig. 11).

While our optimal algorithm is more costly than the greedy algorithm, the cost is not prohibitive with a performant implementation run on modern hardware, as demonstrated in Section 4.4.

① https://salientposes.com/results/

4.3 Error versus compression: Maya, PCA, ours

Figure 12 presents results that compare the error against compression for animations reconstructed using our algorithm, our implementation of a pose-based PCA reconstruction, and Maya's "Simplify Curves" algorithm.

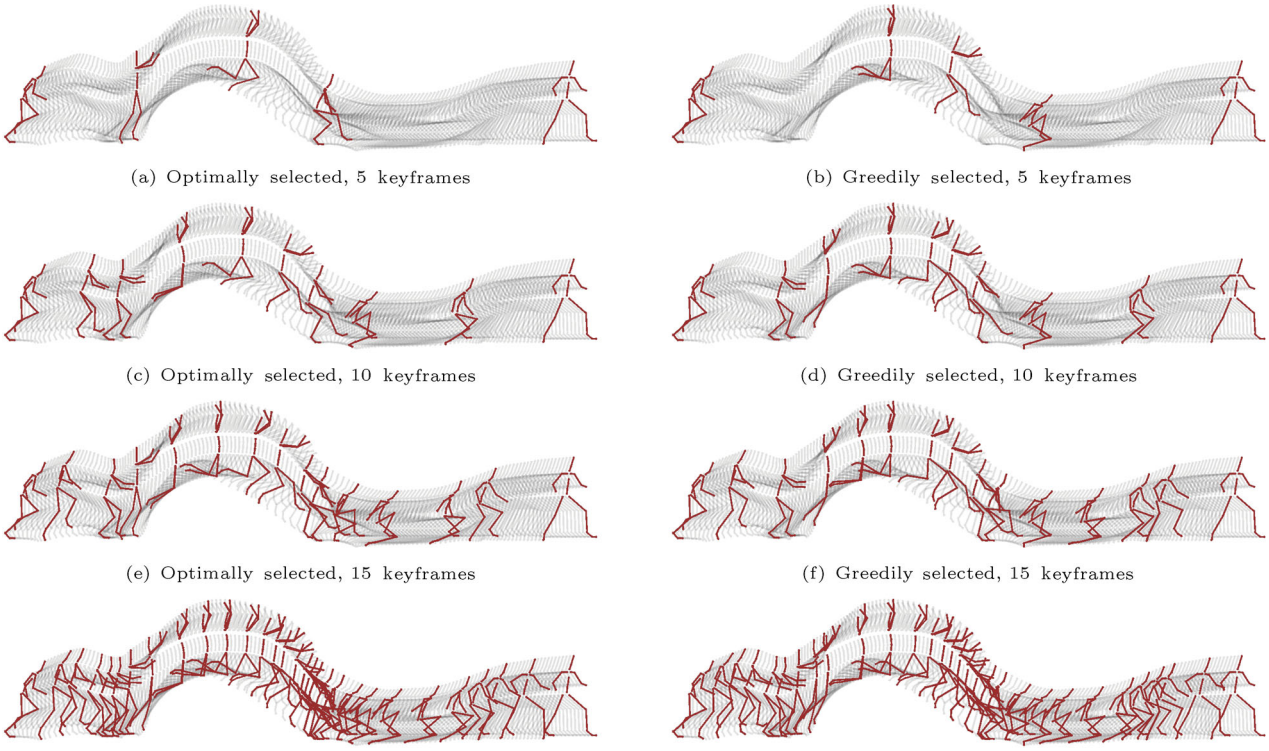
Each algorithm takes a different approach. As outlined in Section 3.2, our algorithm creates a new animation by interpolating a set of selected keyframes. Our PCA approach creates a new animation summarized by eigenvectors rather than keyframes, where each frame is a linear combination of the eigenvectors. Maya's "Simplify Curves" chooses a set of keypoints for each degree of freedom independently, and interpolates those keypoints to recreate the animation.

Each result presents the error of the animation as reconstructed by each algorithm over a range of compression levels. Specifically, we measure overall error as the mean of errors between each pair of frames. The error for a given pair of frames is the average Euclidean distance between corresponding pairs of joints, in centimetres. Intuitively, this error value represents the average distance between all pairs of joints before and after the applying the algorithm. Compression is measured in terms of the ratio between the total amount of data required to store the original mocap sequence and after compression: for both our algorithm and Maya's "Simplify Curves" it consists of both the keypoints and also the tangent points of the interpolating splines across each degree of freedom, while for the PCA approach, it comprises the eigenvectors along with the weights required to reconstruct each frame from them. Note that to obtain the results for our algorithm we created animations using specified numbers of keyframes. For PCA we created animations using specified numbers of eigenvectors. For "Simplify Curves" we entered increasing values for the threshold parameter.

Although our method is not primarily intended for compression, we can see from Fig. 13 that it outperforms Maya's "Simplify Curves" algorithm and is competitive with our pose-based PCA implementation.

From these results, we conclude that Salient Poses can be of secondary benefit to games that use keyframes for compression, since it eliminates

Fig. 7 Comparison of how well keypoints selected by three methods approximate a digitized hand-drawn curve. (a)–(c) result from the greedy approximation (RDP [5]), (d)–(f) result from a finite-differences approach (Short Straw [11]), and (g)–(i) result from our algorithm (Salient Poses, adjusted to serve as an optimal keypoint selection algorithm). In the graph (j), the horizontal axis plots the number of keypoints and the vertical axis plots the approximation error of each solution (the maximum perpendicular distance between a linear interpolation of keypoints and the original curve). The vertical axis units are pixels, obtained from a plotting program used to generate the curves.



(a) Optimally selected, 5 keyframes (b) Greedily selected, 5 keyframes

(c) Optimally selected, 10 keyframes (d) Greedily selected, 10 keyframes

(e) Optimally selected, 15 keyframes (f) Greedily selected, 15 keyframes

(g) Optimally selected, 30 keyframes (h) Greedily selected, 30 keyframes

Fig. 8 Time-lapse renders comparing keyframes when selected optimally by Salient Poses (left) and greedily by an approximation algorithm [20] (right). Keyframes of each animation are drawn in red, while the inbetweens are presented in semi-transparent grey. As further keyframes become available, the optimal algorithm is able to distribute the keyframes to best optimize the objective. The distribution of keyframes given by the greedy algorithm is less useful for editing in sparse cases: in (a), three keyframes are used to summarize the downward motion after the jump, while the greedy algorithm selects no keyframes for the prior upward part; as more keyframes become available (c), the greedy algorithm is able to represent both sides of the jump well, but provides no keyframes for the step before the crouching into the jump. Both greedy and optimal solutions become more similar as further keyframes are added.

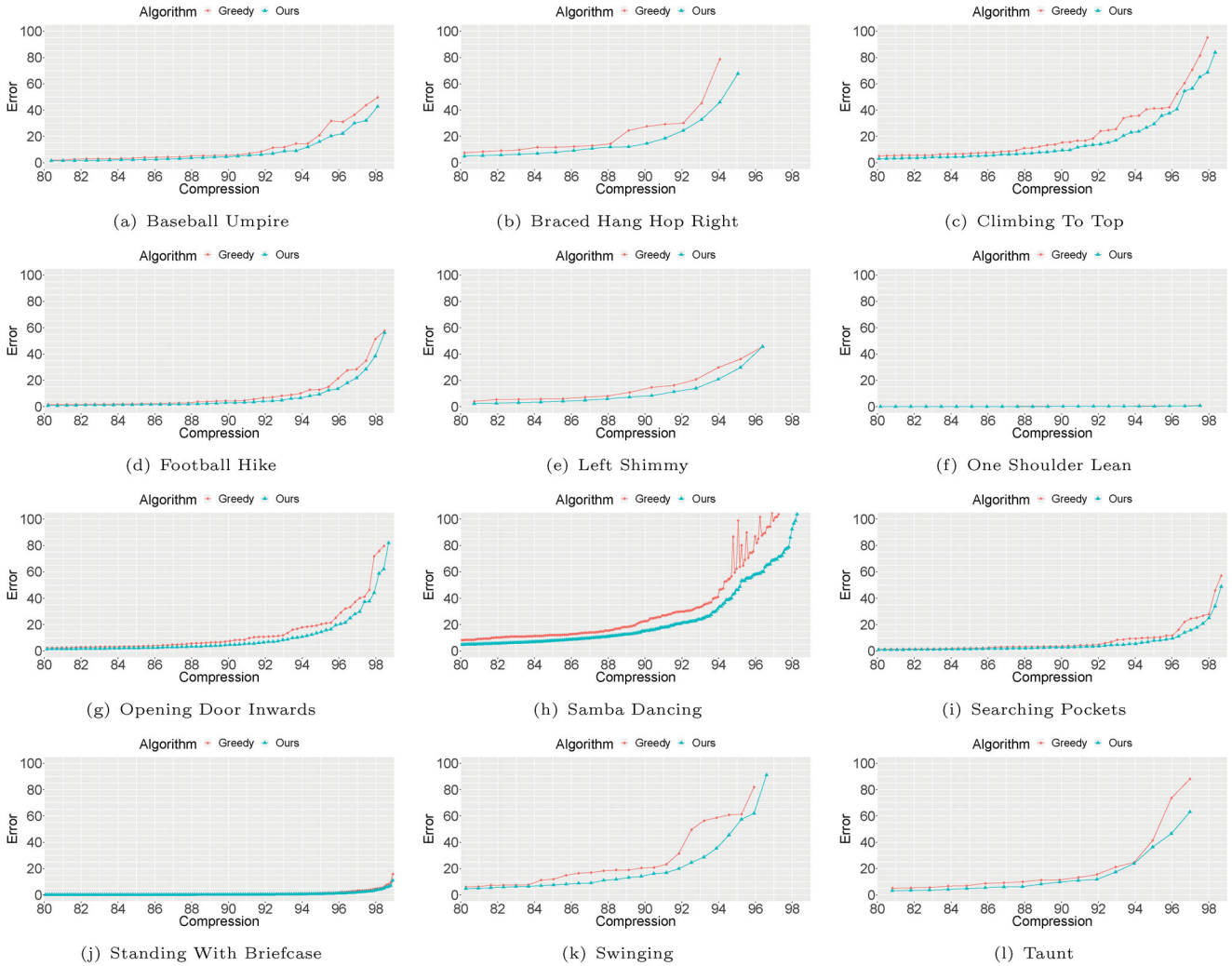
(a) Optimally selected, 14 keyframes (b) Greedily selected, 14 keyframes

Fig. 9 Another time-lapse example. The keyframes are temporally distributed more consistently in the optimal case (a), with respect to the amount of change between successive frames, than in greedy case (b). The difference can be seen most clearly by focusing on the head of the character and examining the number of inbetweens (non-keyframes, gray) between each pair of keyframes (red).

the need to maintain a separate compressed representation[①]. On the other hand, if compression is a primary concern, then a separate compression algorithm should be employed.

Additionally, note that Maya's "Simplify Curves" algorithm *does not* find keyframes, but rather finds keypoints independently for each degree of freedom. This is actually beneficial in terms of reducing approximation error, since the keypoints can be chosen independently for each DOF. However, this approach is disadvantageous for editing, as artists prefer to work with KFs where possible. In any case, the results from Maya's "Simplify Curves" algorithm are not competitive.

① In games it is generally desirable, or necessary, to compress the motion, as the expected movement sets of all characters must either be stored in limited GPU memory (along with the geometry and textures of scene objects), or the motion of every character must be streamed to the GPU. In either case the resource (memory for storage or bandwidth for streaming) is in demand and its usage should be optimized.



(a) Baseball Umpire (b) Braced Hang Hop Right (c) Climbing To Top

(d) Football Hike (e) Left Shimmy (f) One Shoulder Lean

(g) Opening Door Inwards (h) Samba Dancing (i) Searching Pockets

(j) Standing With Briefcase (k) Swinging (l) Taunt

Fig. 10 A comparison between Salient Poses and the greedy approximation algorithm [20]. The horizontal axes present compression. In this case, compression is measured as the percentage of keyframe reduction: the ratio between the number of selected keyframes and the number of keyframes in the original mocap sequence. The vertical axes depict approximation error. In this case, error is quantified as the maximum distance between a piecewise linear interpolation of the selected keyframes and each frame of the original animation. See Section 3.3 for further detail.

(a) Compression 80%–85% (b) Compression 85%–90% (c) Compression 90%–95% (d) Compression 95%–100%

Fig. 11 To summarize Fig. 10, these box and whisker plots compare the median approximation error of selections from Salient Poses and the greedy approximation algorithm [20] for all 20 examples together. To combine the errors of all examples, we normalize the error measured for a given selection by the error measured when using only the start and end frames as keyframes. We then organize the errors into groups based on the level of compression. Each box presents the one-quartile spread of the data about the median (represented by the horizontal line) for a range of compression.

Finally, note that other approaches to PCA are possible. For example an alternative approach would be to compress the animation curve for each degree of freedom, rather than the poses. Another alternative would be to preprocess the mocap into space–time windows, and then apply the compression. Such alternatives may enable higher levels of compression than our pose-based implementation.

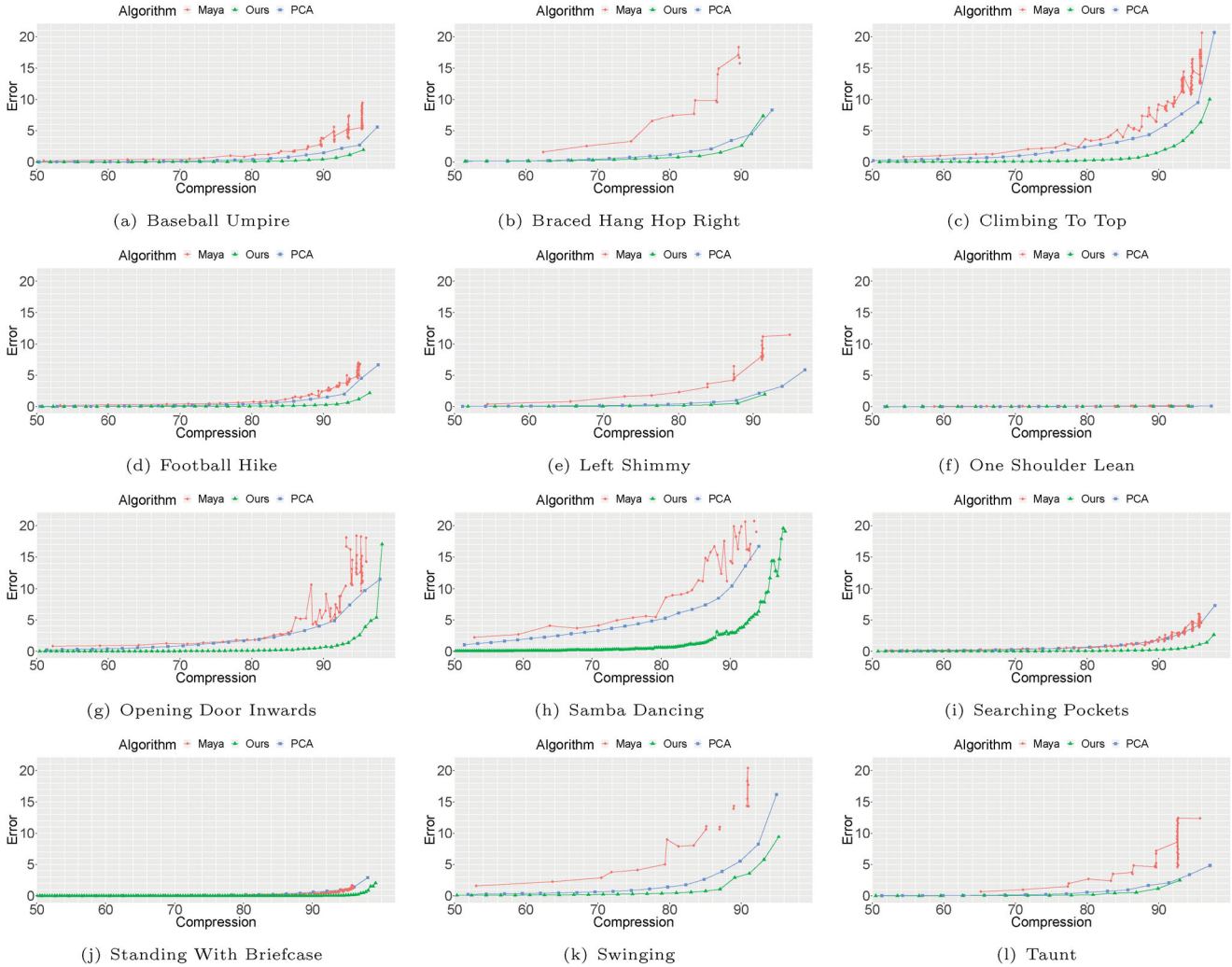


Fig. 12 Comparison between our optimal approximation, our implementation of a pose-based PCA compression algorithm, and Maya's "Simplify Curves" algorithm. The horizontal axes correspond to the total amount of compression. The vertical axes depict the distance between the original animation and that reconstructed from the compressed interpretation. See Section 4.3.

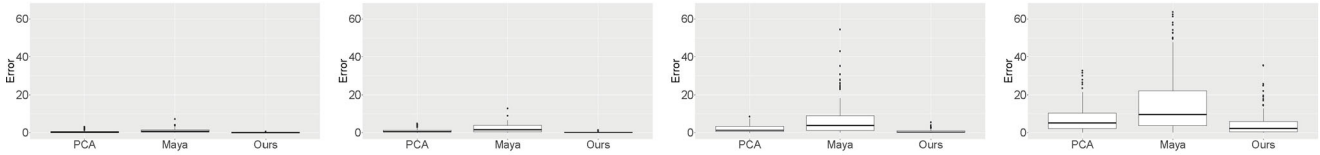


Fig. 13 To summarize Fig. 12, these box and whisker plots compare the median distance between the original animation and those reconstructed using our optimal approximation, our implementation of a pose-based PCA compression algorithm, and Maya's "Simplify Curves" algorithm for all 20 examples together. We organize the measured distances into groups based on the level of compression. Each box represents the one-quartile spread of the data about the median (represented by the horizontal line) for a range of compression.

4.4 Run time

Figure 14 shows the time required to build the all pairs table when executed on a NVIDIA GeForce GTX 1080. Computing the table for the animation of around 800 frames took under 1.5 seconds. This calculation need only be done once per animation and is a marginal overhead when performed whilst importing the motion. Smaller animations take a fraction of this time.

Figure 15 shows the time required, when executed on an Intel Core i7-8700 @ 3.20 GHz, for Salient Poses to find all solutions up to K keyframes (given

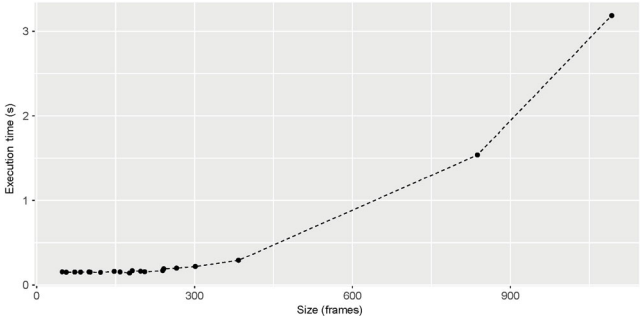


Fig. 14 Time required to build the all pairs table for each of the 20 mocap examples, when executed on a NVIDIA GeForce GTX 1080. The horizontal axis gives the size of the animation, in frames. The vertical axis gives execution time, in seconds.

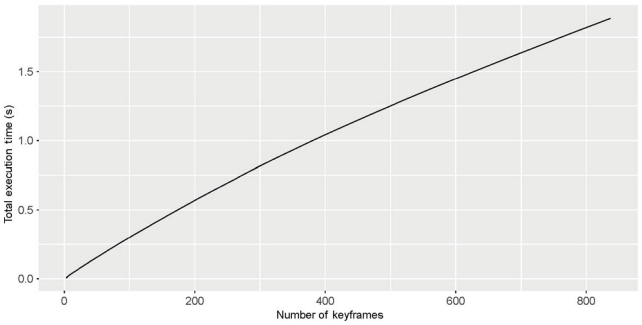


Fig. 15 Time required to compose all solutions for the "Standing With Briefcase" mocap example (837 frames), when executed on an Intel Core i7-8700 @ 3.20 GHz. The horizontal axis depicts the number of iterations (i.e., the number of keyframes selected), and the vertical axis depicts execution time in seconds. After slightly over one second, Salient Poses has computed all selections that correspond to at least 50% compression ($K \leqslant 417$ keyframes).

the all pairs table) for an animation of 837 frames. Within one second, this single execution provides all solutions with at least 50% compression ($K \leqslant N/2$). The solutions can be stored in an index table and then browsed interactively.

For comparison, an evolutionary optimization technique that minimizes a weighted combination of compression and approximation error requires execution time of 19 seconds for an animation of 316 frames, and 28 seconds for 481 frames [22]. Importantly, the time is for only *one* solution and, consequently, an artist would need to execute the algorithm repetitively to explore a range of solutions. The longer time cannot support interactive editing.

4.5 Editing

Unfortunately, analysing the editability afforded by alternative sets of keyframes is beyond the scope of the work. Specifically, editing tasks are hard to quantify due the their subjective nature—the

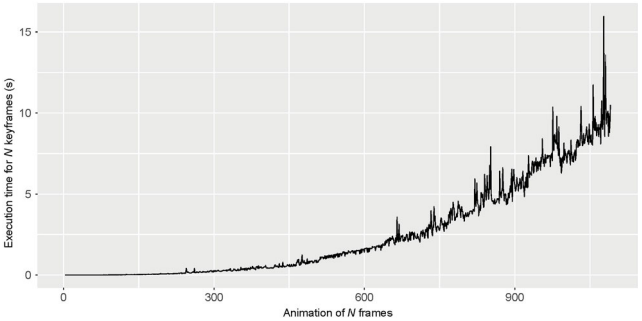


Fig. 16 Execution time versus the number of frames in the animation. The horizontal axis gives the number of frames in the animation, while the vertical axis gives the time required to find all optimal solutions using an Intel Core i7-8700 @ 3.20 GHz. To generate these results, we ran Salient Poses on slices of the "Samba Dancing" example. One slice includes the first three frames, another the first four frames, and so on, with the last slice containing all 1092 frames. Extrapolating the graph shows a cubic trend between the number of frames and execution time, which reflects the $O(N^3)$ complexity of our algorithm described in Section 3.2.2.

editor must make creative decisions with respect to both pose and timing. Furthermore, as described in Section 3.1, artists differ in their preferred ways to work with keyframes. Consequently, further research that quantifies both editing tasks and keyframe techniques is required before it is possible to conduct a formal analysis of *keyframe editability*. Instead, for this research, we make the simplifying assumption that sparsity corresponds to editability: a motion with fewer keyframes can be edited with fewer changes.

As the previous results have demonstrated that Salient Poses provides the best possible trade-off between sparse keyframe representation and approximation error, Salient Poses already meets the goal of providing sparse animations for editing that faithfully recreate the original motion capture.

While formal analysis is beyond our scope, we provide a few examples to demonstrate that the sets of keyframes provided by Salient Poses successfully enable editing. Figures 1, 17, and 18 present three examples of mocap animations before and after editing. In each case, the artist first applied the Salient Poses algorithm and then identified a particular set of keyframes suitable for their editing task. Salient Poses then constructed a new keyframe animation, which the artist edited using only the standard keyframing utilities provided by Autodesk's Maya. Please refer to the video in the ESM for the complete examples.

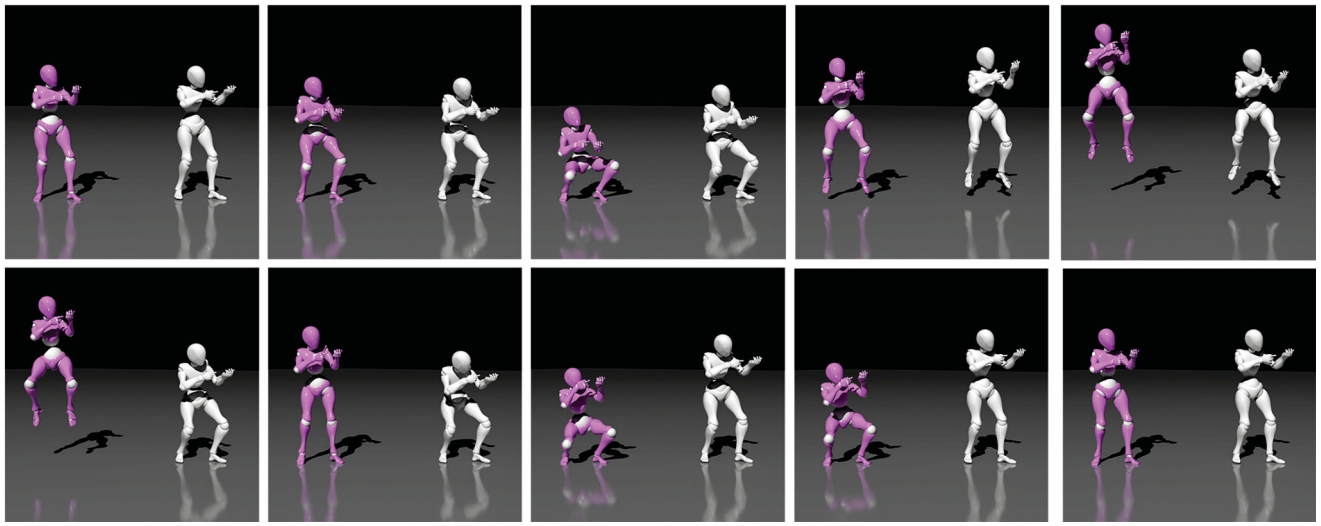


Fig. 17 Uniformly spaced frames extracted from a "Jumping" motion from Adobe's Mixamo [48], before (right character in each panel) and after editing (left character) using our technique. The edits increase the height of the jump and deepen the crouching both before and after the jump. Also see the video in the ESM.

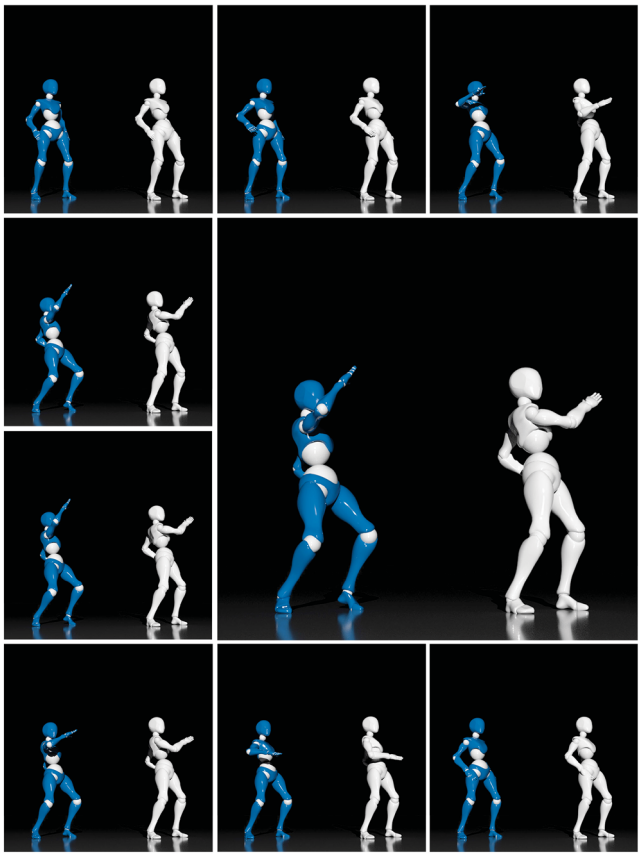


Fig. 18 Uniformly spaced frames extracted from a "Blocking" motion from Adobe's Mixamo [48], before (right character in each panel) and after editing (left character) using our technique. The edits reposition the location of the right arm and also adjust the character's centre of gravity. Also see the video in the ESM.

5 Discussion and conclusions

The representations and approaches preferred by artists can be characterized by a combination of semantically meaningful parameters, high-level control, and an aesthetic appeal that may relate to simplicity and predictability[①]. The keyframe representation of motion is one such representation that has stood the test of time and continues to be used by artists in both 2D and 3D media and in multiple communities (animation, visual effects movies, and games).

In this paper, we have introduced a solution to the important problem of editing motion capture, by converting mocap into a keyframe representation that supports editing using traditional tools and approaches. Our algorithm is designed to satisfy artists' common preference for keypoints aligned to the same frames, i.e., keyframes. Importantly, the algorithm produces a range of solutions with differing numbers of keyframes, allowing the artist to intuitively and interactively browse the solutions and pick one that offers the desired trade-off between detail and control. Finally, the algorithm is optimal, meaning that each solution features keyframes distributed to best summarize the motion (in so far as the chosen error measure describes the notion of "best").

① For example, meshes that have a desirable distribution of polygons are referred to as having "edgeflow".

In terms of evaluation, our optimal solution outperforms competing greedy algorithms by definition. It also considerably outperforms a leading commercial tool, even with the handicap that the commercial tool produces a similar number of keypoints without grouping them into keyframes.

We have shown our tool to artists at three internationally known companies from the video game, cartoon animation, and visual effects industries. Two animators described a prototype of our algorithm as "a life changing tool", and "a thing of beauty". Another artist commented that exploring the keyframes for different levels of compression makes it "a lot easier to visualize the movement as a series of poses", while another said that it would be "ideal for stylizing motion capture for use in action-focused games". Overall, the artists felt that our algorithm reduces the time and cost required for motion editing and stylization.

The algorithm is also currently being tested for adoption at a major entertainment production company and also by an independent studio.

5.1 Limitations

Our algorithm has two evident limitations.

5.1.1 Asymptotic complexity

Motion pictures are composed of "shots" that typically last between 4 and 6 seconds [49] or often less in action-heavy visual effects sequences. While our method gives interactive performance for a mocap of this typical length, games and even some movies can require motion clips that are longer than five seconds. Due to the cubic computational cost as well as the quadratic memory needed for the all pairs table, very long shots must be split before processing. This can be done automatically by detecting keyframes using a local heuristic [50], or otherwise manually by an artist by selecting a small number of keyframes to serve as split points.

5.1.2 Sweet spots

While any choice of K keyframes resulting from our algorithm is optimal by definition, our experimental results show that approximation error does not always decay smoothly as more keyframes are added. On the contrary, there are "error cliffs" where the addition of a single KF produces a disproportionate reduction in error, followed sometimes by "plateaus" where a few additional keyframes provide little improvement (examine the shape of the curves in Fig. 10, c.f., 95%–99% compression in Fig. 10(g)). While we were initially surprised by the appearance of these cliffs, they can be explained intuitively (see Fig. 19).

In terms of quality, the best sets of keyframes relative to the greedy RDP algorithm occurs at the base of a cliff, i.e., at an m such that $|E^m_{1,N} - E^{m+1}_{1,N}|$ is relatively large. While such cliff locations could

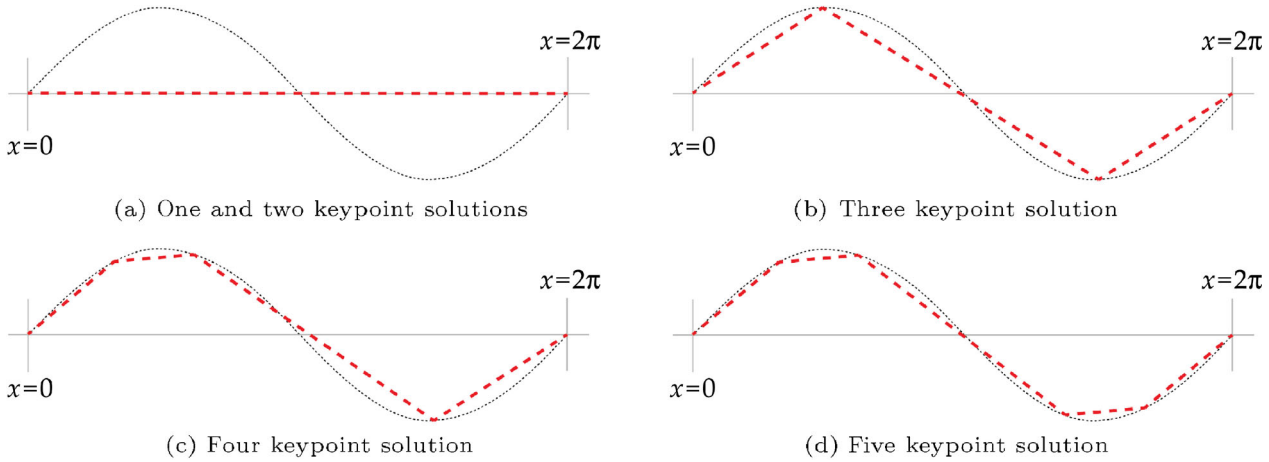


Fig. 19 The error between a piecewise linear interpolation of keyframes and the original motion may decrease irregularly as more keyframes are used. This lack of proportionate error reduction can be explained intuitively using this example of a sinusoid. Take the canonical sinusoid $\sin(x)$ through the interval $[0, 2\pi]$. First, note that optimal solution for both the one keypoint and two keypoint linear approximation features the same flat line (a). The optimal three keypoint solution (b) captures both extrema and, consequently, the error reduces significantly. In contrast to the optimal three keypoint solution, the optimal four (c) and five (d) keypoint solutions reduce error only slightly. From this example, we can see that error reduces irregularly with increasing numbers of keypoints. Notably, the *error cliff* phenomenon (see text) occurs with the three-piece solution. As articulated motion tends to follow smooth trajectories, at least at a coarse scale, it is unsurprising that a similar error cliff pattern is seen with piecewise linear approximations. Intuitively, an error cliff occurs as enough keyframes become available to capture all extrema for a given level of detail.

perhaps be detected automatically, in the absence of further analysis we prefer to let the artist interactively browse the solutions for various K and select the best solution according to their judgement.

5.2 Future work

While we feel that our algorithm provides a good general solution for converting mocap into an editable representation, there remains much work to be done to more fully formalize and support the motion editor's craft.

5.2.1 Intent-directed editing

The best set of keyframes depends on the creative intent. For example, consider a shot in which the actor stands nearly motionless for a few seconds and then begins to run. The director may wish to emphasize the dramatic quality of springing into motion, or, they may wish to focus on the subtle movements (perhaps reflecting nervousness, for example) while the person is standing. Accommodating such "intent" in a computer-assisted editing framework remains an open problem.

5.2.2 Joint-weighted editing

The importance of accurately capturing the motions of different body parts can vary depending on the task. For example, the position of the hand is important when interacting with a door, or a gun, yet the hand position during a walking motion may be more forgiving. Configuring the importance of each joint can be incorporated by weighting how much each joint, or perhaps even each degree of freedom, contributes to the approximation error. Initially, each joint has a default weight of one, which the animator can override for particular joints with a time-varying animation curve.

5.2.3 Animation concepts

A challenging open problem is to more fully express animation (as opposed to motion editing) concepts such as "leading part" [51] in a keyframe-based editing framework. We speculate that this challenging problem will, at least, require re-thinking the holistic one-fits-all approximation error we have employed in this work.

5.2.4 Spline fitting

Another topic for future work is that of curve fitting. While curve fitting is often considered to be a solved problem, many published fitting algorithms assume that parametric continuity is desired across the entire curve. In the context of our problem continuity is not always desirable; for example, the motion of a fist hitting a wall should introduce a derivative discontinuity. Curve fitting algorithms also tend to rely on the ability to add keypoints when faced with sections of the curve that are hard to approximate with a spline. In our case, additional keypoints should not be added, since the desired keypoints are prescribed by the keyframes. In our experience, in which we applied a variation of Schneider's algorithm (see Section 3.2.3), we were able to obtain clear improvements with manual adjustments to the curve tangents.

Acknowledgements

Many researchers and artists have contributed important insights to this research. The authors would like to give special thanks to Ayumi Kimura and other staff of OLM Digital, to Johan Andersson, Ida Winterhaven, and Binh Le of SEED, Electronic Arts, and also to Ian Loh and other staff of Victoria University of Wellington's Computational Media Innovation Centre and Virtual Worlds Lab. The authors would also like to thank the Moveshelf team for supporting the web-based presentation of our results.

Electronic Supplementary Material Supplementary material is available in the online version of this article at `https://doi.org/10.1007/s41095-019-0138-z`.

References

[1] Lam, D. Personal communication. Electronic Arts, 2017.

[2] Shelton, D. Personal communication. Electronic Arts, 2017.

[3] White, T. *Animation from Pencils to Pixels: Classical Techniques for Digital Animators*. Burlington: Focal Press, 2006.

[4] Roy, K. *How to Cheat in Maya 2014: Tools and Techniques for Character Animation*. Burlington: Focal Press, 2013.

[5] Ramer, U. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing* Vol. 1, No. 3, 244–256, 1972.

[6] Bertsekas, D. P. *Network Optimization: Continuous and Discrete Models*. Belmont: Athena Scientific, 1998.

[7] The U.S. game industry has 2,457 companies supporting 220,000 jobs. 2018. Available at https://venturebeat.com/2017/02/14/the-u-s-game-industry-has-2457-companies-supporting-220000-jobs.

[8] The games industry in numbers. 2018. Available at https://https://ukie.org.uk/research.

[9] Wang, X.; Chen, Q.; Wang, W. 3D human motion editing and synthesis: A survey. *Computational and Mathematical Methods in Medicine* Vol. 2014, Article ID 104535, 2014.

[10] Miura, T.; Kaiga, T.; Shibata, T.; Katsura, H.; Tajima, K.; Tamamoto, H. A hybrid approach to keyframe extraction from motion capture data using curve simplification and principal component analysis. *IEEJ Transactions on Electrical and Electronic Engineering* Vol. 9, No. 6, 697–699, 2014.

[11] Wolin, A.; Eoff, B.; Hammond, T. ShortStraw: A simple and effective corner finder for polylines. In: Proceedings of the 5th Eurographics Conference on Sketch-based Interfaces and Modeling, 33–40, 2008.

[12] So, C. K. F.; Baciu, G. Entropy-based motion extraction for motion capture animation. *Computer Animation and Virtual Worlds* Vol. 16, Nos. 3–4, 225–235, 2005.

[13] Cuntoor, N. P.; Chellappa, R. Key frame-based activity representation using antieigenvalues. In: *Computer Vision – ACCV 2006. Lecture Notes in Computer Science, Vol. 3852*. Narayanan, P. J.; Nayar, S. K.; Shum, H. Y. Eds. Springer Berlin Heidelberg, 499–508, 2006.

[14] Wei, X. P.; Liu, R.; Zhang, Q. Key-frame extraction of human motion capture data based on least-square distance curve. *Journal of Convergence Information Technology* Vol. 7, 11–19, 2012.

[15] Bulut, E.; Capin, T. Keyframe extraction from motion capture data by curve saliency. Available at http://www.people.vcu.edu/∼ebulut/casa.pdf.

[16] Halit, C.; Capin, T. Multiscale motion saliency for keyframe extraction from motion capture sequences. *Computer Animation and Virtual Worlds* Vol. 22, No. 1, 3–14, 2011.

[17] Marr, D. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. New York: Henry Holt and Co., Inc., 1982.

[18] Douglas, D.; Peucker, T. K. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature.*Cartographica: The International Journal for Geographic Information and Geovisualization* Vol. 10, No. 2, 112–122, 1973.

[19] Lowe, D. G. Three-dimensional object recognition from single two-dimensional images. *Artificial Intelligence* Vol. 31, No. 3, 355–395, 1987.

[20] Lim, I. S.; Thalmann, D. Key-posture extraction out of human motion data. In: Proceedings of the 23rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Vol. 2, 1167–1169, 2001.

[21] Seol, Y.; Seo, J.; Kim, P. H.; Lewis, J. P.; Noh, J. Artist friendly facial animation retargeting. *ACM Transactions on Graphics* Vol. 30, No. 6, Article No. 162, 2011.

[22] Liu, X.-M.; Hao, A.-M.; Zhao, D. Optimization-based key frame extraction for motion capture animation. *The Visual Computer* Vol. 29, No. 1, 85–95, 2013.

[23] Zhang, Q.; Zhang, S.; Zhou, D. Keyframe extraction from human motion capture data based on a multiple population genetic algorithm. *Symmetry* Vol. 6, No. 4, 926–937, 2014.

[24] Zhang, Y.; Cao, J. 3D human motion key-frames extraction based on asynchronous learningfactor PSO. In: Proceedings of the 5th International Conference on Instrumentation and Measurement, Computer, Communication and Control, 1617–1620, 2015.

[25] Chang, X.; Yi, P.; Zhang, Q. Key frames extraction from human motion capture data based on hybrid particle swarm optimization algorithm. In: *Recent Developments in Intelligent Information and Database Systems. Studies in Computational Intelligence, Vol. 642*. Król, D.; Madeyski, L.; Nguyen, N. Eds. Springer Cham, 335–342, 2016.

[26] Bellman, R.; Kashef, B.; Vasudevan, R. Splines via dynamic programming. *Journal of Mathematical Analysis and Applications* Vol. 38, No. 2, 471–479, 1972.

[27] Lewis, J. P.; Anjyo, K. Identifying salient points. In: Proceedings of the ACM SIGGRAPH ASIA 2009 Sketches, Article No. 41, 2009.

[28] Witkin, A.; Popovic, Z. Motion warping. In: Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques, 105–108, 1995.

[29] Lee, J.; Shin, S. Y. A hierarchical approach to interactive motion editing for human-like figures. In: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, 39–48, 1999.

[30] Witkin, A.; Kass, M. Spacetime constraints. *ACM SIGGRAPH Computer Graphics* Vol. 22, No. 4, 159–168, 1988.

[31] Gleicher, M. Animation from observation: Motion capture and motion editing. *ACM SIGGRAPH Computer Graphics* Vol. 33, No. 4, 51–54, 1999.

[32] Guay, M.; Cani, M.-P.; Ronfard, R. The line of action: An intuitive interface for expressive character posing. *ACM Transactions on Graphics* Vol. 32, No. 6, Article No. 205, 2013.

[33] Choi, B.; i Ribera, R. B.; Lewis, J. P.; Seol, Y.; Hong, S.; Eom, H.; Jung, S.; Noh, J. SketchiMo: Sketch-based motion editing for articulated characters. *ACM Transactions on Graphics* Vol. 35, No. 4, Article No. 146, 2016.

[34] Kovar, L.; Gleicher, M.; Pighin, F. Motion graphs. *ACM Transactions on Graphics* Vol. 21, No. 3, 473–482, 2002.

[35] Casas, D.; Tejera, M.; Guillemaut, J.-Y.; Hilton, A. 4D parametric motion graphs for interactive animation. In: Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, 103–110, 2012.

[36] Huang, P.; Tejera, M.; Collomosse, J.; Hilton, A. Hybrid skeletal-surface motion graphs for character animation from 4D performance capture.*ACM Transactions on Graphics* Vol. 34, No. 2, Article No. 17, 2015.

[37] Park, M. J.; Shin, S. Y. Example-based motion cloning. *Computer Animation and Virtual Worlds* Vol. 15, Nos. 3–4, 245–257, 2004.

[38] Peng, X. B.; Abbeel, P.; Levine, S.; van de Panne, M. DeepMimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics* Vol. 37, No.4, Article No. 143, 2018.

[39] Assa, J.; Caspi, Y.; Cohen-Or, D. Action synopsis: Pose selection and illustration. *ACM Transactions on Graphics* Vol. 24, No. 3, 667–676, 2005.

[40] Yasuda, H.; Kaihara, R.; Saito, S.; Nakajima, M. Motion belts: Visualization of human motion data on a timeline. *IEICE Transactions on Information and Systems* Vol. E91-D, No. 4, 1159–1167, 2008.

[41] Hu, Y.; Wu, S.; Xia, S.; Fu, J.; Chen, W. Motion track: Visualizing variations of human motion data. In: Proceedings of the IEEE Pacific Visualization Symposium, 153–160, 2010.

[42] Arikan, O. Compression of motion capture databases. *ACM Transactions on Graphics* Vol. 25, No. 3, 890–897, 2006.

[43] Huang, K.-S.; Chang, C.-F.; Hsu, Y.-Y.; Yang, S.-N. Key probe: A technique for animation keyframe extraction. *The Visual Computer* Vol. 21, No. 8, 532–541, 2005.

[44] Tournier, M.; Wu, X.; Courty, N.; Arnaud, E.; Reveret, L. Motion compression using principal geodesics analysis. *Computer Graphics Forum* Vol. 28, No. 2, 355–364, 2009.

[45] Xia, G.; Sun, H.; Niu, X.; Zhang, G.; Feng, L. Keyframe extraction for human motion capture data based on joint kernel sparse representation. *IEEE Transactions on Industrial Electronics* Vol. 64, No. 2, 1589–1599, 2016.

[46] Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik* Vol. 1, No. 1, 269–271, 1959.

[47] Schneider, P. J. An algorithm for automatically fitting digitized curves. In: *Graphics Gems*. San Diego: Academic Press Professional, Inc., 612–626, 1990.

[48] Adobe Mixamo. 2018. Available at https://www.mixamo.com.

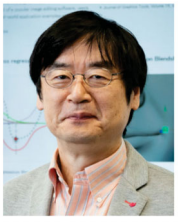
[49] Kaufman, J. C.; Simonton, D. K. *The Social Science of Cinema*. Oxford University Press, 2013.

[50] Miura, T.; Kaiga, T.; Katsura, H.; Tajima, K.; Shibata, T.; Tamamoto, H. Adaptive keypose extraction from motion capture data. *Journal of Information Processing* Vol. 22, No. 1, 67–75, 2014.

[51] Lasseter, J. Principles of traditional animation applied to 3D computer animation. *ACM SIGGRAPH Computer Graphics* Vol. 21, No. 4, 35–44, 1987.

Richard Roberts researches into artist-directed tools for animation and visual effects work. Roberts is a currently a research fellow, developing a facial mocap and animation pipeline for the production of a VR narrative experience. He has worked briefly in industry, receiving credit in *the Adventures of Tintin*, and also has a background developing virtual machines for high-level programming languages.

J. P. Lewis is a numerical programmer and researcher. He is principal research scientist at SEED, the new research lab of Electronic Arts, and is an adjunct associate professor in the machine learning group of Victoria University of Wellington. His interests include computer vision and machine learning applications in entertainment. He has received credits on a few movies including *Avatar* and *the Matrix* sequels, and several of his algorithms have been adopted in commercial software including Maya and MATLAB.



Ken Anjyo set up and headed the research and development division of OLM Digital, the digital production company in Tokyo famous for the *Pokémon* movies and other 3D animated feature films. He became the company's CTO and is now its executive R&D adviser. He is a board member of VFX-JAPAN, the Japanese association of domestic digital production companies, and a member of the Visual Effects Society. Since 2018, he has also been working as the director of the Computational Media Innovation Center at Victoria University of Wellington.

Jaewoo Seo is a director of R&D at Pinscreen. His research interests include facial animation, motion capture, and GPU programming. Before joining Pinscreen, he was in the visual effects industry as an R&D engineer at ILM, Weta Digital, and OLM Digital. He received his Ph.D. and M.S. degrees in culture technology from KAIST and B.S. degree in digital media and in computer and information engineering from Ajou University.

Yeongho Seol is interested in developing fundamental computer graphics and vision technologies and making them useful in real-world VFX and animation production. He is experienced in a range of motion capture related technologies and works as senior motion capture developer in Weta digital.