

Converting Motion Capture into Editable Keyframe Animation

Fast, Optimal, and Generic Keyframe
Selection

by

Richard Roberts

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Computer Science and Design.
Victoria University of Wellington
2018

Abstract

Motion capture is attractive to visual effects studios because it offers a fast and automatic way to create animation directly from actors' movements. Despite extensive research efforts toward motion capture processing and motion editing, animations created using motion capture are notoriously difficult to edit. We investigate this problem and develop a technique to reverse engineer editable keyframe animation from motion capture.

Our technique for converting motion capture into editable animation is to select keyframes from the motion capture that correspond to those an animator might have used to create the motion from scratch. As the first contribution presented by this thesis, we survey both traditional and contemporary animation practice to define the types of keyframes created by animators following conventional animation practices. As the second contribution, we develop a new keyframe selection algorithm that uses a generic objective function; using different implementations, we can define different criteria to which keyframes are selected. After presenting the algorithm, we return to problem of converting motion capture into editable animation and design three implementations of the objective function that can be used together to select animator-like keyframes. Finally, as a minor contribution to conclude the thesis, we present a simple interpolation algorithm that can be used to construct a new animation from only the selected keyframes.

In contrast to previous research in the topic of keyframe selection, our technique is novel in that we have designed it to provide selections of keyframes that are similar in structure to those used by animators following conventional practices. Consequently, both animators and motion editors can adjust the resulting animation in much the same way as their own, manually created, content. Furthermore, our technique offers an optimal guarantee paired with fast performance for practical editing situations, which has not yet been achieved in previous research. In conclusion, the contributions of this thesis advance the state of the art in the topic by introducing the first fast, optimal, and generic keyframe selection algorithm. Ultimately, our technique is not only well suited to the problem of recovering editable animation from motion capture, but can also be used to select keyframes for other purposes – such as compression or pattern identification – provided that an appropriate implementation of the objective function can be imagined and employed.

Acknowledgements

When recently returning to studies after a summer internship I was excited to practice animation. I turned to motion capture with the hope that I could use it to create exemplary motion, which I could later edit, to help myself learn more about the craft of animation. To my surprise, there was little in the way of accessible software that provided support for editing motion capture.

I transitioned into computer science with hopes to create my own tools for editing motion capture, but after two years as a doctorate student I was humbled to find that developing them was going to be particularly difficult – a challenge that required both creative and technical skills to meet. In search of better ideas I travelled to Tokyo to learn from a local visual effects studio. It was through discussion with their staff that I gained key insights; prerequisites to the techniques that I present in this document. After returning to New Zealand, to complete my final year as a doctorate student, I was finally successful in my efforts to realize a technique for converting hard-to-edit motion capture into easier-to-edit animation.

I would like to thank the academic staff of the university, without their help I would have never been able to progress through the design and engineering problems that I was faced with. And, just as importantly, I am forever in the debt of the university's exceptional supporting staff that have ensured my physical, mental, and financial survival. My thanks continue to my fellow students, who have revived my motivation countless times; and to industry, to WETA DIGITAL whose staff raised me as a programmer and to OLM DIGITAL who provided me with a life-changing exchange in Tokyo. To my supervisors, to whom I inflicted pain and received only knowledge in return. And finally to my wife, my daughter, and my family who have made this endeavour not only possible but worth-while.

Contents

1	Introduction	1
1.1	Motion Editing	4
1.2	Related Work and Technical Challenges	11
1.3	Research Contributions and Outline	13
1.4	Thesis Scope	14
2	Related Work	17
2.1	Animation Practice	18
2.1.1	Traditional Animation	18
2.1.2	Computer Animation	21
2.1.3	Terminology	26
2.2	Keyframe Selection	26
2.2.1	Detection	28
2.2.2	Approximation	31
2.2.3	Optimization	34
2.2.4	Factorization	37
2.2.5	Discussion	39
3	Optimal Keyframe Selection	41
3.1	Salient Poses	43
3.1.1	Representing the Animation as a Graph	43
3.1.2	Partial Graphs provide Optimal Substructure	45
3.1.3	Optimal Selection through Partial Graphs	46
3.1.4	Number of Computations Required	48
3.2	Results	51
3.2.1	Comparison to Related Techniues	53
3.2.2	Execution Times for Keyframe Selection	62
3.3	Summary	65

4	Keyframe Selection for Blocked Animation	67
4.1	Keyframes for Editable Animation	68
4.2	Complications for Keyframe Selection	68
4.3	Three Layers of Keyframe Selection	73
4.3.1	Segmenting the Motion	75
4.3.2	Selecting Keyposes	76
4.3.3	Selecting Breakdowns	76
4.3.4	Composing the Selections	77
4.4	Results	78
4.4.1	Keyframes from Keyframe Animation	78
4.4.2	Keyframes from Motion Capture	89
4.5	Summary	94
5	Reconstructing the Animation	97
5.1	Optimization-Based Spline Fitting	101
5.1.1	Curve Fitting as a Minimization Problem	101
5.1.2	Design of the Objective Function	101
5.1.3	Accuracy Term	102
5.1.4	Angle Term	106
5.1.5	Time complexity	106
5.2	Results	107
5.3	Summary	115
6	Conclusion	117
6.1	Summary of Contributions	119
6.2	Summary of Limitations	120
6.3	Future Work	121
	Appendices	123
A	Simple Curve Approximation	125

Chapter 1

Introduction

An animator creates an animation by plotting keyframes. To create a keyframe, the animator first chooses a point in time and then adjusts the character so that they strike a particular pose. To create a fluid animation from the keyframes, the animation software interpolates through the animator’s keyframes to create inbetween poses. The animation software creates the animation in real time and, as such, the animator can interactively configure any of the keyframes to edit the animation. In this thesis, we will use the term “keyframe animation” to describe the type of animation created by an animator following conventional computer animation practice.

Keyframes are a useful abstraction for animators. Most importantly, changes to the pose of a particular keyframe affect not only that keyframe itself but also all of the inbetweens (the poses created by the animation software) nearby. Since keyframes define the animation, an animator can edit the motion via the keyframes. Consequently, as more keyframes are added to the animation, the animator needs to perform more adjustments to change the animation and conversely, fewer adjustments are needed with fewer keyframe. In other words, animations tend to be easier to edit when fewer keyframes are used.

Since animators can guide the animation software to realize a fluid animation without needing to key in every pose that the character should strike, keyframe animation has long been a standard practice for content generation in animated feature films. Furthermore, keyframe animation is also attractive to visual effect studios since animators are able to preserve the ability to edit their motions throughout the animation process (at least until entering the final detailing stage, described in more detail in Section 2.1).

Despite the advantages present in keyframe animation, it is an expensive approach to generating animations. A professional from the industry reports that a conservative estimate of the time required to complete the animation for a contemporary Hollywood animated movie may be as high as two years, with a staff of 100 or more animators [48].

In contrast, motion capture technology offers an automatic solution for mapping the movements of real life actors onto virtual characters. With the ability to automatically generate new animation, it is of little surprise that motion capture technology has been well received by visual effects studios.

One of the first lead characters to be animated with motion capture was Gollum, a hobbit character in Peter Jackson’s *THE LORD OF THE RINGS* trilogy [39]. To animate Gollum, the visual effects studio Weta Digital recorded the movements of Andy Serkis (a real-life actor) performing as Gollum on a motion capture stage. The application of motion capture in films has continued to expand and today the animation used in many of the recent blockbusters films have been produced with extensive use of the technology. Weta Digital employed motion capture to animate both the body and face of the protagonist monster in Peter Jackson’s *KING KONG* [40], many of the mythical creatures in Peter’s Jackson’s *THE HOBBIT* trilogy [41], and also the native indigenous characters in James Cameron’s *AVATAR* [13]. Motion capture is also being used extensively for the production of animation at other large visual effects studios, such as Industrial Light and Magic and Double Negative, and also for the production of animation in video games: Konami employed motion capture animation in Hideo Kojima’s *METAL GEAR SOLID* video-game series [45] and Square Enix in Hajime Tabata’s *FINAL FANTASY XV* [69].

Despite the wide-spread success of motion capture, an under-reported problem is that the animation resulting from the use of motion capture is notoriously difficult to edit. In order to ensure that the details of an actors’ performances are preserved accurately in the recorded data, motion capture tools record many snapshots of an actor’s poses per second (typically 120 snapshots are used for standard motions). Due to the high number of snapshots recorded by these tools, the data that results from motion capture is both dense and unstructured, which makes the animation difficult to edit [71, p. 3]. The problem is perhaps best summarized by Gleicher [25, p. 53]:

The data is most certainly inconvenient for editing. Motion capture systems typically provide a pose for every sample or frame of the motion, not just at important instants in time. This means that a lot of data must be changed to make an edit. There is nothing but the data to describe the properties of the motion. There is little indication in the data to show what the important properties of the motion are, and what should be changed to effect the motion, nor is there an animator familiar with the ‘why’ of the motion.

The large number of snapshots, henceforth referred to as keyframes, present in

the motion capture animation means that an editor wishing to adjust the motion must make many adjustments, since every pose in the motion capture animation is equivalent to a keyframe and as such adjusting a keyframe impacts only that keyframe and not the surrounding keyframes. In contrast to motion capture animation, keyframe animation is purposefully constructed with fewer keyframes so that adjustments to any of those keyframes is impactful (adjusting a keyframe has impact over all the neighbouring inbetweens). Thus, an advantage of keyframe animation over motion capture is that the resulting animation can be edited with significantly fewer adjustments.

Motion capture often needs to be edited to remove artefacts in the motion that result from errors in the recording process or the following process of mapping the recorded data on to a virtual character. Furthermore, editing of the motion may also be required to meet changes requested by a director, perhaps to exaggerate the motion beyond reality. As visual effects studios continue to rely on motion capture more heavily, we hypothesize that the ability to convert motion capture into an editable form will become essential.

In this thesis we address the problem of motion capture animation being difficult to edit and propose to solve this problem by reducing the number of keyframes present in the motion capture animation. Specifically, we look to develop a keyframe selection technique to find the important poses in the motion and, by doing so, effectively reverse engineer editable keyframe animation from motion capture. In particular, we intend for our technique to identify the keyframes that an animator may have otherwise used to animate the motion themselves. Once we have obtained a reasonable selection of keyframes we construct an entirely new animation by interpolating only those keyframes in a way that preserves the detail of the original motion.

Despite the simplicity of our solution, three primary technical challenges are involved:

- **Accuracy against compression.** Removing a higher number of keyframes from the motion capture animation increases the impact of adjustments to those keyframes since the number of inbetweens increases as keyframes are removed. However, as a higher number of keyframes are removed it also becomes increasingly difficult to interpolate the remaining keyframes in a way that preserves the details of the motion. Thus, the first technical challenge is to develop an algorithm that can select keyframes with an appropriate balance between accuracy (the potential for the selected keyframes to reconstruct the animation) and the amount of compression (the number of keyframes removed).
- **Best distribution.** As the keyframes that remain after reduction serve as the

points of control over the animation, it is also important that the selected keyframes are distributed in a way that provides the best control over the details of the animation. For example, an animation that features many keyframes before a character begins a jump and only few keyframes after the character later contacts the ground would not feature a useful distribution. While the keyframes before the jump enable accuracy, adjustments to the keyframes in that earlier part of the motion are not impactful. Conversely, the keyframes after the jump enable impactful changes but lack accuracy. Instead, distributing the keyframes through the motion based on the amount of change between poses leads to a more editable animation. Thus, the second technical challenge is to develop an algorithm that distributes keyframes well throughout the motion.

- **Accurate reconstruction.** Finally, we require an algorithm to interpolate the selected keyframes in a way that leads to a new animation that is a faithful representation of the original motion. Thus, the third technical challenge is to develop an algorithm for accurately fitting an interpolation of the selected keyframes to the original motion.

With an implementation of these contributions, visual effects studios can fully benefit from motion capture technology without sacrificing their ability to edit the resulting animations easily. From that perspective, our contributions are placed to help simplify the task of motion editing. Furthermore, converting motion capture into keyframe animation provides an interpretation of motion capture that will be familiar to animators and therefore our contributions may also help to integrate the practices of motion editing and keyframe animation in the future.

In order to provide context for these contributions, we will first provide a description of how contemporary software supports motion editing, in which we highlight why motion editing can be a difficult task (Section 1.1). We then briefly review the previous research toward keyframe selection and examine the technical challenges surrounding the topic (Section 1.2). Finally, we introduce our solution in more detail and summarize the contributions that will be presented throughout this thesis (Section 1.3).

1.1 Motion Editing

Editing motion capture by tweaking each of its poses is difficult in much the same way that adjusting an image by tweaking individual pixels would be difficult; the representation is too dense to make impactful changes and is void of any structure that assists an editor. Consequently, complications arise during production when

adjustments need to be made to motion capture. In this section we present the motivation for this thesis by outlining the tasks that motion editors must perform to prepare motion capture for use in a production. In particular, we describe the level of support for these tasks afforded by contemporary software and outline where bottlenecks can occur.

VICON is an optical motion capture technology and a popular choice among visual effects studios for recording body motion. The system records the position of retro-reflective orbs, typically referred to as markers, that are attached to a black suit worn by an actor. In the typical set up, a number of specialized cameras surround a stage upon which actors' movements may be recorded. Each camera films the stage from a different angle using a high frame rate, often 120 snapshots per second. An algorithm triangulates the markers' positions from the synchronized video feeds to provide a set of densely sampled 3D points for each of the markers. As high frame rates are used, the sampled points form curves that arc through the scene (see Figure 1.1a).

Unfortunately, raw motion capture data contains errors that need to be fixed. One of these errors is noise in the motion which results from an incorrect calculation of where the markers are in space. Noise can give the motion a jittery appearance (an example of noise in an animation curve is depicted in Figure 1.2). Removing the distortion caused by noise is often the first task undertaken by a motion editor. Contemporary animation software such as Autodesk's MOTIONBUILDER [5] provide filtering algorithms to support noise removal. These filtering algorithms are based on seminal research on noise removal [74, 9]. More recently advanced algorithms using wavelet encoding [21] and optimization methods have been proposed [22], but have not yet been integrated into MOTIONBUILDER.

Once noise has been removed from the motion capture data, the data needs to be applied to animate a virtual character. In order to map the motion capture onto a virtual character, the data needs to be solved, first, for a performance rig. Each joint of the performance rig is configured based on the recorded positions. Once animated, the performance rig's movements are retargeted onto the virtual character (see Figure 1.1c). Advanced solving and retargeting techniques have also been presented in research: techniques that resolve proportional differences between the actor and characters [24, 56], techniques that ensure the spatial relationships between the actor and their environment are preserved [33], and even techniques to map motion onto non-human characters [77, 20]. Like the recent research on noise removal, much of the recent research on solving and retargeting has not yet been integrated into tools such as MOTIONBUILDER.

Unfortunately, both the processes of solving for the performance rig and retargeting the motion onto the virtual character can produce invalid poses, especially when there

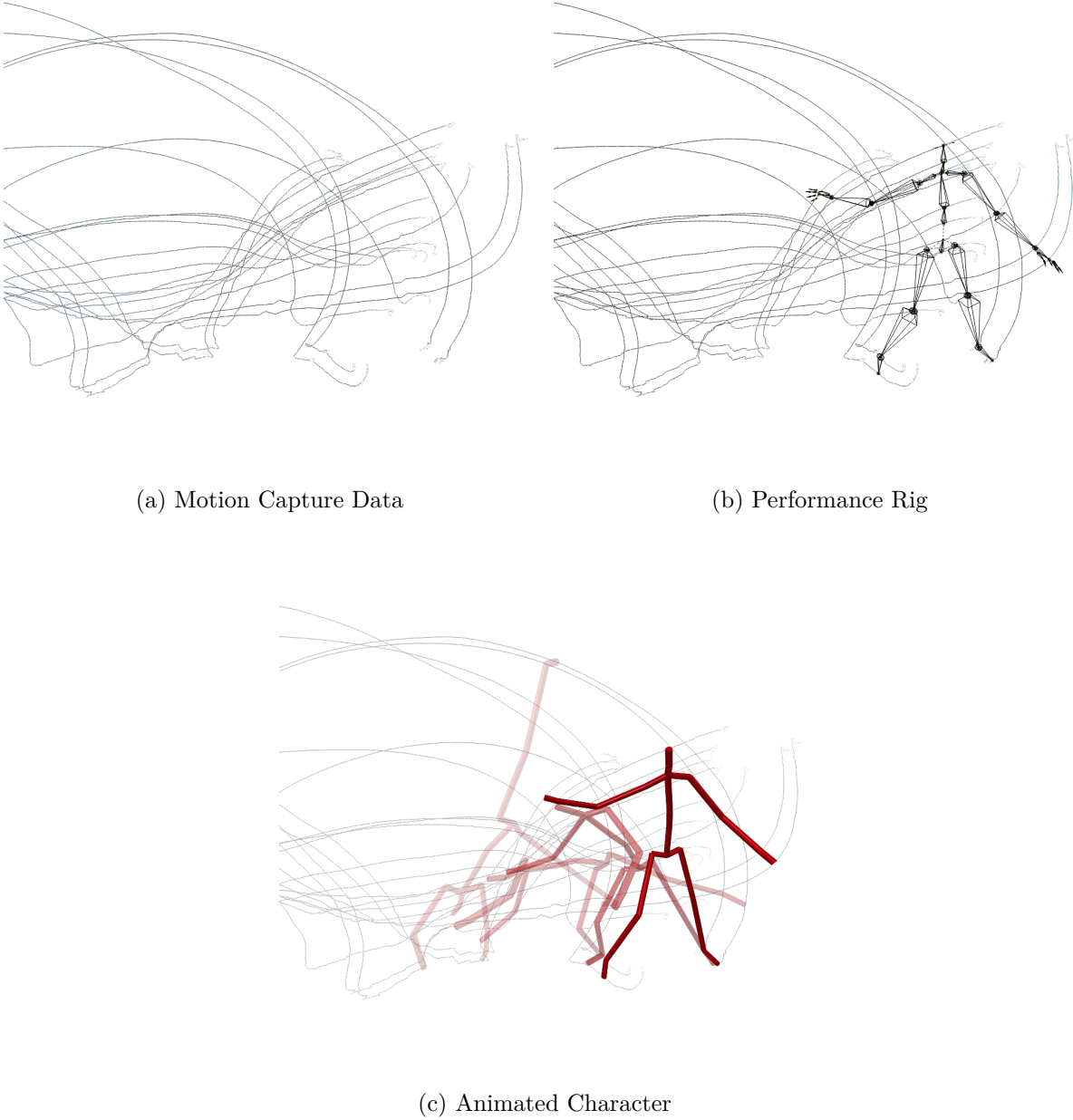


Figure 1.1: In visual effects production, processes called solving and retargeting are used to map recorded movements onto a virtual character. The recorded movements consist of a set of curves (a). First the poses of a performance rig are solved for the recorded motion, by mapping each part of that performance rig onto the corresponding curves (b). Finally, poses from the performance rig are retargeted onto the virtual character for each frame of the animation (c).

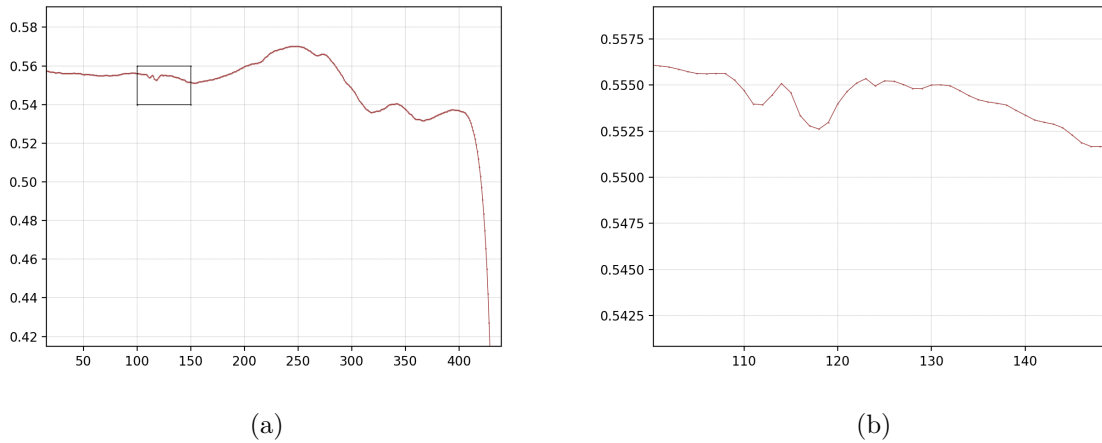


Figure 1.2: Small errors in the calculation of the markers' positions can occur when markers are occluded from the view of multiple cameras. These errors cause intermittent periods of noise throughout the motion, such as the noise in the curve depicted above (a). Figure 1.2b presents a zoomed-in portion of the curve.

are differences in the shape or the proportion of the actor and the virtual character. For example, when the length of the character's legs are shorter than that of an actor's, the solving or retargeting process can result in the character's feet sliding along the ground to catch up to the actor's feet (a problem known as foot sliding that has already been addressed in research [46, 26, 37]). Other problems can also occur, such as the character's hands becoming misplaced and consequently missing the location for important interactions. Correcting poses that are distorted by proportional differences and other defects in the mapping process is typically the second task undertaken by a motion editor when preparing an animation for production. MOTIONBUILDER provides a constraint system that motion editors can use to force the character's feet or hands into a precise position at given times. These constraints can also be animated and are commonly employed by motion editors to clean up any erroneous poses that result from solving or retargeting.

After recording and noise removal and solving and retargeting and any further cleaning of the motion has been completed, the recorded data has been successfully applied to the virtual character. In some cases, the resulting animation is sufficient for production at this point and the job for the motion editor is complete. However, there are also situations when further editing is required. For example a director may request that a fire hydrant be included to interrupt an already animated character as they walk along a footpath. One option available to the studio to accommodate the director's request is to record the motion a second time. However, doing so may no

longer be possible as the actor may be booked for other projects (or even if they are available, this option is expensive as the stage needs to be set up again and the actor and supporting staff rescheduled). Alternatively, the studio may task a motion editor, or team of motion editors, with modifying the animation to meet the new restrictions of the scene.

Contemporary animation software provides some higher level editing tools. For example, MOTIONBUILDER includes the three following tools:

- **Retiming.** In keyframe animation, an animator can speed up or slow down transitions in the motion by shifting keyframes along a timeline (illustrated in Figure 1.3). MOTIONBUILDER provides a tool that enables a motion editor to select frames in the motion capture animation and shift them along the timeline as though they were keyframes in a keyframe animation. The MOTIONBUILDER software automatically retimes the motion capture animation to match the editor's adjustments.
- **Falloff Editing.** Another approach to editing provided by MOTIONBUILDER is a method to smoothly blend a change to a pose onto the surrounding animation curve, where the amount of distortion is defined by a falloff function, as illustrated in Figure 1.4. Note that falloff editing is usually applied to the animation's curves rather than the poses of the animation.
- **Layer Editing.** In keyframe animation an animator can combine different layers of animation together, as illustrated in Figure 1.5. MOTIONBUILDER also provides a similar layer editing tool for editing motion capture animation, which allows motion editors to add a layer of keyframe animation on top of the motion capture.

Continuing from the example of the fire-hydrant described above, the editor may begin the editing task by using falloff editing to adjust the animation curves of the character's position in space, so that the character walks around the added fire hydrant. The change in the character's position as they walk through the scene may result in new footsteps being required. After adding these footsteps using layer editing, the motion editor may apply retiming to ensure that the character transitions through each of these steps in a way that appears natural. During retiming, the motion editor may also need to tweak the bend of the spine and the swing of the arms to match the adjusted footsteps. Lastly, the motion editor may tweak a few poses throughout the scene to ensure that the character appears as though they notice the fire hydrant before they turn to avoid it.

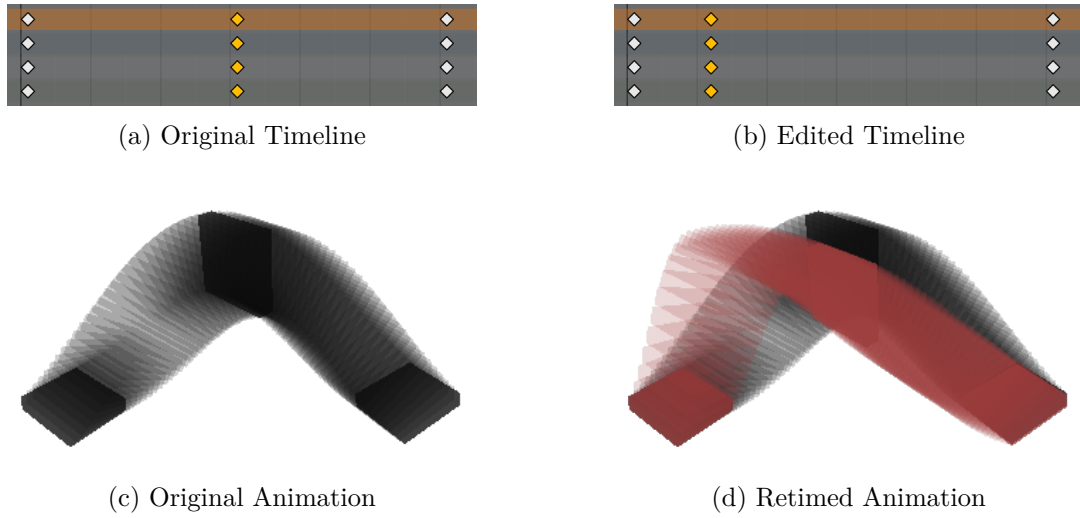


Figure 1.3: In contemporary animation software such as Autodesk’s Maya [4] and Blender Foundation’s Blender [7] animators can use the timeline interface to reconfigure the time upon which keyframes occur. In the illustration above, the second keyframe originally occurred at frame 30 (a) and was moved to frame 10 (b), which results in a speed up for the earlier half of the motion and a slow down for the later half.

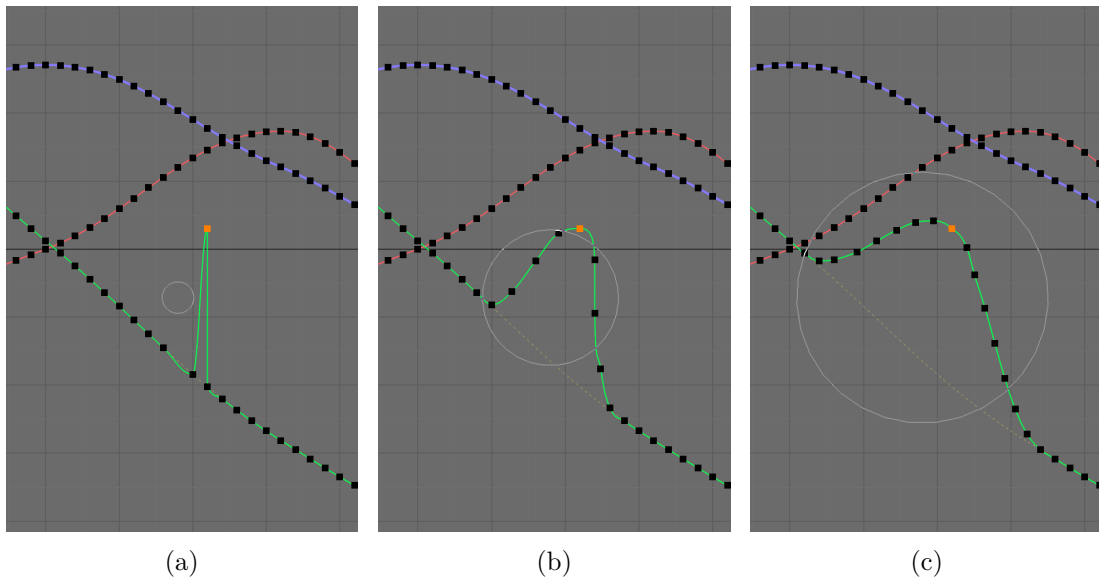
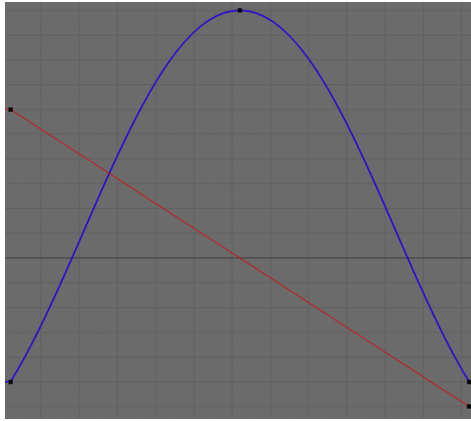
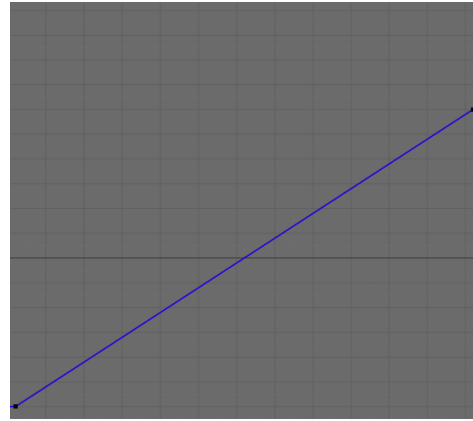


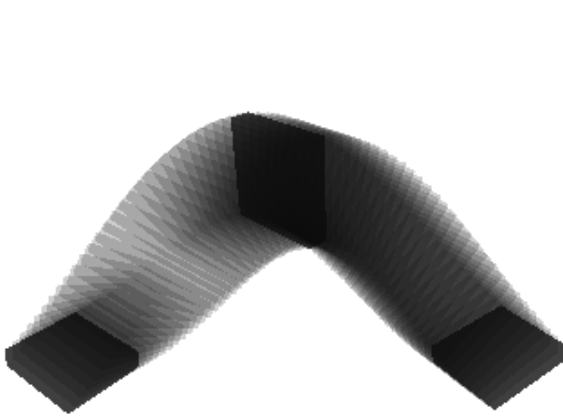
Figure 1.4: Falloff editing is a technique where an editor can blend an adjustment onto a motion curve, where the amount of change to each frame smoothly falls off further from the adjusted frame. The size of the falloff can be adjusted by the motion editor, which is illustrated by the size of the circle above.



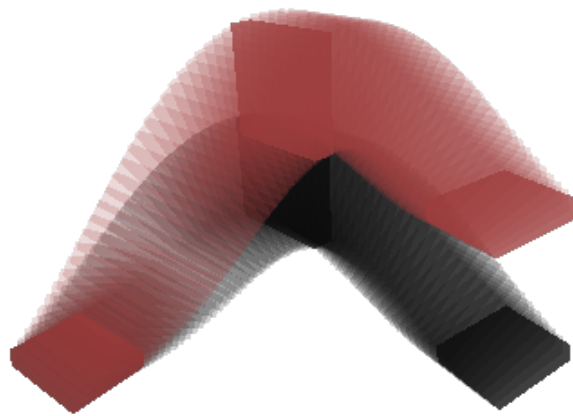
(a) Layer 1, Interpolation



(b) Layer 2, Interpolation



(c) Layer 1, Animation



(d) Layer 2, Animation

Figure 1.5: Animation layers enable animators to split different components of an animation into sub-animations, where the complete animation is constructed by adding together the corresponding curves of each sub-animation. Splitting the animation into layers in this way can be useful to decouple complicated motions, such as splitting the animation of a character shaking in fear into two parts: one layer for the larger scale details, such as the character raising their arms to hide themselves; and another for the finer scale details, such as the trembling of the character's hands. Layer editing tends to be used sparingly as the different animation layers can be unwieldy when not carefully planned and maintained.

In summary, contemporary motion editing software like MOTIONBUILDER support motion editors by providing tools to automatically remove noise resulting from incorrect marker calculations or providing constraints to help editors fix incorrectly mapped poses. Furthermore, a few editing tools are also provided for making higher level changes to the animation. Among those tools, retiming is useful for speeding up or slowing down different transitions in the motion. Beyond adjusting the timing of the motion, falloff and layer editing provide utilities for editing poses in the motion. While falloff editing works well to make a few adjustments, it is impractical to use falloff editing for extensive adjustments. As noted by Lowe, during a recent presentation at Game Developers Conference [53], performing extensive edits to motion capture animation using falloff editing tends to degrade quality of the motion since realistic movements rarely feature the smooth types of changes that falloff editing encourages. Furthermore, since adjustments made using falloff editing can unintentionally distort frames nearby the edited frame, the approach can lead to cycles of adjustment and readjustment and eventually the finer details in the motion become distorted. Finally, layer based editing can be used to enable an editing process more similar to that of keyframe animation but leaves the motion editor with the problem of adjusting the difference between the recorded motion and that which they envision, rather than editing the motion directly.

1.2 Related Work and Technical Challenges

The notion of using keyframe selection to simplify motion capture animation has already been addressed in previous research. Among the existing research toward keyframe selection (reviewed in detail in Section 2.2), there are four distinct approaches for identifying which poses of the original motion capture animation should be considered as the keyframes:

- One of the approaches is to detect the occurrence of a feature in the motion, in which poses that contain the feature are considered as keyframes [2, 66, 18, 12, 28, 44, 54, 55, 79].
- Other approaches encode the motion as a high-dimensional curve and approximate that curve [50, 70, 2, 76, 57, 55], in which keyframes correspond to the vertices of the fitted approximation.
- Some approaches employ optimization methods to choose a set of keyframes that optimally approximate the original motion [52, 81, 82, 15], in which the notion of how well the approximation fits the motion is encoded using an objective function.

- Finally, other approaches interpret keyframes as the factors of a motion [46, 51, 58, 6, 80, 36, 42, 80, 75]. These approaches choose keyframes to form a linear basis of the motion, and reconstruct the animation as a weighted blend of the poses in that linear basis. *Due to their different interpretation of keyframes (keyframes as factors), these techniques cannot be applied to our keyframe selection problem.*

The techniques listed above can begin to address some of the challenges present in the problem of converting motion capture into keyframe animation. In this section we expand our previous description of the technical challenges inherent to our problem. We also summarize where further development is required to fully realize a complete solution.

A central challenge in the design of our keyframe selection algorithm is to develop a mechanism for balancing the level of accuracy against the level of compression. In some editing situations, particularly those containing a highly dynamic motion (perhaps acrobatic motions or fighting motions), the motion editor may be willing to sacrifice the impact that the keyframes have over the motion to ensure that all large and medium scales of detail are preserved accurately. Conversely in other situations where the motion editor is intending to make extensive changes, they may be willing to sacrifice some of the detail in order to ensure that the resulting keyframes enable a high level of impact over the motion.

The approximation-based technique can begin to address this problem. Approximation techniques typically begin with only two keyframes selected (the first and last frames of the motion). Keyframes are then successively added until the animation that would result from those keyframes is a close approximation of the original. With an implementation of such an algorithm, a motion editor can first generate all selections of keyframes and then choose the particular selection that features a balance of accuracy against compression that is appropriate for the editing task at hand. However, a limitation of these approximation-based techniques is that they cannot rearrange previously selected keyframes, as higher number of keyframes become available, to best approximate the animation.

Optimization-based techniques achieve better selections than approximation-based techniques, in that they encode both the qualities of accuracy and compression into an objective function that can be minimized using an optimization method. By specifying how strongly the notions of accuracy and compression appear to the objective function, an editor can bias the optimization-based technique to favour more accurate or more impactful selections of keyframes. Regardless of the configuration, the optimization-based techniques ensure an optimal selection of keyframes (at least when they converge correctly to a global optimum), and as such the given selection of keyframes approximates the motion as best as possible for the

given level of compression. However, optimization-based techniques are significantly more expensive to compute than the approximation-based techniques.

We develop an approach that is designed as a hybrid of these two types of keyframe selection techniques: we borrow from optimization-based techniques the notion of an objective function and develop an algorithm that composes each successive selection of keyframes from the previous (similar to the iterative process seen in approximation-based techniques). By doing so, we are able to offer a new keyframe selection that is both fast and optimal.

In summary, the topic of keyframe selection has already been addressed in previous research. While the existing techniques progress toward a keyframe selection algorithm suited to converting motion capture into editable animation, none of the existing techniques can offer a solution that is both fast and optimal.

In this thesis, our goal is to provide the first fast and optimal keyframe selection technique. Such a technique would be attractive to motion editors because (1) the optimal quality ensures that each of the selections feature the best trade-off between accuracy and compressions and (2) the technique is fast enough to generate all selections quickly, so as not to obstruct the motion editor’s workflow by making them wait for expensive computations.

1.3 Research Contributions and Outline

Here we summarize three primary contributions of this thesis.

- **Optimal Selection Space.** Much like the approximation-based techniques, our technique successively adds keyframes to an initial selection. Through expanding the selection in this way we can offer an entire range of selections which include both highly accurate selections (low compression) and highly impactful selections (high compression), as well as all those inbetween. Distinct from optimization-based techniques that only provide a single selection, we develop a method that forms all possible optimal selections (one for each level of compression). Thus, once our keyframe selection terminates we can provide the editor with a range of optimal keyframe selections, from which they can choose a particular selection whose level of compression is most suited to their editing task. To ensure that all selections provided to the motion editor are optimal in terms of the given objective function, we formulate the problem of keyframe selection using a graph that exhibits an optimal substructure. The graph formulation and how it can be used to realize a fast and optimal technique are presented in Chapter 3.
- **Generic Objective Function.** Much like the optimization-based techniques,

our technique employs an objective function to describe which of the potential selections best describe the original motion. Distinct from the existing optimization-based techniques we purposefully leave the implementation of the objective function abstract. That is, we have designed our technique to be able to optimize any implementation of the objective function. In Chapter 4, we describe three implementations that can be used together to select animator-like keyframes.

- **Accurate Reconstruction.** Finally, in order to reconstruct a new editable animation from only the selected keyframes we require a technique to interpolate those keyframes in a way that closely approximates the detail of the original motion. To do so, we develop a new curve fitting technique that excels at accurately reconstructing each of an animation’s curves using only the given keyframes.

Our contributions are novel because they realize the first fast and optimal technique for converting motion capture into editable keyframe animation. We title our keyframe selection algorithm *Salient Poses*; the title stems from an algorithm, titled *Salient Points*, that was presented by Lewis and Anjyo [49]. We introduce Lewis and Anjyo’s algorithm before presenting our keyframe selection in Chapter 3.

Supplementary material – including reference and performant implementations, further documentation, and video materials that help to illustrate how these contributions can be integrated with animation software – are provided online ¹.

1.4 Thesis Scope

Here we provide a few further details that help to outline the scope of this thesis.

- **Motion Capture Data** As with body animation, motion capture systems for facial motion are also a core component of animation production in contemporary visual effects pipelines. Beyond body and facial animation, prototype motion capture systems for recording hand and eye movements are also emerging, but are not yet typical in production. While each type of motion capture is an attractive research topic, we address only body motion in this thesis. We choose to focus on body animation because there are a number of instructional resources for animators surrounding body animation that can support the ideas within this thesis and relatively few for facial, hand, and eye animation. Furthermore, Carnegie Mellon University provides a free motion capture database of body

¹See the online repository at <http://github.com/richard-roberts/PhD>.

animation [14]. Similar databases are not yet freely available for the other types of motion capture. Despite our focus on body animation, we have intentionally designed our technique to be robust so that it may be easily adapted to other types of motion capture in the future.

- **Problem Size** In this thesis we limit the size of motion capture problems to relatively short clips. The motivation behind this is to present a study focused on recovering editable animation from clips of motion capture that would typically be edited, rather than a study focused on scaling the technique to larger problems. The length of the motion capture clips used in our motion capture examples are based on the average shot length observed between camera changes in films. In recent films, average shot length has been observed to be between four and six seconds [59, p. 166], [10, p. 120], [43, p. 126]. Based on these observations, we assume that a typical motion capture clip for a shot would contain between 480 and 720 frames (assuming a frame rate of 120 frames per second). Consequently, we propose that clips of motion capture between 480 and 720 frames are indicative of what the content to be adjusted in typical editing tasks.

Chapter 2

Related Work

In this chapter we expand our discussion of animation practice and keyframe selection to provide a context and a foundation for later discussion in this thesis.

We begin in Section 2.1 with a narration of how conventions in animation practice emerged from traditional animation and how these conventions inspired early computer animation software. This narrative will provide further background on the processes used by animators to create keyframe animation and, in particular, the types of keyframes that animators use to when creating keyframe animation that remains editable. To support that the concepts we introduce through the narrative reflect conventional animation practice, we surveyed key instructional resources in animation: Williams’s “Animation’s Survival Kit” [73], Thomas and Johnston’s detailed review of the history and practices of Walt Disney Studios during the production of classic animated films [23], and also animation guidebooks by White [72] and Roy [63]. Additionally, we discussed our interpretation of animation practice with the head of research [1] and animators at OLM Digital, a Japanese animation studio utilizing both traditional and computer animation practices together.

We then continue into Section 2.2, in which we review both seminal and recent research toward keyframe selection. The survey is intended as a detailed review of the topic of keyframe selection for motion capture and provides a context in which we can state the limitations of existing keyframe selection techniques when applied to our problem of converting motion capture into editable keyframe animation. The survey also provides a context in which we can place our contributions to keyframe selection.

We refer readers interested in a broader introduction to techniques on motion capture data processing, editing, and keyframe selection to Wang et al.’s survey [71], which locates the seminal and recent research in those topics.

2.1 Animation Practice

In computer animation practice, animators typically create motion in three phases: planning, in which reference materials are gathered and the motion is sketched by hand or drafted on simple test characters; blocking, in which keyframes are plotted into the animation software and then adjusted until the animator is satisfied that the motion is a plausible and compelling approximation of the envisioned motion [60, 72, 63]; and refinement, in which the animator adds and tweaks keyframes and their interpolating curves to add finer scale detail to the motion.

The second blocking phase in keyframe animation is of particular interest to the design of our keyframe selection algorithm. As we describe later in this section an animator, when following conventional animation practices, will create two types of keyframes during the second blocking phase: *keyposes* and *breakdowns*. The keyposes are keyframes that define the shape of the important poses in the motion and breakdowns are keyframes that define how a character transitions through those keyposes. Once the animator has completed blocking, the keyposes and breakdowns together comprise a close approximation of the eventual animation (missing only the finer scale of detail). An animation is most editable in this blocked form, since it typically contains the fewest keyframes required to express the motion well.

2.1.1 Traditional Animation

As narrated by Thomas and Johnston, the *Walt Disney Productions* animation studio purposefully neglected a strict hierarchical system and instead favoured a workflow that allowed them to swiftly incorporate practices that improved efficiency or quality while discarding those that did not [23, p. 29-46]. An early example of such an adaptation was the introduction of the assistant animator, responsible for cleaning up the line work in drawings that were otherwise ready for production. The role was introduced after the lead producer, Walt Disney, noted that rough sketches of a motion held a greater sense of vitality and liveliness than a carefully planned and executed sequence of drawings. Splitting the animation process into two roles (the lead animator and the assistants) enabled leads to forgo concern of production quality and because of this they were able to focus on generating lively animations.

The studio continued to adapt throughout productions such as David Hand’s *SNOW WHITE AND THE SEVEN DWARFS* [29] and *BAMBI* [30] and Ben Sharpsteen’s *DUMBO* [65]. Each generation of animators contributed new ideas that continued to build toward the studio’s production processes.

A conventional workflow emerged as the studio’s practices matured. First a team of directors, layout artists, and storyboard artists drafted the production: the directors



PROD.	REC.	SCENE	"CHARACTER GA-GA"					SHEET
	(D)	(S)						(1)
			MOUTH	EYES	ARM	BODY		
ACTION	DIAL	EXTRA	M	E	A	B	EXTRA	CAMERA INSTRUCTIONS
01			(M-1)	(E-1)	(1)	(B-1)	(BGS)	 (START)
02								
03					3			
04					(5)			
05								 NOTE: ARM DRAWINGS ARE RE-USED HERE!
06					7			
07				(9)	9			
08				(11)	(11)			
09					(9)			
10					(7)			
11				(15)	(7)			
12				17	(5)			
13				(19)	(3)			
14					(1)			
15			(23)					
16			(25)					
17			(27)					
18					29			
19					(31)			
20								
21								
22								
23								
24								
25								
26								
27								
28								
29								
30								
31								
32								

Figure 2.1: White, in their guidebook on animation practice, illustrate how traditional animators tabulated keyposes (circled) and breakdowns (parenthesized) using a dopesheet [72, p. 356]. In this example, the animator has used the columns to denote each part of the character: their mouth, eyes, arms, and body. A dopesheet for a more advanced scene may include columns for interactions between the character and objects in the scene and also list any spoken dialogue. *This image has been copied from White's book [72] and has been reprinted with permission from the author. © 2006 by Tony White.*

would plan out the events that occurred in each scene, storyboard artists would create comic panels for a few of the main character’s keyposes, and layout artists would composite sketches of characters in their scenes. The scene information, along with notes about sound effects and dialogue, would be compiled into a document called a dopesheet (illustrated in Figure 2.1).

Once a scene had been drafted, the materials along with the dopesheet would be handed off to a small animation team. These small teams typically contained one lead animator paired with a few assistants. The lead animator drew the keyposes and the more important breakdowns, while assistants would draw the inbetweens. Once each of the frames had been drawn, the drawings were organized and submitted for clean up and colouring. Finally, the drawings were stacked upon one another to form the complete scene and then photographed to create a frame in the film [23, p. 225-229].

Thomas and Johnston [23] and Williams [73] recount a set of *twelve animation principles* that summarize the key practices of professional animators at the Walt Disney Productions studio:

- The principle of “arcs” states that each of a character’s joints should trace smooth arcs through space as the character moves through the scene.
- The principle of “anticipation” explains that a character should show preparation for a significant movement (such as crouching before a jump), while the principle of “follow-through” describes that a character should react after a significant movement (such as stumbling when contacting the ground again after the jump).
- The principles of “slow in and slow out” and “squash and stretch” explain how the distribution of mass in a character changes with respect to the speed, and direction of, changes in their pose.

Together the principles serve as a framework for animators and can guide their understanding of how a character should move in order to produce a compelling motion that communicates a narrative clearly while also appearing compelling to the audience.

Of particular interest to this thesis is another principle titled “Straight Ahead Action and Pose to Pose”, which proposes an ideal workflow for creating animation. When using the *straight ahead* approach, an animator simply draws each pose in the motion in chronological order. Conversely, when following the *pose to pose* approach, an animator first creates keyposes and then adds breakdowns. A keypose should be created for all important poses in the motion, such as moments where the character transitions through an extreme position (perhaps the peak of a jump) or otherwise interacts with the scene (such as their foot colliding with the floor). Breakdowns

should then be added to describe how the character moves between the keyposes more precisely.

To help illustrate how an animator structures keyposes and breakdowns, Figure 2.2 presents the keyposes and breakdowns used in an animation where an Agent character first turns backward to spot an attacker and then swings their suitcase outward in defence (in the figure, each pose is slightly offset from the previous to indicate how time passes through the motion). From Figure 2.2a, we can observe that the animator has created keyposes before and after the turn-backward and turn-forward action and also to surround other action where the Agent swings their suitcase outward. Breakdowns were placed to refine the timing of the turn actions and also to define the arc along which the Agent swings their suitcase.

As summarized by Williams, following a straight ahead approach instils a sense of liveliness in the motion but makes it difficult to ensure that the timing of important events and interactions occur as the animator has intended. In contrast, the pose to pose approach enables the animator to plot the important events and interactions first and then continue to develop the style of the performance. However, when the pose to pose approach is used extensively, the resulting motion can lack a feeling of liveliness in comparison to the motion that would have otherwise resulted from using the straight ahead approach. Ultimately, Williams suggests that an ideal workflow is found when combining both approaches: the animator should first block out keyposes and breakdowns and then construct the more granular transitions through each of those keyframes using the straight ahead approach. In particular, employing pose to pose first ensures that the character will transition through each of the important events and interactions at the correct time, while the later use of straight ahead instils a sense of liveliness to the scene [73, p. 47-68].

2.1.2 Computer Animation

The use of keyframes in 3D computer animation was pioneered in Stern’s Animation System [67], and was popularized by an early animation tool developed at Pixar titled the “Menv Modelling and Animation Environment” [11, p. 46-48]. Reeves et al. [62] describe how animators, using this software, could create animation by plotting keyframes into a grid interface called a cue sheet. The cue sheet interface is markedly similar to the dopesheet used in traditional animation. Each column of the cue correlates to a frame number and each row correlates to a degree of freedom of the character’s pose (Figure 2.3b). An animator creates a keyframe by specifying a value for one or more of the cells in a particular column of the grid. The system interpolates through each of the keyframe values and by doing so automatically



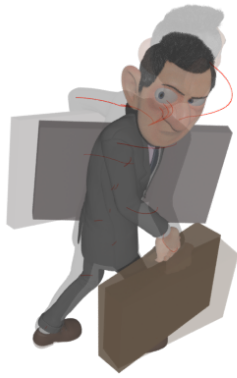
(a) Animation Sequence



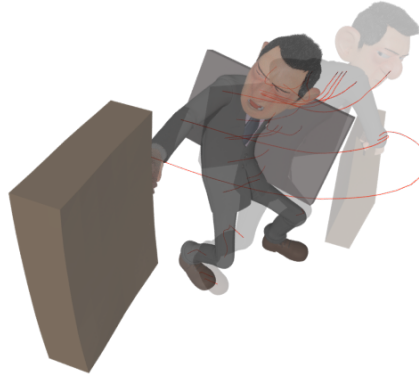
(b) Segment Two



(c) Segment Six

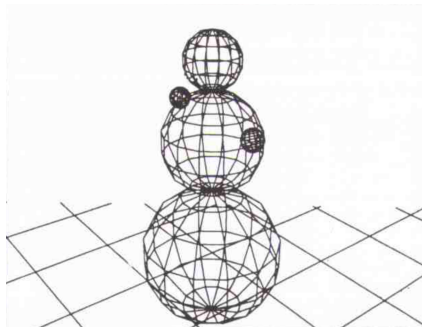


(d) Arcs of Segment Two



(e) Arcs of Segment Six

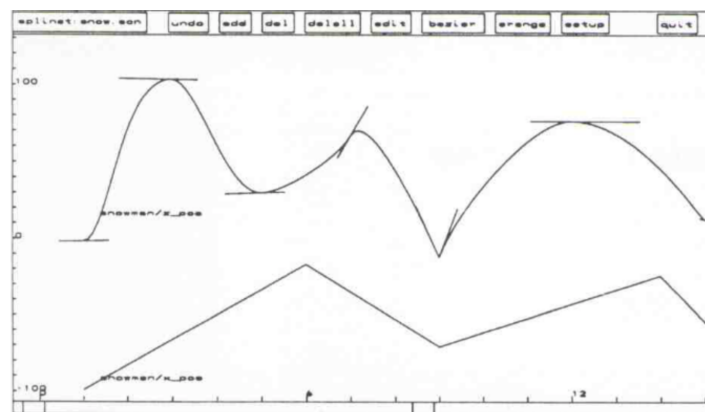
Figure 2.2: In this animation an agent character turns backward to spot an assailant before moving into a crouch and attacking them with their suitcase. The keyposes of the animation are shown in full colour, the breakdowns are silhouetted in blue, and the transparent silhouettes are inbetweens. Each frame of the animation has been offset horizontally to illustrate the amount of time between keyframes. Each pair of keyposes forms a segment in which a transition takes place. For example, the second and third keyposes form a segment in which the character turns their head (b) and the last segment encapsulates the outward swing of the arm (c). The arcs of these two segments are illustrated by Figures 2.2d and 2.2e.



(a) View

edit	end	son	undo	add	del	sel	air	move	copy	paste	linear	scene	class	setup	quit
3.000	even		0	1	3	5	7	9	12						
-66.24	x_pos		-99.06			-17.01		-71.03							
0.00	y_pos														
103.33	x_pos		-1.71	103.33	29.69		66.16	-12.29	76.01						
-31.37	scale		-31.37												
20.07	rotation		20.07												
	get eye														

(b) Cue Sheet



(c) Spline Tool

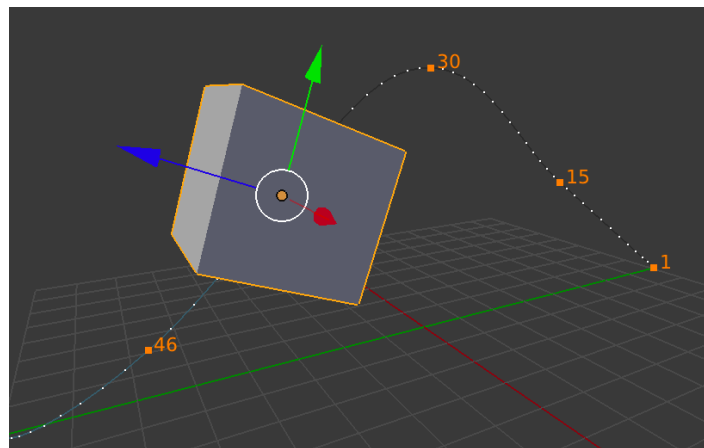
Figure 2.3: The Menv Modelling and Animation Environment system provided a toolkit for computer animators that featured three interfaces for creating animation: a 3D view of the scene (a), a dopesheet-inspired interface called the cue sheet (b), and also a splining tool called the graph editor (c). Animators could adjust the values describing the keyframes' poses using the cue sheet interface and interactively move the control points of the interpolating curves to manipulate inbetweens. *This image has been copied from Reeves et al.'s publication [62] and has been reprinted with permission from the Wiley Global Permissions team. © 1990 by John Wiley & Sons, Ltd.*

constructs smooth animation curves for each part of the animated character. Beyond the dopesheet interface, the system also provided a graph editor through which animators could adjust the shape of the curves interpolating through the keyframes (Figure 2.3c), which is typically used in the later refinement stage of animation when finer scales of detail are added to the animation.

The Menv Modelling and Animation Environment system was inspired by concepts that emerged from traditional animation practice; in particular the dopesheet interface and the notion of keyframes and inbetweens. The core components of the early software – its viewport, cue sheet, and graph editor – remain as the foundations of contemporary animation software: Autodesk’s Maya [4] and Blender Foundations’s Blender [7] both provide a viewport, a dopesheet, and a graph editor (Blender’s is illustrated in Figure 2.4). The implementation of both these utilities have diverged little from their initial design in the Menv Modelling and Animation Environment system. One significant difference, however, is that in contemporary software the keyframes can be specified by posing the virtual character interactively in the 3D view (rather than keying them manually into the dopesheet), in which the character functions as a poseable figure.

Animation software enables animators to pursue a workflow similar to that of traditional animation. By providing inbetweens automatically through an interpolation of the keyframes, the software plays the role of the assistant and leaves the user to work in much the same way as a traditional lead animator. The dopesheet interface mirrors the functionality of the traditional dopesheet and encourages animators to block out their animations as a sequence of keyposes and breakdowns. Despite the differences in their mediums, the workflows of the modern day computer animator have changed little from their traditional counterparts and, consequently, an animator can work traditionally or on the computer using the same set of animation principles: the motion is first imagined, then blocked out using keyposes and breakdowns, and then refined by adding finer scales of detail. The only differences, in so far as the animator is concerned, is that they use either the physical toolkit and hand-off to an assistant for inbetweens or that they use the digital toolkit and hand-off the animation to the computer for inbetweens.

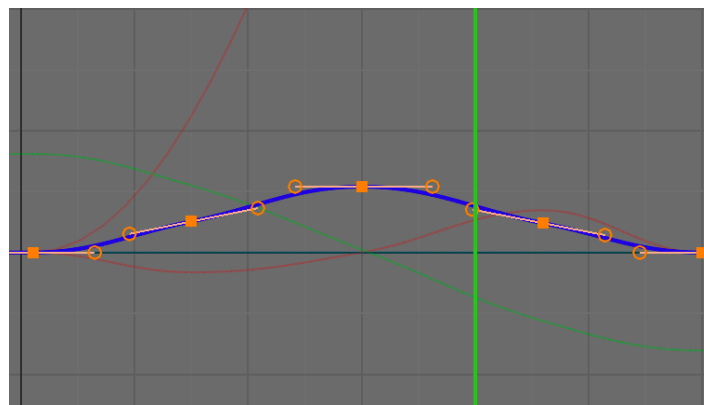
In conclusion, ideas from traditional animation have had a significant impact on the development of animation software. In the context of this thesis, we propose that the animation principles can help to define what it means for an animation to be editable. In particular, the idea that animators use a combination of keypose and breakdowns. For the scope of this thesis, we define an editable animation as having a sparse set of keyframes that are distributed in such a way that they are similar in structure to those keyposes and breakdowns that may have otherwise been created by an animator.



(a) View



(b) Dopesheet Sheet



(c) Spline Tool

Figure 2.4: Blender Foundation’s Blender [7] is a contemporary and open source animation software. Like the Menv Modelling and Animation Environment system, Blender includes a 3D view of the scene (a), a dopesheet (b), and a graph editor (c). Animators can pose their character in the view of the scene and choose to have their poses automatically entered into the system as keyframes at the current scene time (indicated by the vertical green bar in (b) and (c)). As keyframes are added, Blender automatically populates the dopesheet with columns of keys. Animators can slide these keyframes back and forth to change the timing of the animation and also annotate these keyframes as either keyposes or breakdowns (indicated by the different colours shown in (b)).

2.1.3 Terminology

We now summarize a number of terms that will be referred to throughout this thesis. Primarily, these terms are used to help communicate how we designed our objective functions to realize the selection of keypose-like and breakdown-like keyframes (Chapter 4), and in the concluding remarks regarding the contributions of this thesis (Chapter 6).

- **Keypose** A type of keyframe used during blocking. The poses of these keyframes are typically authored first to define extreme events in the character’s motion and to denote interactions between the character and scene. Once the initial set of keyposes have been authored, the narrative and timing of the motion should be evident through them.
- **Breakdown** Another type of keyframe used during blocking. Breakdowns are used to define how a character transitions through pairs of keyposes. Any number of breakdowns may be used but typically one breakdown occurs halfway between two keyposes for simple transitions and between two and five breakdowns for more complex transitions.
- **Blocked animation** An animation created by interpolating the complete set of keyposes and breakdowns together. The number of keyframes used tends to range between a minimum of two per second (one keyframe per 12 frames of standard animation) and a maximum of twelve per second (one keyframe per two frames of standard animation). A blocked animation does not include finer scale details.

2.2 Keyframe Selection

Keyframe selection is perhaps best described as a specialized topic in data reduction. The idea of keyframe selection for motion capture is simple: to select a set of keyframes that summarize the given motion. By creating a new animation that consists only of the selected keyframes, one can reduce the amount of data required to represent the original motion.

One application of keyframe selection is compression. Compressing motion capture animations enables faster loading and viewing of the motion, which is useful for browsing or processing large databases of motion capture [50, 36, 44, 52, 80, 82]. For example, one type of common processing operation applied to larger databases is to compare the motions for similarity. A comparison operation that visits all the frames between two or more animations may be expensive, whereas comparing only the poses of selected keyframes enables a faster operation. Faster comparisons

between animations are especially useful for searching motion databases or for retrieving animations that contain a particular sequence of poses [46, 51, 66, 36, 76, 6].

Keyframe selection has also been applied to visualize motion capture animation. For example, an entire motion can be summarized by selecting keyframes, rendering the pose of each of those keyframes, and compositing those renders together into an image [2, 3] or a timeline [78, 35]. Visualization can be helpful to motion editors looking to preview adjustments to a motion, for showing differences between motions, and also for browsing databases of animation.

Finally, and most importantly in the context of this thesis, keyframe selection is advantageous in terms of editing. In particular, removing excessive keyframes from the motion means that the remaining keyframes can have a higher level of impact over the animation when adjusted [57, 28, 54, 55].

To the best of our knowledge, the existing research is yet to explore the potential for keyframe selection techniques to identify keyframes that are similar to those created by animators for keyframe animation. Consequently, we place the contributions of our thesis as the first application of a keyframe selection technique specifically designed to reverse engineer animator-like keyframes from animations created with motion capture data.

In this section, we provide a detailed survey of four types of techniques that have been proposed for keyframe selection with respect to animation created from motion capture. We summarize the four types of techniques as follows:

- *Detection*, the process of selecting keyframes by detecting the occurrence of features or patterns within the poses or curves of the animation. We categorize the following research as detection-based: [2, 66, 18, 12, 28, 44, 54, 55, 79].
- *Approximation*, the process of fitting an approximation to an animation's curves and selecting keyframes as the vertices of the resulting approximation. We categorize the following research as approximation-based: [50, 70, 2, 76, 57, 55].
- *Optimization*, the process of selecting keyframes in order to minimize an objective function that encodes both the levels of accuracy and compression afforded by the selected keyframes. We categorize the following research as optimization-based: [52, 81, 82, 15].
- *Factorization*, the process of choosing a subset of frames to serve as a linear basis from which the animation can be reconstructed. We divide the factorization-based techniques into two groups: those that employ clustering algorithms [46, 51, 58, 6, 80] and others that employ matrix factorization algorithms [36, 42, 80, 75].

The focus of this survey is to introduce each of the four types of technique and to examine their advantages and disadvantages. To conclude the survey, we will continue to outline how the existing research can support our goal of selecting animator-like keyframes. We will also isolate the limitations that need to be addressed before realizing that goal.

Finally, it should be noted that we have omitted discussion of keyframe selection techniques designed for video data, as these techniques are not readily compatible with motion capture. For an introduction to video-based keyframe selection techniques readers may refer to the seminal works [19, 27, 17] or to Sujatha and Mudenagudi’s survey of the topic [68].

2.2.1 Detection

In detection-based techniques keyframes are selected by identifying the occurrence of distinctive features in the motion. Distinctive features may be considered with respect to either the animation’s curves, the changes between poses in the animation, or otherwise with respect to changes in a parameter derived from the motion.

In order to distinguish frames as keyframes, the algorithms used in detection-based techniques typically scan over each frame of the motion and measure how strongly the chosen feature occurs at that point in the animation. Frames that appear significant are either selected directly as keyframes or otherwise specified as candidates keyframes. In the latter case a further refinement step is used to finalize the selection, typically by dividing the candidates into groups based on proximity and then selecting one final keyframe from each group.

The design of detection-based techniques vary from simple to advanced. Miura et al. [54] propose a simple technique that measures the sum of the velocities of the character’s joints at each frame in the motion. Based on the notion that actions in a motion are usually separated by moments of low-velocity, Miura et al.’s technique selects all frames that feature sufficiently low velocity as keyframes.

As an example of a more advanced detection-based technique, Bulut and Capin [12] propose employing a saliency filter¹ to select keyframes as frames that feature significant differences in curvature when measured at different scales. In their technique, the saliency filter is applied to measure the differences in curvature for all points in each

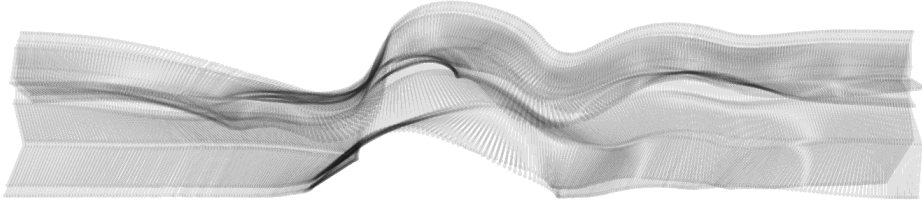
¹Itti et al. [38] proposed a filter that detects distinctive elements in images (such as a sign in the foreground of a photograph). The filter uses a weighted averaging algorithm to measure differences in curvature over smaller and larger localized areas in an image. A significant difference between these averages suggests that the curvature is smooth at one scale but not at another. Itti et al. propose that differences between the averaged curvatures correspond to interesting details and therefore the filter can be used to identify distinctive elements.

of the animation’s curves. Any frames that feature a greater than average difference in curvature are selected as candidate keyframes. The final selection of keyframes is then obtained by organizing the candidates into groups based on closeness in time and then selecting a single keyframe from each group. Halit and Capin [28] later revisit Bulut and Capin’s technique and propose to apply the saliency measure to the principle components of the animation’s curves (rather than the animation’s curves directly).

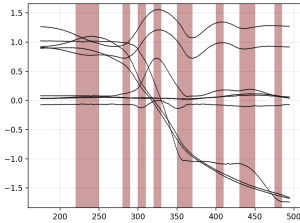
Other detection-based techniques have been proposed to select keyframes based on changes between the poses of successive frames in the animation. So and Baciú [66] propose a technique that measures the entropy of changes between keyframes using a mutual information algorithm: when a change between two successive frames features a low level of entropy, the first of those two frames is selected as a keyframe. A similar technique has been proposed by Cuntoor and Chellappa [18], in which the changes between frames are instead represented by transformation matrices. Finally, Kim et al. [44] propose a technique that first divides the animation into small segments that each contain the same number of frames. A keyframe is then selected in any segment that cannot be accurately reconstructed with a simple interpolation (through the first and last frames of that segment).

A notable disadvantage of detection-based techniques is that a motion editor cannot control the level of compression (the number of keyframes being selected). While some of the detection-based techniques propose using a parameter to define how significantly a feature must occur to be selected as a keyframe [66, 18, 54], there is no way to increase or decrease the number of keyframes precisely. In more advanced detection-based keyframe selection techniques, the lack of control over the number of keyframes being selected is further exaggerated. For example, Halit and Capin [28] suggest omitting finer scales of detail from the animation to reduce the number of keyframes, or otherwise to tweak the scales at which the saliency filter measures curvature. Unfortunately, neither of these mechanisms enables a simple and intuitive solution for configuring the level of compression. Without a simple mechanism to express the level of compression, motion editors cannot affect the balance between the accuracy and compression of the animation that results from the keyframe selection (which may be required for different editing tasks).

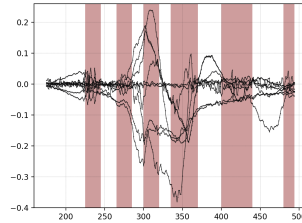
Since the keyframes selected by detection-based techniques tend to correlate to visually iconic poses, an advantage of these approaches are that the selected keyframes effectively divide the motion into small and distinctive segments. As such, detection algorithms excel at separating an animation into regions that contain different scales of detail. For example, a detection-based algorithm may divide a jump animation into segments such that one segment is formed for the part of the motion before the jump, another for the crouch, another for the leap into the air, and so on (see Figure 2.5).



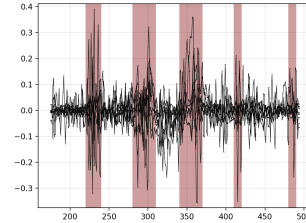
(a) Animation



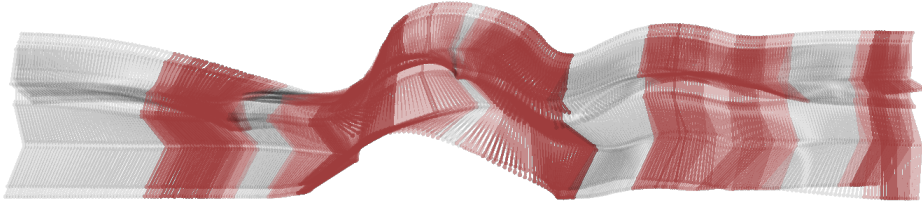
(b) Position Curves



(c) Velocity curves



(d) Acceleration Curves



(e) Distinctive Poses

Figure 2.5: Detection-based keyframe selection techniques tend to select keyframes that correlate to visually distinctive poses in the animation. In the case of the jump animation presented above (a), the position of the character’s joints will tend to feature the most significant changes in terms of position (b), velocity (c), and acceleration (d) when the character first crouches to leave, and then later contacts the ground (the highlighted regions depict the larger changes among each set of curves). Figure 2.5e highlights regions of the animation based on the scale of the positional changes.

Another important advantage of detection-based techniques is that their design tends to enable a fast implementation. Since each frame in the animation need only be visited once to measure the occurrence of distinctive features, these algorithms tend to feature linear time complexity with respect to the number of frames in the animation.

2.2.2 Approximation

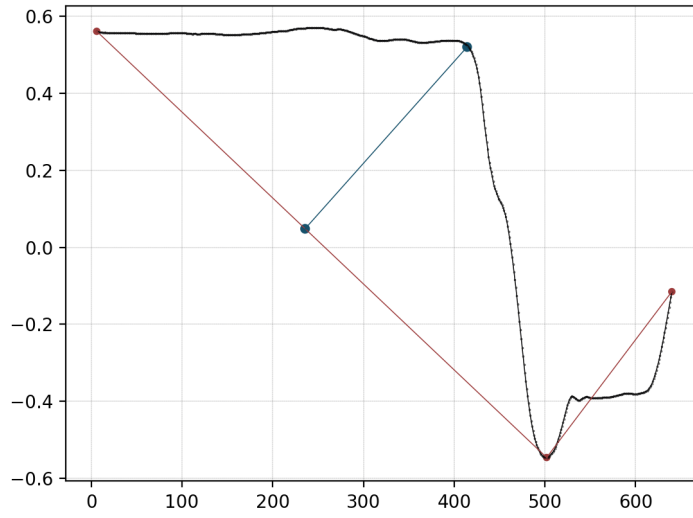
Approximation-based keyframe selection techniques fit an approximation onto the animation's curves. The approximation begins with only the first and last frame of the animation selected and iteratively adds keyframes until the approximation (resulting from the keyframes in some way) is a sufficient accurate replication of the original animation.

Lim and Thalmann [50] proposed a seminal approximation-based technique. They model the animation as a single high dimensional curve and extend a classic approximation algorithm, illustrated in Figure 2.6, to iteratively select further keyframes: instead of fitting a 2D polyline to a 2D shape, Lim and Thalmann fit a high-dimensional polyline to the high dimensional curve representation. Keyframes continue to be selected, as the point in the high-dimensional animation curve furthest from linear interpolation of the previously selected keyframes, until the approximation is sufficiently similar to original animation.

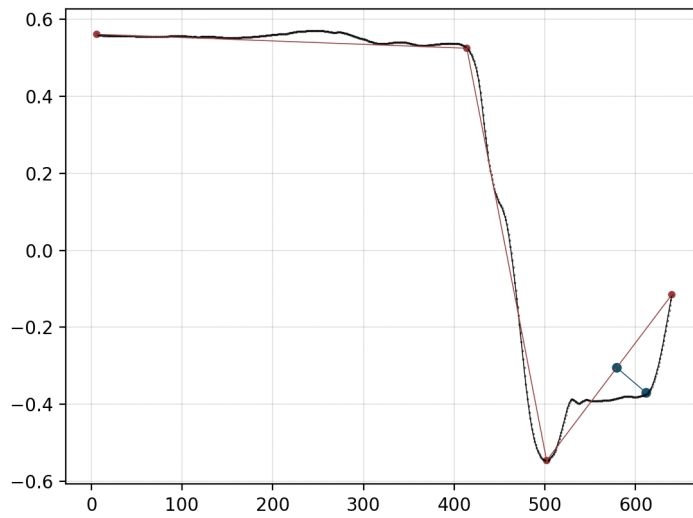
Togawa and Okuda [70] proposed a similar approximation-based technique. Similar to Lim and Thalmann, they also propose a greedy selection process. However, they initialize their technique with all frames selected as keyframes. Then, in each iteration, a keyframe is removed rather than added. The keyframe removed in each iteration is the one that distorts the approximation the least when removed. The technique continues to remove keyframes while the approximation remains sufficiently similar to the high dimensional curve representing the animation.

A drawback of these approximation-based techniques is that their keyframes are suboptimal, in the sense that there are other possible selections containing the same number of keyframes that better approximate the motion. The suboptimal selection is a consequence of the greedy design of these techniques (the choice of previously selected keyframes cannot be adjusted to make better use of decreasing levels of compression). As illustrated in Figure 2.7, suboptimal selection leads to a poor distribution when lower levels of compression become available. Furthermore, suboptimal selection is also problematic in situations where artefacts such as spikes are present in the animation's curves (see Figure 2.7c).

An important advantage of approximation-based techniques is that they can offer precise control over the number of keyframes to be selected. A motion editor can simply

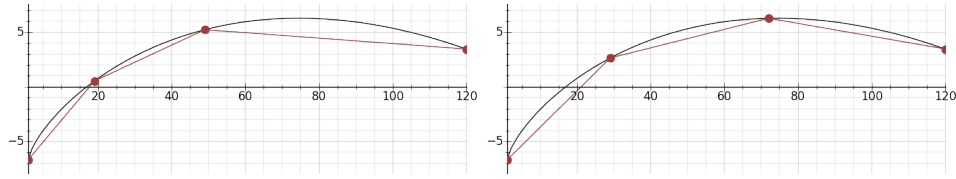


(a) Three Points Selected

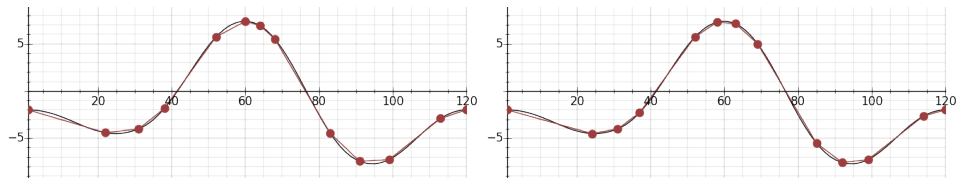


(b) Four Points Selected

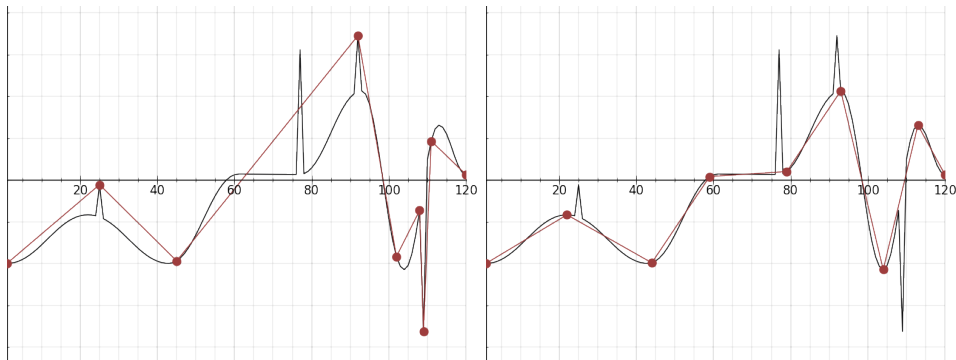
Figure 2.6: The Ramer Douglas Peucker algorithm [61] is an approximation-fitting technique, which appears throughout many areas of computer graphics research and has been attributed to many researchers who proposed the technique simultaneously [32]. To initialize the fitting, the end-points of the curve are linearly interpolated and the success of the approximation is quantified as the maximum perpendicular Euclidean distance between the original curve and the approximation (a). The algorithm features a simple design, in which the point responsible for the most significant difference is added to the approximation (b). The algorithm is repeated until the approximation is sufficiently close to the original. Lim and Thalmann's technique [50] is effectively a high dimensional extension of the Ramer Douglas Peucker algorithm.



(a) Poor Distribution in Large Arcs



(b) Poor Distribution Around Peaks



(c) Poor Distribution around Artefacts

Figure 2.7: Two problems occur with the greedy selection of keyframes. The first is that the selected keyframes cannot be redistributed when larger numbers of keyframes become available. Figure 2.7a presents a simple case where the greedy approach produces keyframes that under-represent the curve (left). A better distribution is depicted for comparison (right). The distribution is better because the keyframes are distributed to have equal impact over the curve. In Figure 2.7b the keyframe at frame 64 of the suboptimal selection (left) has less impact than its neighbours. A better selection would distribute the keyframes around this peak evenly so that the two centred keyframes have an equal impact over that part of the animation (right). As depicted by Figure 2.7c, in cases where animation curves feature artefacts, such as spikes, the greedy approach risks selecting outliers (left) whereas a better selection would consider all points to obtain a closer approximation (right).

define the number of fitting steps to perform, which corresponds exactly to the number of keyframes selected. Furthermore, these approximation-based techniques enable fast computation since each successive selection of keyframes can be obtained by adding a single keyframe to the existing selection. Thus the technique only needs to visit each of the potential keyframes once to complete each iteration.

Finally, a few keyframe selection techniques have been proposed that employ a hybrid of detection-based and approximation-based techniques. Assa et al. [2] proposed a technique that first encodes the position, orientation, and velocity of each of the character’s joints into matrices. A low dimensional interpretation of the animation is then derived from those matrices using a dimensionality reduction algorithm. An initial set of keyframes are detected as minima in the low-dimensional representation and then a greedy selection process, similar to Lim and Thalmann’s approximation technique [50], is used to expand the selection of keyframes until the approximation accurately represents the original motion. As another example, Miura et al. [55] proposed to form an initial selection of keyframes by detecting the minima of the first principle component of the animation’s curves and then to add further keyframes with Lim and Thalmann’s approximation technique. The hybrid approach is useful in that such techniques can select keyframes that both divide the animation into regions of distinctive actions (a consequence of the detection-part) and also ensure a high level of accuracy (a consequence of the approximation-part).

2.2.3 Optimization

Recently, researchers have proposed optimization-based techniques for keyframe selection. These techniques encode the selection of keyframes as a binary sequence, in which each component in that sequence corresponds to a frame of the animation. When a component in the binary sequence is one, its corresponding frame is considered to be a keyframe and, conversely, a value of zero corresponds to a non-keyframe. In order to derive the best sequence, and thus the obtain a selection of keyframes, these methods employ optimization algorithms to continuously modify the sequence until it minimizes, or sometimes maximizes, a goal that defined through an *objective function*. The objective function typically encodes a balance between accuracy and compression.

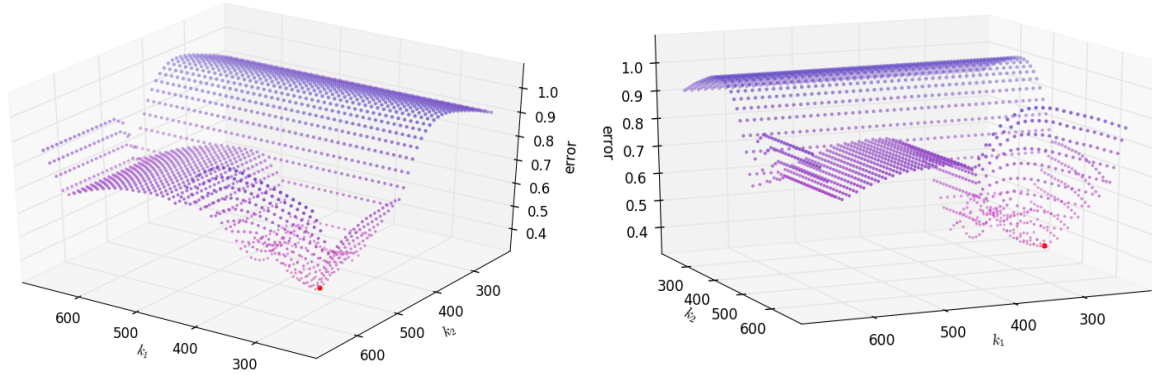
Liu et al. [52] proposed one of the first optimization-based techniques, in which they present an objective function that combines two terms relating to accuracy and compression. Specifically, the accuracy term is measured as the difference between the position, orientation, and velocity of each joint in the original animation and the linear interpolation of the keyframes. The compression term is measured as the ratio

between the number of frames in the motion against the number of keyframes in the selection. The objective function combines its two terms with a user-specified weighting parameter: increasing the parameter biases the technique toward favouring more accurate selections and, conversely, decreasing the parameter makes the technique favour more compressed selections. Thus, the motion editor can use the parameter to specify a balance between accuracy and compression that is suited to their editing task.

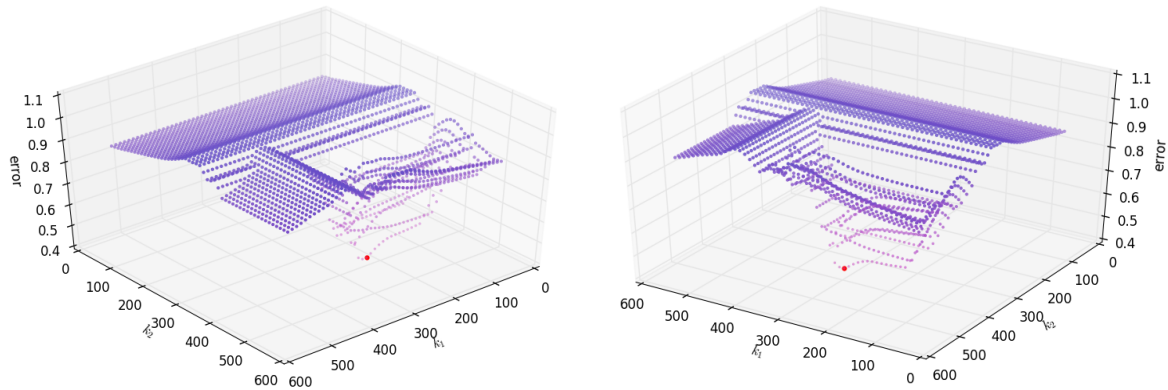
In order to optimize the objective function, Liu et al. employ a type of genetic algorithm. The algorithm begins by randomly generating many possible selections and sorts those selections based on the measure prescribed by their objective function. Next, a simplex method is applied to improve a subset of the most promising selections and then the crossover and mutation steps are applied to stochastically modify the selections (a standard process used in genetic algorithms). The algorithm iterates through these phases of sorting the selections, improving the best selections, and then mutating the selections until the algorithm either converges (cannot improve further) or stagnates (the algorithm has performed too many iterations and terminates without the optimal result).

The more recent research toward keyframe selection using optimization methods has diverged little from Liu et al.'s initial design. Zhang et al. revisit the technique using an alternative genetic algorithm and propose slight modifications to the objective function [81], while Zhang and Cao [82] and Chang [15] propose to employ particle swarm optimization algorithms in place of the genetic algorithm.

A drawback of these optimization-based techniques is that they rely on computationally demanding optimization methods. Liu et al. report that their implementation requires 18.4 seconds to select keyframes for a motion capture animation of 316 frames and 40 seconds for another of 685 frames [52, p. 92, Table 1]. Furthermore, the technique must be executed separately for every new selection and, as such, the motion editor cannot explore the balance between compression and accuracy without waiting for a new selection to be computed many times over. As the space through which the optimization algorithms must navigate to converge toward an optimal solution is highly non-linear space, as illustrated by Figure 2.8, the problem of expensive computations cannot be avoided (at least without extensive engineering efforts). Another drawback is that the optimization algorithms used to traverse the space, in this case genetic and particle swarm algorithms, are non-deterministic. A lack of determinism can result in a failure to produce the optimal solution and in such cases the motion editor may need to restart the algorithm, which only serves to exasperate the problem of the technique being computationally expensive.



(a) Cartwheel



(b) Lay Up

Figure 2.8: Here we present an illustration of the highly non-linear space that results from the problem of selecting keyframes in order to minimize a distance-based objective function. For this example we select only two keyframes: the axes k_1 and k_2 denote the choice of the two keyframes and the vertical axis, titled error, presents the sum of distances between the poses of the original animation and a uniform sampling of a linear interpolation through the selected keyframes. The space is complex because it contains several local minima and discontinuous error surfaces (note the shape of the surface when the keyframes switch from $k_1 < k_2$ to $k_1 > k_2$). The complexity of the space increases with both the number of frames in the animation and also the number of keyframes being selected (at least until the level of compression drops below 80%).

Despite their drawbacks in terms of computational costs, a distinct advantage of these optimization-based techniques is that their keyframes optimize the given objective function (at least when the optimization algorithm converges successfully). Providing the ability to produce optimal selections of keyframes is important because it provides the guarantee that the selected keyframes maximize their accuracy for the given level of compression.

2.2.4 Factorization

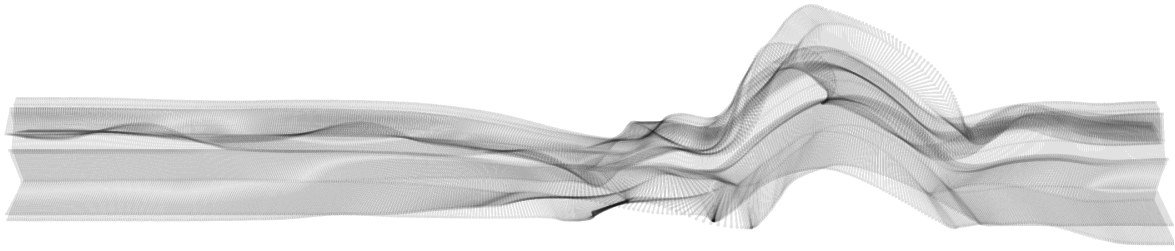
Factorization-based techniques select keyframes in order to form a linear basis that can be used to reconstruct the animation. In this way, factorization-based techniques interpret keyframes as the *blend shapes* of an animation and propose to choose keyframes in a way that any frame of the original animation can be reconstructed as weighted sum of the keyframes (in contrast, the other keyframe selection techniques treat keyframes as poses that are to be interpolated to reconstruct the animation).

We categorise two types of techniques as factorization-based:² *clustering* techniques first organize the frames of the animation into groups and then select a single keyframe from each group (illustrated in Figure 2.9); and (*matrix factorization*) techniques that first encode the animation into matrices and then employ decomposition algorithms to recover a new matrix containing the selected keyframes.

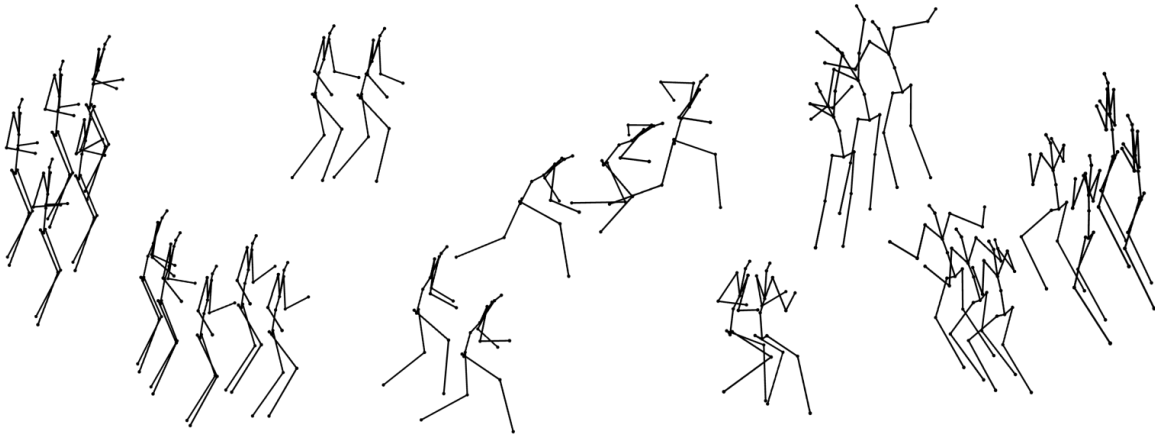
Clustering Park and Shin [58] proposed a clustering techniques. In their technique, the two most distant poses in the animation are used as an initial selection of keyframes. Next, the frames of the original animation are organized into groups, such that the poses in each group are most similar to one another. A new selection of keyframes is derived by selecting the frames whose poses are nearest to the average of those in its group. Finally, their technique approximates the original motion via a scattered data interpolation and selects the frame of the original animation that is furthest from the approximation as a new keyframe. This process of grouping, updating the selection of keyframes to the centroids of the groups, and then adding a new keyframe is repeated until the approximation of the motion is sufficiently similar to the original animation.

Matrix Factorization Huang et al. [36] proposed a matrix decomposition techniques for factorization. In their technique, a matrix is formed to represent the original animation, in which each frame of the animation correlates to a column of

²Much of the research on keyframe selection divides these works in two separate categories. We categorize these techniques together under the label of factorization since both of the clustering and the matrix factorization techniques that we surveyed interpret keyframes as factors of the reconstructed animation.



(a) Original Animation



(b) Encoding as Groups



(c) Factorization

Figure 2.9: In clustering, an algorithm first groups the frames of the animation (a) based on similarity of the shape of their poses (b). The groups may be reorganized multiple times by the algorithm, and eventually the algorithm will terminate. After termination, a keyframe is selected from each group (c).

that matrix. The animation matrix is then decomposed into a keyframe matrix and a weights matrix, the product of which approximates the original animation. As with Park and Shin’s technique, the frame furthest from the approximation is added as a new keyframe, forming a new column in the keyframe matrix. The approximation is updated by solving for the weights matrix and again the furthest frame is added as a new keyframe. This process of alternating between adding a keyframe and updating the weights matrix to best approximate the motion is continued until the original animation can be approximated well by the selected keyframes.

From the two types of techniques that we categorize under factorization, the clustering techniques are faster to compute and the matrix decomposition techniques tend to produce more accurate selections, but at the cost of needing to perform the more expensive matrix inversion operations (required to solve for the weights).

Unfortunately, factorization-based techniques cannot be applied to select keyframes for converting motion capture into animator-like keyframe animation since they interpret the animation as a blend, rather than an interpolation, of the poses of the keyframes. While not appropriate for application to our problem, the factorization-based techniques excel at compressing animation, since fewer keyframes are required to form an effective linear basis (especially in cyclic motions such as walking and running).

2.2.5 Discussion

In this section, we have provided a detailed review of the existing research on keyframe selection for motion capture animation. We identified four types of techniques:

- the detection-based techniques that select keyframes based on the occurrence of features in the motion,
- the approximation-based techniques that select keyframes as vertices of an approximation fitted to the motion,
- the optimization-based techniques that minimize an objective function that expresses both the level of accuracy and compression afforded by the selected keyframes, and
- the factorization-based techniques that interpret keyframes as factors of the motion.

In summary of the advantages from our perspective: detection-based techniques excel at coarsely dividing the motion into regions that contain distinctive actions;

approximation-based techniques offer the benefit of fast selection and explicit control over the level of compression; optimization-based techniques offer the only method for optimal keyframe selection; and finally factorization-based techniques enable the highest levels of compression while preserving the accuracy of the selected keyframes (enabled by their alternative interpretation of keyframes).

In summary of the disadvantages from our perspective: detection-based techniques cannot offer explicit control over the number of keyframes selected; approximation-based techniques are bound to produce suboptimal selections due to their greedy design; optimization-based techniques offer optimality at the cost of heavy computational demands; and factorization-based techniques cannot be applied to our keyframe selection problem.

From this review, we highlight four key limitations:

- **Lack of optimality.** Due to their greedy design, the approximation-based techniques cannot rearrange previously selected keyframes as a higher number of keyframes become available. As such, the denser selections of keyframes produced by these techniques tend to be suboptimal and therefore it would have been possible to use fewer keyframes to achieve a similar level of accuracy.
- **Lack of control of criteria.** None of the existing techniques offer a mechanism to express different criteria from which keyframes are selected. Some techniques offer weighting schemes that may be used to tweak which parts of a character’s motion contribute most significantly to the keyframe selection, but these are difficult to use as many dimensions of weights need to be configured.
- **Expensive computation.** Optimization-based techniques, the only type to offer optimal selection, require expensive computations that would halt a motion editor’s workflow while they wait for the technique to complete.
- **Lack of determinism.** The optimization-based techniques employ genetic algorithms that are non-deterministic: the set of initial selections tend to be randomly initialized and then stochastically improved. A lack of determinism is problematic because different selections of keyframes may be selected for two or more animations containing similar, or even identical, motions.

Given these limitations we propose that a new keyframe selection technique that can (1) select keyframes to optimize an objective function, (2) offer the ability to configure the criteria to which keyframes are selected, (3) offer fast computation, and (4) offer precise control over the level of compression would be a new and novel contribution to the topic of keyframe selection; we present such a technique through the remainder of this thesis.

Chapter 3

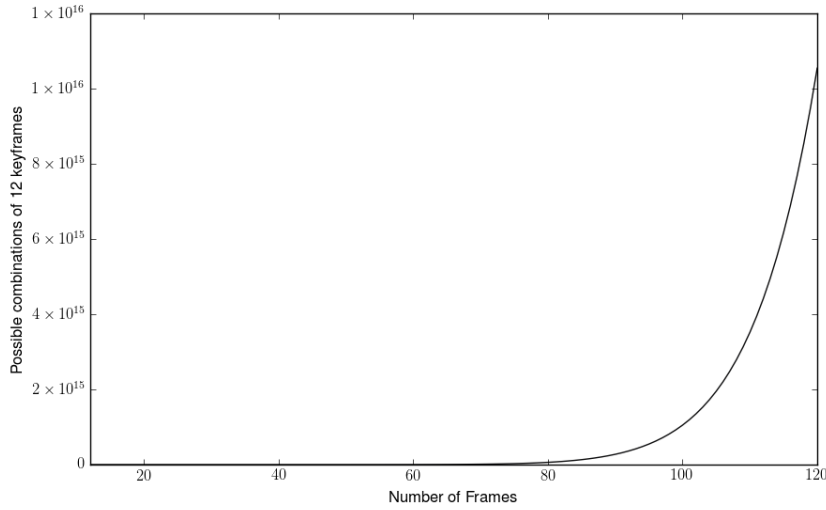
Optimal Keyframe Selection

The problem of selecting keyframes to minimize an objective function is generally a combinatoric problem. The combinatoric nature of the problem means that the number of possible selections of keyframes increases combinatorially as the size of an animation increases. Thus, employing a simple brute-force algorithm that compares all possible selections is not possible (see Figure 3.1).

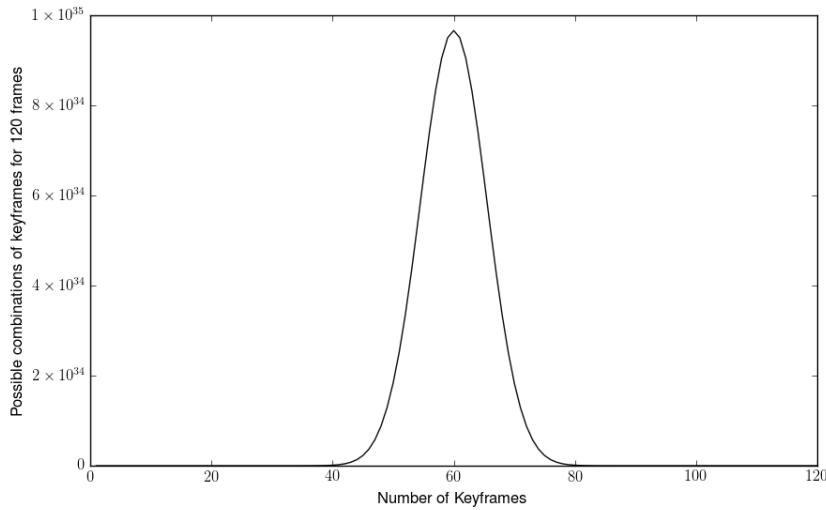
One approach to avoid the problem of needing to check all possible selections is to employ a method that can traverse through a space to find an optimal selection. As described in Section 2.2.3, the optimization-based keyframes techniques take this approach to enable optimal keyframe selection but, as a consequence of this approach, are computationally expensive due to the complexity of the space.

We propose a different technique that realizes a fast approach for creating optimal selections, in which we decompose the problem to reduce the number of computations required to derive an optimal selection. To do so we first represent the animation as a graph that can be divided into smaller graphs, in which the notion of a smaller graph correlates to a smaller part of the animation. Once we have divided the animation into these graphs, we begin the process of keyframe selection by calculating the optimal selection with respect to the smallest graph. We then reuse the selections calculated for the smallest graph to assist the computation of the optimal selections for next smallest graph. By continuing to reuse the selections from smaller graphs to compute those for larger graphs in this way, we are able to realize an optimal selection of keyframes for the animation without requiring extensive computational effort.

Our idea to apply this type of decomposition in the context of keyframe selection was inspired by a technique titled *Salient Points*, which is a technique for identifying sets of points that optimally approximate 2D shapes. In the design of their technique, Lewis and Anjyo [49] made the important realization that the optimal set of points for representing a shape contains within it the smaller optimal set of points for representing a part of that shape. When such a decomposition is present - the optimal solution to a



(a) Combinations as Frames Increase



(b) Combinations as Keyframes Increase

Figure 3.1: Selecting keyframes in order to minimize an objective function is a combinatoric problem, which means that the number of possible selections increases combinatorially with respect to the number of frames in the animation and binomially with respect to the number of keyframes being selected. Figure 3.1a illustrates the number of possible ways to select twelve keyframes as the number of frames in the given animation increases, while Figure 3.1b depicts the number of possible combinations of selecting an increasing number of keyframes for an animation containing 120 frames. To illustrate the severity of the combinatoric problem: it would take over 300 years to check every possible selection of 12 keyframes for 120 frames of animation if an algorithm were able to check one selection every microsecond (approximately $\binom{120}{12} = 1.0542859^{16}$ solutions are possible).

partial problem appears in the optimal solutions to the overall problem - the problem is said to have an *optimal substructure*. Due to the presence of an optimal substructure, Lewis and Anjyo were able to realize a fast and optimal technique. Similar to Lewis and Anjyo, we realize an optimal substructure for our keyframe selection problem through our graph representation.

This chapter is focused on describing Salient Poses in detail and demonstrating that it is fast and optimal. To describe Salient Poses we need to describe how we formulate the problem using a graph (Section 3.1.1), how we decompose that graph into smaller graphs to realize optimal substructure (Section 3.1.2), and how we can compute optimal selections of keyframes in practice (Section 3.1.3). Finally, we express precisely how much computation is required to generate the optimal keyframe selections for any animation (Section 3.1.4).

Once we have described our technique, we present an evaluation in two parts. First, we provide a comparison of how well the keyframes selected by our technique approximate six motion capture animations against those selected by other related techniques (Section 3.2.1). We also present execution times required for Salient Poses to generate all optimal selections for each of the six motion capture animations (Section 3.2.2). Together, these two results demonstrate that our technique is both fast and optimal.

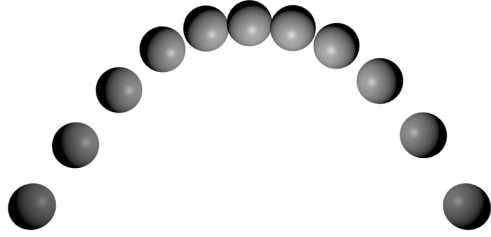
Later, in Chapter 4, we will apply Salient Poses to the problem of recovering blocked animation from motion capture, in which we are able to use Salient Poses to select keyframes that are similar in structure to the keyposes and breakdowns used by animators to create blocked animation.

3.1 Salient Poses

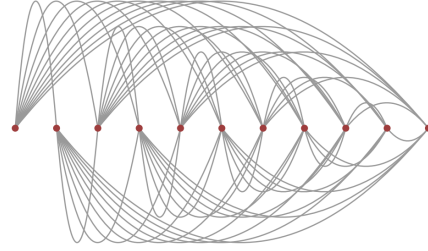
3.1.1 Representing the Animation as a Graph

As briefly described in the previous section, we represent the animation as a graph. In particular, a node is added to the graph for each frame in the animation and a directed edge is formed between that node and the nodes of all following frames (those that occur later in time). An illustration of the graph is provided by Figure 3.2b.

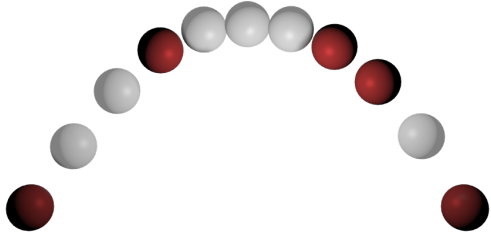
We express a selection of keyframes as a traversal through that graph. A traversal must start from the first node (the first frame of the animation) and end at the last node (the last frame). A traversal can pass through any number of nodes and must follow the direction of edges in the graph. Finally, the length of each directed edge is



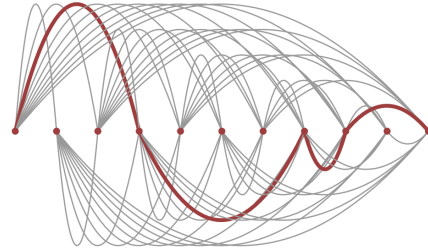
(a) Animation



(b) Graph



(c) Selection



(d) Traversal

Figure 3.2: We use a graph to represent the keyframe selection problem. Say we have the animation of the ball presented in Figure 3.2a, which contains 11 frames. We add a node to the graph for each of the animations frames, and furthermore, we add directed edges between each node and the nodes of the frames that occur later on in time (b). Finally, every traversal of the graph denotes a unique selection of keyframes. For example, Figure 3.2c depicts the keyframe selection containing frames $\{1, 4, 8, 9, 11\}$. The graph traversal for this selection is visualized in Figure 3.2d.

prescribed by an objective function given to Salient Poses.¹

The nodes that a traversal passes through are that traversal's keyframes and, consequently, a traversal represents a selection of keyframes that is inherently sorted and cannot contain the same keyframe twice. Furthermore, we say that the length of a traversal is the sum of the lengths of the directed edges travelled by the traversal.

With the notion of a traversal, we can restate the problem of selecting a given number of keyframes to best approximate an animation as the problem of choosing the shortest traversal of that many keyframes through the graph representing the animation.

Next in Section 3.1.2, we examine the optimal substructure of this graph interpretation and make an important observation that leads to a fast and optimal technique for creating traversals that minimize their edge length.

3.1.2 Partial Graphs provide Optimal Substructure

We now introduce the notion of *partial graphs*, which simply divide the graph representing the animation into smaller graphs. The smallest partial graph contains only three nodes (the first, second, and third frames). We refer to this as the *graph of three frames*. The next partial graph contains four frames (the first, second, third, and fourth frames). We refer to this as the *graph of four frames*. In a similar way, the *graph of 50 frames* contains the first 50 frames of the animation.

In order to show how our partial graphs lead to an optimal substructure, we begin this section with a contrived example. The example describes how the optimal traversals of smaller graphs must occur as part of the optimal traversal in some of the larger graphs. Once given, this example leads to an important insight for realizing an algorithm that requires significantly fewer computations required to create optimal traversals.

Consider an animation of just 10 frames and assume that the optimal traversal of four keyframes for that animation is $\{1, 3, 7, 10\}$. Since we know that the optimal traversal minimizes the length (given by the objective function), we can see that the optimal traversal of three nodes through the graph of 7 frames must be $\{1, 3, 7\}$.

To see why, assume temporarily that $\{1, 5, 7\}$ were to be the optimal traversal. If this were to be true, then it must be the case that the length of $\{1, 5, 7\}$ is less than

¹It should be noted that the objective function is purposefully left undefined for Salient Poses; any implementation of our objective function can be provided for use with Salient Poses. This is important because it means that Salient Poses can be applied to select keyframes for any criteria that can be expressed through its objective function. Later, in Chapter 4, we will present three specific objective functions that can be used with Salient Poses to realize the selection of animator-like keyframes.

$\{1, 3, 7\}$. We can express this less-than relation mathematically using a function o_* that describes the optimal length from some frame i to another frame j :

$$o_*(1, 5) + o_*(5, 7) < o_*(1, 3) + o_*(3, 7)$$

However by adding the length of the edge between frames seven and ten to both sides of the comparison (this cannot change which side is greater) we can see that we arrive at a contradiction:

$$o_*(1, 5) + o_*(5, 7) + o_*(7, 10) < o_*(1, 3) + o_*(3, 7) + o_*(7, 10)$$

The above expression is a contradiction because we know that $\{1, 3, 7, 10\}$ is an optimal traversal, and therefore its length must be less than that of $\{1, 5, 7, 10\}$. The same contradiction arises when the same argument is applied to any other choice of traversal through the graph of seven frames. As such, we know that $\{1, 3, 7\}$ must also be optimal.

The observation that optimal traversals to smaller graphs appear in the solutions to larger graphs is crucial to the design of Salient Poses: if we already know the optimal traversals for smaller graphs then we can reuse them to reduce the amount of computation required for calculating the optimal traversals for larger graphs.

As we will illustrate in the next section, reusing computations in this way enables optimal traversals to be calculated with significantly fewer computations.

3.1.3 Optimal Selection through Partial Graphs

With the observation that the optimal traversals through smaller graphs occur in larger graphs, we can propose a fast technique for computing optimal traversals. We employ an iterative technique to do so. The first iteration calculates the optimal traversals of three keyframes for all partial graphs. The second iteration calculates optimal traversals of four keyframes for all partial graphs and, importantly, borrows from the results of previous iteration to enable a faster computation. We continue with this iterative process, each iteration adding one keyframe to the traversal of each partial graph, until we arrive at the optimal traversals of the required number of keyframes for the full graph.²

²Note that the optimal traversals are formed between first frame of the animation and all other frames. If instead the optimal traversals between the last frames and all other frames are desired, we can apply the partial graph decomposition in the reverse (the graph of three frames contains only the last three frames, the graph of four frames contains only the last four frames, and so on).

We start from the smallest graph of three frames. This graph can only have one traversal: $\{1, 2, 3\}$. Therefore, we have already found the optimal traversal of three keyframes for this graph. The graph of four frames is more interesting, there are two possibilities traversals: $\{1, 2, 4\}$ and $\{1, 3, 4\}$. We compare the length of these two traversals, recording the optimal one and discarding the other. We continue to the next graph of five frames. This graph has three possible traversals: $\{1, 2, 5\}$, $\{1, 3, 5\}$, and $\{1, 4, 5\}$. We compare the traversals, store that which is optimal and discard the others. We continue this same approach for the following graphs of six, seven, eight, nine, and ten frames, and continue until all partial graphs have been assigned an optimal selection.

Continuing onto the second iteration, in which the goal is to calculate all optimal traversals of four keyframes, we begin from smallest possible graph of four frames that only has one possible traversal: $\{1, 2, 3, 4\}$. The next graph of five keyframes has three possible traversals: $\{1, 2, 3, 5\}$ and $\{1, 2, 4, 5\}$, and $\{1, 3, 4, 5\}$. Recall that in the first iteration we have already found the optimal traversal of three keyframes through both the smaller partial graphs (of three frames and of four frames). Since we have already calculated the optimal traversal we can reuse it and, therefore, we need only compare the traversals $\{1, x, 3, 5\}$ and $\{1, x, 4, 5\}$ (that is, the value for the second keyframe x can be recalled from the first iteration). Due to this abstraction over the second keyframe, we can simply iterate through the possible choices for the third keyframe in each partial graph to complete the second iteration.

To illustrate the abstraction further, Table 3.1 presents all traversals visited and compared in the second iteration for an animation of 10 frames. After the iteration, the optimal traversal for each partial graph is recorded and the others discarded (that is, one traversal is recorded for each row in the table). As further illustration, Table 3.2 displays the traversals that need to be visited for the third iteration, with the same abstraction over the value in both the second and third keyframes. In all situations, the values of x and y can be obtained by recalling optimal traversals that were computed in the previous iterations.

Finally, we need to describe how the final selection of keyframes can be composed. The solution is simple: we iterate the algorithm to compute the optimal traversals for all partial graphs until those traversals contain one less keyframe than what we need for the final selection. We then apply the same abstraction as used in each iteration, iterate through the possible choices for the second to last keyframe, and choose the partial traversal that features the shortest edge length.

We conclude our description of the algorithm next in Section 3.1.4, in which we express the number of traversals that we need to visit in order to find the optimal selection of keyframes for any animation.

$n_f = 4$	$\{1, x, 3, 4\}$						
$n_f = 5$	$\{1, x, 3, 5\}$	$\{1, x, 4, 5\}$					
$n_f = 6$	$\{1, x, 3, 6\}$	$\{1, x, 4, 6\}$	$\{1, x, 5, 6\}$				
$n_f = 7$	$\{1, x, 3, 7\}$	$\{1, x, 4, 7\}$	$\{1, x, 5, 7\}$	$\{1, x, 6, 7\}$			
$n_f = 8$	$\{1, x, 3, 8\}$	$\{1, x, 4, 8\}$	$\{1, x, 5, 8\}$	$\{1, x, 6, 8\}$	$\{1, x, 7, 8\}$		
$n_f = 9$	$\{1, x, 3, 9\}$	$\{1, x, 4, 9\}$	$\{1, x, 5, 9\}$	$\{1, x, 6, 9\}$	$\{1, x, 7, 9\}$	$\{1, x, 8, 9\}$	
$n_f = 10$	$\{1, x, 3, 10\}$	$\{1, x, 4, 10\}$	$\{1, x, 5, 10\}$	$\{1, x, 6, 10\}$	$\{1, x, 7, 10\}$	$\{1, x, 8, 10\}$	$\{1, x, 9, 10\}$

Table 3.1: All traversals visited in the second iteration of Salient Poses (selecting four keyframes) for an animation of 10 frames.

$n_f = 5$	$\{1, x, y, 4, 5\}$						
$n_f = 6$	$\{1, x, y, 4, 6\}$	$\{1, x, y, 5, 6\}$					
$n_f = 7$	$\{1, x, y, 4, 7\}$	$\{1, x, y, 5, 7\}$	$\{1, x, y, 6, 7\}$				
$n_f = 8$	$\{1, x, y, 4, 8\}$	$\{1, x, y, 5, 8\}$	$\{1, x, y, 6, 8\}$	$\{1, x, y, 7, 8\}$			
$n_f = 9$	$\{1, x, y, 4, 9\}$	$\{1, x, y, 5, 9\}$	$\{1, x, y, 6, 9\}$	$\{1, x, y, 7, 9\}$	$\{1, x, y, 8, 9\}$		
$n_f = 10$	$\{1, x, y, 4, 10\}$	$\{1, x, y, 5, 10\}$	$\{1, x, y, 6, 10\}$	$\{1, x, y, 7, 10\}$	$\{1, x, y, 8, 10\}$	$\{1, x, y, 9, 10\}$	

Table 3.2: All traversals visited in the third iteration of Salient Poses (selecting five keyframes) for an animation of 10 frames.

3.1.4 Number of Computations Required

Regardless of the iteration, the smallest partial graph - the graph whose number of frames is equal to the number of keyframes currently being selected - contains only one possible traversal once the abstraction has been applied. The next partial graph contains only two possible traversals (again once the abstraction has been applied). In a similar way, the third partial graph contains three possible traversals that each need to be visited. More generally, each successive graph contains one more possible traversal than the previous.

In our implementation of Salient Poses, we iterate through a list of possible traversals for each partial graph and record the one with the least length as optimal and discard the rest. For simplicity, assume that our technique demands one unit of computation to visit each traversal. As such, we can simply express the number of computations required for an iteration of Salient Poses as the total number of traversals that need to be visited. Using n_f to denote the number of frames and n_k to denote the number of keyframes, we can express the total number of traversals visited in an iteration as:

$$\sum_{i=n_k}^{n_f} (i - n_k + 1)$$

Table 3.3 provides some examples of the number of traversals visited for different

n_k	3	4	5	6	7	8	9	10	11	12
$n_f = 3$	1									
$n_f = 4$	2	1								
$n_f = 5$	3	2	1							
$n_f = 6$	4	3	2	1						
$n_f = 7$	5	4	3	2	1					
$n_f = 8$	6	5	4	3	2	1				
$n_f = 9$	7	6	5	4	3	2	1			
$n_f = 10$	8	7	6	5	4	3	2	1		
$n_f = 11$	9	8	7	6	5	4	3	2	1	
$n_f = 12$	10	9	8	7	6	5	4	3	2	1
Σ	55	45	36	28	21	15	10	6	3	1

Table 3.3: Examples of how many traversals are visited by Salient Poses to compute the optimal solution for each partial graph. Each row in the table represents a partial graph and each column in the table represents an iteration. For example, the cell where $n_f = 10$ and $n_k = 5$ describes that the partial graph of 10 frames features six possible traversals of five keyframes once the abstraction has been applied.

iterations. We can also express the total number of traversals that need to be visited to complete all iterations required to create the optimal traversal of a given number of keyframes for an animation of any size (the variable N denotes the total number of keyframes to be selected):

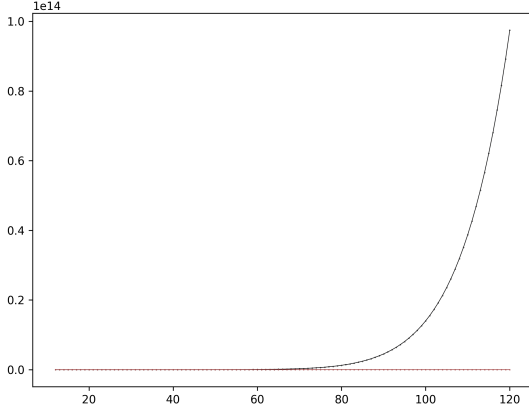
$$\sum_{n_k=3}^N \left(\sum_{i=n_k}^{n_f} (i - n_k + 1) \right)$$

Finally, Table 3.4 provides some examples of how many traversals need to be visited to complete all iterations.

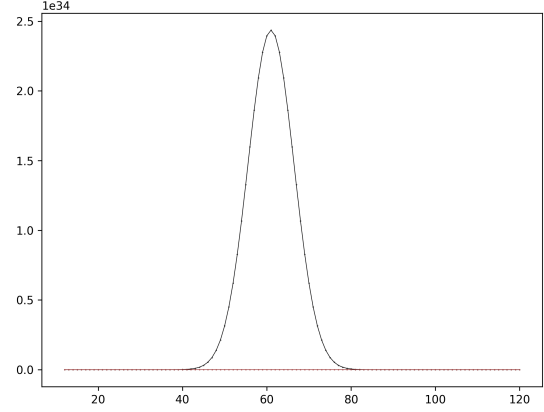
Finally, to prove that our this technique is able to significantly reduce the numbers of computations required to obtain the optimal solution, Figure 3.3 depicts the number of possible traversals for the given keyframe selection problem against the number of traversals that are visited using our technique. In conclusion, we are able to realize a fast technique for computing selections due to the significantly reduced number of traversals that need to be visited in order to provide the optimal solution.

We conclude this section with two remarks:

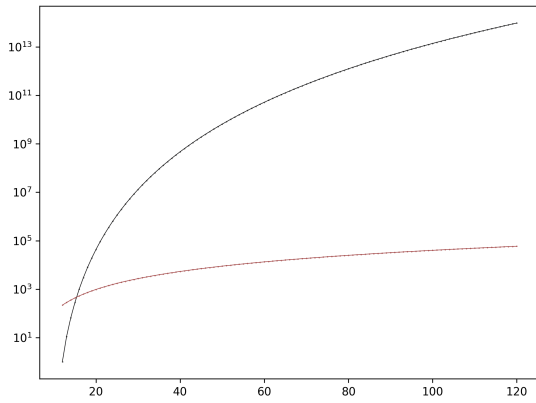
Selection Space. If we simply continue iterating Salient Poses until the largest possible number of keyframes have been computed, then the optimal keyframe



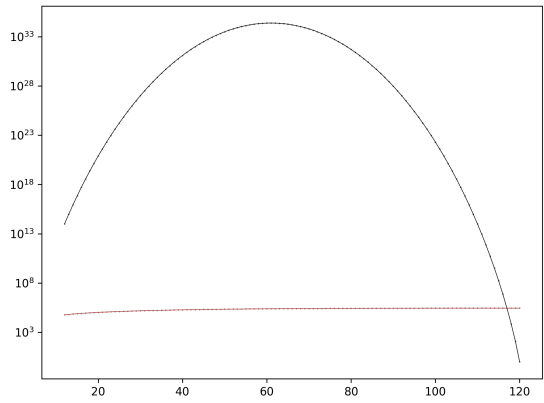
(a) Frames Increase



(b) Keyframes Increase



(c) Frames Increase



(d) Keyframes Increase

Figure 3.3: This figure builds upon Figure 3.1 and compares the number of possible traversals (black line) for a given keyframe selection problem against the number of traversals that need to be visited by Salient Poses to produce an optimal selection (red line). Figure 3.3a depicts how the number of traversals changes with respect to the problem of selecting 12 keyframes as the number of frames in an animation increases from 12 up to 120. In a similar way, Figure 3.3b depicts how the number of traversals changes with respect to the problem of selecting between 3 and 120 keyframes in an animation of 120 frames. Figure 3.3c and Figure 3.3d present the same figures, but using a logarithmic scale instead.

n_k	3	4	5	6	7	8	9	10	11	12
$n_f = 3$	1									
$n_f = 4$	2	4								
$n_f = 5$	3	8	10							
$n_f = 6$	4	13	18	20						
$n_f = 7$	5	19	28	33	35					
$n_f = 8$	6	26	40	49	54	56				
$n_f = 9$	7	34	54	68	77	82	84			
$n_f = 10$	8	43	70	90	104	113	118	120		
$n_f = 11$	9	53	88	115	135	149	158	163	165	
$n_f = 12$	10	64	108	143	170	190	204	213	218	220

Table 3.4: Examples of how many traversals are visited by Salient Poses to complete all iterations. Each row in the table represents an animation containing the specified number of frames. Each column in the table represents the problem of selecting a given of keyframes. For example, the cell where $n_f = 12$ and $n_k = 6$ describes that Salient Poses needs to visit 190 traversals to obtain the optimal selection of six keyframes for an animation of 12 frames.

selection for any level of compression (any number of keyframes) is already available among the optimal selections already computed via the iterations. We henceforth use the term *selection space* to denote a collection of keyframe selections, in which there is one selection for a range of levels of compression (that is, for each possible number of keyframes).

Number Sequences. Another interesting insight is that the number of traversals required to complete an iteration is expressed by the triangle number sequence - the third diagonal of Pascal’s triangle (Table 3.5). Furthermore, the number of traversals required to form the the entire selection space (one selection for all possible levels of compression) can be expressed exactly by the tetrahedral number sequence - the fourth diagonal of Pascal’s triangle.

3.2 Results

In this section, we present results with two objectives. The first objective is to demonstrate that Salient Poses always produces selections that are better than, or otherwise equal to, those of other related keyframe selection techniques. In Section 3.2.1 we present results to meet this objective, with a comparison between

each limb (from shoulder to wrist and hip to ankle). We exported the positional data into the JSON file format. Among these animations’ curves many different examples of large, medium, and finer scales of detail can be found.

A reference implementation, used to create the results presented in this section, can be viewed online ³.

3.2.1 Comparison to Related Techniques

In this section we examine the selection space given by Salient Poses and compare it with the selection space given by four other keyframes selection techniques. We visualize the selection space of each algorithm using a plot of the level of accuracy afforded by each selection against its level of compression. Figure 3.4 illustrates how accuracy is measured and Figure 3.5 provides an example of the visualization.

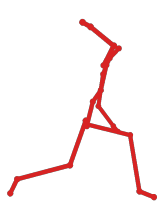
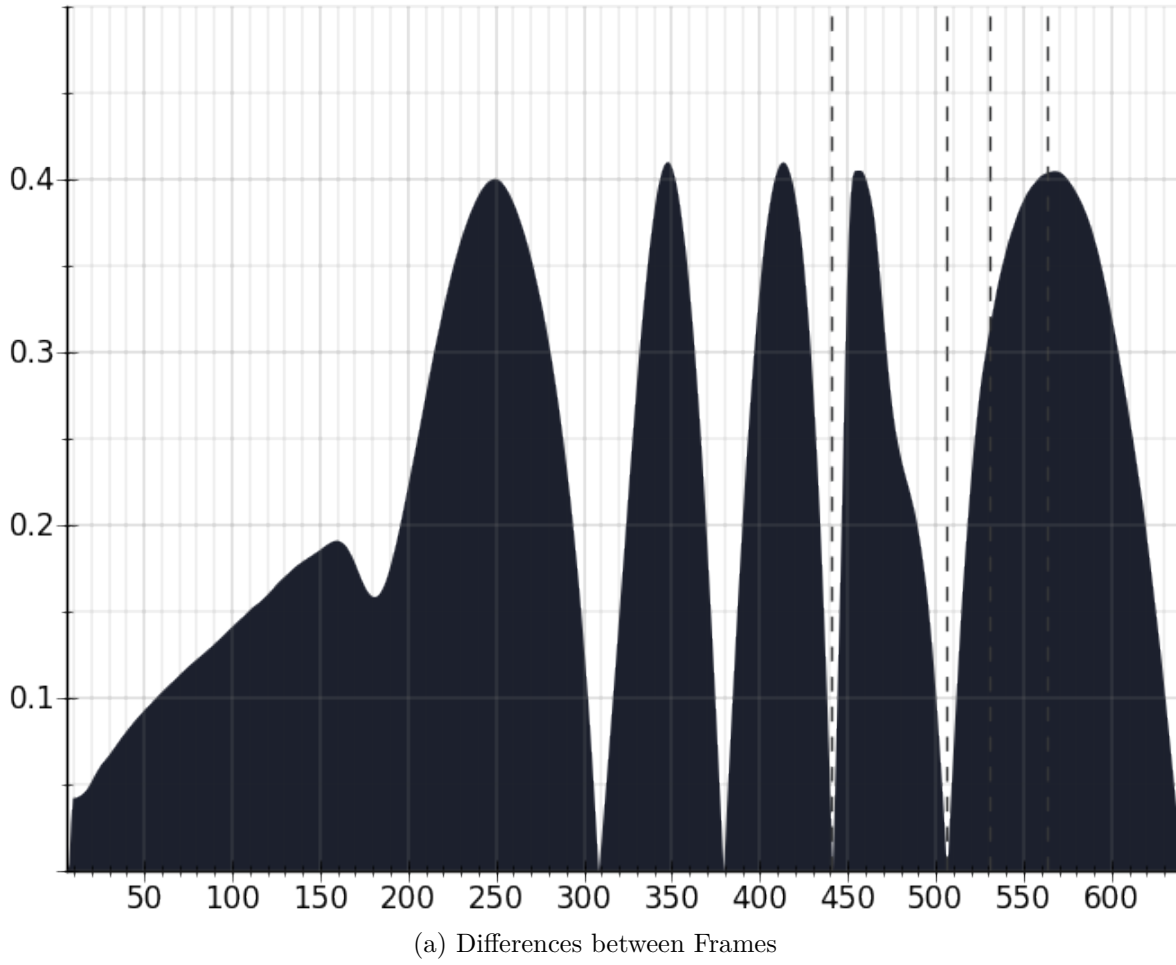
Note that our visualization of the selection space only displays selections that feature a level of compression (between 80% and 100%). As our motion capture examples were recorded at 120 frames per second, and keyframe animation typically is created with 24 frames per second, an 80% compression of the motion capture roughly translates to 24 keyframes selected per second - thus, 80% compression is roughly equivalent to selecting a keyframe on every frame. Since it is unlikely that more than 24 keyframes would be useful for any editing task, we focus our comparison on the selections with 80% compression or more.

We compare Salient Poses to four other algorithms:

- Lim and Thalmann’s approximation-based technique [50],
- Togawa and Okuda’s approximation-based technique [70],
- an adapted implementation of Liu et al.’s optimization-based technique [52], and
- an adapted implementation of Park and Shin’s factorization-based technique [58].

As discussed in Section 2.2.5, Salient Poses is designed to be an optimal interpretation of the approximation-based techniques. We compare to the approximation-based techniques to demonstrate that our optimal guarantee significantly improves the accuracy afforded by selection space. While more recent techniques have borrowed Lim and Thalmann’s technique [57, 55], no significant changes to the technique have been proposed. Thus, we propose Lim and Thalmann’s [50] and Togawa and Okuda’s [70] techniques remain indicative of

³See the SELECTOR tool provided by the MOCAPPIE reference implementation: <http://github.com/richard-roberts/PhD>.



(b) Frame 441



(c) Frame 506



(d) Frame 531



(e) Frame 563

Figure 3.4: The accuracy afforded by a given selection is prescribed by Salient Poses’s objective function. For example, suppose the objective function measures the average distance between the original motion and the interpolation of the selected keyframes. Figure 3.4a displays this measure (vertical axis) as calculated for each frame in an example animation (horizontal axis). Figures 3.4b to 3.4e present some examples to illustrate how the measure corresponds to differences in pose between the original motion and a linear interpolation of the keyframes (the vertical dashed lines in Figure 3.4a correspond to pairs of poses in Figures 3.4b to 3.4e). For this particular objective function, the accuracy for the entire animation is calculated as the sum of the values in Figure 3.4a.

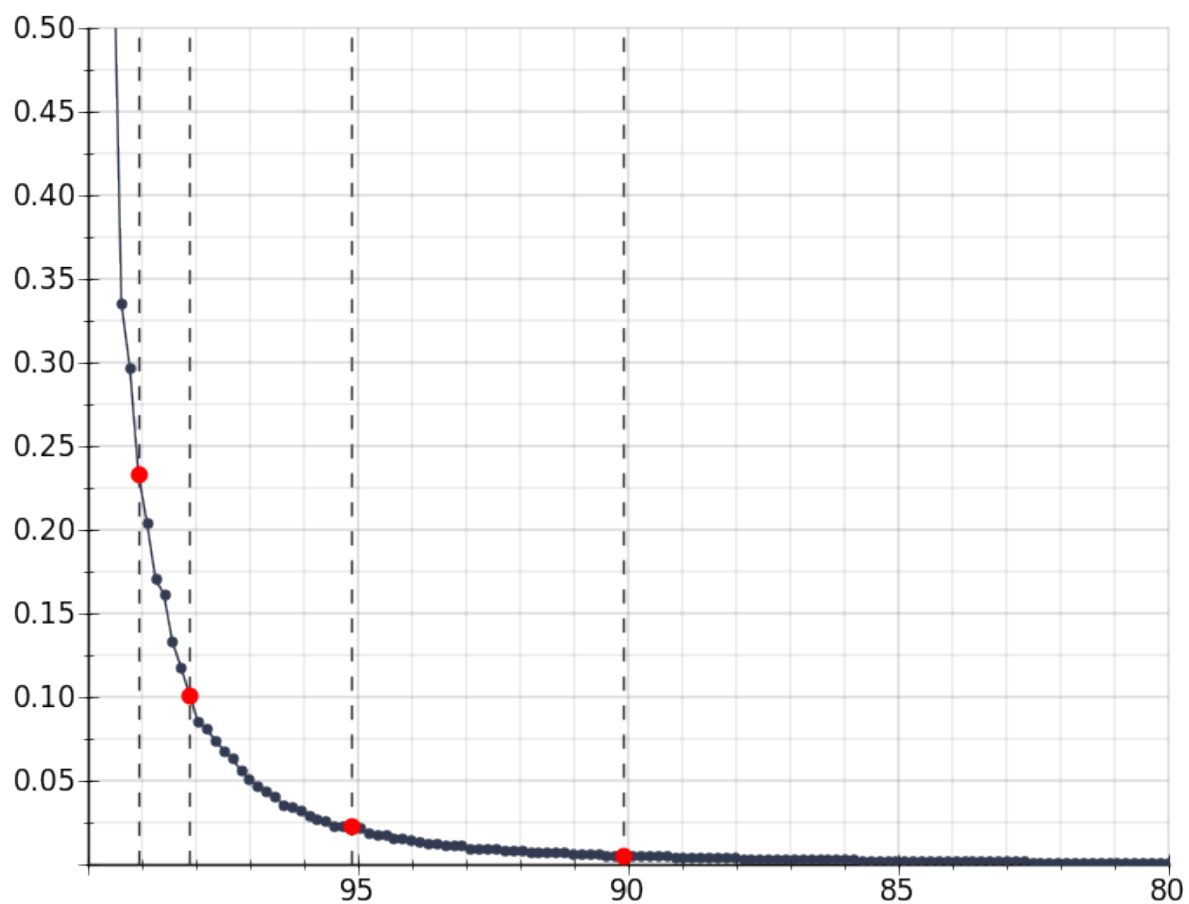


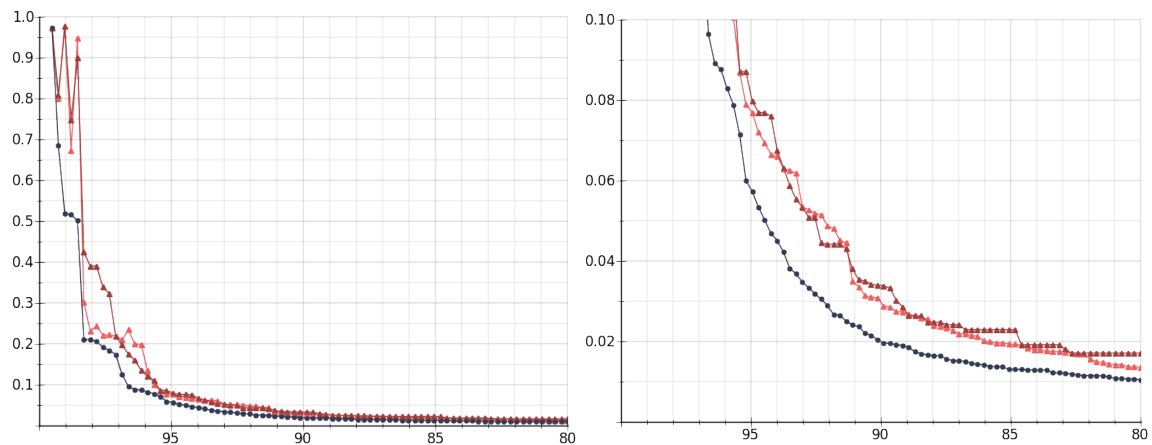
Figure 3.5: A visualization of the selection space. Each point corresponds to a particular selection. The horizontal axis depicts the level of compression, with higher compression toward the left and lower compression toward to right. Finally, the vertical axis depicts the measure of accuracy, as described in Figure 3.4.

approximation-based techniques. Finally, as these techniques employ a greedy selection process, we can form the selection space simply by recording the selection that results from each iteration.

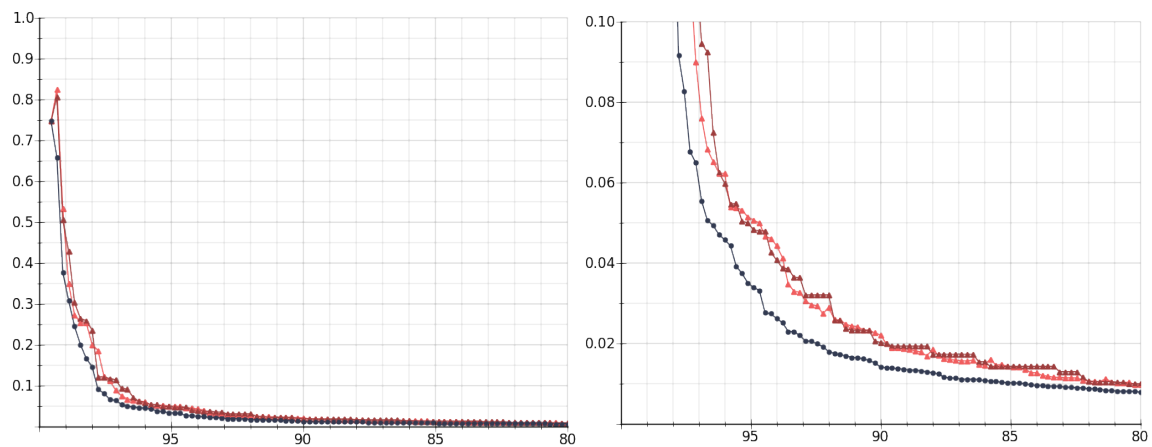
It is also important that we compare to optimization-based techniques. To support the claim that our algorithm is optimal, we expect similar results between Salient Poses and the optimization-based technique in terms of accuracy (except in situations when the optimization-based technique fails to converge to the optimal selection). Unfortunately it is difficult to obtain a selection space using the optimization-based techniques, due to their use of the binary sequence which does not allow explicit control over the number of keyframes being selected. Instead of comparing directly to Liu et al.’s [52] technique, we instead use another algorithm that we have designed as an adapted version of their technique. Our implementation replaces the binary sequence with an integer sequence. The integer sequence contains the same number of components as there are keyframes (each component denotes a frame number that is to be considered as a keyframe). We can restrict the number of keyframes in the resulting selection by setting the length of the integer sequence as a constant, and as such we are able to form the selection space by executing the algorithm for all levels of compression (one for each possible number of keyframes). Finally, for the optimization method we employ a simple genetic algorithm that uses the typical steps of crossover and mutation to optimize selections.

It should be noted that replacing the binary sequence with an integer sequence has the consequence of increasing the complexity of the optimization space, which means that our adapted version is more prone to failing than Liu et al.’s algorithm. To circumvent this problem, we allow the algorithm to execute 10 times for every selection and only include the best of these ten selections in the final selection space.

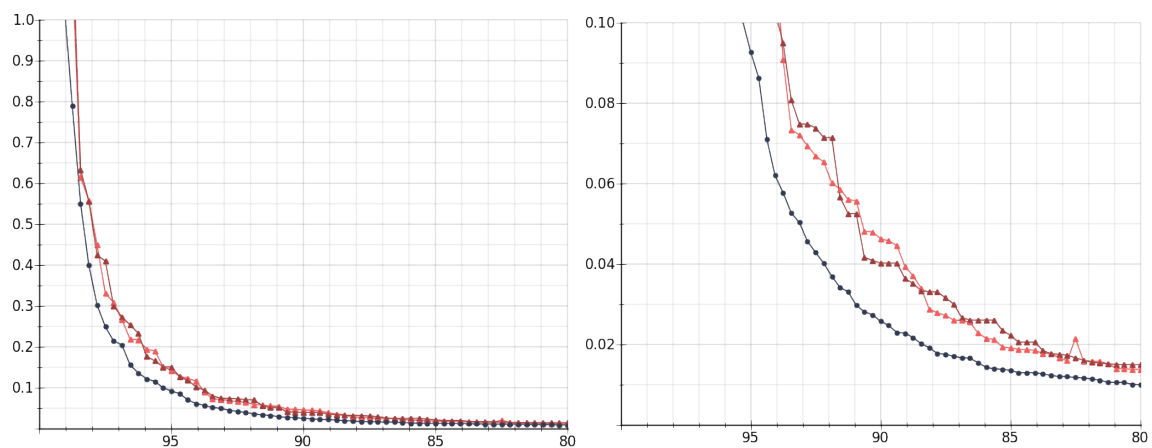
Finally, we also compare to an adaption of Park and Shin’s [58] technique. Our implementation, a k-means algorithm, begins with a random selection of keyframes. The frames of the animation are then divided into groups based on their similarity to the currently selected keyframes. Each of the selected keyframes are then updated to the frame of each group whose pose is nearest the average of its group’s poses. The algorithm continues to alternate between dividing the frames into groups based on the currently selected keyframes and then updating the keyframes based on the groups for a total of 100 iterations. We execute this factorization-based technique for each number of keyframes. We also allow this algorithm to execute 10 times for every selection and include only the most accurate of the 10 selections in the final selection space.



(a) Walk

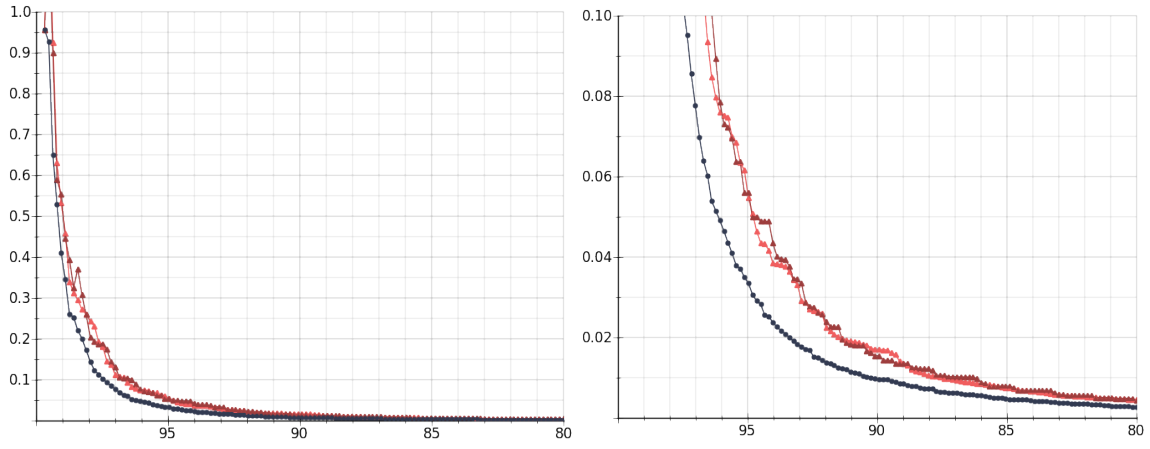


(b) Jump

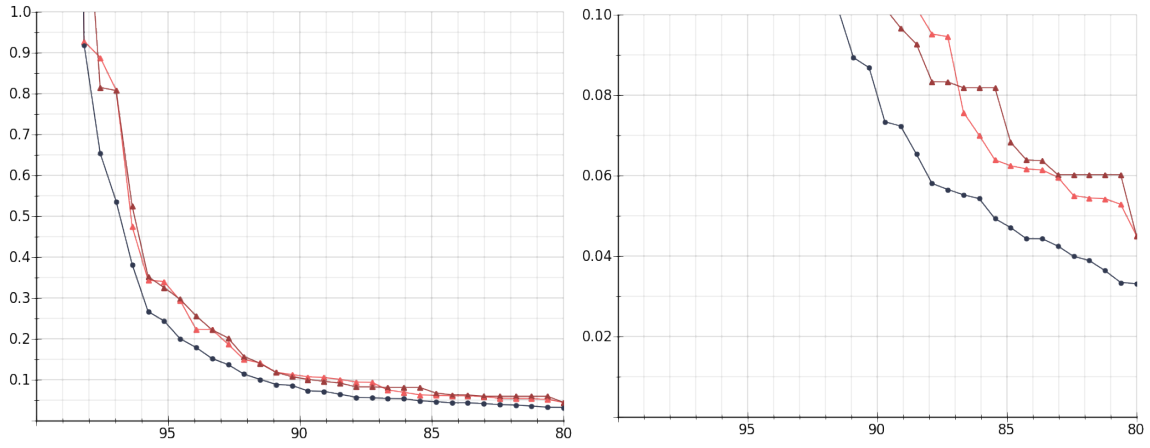


(c) Pitch

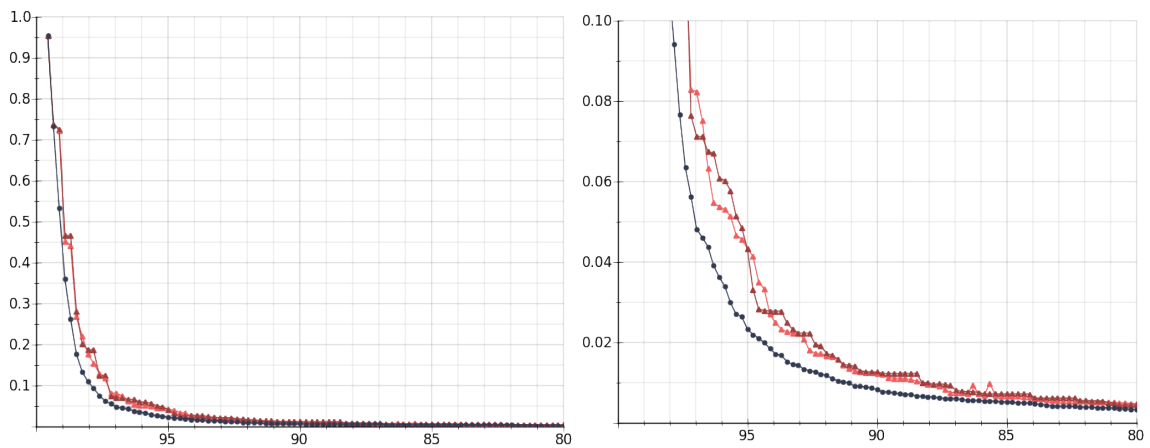
Figure 3.6: The selection space of Salient Poses (black line) visualized against that of Lim and Thalmann's (dark red line) and Togawa and Okuda's (light red line) techniques, for each of the six motion capture examples. A zoomed-in view of the selection space is depicted on the right-side.



(d) Run & Turn



(e) Cartwheel



(f) Lay Up

Figure 3.6: continued.

Comparison to Approximation

Both the approximation-based techniques quantify accuracy of their selection in terms of approximation error (less error is better). The approximation error is calculated as the maximum perpendicular Euclidean distance between the curves of the original animation and a linear interpolation of the keyframes for each of those curves. To favour a fair comparison, we select keyframes with Salient Poses using an implementation of the objective function that defines edge length in the same way (the maximum perpendicular Euclidean distance between the original animation and the linear interpolation).

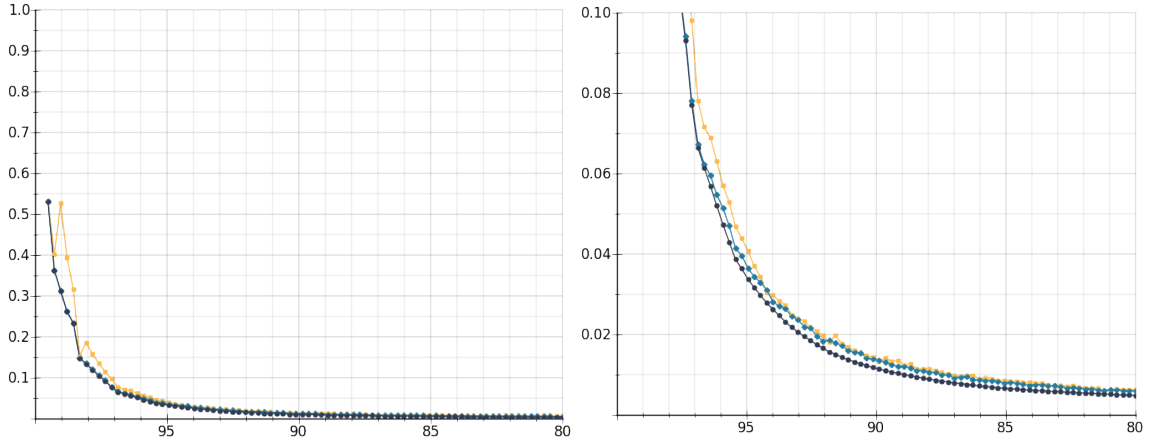
Figure 3.6 compares the selection space of Salient Poses against that of the approximation-based algorithms. The vertical axis displays the maximum perpendicular Euclidean distance (the distance has been normalized by the largest Euclidean distance between any two poses in the original animation). The accuracy of the selection spaces of the two approximation-based algorithms appears similar in all cases. The accuracy of Salient Poses’s selection space features a lower approximation error in all cases. Thus, we can observe that Salient Poses produces better or equal selections in all cases.

In Figure 3.6a, an interesting observation can be made regarding the visualization of the selection space belonging to Salient Poses. The selection space features two sharp drops in approximation error, the first occurring at approximately 99% compression and the other two at 98.2%. As the walk animation is cyclic, a sharp drop in reduction occurs as soon as enough keyframes become available to represent a level of detail in the cycle effectively. Since the presence of these drops indicates a significant increase in accuracy for only a small reduction in compression, the selections corresponding to these drops are more likely to feature an effective balance between accuracy and compression - an ideal for candidate for editing purposes. Later, in Section 4.3.4, we describe a simple threshold-based approach for automatically choosing a selection from the selection space that uses the notion of these sharp drops.

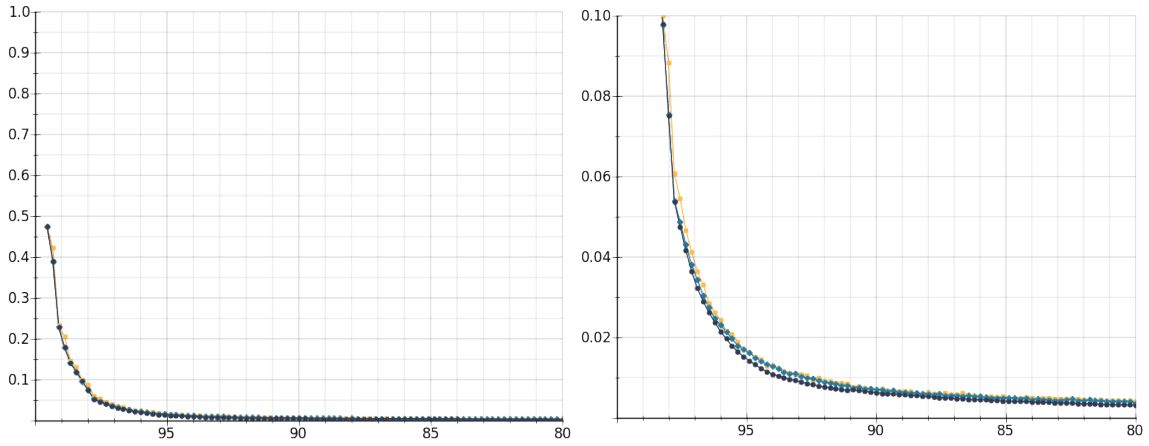
Comparison to Optimization and Factorization

Our implementation of the optimization-based technique measures accuracy as the sum of squared Euclidean distances between the curves of the original animation and the linear interpolation of the keyframes for each of those curves. To favour a fair comparison, we use the same objective function for both the optimization-based technique and Salient Poses.

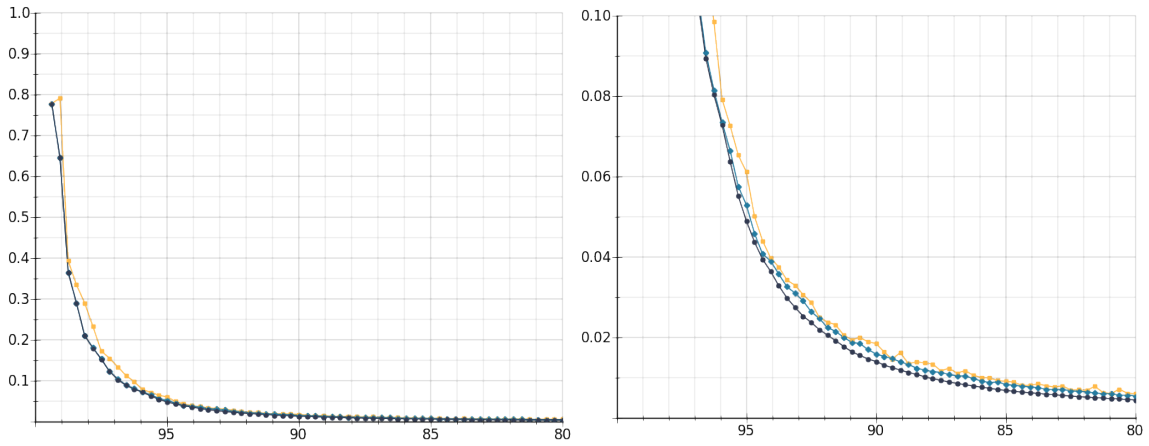
Figure 3.7 compares the selection space of Salient Poses against that of the optimization-based and factorization-based algorithms. The vertical axis denotes



(a) Walk

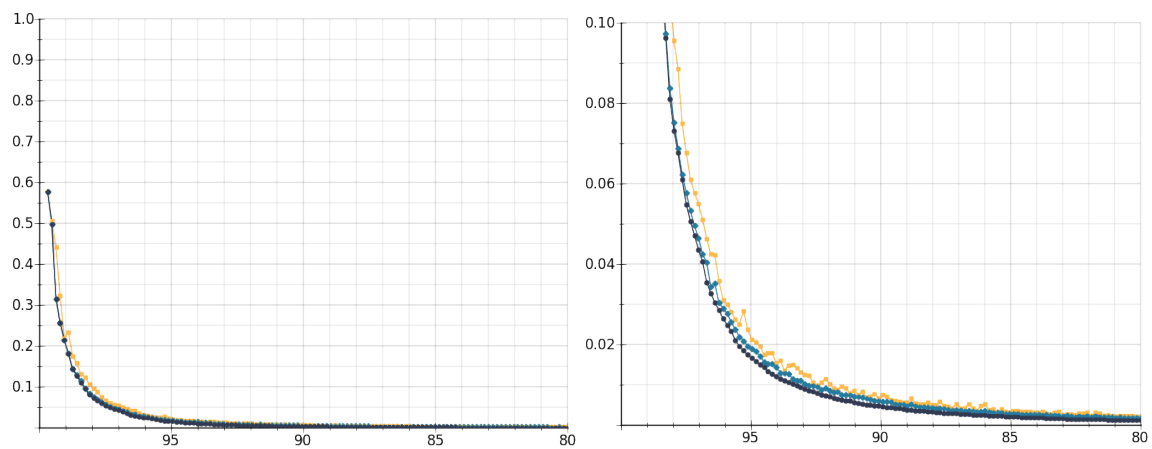


(b) Jump

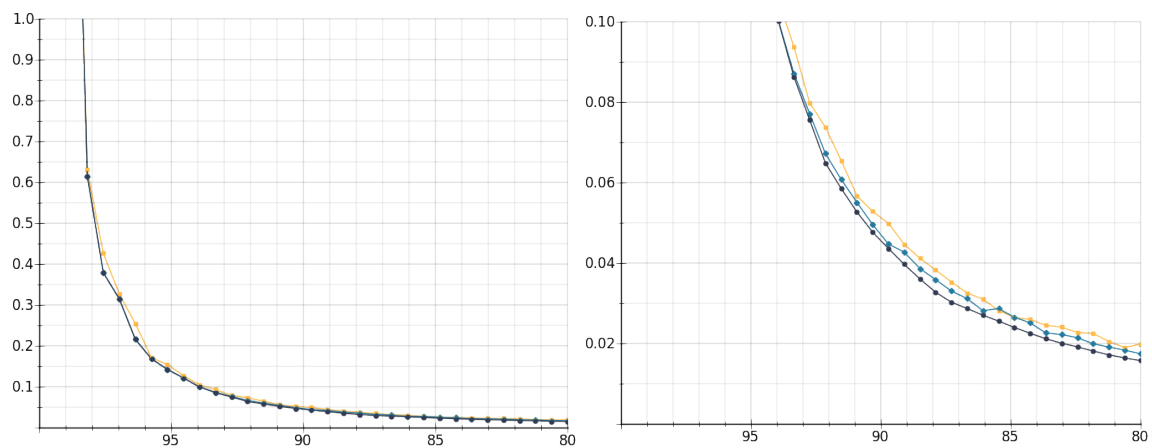


(c) Pitch

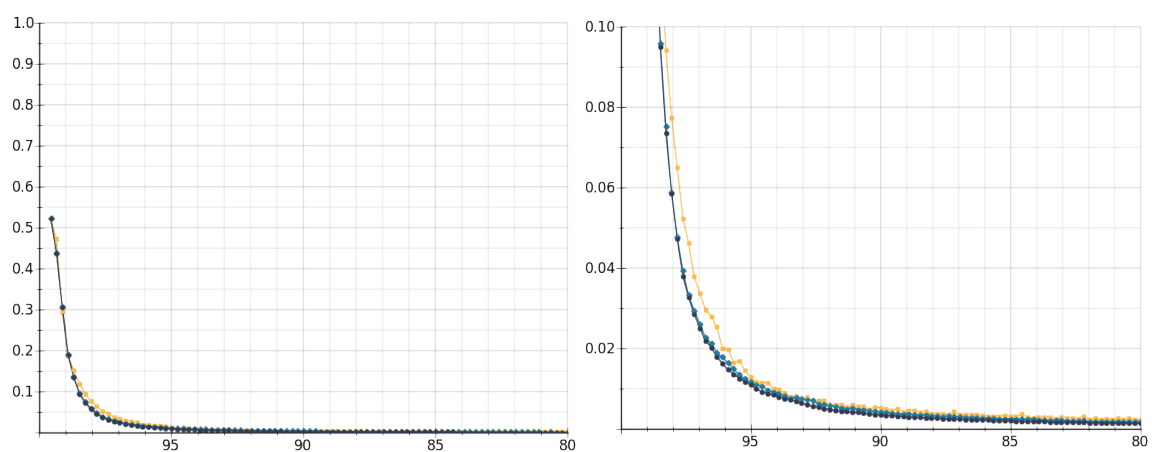
Figure 3.7: The selection space produced by Salient Poses (black line) compared to the selection space produced by the optimization-based (blue line) and factorization-based (yellow line) algorithms. A zoomed-in view of the selection space is depicted on the right-side.



(d) Run & Turn



(e) Cartwheel



(f) Lay Up

Figure 3.7: continued.

approximation error (again normalized by the largest Euclidean distance between any two poses of the original animation). Again, in all cases Salient Poses was able to produce selections that were better or equal to the other techniques.

In many of the cases it appears that Salient Poses produces selections that feature slightly less approximation error than the optimization-based technique, which is due to the optimization-based technique converging on non-optimal selections. However, for many of the selections the results produced by Salient Poses and these two techniques had a virtually equal approximation error.

In Figure 3.7d and Figure 3.7f, the reduction in approximation error as the level of compression reduces is more noisy than the other techniques. The noisiness further illustrates that the optimization space for keyframe selection is complex (highly non-linear).

3.2.2 Execution Times for Keyframe Selection

In order to demonstrate that our technique is fast, we present the execution times required to calculate the selection space for each of the six motion capture animations. The results presented in this section are measured from our implementation of Salient Poses running on the INTEL CORE i5 processor (2.9 GHz). We developed the implementation in the C++ programming language and compiled it the APPLE LLVM 8.1 compiler with all optimizations enabled. Finally, note that the results presented here do not include the computation of edge lengths.⁴

Figure 3.8 presents the execution time required to complete each iteration of Salient Poses for each of the six motion capture examples. The downward trend in computation time occurs in further iterations because each successive iteration requires visiting fewer traversals (one less partial graph in each case, and one less solution per partial graph).

Figure 3.9 depicts how the execution time increases as the number of frames in the animation increases. To produce these results, we applied Salient Poses ten times for each number of frames and averaged the measurements of execution time. Most importantly, these results demonstrate that Salient Poses can produce the entire selection space for an animation of 600 frames in one second (averaged at 1.1 seconds). Furthermore, the range of selections that would be of most interest for

⁴The length of the edges in the graph formulation used by Salient Poses need only be computed once for each animation. Thus, we applied each of our objective functions to prescribe edge length only once after obtaining each of the six motion capture animations and stored the measurements to a JSON file. The execution time required for these calculations is dependent on the design of each objective function and, therefore, is not relevant to the comparisons presented in this section. Furthermore, the calculations of the edge lengths are independent from one another and as such parallel processing can be used to enable mitigate much of the computation required for this edge-length step.

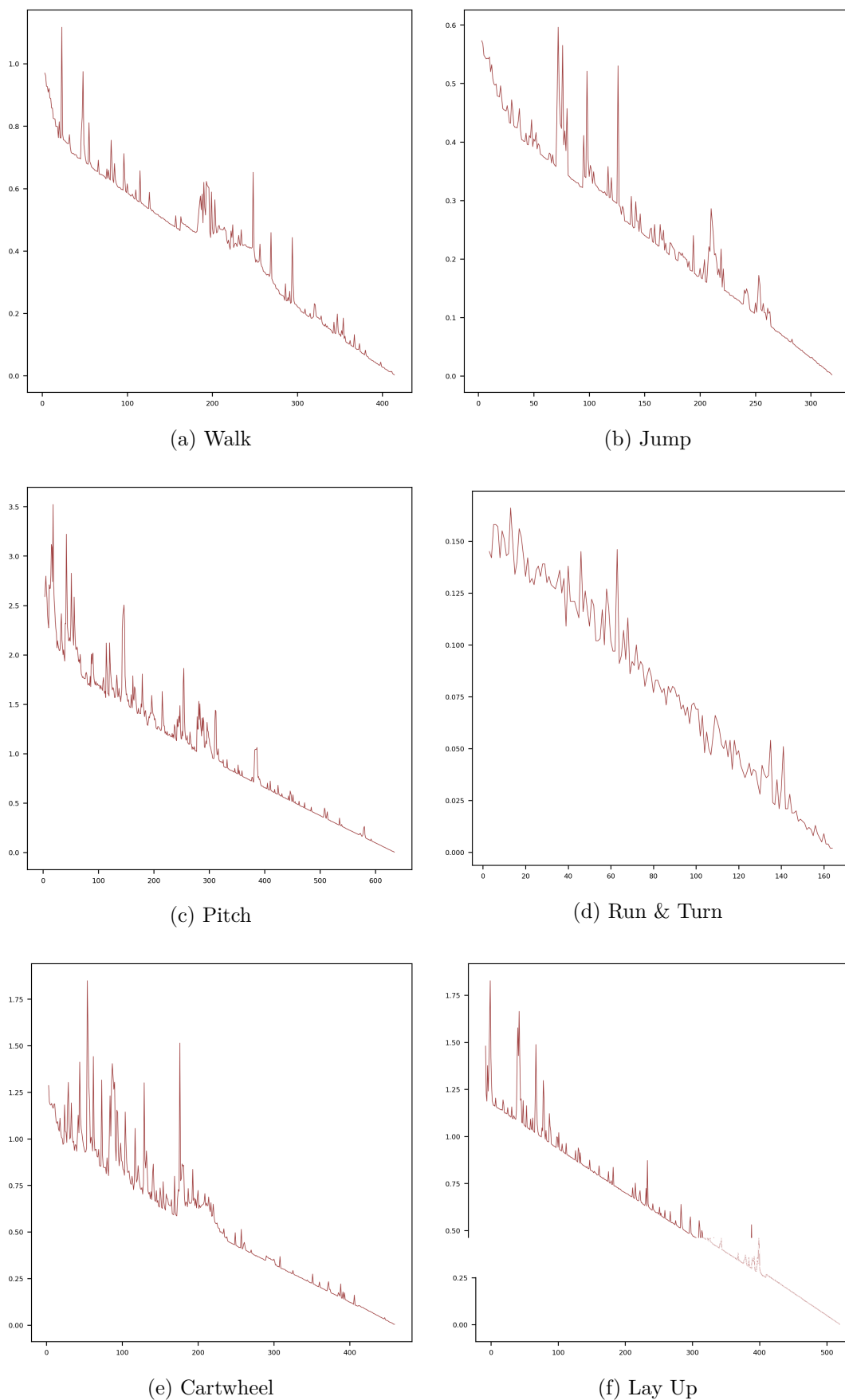


Figure 3.8: Graphs that depict the time required to compute the optimal selections for each iteration. The horizontal axis of denotes the number of keyframes being selected (equivalent to the number of iteration completed) and the vertical axis denotes time in milliseconds.

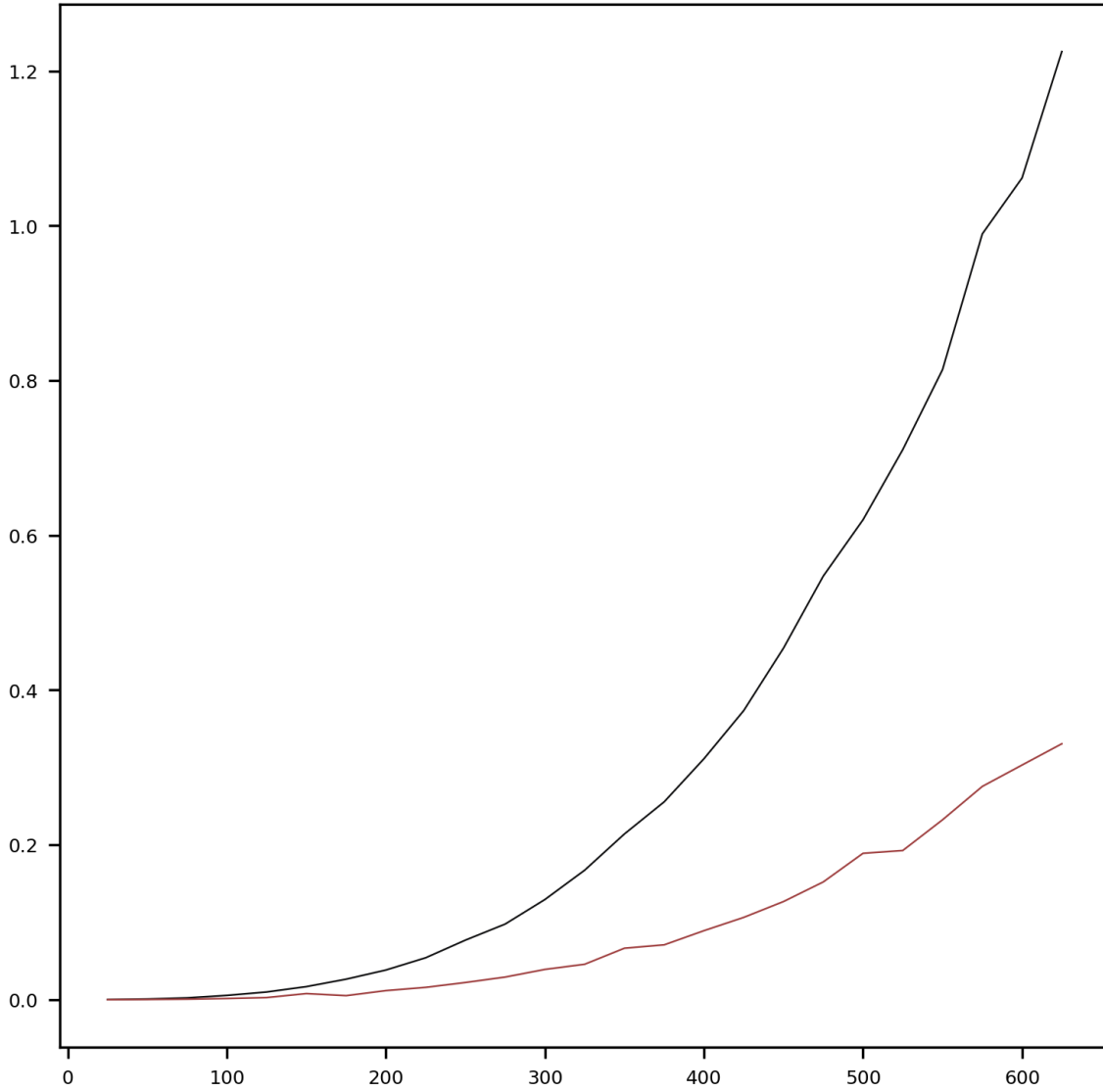


Figure 3.9: The time required to compute keyframes as the number of frames increases (in intervals of 25 frames). The black line depicts execution time that we observed when computing the entire selection space and the red line depicts the execution time that we observed when computing only the selections featuring 80-100% compression.

editing purposes (those with 80-100% compression) can be obtained in a fraction of a second (averaged at 0.3 seconds).

The only other techniques that can produce optimal selections of keyframes are the optimization-based methods. As described earlier in Section 2.2.3, Liu et al. reported execution times between 20 and 40 seconds (for 316 and 685 frames respectively). Furthermore, these times are with respect to a single selection (and not an entire selection space as produced by Salient Poses).

3.3 Summary

In this chapter we have presented Salient Poses: the first fast, optimal, generic, and deterministic keyframe selection technique. In order to enable fast and optimal selection, we introduced a graph that can be used to represent the space of possible keyframe selections. Most importantly, the graph can be decomposed in a way that features an optimal substructure and therefore allows Salient Poses to first compute optimal selections for smaller graphs and reuse them in the computation of optimal selections for larger graphs. Through the decomposition, we were able to offer fast computation while preserving our optimal guarantee.

Through the results presented in Section 3.2 we have demonstrated that Salient Poses can produce an entire space of optimal selections (in which all selections are better or equal to those produced by other techniques, at least in terms of the criteria expressed by the objective function), while retaining the ability to terminate at near-interactive speeds.

Fundamentally, Salient Poses is a dynamic programming technique for finding optimal traversals through two points in a graph. Since dynamic programming is included in the curriculum of most computer science and software engineering programs, an implementation of Salient Poses should be accessible to not only larger visual effects studios but also to individual researchers and even hobbyists.

One drawback of our keyframe selection technique is that the number of computations for forming the optimal selection space begins to increase quickly with respect to the number of frames in the animation (this can be extrapolated from Figure 3.9). As such, Salient Poses eventually incurs higher computational costs when applied to larger problems. We discuss why this is not a significant problem later in Section 6.2.

Finally, a crucial advantage of Salient Poses is that it employs a generic objective function. This means that we can plug-in different criteria for which the selections of keyframes are to be optimized. Later, in Chapter 4, we will propose how different implementations of the objective function can be used together to realize a solution to

our problem: a solution to selecting keyframes that are similar in structure to those used in conventional computer animation practice.

Chapter 4

Keyframe Selection for Blocked Animation

In this chapter, we apply Salient Poses to select keyframes similar to those used by animators. As described through Chapter 3, we have designed Salient Poses to optimize the selection of keyframes for a generic objective function. This abstraction over the implementation of our objective function is important because it means that we are able to apply Salient Poses to select keyframes for any criteria that we can express with an implementation of that objective function. We take advantage of the abstraction in this chapter and propose a three layer keyframe selection technique. Each layer applies Salient Poses with a different implementation of the objective function; that is, each layer selects keyframes according to a different criteria.

We begin this chapter by briefly revisiting the qualities of blocked animation that make it easy to edit (Section 4.1) and then examine two features of motion that complicate the problem of selecting keyframes similar to those used by animations (Section 4.2). We then introduce the three objective functions used in our three layer approach (Section 4.3), along with a new threshold-based algorithm that can automatically choose a selection from the selection space provided by each layer (Section 4.3.4).

In Section 4.4.1 we evaluate our technique by demonstrating that we are able to recover the keyframes of three different keyframe animations that were created by animators. In Section 4.4.2, we show that the same style of keyframes are selected when we apply our technique to motion capture and therefore demonstrating that we are able to select animator-like keyframes from motion capture. As keyframe animation is editable by design, we conclude that our technique can select the keyframes required to convert motion capture into editable keyframe animation. Later in Chapter 5, we describe a new curve fitting technique for reconstructing the new animation by

interpolating the given keyframes, which is the final step required to create this new editable animation.

4.1 Keyframes for Editable Animation

Recall from Section 2.1 that animators use keyposes and breakdowns to create a blocked animation: the keyposes are keyframes that define extreme poses in the motion and the breakdowns are keyframes that define how a character transitions through those keyposes. Both the keyposes and breakdowns together comprise an editable animation.

There are two qualities about a blocked animation that make it easy to edit:

Sparsity Since the keyframes define the poses of the inbetweens, adjustments to keyframes have a high level of impact over the motion when those keyframes are sparse in number. Animators purposefully use keyframes sparingly to preserve the impact that their adjustments have over the motion (at least until the final refinement stage of the animation process).

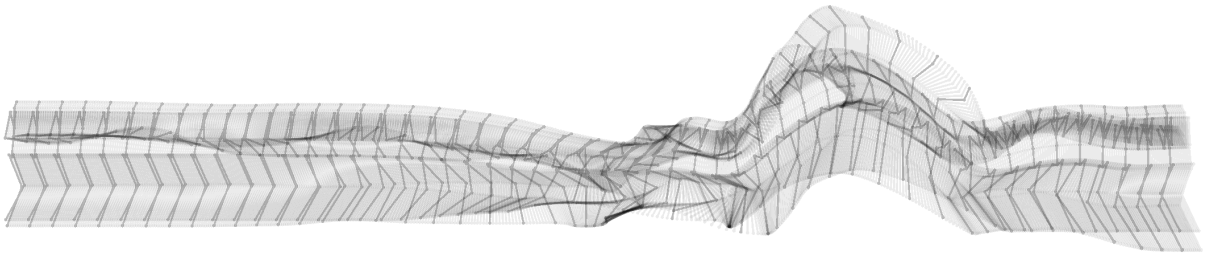
Distribution Animators distribute their keyframes through the motion to enable a higher level of control over the motion’s detail. Conventionally, keyframes are clustered together more densely in parts of the motion that feature higher frequency in their detail (see Figure 4.1) and more sparsely in other parts that feature lower frequency in their detail. Thus, we propose that editable keyframes should be distributed through the motion in a such a way that corresponds to the frequency of the motion’s detail.

4.2 Complications for Keyframe Selection

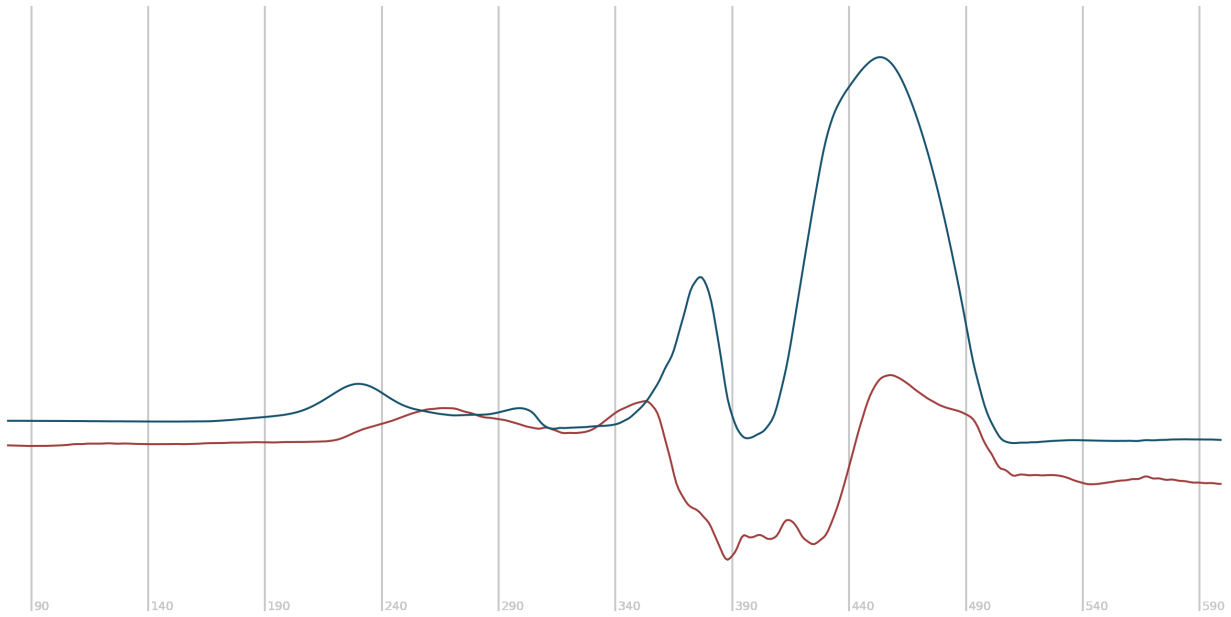
There are two factors that complicate the problem of selecting a set of sparse and well distributed keyframes.

Fluctuation in Scale As we described in Section 4.1, the details of a motion change in frequency throughout a motion. However, as well this, the scale of those details also increases and decreases through the motion. As illustrated in Figure 4.2, the scale of details is small during the earlier phase of the motion when the actor is standing relatively still, becomes slightly larger when the actor moves forward over two steps, increases significantly as they jump into the air, and reduces again as they regain their balance.

Fluctuation in the scale of details is one factor that complicates the problem of selecting well distributed keyframes. For example, applying maximum perpendicular



(a) Animation



(b) Curves of Right Foot

Figure 4.1: The frequency of details increases and decreases throughout a motion. The curves of the right foot (b) of the LAY UP motion capture animation (a) provide an example of how the frequency of details change. The actor is standing still during the first part of the motion (frames 90-210) and, consequently, the recorded animation curves corresponding to the feet feature almost no changes. The actor begins to move forward toward the basketball hoop over two steps (frames 240-340), crouches slightly (frames 340-360), and then launches into the air to shoot the ball (frames 360-470). The frequency of details increase most significantly during the two steps (frames 240-340) and when transitioning through the jump (frames 390-410). To realize an editable keyframe selection for this motion, it would be important to select keyframes that are clustered more densely together in these parts containing higher frequency detail (to ensure that an editor would have the ability to adjust these detailed parts of the motion precisely).

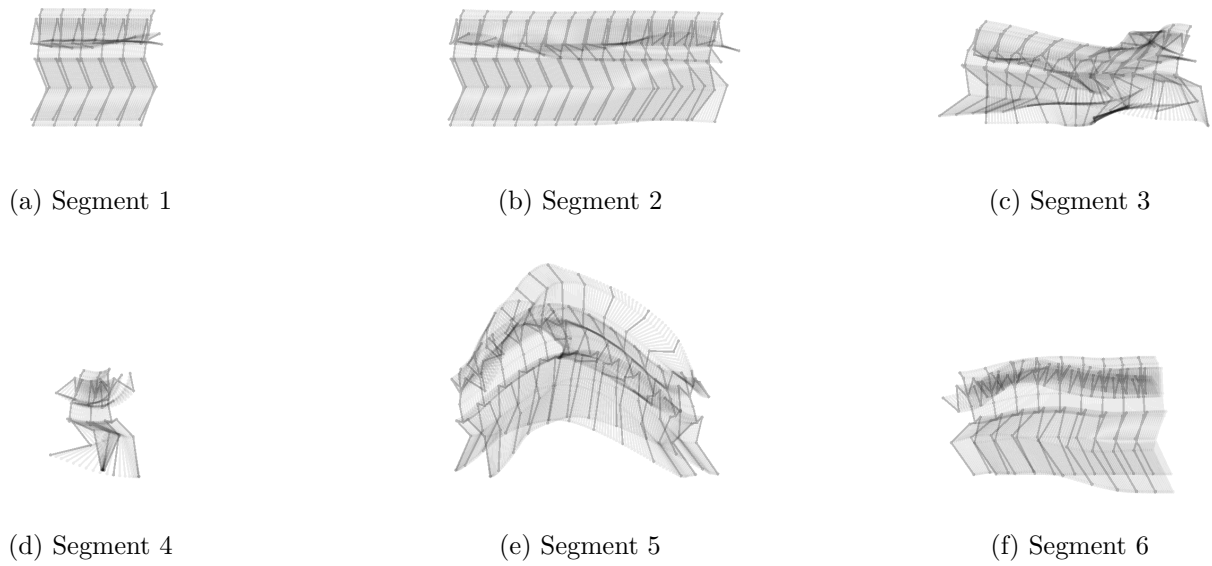
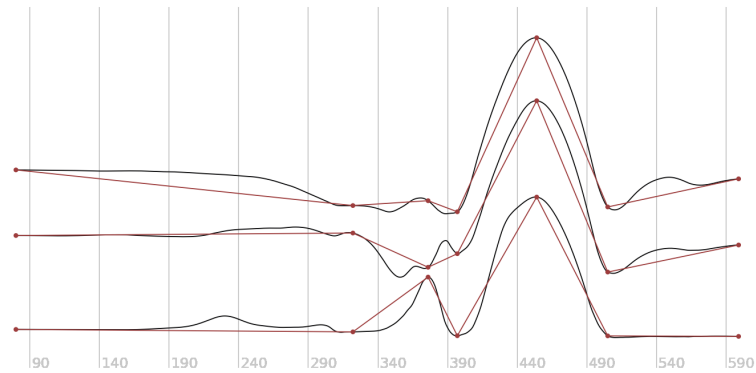


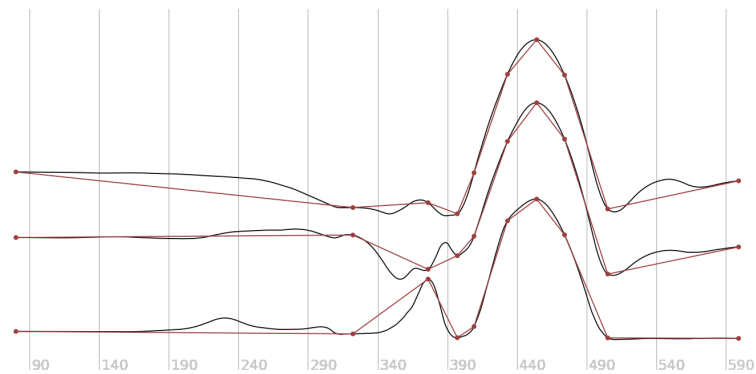
Figure 4.2: We describe the idea that the amount of change between poses increases and decreases throughout the motion as *fluctuation*. Again referring to the LAY UP example: the actor is standing almost perfectly still at first (a) and as such the changes between poses are minimal. They then begin to move their arm to reposition the basketball (b) and thus the amount of change between poses slightly increases through this segment. The actor then takes two steps forward (c) and crouches slightly (d), during which time the changes between the poses become larger again. The actor then launches into the air to shoot the ball (e), leading to significantly larger changes between poses. Finally the character regains their balance (f) and the amount of change between poses reduces again.



(a) 5 Keyframes Selected



(b) 8 Keyframes Selected



(c) 10 Keyframes Selected

Figure 4.3: The maximum perpendicular Euclidean distance measure initially distributes keyframes well through the motion (a and b), but as further keyframes are added the larger arcs, that can already be controlled well using the given keyframes, receive further keyframes before other parts of the motion that cannot be yet controlled easily (frames 190-300, 340-390, and 520-560). Thus, while this criteria is useful for initially segmenting the motion it can lead to poor distributions of keyframes when higher numbers of keyframes become available.

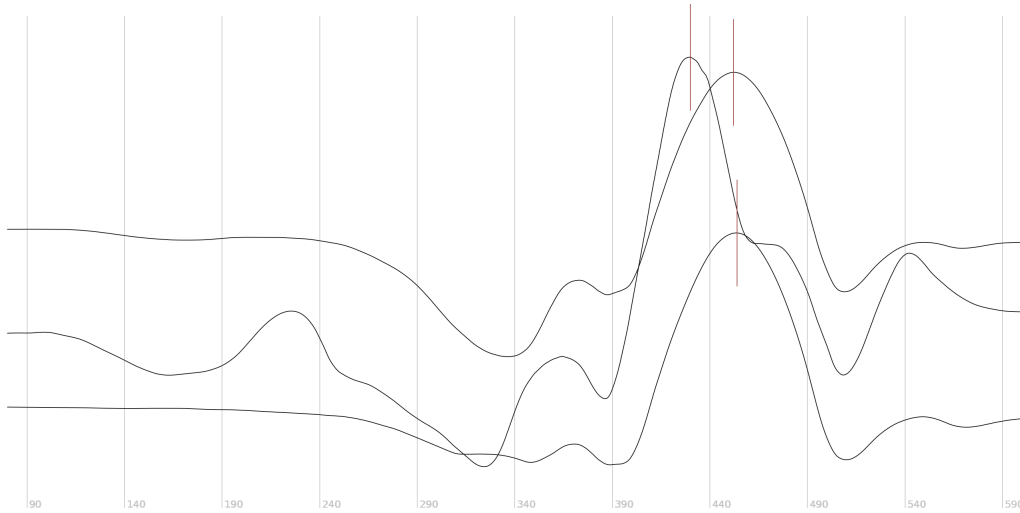


Figure 4.4: Different joints of an animation transition through their extreme poses offset from one another. This offsetting effect can be noticed easily among different curves of the LAY UP animation. The curves above depict the height of left hand, the head, and the hips through the motion. The most significant offsetting in the extrema of these curves can be seen through the peak of the jump (frames 430-450). Other offsets can be seen in the two extrema that occur before the jump (frames 350-370 and 380-400) and also the two extrema that occur after (frames 500-520 and 530-540).

Euclidean distance as the criteria for keyframe selection results in the larger details being over-represented by the keyframes and the smaller details being under-represented (see Figure 4.3). Such a selection of keyframes does not have a distribution suitable for editing because too much control is provided over parts of the motion that feature larger changes between poses and too little control is provided over parts with smaller changes between poses.

Offset Extremes Extreme poses for each of the joints in a motion can occur offset from one another in time. Referring again to the LAY UP example in Figure 4.4, we can observe that a number of extrema between the curves are related in that they occur within a small period of time (most notably the extrema corresponding to the peak of the jump in frames 430-450). The offsetting of extreme poses between joints can be observed in almost all natural movements in which there is a chain of joints: in a ball swinging on a chain, in the tails of dogs and cats, along the body of a snake when slithering, in the arm of a baseball player pitching a ball, and even subtly along the spine and limbs of a human walking. The offset quality complicates the problem of keyframe selection because each extrema indicates a possible keyframe (in the case of the LAY

UP motion, any frame between frames 430-450 might be considered a keyframe). Thus, a keyframe selection technique must employ some process to determine which extrema are more important than others to avoid choosing too many keyframes around each extreme pose.

4.3 Three Layers of Keyframe Selection

As we outlined during the introduction to this chapter, our approach to selecting editable keyframes is to apply Salient Poses with three different objective functions. This three layer design is able to produce sparse and well distributed selections of keyframes despite the presence of fluctuation in the animation.

To address the offset problem, we borrow the concept of a *leading part*. As described by Lasseter [47, p. 39]:

In the movement of any object or figure, the actions of the parts are not simultaneous: some part must initiate the move, like the engine of a train. This is called the lead. In walking, the action starts with the hips. As the hip swings forward, it sets a leg in motion. The hip “leads”, the leg “follows.” As the hip twists, the torso follows, then the shoulder, the arm, the wrist, and finally the fingers.

We propose to use the idea of a leading part to identify which part of the animation is most indicative of extreme poses. That is, an animator simply selects one or more joints to be considered as a leading part.

Our first layer of keyframe selection is designed the motion into segments based on fluctuations in the motion. The motion is divided by applying Salient Poses with the first objective function (the implementation of this objective function is described next in Section 4.3.1). The objective function is applied to only the subset of the motion that corresponds to the leading part. We then choose a selection from the resulting selection space that coarsely approximates the motion of the leading part. We then use that selection to divide the motion into separate parts - henceforth denoted as *segments* - in a way that each of the segment tends to feature no more than one fluctuation (note how the keyframes displayed in Figure 4.3b are effective in separating fluctuations that occur within the curve).

As an alternative to our leading part idea, it would of course be possible to take a weighting approach: the editor configures a set of weighting parameters defined for each joint (or perhaps each dimension). The values in the each curve, of the motion capture, would then be scaled based on their corresponding parameter. While the

weighting parameters enable the editor to provide a more detailed description of which parts of the motion are significant, configuring individual parameters for each joint (or dimension) would be tedious for the editor. Furthermore, it would be unclear as to how each parameter should be modified to obtain different distributions of keyframes. Our approach, using the idea of the leading part, favours simplicity for the editor at the cost of precision. If precision is required, a developer may help the editor make use of weights by simply adding them to the relevant objective function.

Our second layer is designed to select any further extremes as keypose-like keyframes. It applies Salient Poses, again, to each of the segments individually and instead uses the second objective function (described next in Section 4.3.2). Note that this step is unlikely to suffer from the fluctuation problem since we have already coarsely divided the animation to avoid segments containing multiple fluctuations, via the selection from the first layer. Furthermore, as we are able to treat each segment individually, we can choose a selection of keyframes that is appropriate for the frequency of details in that segment: we chose a selection from the selection space that captures any further extreme poses of the leading part that appear significant relative to the scale of the details in the corresponding segment. Finally, we combine together the keyframes selected for each segment of the second layer together and segment the animation again.

Finally, the third layer applies Salient Poses with the third objective function to select breakdown-like keyframes. For this third layer Salient Poses is applied, again to each of the segments that result from the second layer, and instead uses the third objective function (described next in Section 4.3.2). The third objective function is designed to selected any further keyframes required to ensure that the complete set of keyframes has the potential to accurately reconstruct the original motion when interpolated with the type of curves used in animation software (we introduce these curves in more detail in Chapter 5).

In summary, our three layer technique can be used to select animator-like keyframes from motion capture: by segmenting the motion first, based on a coarse linear approximation, we circumvent the problem that fluctuations in the motion might lead to selections of keyframes that are overly dense in areas containing larger scale detail. In other words, the design enables our technique to avoid the problem of over-representing larger arcs in the motion while also under-representing smaller arcs. To address the offset problem, we have borrowed the notion of a leading part that enables us to select keyframes, in the first two layers, according to a subset of the motion.

Finally, by recording which layer selected each keyframe, we can categorize the selection's keyframes as keyposes and breakdowns: the keyframes that result from the

application of the first two layers comprise a selection of keyposes and those added by the third layer comprise a selection of breakdowns.

We now describe the objective functions used for the three layers: the first in Section 4.3.1, the second in Section 4.3.2, and the third in Section 4.3.3. To conclude the section we introduce a simple threshold-based algorithm that can be used to automatically chose a selection from the selection space resulting from each layer Section 4.3.4.

4.3.1 Segmenting the Motion

Recall from Section 3.1.2 that Salient Poses applies the objective function to measure the length of each edge in the graph; or in other words, the objective function is applied to all possible segments of the motion. Thus, to implement an objective function we need only design a mechanism to measure edge length with respect to a segment of the motion.

As outlined in Figure 4.3, the maximum perpendicular Euclidean distance measure is a useful criteria for selecting keyframes that outline fluctuation among an animation’s curve. We employ this measure for the objective function of the first layer and by doing so enable a mechanism for dividing the motion into segments in a way that each segment is unlikely to contain more than one fluctuation (the first layer divides the motion into segments using a coarsely fitted linear interpolation). Since we define a fluctuation as the transition from a larger scale of detail to a smaller, or a smaller to larger, we can assume that a coarsely fitted linear interpolation would include a keyframe before, after, or otherwise during the fluctuation.

In much the same way as Lim and Thalmann’s approximation-based keyframe selection technique [50], we calculate the measure of distance by first interpolating the given segment with a high dimensional line. We then measure the distance between each of the frames in that segment and the interpolating line, and choose the largest distance as the value prescribed by the objective function. More concisely, we calculate the measure as the maximum perpendicular Euclidean distance between the high dimensional curve of the animation and the linear interpolation of the given segment.

Equation (4.1) presents a formula for this first objective function. The function’s parameters (denoted i and j) correspond to the first and last frames in the segment. We then choose the variable k such that the distance between the interpolating line and the original motion is largest. Finally, M_k denotes the point in the curve that occurs at frame k of the animation and L_k denotes the point on the line which is nearest to

M_k .

$$o_1(i, j) = \max_{k \in (i, j)} \|M_k - L_k\|^2 \quad (4.1)$$

4.3.2 Selecting Keyposes

The next objective function expresses accuracy in terms of a more typical sum of squared Euclidean distances between the original animation and the same linear interpolation as described in Section 4.3.1. Equation (4.2) presents the formula for our second objective function, in which the denotations are the same as in Equation (4.1).

$$o_2(i, j) = \sum_{k=i}^j \|M_k - L_k\|^2 \quad (4.2)$$

Note that Equation (4.2) simply replaces Equation (4.1)’s maximization operation with a summation. Replacing the maximization with the use of the summation is not essential, but helps to ensure that the keyframe selection is not obstructed by the presence of artefacts, such as noise or spikes, among the animation’s curves.

4.3.3 Selecting Breakdowns

The last objective function expresses accuracy in terms of the maximum perpendicular Euclidean distance between the animation and a curve interpolation of the keyframes that has been fitted to the segment. Using curve approximation in place of the linear interpolation enables keyframes to be selected with respect to how easily the animation can be reconstructed using curve-based interpolation and, furthermore, ensures that keyframes are not added when the current set of keyframes already summarizes the motion well (when interpolated with the appropriate curves, but perhaps not a line).

Equation (4.3) presents the formula for our third objective function, using the same denotations as in Equation (4.1) and Equation (4.2). The variable C_k denotes the point in the fitted curve that is nearest to M_k .

$$o_1(i, j) = \max_{k \in (i, j)} \|M_k - C_k\|^2 \quad (4.3)$$

Appendix A provides a brief description of the curve fitting technique that we use for our implementation of Equation (4.3).

4.3.4 Composing the Selections

As briefly outlined at the start of this section, we can use the three objective functions together to select keyframes suited to editing. To do so, we obtain the selection space for the first layer and choose from it a particular selection that we will use to segment the motion, and then apply the next two layers in the same fashion.

The choice of selection from each layer can be made either manually or automatically. For manual choice, the selection space is computed quickly by Salient Poses and then browsed interactively using a slider interface (perhaps displaying the keyframes through a dopesheet interface or through a viewport displaying the poses of the keyframes in the 3D scene). For automatic choice we employ a simple threshold algorithm.

Recall our observation, from Section 3.2.1, that in the selection space for the WALK animation some of the selections featured a sharp drop in approximation error in comparison to the selection with one less keyframe. We base our approach for automatically choosing a selection from a selection space on this idea of drops in accuracy between successive selections.

The approach is simple, we first eliminate all selections from the space that do not contain a sufficient level of accuracy (defined by a configurable threshold). We then eliminate all selections from the space that do not feature a significant drop (defined by another configurable threshold). After these elimination steps, we choose the selection from the space that has the highest level of compression.

As indicated above our approach uses two thresholds: an *accuracy threshold* to define the level of accuracy that a selection must have to avoid elimination and a *significance* threshold to define the amount of improvement that a selection must have to avoid elimination.

Accuracy Threshold The accuracy threshold employs a normalization term, which divides the measure of accuracy assigned to each selection by the largest Euclidean distance between any two poses within the given segment. By doing so, the accuracy threshold can be expressed in terms of a percentage. For example, setting the threshold to 20% and using the maximum perpendicular Euclidean distance objective function would ensure that the selections remaining after elimination can be interpolated in a way that the largest distance between the approximation and the animation is no more than 20% of the largest distance between any two poses in that segment. Applying the normalization term per segment is critical, because it enables more keyframes to be added to segments that feature higher frequency, but smaller scales, of detail. And conversely, the normalization also enables fewer keyframes to be added to segments that feature lower frequency, but larger scales, of detail.

Significance Threshold Our significance threshold, also percentage based, is useful for increasing the chance that selections who feature a good trade-off between accuracy and compression are more likely to be chosen, which helps to ensure higher levels of sparsity in the final selection. To illustrate how the threshold works, suppose that we configure the threshold to 10% and have a selection of 10 keyframes featuring a normalized measure of accuracy of 15%. In this case, for the next selection of 11 keyframes to be considered significant, that selection must feature a normalized measure of accuracy of 13.5% or less. Therefore, the significance threshold encodes how much a given selection must improve in comparison to its predecessor to avoid elimination.

4.4 Results

The objective of the results presented within this section is to demonstrate that our three layer design, together with the simple threshold-based algorithm, can select keyframes from motion capture that are similar in sparsity and distribution to the those found in keyframe animation.

In this section we will first present results that will demonstrate how our technique can recover keyframes similar to those created by professional animators. Specifically, we convert three keyframe animations that were created by professional animators into a data set, which is similar to motion capture, and then apply our keyframe selection technique (Section 4.4.1). By doing so, we can compare the animators' keyframes directly to those selected by our technique. We then continue into Section 4.4.2 where we apply the same implementation of our technique to animations created using motion capture. We discuss how the keyframes selected from motion capture feature a sparsity and distribution similar to those from the keyframe animation examples.

4.4.1 Keyframes from Keyframe Animation

Blender Institute's BLENDER CLOUD [8] is an open source repository containing tutorials for visual effects paired with scene files from Blender Institute's open movies projects (short animated films). We have selected three exemplary keyframe animations from this repository, each created by a professional animator:

- **Robotic Walk.** The first example that we use features a robot dog character from the TEARS OF STEEL open movie project. Through an animation of 40 frames, the robot takes two steps forward (the animation can be looped to realize a walk cycle from these two steps). The animator used a total of nine keyposes and no breakdowns.

- **Sneaking Agent.** The second example that we use features an agent character from the AGENT 327 open movie project. In this animation of 74 frames, the agent is sneaking through a passageway. In the motion, the agent takes two forward steps and then begins a third, in which they carefully place their foot forward but do not shift their weight onto that leg. The animator used a total of nine keyposes and eight breakdowns.
- **Attacking Agent.** The third example that we use also features the agent character from the AGENT 327 open movie project. In this animation of 105 frames, the agent is about to be ambushed by an assailant. The agent first turns backward to observe the assailant, then turns forward again and lowers into a crouch before swinging their suitcase around to stop the attack. The animator used a total of seven keyposes and eleven breakdowns.

In order to apply our technique we first need to convert the keyframe animation into ersatz motion capture. To do so, we recorded the positions of 17 joints at 120 frames per second: five joints along the spine and three joints along each limb. Thus, the ersatz motion capture is similar in terms of its data to the motion capture examples that we used in Section 3.2.

Finally, we determined which keyframes correspond to keyposes and which to breakdowns for each of these three keyframe animations. In the case of the ROBOTIC WALK example, the keyposes are tabulated clearly in the dopesheet interface. However in the SNEAKING AGENT and ATTACKING AGENT examples not all of the keyframes were annotated as keyframes or breakdowns. In order to distinguish whether unlabelled keyframes were intended as keyposes or breakdowns we examined the number of the dimensions that featured a keyed value for the frame. When all dimensions, or almost all (no more than 10 dimensions missing keyed values), featured a keyed value we assume the given frame to be a keypose. We considered frames featuring keyed values for only a few of the dimensions (fewer than 10) to be detail keyframes that were likely added after the blocking stage and treated them as non-keyframes. Finally, we consider all other unlabelled keyframes as breakdowns.

Algorithm Set Up

Recall from Section 4.3 that the first two layers of our technique apply to a leading part. For the ROBOTIC WALK example we defined the leading part as the hip joint, since extrema in the animation’s curves for the hips are typically indicative of the occurrence of contact poses in walk cycles. For the SNEAKING AGENT animation we defined the leading part as the feet, since the positioning of the feet is critical in that motion (which has a tip-toe effect). And for the ATTACKING AGENT example we

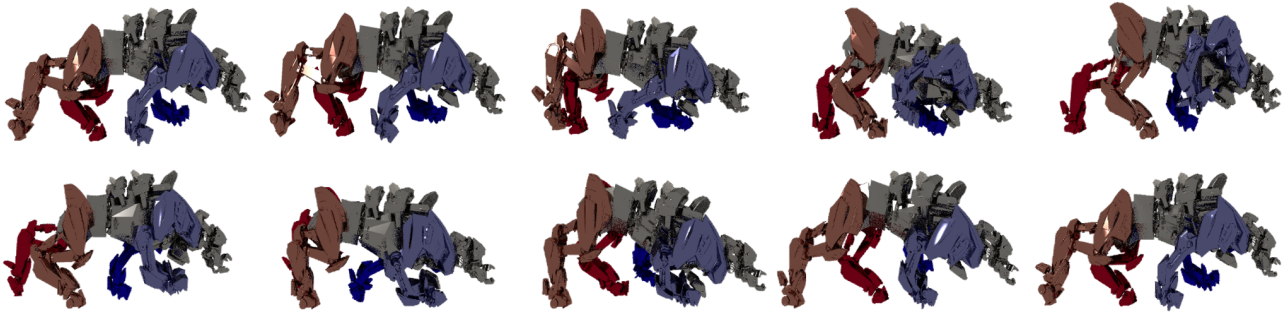


Figure 4.5: The frames of the **ROBOTIC WALK**. The nine keyposes created by the animator are presented as renders, with the virtual camera centred on the character. The keyposes are organized to presented those used in the first step (front left leg being planted and the right lifted forward) and the second (front right leg being planted and the left lifted forward). Note that the right-most image in the top row is the same keypose as the left-most image in the bottom row.

defined the leading part as the neck and head, since most of the movement in the early part of the animation involves the head (the agent turns back to see the assailant, then forward, and then back again while swinging the suitcase).

In each of these three cases, we configure the parameters for the threshold algorithm in the same way: the first and second layer set the accuracy threshold to 20% (to encourage sparsity), while the third layer sets it to of 1% (to ensure a high level of accuracy); all layers set the significance threshold to 10%.

Robotic Walk

Figure 4.5 displays the nine keyposes that were created by the animator for the **ROBOTIC WALK** animation.

We applied the first layer, whose selection space as depicted in Figure 4.6a. The first selection of three keyframes already features a normalized measure of distance below 20% and as such it is automatically chosen as the selection for the first layer.

We then applied the second layer to each of the two segments that result from the first selection. Our threshold-based algorithm chose not to add any keyframes from the second layer, since the movement of the leading part was already well approximated by the linear interpolation of the existing keyframes.

We then applied the third layer to each of the two segments. As demonstrated by the visualization of the selection space in Figure 4.6b and Figure 4.6c, our threshold-based technique automatically chose a selection of five keyframes for both segments. We then composed the selections together, which resulted in a total of nine keyframes (displayed as two segments in Figure 4.7).

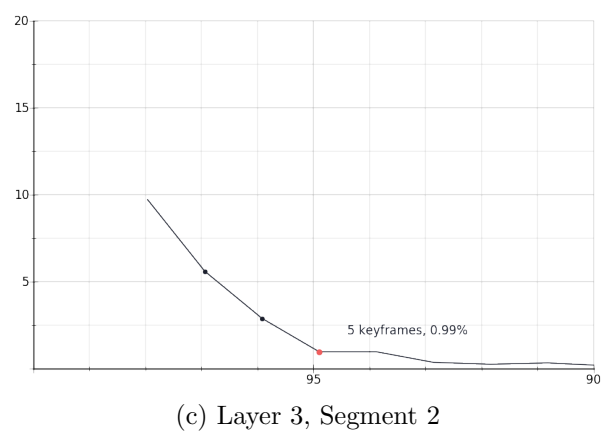
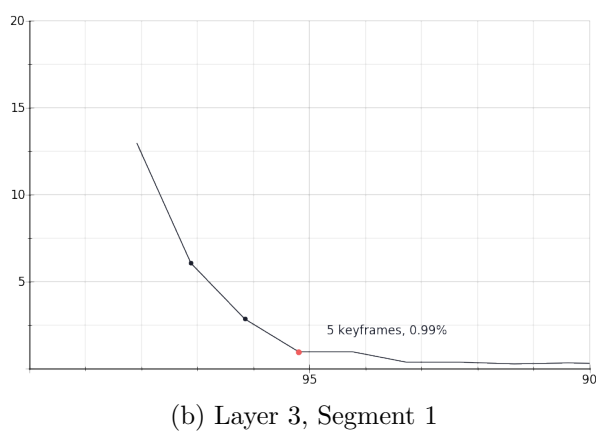
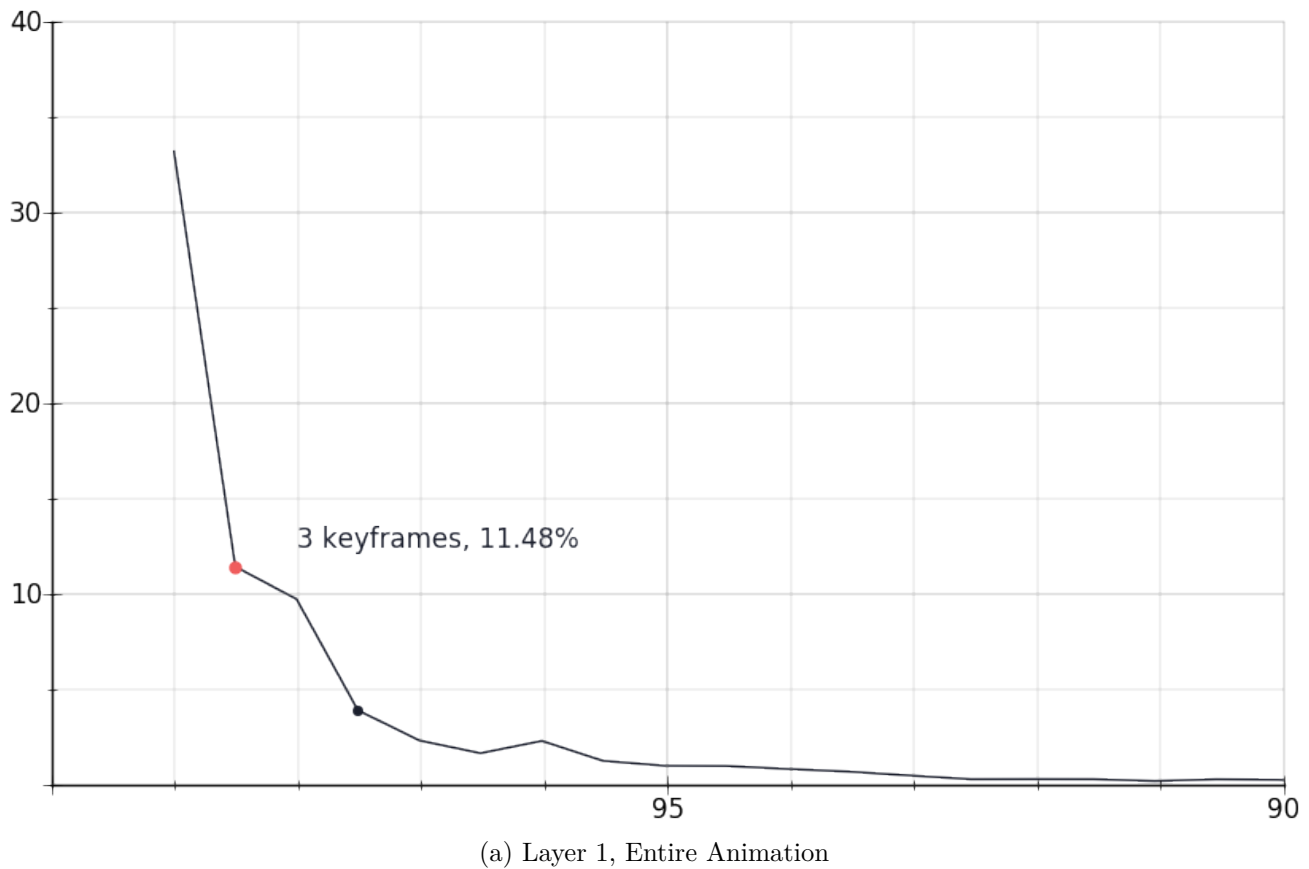


Figure 4.6: The selection space produced by Salient Poses in the first and third layer (no keyframes were added by the second layer). Each of the selections chosen by our threshold-based algorithm are highlighted in red and are annotated with their measure of normalized distance.



(a) Segment 1 from Layer 1



(b) Segment 2 from Layer 2

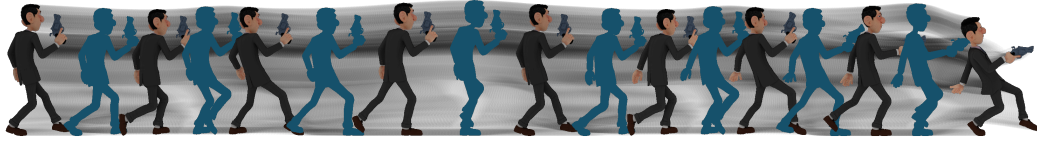
Figure 4.7: The keyframes that result from the application of our technique. The first layer introduced two segments as displayed above. No keyframes were added by the second layer. The keyframes added by the third layer are depicted by the red silhouettes.

The final selection of keyframes were *identical to the animator’s original keyframes*. Since our technique was able to recover the animators keyframes exactly, we conclude that our technique was able to recover editable keyframes.

A slight drawback is highlighted by this result: our third layer selected the animator’s keyposes, rather than the second. As discussed through Section 4.3, we designed the objective functions of the first two layers to realize the selection of keyposes and the third layer to realize the selection of breakdowns. To interpret why our selected keyframes appeared as breakdowns and not keyposes, we reviewed the discussion on walk cycles presented by Williams [73, p. 102 - 117], in which an illustration of a typical walk cycle is provided. Their illustration features the two contact poses (which appear to be keyposes) and three passing poses (which appear to be breakdowns). Since the categorization of the keyframes selected by our technique is consistent with the example given by Williams, we propose that the difference in categorization is of no significant concern.

Sneaking Agent

Figure 4.8 compares the animator’s keyframes for the SNEAKING AGENT example to the selection that we obtained from our technique. The animator has used keyposes at each contact pose in the steps as well as one further keypose half way through each step. Additionally, one breakdown was used between each pair of keyposes. In total, each step is encapsulated by five keyframes.



(a) Animator's Keyframes



(b) Keyframes as Selected

Figure 4.8: The keyframes used by the animator to create the SNEAKING AGENT animation (a) compared to those selected by our algorithm (b). The animator's keyposes are displayed in full colour and their breakdowns are displayed as blue silhouettes. In a similar way, the keyframes selected in the first and second layers of our technique are displayed in full colour and the keyframes added by the third layer are displayed as blue silhouettes.



Figure 4.9: A closer view of the fifth keypose from Figure 4.8. The animator's keypose is presented as partially transparent with the keypose, as selected by Salient Poses, displayed as a fully-opaque render.

The selection of keyframes provided by our technique is almost equivalent to the animator’s keyframes. In particular, our selection features exactly five keyframes per step with the same ordering of keyposes and breakdowns: keyposes occur at contact poses and half way through each step, with one breakdown between each keypose.

Minor differences occur in the fifth and eighth keypose of our selection, which differ slightly from the animator’s keyposes (a closer view of the fifth keypose is provided in Figure 4.9 to help illustrate the difference). These differences are a consequence of how we have designed the second layer of Salient Poses, which makes use of a linear interpolation paired with the sum-squared distance measure to add any further poses that appear locally as extremes. Consequently, the keyposes in our selection reflect the extremes among the motion curves of the leading part that we selected, more so than the animator’s keyposes. The reason for the difference may be due to the animator choosing their keyframes based on a different part or, alternatively, the animator may have purposefully placed the keypose on a non-extreme to more evenly distribute the inbetween frames.

Minor differences can also be seen when comparing the third, fourth, and seventh breakdowns between the animations. In the case of the animator’s keyframes, it appears that the breakdowns are distributed to favour having a similar number of inbetweens assigned to each keyframe. In our case, we designed the third layer to select breakdowns that lead to the closest fitting curve interpolation.

We expect that the differences described above will not have an adverse effect on how easily an editor can edit the animation. Our selection provides keyframes, and thus control, over the extreme poses in the motion. Furthermore, our selection has the same number of keyframes as the original animation and therefore features an identical level of sparsity, which means that the editor can use the same number of adjustments to the edit the animation. Finally, the few differences in the distribution of keyframes appears incidental – it may be the case that different animators would reproduce the same animation using slightly differing distributions of keyframes. Based on these observations, we conclude our technique was again able to recover editable keyframes from the motion capture interpretation of the keyframe animation.

Attacking Agent

Figure 4.10 compares the animator’s keyframes for the ATTACKING AGENT example to the selection that we obtained from our technique. Again, the sparsity and distribution of our selection appears to match that of the animator’s keyframes.

One notable difference can be seen regarding the categorization of the third keyframe, which was annotated as a breakdown by the animator and a keypose by our technique. In total the animator used 18 keyframes (seven keyposes and eleven



(a) Animator's Keyframes



(b) Keyframes as Selected

Figure 4.10: The keyframes used by the animator to create the *ATTACKING AGENT* animation (a) compared to those selected by our algorithm (b). The keyframes are presented using the same colouring scheme as was used for the previous example (Figure 4.8).

breakdowns), whereas our technique selected 17 keyframes (seven keyposes and ten breakdowns). Thus, we suggest that our technique missed one keyframe in the earlier section of the motion (perhaps the animator's fourth keyframe).

Section 4.4.1 and Section 4.4.1 provide another comparison between the animator's keyframes and our selection, which helps to show the difference in structure between the two sets of keyframes. Note that in the first row the animator's keyframes feature one more breakdown than ours. Despite the missed keyframes, both the sparsity (the number of segments) and the distribution (the size of each segment) are similar between both our selection and the animator's keyframes. As with the previous example, we conclude that our technique was again able to recover editable keyframes from the motion capture interpretation of the animation: our technique offered a set of keyframes that are similar in terms of pose, sparsity, and distribution to those created for the original animation.

Finally, we include another illustration of the selections that were automatically chosen by our threshold-based algorithm. As depicted by Figure 4.13, we automatically selected six keyframes from the first layer, one additional keyframe from the second segment of the second layer, and ten additional keyframes between the segments of the third layer.



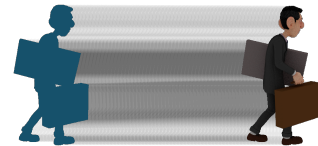
(a) Segment One



(b) Segment Two



(c) Segment Three



(d) Segment Four



(e) Segment Five



(f) Segment Six

Figure 4.11: The segments resulting from the animator's keyframes for the AGENT ATTACKING example.



(a) Segment One



(b) Segment Two



(c) Segment Three



(d) Segment Four

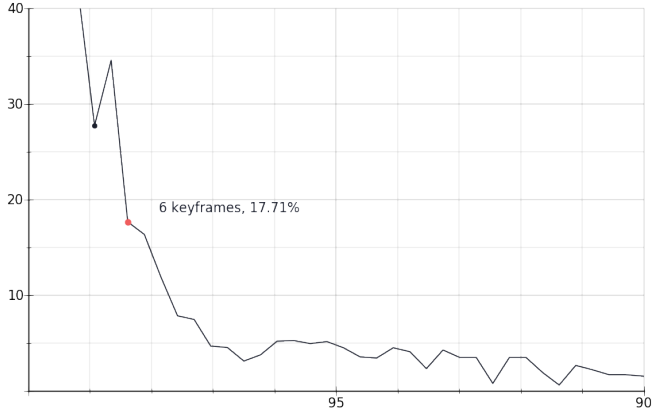


(e) Segment Five

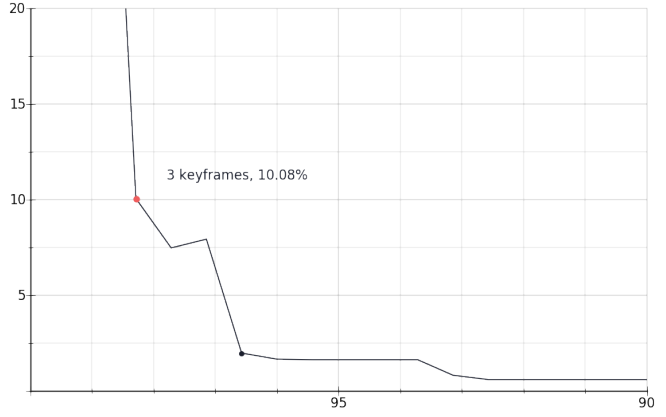


(f) Segment Six

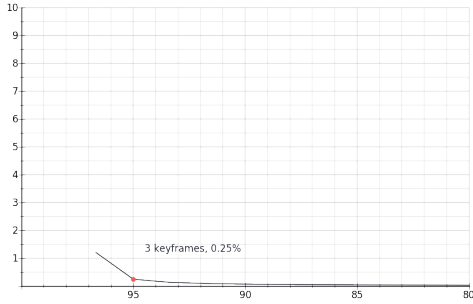
Figure 4.12: The segments resulting from the keyframes selected by our technique for the AGENT ATTACKING example.



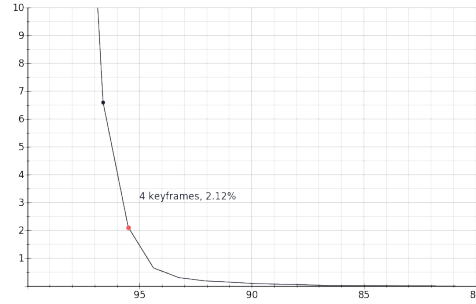
(a) Layer 1, Entire Animation



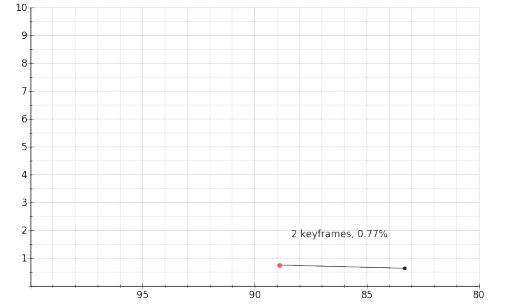
(b) Layer 2, Segment 2



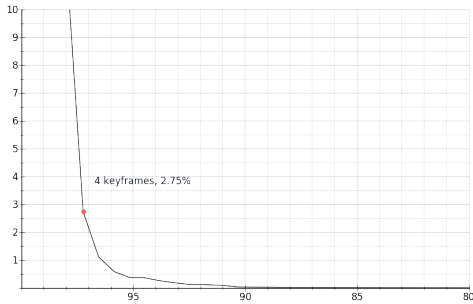
(c) Layer 3, Segment 1



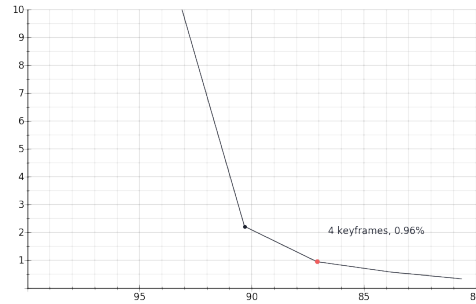
(d) Layer 3, Segment 2



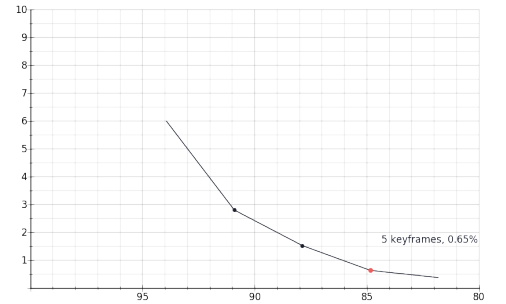
(e) Layer 3, Segment 3



(f) Layer 3, Segment 4



(g) Layer 3, Segment 5



(h) Layer 3, Segment 6

Figure 4.13: The selections chosen by our threshold-based algorithm for the AGENT ATTACK example.

4.4.2 Keyframes from Motion Capture

The objective of this section is to demonstrate that our technique also selects editable keyframes when applied to motion capture animation. We consider results successful when the selected keyframes exhibit sparsity and structure in a similar way to the results we presented for the keyframe animations (Section 4.4.1).

Note that some of the inbetween frames are rendered more opaquely than others in the following results. The more opaque frames indicate the keyframes that would be required to preserve even the finest scale of noise in the animation. We selected these by repeating the third layer a second time with an accuracy threshold of 0.1%. We do not discuss these additional keyframes further, but provide them to illustrate the number of keyframes that would be required to achieve near perfect reconstruction.

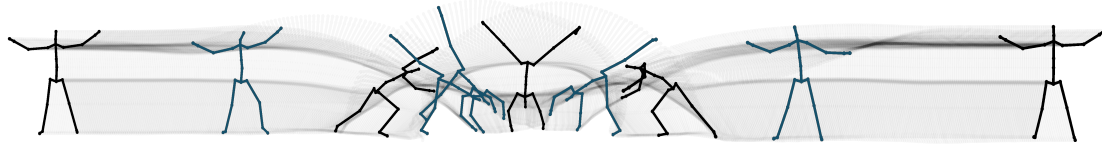
Cartwheel

For the CARTWHEEL example we selected both hands as the leading part and applied the three layers of keyframe selection. The selected keyframes along with the segments are displayed in Figure 4.14.

The motion begins with the actor standing still, slightly turning their arms as they prepare to perform the cartwheel. Our technique selects a breakdown at the point where the arms begin to turn more quickly. The following keypose captures the actor as they twist toward the ground. Two further breakdowns are seen before the next keypose, at which point the actor is precisely half way through the cartwheel. Interestingly, the selection of keyframes appears symmetrical: the first and last segments and the second and third segments appear as mirror images on one another (aside from the additional breakdown in the second segment).

The selection of keyframes is both sparse and distributed well through the motion for editing. In terms of the distribution, we can observe that the higher frequency parts of the motion surrounding the cartwheel have more keyframes than the lower frequency parts where the character is standing relatively still. Moreover, our keyframes enable precise control over extreme poses in the animation: using the keyframes an editor can adjust the actor’s pose as they begin to turn their arms (first breakdown), how quickly they twist toward the ground (second keypose), and how they flick their legs above themselves into the fully inverted pose (second and third breakdowns).

Since the keyframes selected by our technique are both sparse and well distributed, we conclude that our technique was able to produce a selection sufficient for editing.



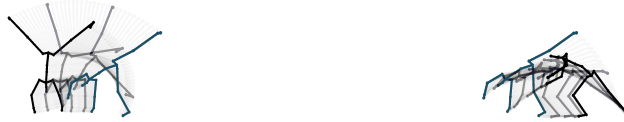
(a) Selected Keyframes



(b) Segment 1



(c) Segment 2



(d) Segment 3



(e) Segment 4

Figure 4.14: The keyframes selected for the CARTWHEEL example, along with the segments that result from those keyframes. The segments are divided into groups and numbered based on the selected keyposes, which are the set of keyframes selected by the first and second layers of Salient Poses (presented in black). We consider the keyframes selected in the third layer as breakdowns; presented in blue. The inbetweens are displayed in transparent grey. Finally, as an additional result, we also presented some inbetweens in a darker grey - a fourth layer of keyframes required to create an exact replication of the original animation.

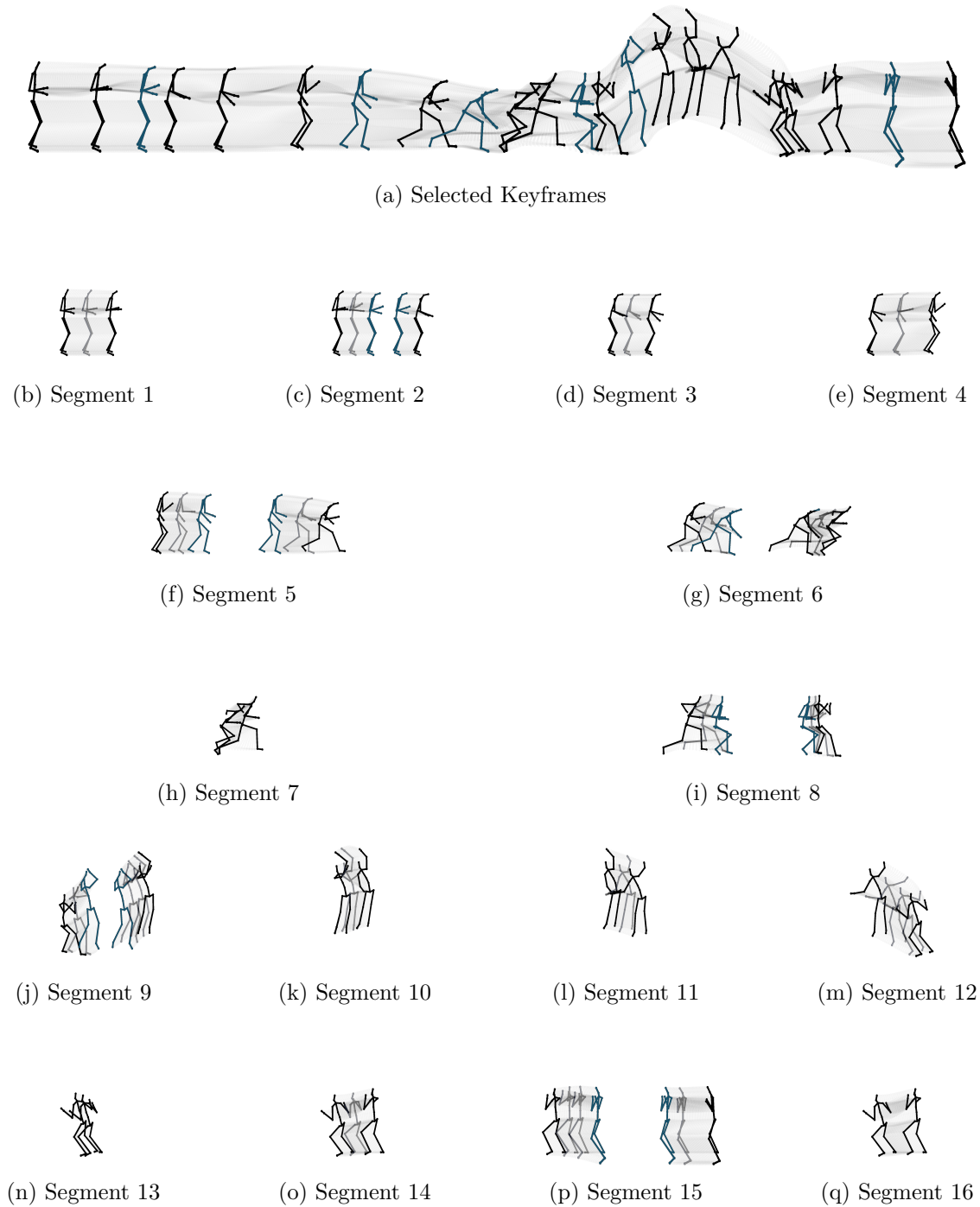


Figure 4.15: The keyframes selected for the LAY UP example.

Lay Up

For the LAY UP example we selected the right hand as the leading part (the player shoots the basketball with their right hand) and applied the three layers of keyframe selection. The resulting segments are displayed in Figure 4.15.

We can observe that the selected keyframes are distributed well for editing in that the keyframes are most densely clustered around the higher frequency parts of the motion: the two steps and crouch before the jump (segments 5-8) and the recoil after first contacting the ground (segments 13 and 14). The distribution is slightly sparser through the jump (segments 9-12) and is most sparse in the lower frequency parts of the motion (segments 1-4 and 15-16).

Keyposes occur at each of the extremes poses, which include the contact poses of the steps (segments 1-6 and 14-16), both the crouch and the recoil before and after the jump (the end of segment 6 and segment 13), and the peak of the jump (between segments 10 and 11). Furthermore, breakdowns correctly occur in between the contact poses of the steps and also as the character launches into the air.

A minor drawback of our selection is that the keyposes that occur when the character is moving the ball (segments 1-4) and when the character is falling after the jump (between segments 11 and 12) would be better represented as breakdowns since they are not extreme poses.

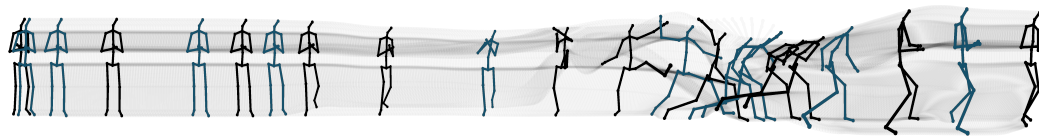
In conclusion, the selection of keyframes produced by our technique for this example are both sparse and well distributed through the motion. Thus, we conclude that our technique was able to produce a selection sufficient for editing.

Pitch

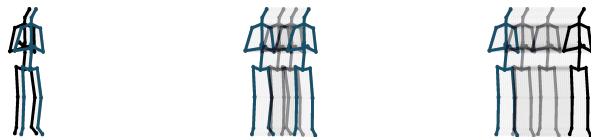
For the PITCH example we selected the right hand as the leading part (the player pitches the ball with their right hand) and applied the three layers of keyframe selection. The selected keyframes, along with the segments they produce, are displayed in Figure 4.16.

As with the previous two examples, we can observe that the selected keyframes are both sparse and distributed well through the motion. The animation's curves contain a higher frequency of detail when the character pitches the ball (segments 7-9) and accordingly our selection clusters keyframes together more closely through that region of the motion and, conversely, the other parts of the motion that have a lower frequency of detail have fewer keyframes (segments 4-6 and 10-11).

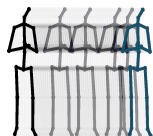
The extreme poses in the animation have each been selected as keyposes: the contact pose of the step (between segments 4 and 5), the pose in which the actor has drawn their arm back (between segments 6 and 7), the last pose during the swing of the arm (between segments 9 and 10), and the contact pose of the final step (between segments



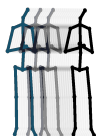
(a) Selected Keyframes



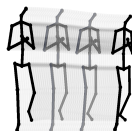
(b) Segment 1



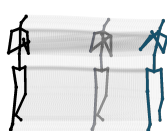
(c) Segment 2



(d) Segment 3



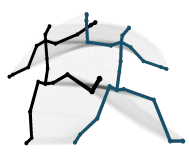
(e) Segment 4



(f) Segment 5



(g) Segment 6



(h) Segment 7



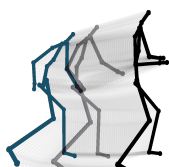
(i) Segment 8



(j) Segment 9



(k) Segment 10



(l) Segment 11



Figure 4.16: The keyframes selected for the PITCH example.

10 and 11). Breakdowns are present when the character is winding back their arm (segment 5), thrusting their arm forward (segments 7 and 8), and half way through each of the steps (segments 10 and 11).

A drawback with the selection is that excessive keyframes were selected in the early phase of the animation (segments 1-3), which was due to the presence of finer scale detail among the curves corresponding to the leading part (the right hand). While the details were small, they were able to effect the selection of keyframes due to the normalization (these details appeared significant in the second layer and consequently the sparser selections were eliminated by our threshold-based algorithm).

Despite the additional keyframes in the early phase of the animation, the overall selection of keyframes remains sparse and well distributed for editing. In the case that the editor needs to adjust the motion of the hand (perhaps to make the character shake the ball more vigorously before the pitch) these additional keyframes would even be helpful. Again, we conclude that our technique was able to produce a selection that is sufficient for editing.

4.5 Summary

In this chapter we returned to the problem of selecting keyframes from motion capture that are suited to editing. Through Section 4.1 we discussed two qualities of keyframe animation that are important for editing: sparsity, to ensure that adjustments to the poses of the keyframes enable a high level of impact over the animation; and distribution, to ensure that a high level of control over the motion's details is provided. We then indicated how the presence of fluctuation and offset among a motion capture animation's curves complicate the keyframe selection problem (Section 4.2). The presence of offset and fluctuation motivated our three layer technique.

Through Section 4.3 we described the implementation of our objective functions, along with the threshold-based algorithm, that can be used together to obtain animator-like keyframes from motion capture. In summary, our first objective function makes use of a maximum perpendicular Euclidean distance measure to divide the motion of the leading part into segments; our second objective function uses a more typical sum of squared Euclidean distance measure to ensure that any extreme poses of the leading part are selected as keyposes; and finally, our third objective function uses a curve-based distance measure to select any further breakdowns that would be required to accurately reconstruct the animation.

In Section 4.4 we evaluated our technique. We first demonstrated that our technique could recover keyframes used by professional animators to create three

different keyframe animations. We then described how the keyframes that our technique selected for motion capture animations were similar in terms of sparsity and distribution to the animators' keyframes. Since we have shown that our technique can realize the selection of animator-like keyframes – which are editable by nature of the conventional animation process – we conclude that our technique meets the primary objective of this thesis: to reverse engineer editable keyframes from motion capture.

Next in Chapter 5, we describe our optimization-based curve fitting technique that can be used to accurately reconstruct each of a motion capture animation's curves using an interpolation paired with a given selection of keyframes. By first selecting editable keyframes with our technique and then applying our curve fitting technique to reconstruct the original motion, we are finally able to convert motion capture into editable keyframe animation.

Chapter 5

Reconstructing the Animation

Previously in Chapter 4 we applied Salient Poses to select keyframes, in the style of keyposes and breakdowns, from motion capture. The final step is to construct a new animation from only the selected keyframes.

Since their initial success in the Modelling and Animation Environment (see Section 2.1), 2D cubic Bézier curves have been employed by animation software to generate transitions between keyframes. Each keyframe, aside from the first and last, is assigned a pair of control points that can be used to change the shape of the interpolation between keyframes. These curves are favoured by animation software because of the wide range of shapes that they can express.

Schneider’s curve fitting technique [64] is a well known technique for fitting a chain of 2D cubic Bézier curves to an ordered set of 2D points. Schneider’s technique begins by interpolating the input data using a single 2D cubic Bézier curve and, unless the fitted curve already approximates the input well, a dividing point is selected and a chain of curves are fitted (see Figure 5.1b). These steps, of adding dividing points and then fitting a chain of curves, are repeated until the fitting is an accurate representation of the input.

Schneider’s technique addresses not only the problem of fitting the curves but also the problem of choosing the dividing points. In our case, the selected keyframes already define the dividing points that should be used when fitting. Since further dividing points would appear to the animation software as additional keyframes, having the fitting technique add further keyframes is problematic because we want to preserve both the sparsity and distribution of the keyframes selected by our technique.

In the context of our problem, of constructing a new animation using keyframes selected from motion capture, we need to replace each of the motion capture animation’s curves with an chain of fitted 2D Bézier curves. The fitted curves must interpolate the keyframes and also accurately represent the animation curve.

The fitting stage of Schneider’s technique, as described in [64], can be used to

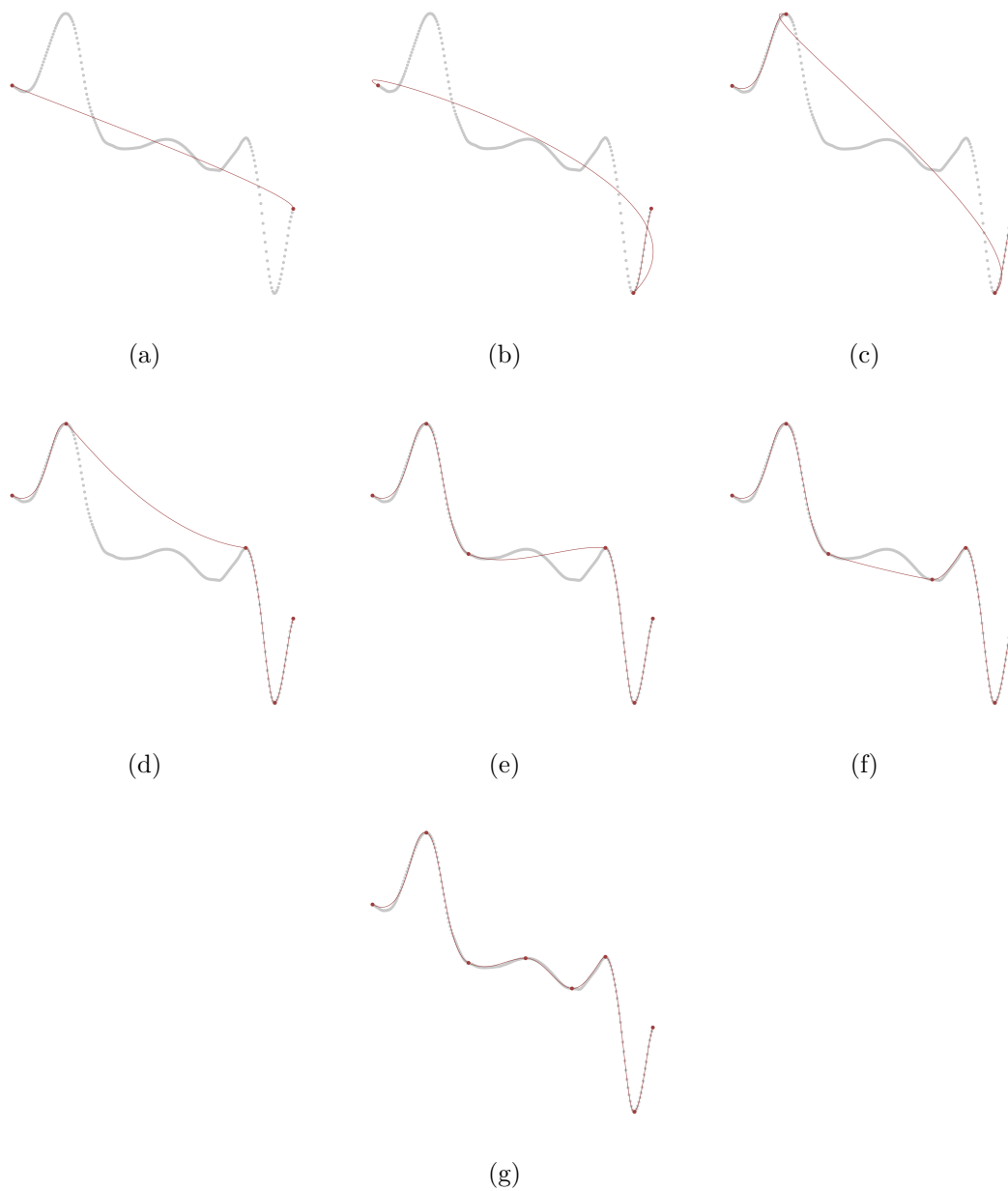


Figure 5.1: Schneider's curve fitting technique is initialized by fitting a single Bézier curve to the input (a). Next, a dividing point is selected as the point of the input furthest from the fitted curve. The technique then updates the fitting, now using two Bézier curves. These steps of adding a further dividing point and then fitting the chain of curves to the input are repeated until the fitting provides an accurate approximation (b to g).

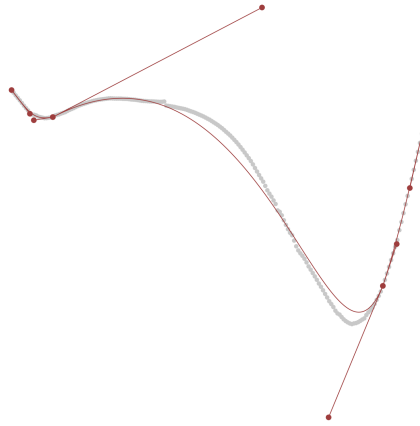
address our problem:

- As input to the fitting step we provide an ordered set of 2D dimensional points, composed from the values of the given animation curve (each value is annotated by the time upon it occurs),
- we set the dividing points based on a predefined set of keyframes,
- we calculate the tangent information of the input using the finite difference method, and finally
- we apply an optimization method to adjust the control points to best fit the input.

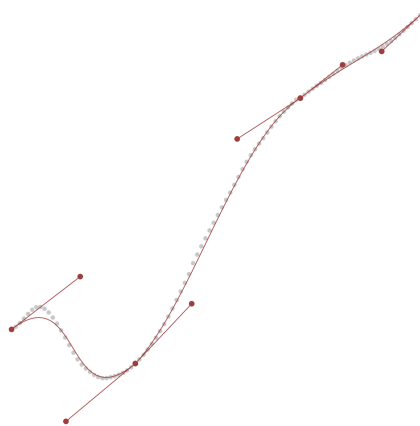
Using animation software, such as Autodesk’s Maya [4] or Blender Foundation’s Blender [7], we can create the new animation by first importing the motion capture data. We then remove all poses that are not keyframes and, as we remove those poses, the animation software automatically replaces them with interpolating curves that use the same type of control points as those used by the fitting technique. Consequently, we can configure the control points of the animation software’s curves based on those that result from the fitting technique. By following these steps we can create a new animation whose data consists only of the selected keyframes paired with the fitted curves and in this way the new animation has the same structure as a keyframe animation that may have otherwise been created by an animator.

While Schneider’s technique works well to solve our fitting problem in general, there are some cases where the technique does not achieve the best possible result. In Schneider’s technique the control points, of the fitted curves, are restricted to occur along the tangents of the input at the dividing points. Restricting the control points to the tangents in this way is useful because it ensures that the fitted curves preserve geometric continuity (the fitted curves do not have corners), but it also means that the algorithm cannot freely move the control points to best approximate the input (see Figure 5.2). Furthermore, the restriction also means that poor fitting results occur when noise in an animation’s curve obscures the finite-difference-based tangent calculation.

In order to provide curves that fit the animation as accurately as possible, we introduce a new optimization-based curve fitting technique. Like the fitting step from Schneider’s technique, we encode the accuracy of the fitted curves as the sum of squares difference between the original data and the chain of fitted curves. Distinct from Schneider’s technique, we allow our optimization technique to configure all control points freely and instead encode a preference for geometric continuity into the objective



(a)



(b)

Figure 5.2: Two examples where Schneider's technique did not fit the input as accurately as possible. In Figure 5.2a the fitted curves lose some of the detail in middle section of the input curve. In Figure 5.2b the curve approximates the input well overall but misses detail surrounding the first peak. Since Schneider's technique enforces that the control points of the fitted curves occur along the tangents of the dividing points, the technique cannot move the control points freely to produce the best fitting possible.

function. By pairing free control of the control points with a preference for geometric continuity we are able to realize more accurate fittings than what is possible with Schneider’s technique, while preserving geometric continuity in general.

We introduce our optimization-based curve fitting technique in Section 5.1, in which we describe how we have realized a balance between accuracy and the preference for geometric continuity using a dual-term objective function (Section 5.1.2). Further details on the accuracy and angle terms are provided in Section 5.1.3 and Section 5.1.4. We conclude, in Section 5.2, with a presentation of results that demonstrate our approach can produce more accurate fittings than the fitting step from Schneider’s technique.

5.1 Optimization-Based Spline Fitting

In this section we present the design of our dual-term objective function that expresses how accurately the fitted curves approximate a given curve from the motion capture animation. We will first express the curve fitting problem as a minimization and then continue to examine how the design of the dual-term in our objective function enables our technique to produce fittings that are accurate and are generally smooth.

5.1.1 Curve Fitting as a Minimization Problem

We can express the problem of curve fitting as a minimization problem, in which the position of the control points are independent variables and the quality to be minimized is the measure prescribed by the objective function. We will denote the input curve (belonging to the motion capture) as C and the fitted curves as C' . The position of the control points of the fitted curves are denoted as H_1 for the first, H_2 for the second, and H_n for the last. Finally, we denote the objective function as Δ to indicate that it describes the difference between the input curve and the fitted curves:

$$\min_{H_1, H_2, \dots, H_n} \Delta(C, C') \quad (5.1)$$

5.1.2 Design of the Objective Function

There are two qualities that we want to preserve in the fitted curves. The fitted curves must approximate the input curve accurately and the fitted curves should be smooth where possible. In particular, geometric continuity is always preferred provided that the animation curve is also smooth (we consider breaks in geometric continuity, for the fitted curve, acceptable in cases where the animation curve is not smooth).

Given the minimization problem, we need only to design the objective function in a way that it assigns the lowest measure of difference when the fitted curves are accurate and generally smooth. We introduce a dual-term objective function: the first term measures the distance between the two curves and then second measures difference in angle of the two vectors between a keyframe and its neighbouring control points. With these two terms, the objective function assigns the lowest measure of difference when the fitted curves are (1) as close as possible to the input curve and (2) contain fewer breaks in geometric continuity.

We now extend Equation (5.1) to express the dual-term. In the following equation Δ_1 denotes the accuracy term, Δ_2 denotes the angle term, and ω_1 and ω_2 denote the weights used to sum those two terms:

$$\min_{H_1, H_2, \dots, H_n} \omega_1 \Delta_1(C, C') + \omega_2 \Delta_2(C, C') \quad (5.2)$$

Given this design, we can employ typical numerical optimization methods to fit an animation's curves in a way that is both accurate and generally smooth.

5.1.3 Accuracy Term

A conventional approach for measuring the difference between the input and fitted curves is to measure the distance between pairs of points. Conventionally, each point of the input is paired to the nearest point in the fitted curves. With these pairs of points, the difference is expressed as the sum of squared Euclidean distances between all pairs. We express this conventional approach in Equation (5.3), in which C again denotes the input curve and C' the fitted curves. The input contains n points and we denote its i^{th} point as C_i . In a similar way, we denote the point nearest to C_i , in the fitted curve, as C'_i .

$$\Delta_1 = \sum_{i=1}^n \|C_i - C'_i\|^2 \quad (5.3)$$

One drawback of the conventional approach is that finding the point in the fitted curves that is nearest to a given input point is a difficult problem. Each fitted curve is a function that can be sampled, as illustrated by Figure 5.3. Thus, the problem of finding the nearest point is equivalent to the problem of choosing the value for that function that corresponds with that nearest point. We can express this as follows, in which we denote a value u applied to the curve's function as $C(u)$:

$$C'_i = \min_u \|C_i - C(u)\|^2 \quad (5.4)$$

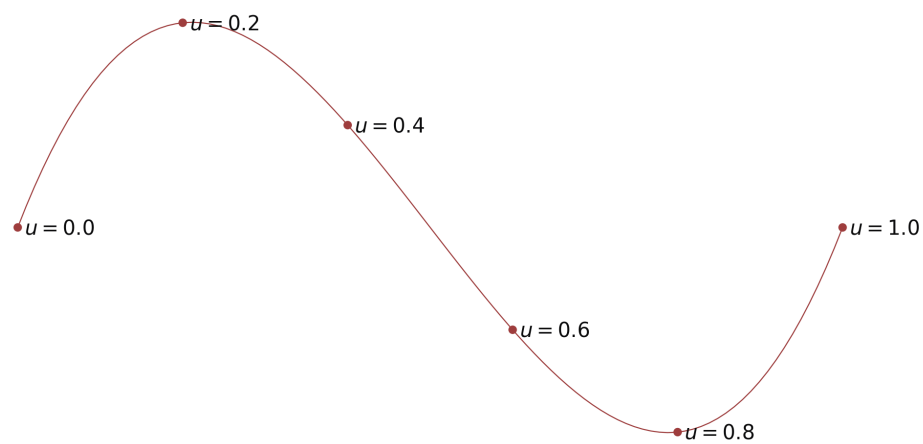


Figure 5.3: Each fitted curve is a function with a parameter, denoted here as u . The curve can be sampled with different values of that parameter to produce points along the curve. A value of zero corresponds to the start of the fitted curve and a value of one to end of that curve. One might imagine the sampling parameter akin to a percentage of the way along the fitted curve (in the sense that a value of 0.2 corresponds to the point roughly 20% of the way through the curve, when traced from the first point to the last).

Typically, and in the case of Schneider's technique, curve fitting techniques have employed *root finding* to solve this minimization problem. While root finding can achieve this task, it is moderately expensive to compute and also difficult to formulate a symbolic derivative for.

Numerical optimization methods can offer a significant reduction in computation time when derivative information is available. As such, we have designed our accuracy term to find pairs of points in a way that it is simple to formulate a derivative, which means that we can take advantage of reduced computation time (at least when using optimization methods that make use of tangent information).

We now describe our accuracy term and how it identifies pairs of points. Note that our accuracy term applies the same calculation to each of the curves in the fitted chain of curves, and as such we describe the accuracy term with respect to only one fitted curve.

We first obtain a set of samples from the fitted curve, using some number of uniformly spaced values for the sampling parameter (in practice, we use 10 per curve). We express this sampling of the fitted curve in the following equation, where we have denoted the set of sampled points as S and the number of samples as n :

$$S = \left\{ C'(\frac{0}{n}), C'(\frac{1}{n}), \dots, C'(\frac{n}{n}) \right\} \quad (5.5)$$

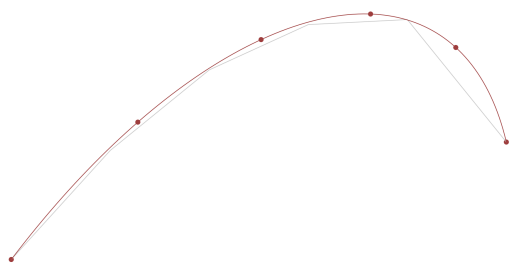
We then need to match each of these sampled points with a point from the original curve. To do so, we first calculate the floor and the ceiling of the x-position of each sampled point. Since the x-positions of the points of the input curve each correspond to an integer frame number, the values given from the floor and ceiling operations provide the indices of the frames that occur directly before and after the sampled point. We obtain the points corresponding to those indices, from the input curve, which we now denote as C_a and C_b (we denote the x-position of the sampled point as $S_{i,x}$):

$$\begin{aligned} C_a &= C_{\lfloor S_{i,x} \rfloor} \\ C_b &= C_{\lceil S_{i,x} \rceil} \end{aligned}$$

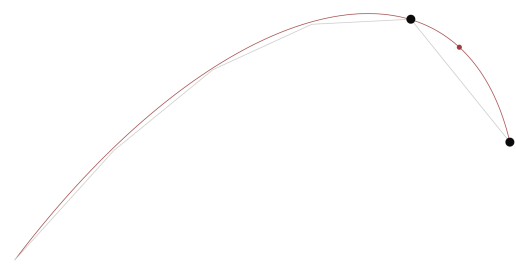
We then use a linear interpolation of the y-values of these two points to obtain a y-value value that represents the point, on the input curve, that occurs at the same time as the sampled point S_i (see Figure 5.4c).

$$O_{i,y} = C_{a,y} + (C_{b,y} - C_{a,y}) \cdot \left(\frac{S_{i,x} - C_{a,x}}{C_{b,x} - C_{a,x}} \right) \quad (5.6)$$

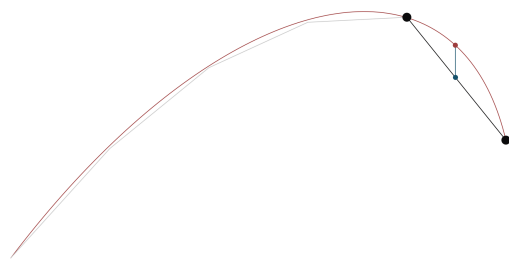
Finally, we can express our complete accuracy term as the sum of squared differences between the y-positions of the points sampled for the fitted curves and those sampled



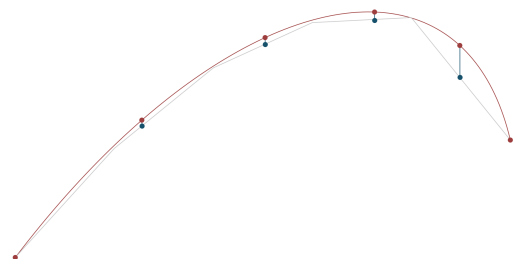
(a) A Fitted Curve



(b) Nearest Points to Sample



(c) Linear interpolation



(d) Accuracy Measure

Figure 5.4: In order to pair each of the samples of the fitted curves with a point from the original curve (a), we first obtain the two points that occur immediately before and after the sampled point (b). We then linearly interpolate these two nearest points to approximate the input curve at the same x-position of the sampled point (d). Thus, we are able to measure accuracy in terms of distances between these pairs of points (d).

in reverse from the input curve (see Figure 5.4d):

$$\Delta_1 = \sum_{i=1}^n (S_{i,y} - O_{i,y})^2 \quad (5.7)$$

As we indicated above, the primary advantage of our reversed-sampling approach is that it leads to a simple derivative. Recall that our formulation is applied to each curve in the chain of fitted curves individually and that the shape of each fitted curve is defined by two control points. The following equation expresses the partial derivative of our accuracy term with respect to the position of the control points (denoted as H_i):

$$\frac{d\Delta_1}{dH_i} = 2 \sum_{i=1}^n f \frac{df}{dH_i} \quad (5.8)$$

In the equation above, f encapsulates the expression $S_{i,y} - O_{i,y}$. The derivative of the expression f with respect to the position of the handles can be expressed in terms of the derivatives of the 2D cubic Bézier interpolation function and the line intersection equation.

5.1.4 Angle Term

With the accuracy term realized, we need only design a term to penalize the occurrence of breaks in geometric continuity through the fitted curves. For this we use a summation of the differences in angle formed between the lines from a keyframe and its two associated control points, which evaluates to zero when those control points form a straight line through the keyframes and increases as they form a sharper corner. The derivative of this term can be obtained from the conventional trigonometric equations and their derivatives.

5.1.5 Time complexity

Optimization methods adjust independent variables (in our case, the location of each control point) in order to minimize the measure prescribed by an objective function. The number of iterations required for convergence depends on the shape of the space produced by the objective function and also the optimization method being used. While the number of steps required for convergence cannot be predicted, we can instead analyse how the time complexity of each step grows with respect to the number of frames in the animation and the number of keyframes used.

Our accuracy term involves calculating the distance between pairs of points in the fitted curves and the input curve (a curve from the motion capture animation). As described in Section 5.1.3, we uniformly sample the fitted curves and then match each of those sampled points with a point from the input curve. Therefore, the time

complexity of a step can be expressed as the number of samples multiplied by the number of segments in the curve.

We can express that the complexity of the accuracy term increases linearly with respect to the number of segments (the number of keyframes minus one) as follows. We denote the number of samples used in each segment as n_s and the cost of sampling the fitted curve as well as performing the linear interpolation step (to obtain the sample's pair) as c_1 :

$$O(n_k - 1) \cdot n_s \cdot c_1$$

The collinearity term evaluates the difference in angle between the two vectors formed between a keyframe and its control points. The first and last keyframe need not be evaluated, as only one control point affects it and therefore the position of the point cannot break geometric continuity. We can express that the complexity of the collinearity term also increases linearly with respect to the number of keyframes as follows. Denoting the cost of measuring the difference in angle between the two vectors as c_2 :

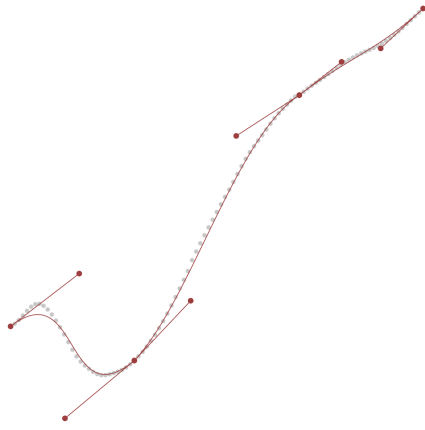
$$O(2(n_k - 2)) \cdot c_2 = O(n_k) \cdot c_2$$

Since the time complexity of both terms increase linearly with respect to the number of keyframes, we conclude that the time complexity of our optimization-based curve fitting technique increases linearly with respect the number of keyframes and that its complexity is invariant with respect to the number of frames in the animation.

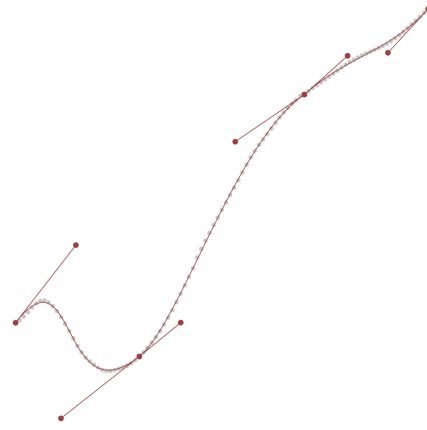
5.2 Results

In this section we present a comparison between the fitting phase of Schneider's technique to ours. The objective of the results presented here is to demonstrate that (1) our technique can produce highly accurate fittings and (2) that our technique can produce smooth curves.

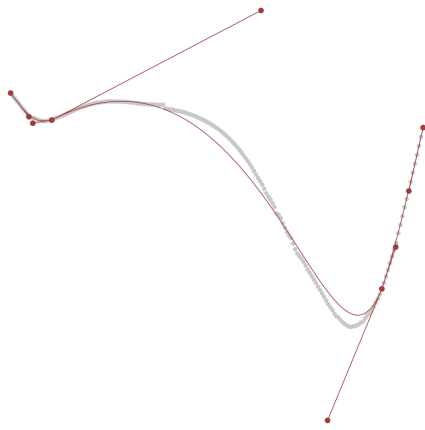
We first present a comparison between Schneider's technique and ours for the example problems given earlier in Figure 5.2. To obtain these results, we provided each curve and the keyframes as input to both techniques. To produce the results presented for Schneider's technique, we applied their tangent calculation and optimization steps as described in [64]. And to produce the results presented for our optimization-based technique we applied the BFGS algorithm, provided via the



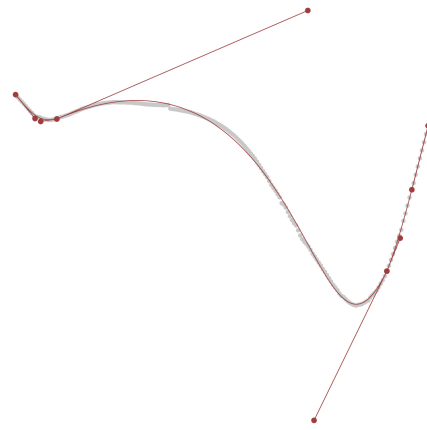
(a) Schneider's



(b) Ours



(c) Schneider's



(d) Ours

Figure 5.5: A comparison between the fittings provided by Schneider's technique (a and c) and ours (b and d) for the two examples provided earlier in Figure 5.2.

SCI-PY library, to optimize our dual-term objective function. We used a weighting parameter of $\omega_1 = 1$ for the accuracy term and a weighting parameter $\omega_2 = 1000$ for the angle term. A reference implementation, used to create the results presented in this section, can be viewed online ¹.

We visualize the keyframes and their corresponding control points using a red line connecting three circles. The middle point represents the keyframe and the end points represent its control points. The input motion capture curve is displayed in grey and the chain of fitted 2D cubic Bézier curves in red. As depicted in Figure 5.5, our curve fitting technique fits both of the input curves more accurately.

We now present and describe the results of both techniques applied to fit curves from motion capture. To obtain each result we chose one of the six motion capture animations at random and then applied Salient Poses to obtain a selection of keyframes. We then chose one dimension from the motion capture data, again at random, and applied both techniques.

Figure 5.6 present the results of applying the two fitting techniques to a relatively simple curve, from the hip joint in the PITCH animation. Both techniques approximate the overall shape of curve well, but the fitting produced by Schneider’s technique does not preserve the shape of the first rise in the curve (frames 100-240). Furthermore our technique is able to preserve the shape of the twist that occurs near frame 335.

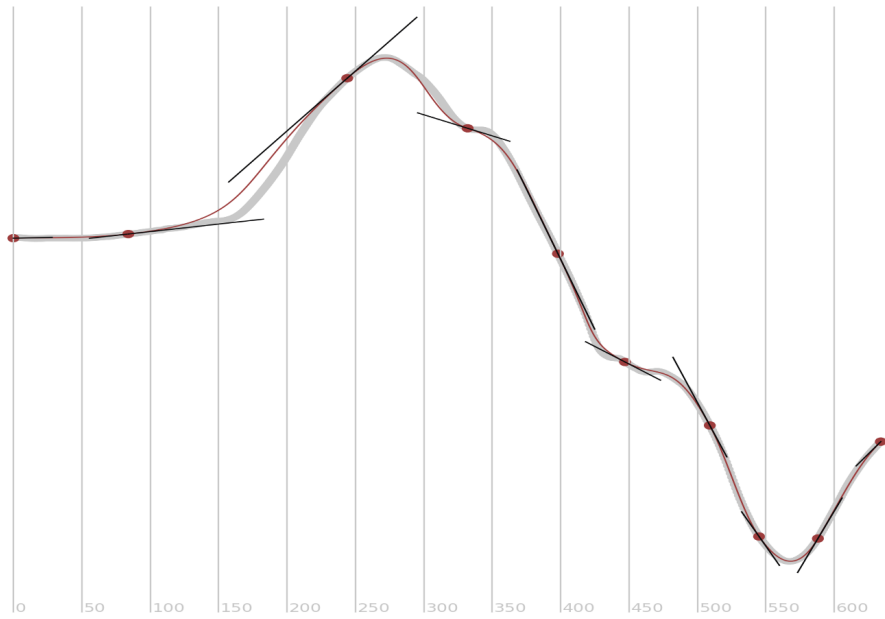
Figure 5.7 features a curve from the hip joint of the CARTWHEEL animation. Since our technique is able to move the control points freely, namely the keyframe occurring at frame 304, we are able to fit the first section of the curve more accurately (frames 220 - 360) as well as the trough that occurs following the large peak (frames 480 - 520).

Figure 5.8 presents the fittings for a curve from the right foot of the LAY UP motion capture. Medium scale details can be seen in the earlier part of the curve and two large arcs in the later part. Both techniques produced accurate fittings for the larger arcs in the curve (frames 260 - 420), but Schneider’s technique was not able to preserve the medium scale details that occur earlier in the curve (frames 90 - 230). Our technique was able to do so because it has the ability to move the control points freely.

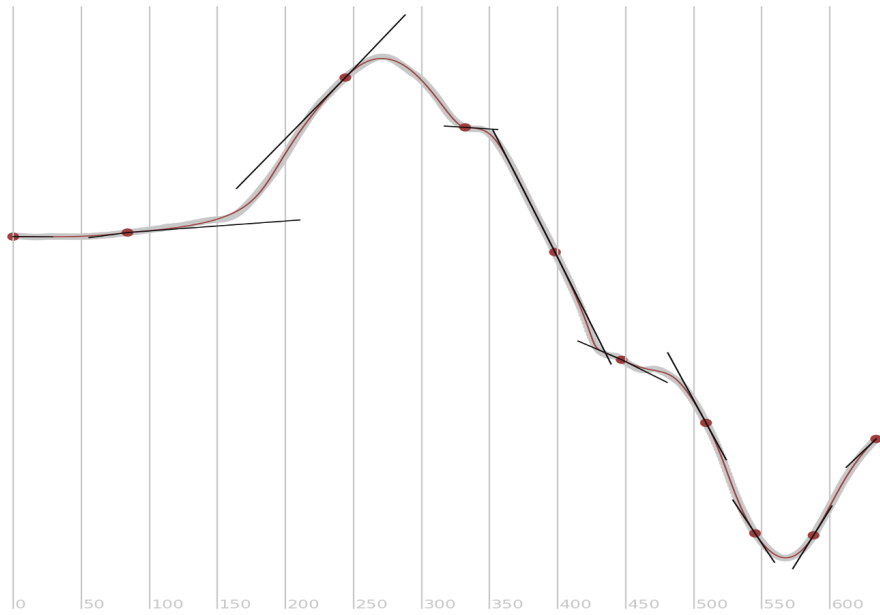
In a similar way, our technique was able to fit the finer scale details in Figure 5.9 more accurately than Schneider’s technique. Differences between the fittings are most notable surrounding the denser clusters of keyframes that occur in the troughs of the input curve (frames 50-150, 190-270, and 320-400). Again, our technique is able to better fit the animation’s curve through these regions because the control points can be moved freely.

Finally, as a stress test, we applied both techniques to fit a noisy animation curve.

¹See the INTERPOLATOR tool provided by the MOCAPPIE reference implementation: <http://github.com/richard-roberts/PhD>.

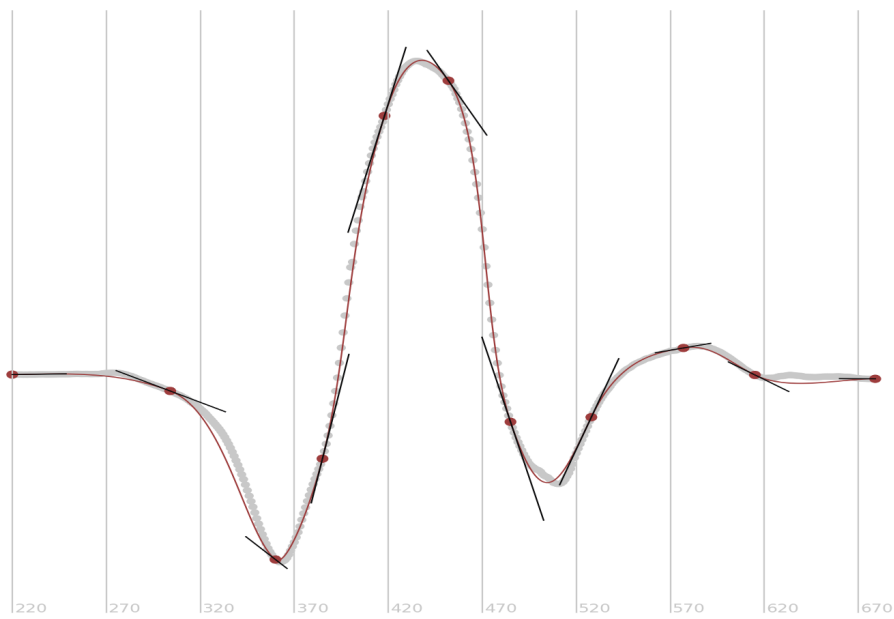


(a) Schneider's

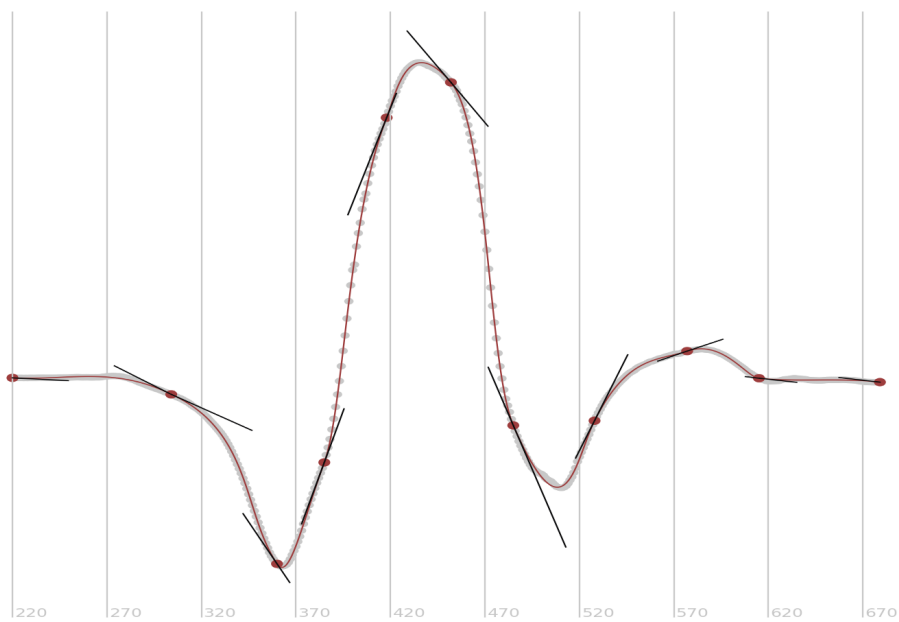


(b) Ours

Figure 5.6: The result of applying the two techniques to fit a curve from the PITCH motion capture animation.

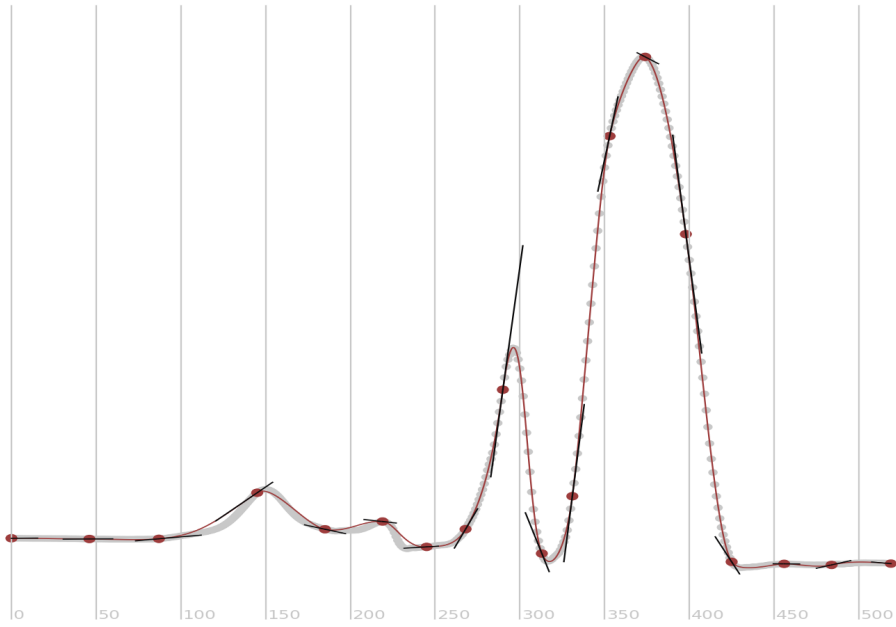


(a) Schneider's

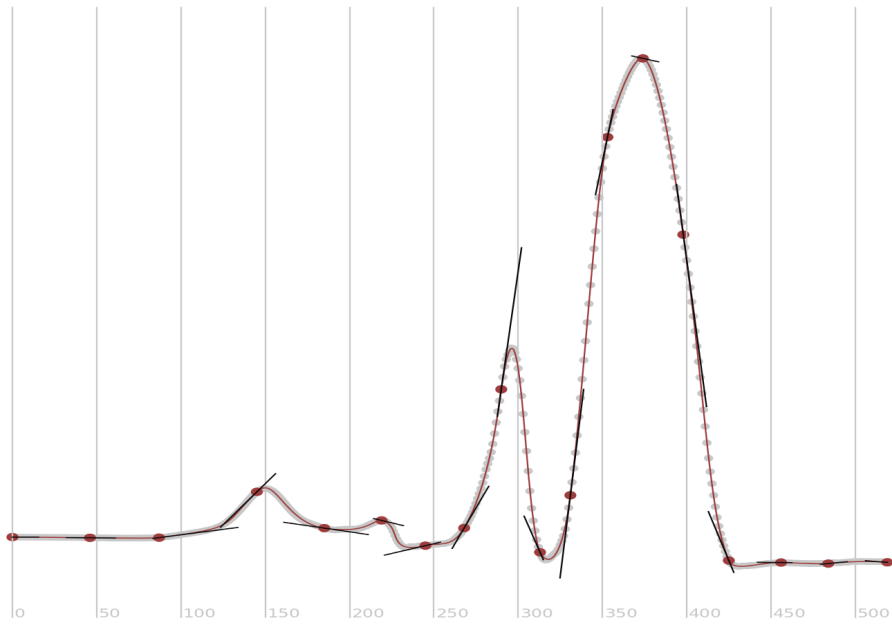


(b) Ours

Figure 5.7: The result of applying the two techniques to fit a curve from the CARTWHEEL motion capture animation.

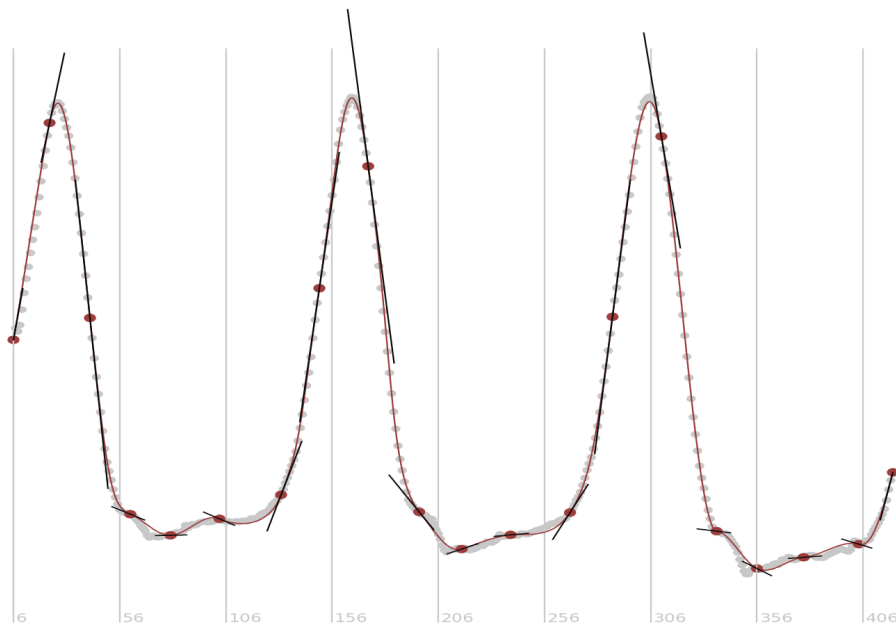


(a) Schneider's

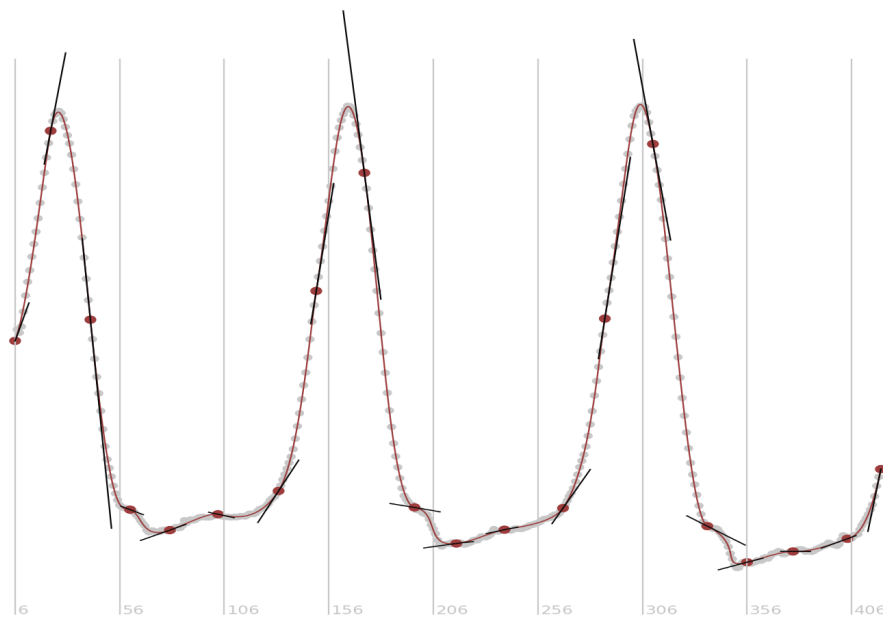


(b) Ours

Figure 5.8: The result of applying the two techniques to fit a curve from the LAY UP motion capture animation.

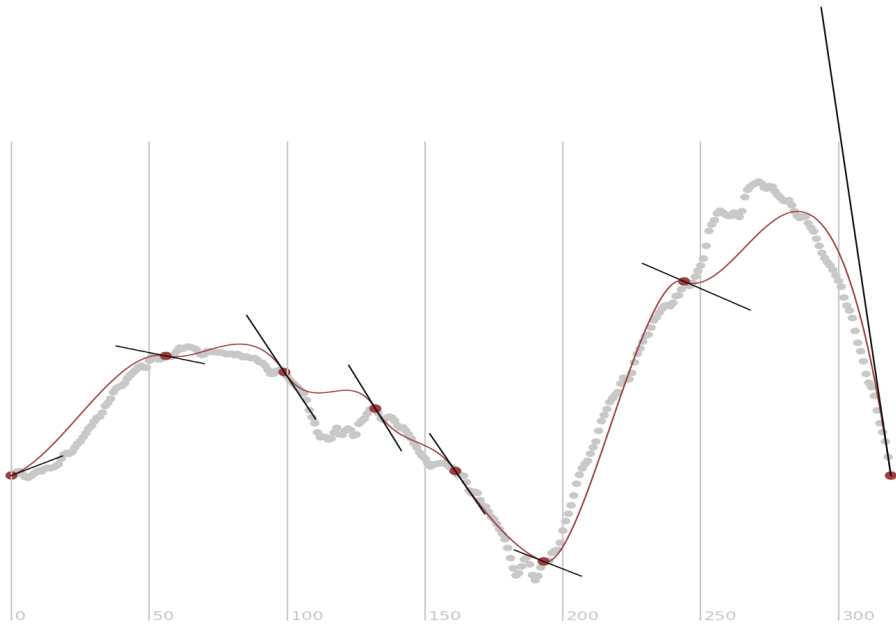


(a) Schneider's

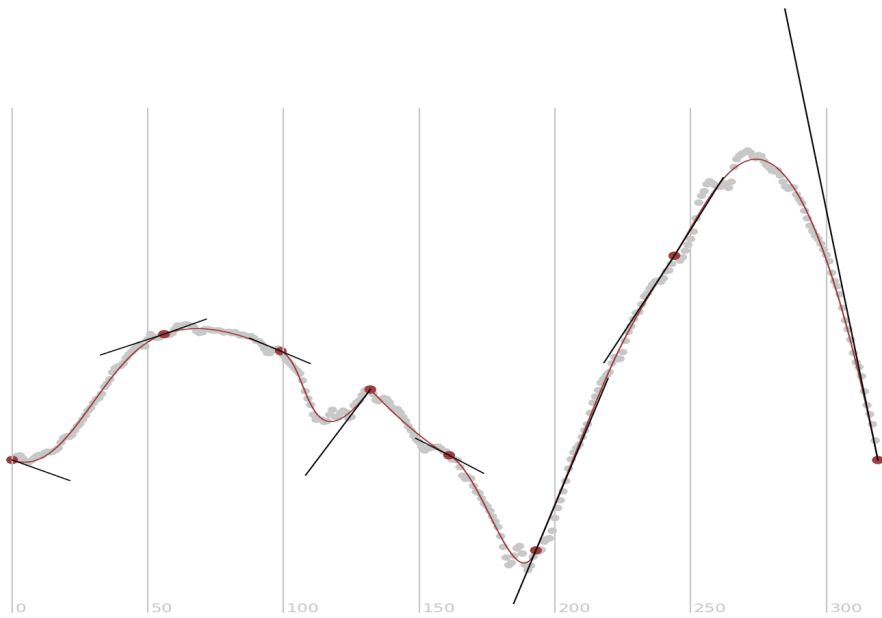


(b) Ours

Figure 5.9: The result of applying the two techniques to fit a curve from the WALK motion capture animation.



(a) Schneider's



(b) Ours

Figure 5.10: The result of applying the two techniques to fit a curve from the JUMP motion capture animation.

As depicted by Figure 5.10, Schneider’s technique cannot approximate the curve effectively, in part due to the presence of noise in the curve (which disrupts the computation of the tangent information). By comparison, our technique is more robust against higher levels of noise and is still able to approximate most of the curve well.

An undesirable break in geometric continuity can be seen surrounding the keyframe that occurs at frame 307. Our technique allowed the break because doing so significantly reduced the value prescribed by the accuracy term.

5.3 Summary

In this chapter we have presented an optimization-based technique for fitting an animation’s curves accurately. We have designed our technique specifically for interpolating the selections given by Salient Poses in a way that accurately preserves the larger, medium, and finer scales of detail from the original animation; our technique is able to move the control points of each of the fitted curves freely to ensure the higher level of accuracy. With the ability to interpolate keyframes in a way that accurately approximates each curve of the original motion, we have the final step required to convert motion capture into editable keyframe animation.

The primary contribution of our curve fitting technique is the design of two terms used in its objective function. The accuracy term quantifies how well the fitted curves approximate the original motion capture animation’s curve, while the angle term penalizes breaks in geometric discontinuity. Finally, as a secondary contribution, both the terms feature a simple symbolic derivative that optimization methods can utilize to reduce computation time by up to an order of magnitude (in particular, we achieve this by replacing the typical root-finding operation with our reverse sampling approach).

The primary limitation of our technique is that the numerical optimization methods that we rely on can be computationally expensive. Especially in cases where keyframes are clustered closely together, the space given by our objective function can be non-linear and consequently difficult for the optimization method to resolve. As such, we propose that our curve fitting technique is best applied to create an editable animation after the motion editor has decided upon a particular selection. In practice, a motion editor might browse the selection space while using Schneider’s technique to preview each alternative selection in real time. Once they have identified a selection suited to their editing task, our optimization-based curve fitting technique can be employed to construct the final animation. Alternatively, if more intensive engineering efforts are possible a faster implementation of an optimization method could be explored,

perhaps by taking advantage of asynchronously computation via the graphics card (the problems of fitting each of the animation's curves are independent).

Chapter 6

Conclusion

In this thesis, we have focused on the problem that motion capture is difficult to edit and have proposed a technique that can automatically recover editable animation. We began in Section 2.1, where we examined conventional animation practice in an effort to define editable animation. We found that animators employ keyposes and breakdowns in order to construct blocked animation, in which they purposefully retain a sparse set of keyframes to ensure that their animations can be edited using fewer adjustments. Based on these ideas, in particular those pertaining to the pose to pose animation convention, we proposed to solve the problem that motion capture is difficult by offering a technique that rebuilds motion capture as though it had been blocked out by an animator using keyposes and breakdowns.

In order to realize our solution we proposed to design a method that can select keyframes to optimize a generic objective function. In order to achieve the optimal selection without incurring significant computational costs we needed to avoid checking all possible solutions (which increase combinatorially with respect to the numbers of frames in the given animation). Through Chapter 3 we proposed a new graph-based representation of the keyframe selection problem, which can be used to significantly reduce the number of computations required to ensure optimal selections. Our decomposition lead to Salient Poses; the first fast, generic, optimal, and deterministic keyframe selection technique.

We continued into Chapter 4, where we applied Salient Poses specifically to the problem of selecting keyframes that are similar in structure to the keyposes and breakdowns used by animators. To do so we introduced our three layer technique, in which each layer used a different objective function. To conclude the chapter, we demonstrated how sparse and well distributed sets of keyframes could be realized by composing together selections from those layers, where we used our threshold-based algorithm to automatically choose selections from the selection spaces that resulted from each layer.

We concluded in Chapter 5, where we described our optimization-based curve fitting technique. The technique was designed specifically to excel at fitting an interpolation of the selected keyframes in a way that preserves the detail of the original motion. To conclude the chapter, we demonstrated that our technique enables an animation’s curves to be fitted with a high level of accuracy.

In summary, we can convert motion capture into editable keyframe animation using our keyframe selection technique together with our curve fitting technique. Returning the context of the problem that we outlined in the beginning, our approach enables an editor at a visual effects studio to edit motion capture easily:

- the editor first loads the motion capture into their editing software of choice, from which the software automatically animates either a performance rig or a virtual character,
- the editor selects the leading part and then opens the dopesheet interface to identify a segment to edit before applying Salient Poses to generate a selection space,
- the editor explores the selection space manually, or otherwise applies the threshold-based algorithm, and finally
- the editor applies our fitting technique to create the editable animation.

After completing the steps listed above, a new animation is available to the editor; the editor can adjust this new animation in the same way that animators adjust their own keyframe animations.

In terms of the interface we expect that a typical slider device that describes a number of keyframes, to be used by in the new animation, would be most useful for exploring the space of potential selections. As the editor moves the slider back and forth, the dopesheet and the viewport of the software would display the keyframes interactively (perhaps using silhouette-style renders in the viewport). By using this type of an interface, an editor can explore how the distribution of keyframes changes in relation to both the number of keyframes (via the dopesheet) and also the poses themselves (via the viewport). When no appropriate selections are found, the editor can either change the lead part or ask a developer to help configure the objective functions used by Salient Poses. An alternative interfaces might visualize the selection space using the graphical style of presentation that we used in Section 3.2.

Finally, we have intentionally designed Salient Poses and our curve fitting technique to be simple to implement. In particular, neither of these techniques require specialized knowledge to implement: dynamic programming algorithm are

taught in most computer science curriculums and numerical optimization techniques are freely available in some scientific computing libraries (such as Python's `SciPy`).

6.1 Summary of Contributions

Keyframe Selection

The central contribution of this thesis was a new fast, generic, optimal, and deterministic keyframe selection technique. In Chapter 3 we described how we represented the problem of optimal keyframe selection using a graph. The graph representation was pivotal in enabling our technique to be both fast and optimal. Specifically, from the graph representation we were able to decompose the problem into partial graphs. By iteratively composing together optimal selections of keyframes for smaller graphs and reusing those selections in larger graphs we were able to significantly reduce the amount of computation required to obtain an optimal selection space. Additionally, in Section 3.1.4, we provided insight into exactly how many traversals must be visited to form any optimal selection.

We later applied Salient Poses to the problem of recovering blocked animation from motion capture (Chapter 4). In particular, we developed three implementations of Salient Poses's objective function that enabled:

- keyframes to be selected to divide the motion into segments based on fluctuation among the curves of the animation that correspond to the leading part,
- keyframes to be selected based on the occurrence of extremes present among the curves of the animation that correspond to the leading part, and
- keyframes to be selected in order to ensure that the animation's curves can be reconstructed accurately using a curve-based interpolation.

Finally, we introduced our simple threshold algorithm to automate the choice of selection from a selection space. With the algorithm, much of our technique for selecting editable keyframes can be performed automatically.

Contributions to Interpolation

The secondary contribution of this thesis was the optimization-based curve fitting technique. We designed the technique especially for the problem of reconstructing a new animation, from only the given keyframes, in a way that preserves most of the detail of the original animation.

In Chapter 5, we introduced the dual-term objective function that composes together the notions of accuracy and collinearity.

Accuracy was expressed using a typical sum of squared Euclidean distance measure. However, where curve fitting techniques usually employ an expensive root-finding step to compare the original and fitted curves, we proposed a new alternative approach: we uniformly sample each fitted curve segment and find the nearest points by interpolating the original curve to find points that occur at the same time as the samples. Our reversed sampling approach avoids the need to perform the expensive root-finding step.

The collinearity term was a simple comparison between the angle of the two vectors from each keyframe to its control points. When weighted effectively this additional collinearity term encourages, but does not enforce, geometric continuity between the fitted curves.

Finally both of the terms lead to a simple symbolic derivative, which meant that our curve fitting technique was well suited to the type of numerical optimization methods that can take advantage of derivative information.

6.2 Summary of Limitations

Application to Larger Problems

The most significant limitation to Salient Poses is that it cannot be applied to select keyframes for long sequences of motion capture. While our graph representation could reduce the amount of computation required for keyframe selection (see Section 3.1.1 and Section 3.1.2), too many traversals need to be visited in longer sequences of animation and therefore Salient Poses eventually becomes too slow for editing purposes. In particular, we can extrapolate from ?? that the execution time required to compute the selection space (for a compression level of 80%-100%) increases beyond one second when the animation contains approximately 1000 frames and, furthermore, that the execution time would increase sharply thereafter. The sharp increase in execution time is due to the combinatorial increase in the number of traversals that need to be visited. Consequently, we suggest that applying Salient Poses directly to problems that are larger than the average shot-length is impractical, especially for editing purposes in which the editor should be able to work interactively with their software.

It is not a significant problem that Salient Poses cannot be applied to larger problems without demanding more extensive computation. The adjustments performed by motion editors are likely to be within the scope of the average shot length, in which case the limitation is irrelevant. Furthermore, in cases where the

editor requires keyframe selections for larger sequences of animation, we can employ a detection-based technique. Recall from Section 2.2.1 that detection-based techniques excel at selecting keyframes to coarsely dividing large animation into smaller segments, are fast, and are also simple to implement. Consequently, we can still realize a fast solution using Salient Poses by employing a detection-based technique to pre-process larger sequences into shot-sized segments. By applying Salient Poses to each of the resulting segments, rather than the entire sequence, we avoid suffering an increase in execution time.

Poor Fitting for Very Sparse Selections

Our curve fitting technique is designed to be able to interpolate a given selection of keyframes in a way that accurately approximates the original motion.

In some cases, our technique produces fittings that do not preserve geometric continuity, even when they are desirable. For example, in Figure 5.10, the animation’s curve was smooth and so it would have been better if our technique also produced a smoother interpolation. It may be possible to develop additional terms for the algorithm’s objective function, perhaps using information about higher order derivatives of the curve being fitted, that encourage the algorithm to discard poorly fitted segments and replace them with a smoother alternative. Unfortunately such additional terms would complicate the space traversed by optimization method and, consequently, the technique would require a significant number of steps before converging to a solution.

We propose that the occasional breaks in geometric continuity produced by our algorithm are not a significant drawback. To resolve any segments in which the curve has been fitted poorly, a motion editor can choose to add a keyframe manually or can choose another selection that features a higher level of accuracy; once the fitting technique has been reapplied to the new set of keyframes, the problem will likely be resolved.

6.3 Future Work

The most interesting avenue for future work in keyframe selection would be to explore other types of information that can be used as part of the criteria to which keyframes are selected. For example, Ho et al. proposed a retargeting technique that preserves not only movements of the character but also the shape of the negative space between that character and the scene [33]. Salient Poses could be applied to select keyframes with respect to the negative space by expressing qualities of that negative space through

an objective function. Other criteria that may be interesting to keyframe selection may include information regarding the timing between poses (explored in [6]) and also alternative descriptions of a character's pose (explored in [16, 31, 34]).

Ultimately, an entire library of the objective functions for Salient Poses could be developed. Each function could be crafted for a particular purpose, and then composed together to recover keyframes for all kinds of qualities about an animation; not only for body movements but also facial and hand animation, physics and fluid simulation, and beyond.

Appendices

Appendix A

Simple Curve Approximation

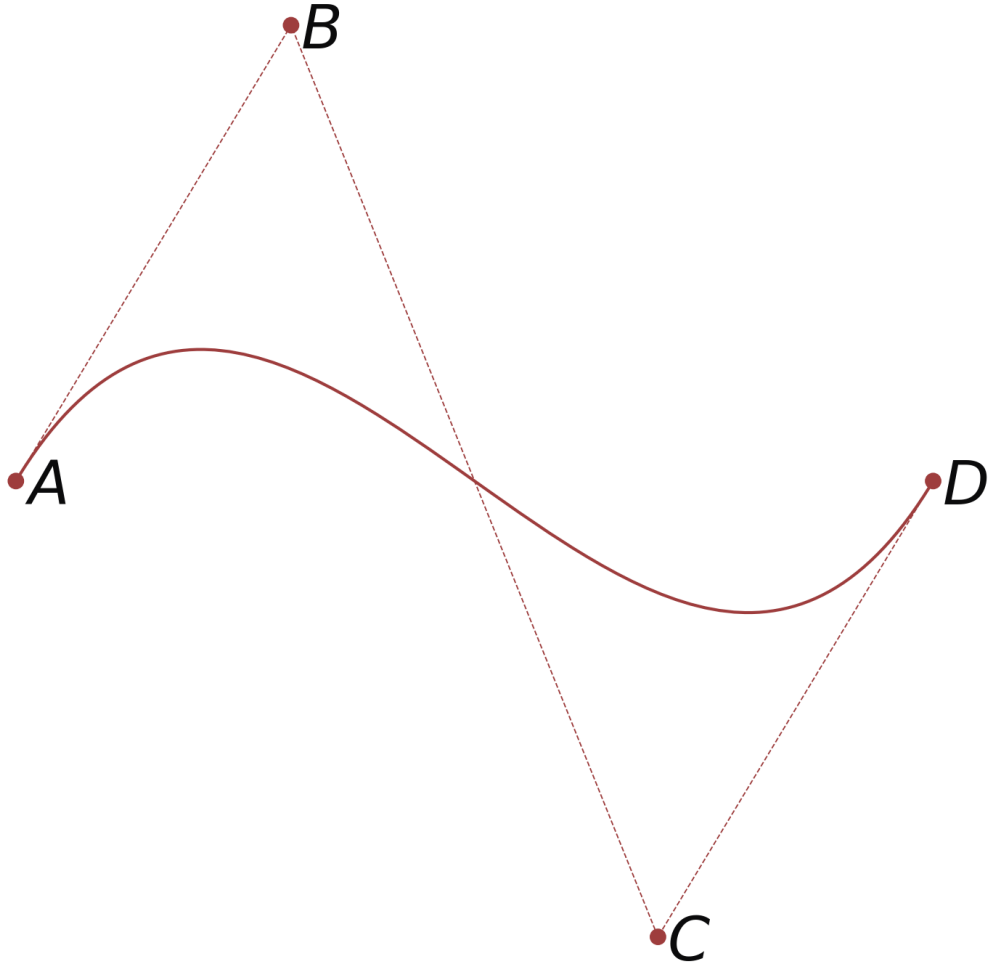
2D cubic Bézier curves are defined as a weighted blend of four control points, as illustrated in Figure A.2. In practice, these curves are functions that are sampled with a single parameter. Equation (A.1) presents the formula for the 2D cubic Bézier function, in which the sampling parameter is denoted as u :

$$f(u) = (u - 1)^3 A + 3(u - 1)^2 u B + 3(u - 1) u^2 C + u^3 D \quad (\text{A.1})$$

To realize a curve-based approximation of a set of points, we can choose B and C in such a way that their curve is as close as possible to the given points, as illustrated by Figure A.2. More precisely, we choose the position of B and C to minimize the sum of squared Euclidean distances between the given points and a uniform interpolation of the curve. This minimization is expressed in Equation (A.2), in which n denotes the number of given points and P_i denotes one of the given points.

$$\min_{B,C} = \sum_{i=1}^n \left\| P_i - f\left(\frac{i}{n-1}\right) \right\|^2 \quad (\text{A.2})$$

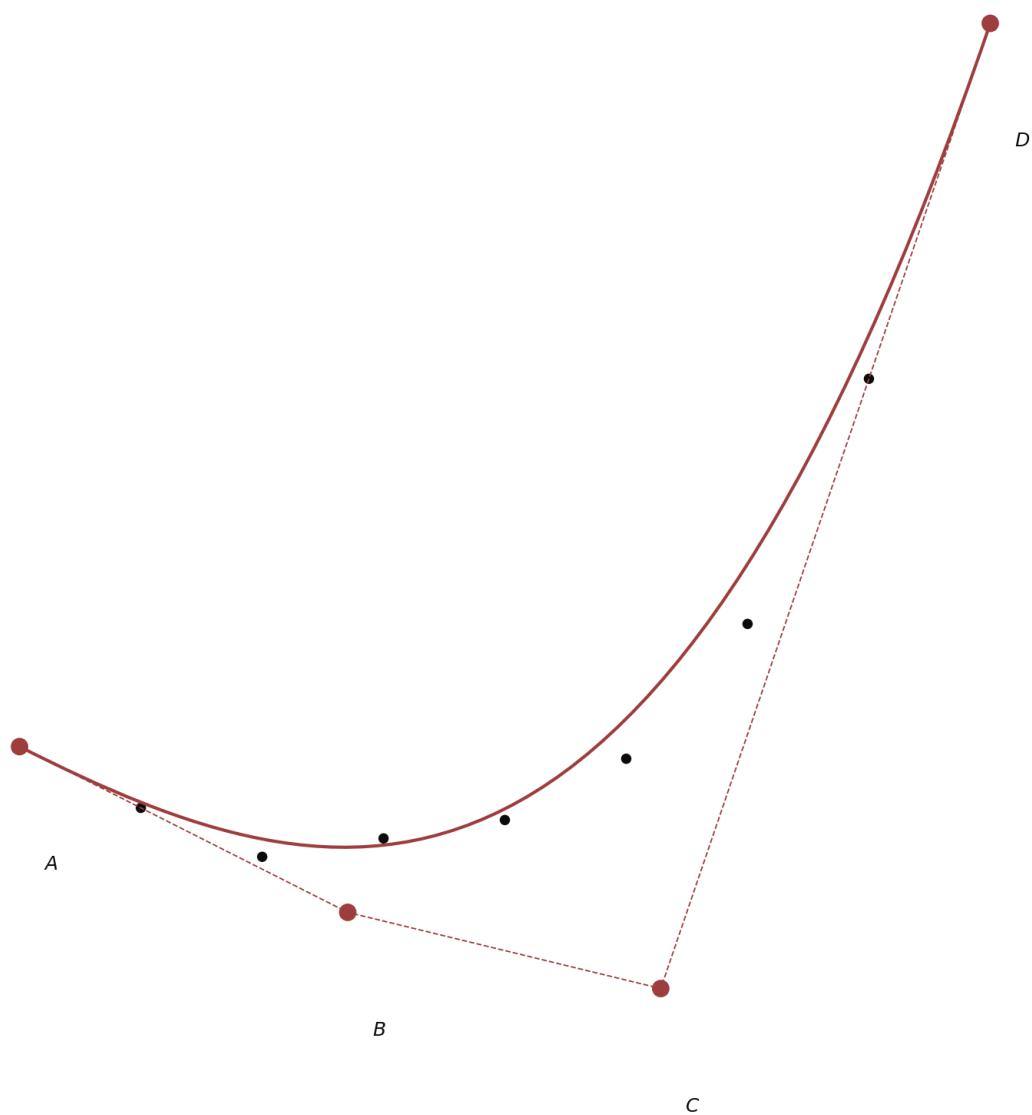
Equation (A.2) can be formulated as a set of simultaneous equations and can easily be solved using standard numerical techniques such as matrix inversion.



(a) A Cubic Bézier Curve

(b) $u = \frac{0}{4}$ (c) $u = \frac{1}{4}$ (d) $u = \frac{3}{4}$ (e) $u = \frac{4}{4}$

Figure A.1: 2D cubic Bézier curves are defined by two points that must be interpolated (denoted A and D) and two control points (denoted B and C). The curve can be sampled through the interval $[0, 1]$ to produce points along the curve. Figure A.1b to Figure A.1e present some examples for different values of the sampling parameter, which is denoted by u .



(a) A Cubic Bézier Curve

Figure A.2: We fit a set of given points by choosing the control points, denoted B and C , in such a way that the distance between each point and the curve is minimized.

Bibliography

- [1] ANJYO, K. Personal Communication, Feb. 2016.
- [2] ASSA, J., CASPI, Y., AND COHEN-OR, D. Action Synopsis: Pose Selection and Illustration. *Transactions on Graphics* 24, 3 (July 2005), 667–676.
- [3] ASSA, J., COHEN-OR, D., YEH, I.-C., AND LEE, T.-Y. Motion Overview of Human Actions. *Transactions on Graphics* 27, 5 (Dec. 2008), 115:1–115:10.
- [4] AUTODESK. Maya. <https://www.autodesk.com/products/maya/overview>.
- [5] AUTODESK. MotionBuilder. <https://www.autodesk.com/products/motionbuilder/overview>, 2016.
- [6] BAAK, A., MÜELLER, M., AND SEIDEL, H.-P. An Efficient Algorithm for Keyframe-based Motion Retrieval in the Presence of Temporal Deformations. In *Proceedings of the 1st ACM International Conference on Multimedia Information Retrieval* (2008), Association for Computing Machinery, pp. 451–458.
- [7] BLENDER FOUNDATION. Blender. <https://www.blender.org>.
- [8] BLENDER INSTITUTE. Blender Cloud. <https://cloud.blender.org>, 2017.
- [9] BRUDERLIN, A., AND WILLIAMS, L. Motion Signal Processing. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques* (1995), Association for Computing Machinery, pp. 97–104.
- [10] BUCKLAND, A., AND BUCKLAND, W. *Film Theory and Contemporary Hollywood Movies*. Taylor & Francis, 2009.
- [11] BUCKLEY, A. *Pixar: The Company and Its Founders*. ABDO Publishing Company, 2011.
- [12] BULUT, E., AND CAPIN, T. Key Frame Extraction from Motion Capture Data by Curve Saliency. 2007.
- [13] CAMERON, J. Avatar, Dec. 2009.

- [14] CARNEGIE MELLON UNIVERSITY. Motion Capture Database. <http://mocap.cs.cmu.edu/>, 2016.
- [15] CHANG, X., YI, P., AND ZHANG, Q. *Key Frames Extraction from Human Motion Capture Data Based on Hybrid Particle Swarm Optimization Algorithm*. Springer International Publishing, 2016, pp. 335–342.
- [16] CHEN, C., ZHUANG, Y., NIE, F., YANG, Y., WU, F., AND XIAO, J. Learning a 3D Human Pose Distance Metric from Geometric Pose Descriptor. *IEEE Transactions on Visualization and Computer Graphics* 17, 11 (Nov 2011), 1676–1689.
- [17] COOPER, M., AND FOOTE, J. Summarizing Video using Non-negative Similarity Matrix Factorization. In *IEEE Workshop on Multimedia Signal Processing* (Dec. 2002), pp. 25–28.
- [18] CUNTOOR, N. P., AND CHELLAPPA, R. Key Frame-based Activity Representation Using Antieigenvalues. In *Asian Conference on Computer Vision* (2006), pp. 499–508.
- [19] DOULAMIS, N. D., DOULAMIS, A. D., AVRITHIS, Y., AND KOLLIAS, S. D. A Stochastic Framework for Optimal Key Frame Extraction from MPEG Video Databases. In *IEEE 3rd Workshop on Multimedia Signal Processing* (Sept. 1999), pp. 141–146.
- [20] DU SEL, Y. P., CHAVEROU, N., AND ROUILLÉ, M. Motion Retargeting for Crowd Simulation. In *Proceedings of the 2015 Symposium on Digital Production* (2015), Association for Computing Machinery, pp. 9–14.
- [21] FEISZLI, M., AND JONES, P. W. Curve Denoising By Multiscale Singularity Detection And Geometric Shrinkage. *Applied and Computational Harmonic Analysis* 31, 3 (2011), 392–409.
- [22] FENG, Y., XIAO, J., ZHUANG, Y., YANG, X., ZHANG, J. J., AND SONG, R. Exploiting Temporal Stability and Low-rank Structure for Motion Capture Data Refinement. *Information Sciences* 277 (2014), 777–793.
- [23] FRANK THOMAS, O. J. *The Illusion of Life : Disney Animation*. Hyperion, 1995.
- [24] GLEICHER, M. Retargetting Motion to New Characters. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (1998), Association for Computing Machinery, pp. 33–42.

- [25] GLEICHER, M. Animation from Observation: Motion Capture and Motion Editing. *SIGGRAPH* 33, 4 (Nov. 1999), 51–54.
- [26] GLEICHER, M. Motion Path Editing. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics* (2001), Association for Computing Machinery, pp. 195–202.
- [27] GONG, Y., AND LIU, X. Video Summarization using Singular Value Decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (June 2000), vol. 2, pp. 174–180.
- [28] HALIT, C., AND CAPIN, T. Multiscale Motion Saliency for Keyframe Extraction from Motion Capture Sequences. *Computer Animation and Virtual Worlds* 22, 1 (2011), 3–14.
- [29] HAND, D. Snow White and the Seven Dwarfs, Dec. 1937.
- [30] HAND, D. Bambi, Aug. 1942.
- [31] HAUBERG, S., AND STEENSTRUP PEDERSEN, K. Spatial measures between human poses for classification and understanding. In *Proceedings of the 7th International Conference on Articulated Motion and Deformable Objects* (2012), Springer-Verlag, pp. 26–36.
- [32] HECKBERT, P. S., AND GARLAND, M. Survey of Polygonal Surface Simplification Algorithms. Tech. rep., DTIC Document, 1997.
- [33] HO, E. S. L., KOMURA, T., AND TAI, C.-L. Spatial Relationship Preserving Character Motion Adaptation. *Transactions on Graphics* 29, 4 (July 2010), 33:1–33:8.
- [34] HOU, J., CHAU, L. P., MAGNENAT-THALMANN, N., AND HE, Y. Human Motion Capture Data Tailored Transform Coding. *IEEE Transactions on Visualization and Computer Graphics* 21, 7 (July 2015), 848–859.
- [35] HU, Y., WU, S., XIA, S., FU, J., AND CHEN, W. Motion Track: Visualizing Variations of Human Motion Data. In *IEEE Pacific Visualization Symposium* (Mar. 2010), pp. 153–160.
- [36] HUANG, K.-S., CHANG, C.-F., HSU, Y.-Y., AND YANG, S.-N. Key Probe: A Technique For Animation Keyframe Extraction. *The Visual Computer* 21, 8 (2005), 532–541.

- [37] IKEMOTO, L., ARIKAN, O., AND FORSYTH, D. Knowing when to Put Your Foot Down. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games* (2006), Association for Computing Machinery, pp. 49–53.
- [38] ITTI, L., KOCH, C., AND NIEBUR, E. A Model of Saliency-Based Visual Attention for Rapid Scene Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20, 11 (Nov. 1998), 1254–1259.
- [39] JACKSON, P. The Lord of the Rings, Dec. 2001. Trilogy.
- [40] JACKSON, P. King Kong, Dec. 2005.
- [41] JACKSON, P. The Hobbit, Nov. 2012. Trilogy.
- [42] JIN, C., FEVENS, T., AND MUDUR, S. Optimized Keyframe Extraction for 3D Character Animations. *Computer Animation and Virtual Worlds* 23, 6 (2012), 559–568.
- [43] KAUFMAN, J., AND SIMONTON, D. *The Social Science of Cinema*. Oxford University Press, 2014.
- [44] KIM, M. H., CHAU, L. P., AND SIU, W. C. Keyframe Selection for Motion Capture using Motion Activity Analysis. In *IEEE International Symposium on Circuits and Systems* (May 2012), pp. 612–615.
- [45] KOJIMA, H. Metal Gear Solid, Nov. 2001. Series.
- [46] KOVAR, L., GLEICHER, M., AND PIGHIN, F. Motion Graphs. *Transactions on Graphics* 21, 3 (July 2002), 473–482.
- [47] LASSETER, J. Principles of Traditional Animation Applied to 3D Computer Animation. *SIGGRAPH* 21, 4 (Aug. 1987), 35–44.
- [48] LEWIS, J. P. Personal Communication, Mar. 2017.
- [49] LEWIS, J. P., AND ANJYO, K. Identifying Salient Points. In *SIGGRAPH Sketches* (2009), Association for Computing Machinery, p. 41:1.
- [50] LIM, I. S., AND THALMANN, D. Key-posture Extraction Out of Human Motion Data. In *Proceedings of the 23rd Annual International Conference of the IEEE* (2001), vol. 2, pp. 1167–1169.
- [51] LIU, F., ZHUANG, Y., WU, F., AND PAN, Y. 3D Motion Retrieval with Motion Index Tree. *Computer Vision and Image Understanding* 92, 2-3 (Nov. 2003), 265–284.

- [52] LIU, X.-M., HAO, A.-M., AND ZHAO, D. Optimization-based Key Frame Extraction for Motion Capture Animation. *The Visual Computer* 29, 1 (2013), 85–95.
- [53] LOWE, D. Animation Bootcamp: The 'Animate' Button. Game Developers Conference, 2016.
- [54] MIURA, T., KAIGA, T., KATSURA, H., TAJIMA, K., SHIBATA, T., AND TAMAMOTO, H. Adaptive Keypose Extraction from Motion Capture Data. *Journal of Information Processing* 22, 1 (2014), 67–75.
- [55] MIURA, T., KAIGA, T., SHIBATA, T., KATSURA, H., TAJIMA, K., AND 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* 9, 6 (2014), 697–699.
- [56] MONZANI, J.-S., BAERLOCHER, P., BOULIC, R., AND THALMANN, D. Using an Intermediate Skeleton and Inverse Kinematics for Motion Retargeting. In *Computer Graphics Forum* (2000), vol. 19, Wiley Online Library, pp. 11–19.
- [57] ONDER, O., GUDUKBAY, U., OZGUC, B., ERDEM, T., ERDEM, C., AND OZKAN, M. Keyframe Reduction Techniques for Motion Capture Data. In *The True Vision - Capture, Transmission and Display of 3D Video* (May 2008), pp. 293–296.
- [58] PARK, M. J., AND SHIN, S. Y. Example-based Motion Cloning. *Computer Animation and Virtual Worlds* 15, 3-4 (2004), 245–257.
- [59] PRAMAGGIORE, M., AND WALLIS, T. *Film: A Critical Introduction*. Laurence King, 2005.
- [60] PULLEN, K., AND BREGLER, C. Motion Capture Assisted Animation: Texturing and Synthesis. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (2002), Association for Computing Machinery, pp. 501–508.
- [61] RAMER, U. An Iterative Procedure for the Polygonal Approximation of Plane Curves. *Computer Graphics and Image Processing* 1, 3 (1972), 244 – 256.
- [62] REEVES, W. T., OSTBY, E. F., AND LEFFLER, S. J. The Menv Modelling and Animation Environment. *The Journal of Visualization and Computer Animation* 1, 1 (1990), 33–40.

- [63] ROY, K. *How to Cheat in Maya 2014 : Tools and Techniques for Character Animation*. Focal Press, Burlington, MA, 2013.
- [64] SCHNEIDER, P. J. Graphics Gems. Academic Press Professional, 1990, ch. An Algorithm for Automatically Fitting Digitized Curves, pp. 612–626.
- [65] SHARPSTEEN, B. Dumbo, Oct. 1941.
- [66] SO, C. K. F., AND BACIU, G. Entropy-based Motion Extraction for Motion Capture Animation. *Computer Animation and Virtual Worlds* 16, 3-4 (2005), 225–235.
- [67] STERN, G. *Garland’s Animation System (Gas) a System for Computer-aided Keyframe Animation*. PhD thesis, 1978. AAI7822833.
- [68] SUJATHA, C., AND MUDENAGUDI, U. A Study on Keyframe Extraction Methods for Video Summary. In *International Conference on Computational Intelligence and Communication Networks* (Oct. 2011), pp. 73–77.
- [69] TABATA, H. Final Fantasy XV, Nov. 2016.
- [70] TOGAWA, H., AND OKUDA, M. Position-Based Keyframe Selection for Human Motion Animation. In *11th International Conference on Parallel and Distributed Systems* (July 2005), vol. 2, pp. 182–185.
- [71] WANG, X., CHEN, Q., AND WANG, W. 3D Human Motion Editing and Synthesis: A Survey. *Computational and mathematical methods in medicine* 11 (2014).
- [72] WHITE, T. *Animation from Pencils to Pixels : Classical Techniques for Digital Animators*. Focal Press, Burlington, MA Oxford, 2006.
- [73] WILLIAMS, R. *The Animator’s Survival Kit*. Faber, 2001.
- [74] WITKIN, A. P. Scale-space Filtering. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (1983), Morgan Kaufmann Publishers, pp. 1019–1022.
- [75] XIA, G., SUN, H., NIU, X., ZHANG, G., AND FENG, L. Keyframe Extraction for Human Motion Capture Data Based on Joint Kernel Sparse Representation. *IEEE Transactions on Industrial Electronics* PP, 99 (Sept. 2016), 1589–1599.

- [76] XIAO, J., ZHUANG, Y., YANG, T., AND WU, F. An Efficient Keyframe Extraction from Motion Capture Data. In *Proceedings of the 24th International Conference on Advances in Computer Graphics* (2006), Springer-Verlag, pp. 494–501.
- [77] YAMANE, K., ARIKI, Y., AND HODGINS, J. Animating Non-humanoid Characters with Human Motion Data. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2010), Eurographics Association, pp. 169–178.
- [78] YASUDA, H., KAIHARA, R., SAITO, S., AND NAKAJIMA, M. Motion belts: Visualization of human motion data on a timeline. vol. E91.D, pp. 1159–1167.
- [79] ZHANG, Q., XUE, X., ZHOU, D., AND WEI, X. Motion Key-frames Extraction Based on Amplitude of Distance Characteristic Curve. *International Journal of Computational Intelligence Systems* 7, 3 (2014), 506–514.
- [80] ZHANG, Q., YU, S.-P., ZHOU, D.-S., AND WEI, X.-P. An Efficient Method of Key-frame Extraction Based on a Cluster Algorithm. *Journal of Human Kinetics* 39, 1 (2013), 5–14.
- [81] ZHANG, Q., ZHANG, S., AND ZHOU, D. Keyframe Extraction from Human Motion Capture Data Based on a Multiple Population Genetic Algorithm. *Symmetry* 6, 4 (2014), 926.
- [82] ZHANG, Y., AND CAO, J. 3D Human Motion Key-Frames Extraction Based on Asynchronous Learning Factor PSO. In *Fifth International Conference on Instrumentation and Measurement, Computer, Communication and Control* (2015), pp. 1617–1620.