Motion Composition for Character Animation

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Anna Dagna Majkowska

2007
The dissertation of Anna Dagna Majkowska is approved.

Michael G. Dyer

Gerassimos Orkoulas

Demetri Terzopoulos

Petros Faloutsos, Committee Chair

University of California, Los Angeles

2007
For Andrew
# Table of Contents

1 Introduction .......................................................... 1

2 Related Work ......................................................... 10
   2.1 Work Related to Dynamic Response for Motion Capture Animation 12
   2.2 Work Related to Motion Composition for Acrobatics .............. 13
   2.3 Work Related to Automatic Splicing for Hand and Body Animations 15

3 Overview of Motion Capture Technology .......................... 17

4 Dynamic Response for Motion Capture Animation .............. 21
   4.1 Motion Selection .................................................. 25
   4.2 Transition Synthesis ............................................. 27
   4.3 System Implementation ......................................... 29
      4.3.1 Dynamic Simulation ....................................... 29
      4.3.2 Motion Capture Database ................................. 30
   4.4 Results .......................................................... 31
   4.5 Discussion ..................................................... 32

5 Motion Composition for Acrobatics ............................... 38
   5.1 Introduction ..................................................... 38
5.2 Overview ................................................. 41

5.3 Pre-processing Stage ................................. 43
  5.3.1 Ballistic Phase Detection ......................... 43
  5.3.2 Estimating Mass Distribution ..................... 44
  5.3.3 Momentum Computation .......................... 48

5.4 Run-Time Jump Generation Stage .................. 48
  5.4.1 Editing Operations and Their Impact on Momentum ... 50
  5.4.2 Search and Sequencing ........................... 51
  5.4.3 Retiming .......................................... 56
  5.4.4 Repositioning ..................................... 58

5.5 User Modifications .................................... 59
  5.5.1 Changes in Direction of L ....................... 60
  5.5.2 Changes in Magnitude of L ....................... 63
  5.5.3 Final Retiming and Repositioning ................. 63

5.6 Results ................................................. 64

5.7 Experimental Evaluation ............................. 69

5.8 Discussion ............................................. 71

6 Automatic Splicing for Hand and Body Animations....... 74

  6.1 Matching Algorithm ................................. 77
  6.2 Distance Metrics for Phase Alignment ............... 80
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Motion Composition Techniques Diagram</td>
<td>9</td>
</tr>
<tr>
<td>3.1</td>
<td>Dancer in a Motion Capture Suit</td>
<td>20</td>
</tr>
<tr>
<td>4.1</td>
<td>Dynamic Response for Motion Capture Animation: Example Output from Our System</td>
<td>22</td>
</tr>
<tr>
<td>4.2</td>
<td>Dynamic Response for Motion Capture Animation Diagram</td>
<td>23</td>
</tr>
<tr>
<td>4.3</td>
<td>Dynamic Response for Motion Capture Animation: Range of Examples Deriving from a Single Pair of Motion Clips</td>
<td>33</td>
</tr>
<tr>
<td>4.4</td>
<td>Dynamic Response for Motion Capture Animation: Three Sample Animations</td>
<td>36</td>
</tr>
<tr>
<td>4.5</td>
<td>Dynamic Response for Motion Capture Animation: More Animation Examples</td>
<td>37</td>
</tr>
<tr>
<td>5.1</td>
<td>Motion Composition for Acrobatics Diagram</td>
<td>39</td>
</tr>
<tr>
<td>5.2</td>
<td>Motion Composition for Acrobatics: Schematic Diagram of Our Method’s Pre-processing and Run-Time Jump Generation Stages</td>
<td>43</td>
</tr>
<tr>
<td>5.3</td>
<td>Motion Composition for Acrobatics: Outline of the Run-Time Jump Generation Stage of Our Algorithm</td>
<td>49</td>
</tr>
</tbody>
</table>
5.4 Motion Composition for Acrobatics: Differences in the Angular Momentum Direction Between an Original Motion and a Motion Modified by an Animator ......................................................... 62
5.5 Motion Composition for Acrobatics: Errors in the Angular Momentum Magnitude in a Motion Modified by an Animator Before and After Retiming ..................................................... 64
5.6 Motion Composition for Acrobatics: Double Somersaults Generated with Our Technique ................................................................. 66
5.7 Motion Composition for Acrobatics: Various Twist Jumps Generated with Our Method ................................................................. 67
5.8 Motion Composition for Acrobatics: Joining Together an Aerial Cartwheel Motion and a Forward Flip ..................................................... 68
5.9 Motion Composition for Acrobatics: The Results of a User’s Editing Operations that Do Not Change the Direction of the Angular Momentum Vector ......................................................... 68
5.10 Motion Composition for Acrobatics: The Results of a User’s Editing Operations that Change Both the Direction and the Magnitude of the Angular Momentum Vector ......................................................... 73
6.1 Automatic Splicing for Hand and Body Animations: Mudras in Indian Dancing ................................................................. 75
6.2 Automatic Splicing for Hand and Body Animation Diagram ................................................................. 76
6.3 Automatic Splicing for Hand and Body Animations: Dynamic Time Warping for Two Motions ........................................ 81

6.4 Automatic Splicing for Hand and Body Animations: Our Distance Function Produces Better Results Than a Function Using the Values of the Derivatives ............................................. 81

6.5 Automatic Splicing for Hand and Body Animations: Matching Frames Within Phases .............................................. 86

6.6 Automatic Splicing for Hand and Body Animations: Computing $D_2$ ................................................................. 87

6.7 Automatic Splicing for Hand and Body Animations: Example of Directive Gestures ..................................................... 87

6.8 Automatic Splicing for Hand and Body Animations: Example of a Charade ............................................................... 89

6.9 Automatic Splicing for Hand and Body Animations: Counting Example Shows Good Synchronization of Hand and Body Movement .............................................. 90

6.10 Automatic Splicing for Hand and Body Animations: Indian Dance: Side by Side Comparison ........................................... 91

D.1 Automatic Splicing for Hand and Body Animations: Camera Setup for Hand Capture ...................................................... 103

D.2 Automatic Splicing for Hand and Body Animations: Hand Markers ................................................................. 104

D.3 Automatic Splicing for Hand and Body Animations: Full Body Capture ................................................................. 105
D.4 Automatic Splicing for Hand and Body Animations: Hand capture. 106
LIST OF TABLES

5.1 Motion Composition for Acrobatics: Comparison of Take-off Poses for a Single and Double Backflip ........................................ 57

5.2 Motion Composition for Acrobatics: Results of Our Perceptual Study .......................................................... 71

6.1 Automatic Splicing for Hand and Body Animations: Types of Motion Phases and Their Acceleration and Velocity Profiles ........ 83

6.2 Automatic Splicing for Hand and Body Animations: Parameters of Motion Sequences ............................................. 88

6.3 Automatic Splicing for Hand and Body Animations: Running Time of Our Algorithm ............................................. 89

A.1 Dynamic Response for Motion Capture Animation: Weights and Masses Used in Our Implementation ................................ 97

C.1 Dynamic Response for Motion Capture Animation: Weights and Masses Used in Our Implementation ................................ 102
ACKNOWLEDGMENTS

There are many people who helped me during my years as a grad student. My chief debt is to my advisor Petros Faloutsos who helped me tremendously in my research and made this work possible. He also gave me a lot of great advice, some of which I stubbornly refused to take. I also owe a great deal to Victor Zordan, my advisor at UC Riverside who introduced me to the fascinating area of computer graphics and animation. I would also like to thank my labmates from the Magix Lab: Sung Hee Lee, Brian Allen, Shawn Singh, Gabriele Nataneli, Derek Chu, and Ari Shapiro for sharing their knowledge with me and providing superb company, especially during the sleepless nights before submission deadlines. Bill Chu from the Riverside Graphics Lab also put in some long hours during our work on the dynamic response project and was a great conversation partner.

I learned a lot from the professors at UCLA and UCR. I would like to thank them all, especially Eamon Keogh and Richard Korf who gave me many useful ideas for my research. Members of my doctoral committee: Michael Dyer, Gerasimos Orkoulas and Demetri Terzopoulos helped me to improve this dissertation with their comments and fresh perspective on my work.

I would also like to thank my friend Yulia Zabiyaka for making my years at UCLA much more exciting and Verra Morgan, the graduate student advisor, for helping me to stay out of trouble with the Graduate Division and otherwise. Finally, I couldn’t thank enough my fiancé Andrew Sorensen, who kept me going each time I wanted to give up.
Vita

1978 Born, Warsaw, Poland.

2000 B.S., Computer Science
Warsaw University
Warsaw, Poland

2000–2003 Java Developer
Scott Tiger SA
Warsaw, Poland

2002 M.S., Computer Science
Warsaw University
Warsaw, Poland

2003–2005 Teaching Assistant
Department of Computer Science
University of California, Riverside

2004 Research Intern
Lawrence Livermore National Laboratory
Livermore, California

2006 Teaching Assistant
Department of Computer Science
University of California, Los Angeles
2006 Engineering Intern
Google Inc.
Mountain View, California

PUBLICATIONS


Abstract of the Dissertation

Motion Composition for Character Animation

by

Anna Dagna Majkowska

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2007

Professor Petros Faloutsos, Chair

Motion composition is a motion synthesis approach which allows for the creation of new animations by joining together existing motion fragments. Because it preserves the naturalness and the richness of detail of the original motion capture animations, motion composition is becoming increasingly popular with the growing availability of motion capture data.

In this dissertation, we present three novel motion synthesis techniques based on different composition strategies. First, we propose a method for incorporating unexpected impacts into motion-capture driven animations. This technique automatically generates a reaction to an impact by using physics-based simulation, and then switches to the most appropriate response clip selected from a motion capture library. By using motion capture jointly with dynamic simulation we obtain motions that are both physically valid and human-like.

The second technique presented generates complex acrobatic motions that adhere to the laws of physics. Multi-flip jumps are created from single flip an-
imations by rotating and composing together fragments of input motion while maintaining the property of momentum conservation. Our approach also allows an animator to interact with the system by introducing modifications to a ballistic phase of a motion. Our algorithm then automatically adjusts motion rotation, timing and trajectory, to assure physical validity of the motion after the modifications. In our approach, masses and inertia moments of the character’s body parts, necessary for momentum computations, are computed directly from the data using a novel estimation method based on the conservation of linear momentum. To validate our approach we present the results of a study of user sensitivity to errors in angular momentum and take-off angle. The study shows that small changes of these parameters introduced by our method are not perceptible to a viewer.

The final proposed method adopts a different approach - layering motions in time to obtain a multi-resolution effect. By synchronizing hand and full-body motions recorded in separate sessions, our technique creates character animations with detailed hand movement. In addition, we provide a method for supplying user input, which provides an animator with variable degree of control over the synchronization process.
CHAPTER 1

Introduction

Human motion is remarkably complex; 206 bones and more than 600 muscles produce a seemingly infinite number of possible movements and body configurations. However, if one was to attempt the Herculean task of simulating the exquisite machinery of the human body for the purpose of generating human motion, the hardest organ to simulate would not be bones or muscles but the human brain. It is the brain that allows humans to create movements so complex, so rich in minute detail that sometimes they require years or decades to master.

In the face of such a tremendous intricacy of human motion, one can hardly hope to construct an universal algorithm that would generate all possible varieties of human movement. Instead, three different motion synthesis approaches have been devised for use in computer animation: handcrafting, physics-based simulation, and motion capture adaptation. Handcrafting, the most traditional approach, requires an animator to generate movements manually from observation and their knowledge of human motion. Like any other form of art, the results can be poor or outstanding depending on the skills of the creator, but the process is always time-consuming.
Another solution is simulation that, out of necessity, ignores all the troublesome complexities of human body and brain, and settles for decidedly simpler models as an approximation. Control and optimization methods fall into that category. The control technique attempts to model the motor-control function of the brain as an algorithm. This in essence amounts to creating a sophisticated virtual robot. Predictably, results are physically valid but robotic-looking. This is due to the fact that there are many ways to transition from one body configuration to another and only a few of these transitions look human-like. Optimization methods attempt to solve this problem by adding an objective function which measures the naturalness of all possible transitions in order to find an optimal one. The objective function typically includes many components such as energy expenditure or joint preferences. The optimization approach often produces good results. However, unsurprisingly, there is no single objective function that would work well with all types of motion.

The final approach - motion capture adaptation, became possible thanks to an increased availability of motion capture technology, which allows accurate recording of human movements and their transfer to animated characters (see Chapter 3). However, with the infinite variety of possible human motions it is not possible to record them all. Hence the need for motion adaptation methods which allow for modifying existing motions to match an animator’s demands. Amongst them are interpolation between motions, style transfer, techniques for retargeting existing motions to new characters and motion composition - an approach for obtaining new animations by joining together multiple motion segments. Our
work falls into the last category.

Just like the other approaches, motion composition does not offer a one-size-fits-all solution and different composition methods have been developed for different motion types and animation tasks. To date, the main focus of research in the area of motion composition is the development of new methods for creating motions that meet space and time constraints. Composition algorithms typically search the motion database to find appropriate motion segments according to an animator’s specifications and join them together. Increasing the search efficiency and assessing the quality of transitions between motion segments are other areas of focus (see Chapter 2).

In contrast, our work concentrates on exploring new tactics for joining motions which allow us to achieve novel effects and obtain motions that cannot be generated by simply “gluing” motions together (Figure 1.1 a). These tactics are: interjecting physical simulation between motion segments (Figure 1.1 b), rotating, repositioning and retiming motion segments to conserve momentum (Figure 1.1 c), and layering motions (Figure 1.1 d). During the composition process special care is taken to maintain the physical validity of resulting motions. Unlike most methods based on gluing motions, our techniques produce physically valid and natural-looking animations even when the composed motions are significantly different from each other.

First, we introduce a technique for incorporating unexpected impacts into a motion capture-driven animation. This method automatically generates a reaction to an impact by using physics-based simulation, and then switches to the
most appropriate response clip selected from a motion capture library. (Figure 1.1 b). Creating a believable response following an impact is challenging for an animator, as it entails changes to a character’s bodies velocities and accelerations. Further, the handcrafting of motion is often at odds with the goal of maintaining interactivity. With our method an animator can automatically create physically valid and natural-looking impact responses.

Next, we propose a technique for creating complex multi-flip acrobatic motions from simple, single-flip animations. To obtain this effect our algorithm rotates and composes fragments of motion while maintaining physically valid momentum profiles (Figure 1.1 c). However, simply rotating and joining motion fragments is often not sufficient, as multi-flip jumps vary significantly from their single-flip counterparts. In the multi-flip motions the rotation is faster and the flight phase longer. The challenge here lies in modifying the constructed motion to assure proper momentum build-up and release. Our technique adjusts motion timing and modifies flight trajectories to guarantee physically valid momentum profiles. Additionally, we present a method for adapting ballistic motions after user modifications to assure their physical validity. An animator can change a character’s joint angles and body parts’ positions during the flight. Our system then automatically adapts the modified motion by rotation and retiming to ensure conservation of momentum. To validate our approach we present the results of a study of user sensitivity to errors in angular momentum and take-off angle. The study shows that small changes of these parameters introduced by our method are not perceptible to a viewer.
While the previous two techniques combine motion fragments along the time axis, the last proposed method of motion composition layers motions that occur simultaneously (Figure 1.1 d). This technique takes as an input motion capture data of hand and full-body motions recorded in separate sessions and splices them seamlessly together to create character motion with detailed hand gesticulation. Although it is possible to record hand movement together with full-body motion in a simultaneous capture, there are compelling reasons for recording hand motion separately. These include greater flexibility in re-use of hand motions across animated characters and scenes and increased accuracy and control over the hand capture. Obtaining correct synchronization between the two motion capture sources can be a challenging problem due to differences in motion execution. Our matching algorithm eliminates these differences by applying dynamic time warping in combination with a novel distance measure, based on research on human gestures. Additionally we provide a mechanism which allows animators to guide the synchronization process.

The three proposed methods target different types of motion. This is due to the fact that each of the methods utilizes unique properties of a particular motion type to simplify the synthesis process and make the generation of novel motions possible. In case of the dynamic response technique it is the concept of burst - a short time period before a human reaction takes place, that guides our simulation of the impact response. In motion composition for acrobatics, the simple form of momentum conservation laws during the ballistic phase allows us to create and modify acrobatic motions in a physically-valid manner. Finally, during the
splicing process for hand and full-body animations we use our knowledge of human
perception of gestures to correctly align hand and body motions.

While the proposed techniques apply to different kinds of motions, they share
the same goal: to use existing animations for creating motions that would be
either impossible or dangerous to record with motion capture technology. Our
dynamic response method (Chapter 4) creates reactions to strong impacts that
would be unsafe and painful for an actor to perform. Motion composition for
acrobatics (Chapter 5) allows for efficient generation of complex acrobatic jumps
that lie near or outside the limits of human athletic abilities. Automatic
splicing for hand and body animations method (Chapter 6) produces full-body
motion with detailed hand gesticulation that would be difficult to record with
motion capture technology due to technical limitations.

The main contributions presented in this dissertation are:

- **A novel technique for incorporating unexpected impacts into a
  motion capture-driven simulation.** Our system combines physical sim-
  ulation, which responds to contact forces, and a specialized search routine
  which determines the best re-entry point to create animated characters
  which can respond to unexpected impacts.

- **Efficient methods for creating and editing complex, physically
  valid acrobatic motions.** Our algorithm takes as an input a simple
  ballistic motion, such as a single somersault, and creates a more complex
  motion, such as a multiple somersault, by rotating and repeating fragments
of motion, while maintaining physically valid momentum profiles. The proposed approach also allows for user motion modifications during a ballistic of motion. Our algorithm adapts the modified motion to assure its physical validity.

- **A method for estimating mass distribution from motion capture data.** Taking advantage of the momentum conservation laws, our technique applies linear least squares optimization to the motion data during the flight segment to estimate a character’s mass distribution.

- **A simple and effective technique for hand and body motion alignment.** Our method uses dynamic time warping together with a novel distance metric to align motions with significant amplitude differences. Key features of human gesticulation, supported by research on gestures, create the foundation for our distance metric.

- **A method for incorporating user-specified constraints directly into the dynamic time warping algorithm.** By modifying the shape of the dynamic time warping band our method allows a user to easily choose from a continuous spectrum of control options: from fully automatic matching with no user input, to partial control, where the user suggests matching regions, to complete control, with the user listing specific pairs of frames to be matched.

The remainder of this dissertation is organized as follows: Chapter 2 reviews relevant research in motion capture editing. Chapter 3 presents an overview of
motion capture technology. Chapter 4 describes a technique for incorporating unexpected impacts into a motion capture-driven animation. Chapter 5 details a method of creating complex, multi-flip acrobatic motions from simple, single-flip animations and an algorithm for adapting motions after user modifications. Chapter 6 presents a technique of creating motion with detailed hand gesticulation by layering hand and full-body motions. Chapter 7 concludes the dissertation.
Figure 1.1: Motion composition techniques. a) Simple motion "gluing" with smoothing filter (SM), prevalent in most composition methods. b) Interjection of physical simulation into composed motions allows incorporation of impacts into motion capture-driven animation. c) Rotating and repeating of motion fragment B creates complex, multi-flip acrobatic motion. Repositioning and retiming assure correct momentum profiles. d) Layering of full-body and hand motions produces character animation with detailed hand movement. Dynamic Time Warping (DTW) assures correct synchronization of the two motions.
CHAPTER 2

Related Work

Researchers have introduced numerous techniques for synthesis of new motions by editing and combining existing motion data. The most basic form of motion composition involves "gluing" together multiple motion segments along a time axis to obtain new, more complex movements that meet an animator’s constraints [33, 67]. To remove motion discontinuities, transitions between segments are created by blending motion frames. Generation of the transitions and assessment of their quality is an active area of research. Wang and Bodenheimer propose a method for evaluating an optimal length of the transition [71] and for computing a set of optimal weights for the cost function which identifies good transition points [70]. Matsunaga and Zordan present an alternative algorithm for computing weights based on character’s physical properties [42]. Kovar and Gleicher propose a blending algorithm which automatically determines relationships between input motions to increase the range of motions that can be successfully joined together [30]. Ikemoto et al. offer a different algorithm which enforces foot constraints to avoid sliding [22]. Safanova and Hodgins show that simple linear interpolation can be used in many cases to produce natural-looking transitions [56].
In large motion databases, testing all possible motion composition variations would be prohibitively expensive. A structure known as a motion graph organizes multiple motions into a directed graph form in order to make the motion synthesis process more efficient. Typically, a node in the motion graph contains a single motion frame and a directed edge denotes a possible transition between two frames, as proposed in [3, 32, 36]. Alternatively, nodes contain fragments of motion [37, 55] or even clusters of motions representing similar tasks [19]. In order to assess the quality of a generated graph Reitsma and Pollard propose an algorithm for determining how many goals in a given environment are reachable [53]. A number of techniques address a problem of efficient search in motion graphs: [3, 32, 35, 36, 64, 66]. Other works integrate motion interpolation into the graph design or search process to create novel motions [19, 57].

Motion composition often involves synthesis of motions that match a user’s specifications detailing the type of movement to be used (such as running or jumping). Therefore it is often necessary to efficiently retrieve a given motion type from a large motion collection. Kovar and Gleicher present a method for an efficient retrieval of motions similar to a given motion sample [31]. Their algorithm aligns motions in time to eliminate timing differences and utilizes a root-invariant similarity measure for motion comparison. Forbes and Fiume use weighted PCA to reduce motion dimensionality and improve search efficiency [17]. Müller et al. use logical predicates that describe the motion instead of motion samples which can sometimes be difficult to obtain [46].
2.1 Work Related to Dynamic Response for Motion Capture Animation

While many works concentrate on efficient search and creating natural-looking transitions, fewer researchers present composition approaches which combine motion capture with other motion generation methods, such as physical simulation. Our dynamic response method falls into that category. We use motion capture jointly with physical simulation with the goal of creating a responsive character. Most of the other works in this area focus on balanced motions and propose methods suited for weak impacts [4, 29, 48, 74]. More closely related work, presented by Shapiro et al. [62] introduces hybrid methods for moving between motion capture and dynamic simulation. In contrast, we propose a specialized method for impact response including a customized search method (Chapter 4). In addition, in our method we move from and to motion capture with a minimal, focused simulation phase. Mandel [41] suggests specialized controllers for falling and recovery which take over once impact is detected. In our framework, we give preference to the motion capture animation and use the simulation only as a vehicle for creating an immediate response. We propose a shift away from building specialized controllers and instead treat the controlled simulation as an interim system, with emphasis placed on motion capture examples. In a subsequent work, Zordan et al. introduce a few improvements to our technique that allow it to run in real-time [75].
2.2 Work Related to Motion Composition for Acrobatics

Our next proposed composition method creates more complex motions, such as acrobatic flips and stunts (Chapter 5). These have been typically generated either with physical simulation or optimization techniques rather than with motion capture editing methods. Simulation with control mechanisms has been successfully applied to create a wide range of motions, including acrobatics. Hodgins et al. [20] designed a set of controllers with a state machine, which allowed a physically simulated character to perform a somersault over a vault. Wooten et al. [73] created controllers for high diving and leaping and Faloutsos et al. [14] constructed a kip stunt controller. Laszlo et al. [34] proposed an algorithm for interactive creation of various flips, kips and somersaults. While control algorithms create physically valid (though sometimes robotic-looking) motions, the design of such controllers is still a difficult and time-consuming process.

Optimization techniques with physics constraints have become a prominent method for generating realistic highly dynamic motions [15,39,40,50,58,72]. The efficiency of such methods have been significantly improved in recent works by using a simple set of dynamic constraints [39], or using constraint functions which derivatives can be computed in linear time [15], or by reducing motion dimensionality [58]. However, even with these improvements, generating motions with optimization techniques typically takes a few minutes, which limits their applicability in real-time and interactive applications. Many works apply spacetime optimization to the problem of editing ballistic motions [2,43,55,58,65,72]. These
Techniques allow animators to synthesize new, physically valid motions while preserving realism and style of movement. While many of these techniques could be applied to the problem of creating complex acrobatic stunts, this work shows that for such motions time-consuming optimization techniques are not necessary. High-effort ballistic motions can be synthesized and modified extremely efficiently using only basic editing operations.

The search algorithm used in our technique compares frames of motion consisting of the character’s joint angles and the joints’ angular velocities. Similar comparison methods have been employed in many motion capture editing works such as [3,32,36,76] and others. Similarly to [76] we assign higher weights to joints close to the body root, as differences in limb positions can be easily smoothed by blending. However, in contrast to the above methods, we rotate the skeleton around the momentum vector before the comparison to minimize differences in body orientation between compared frames.

The results of our perceptual study contribute to the area of human perception of motion. Reitsma and Pollard [52] studied human sensitivity to changes in vertical and horizontal acceleration and modifications of gravity. Safanova and Hodgins [56] observed that motion interpolation can cause large fluctuations in angular momentum which often go unnoticed if the resulting changes in angular velocities are small. O’Sullivan et al. [49] studied perception of angular distortions and velocity changes in post-collision trajectories for simple objects such as spheres, blocks and T-shapes. We contribute to this body of knowledge by measuring human sensitivity to distortions in angular momentum and take-off
angle.

Our method for estimating mass distribution is similar to the approach proposed in [6]. In their work, Bhat and colleagues apply optimization to estimate motion and physical parameters of a rigid body in free flight from video, given the body’s mass distribution as an input. In contrast, our technique uses 3D motion data to estimate not only motion parameters, but mass distribution as well.

2.3 Work Related to Automatic Splicing for Hand and Body Animations

While the previously described approaches combine cuts made along the time axis, fewer examples in the literature describe methods for layering motions that occur simultaneously, which is the case in our technique for creating animated characters with detailed hand motion (Chapter 6). Most of work in this area has focused on extracting style elements from one motion and layering them over another [7, 21, 51, 54, 61, 68]. In our method we aim our efforts at automatically combining different motion clips onto a single skeleton by allowing each clip to control its own portion of the skeletal hierarchy. The work of Ikemoto et al.[23] shares the characteristic of integrating two motions, which control different parts of the character skeleton, into a single hierarchy. Our technique is unique in that we are interested in combining recordings from different sources by finding correlations within the motion clips and synchronizing the timing of the clips to
align them in a coherent manner. In their work, Ikemoto et al. transplant limbs from one clip to another by selecting random pairs from a library and forming an arbitrary transplantation and then accessing the quality based on a set of rules. In contrast, we focus on combining known, user-specified clips, and place our efforts in the careful alignment over time of the motions contained in the given sources. Our technique is also similar to that of Dontcheva et al. [12] in our effort to align motion sequences recorded in separate motion-capture takes and to apply them to a single skeleton. However, our approach, driven by the specific problem domain, accounts for amplitude differences that change over time in a non-uniform and non-discrete manner. This is crucial in the area of hand gesticulation, where different recordings of the same gesture sequence tend to have large, varying amplitude differences.

Although other researchers have used time warping for motion alignment, for example [17, 30, 31], we applied it to a novel domain of human gesticulation. This posed new challenges to our system. While existing work concentrated on more dynamic motions, such as walking or jogging, human gesticulation tends to have more kinematic qualities with significantly more variation than motions controlled to a large extent by physical constraints. Because of these differences, previously proposed comparison metrics do not work well in the domain of gesticulation. Additionally, although the idea of a user controlling the alignment by specifying pairs of matching frames has been used before (for example by Rose et al. [55]), we are the first ones to incorporate the constraints directly into the DTW band in order to allow for a varying degree of user control.
CHAPTER 3

Overview of Motion Capture Technology

The development of new methods for motion synthesis, such as motion composition, has been spurred on by an increased availability and variety of motions recorded with motion capture technology. Motion capture, a technique for digitally recording three-dimensional motions, is becoming increasingly popular in a wide range of applications. Aside from use in computer animation, motion capture has been employed in sports training, biomechanics research and medicine. For example, spinal capture was performed on astronauts in a space station to observe the effects of zero gravity on their spines [45]. The earliest applications of motion capture were in the military technology, where it was used to track targets and troop movements. Motion capture can be used on a wide variety of subjects. Apart from humans, it has been used to record movements of horses, kangaroos, iguanas and even ants [45]. Recording environments also vary: motion capture has been used in space, under water, on climbing walls, even in New York City.

During motion capture recording, a performer typically wears a leotard with multiple markers, which are positioned near joints on all body parts. Specialized, high speed cameras track the markers’ positions throughout the performer’s motion. In the post-processing step positions of body parts and joint angles are
calculated from the recorded marker data. While most motion capture technologies employ markers or sensors, there are also other solutions. For example, mechanical systems employ skeleton-like structure, called exoskeleton, which is attached to a performer’s limbs.

The two most commonly used motion capture technologies are optical and magnetic systems. In our work we utilized motions recorded with passive optical systems, in which markers covered with retroreflective material reflect light that is emitted by specialized cameras. In contrast, active optical systems markers contain small LEDs which emit light. Magnetic systems employ sensors which measure the magnetic field created by a source to obtain body parts’ positions and joints angles. Typically, the recording range in magnetic systems is significantly smaller than in optical ones.

The main reasons for motion capture popularity in computer animation are the rapid (sometimes even real-time) generation of results and the high quality of obtained motions. Employing motion capture technology can reduce both cost and time of producing an animation compared to traditional methods, where motions are handcrafted by an animator. The resulting motions are typically more detailed and natural-looking than the ones created with traditional methods or physical simulation.

The disadvantages of motion capture include high cost and specific recording space and equipment requirements. The recording space needs to be large enough to make it possible to record a wide variety of motions, such as running, jumping and martial arts moves. Typically 120Hz cameras are used to adequately capture
fast (high frequency) motions. Additionally, cameras’ resolution must be high enough to accurately record secondary motion details. The two most common artifacts that make motion capture process challenging are occlusion and marker swapping. Occlusion occurs when a marker is obstructed from a view of a camera. In order to compute a 3D position of a marker in space, the marker has to be visible to least two cameras. When this is not the case (for example when a marker is obstructed from view by a body part), it leads to a loss of data. Another artifact - marker swapping can occur when two (or more) markers move so close to each other that the tracking system of the cameras confuses one marker for another.

However, perhaps the most significant limitation of motion capture systems lies in the fact that they produce simply a recording of motions. Unlike in the case of physical simulation techniques, generated motions cannot adapt to new environments or respond to new interactions. The infinite variety of human motions makes it impossible to record and store them all. This is where motion adaptation techniques such as motion composition step in. In this work we present a number of techniques that address some of motion capture’s limitations.
Figure 3.1: Dancer in a motion capture suit with passive optical markers.
CHAPTER 4

Dynamic Response for Motion Capture Animation

With the increasing popularity of motion capture animation in game and movie industries, techniques for data synthesis and adaptation have become increasingly important. In particular, methods that allow animated characters to interact with each other and the environment, and respond to unexpected impacts are especially useful in gaming applications. Impact reactions are difficult to generate by manual motion editing as they require making changes to the character’s body velocities and accelerations. Changes to motion trajectories that violate the laws of physics can often cause the motion to appear unnatural and physically flawed [52, 71]. Additionally in gaming applications the hand crafting of motion trajectories is often at odds with the goal of maintaining interactivity. Automatic motion synthesis methods that generate physically correct responses are especially valuable because they allow characters to react and interact with their environment while maintaining consistency and richness of detail in generated motions.

Physical simulation is commonly used to create responsive characters by gen-
Figure 4.1: Example output from our system. The generated motion (red) fills in the gap between the blue and green motion capture segments. The synthesis and motion selection (of the green clip) are computed automatically by our system given the interaction which is about to take place in the left panel above. The plot shows the Y-value of the waist joint before, during, and after the interaction, and two intermediate trajectories of simulated motion used by our system.

erating a dynamic response to collision forces following the impact. In many applications, when impacts are detected, motion-capture driven characters are turned over to a dynamics engine to generate a response. However, the reaction of the character is most often passive and dissipates over the time period just following the impact. The character either resumes following the original motion sequence [48, 74] or goes limp and falls to the ground like a "rag-doll", as
Figure 4.2: Our method creates a dynamic response to an impact by interjecting physical simulation into a motion capture-driven animation.

if rendered unconscious after the impact. Such an approach not only generates unrealistic reactions but also offers no procedure for returning the simulation to motion capture control once the effects of an impact are over. In this chapter, we present a technique which automatically computes a dynamic reaction to an unanticipated impact and returns the character to an available clip from a motion capture repository. In other words our method composes two motion capture segments with a brief period of simulation in-between to create a responsive animated character (see Figure 4.2).

A key observation that makes our approach practical is that dynamic simulation is often only needed for a short time period (a burst) during which the effects of a collision change the velocities and accelerations of a character’s body parts. After that, the utility of the dynamics decreases and without a good fall controller the resulting motion of the simulation becomes less reliable than a motion capture sequence. We take advantage of this observation and propose an automated system which selects a best suited clip from a motion capture repository for soon after the interaction (the green clip in Figure 4.1) and computes a new motion (shown red in Figure 4.1) that allows the character to respond to the impact and prepare for its newly determined, upcoming animation. Utiliz-
ing a motion library with a small set of real human reactions (such as falls and
dodges), we find good matches given a preliminary, "naive" response trajectory
computed using forward simulation (labeled as "Simulation 1" in the Figure 4.1).
The second response (labeled "Simulation 2") is generated using the knowledge
of the selected reaction and acts in anticipation of the found data. In the last
step, the system generates the final animation (labeled "Final Blend") by inter-
polating the dynamic motion so that it matches the selected response at the end
of the generated segment.

Our system consists of two main components: a search engine that compares
the initial simulated response with reaction segments from a motion library and a
joint-torque controller that generates the "naive" and active simulations. In our
implementation on these components, we draw from and contribute to two inde-
pendent areas of previous work, building on reported techniques while combining
them to meet our specific needs. Our search engine is similar to those suggested
previously for composing motion capture data but is crafted to give importance
to the main bulk of the mass (i.e. the trunk) as well as the ground-support body,
targeting relevant dynamic effects and contact constraints in the motion compar-
ison. Although our control approach is not novel, our application of simulation to
generate an unexpected forceful interaction in the motion immediately following
is different than solutions presented in previous work.

The main contributions of our system are:

• a simple algorithm for creating a physically-valid and natural-looking re-
  sponse to an impact,
• a search engine which employs optimizations that reduce the search time without sacrificing the search quality by utilizing the concept of unique frames,

• an active simulation, which creates an illusion of a character consciously reacting in preparation of the upcoming fall.

### 4.1 Motion Selection

As the first step of our algorithm, we find a response motion from our repository to be used both for generating the active simulation and for the animation following the interaction. To do this, we first use forward simulation to synthesize the motion that acts in response to forces present during the interaction. Next, we compare the simulated data with sequences in our motion library to find the best sequence to transition to as well as the precise time and root transformation that aligns the found sequence with the simulated trajectory.

In our search, each frame is defined as a vector \( f = (p_1, \theta_1, \ldots, p_n, \theta_n)^T \) where \( p_i \) are the positions and \( \theta_i \) are the orientations of each of the \( n \) body parts. We compare frame windows, similarly to Kovar et al. [32], between the simulation segment and candidate motion capture sequences. Within windows, the distance function for pairs of frames \( f_{1i} \) and \( f_{2i} \) is computed as the weighted sum of distances between the positions and orientations of matching body parts. The distance between two windows is found as the sum of distances between the corresponding frame pairs. To remove coordinate-frame differences, we normalize...
by aligning the roots in the start frames of each window. The distance $D$ between windows $W_1 = \{f_{1i}\}_{i=s}^e$ and $W_2 = \{f_{2i}\}_{i=s}^e$ is then defined as:

$$D(W_1, W_2) = \sum_{i=s}^e d(f_{1i}, f_{2i}),$$

where (4.1)

$$d(f_{1i}, f_{2i}) = w_i \left( \sum_{b=1}^n w_{pb} \|p_b(f_{1i}) - p_b(f_{2i})\| + w_{\theta b} \|\theta_b(f_{1i}) - \theta_b(f_{2i})\| \right),$$

and $w_i$ is the window weight, a quadratic function which returns the highest value for the start frame and decreases for subsequent frames in the window. The weights $w_{pb}$ and $w_{\theta b}$ scale the linear and angular distances for each body $b$ (see Appendix A).

To capture the dynamic properties existing in the simulated motion, we assign high weights to the trunks. We use lower weights for the limbs, as we found that differences in the positions of the limbs can be more easily modified during the active simulation. Additionally, to reduce the problem of sliding ground contact, we compute the center of mass at the first frame of each window and assign a high weight to the closest ground-support body for the duration of the window.

To increase search efficiency we pre-process our motion database and find unique frames. This approach takes advantage of the fact that neighboring frames tend to be very similar and therefore we can reduce the number of frame comparisons without sacrificing the quality of the search. We define unique frames recursively. The first frame of a motion is always defined as unique. For the subsequent frames, a frame is unique only if it is significantly different from the last unique frame found. In other words for a frame sequence $\{f_i\}_{i=0}^m$, we define
unique frames \( \{F_k\} \) as:

\[
F_0 = f_0,
\]

for \( i > 0 \):

\[
F_k = f_i \iff F_{k-1} = f_j \land i > j \land d(F_{k-1}, f_i) > T \land \forall j < l < i d(F_{k-1}, f_l) \leq T,
\]

where \( T \) is a tolerance constant found experimentally (see Appendix A) and the distance function \( d() \) is defined in the Equation 4.1. In our search we first compare windows starting at unique frames. Once we find a best matching pair, we test all possible windows in the surrounding intervals between the neighboring unique frames to find the closest match.

### 4.2 Transition Synthesis

Given the motion capture response we found in the previous step, we must generate a reaction to fill in the burst between the beginning of the interaction and the time the found motion is played. We synthesize this transition with two goals in mind: to create a physically-valid reaction and to meet the desired state (found response) as closely as possible. Our system finds the transition in two steps based on these goals. We create the second simulation utilizing a joint controller which uses the information of the upcoming motion sequence in its desired state calculation. The joint-torque controller allows the character to appear conscious during the transition but lacks the knowledge to make directed
coordinated adjustments to the motion. Therefore, as a last step we perform interpolation between the simulated motion and the found reaction clip to create the final motion.

During the transition period, our controller follows a sequence of joint angles, $\theta_d(t)$, blended between the initial motion capture segment and the response clip. The controller uses an inertia-scaled PD-servo [74] at each joint to compute torques as:

$$\tau = I(k_p(\theta_d(t) - \theta) - k_v(\dot{\theta})),$$

(4.3)

where $\theta$ and $\dot{\theta}$ are the simulation’s current joint angles and velocities. $I$ is the inertia matrix of the outboard body for each joint. $\theta_d(t)$, is generated by interpolating the intermediate postures from the two motion capture sequences before and after the transition. The blend is generated with spherical linear interpolation (slerp) with an ease-in/ease-out weighting parameter over the transition interval. A hand-crafted values of $k_p$ and $k_v$ are held fixed for the entire transition (see Appendix A). (After an initial tuning phase, we found satisfying results across many examples without the need to modify these values.) Our controller does not include joint limits as it is expected to make relatively small adjustments and would likely break for poor matches.

Timing is a critical factor in making the character’s action appear realistic. To render the character’s reaction more human-like, we introduce a deliberate delay before the introduction of the active response simulation. With a small delay or no delay, the character appears to be anticipating the contact and with a large delay, the character responds slowly, as if stunned. We used the delay time
of 0.1-0.2 s (which falls in the range of real human response time), which made the character’s response to the interaction much more believable. Because the controller acts while the impact is happening, our system creates both active and passive dynamic responses simultaneously. The generated active response is then blended into the found response clip using interpolation, to remove remaining differences. We linearly interpolate the root node position, distributing the global position error across the simulated sequence. Joint angles are interpolated by slerping using a simple linear weighting in time across the transition. While this approach is extremely simple, it produced equivalent or better results when compared to more elaborate optimization schemes we investigated.

4.3 System Implementation

The character we chose included 16 body parts connected by ball joints (see Appendix A). The joint angles together with the global position constituted 51 degrees of freedom. We selected a skeleton with the proportions between the two recorded subjects, both medium build with heights of 5’ 11” and 6’ 2”. The search engine intermittently selected motions from both actors and does not show a preference towards motions of a particular performer.

4.3.1 Dynamic Simulation

The dynamic simulation with collisions was generated by Open Dynamic Engine (ODE) [63]. We chose 3D ball joints so that we could easily switch between
motion capture and dynamics. Mass and inertial parameters were derived from basic geometric models for the bodies similar to those seen in the figures (see Appendix A). The same model parameters were used for all motions and in our custom collision handler for the foot/ground contact. The handler used a penalty method with Coulomb friction plus additional rotational friction to impede free spinning movement between the foot and the ground. Since no collisions were computed during the final blend, some inter-penetration is visible in the final animation results.

In order to create believable exchanges between characters, heavy impact-based interactions required impact simulation of both the recipient and the attacker. When the aggressor simply followed the original attack motion sequence it typically lead to unrealistic collision forces. This forced us to include the additional effect of the response for the attacker into our simulation. Conveniently, we observed that first following the simulated motion for a few of frames after the impact and then blending back to the original motion through interpolation lead to a convincing attack motion. With a quick but visible perturbation, the hitter responded to the impact and then continued to complete the attack motion.

4.3.2 Motion Capture Database

A wide variety of believable responses in the motion library is important for producing high quality results. In our motion collection we included recordings of two actors sparring as well as practicing kenpo and judo strikes and recoveries. The repository contained segmented strikes and pushes as well as various
reactions based on contact varying from light to heavy. Although strong strike reactions were not recorded to avoid injury to the actors, strong pushes were captured without any harm to the performer. Front, side and back pushes were recorded with reactions ranging from balanced recovery that required minimal foot repositioning, to a leap or step recovery, to a loss of balance resulting in a forward or backward judo-style somersault. Strikes included various kenpo kicks and punches. In total, our library contained 110 single-actor clips which ranged from 3 to 10 seconds each.

4.4 Results

With our system we are able to create a variety of dynamic responses under various conditions. Figure 4.3 shows a range of such responses generated from a pair of motion clips found by simply varying the facing direction of one of the characters. Figures 4.1, 4.4 and 4.5 show several different attack scenarios and their resulting responses. The scenarios shown in our results were set-up to assure strong impact. However, our system can as well manage the situations when the contact leads to small disturbances. In such cases the optimal solution for the attacked is often to complete the current motion capture playback after a brief period of simulation, as done by the attacker. For heavier impacts, we also anticipate good performance with the limiting factor being the collision handler. ODE’s collision solver uses constraints to resolve collisions which can lead to undesirable inter-penetrating bodies for large contact forces.
During run-time, the search was the slowest part of the system, taking about seventy percent of the computation time, and the results shown here were computed at approximately one minute each (without graphics.) While this is admittedly too slow for games, we note what Mandel describes in detail in his thesis [41]: that search time can be controlled directly by choosing the number of reaction examples in the search database. And indeed, when we were developing our system, we found searching over a core set of reactions lead to runs much closer to real-time. We did choose to include our entire database of reactions to produce the highest quality motion and our timing reflects this extreme.

4.5 Discussion

We propose a simple method for generating physically plausible response to an unexpected impact by composing motion capture fragments with a brief injection of dynamic simulation. Our approach takes advantage of the concept of a burst following an impact to create natural-looking response without the need for a complicated implementation. It may seem tempting to further simplify our method by performing basic interpolation following the first, naive rag-doll-like simulation trajectory described. However, the active control is critical for generating realistic, lifelike motions. Our active simulation’s controller follows a changing trajectory (based on the found response motion) and creates the illusion of the character moving in a conscious manner. And since the active simulation moves towards the desired posture found in the selected response, both an active reaction
Figure 4.3: Range of examples deriving from a single pair of motion clips with varying facing direction of the kicked character.
and a passive response to the collision are simultaneously incorporated into the synthesized motion in a physically-based manner. Additionally, the active simulation produces visible secondary effects, such as physically based foot slipping which is more desirable than the foot skate caused by interpolation. Sometimes physically based slipping is appropriate, when a reaction calls for a fast corrective change in stance for example. Similarly, the active control produces correct changes in momentum when certain actions are performed, such as throwing out an arm or leg for balance. These physically based characteristics can change the resulting motion in a manner consistent with the upcoming motion, but are only achievable if the simulated character is actively moving in anticipation of the chosen response in addition to passively responding to the collisions happening based on the current interaction.

Currently, our system does not manage multiple contacts in a series which would require a secondary search (or beyond) to find a good transition-to clip. Therefore a one-two punch would likely lead to an undesirable behavior since we do not anticipate the subsequent contact in the motion search. Also, sustained contact, like holds in wrestling, remains unexplored and provides an interesting direction for future work.

In this chapter we presented a method for generating physically-valid impact responses by incorporating physical simulation into the motion composition process. In the next chapter we show that certain types of motion, such as acrobatic flips and jumps, can be generated in a physically correct manner without the need for physical simulation. This can significantly improve the performance of
motion synthesis, as dynamic simulation remains a computational bottleneck.
Figure 4.4: Three sample animations (vertically top to bottom) show a range of possible uses for our technique.
Figure 4.5: More animation examples (vertically top to bottom) and a comparison of generated motion with original motion capture recording (in yellow).
CHAPTER 5

Motion Composition for Acrobatics

5.1 Introduction

In 1905 Francis Gouleau of France achieved the first double backward somersault from a stand [69]. Many modern acrobats have tried to repeat this feat and failed. On the other hand, a single backward somersault from a stand can be performed by even a beginning gymnast. In this chapter we propose a technique that can recreate Gouleau’s performance in a fraction of a second using a motion capture recording of a single backward flip (see Figure 5.6a). Our algorithm takes as an input a simple ballistic motion, such as a single somersault, and creates a more complex animation, such as a multiple somersault, by rotating and repeating fragments of motion, while maintaining physically-valid momentum profiles (see Figure 5.1).

Our algorithm automatically applies simple motion editing operations to create superhuman motions or acrobatic stunts that would be difficult to record in a motion capture studio. Because it does not employ computationally expensive algorithms, our work can produce complex acrobatic motions in milliseconds after a pre-processing step of two seconds. Therefore, our technique can be used to
create multiple animated characters performing acrobatic motions in real-time. At the same time, by exploiting characteristics of ballistic motions, the proposed technique creates animations that satisfy the laws of linear and angular momentum conservation. Our algorithm can be applied to a variety of ballistic motions but is especially useful in motions with a significant amount of rotation during the flight phase, such as acrobatic flips and twists, skiing and snowboarding tricks, or martial arts moves.

Additionally, we present a method which allows an animator to modify an existing motion by changing the character’s poses during flight. Such pose modifications can cause changes in the character’s center of mass position and overall body inertia. Our algorithm automatically adapts the modified motion to assure conservation of linear and angular momentums.

Reusing motion capture data allows us to preserve the unique style of a performer more easily than with other methods, such as physical simulation or optimization techniques. While new optimization methods, for example the one
proposed by Liu et al [38] based on joint preferences, go a long way towards explaining biological aspects of motion style, there are still elements of style that cannot be easily characterized and are best captured by data-driven techniques. For example, acrobatic coach and human motion researcher Hartley D. Price compliments Russian athletes on their skills: ”Russians were not content to merely perform the skill. They went one step farther and added a breathtaking flavor to the performance, leaving no doubt where the complete understanding was.” [69]. While highly skilled athletes can add certain flavors to their performance, beginners often make small errors which also make their motion unique and difficult to create with generative methods.

The challenge of reusing motion capture data lies in the fact that complex, multi-flip jumps differ considerably from their simple, single-flip counterparts. While the take-off and landing poses do not change significantly with a varying number of flips (see Section 5.4.3), in the multi-flip motions the rotation is faster and the flight phase longer. This results in the need for a more rapid build-up of momentum in the take-off phase and its quicker release during landing. To account for these differences, our technique adjusts motion timing and modifies flight trajectory. The trajectory modifications can introduce small changes in the take-off angle, while numerical approximations during the retiming step can cause small errors in angular momentum computations. However, we found that these changes are not perceptible in the motions generated with our method. To validate this result, we conducted a user study. We knew from previous research that most humans have difficulties noticing even significant discrepancies
in angular momentum when observing motions of simple objects [49]. We were curious to find out if these results could be extended to human motion as well. Our results show that motions with small changes in angular momentum (both smooth and abrupt) as well as changes in take-off angle were not perceived as less correct than the original motions. In fact, surprisingly, sometimes they were perceived as more correct. The detailed results of our study are presented in Section 5.7.

In this chapter we present the following contributions:

• an efficient technique for creating complex acrobatic motions with multiple flips from simple jumps, while obeying momentum conservation laws,

• an algorithm for adapting motions modified by an animator, to assure the conservation of linear and angular momentum after the modifications,

• a method for estimating mass distribution from motion capture data, which increases the accuracy of momentum computation,

• results of a study on human perception which measures sensitivity to errors in angular momentum and take-off angle.

5.2 Overview

Our algorithm works in two stages: a pre-processing stage, which needs to be executed only once when a new motion is added to the system, and a run-time jump generation stage (see Figure 5.2). In the pre-processing stage, we first find a
ballistic phase in the input motion. Next, taking advantage of linear momentum conservation, we estimate our character’s mass distribution and inertia tensors for each body part. Finally, given the masses and the inertias, we compute the linear and angular momentum in each motion frame.

In the run-time jump generation stage (see Figure 5.3), we first employ a search algorithm to identify fragments of motion that can be rotated and joined together to create a more complex performance. Next, we retime the resulting motion to maintain continuity of momentum curves. As a last step, our algorithm repositions the character so its center of mass follows physically correct trajectory.

We also provide functionality for user motion editing. An animator can, for example, change the position of the legs from tucked to straight to create a more difficult jump. When legs are straightened, their distance from the center of mass increases and the jump rotation must slow down. Additionally, if limbs are positioned asymmetrically, changes in body inertia can affect the body’s axis of rotation. Our algorithm can automatically adjust the edited motion to assure physical correctness. Furthermore, different jumps can be joined together during the flight and adjusted to ensure correct momentum profiles.

We describe the preprocessing steps of our algorithm in Section 5.3 and the run-time jump generation stage in Section 5.4. The user motion editing functionality is presented in Section 5.5.
5.3 Pre-processing Stage

In the pre-processing stage, our algorithm computes motion parameters such as mass distribution, linear and angular momentums, and jump duration (see Figure 5.2). These parameters need to be computed only once for each motion and can then be used multiple times in the run-time jump generation stage to create various ballistic motions.

5.3.1 Ballistic Phase Detection

In this step of our algorithm we determine where ballistic phases occur in the input motion. This information is first used to estimate the character’s masses, then to run the search algorithm, and finally to create a new motion and retime it correctly. Our jump phase detection algorithm is based on the constraint method proposed by [39]. That is, we recognize that during the take-off and landing phases the character’s feet or hands remain planted on the ground for a certain amount of time. We don’t know the exact position of the ground in
our motions (there can be multi-level environments). However, we can assume that intervals where at least one end effector is not moving significantly are good candidates for border frames of the ballistic motion.

Sometimes during the ballistic phase, one of the limbs can remain in the same place for significant time. Therefore, we add an additional condition, based on the pattern of linear momentum build-up and release. Typically, during take-off, linear momentum slightly drops and then suddenly increases. Similarly, during landing, there is first a sudden drop in momentum and then a slight increase. We used the momentum change conditions (first down, then up) together with the constraint method to find the longest ballistic interval in each motion. With this technique we managed to correctly find jump phases in all of our motion clips. Because in this step we do not yet know the character’s estimated mass distribution, we use an average mass distribution of a human body (obtained from [11]) to compute the linear momentum. In our results this approximation did not affect the accuracy of the output.

5.3.2 Estimating Mass Distribution

Applying momentum conservation laws to motion capture data is a key element of our approach. The knowledge of a character’s mass distribution, though typically not provided with the motion data, is necessary for the computation of linear and angular momentums. Since we are working with motions with long ballistic phases, we have the advantage of being able to compute the mass distribution from the input motion alone. During a flight phase, the character’s center of
mass (COM) over time follows a line in the $x$ and $z$ dimensions and a parabola in the $y$ (vertical) dimension due to gravity. Using this property we compute the masses of the body parts which minimize the distance between the character’s COM calculated from the motion data and the correct COM trajectory, which is a line in $x, z$ dimensions and a parabola in $y$.

To achieve this, we formulate a simple linear least-squares optimization problem:

$$
\min_{m_1,\ldots,m_n; a_x,b_x,a_z,b_z; a_y,b_y,c_y} \left\| \left( \sum_{i=1}^{n} \frac{m_i}{M} \text{pos}_i(t) \right) - \text{COM}(t) \right\|^2 \quad \text{for } t = 0 \ldots T, \quad (5.1)
$$

with additional conditions that the sum of the masses is constant: $\sum_{i=1}^{n} m_i = M$ and that $m_i \geq 0$. In the above equations $T$ is the duration of the flight phase, $\text{pos}_i(t)$ is the position of $i$-th body part at time $t$ obtained from the motion data and $\text{COM}(t) = [a_xt+b_x, a_yt^2+b_yt+c_y, a_zt+b_z]^T$ is the correct COM trajectory.

Note that in our optimization we search both for mass distribution and the COM trajectory parameters $a_x, b_x, a_y, b_y, c_y, a_z, b_z$. We do not use the real-world gravity coefficient in our model because in motion capture data the character’s joint lengths are often scaled compared to the real-world values. Using the results of our optimization we can compute the scaling factor as: $s = 2a_y/g$, where $g$ is the real-world gravity coefficient.

The $3(T+1)$ equations constructed from (5.1) don’t always have a unique solution. To minimize the number of optimized variables, we assume perfect symmetry between right and left limbs. Even with this modification, we found that the optimization still commonly produced solutions unrealistic for human body

45
mass distribution. Although our optimization algorithm allowed us to specify a reasonable mass distribution $m_i^0, \ldots, m_n^0$ as a starting point, the results of the optimization still suffered from overfitting. Overfitting occurred because the optimization algorithm attempted to "explain" inherent errors in the motion data. This caused large shifts in mass distribution, which resulted in only small improvements in the optimization results. To alleviate this problem, we added another equation to our problem formulation, which made the optimization algorithm favor solutions close to the initial distribution:

$$\min_{m_i} W \|m_i - m_i^0\|^2 \text{ for } i = 0 \ldots n,$$

(5.2)

where $W$ is a small constant. This formulation removes the overfitting problem as only large improvements in the optimization can sway the results significantly from the initial solution.

Given the mass distribution and the skeleton proportions obtained from the motion capture data, we estimate the inertia tensors of all body parts. First, assuming constant density of each body part, we compute the parts volumes. We obtain typical densities of human body parts from [11]. Next, we approximate each part as a box, whose length is determined by the skeleton proportions and whose width and depth remain in a constant ratio. We derive the ratio for each box from human body measurements ([11]) and use it to calculate the depth and width of the box. Finally, given the body parts’ dimensions and masses, we compute inertia moments for each part using the formula for solid cuboid:

$$I_i = \frac{1}{12} m_i \left[ w_i^2 + d_i^2, l_i^2 + d_i^2, l_i^2 + w_i^2 \right]^T,$$

(5.3)
where \( l_i, w_i \) and \( d_i \) are \( i \)-th body part length, width and depth respectively and \( m_i \) is its mass.

In our method we take advantage of the regular shape of COM trajectory during the flight phase due to the linear momentum conservation. We also tried to utilize the angular momentum conservation property to increase the number of constraints in the optimization problem. Unfortunately, computed derivatives contained too much noise, due to numerical errors in our motion data, to be useful in our optimization.

Our technique of mass estimation is useful for momentum computations as it reduces the deviation of linear momentum during the flight phase from the physically correct pattern. In our computations the reduction was 30\% compared to the momentum values computed with the initial mass distribution. It is difficult to evaluate how well our method estimates the real-world mass distribution of a performer, since we do not know the actual masses of our performer’s body parts (hence the need for estimation). However, a video recording of the motion capture session showed that the performer of our motions was tall. Tall males typically exhibit lower weight of limbs and higher weight of the trunk as the percentage of total weight, compared to a person of an average height [10]. The results of our optimization showed the same pattern in the estimated masses compared to the average mass distribution (see Appendix C).
5.3.3 Momentum Computation

Given the mass distribution, computing the momentums is straightforward. The linear momentum $p$ at time $t$ can be computed as:

$$p(t) = \sum_{i=1}^{N} m_i \mathbf{v}_i(t),$$  
(5.4)

where $m_i$ is the mass of $i$-th body part and $\mathbf{v}_i(t)$ is its velocity at time $t$. Angular momentum $L$ for a system of rigid bodies can be computed as follows:

$$L(t) = \sum_{i=1}^{N} \mathbf{R}_i(t)\mathbf{I}_i \mathbf{R}_i(t)^T + m_i (pos_i(t) - COM(t)) \times \mathbf{v}_i(t),$$  
(5.5)

where $\mathbf{R}_i(t)$ is the rotation matrix for $i$-th body part in world coordinates, $pos_i(t)$ and $\mathbf{I}_i$ are the $i$-th body position and inertia matrix respectively, and $COM(t)$ is the character’s center of mass at time $t$.

5.4 Run-Time Jump Generation Stage

In the run-time jump generation stage, our algorithm creates a complex multi-flip motion from an easier, single-flip jump using basic motion editing operations (see Figures 5.2 and 5.3). These operations and their impact on momentum are described in Section 5.4.1. The jump generation stage starts with a search, which finds and joins together fragments of motion to produce a longer sequence (Section 5.4.2). Next, the resulting motion is retimed to assure continuity of angular and linear momentums during take-off, flight, and landing phases (Section 5.4.3). Finally, the motion frames are repositioned to maintain a correct COM trajectory in the ballistic phase (Section 5.4.4).
Figure 5.3: Outline of the run-time jump generation stage of our algorithm. a) Input motion. b) First, the search algorithm finds a self-looping sequence B; root positions are ignored. c) Next, the sequence B is rotated around the angular momentum vector and repeated to obtain a double-flip. d) The resulting sequence is retimed to assure continuity of linear momentum during take-off and landing. e) In the final step, the character’s COM is repositioned to follow a correct trajectory under gravity.
5.4.1 Editing Operations and Their Impact on Momentum

In our algorithm, we employ three basic motion editing operations: repositioning, rotation and retiming by interpolation. In this section we analyze their impact on linear and angular momentum. We also show how these operations are used in our method to maintain the momentum conservation property during flight and assure continuity of momentum during take-off and landing.

**Repositioning of the center of mass.** Changes in the COM trajectory do not affect the character’s angular momentum, which is computed with respect to the character’s COM. We use this property in the last step of the run-time jump generation stage, when adjusting the COM trajectory to the longer flight phase.

**Rotation around the angular momentum vector.** Rotating a fragment of a ballistic motion around the axis defined by a character’s center of mass and the angular momentum vector will not affect the angular momentum within this motion fragment. Intuitively, angular momentum is a vector that remains constant in the given reference frame during the jump duration. If we rotate our reference frame around the axis defined by that vector and the COM, then all points on that axis will remain unchanged. More formally, angular momentum of a system around its COM can be defined as:

\[ L = \sum_i m_i (r_i \times v_i) = \sum_i m_i (|r_i||v_i| \sin \theta_i \mathbf{n}), \]

where \( m_i \) is the mass of a particle, \( r_i \) is the distance from the COM, \( v_i \) is the particle’s velocity w.r.t. the COM, \( \theta_i \) is the angle between \( r_i \) and \( v_i \), and \( \mathbf{n} \) is a unit vector perpendicular to \( r_i \) and \( v_i \). If we rotate the reference frame by some
angle $\phi$ around the axis defined by the center of mass and the angular momentum vector the new angular momentum will be:

$$L_{\text{new}} = \sum_i m_i (|r_i||v_i| \sin \theta_i R(\phi) \mathbf{n})$$

$$= R(\phi) \sum_i m_i (r_i \times v_i) = R(\phi) \cdot L = L,$$  \hspace{1cm} (5.7)

because the axis of rotation is parallel to $L$. We use this property in the sequencing step to increase the number of flips, while conserving angular momentum.

**Motion retiming.** Uniform retiming scales the linear and angular momentum linearly. A motion fragment with linear momentum $p$ and angular momentum $L$ retimed by a factor of $n$ has momentums equal to $np$ and $nL$. This is due to scaling of velocities during retiming:

$$v_i^{\text{new}} = \frac{d}{dt} r_i(nt) = \frac{d}{d(nt)} \frac{d(nt)}{dt} v_i = n v_i.$$ \hspace{1cm} (5.8)

While this is true in the continuous case, applying this rule to motion capture data, where derivatives are computed numerically, causes errors. However, for $\frac{1}{2} \leq n \leq 2$, these errors in our motions are within 20% range and according to our study (Section 5.7) are not perceptible in the resulting motion. We use retiming to assure continuity of linear momentum during take-off and landing and to adjust momentums to motion changes made by an animator.

### 5.4.2 Search and Sequencing

The search algorithm attempts to find a fragment of ballistic motion, which can be repeated in the new sequence after rotation around the axis defined by the
angular momentum vector $L$ and the character’s COM. In other words, it finds a sequence of frames $F_B = f_m, f_{m+1}, \ldots, f_n$ and an angle $\phi$, such that:

$$R(f_m, L, \phi) \approx f_n. \quad (5.9)$$

That is, frame $f_m$ after rotation by $\phi$ around $L$ is similar to $f_n$ (we describe the similarity measure in Section 5.4.2.1). We call the jump fragment $F_B$ with this property a *self-looping sequence*. $F_B$ is then progressively rotated and repeated multiple times to create a longer jump with more flips. For example, in Figure 5.3c the self-looping motion is repeated four times to create a double flip.

In other words, we use the initial motion sequence:

$$F = [F_S, F_B, F_E], \quad (5.10)$$

where $F_S = f_1, \ldots, f_{m-1}$ and $F_E = f_{n+1}, \ldots, f_{end}$, to create a new, longer sequence with more flips:

$$F^{new} = [F_S, F_B, R_B(\phi), R_B(2\phi), \ldots, R_B(r\phi)], \quad (5.11)$$

where $R_B(\phi)$ is the sequence $F_B$ with all frames rotated by $\phi$ around $L$. The parameter $r$ denotes the number of repetitions of $F_B$ in $F^{new}$ and depends on the number of flips we want to obtain in the new motion. For example, in Figure 5.3 we construct a double flip, which involves the root rotation of $4\pi$ around $L$. The rotation around $L$ in $F_B$ is $\phi \approx \frac{3\pi}{4}$ and rotations in $F_S$ and $F_E$ are close to $\frac{\pi}{2}$. Therefore we need to repeat the self-looping sequence four times. We compute $r$ as $r = \lceil (2\pi \cdot \text{num_flips} - \phi_S - \phi_E) / \phi \rceil$, where $\phi_S$ and $\phi_E$ are amounts of rotation around $L$ in $F_S$ and $F_E$ respectively. Note that since we are rotating the self-looping sequence around its angular momentum vector, the angular momentum in
\[ [F_B, R_B(\phi), \ldots, R_B(r\phi)] \] will remain approximately constant when computed with respect to the COM.

The newly created sequence \( F^{new} \) contains a take-off phase and multiple rotations. Next, we perform a second search, to add the landing phase to \( F^{new} \). We find two frames \( f \in [R_B((r - 1)\phi), R_B(r\phi)] \) and \( g \in F_E \), such that \( f \approx g \). The resulting sequence \( F^* \) will consist of \( F_S, [F_B, R_B(\phi), \ldots, R_B(r\phi)] \) truncated to end at frame \( f \), and \( F_E \), truncated to start at frame \( g \).

If the ballistic motion does not contain self-looping sequences (there are no similar frames \( f_m \) and \( f_n \)), our algorithm can still be applied. The transition between any two chosen frames of ballistic motion can be created by an animator or automatically by smoothing. Although during the transition the angular momentum will not be constant, we can adjust it by applying non-uniform motion retiming as described in Section 5.5. The same algorithm can be easily modified to join two different motions during the jump phase. As long as the two motions have the same angular momentum direction, the magnitudes can be adjusted by retiming.

### 5.4.2.1 Similarity Measure

In our search algorithm we use a comparison function which computes the distance between two poses while ignoring their positions in space. Our comparison method is similar to previous approaches [3, 32, 36, 76]. However, in contrast to the above methods, before the comparison we first rotate the skeleton around the angular momentum vector \( \mathbf{L} \) to minimize differences in body orientation between
compared frames.

In more detail, each frame in our system $f = [p, q^r, q^1, \ldots, q^N]$ is described by the character’s root position $p$, root orientation $q^r$ represented as a quaternion and a vector of joint angles $q^1, \ldots, q^N$ also stored as quaternions. Given two frames $f_1 = [p_1, q^r_1, q^1_1, \ldots, q^N_1]$ and $f_2 = [p_2, q^r_2, q^1_2, \ldots, q^N_2]$, we first compute rotation angle $\phi$ around $L$ which minimizes differences between global orientations $q^r_1$ and $q^r_2$. To achieve this we project the angle between root orientations in $f_1$ and $f_2$ on a plane perpendicular to $L$:

$$\phi = P(q^r_2 q^r_1^{-1}, L),$$  \hspace{1cm} (5.12)

where function $P()$ denotes the projection of the angle defined by the quaternion $q^r_2(q^r_1)^{-1}$ on a plane perpendicular to $L$. After computing $\phi$ and obtaining the quaternion $q_\phi$ from angle $\phi$ and axis $L$, we proceed to calculate the distance between the two frames as:

$$D(f_1, f_2) = w_r |angle(q^r_2 q^r_1^{-1})| + \sum_{i=1}^{N} w_i |angle(q^i_2 q^i_1^{-1})|,$$  \hspace{1cm} (5.13)

where function $\text{angle}()$ extracts the rotation angle from quaternion representation and $w_r, w_1, \ldots, w_N$ are the weights for global orientation and joint angles. Similarly to [76], we assign higher weights to joints close to the trunk, as differences in limb positions can be easily smoothed by blending. We used the same set of weights ($w_{\text{root}} = 2$, $w_{\text{trunk}} = w_{\text{thigh}} = 0.7$, $w_{\text{knee}} = 0.5$, $w_{\text{shoulder}} = 0.3$, $w_{\text{elbow}} = 0.1$) for all of our motions.
5.4.2.2 Smoothing

To eliminate small discontinuities in transitions between motion fragments, we smooth joint angles by performing spherical linear interpolation (slerp) over small windows of frames. In most cases this operation causes negligible changes in angular momentum. If two adjoining frames are dissimilar, resulting changes in momentum can be adjusted by non-uniform retiming, the same way as in the case of motion modification (Section 5.5).

Given two neighboring transition frames $f_n$ and $f_{n+1}$, we smooth joint angles by slerping over small frame windows of equal lengths: $W_1 = [f_{n-w}, \ldots, f_n]$ and $W_2 = [f_{n+1}, \ldots, f_{n+1}]$. Special care must be taken when smoothing global orientations, since the rotation around the angular momentum vector $\mathbf{L}$ is already continuous. To maintain this continuity and avoid introducing undesirable artifacts in overall body rotation, we remove the rotation around $\mathbf{L}$ from our slerp for global orientations. To achieve this effect, we split global orientation $q_i^r$ in frame $i$ into two quaternion rotations:

$$q_i^r = q_i^P q_i^L,$$

(5.14)

where $q_i^L$ denotes quaternion rotation around $\mathbf{L}$ and $q_i^P$ is a rotation around an axis perpendicular to $\mathbf{L}$. When smoothing, we slerp only the portion of global orientation defined by $q_i^P$.

Smoothing between the first and the last frame of self-looping sequence needs to be computed only once and can be then re-used for all of the self-looping sequence repetitions. Root positions will be adjusted in the repositioning phase.
(Section 5.4.4) and are not smoothed.

5.4.3 Retiming

In the search step we constructed a complex motion with multiple rotations and a longer ballistic phase compared to the original motion. Differences in the flight phase duration and COM trajectory affect the momentum build-up during take-off and its release during landing. One of challenges here is to adapt the take-off and landing phases to reflect the changes in the flight phase and to assure continuity of linear and angular momentums.

For low-effort jumps, changes in momentum build-up are associated with significant changes in take-off poses. For example, a character might bend the knees more to achieve a longer or higher jump. Surprisingly perhaps, the same is not true for the high-effort ballistic motions, such as acrobatic flips or long jumps, where the athletes are operating close to the limit of their physical abilities. Research suggests that the approach speed is the main factor affecting the performance in high-effort motions, while take-off poses vary little with changing performance levels. Bridgett and Linthorne [8] evaluate long jumps performed by professional athletes with maximum effort and conclude that variations in run up speed account for 96% of the variation in jump distance. Seyfarth, Blickhan and van Leeuwen [59] report that the optimal jumping technique defined by leg angles is not affected by changes in approach speed or muscle design. In acrobatic motion research, King and Yeadon [26, 27] present data on joint angles during take-off for a single and double backflip from a round-off (see Table 5.4.3).
<table>
<thead>
<tr>
<th>Joint</th>
<th>Single flip (deg)</th>
<th>Double flip (deg)</th>
<th>Ang. vel. (deg/sec)</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ankle</td>
<td>125</td>
<td>125</td>
<td>821</td>
<td>0.00</td>
</tr>
<tr>
<td>knee</td>
<td>174</td>
<td>168</td>
<td>278</td>
<td>0.02</td>
</tr>
<tr>
<td>hip</td>
<td>175</td>
<td>202</td>
<td>715</td>
<td>0.02</td>
</tr>
<tr>
<td>trunk</td>
<td>78</td>
<td>99</td>
<td>931</td>
<td>0.02</td>
</tr>
<tr>
<td>shoulder</td>
<td>154</td>
<td>153</td>
<td>157</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Table 5.1: Comparison of take-off poses for a single and double backflip. Differences between the joint angles in the two take-offs are negligible: with given angular velocities, it takes less than a duration of one frame to transition between them. Data adapted from [26] and [27].

Differences in joint angles are negligible: it would take less than 0.02 second to transition between these two take-offs, due to high angular velocities of body parts.

While there is less scientific data on the landing phase, our conversations with gymnastic coaches [5, 47] reveal that a perfectly executed double flip will have a very similar landing to a correctly carried out single flip, but will be performed faster.

Since in high-effort motions take-off and landing do not vary significantly with performance level, we can achieve natural-looking motions and eliminate momentum discontinuities by simply retiming motion fragments so that they are performed faster. We retime the take-off, flight and landing phases by the same factor $k$. Retiming will scale the angular momentum by a factor of $k$ in all phases.
of motion. During flight, linear momentum will be adjusted by repositioning of the character’s COM, but retiming also increases linear momentum \( k \) times during take-off and landing, and reduces the duration of the ballistic phase by a factor of \( k \). In Appendix B we show that there exists a single scaling factor \( k^* \) which assures the continuity of the linear momentum curve. In practice however, small discontinuities in linear momentum are not perceptible to the human eye. Therefore we can adjust \( k \) to obtain different jump heights and rotation speeds. In our results we typically chose \( k \approx 0.7k^* \), as we found higher jumps with slower rotations to be more aesthetically pleasing.

### 5.4.4 Repositioning

In this step we adjust the character’s center of mass in each flight frame of the newly constructed motion, so that it follows a physically valid trajectory. Computing the COM position along \( x \) and \( z \) axes is simple, given values \( a_x, b_x, a_z \) and \( b_z \) computed in the mass estimation step (Section 5.3.2) and retiming factor \( k \) chosen in the retiming step (Section 5.4.3):

\[
\begin{align*}
COM_{x}^{new}(t) &= k a_x t + b_x \\
COM_{z}^{new}(t) &= k a_z t + b_z \\
t &= 0 \ldots T^{new}.
\end{align*}
\]  

(5.15)
where $T^{\text{new}}$ is the duration of the ballistic phase in the new motion. We also offset the COM trajectory after the jump to maintain the continuity of motion:

\[
\text{offset}_x = \text{COM}^{\text{new}}_x(T^{\text{new}}) - \text{COM}_x(T)
\]
\[
\text{offset}_z = \text{COM}^{\text{new}}_z(T^{\text{new}}) - \text{COM}_z(T).
\] (5.16)

In the case of the $y$ (vertical) axis, we need to modify the parabolic trajectory by scaling the initial velocity $b_y$ at $t = 0$:

\[
\text{COM}^{\text{new}}_y(t) = a_y t^2 + k b_y t + c_y \quad \text{for } t = 0 \ldots T^{\text{new}},
\] (5.17)

The change in initial velocity causes a (typically small) change in the take-off angle. However, our study indicates (Section 5.7) that such small changes are not perceptible.

Using the above formulas, we can also make non-trivial modifications to the character’s trajectory by modifying $c_y$ and adjusting the scaling factor $k$ (see Appendix B). For example in Figure 5.8 we changed the height from which the character takes off. In the original motion the take-off and landing were at the same level.

### 5.5 User Modifications

Our method allows an animator to modify an existing motion by changing the character’s poses during the flight. For example, an animator can extend the character’s arms or change the legs position from tucked to straight. Such modifications can alter the character’s overall body inertia which in turn can cause
changes in angular momentum that violate the law of momentum conservation. Our algorithm automatically adjusts the edited motion to assure conservation of linear and angular momentums (p and L) after the modifications. The method works in three steps. First, we adjust the character’s orientation in each frame to assure that the direction of L remains constant during flight. Next, we perform non-uniform retiming to eliminate changes in the magnitude of L. Finally, we assure conservation of linear momentum p by uniform retiming and repositioning of the character’s COM.

5.5.1 Changes in Direction of L

Motion modifications can introduce changes in a character’s inertia and alter the axis of jump rotation. For example, high divers often position their arms asymmetrically: one arm above the head and the other one extended to the side, to perform a twist around the vertical axis together with a somersault [13]. We would like our character to respond in the same manner when an animator adjusts its arm positions (see Figure 5.10). To achieve that effect, in the first step of our algorithm we modify the character’s orientation (differently in each frame) so that the direction of angular momentum remains constant during flight.

In our angular momentum computations we use finite differences to estimate linear and angular velocities. In other words, we compute angular momentum at a frame f (denoted as Lf) as a function of the character’s pose (joint angles) in
frames $f - 1$, $f$ and $f + 1$:

$$L_f = L(\text{pose}_{f-1}, \text{pose}_f, \text{pose}_{f+1}),$$  \hspace{1cm} (5.18)

where $L()$ is our angular momentum computation function (see Section 5.3.3).

Since modifications are introduced during the flight, at the beginning of the jump (at frame $s$) angular momentum remains unchanged:

$$L_{\text{new}}^s = L^*,$$  \hspace{1cm} (5.19)

where $L_{\text{new}}^s$ is the angular momentum at frame $s$ after the modifications and $L^*$ is the constant angular momentum during the flight phase before the modifications.

At some point during the flight phase, the introduced modifications can cause changes in the angular momentum vector. Let $f^*$ be the first frame at which the change occurs:

$$L_{f^*}^{\text{new}} \neq L^* \text{ and } L_{f^*-1}^{\text{new}} = L^*.$$  \hspace{1cm} (5.20)

Since $L_{f^*-1}^{\text{new}}$ is computed based on frames $f^* - 2$, $f^* - 1$ and $f^*$, the character’s orientation in these frames must be correct. Therefore to assure that the direction of angular momentum is preserved in the frame $f^*$, we have to adjust the character’s orientation in the frame $f^* + 1$. In other words, we need to find a rotation $R$ such that:

$$\text{unit}(L(\text{pose}_{f^*-1}, \text{pose}_{f^*}, R \text{pose}_{f^*+1})) = \text{unit}(L^*),$$  \hspace{1cm} (5.21)

where $\text{unit}(v) = \frac{v}{\|v\|}$ is the unit vector in the direction of $v$.

Although computing the rotation $R$ analytically is not trivial, $R$ can be easily estimated with optimization methods. In our algorithm we employed the
Quasi-Newton BFGS method [9, 16, 18, 60] provided by the Matlab optimization toolbox. As a starting point (initial solution estimate) for our optimization we choose the rotation between $L_{f_*}^{new}$ and $L^*$, which typically assures fast convergence to the optimal solution. Once we find $\mathbf{R}$ and rotate our character in frame $f^* + 1$, we repeat the same process for all subsequent frames. At that point we use the rotations found in the previous iterations as starting points. Due to the small number of optimized variables and the good initial solution estimate, our algorithm manages to practically eliminate errors in the direction of $\mathbf{L}$ (see Figure 5.4).

![Figure 5.4: Differences in the angular momentum direction between an original motion and a motion modified by an animator (in radians) before and after the first step of our algorithm.](image)
5.5.2 Changes in Magnitude of $\mathbf{L}$

After changes in direction of $\mathbf{L}$ are eliminated, differences in the magnitude of $\mathbf{L}$ can be removed by non-uniform retiming of the motion. Since momentum in the modified motion changes over time, the retiming factor must change as well. We create a new, retimed frame sequence by computing which frame should be displayed next, given the number of the previous frame in the retimed sequence:

$$\text{next\_frame} = \text{last\_frame} + \frac{1}{\text{ratio}(\text{last\_frame})},$$

(5.22)

where $\text{ratio}(f) = ||\mathbf{L}_{f}^{\text{new}}||/||\mathbf{L}^*||$ is a ratio between the angular momentum magnitude in the modified motion (after direction adjustments) at a frame $f$ and the angular momentum magnitude in the original motion during flight. For in-between frames, we linearly interpolate the joint angles and compute the angular momentums from the interpolated values. While retiming does not eliminate all angular momentum errors, it reduces them significantly (see Figure 5.5). Before the retiming, angular momentum error was over 50% and according to our perception study could be detected by most viewers (see Table 5.2). Our method reduces the error to 20% or less, so that according to our study it is no longer perceptible.

5.5.3 Final Retiming and Repositioning

In the two previous steps we assured conservation of angular momentum by adjusting the direction and magnitude of the angular momentum vector in the modified motion. In the final step we ensure the property of linear momentum
Figure 5.5: Errors in the angular momentum magnitude in a motion modified by an animator before and after retiming relative to the original motion. Retiming causes significant reduction in the relative error.

conservation. The non-uniform retiming described in the previous section can cause changes in the jump duration. To ensure proper momentum build-up and release, we retime the motion uniformly, similarly as in Section 5.4.3. Next, we reposition the character’s center of mass (as in Section 5.4.4) so that it follows a correct trajectory under gravity.

5.6 Results

We applied the described techniques to a variety of acrobatic motions from the CMU motion capture database [1] and commercial databases. Figure 5.6 presents three different double somersaults generated from their single-flip counterparts. The generated motions exhibit faster rotation and more rapid take-off and landing compared to the original jumps, to accommodate changes in momentum due to the longer ballistic phase. The flight trajectory is also adjusted, resulting in
higher and longer jumps. Our method can be also applied to simpler, non-acrobatic motions, such as twist jumps. Although in human twist jumps take-off poses can vary significantly with the achieved twist rotation, small adjustments of the amount of rotation typically produce natural-looking results. Figure 5.7 shows a range of different twists generated with our method varying in rotation from 180 to 270 degrees. In the generated motions the take-off, angular velocity and jump height increase with the achieved degree of rotation.

Figure 5.8 presents the results of joining together two acrobatic motions, an aerial cartwheel and a forward flip, to create a new stunt. The motions are retimed so that the angular momentums during the flight phase are equal. This example also shows the capability of our technique to easily change the height from which the character takes off.

Figure 5.9 shows the results of user’s editing operations that affect only the angular momentum magnitude. Tucked forward and backward flips are modified by an animator to become straight-leg jumps. Our system retimes and repositions the motions to adapt to inertia changes. This causes the flight phase to be longer with slower rotation and greater jump height compared to the original motion. Take-off and landing phases are also modified due to the change in length of the flight phases.

Lastly, Figure 5.10 presents the result of a user’s editing operations that affect both the direction and magnitude of the angular momentum. In this high dive our character extends its arms asymmetrically, similarly to real-word divers, to achieve a somersault with a twist. In the first step, our algorithm creates the twist
by automatically adjusting the character’s orientation to assure conservation of the angular momentum direction. In the second step, the motion is retimed to conserve the angular momentum magnitude. Finally, the repositioning step ensures correct trajectory under gravity and guarantees that the character’s body does not touch the platform during the fall phase.

**Running time.** The pre-processing stage of our algorithm took about 2 seconds. The run-time jump generation stage for a two-flip jump took 3.69 ms, a three-flip jump 4.60 ms and a four-flip jump 5.09 ms.
Figure 5.7: Various twist jumps generated with our method. a) An original 180 degrees twist jump. b) A 210 degrees twist jump. c) A 240 degrees twist jump. d) A 270 degrees twist jump. e) Comparison of all twist jump motions.
Figure 5.8: Joining together an aerial cartwheel motion (a) and a forward flip (b) creates a new acrobatic motion (c).

Figure 5.9: The results of a user’s editing operations that do not change the direction of the angular momentum vector. The original motions (in yellow) are adapted by non-uniform retiming to assure momentum conservation. a) A straight-leg forward flip. b) A straight-leg backflip.
5.7 Experimental Evaluation

We studied human sensitivity to errors in angular momentum and take-off angle, to find out how the approximations introduced by our method affect viewers’ perception of the generated motions.

Participants. The study involved 19 student volunteers (3 women and 15 men) ranging in age from 20 to 25 years (3 participants did not specify their age). None of the volunteers had significant experience in computer animation or gymnastics.

Stimuli. Animations of single-flip forward and backward somersaults were created as a stimuli. All animations were rendered in the same style with the same camera configuration and the same take-off position. Original motion capture clips were used as base motions. Errors in angular momentum were created by the motion retiming during the flight phase. We tested both abrupt changes - sudden introduction of retiming in the middle of the flight phase, and smooth changes in angular momentum - retiming factor growing linearly and achieving maximum value just before landing. Both abrupt and smooth errors ranged from -50% to 50% of the original angular momentum magnitude. The errors didn’t change the direction of the angular momentum vector. Errors in take-off angle were introduced by modifying the linear velocity in the flight phase along the direction of the jump and ranged from 0 to 40%.

Procedure. Participants were told that they were about to see a sequence of motions, some of which were physically correct and some of which were not. They were not informed how many of the motions contained errors or what kind
of errors were introduced, but they were shown examples of correct motions to give them a reference point. Participants were instructed to mark each of the test motions either as correct or incorrect. The test motions were arranged randomly within each error group and then interleaved to reduce the learning effect. Each motion was shown twice.

**Results.** Similar to findings reported by [49] for simple objects, our subjects were not sensitive to even significant changes in angular momentum during ballistic motion. For example, for both smooth and abrupt changes, a 25% increase in angular momentum was imperceptible and as likely to be classified physically valid as the original motion. Surprisingly, motions with smooth decreases in momentum often scored higher than the original motions. This might be caused by humans overestimating the effect of air friction on the motion.

Likewise, our subjects proved not to be sensitive to errors in the take-off angle. A 30% change was perceived as physically valid as often as the original motion. Perhaps even more surprising is the fact that the higher error didn’t necessarily cause increased sensitivity: a 10% change scored significantly higher than the original motion and a 30% change was viewed as correct more often than a 20% change. This shows that most humans have relatively little experience with high-effort ballistic motions and find it difficult to estimate their correctness. The results of the study are summarized in Table 5.2.
<table>
<thead>
<tr>
<th>Angular momentum modifications</th>
<th>0%</th>
<th>17%</th>
<th>25%</th>
<th>37%</th>
<th>50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>smooth</td>
<td>72</td>
<td>71</td>
<td>72</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>increase</td>
<td>(.00)</td>
<td>(.05)</td>
<td>(.00)</td>
<td>(.81)</td>
<td></td>
</tr>
<tr>
<td>smooth</td>
<td>72</td>
<td>79</td>
<td>78</td>
<td>78</td>
<td>62</td>
</tr>
<tr>
<td>decrease</td>
<td>(.00)</td>
<td>(-.22)</td>
<td>(-.18)</td>
<td>(-.18)</td>
<td>(.27)</td>
</tr>
<tr>
<td>abrupt</td>
<td>72</td>
<td>76</td>
<td>78</td>
<td>48</td>
<td>44</td>
</tr>
<tr>
<td>increase</td>
<td>(.00)</td>
<td>(-.13)</td>
<td>(-.18)</td>
<td>(.63)</td>
<td>(.75)</td>
</tr>
<tr>
<td>abrupt</td>
<td>72</td>
<td>83</td>
<td>68</td>
<td>33</td>
<td>22</td>
</tr>
<tr>
<td>decrease</td>
<td>(.00)</td>
<td>(-.38)</td>
<td>(.11)</td>
<td>(1.0)</td>
<td>(1.4)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Take-off angle modifications</th>
<th>0%</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
</tr>
</thead>
<tbody>
<tr>
<td>increase</td>
<td>63</td>
<td>81</td>
<td>50</td>
<td>63</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>(.00)</td>
<td>(-.53)</td>
<td>(.34)</td>
<td>(.00)</td>
<td>(1.3)</td>
</tr>
</tbody>
</table>

Table 5.2: Results of our study: mean ratings (in percent) and sensitivity levels (in parenthesis). Sensitivity less or equal to zero means that participants could not detect errors.

### 5.8 Discussion

We presented a method for creating complex ballistic motions from simpler jumps and a technique for adapting user-modified motions to assure their physical validity. The proposed techniques are efficient as they use only basic editing operations. Our approach works especially well with high-effort motions, which do
not exhibit significant changes in take-off and landing poses with varying levels of effort. Nevertheless, the presented methods can be also applied to low and medium-effort motions, provided that the resulting changes in energy expenditure at take-off are small. However, extreme changes, such as transforming a 45 degree twist jump into a 360 degree one, would be perceived as unnatural. It would be interesting to explore the possibilities of using our technique jointly with optimization methods, such as the ones proposed in [65] or [39], in order to adapt take-off and landing poses to varying levels of effort.

Furthermore, our method for adjusting motions after user modifications (Section 5.5) does not modify the character’s poses but only alters rotation and timing of the motion. This may produce unrealistic results, as humans often adapt their bodies differently, depending on the direction of the rotation. In our method, the burden of assuring the correct positioning of the character’s body lies on an animator.
Figure 5.10: The results of a user’s editing operations that change both the direction and the magnitude of the angular momentum vector. a) Input motion. b) Motion after znanimator’s modifications. c) First step of our algorithm: the character is rotated in each frame to assure conservation of the angular momentum direction. d) Final steps of our algorithm: non-uniform retiming assures conservation of the angular momentum magnitude, repositioning guarantees linear momentum conservation.
CHAPTER 6

Automatic Splicing for Hand and Body Animations

The previously described methods compose motion fragments along the time axis. In this chapter we propose a different approach - layering motions in time to obtain a multi-resolution effect (see Figure 6.2). We apply this tactics to add detailed hand motion to a full-body animation. In computer animation, hand motion is typically either keyframed by an animator or added painstakingly by hand, starting from motion capture data recorded during dedicated hand capture sessions. While the development of methods for re-use and modification of motion capture data is an active area of research, little attention has been paid to automatic methods for integrating hand motion into full-body animations. Although it is possible to record hand movement together with full-body motion in a simultaneous capture, there are compelling reasons for recording hand movement separately. These include greater flexibility in the use and re-use of hand motions across actors and animated scenes, and increased accuracy and control over the hand capture. The latter is especially true for optical technologies, where resolution and occlusion are key factors in the quality of the captured motion.
Figure 6.1: Mudras in Indian dancing.
Figure 6.2: Layering of full-body and hand motions produces character animation with detailed hand movement. Dynamic Time Warping (DTW) assures correct synchronization of the two motions.

We present a motion composition technique that seamlessly splices together hand and body motions recorded in separate sessions, while maintaining synchronization between the two motion capture sources. Finding the correct matching between hand and body motions recorded separately can be a challenging problem due to differences in motion execution. In particular, timing and amplitude variations can cause two recordings of the same movement to have very different numerical representations. Differences in timing arise when parts of two motions are performed at different speeds. Amplitude variations are caused by different execution of the motion, for example, a performer extending his arm further in one performance over another. This problem is prevalent when high resolution hand motion is recorded with optical systems because the motion is often constrained to a small, restricted area to avoid marker occlusion.

Despite the timing and amplitude variations, humans can easily recognize and interpret different executions of the same hand movement. Research on human gestures shows that the hand movement can be segmented into distinct phases using objective measures [28]. Specifically, *phase boundaries in hand movement*
are marked by an abrupt change of direction with a discontinuity in velocity profile before and after the direction change. Exploiting this criteria, we propose a novel distance metric that assesses the signs of the first and second derivatives of motion trajectories over time in order to detect phase boundaries.

To find correct correspondence between hand and full-body motions, we use our metric for hand motion segmentation jointly with dynamic time warping (DTW). Our use of DTW for such a task is novel because, although DTW has been introduced to the graphics community previously for temporal alignment, the extreme amplitude differences seen between performances in our case have not been addressed in previous work.

We employ the DTW algorithm at two levels of refinement. First, we use it at the coarse level to identify phase similarity. By aligning phases of motion our method overcomes gross amplitude and timing differences. Next, our algorithm performs a second DTW pass within each matched phase in order to fine-tune timing correspondence. We extended the basic DTW algorithm to allow for user input in order to guide the splicing process.

6.1 Matching Algorithm

Our technique employs dynamic time warping (DTW) to generate frame correspondences between motions with timing differences. Our algorithm works in three stages: first, using time warping, we match movement phases based on the velocity and acceleration profile. Next, we modify the resulting match by adjust-
ing the alignment of the frames within the phases, again with DTW. Finally, we create the combined hand and body animation, smoothing the resulting motion where necessary.

To match our source motions, we apply DTW with distance functions computed from four marker positions (two on the wrist, one on the hand, and one on the forearm). We found that this set was typically easy enough to include in both full-body and hand motion data during our captures, however the distance function can easily be adapted for use with different markers or with joint angle data.

**Dynamic time warping:** For detailed description of the time warping technique see [25]. Briefly, given the sequences \( p = \{p_1, p_2, \ldots, p_n\} \) and \( q = \{q_1, q_2, \ldots, q_m\} \), the DTW algorithm computes a cumulative distance matrix \( \Gamma = \{\gamma_{i,j}\}_{1,1}^{n,m} \) using a recurrence equation:

\[
\gamma_{i,j} = R(i, j, \gamma_{i-1,j-1}, \gamma_{i-1,j}, \gamma_{i,j-1}) \quad (6.1)
\]

where

\[
R(i, j, v_1, v_2, v_3) = D(p_i, q_j) + \min \left[ \begin{array}{c} v_1 \\ v_2 \\ v_3 \end{array} \right], \quad (6.2)
\]

and \( D(p_i, q_j) \) is the distance between points \( p_i \) and \( q_j \). \( \gamma_{i,j} \) represents the the minimal cost of a monotonic path between points \((0,0)\) and \((i,j)\). This path defines the correspondences between the frames of two motions, where each frame of one motion is matched to one or more frames of the other.

In order to limit the number of frames that can be matched to one frame
and to reduce the computation cost, the search for an optimal path is typically limited to a restricted area of the matrix, called the band (see Figure 6.3 a)). To incorporate the band constraint, we compute a new matrix $\Gamma'$ as follows:

$$
\gamma'_{i,j} = \begin{cases} 
R(i, j, \gamma'_{i-1,j-1}, \gamma'_{i-1,j}, \gamma'_{i,j-1}) & \text{if } j \geq \frac{m}{n} \cdot i - \Delta, \\
\gamma'_{i,j-1} & \text{if } j < \frac{m}{n} \cdot i - \Delta, \\
+\infty & \text{otherwise}
\end{cases}
$$

(6.3)

where $\Delta = W \cdot \sqrt{\frac{n^2 + m^2}{n}}$ and $W$ denotes the width of the band.

We extend the DTW algorithm to add functionality for user input; a user can specify which pairs of frames in the two motions should be matched and the algorithm will adapt the search band to the user’s specifications (see Figure 6.3 b)). A user can choose if frames need to be matched exactly or with some tolerance; in the latter case the search band will also include neighboring frames. We need to further modify the Equation 6.3 to accommodate user specified constraints, which are represented as a set of matching points $(x, y)^i_{k=1}$, where $x_i$ and $y_i$ are frame indexes in $p$ and $q$ respectively, $x_0 = 0 < x_1 < x_2 < \ldots < x_l < x_{l+1} = n$ and $y_0 = 0 < y_1 < y_2 < \ldots < y_l < y_{l+1} = m$. First we need to modify the band shape, so it includes the matching points:

$$
\gamma''_{i,j} = \begin{cases} 
R(i, j, \gamma''_{i-1,j-1}, \gamma''_{i-1,j}, \gamma''_{i,j-1}) & \text{if } j_k \geq \frac{|y_k|}{|x_k|} \cdot i_k - \Delta_k, \\
\gamma''_{i,j-1} & \text{if } j_k < \frac{|y_k|}{|x_k|} \cdot i_k + \Delta_k, \\
+\infty & \text{otherwise}
\end{cases}
$$

(6.4)

where $k$ is an index such that $x_{k-1} < i \leq x_k$, while $i_k = i - x_{k-1}$, $j_k = j - y_{k-1}$,
\[ |x_k| = x_k - x_{k-1}, \quad |y_k| = y_k - y_{k-1} \quad \text{and} \]
\[ \Delta_k = W \cdot \sqrt{\frac{|x_k|^2 + |y_k|^2}{|x_k|}}. \]

Next, we limit the band area so that the minimal path is constrained to pass in the neighborhood of the points \((x_k, y_k)\), within the user-specified tolerance \(T\). We compute the final matrix \(\Gamma^*\) as follows:

\[
\gamma_{i,j}^* = \begin{cases} 
R(i, j, \gamma_{i-1,j-1}^*, \gamma_{i-1,j}^*, \gamma_{i,j-1}^*) & \text{if } lb \leq j_k \leq ub \\
+\infty & \text{otherwise} 
\end{cases} \tag{6.5}
\]

where \(lb\) and \(ub\) are defined as:

\[
lb = \max\{1, y_k - T, \frac{|y_k|}{|x_k|} \cdot i_k - \Delta_k\} \\
ub = \min\{m, y_k + T, \frac{|y_k|}{|x_k|} \cdot i_k + \Delta_k\}.
\]

The above formula can be easily transformed into an efficient scheme for computation of matrix \(\Gamma^*\) and finding optimal alignment. Algorithm 1 shows a possible implementation.

### 6.2 Distance Metrics for Phase Alignment

**Phase Matching:** Studies of human gesticulation [24, 28, 44] show that human gestures can be segmented into a sequence of discrete phases of different types, based on velocity and acceleration profile (see Table 6.1). For example, the "counting footsteps" motion, (Figure 6.9) contains a repeated sequence of P,H,R phases, while the gesture for "Go down the driveway" in the "directions" sequence (Figure 6.7) can be decomposed into phases P,S,H,R.
Figure 6.3: Dynamic time warping for motions P and Q. a) We compute the distance matrix for pairs of frames from P and Q. We limit our search to a portion of distance matrix called a band (diagonal region). The darker areas on the band denote lower cost. The bright path denotes the optimal alignment between frames that minimizes the total cost. b) Our technique allows for user input by modifying the band. A user can specify pairs of matching frames either exactly (band limited to one point) or with some tolerance (band limited to a narrow area).

Figure 6.4: In the first stage of our algorithm, we align movement phases. For motions with significant amplitude differences, our distance function (center), which compares the signs of the first two derivatives over time, produces better results than a function using the values of the derivatives (right).

To match the corresponding phases in the two motions we use a distance function that evaluates the signs of the first and second derivatives over time for the two motion trajectories. The change in signs of derivatives reveals the changes in direction and velocity discontinuities that separate each hand motion phase. We choose to compare signs of derivatives rather than derivative values.
Algorithm 1 Compute matrix $\Gamma^*$

initialize all elements of $\Gamma^*$ to $+\infty$

for all $k$ such that $0 < k \leq l + 1$ do

for all $i$ such that $x_{k-1} < i \leq x_k$ do

$|x_k| = x_k - x_{k-1}, \quad |y_k| = y_k - y_{k-1}$

$\Delta = W \cdot \sqrt{|x_k|^2 + |y_k|^2} / |x_k|$

$i_k = i - x_{k-1}$

$lb = \max \{ 1, y_{k-1} - T, \frac{|y_k|}{|x_k|} \cdot i_k - \Delta_k \}$

$ub = \min \{ m, y_k + T, \frac{|y_k|}{|x_k|} \cdot i_k + \Delta_k \}$

for all $j$ such that $lb \leq j - y_{k-1} \leq ub$ do

$\gamma^*_{i,j} = D(p_i, q_j) + \min \begin{bmatrix} \gamma^*_{i-1,j-1} \\ \gamma^*_{i-1,j} \\ \gamma^*_{i,j-1} \end{bmatrix}$

because, for motions with significant amplitude differences, the function based on derivative values produces non-uniform matching within phases, where many frames of one motion are aligned to a single frame of the other (see Figure 6.4).

To compute our function, we first subtract in each frame the shoulder position from the four hand markers in full-body animation. This isolates the movement of the arm and the hand from the transition of the full-body motion. For example, when the character is walking and gesticulating, subtracting the shoulder position will extract the hand gesticulation and remove the transition effect from the hand markers. Next, we apply the time warping algorithm with distance metric defined
<table>
<thead>
<tr>
<th>Phase Type</th>
<th>Description</th>
<th>( v ) and ( a ) profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stroke (S)</td>
<td>More force is exerted than in neighboring phases, indicated by acceleration</td>
<td>either ( a &gt; 0 ) or ( a &lt; 0 )</td>
</tr>
<tr>
<td></td>
<td>or deceleration.</td>
<td></td>
</tr>
<tr>
<td>Hold (H)</td>
<td>Hand held still.</td>
<td>( a = 0, v = 0 )</td>
</tr>
<tr>
<td>Preparation (P)</td>
<td>Non-stroke phase that departs from a resting position or joins two stroke phases.</td>
<td>( a = 0, v &gt; 0 )</td>
</tr>
<tr>
<td>Retraction (R)</td>
<td>Non-stroke phase that arrives at a resting position or switches into preparation (partial retraction).</td>
<td>( a = 0, v &lt; 0 )</td>
</tr>
</tbody>
</table>

Table 6.1: Types of motion phases and their acceleration (\( a \)) and velocity (\( v \)) profiles (adapted from [28]). Positive velocity denotes movements outwards from the body, negative - towards the body.

as:

\[
D(F_1, F_2) = \sum_{i=1}^{4} \| \text{sgn}(\mathbf{p}_i) - \text{sgn}(\mathbf{q}_i) \|^2 + \| \text{sgn}(\dot{\mathbf{p}}_i) - \text{sgn}(\dot{\mathbf{q}}_i) \|^2
\]

\[
q_i = R_y(\theta) \cdot p'_i,
\]

where \( \mathbf{p}_i = [x_i, y_i, z_i] \) and \( \mathbf{p}'_i = [x'_i, y'_i, z'_i] \) are the positions of the \( i \)’th marker in frames \( F_1 \) and \( F_2 \) respectively, \( \dot{\mathbf{p}}_i = [\dot{x}_i(t), \dot{y}_i(t), \dot{z}_i(t)]^T \) is the first derivative of marker positions over time and \( \ddot{\mathbf{p}}_i \) is the second derivative. The sign function applied to a vector takes the sign of each vector element: \( \text{sgn}([x_i, y_i, z_i]^T) = [\text{sgn}(x_i), \text{sgn}(y_i), \text{sgn}(z_i)]^T \).
Rotation about the vertical axis, $R_y(\theta)$, locally aligns two motion fragments in space and can be computed with formula introduced in [32], using small windows of neighboring frames around $F_1$ and $F_2$:

$$\theta = \arctan \frac{\sum_{j=1}^{n}(x_jz_j' - x'_jz_j) - \frac{1}{n}(\bar{x}\bar{z}' - \bar{x}'\bar{z})}{\sum_{j=1}^{n}(x_jx'_j + z_jz'_j) - \frac{1}{n}(\bar{x}\bar{x}' + \bar{z}\bar{z}')},$$

(6.7)

where barred terms are defined as $\bar{\alpha} = \sum_{j=1}^{n} \alpha_j$ and $n$ is the total number of marker positions in a window. In our experiments we used windows with 20 marker positions (4 markers per frame for 5 frames).

Alternatively, $\theta$ can be computed as an angle difference between the trunk orientations. While for the full-body sequence trunk orientation can be easily computed from the waist markers, in general it might not be available for the hand motion. However, if the hand data was recorded without significant trunk movement, like in our case, the trunk orientation is easy to obtain and constant throughout the motion. In practice, the method of computing $\theta$ from trunk angles is more robust but requires additional information. In our experiments, $\theta$ obtained from Equation 6.7 worked well for all motions except the Indian dancing, which involves a lot of variation and frequent orientation changes. For this motion we used the trunk angle method.

**Matching frames within phases:** The first stage of the algorithm aligns the corresponding phases of motions. Next, to refine the frame correspondence within the motion phases, we align the forearms of the hands in the two motions and minimize the angle differences between the palms of the hand in each frame (see Figure 6.6). Specifically, to compute the cost function between two frames
F_1 and F_2, we first compute the forearm vectors f and f' and hand vectors h and h' as:

\[

g f = p_3 - \frac{p_1 + p_2}{2} \quad f' = p'_3 - \frac{p'_1 + p'_2}{2}
\]
\[
gh = p_4 - \frac{p_1 + p_2}{2} \quad h' = p'_4 - \frac{p'_1 + p'_2}{2},
\]

(6.8)

where p_1, p_2, p_3, p_4 and p'_1, p'_2, p'_3, p'_4 are the two wrist markers, the forearm marker and the hand marker for the full-body and hand motion respectively.

Next we compute the rotation matrix R that aligns the two forearm vectors f and f' in space and compute the angle between two hand vectors:

\[
D_2(F_1, F_2) = \arccos\left(\frac{\mathbf{h} \cdot R \mathbf{h}'}{\|\mathbf{h}\| \|R \mathbf{h}'\|}\right).
\]

(6.9)

To adjust frame correspondence, we again use DTW along with the cost function D_2. Here we use a narrower band to preserve phase correspondences while allowing adjustments to frame alignment within the phase. In our experiments the time warping band width was equal to 30\% and 10\% of the full-body motion length in the first and second stage respectively. If necessary, we recompute the user’s input so the frame numbers that need to be matched correspond to frames in the aligned motion.

**Merging and smoothing:** The frame correspondence resulting from the previous two steps goes through a final smoothing pass before rendering. Recall, the time warping algorithm produces matching sequences which may align a frame from one motion to one or more frames from the other motion. For the final animation, whenever a single frame from the full-body motion is matched to multiple frames from the hand sequence, we choose the median position of each marker
Figure 6.5: After aligning phases of motion we refine frame correspondences within each phase. a) Sequences of full-body motion (P) and hand motion (Q) belong to the same phase. b) Graphs show wrist marker positions and hand angles for sequences P and Q. c) Because all frames within a given phase have the same velocity and acceleration profile, during the first stage of our algorithm their alignment is arbitrary. d) Second stage of the algorithm refines frame correspondences based on angle between forearm and palm of a hand.

from the frames and use it in the matched sequence. Because this frame averaging process can produce small discontinuities in the hand motion, as a final step, we blend discontinuities over a window of frames to create a smooth animation.
Figure 6.6: $\mathbf{D}_2$ denotes the angle between the hands in two motions. First we compute the forearm and hand vectors $\mathbf{f}$ and $\mathbf{h}$ for full-body motion (a) and $\mathbf{f}'$ and $\mathbf{h}'$ for hand motion (b). Next we align the forearm vectors $\mathbf{f}$ and $\mathbf{f}'$ and compute the distance function as the angle $\alpha$ between hand vectors $\mathbf{h}$ and $\mathbf{h}'$ after the alignment (c).

Figure 6.7: Example of directive gestures.

6.3 Results

We test our algorithm on various motions with complex hand gesticulation, where synchronization between the hand and body motions is crucial. We show sample frames of the resulting animations in Figures 6.1, 6.7 and 6.9. Examples include: a series of counting animations, where the hand keeps count of the foot steps, an animation of a person giving a complex sequence of directions, Indian dancing motion where elaborate hand gestures (mudras) are used to tell a story and a
more light-hearted animation of a charade for ‘peeling a banana’. Each sequence contains multiple examples of aligned gesture phases (see Table 6.2).

All motions were captured with Vicon optical capture equipment (www.vicon.com). While the full-body animation was recorded without space restrictions, the hand motion was captured in a constrained area, roughly a 20” cube, where the arm’s movement was severely limited. In all test scenarios our algorithm produced good results without the need for additional user input. For Indian dancing motion we specified 3 matching points to further increase the accuracy of the alignment. In the obtained motions the hand gestures are correctly synchronized with corresponding full-body movement. In the accompanying video we compare the resulting animations to the original human performer’s moves during the motion capture recording. We also show an example of how user-specified constraints can be employed to achieve special effects, such as counting every left step.

<table>
<thead>
<tr>
<th>Motion Sequence</th>
<th>Length (sec)</th>
<th>Number of Gestures</th>
<th>Number of Phases</th>
<th>Matching Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directions</td>
<td>18</td>
<td>6</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>Counting</td>
<td>16</td>
<td>8</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>Dance</td>
<td>18</td>
<td>34</td>
<td>52</td>
<td>3</td>
</tr>
<tr>
<td>Charade</td>
<td>9</td>
<td>4</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6.2: In our experiments we used 4 motion sequences with varying number of gestures, from a simple charade to a complex Indian dance. All motions were aligned without user input, except for the Indian dance, where we specified 3 matching points. Numbers of gestures and phases are reported collectively for both hands.
### Table 6.3: Running time of our algorithm using Matlab implementation.

<table>
<thead>
<tr>
<th>Motion Sequence</th>
<th>Number of Frames</th>
<th>Matching Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directions</td>
<td>569</td>
<td>10.95</td>
</tr>
<tr>
<td>Counting</td>
<td>503</td>
<td>8.72</td>
</tr>
<tr>
<td>Dance</td>
<td>560</td>
<td>10.81</td>
</tr>
<tr>
<td>Charade</td>
<td>280</td>
<td>2.75</td>
</tr>
</tbody>
</table>

Our straightforward but effective technique for composing hand and full body motion exploits characteristics of human gestures in order to properly align motions with large amplitude differences that were captured separately, and at different resolutions. Thanks to its simple design, the proposed algorithm is efficient (see Table 6.3) and easy to implement. We have shown the power of our technique on motions with complex gesticulation and obtained correctly synchronized, natural-looking animation results.

There are several classes of motions for which our solution will likely fail. While our algorithm found appropriate matches for many free space gestures,
Figure 6.9: Counting example shows good synchronization of hand and body movement.
Figure 6.10: Indian dance: side by side comparison of the original motion and our results.
gesticulation which involves specific spatial cues would require special consideration. For example, generating a motion with a constraint which ensures that a finger makes contact when touching the tip of one’s nose lies outside the scope of the proposed solution. Additionally, motions which rely heavily on dynamics may not be amenable to the splicing described here. An example of such a motion may include shaking hands energetically, where the movement of the arms directly dictates the motion of the hands. Finally, animations generated using our approach require that the motion be reasonably performed in a limited workspace for the hand capture. This is not the case in certain motions, such as performing a gymnastic routine.
CHAPTER 7

Conclusion

In this work we have presented three novel motion synthesis techniques based on different strategies of composing motion data. By interjecting motion capture animation with dynamic simulation we created realistic human responses to an unexpected impact (Chapter 4). We have shown how to create complex acrobatic motions from simpler animations by composing fragments of motion capture data and how to adjust these motions after user modifications to assure conservation of momentum (Chapter 5). Finally, we layered motions in time to obtain a multi-resolution effect: detailed hand gesticulation with full-body movement (Chapter 6).

The presented motion composition methods create novel animations from existing motion capture clips while maintaining the natural look, richness of detail, and physical validity of the original motions. Our methods are useful as they generate motions that cannot be replicated with motion capture recordings, due to high level of motion difficulty, safety concerns or technical limitations.

The proposed techniques are specialized - they create natural-looking and physically valid motions in an efficient manner by taking advantage of proper-
ties particular to a given motion type. This specialization has downsides - our methods cannot be applied across the range of all possible motions. Due to the immense complexity of human movement, we try to solve the problem of human motion synthesis step by step, one motion type at a time. With the proposed techniques we have expanded the space of human motions that can be generated with automated synthesis. We believe that experiences and learnings acquired in the process will allow us to develop general techniques in the future.

Perhaps, thanks to such techniques, one day it will be possible to create virtual actors that are indistinguishable from human ones. Simulated characters that acted and moved naturally in a wide variety of situations would deeply affect the movie industry. They could be used to perform dangerous stunts, thus avoiding the risk of injury to a human actor while still maintaining the actor’s individual style of movement. Long gone movie stars could be brought back to life by virtual actors who looked and moved just like their human counterparts. In video games, life-like characters that reacted naturally in different scenarios would significantly improve games’ interactivity and players’ satisfaction. Hopefully, these advances, once developed, will bring us more enjoyment and an enriched experience of viewing human motion on the screen.
APPENDIX A

Dynamic Response for Motion Capture

Animation: Implementation Details

In our search function we use a similarity function which compares windows of frames. Recall that the distance $D$ between windows $W_1 = \{f_{1i}\}_{i=s}^e$ and $W_2 = \{f_{2i}\}_{i=s}^e$ is defined in Equation 4.1 as:

$$D(W_1, W_2) = \sum_{i=s}^{e} w_i \left( \sum_{b=1}^{n} w_{pb} ||p_b(f_{1i}) - p_b(f_{2i})|| + w_{\theta b} ||\theta_b(f_{1i}) - \theta_b(f_{2i})|| \right),$$

where $w_i$ is the window weight and the weights $w_{pb}$ and $w_{\theta b}$ scale the linear and angular distances for each body $b$.

We compute $w_i$ as:

$$w_i = -\left( \frac{i - s}{e - s} \right)^2 + 1,$$

where $s$ and $e$ are respectively the start and end frames of the window. Weights $w_{pb}$ and $w_{\theta b}$ are provided in Table A.1. The constant $T$ used in the computation of unique frames (Equation 4.2) is defined in our system as $T = 0.5$.

Once the search function finds the best motion to transition to, the transition between the motions is created by the torque controller (see Section 4.2). We use the following constants in the controller formulation: spring constant $k_p = 1.0,$
positional damping constant $k_v = 2.0$ and rotational damping constant $k_\omega = 0.05$.

Timing is a difficult aspect of the problem we address. Because we want to minimize the time spent in simulation, we limit our motion search to a short time window just after the initial impact from 0.1 to 1.0 s. During operation, the system automatically shrinks this window by increasing the start time for the search (up to a limit of 0.25 s) based on contact to ensure that a match is not found in the time before contact is over (as the dynamic effects have not yet fully changed the state of the character.) The upper limit manages cases with very long or sustained contact. We found that good contact required running the collisions at a simulation time step of about 0.0005 s and to speed up the system, we built in an automatic rewind that looks for the first contact at 30 fps and then turns back time to the last motion capture frame, computes the simulation state and begins the simulation at the reduced time step. Ground friction was another sensitive parameter in our system and we chose to set it in the first naive simulation calculation to 1.0 (a rough surface) but dropped the friction to 0.5 for the second pass to allow foot slippage.
<table>
<thead>
<tr>
<th>Body Part</th>
<th>Mass</th>
<th>Linear weight $w_{j_b}$</th>
<th>Angular weight $w_{\theta_b}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Head</td>
<td>3.66</td>
<td>0.1</td>
<td>0.01</td>
</tr>
<tr>
<td>Left Upper Arm</td>
<td>2.16</td>
<td>0.5</td>
<td>0.05</td>
</tr>
<tr>
<td>Right Upper Arm</td>
<td>2.16</td>
<td>0.5</td>
<td>0.05</td>
</tr>
<tr>
<td>Left Forearm</td>
<td>1.37</td>
<td>0.2</td>
<td>0.02</td>
</tr>
<tr>
<td>Right Forearm</td>
<td>1.37</td>
<td>0.2</td>
<td>0.02</td>
</tr>
<tr>
<td>Chest</td>
<td>12.83</td>
<td>1.0</td>
<td>0.10</td>
</tr>
<tr>
<td>Stomach</td>
<td>8.61</td>
<td>1.0</td>
<td>0.10</td>
</tr>
<tr>
<td>Pelvis</td>
<td>8.70</td>
<td>1.0</td>
<td>0.10</td>
</tr>
<tr>
<td>Left Thigh</td>
<td>5.17</td>
<td>0.5</td>
<td>0.05</td>
</tr>
<tr>
<td>Right Thigh</td>
<td>5.17</td>
<td>0.5</td>
<td>0.05</td>
</tr>
<tr>
<td>Left Shank</td>
<td>2.14</td>
<td>0.2</td>
<td>0.02</td>
</tr>
<tr>
<td>Right Shank</td>
<td>2.14</td>
<td>0.2</td>
<td>0.02</td>
</tr>
<tr>
<td>Left Foot</td>
<td>1.88</td>
<td>0.1 / 1.0</td>
<td>0.01 / 0.10</td>
</tr>
<tr>
<td>Right Foot</td>
<td>1.88</td>
<td>0.1 / 1.0</td>
<td>0.01 / 0.10</td>
</tr>
<tr>
<td>Left Hand</td>
<td>1.48</td>
<td>0.1</td>
<td>0.01</td>
</tr>
<tr>
<td>Right Hand</td>
<td>1.48</td>
<td>0.1</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Table A.1: Weights and masses used in our implementation. Feet weights are higher for supporting limbs.
APPENDIX B

Motion Composition for Acrobatics: Computing Retiming Factor $k^*$

The retiming process affects the linear momentum in two directions. First, it shortens the take-off and landing and increases momentum in these phases (assuming that the retiming factor is positive). Second, it also shortens the duration of the flight and decreases the necessary momentum build-up. Due to these two consequences of motion retiming, there exists a single retiming factor $k^*$ which assures continuity of linear and angular momentum curves during all jump phases.

To compute $k^*$ we first observe that retiming scales the take off velocity linearly:

$$v_0^* = k^* v_0,$$

where $v_0$ is the take-off velocity along the $y$ axis in the original motion. Next, we note that the flight duration is also linearly scaled, by the factor $1/k^*$:

$$T^* = T/k^*, \quad (B.2)$$

where $T$ is the duration of the flight phase before retiming. The motion of the
character’s center of mass along the $y$ axis is described by a parabola:

$$y^*(t) = -\frac{1}{2}gt^2 + v_0^* t + y_{\text{start}},$$  \hspace{1cm} (B.3)

with the end condition: $y^*(T^*) = y_{\text{end}}$, where $y_{\text{start}}$ and $y_{\text{end}}$ are the take-off and landing levels and $g$ is the gravity coefficient. Substituting (B.1) and (B.2) in the parabola equation (B.3) we obtain:

$$-\frac{1}{2}g \left( \frac{T}{k^*} \right)^2 + v_0 T + y_{\text{start}} = y_{\text{end}}.$$  \hspace{1cm} (B.4)

After solving for $k^*$ we receive:

$$k^* = \sqrt{\frac{1}{2}gT \left( \frac{T}{v_0} \right) - \Delta y T},$$  \hspace{1cm} (B.5)

where $\Delta y = y_{\text{end}} - y_{\text{start}}$.

**Height changes.** We can use the above formulas to adjust the take-off and landing levels. In the retiming step we substitute $\Delta_{\text{new}} y = y_{\text{end}}^{\text{new}} - y_{\text{start}}^{\text{new}}$ when computing $k^*$, where $y_{\text{start}}^{\text{new}}$ and $y_{\text{end}}^{\text{new}}$ are the new take-off and landing levels respectively. Next, in the repositioning step we use $y_{\text{start}}^{\text{new}}$ to compute the parabola equation. We can even eliminate the need for retiming by appropriately changing the take-off and landing levels to assure that $\Delta_{\text{new}} y = \frac{1}{2}gT^2 + v_0 T$ and therefore $k^* = 1$. There are however limits to the height changes if $\Delta_{\text{new}} y > 0$. When computing $k^*$ we need to ensure that the radicant in (B.5) is positive and therefore $\Delta_{\text{new}} y$ must be less than $v_0 T$.  

99
APPENDIX C

Motion Composition for Acrobatics:
Implementation Details

Table C.1 presents body parts’ densities and width to depth ratios that we used in our mass optimization algorithm (Section 5.3.2). The initial mass distribution, which was a starting point of our optimization, was computed from the typical body parts dimensions and densities (obtained from [11]) scaled to match our skeleton. Note, that since the skeleton we used (provided with motion data) was not symmetric, the initial weight distribution between the left and right limbs is not fully symmetric either. However, during the optimization process we assumed symmetry to reduce the number of optimized variables. Therefore in the resulting distribution, the body parts of the left limbs have equal weights to their right limbs’ counterparts.
<table>
<thead>
<tr>
<th>Body Part</th>
<th>Density</th>
<th>Width/Depth Ratio</th>
<th>Initial Mass Distribution %</th>
<th>Optimized Mass Distribution %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root</td>
<td>1.029</td>
<td>1.40</td>
<td>7.4</td>
<td>6.7</td>
</tr>
<tr>
<td>Left Femur</td>
<td>1.040</td>
<td>0.86</td>
<td>10.2</td>
<td>11.8</td>
</tr>
<tr>
<td>Left Tibia</td>
<td>1.079</td>
<td>0.89</td>
<td>7.9</td>
<td>5.3</td>
</tr>
<tr>
<td>Left Foot</td>
<td>1.066</td>
<td>2.00</td>
<td>0.3</td>
<td>0.0</td>
</tr>
<tr>
<td>Left Toes</td>
<td>1.066</td>
<td>3.00</td>
<td>0.1</td>
<td>0.0</td>
</tr>
<tr>
<td>Right Femur</td>
<td>1.040</td>
<td>0.86</td>
<td>9.8</td>
<td>11.8</td>
</tr>
<tr>
<td>Right Tibia</td>
<td>1.079</td>
<td>0.89</td>
<td>7.6</td>
<td>5.3</td>
</tr>
<tr>
<td>Right Foot</td>
<td>1.066</td>
<td>2.00</td>
<td>0.3</td>
<td>0.0</td>
</tr>
<tr>
<td>Right Toes</td>
<td>1.066</td>
<td>3.00</td>
<td>0.1</td>
<td>0.0</td>
</tr>
<tr>
<td>Lower Back</td>
<td>1.029</td>
<td>1.20</td>
<td>5.8</td>
<td>5.8</td>
</tr>
<tr>
<td>Upper Back</td>
<td>1.029</td>
<td>1.08</td>
<td>7.4</td>
<td>7.4</td>
</tr>
<tr>
<td>Thorax</td>
<td>1.009</td>
<td>1.00</td>
<td>8.9</td>
<td>8.8</td>
</tr>
<tr>
<td>Lower Neck</td>
<td>1.009</td>
<td>1.00</td>
<td>1.0</td>
<td>0.9</td>
</tr>
<tr>
<td>Upper Neck</td>
<td>1.009</td>
<td>1.00</td>
<td>0.6</td>
<td>0.8</td>
</tr>
<tr>
<td>Head</td>
<td>1.171</td>
<td>1.00</td>
<td>5.9</td>
<td>7.4</td>
</tr>
<tr>
<td>Left Clavicle</td>
<td>1.009</td>
<td>1.17</td>
<td>5.9</td>
<td>6.5</td>
</tr>
<tr>
<td>Left Humerus</td>
<td>1.068</td>
<td>1.00</td>
<td>3.6</td>
<td>3.9</td>
</tr>
<tr>
<td>Left Radius</td>
<td>1.102</td>
<td>1.17</td>
<td>2.6</td>
<td>3.1</td>
</tr>
<tr>
<td>Left Wrist</td>
<td>1.102</td>
<td>1.00</td>
<td>0.7</td>
<td>0.5</td>
</tr>
<tr>
<td>Left Hand</td>
<td>1.070</td>
<td>4.00</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Body Part</th>
<th>Density</th>
<th>Width/Depth Ratio</th>
<th>Initial Mass Distribution %</th>
<th>Optimized Mass Distribution %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left Fingers</td>
<td>1.070</td>
<td>8.00</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Left Thumb</td>
<td>1.070</td>
<td>2.00</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Right Clavicle</td>
<td>1.009</td>
<td>1.17</td>
<td>7.5</td>
<td>6.5</td>
</tr>
<tr>
<td>Right Humerus</td>
<td>1.068</td>
<td>1.00</td>
<td>3.5</td>
<td>3.9</td>
</tr>
<tr>
<td>Right Radius</td>
<td>1.102</td>
<td>1.17</td>
<td>2.3</td>
<td>3.1</td>
</tr>
<tr>
<td>Right Wrist</td>
<td>1.102</td>
<td>1.00</td>
<td>0.6</td>
<td>0.5</td>
</tr>
<tr>
<td>Right Hand</td>
<td>1.070</td>
<td>4.00</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Right Fingers</td>
<td>1.070</td>
<td>8.00</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Right Thumb</td>
<td>1.070</td>
<td>2.00</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table C.1: Weights and masses used in our implementation. Feet weights are higher for supporting limbs.
APPENDIX D

Automatic Splicing for Hand and Body Animations: Data Acquisition

Figure D.1: Our camera setup for hand capture.
We captured our motions with Vicon optical capture equipment (www.vicon.com). The full-body animations were recorded in a large studio, without space restrictions (see Figures D.3 and 6.10) with a standard camera setup. The hand motion was captured in a constrained area, roughly a 20” cube, where the arms movement was severely limited (see Figure D.4). Additionally, for the hand recording we positioned the cameras close together (Figure D.1) to increase capture accuracy and reduce artifacts such as occlusion and marker swapping.

![Hand markers used in detailed hand capture. Four markers: VLA, VWRA, VWRB and VH1 were common for both hand and full body capture.](image)

Full-body motions were recorded using 41 markers. For hand detail we employed 22 optical markers (see Figure D.2). For the alignment purposes we used four common markers on each hand: VLA, VWRA, VWRB and VH1, which
Figure D.3: Capture of full body motion. Dancer moves in a large room.

were attached to actors’ hands during both full-body and hand recordings.
Figure D.4: Capture of detailed hand motion. Dancer movement is restricted to a small area.
References

[1] CMU graphics lab motion capture database. mocap.cs.cmu.edu.


