Reducing the search space for physically realistic human motion synthesis

Alla Safonova

CMU-CS-06-157

September 2006

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Jessica K. Hodgins, Carnegie Mellon University (chair)
Nancy S. Pollard, Carnegie Mellon University
Christopher G. Atkeson, Carnegie Mellon University
Jovan Popovic, Massachusetts Institute of Technology

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the National Science Foundation under grant nos. CNS-0196217, IIS-0205224, and IIS-0326322 and supported by generous software donations from Alias/Wavefront and Autodesk.

Keywords: computer graphics, animation, optimization, motion capture, human motion, motion graphs, interpolation

Abstract

Being able to animate a human character in a way that does not require the expertise of a professional animator can be useful in many different applications: children would be able to animate stories, witnesses would be able to visually describe an accident to lawyers, football fans would be able to re-produce their favorite football plays. These applications and others like them have motivated this thesis: it focuses on the development of methods that given a sketch of the desired motion, can interactively create a physically realistic motion that matches that sketch.

This problem is very hard to solve in part because human-like characters are high-dimensional and therefore the space of their motions also appears to be high-dimensional. However, the high dimensionality of the problem is an artifact of the problem representation because most dynamic human motions are intrinsically low-dimensional with legs and arms operating in a coordinated way. For example, six to eight dimensions are enough to represent a human jump that looks quite similar to the original high-dimensional version.

In this thesis, we experiment with two different approaches that use this observation to build a compact (reduced-space) representation of the motion based on available motion capture data. In the first part of the thesis we build a continuous low-dimensional representation of the desired motion. By confining the solution to a smaller search space, we are able to synthesize physically realistic motion for a human character that matches a rough sketch provided by the user.

In the second part of the thesis, we build a discrete reduced-space representation of the desired motion. This representation can be viewed as a combination of the motion graph and interpolation techniques. The final motion is an interpolation of k time-scaled paths through the motion graph. We assess its physical correctness using our analysis of the physical correctness of interpolated motions. The optimization in the discrete space supports interactive frame rates and allows for the synthesis of less dynamic motions and motions that are sequences of different behaviors. We also demonstrate how to search a motion graph of a reasonable size for an optimal (or nearly-optimal) solution.

For both, the continuous and discrete optimization approaches the synthesized motion is likely to contain natural coordination patterns because the solution is constrained to a much smaller search space computed based on motion capture data. This objective is difficult to describe mathematically and is often not achieved when optimizing in the full search space. In the last chapter of the thesis, we compare the two approaches and provide some intuition as to when one is more applicable than the another.

Acknowledgments

First, I would like to thank my advisor Jessica Hodgins. I sincerely appreciate all the help and support she has provided to me during my Ph.D. studies. I would also like to thank the other members of my thesis committee, Nancy Pollard, Chris Atkeson and Jovan Popovic, for their guidance and criticism that helped to shape up my research. In addition, I would also like to thank Jarek Rossignac with whom I have really enjoyed working while at Georgia Tech.

I would also like to thank a number of my colleagues here at CMU for making the experience more fun. This list includes Kiran Bhat, Preethi Bhat, Paul Reitsma, Christopher Twigg, Liu Ren, Jin-Xiang Chai, Jehee Lee, Jernej Barbic, James Hays, Steve Wen-Chieh Lin, Steven Osman, Jeff Smith and Thomas Kang. Special thanks to Moshe Mahler for making all my animations look good.

Finally, and most importantly, I would like to thank my family. I am very grateful to my husband, Max, for his moral support during these years and to my beautiful daughter, Sasha, for making the time outside of CMU a lot of fun. I would also like to thank my parents and grandparents for their love and help.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Intuitive interfaces for animating human characters would allow children to tell stories, sport fans to visually describe a football play or witnesses to describe an accident to lawyers. In games, intuitive interfaces would allow a motion of the character through a complex environment to be specified by simply sketching a path. These applications and others like them motivate our work. We would like to enable naive users to create animations of complex characters, such as humans, in an easy and intuitive way. We therefore focus on interfaces that require users to provide only a small amount of information to describe the desired motion. That information should be specified in an intuitive way that does not require learning complicated animation systems such as those used by professional animators. For example, a human animation might be created by posing an articulated doll or sketching a path of the character (Figure 1.1). The doll interface might allow children to use animation as an expressive tool for storytelling. The sketch interface might be useful for guiding characters in strategy games.

Given a sketch of the desired motion, our system should interactively create a motion that matches that sketch. Figures 1.2– 1.3 shows a few examples. Our techniques should satisfy the following three requirements:

- Synthesize a physically realistic and natural looking motion.
- Animate a character with a plausible number of degrees of freedom for a human.

- Match a rough sketch from the user.

In addition, we would also like our techniques to satisfy the following three properties:

- Support an interactive interface where the user refines the motion after viewing it. This requirement implies that the motion should be synthesized within a few minutes.
- Support less dynamic and more stylized motions.
- Support complex motions that involve more than one behavior (for example, a walk with few jumps).

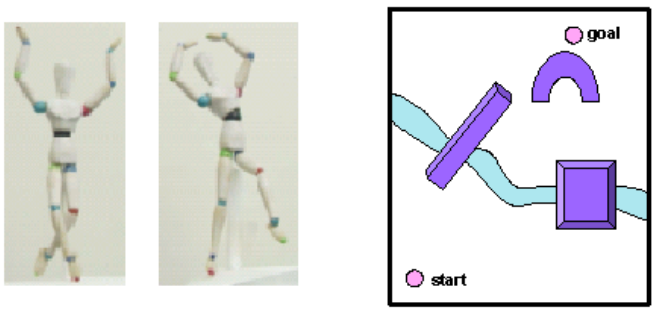

Figure 1.1: (a) The Doll Interface created at Princeton University. This interface allows novice users to easily pose a wooden doll in any configuration. Two cameras are then used to capture the 3D pose of the doll [1]; (b) Another intuitive interface would be to have the user specify the sketch of the path of the character through a complex environment or specify just the start and end of the path.

This problem is very hard to solve in part because it is very high dimensional. About 60 degrees of freedom are required to represent the pose of a humanlike character at each point in time (Figure 1.5). For an animation with T frames this results in $60T$ unknown variables each of which is a continuous variable defined on an interval of the real line.

As we discuss in this thesis, the high dimensionality of the problem is, however, an artifact of the problem representation. Most dynamic human motions are intrinsically

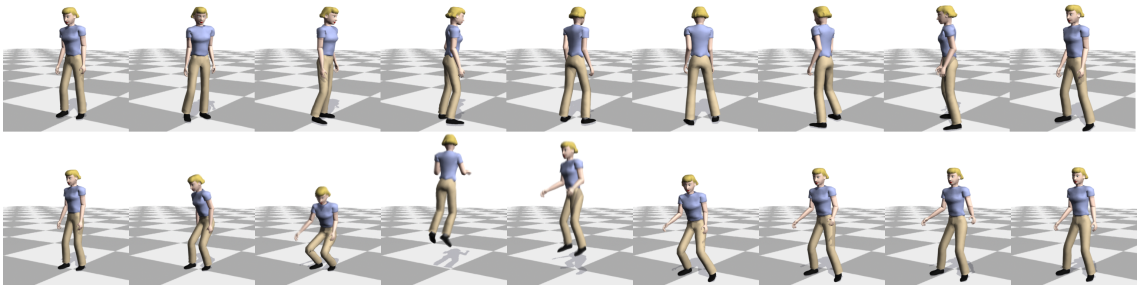

Figure 1.2: Synthesizing a vertical jump with a 360^o turn. (First row) Sketch of the desired motion created by linearly interpolating start and end poses provided by the user. (Second row) Synthesized, physically correct motion.

low dimensional with legs and arms operating in a coordinated way as we will show in Chapter 3. For example, six to eight dimensions are enough to represent a human jump that looks quite similar to the original high-dimensional version. In this thesis, we experiment with two different approaches that exploit this observation to build a compact (reduced-space) representation of the motion based on available motion capture data. In Chapter 3 we build a continuous low-dimensional representation of the desired motion. In Chapter 4 we build a discrete reduced-space representation of the desired motion. We outline these two approaches in the next two sections of this chapter. Throughout the rest of the thesis we will refer to the them as the "continuous optimization approach" and the "discrete optimization approach."

1.1 Continuous low-dimensional subspace

Continuous optimization [74] is a common technique for finding a motion when only a rough sketch is provided. The user specifies a set of constraints (such as pose) and an objective function. The optimization problem is then to minimize the objective function while satisfying user-specified and physics constraints (which preserve the physical validity of the motion). Optimization-based techniques rely on physical laws to constrain the search to the appropriate part of the state space.

Constrained optimization has been extensively explored as a technique for charac-

Figure 1.3: (Left image) User specified the sketch of the path the character needs to follow; (Right image) Synthesized motion.

ter animation. The optimization problem is usually represented in a continuous domain, where the variables (or unknowns) of the optimization can take on any values from intervals of the real line. For complex articulated characters such as humans, the optimization problem is hard to solve for three reasons:

1. A large number of unknowns are required to realistically represent human characters and as a result the search space is high dimensional.

2. A highly non-linear optimization function and constraints are generally required to represent the physics of the character resulting in a complex landscape for the optimization function and constraints. Most methods used to solve continuous optimization problems make use of first and second derivatives of the objective function and constraints and are therefore highly sensitive to the landscape of the optimization function and the constraints.

3. Defining an objective function that reliably results in natural human motion for many different human behaviors is difficult, particularly for less dynamic behaviors where the physical constraints do not restrict the motion significantly.

We address these problems by confining the solution of the optimization problem to a

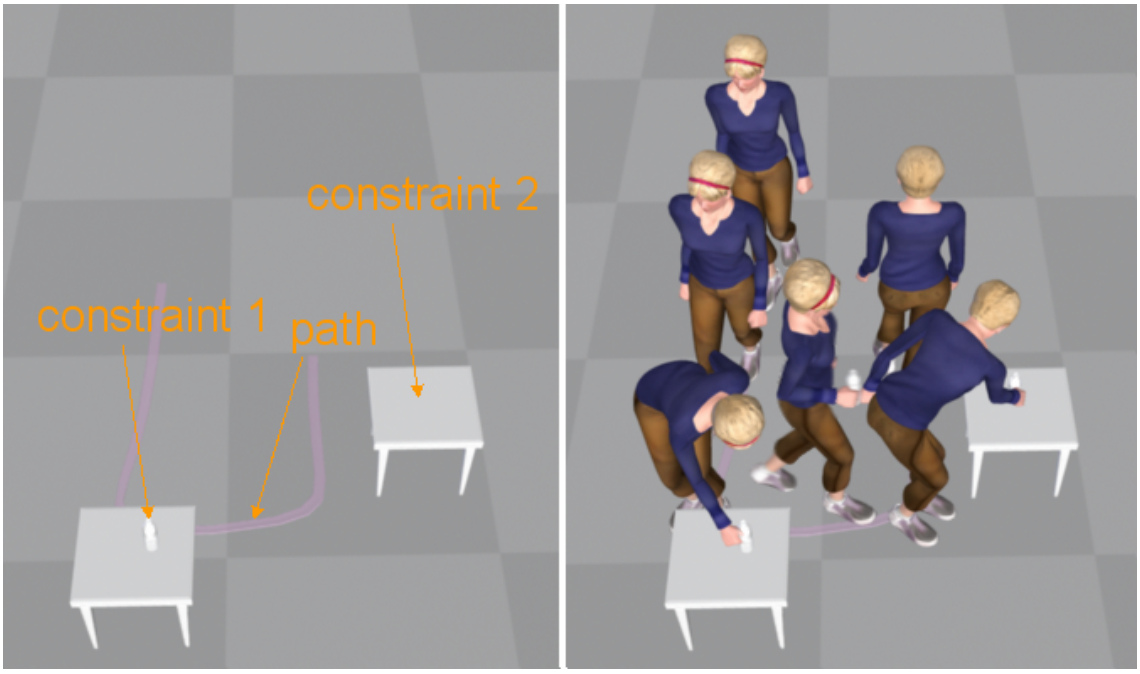

Figure 1.4: (Left image) User specified the sketch of the path the character needs to follow and two constraints—the character needs to pick up a bottle from the first table and put it on the second table; (Right image) Synthesized motion.

low-dimensional space that captures the type of behavior desired by the user. For example, if the user wants to generate a jumping motion, the subspace should be general enough to generate jumping motions of various heights and lengths. We find this space by running Principal Component Analysis (PCA) on a collection of poses from a few motions selected from an existing motion capture database that are similar to the behavior desired by the user. Each frame of the desired motion is then represented as a linear combination of six to ten basis vectors computed by PCA. Optimization is used to find the linear coefficients that relate these vectors and produce the desired motion. The optimization problem is solved in the space of a lower dimensionality, but the linear coefficients specify a physically valid motion for the full 60 DOF character.

By representing the problem in a low-dimensional space, we reduce the complexity of the optimization problem. As a result, we are able to generate motion for complex

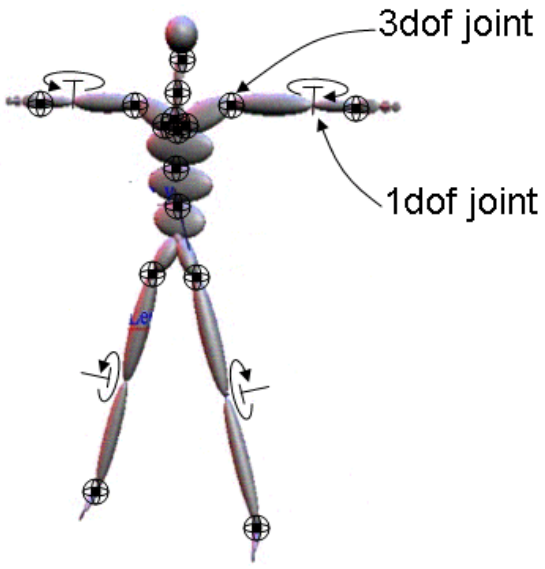

Figure 1.5: Degrees of freedom of the character.

characters, such as humans, with physics constraints imposed and using highly non-linear optimization functions, such as the sum of squared torques. The solution also contains natural coordination patterns because they are enforced by the low-dimensional space. This objective is difficult to describe mathematically and it is often not achieved when optimizing in the full-dimensional space.

Although optimization in the lower-dimensional space makes the problem tractable, constraints such as foot contact cannot always be satisfied exactly in this space. We solve this problem by using inverse kinematics to meet the constraints exactly while including a term in the optimization function that keeps the motion close to the low-dimensional basis. The optimizer then solves for a motion that is very close to the low-dimensional space, satisfies all the user-specified and physics constraints and minimizes such criteria as energy expenditure. We demonstrate the power of this approach through a number of different examples (Figure 1.6) and compare them to ground truth motion capture data.

All three of the properties we listed as required in the problem statement are satisfied

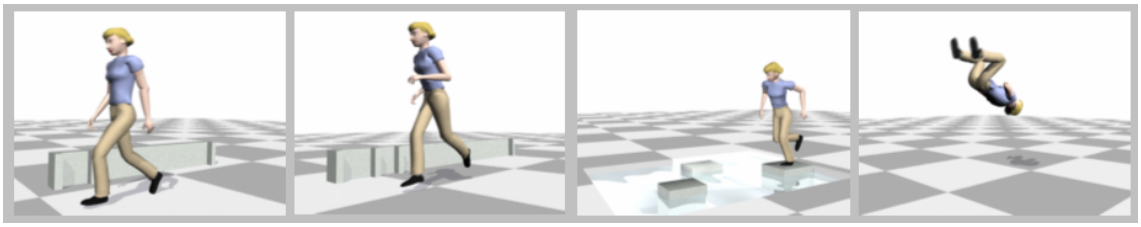

Figure 1.6: Synthesized motions: walking, running, jumping between stepping stones and a back flip.

with this approach: it generates a physically realistic motion for a human character and the motion matches the rough sketch provided by the user. However, not all the properties listed as desired are satisfied. The algorithm runs at interactive frame-rates only for very short motions. The running time of the algorithm is somewhere from three minutes to an hour depending primarily on the length of the motion and some other parameters. Because the motion is computed by minimizing a physics-based objective function (such as energy) this approach is most suitable for dynamic motions. The lower-dimensional space limits the solution to behaviors similar to the one that was used to compute it. Therefore motions of only a single behavior can be computed in each low-dimensional space.

1.2 Discrete low-dimensional subspace

Motion graphs are another common way to solve for a desired motion based only on a rough sketch [41, 6, 33, 38, 6, 7]. Given a database of human motions, a motion graph is constructed by finding natural transitions between different poses in the database. A motion can then be generated simply by traversing a *path* through the graph. Figure 1.7 gives an example of a simple motion graph constructed from two walking motions and also shows an example of a path through the motion graph. A *path* through the graph can be formally defined as an ordered sequence of poses, where any two consecutive poses are connected by an edge in the graph.

Discrete search techniques can be used to search a motion graph for a motion that satisfies user-specified constraints. Because the solution is constrained to a sequence of

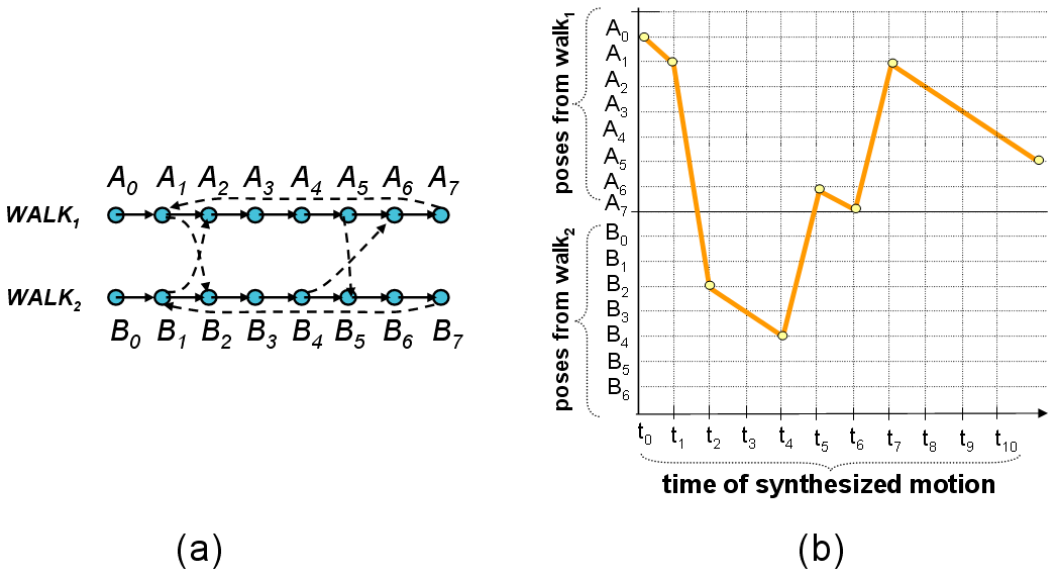

Figure 1.7: (a) Simple motion graph for two walking motions. States A_1 and B_1 are similar and therefore two transitions are added to the motion graph: a transition from state A_1 to state B_2 and a transition from B_1 to state A_2; Transitions are shown by dashed arrows. (b) The motion is generated by traversing the *path* through the motion graph shown in (a). The X axis represents the time of the synthesized motion and the Y axis represents an index of the frames of the motions in the database.

motion segments from the motion capture database, a motion graph provides a compact representation of a human motion. This representation, however, is quite restrictive. For example, it would be impossible to synthesize a motion for picking up a cup from a table that is 1.0 meter high if the database contains only motions for picking up a cup from tables that are 0.5 and 1.5 meters high.

To relax this restriction, we instead consider a more general discrete low-dimensional representation of the data. The new motion is an interpolation of two or more motions, where each motion is generated by traversing a path through the motion graph. More formally, each pose of the new motion, $P_{new}(t)$, is represented as an interpolation of k poses from the motion capture database:

$$P_{new}(t) = P_1(t)w_1 + P_2(t)w_2 + \ldots + P_k(t)w_k. \quad (1.1)$$

where, $P_i(t)$ are poses from the motion capture database and w_i are weights that interpolate these poses. Any two consecutive poses in each $P_i(t)$ must be connected by an edge in the motion graph. In other words, each $P_i(t)$ is a path through the motion graph. Interpolation weights can be constant or can be a function of time. In our implementation we want to synthesize motions of multiple behaviors and therefore we allow weights to change with time. We also allow for time-scaling of these paths which is often required to synchronize interpolated motions. This representation, therefore, can be viewed as a combination of motion graph and interpolation techniques. The final motion is an interpolation of k time-scaled paths through the motion graph. Figure 1.8 shows an example of two paths through the motion graph that are interpolated to create the desired motion.

Why is this a good representation?

- Because the new motion is computed by interpolating existing poses from the motion capture database, it is likely to be natural looking. Interpolation is a very simple and yet a very powerful technique for generating motions that are variations of motions in the database. Interpolation has been shown to produces surprisingly natural results. In Chapter 5 we analyze interpolated motion for physical correctness and show that interpolation produces motions that are close to physically correct in many cases. This makes interpolation a good technique for creating new human motions that are close to physically correct.

- The search space is greatly reduced from the full 60 dof search space. If we were to discretize each of the 60 degrees of freedom of the character into 100 values, we would have 100^{60} possible poses for this character. Many of these poses, however, are not natural. The motion capture database at Carnegie Mellon University [2] (which is considered quite big) contains only about $5 * 10^5$ poses (assuming that each motion is sampled at 30 frames per second) and even then many of these poses are quite similar despite an attempt to put a wide variety of human motions into the database. By representing a new motion as an interpolation of existing poses, we

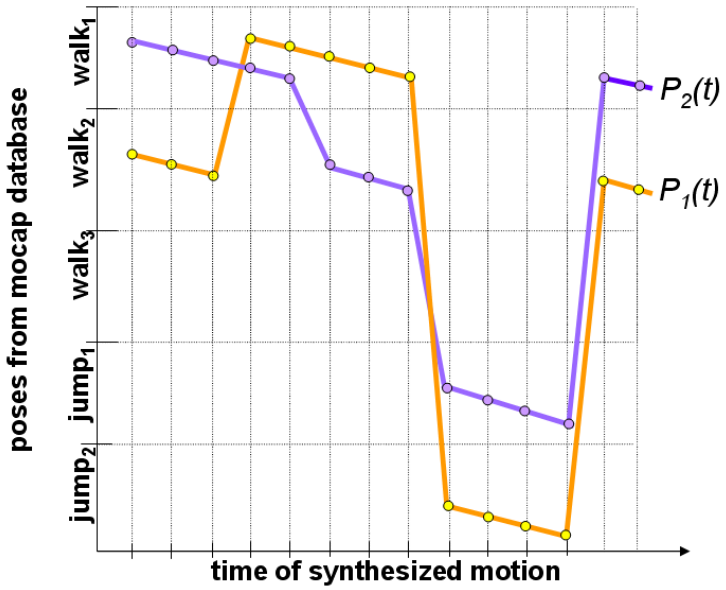

Figure 1.8: A database consisting of five motions: three walks and two jumps. For this example, we assume that $k = 2$ in equation 1.1. The resulting motion is an interpolation of two paths through the motion graph, $P_1(t)$ and $P_2(t)$. The orange curve represents $P_1(t)$ and the purple curve represents $P_2(t)$. The X axis represents the time of the synthesized motion and the Y axis represents an index of the frames of the motions in the database. $P_1(t)$ and $P_2(t)$ can transition from one motion to another independently of each other as each is just following a path through the motion graph.

greatly reduce the number of possible poses, and also explore a higher percentage of natural looking poses.

- This representation is more general than motion graphs because it can generate variations of motions in the motion capture database. It is more general than existing interpolation techniques because it can generate complex motions that involve more than one behavior and does not require motions to be preprocessed a priori into short clips of similar structure.

  This representation is less general than the continuous space described in the previ-

ous section because it is restricted to interpolation of sample points used to construct the space (see Chapter 6 for a more detailed comparison of discrete and continuous spaces). It is, however, more compact than the continuous space because it encodes the dynamics of the motions.

- We can use discrete search techniques to solve this problem. Because optimization in a discrete domain does not require derivatives of the constraints or the optimization function it is less sensitive to the landscape of the optimization function than continuous optimization. Discrete optimization can, therefore, more gracefully handle highly non-linear objective functions and constraints. Discrete search finds a global minima (or a close approximation to it) and as a result will not get stuck in bad local minima as continuous search on a 60 dimensional space often does. On the other hand, the global characteristic of discrete search restricts the size of the search space that can be efficiently searched. We greatly decrease the size of the search space by representing the unknowns as an interpolation of existing poses, by compressing the motion graph into a practically equivalent but much smaller graph and by developing an informative heuristic function that focuses the search efforts only on relevant parts of the graph.

To assess the physical correctness of the resulting motion, we analyze the physical correctness of interpolated motions in Chapter 5 [63]. This analysis studies the interpolated motion in terms of a number of basic physical properties: (1) linear and angular momentum during flight; (2) foot contact, static balance and friction with the ground during stance; (3) continuity of position and velocity between phases. The analysis shows that with a few simple modifications to the straightforward interpolation technique proposed by others, we can prove that these physical properties are satisfied for a wide range of different kinds of motions. The interpolated motion will satisfy these physical properties if the motions used for interpolation do not include significant rotation during the flight phase (runs, forward and vertical jumps, for example), rotate around approximately the same principal axis by approximately the same amount (jumps with turns, for example) or have no flight phase (walks or kicks, for example). In Chapter 4 we show how this analysis applies to the framework of interpolation described by equation 1.1.

As with continuous optimization, optimization in our discrete space also satisfies the three required properties: it generates a physically realistic motion for a human character and the motion matches the rough sketch provided by the user. In contrast to optimization in the continuous space however, discrete optimization also satisfies the other three desired properties: it works at interactive frame rates, it supports less-dynamic or more stylized motions, and it can generate motions that consist of multiple behaviors. Our experiments show that it usually takes between few seconds to few minutes for the discrete optimization to generate a solution. Its runtime depends on the length of the desired motion and the size of the database. The discrete optimization method has other disadvantages when compared to continuous optimization however. When generating a short, single-behavior motion, discrete optimization lacks the generative power of the continuous optimization method: the latter is capable of generating motions whose frames are more than just the interpolation of existing frames in the motion capture database.

Figure 1.9 offers a graphical explanation of the differences between the two methods. The continuous optimization approach constructs a low-dimensional space by treating poses in the sample motions as independent samples. It only considers correlations between joint angles in the same pose and not how they change with time. This corresponds to Figure 1.9(a). The discrete optimization approach, on the other hand, considers not only how the joint angles relate to each other within the same pose but also how they change with time. The poses that it considers are the interpolations of the motion graph poses, the transitions that it considers are the interpolations of the motion graph transitions. This is graphically shown in Figure 1.9(c). As a result, the space constructed by the continuous optimization approach contains more novel motions but is also larger and, consequently, harder to optimize in than the space constructed by the discrete optimization approach. We provide more detailed comparison of two spaces in Chapter 6.

The continuous and discrete approaches are two points in the spectrum of all possible approaches. Figure 1.9(b) shows that we might consider developing a method that compromises between our two proposed methods by taking into account the dynamics of the joint angles in a short-term window in time. Figures 1.9(d-f) suggest that it might be useful to consider splitting joint angles within the same pose into two or more sets, for

example, lower body and upper body joint angles (see for example work of Ikemoto and Forsyth [25]). The spaces for each set can then be constructed independently and the full body motion is the result of combining the searches in each of the spaces.

The structure of the thesis document is as follows. The next chapter reviews the related work. Chapter 3 provides the details of our continuous optimization algorithm and Chapter 4 provides the details of our discrete optimization algorithm. Chapter 5 provides the details of the analysis of the physical correctness of interpolated motion. Chapter 6 discusses in detail the differences between our two methods and concludes with a summary of the contributions of the thesis.

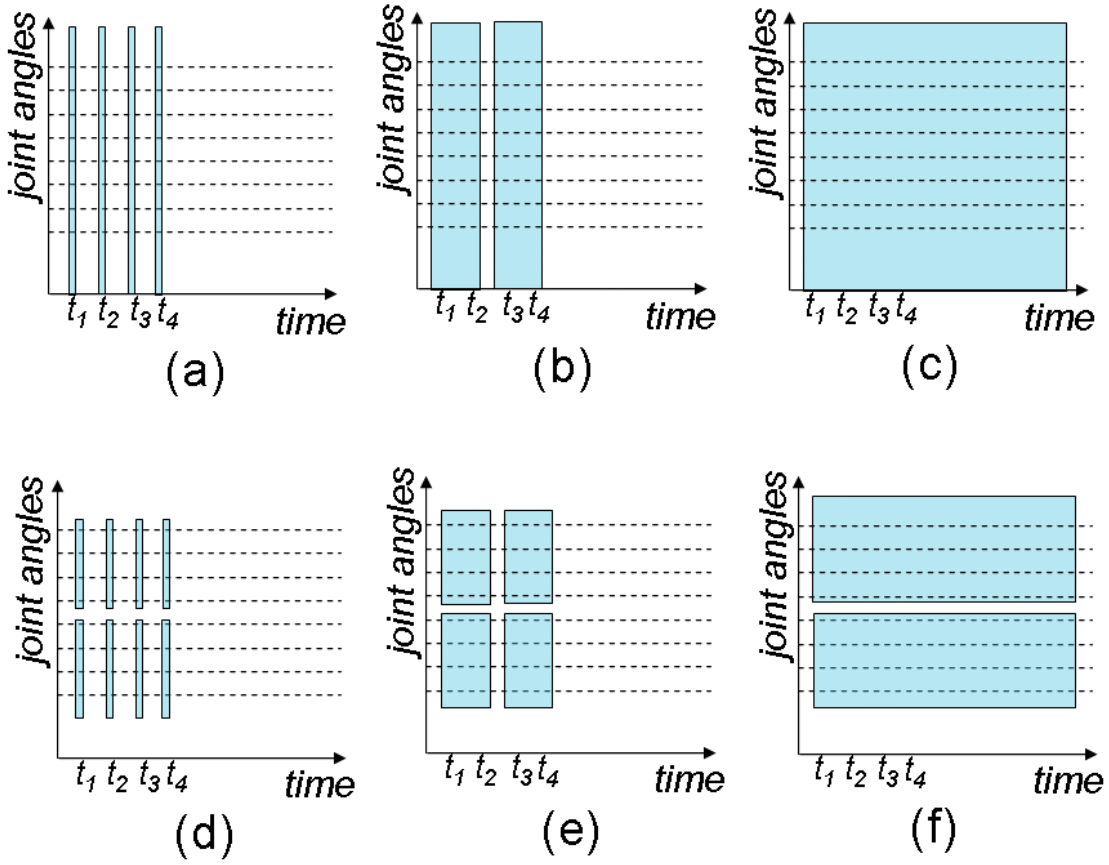

Figure 1.9: A graphical representation of the differences in how the spaces are constructed by the two approaches. The continuous optimization approach considers correlations between joint angles only within the same pose (the graph in (a)). The discrete optimization approach considers the relationship between angles within the same pose and how they change with time (the graph in (c)). The graphs in (b,d-f) suggest other alternatives. The methods in (b) and (e) would analyze correlations between joint angles in poses that are within a short-term window in time. The methods portrayed in (c-e) would analyze upper and lower body joints separately.

Chapter 2

Related work

We first review the prior research in continuous constrained optimization used for the synthesis of a human character animation. This research is related to our continuous representation (Chapter 3). We then review the prior work in the interpolation of human motions and in motion graphs. This work is relevant to our discrete optimization approach (Chapter 4) and to the analysis of physical correctness of interpolated motions (Chapter 5).

2.1 Continuous constrained optimization

Constrained optimization techniques were first introduced to the graphics community by Witkin and Kass [74]. They demonstrated the viability of this approach with a jumping Luxo lamp; its motion was quite compelling as it crouched in anticipation of a jump and compressed to absorb the impact. The user specified the start pose, end pose, and a physically based objective function; the optimizer computed the details of the motion. Despite this promising beginning, optimization has proven difficult for complex articulated characters, and subsequent research has focused on ways to make the approach viable for these more complex systems. Although no systematic studies have yet been published, the problem appears to be made more difficult by higher degree-of-freedom systems, physics constraints, torque-based optimization functions, and longer animations while domain knowl-

edge in the form of existing control laws, motion data, or a close initial guess make the problem more tractable.

One way to keep the problem tractable is to reduce the number of degrees of freedom. Popović and his colleagues [54] developed a interactive system that gave the user fine control over the motion of a single rigid body. Their system was able to produce a wonderful example of a hat spinning as it was tossed onto a hat rack. A number of researchers have shown that the freefall portion of a dive can be efficiently optimized for a simplified character [44, 10, 4]. Huang and his colleagues [23] computed the motion of characters of similar complexity performing such actions as weight lifting and pushups.

Simplifying a complex character also simplifies the problem: Popović and Witkin [55] showed that significant changes to motion capture data can be made by manually reducing the character to the degrees of freedom most important for the task. The optimized motion was then mapped back up to the full character. Our continuous optimization approach [64] is similar in that we also perform the optimization in a lower degree of freedom space, but we find that representation automatically and the motion we compute is physically correct for the full 60 DOF character.

Human motion with many degrees of freedom can be optimized when the animator provides closely spaced keyframes without exact timing information [45]. A related problem is dynamic filtering where an existing motion is optimized to make it physically realistic. In this formulation, the original motion can be thought of as a very closely spaced set of keyframes that function as soft constraints [11, 75, 53]. Short segments of motion can be computed for characters with many degrees of freedom as Rose and his colleagues [61] demonstrated when they computed optimal transitions between human motion segments that began and ended with different but similar poses.

Physics constraints and an optimization function based on torque often make the problem more difficult to optimize. In contrast, purely kinematic techniques give the animator interactive control for making significant changes to the motion [17, 40]. Simplified physical constraints also create tractable problems. Liu and Popović [43] show that some dynamic effects can be preserved by enforcing patterns of linear and angular momentum that do not require the computation of such dynamic parameters as contact forces and joint

torques.

If the dynamics of the system can be made more efficient, the search process also becomes more efficient. Fang and Pollard [13] derived an algorithm for efficiently computing first derivatives of a broad range of physics constraints. Because their system never computed torques, they used a sum of weighted, squared accelerations as an optimization function. They synthesized swinging and leaping motions for characters having from 7 to 22 degrees of freedom. Grzeszczuk and his colleagues [19] developed a neural network approximation of dynamics for a number of systems and used that approximation in the optimization step to reduce the computational cost of the gradient search. Outside of character animation, reduced order models of dynamics have been explored extensively in computer graphics and other fields (e.g., [51] [26] [35]), including for their use in speeding up optimization (e.g., [57]). In our work, we do not simplify our representation of the dynamics of the system (inverse dynamics calculations are performed in the high-dimensional space); instead we reduce the dimensionality of the configuration space of the character that is explored during optimization.

Work in biomechanics and robotics (Pandy and Anderson [5, 47], Hardt [21]) can solve the optimization problem in a high-dimensional space when computing short motion segments of a single behavior (for example one walk cycle). These systems usually program the behavior specific details into the system, such as the fact that the legs and arms move together during a jump. In our work we solve the optimization problem in a low-dimensional space which is also designed for a particular single behavior, but we find this space automatically from examples of motion capture data.

In work done in parallel with our continuous optimization approach, Sulejmanpašić [68] showed that adaptation of ballistic motions with full physics is possible for high DOF characters if careful attention is paid to details such as proper variable scaling and initialization. Sulejmanpašić also explored using PCA on one or two captured motions to reduce the dimensionality of the search space, but did not find it to be effective, in part because reducing dimensionality below 16 DOF made it difficult to satisfy constraints during optimization. In our continuous optimization approach, we show that by constructing a basis set from multiple examples of a behavior and by adding inverse kinematics (IK) to

the optimization process, a smaller number of DOF can be used successfully even without a good initial guess. We speculate that our use of PCA was successful while theirs was not because we used several examples of the desired behavior to create the basis while they used only one.

Finally, we note that many researchers in computer graphics, robotics, computer vision, machine learning and biomechanics have also explored the use of dimensionality reduction techniques to aid in clustering, modelling, and other processing of motion (see, e.g., [27] [65] [12] [8] [41]).

2.2 Interpolation

Interpolation is a very simple and yet a very powerful technique for generating motions that are variations of motions in the database. Perlin [52] proposed one of the first systems that included interpolation. He used blending operations on a set of base motions to create new motions and transitions between them. Wiley and Hahn [73] and Guo and Roberge [20] used linear interpolation on a set of hand-selected example motions to produce modified motions within that set. For example, Wiley and Hahn were able to interpolate among a set of reaching and pointing motions to have the character point to other places in the space. The set of example motions was quite small in this work as motion capture data was not yet easy to obtain.

Rose and his colleagues [60] implemented a very impressive system that used radial basis functions to represent motions for interpolation. The motions included a set of walks and runs of varying speed and emotion. The key events in the motions were selected by hand so that the motions could be appropriately aligned in time for interpolation.

Kovar and Gleicher [32] added a search technique for identifying a set of motions with similar time events that could then be interpolated. They also presented techniques for automatically registering the motions for interpolation [31].

Linear interpolation has been used extensively for creating transitions between motions [72, 60, 52]. Transitions are created by blending portions of two motions with a

weight that changes over time.

Abe and his colleagues [3] used optimization to synthesize a family of highly dynamic motions based on a given motion capture clip and then used interpolation to create intermediate motions. They observed that the space of motions does not need to be sampled very densely for the interpolation to produce good results.

All of these methods require a set of carefully crafted example motions of the same behavior before interpolation. Our discrete search method also relies on interpolation to synthesize variations of existing motions, but it does not require carefully crafted example motions — it operates on the whole motion capture database. We also synthesize complex multi-behavior motions rather than interpolating just single behaviors.

In most of the previous work, the weights that interpolate motions were found based on user-specified parameters such as kick position, speed of the motions or their style. For example, given a user-specified kick position, these methods would select a few motions with similar kick positions and then compute weights that interpolate these motions based on the difference between the kick positions in these motions and the desired kick position. For our problem we need to compute a motion that follows user sketch, satisfies a set of user specified constraints and minimizes some objective function (such as minimizing energy). We also want this motion to be close to physically correct. Because current interpolation approaches do not do any optimization (or search) it is not clear how to extend them to solve this problem. Also, none of these methods consider the physical correctness of the interpolated motion.

2.3 Motion Graphs

In contrast to interpolation approaches that are good at synthesizing short motions, motion graphs are good at synthesizing long, multi-behavior motions and they do not require motions to be split into similar structure motion clips. It is also possible to search motion graphs for solutions that optimize some objective functions and satisfy user constraints.

Inspired by the technique of Schödl and his colleagues [66] that allowed a long video to

be synthesized from a short clip, motion graphs were developed simultaneously by several research groups in 2002 and extended in subsequent years.

Motion graph related approaches can be roughly divided into *on-line* and *off-line* approaches. In *on-line* approaches the motion is generated in response to current user input (from a joystick, for example). These algorithms therefore do not need to worry about synthesizing long motions that minimize an objective function and therefore they perform only a very local search or no search at all—just pick an action from the current state that matches user input the best. This is different from *off-line* search techniques where we know the full motion specification beforehand and are interested in finding a solution that matches this specification and minimizes some objective criteria – energy for example. Our work falls into the category of *off-line* techniques.

A number of *on-line* approaches were created in the past few years. Lee and his colleagues [38] use a local search algorithm to generate motions at interactive rates for three different user interfaces. Sung and his colleagues [69] use local search to generate motions for crowds. Gleicher and his colleagues [18] use a semi-automatic process to split a corpus of motion capture data into a set of short clips that can be concatenated to create a continuous stream of motions to synthesize motions for virtual environments. Mnardais and his colleagues [46] present a real-time synchronization algorithm which in real time automatically (based on contacts and priorities) detects which motions are incompatible for blending.

Several *on-line* approaches are closely related to our work — they also combine motion graph and interpolation techniques. Park and his colleagues [49, 48] manually preprocess motion into short segments, arrange segments with similar structure into nodes in a graph. They blend segments at each node and use local search to generate locomotion in real-time. The graph construction was later automated by Kwon and his colleagues [34] for locomotive motions. They proposed an automatic way for cutting motions into segments and for grouping similar segments into nodes. In 2006, Shin and Oh [24] extended these techniques to include more behaviors. They use a method similar to the one proposed by [18] to semi-automatically build a fat graph in which a node corresponds to a pose and its incoming and outgoing edges represent motion segments starting from and ending at

similar poses. A group of similar motion segments leaving particular node are parameterized and blended using traditional motion blending techniques.

The main difference from our work is that these techniques are tailored for interactive joystick-like control—they do not do any search, they follow the graph based on current user input (joystick for example) and interpolate motion segments leaving the current node. Because there is no search, these methods are fast enough for continuous interactive control with a joystick. Our method is good for an optimal motion synthesis based on a sketch. It is slower, but can compute a motion that matches a user sketch, satisfies user constraints and minimizes an objective function. Also, the previous approaches build the graph with contrived structure (edges correspond to similar motion segments) in a semi-automatic way. Our method works on standard motion graphs. Also, we do not parameterize similar motions to compute weights for interpolation. Weights are found as part of the search process.

Therefore our method falls into the category of *off-line* search techniques. If the user specification for the entire motion is known before the search starts then a motion graph can be searched for a complete motion using global search techniques. A number of sub-optimal algorithms were developed to search motion graphs for solutions at interactive frame rates. Kovar and his colleagues [33] employed a branch and bound algorithm to get an avatar to follow a sketched path. Arikan and Forsyth [6] created a hierarchy of graphs and employed a randomized search algorithm for the synthesis of a new motion subject to user-specified constraints. Pullen and Bregler [56] segmented motion data into small pieces and rearranged them to match user-specified keyframes. In 2003, Arikan and his colleagues [7] presented a new search approach based on dynamic programming that supports user-specified annotations of the motion. Choi and his colleagues [9] presented a scheme for planning natural-looking locomotion of a biped figure based on a combination of probabilistic path planning and hierarchical displacement mapping. Sung and his colleagues [70] used probabilistic roadmaps and displacement mapping to synthesize motion for crowds. Li and his colleagues [41] created a two level statistical model to produce a dance motion with variations in the details. Srinivasan and his colleagues [67] pre-compute a mobility map that for every state in the motion graph encodes all locations

the character can move to within a fixed number of "smooth" actions. At run-time they perform a local greedy search for the best alternative based on the current goal.

To guarantee fast performance these approaches do not find optimal solutions but instead use sub-optimal search techniques. To find an optimal solution at interactive frame rates, Lau and Kuffner [37] manually created an abstracted behavior-based motion graph with a very small number of nodes so that it can be searched efficiently. In later work [36] they show how to pre-compute search trees for this behavior-based motion graph and use them to make the search run even faster. This, later approach gives up optimality but for the examples they presented in the paper the cost of the resulting solution is very close to an optimal one. Lee and Lee [39] present a pre-computation method to compute a policy that, for each possible control input and avatar state, indicates how the avatar should move. This approach allows avatars to be animated and controlled interactively with minimal run-time cost at the expense of memory and limitations in the possible control inputs.

In our work we compress the motion graph into an almost identical but much smaller graph and develop informative heuristic functions that allow us to search the motion graph as well as the interpolated motion graph for motions that are close to optimal. In the results section of Chapter 4 we show the importance of finding nearly optimal solutions—they are usually much more natural than sub-optimal ones because they avoid dithering and inefficient motion strategies common in suboptimal solutions.

Also, the previously developed *off-line* based motion graph approaches essentially can only re-sequence existing motion segments based on user-specified constraints. They can not synthesize a motion that is a variation of existing motions. This restriction makes it difficult to satisfy user constraints exactly. For example, it would be impossible to synthesize a motion for picking up an object from a table of a particular height if the database does not contain a motion for exactly that height. In our discrete optimization approach we overcome this problem by introducing an interpolated motion graph and showing how to search it efficiently.

Chapter 3

Motion synthesis in a continuous space

In this chapter we show how to build a continuous low-dimensional representation of a human motion and use it to synthesize the desired motion. Continuous optimization [74] has been a popular approach for finding a human motion when only a rough sketch is provided. The optimization problem, however, is hard to solve in part because of its high-dimensionality and difficulty in defining a good objective function that would result in a natural looking motion. The algorithm described in this chapter tries to address these issues. It is based on two observations. First, many dynamic human motions can be adequately represented with only five to ten degrees of freedom. Second, motions with similar behavior can be used to construct a low-dimensional space that can represent well other examples of the same behavior. The algorithm therefore constrains the optimization to this low-dimensional space. This makes the job of the optimizer much easier, and the solution becomes more likely to be a naturally looking motion.

Fifty to sixty dimensions are often used to represent a high quality human motion. For many behaviors, however, the movements of the joints are highly correlated. For example, during a walk cycle, the arms, legs and torso tend to move in a similar oscillatory pattern. As a result, the dimensionality of motions can be greatly reduced by applying a simple dimensionality reduction technique such as PCA (Principal Components Analysis) to poses taken from human motion sequences. Figure 3.1 shows the average squared angle

error between a number of motions and their projections onto optimal low-dimensional linear subspaces obtained using PCA. For the common human behaviors that we have tested, the error becomes small for representations with more than five to ten dimensions.

Figure 3.2 compares the error of representing forward jump motions in six different low-dimensional spaces. The error is averaged over twenty different jumping motions of varying height and length. The low-dimensional space computed from the motion that is being represented, (a), naturally provides the best representation. During the synthesis process, however, the desired motion is not known and thus this space cannot be computed. The low-dimensional space constructed from the motions of the same behavior (whether it is just three similar motions as in (b) or a set of twenty motions as in (c)) can represent the motion quite well with seven dimensions. The low-dimensional space computed from one example of a given behavior, (d), requires higher dimensionality because it does not have enough generality to represent other motions well. When we incorporate motions with different behaviors, the required dimensionality of the space increases. A general mix of 150 motions, (e), however, provides a better representation than a behavior-specific representation for the wrong behavior, (f). Figure 3.2 illustrates the results of dimensionality reduction for forward jumping, but we have obtained similar results for other behaviors such as running and walking.

A seven-dimensional space constructed from several motions of the same behavior generally represents the desired motion well. When visual artifacts remain in this representation, they are usually the result of contacts. For example, during the initiation of a forward jump, both feet are planted on the ground which constrains the relationship between the hip, knee, and ankle angles. This constraint may not be represented with sufficient accuracy in the low-dimensional space even though that space does model the basic fore/aft swinging of the arms and legs. We incorporate inverse kinematics as part of our optimization approach to reduce artifacts caused by this limitation (Section 3.1.2).

In Section 3.2, we demonstrate that a low-dimensional space constructed from a few motions of a particular behavior, such as spaces (b) and (c), can be used to synthesize physically realistic, natural-looking motions of same behavior but with quite different parameters (e.g., length, height or degree of turn for jumping or step length for running).

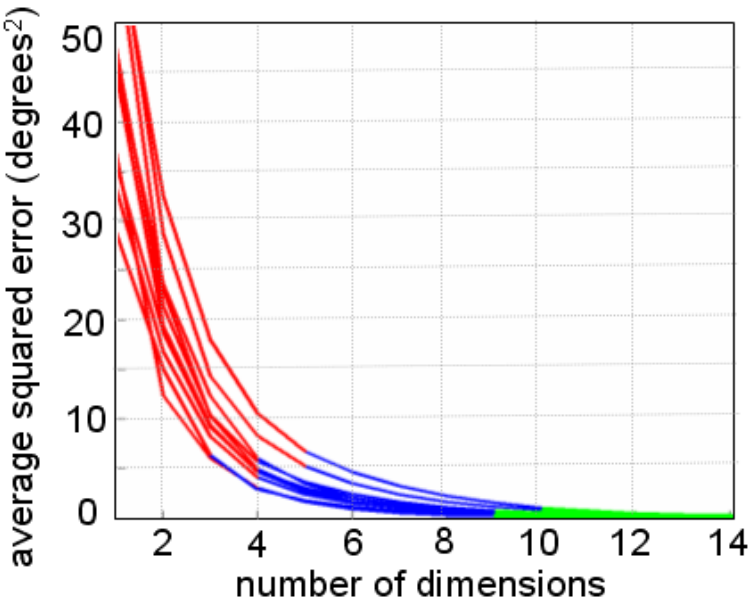

Figure 3.1: Error between the original motion and the corresponding k-dimensional representation for a number of behaviors: running, walking, jumping, climbing, stretching, boxing, drinking, playing football, lifting objects, sitting down and getting up. The error was averaged over ten to twenty motions within each behavior. Each motion was represented as a collection of poses. The k-dimensional representation was computed by first using PCA to compute principal components for the set of full-dimensional poses and then projecting each full-dimensional pose onto the k components with the most variation in the data. The error is a squared error between the angles of the full-dimensional motion and its k-dimensional representation averaged over all joint angles and all poses in the motion. The curve color gives an approximate measure of the visual quality: red color indicates motions with large visual artifacts; blue color indicates motions that look very similar to the full-dimensional motion except for some sliding of the feet; and green color indicates motions that look nearly indistinguishable from the full-dimensional motion.

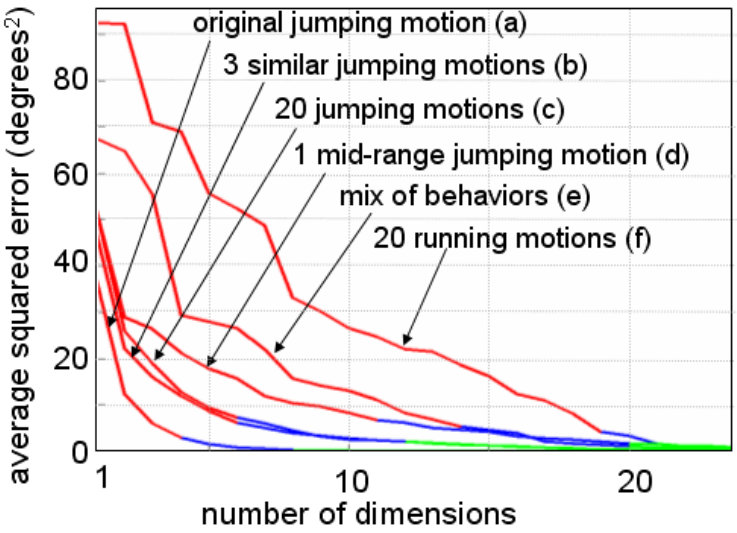

Figure 3.2: Motion representation error of a full-dimensional motion in a k-dimensional space averaged over twenty different jumping motions of varying height and length. For *each* of the motions, the k-dimensional space is spanned by k principal components that are computed from: (a) the motion that is being represented, (b) three jumping motions that are visually similar to the motion being represented, (c) a set of twenty jumping motions, (d) a single mid-range jumping motion, (e) a mix of 150 behaviors (walking, running, jumping, climbing, punching, dribbling a basketball, lifting, drinking and other common human activities) and (f) twenty running motions. The sets of motions used for constructing spaces (c-f) are the same for each of the twenty motions. As in Figure 3.1, each curve is colored to indicate the visual quality.

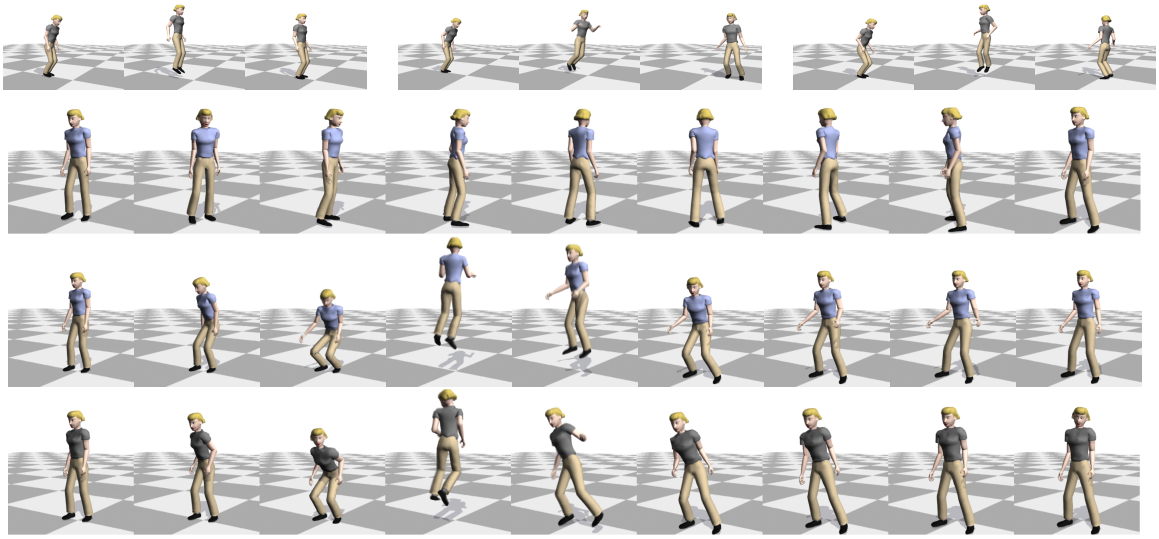

Figure 3.3: Synthesizing a vertical jump with a 360^o turn. (Top row) Motion capture data used to compute the low-dimensional space for the optimization: a forward jump, a forward jump with a 90^o turn, and a vertical jump with a 180^o turn. (Second row) Initial guess. Constraints were set on the first and the last pose of the motion and on the position of the feet during the stance phases. The duration of each stance phase and the desired height of the jump were also specified. (Third row) Synthesized motion. (Last row) Motion capture data of a similar motion for comparison.

A low-dimensional space constructed from one example of a particular behavior, such as space (d), may produce unnatural motions. A low-dimensional space constructed from motions with different behaviors (such as spaces (e) and (f) in Figure 3.2) requires solving an optimization problem of higher dimensionality which in our experience does not produce reliable results. In a higher-dimensional space, the problem is harder to solve and more dependent on having an optimization criterion which defines natural human motions, something that is difficult to define mathematically.

The next Section, 3.1, provides the details of the optimization problem in the continuous low-dimensional space. Section 3.2 describe the experimental results and Section 3.3 summarizes our approach.

3.1 Low-dimensional Optimization

To use this system, the user specifies a rough sketch of the desired motion, M_s, and the constraints that should be enforced. The user also selects the motions used to define the low-dimensional space for the desired motion. Constrained optimization is then used to automatically find a motion that minimizes some objective function subject to satisfying the user-specified and physics constraints (Figure 3.3).

We formulate the optimization problem by solving for the joint angles and the root position of the character's motion $M(t)$ as opposed to solving for a force/torque function $T(t)$ where $T(t) = \{\tau_1(t) \dots \tau_n(t)\}$ and $\tau_i(t)$ is the torque applied to joint i at time t. Inverse dynamics equations are used to compute the force/torque function $T(t)$ at any given time t. A number of constraints must be set on the force/torque function to preserve physical validity. The constraints are enforced at discrete times. The motion may be physically infeasible in between these points, but these constraints are enough to generate visually pleasing motion.

Because the unknown of the optimization problem, $M(t)$, is expressed in a low-dimensional space, it is easy to make this formulation of the optimization problem low-dimensional. Computing a reasonable initial guess for a motion $M(t)$ from a user sketch is also easier than computing a reasonable initial guess for the force/torque function $T(t)$. Finally, inverse dynamics is usually easier to solve than forward dynamics [Featherstone 1987, page 79].

3.1.1 Low-dimensional Problem Representation

Each motion M consists of a sequence of frames $M(t) = \{p(t), Q(t)\}$, where $Q(t) = \{q_1(t) \dots q_n(t)\}$ are the angles of all of the joints (including the root orientation) and $p(t)$ is the position of the root segment. $Q(t)$ is a point in an n-dimensional space. Let us consider a d-dimensional linear subspace of the original space that is spanned by unit length orthogonal vectors, $B_1 \dots B_d$, with origin $Q_m = (1/T) \sum_{i=1}^{T} Q(t_i)$, where T is the number of time samples. Then, we can approximate $Q(t)$ by a linear combination of the

basis vectors $B_1 \ldots B_d$ using only d scalar coefficients $A_1(t) \ldots A_d(t)$, as:

$$Q'(t) = Q_m + B_1 A_1(t) + B_2 A_2(t) + \ldots + B_d A_d(t). \tag{3.1}$$

PCA is a technique that finds $B_1 \ldots B_d$ so that the error $E = \sum_{i=1}^{T} (Q(t_i) - Q'(t_i))^2$ is minimized [28].

Given motions $M_{B1} \ldots M_{BK}$, selected by a user, we use PCA to find the d-dimensional subspace, L, that represents them with the smallest error E. If we now use Equation 3.1 for the representation of $M(t)$, then the unknowns of the optimization are root position $p(t)$, the mean of the joint angles Q_m, and the coefficients $A_1(t) \ldots A_d(t)$. We follow a standard approach of representing each $A_i(t)$ and $p(t)$ using cubic B-splines. The root position $p(t)$ is only unknown during the stance phase; during the flight phase the position of the center of mass (COM) of the character can be computed from the lift-off velocity, and $p(t)$ can then be computed from the COM position and the angles of the character.

The lower-dimensional space reduces the complexity of the optimization problem considerably. Let K be the number of control points in a B-spline curve used to approximate $p(t)$ and either $q_i(t)$ or $A_i(t)$. The full space has about $(n+3)K$ unknowns (where $n \approx 60$ for a human character), while the reduced dimensional space has $(d+3)K + n$ unknowns, with $7 < d < 9$ for the examples in this work. Therefore, the number of unknowns is reduced by a factor of six to seven, which in our experience results in significantly faster and more consistent convergence of the optimization problem. In many cases, the mean of the joint angles, Q_m, can also be computed from example motions and excluded from the optimization function, further reducing the number of the unknowns. However, we kept Q_m as an unknown in the examples reported here.

Because we include root orientation in the dimensionality reduction analysis, we first preprocess all the motions by rotating them so that the character faces the positive Z direction on the first frame of the motion. Keeping the root orientation in the basis worked well in the examples we tried, but for some motions with significant change in orientation it may be better to treat root orientation as an additional variable and add 3K extra parameters to the optimization problem.

To estimate the dimensionality, d, of the linear subspace L we use a standard heuristic [15]. We choose the smallest d such that:

$$E_r = \frac{\sum_{i=1}^{d} \lambda_i}{\sum_{i=1}^{n} \lambda_i} \geq 0.9 \tag{3.2}$$

where $\lambda_i, i = 1 \ldots n$, are the eigenvalues computed by PCA and sorted in decreasing order. Because the variance along the i-th principal component is given by the i-th eigenvalue, E_r is an indicator of how much information is retained when all data is represented in the optimal d-dimensional linear subspace.

3.1.2 Inverse Kinematics for Limbs in Contact

In a low-dimensional space, the constraints that relate two or more points on the character's body may not be satisfied exactly. Consider, for example, the contact constraints on the character's feet during the double support phase of a jump. The low-dimensional space may not include a pose that would exactly satisfy constraints for both feet (or the pose may be unnatural). We address these problems by using inverse kinematics (IK) to transform the degrees of freedom in the optimization to a set that can be independently specified while maintaining constraints. IK is used to represent the angles for the arms or legs that are in contact with the environment. Intuitively, this allows the optimizer to satisfy contact constraints exactly by allowing the resulting motion to move slightly outside the space specified by the basis.

Consider a human arm consisting of three limb segments: upper arm, lower arm and hand. It can be represented by a seven DOF kinematic chain with one spherical (three DOF) joint at the shoulder, one at the wrist, and one revolute (one DOF) joint at the elbow. When an arm is in contact with the environment, the position and the orientation of the hand segment is fixed and the location of the shoulder joint is known. There is then a one DOF redundancy in the seven DOF kinematic chain representing the arm. As was pointed out by Korein and Badler [30] and later by Lee and Shin [40], this redundancy is in the "elbow circle" of the arm; the elbow can rotate even when the hand is in contact and the position of the shoulder joint is fixed. In this case all seven angles for the arm linkage can

be analytically expressed through only one parameter, P, representing the elbow rotation (see [71] for details).

Let $Q_u(t) = \{q_1(t)...q_k(t)\}$ be all the angles of the character joints excluding the joints that belong to the arms and legs that are in contact with the environment. As before, we represent these angles using a d-dimensional representation:

$$Q_u(t) = Q_m + B_1 A_1(t) + B_2 A_2(t) + \ldots + B_d A_d(t) \tag{3.3}$$

We represent the angles for all limbs (e.g., arms and legs) that are in contact with an environment, $Q_k(t) = \{q_{k+1}(t)...q_n(t)\}$ using IK:

$$Q_k(t) = F(P_1(t), ..., P_r(t)) \tag{3.4}$$

where r is the number of limbs in contact, F is the IK function and $P_1(t)...P_r(t)$ are free parameters that define angles for the limbs in contact. Because there are four limbs (two arms and two legs), we are adding at most $4K$ unknowns to the optimization problem. When we use IK to compute some of the joint angles, those joint angles are no longer constrained to the low-dimensional space. To compensate for this, we add an additional term to the optimization function that favors poses in the low-dimensional space (see Section 3.1.4).

The model we use of the human leg (upper leg, lower leg, foot and toe segments) is similar in degrees of freedom to that of the arm except for the addition of the toe joint. If both toe and foot segments are constrained then the orientation of the foot segment needed for IK is known. If only the toe segment is constrained, then the angle between the toe and the foot segments is treated as part of $Q_u(t)$ and the foot segment orientation can be computed from this angle and the contact information given for the toe segment.

3.1.3 Constraints

This system includes two types of constraints: user-defined constraints that allow the user to control the resulting motion and constraints that ensure physical validity of the motion.

The most common user-specified constraints are pose constraints, contact constraints, and time constraints. We used pose constraints to fix initial, final, and key postures, such as a particular pose during the aerial phase of a back flip. The constraint poses are specified in the full-dimensional space and then projected onto the low-dimensional space. We used contact constraints to specify foot configurations for ground contact, and time constraints were used to specify the durations of various phases of the motion. Some high-level user controls, such as height or length of a jump, speed of a walk or height of a back flip were also provided. If time was not provided and could not be computed from other constraints (e.g. height of a jump), then it was set as an additional variable in the optimization.

Constraints on the physical validity of the motion are added automatically by the system and are designed to preserve physical validity of the motion. They include joint angle limits, torque limits, and constraints on aggregate force. Joint angle limits are straightforward. (The limits were estimated based on a large motion capture database.) Aggregate force constraints are set as in Fang and Pollard [13], and include constraints for conservation of momentum during flight, as well as constraints on ground contact forces. For torque constraints, inverse dynamics allows us to compute torques for the character with a single point of contact with the environment. When a closed loop is formed by two or more simultaneous contacts, however, the problem is undetermined. We use the approximation method proposed by Ko and Badler [29].

During the implementation we found it beneficial to run the optimization problem in two steps: first, we solve a simpler problem with no constraints on torque limits and aggregate forces; second, we solve a full problem with those constraints added. The solution from the first optimization provided a good initial guess for the second optimization, making it easier to solve.

3.1.4 Objective Function

In our implementation, the objective function, $G(M)$, is a weighted sum of three components:

$$G(M) = w_T G_T(M) + w_A G_A(M) + w_P G_P(M) \tag{3.5}$$

The component $G_T(M)$ minimizes the sum of squared torques:

$$G_T(M) = \int \sum_{i=1}^{n} (\tau_i^2(t))dt \tag{3.6}$$

The component $G_A(M)$ ensures smoothness of the joint angle trajectories and root trajectory over time. It minimizes the sum of squared joint accelerations and sum of squared root accelerations:

$$G_A(M) = \int (\ddot{p}^2(t) + \sum_{i=1}^{n} \ddot{q}_i^2(t))dt \tag{3.7}$$

The component $G_P(M)$ ensures that the resulting motion has correlations between the angles and a distribution of poses around the mean pose similar to the ones found in the motions used to construct the basis. When we run PCA on motions $M_{B1} \ldots M_{BK}$, selected by the user, we obtain both the principal components of the d-dimensional linear subspace L, as well as the variance of the data points in these motions along the principal components, given by corresponding eigenvalues. The $G_P(M)$ component penalizes the deviation of coefficients from zero in inverse proportion to the standard deviation along the corresponding principal component:

$$G_P(M) = \int \sum_{i=1}^{d} (A_i^2(t)/\lambda_i)dt \tag{3.8}$$

From our experience, increasing the weight of the $G_T(M)$ component results in a more realistic motion, but the optimization problem takes longer to converge. Increasing the weight of the $G_A(M)$ component results in a faster convergence of the optimization problem, but the motion may not be energy efficient and as a result may not look as good. Increasing the weight of the $G_P(M)$ component results in a solution that more closely resembles the basis motions. Decreasing the weight, on the other hand, usually results in a solution that minimizes the sums of squared torques and accelerations better. The solution,

however, may often look unrealistic (see Section 3.2). Optimizing in a low-dimensional space as well as using the $G_P(M)$ term results in natural coordination patterns.

3.1.5 Implementation Details

In our implementation we used a sequential quadratic programming package, SNOPT [16], a commercially available library that solves general nonlinear constrained optimization problems. We also used a modeling language, AMPL [14], that allows the user to easily formulate linear and nonlinear optimization problems in mathematical terms and automatically generates code appropriate for various solvers.

AMPL uses automatic differentiation to compute derivatives for the optimization function and the constraints. Automatic differentiation takes as input a section of code that computes the value of a function and outputs a new piece of code that computes analytical derivatives for that function. Unlike numerical differentiation methods, automatic differentiation is based on the chain rule computation of derivatives and is therefore considered an analytical differentiation method. It yields exact derivatives within machine accuracy.

3.2 Experimental Results

In this section we analyze the performance of the algorithm and show a number of motions generated for a human character with 60 degrees of freedom using our approach. For each example, a user specified the start and end poses for the motion, the contact information and the timing for the stance and flight phases. The user also selected a few motions from the database that contained similar behaviors to the desired motion. Our system then automatically found a motion that minimized the optimization function $G(M)$ (Equation 3.5) and satisfied the user-specified and the physics constraints. For all examples, the initial guess was a linear interpolation of the parameters (angles, position, IK parameters) for the starting and ending poses.

Each optimization took from three to sixty minutes to converge. The timing depends

on many parameters: the length of the motion, the dimensionality of the low-dimensional space, the weight of the energy component of the optimization function, $G_T(M)$, and the intrinsic parameters set for the optimizer (e.g., the number of major iterations, which we set to 1500). For example, when the problem was represented in a seven-dimensional space all the jumping motions took less than ten minutes to converge with equal weight set for both energy and smoothness components of the optimization function. Each jump was about two seconds in length. All the experiments were run on a 3GHz Pentium 4 computer with 1GB of RAM. The videos for most experiments is available in the following website: $http://graphics.cs.cmu.edu/projects/lowd\_optimization/$

The bias toward "realistically" looking motions. We found that optimizing in a low-dimensional space as well as adding the $G_P(M)$ component to the optimization function biases the solution towards natural-looking motions. Optimizations run in higher-dimensional spaces with the weight on the $G_P(M)$ component set to zero usually result in a solution that minimizes the sum of squared torques and accelerations ($G_T(M)$ and $G_A(M)$ components) better, but has unnatural visual artifacts.

We generated the same forward jump three times: first, by representing the problem in a six-dimensional space with non-zero weight on the $G_P(M)$ component; second, by representing the problem in a twenty-dimensional space with the same weight on $G_P(M)$ component; third, by representing the problem in the same twenty-dimensional space with zero weight on $G_P(M)$ component. The solution that we obtained in the first and the second cases was natural looking, although running the optimization in higher dimensions was less reliable in general. The solution in the third case did not look natural, despite the fact that it was more optimal with respect to the $G_T(M)$ component. We observed similar results for other experiments.

The generality of the low-dimensional space. The same low-dimensional subspace can be used to synthesize a variety of motions. To demonstrate this, we used three different jumps to find a basis: a forward jump, a forward jump with a 90^o turn and a vertical jump with 180^o turn (Figure 3.3). We then synthesized a number of different jumps using that basis, varying the length of the jump and the size of the turn. The synthesized motion of the arms and legs is natural, and as the length of the jump increases it appears as if the

character tries harder. We were also able to synthesize a high vertical jump with a 360^o turn (Figure 3.3) although the basis only contained a low vertical jump with a 180^o turn. In a separate set of experiments, we also combined twenty forward jumps (the same jumps as were used in Figure 3.2) to compute a basis. According to equation 3.2 the dimensionality of the basis was eight. Jumps with varying height and length could also be synthesized in this space.

If the basis does not adequately represent the desired motion, the optimizer will produce a result that looks unnatural or violates the physics constraints. For example, we synthesized a long jump using a basis computed from a very short jump. The resulting motion looked physically realistic, but the arm motion was restricted in an unnatural way because the low-dimensional space did not allow sufficient arm motion. Synthesizing a jump using a basis computed from twenty running motions converged to an even more oddly looking motion - the arms and legs of the character coordinated in the similar pattern, as they would for a run.

Other motions. We demonstrate the generality of this approach by synthesizing additional behaviors: running, running across stones, walking, and an acrobatic back flip.

Running (Figure 3.4). We used eight forward running motions and one motion of running across stepping stones to find the basis. We then synthesized three running motions with different step lengths, three motions that ran across stepping stones with varying placement and speeds and one running motion with a jump over water.

Walking (Figure 3.5). We used seven forward walking motions and one walking motion with the step over an obstacle to compute the basis. We synthesized three walks with different step lengths, including an exaggerated walk with a very large step length. We also synthesized two motions that stepped over obstacles of different heights.

Acrobatic Back Flip (Figure 3.6). We used one acrobatic back flip motion to compute the basis and synthesized two back flips of different distances and heights.

User Control. The optimizer finds a motion that minimizes a given objective criterion and is physically valid. Specifying additional constraints on the motion allows finer control over the details of the motion. We synthesized a back flip with straight legs in the middle

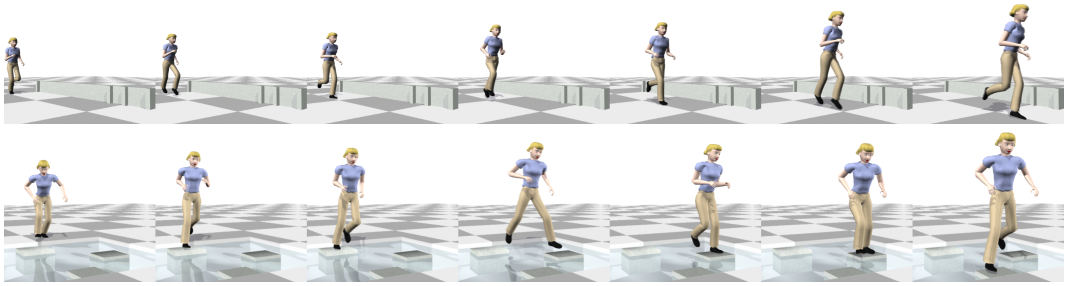

Figure 3.4: A forward run and a run across stepping stones.

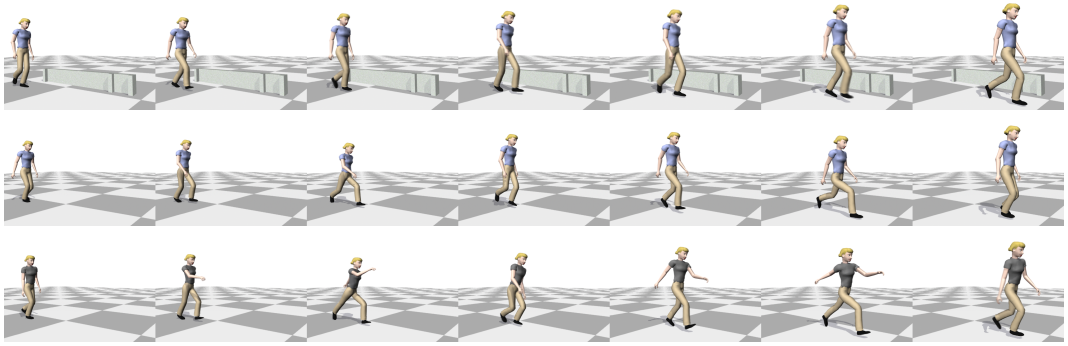

Figure 3.5: A normal walk, a walk with an exaggerated step length, and motion capture data of a walk with an exaggerated step length for comparison.

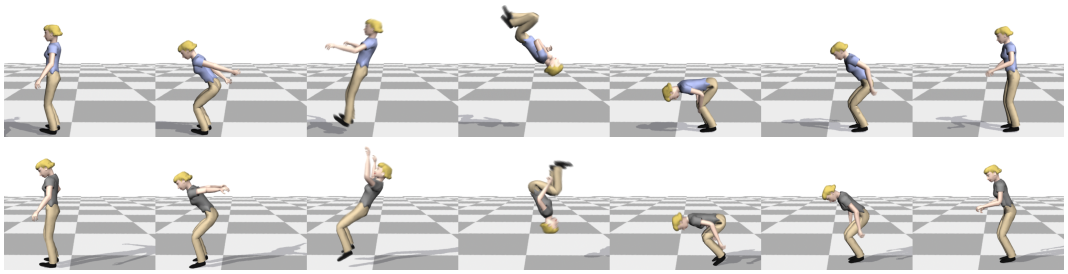

Figure 3.6: A back flip and motion capture data for comparison.

of the flip by specifying a middle pose for the flip. We synthesized a run with a jump over water where the user controls the spread of the legs by adding a straddle pose in the middle of the jump. We also modified the style of the run over stepping stones by specifying two additional poses to generate a run where the feet are raised higher.

3.3 Discussion

The key insights of the continuous optimization technique we have presented in this chapter are that optimization of human motion can occur much more effectively in a lower-dimensional space and that such a space can be easily created from motion capture clips of similar behaviors. The low-dimensional space allows us to generate natural-looking and physically valid motion for characters that have 60 degrees of freedom and only a rough sketch as the initial guess. This approach has proven to be effective for a wide variety of different human motions, both highly dynamic (back flip) and less dynamic (walking). The approach is quite robust to the choice of motion clips that are used to form the lower-dimensional space as long as they are not overly restrictive (e.g., a basis from just a single motion) and are similar behaviors to the desired behavior. It provides good control for the user, allowing him or her to specify the location of the footfalls for a path of stepping stones or to specify an intermediate pose for the flight phase of a back flip.

Solving the optimization problem in an appropriate lower-dimensional space makes it not only more likely that the optimizer will converge to a feasible solution but also more likely that it will converge to a solution that matches the strategy a human would have used to perform that task. This feature of our approach reduces the burden on the user because fewer constraints are required to guide the optimizer. To generate a walk motion, for example, Hardt and his colleagues had to specify symmetry and anti-symmetry constraints for human joint angles and contact forces [22]. The in- and out-of-phase motion of the walk was captured by our basis automatically, allowing us to generate a walk without explicitly specifying these constraints.

In the examples presented here, the user selected the motions for the basis. This task

was not burdensome and only required a few minutes of browsing a reasonably sized database (such as the publicly available database at mocap.cs.cmu.edu). However, it should be possible to search a database automatically for appropriate motions based on the constraints on the desired motion provided by the user. Liu and Popović [43] solved a related problem; their system searched a motion capture database for transition poses that separated constrained and unconstrained phases. They used the following training parameters: flight distance, flight height, previous flight distance, takeoff angle, landing angle, spin angle, foot speed at takeoff and landing, and the average horizontal speed. We believe that a similar technique could be used to search the database for the motions required to find the low-dimensional space. We have implemented a simpler algorithm that looks only at contact information and the overall direction of the motion to select motions from the database. We found that even this very simple approach generally selected appropriate motions.

Chapter 4

Motion synthesis in a discrete space

In this chapter, we explore a discrete representation of human motion. As with the continuous approach discussed in the previous chapter, the key insight is to restrict the search space. The full search space for a human character with 60 degrees of freedom has 100^{60} possible poses if we discretize each degree of freedom of the character into 100 values. The majority of these poses, however, are not natural. Instead, we would like to find a representation of human motion that is compact and contains a high percentage of natural poses and natural velocities while avoiding unnatural poses. At the same time, the representation should be general enough to represent a variety of human motions.

Motion graphs provide one compact representation of human motion [41, 6, 33, 38, 6, 7]. Given a database of human motions, a motion graph is constructed by finding similar poses and creating transitions between these poses. The transitions are associated with probabilities: the transitions between more similar poses receive higher probability values, and the transitions between less similar poses receive lower probability values. Figure 4.1(a) gives an example of a motion graph constructed from two walking motions. A new motion can be generated simply by traversing a *path* through the graph (Figure 4.1(b)). A *path* through the graph can be formally defined as an ordered sequence of poses, where any two consecutive poses are connected by an edge in the graph. Discrete search techniques can be used to search the motion graph for a path that satisfies

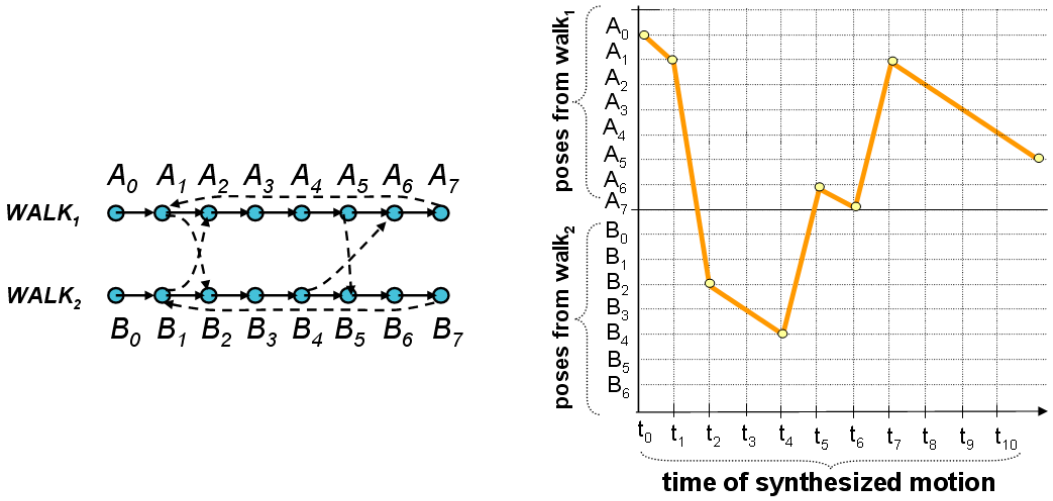

Figure 4.1: (a) Simple motion graph for two walking motions. States A_1 and B_1 are similar and therefore two transitions are added to the motion graph: a transition from state A_1 to state B_2 and a transition from B_1 to state A_2; (b) The curve represents a motion that is generated by traversing a path through the motion graph shown in (a). The X axis represents the time of the synthesized motion and the Y axis represents an index of the frames of the motions in the database.

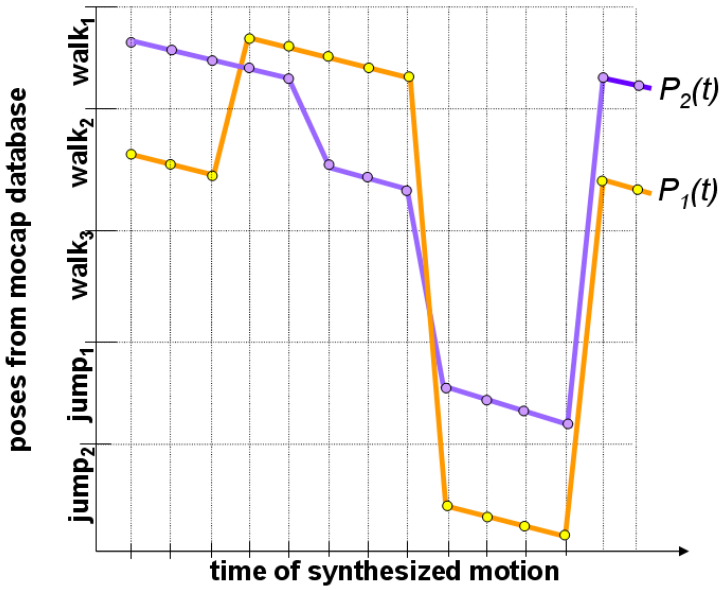

Figure 4.2: The database consists of five motions: three walks and two jumps. For this example we assume that $k = 2$ in equation 4.1. The resulting motion is an interpolation of two paths through the motion graph, $P_1(t)$ and $P_2(t)$. The orange curve represents $P_1(t)$ and the purple curve represents $P_2(t)$. The X axis represents the time of the synthesized motion and the Y axis represents an index of the frames of the motions in the database. $P_1(t)$ and $P_2(t)$ can transition from one motion to another independently of each other as each is just following a path through the motion graph.

user-specified constraints. Because the solution is constrained to a sequence of poses from a motion capture database, the motion graph provides a reduced-space representation of human motion. No unnatural poses will appear in the final motion because each pose appears in the original database. This representation, however, is quite restrictive. For example, it would be impossible to synthesize a motion for sitting down on a medium height chair if the database contains only motions for sitting down on low and high chairs. Because it is impossible to capture all possible variations of even a single behavior this will be a problem for databases of any size.

To relax this restriction, we instead consider a more general discrete representation of

the data. The new motion is an interpolation of two or more motions, where each motion is generated by traversing a path through the motion graph. More formally, each pose of the new motion, $P_{new}(t)$, is represented as an interpolation of k poses from the motion capture database:

$$P_{new}(t) = P_1(t)w_1 + P_2(t)w_2 + \ldots + P_k(t)w_k. \tag{4.1}$$

where $P_i(t)$ are poses from the motion capture database, w_i are weights that interpolate these poses, and any two consecutive poses in each $P_i(t)$ are connected by an edge in the motion graph. In other words, each $P_i(t)$ is a path through the motion graph. We also allow for time scaling of these paths which is often required to synchronize interpolated motions. This representation, therefore, can be viewed as a combination of motion graph and interpolation techniques: the final motion is an interpolation of k time-scaled paths through the motion graph. Figure 4.2 shows an example of interpolating two paths through the motion graph. Weights, $w_1 ... w_k$ can be constant or can be a function of time. A single constant weight has been shown to work well when interpolating motions that consist of only one behavior. In our implementation we want to synthesize motions of multiple behaviors and therefore we allow weights to change with time (see Section 4.3.1 for details).

Because the new motion is computed by interpolating existing poses from the motion capture database, it is likely to be natural looking. In Chapter 5, we analyze interpolated motions for physical correctness and show that interpolation produces motions that are close to physically correct in many cases. These properties make interpolation a good candidate to represent human motions that are close to physically correct.

To find a motion described by equation 4.1 we construct a graph that supports interpolation of motion paths and search it for the user-specified motion. We call this graph an *interpolated motion graph*. To simplify the explanation, we divide equation 4.1 into two cases: (1) $k = 1$—no interpolation; (2) $k > 1$—with interpolation. For $k = 1$ the construction of the search graph is similar to how it was done previously [41, 6, 33, 38, 6, 7]: we first construct a standard motion graph, MG, from the motions in the database; we next construct a full search graph, SG, by unrolling graph MG into the environment. During the unrolling step, each state int in graph MG is augmented with the global position and

orientation of the root causing the size of the graph to grow by the number of possible positions and orientations of the character in the environment. This step is required so that we can search for motions that satisfy user-specified global position constraints and avoid obstacles. For $k > 1$ the process is similar. We first construct a motion graph that supports interpolation which we call IMG—interpolated motion graph. We next construct a full search graph, ISG, by unrolling graph IMG into the environment.

We explain our algorithms first for $k = 1$ using graphs MG and SG and then extend them to $k > 1$ for graphs IMG and ISG. In the actual implementation we do not need to construct graph SG, it is defined here to simplify the explanation. Section 4.2 describes the details of the construction of all four graphs.

We are interested in finding a globally optimal solution or a close approximation to it. In contrast to locally optimal and greedy solutions, globally near-optimal solutions will not get stuck in a bad local minima and will avoid the dithering and inefficient patterns of motion that sub-optimal solutions often have (see experimental results Section 4.4). On the other hand, the global characteristic reduces the size of the search space that can be efficiently searched. Although unrolled graphs SG and ISG are much smaller than the full search space for a character with 60 degrees of freedom, they are still very large and are impossible to search for an optimal or nearly-optimal solution at interactive frame rates (within few minutes). Section 4.3 analyzes the size of these graphs for one example. Even graph SG is too large to be searched efficiently for an optimal or nearly-optimal solution for a database consisting of approximately 10000 poses (motions are sampled into poses at 30 frames per second). All existing approaches in the literature either find a solution using a greedy approach with no guarantee on sub-optimality of the solution or search a manually constructed graph with a small number of states.

In Section 4.3, we present two techniques that allow us to reduce the size of the search space sufficiently that a solution can be found at interactive frame rates—our experiments show that it usually takes between few seconds to few minutes for the discrete optimization to generate a solution. The first technique greatly reduces the size of graphs MG and IMG by culling states and transitions that are redundant and/or clearly sub-optimal and will never be chosen to appear in an optimal solution. The second technique reduces the

portion of graphs SG and ISG that are explored by search. It computes an informative heuristic function that guides the search toward states that are more likely to appear in an optimal solution. All experimental results presented in this thesis are for $k = 1$ (no interpolation) and for $k = 2$ (interpolation of two paths).

In the next section we define the unknowns, constraints and objective function for our optimization problem. In section 4.2, we explain how the graphs are constructed. In Section 4.3, we present the methods for reducing the size of the graphs and searching them efficiently. We conclude with experimental results in Section 4.4.

4.1 Optimization problem setup

We assume that a user provides a sketch of the desired motion. The sketch includes a roughly sketched $2D$ path that the character should follow in the environment and a set of constraints. For example, the user may want the character to start from position A, pick up an object from a table at position B, jump over a river at position C and arrive at position D (Figure 4.3a). This motion has 4 constraints: (1) start at position A; (2) pick an object from the table at position B; (3) jump over river at position C and (4) arrive at position D. If the user does not specify the path, then we compute it automatically as a shortest path in the ground plane that passes through all user-specified constraints and avoids obstacles (Figure 4.3b).

Unknowns: The unknowns of the optimization problem are the variables from equation 4.1. In our implementation k is set either to one (no interpolation) or two (interpolation of two paths). For $k = 1$ path $P_1(t)$ is the only unknown. For $k = 2$ the unknowns are $P_1(t)$, $P_2(t)$, $w_1(t)$. $w_2(t)$ is computed as $1 - w_1(t)$.

Constraints: In our implementation, both a user-specified $2D$ path of the character and other user constraints are implemented as hard constraints and are not part of the optimization function. When the user specifies a path, the root of the character in the generated motion is constrained to stay inside a $2D$ corridor around that path. The corridor is created automatically based on the user-specified path. In our experiments we generally

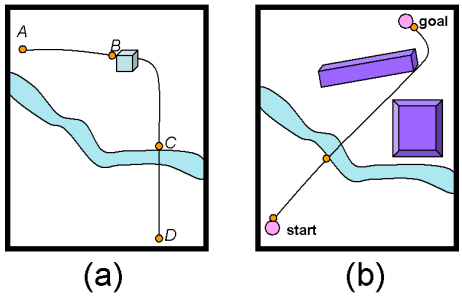

Figure 4.3: (a) User sketch. The character starts at position A, picks up an object from a table at position B, jumps over a river at position C and arrives at position D. (b) User specified only start and end positions. The system automatically creates a sketch of the $2D$ path from start to goal while avoiding obstacles and adds three constraints: (1) start at "start state"; (2) jump over river; (3) arrive at "goal state."

set the width of the corridor to 0.5 meters. While staying inside the corridor, the generated motion must also not violate the environmental constraints. For example, the character must avoid walls whenever they are inside the corridor and jump or step over water and ditches. The motion must also satisfy other user-specified constraints to within a small error tolerance.

Objective function: The objective function for discrete optimization is very similar to the one for continuous optimization and includes a weighted average of two terms. The first term minimizes the sum of squared torques (an approximation of the energy needed to compute the motion). The torques are computed by inverse dynamics. The second term maximizes the sum of probabilities of transitions associated with edges in graphs MG and IMG and thus maximizes the smoothness of the motion.

4.2 Graph Construction

A standard motion graph MG is constructed as a pre-processing step (Sections 4.2.1 and 4.2.2) much as it was done in previous motion graph implementations. We construct graph IMG by extending graph MG to support interpolation. Graph IMG, can also be con-

structed as a pre-processing step for $k \leq 2$. For larger k, this graph will be very large because of the combinatorics of combining k motions, and therefore will need to be constructed at runtime on as needed basis (Sections 4.2.1 and 4.2.2 provide the details). As in the previous work on motion graphs, only the position and orientation of the character *relative* to the previous frame is stored with each state in graphs MG and IMG. To be able to find motions that satisfy user-specified position constraints and avoid obstacles, we unroll graphs MG and IMG into the environment. The corresponding unrolled graphs, SG and ISG, are very large because an individual pose now appears many times in the graph, once for each reachable position and orientation. These graphs, therefore, should be constructed at a runtime on as needed basis. Usually only a small portion of the graph is explored by the search at a runtime before an optimal or near-optimal solution is found (see Section 4.4).

In Sections 4.2.1-4.2.2, we explain what defines a state in each graph and which states are connected by transitions. In Section 4.2.3 we describe the search algorithm we use to search graphs SG and ISG for the desired motion. To simplify the explanation of the construction of graphs IMG and ISG, we assume that k is equal to two in equation 4.1. Extension for $k > 2$ is trivial from an algorithmic point of view but very costly in terms of the size of the graph and the search time required.

We assume that we have a database of motions. Each motion is sampled into an ordered sequence of poses. We use a right handed coordinate system XYZ with the X and Z axes spanning the ground plane and the Y axis pointing up. Each pose is represented by (1) Q, the absolute values of all joint angles plus an absolute rotation of root of the character around the X and Z axes, (2)$P_{vertical}$, the absolute position of the root of the character along the vertical axis, (3)RP_{plane}, the relative position of the root on ground plane and (4) RQ_{yaw}, the relative rotation of the root around the vertical axis. All relative values are computed relative to the previous pose in the motion and the values for the first pose are set to zero.

4.2.1 State definition

Graph MG: Each state in graph MG is defined as $S = (PoseIndex)$, where $PoseIndex$ is an index of the pose in the motion capture database.

Graph IMG: Similarly, each state in graph IMG is defined as $S = (PoseIndex_1, PoseIndex_2, w_1)$, where $PoseIndex_1$ and $PoseIndex_2$ are the indices of the two poses from the motion database to be interpolated and w_1 is the interpolation weight from the equation 4.1 (we assume that weight $w_2 = 1 - w_1$ because $k = 2$). We only want to interpolate poses with the same contact state. Therefore, poses $PoseIndex_1$ and $PoseIndex_2$ in a given state S are required to have the same contacts (for example, both have only the left foot on the ground).

Graph SG: Graph SG is computed by unrolling graph MG into the environment. Therefore, each state in graph SG is a state in graph MG with two additional variables representing the global position and orientation of the character. $S = (PoseIndex, P_{plane}, Q_{yaw})$, where P_{plane} is the global position of the character in the ground plane (X and Z values) and Q_{yaw} is the global orientation of the character about the vertical axis. Global positions P_{plane} and P_{yaw} are computed by integrating the relative ones.

Graph ISG: Similarly to graph SG, graph ISG is computed by unrolling graph IMG into the environment. Therefore, each state in graph ISG is a state in graph IMG with two additional variables representing the global position and orientation of the character. $S = (PoseIndex_1, PoseIndex_2, w_1, P_{plane}, Q_{yaw})$, where P_{plane} is the global position of the character in the ground plane (X and Z values) and Q_{yaw} is the global orientation of the character about the vertical axis. Global positions P_{plane} and P_{yaw} are computed by integrating the interpolated relative ones.

4.2.2 Transitions

Graph MG: Transitions in graph MG are constructed by identifying similar poses in the motions and connecting them with edges in the motion graph as was done by Lee and his colleagues in [38]. They only allowed transitions at poses where the contact of the char-

acter with the environment changed. This drastically reduces the number of transitions in the motion graph and makes the graphs smaller and faster to search. But, this simplification may also remove many good transitions that occur in the middle of contact phases. In contrast, we allow transitions inside of contact phases but use a pruning technique that is lossless for many domains. Section 4.3.1 provides the details. To compute similarity between two states we use the same similarity function as Lee and his colleagues [38]. Two poses are similar if the similarity between them is within an error threshold. We experimentally picked our threshold to have a small perceptual error during transitions. The transitions in graph MG are associated with probabilities: the transitions between more similar poses receive higher probability values, and the transitions between less similar poses receive lower probability values.

We found that a single threshold for picking good transitions often does not work well. A low threshold results in most transitions occurring within a single behavior (walks for example) and very few transitions between motions of different behaviors (walks and jumps for example). A high threshold, on the other hand, results in many low quality transitions within a single behavior even though these transitions are not needed. We allowed higher threshold for transitions between different behaviors and lower threshold for transitions within the same behavior. An automatic method for computing a motion graph with "good" connectivity would be very helpful.

Graph IMG: Given state A defined by $(PoseIndex_1^A, PoseIndex_2^A, w_1^A)$ we need to compute the set of successor states—the states that can be reached from state A via a single transition in the graph IMG. Consider state B defined by $(PoseIndex_1^B, PoseIndex_2^B, w_1^B)$. State B is a successor of state A if and only if $PoseIndex_1^B$ is a successor of $PoseIndex_1^A$ and $PoseIndex_2^B$ is a successor of $PoseIndex_2^A$ in the motion graph MG (Figure 4.4).

We can think of constructing graph IMG for $k = 2$ as taking a "product" of two, identical, motion graphs. Figure 4.5 gives a detailed example of the construction. Graph MG is shown on left. Graph IMG is gradually constructed on the right. In Figure 4.5(a) two copies of graph MG are shown on the left. One state from IMG is shown on the right. It is defined as interpolation of two poses—one from the first MG and one from the second

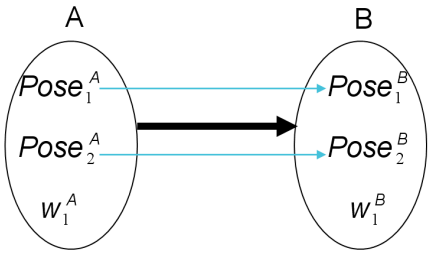

Figure 4.4: States A and B are connected by an edge in graph IMG (thick arrow) only if $Pose_1^A$ is connected to $Pose_1^B$ and $Pose_2^A$ is connected to $Pose_2^B$ in the original motion graph, MG (thin arrows).

MG. In this example state A from IMG is computed by interpolating poses A_1 and A_2. In Figures 4.5(b)-(c) to compute transitions leaving state A we take all possible combination of transitions that leave poses A_1 and A_2 (shown in orange on the left). Figure 4.5(d) shows a segment of graph IMG starting at state A. In this example we assume that the interpolation weight w is constant throughout the graph.

We also need to allow the paths which are being interpolated to be scaled in time so that their contact changes can be synchronized. For example, a short jump and a long jump have different time durations and therefore we need to align their contact changes before interpolation. We defer the discussion of how we do time scaling until Section 4.3.1.

The maximum number of states in graph IMG is N^kW, where N is the number of poses in the motion capture database and W is the number of possible weight values. In our implementation $k = 2$ and therefore we were able to pre-compute and store graph IMG in memory. For larger k, this graph will need to be constructed at a runtime on as needed basis.

Graph SG: The successor states for a state in graph SG are computed by following the transitions in graph MG and computing the global root position and orientation of the character from the relative ones stored in MG.

Graph ISG: The successor states for a state in graph ISG are computed in a similar fashion from graph IMG. The only difference is that we now also need to interpolate the

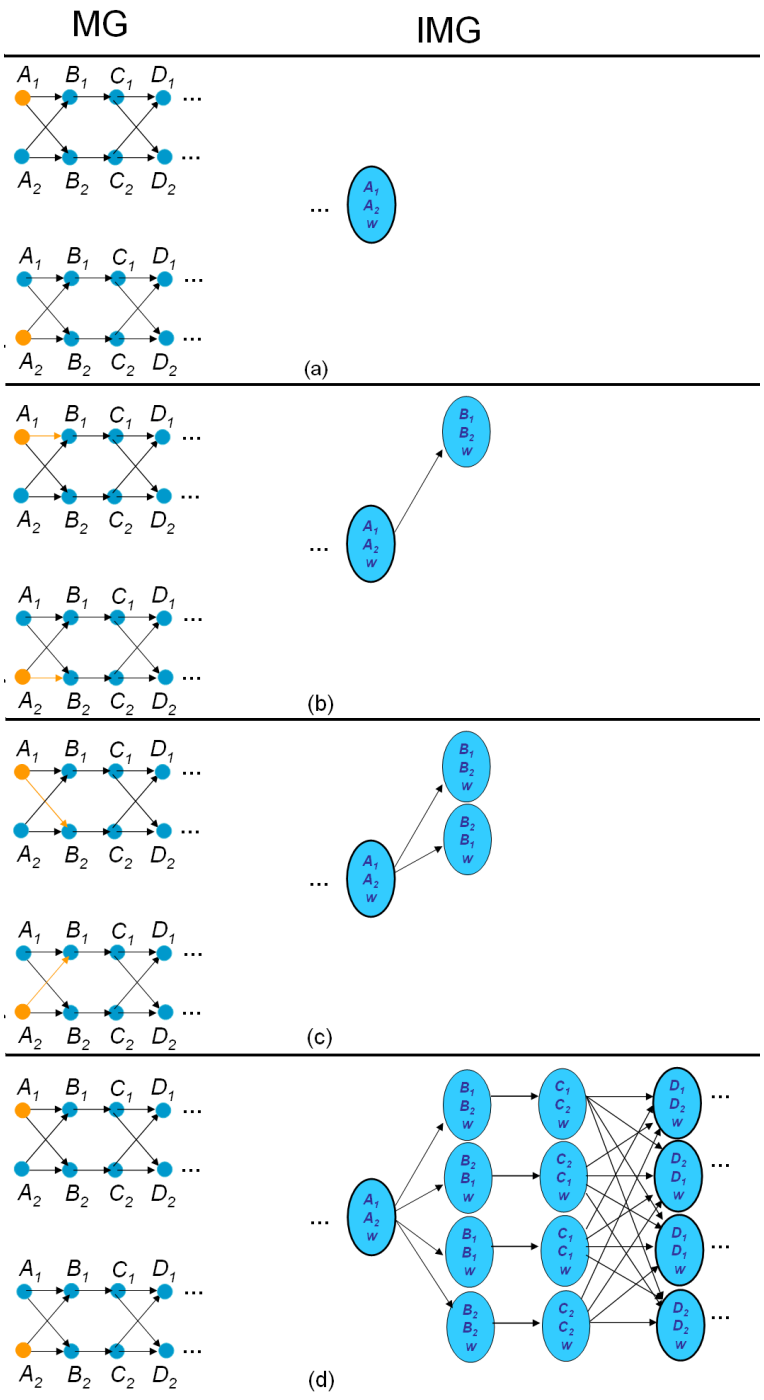

Figure 4.5: Example of construction of graph IMG.

relative positions and orientations to compute the global ones.

In both search graphs, SG and ISG, each transition is associated with a cost. This local cost is derived according to our objective function defined in Section 4.1. It is a weighted average of two terms: the first term minimizes the sum of squared torques (computed using inverse dynamics) and the second term maximizes the sum of probabilities of transitions associated with edges in graphs MG and IMG and thus maximizes the smoothness of the motion.

4.2.3 Search

We use A^* search, and in particular its anytime extension ARA^* [42], to find a sequence of motion graph poses that result in the desired motion. A^* is a well-known and well-studied search algorithm for efficiently finding an optimal solution [50]. For a given graph and heuristics, A^* computes the minimum number of states in the graph to guarantee the optimality of a solution [62]. The algorithm takes a graph as input, where each edge has a strictly positive cost, a start state, s_{start}, and a goal state, s_{goal}. It then searches the graph for a path from s_{start} to s_{goal} that minimizes the cumulative cost of the transitions in the path. A^* uses a problem-specific heuristic function to focus its search on the states that are more likely to appear on the optimal path because they have low estimated cost. For each state s in the graph, the heuristic function must return a non-negative value $h(s)$ that estimates the cost of a path from s to s_{goal}. To guarantee the optimality of the solution and that each state is expanded only once, the heuristic function must satisfy the triangle inequality: for any pair of states s, s' such that s' is a successor of s, $h(s) \leq c(s, s') + h(s')$ and for $s = s_{\text{goal}}$, $h(s) = 0$, where $c(s, s')$ is the cost of a transition between states s and s'. In most cases, if the heuristic function is admissible (i.e., does not overestimate the minimum distance to the goal), the triangle inequality holds.

The heuristic can tremendously speed up the search, especially if it is informative. Moreover, the anytime extension of A^*, ARA^* search [42], trades solution quality for search time by using an inflated heuristic (h-values multiplied by $\epsilon > 1$). The inflated heuristic often results in a speedup of several orders of magnitude but the solution is no

longer guaranteed to be optimal. The cost of the solution, however, is guaranteed to be no more than ϵ times the cost of an optimal solution. ARA^* finds the best solution it can within a specified time window. It starts by finding a suboptimal solution quickly using a large ϵ and then gradually decreases ϵ and recomputes the solution until it runs out of time or finds a provably optimal solution. The algorithm reuses previous search results when ϵ is decreased.

In Sections 4.3.2 and 4.3.3, we describe the heuristic function we use and in the experimental results Section 4.4 we show that this heuristic function together with the technique that reduces the size of the motion graph (see Section 4.3.1) makes it possible to find solutions at interactive frame rates.

4.3 Reducing the search space

Although graphs SG and ISG are much smaller than the full search space for a 60 dof character, they are still very large and are impossible to search at interactive frame rates. For example, suppose we have a database of a few behaviors: walks, jumps, and picking an object. The database has about $N = 10000$ poses (motions are sampled into poses at 30 frames per second). As was explained in Section 4.2.1, each state in the ISG graph is defined by $S = (PoseIndex_1, PoseIndex_2, w_1, P_{plane}, Q_{yaw})$ when $k = 2$. If we discretize w_1 into 10 values, P_{plane} into 1000 by 1000 values and Q_{yaw} into 1000 values, we will have $NumStates = (10000)^2 1000^3 * 10 = 10^{18}$ possible states. Because we constrain the character to stay inside the corridor around the user-specified path, the number of possible states will be smaller. Assuming that only 1% of positions/orientation values will be allowed as a result of this restriction the maximum number of states will be 100 time smaller than before: $NumStates = 10^{16}$. The complexity of A* algorithm is $O(E + SlogS)$, where E is the number of edges in the graph. With $S = 10^{16}$, the graph cannot be searched at interactive frame rates. Even with $k = 1$ (no interpolation) the maximum number of states is $S = 10^{11}$ which is still too large to search for an optimal or close to optimal solution at interactive frame rates. All existing approaches in the literature either find a solution using a greedy approach with no guarantee on sub-optimality of the solution

or search a manually constructed graph with a small number of states.

To address this problem, we developed two techniques that significantly decrease the number of states the search will have to visit to find a solution. The first technique, described in Section 4.3.1, greatly reduces the size of the motion graph MG by culling states and transitions that are redundant and/or clearly sub-optimal and will not appear in an optimal solution. It then builds the reduced graph IMG from the reduced graph MG. The second technique, described in Sections 4.3.2 and 4.3.3, reduces the portion of graphs SG and ISG that are explored by search by computing an informative heuristic function that guides the search toward the states that are more likely to appear in an optimal solution. We describe this technique first for graph SG in Section 4.3.2 and then extend it to graph ISG in Section 4.3.3. In the experimental section we show that the combination of these techniques makes it possible to find an optimal or a close to an optimal solution for a reasonable size database at interactive frame rates for $k = 1$ and $k = 2$.

4.3.1 Culling unnecessary states and transitions

Culling of states and transitions is easier to explain if we first describe a hypothetical graph MG^* which could be constructed from graph MG. The graph MG^* is fully equivalent to graph MG, meaning that every path in MG is a valid path in MG^* and vice versa but MG^* contains only selected states from MG and all other states become part of the transitions in between those selected states in MG^*. Formally, the states and transitions in graph MG^* are defined as follows:

- **States in graph** MG^*: We define a "contact change" state as a state that starts a new contact phase in the motion graph (states A, B, C and D in Figure 4.6). The states in MG^* are the "contact change" states in graph MG.

  To determine the "contact change" states we process all motions in the motion capture database by separating them into phases based on contact with environment (see Figure 4.7 for an example). We use the same technique as Lee and his colleagues in [38] to detect contacts.

- **Transitions in graph MG^*:** A transition between states S_1 and S_2 in graph MG^* is a path between corresponding states in graph MG such that all states on this path except for S_2 have the same contact. We call such paths "single contact" paths. There can be multiple "single contact" paths in graph MG that connect two "contact change" states (see states A and D in Figure 4.6(a) for example). There can even be infinitely many "single contact" paths between a pair of states if the graph has cycles. Consequently, there can be infinitely many transitions between a pair of states in graph MG^*. Figure 4.6(b) shows an example of graph MG^*.

In the next section, we will explain how to cull redundant transitions from graph MG^* and then explain how it can be done directly in graph MG without explicitly constructing graph MG^*. Then we explain how to cull redundant states from MG^*, also without explicitly constructing it. After culling both states and transitions, we obtain a much more compact version of graph MG—$MG_{compact}$. In the last section we compute a compact version of graph IMG from graph $MG_{compact}$.

Culling transitions: Graph MG^* can have infinitely many transitions between any two "contact change" states. Only one of these transitions is optimal with respect to our optimization function, however, and the rest can be culled without affecting the optimality of the solution in most cases. We pre-compute just one optimal "single contact" path between any two "contact change" states because when unrolled into the environment, all "single contact" paths between a pair of states end with the character at exactly the same place in the environment. For "single contact" paths between states S_1 and S_2, the global position and orientation of the root of the character at state S_2 is solely defined by the global position and orientation of the root at state S_1 and is independent of the actual path from S_1 to S_2. The point of contact with the environment for state S_2 is defined by state S_1 (because we are considering "single contact" paths), whereas all the joint angles are defined by the state S_2 (see Figure 4.8(a)). The contact position and orientation and the values of the joint angles uniquely determine the global position and orientation of the root of the character for state S_2.

We consider a flight phase also as a contact phase (with zero contacts). All in flight "single contact" paths between two states, S_1—lift-off and S_2—landing, also end at ex-

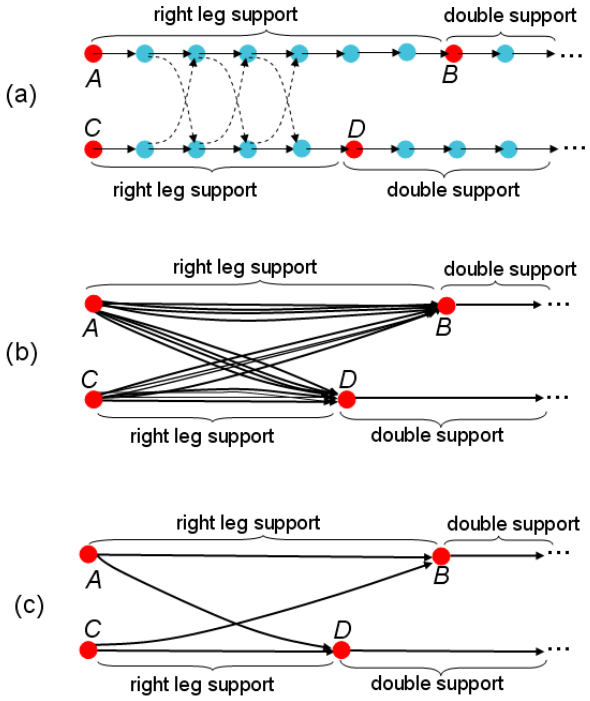

Figure 4.6: (a) Motion graph MG. States A, B, C and D (shown in red) are "contact change" states. The character enters and exits a contact phase through a "contact change". For example, if the character enters state A (that initiates a right leg support phase) it can exit only through state B or state D. (b) A hypothetical graph MG^* constructed from graph MG. (c) Compact motion graph $MG_{compact}$ for the motion graph shown in (a). It only contains "contact change" states. A transition between any two states in graph $MG_{compact}$ represents an optimal path in graph MG.

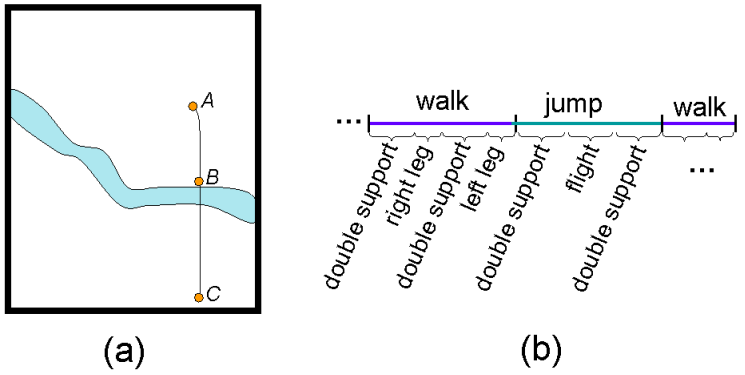

Figure 4.7: (a) A character leaves from point A and arrives at point C after jumping over a river at point B; (b) The motion can be split into phases based on contact information.

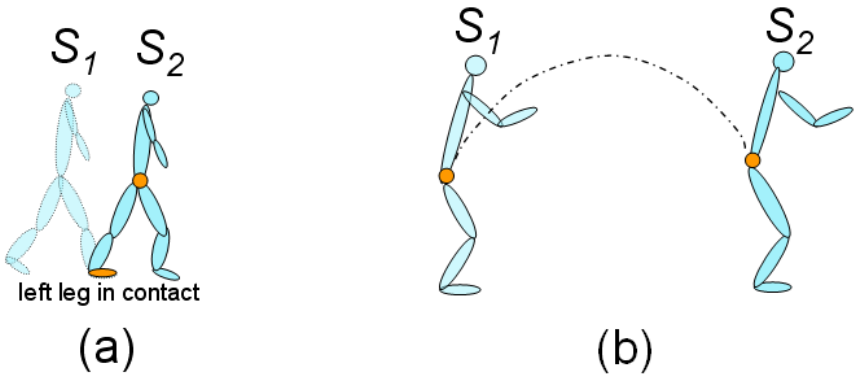

Figure 4.8: (a) For "single contact" paths between a pair of states S_1 and S_2, the global position and orientation of the root of the character at state S_2 is uniquely determined by contact position and orientation at state S_1 and the values of the joint angles at state S_2. (b) The position of the center of mass at landing (state S_2) is uniquely defined by the intersection of the flight trajectory and the center of mass of the character at state S_2. The trajectory of the center of mass for the root of the character is defined by the lift-off velocity from state S_1.

actly the same place in the environment. The trajectory of the center of mass for the root of the character is defined by the lift-off velocity from state S_1. The position of the center of mass at landing (state S_2) is uniquely defined by the intersection of the flight trajectory and the center of mass of the character in state S_2 (see Figure 4.8(b)).

Because all unrolled, "single contact" paths between two states S_1 and S_2 end with the character at exactly the same place with respect to state S_1, we can just compute one optimal "single contact path" from S_1 to S_2 prior to runtime and cull the other less optimal paths. We call the graph that remains after culling $MG_{compact}$. $MG_{compact}$ has the same states as MG^* but any two states in graph $MG_{compact}$ are connected by at most one transition.

$MG_{compact}$ can be computed directly from MG by searching for an optimal "single contact" path between each pair of "contact change" states in graph MG. The transition is not created if such a path does not exist. Figure 4.6(c) shows an example of a compact motion graph.

Our assumption that $MG_{compact}$ is equivalent to the full MG will fail if the user specification requires controlling the details of the motion inside a "single contact" path. This situation might arise because of environmental constraints, user constraints or an optimization function.

The optimal "single contact" path between two states may violate environmental constraints when another less optimal path between the same two states might not (Figure 4.9). We seldom saw this problem in our experiments probably because if the optimal path violates environmental constraints then all other paths between the two contact states are also likely to. The start and end states of the optimal path differ by a single change in contact and so the root positions of the two states are quite close (except for flight phase). Neither state violates the environmental constraints (or this part of the graph would not be searched at run-time), so it is unlikely that while the optimal path violates a constraint, another path between the two states does not. If this problem were to occur in some domain, the search could switch from the compact motion graph to the original one if it detects that the optimal path violated constraints.

A user may also specify a constraint on a frame of the motion that is inside a "single contact" path. Specifying a wave while standing would be an example of such a motion. In our work, and in other previous work, user constraints generally change the contact information (picking a cup or stepping onto a stone for example) and therefore occur at "contact change" states. If the constraint does occur inside a "single contact" phase, the search can switch to the original motion graph in the vicinity of that constraint. Alternatively, $MG_{compact}$ could be constructed with actions other than contact changes as the start and end points of "single contact" paths.

The optimization function we use in this thesis (see Section 4.1) allows us to compute optimal paths prior to runtime because it is independent of the particular problem the user specifies. Many functions have this property: minimizing energy (while satisfying user constraints), minimizing sum of squared accelerations, maximizing smoothness, minimizing the total distance traversed from start to goal, minimizing total time of the motion, satisfying specified annotations (for example behaviors or styles as in [7]) and many other functions. If the optimization function depends on what the user specifies, however, we then either need to have that function during pre-computation or approximate it with another function that is dependent on the user problem only at "contact change" states. For example, instead of minimizing the distance between every frame of the motion and the user-specified curve, we can minimize the distance only between the "contact change" states and the curve.

Therefore, for many problem specifications we can pre-compute optimal paths between "contact change" states for a given motion graph. This computation is efficient because it occurs in graph MG which is much smaller than the unrolled graph SG. In experimental Section 4.4 we show that graph $MG_{compact}$ has many fewer states and transitions than graph MG and therefore is much faster to search.

This preprocessing step is not the same as just removing all the transitions from the motion graph except for the ones in between "contact change" states, as others have done [38]. With that approach there would be no path between states A and D in Figure 4.6(a) even though one exists in the original motion graph. The preprocessing presented here retains many more transitions which is important for finding transitions between different behav-

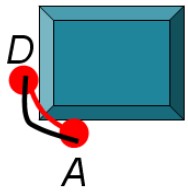

Figure 4.9: States, A and D, do not intersect the obstacle. An optimal path (shown in red) intersects the obstacle. Another path from A to D (shown in black) does not intersect the obstacle. This case is unlikely because states A and D share a common foot contact and therefore their root positions are close to each other. From our experiments this practically never happens.

iors such as a walk and a jump.

Culling states: A lot of redundant data is usually included in a motion graph, in order to capture natural transitions between behaviors. For example, to include natural transitions between walking and jumping, we had to include many similar walking segments leading to those transitions to jumping. As a result, each state in the compact motion graph may have many outgoing transitions that are very similar. For example, a state A in Figure 4.10 has three successors: S_1, S_2 and S_3, that are similar to each other. Because the states are similar, all three transitions will end at the approximately the same position in the world when graph $MG_{compact}$ is unrolled into the environment to create graph SG. We can cull the redundant states by merging all three states S_1, S_2 and S_3 into one and keeping only the lowest cost transition.

When two or more states are merged to form a new state, we define successors of that state to be the union of the successors of the merged states. Similarly, we define the predecessors to be the union of the predecessors of the merged states. After we merge all possible states we will have redundant transitions between many of them. For example in Figure 4.10, transitions t_1, t_2 and t_3 become redundant after states S_1, S_2 and S_3 are merged. We keep just the lowest cost transition. We merge states in the order of their similarity: the most similar states are merged first and less similar states that still fall

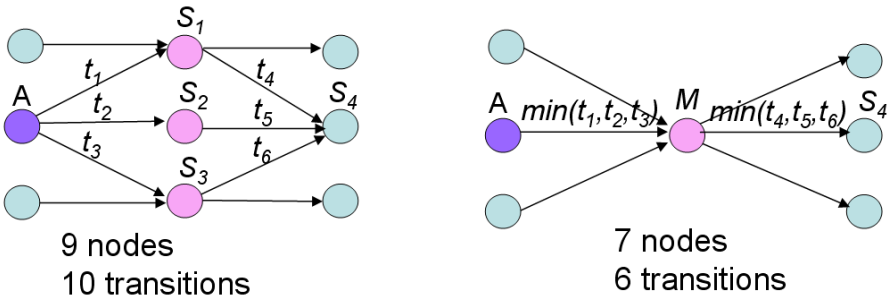

Figure 4.10: (a) States S_1, S_2 and S_3 are similar to each other. As a result, optimal transitions t_1, t_2 and t_3 are also very similar and all end with character at approximately the same position. (b) We can remove this redundancy if we merge similar states S_1, S_2 and S_3 into one state M and only keep the lowest cost transition.

below the threshold for the similarity error are merged later.

The similarity threshold for merging could be the same value that we used to identify similar states while constructing the motion graph, but we observed that that threshold left too many similar transitions in the graph. We therefore increased the threshold by a factor of 1.5 to 2. During motion graph construction, the threshold must be low enough that we do not introduce any perceptible errors in the transitions between states. For merging, however, a higher threshold is appropriate because it just removes flexibility in the motion graph without introducing perceptible error.

Building graph $IMG_{compact}$: The graph $IMG_{compact}$ can be computed from graph $MG_{compact}$ in almost the same way graph IMG was computed from graph MG. Each state in graph $IMG_{compact}$ is defined as $S = (A, C, w_1)$, where A and C are the indices of two poses in $MG_{compact}$ with the same contact. w_1 is the interpolation weight from equation 4.1. State $S_2 = (B, D, w_2)$ is a successor of state $S_1 = (A, C, w_1)$ if and only if pose B is a successor of pose A and pose D is a successor of pose D in the graph $MG_{compact}$. We also require that the interpolation weight changes gradually. So w_1 must be close to w_2 (we discretize the weight into 10 values from 0 to 1). Figure 4.11 shows an example of this graph.

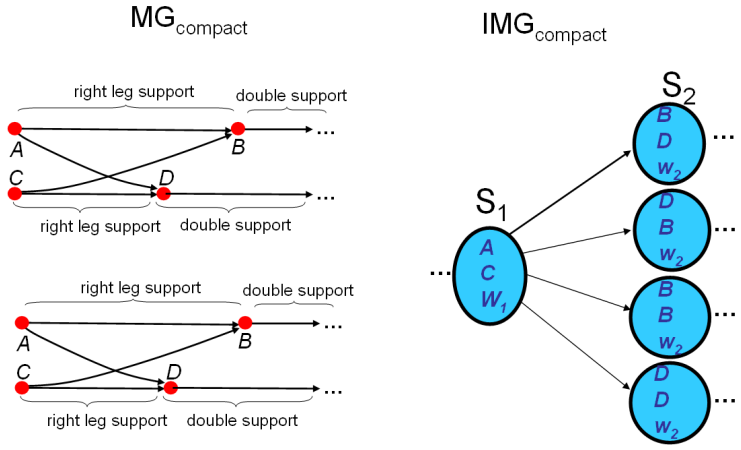

Figure 4.11: Two identical versions of graph $MG_{compact}$ are shown on the left. A small segment of graph $IMG_{compact}$ starting from the state $S_1 = (A, C, w_1)$ is shown on the right.

The transition between any two states S_1 and S_2 in graph $IMG_{compact}$ is actually a sequence of poses, where each pose is an interpolation of corresponding poses in the transitions from A to B and from C to D. Figure 4.12 shows an example. The interpolation uses a constant weight w_1 throughout the transition. The durations of the transitions from A to B and from C to D may differ. We assume a uniform time scaling with the time of the interpolated segment computed according to the following equation:

$$T = \sqrt{T_1^2 w + T_2^2 (1 - w)} \tag{4.2}$$

T_1 and T_2 are the time durations of the first and second segments being interpolated.

We have also made two other changes to the standard interpolation scheme: (1) during flight we interpolate the center of mass positions (instead of the root positions) and all the joint angles; (2) during ground contact we interpolate the positions of the feet, the center of mass positions and all non-redundant degrees of freedom to prevent the feet from sliding on the ground. Chapter 5 describes it in more details.

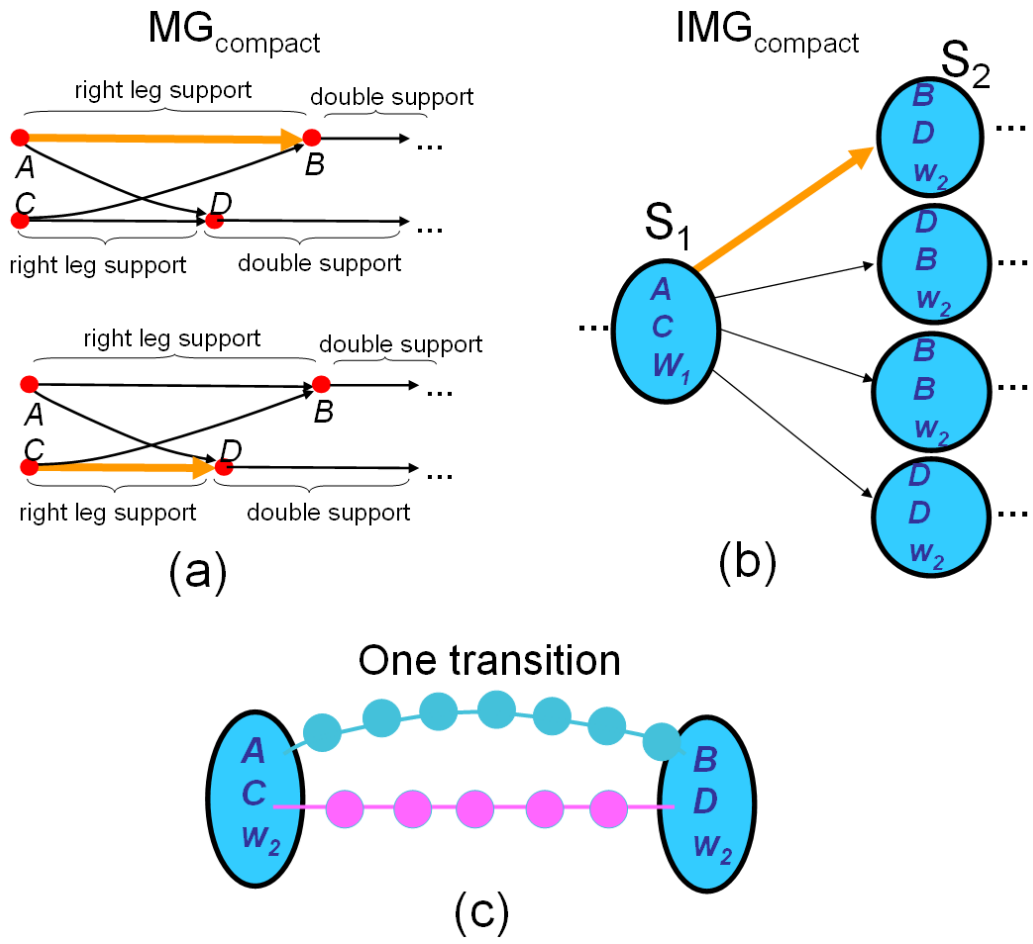

Figure 4.12: (a)Two identical versions of graph $MG_{compact}$. (b) $IMG_{compact}$. The transition between states S_1 and S_2 in graph $IMG_{compact}$ is an interpolation of a path from A to B and a path from B to D in graph $MG_{compact}$. These two paths are shown by thick arrows in (a). (c) Shows these two paths in more detail. Circles represent frames. Because the paths can be of different length we need to scale them in time.

Uniform time scaling and the interpolation scheme described above, will ensure that many transitions will be physically correct. As will be shown in Chapter 5, interpolation of two physically valid motion segments often produces a motion segment that is close to physically correct. We can also use the results from our analysis in Chapter 5 to quickly check whether the interpolation of the two motion segments that correspond to the transitions from A to B and from C to D will produce a physically valid motion. The transition from S_1 to S_2 is removed from graph $IMG_{compact}$ if it is found to be physically incorrect. Instead of using the analysis in Chapter 5 to check each transition for physical correctness, we can also just use inverse dynamics to check that the transition approximately satisfies all the necessary physical properties. The main benefit of the analysis is that it ensures that most of the transitions will generally be found physically correct.

After $IMG_{compact}$ is constructed we can also make a pass over its states and check for significant discontinuities in joint angles and joint angle velocities and root velocities. These discontinuities could occur due to slightly different interpolation schemes during flight and contact phases, substantial changes in time scaling between incoming and outgoing transitions, the interpolation of motions that involve a simultaneous translation and rotation, but of a different degree (described in Chapter 5). In practice, however, the last case can be prevented by not allowing for such transitions on the first place, while other cases do not seem to produce a visually perceivable error.

4.3.2 Deriving informative lower bounds (heuristics) for graph MG

The techniques we described in the previous sections substantially decrease the size of the motion graph MG. For example, in our experiments, the size decreases from $12,000$ nodes and $250,000$ transitions in graph MG to 300 nodes and $60,000$ transitions in graph $MG_{compact}$. But the search space is still very large because we unroll this motion graph into the environment in order to satisfy user-specified position constraints and avoid obstacles.

We use an A^* search algorithm to find an optimal path in the graph SG. The number of states that an A^* search explores depends on the quality of the heuristic function—the lower bounds on cost-to-goal values. Informative lower bounds can drastically reduce the

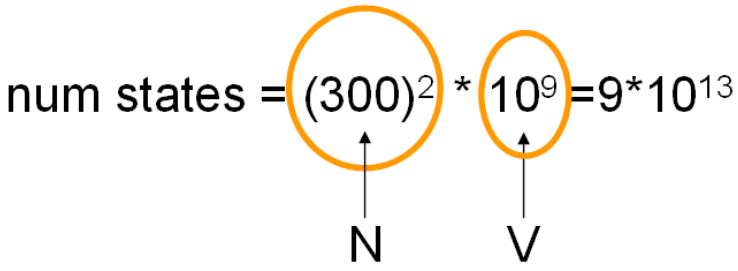

Figure 4.13: This figure shows maximum size of the full search graph, ISG (we assume a constant weight w in this example, otherwise the size would be multiplied by the number of weight values). It is the product of the number of states in the motion graph $MG_{compact}$ ($N = 300$ in this example) and the number of possible positions and orientations of the character in the world. If we discretize root position in ground plane (XZ) and orientation about vertical axis (Y) into 1000 values we have $V = 1000^3 = 10^9$ possible values.

search space exploration. In this section, we present a method for computing such bounds and in the experimental results (Section 4.4) we show that this heuristic function usually speeds up the search by several orders of magnitude and is often the determining factor for whether a solution can be found. We first show how to compute the heuristic for graph MG and then extend it to graph IMG.

The maximum size of the unrolled graph, SG, is defined by the product of two numbers: (1) the number of states in the motion graph $MG_{compact}$—N and (2) the number of possible positions and orientations of the character in the world—V. N is approximately 100 to a 1000 and V is approximately 10^9. Individually N and V are not too large, but the product of the two is. For interpolated motion graph, ISG, the product becomes even larger because the number of states in graph $IMG_{compact}$ is on the order of N^2 (see Figure 4.13). This motivates our heuristic function. We find one heuristic function based only on the character's location in the world and the second heuristic function based only on the motion graph and then combine the two to obtain an informative heuristic function.

Each state in graph SG is defined as $S = (PoseIndex, P_{plane}, Q_{yaw})$, where $PoseIndex$ is an index of the pose in the motion capture database, P_{plane} is the global

position of the character in the ground plane (X and Z values) and Q_{yaw} is the global orientation of the character about the vertical axis. The heuristic function is an estimate of the cost of getting to the goal state G from state $S = (PoseIndex, P_{plane}, Q_{yaw})$. Our first heuristic function, H_{2D}, is based only on the character location in the world (P_{plane}). It ignores other parameters of the state—$PoseIndex$ and Q_{yaw}. If the character could walk along any path, then H_{2D} would provide a perfect heuristic. However, what the character can do depends not only on the character's position in the world, P_{plane}, but also on the character's state in the motion graph, $PoseIndex$, and its orientation, Q_{yaw}. For example, if on the way from state S to the goal the character needs to jump across an obstacle and it is very difficult to reach a jumping motion from state S, then the cost-to-goal at state S should be very large. H_{2D} will grossly underestimates this cost and the A^* search would needlessly explore this part of the space. The second heuristic function, H_{mg}, addresses this problem by computing a cost based on $PoseIndex$ while ignoring P_{plane} and Q_{yaw}. H_{mg} takes into account the capabilities of the character that are encoded in the motion graph.

Computing these heuristic functions involves searching a much smaller space than the full search space. H_{mg} can be pre-computed for a given motion graph as it does not depend on the specific user problem. H_{2D} depends on the user sketch and therefore cannot be pre-computed. However, it is fast to compute because it only requires search in a $2D$ space and it needs to be computed only once before the full search starts. We prove that the heuristic function that results from combining H_{2D} and H_{mg} satisfies the required admissibility and consistency properties at the end of this section. We now describe how to compute H_{2D} and H_{mg} and how to combine the two.

The heuristic function based on the character location in the world (H_{2D}): $H_{\mathrm{2D}}(S, G)$ is an estimate of the cost of getting to the goal state G from state $S = (PoseIndex, P_{plane}, Q_{yaw})$ based only on P_{plane} information, the environment constraints and the user-specified $2D$ path of the character. When computing $H_{\mathrm{2D}}(S)$, we optimistically assume that the path to the goal is independent of $PoseIndex$ and Q_{yaw}.

We first compute the distance from the location of the character at state S to the goal location, $d(P_{plane})$. This distance is computed as the shortest path in $2D$ from $P_{plane} = (x, z)$

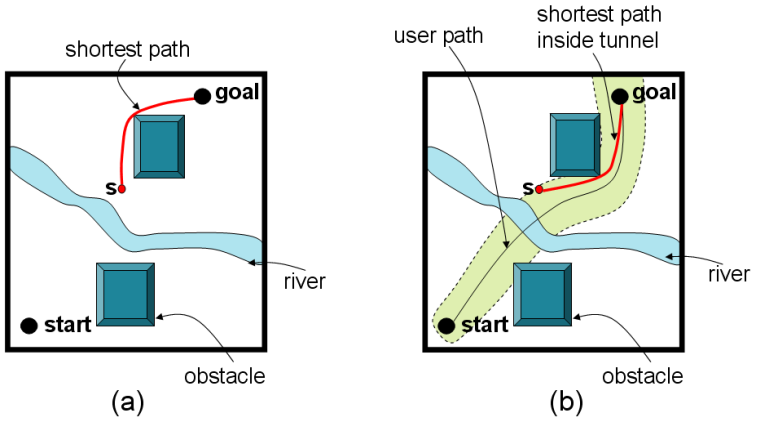

Figure 4.14: (a) $H_{2D}(S, G)$ is the shortest path from the position of the character at state S to the goal. (b) If the user wants the character to stay close to a specified path, the shortest path is constrained to stay inside the tunnel.

(the coordinates of the character at state S) to the (x, z) coordinates of the goal taking into account obstacles in the environment (Figure 4.14(a)). We use a single $2D$ Dijkstra's search to obtain such path from every cell to the goal right before the full search starts. In our implementation, when computing the H_{2D} function, the environment is discretized into 0.2 by 0.2 meter cells. During the full search, the root position of the character was discretized into 0.02 by 0.02 meter cells. If we want the character to stay close to a user-specified path, we define a corridor around the path and the distance is computed based only on the x, z cells that fall within the corridor (Figure 4.14(b)). $H_{2D}(S, G)$ is computed as the product of $d(P_{plane})$ and $c_{\min}$, where $c_{\min}$ is the minimum cost of travelling one unit of distance. For example, when the minimization function is the sum of squared accelerations, we estimate $c_{\min}$ as the minimum acceleration required to traverse one meter based on the motion graph data.

The heuristic function based on motion graph state (H_{mg}): The user provides a set of constraints. For example, the user may want the character to start from position A, pick up an object from a table at position B, jump over a river at position C and arrive at position D (Figure 4.15). This motion has four constraints: (1) start at position A; (2) pick an object from the table at position B; (3) jump over river at position C and (4) arrive

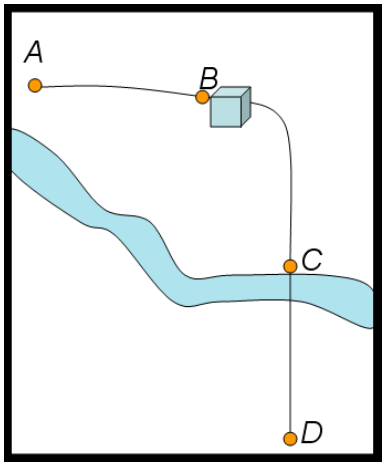

Figure 4.15: User sketch. The character starts at position A, picks up an object from a table at position B, jumps over a river at position C and arrives at position D.

at position D. For each of these constraints, we compute the minimum cost of passing it based on the available paths in the motion graph and use it as the heuristic H_{mg}.

We currently support two types of constraints: picking and jumping. The framework we use for these constraints should generalize to other types of constraints such as kicking, sitting on a chair, and stepping onto obstacles of different height. For each type of the constraint $H_{\text{mg}}(S, C)$ is computed as the minimum cost of getting to the pose in the motion graph that satisfies constraint C from the state S:

- Picking: For the picking constraint, $H_{\text{mg}}(S, C)$ represents the minimal cost of a path in the motion graph from state S to a state that represents the picking up of an object. We can automatically identify poses in the motion graph that represent picking up an object based on contact information. Each "picking" pose is defined by two parameters: *height* and *reach* (Figure 4.16). *Height* defines how high the object is located with respect to the ground. *Reach* defines how far the character reaches out to pick up the object (distance between the root and the hand projected onto ground).

  For each state in the motion graph, MG, we compute a table (Figure 4.17). Each

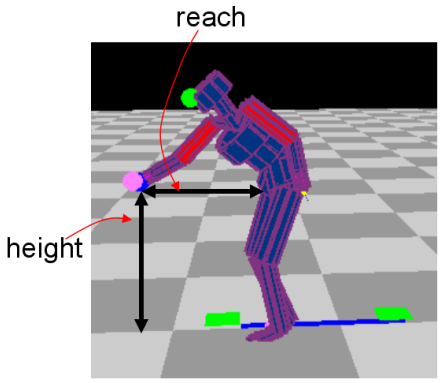

Figure 4.16: We identify states where objects are picked up in the motion graph. Each such pose is parameterized by two parameters: *height* and *reach*.

entry in the table represents the minimal cost of getting to a "picking" pose where *height* and *reach* are in the given range. This pre-computation is independent of the user-specified problem and therefore needs to be computed only once for a given motion graph. For each entry in the table and each state in MG, we search graph MG to compute the cost for that entry. MG is quite small and therefore this pre-computation is reasonably fast.

- Jumping: For the jumping constraint, $H_{\text{mg}}(S, C)$ represents the minimal cost of a path in the motion graph from state S to any state that initiates a flight phase. As with picking, we can use contact info to automatically identify poses in the motion graph that start a flight phase. Each such "jumping" pose is defined by two parameters: *height* and *length*, where *height* defines the highest point of the flight phase of the jump with respect to the ground and *length* defines the distance traveled during the flight phase of the jump. For each state in the motion graph we compute a table similar to the one for picking (Figure 4.17), where each entry in the table represents the minimal cost of getting to a "jumping" pose with *height* and *length* parameters in the given range.

Combining the two heuristics: While $H_{\text{2D}}(S, G)$ is an estimate of the cost of getting to the goal, $H_{\text{mg}}(S, C)$ is an estimate of the cost of satisfying a particular constraint. Because

state S_i

height \ reach	0.2-0.4 meters	0.4-0.6 meters	0.8-1.0 meters	1.0-1.2 meters
0.2-0.4 meters	cost	cost	cost	cost
0.4-0.6 meters	cost	cost	cost	cost
0.8-1.0 meters	cost	cost	cost	cost
1.0-1.2 meters	cost	cost	cost	cost

Figure 4.17: For each state in the motion graph we pre-compute this table. Each entry in the table represents the minimal cost of getting to a "picking" state where the height and reach parameters are within the given range.

the goal is usually much further away from state S than the constraint, $H_{\text{2D}}(S, G)$ will usually be much larger than $H_{\text{mg}}(S, C)$ (Figure 4.18). To get a tighter estimate on the cost, we estimate the cost of satisfying the next constraint and then getting to the goal from the constraint location: $H_{\text{mg}}(S, C) + H_{\text{2D}}(C, G)$, where $H_{\text{2D}}(C, G)$ is the cost of getting to the goal from the constraint location.

Just as in the example in Figure 4.15, there may be multiple constraints. Suppose there are n constraints that still remain to be satisfied at state S. For each of the remaining constraints c_i we can then compute the heuristic as $H_{\text{mg}}(S, C_i) + H_{\text{2d}}(C_i, G)$. We thus obtain n heuristic values based on the motion graph. In addition, we have an H_{2d}-heuristic for state S, the cost of going directly to the goal. These heuristic values can be combined together via a single $\max$ operator to select the most informative heuristic value:

$$H(S) = \max(H_{\text{2d}}(S, G), H_{\text{mg}}(S, C_1) + H_{\text{2d}}(C_1, G), ..., H_{\text{mg}}(S, C_n) + H_{\text{2d}}(C_n, G)) \tag{4.3}$$

Figure 4.19 shows an example of heuristic function for a problem where the character needs to get from A to D along the sketched path, pick up a cup at point B and jump over

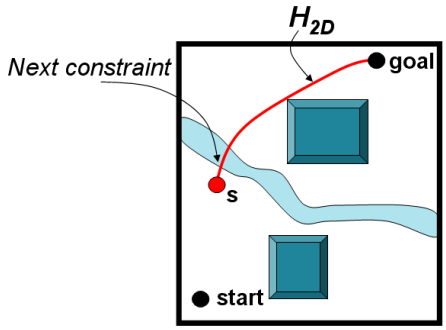

Figure 4.18: H_{2D} will often be much larger than H_{mg} because it estimates the cost of getting all the way to the goal. We need to add another term to H_{mg} that estimates the cost of getting from constraint location to the goal.

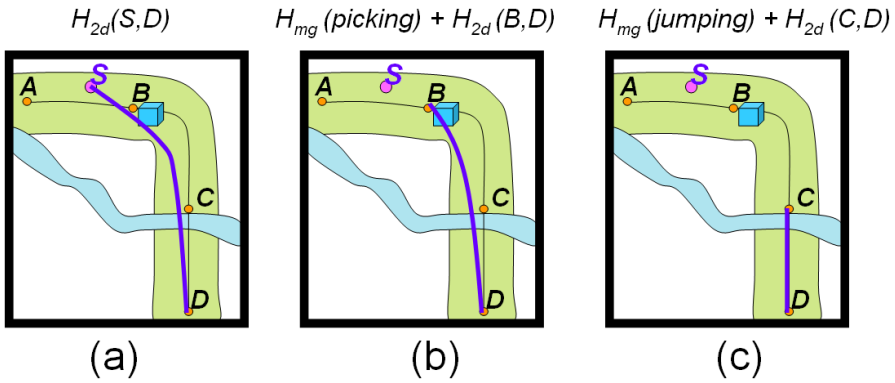

Figure 4.19: Heuristic function for state S. The character needs to get from A to D along the sketched path, pick up a cup at point B and jump over river at point C. The three heuristic functions shown in the figure are computed—(a),(b) and (c)—and combined together using max operator.

river at point C. Three heuristic functions shown in the figure are computed and combined together using max operator.

Proof that the heuristic function is correct: It can be shown that the resulting heuristic function satisfies the consistency requirement (i.e., the triangle inequality). This property is required to guarantee the optimality of the solution returned by A^*. The proof that h-values are consistent takes an arbitrary state S and its arbitrary successor state S' and shows that $H(S) \leq c(S, S') + H(S')$. Let us define state S as $(PoseIndex, P_{plane}, Q_{yaw})$ and state S' as $(PoseIndex', P'_{plane}, Q'_{yaw})$. The proof proceeds as follows.

First, let us introduce the function $H^i_{mg}(S)$ defined as $H_{\mathrm{mg}}(S, C_i) + H_{\mathrm{2d}}(C_i, G)$ if the constraint C_i has not been satisfied at S yet and as $H_{\mathrm{2d}}(S, G)$ otherwise. It should be clear that $H(S) = \max(H_{\mathrm{2d}}(S, G), H^1_{mg}(S), \ldots, H^n_{mg}(S))$.

For the sake of a simpler proof we will now prove the consistency using the latter definition of $H(S)$ function.

We first note that the maximum of two or more consistent heuristic functions is also a consistent heuristic function (see **??** for example). We therefore only need to prove that $H_{\mathrm{2d}}(S, G)$ and $H^i_{\mathrm{mg}}(S)$ for any constraint i are consistent functions.

Let us first prove that $H_{\mathrm{2d}}(S, G)$ is a consistent function. We need to show that $H_{\mathrm{2d}}(S, G) \leq c(S, S') + H_{\mathrm{2d}}(S', G)$. According to the definition of H_{2d} function: $H_{\mathrm{2d}}(S, G) = d(P_{plane}) * c_{\min}$ and $H_{\mathrm{2d}}(S', G) = d(P'_{plane}) * c_{\min}$. We thus need to show that $d(P_{plane}) * c_{\min} \leq c(S, S') + d(P'_{plane}) * c_{\min}$. Let δ denote the distance between P_{plane} and P'_{plane}. Then $c(S, S') \leq \delta * c_{\min}$. We thus need to show that $d(P_{plane}) * c_{\min} \leq \delta * c_{\min} + d(P'_{plane}) * c_{\min}$. Dividing the inequality by $c_{\min}$ we obtain $d(P_{plane}) \leq \delta + d(P'_{plane})$. This inequality holds because the shortest distance to the goal from P_{plane} is at most the the length of the path from P_{plane} to the goal via the point P'_{plane}.

Let us now prove that $H^i_{\mathrm{mg}}(S)$ is also a consistent function. We need to show that $H^i_{\mathrm{mg}}(S) \leq c(S, S') + H^i_{\mathrm{mg}}(S')$. Suppose first the constraint C_i has already been satisfied at S. Then the inequality reduces to $H_{\mathrm{2d}}(S, G) \leq c(S, S') + H_{\mathrm{2d}}(S', G)$ and we have already shown it to hold.

Suppose now the constraint C_i has not been satisfied at S' (and consequently at S). We then need to show that $H_{\text{mg}}(S, C_i) + H_{\text{2d}}(C_i, G) \leq c(S, S') + H_{\text{mg}}(S', C_i) + H_{\text{2d}}(C_i, G)$. Let δ denote the cost of getting from $PoseIndex$ to $PoseIndex'$ in the graph MG. The least cost of satisfying the constraint C_i from $PoseIndex$, given by $H_{\text{mg}}(S, C_i)$, is at most the cost of satisfying this constraint from $PoseIndex$ while passing $PoseIndex'$ on the way. That is, $H_{\text{mg}}(S, C_i) \leq \delta + H_{\text{mg}}(S', C_i)$. Since $\delta \leq c(S, S')$, it then follows that $H_{\text{mg}}(S, C_i) \leq c(S, S') + H_{\text{mg}}(S', C_i)$. After adding $H_{\text{2d}}(C_i, G)$ on both sides this is the desired inequality.

Let us now consider the last situation: when the constraint is satisfied at S', while it is not at S. Then we need to show that $H_{\text{mg}}(S, C_i) + H_{\text{2d}}(C_i, G) \leq c(S, S') + H_{\text{2d}}(S', G)$. In order to satisfy the constraint at state S', the character must be at the required location. Thus, $H_{\text{2d}}(S', G) = H_{\text{2d}}(C_i, G)$. Subtracting this term on both sides of inequality, we get $H_{\text{mg}}(S, C_i) \leq c(S, S')$. And this inequality is guaranteed to hold because $H_{\text{mg}}(S, C_i)$ is the least cost of satisfying the constraint from $PoseIndex$ in graph MG, while $c(S, S')$ is the cost of satisfying the constraint while taking into account other variables in S in addition.

4.3.3 Deriving informative lower bounds (heuristics) for graph IMG

In this section we show how to extend the heuristic function described in the previous section to graph ISG. In graph ISG, $S = (PoseIndex_1, PoseIndex_2, P_{plane}, Q_{yaw}, w_1)$, where $PoseIndex_1$ and $PoseIndex_2$ are two poses of the character being interpolated, P_{plane} is the the character's global root position in the XZ plane, Q_{yaw} is the the character's global root orientation around Y axis and w_1 is the interpolation weight. As with graph SG, we find one heuristic function, H_{2D}, based only on the character location in the world and a second heuristic function, H_{mg}, based only on the motion graph, IMG.

The computation of the H_{2D} heuristic is exactly the same as for graph SG because we store the interpolated root position, P_{plane}, for each state in graph ISG. The computation of H_{mg} is also very similar except that it is now performed on graph IMG instead of MG. We describe how to compute H_{mg} for the picking constraint. The heuristic functions for

jumping constraint can be computed similarly.

- Picking: We first identify each state in the graph IMG that represents picking up an object, a "picking" state, $p = (PoseIndex_1, PoseIndex_2, w_1)$ where both poses, $PoseIndex_1$ and $PoseIndex_2$ are states where an object can be picked up. This state can achieve a *height* and *reach* value that is an interpolation of poses $PoseIndex_1$ and $PoseIndex_2$ with weight w_1. For each state in graph IMG, we then compute a table as for graph MG (Figure 4.17).

  This pre-computation is independent of the user-specified problem and therefore can be computed once for a given graph IMG. To compute this table, we need to search graph IMG. For $k = 2$ the pre-computation is relatively fast but the computations and memory required for this table grows exponentially as k grows. For $k = 2$ the table fits into the memory. For larger k, it may need to be stored on a hard drive and memory management techniques may be required.

The advantage of the H_{mg} heuristic function is that it takes into account the capabilities of the character encoded in the motion graph. For graph ISG, however, the H_{mg} heuristic function also helps to synchronize the contact patterns of the two paths being interpolated. For example, to reach a "picking" state P from state A the system needs to interpolate two sequences of "contact change" states which both should end at a "picking" state and should also be synchronized in contact pattern (see Figure 4.20). In our experiments about 95% of the states in graph MG can reach a "picking" state. But, because of the need to synchronize contacts for interpolated paths, only about 20% of the states in graph IMG can reach a "picking" state. The H_{mg} heuristic guides the search toward the states that *can* actually reach a "picking" state and avoids exploring those that cannot.

4.4 Experimental results

This section evaluates the effectiveness of our approach with a number of different experiments. The motions illustrating most of the experiments are available on the web:
www.cs.cmu.edu/∼alla/thesis/

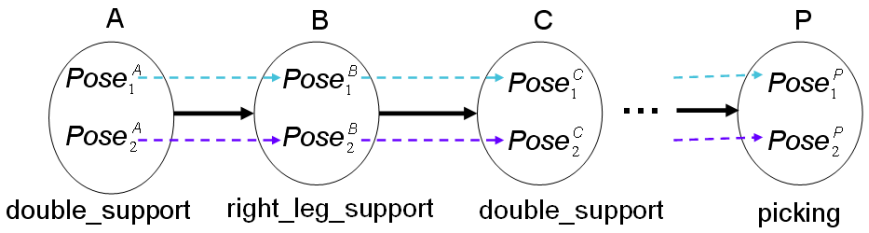

Figure 4.20:

4.4.1 Examples of motions

To illustrate the effectiveness of our approach, we generated a variety of different examples. For each experiment, the user specified a 2D path in the environment that the character should follow. The width of the corridor around the path was set to 0.5 meters. In some experiments, the user also specified additional constraints such as jumps and contact with environment at specified locations.

The examples include walking along curves of different curvature, picking and placing an object in various locations, jumping over stones with variable spacing, forward jumps of different lengths, vertical jumps with different amounts of rotation and forward walks of different step lengths. Figures 4.21– 4.24 show images for some of the results. It took less than 3 minutes to compute a close to an optimal solutions for all examples. The first, sub-optimal solution is usually found in just a few seconds.

4.4.2 The benefit of interpolation

The first experiment shows why interpolation in conjunction with motion graphs allows us to satisfy user-specified constraints within a small error tolerance. We used the following three test problems: (1) a character needs to start at position A and pick up a small object at position B (see Figure 4.25a); (2) a character needs to start at position A and pick up a small object at position B but now we also constrain the root position of the character to position C while picking a small object (see Figure 4.25b); (3) a character needs to start

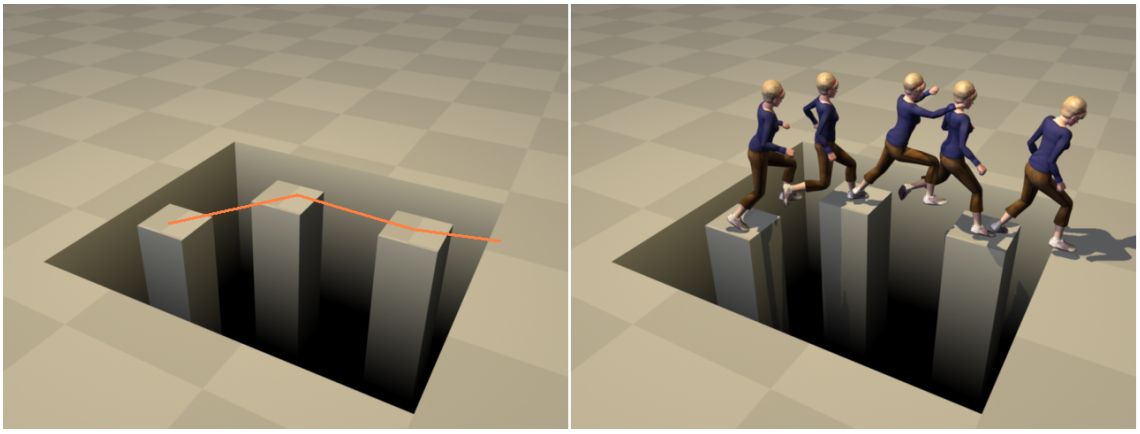

Figure 4.21: Left: the user sketched the path the character should follow; Right: synthesized motion.

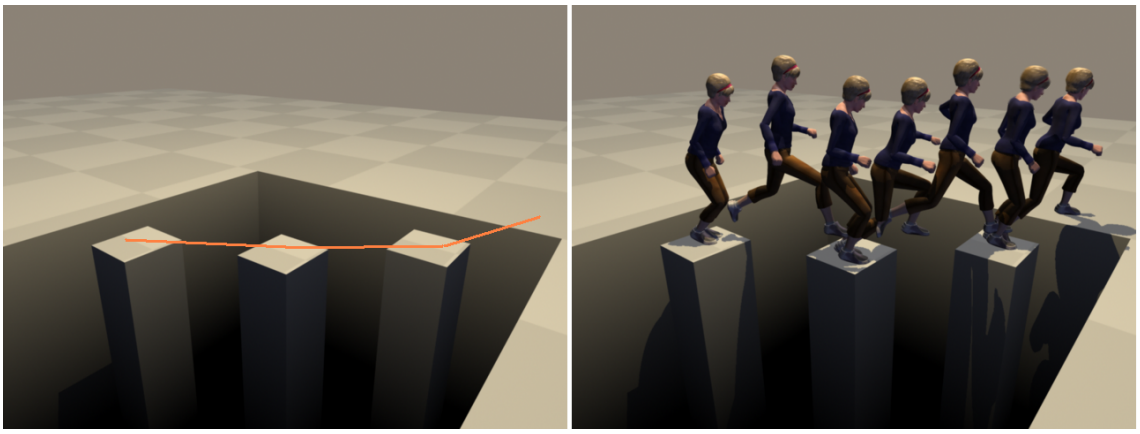

Figure 4.22: Left: the user sketched the path the character should follow; Right: synthesized motion.

Figure 4.23: Left: the user sketched the path the character should follow; Right: synthesized motion.

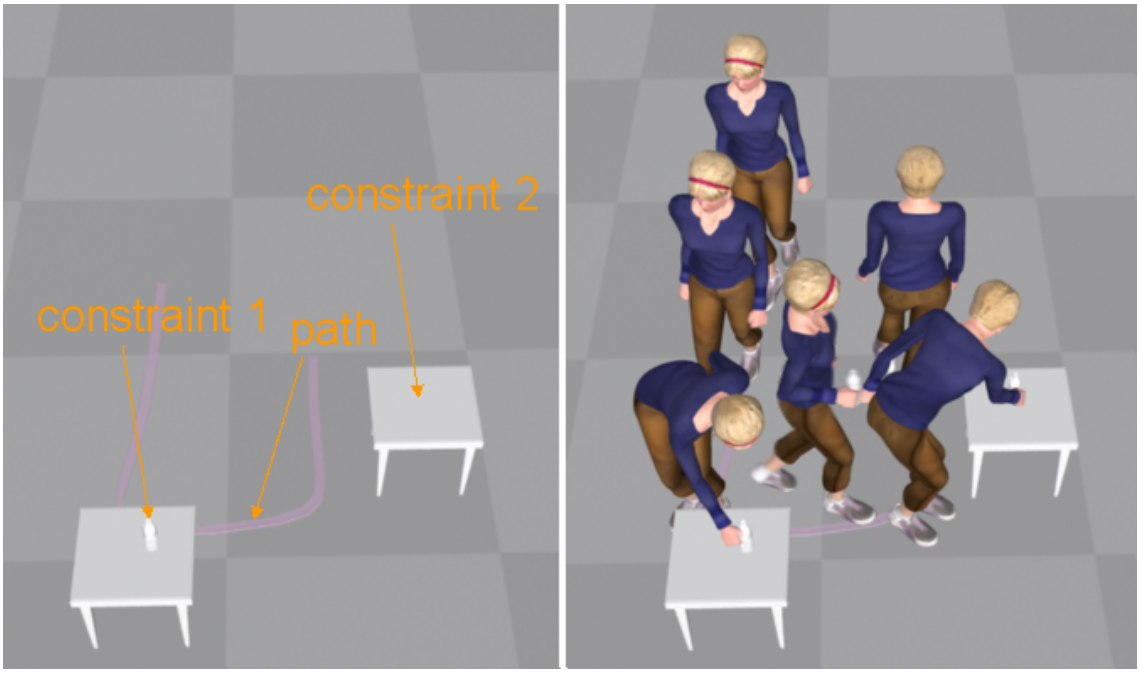

Figure 4.24: Left: the user sketched the path the character should follow and two constraints—the character needed to pick up a bottle from the first table and put it on the second table; Right: synthesized motion.

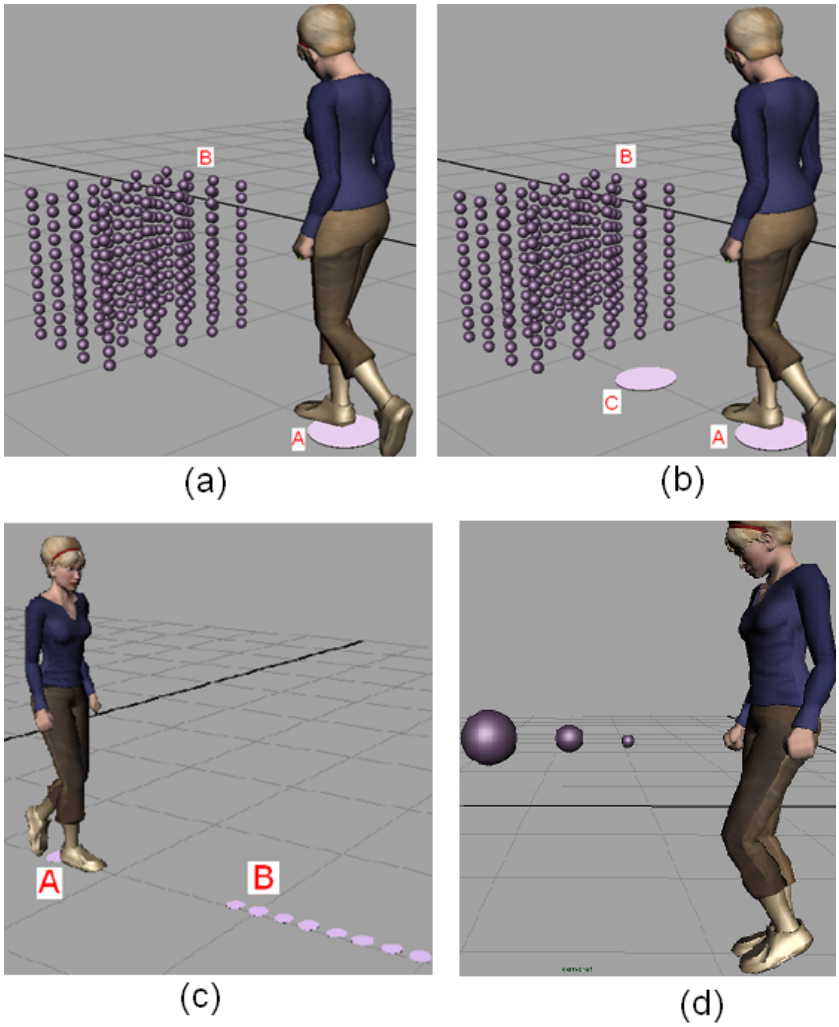

Figure 4.25: Test problems: (a) A character needs to start at position A and pick up a small object at position B. We sample the location of constraint B; (b) A character needs to start at position A and pick up a small object at position B but now we also constrain the root position of the character to position C while picking a small object. We sample the location of constraint B; (c) A character needs to start at position A and walk to position B with one walk cycle. We sample the location of constraint B; (d) Error tolerances from smallest to largest in centimeters: 2.5, 5 and 10.

at position A and walk to position B with one walk cycle (see Figure 4.25c).

We ran many experiments for each problem by sampling the locations of constraint B (179 samples for first two problems and 10 samples for third problem). Tables 4.1 summarize the results for each problem for $k = 1$ and for $k = 2$. The results show that with interpolation ($k = 2$) the system consistently finds solutions for small error tolerances. Without interpolation ($k = 1$), we need to set the error tolerances to much higher values to get consistent results. Figure 4.25d compares the tolerances visually.

We also computed the average cost (sum of squared torques) of the solutions for the second problem. Only experiments that succeeded were included in the computation of the average cost. The average solution cost is about 1.35 times higher without interpolation, especially for small tolerances. Without interpolation the search has much less freedom to satisfy the constraints. Therefore even if the search is able to find the solution, this solution is more likely to contain dithering and inefficient motion patterns that result in the higher solution cost (see website for examples of motions).

4.4.3 Interpolated motion graphs versus warping

The examples in this section show the advantage of our interpolated motion graphs approach over the alternative approach of first finding an approximate solution for $k = 1$ and then warping this solution to satisfy the user-specified constraints. Interpolation usually produces solutions with better, more optimal strategies and is also more robust (do not need to tweak the threshold as will be explained next).

The example in Figure 4.26 shows a situation where the solution that satisfies all user-specified constraints does exist for $k = 1$. In this example the character needs to get over three columns. The strategy found for $k = 2$ is more optimal than the one for $k = 1$. For $k = 2$, the character walks from the first to the second column and jumps from the second to the third column. For $k = 1$, on the other hand, the character jumps between each column. Because the optimal strategy for $k = 1$ is different than the one for $k = 2$ it would be impossible to warp the solution for $k = 1$ into the one for $k = 2$.

The example in Figure 4.27 shows a situation where the solution that satisfies all user-

	error tol=2.5 cm	error tol=5 cm	error tol=10 cm
% succeeded for $k = 1$	31%	70%	99%
% succeeded for $k = 2$	99%	100%	100%

	error tol=2.5 cm	error tol=5 cm	error tol=10 cm
% succeeded for $k = 1$	16%	45%	87%
% succeeded for $k = 2$	96%	99.5%	100%

	error tol=2.5 cm	error tol=5 cm	error tol=10 cm
% succeeded for $k = 1$	40%	75%	100%
% succeeded for $k = 2$	100%	100%	100%

Table 4.1: Three tables showing the success rate for each of the three test problems. Top table (first problem): we have uniformly sampled 179 locations in $3D$ where the character must pickup a small object (see Figure 4.25a). Middle table (second problem): again, we have uniformly sampled 179 locations in $3D$ where the character must pickup a small object (see Figure 4.25b). Bottom table (third problem): we have uniformly sampled the location of the goal along the X axis (Figure 4.25c); because the walk is constrained to one walk cycle, this resulted in walks with different step lengths. Each table entry shows the percent of the experiments that succeeded—the search found a solution within two minutes and within the specified error tolerance around the small object location. The first row is for no interpolation ($k = 1$) and the second row is for interpolation with $k = 2$. We repeat the experiment for three different error tolerances (see Figure 4.25d).

specified constraints within the user-specified tolerance does not exist for $k = 1$. In this example the character needs to follow the curve within the user-specified tolerance (represented as a corridor around the curve). For $k = 2$, the solution exists and the character can track the curve very well. Because the solution does not exist for $k = 1$ we need to increase the corridor width around the curve to find a solution for $k = 1$. In the resulting solution, however, the character takes only five steps, whereas for $k = 2$ she takes six steps. It is therefore unclear how to warp this solution to satisfy the tolerance constraints. As in the previous example, this is a different strategy for the solution and it would not be possible to warp the solution for $k = 1$ to the solution for $k = 2$.

It is also often hard to decide by how much to increase the tolerance. Increasing tolerance too much may result in a solution that would be impossible to warp. For example, we often cannot find any solution for $k = 1$ for jumping across arbitrary placed columns. If we increase the width of the column too much the character walks across the columns instead of jumping but to satisfy the original column width the character actually needs to jump. In this case the algorithm can not find any solution at all for $k = 1$ because it is impossible to warp a walk into a jump.

In general, often there is no solution for $k = 1$ that satisfies user constraints within the user-specified tolerance. Increasing the tolerance allows a solution to be found. But then, the found solution often follows a different, less optimal strategy than the $k = 2$ solution and therefore cannot be warped to match.

4.4.4 The benefit of optimality

In this experiment we show that globally near-optimal solutions avoid the dithering and inefficient patterns of motion that sub-optimal solutions often have. We show how the cost of the solution changes as its optimality increases during our search. Tables 4.2 show the results for three motions: a walk from start to goal; a walk from start to goal that requires crossing three rivers of different width; a walk to a place where the character needs to pick up an object. As the optimality of the solution increases the character finds more efficient motion patterns (see website for examples of motions).

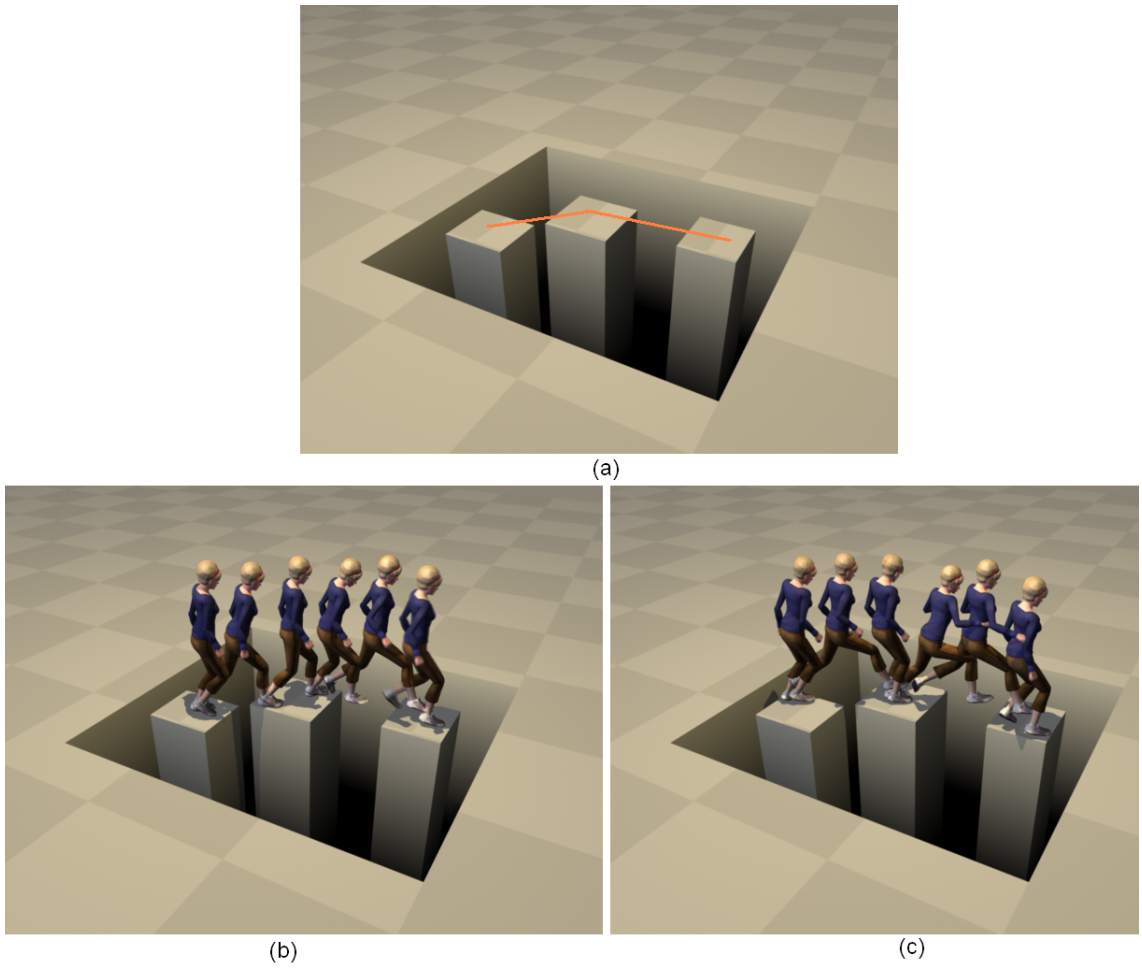

Figure 4.26: (a) User sketch. (b) The optimal solution for $k = 2$. (c) The optimal solution for $k = 1$. For $k = 2$, the character walks across the first column and jumps across the second column. For $k = 1$, the character jumps between each pair of columns. Because the optimal strategy for $k = 1$ is different than the one for $k = 2$, it is impossible to warp the solution for $k = 1$ into the solution for $k = 2$.

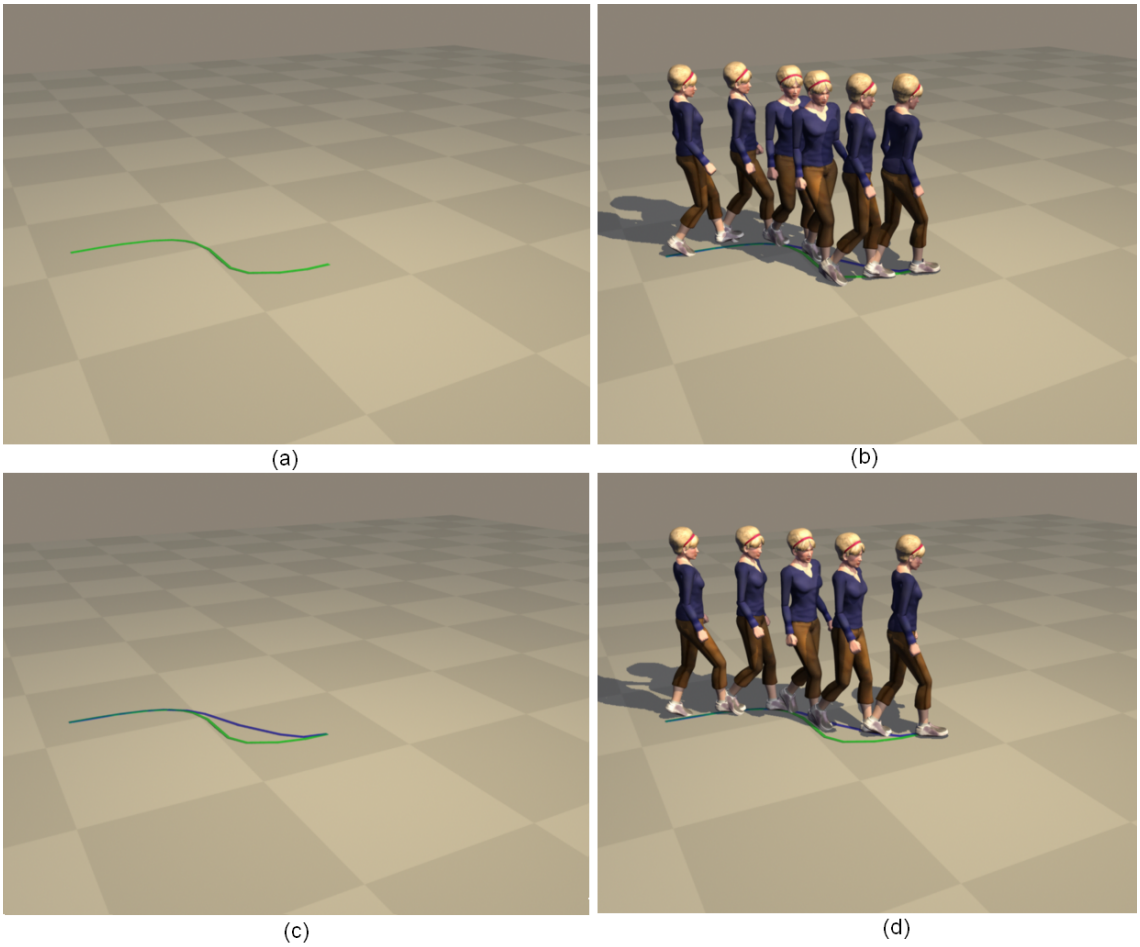

Figure 4.27: (a) User sketch. (b) The solution with interpolation, $k = 2$. The character can track the curve, using a very small corridor width around the curve. (c) The blue curve shows the trajectory of the root for the best solution with $k = 1$. In order to find any solution for $k = 1$, we needed to increase the corridor width around the curve. (d) The found solution for $k = 1$. The character takes five steps, whereas for $k = 2$ she takes six steps. It is therefore impossible to warp the solution for $k = 1$ into the one for $k = 2$.

Search Time (secs)	Solution Cost	Optimality Bound (ϵ)
0.67	2,700,000	10.0
3.42	1,800,000	2.0
50.00	1,600,000	1.0

Search Time (secs)	Solution Cost	Optimality Bound (ϵ)
0.016	10,700,000	10.0
0.032	8,200,000	2.0
0.844	6,250,000	1.0

Search Time (secs)	Solution Cost	Optimality Bound (ϵ)
0.70	1,200,000	10.0
9.10	650,000	2.0
20.10	550,000	1.0

Table 4.2: Top table (motion 1): walk from start to goal; the first solution is very suboptimal—the character makes two really large steps to reach the goal position (Figure 4.28a); the second solution is better—the character makes smaller steps but the walk is a bit unnatural because the steps are of different length (Figure 4.28b); the final solution is optimal and looks natural (Figure 4.28c). Middle table (motion 2): walk with jumps over three rivers; the first solution is suboptimal—the character makes inefficient two legged jumps to cross all three rivers; the second solution is more natural, the character now uses a one-legged jump to cross the rivers; in the optimal solution the character does not jump but steps over the last (the smallest) river. Bottom table (motion 3): the character starts on the small rectangle and needs to walk toward and pick a small object shown by a sphere in Figure 4.29; the first solution is very suboptimal, the character bends way too far to pick up a small object (Figure 4.29a); the second solution is better but the character is reaching from the side which in the absence of constraints appears unnatural(Figure 4.29b); the final solution is optimal and looks natural (Figure 4.29c).

Figure 4.28: (a) The first, sub-optimal solution. (b) A better solution, but still not optimal because of variation in step length. (c) A final, optimal solution.

Figure 4.29: The character starts on the small rectangle and needs to walk toward and pick a small object shown by a sphere. One frame from each motion showing the character's pose when she touches the object. (a) The first solution is very suboptimal, the character bends way too far to pick up a small object; (b) The second solution is better but the character is reaching from the side which in the absence of constraints appears unnatural; (c) Final solution looks natural.

4.4.5 The discrete versus continuous approach

In this experiment we compare our discrete and continuous optimization approaches. We use discrete optimization to synthesize motions that are similar to these we have already synthesized using continuous optimization. The generated motions for all experiments can be found on the website.

To synthesize jumping motions we had to use more example motions to compute a motion graph for discrete approach than we used to construct a basis needed to synthesize the same examples for continuous approach. For continuous approach we could synthesize various jumps from a basis constructed from just three jumps: a forward jump, a forward jump with a 90 degree turn and a vertical jump with a 180 degree turn. We need to add more jumps to the discrete space because the solution is restricted to interpolation of existing examples. We discuss it in more details in Chapter 6.

Using discrete optimization, we synthesized vertical jumps with various amounts of rotation—from 0 to 360 degrees. We used three motions to compute the discrete space: a vertical jump of 0 degrees, a vertical jump of 90 degrees and a vertical jump of 360 degrees.

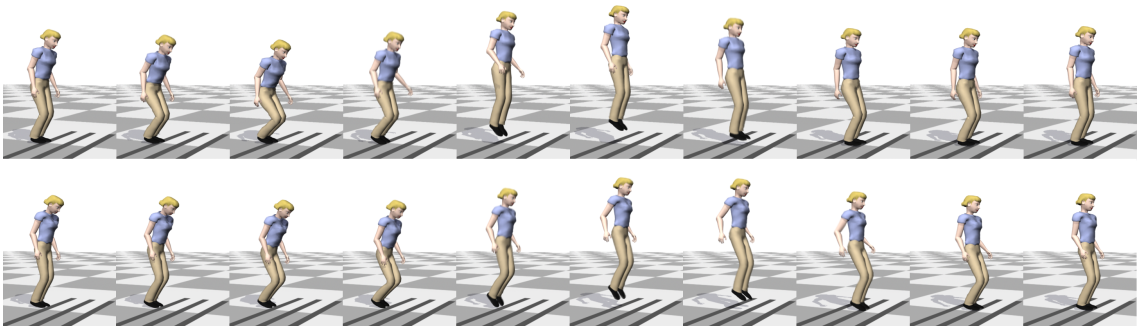

Figure 4.30: Short forward jump. Top: a solution produced using our continuous optimization approach. Bottom: a solution produced using our discrete optimization approach.

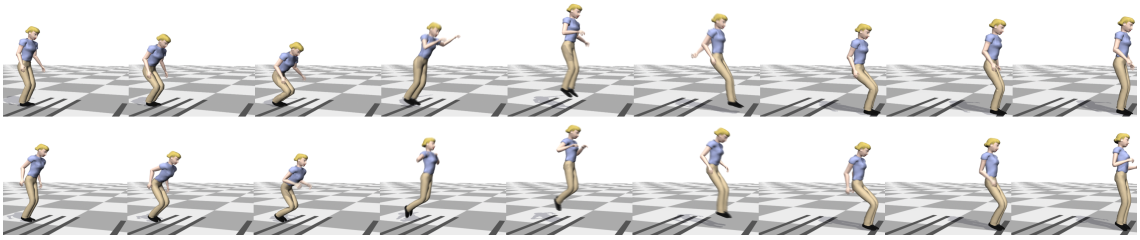

Figure 4.31: Long forward jump. Top: a solution produced using our continuous optimization approach. Bottom: a solution produced using our discrete optimization approach.

We also synthesized forward jumps of various length—from 0.2 to 1.5 meters using three motions to compute the discrete space: a forward jump of 0.2 meters, a forward jump of 1 meter and a forward jump of 1.5 meters.

We also synthesized forward walks of different step length—from very small to exaggerated. We used the same seven forward walking motions with different step length to compute the discrete space as we used to compute a basis for the continuous space in Chapter 3.

Figures 4.30– 4.32 compare discrete and continues solutions for the two jumping motions and the walking motion. Movies that can be found on the website give a better comparison than still images. Both approaches generate natural looking motions.

The continuous space can generalize quite well beyond the sample motions (Sec-

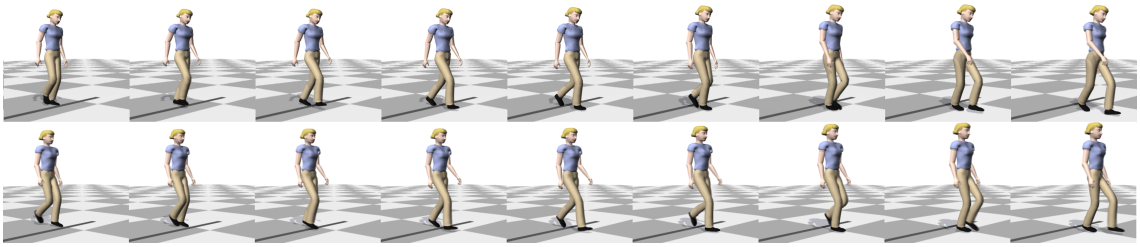

Figure 4.32: One cycle of forward walk. Top: a solution produced using our continuous optimization approach. Bottom: a solution produced using our discrete optimization approach.

tion 3.2). To assess how well the discrete space can extrapolate beyond the sample points, we tried to compute a vertical jump of 360 degrees in a discrete space computed from three vertical jumps: one of 90, one of 180 and one of 200 degrees. The resulting motion did not look natural because it required too much extrapolation. On the other hand, a 250 degree jump could be synthesized successfully. In general, only small amount of extrapolation seems to be visually acceptable. We repeated the extrapolation experiments for forward jumps and obtained a similar result.

The time it takes to synthesize motions in discrete space is much less than in continuous space. It took less than a second to synthesize jumping motions and less than 5 seconds to synthesize walking motions described in this section. Section 6.1 provides a more detailed comparison of our continuous and discrete optimization approaches.

4.4.6 The benefit of motion graph compression

In this experiment, we evaluate the effect of the motion graph compression. Table 4.3 shows general statistics for three different databases: (1) walking and jumping motions; (2) walking and picking motions; (3) just walking motions. For each database, we computed the number of states and transitions in the motion graph before compression, after the first compression step (merging transitions) and after the second compression step (merging states). The table also gives the time required to compress the graph (a pre-computation

	Before Merging	After merging transitions	After merging states	Compression Time
DB 1	states=12,000 trans=250,000	states=700 trans=60,000	states=305 trans=15,000	60 min
DB 2	states=6,500 trans=70,000	states=430, trans=15,000	states=200 trans=2,200	30 min
DB 3	states=2,000 trans=25,000	states=173 trans=3,700	states=100 trans=1,000	2 min

Table 4.3: Compression for three motion graphs. The first graph is computed from walking and jumping motions. The second graph is computed from walking and picking motions and the third one is computed from just walking motions.

step performed only once for each database). Compression techniques reduce the size of the graph by a factor of 20 to 40. For example, after the first compression step the number of states in the first database is reduced by a factor of 17 and the number of transitions by a factor of 4. After the second compression step the number of states is reduced by a factor of 39 and the number of transitions by a factor of 16.

4.4.7 The benefit of the heuristic function

We also evaluated the effectiveness of our heuristic function. The results shown in Table 4.4. We compare four heuristics: (1) the Euclidean distance to the goal; (2) only the H_{2D} component of our objective function; (3)only H_{mg} component of our objective function; (4) the combined heuristic function with both H_{2D} and H_{mg} components. The results demonstrate that our heuristic function is essential for making search efficient and often makes the difference between finding a good solution and not finding one at all. The table also shows that both components of the heuristic function are important for getting good results—leaving just one component and disabling the other makes the search substantially less efficient.

ε	Euclidean distance			H_{2D}			H_{mg}			$H_{combined}$		
	time	exp	solved	time	exp	solved	time	exp	solved	time	exp	solved
10.0	8.0	185,813	100%	8.1	160,718	100%	11.6	72,004	100%	0.8	9,332	100%
3.0	17.1	481,321	100%	16.8	406,149	100%	15.1	103,000	100%	1.6	16,068	100%
1.0	100.2	1,832,347	20%	97.8	1,748,620	20%	48.1	270,812	80%	49.5	275,712	80%

Table 4.4: Evaluation of the heuristic function. Each column shows the runtime of the search in seconds and the number of states expanded by it for different heuristic functions. The first column is for the Euclidean distance to the goal. The second column uses only the H_{2D} component of our objective function. The third column uses only H_{mg} component of our objective function. The last column shows the results for the combined heuristic function. The first row shows search efforts to obtain a solution whose cost is at most 10 times the optimal one. The sub-optimality bound for the second row is 3. The solution in the last row is optimal.

4.5 Discussion

In this chapter, we built a discrete reduced-space representation of human motion. This representation can be viewed as a combination of a motion graph and interpolation techniques. The motion that can be generated with this representation is an interpolation of k time-scaled paths through the motion graph. Finding a solution in this smaller search space is much easier than finding a solution in the full search space. In addition, the synthesized motion is likely to contain natural coordination patterns. This objective is difficult to describe mathematically and is therefore hard to achieve when searching the full search space. We have shown that the optimization in this discrete space supports interactive frame rates and allows for the synthesis of less dynamic motions and longer motions that are composed of different behaviors. It takes less than 3 minutes to compute a close to an optimal 10 sec long motion and the first sub-optimal solution is usually found in just a few seconds.

The quality of the results largely depend on the quality of the motion database used to construct a motion graph. For example, if the database contains only a motion of sitting on a tall chair then we cannot synthesize a motion for sitting on a medium or a low height chair because there are no two motions whose interpolation would give us the desired

motion.

We also found that it is important that the resulting motion graph has a "good" connectivity. For example, if it is impossible to reach a "picking" state from other states in the motion graph then we cannot synthesize motions that satisfy picking constraints. Our experiments show that to obtain good results it is important that many states in the motion graph can quickly reach a constraint state (such as "picking" state) and many states can be quickly reached from constraint states. An automatic technique for evaluating the connectivity of a motion graph would be very helpful. In their work, Reitsma and Pollard [58] evaluated the quality of a motion graph for navigational tasks. Extending this evaluation to our domain would be useful.

Better automatic methods for constructing motion graphs with "good" connectivity would also be very helpful. We found that a single threshold for picking good transitions often does not work well. A low threshold results in most transitions occurring within a single behavior (walks for example) and very few transitions between motions of different behaviors (walks and jumps for example). A high threshold, on the other hand, results in many low quality transitions within a single behavior even though these transitions are not needed.

We compute motions that minimize sum of squared torques as an objective function while satisfying user constraints. Our objective function together with the requirement that we interpolate motion segments with the same contact seems to work well to pick and synchronize motion segments that when interpolated result in natural motions. For example, two walk segments with both arms to the side are more likely to be picked by the search for interpolation than segments with one arm waving because the latter requires more energy (unless the constraints specifically require waving).

Our experimental results show that our approach works well for a database with 12,000 frames (motions are sampled at 30 frames a second). Scaling our approach to the larger databases, however, will require additional work. We plan to experiment with automatic clustering of motions into behaviors and only interpolating motions within the same class to further reduce the size of the problem.

All of our results are for $k = 1$ (no interpolation) and $k = 2$ (interpolation of two paths). In all of our experiments, $k = 2$ was sufficient to find a solution that met the users constraints and produced natural looking motion. Some problems, however, may require $k > 2$ in order to satisfy the constraints. We plan to try to scale our approach to $k = 3$. If this proves to be infeasible, we can also try an iterative approach: first compute the best possible solution for $k = 2$; then compute another solution for $k = 2$ which, when interpolated with the solution we already have, produces a better result; continue in this manner until no further improvement is possible. This iterative, greedy, approach did not work for $k = 1$, but if $k = 2$ is sufficient to land the right strategy then small refinements should work well in a greedy fashion.

Our approach can currently find a close to an optimal solution for motions that are approximately 10 seconds long. Scaling our approach to longer motions is part of the future work. As the length of the desired motion increases, the complexity of the search greatly increases. Suppose the final motion consists of P contact phases. We can decrease the complexity of the search by limiting the number of contact phases in the final solution that will involve interpolation. For example, only five out of P contact phases of the final motion will be computed by interpolating motion graph segments and the other will be original segments from the motion graph. Interpolation is most often required whenever the motion needs to satisfy user-specified constraints (such as pick position or curvature) and is generally not necessary when the character is moving in free space. Deciding when to allow interpolation and when not to in advance would be difficult. To satisfy a picking constraint for example, we may need to interpolate few steps before the picking to position the character in front of the object. We can limit interpolation by adding one more variable to each state in the graph that counts the number of motion segments that have been interpolated so far. We can then limit this variable during the search and therefore control the maximum number of segments interpolated in the solution. Interpolating only when it is real benefit should greatly reduce the complexity of the search.

Chapter 5

Analysis of the physical correctness of interpolated motion

Over the past ten years, interpolation of motion capture data has been shown to be a very powerful technique for generating high quality and natural looking motion. This technique is successful in part because the naturalness of the original motions is not destroyed by the relatively small changes made in the process of interpolation. However, larger changes may also produce natural looking motion if interpolation is performed within a well-defined class of behaviors such as kicking [32] or walking [60] where significant events such as foot contact can be aligned.

For these larger changes, in particular, it is not immediately obvious why interpolation should produce such good results. For example, straightforward linear interpolation could well introduce visually apparent errors in the physics of the motion. In this chapter of the thesis, we analyze the physical correctness of motions created by interpolating a few, presumably physically correct, human motions.

We analyze the interpolated motion in terms of a number of basic physical properties: (1) linear and angular momentum during flight; (2) foot contact, static balance and friction with the ground during stance; (3) continuity of position and velocity between phases. We assume that the motions used for interpolation are physically correct themselves, have the

same skeleton, can be aligned in time by picking corresponding key events and that linear interpolation is used to interpolate parameters of motions between these key events.

Our analysis shows that with a few simple modifications to the straightforward interpolation technique proposed by others, we can prove that these physical properties are satisfied for a wide range of different kinds of motions. The interpolated motion will satisfy these physical properties if the motions used for interpolation do not include significant rotation during the flight phase (runs, forward and vertical jumps, for example), rotate around approximately the same principal axis by approximately the same amount (jumps with turns, for example) or have no flight phase (walks or kicks, for example).

The analysis presented in this chapter should at least partially resolve a concern that has been raised about interpolation–that it is not a suitable technique for highly dynamic motions because the physics of the resulting motion is incorrect. While the main contribution of this work lies in its analysis, the few simple modifications to the interpolation scheme that we describe can also significantly improve the visual quality of certain classes of interpolated motions while guaranteeing their physical correctness. We also use the analysis presented in this chapter to assess the physical correctness of the interpolated motion computed using our discrete optimization approach (as was described in Chapter 4).

5.1 Problem Description

The interpolation problem is defined as follows: Given k human motions M_1, M_2,...,M_k compute motion M by interpolating the parameters of these example motions. Each motion is defined as a sequence of frames $M(t) = \{P_{root}(t), Q(t)\}$, where $P_{root}(t)$ is the position of the root segment of the character, $Q(t) = \{q_1(t)...q_n(t)\}$ is the orientation of the root and the relative angles of the character's joints and $t = 0..T$ is the time of a particular frame. In this work we use Euler angles to represent rotations although most of the analysis is independent of the rotation representation.

Same as in the previous chapters, we use a right handed coordinate system for all motions, with the X and Z axes spanning the ground plane and the Y axis pointing up.

Positive rotation is assumed to be counterclockwise about the axis of rotation.

Using a technique proposed by a number of other researchers (including Rose and his colleagues [60]), we compute motion M by interpolating the root positions and all the joint angles of the example motions. The example motions must be scaled in time, or time-warped, to align key events such as foot contacts. We use a time-warping scheme similar to the one proposed by Rose and his colleagues [60]. We assume that a set of matching key frames for the input motions is provided (either by the user or computed automatically) and that the motion segments between these key frames can be scaled uniformly.

In our work, as in most other approaches to interpolation, we automatically locate these key frames at changes in the contact with the environment because the physical laws governing the motion change with contact. Motions M_1, M_2,...,M_k are split into phases based on these key frames and the corresponding phases are interpolated. For example, a jumping motion would consist of three phases: lift-off, flight and landing. Additional key frames can be added during long contact phases to better align the motions without violating the assumptions behind our analysis.

We compute each phase of motion M by interpolating corresponding phases of motions M_1, M_2,...,M_k with a constant set of weights, w_1, w_2,...,w_k:

$$M = w_1 M_1 + w_2 M_2 + ... + w_k M_k \tag{5.1}$$

where $\sum_{i=1}^{k} w_i = 1$. The analysis in this work assumes that the weights sum to one so our results are limited to interpolation and do not generalize to extrapolation. The analysis is presented for interpolation of only two motions, M_1 and M_2 but generalizes to the interpolation of k motions because equation 5.1 can be recursively computed by interpolating two motions at a time. The weights for each interpolation sum to one and the final interpolation produces a motion with the weighting given in equation 5.1.

Consider a particular phase Φ. At each time t of that phase we compute motion $M(t, w)$ as follows:

$$M(t,w) = \begin{cases} P_{root}(t) = wP_{1root}(t_1) + (1-w)P_{2root}(t_2) \\ Q_i = wQ_{1i}(t_1) + (1-w)Q_{2i}(t_2), \quad \text{for } i = 1..n \end{cases} \tag{5.2}$$

where $w = 0..1$ is the interpolation weight, T_1, T_2 and T are the time of phase Φ in motions

M_1, M_2 and M respectively, and $t_1 = tT_1/T$ and $t_2 = tT_2/T$ are time indices into motions M_1 and M_2.

We analyze the physical correctness of motions computed by linear interpolation of two motions with a constant weight w. This analysis includes: (1) the flight phases of the motion, (2) the contact phases and (3) the transitions between the flight and contact phases. During flight the only force acting on the character is gravity. During contact the feet of the character should not slide, contact forces should not require an unreasonably high coefficient of friction, and when the character is in static balance, the center of mass of the character should fall within the support polygon of the feet. The transition between contact and flight phases must maintain continuity (for example, the velocity and position at the end of the flight phase should match that at the beginning of the contact phase). In the next three sections, we present our analysis and suggest some improvements over existing interpolation schemes.

5.2 Analysis of the flight phase

In this section we analyze the linear and angular momentum of the interpolated motion during flight. We verify that during flight the net external force acting on the character is gravity and that for a restricted model of the character angular momentum is conserved.

5.2.1 Linear momentum during flight

Because the net external force acting on the character during flight is gravity, the trajectory of the center of mass should be a parabola:

$$R_{com}(t) = R_{0com} + V_{0com}t + 0.5Gt^2 \tag{5.3}$$

where R_{0com} and V_{0com} are position and velocity of the center of mass of the character at the start of the flight phase and $G = (0, -9.8, 0)$ is the acceleration due to gravity.

Figure 5.1 shows the Z component of the trajectory of the center of mass when a forward jump with no turn and a forward jump with a 360 degree turn are interpolated

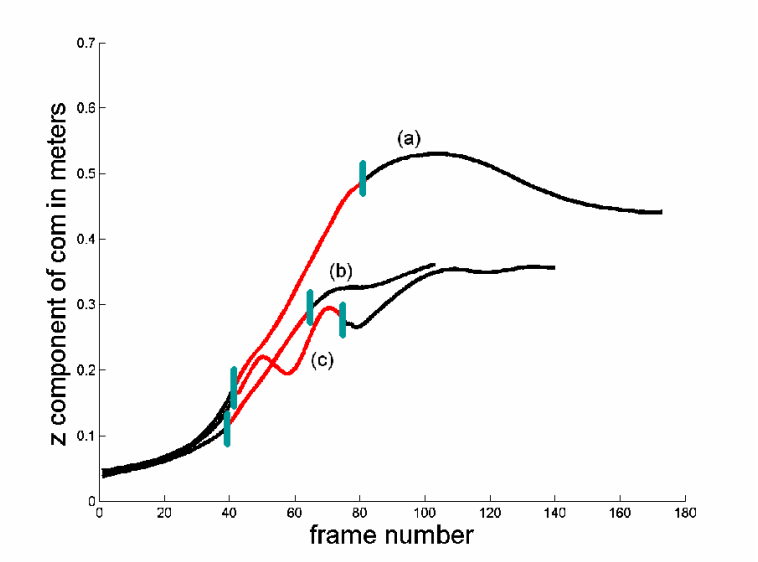

Figure 5.1: The Z component of the trajectory of the center of mass for: (a) a forward jump with no turn (motion M_1); (b) a forward jump with 360 degree turn (motion M_2); (c) the motion that results from interpolating motions M_1 and M_2. Vertical bars are used to indicate the beginning and ending of the flight phase for each motion. The trajectory of the center of mass of the interpolated motion during flight is not a straight line as it should be.

using equation 5.2. Because gravity only acts in the vertical, Y, direction, the Z component should be a straight line during flight but it is not. The trajectory appears to contain additional forces that act on the character during flight.

As the example in Figure 5.1 shows, linear interpolation of the root position and the joint angles of the character can result in a *non-linear trajectory* for the center of mass. A simple fix is to interpolate the center of mass trajectories instead of the root positions. The root position can then be computed from the new center of mass position and joint angles (see Appendix A). The interpolation equation is now:

$$M(t,w) = \begin{cases} P_{com}(t) = wP_{1com}(t_1) + (1-w)P_{2com}(t_2) \\ Q_i(t) = wQ_{1i}(t_1) + (1-w)Q_{2i}(t_2), \quad \text{for } i = 1..n \\ P_{root}(t) = F(P_{com}(t), Q(t)) \end{cases} \tag{5.4}$$

where F is the function that computes the root position from the center of mass and the

joint angles. With this small change, we can now prove that the net external force acting on the character during flight is gravity. According to Newton's second law:

$$\frac{dP}{dt} = F_{net} = \bar{m}G \tag{5.5}$$

where P is the total linear momentum of the character, F_{net} is the net external force acting on the character, $\bar{m}$ is the total mass of the character and $G = (0, -9.8, 0)$ is the acceleration due to gravity.

Proof: Linear momentum of the articulated character $P = \bar{m}V_{com}$. Taking the derivative of P with respect to time:

$$\begin{aligned}
\tfrac{dP(t)}{dt} &= \bar{m}A_{com}(t) \\
&= \bar{m}(wA_{1com}(t_1)(\tfrac{T_1}{T})^2 + (1-w)A_{2com}(t_2)(\tfrac{T_2}{T})^2) \\
&= w(\tfrac{T_1}{T})^2\bar{m}A_{1com}(t_1) + (1-w)(\tfrac{T_2}{T})^2\bar{m}A_{2com}(t_2) \\
&= w(\tfrac{T_1}{T})^2\bar{m}G + (1-w)(\tfrac{T_2}{T})^2\bar{m}G \\
&= \bar{m}G(w(\tfrac{T_1}{T})^2 + (1-w)(\tfrac{T_2}{T})^2) \\
&= \bar{m}G
\end{aligned} \tag{5.6}$$

The transition from the first to the second line is obtained by taking second derivative of the position of the center of mass in equation 5.4 with respect to time (see Appendix B). The transition from the second to the third line is obtained by rearranging terms in the equation. The transition from the third to the fourth line is obtained by substituting $\bar{m}A_{1com}(t_1) = \bar{m}G$ and $\bar{m}A_{2com}(t_2) = \bar{m}G$. This substitution is valid because we assume that motions M_1 and M_2 are physically correct. The transition from the fourth to the fifth line is obtained by rearranging terms in the equation. The transition from the fifth to the sixth line is obtained by substituting:

$$T = \sqrt{T_1^2 w + T_2^2(1-w)} \tag{5.7}$$

This equation defines the choice of the time, T, which will ensure that gravity is correct during flight.

In the literature, the time of an interpolated motion has generally been computed as:

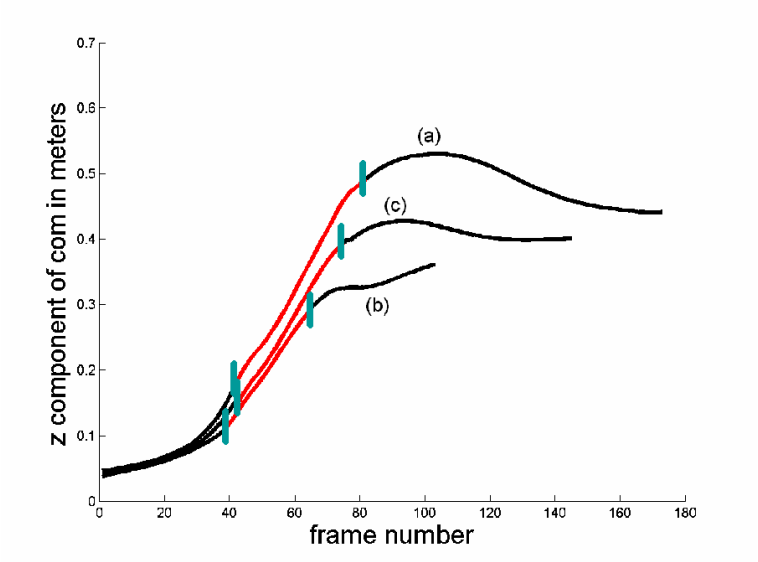

Figure 5.2: Example from figure 5.1 but with the flight phase of the interpolated motion computed by interpolating the center of mass positions of the input motions instead of the root positions and with the time of the flight phase computed as $T = \sqrt{T_1^2 w + T_2^2 (1-w)}$.

$T = wT_1 + (1-w)T_2$. But setting time in this way results in scaling gravity by:

$$\frac{wT_1^2 + (1-w)T_2^2}{(wT_1 + (1-w)T_2)^2} \tag{5.8}$$

In many cases this error will be small and will not be noticeable. Reitsma and Pollard [59] determined that if gravity is between $-9.0m/s^2$ and $-12.7m/s^2$ the error is not visible to the human observer.

Figure 5.2 shows the example from Figure 5.1 with the interpolated motion during the flight phase computed according to equation 5.4 and with time $T = \sqrt{T_1^2 w + T_2^2 (1-w)}$. The Z component of the trajectory of the center of mass during flight is now a straight line.

The difference between the interpolation schemes in equation 5.2 and equation 5.4 becomes most apparent when interpolating dissimilar motions (as in the example in Figure 5.1) or motions that involve significant movement of the root of the character with respect to the center of mass during flight. In our experiments, we found that for many

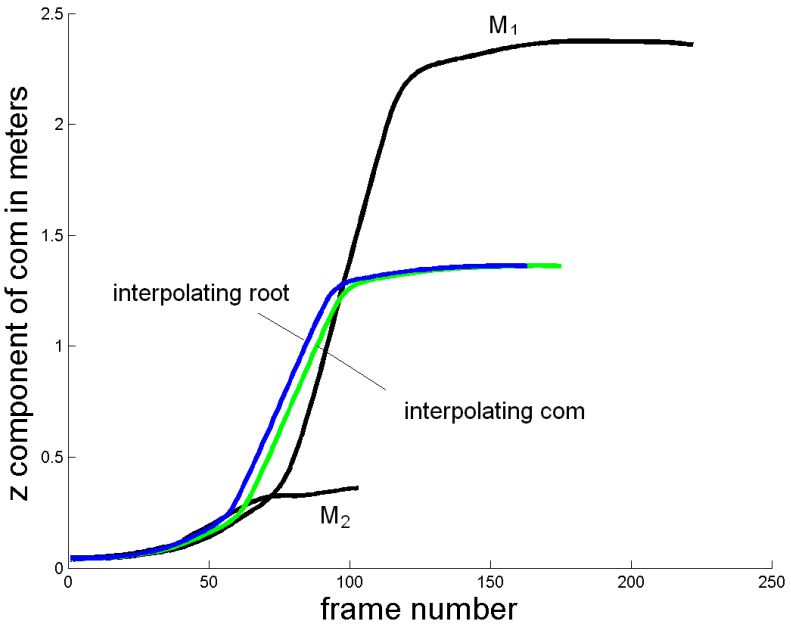

Figure 5.3: Interpolating a very small forward jump, 0.4 meters, (motion M_1) and a very large forward jump, 2.5 meters, (motion M_2). The Z component of the center of mass is shown for motion M_1, motion M_2 and two interpolated motions, one computed by interpolating root positions and one computed by interpolating the center of mass positions. The two trajectories for the Z component of the center of mass are very similar.

motions linear interpolation of the root resulted in an almost linear interpolation of the center of mass. See Figure 5.3 for an example.

The pelvis is often chosen as the root of the character. Because it is generally very close to the center of mass of the entire body, it often moves similarly. Figure 5.4 compares the position of the center of mass to the position of the root for the three motions used in Figures 5.1 and 5.3. For the forward jumps, the root moves similarly to the center of mass but for the jump with a 360 degree turn, the root moves along a different trajectory. As a result, interpolating the root positions of two forward jumps produces a natural looking motion and interpolating the root positions of a forward jump and a forward jump with a turn produces an unnatural looking result.

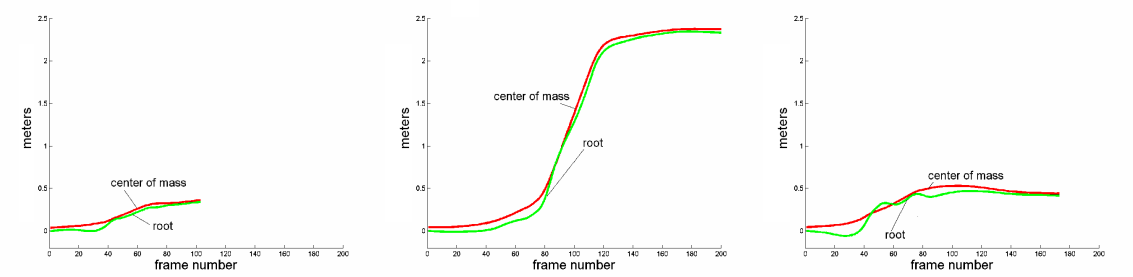

Figure 5.4: Comparing the center of mass trajectory to the root trajectory for three different jumps. Z components are shown. Left: small forward jump, middle: large forward jump, right: forward jump with 360 degree turn. These jumps were used to compute the interpolated motions in figures 5.1 and 5.3.

5.2.2 Angular momentum during flight

Because the only force acting on the system during flight is gravity, and gravity acts at the center of mass, the angular momentum of the system about the center of mass should be constant during flight. In general, angular momentum will not be constant for a motion computed by interpolating two arbitrary motions. For example, the upper row in Figure 5.5 shows an interpolation between a forward jump and a vertical jump with a 360 degree turn. The angular momentum of the interpolated motion is not constant during flight.

However, even relatively large fluctuations in angular momentum are often unnoticed by the viewer if they do not create large changes in angular velocities. Because angular momentum, H, is equal to the product between inertia of the body and angular velocity ($H = I\Omega$), large changes in angular momentum result in small changes in angular velocity if the corresponding inertia is also large. For example, in Figure 5.5 (upper row, rightmost image) angular momentum changes significantly around the X axis but the motion still appears natural. The change in angular momentum is hard to detect because the inertia around the X axis is large (because the longitudinal axis of the body is perpendicular to the X axis) and the resulting change in angular velocity is small.

Although the angular momentum is not necessarily preserved when interpolating two arbitrary motions it is possible to show that for a single rigid body, angular momentum

is conserved during flight if both motions rotate around the same principal axis or one or both contain no rotation. In many common motions, visible rotation (with large angular velocity) during flight is either absent (for example, a short forward jump or run) or happens around only one principal axis (for example, a longer forward jump, a flip or a vertical jump with a turn). Approximating the character with a single rigid body is not an accurate model for most motions but this proof still provides some insight into when angular momentum will be preserved.

Proof: If a rigid body rotates around a principal axis then the angular momentum, H, is equal to the product of inertia of the body I, and the angular velocity of the body, Ω around the axis of rotation. Let $H_1 = I_1\Omega_1$ and $H_2 = I_2\Omega_2$ be the angular momentum for the first and second motions respectively. If we interpolate the center of mass positions and the rotation angles, the angular momentum of the interpolated motion is

$$H = I_3\Omega = I_3(w\Omega_1\frac{T_1}{T} + (1-w)\Omega_2\frac{T_2}{T}) = constant \tag{5.9}$$

The angular momentum H is constant because I_3, Ω_1 and Ω_2 are constant during flight.

The bottom row of Figure 5.5 shows the angular momentum for a motion computed by interpolating two forward jumps. Because both jumps involve a rotation around the same axis the angular momentum in the resulting motion remains relatively constant during flight.

5.3 Analysis of the contact phase

In this section, we analyze the physical correctness of the motion while one or both feet are in contact with the environment. The following conditions should hold for the motion to be physically valid: (1) the feet of the character should not slide; (2) when the character is in static balance its center of mass should fall within the support polygon of the feet; (3) the contact forces that correspond to the motion should not require an unreasonably high coefficient of friction. We now analyze each of these requirements.

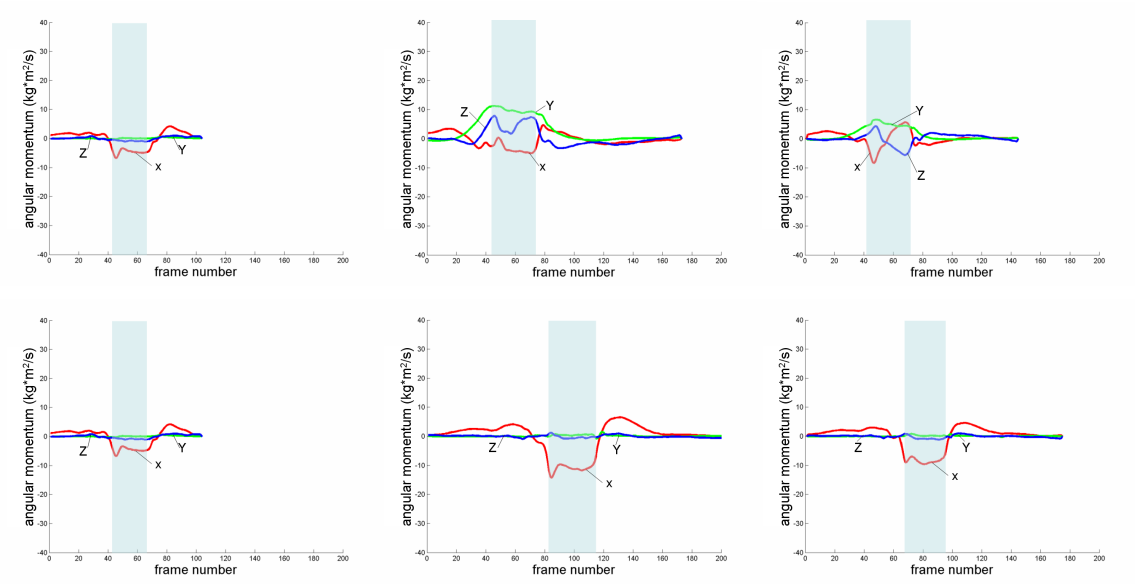

Figure 5.5: Upper row, from left to right: Angular momentum curves for a forward jump, a vertical jump with a 360 degree turn about the vertical axis and a motion computed by interpolating those motions. Lower row, from left to right: Angular momentum curves for a small forward jump, a very large forward jump and a motion computed by interpolating those motions. X, Y and Z components of angular momentum are shown for each graph. The shaded area represents the flight phase.

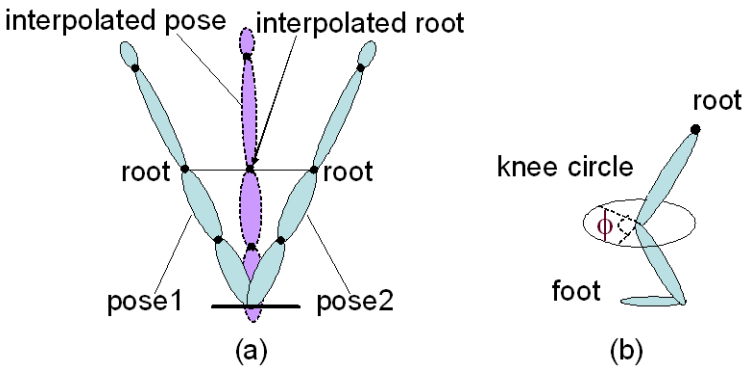

Figure 5.6: (a)Two poses of a simplified character that have the same contact are interpolated with weight $w = 0.5$. The resulting pose penetrates the ground. (b) The redundant degrees of freedom of each leg can be intuitively parameterized by one parameter, Φ, that represents the "knee circle" of the leg.

5.3.1 Non-sliding Foot Contact

We assume that when one or both feet of the character are in contact with the ground, the position of the feet should not move (the character does not slip). This condition, however, does not hold for either the center of mass or root interpolation schemes presented above. Consider the example in Figure 5.6(a): two poses of a simplified character that have the same contact point are interpolated with weight $w = 0.5$ resulting in a foot position below the ground.

Other researchers have addressed this problem by rooting the character at a foot that is in contact. This solution works well when there is only one foot in contact but may result in sliding of the other foot if it is also in contact. In general, preserving the contact positions of both feet and computing joint angles via interpolation is not possible because the system is over-constrained. A common solution is to to eliminate foot sliding in the interpolated motion with a post processing step (see, for example [40] and [60]). This additional editing would be hard to analyze for physical validity.

An alternative solution that preserves physics is to interpolate only the non-redundant degrees of freedom (dofs) of the character and the constraints. As we discussed in Chap-

ter 3, each constraint reduces the number of available dofs. Thus, if the character originally had $n + 3$ dofs (n rotational and 3 translational), then when both feet are in contact, the number of degrees of freedom is reduced by 12. Korein and Badler [30] and later Lee and Shin [40] showed that the degrees of freedom of a leg in contact with the ground can be controlled by just one parameter, Φ, assuming that the hip position of the leg is also known. Intuitively, that parameter represents the "knee circle" of the leg (Figure 5.6(b)). Thus, when both legs are in contact the non-redundant degrees of freedom of the character are (1) root position; (2) all the joint angles of the character except the legs; and (3) two "knee circle" parameters, one for each leg. We can now interpolate these non-redundant degrees of freedom and the constraints that include the positions and orientations of both feet and the feet will not slide.

Because there is no real advantage in interpolating the root as opposed to interpolating the center of mass on the ground, we can interpolate the center of mass as we did for the flight phase:

$$M(t,w) = \begin{cases} P_{com}(t) = wP_{1com}(t_1) + (1-w)P_{2com}(t_2) \\ Qnr_i(t) = wQnr_{1i}(t_1) + (1-w)Qnr_{2i}(t_2) \\ C_j(t) = wC_{1j}(t_1) + (1-w)C_{2j}(t_2) \\ P_{root}(t) = F2(P_{com}(t), Q_{nr}(t), C(t)) \end{cases} \tag{5.10}$$

where Qnr_i are the non-redundant dofs of the character not including the root, C_j are constraints such as feet positions and orientations, and $F2$ is the function that computes the root position of the character from the center of mass position, non-redundant dofs and the constraints (see Appendix C for details). To preserve continuity of the motion, we use equation 5.10 independent of whether one foot or both feet are in contact. With this interpolation scheme, the feet will not slide during contact and we can prove that the static balance condition holds and that the ground reaction forces are within the friction cone.

5.3.2 Static Balance

Static balance exists when the projection of the center of mass of the character onto the ground is within the support polygon of the feet. We do not assume that input motions

are statically balanced but we show that if they are, the interpolated motion is as well. We assume that M_1 and M_2 have the same support polygon.

The position of the center of mass of the interpolated motion at time t is equal to the interpolation of the center of mass of motion M_1 at time t_1 and of center of mass of motion M_2 at time t_2 (equation 5.10). Therefore, the center of mass of the interpolated motion, P_{com}, will lie on a segment connecting points P_{1com} and P_{2com}. The projection of P_{com} onto a plane of contact will also lie on a segment connecting the projections of P_{1com} and P_{2com}. Let us call these projections P^g_{com}, P^g_{1com} and P^g_{2com}, then:

$$P^g_{com}(t) = wP^g_{1com}(t_1) + (1-w)P^g_{2com}(t_2) \tag{5.11}$$

Because P^g_{1com} and P^g_{2com} lie within the support polygon of the feet, P^g_{com} will also lie within the support polygon assuming the support polygon is convex and $0 \leq w \leq 1$.

5.3.3 Friction cone

For the motion to be physically valid, ground contact should not require an unreasonably high coefficient of friction. We use a Coulomb friction model to analyze the ground contact. If contacting surfaces do not move with respect to each other (static friction) the ratio of the absolute values of the tangential component of the ground reaction force, F^{grf}_t, and the normal component of the ground reaction force, F^{grf}_n, should be smaller than the coefficient of static friction:

$$\frac{F^{grf}_t(t)}{F^{grf}_n(t)} < \mu_s \tag{5.12}$$

Geometrically this constraint means that the ground reaction force must fall within a friction cone oriented along the contact normal (Figure 5.7).

Assuming a single support polygon, Newton's second law says that the ground reaction force, $F^{grf}(t)$ is

$$F^{grf}(t) = \bar{m}A_{com}(t) - \bar{m}G \tag{5.13}$$

where $\bar{m}$ is the total mass of the system, $A_{com}(t)$ is the acceleration of the center of mass of the system and $G = (0, -9.8, 0)$ is the acceleration due to gravity. The ground reaction

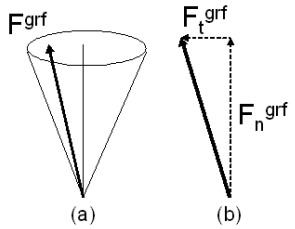

Figure 5.7: (a) The ground reaction force must fall within a friction cone oriented along the contact normal. (b) The tangential, F_t^{grf}, and the normal, F_n^{grf}, components of the ground reaction force.

force of the interpolated motion computed according to equation 5.10 is an interpolation of the ground reaction forces of motions M_1 and M_2 (see Appendix D for the proof):

$$F^{grf}(t) = wF_1^{grf}(t_1)\left(\frac{T_1}{T}\right)^2 + (1-w)F_2^{grf}(t_2)\left(\frac{T_2}{T}\right)^2 \tag{5.14}$$

The proof requires that we set the time T of the contact phase as

$$T = \sqrt{T_1^2 w + T_2^2(1-w)}$$

, which is the same formula we used for the flight phase. Now we need to show that equation 5.12 holds for the interpolated motion. From equation 5.14, we know that the tangential and normal components of the ground reaction force can be computed by interpolating the corresponding components of motions M_1 and M_2:

$$\frac{F_t^{grf}(t)}{F_n^{grf}(t)} = \frac{wF_{t1}^{grf}(t_1)(\frac{T_1}{T})^2 + (1-w)F_{t2}^{grf}(t_2)(\frac{T_2}{T})^2}{wF_{n1}^{grf}(t_1)(\frac{T_1}{T})^2 + (1-w)F_{n2}^{grf}(t_2)(\frac{T_2}{T})^2} \tag{5.15}$$

To show that equation 5.12 holds for the interpolated motion we first show that for any positive numbers a,b,c and d, if $\frac{a}{b} < \mu$ and $\frac{c}{d} < \mu$ then $\frac{a+c}{b+d} < \mu$ (see Appendix E for the proof). From this result, and because we know that equation 5.12 holds for motions M_1 and M_2 we can conclude that equation 5.12 holds for the interpolated motion.

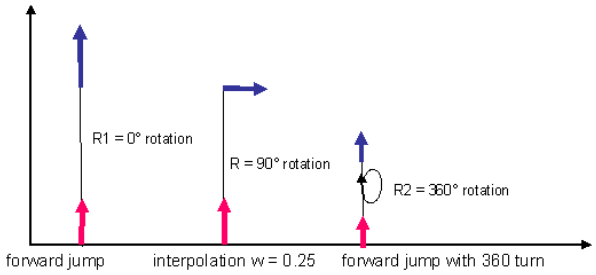

Figure 5.8: Interpolation between a long forward jump and a forward jump with a 360 degree turn (weight $w = 0.75$). Motions are shown schematically: the center of mass is projected onto the ground; the arrows represent the facing direction of the character.

5.4 Transition between phases

We have analyzed the motion of the character during flight and contact phases independently so we also need to analyze the continuity of the motion across the transition between phases. The continuity of the position of the center of mass follows trivially from the fact that it is computed by interpolating the center of mass of motions M_1 and M_2 which are themselves assumed to be continuous. The velocity of the center of mass may be discontinuous during the transition because different time scalings are applied to adjacent phases. We have found, however, that this discontinuity is not noticeable in practice.

Motions with rotation during the flight phase, however, may have significant discontinuities at the transition between flight and stance phases because the orientation of the interpolated motion may not match that of the original motions after the flight phase. For example, consider the interpolation of a long forward jump with a forward jump with a 360 degree turn (schematically shown in Figure 5.8). In the original motions the character lands facing the positive Z axis but in the interpolated motion, the character lands facing the positive X axis (rotated 90 degrees clockwise about vertical). Both the original interpolation scheme (equation 5.2) and the modified scheme (equation 5.10) will have problems with this transition. The resulting motion will have either significant foot sliding because the motion of the root does not match the motion of the joint angles or a discontinuity in the joint angles.

To reduce these problems, the subsequent motion of the root (or center of mass) in the original motions can be rotated to align them with the interpolated motion at the end of the flight phase. This operation brings the root motion into alignment with the character's facing direction and joint movements but it may introduce a discontinuity in the velocity of the center of mass. For example, in Figure 5.8, the landing velocity of the center of mass for the interpolated motion will be rotated instantaneously by 90 degrees in the transition to the stance phase. The discontinuity will be small when the required rotations are small or the ground plane components of the velocity at landing, $V_{landing}$, are small. The problem will be worse for motions with more complicated rotations such as forward or twisting flips.

5.5 Summary of Analysis

We have made three changes to the interpolation scheme of equation 5.2: (1) during flight we interpolate the center of mass positions instead of the root positions; (2) during ground contact we interpolate the positions of the feet, the center of mass positions and all non-redundant degrees of freedom to prevent the feet from sliding on the ground; (3) the timing of each phase is computed as $T = \sqrt{T_1^2 w + T_2^2 (1 - w)}$. With these changes, we can prove the following properties about the physical correctness of the interpolated motion:

- The net force acting on the character during flight will be equal to gravity.
- During contact, the feet of the character will not slide.
- If the character is balanced in the original motions, it will also be balanced in the interpolated motion.
- If the ground reaction force in both original motions is within the friction cone, it will also be within the friction cone for the interpolated motion.
- If we interpolate two motions that do not have visible rotation during the flight phases (for example, runs, short forward jumps and vertical jumps) or motions that

rotate about approximately the same principal axis (for example, flips and longer forward jumps), the angular momentum in the interpolated motion will be close to constant during flight. This analysis of angular momentum holds when the character can be reasonably approximated by a rigid body during flight.

- If we interpolate two motions that either (1) do not have visible rotation during their flight phases or (2) rotate by approximately the same angle about the vertical axis in both original motions or (3) occur mostly in the vertical direction (for example, a vertical jump), then the continuity of the velocity of the center of mass will be preserved during the transition from flight to contact (ignoring the discontinuity due to differences in time scaling of two phases).

5.6 Experimental results

Our experimental results consist of two parts. We first demonstrate that a variety of dynamical and non-dynamical motions can be successfully interpolated to generate realistic looking motions. The motions are generated by interpolating the trajectories of the center of mass and joint angles during the flight phases and during the stance phases placing the root at one of the feet in contact and interpolating the root and joint angles. The motions are also aligned as described in Section 5.4. The motions are all included in the video that can be found at the following website http://graphics.cs.cmu.edu/projects/interpolation_analysis/.

We performed the following experiments: (1) the interpolation of two forward jumps of very different lengths with no rotation; (2) the interpolation of two forward jumps of different lengths, each with a 90 degree turn; (3) the interpolation of two vertical jumps of different heights and different amounts of rotation; (4) the interpolation of two motions in which the actor stepped over obstacles of different heights; (5) the interpolation of running and a running jump. In each of these experiments, the original motions had the properties required to guarantee the physical correctness of the interpolated motions according to our analysis. The interpolated motions did indeed look visually realistic.

We also compare linear interpolation using root positions during flight with interpo-

lation using the position of the center of mass. We interpolated root and center of mass for a forward jump with no turn and forward jump with 360 degree turn (the example in Figure 5.1). This interpolation results in unnatural motion during the flight phase if the root is interpolated and natural looking motion if the center of mass is interpolated.

Interpolation of either the center of mass position or of the root position may cause the feet to slide or penetrate the ground. We demonstrated this by interpolating motions of a person sitting down on two seats of different heights. Interpolating root position results in significant sliding of the feet. Even simply placing the root at one foot of the character significantly reduces the problem although the second foot still moves slightly with respect to the ground.

Our last experiment demonstrated that if two motions with different amounts of rotation are interpolated there may be a visible discontinuity in the velocity of center of mass at landing (Section 5.4). This phenomena is demonstrated on the interpolation of a forward jump and a vertical jump with a 360 degrees turn. The resulting motion is quite unnatural.

5.7 Discussion

We, like others who have experimented with interpolation, have observed that the matching of key events is crucial for good results. Some key events such as foot contact are relatively easy to detect automatically. Others such as oscillations in arm swing, are more difficult to detect and match accurately. However, if two jumps are interpolated, one with a double arm swing during the landing phase and one with a single arm swing, the resulting motion will not be natural. Problems such as this will have to be addressed in an automatic fashion to make interpolation useful in situations where the details of the original motions are not a good match.

The analysis presented here only looked at physical correctness. It will not catch unnatural motions like the ones described in the previous paragraph or intersections of the segments of the body that will sometimes arise when two natural motions are interpolated. Those errors will have to be detected and fixed using editing techniques which may

themselves introduce errors in the physical correctness of the motion.

In our analysis we assumed a time warping based on a set of matching keys. A method for performing dynamic time warping is presented in [31]. While our analysis cannot be directly applied to this approach because it violates our assumption of uniform time-scaling, it might be possible to extend the analysis.

In our work we have studied most of the main physical properties. There are, however, other physical properties that we haven't looked at, such as, for example, the fact that the center of pressure should fall within the support polygon of the feet.

The results presented here lead to a number of interesting further questions. First, in situations in which the interpolated motion is not going to be physically correct, we need better guidelines on how much error is acceptable. Reitsma and Pollard took a step in that direction [59] and observed that errors in horizontal velocity were easier to detect than errors in vertical velocity but we need a much more complete understanding of what errors will be perceptible and which will not be noticed. For example, the angular momentum of the interpolated motion is not conserved during flight in the general case. From our experiments, however, even relatively large fluctuations in angular momentum are not noticed by the viewer if they do not result in large changes in angular velocities. A deeper understanding of this observation might be useful in developing better guidelines for interpolation. Similarly, it would be useful to have a guideline for when the discontinuity in the transitions from flight to stance will be visible.

5.8 Appendix A

Given the position of the center of mass of the character, P_{com}, and the values of all joint angles, Q_i, $i = 1..n$, we compute the position of the root of the character, P_{root} as follows: (1) for the given Q_i compute the center of mass of the character, P^0_{com} assuming that the root is at the origin; this calculation gives us the relative position of the root of the character with respect to the center of mass of the character; (2) compute root position: $P_{root} = P_{com} - P^0_{com}$.

5.9 Appendix B

The velocity of center of mass is computed by taking the derivative of the position of the center of mass in equation 5.4 with respect to time. Similarly, acceleration of center of mass is computed by taking the derivative of the velocity:

$$\begin{aligned} V_{com}(t) &= \frac{dP_{com}(t)}{dt} \\ &= \frac{d(wP_{1com}(t_1)+(1-w)P_{2com}(t_2))}{dt} \\ &= wV_{1com}(t_1)\frac{T_1}{T} + (1-w)V_{2com}(t_2)\frac{T_2}{T} \end{aligned} \tag{5.16}$$

$$\begin{aligned} A_{com}(t) &= \frac{dV_{com}(t)}{dt} \\ &= \frac{d(wV_{1com}(t_1)\frac{T_1}{T}+(1-w)V_{2com}(t_2)\frac{T_2}{T})}{dt} \\ &= wA_{1com}(t_1)(\frac{T_1}{T})^2 + (1-w)A_{2com}(t_2)(\frac{T_2}{T})^2 \end{aligned} \tag{5.17}$$

where $w = 0..1$ is the interpolation weight, T_1, T_2 and T are the overall time of phase Φ in motions M_1, M_2 and M. $t_1 = tT_1/T$ and $t_2 = tT_2/T$ are scaled time indices into motions M_1 and M_2.

5.10 Appendix C

To obtain the root position given the center of mass position, the values of all non-redundant dofs and the values of constraints, such as feet position and orientation, we first note that the center of mass of the character can be decomposed into the summation of three quantities: the center of mass for the upper body, P_{upcom}, for the left leg, P_{llcom} and for the right leg, P_{rlcom}. The center of mass of the entire body, P_{com}, is then $P_{com} = P_{upcom}M_{upcom} + P_{llcom}M_{llcom} + P_{rlcom}M_{rlcom}$. In this derivation we assume both legs are in contact but when only one leg is in contact the derivation is very similar. The position of the center of mass of the upper body can be re-expressed in terms of the root position (an unknown) and the center of mass of the upper body assuming the root is at the origin (see Appendix A):

$$P_{com} = (P_{root} + P^0_{upcom})M_{upcom} + P_{llcom}M_{llcom} + P_{rlcom}M_{rlcom} \tag{5.18}$$

The position of the center of mass for a leg can be expressed in terms of the position of the root, the position of a knee joint and the position of the foot joint. For example, for the left leg

$$P_{llcom} = 2\,P_{root} + 2/3\,(P_{lknee} - P_{root}) + 1/2\,(P_{lfoot} - P_{lknee}) + P_{lfcom} \tag{5.19}$$

where P_{lknee} is the position of the left knee joint, P_{lfoot} is the position of the left foot joint and P_{lfcom} is the position of the center of mass of the left foot. P_{lknee} in its turn can re-expressed as a function P_{root} as was shown in [30], leaving us with P_{root} as the only unknown in equation 5.18. Solving the equation for P_{root} (either analytically if possible or numerically if not) will give us the desired result.

5.11 Appendix D

It can be shown that ground reaction force of an interpolated motion computed according to equation 5.10 is a time-scaled interpolation of the ground reaction forces of motions M_1 and M_2 if we compute time T for that contact phase as $T = \sqrt{T_1^2 w + T_2^2(1-w)}$:

$$F^{grf}(t) = F_1^{grf}(t_1)(\frac{T_1}{T})^2 w + F_2^{grf}(t_2)(\frac{T_2}{T})^2(1-w) \tag{5.20}$$

Proof: Assuming a single support polygon, by Newton's second law the ground reaction force, $F^{grf}(t)$ is

$$F^{grf}(t) = \bar{m}A_{com}(t) - \bar{m}G \tag{5.21}$$

where $\bar{m}$ is the total mass of the system, $A_{com}(t)$ is the acceleration of the center of mass of the system and $G = (0, -9.8, 0)$ is an acceleration due to gravity. Substituting equation 5.17 into 5.21 yields:

$$F^{grf}(t) = \bar{m}A_{1com}(t_1)(\frac{T_1}{T})^2 w + \bar{m}A_{2com}(t_2)(\frac{T_2}{T})^2(1-w) - \bar{m}G \tag{5.22}$$

From Newton's second law $\bar{m}A_{1com}(t_1) = F_1^{grf}(t_1) + \bar{m}G$ and $\bar{m}A_{2com}(t_2) = F_2^{grf}(t_2) + \bar{m}G$. Substituting this into the equation above:

$$\begin{aligned} F^{grf}(t) &= (F_1^{grf}(t_1) + \bar{m}G)(\tfrac{T_1}{T})^2 w + \\ &\quad (F_2^{grf}(t_2) + \bar{m}G)(\tfrac{T_2}{T})^2 (1-w) - \bar{m}G \end{aligned} \tag{5.23}$$

Rearranging the terms we have:

$$\begin{aligned} F^{grf}(t) &= F_1^{grf}(t_1)(\tfrac{T_1}{T})^2 w + F_2^{grf}(t_2)(\tfrac{T_2}{T})^2 (1-w) + \\ &\quad \bar{m}G(\tfrac{T_1^2 w + T_2^2(1-w)}{T^2}) - \bar{m}G \end{aligned} \tag{5.24}$$

Because $T = \sqrt{T_1^2 w + T_2^2(1-w)}$, the ground reaction force for the interpolated motion is the interpolation of ground reaction forces from first and second motions:

$$F^{grf}(t) = F_1^{grf}(t_1)(\frac{T_1}{T})^2 w + F_2^{grf}(t_2)(\frac{T_2}{T})^2 (1-w) \tag{5.25}$$

5.12 Appendix E

It is easy to show that for any positive numbers a,b,c and d, if $\frac{a}{b} < \mu$ and $\frac{c}{d} < \mu$ then $\frac{a+c}{b+d} < \mu$.

Adding equations $a < \mu b$ and $c < \mu d$ together we have: $(a + c) < \mu(b + d)$. Now rearranging terms yields: $\frac{a+c}{b+d} < \mu$

Chapter 6

Summary and discussion

In this thesis, we experimented with optimization in two different subspaces: a low-dimensional continuous subspace (Chapter 3) and a compact discrete subspace (Chapter 4). We have shown that the optimization problem becomes much easier to solve when the solution is restricted to one of these compact subspaces. This chapter concludes the thesis with a comparison of the two approaches and a summary of our contributions.

6.1 Comparison of the two optimization approaches

We compare the continuous and discrete optimization approaches on three different axes: search space complexity, generality, and physical correctness.

6.1.1 Complexity of the search space

The continuous space is usually much more complex than the discrete space. In our experiments, it takes five to ten dimensions to represent motions of one behavior. For example, to adequately represent forward walking with different step length behavior, the continuous optimization approach needs seven dimensions. The optimizer takes about 20 minutes to generate a 2 meter walk (about 1.5 second time duration) in this 7-dimensional space. The

discrete approach, on the other hand, needs only eight motions (less than 3 seconds each) to construct a motion graph that represents a the same walking behavior. After compression, the corresponding compact motion graph has about 15 states, and the search explores a few thousand states (with interpolation and global position). The discrete approach requires less than 5 seconds to generate an optimal 2 meter walk.

These run-times are typical for the two approaches when generating short single-behavior motions and the ratio of the run-time is about 250:1. Furthermore, the solution found by the continuous optimization is a local minimum within a low-dimensional subspace which is not guaranteed to be a global minima, while the discrete optimization finds a global minimum within the database. However in our experiments, the local minimum found in the low-dimensional continuous space is usually a natural-looking motion despite not necessarily being a global optima.

If we need to generate a motion involving more than one behavior, then the continuous space can no longer be limited to 7-9 dimensions. As shown in Figure 3.2, the required dimensionality grows as the number of the behaviors increases. Dimensionalities greater than 15 usually prevent the optimizer from converging within reasonable time (approximately 5 hours) or results in an unnatural looking motion. A similar problem arises when synthesizing very long motions with a single behavior. The number of unknowns in the solution becomes too large and the optimizer does not converge within a reasonable amount of time. The number of behaviors and the duration of the desired motion also affects the complexity of the search in the discrete motion but it can handle these complexities better. As was shown in Section 4.4, discrete optimization can generate motions that are 10 seconds long and involve multiple behaviors (in our experiments all the solutions are found in less than 3 minutes).

6.1.2 Generality

The continuous space is more general than the discrete space. As was shown in Section 3.2, the continuous subspace allows the optimizer to generalize well beyond the sample motions. For example, we can synthesize a vertical jump with a 360 degree turn in a subspace

constructed from the examples of three vertical jumps, one of 90 degrees, one of 180 degrees and one of 200 degrees. We can also synthesize many different jumps of varying length and amount of turn from examples of just three jumps: a forward jump, a forward jump with a 90 degree turn and a vertical jump with a 180 degree turn. Because of its generality, optimization in the continuous space can handle a dense set of constraints and can satisfy them exactly.

The discrete space is less general than the continuous space because a solution in the discrete space is constrained to the interpolation of sample points. We can extrapolate only slightly beyond these sample points because substantial extrapolation results in an unnatural solution. For example, a vertical jump with a 360 degree turn synthesized from the same examples of three vertical jumps (90, 180 and 200 degrees) does not look natural. On the other hand, a 250 degree jump can be synthesized successfully. The same holds for forward jumps. In general, only a small amount of extrapolation is visually acceptable (see website for examples).

In order to make the discrete space as general as the continuous space, we need to add more samples. For example, if we add two more vertical jumps—one with a 360 degree turn and one with a 0 degree turn—to the sample set of three vertical jumps described above, then we can synthesize natural looking vertical jumps of all rotations between 0 and 360 degrees using discrete optimization. Our experiments show that given enough examples we can construct a discrete space that is capable of synthesizing the same examples that we synthesized with continuous optimization (see website for examples).

6.1.3 Physical correctness

The continuous space is not limited to just physically correct motions. To ensure that the resulting motion is physically correct, we add a physics constraint to the optimization function. Maintaining the physics constraint and minimizing a physic-based criterion such as energy makes it possible to synthesize motions that are quite different from the sample motions. On the other hand, these constraints make the optimization problem much harder to solve because they make the search space more complex.

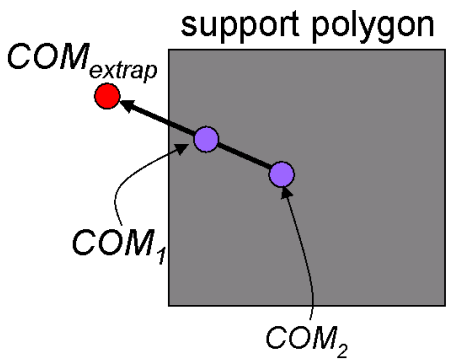

Figure 6.1: Center of mass of an extrapolated motion, COM_{interp}, can move outside of the support polygon. The shaded rectangle represents the support polygon for the motions that are being extrapolated.

The graph that we construct for the discrete space represents only motions that are close to physically correct (as we have shown in Section 4.3.1) and therefore we do not need to maintain physics constraints during search just simply minimize energy. If we add extrapolation to the discrete optimization, however, we can no longer guarantee physical correctness because many of the physical properties that we have proven for interpolated motions will not hold for extrapolated motions. For example, even if the center of mass of a character was inside the polygon of support for two source motions, the extrapolated center of mass may move outside the support polygon (see Figure 6.1). Our experiments indicate that only a small amount of extrapolation can occur before the motion becomes unnatural. Continuous optimization, on the other hand, can produce motions that are a significant extrapolations of the sample motions and are still physically valid.

In conclusion, the discrete space is more compact than the continuous space because it only contains physically correct and natural looking motions. On the other hand, it is less general than the continuous space because it is restricted to interpolation of the sample points used to construct the space.

6.2 Contributions

The main contribution of this thesis is the development of two methods that support intuitive interfaces for animating human characters. Our methods allow the synthesis of physically realistic motions for complex characters such as humans based on only a rough sketch of the desired motion. We hope in the near future to use these methods to create a system that would allow naive users to animate complex characters, such as humans, in an easy and intuitive way.

We have developed two different approaches that both build a compact (reduced-space) representation of the motion based on available motion capture data. Finding a solution in this smaller search space is much easier than finding a solution in the full search space. In addition, the synthesized motion is likely to contain natural coordination patterns. This objective is difficult to describe mathematically and is therefore hard to achieve when searching the full search space.

In the first approach we have shown how to build a continuous low-dimensional representation of the desired motion. With this approach, we were able to synthesize physically realistic motions for a human character that matched rough sketches provided by a user. This work was presented at the ACM SIGGRAPH 2004 conference [64]. In the second approach, we have shown how to build a discrete reduced-space representation of the desired motion. This representation can be viewed as a combination of motion graph and interpolation techniques. The motion that can be generated with this representation is an interpolation of k time-scaled paths through the motion graph. We have shown that the optimization in discrete space supports interactive frame rates (less than 3 minutes to compute a close to an optimal 10 sec long motion) and allows for the synthesis of less dynamic motions and longer motions that are composed of different behaviors.

Another contribution of this thesis is that we have demonstrated that it is possible to search a motion graph using a globally optimal search algorithm, A*. Two contributions made this possible: compression of the motion graph into a practically equivalent but much smaller graph and a search heuristic that guides search well for many human motions. We have demonstrated that the global search is effective by creating long example motions

and showing that the optimal and near-optimal solutions avoid the dithering and inefficient patterns of motion seen in many motion graph implementations.

Finally, we have provided an analysis of the physical correctness of motions created by interpolating a few, presumably physically correct, human motions. We have analyzed the interpolated motion in terms of a number of basic physical properties: (1) linear and angular momentum during flight; (2) foot contact, static balance and friction with the ground during stance; (3) continuity of position and velocity between phases. We assumed that the motions used for interpolation are physically correct themselves, have the same skeleton, can be aligned in time by picking corresponding key events and that linear interpolation is used to interpolate parameters of motions between these key events. Our analysis shows that with a few simple modifications to the straightforward interpolation technique proposed by others, we can prove that these physical properties are satisfied for a wide range of different kinds of motions. The interpolated motion will satisfy these physical properties if the motions used for interpolation do not include significant rotation during the flight phase (runs, forward and vertical jumps, for example), rotate around approximately the same principal axis by approximately the same amount (jumps with turns, for example) or have no flight phase (walks or kicks, for example). This work was presented at the ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2005 conference [63].

Bibliography

[1] The doll interface created at princeton university. (document), 1.1

[2] Motion capture database at carnegie mellon university. 1.2

[3] Yeuhi Abe, C. Karen Liu, and Zoran Popović. Momentum-based parameterization of dynamic character motion. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 173–182, 2004. 2.2

[4] J. V. Albro, G. A. Sohl, and J. E. Bobrow. On the computation of optimal high-dives. In *Proc. IEEE Intl. Conference on Robotics and Automation*, pages 3958–3963, 2000. 2.1

[5] F. C. Anderson and M. G. Pandy. A dynamic optimization solution for vertical jumping in three dimensions. *Computer Methods in Biomechanics and Biomedical Engineering*, (2):201–231, 1999. 2.1

[6] O. Arikan and D. A. Forsyth. Interactive motion generation from examples. *ACM Trans. on Graphics*, 21(3):483–490, July 2002. 1.2, 2.3, 4, 4

[7] Okan Arikan, David A. Forsyth, and James F. O'Brien. Motion synthesis from annotations. *ACM Trans. Graph.*, 22(3), 2003. 1.2, 2.3, 4, 4, 4.3.1

[8] Matthew Brand and Aaron Hertzmann. Style machines. In *Proc. of SIGGRAPH 2000*, pages 183–192, 2000. 2.1

[9] Min Gyu Choi, Jehee Lee, and Sung Yong Shin. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Trans. Graph.*, 22(2):182–203, 2003. 2.3

[10] L. S. Crawford and S. S. Sastry. Biological motor control approaches for a planar diver. In *Proc. IEEE Conference on Decision and Control*, pages 3881–3886, 1995. 2.1

[11] A. Dasgupta and Y. Nakamura. Making feasible walking motion of humanoid robots from human motion capture data. In *Proc. IEEE Intl. Conference on Robotics and Automation*, pages 1044–1049, 1999. 2.1

[12] F. De la Torre and M. J. Black. A framework for robust subspace learning. In *Intl. Journal of Computer Vision*, volume 54, pages 117–142, 2003. 2.1

[13] Anthony C. Fang and Nancy S. Pollard. Efficient synthesis of physically valid human motion. *ACM Trans. on Graphics*, 22(3):417–426, 2003. 2.1, 3.1.3

[14] R. Fourer, D. Gay, and B. Kernighan. AMPL: A mathematical programming language, 1989. 3.1.5

[15] K. Fukunaga. Statistical pattern recognition - 2nd edition. John Hopkins University Press, Baltimore, 1989. 3.1.1

[16] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. Technical Report NA–97–2, San Diego, CA, 1997. 3.1.5

[17] M. Gleicher. Motion editing with spacetime constraints. In *Proc. of the 1997 Symposium on Interactive 3D Graphics*, pages 139–148, Providence, RI, April 1997. 2.1

[18] M. Gleicher, H. Shin, L. Kovar, and A. Jepsen. Snap-together motion: Assembling run-time animation. 2003. 2.3

[19] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proc. of SIGGRAPH 98*, pages 9–20, July 1998. 2.1

[20] S. Guo and J. Roberge. A high-level control mechanism for human locomotion based on parametric frame space interpolation. In *EGCAS '96: Seventh International Workshop on Computer Animation and Simulation*, 1996. 2.2

[21] M. Hardt. Multibody dynamical algorithms, numerical optimal control, with detailed studies in the control of jet engine compressors and biped walking. In *Thesis*. University of California San Diego, 1999. 2.1

[22] M. Hardt. *Multibody Dynamical Algorithms, Numerical Optimal Control, with Detailed Studies in the Control of Jet Engine Compressors and Biped Walking*. PhD thesis, University of California San Diego, 1999. 3.3

[23] G. Huang, J. Lo, and D. Metaxas. Human motion planning based on recurive dynamics and optimal control techniques. In *Proc. of Computer Graphics International 2000*, pages 19–28, 2000. 2.1

[24] Hyun Seok Oh Hyun Joon Shin. Fat graphs: Constructing an interactive character with continuous controls. In *2006 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, September 2006. 2.3

[25] Leslie Ikemoto and David A. Forsyth. Enriching a motion collection by transplanting limbs. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 99–108, 2004. 1.2

[26] Doug L. James and Kayvon Fatahalian. Precomputing interactive dynamic deformable scenes. *ACM Trans. on Graphics*, 22(3):879–887, 2003. 2.1

[27] O. C. Jenkins and M. J. Mataric. Deriving action and behavior primitives from human motion data. In *IEEE/RSJ Intl. Conference on Intelligent Robots and Systems (IROS)*, pages 2551–2556, 2002. 2.1

[28] I. Jolliffe. Principal component analysis. Springer Verlag, 1986. 3.1.1

[29] H. Ko and N. I. Badler. Animating human locomotion with inverse dynamics. *In IEEE Computer Graphics and Applications*, pages 50–59, 1996. 3.1.3

[30] J.U. Korein and N.I. Badler. Techniques for generating the goal-directed motion of articulated structures. In *IEEE Computer Graphics and Applications*, volume 2, pages 71–81, November 1982. 3.1.2, 5.3.1, 5.10

[31] Lucas Kovar and Michael Gleicher. Flexible automatic motion blending with registration curves. In *2003 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 214–224, August 2003. 2.2, 5.7

[32] Lucas Kovar and Michael Gleicher. Automated extraction and parameterization of motions in large data sets. *ACM Transactions on Graphics*, 23(3):559–568, August 2004. 2.2, 5

[33] Lucas Kovar, Michael Gleicher, and Fred Pighin. Motion graphs. 1.2, 2.3, 4, 4

[34] Taesoo Kwon and Sung Yong Shin. Motion modeling for on-line locomotion synthesis. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 29–38, July 2005. 2.3

[35] S. P. Lall, P. Krysl, and J. E. Marsden. Structure-preserving model reduction for mechanical systems. In *Physica D 184*, pages 304–318, 2003. 2.1

[36] Manfred Lau and James Kuffner. Precomputed search trees: Planning for interactive goal-driven animation. In *2006 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, September 2006. 2.3

[37] Manfred Lau and James J. Kuffner. Behavior planning for character animation. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 271–280, 2005. 2.3

[38] Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. Interactive control of avatars animated with human motion data. 1.2, 2.3, 4, 4, 4.2.2, 4.3.1, 4.3.1

[39] Jehee Lee and Kang Hoon Lee. Precomputing avatar behavior from human motion data. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 79–87, 2004. 2.3

[40] Jehee Lee and Sung Yong Shin. A hierarchical approach to interactive motion editing for human-like figures. In *Proc. of SIGGRAPH 99*, pages 39–48, August 1999. 2.1, 3.1.2, 5.3.1, 5.3.1

[41] Y. Li, T. Wang, and H.-Y. Shum. Motion texture: a two-level statistical model for character motion synthesis. *ACM Trans. on Graphics*, 21(3):465–472, 2002. 1.2, 2.1, 2.3, 4, 4

[42] M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems (NIPS) 16*. Cambridge, MA: MIT Press, 2003. 4.2.3

[43] C. Karen Liu and Zoran Popović. Synthesis of complex dynamic character motion from simple animations. *ACM Trans. on Graphics*, 21(3):408–416, 2002. 2.1, 3.3

[44] Z. Liu and M. Cohen. Decomposition of linked figure motion: Diving. In *5th Eurographics Workshop on Animation and Simulation*, 1994. 2.1

[45] Z. Liu and M. F. Cohen. Keyframe motion optimization by relaxing speed and timing. In *6th Eurographics Workshop on Animation and Simulation*, pages 144–153, 1995. 2.1

[46] S. Ménardais, R. Kulpa, F. Multon, and B. Arnaldi. Synchronization for dynamic blending of motions. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 325–335, 2004. 2.3

[47] M. G. Pandy and F. C. Anderson. Dynamic simulation of human movement using large-scale models of the body. In *Proc. IEEE Intl. Conference on Robotics and Automation*, pages 676–681, 2000. 2.1

[48] Sang Il Park, Hyun Joon Shin, Tae Hoon Kim, and Sung Yong Shin. On-line motion blending for real-time locomotion generation. *Computer Animation and Virtual Worlds*, 15(3-4):125–138, 2004. 2.3

[49] Sang Il Park, Hyun Joon Shin, and Sung Yong Shin. On-line locomotion generation based on motion blending. In *ACM SIGGRAPH Symposium on Computer Animation*, pages 105–112, July 2002. 2.3

[50] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984. 4.2.3

[51] A. Pentland and J. Williams. Good vibrations: Modal dynamics for graphics and animation. In *Computer Graphics (Proc. of SIGGRAPH 89)*, volume 23, pages 215–222, August 1989. 2.1

[52] Ken Perlin. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):5–15, 1995. 2.2

[53] N. S. Pollard and P. S. A. Reitsma. Animation of humanlike characters: Dynamic motion filtering with a physically plausible contact model. In *Yale Workshop on Adaptive and Learning Systems*, 2001. 2.1

[54] Jovan Popović, Steven M. Seitz, Michael Erdmann, Zoran Popović, and Andrew P. Witkin. Interactive manipulation of rigid body simulations. In *Proc. of SIGGRAPH 00*, pages 209–218, July 2000. 2.1

[55] Zoran Popović and Andrew P. Witkin. Physically based motion transformation. In *Proc. of SIGGRAPH 99*, pages 11–20, August 1999. 2.1

[56] Katherine Pullen and Christoph Bregler. Motion capture assisted animation: texturing and synthesis. 2.3

[57] S.S. Ravindran. Reduced-order adaptive controllers for fluid flows using POD. In *Journal of Scientific Computing*, volume 15, pages 457–478, 2000. 2.1

[58] P. S. A. Reitsma and N. S. Pollard. Evaluating motion graphs for character navigation. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 89–98, 2004. 4.5

[59] Paul Reitsma and Nancy Pollard. Perceptual metrics for character animation: Sensitivity to errors in ballistic motion. *ACM Transactions on Graphics*, 22(3):537–542, August 2003. 5.2.1, 5.7

[60] Charles Rose, Michael F. Cohen, and Bobby Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics & Applications*, 18(5):32–40, September 1998. 2.2, 5, 5.1, 5.3.1

[61] Charles F. Rose, Brian Guenter, Bobby Bodenheimer, and Michael F. Cohen. Efficient generation of motion transitions using spacetime constraints. In *Proc. of SIGGRAPH 96*, pages 147–154, August 1996. 2.1

[62] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice-Hall, 2003. 4.2.3

[63] Alla Safonova and Jessica K. Hodgins. Analyzing the physical correctness of interpolated human motion. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 171–180, 2005. 1.2, 6.2

[64] Alla Safonova, Jessica K. Hodgins, and Nancy S. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Trans. Graph.*, 23(3):514–521, 2004. 2.1, 6.2

[65] M. Santello, M. Flanders, and J. F. Soechting. Patterns of hand motion during grasping and the influence of sensory guidance. In *Journal of Neuroscience*, volume 22, pages 1426–1435, 2002. 2.1

[66] Arno Schödl, Richard Szeliski, David H. Salesin, and Irfan Essa. Video textures. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 489–498, July 2000. 2.3

[67] Madhusudhanan Srinivasan, Ronald A. Metoyer, and Eric N. Mortensen. Controllable real-time locomotion using mobility maps. In *GI '05: Proceedings of the 2005 conference on Graphics interface*, pages 51–59, 2005. 2.3

[68] Adnan Sulejmanpašić and Jovan Popović. Adaptation of performed ballistic motion. *ACM Trans. Graph.*, 24(1):165–179, 2005. 2.1

[69] Mankyu Sung, Michael Gleicher, and Stephen Chenney. Scalable behaviors for crowd simulation. *Computer Graphics Forum*, 23(3):519–528, September 2004. 2.3

[70] Mankyu Sung, Lucas Kovar, and Michael Gleicher. Fast and accurate goal-directed motion synthesis for crowds. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 291–300, July 2005. 2.3

[71] D. Tolani, A. Goswami, and N. Badler. Real-time inverse kinematics techniques for anthropomorphic limbs. In *Graphical Models 62 (5)*, pages 353–388, 2000. 3.1.2

[72] Jing Wang and Bobby Bodenheimer. Computing the duration of motion transitions: an empirical approach. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 335–344, 2004. 2.2

[73] Douglas J. Wiley and James K. Hahn. Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics Applications*, 17(6):39–45, 1997. 2.2

[74] Andrew Witkin and Michael Kass. Spacetime constraints. In *Computer Graphics (Proc. of SIGGRAPH 88)*, volume 22, pages 159–168, August 1988. 1.1, 2.1, 3

[75] K. Yamane and Y. Nakamura. Dynamics filter – concept and implementation of online motion generator for human figures. In *Proc. IEEE Intl. Conference on Robotics and Automation*, pages 688–695, 2000. 2.1