Evaluating Data Driven Character Animation

Paul S. A. Reitsma

CS-CMU-06-161
October 2006

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Nancy S. Pollard, Chair
Jessica K. Hodgins
James J. Kuffner
Michiel van de Panne (UBC)

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 no. IIS-0326322.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: computer graphics, animation, experimentation, measurement, reliability, motion capability, capability metrics, motion capture, human motion, motion graphs, motion graph embedding, editing model

Abstract

Humanlike characters are an important area of computer animation. These characters are vital components of many applications, such as computer games, cinematic special effects, virtual reality training, and artistic expression. There are two main challenges to animating humanlike characters. First, our daily familiarity with human motion makes creating realistic humanlike motion particularly difficult, since viewers have an extremely high sensitivity to artifacts or errors in human motion; this is the problem of producing animation of sufficient quality. Second, animators require a motion generation system that can produce all motions required to animate the character through the full range of tasks in the target application; this is the problem of ensuring the motion generation system has sufficient capability.

While several technologies have been developed to address these problems, our understanding of such technologies is still developing. Accordingly, techniques for producing high-quality animations—such as motion capture—and for producing high-capability animation systems—such as motion graphs—typically rely heavily on the intuition and abilities of a skilled animator to achieve acceptable results, and hence are difficult to use or automate. The goal of this work was to augment the animator's intuition by providing ways to more objectively quantify and measure a system's capability to produce the required motion, and the quality to viewers of the motion so produced.

We present a technique for evaluating motion quality by measuring user sensitivity to editing-based artifacts in captured motion, derive practical animation guidelines from the results, and discuss several interesting systematic trends we have uncovered in the experimental data. We also present an efficient technique for evaluating the capability of a motion graph to fulfill the requirements of a given scenario, along with an examination of capability deficiencies it uncovers and the effect on those deficiencies of some methods commonly used to improve motion graphs. Finally, we propose a wide range of extensions and applications enabled by these new evaluation tools, such as automatic optimization and vetting of motion graphs.

Acknowledgments

For Markus, who would have appreciated the complexities inherent in adding another D.

For Dave, who would have appreciated the heft of this thing when rolled up tightly.

For Schelte and Albert, who would (I hope) have been pleasantly surprised by their role.

For family and friends who have not yet abandoned me despite my years cloistered in an ivory tower. Prior work [Rapunzel et al., 1812] suggests that perhaps I should have forgone the haircut.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Animation Applications

Character animation has a prominent role in many applications, from entertainment in games or movies to safely training for dangerous situations in virtual reality to expanding the range of artistic expression. Realistic and directable humanlike characters are an ongoing goal in computer animation, playing vitally important roles in all types of applications that use computer animation.

One very common use of such animated characters is in computer games such as "Prince of Persia: the Sands of Time", "Madden NFL 2004", or "Max Payne 2", to name a few examples. As with many games, these titles focus the player's attention on human characters for the large majority of the playing experience. Similarly, many training scenarios, such as the first-responder trainer "Project Biohazard" [Brooks et al., 2003] or the military virtual reality conflict-resolution training "Mission Rehearsal Exercise" [Hill et al., 2003] rely on interaction with realistic humanlike characters. Many special effects for movies or television require realistic and directable humanlike characters, such as the background character scenes now common to many movies, or the foreground characters in such movies as "Spiderman 2" or "Van Helsing". Artistic endeavors are often similar; even such a highly stylized piece as "Ryan" (Chris Landreth, 2004) makes use of real-

istic human motion as a counterpoint to the fantastical visual effects. Some applications even fill multiple roles, such as the US military's "Full Spectrum Warrior", which is used for entertainment, recruitment, and training. In all of these applications, realistic human motion is important to create an immersive and engaging result. Similarly, directable characters are vital for interactive applications where no human intervention is possible, and tremendously useful for authoring in other situations.

1.1.1 Challenges of Animation

Animating such realistic and directable characters is a particularly challenging goal; because we are so intimately familiar with human motion from our day-to-day lives, even untrained viewers have strict requirements for what they will consider to be realistic human movement. Our sensitivity to subtle details of human motion is very high [Kozlowski and Cutting, 1977], meaning seemingly minor details of human motion carry significant information and impact.

A character with unrealistic motion or whose motion contains details that are ill-suited to the scenario can detract significantly from the goal of the application. Entertainment applications—which often rely on engaging and immersing the viewer—suffer when a motion appears implausible or ill-suited, and training applications—particularly those which require the emotional involvement of the trainee (e.g., Hill et al. [2003])—may suffer similar reductions in effectiveness. Possibly worse, a training application meant to teach skills with a physical component could be unreasonably hard to learn from or could even teach bad practices if the animated character's motion cannot be made sufficiently similar to the motion of the human teacher from which the lesson was originally derived. Similarly, inappropriate subtle details to a motion can adversely affect the artistic or emotional impact of an animation, necessitating labor-intensive manual alteration of sub-standard motions.

Interactive or online applications have an additional requirement, namely that the animated character must have the capability to perform all required tasks; in other words, the requirement that a user interacting with the system can cause the character to undertake the desired motion at the proper time and without undue delay. For example, a ball-carrier

Figure 1.1: An actor prepared for a motion capture session with bodysuit and reflective markers.

in a football game should always be able to turn or sidestep to avoid a tackle; if the character lacks the capability to undertake evasive motion in a timely manner, the game will be frustrating to play and is likely to be less successful commercially.

1.2 Motion Capture and Motion Graphs

One promising technology for acquiring high-quality human motion that has become available in recent years is *motion capture*. In a nutshell, motion capture directly records the motion of an actor for later playback through an animated character. For optical motion capture—the most common type—the actor is prepared with several dozen reflective markers placed on specific locations on the body (see Figure 1.1). A set of 6–12 calibrated near-infrared cameras track the 3D position of these markers at 30–120Hz, after which this raw positional data is combined with anatomical knowledge about the character to generate a sequence of character poses that—when played back at the same rate as the original data capture—will result in a very close approximation of the actor's movements. This pose information is stored in joint angle format (i.e., the left elbow is at X degrees at frame 43, at Y degrees at frame 44, and so on).

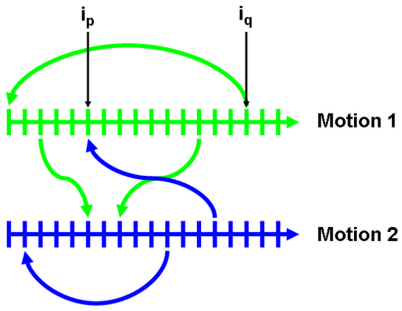


Figure 1.2: A motion graph is formed from the additional transitions added between frames of motion capture data. The frames of Motion 1 from $\mathrm{i_p}$ to $\mathrm{i_q}$ represent one clip in the motion graph. In this diagram, each vertical hashmark is a frame of motion; connecting arrows represent these additional transitions.

Typical accuracy for 3D tracking of motion capture markers is a few millimeters; combined with the known anatomical restrictions of the situation (e.g., the lengths of the actor's bones remain constant), this level of accuracy allows detailed reconstruction of the actor's motion. The result is a portrayal of the original actor that is sufficiently accurate and captures enough subtle details that observers can typically distinguish between different actors—or recognize a familiar actor—quickly and easily.

1.2.1 Motion Graphs

The intention of a motion graph is to allow additional flexibility when using motion capture data. Normally, motion capture data is played sequentially; to replay the motion of clip i, the animation transitions from frame i_k to frame i_{k+1} for $k = 1, 2, ...n - 1$. Motion graph data structures take motion capture data, break the motions into short clips, and allow the clips to be pieced back together in any order and combination that does not require transitions between clips whose measured quality is below a certain threshold. To create a motion graph, the data is examined to find additional transitions of the form i_k to j_m (i.e., clip i, frame k to clip j, frame m) that will result in acceptable motions. One common

	Automatic Motion Graph	**Hand-Tuned Network**
Setup	Fast, automatic	Slow, manual
Transitions	Natural	Hand-tuned
Capabilities	Not known	Known by design

Table 1.1: Comparison of automatic motion graphs vs. hand-designed motion networks.

criterion for selection of such transitions is that frames i_k and j_m are sufficiently similar (e.g., the sum of the squared differences between the joint angles and joint angle velocities of the two characters is below a threshold).

A motion graph is formed by considering these additional transitions to be the edges of a directed graph (see Figure 1.2), with nodes in the graph representing *clips* (sequences of frames extending between two of these additional transitions within an original source motion; i.e., frame i_p to frame i_q). Traversing this graph consists of playing a sequence of these clips, with all clip-to-clip boundaries being at one of these additional transitions. For example, a clip of a character walking three steps might be broken into three clips—one for each step—and a five-step sequence could be synthesized by piecing together the three steps in the order 12321, 32321, or even 12121. To allow continuous motion generation, which is crucial for applications where the entire motion plan is not known in advance (e.g., interactive applications), only the largest strongly connected component (SCC) of the motion graph is used.

Motion graphs hold much promise for achieving the goals we have set out for animation generation systems. One particular advantage is their direct reliance on motion capture, meaning that motion graphs can easily benefit directly from the skills of a particular actor, capturing the nuances of his or her performance to generate realistic motion evoking the desired responses in the viewer. When coupled with motion capture and motion editing, motion graphs have the potential to allow automatic and efficient construction of realistic character animations in response to user or animator control.

Unfortunately, many of these potential benefits are difficult to realize with motion graphs. In particular, motion graphs often suffer from a lack of flexibility in terms of character capabilities. This problem is sometimes addressed with hand-tuned motion networks [Mizuguchi et al., 2001], which are carefully designed to give characters the necessary motion capabilities. These networks, however, have the disadvantages that they require extensive manual effort on the part of skilled animators, and can suffer from less natural transitions than an automatic motion graph would have, due to the latter's ability to automatically process very large motion databases and pull from the database appropriate transitions that the actor actually performed. Table 1.1 gives a summary of the comparison between automatically-generated motion graphs and hand-tuned motion networks.

1.2.2 Choosing Motion Graphs

We chose to work with motion graphs due to their following advantages:

1. Leverage and preserve inherent realism of motion capture
2. Largely preserve subtleties of actors' performances
3. Function effectively for many different types of actions
4. Large body of supporting work
5. Modest computational and resource requirements
6. Modest requirements for animator time and effort
7. Modest requirements for animator skill

In particular, we believe that motion graphs can offer an excellent mix of easy motion creation for naive users and actor-derived control over motion subtleties for more demanding tasks.

One reason we used motion graphs in our research is due to a long-term goal of providing guaranteed-good online motion generation. Motion graphs stitch together high-quality

motion capture data, meaning that motion quality and motion-generation capability can be somewhat decoupled in the system, and hence can be reduced to some extent into simpler sub-problems. We address these two sub-problems in the two evaluation threads of our research.

Additionally, we selected motion graphs for the initial implementation of our work because they are well-supported by high-quality research from many other researchers, they tend to produce good results, and they are fast and efficient to use. All of these contribute to our final reason, which is that motion graphs and similar technologies are already in use in several applications which we believe would benefit from our work.

It is worth noting that, while this research deals only with motion graphs, it could be applied to the other motion generation methods, at least in a general sense. Any motion generation method could be used to create sample motions from which a motion graph could be formed, and this motion graph could be evaluated as we describe (although the capabilities and task space coverage of the other motion generation method might be underestimated, depending on the quality of the sample motions generated). Additionally, results from studies on the perceptual quality of single motions will need to be reinterpreted for different types of motion editing used by each motion generation method to ensure the correctness of the algorithm we describe.

1.3 Motion Editing

Motion capture data, while detailed and realistic, is also extremely inflexible; by itself, the data can only be replayed to animate exactly the motions the original actor performed. Due to this rigidity, virtually every application that uses motion capture data edits and changes that data because the recorded motion clips need to be tweaked to fit the requirements of the particular scenario. Even very simple applications, such as making a character walk forward five meters, will require motion editing unless precisely that action was captured.

1.3.1 Types of Motion Editing

There are four major categories of motion editing:

1. Splicing two clips together
2. Warping a single clip
3. Interpolating between multiple clips
4. Transplanting body parts from one clip into another

The key motion editing techniques for motion graphs are splicing and warping, so our work focuses on those approaches. Future work involving alternative motion generation algorithms may place more emphasis on interpolating or transplanting as methods of modifying clips.

Splicing

Splicing motion capture data refers to combining two motion clips. This edit involves taking two clips of motion capture data and attaching the beginning of one motion to the end of the other. The result is a longer animation than either input clip, possibly creating an overall motion that the original actor never performed. For example, splicing a two-meter walk onto the end of another two-meter walk would produce a four-meter walk, even if the actor was only ever recorded walking two meters at once. Similarly, animating a character for an extended period of time requires splicing together motion clips, otherwise the animation driving the character will hit the end of the clip and simply stop.

Splicing motion clips together in this way tremendously increases the range of animations possible from a collection of motion capture data, and is the key technique behind motion graphs. Accordingly, splices are an indispensable type of motion editing.

Warping

Warping motion capture data refers to altering details of a single motion clip. For example, one could take a motion clip that makes the character walk directly forward two meters and warp it into a motion clip where the character walks 1.8 meters at an angle of ten degrees from forward.

This type of motion editing is crucial for having a character perform tasks as specified by the animator. Raw motion capture data—even when pieces are spliced together—is limited to replaying what the actor originally performed. If the actor was captured reaching for an object at waist height and at head height, but was not captured reaching for an object at shoulder height, no amount of splicing or replaying of the original data will produce that shoulder-height reach. Warping the head-height reach clip, however, can produce the required shoulder-height reach. Warping of motion clips is similarly important for locomotion to a specified target point, and for almost all similar tasks where motion must be generated that closely fulfills requirements set by the animator. Accordingly, motion warping is also an indispensable type of motion editing.

Interpolating

Interpolating motion capture data refers to combining two or more captured motions to create a new motion that is different from any of the sources. For example, one could take a motion clip that makes the character punch at head height and a motion clip that makes the character punch at chest height, and interpolate between to make a new motion clip that makes the character punch at shoulder height. Interpolation and warping share many similarities, and interpolation is valuable in many of the same circumstances.

Transplanting

Transplanting refers to taking motion for different parts of the character's body from different motion clips, and is used to enrich a motion database. For example, a clip of a character running and a clip of a character catching a ball can be combined by transplant-

ing the ball-catching torso onto the running legs to create a new clip of a character catching a ball while running.

1.3.2 Implications of Motion Editing

Motion graphs rely heavily on splicing clips together to form the additional transitions among the source motions, and on motion warping to make the resulting motions general enough to fulfill task requirements. This need for splicing transitions and warping motion clips means that motion graphs require extensive use of motion editing; unfortunately, that motion editing is problematic because it undermines the key benefit of motion capture data: the realistic portrayal of the original actor's motion. Due to the way motion editing changes both large and small details of a motion captured clip, there is no longer any assurance that the resulting motion will be at all realistic or acceptable to a viewer.

The benefit of motion editing is that the character's capabilities are greatly enhanced, making it possible for motion capture data to be used to achieve animator (or user) goals for a character's motion.

These two effects of motion editing largely oppose each other, creating a tension that must be carefully understood and controlled as the animator tries to deal with the twin problems of motion quality and motion capability.

1.3.3 Perceptible Flaws in Motion Quality

While measurement error of the motion capture markers and approximation error incurred when fitting human motion to a simplified skeletal representation both result in less-than-perfect human motion, the largest and most perceptually important problems in most applications that use motion capture stem from motion editing. Motion editing techniques can often introduce artifacts into the motions they produce, and such artifacts can be detectable by and even disturbing to viewers. Figure 1.3 shows how a mismatch when splicing together motion capture data can lead to an artifact. It is not yet well understood how much motion editing can be applied to a motion before artifacts are introduced that violate the

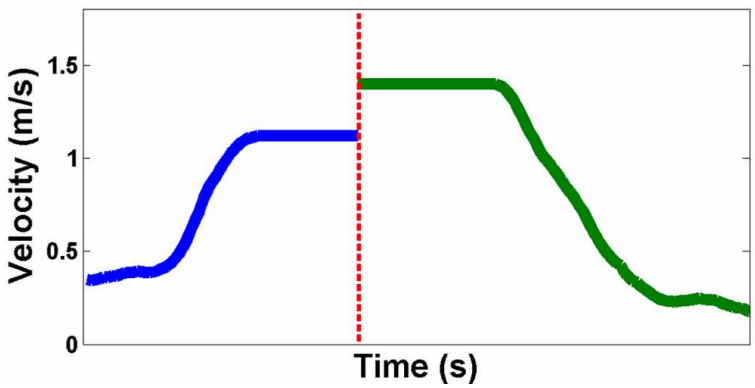


Figure 1.3: An example of an editing-caused artifact in a clip. The two motions being spliced are not sufficiently similar in this data channel, and the result—even after smoothing—will be jerky motion.

particular application's quality and verisimilitude requirements, and motion editing is currently applied largely according to rules of thumb. Two common examples of such editing errors are footsliding and jerkiness in the motion.

The common approach taken to examine the quality of motion produced by an animation system is for the animator to become familiar with the system and use their intuition to judge the applications in which the motion should be acceptable. Unfortunately, for most applications the animator is not able to exhaustively test the range of motions the system could produce, leading to the constant threat of motions being produced that are far from acceptable under the application's requirements. Additionally, typical viewers of animation may perceive an animated character's motion differently than skilled animators do, much like a skilled martial artist will understand a movie fight scene differently than an average viewer. Accordingly, the animator's intuition may be too harsh, throwing out acceptable animations and wasting resources, or may simply be inappropriate to judge what will or will not be acceptable to actual average viewers.

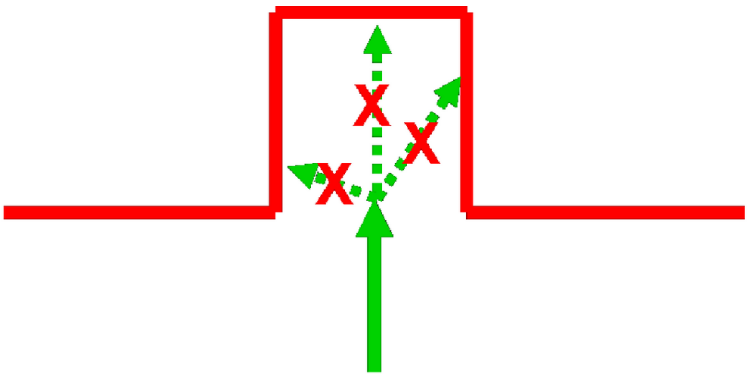


Figure 1.4: An abstract view of a character reaching a dead end in the environment due to a lack of appropriate motion capability.

1.4 Gaps in Motion Capability

Even before the animator can worry about how viewers will perceive the quality of a motion, that motion must first be created by the animation system. If a required motion cannot be created without using excessive motion editing, the animator may be faced with an unpleasant choice between a bad-looking motion and no motion at all. Even worse is when this dilemma comes up during interactive motion creation applications—such as games—when no animator is present to make the best choice in a bad situation. One example of this problem is the motion graph lacking the movements required to navigate the character into and out of a restricted location in the environment, such as a narrow passage (see Figure 1.4); a second example is the motion graph lacking sufficient flexibility to have the character follow natural and reasonable paths within a cluttered environment (see Figure 1.5).

Again, addressing this problem commonly relies heavily on animator intuition and skill. A typical approach is for the animator to carefully hand-tune or even hand-craft a system which their intuition tells them can create all the necessary motions. This suffers from the same pitfalls, however, of the animator not being able to exhaustively test the system's ability to produce adequate character capabilities, leading to the risk of necessary motions being unavailable. Generally, unpleasant motions are considered preferable to

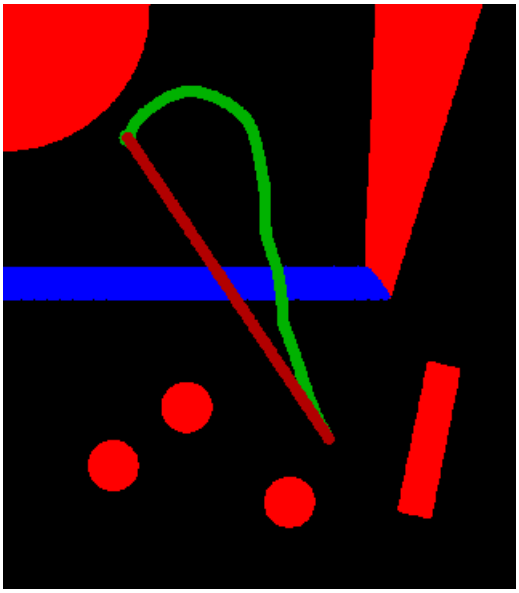


Figure 1.5: An overhead view of a cluttered environment showing two paths between a randomly chosen start and end point. The best path available to the character using the motion graph (green) is unnaturally circuitous as compared to the ideal reference path between start and end positions (red).

unavailable motions, and this risk is avoided by unconstrained use of motion editing, with a predictable effect on motion quality.

One of the key parts of this major stumbling block is a lack of understanding of the space of motions created by the motion graph data structure; it is not known with any confidence what a character can or cannot do when animated by a particular motion graph in a particular environment. Due to this lack of knowledge of a motion graph's capabilities, they are not reliable enough for many applications, especially interactive applications where the character must always be in a flexible and controllable state regardless of the control decisions of the user. Even if a particular motion graph does happen to fulfill all our requirements, that reliability will go unknown and – for a risk-averse application – unused without a method to evaluate and certify the capability of the motion graph.

Chapter 2

Approach

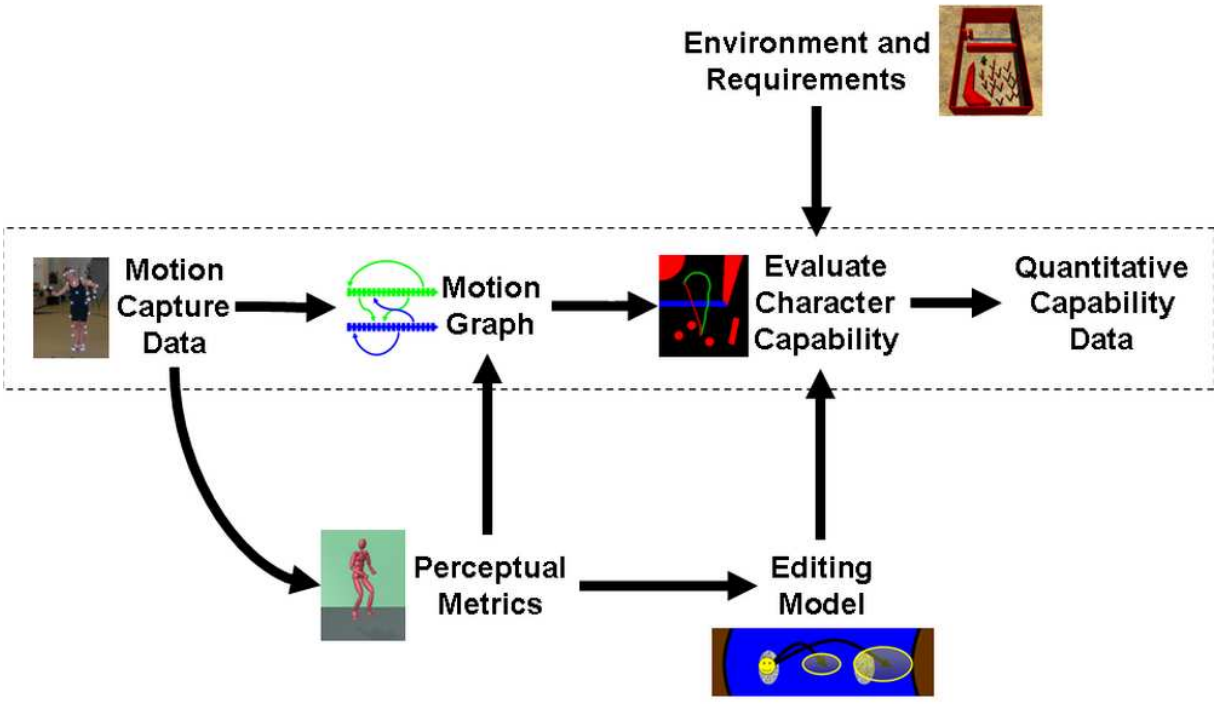


Figure 2.1: A schematic diagram of the overall evaluation approach.

Figure 2.1 shows a diagram of our approach. Motion capture provides the source

motions for our evaluations. Our Perceptual Metrics approach (Chapter 4) allows us to examine the perceptual quality of motion necessary for humanlike or simple object characters to effectively depict realistic motion of one particular type. We use this approach to build insight into how carefully details of a motion must be controlled for different applications, and how this can be exploited for practical animation tasks (arrows to Motion Graph and Editing Model in Figure 2.1). Our method for evaluating the capability of a motion graph (Evaluate Character Capability in Figure 2.1) permits us to evaluate how well-suited a particular motion graph is for a particular *scenario*, by which we mean the motion task space, specific environment, and animator requirements for tasks the character must perform in a particular application. This method provides insight on what properties of motion graphs are necessary to ensure desirable properties such as the ability to animate a character with sufficient motion capability to perform all required tasks. This capability evaluation uses the motion graphs and editing models informed by the perceptual metrics evaluation, as shown in Figure 2.1. We conclude with a selection of future research directions enabled by this work (Chapter 6), representing directions this work might be extended either separately or by using the two evaluation techniques in conjunction.

2.1 Problem Statement

We investigated the following key questions:

1. How does one measure the subjective quality of a motion clip?
    - How can those measurements be used to derive practical animation guidelines?
    - Do differences in character type between a human figure and a simple rigid body affect a user's sensitivity to errors in that motion?
2. How does one measure the capability of a motion graph to fulfill a scenario's requirements?
    - What is a functional and actionable definition of capability?

- How can this evaluation be efficiently applied to the large scenarios practical use demands?
- What specific deficiencies in capability can be identified from the resulting measurements?
- How well do common techniques resolve these deficiencies?

Chapter 3

Related Work

The two main threads of our work – evaluating a motion's subjective quality and evaluating a motion graph's capability – draw largely from substantially different bodies of related work, and so we address them individually. For each of the two threads, we examine important prior or background work in the area and how that work is used by or differs from our work.

3.1 Motion Quality

The first of our key questions is: how does one measure the subjective quality of a motion clip?

A substantial quantity of work in multiple fields has addressed the problem of understanding perception of human motion (Section 3.1.1); however, that work does not address the notion of the subjective *quality* of the motion. Work evaluating the quality or realism of a motion has been undertaken (Section 3.1.2), but generally in terms of rigid bodies and similar objects; only recently has such work addressed the problem of evaluating the quality of a human motion (Section 3.1.3).

Our work is one of a few studies that examines the questions of how perceptual sensi-

tivity to errors in full-body motion varies with changes in parameters such as error magnitude, and how that information can be applied to create practical guidelines for animators. We focus on the ballistic motion of human and rigid-body characters.

3.1.1 Perception of Human Motion

Light dot experiments, in which small points of light attached to key areas of a human's body are the only information given to the subjects about that human's motion, have yielded interesting insights. Researchers have found that observers can make very fine discriminations when presented with a sparse representation of actual human motion, for example accurately estimating lifted weight [Runeson and Frykholm, 1981] and pulled weight [Michaels and de Vries, 1998] from light dot displays. It is argued that a high level of performance is possible because the relevant complexity of the motion is small; the weight of the manipulated object correlates well with observable parameters such as elbow velocity [Bingham, 1987] or center of mass position and velocity [Michaels and de Vries, 1998].

Brain imaging studies (e.g., Grossman et al. [2000], Pelphrey et al. [2003]) carried out with point light displays demonstrate that there are differences in the regions of the brain activated when subjects view biological motion as compared to coherent non-biological motion. We were motivated to examine whether this insight might play a role in the sensitivity of viewers to errors in animated motion.

3.1.2 Evaluation of Motion Quality

Perception of physically unrealistic motions has been studied extensively for rigid bodies and simple mechanisms. It is generally acknowledged that people perform poorly on abstract physical reasoning tasks [Proffitt, 1999], even about something as apparently simple as constant velocity (Michotte [1963], Cohen [1964], Runeson [1974]), although there is evidence that richer stimuli can improve performance for some tasks [Stappers and Waller, 1993].

Animation has also been shown to improve performance [Kaiser et al., 1992, Hecht and Bertamini, 2000], but good performance is observed only for simple motions. In the study of Kaiser et al., for example, animation improved performance for discriminating correct and incorrect ballistic motion of a spherical body, but anomalies in a spinning motion that involved changing inertia (a satellite with extending and retracting solar panels) were not identified unless the spinning motion completely stopped or reversed direction. Degradation of performance with complexity was also noted by O'Sullivan and Dingliana [2001], who showed that anomalies in collisions between complex objects were more difficult to detect than anomalies in collisions with spherical objects.

3.1.3 Evaluation of Quality of Human Motion

Hodgins et al. [1998] varied a synthetic running motion and were able to show that for the types of variations tested—torso rotation, arm swing magnitude, or additive noise—subjects were more sensitive to these variations when the character was rendered using a polygonal model than when a stick figure was used for rendering. Oesker et al. [2000] showed that perceived skill level of animated soccer players increased when more detailed and realistic motion was used for the players.

Wang and Bodenheimer used viewer perception of animation quality to validate their transition-point selection method [2003], and to examine the effect of transition length on motion quality [2004]. Harrison et al. [2004] studied the ability of viewers to detect changes of length during animation under the influence of differing attention levels. Recently, researchers have compared the results of motion-naturalness classifiers to the classifications of users [Ren et al., 2005, Ikemoto et al., 2006, Arikan et al., 2005]. McDonnell et al. [2006] studied how user preference varied with level-of-detail simplification of clothing simulations for characters in animated crowds.

3.2 Motion Capability

The second of our key questions is: how does one measure the capability of a motion graph to fulfill a scenario's requirements?

Many techniques for generating animation have been examined by researchers (Section 3.2.1), although none have been able to completely solve the problem of creating human animations. For the reasons given in Section 1.2.1, we are using motion graphs as the testbed for our evaluation system. However, motion graphs have several important deficiencies, such as a poor ability to respond rapidly to interactive control or to create the precise motion sequence requested. A number of researchers have looked at addressing these problems (Section 3.2.2), but our work is unique in that it is the first systematic evaluation of the problems associated with motion graphs.

Evaluating a motion graph requires taking the details of the environment in which it is to be used into account. Several researchers have developed methods for taking the environment into account for an animation system (Section 3.2.3); however, our evaluation goals required a new algorithm that would allow the full capabilities of the motion graph to be examined efficiently enough to be used in large and complex environments.

3.2.1 Animation Generation Techniques

Many techniques have been suggested to address the problem of creating high-quality animated characters which are capable of fulfilling user requirements. Unfortunately, none of the methods currently available have done so with complete success.

Motion graph approaches to motion generation (e.g., Molina-Tanco and Hilton [2000], Lee et al. [2002], Kovar et al. [2002a], Arikan and Forsyth [2002], Li et al. [2002], Kim et al. [2003], Arikan et al. [2003], Kwon and Shin [2005], Lee et al. [2006]) maintain much of the realism and quality of the underlying motion capture data, but often suffer from poor controllability due to their use of discrete motion clips.

Planning approaches to motion generation (e.g., Kalisiak and van de Panne [2001], Hsu et al. [2002], Julien Pettre Jean-Paul Laumond [2003], Yamane et al. [2004]) create

a collision-free path through an environment with arbitrary obstacles by searching the configuration space of the system. Heuristic planning systems can successfully operate on systems with high degree of freedom, but such systems can be difficult to construct and control, computationally expensive, and even when informed by motion capture data can fail to find a natural-looking solution.

Controller approaches to motion generation (e.g., Wooten and Hodgins [2000], Faloutsos et al. [2001], Zordan and Hodgins [2002], Zordan et al. [2005]) can conceivably generate arbitrary motion that conforms to the laws of physics to the extent that the animator specifies. However, controllers are often complex and difficult to construct, even for experienced users, and are often only reliable and stable within a relatively narrow space of motions.

Optimization approaches to motion generation (e.g., Witkin and Kass [1988], Gleicher [1997], Lo and Metaxas [1999], Popović and Witkin [1999], Liu and Popović [2002], Fang and Pollard [2003], Safonova et al. [2004]) again can conceivably generate arbitrary motion conforming to arbitrary physical laws. Motion that is largely specified by the laws of physics (such as jumping) are particularly well-suited to optimization techniques as they are already highly constrained. Motions that are less heavily dependent on the body's dynamics (such as gesturing) require possibly very complicated user-specified constraints to force the solution to move in the desired manner, and may be less well-suited to optimization methods. Additionally, there is no guarantee that the optimization will converge to an acceptable solution.

Interpolation approaches to motion generation (e.g., Wiley and Hahn [1997], Rose et al. [1998], Rose. et al. [2001], Sloan et al. [2001], Zordan and Hodgins [2002], E. Drumwright [2004], Kovar and Gleicher [2004], Mukai and Kuriyama [2005]) ensure that motion can be generated for any point in a parameter space. However, there are situations where interpolation will produce poor results if there are too few samples (e.g., ballistic motion [Reitsma and Pollard, 2003]). To assist artists in creating an interpolation function that produces pleasing results, Sloan and his colleagues [Sloan et al., 2001, Rose. et al., 2001] provide an interactive editing environment for adjusting example motions, creating new examples, and placing the new examples in the parameter space.

3.2.2 Motion Graph Manipulation

Some previous research has looked at the problem of altering motion graphs to overcome their disadvantages. Mizuguchi et al. [2001] examined a common approach used in the production of interactive games, which is to build a motion-graph-like data structure by hand and rely heavily on the skills and intuition of the animators to tune the results. Gleicher et al. [2003] developed tools for editing motion data in a similar manner, in particular the creation of transition-rich "hub" frames. Lau and Kuffner [2005, 2006] use a hand-tuned hub-based motion graph to allow rapid planning of motions through an environment. Ikemoto et al. [2006] use multi-way blending to improve the transitions available within a set of motion capture data. Recently, some research has examined the problem of how to make motion graphs more responsive at a local level, allowing limited interpolation [Shin and Oh, 2006, Heck and Gleicher, 2006].

3.2.3 Environment Effects on Motion Graphs

It can be advantageous to embed a motion graph into a particular environment (e.g., see Figure 3.1). Embedding a motion graph into an environment reveals how the specific features of that environment will affect the properties and capabilities of the motion graph.

Four other projects have considered embedding a motion graph into an environment in order to capture its interaction with objects. Research into this area was pioneered by Lee et al. [2002]. Their first approach captured character interactions with objects inside a restricted area of the environment. A second approach [Choi et al., 2003] demonstrated how to embed a portion of a motion graph into an environment by unrolling it onto a roadmap [Kavraki and Latombe, 1998] constructed in that environment. Suthankar et al. [2004] embedded a motion graph to model the physical capabilities of a synthetic agent. Lee et al. [2006] embedded motion data into repeatable tiles.

Our evaluations required a new approach which would embed the full set of options available to the motion graph into all parts of the environment. In order to make embedding large motion graphs into large environments a tractable task, we use a grid-based approach

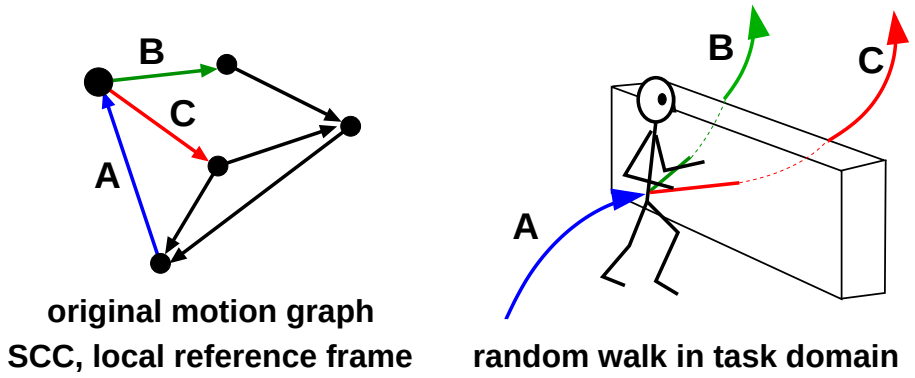


Figure 3.1: A motion graph may be constructed using a reference frame local to the character to preserve flexibility in the character's motion. However, driving the character through this motion graph in an environment with obstacles can result in dead ends that might thwart a local planner. In other words, forming a strongly connected component (SCC) in original motion graph does not guarantee that the character can wander through a given environment indefinitely.

inspired by grid-based algorithms used in robotic path-planning (e.g., [Latombe, 1991, Lozano-Pérez and O'Donnell, 1991, Donald et al., 1993]).

3.3 Differences and Contributions

This work is, to our knowledge, first in several ways:

1. First to measure perceptual quality of full-body human motion with a goal of producting concrete animator guidelines for amount of allowable error [Reitsma and Pollard, 2003]

2. First to derive practical guidelines for animation of human characters from perceptual data of human motion [Reitsma and Pollard, 2003] [1]

[1] Additional work in this area has since appeared [Harrison et al., 2004].

3. First to define and evaluate the capability of a motion graph to fulfill scenario requirements [Reitsma and Pollard, 2004]

The main contributions of this work are the following:

- **A statement of the problem of evaluating the perceptual magnitude of errors in animated human motion.**

We demonstrated how to measure the perceptual quality of an animated human motion, and how to derive practical animation guidelines from those results. Our experiments uncovered a number of systematic trends in user sensitivity to errors in animated human motion, including significant differences between error types, error directions, and character types.

- **A statement of the problem of evaluating global properties of motion generation algorithms.**

We provided a concrete method for evaluating global task-space properties for motion graphs in a manner efficient enough to be carried out in scenarios of practical size. Our experiments uncovered a number of systematic trends, including the tendency of a motion graph's capability to degrade rapidly with increasing complexity in either required tasks or target environment and the effect on a motion graph's capability of using some common motion graph improvement techniques.

Chapter 4

Perceptual Metrics

4.1 Introduction

This chapter presents the results of three studies of user sensitivity to errors in the ballistic phase of motion. Errors were added to motion captured jumps by manipulating the translational velocity of the center of mass in a systematic way, and user sensitivity to these errors was measured.

For the first two studies, we found that sensitivity varied with the level and variety of error added and that a significant level of error may be acceptable for many applications. Our specific results are subject to many factors, such as the complexity of the geometric model we used for testing [Hodgins et al., 1998]. However, for the specific circumstances

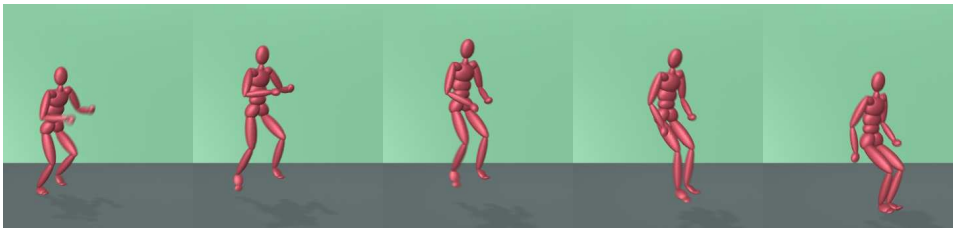


Figure 4.1: Example of a motion used in the first two user studies.

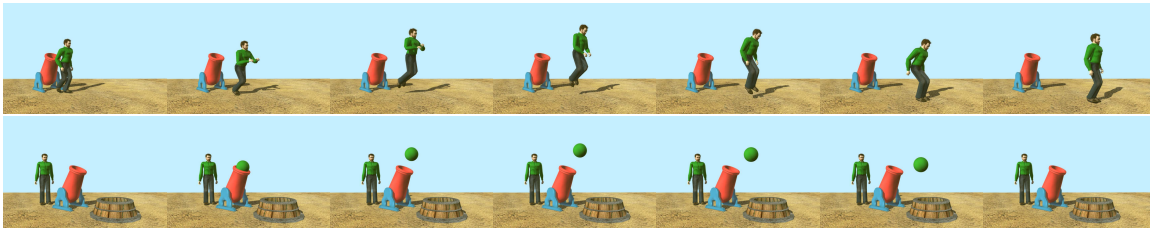


Figure 4.2: An example of one of the motions used in the third user study, depicted on each of the two characters used, a human figure (top) and a cannonball (bottom) which followed the same center of mass trajectory. The second frame is the takeoff point where the character transitions into ballistic motion.

of our test, we define a perceptual metric for evaluating translational errors in animated ballistic human motion.

In addition, this chapter also presents the results of a study comparing user sensitivity regarding errors in the ballistic phase of human jumping motion to errors in the ballistic trajectory of a ball with identical center-of-mass trajectories. Source motions and errors were the same as those used in the first two studies. We found that user sensitivity did not vary systematically between the two types of animations for localized errors (i.e., additional velocity added over a short time window), but varied significantly for altered levels of gravity. This result suggests it may be possible to leverage the work done on user perception of animations of simple objects to infer reasonable guidelines for animation of humanlike characters, but only for types of errors for which user sensitivity does not vary substantially between the two types of characters.

4.2 Experiment: Errors in Ballistic Motion

We chose to study motion containing a flight phase because errors in ballistic motion can be controlled easily and precisely, and because anomalies in ballistic motion are a common result of motion processing techniques. Once the character has left the ground, the trajectory of the center of mass is fully determined. Any changes to that trajectory vio-

late the laws of physics. Such changes can result, however, when motions with differing root velocities are spliced together, creating an anomalous acceleration, or when the effective gravitational constant changes as the result of an editing operation. An incorrect gravitational constant arises, for example, when the height of a jump is changed while timing remains the same. Scaling motion to characters of different sizes can also change the effective gravitational constant.

4.2.1 Method

Three studies were performed, the first to test perception of anomalous accelerations and decelerations, the second to test perception of errors in effective gravity, and the third to compare the sensitivity of users to errors in motions presented on two different characters (a textured human figure and a plain ball).

4.2.2 Study 1: Acceleration

Participants. Participants were obtained by university-wide advertising. Five women and seven men ranging in age from 18 to 42 participated in the study.[1]

Stimuli. Animations of human jumping motions were created as stimuli. All animations were shown in the same rendering style, with the same (fixed) camera configuration (Figure 4.1), with the character beginning the motion at the same position and jumping in the same direction. Shadows were rendered, and a small amount of motion blur was added.

Errors were created by modifying human jumping motion obtained from optical motion capture. Seven different source motions were used. These source motions were performed by the same actor and were similar in overall character, although they varied in distance and height. Error variables were as follows:

- *Error level.* Small, medium, or large.
- *Error variety.* Horizontal or vertical.

[1] One additional subject did not follow instructions and was excluded from the analysis.

- *Error direction.* Acceleration or deceleration.

To generate a horizontal acceleration error, for example, a fixed change in velocity was applied to the character root in the direction of horizontal motion. This change in velocity was added smoothly over a small window of time early in the flight phase of the motion. Details on how errors were created can be found in Section 4.3.

Procedure. Participants were told that they would see a sequence of animated human jumping motions created from motion capture data. They were given some background information on how motion capture data is created and told that some of the motions they would see would contain errors. They were also told that all motions are jumps, slightly less than half have no error, and all errors would appear during the flight phase of the motion. Participants were then shown 12 representative motions. They were told that half contained errors but were not told which specific motions in this training set contained errors.

Tests were prepared by placing motions on video tape in random order. Two tapes were used. Tape order differed for different subjects, and no order effects were observed. Motions included each combination of error variables presented 3 times; an equal number of original motions; and 12 motions with composite errors. Composite errors included both horizontal and vertical acceleration and were introduced to test our intuitions about these interactions. Stimuli were presented on a projection screen in a small conference room. Participants were instructed to categorize each motion as either “no error (unchanged)” or “error” and mark their level of confidence in their answer using a rating scale that ranged from 0 (most confident an error is present) through 9 (most confident an error is not present). Participants categorized training motions as well as test motions, but the data from the training motions was not used for any analyses.

At the end of the study, participants were asked to describe their experience in the study of motion, including involvement in sports, dance, video games, etc. No significant effect of level of experience or gender was noted for this study.

4.2.3 Study 2: Gravity

Participants. Participants were obtained by university-wide advertising. Nine women and two men ranging in age from 19 to 29 participated in the study.[2]

Stimuli. Stimuli were prepared in the same manner as in the first study, with error varieties including increased and decreased gravity.

Procedure. The procedure was identical to that of the first study, except that all participants were shown one tape consisting of 60 motions, including 18 motions with gravity errors, 6 with vertical errors, 6 with horizontal errors, 4 with composite errors, and 26 with no errors. Motions with vertical, horizontal, and composite errors were included to test the validity of comparing data across studies, and performance was consistent with that observed in the first study.

No significant effect of level of experience or gender was noted for this study.

4.2.4 Study 3: Effect of Character Animacy on Error Sensitivity

We extended the method used in the previous studies to examine the effect on user sensitivity to errors of two different character types (a simple ball and a human model; see Figure 4.2), as opposed to the model used previously (Figure 4.1), which was a rough human figure constructed from ellipsoids.

Participants. Participants were obtained by university-wide advertising. 9 women and 13 men ranging in age from 18 to 33 participated in the study.[3]

Stimuli. Animations of human jumping motions were created as stimuli. All animations were shown in the same rendering style, with the same (fixed) camera configuration (Figure 4.2), with the character beginning the motion at the same position and jumping in the same direction. Shadows were rendered, and a small amount of motion blur was added.

[2]Three additional subjects did not follow instructions and were excluded from the analysis.
[3]4 additional subjects did not follow instructions and were excluded from the analysis.

These parameters were chosen to be as close as possible to those used in the first two studies (Section 4.2). The motions for the human character were created in an identical manner to those created for the first two studies, using the same source motions.

Motions for the ball consisted of three parts linked together in a C1-continuous manner, corresponding to the takeoff, ballistic, and landing phases of the jumps performed by the human character. The takeoff phase for the ball consisted of smooth linear acceleration from a fully-hidden position inside the cannon's barrel to the point where the ball was half-emerged from the mouth of the cannon (first and second frame respectively of Figure 4.2). The ball then entered its ballistic phase, and followed the center of mass trajectory of the human character performing that motion (frames three through five of Figure 4.2). Finally, the frame where the human character's feet touch the ground marks the beginning of the landing phase; during this phase, the ballistic motion of the ball was extended by continuing its constant acceleration until it entered a basket placed on the ground plane and disappeared from view (last two frames of Figure 4.2).

Procedure. Participants were given largely the same instructions as for the studies in Chapter 4, with minor differences. Participants were told they would see two types of motions: a human jumping, or a ball traveling through similar arcs; that half of the motions had errors; and that errors were similar in each of the two types of animations. Participants were then shown a training set of 24 motions, consisting of the each of the two characters (human and ball) being used to display the same 12 representative error treatments as used for the first two studies. Source motions for each error treatment were chosen at random, and the presentation order of the full set of 24 animations was randomly permuted.

Each test consisted of 72 unique motions – 2 examples of each error treatment shown on two different source motions, plus the corresponding error-free motions – displayed on each of the two characters, for a total of 144 test motions. The order of presentation of the 144 animations was randomly permuted, and then split into four blocks of 36 trials for display purposes.

All motions were placed on DVD in movie format and played on a commercial DVD player, as that appeared to minimize playback hitches compared to playback from either a

video file on computer or from a DVD in a computer DVD drive. Six DVDs were used, each with a different random assignment of source motions to error treatments, and each with a unique permutation of presentation order for the resulting animations. Each DVD was seen by between 3 and 5 subjects. Stimuli were presented on the same projection screen as for the prior studies, and participants were given the same style of response sheets.

No significant effect of level of experience was noted for this study, but female subjects gave the animations higher ratings than male subjects ($F(1, 3153) = 8.39, P = 0.0038$). The difference came from their ratings of motions containing errors ($F(1, 1574) = 27.81, P < 0.0001$), as there was no difference in their ratings of error-free motions ($F(1, 1577) = 1.81, P = 0.179$). However, this difference was not reflected in a difference in ability to correctly discriminate between errors ($F(1, 20) = 5.19, P = 0.034$, which is not significant for a two-tailed distribution).

4.3 Error Generation

To create a motion with error, we must add the required error velocity or acceleration to the flight phase while minimizing undesirable artifacts. Our primary goals were to ensure that character motion before and after the flight phase was unchanged (except for horizontal landing position) and that the resulting motion had continuous second derivatives (smooth velocities). Achieving these goals required different processing for each error type. Assume a motion M whose flight phase starts at frame T and ends at frame L, sampled at FPS frames per second. Procedures with names in italics are described at the end of the section. All processing is performed on the translational parameters of the character root. Examples of velocity curves with and without errors are shown in Figure 4.3.

Horizontal Errors. Assume motion M sampled at 120Hz, with horizontal direction H. Use routine *AddError(E,H,M, T+12,T+24)* to add a horizontal velocity error with magnitude E to the motion from 0.1s to 0.2s after the start of the flight phase. To prevent a velocity discontinuity at landing, remove this error velocity over the remainder of the

motion with *AddError(-E,H,M,T+25,L)*.

Gravity Errors. Compute new duration t_n = *Duration(M,*G'*,0)* for the new gravity level, and *Timewarp(M,N,*t_m*,*t_n*)* M to create a new motion N. Finally, *Deseam(N,dY,dV)* to remove position discontinuity dY and velocity discontinuity dV at landing. These discontinuities are relatively small and result from discretization errors, motion capture data errors, and differences between takeoff and landing height of the center of mass. Note that preserving average horizontal velocity and initial vertical velocity leads to jumps that cover a shorter horizontal distance in higher gravity cases and a longer distance in lower gravity cases.

Vertical Errors. Compute new duration t_n = *Duration(M,G,*E*)* for the vertical velocity error E, and *Timewarp(M,N,*t_m*,*t_n*)* M to create a new motion N. *AddError(E,V,N,T+12,T+24)* to add a vertical velocity error of magnitude E to the motion over the period from 0.1s to 0.2s after the jump starts. Finally, *Deseam(N,dY,dV)* according to the position discontinuity dY and velocity discontinuity dV at landing. As with gravity errors, preserving the horizontal velocity and gravity and applying these effects over the new duration of the flight phase leads to jumps that travel greater or lesser horizontal distances.

General Processing and Procedure Definitions. Errors were added to the motions at 120Hz, and the motions were downsampled to 30Hz for display. All motions were translated to align the root positions of their first frames and rotated to align their jump direction. The flight phase of a jump was defined as the first frame where the lowest joint was above 7cm from the ground plane through the first frame where the lowest joint was below 7cm. For all motions, the position change of the last frame of the flight phase was added to all subsequent frames to align the landing with the end of the jump. Procedure definitions follow.

AddError(E,D,M,a,b): Add an error velocity of magnitude E along direction D to the root translation of motion M. *Rampin(E,U,a,b)*, then *GetPosition(U,V)*, then simply add the

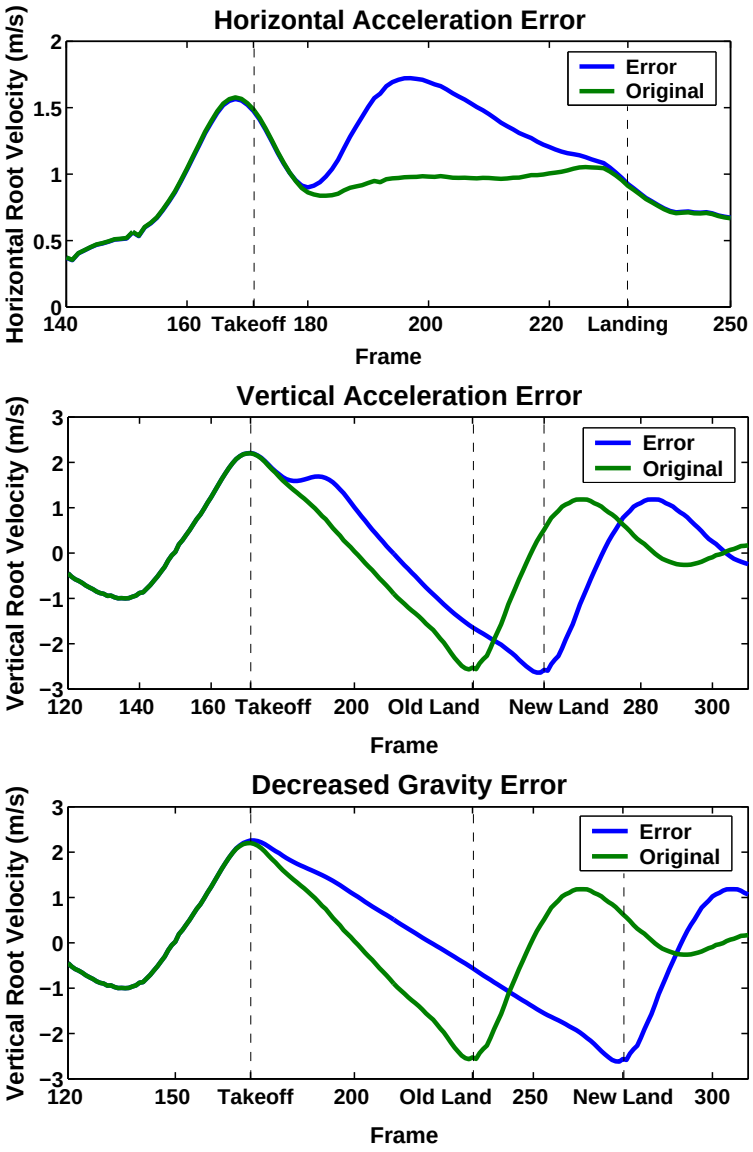


Figure 4.3: Examples of velocities with and without errors. (Top) Horizontal velocities with and without added acceleration in the horizontal direction. (Middle) Vertical velocities with and without added acceleration in the vertical direction. (Bottom) Vertical velocities with and without decreased gravity.

motions: $M[i] = M[i] + V[i] * D$.

Duration(M,G,E): Compute the new flight time for root motion M given gravity G, vertical velocity error E, and initial vertical velocity V_i using basic kinematics:

$$t' = \lfloor (\frac{1}{G}(-(V_i + E) - \sqrt{(V_i + E)^2 + 2G(\frac{E}{20} + \frac{E * FPS}{L - T - 0.2 * FPS}))} \rfloor \quad (4.1)$$

Timewarp(M,N,t_m,t_n): Timewarp the flight phase of M from t_m to t_n seconds in an ease-in/ease-out manner so as not to cause discontinuities in joint velocities, and place the result in N. First, remove gravity and average horizontal velocity from root translation of M, resulting in M'. Perform the timewarp. Add back in the average horizontal velocity and gravity removed earlier but applied over the whole of the new motion, resulting in N.

Deseam(M,dY,dV): Some error types lead to a vertical velocity discontinuity at landing, which is fixed in this postprocessing step. For a given required position correction dY and a given required velocity correction dV, let

$$D[T + 0.2 * FPS + i] = (-2dY + 0.1 * dV)t^3 + (3dY - 0.1 * dV)t^2 \quad (4.2)$$

where $t = \frac{i+1}{L-T-0.2*FPS}$ for $0 \leq i < L-T-0.2*FPS$. D is then added to root translation M to make the corrections specified by dY and dV: $M[i] = M[i] + D[i]$.

Rampin(E,U,a,b): Using the spline $-2x^3 + 3x^2$, smoothly transition from 0 at frame a to E at frame b, and place the result in array U.

GetPosition(U,V): Integrate a velocity error in array U up into a change in position in array V.

4.4 Results

Figure 4.4 shows mean ratings for horizontal errors (top), vertical errors (middle), and gravity errors (bottom) for studies 1 and 2. Blue lines are acceleration (or decreased gravity) and green lines are deceleration (or increased gravity). Results are broken out by

small, medium, and large error levels. The plots show the mean ratings of motions at each magnitude of added error, including unchanged motions. Figure 4.5 shows mean ratings for all error varieties and directions from all studies.

A repeated measures analysis of variance (ANOVA) was run for study 1 with 3 error levels x 2 error varieties (horizontal and vertical) x 2 error directions. All error treatments could be detected at $p < 0.01$ except for small vertical decelerations. A second ANOVA was run for study 2 with 3 error levels x 2 error directions. All error treatments could be detected at $p < 0.01$ except for small and medium levels of increased gravity. A third ANOVA was run for study 3 with 3 error levels x 2 character types (human and ball) x 3 error varieties x 2 error directions. All error treatments of large magnitude could be detected at $p < 0.01$; 9 of 12 error treatments of medium magnitude could be detected at $p < 0.01$ (exceptions: vertical deceleration for both characters, and decreased gravity for the human character); 5 of 12 error treatments of small magnitude could be detected at $p < 0.01$.

No significant interactions were observed between error variety and error direction in study 1 or error variety and error magnitude in studies 1 or 2. There was an interaction between error direction and error level (study 1: $F(2, 426) = 5.7, p = 0.003$; study 2: $F(2, 192) = 5.5, p = 0.005$). This interaction can be seen in Figure 4.4: added acceleration or decreased gravity is proportionately easier to detect for large errors. There was an interaction between error variety and error magnitude for study 3, which was only significant for horizontal errors with small vs. medium error magnitudes ($t = 4.20, P < 0.0001$); this can be seen in the comparatively low level of sensitivity for medium horizontal errors (Figure 4.9). There was an interaction between error variety and error direction in study 3, but it was only significant for gravity. There was a weak interaction between error variety x error direction x character type in study 3 which was only close to significant for horizontal errors ($t = 1.98, P = 0.048$). This can be seen in the higher sensitivity to the human character for horizontal accelerations, but the higher sensitivity to the ball character for horizontal decelerations, which is explored in Section 4.7.

In addition, we found four main effects, reported as F-value differences in the ratings,

and also as paired-t-values on the per-subject sensitivities for study 3:

(1) Subjects found added acceleration easier to detect than added deceleration
Study 1: $F(1, 430) = 38.2, P < 0.001$, mean rating 2.6(0.2) vs. 4.1(0.2) respectively
Study 3 human: $F(1, 524) = 33.8, P < 0.0001$, paired-$t = 3.71, P = 0.0013$, mean rating 2.71(0.16) vs. 4.00(0.16)
Study 3 ball: $F(1, 522) = 18.3, P < 0.0001$, paired-$t = 4.24, P = 0.0004$, mean rating 2.80(0.15) vs. 3.78(0.17)

(2) Subjects found low gravity easier to detect than high gravity
Study 2: $F(1, 196) = 41.9, P < 0.001$, mean rating 2.7(0.2) vs. 4.7(0.2) respectively
Study 3 human: $F(1, 260) = 24.86, P < 0.0001$, paired-$t = 3.84, P = 0.0009$, mean rating 2.50(0.22) vs. 4.02(0.21)
Study 3 ball: $F(1, 262) = 9.99, P = 0.0018$, paired-$t = 2.15, P = 0.043$, mean rating 3.61(0.23) vs. 4.57(0.20)

(3) Subjects found errors in horizontal velocities easier to detect than errors in vertical velocities
Study 1: $F(1, 430) = 18.1, P < 0.001$, mean rating 2.8(0.2) vs. 3.9(0.2) respectively.
Study 3 human: $F(1, 524) = 14.40, P = 0.0002$, paired-$t = 4.90, P < 0.0001$, mean rating 2.93(0.16) vs. 3.79(0.16)
Study 3 ball: $F(1, 522) = 15.71, P < 0.0001$, paired-$t = 3.56, P = 0.0018$, mean rating 2.83(0.16) vs. 3.74(0.16)

(4) Subjects found gravity errors easier to detect with human characters than with ball characters
$F(1, 524) = 13.97, P = 0.0002$, paired-$t(1, 21) = 7.3, P < 0.001$, mean rating 3.26(0.16) vs. 4.09(0.16) respectively.

The studies gave very comparable overall results. Sensitivity to horizontal and vertical errors did not differ significantly between the studies ($F(1, 53) = 1.64, P = 0.20$ for ANOVA on subject sensitivity vs. study 1, study 3 human, or study 3 ball). Sensi-

tivity to gravity errors did not differ significantly between study 2 and the study 3 human ($F(1, 31) = 3.58, P = 0.068$) or between study 2 and the study 3 ball ($F(1, 31) = 3.82, P = 0.060$). This result is especially interesting in light of the difference between the study 3 human and the study 3 ball (main effect 4), suggesting that perhaps the rough human figure used in study 2 was perhaps "perceptually in between" the realistic human figure and the simple ball used in study 3. Since the studies with the ellipsoidal human figure and the studies with the textured human figure and the ball were run separately, however, such a relationship is at this point just conjecture.

4.5 Estimating Acceptable Error

From our results, we can form an estimate of a level of error that should be acceptable. We propose a method for calculating error thresholds based on detection theory [Macmillan and Creelman, 1991]. The ratings given by a subject to unchanged motions form an approximately normal distribution, and the ratings given to motions containing errors form a second, also approximately normal distribution. The subject's sensitivity is the distance between the means of these two distributions, in units of standard deviation. For a simple yes/no classification, sensitivity d is easily calculated from the hit rate (H), the fraction of motions containing errors that are correctly judged to contain errors, and false alarm rate (F), the fraction of original motions that are incorrectly judged to contain errors:

$$d = z(H) - z(F) \tag{4.3}$$

where z is the inverse of the normal distribution function. (See Macmillan and Creelman [1991] for details.) For example, a hit rate of 69% with a false alarm rate of 31% gives a sensitivity of 1.0. Sensitivity does not reflect the biases of the subject, so a hit rate of 26% and a false alarm rate of 5% also represents a sensitivity of 1.0. As a result, sensitivities determined with one set of biases (such as a test setting) may be applied to a situation with different biases (such as playing a game). We computed sensitivities from subjects' ratings using the method in Macmillan and Creelman [1991]. In cases where the sparse data led to a degenerate distribution (e.g., if a participant marked only zeroes and nines),

			DoF	F-value	Probability
Horizontal	**All**		862	310.9	0.0001
	Accel	Small	682	28.8	0.0001
	Accel	Medium		116.1	0.0001
	Accel	Large		196.2	0.0001
	Decel	Small		14.3	0.0002
	Decel	Medium		64.7	0.0001
	Decel	Large		77.8	0.0001
Vertical	**All**		862	128.7	0.0001
	Accel	Small	682	8.1	0.0045
	Accel	Medium		65.5	0.0001
	Accel	Large		186.4	0.0001
	Decel	Small		0.1	0.7125
	Decel	Medium		14.8	0.0001
	Decel	Large		21.5	0.0001
Gravity	**All**		493	63.7	0.0001
	Incr	Small	328	9.6	0.2077
	Incr	Medium		50.3	0.9321
	Incr	Large		128.6	0.0009
	Decr	Small		1.6	0.0021
	Decr	Medium		0.0	0.0001
	Decr	Large		11.2	0.0001

Table 4.1: F-values and probabilities for horizontal, vertical, and gravity errors from studies 1 and 2. All degrees of freedom are of the form $F(1, N)$; for example, $F(1, 862) = 128.7$ for all horizontal errors.

	Small	Medium	Large	Regression Line
H Accel	1.02 (0.27)	1.89 (0.29)	2.53 (0.19)	$0.00 + 2.44E$
H Decel	0.69 (0.24)	1.36 (0.26)	1.53 (0.29)	$0.05 + 1.52E$
V Accel	0.45 (0.07)	1.25 (0.17)	2.48 (0.19)	$-0.25 + 2.32E$
V Decel	0.34 (0.21)	0.60 (0.12)	0.85 (0.29)	$0.00 + 0.81E$
G Decr	0.47 (0.20)	1.20 (0.27)	1.87 (0.22)	$-0.12 + 0.48E$
G Incr	0.16 (0.14)	0.02 (0.14)	0.50 (0.28)	$-0.05 + 0.10E$

Table 4.2: Mean sensitivity levels (and standard error of the mean) for horizontal (H), vertical (V), and gravity (G) errors. A sensitivity of zero means that participants cannot detect errors. The last column contains lines fit to the sensitivity data, also including the point $(0, 0)$. **E** is the magnitude of the error, in m/s for horizontal or vertical errors and in m/s^2 for gravity errors.

we calculated sensitivity using the participant's classification of the motion as "no error (unchanged)" vs. "error" rather than using the numerical ratings.

We apply this technique to the data from studies 1 and 2. Sensitivity levels from these studies are plotted in Figure 4.6 and listed for all error treatments in Table 4.2, along with regression lines fit to these values plus the origin (zero sensitivity at zero error). The F-values and probabilities of these errors are given in Table 4.1. As an example of how error thresholds can be set from these values, consider a hypothetical application. Suppose that for unmodified motion capture data we expect users to think that the motion looks incorrect 10% of the time (a false alarm rate of 10%). Then suppose that for motions containing error, we are willing to tolerate users thinking that the motion looks incorrect half again as often, or 15% of the time (a hit rate of 15%). The resulting sensitivity would be 0.25. Estimating acceptable errors at this sensitivity level results in the threshold values shown in Table 4.3. We emphasize that this is just an example. The actual desired sensitivity (and resulting threshold values estimated from our data) would depend on the application.

Variety	Threshold
Horizontal error over $0.1s$ interval	$[-0.13m/s,\ 0.10m/s]$
Vertical error over $0.1s$ interval	$[-0.32m/s,\ 0.22m/s]$
Gravitational constant	$[-12.7m/s^2,\ -9.0m/s^2]$

Table 4.3: Error thresholds resulting from a desired sensitivity level of 0.25 or less. For reference, average initial velocities in the original jumps were approximately $2m/s$ in the vertical direction and $1.5m/s$ in the horizontal direction.

Composite Errors. Composite errors—those with both horizontal and vertical components—did not produce any surprises. Figure 4.7 plots mean ratings for vertical, horizontal, and composite errors. Results for composite errors fall approximately within the bounds of those for the two types of errors from which they are derived.

4.6 A Ballistic Error Metric

We briefly describe how an error metric for ballistic motion might be designed based on our results, and based on the example sensitivity requirement of 0.25.

First, consider errors in gravity. Gravity is an average effect, and a metric designed to detect incorrect gravity captures errors where the motion is well behaved throughout the flight phase, but the timing of the motion is wrong. From the vertical takeoff velocity of the center of mass $v_v(t_i)$, the vertical landing velocity of the center of mass $v_v(t_f)$, and elapsed time $(t_f - t_i)$, we can compute the effective gravity represented by a motion:

$$g_{eff} = \frac{v_v(t_f) - v_v(t_i)}{(t_f - t_i)} \tag{4.4}$$

Results from our second study suggest that under circumstances similar to this study, values for g_{eff} between $-12.7m/s^2$ and $-9.0m/s^2$ should lead to sensitivity levels below 0.25, resulting in the following constraint:

$$-12.7m/s^2 < g_{eff} < -9.0m/s^2 \tag{4.5}$$

Anomalous accelerations and decelerations are shorter term phenomena, where the motion during the flight phase is not well-behaved over some window in time. One strategy for measuring errors of this type would be to compute error in horizontal or vertical velocity over a sliding time window, checking this error against the given thresholds. For example, horizontal velocity should be constant during flight, and so any change in horizontal velocity during flight is an error.

$$v_{h,err}(t) = v_h(t + 0.1s) - v_h(t) \tag{4.6}$$

$$-0.13m/s < v_{h,err}(t) < 0.10m/s \tag{4.7}$$

where $v_h(t)$ is the horizontal velocity measured at time t, and $v_{h,err}(t)$ is the horizontal velocity error for the time window of $0.1s$ starting at t. Changes in velocity outside this range would flag potentially anomalous motion. The time window of $0.1s$ is chosen because our study results are based on this value.

The metric for vertical velocity is similar, but must accommodate expected change in velocity due to gravity. For a time window of $0.1s$:

$$v_{v,err}(t) = v_v(t + 0.1s) - v_v(t) + 0.98m/s \tag{4.8}$$

$$-0.32m/s < v_{v,err}(t) < 0.22m/s \tag{4.9}$$

where $v_v(t)$ is the vertical velocity measured at time t and $v_{v,err}(t)$ is the vertical velocity error for the time window of $0.1s$ starting at t. When measuring errors against a gravitational constant different from $-9.8m/s^2$, the equation for $v_{v,err}(t)$ should be adjusted for the new value.

Our composite results suggest that vertical and horizontal velocity errors may combine in a straightforward way. The following metric would place limits on combinations of horizontal and vertical errors, which should be an improvement over treating them separately. One possibility is to work with the sum of squares of normalized error values. For example, if the error at time t is a horizontal acceleration (with threshold $0.10m/s$) and a

vertical deceleration (with threshold $-0.32m/s$), the appropriate constraint would be:

$$\left[\left(\frac{v_{h,err}(t)}{0.10m/s}\right)^2 + \left(\frac{v_{v,err}(t)}{0.32m/s}\right)^2\right] < 1 \tag{4.10}$$

The expression on the left hand side of this equation is a squared distance in velocity space, with horizontal and vertical velocities weighted differently. Values in the denominators would change when the direction of the corresponding error changed.

4.7 Discussion

In these studies, we measured sensitivity of human subjects to errors in animated ballistic human and ball motion. We found that sensitivity was correlated with the level of added error, errors in the horizontal component of the motion were easier to detect than errors in the vertical component, added accelerations were easier to detect than added decelerations, decreased gravity was easier to detect than increased gravity, and changes in the level of gravity were easier to detect with a human character than with a ball character. By contrast, there was no significant difference in sensitivity between human characters and ball characters for horizontal or vertical motions.

Why might it be easier to hide errors in the vertical component of the motion than in the horizontal component? One possible explanation is that horizontal velocity during flight, which should be constant, behaves in a simpler manner than vertical velocity, which should have a constant derivative (gravitational acceleration). This difference may make anomalies in horizontal velocity more visually salient.

The acceleration / deceleration discrepancy seemed to us to be less intuitive. One possible explanation is that it may be easier to detect errors when distance or total time of the jump is increased and more difficult when one or both of these parameters are decreased.

It is important to point out that jump heights and distances alone could not have accounted for the effects we observed. For the first two studies, 69% of the motions containing errors were within 10% of the range of heights and distances spanned by unchanged

motions. In addition, although 4 subjects in study 1 mentioned making use of jump distance, timing, or similar indirect observations of error, 11 of the 12 subjects mentioned direct observation of errors, such as changes in jump trajectory. However, overall jump distance does appear to have been a contributing factor in participant ratings. Our study was not designed to test the effect of source motion—source motion was not fully crossed with other error variables. However, we did check for interactions between source motion and other error variables considered individually. The only significant interaction found was between source motion and error direction in study 1 ($F(6, 418) = 4.5, p < 0.001$). Examination of the data (shown in Figure 4.8) suggests that added accelerations are proportionately easier to detect for jumps that cover a longer distance. However, it is interesting to note that sensitivity for accelerations is higher than for decelerations even for the shortest jumps, which would not be the case if jump distance were the primary factor used to detect errors. Because we did not fully cross source motions with all other variables in the study, however, further investigation is required to verify this trend and to understand its implication for designing error metrics.

We note several systematic differences in user sensitivity between motions using the human character and motions using the ball character (Figure 4.9): sensitivity to errors was in general slightly higher when the human character was used (paired-$t(1, 21) = 3.47, P = 0.0023$); however, all of this difference is attributable to the difference for gravity errors (paired-$t(1, 21) = 7.306, P < 0.0001$), as the difference for horizontal and vertical errors was negligible (paired-$t(1, 21) = 0.694, P = 0.495$).

Several possible explanations could account for the systematic difference between sensitivity to gravity errors between motions using the human and the ball characters. One possibility is that because gravity errors correspond to incorrect behavior over the entire jump, rather than a localized disruption, subjects' greater familiarity with human jumping motion than ball trajectories would account for the difference, especially because it is known that humans have a poor intuitive sense for the physics of the ballistic trajectories of simple objects such as balls [Hecht and Bertamini, 2000]. Another possibility is that because a change in the force of gravity corresponds to timescaling of the jump, the animated motions of the limbs and head of the human character offered additional information that

was not present in the ball character. Stappers and Waller [1993] note that richer stimuli improved accuracy and reliability in using the free fall of objects under gravity to estimate visual depth, suggesting that the additional information offered by the limbs of the human character may have improved user results in our experiment. They also note how observed gravity can vary with perceived scale and distance, suggesting the possibility that the human jumping innately embodies a sense of scale, whereas subjects may have some freedom to interpret the size and distance of the cannonball to suit the observed motion, notwithstanding the human figure placed beside the cannon.

The interaction noted in Section 4.4 between character type and error direction for horizontal errors may have its underpinnings deep in the human visual system. Michotte [1963] and Runeson [1974] note that a simple object appears to slow down when receding from a stationary object in its path and speed up when approaching it, and Cohen [1964] reports that this occurs only when the subject fixates on the stationary object. As the test scene we used opened with the ball hidden inside the barrel of the cannon (Figure 4.2), it is likely that our subjects fixated on the cannon to begin with, and hence they may have observed this spurious perceived deceleration. If so, the deceleration caused by this effect could be expected to enhance the perceptual salience of the deceleration actually undergone in the horizontal deceleration error condition, raising the sensitivity to that error treatment. No similar increase in sensitivity would have occurred for the human character, due to the lack of a fixed object in its path, potentially explaining the inversion of their usual sensitivity relationship.

Another, related, difference between the human character and the ball character is that the landing point of the ball is signalled by the basket (Figure 4.2), and hence is known before the motion takes place, but the precise landing point of the human is not displayed beforehand. The slight difference observed between sensitivity to horizontal errors for the human and ball characters, however, suggests that subjects were likely fixating on the cannon at the start of the motion, potentially limiting the extent to which the basket may have affected their judgements. In additional, errors in the motions occurred early in the trajectory of the character (as well as in the rest of the trajectory, for gravity errors), and hence one might suspect that errors would tend to be observed early in the flight phase,

before the ball neared the basket and perhaps before the subject's attention shifted away from the region near the cannon to the region near the basket. Moreover, subject responses in the questionnaire suggested direct observation of errors due to particular characteristics of the motion, rather than inference of error due to indirect factors. For these reasons, we consider it unlikely that signalling the landing point of the ball made a significant difference to measured sensitivity; however, we are considering the design of potential followup experiments to test this hypothesis.

Many parameters can be expected to affect perception of anomalies in animated human motion. There is some evidence that improved graphical quality of animations, for example, may increase the ability of users to detect anomalous motions [Stappers and Waller, 1993, Hodgins et al., 1998, Oesker et al., 2000, Hecht and Bertamini, 2000] We focused on errors added to the character's root motion in hopes that the results would be robust with respect to detail in the rest of the animation, but this assumption remains to be tested.

We note also that camera angle [O'Sullivan and Dingliana, 2001] and camera motion [Strawn et al., 2006] can have a substantial effect on the perceptual salience of errors. To allow our experiments to focus on other aspects of perceptual sensitivity, we used only a single stationary camera angle, selected so as to offer a balanced and representative viewpoint of all error varieties with no apparent bias for or against any one variety.

Perception of anomalies also varies with task [Oesker et al., 2000, Watson et al., 2001]. The task presented to subjects in our study was simply to observe the motion and indicate whether it appeared to be incorrect. We would expect that sensitivity to errors might be higher if the character was a target in a game environment, for example, and lower if the character was not the focus of attention (e.g. a background character in a virtual environment). More research is required to understand how sensitivity to error in the physics of human motion varies with task.

4.8 Conclusions

This work offered a statement of the problem of evaluating the perceptual magnitude of errors in animated human motion, and demonstrated how to measure that perceptual magnitude and to use it to derive practical animation guidelines.

Our experiments demonstrated several interesting asymmetries in the perceptual magnitude of errors, suggesting different error types may become perceptually salient at different rates, allowing a more efficient tradeoff when creating animations. Our experiments also demonstrated a surprising similarity between user sensitivity to some types of errors – such as added horizontal or vertical velocity over a short window – regardless of the character type on which those errors were shown. We are interested in the notion that results from perceptual studies on one character type, such as a rigid body, may be able to inform animations of other character types, such as human figures.

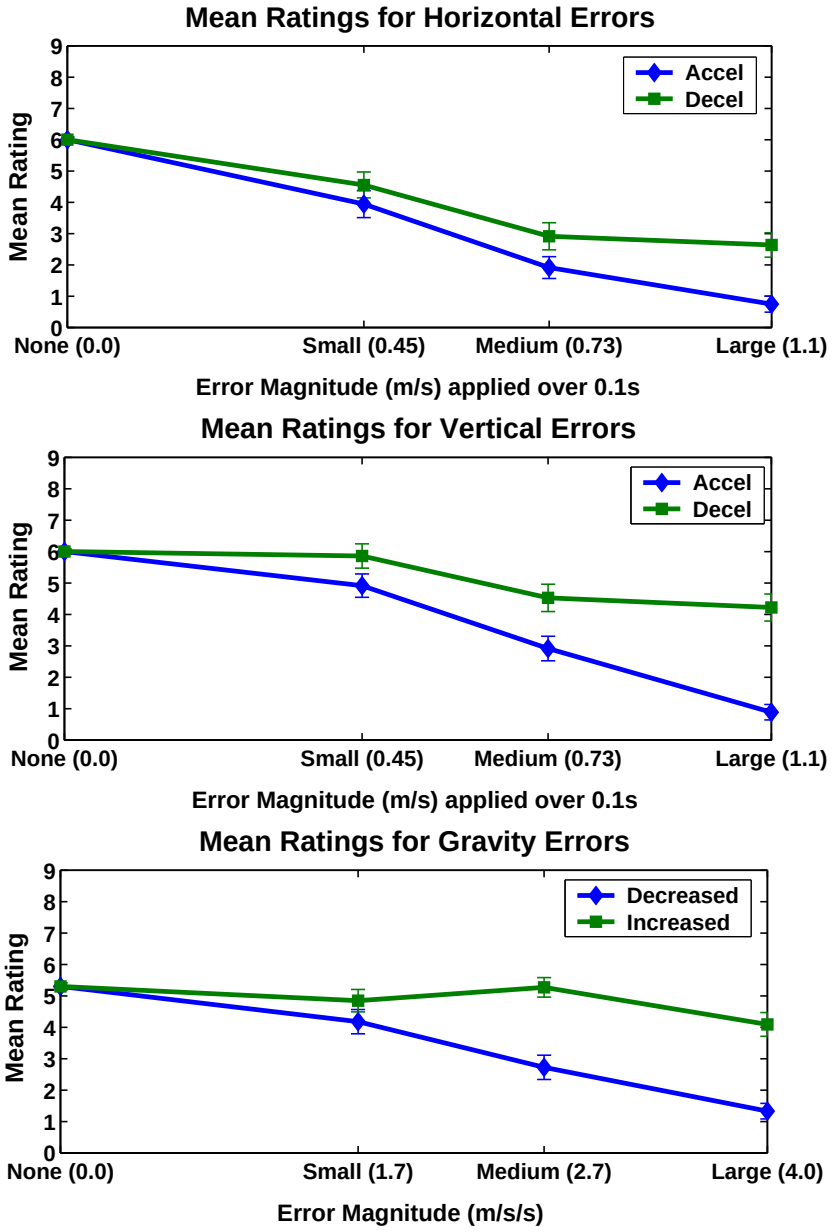


Figure 4.4: Mean ratings of motions with horizontal, vertical, and gravity errors from studies 1 and 2. The mean rating for unchanged motions is plotted for reference. Each plot is broken out by error direction and error magnitude. Error bars show standard error of the mean.

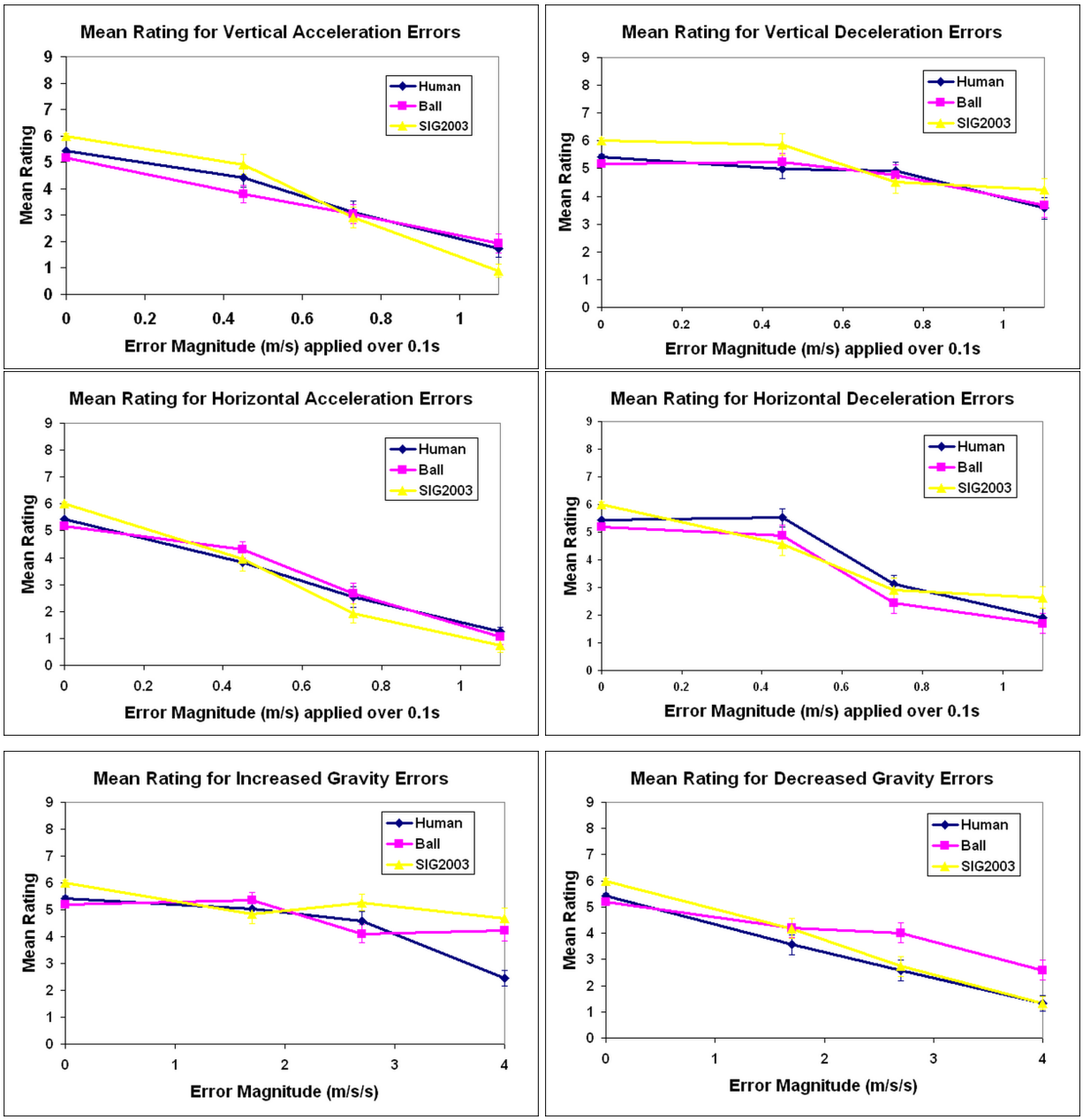


Figure 4.5: Mean ratings for all errors from all studies. Error bars show standard error of the mean.

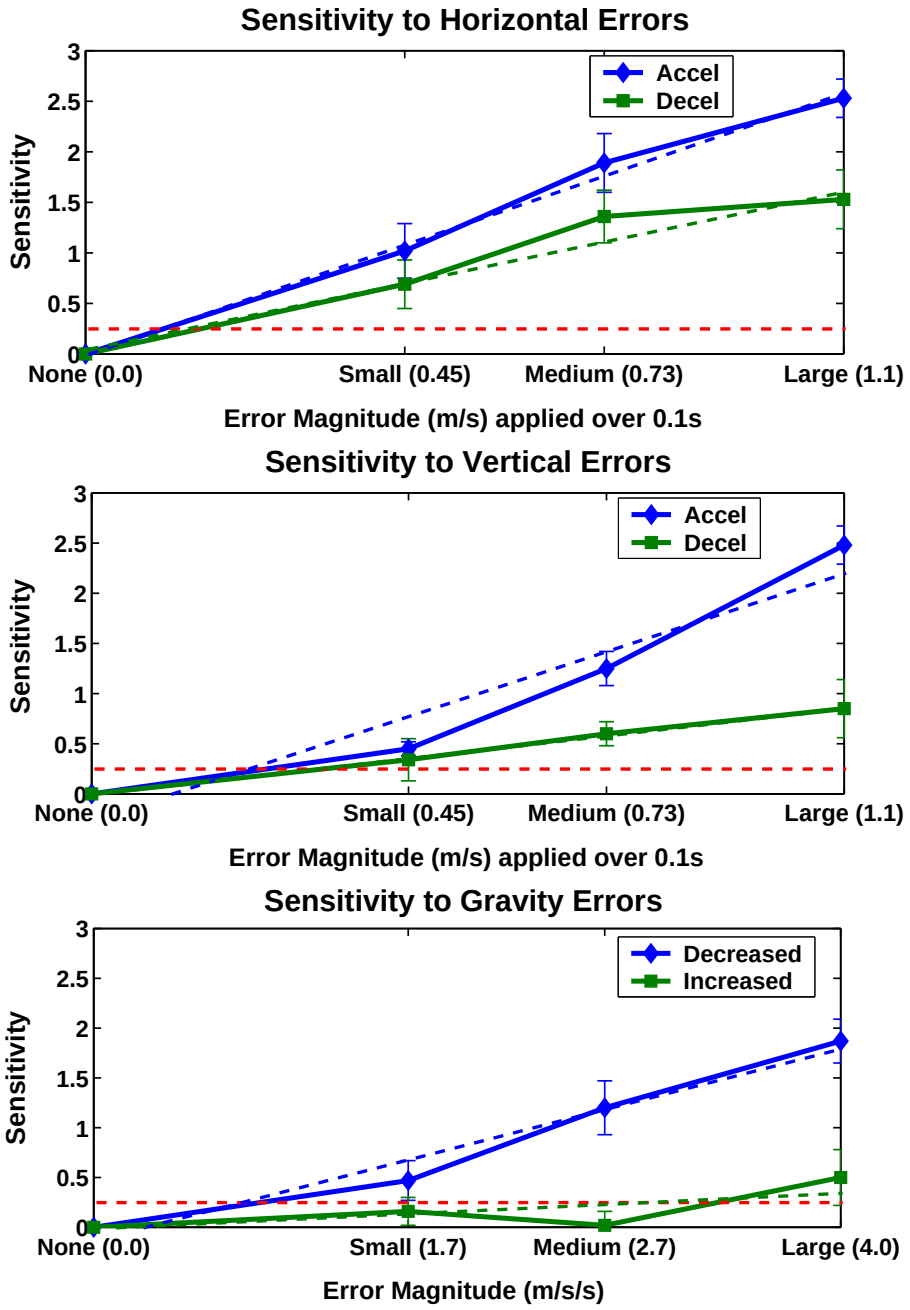


Figure 4.6: Mean sensitivities for all errors from studies 1 and 2, with best-fit linear approximation. Error bars show standard error of the mean.

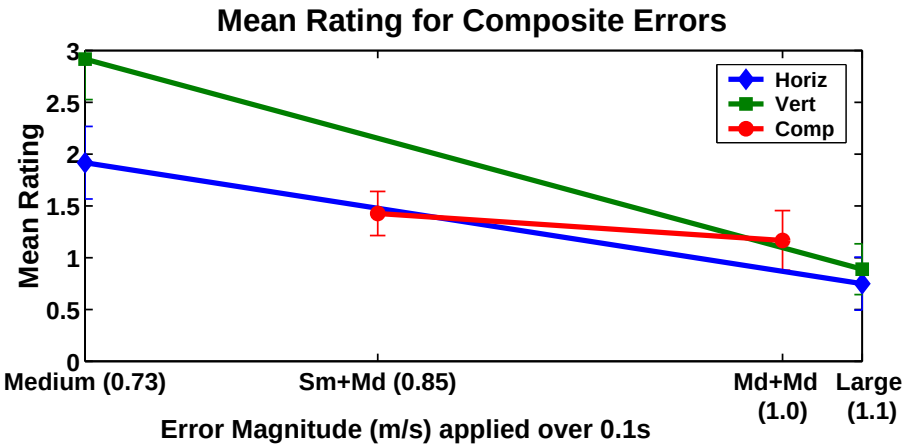


Figure 4.7: Results for composite errors are approximately bounded by results for the types of errors from which they are derived.

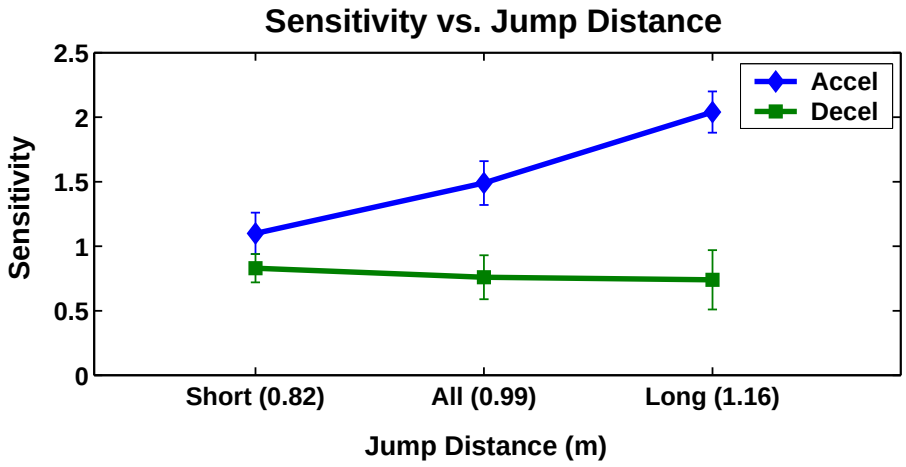


Figure 4.8: Mean sensitivity for study 1 for all seven source motions (middle), the three shortest jumps (left), and the three longest jumps (right). Accelerations are comparatively easier to detect for longer jumps.

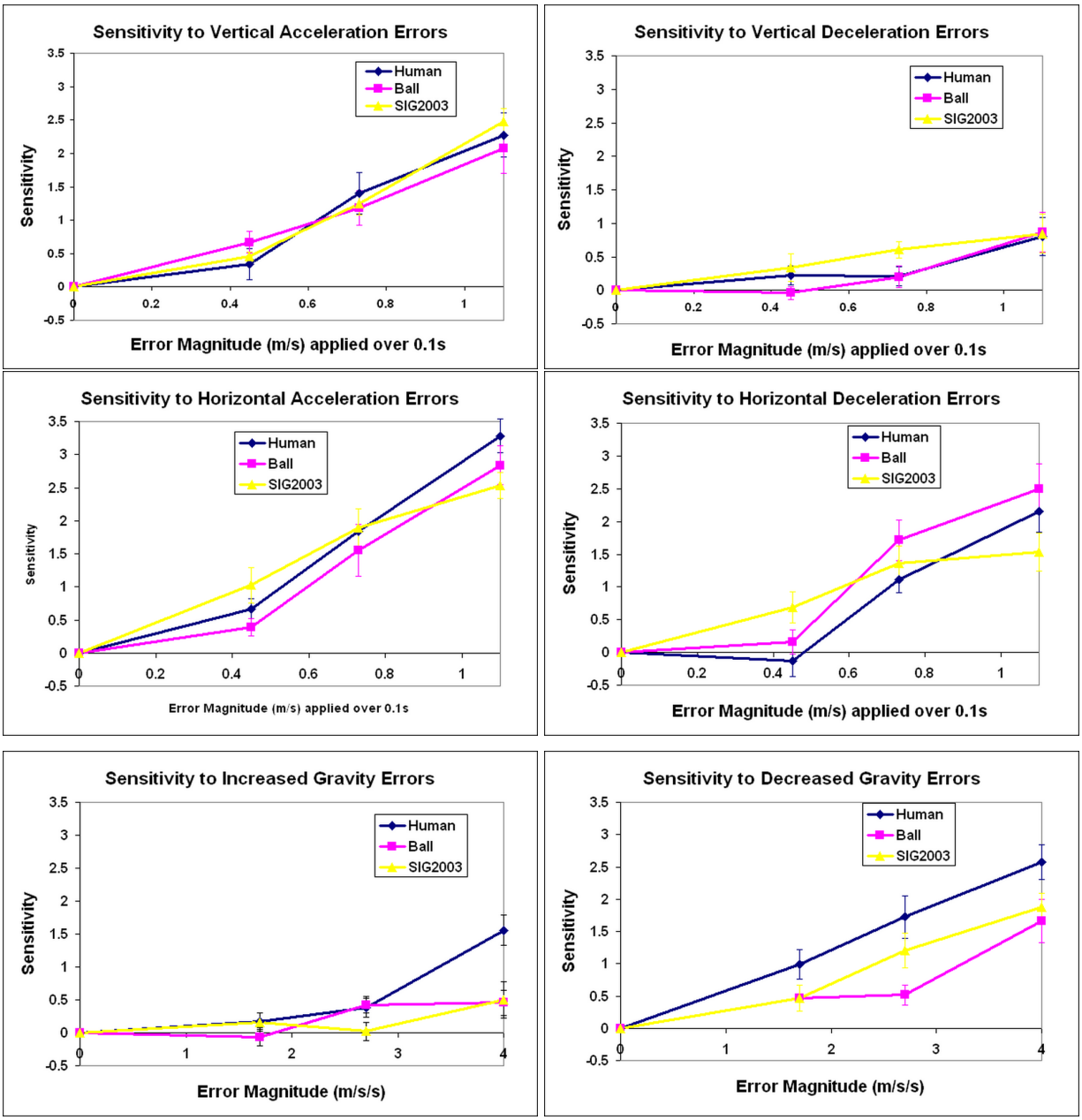


Figure 4.9: Mean sensitivities for all errors from all studies. Error bars show standard error of the mean.

Chapter 5

Evaluating Motion Graphs

This chapter proposes a definition for the notion of a motion graph's capability to create required animations, and describes a method to quantitatively evaluate that capability. The method identifies motion graphs with unacceptable capability and also identifies the nature of their deficiencies, shedding light on possible remedies. One important finding is that the capability of a motion graph depends heavily on the environment in which it is to be used. Accordingly, it is necessary to embed a motion graph into its target environment, and we introduce an efficient algorithm for this purpose. In addition, the information obtained from this evaluation approach allows well-informed tradeoffs to be made along a number of axes, such as trading off motion graph capability for visual quality of the resulting animations or for motion graph size (i.e., answering "how much motion capture data is enough?"). Finally, we conduct experiments to examine the space/time scaling behavior of the evaluation method, as well as its overall stability and validity over various scenarios.

5.1 System Overview

Our system takes as input *(1)* a set of motion capture data, *(2)* visual quality requirements for the animations (represented as a model of acceptable motion editing), *(3)* a set of motion capability requirements (i.e., what tasks the character must be able to accomplish),

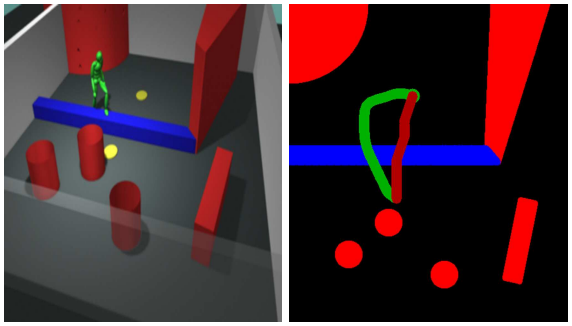


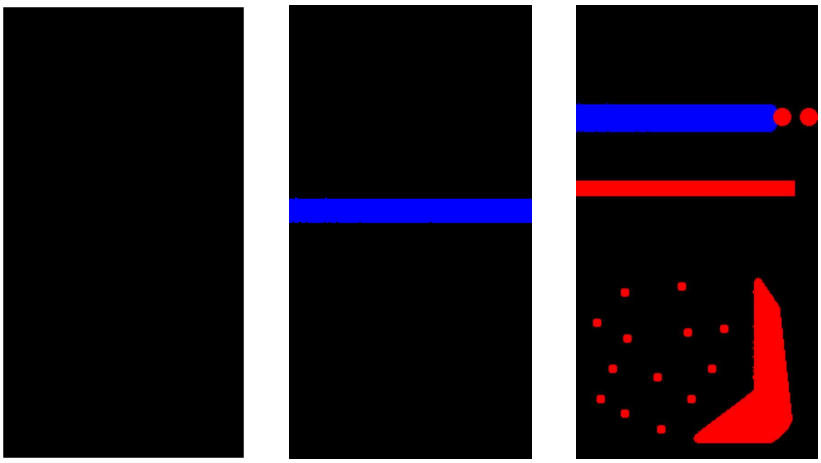
Figure 5.1: A task that cannot be accomplished in a natural way by a simple walking-based motion graph used in some of our tests. The red (dark) path is the desired path. The green (pale) path is the shortest path available using this motion graph.

and *(4)* an environment in which the character is to perform those tasks. These inputs define the *scenario* under analysis.

The motion capture data is processed to form a motion graph (see Section 1.2.1). We further process this motion graph to capture interactions with the target environment (Section 5.1.1 and Section 5.2), based on a model of motion editing (Section 5.1.2). The resulting data structure is used to measure the animated character's ability to successfully complete the required tasks in the given environment (Section 5.1.4 and Section 5.3).

5.1.1 Capturing Motion Graph/Environment Interaction

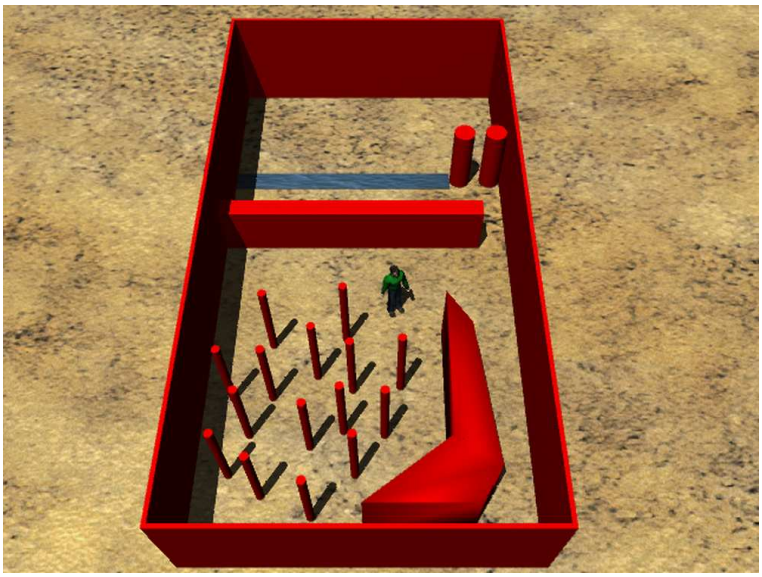
The details of a particular environment can have a very significant impact on the capabilities imparted to an animated character by a motion graph. The simplest example of this is to consider two environments (Figure 5.2(a) and 5.2(b)), one of which has a deep trench bisecting it that must be jumped over; for a motion graph with no jumping motions, the two environments will induce a very different ratio between the total space of the environment and the space which can be reached by a character starting in the lower portion. A more complex example is shown in Figure 5.2(c) and 5.2(d), where small changes to the sizes and positions of obstacles—changes that may appear superficial—produce very



(a) Empty environment

(b) Environment with trench

(c) More complex environment



(d) Complex environment (3D)

Figure 5.2: Example environments. The upper obstacle (blue) in environments (b) and (c) is annotated as a chasm or similar obstacle, meaning it can be jumped over. The environment shown in (c) and (d) is our baseline reference environment.

large differences in the ability of the character to navigate through the environment. Due to this strong environmental influence on the capabilities of a motion graph, the only way to understand how a motion-graph-driven animated character will actually perform in a given environment is to analyze the motion graph in the context of that environment.

For any given scenario to be evaluated, we need a way to capture the influence of the environment on the capabilities of the motion graph. We do this by *embedding* the motion graph into the environment—unrolling it into the environment in the manner described in Section 5.2. After that, we will be able to define how to measure the capabilities of interest of a motion graph, and examine the results of that measurement.

5.1.2 Visual Quality Requirements

Embedding a motion graph into an environment so that its capabilities can be measured requires making a choice of editing model. Generally, the visual quality of motions created in a motion graph will start at a high base due to the verisimilitude of the underlying motion capture data, and will be degraded by any editing done to the motion, either due to the loss of subtle details or to the introduction of artifacts such as foot sliding. Such editing is necessary to transition between the different motion clips of the motion graph and to precisely target the character to achieve certain goals (such as picking up an object or walking through a narrow doorway). Additionally, allowing motions to be edited increases the range of motions available to the animation system, increasing the capabilities of the available animations while decreasing their visual quality.

We assume that current and future research on motion editing (e.g., Kovar et al. [2002b], Mukai and Kuriyama [2005]) and on perceptual magnitude of editing errors (e.g., Reitsma and Pollard [2003], Harrison et al. [2004]) will allow animators to determine the extent to which a motion clip can be edited while maintaining sufficient visual quality to meet the given requirements. Given those bounds on acceptable editing, a motion clip starting at a point p and character facing θ which would normally terminate at a point p' and facing θ' can be edited to terminate at a family of points and facings $\{p_i, \theta_i\}$ containing $\{p', \theta'\}$. In general, the better the editing techniques and the lower the visual quality requirements, the

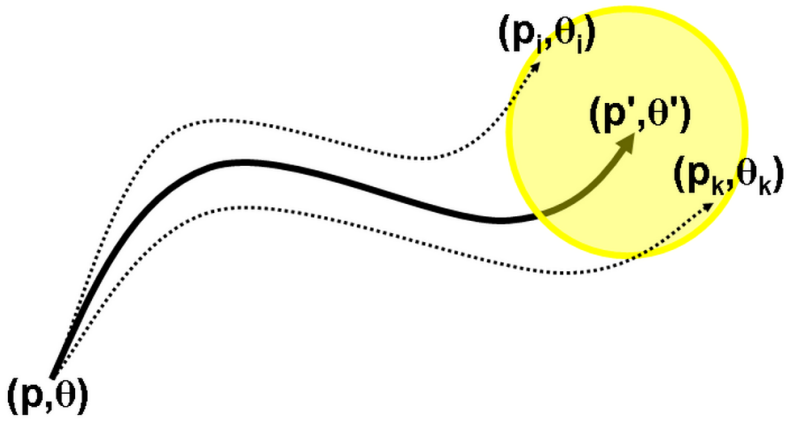


Figure 5.3: From an initial root position p and facing direction of the character θ, a motion clip can be edited to place its endpoint anywhere within its footprint (yellow region). Dotted arrows show the edited paths corresponding to the possible endpoints (p_i, θ_i) and (p_k, θ_k).

larger this family of valid endpoints for a given clip. This family of points is known as the *footprint* of the motion clip; given a specific starting point and direction, the footprint of a clip is all of the endpoints and ending directions which it can be edited to achieve without breaching the visual quality requirements (see Figure 5.3).

Design of motion editing models is an open problem; we use a simple linear model, meaning the amount of acceptable editing grows linearly with both distance traveled and motion clip duration. The intuition behind this model is the observation that taking an error of constant size and spreading it out over more motion will result in a smaller error per unit motion (per metre of distance or per second of duration). To a first approximation, the perceptual results from Chapter 4 suggest that the perceptual magnitude of an error will tend to decrease approximately linearly with decreases in its per-unit magnitude, at least for certain types of error. While this linear model may be somewhat simplistic, we

note that it only affects the size and shape of the editing footprint of a motion, and that the rest of the evaluation system can be used as-is with any alternative method of defining a clip's editing footprint.

5.1.3 Editing Footprints

The editing model is based on the intuition that the amount a motion can be changed will tend to grow with the distance it covers, as well as with the time spanned by the clip—in essence, we assume an error will be acceptably small if it is small relative to the rate at which other actions are taking place.

We specify the amount that V, the root position and orientation, can be adjusted as a linear function of both distance traveled and number of frames in the clip:

$$abs\,(V_i - V') < (s + \alpha d) \begin{bmatrix} r_x \\ r_z \\ r_\theta \end{bmatrix} \tag{5.1}$$

where $(V_i - V')$ is the vector difference between the new and original root configurations, s is the arclength of the original motion clip, d is the duration of the motion clip, α is a scaling factor used to weight arc length vs. duration in terms of editing importance, and $(s + \alpha d)[r_x \;\; r_z \;\; r_\theta]^T$ is the size of the ellipsoidal footprint representing allowable edits to root configuration.

The new path taken by the clip is computed by adding the currently-accumulated portion of $(V_i - V')$ to each frame:

$$V_i(k) = V'(k) + \frac{(s(k) + \alpha d(k))}{s + \alpha d}(V_i - V') \tag{5.2}$$

where $V_i(k)$ is the edited (position,orientation) of the character's root at frame k, $V'(k)$ is the unedited (position,orientation) of the character's root at frame k (i.e., corresponding to original endpoint (p', θ')), $s(k)$ is the arclength of the path through frame k, and $d(k)$ is the duration of the path through frame k.

5.1.4 Motion Capability Requirements

We define the *capability* of a motion graph within an environment as its ability to create motions that fulfill our requirements within that environment. The appropriate metric for evaluating character capabilities depends on the tasks the character is expected to perform. For localized tasks such as punching a target or kicking an object, the task may be to efficiently navigate to the target and contact it with an appropriate velocity profile, while for some dance forms the task may be to string together appropriate dance sequences while navigating according to the rules of the dance being performed. This paper focuses on navigation and localized actions (such as punching, ducking, or picking up an object). We focus on the following requirements, chosen so as to approximate the tasks one might expect an animated character to be required to perform in a dynamic scenario, such as a dangerous-environment training simulator or an adventure game:

- Character must be able to move between all major regions of the environment.
- Character must be able to perform its task suite in any appropriate region of the environment; for example, picking up an object from the ground regardless of that object's location within the environment.
- Character must take a reasonably efficient path from its current position to any specified valid target position.
- The previous two items should not conflict; i.e., a character tasked with performing a specific action at a specific location should still do so efficiently.
- Character must respond quickly, effectively, and visibly to user commands.

5.2 Embedding into the Environment

The manner in which a motion graph can move a character around an environment is strongly affected by the particulars of that environment, such as impassable obstacles (walls), obstacles which can by bypassed only by using particular types of motions (jumpable

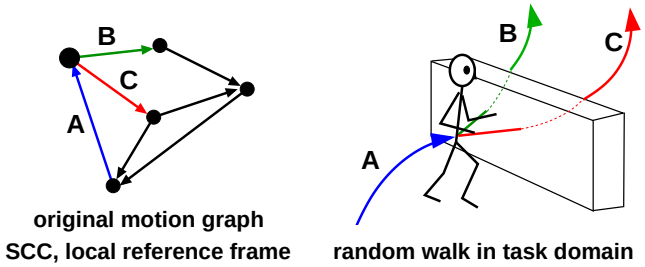


Figure 5.4: (Left) A motion graph may be constructed using a reference frame local to the character to preserve flexibility in the character's motion. (Right) Using this motion graph to drive the character through an environment with obstacles can result in dead ends. Because obstacle information is not available in the local reference frame (left), extracting a strongly connected component (SCC) from the original motion graph does not guarantee that the character can move through a given environment indefinitely.

chasms), or regions where only specific styles of motion are permitted (sneaking). To capture this influence, we must embed the motion graph into the environment, "unrolling" it to see how it interacts with the environment.

5.2.1 Requirements for Embedding

To illustrate the design considerations for an embedding approach, consider attempting to compute the value of a sample metric: Simple Coverage. The value of this metric is just the fraction of the environment through which the character can navigate freely.

One immediate challenge is to form a compact description of the space of character trajectories – simply unrolling the motion graph will rapidly lead to an exponential explosion in the number of paths being considered. A second key challenge is to consider only paths which do not unnecessarily constrain the ability of the character to navigate. For example, avoiding dead ends is necessary for an autonomous character with changing goals, or for a character subject to interactive control. Even though the original motion graph has no dead ends, obstacles can cause dead ends in the environment (see Figure 5.4).

In order to meet these challenges, we discretize the environment, approximating it with a regular grid of cells. At each cell, we embed all valid movement options available to the character. This embedding forms a directed graph, of which we only use the largest strongly connected component (SCC). Note that this SCC is in the *embedded* graph, not within the original motion graph. For example, in Figure 5.4, link A is not valid at that position within the environment, and so would be discarded *for that position only*; link A may not be blocked in other parts of the environment, and hence may be a part of the (embedded) SCC in some positions and not in others.

While computing the embedded SCC has a high one-time cost, that cost is amortized over the many tests run to compute the metrics of interest, each of which is made more efficient by having the information embodied in the SCC available. Computing this SCC in advance, then, allows efficient and correct computations of the capabilities of the motion graph under investigation.

5.2.2 Discretization

We discretize the environment along the following dimensions, representing the state of the character:

X The x-axis groundplane position of the character's root.

Z The z-axis groundplane position of the character's root.

Θ The facing direction of the character.

C The clip which brought the character to this (X, Z, Θ) point (i.e., the character's pose).

C is inherently discrete, but the groundplane position indices (X, Z) are determined by discretizing the environment into a grid with fixed spacing distance between adjacent X or Z bins. Similarly, the facing of the character is discretized into a fixed number of angular bins.

Typically, a ground-plane position specified by only X and Z is referred to as *grid location*; adding the facing angle Θ denotes a *grid point*; finally, specifying the pose of the character by including the motion clip that brought the character to that point specifies a *grid node* in the 4D state space.

Each grid node can be considered a node of a directed graph, where $[x, z, \theta, c]$ has an edge to $[x', z', \theta', c']$ if and only if:

- Clip c can transition to clip c' in the original motion graph
- Given a character with root position (x, z) and facing θ, animating the character with clip c' places (x', z', θ') within the footprint of c'.
- The character follows a valid path through the environment (i.e., avoids obstacles and respects annotations) when being animated from (x, z, θ) to (x', z', θ') by the edited clip c'.

A pair of nodes (u, v) is known as an *edge candidate* if the first two criteria hold (i.e., the edge will exist unless the environment renders it invalid).

For example, suppose clip A can be followed by either clip B or clip C in the original motion graph. In the environment represented in Figure 5.5, the starting point $([1, 1, \frac{\pi}{2}, A]$ is marked with a blue star, endpoints of valid edges with green circles, and edge candidates which are not valid edges are marked with red squares. The edge $([1, 1, \frac{\pi}{2}, A], [5, 1, \frac{\pi}{2}, C])$ is in the embedded graph, since its endpoint is within C's footprint and the path to it is valid. By contrast, there is no edge from $([1, 1, \frac{\pi}{2}, A], [5, 0, \frac{\pi}{2}, C])$, since the edited path of clip C (dotted arrow) is not valid in the environment (it intersects an obstacle).

Note that this discretization approach offers a tradeoff between precision and computation. Our metrics generally give broadly similar results across a range of grid sizes (see Section 5.4.6), suggesting that the values computed in a discretized setting are reasonable approximations of those values in the continuous case.

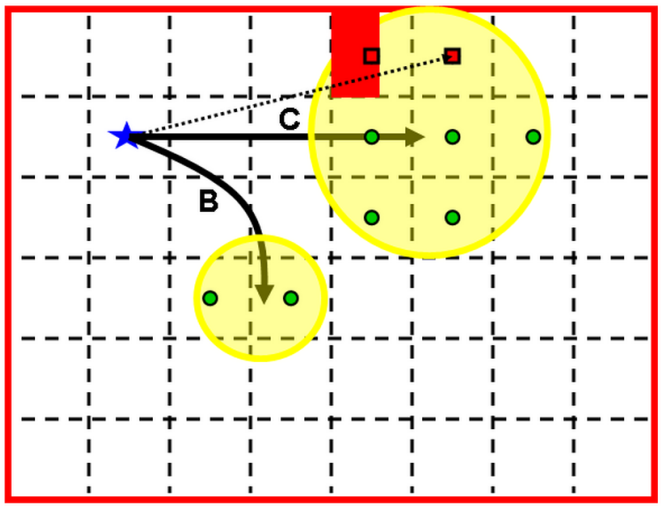


Figure 5.5: Computing edges from a single node; the endpoints of valid edges are marked as green circles. Note that not all nodes within a clip's footprint are the endpoints of edges, as obstacles can obstruct the edited path from the source to the target node (dotted path).

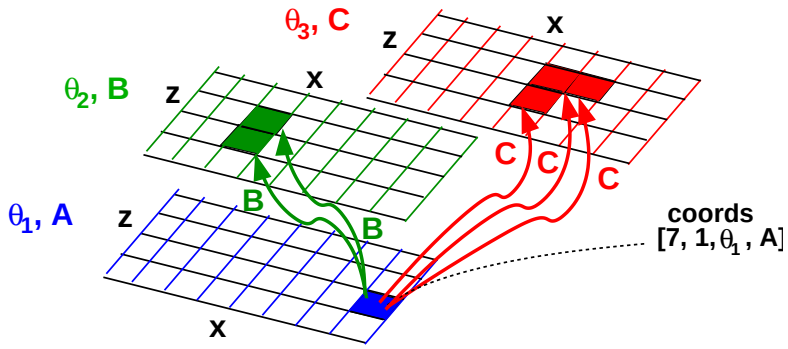


Figure 5.6: Example of forming links for an embedding. The grid lies in a 4D space, indexed by 3D workspace configuration (x, z, θ) and by incoming motion clip (A, B, or C). Links from node $[7, 1, \theta_1, a]$ are formed by looping over all motion clips that follow from A and finding the destination nodes that are within the corresponding editing footprints.

5.2.3 One-Step Unrolling

For motion clips that originate and terminate only at grid cell centers, embedding a motion graph into a task domain can be accomplished using a one-step unrolling process. Figure 5.6 shows an example of this process on the 4D state space defined in Section 5.2.2, with node $[7, 1, \theta_1, A]$ being the state in which the character has arrived at position $(7, 1)$ and orientation θ_1 after playing clip A. For each grid node in the 4D state space, all edge candidates out of that node are created.

Note that repeating this process just once for each grid node results in forming all edges in the graph. All starting configurations, all motion clips, and all of the editing variations that are allowed in our model are taken into account. Note further that many nodes will have multiple incoming edges; while node $[7, 1, \theta_1, A]$ can only ever be reached as a result of playing clip A, it can potentially be reached by playing A from any of several different locations, due to the allowable editing modeled.

After this one-step unrolling process has been performed for all nodes, we use Depth-First Search (DFS) to find the largest strongly connected component (SCC) in the resulting directed graph [Cormen et al., 2001]; this SCC is the desired embedded graph.

5.2.4 Space-Efficient Unrolling

The main limitation of the one-step algorithm (Section 5.2.3) is that it requires explicitly storing all edges needed to compute the embedded graph. While this is an efficient and sensible approach for smaller environments or coarser grids, due to memory considerations it significantly limits the size of environments and the resolution at which they can be processed.

As an alternative that requires much less memory, we propose a flood-based algorithm which takes advantage of the fact that edges outgoing from or incoming to a node can be computed only as needed. For grid nodes u and v, an edge candidate (u, v) will be in the SCC if and only if nodes u and v are both in the SCC, and (u, v) is an edge of the embedded graph (i.e., its associated path is valid within the environment), so storing only

the nodes in the SCC allows computation of the correct edges as needed. Note that the set of incoming edges can be defined exactly analogously, and that these sets must agree.

Finding the SCC for use with this on-the-fly approach can be done efficiently in the discretized environment (see Figure 5.7):

- Choose a source node s (green "S").
- Flood out from s, tagging all reachable nodes (blue vertical hashes).
- Flood into s, tagging all nodes reaching it (red horizontal hashes).
- Intersection of the two tagged sets is the SCC (purple "+").

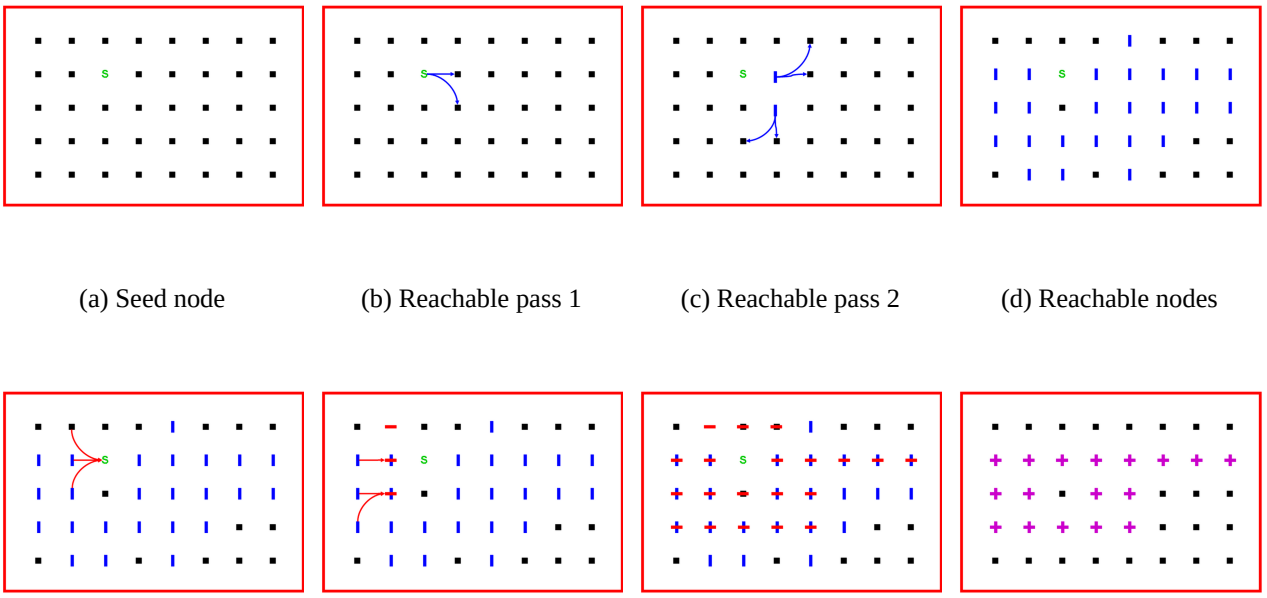
The flood out from s is referred to as the "reaches flood" (i.e., flooding into the places which can be reached from s), and the flood into s is referred to as the "reaching flood".

In practice, motion graphs embedded into environments typically result in an embedded graph with a single SCC whose number of nodes is linear in the total number of nodes in the environment which are not obstructed by obstacles[1]; 10-25% of the total number of nodes in the environment is a typical size for an SCC. Accordingly, s can be chosen uniformly at random until a node inside the SCC is found without an asymptotic increase in time[2].

Flooding is done using a multi-pass Breadth-First Search. For finding the set of Reachable nodes, bit arrays are used to tag which nodes are Active or Reached. Initially only s has its Active bit set, and no nodes have their Reached bits set. At each pass, the algorithm iterates through each node k in the environment. If Active(k) is set, then k is expanded. Expanding a node consists of setting Reached(k) to true, setting Active(k) to false, and finding the set of edges for k. For each of those edges j, Active(j) is set if and only if

[1]There is some evidence this should be expected; see, for example, Muthukrishnan and Pandurangan [2005] regarding this property in random geometric graphs.

[2] In practice, the actual overhead is minimal if fast-fail tests are used. Testing that source nodes have at least 5,000 reaching and reachable nodes efficiently rejects most nodes outside the SCC; in our experiments, testing that the source node can reach itself was the only other fast-fail test required.



(a) Seed node (b) Reachable pass 1 (c) Reachable pass 2 (d) Reachable nodes

(e) Reaching pass 1 (f) Reaching pass 2 (g) Reaching nodes (h) SCC

Figure 5.7: Steps of the embedding algorithm. (a) A seed node (green “S”) is chosen. (b) First pass of the reachable flood marks nodes reachable in one step from the seed node. (c) Second pass marks additional nodes reachable in two steps. (d) All reachable nodes are marked (blue vertical hashes). (e) First pass of the reaches flood marks nodes which reach the seed node in one step. (f) Second pass marks additional nodes reaching in two steps. (g) All reaching nodes are marked (red horizontal hashes). (h) Intersection of set of reachable nodes and set of reaching nodes (purple “+”) is the SCC of the embedded graph.

Active(j) and Reached(j) are both false (i.e., j has not been tagged or expanded yet). The algorithm ends when there is no node k such that Active(k) is set.

Since any node's Active bit is set to true at most once, the algorithm terminates. Since all nodes reachable from s will have their Active bit set, the algorithm correctly computes the set of nodes reachable from s. Each pass of the algorithm touches each node in the environment, but each edge is expanded only once per flood direction. Accordingly, the runtime of the algorithm is the cost of computing whether each edge is valid in the environment plus the cost of reading each node's tags for each iteration until the search is complete; i.e., $\Theta(e) + \Theta(D * N)$, for e the number of edges in the SCC, N the number of nodes in the environment, and D the diameter of the embedded graph (i.e., the maximum depth of the Breadth-First Search). In practice, the $\Theta(e)$ term dominates, making the algorithm approximately as efficient as regular Breadth-First Search.

The set of nodes which can reach s is computed analogously; however, since the set of reachable nodes in already known, only those nodes need to be expanded, as no other nodes can be in the SCC. This offers substantial computational savings.

Note that storing the Reachable and Reaching sets requires two bits of storage per node, and that another bit is required as scratch memory for the Active set, for a total memory requirement of three bits per node.

5.2.5 Space Complexity

We can estimate the theoretical space complexity of the embedded graph with respect to the parameters affecting it. For a given environment, the number of nodes in the embedded graph is

$$O(n) = O(AxzaC) \tag{5.3}$$

and the number of edges in the embedded graph is

$$O(e) = O(Ax^2z^2a^2Cb) = O(nxzab) \tag{5.4}$$

where:

A = the accessible (i.e., character would not collide with an obstacle) area of the environment, in m^2.

x = the discretization of the X axis, in grid cells per m.

z = the discretization of the Z axis, in grid cells per m.

a = the discretization of the character's facing angle, in grid cells per 2π radians.

C = the number of motion clips used (i.e., the number of nodes in the original motion graph).

b = the mean branching factor of the original motion graph.

Equation 5.3 simply notes that the number of nodes in the SCC is linear in the total number of nodes into which the environment is discretized. Equation 5.4 demonstrates how the average number of edges per node is constant with increasing environment size, but increases linearly with increasing resolution (in each dimension), due to more grid points being packed under each clip's editing footprint. Note that a clip's editing footprint increases in size with the length and duration of the clip, but that enforcing a minimum and maximum clip duration when creating the motion graph ensures two clips' footprints will differ in area by at most a constant factor.

The one-step unrolling algorithm presented in Section 5.2.3 stores the edges of this embedded graph explicitly, and so has space complexity $O(nxzab)$. By contrast, the flood-based algorithm presented in Section 5.2.4 has space complexity $O(n)$; i.e., linear in the number of nodes in the embedded graph, rather than in the number of edges, meaning the amount of memory required is reduced by a factor of $\Theta(xzab)$. In practice, the asymptotic nature of the big-Theta notation hides an additional large constant factor of improvement, as only a few bits are required to compute and store each node.

The empirical scaling behavior of these two algorithms with respect to environments of increasing size is examined in Section 5.4.5.

5.2.6 Time Complexity

While the space-efficient algorithm requires asymptotically less space than the one-step unrolling algorithm, it typically does not require asymptotically more time.

The one-step algorithm computes each edge candidate in the environment once, stores valid edges, and accesses each cached edge twice when using Depth-First Search to find the SCC. The runtime for this algorithm is:

$$T_{rp04} = \Theta(oNbf + 2E) = \Theta(oE + 2E) = \Theta(oE) \tag{5.5}$$

where N is the number of unobstructed nodes in the environment, f is the mean number of nodes under the footprint of a clip, E is the number of edges in the environment, and o is the mean number of obstacle-intersection tests required when computing each edge.

When expanding a node, the algorithm of Section 5.2.4 computes either incoming edge candidates or outgoing edge candidates. As a node will be expanded only once in each direction (Reachable/Reaching), each outgoing edge candidate from that node and each incoming edge candidate to that node will be computed only once when finding the SCC. Accordingly, the runtime of our algorithm is:

$$T_{flood} = \Theta(on_rbf + onbf') = O(oE) \tag{5.6}$$

where n_r is the number of nodes reachable from the SCC, and f' is the mean number of nodes under the *reverse footprint* of a clip (i.e., the set of start points $\{p, \theta\}$ such that the footprint of clip c played from that point will contain the end point (p', θ')).

In the typical case, where $n = \Theta(N)$ (i.e., the number of nodes in the SCC is linear in the number of unobstructed nodes in the environment), the runtimes for T_{rp04} and T_{flood} are asymptotically equivalent. Substantial runtime differences can occur from two sources: first, f' will typically be larger than f, due to the manner in which angular edits are made to clips; second, n_r and especially n will typically be smaller than N. Empirical runtimes are examined in Section 5.4.5.

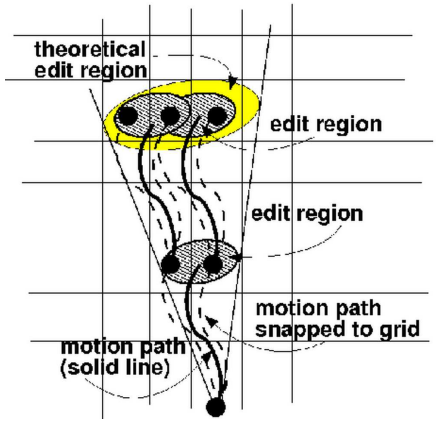


Figure 5.8: In our grid-based algorithm, the end of each motion segment is snapped to the centers of grid points inside the editing footprint for that motion segment. This figure compares how growth in this theoretical edit region compares to the regions actually used in the grid-based algorithm.

5.2.7 Correctness of the Embedding Algorithms

Our embedding algorithm is resolution complete in the sense that a sufficiently fine grid will capture all significant variations in character paths that are possible given our motion graph, task domain, and motion editing model. However, at practical grid resolution, some possible paths will be lost. Figure 5.8 shows an example. Consider a straight walking motion that can repeat. Figure 5.8 shows a sketch of edit regions grown using our approach, which requires forcing motion clips to terminate at cell centers. The theoretical edit region that would result from playing the same motion clip twice in succession is also shown. Eventually, (after playing the motion clip three or four times in succession), the theoretical edit region will contain grid centers that are not captured by our algorithm. As the grid is made finer, the practical effect of this approximation will decrease, and we discuss the results of experiments testing the stability to changes in grid resolution of our analysis approach in Section 5.4.6.

The embedding algorithms are made conservative by making connections only to grid centers within the edit region associated with a motion segment. When the algorithms are

implemented in this way, no paths can be generated that are not possible given the motion graph, task domain, and motion editing model.

5.2.8 Obstacles and Annotation Constraints

Obstacles and other constraints are required to make an environment interesting and realistic. We do collision detection with a cylinder of radius 0.25m around the character's root; hence, all obstacles can be described as 2D shapes (such as circles, axis-aligned rectangles, or general quadrilaterals) with annotations. Annotations can be used to define very general constraints on the motions usable in the environment; annotations we used included:

- Obstacle can not be traversed (high wall).
- Obstacle can be traversed by jumping motions (chasm).
- Obstacle can be traversed by jumping or stepping motions (low wall).
- Region can be traversed by only sneaking motions (area near guard).
- Region can not be traversed by jumping motions (normal locomotion should not be assumed to include jumping for no reason).
- Picking-up motions should not be used unless the user selects the action.

Most of these annotations—either defining different obstacle types which interact differently with different motion types, or defining regions where the character must act in a certain way—are self-explanatory, and are handled automatically during collision-detection; any path which violates an annotation constraint will not be expanded, and hence cannot contribute to the embedded graph. Annotations such as the last one are more complicated, as some motions need to be designated as strictly-optional (and hence cannot be used to connect parts of the embedded graph) but readily-available. We refer to these as *selective actions*.

5.2.9 Selective Actions

Selective actions are those actions which must be strictly optional (i.e., selectable by the user at will, but otherwise never occurring in unconstrained navigation). A user may choose to have the character perform a picking-up action when there are no nearby objects on the floor, for example, but it would look very strange for such a spurious pick-up action to show up in automatically-generated navigation.

An embedded graph created from all motions will not respect this requirement; in particular, parts of the environment may be reachable only via paths which require selective actions, rather than by regular locomotion, and hence those parts of the environment should not be considered reachable.

Selective actions are handled by a slight modification to the embedding algorithm. First, the embedding algorithm described previously is run with only those actions deemed "locomotion" permitted; all other actions are deemed "selective", and are not expanded, although any selective-action nodes reached are marked as such. An exception is made for selective actions which are necessary for traversing an annotated obstacle; e.g., jumping motions are permitted only insofar as they are required to traverse obstacles such as the jumpable-chasm obstacle in Figure 5.2(b). This creates a backbone embedded graph composed of locomotions.

Next, each reachable selective action is tested to see if it can reach the current embedded graph; if so, all paths of reasonable length linking it into the SCC are added to the embedded graph. For reasons of efficiency, these paths are added in an approximate manner; i.e., not all paths from the selective action which are added will actually be in the SCC. Accordingly, the embedding algorithm is rerun to ensure an SCC; this time, however, the resulting SCC must be a subset of the previous embedded graph (i.e., the original SCC plus the added paths). Hence, that information can be used to substantially reduce the number of nodes expanded by the embedding algorithm.

The cost for creating this *augmented SCC* is the cost for creating the initial SCC, plus the cost of adding in paths from each selective action out to a fixed depth, plus the cost of

re-running the flooding algorithm on this subset of nodes. The total cost is:

$$T_{aug} = T_{flood} + T_{paths} + T'_{flood} = O(oE) + O(onbfh) + O(oE) = O(oE) \qquad (5.7)$$

where h is the mean depth in edges of the added paths. Comparing to Equation 5.6, computing the augmented embedded graph is asymptotically equivalent to computing the regular motion graph with all motions treated equally, and in practice requires about twice as much computation time. In addition, storing the "selective" actions during computation of the initial embedded graph and storing that initial SCC during computation of the final, augmented SCC requires an extra bit of storage per node, for a total cost of four bits per node in the environment.

We used this type of embedded graph for all of our experiments.

5.3 Motion Graph Capability Metrics

To measure the capability of a motion graph, we define metrics which evaluate the motion graph's ability to fulfill the requirements identified in Section 5.1.4.

5.3.1 Environment Coverage

The most basic problem a motion graph can have in an environment is simply that it is unable to navigate the character effectively enough to access large regions of that environment. Accordingly, the environment coverage metric is designed to capture the ability of the character to reach every portion of its workspace without becoming irretrievably stuck, similar to the viability domain from viability theory (see below). For navigation, this workspace is represented by discretized grid points $\{X, Z, \Theta\}$ (see Section 5.2.2).

We define grid point (X, Z, Θ) as *covered* by clip c if the center of (X, Z, Θ) is within the footprint of c (see Figure 5.9).[3] From this occupancy information we compute Envi-

[3] An alternative definition of coverage is that any grid point containing any root position of any frame of any clip is covered. In practice, these definitions give essentially identical results, but the former definition is significantly faster to compute.

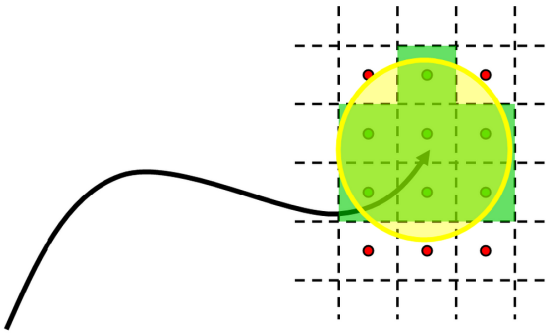


Figure 5.9: A grid cell is covered by a clip if that clip's footprint includes the cell's center. (Covered grid cells are in green.)

ronment Coverage as:

$$C_{XZA} = \frac{\sum_i covered(i)}{\sum_i collisionFree(i)} \tag{5.8}$$

where the numerator contains a count of all 3D grid points which are covered by at least one clip in the embedded graph, and the denominator contains a count of all grid positions which are theoretically valid for character navigation (i.e., some motion type exists for which the character would not collide with an obstacle or violate an annotation constraint; see Figure 5.11). C_{XZA} is a discretized estimate of the fraction of viable (x, z, θ) workspace through which the character can navigate under arbitrary control. C_{XZ}, the 2D equivalent, can be defined analogously.

In order to assist a user in diagnosing and understanding potential problems in a dataset, we use a brushfire algorithm (e.g., Lengyel et al. [1990]) to identify local maxima in terms of distance to the nearest covered grid position (see Figure 5.10). This algorithm can be used to display the locations of the largest gaps or holes. Arrows indicate orientation at local maxima and typically point toward or away from the nearest obstacle. The most obvious problem that can be inferred from this figure is that there are no motions in the simple walking-based dataset used for this example that allow the character to stop and turn in place in this simple example scenario.

Regions with low or no coverage typically indicate that the entrances to those regions

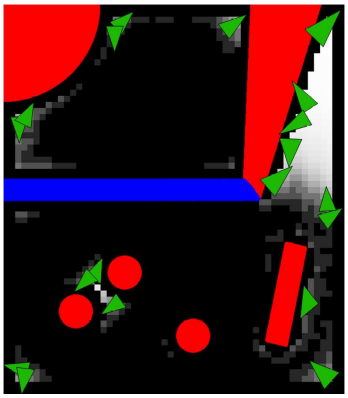


Figure 5.10: Holes in coverage for a simple environment. The brightest regions are the farthest distance from covered, collision-free cells. This figure shows at each x, z position the maximum distance over all orientations θ. Arrows indicate orientations of some of the local maxima.

from the rest of the environment are highly constricted, and/or there are few ways to place the motions required to enter or move within that region without colliding with obstacles or violating annotation constraints.

Viability Theory

Viability theory (see, for example, Aubin [1990]) examines some concepts similar to those we evaluate with our metrics, albeit in a different domain. In particular, our use of a strongly connected component (SCC) for the embedded graph is analogous to evaluating our metrics on the viability kernel of the system, although slightly more restricted. SCCs have the *viability property*, which means that for any start state inside the SCC there exists a path of arbitrary length which stays inside the SCC. Our embedded SCC is hence a *viability domain* of the set of states in the discretized environment which could legally be occupied by the character (i.e., without colliding with an obstacle).

We note, however, that our SCC may not be the *viability kernel* – maximum-size viability domain – of the system; since we take only the largest SCC, a system with a second,

Figure 5.11: Coverage over a simple environment. Obstacles are in red; covered areas are in grey. Lighter grey means more coverage (i.e., the grid location is covered by more clips).

much smaller SCC would include that second, spurious SCC in the viability kernel. In addition, viability theory tends to concern itself with non-deterministic systems where knowledge of the future is difficult or impossible to obtain, and hence cannot take advantage of some of the simplifying properties of our problem, such as the known connectivity of the initial motion graph. Nevertheless, viability theory is potentially an interesting lens through which to view our work and the general question of capability.

5.3.2 Action Coverage

The coverage for action k is defined analogously to Environment Coverage. The 2D version is:

$$C_{XZ,k} = \frac{\sum_i covered_k(i)}{\sum_i collisionFree(i)} \tag{5.9}$$

where $covered_k(i)$ is true if and only if there exists an edge in the embedded graph such that animating the character with that edge causes the central frame of action k to occur in grid location i.

5.3.3 Path Efficiency

The path efficiency metric is designed to evaluate the ability of the motion graph to allow the character to navigate efficiently within the accessible portion of the environment. Each path will have a particular value for path efficiency, and hence the goal is to estimate the distribution of path efficiencies over the set of all possible paths through the environment. To estimate this distribution, we must make an assumption about the manner in which paths will be specified. In this chapter, we examine the problem of point-to-point navigation, where the (X, Z) values of start and end points are specified (such as with a mouse) and a path must be found for the character to travel between these two points[4]. Ideally, a near-minimal-length path would exist for all pairs of start and end positions. The metric for relative path length in point-to-point navigation is:

$$E_P = \frac{pathLength}{minPathLength} \tag{5.10}$$

where $pathLength$ is the length of the shortest path available to the character from the start point to the end point and $minPathLength$ is the length of an ideal reference path between those two points (see Figure 5.12).

When evaluating the path efficiency ratio of a $(start, end)$ pair, we work with the embedded graph described in Section 5.2. Using this graph ensures that only paths which do not result in dead ends are considered, and also significantly improves the efficiency of shortest path calculations.

Given this embedded graph, we use Monte Carlo sampling to estimate the distribution of path efficiency ratios within the 4D space defined by all valid $(start, end)$ pairs. Start and end positions are selected uniformly at random from the set of grid positions which have non-zero coverage (see Section 5.3.1). Value $pathLength$, the shortest path available to the character, is computed using A* search through the embedded graph. Value $minPathLength$, the shortest path irrespective of available motions, is estimated using

[4]Several similar definitions of this metric are possible, such as accepting paths that pass through the target point, rather than only ones ending there. In practical embedded graphs, the results will tend to be very similar, but the definition used here can be computed more quickly, and is closer to the "go to the click" interaction paradigm which motivates the metric.

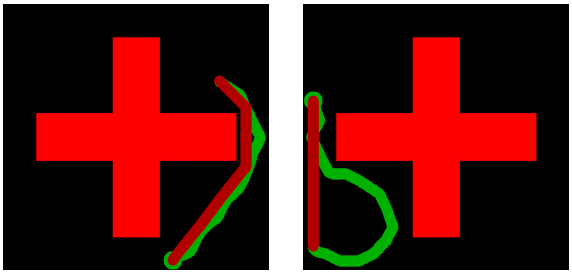


Figure 5.12: Paths through a simple environment. Obstacles are in bright red, the *minPathLength* path is in deep red, and the *pathLength* path is in green (starting point is marked with a wider green dot). (Left) A typical path. (Right) The theoretical *minPathLength* path can head directly into the wall, while the *pathLength* path, which relies on motions in the SCC, must end in a state that allows for further movement.

A* search over the discretization grid imposed on the environment, with each 2D (X, Z) grid location being connected to all neighbors within five hops in either X or Z (i.e., an 11x11 box). Path efficiency is then computed as in Equation 5.10. By default, paths through the embedded graph are computed using all available motions (i.e., all motions which are not specifically banned by annotations in the environment); an alternative approach is to evaluate paths using only a restricted subset of the available motions, such as only pure locomotions (i.e., sneaking and running).

Due to the highly discrete nature of motion graphs, extremely short paths may have unrepresentative path efficiency ratios according to the presence or absence of a clip of particular length in the motion graph; to reduce this erratic influence and to consider paths more representative of typical user-controlled navigational tasks, we throw away $(start, end)$ pairs whose linear distance is less than 4m and re-select start and end candidates. Since the space complexity of $A*$ grows exponentially with path depth and we expect more sophisticated planning algorithms will be used in practice, we similarly throw out $(start, end)$ pairs whose $minPathLength$ is greater than 7m. We believe such longer paths will not have significantly different path efficiency distributions from paths in the $4 - 7$m range, due to the ability to chain shorter paths together to create long ones. Note

also that extremely slow-moving motions, such as idling in place, can skew the results. We treat each clip as having a minimum effective speed of 0.5m/s for the purposes of minimum path computation in order to help filter out spurious paths such as ones which make all turns by stopping and turning in place. Note that this speed is virtual only; while a path which includes two seconds of idling would have an additional 1m added to its length during the search for the shortest path, if selected its true length would be used to calculate the path efficiency ratio.

A poor score on the Path Efficiency metric usually results from the most direct route between the start and end locations passing through areas of very limited coverage (and, hence, very limited path options). A common reason for this type of bottleneck is that all paths connecting one region to the rest of the environment are channeled through a small number of connecting motions, sharply restricting the breadth of valid paths.

5.3.4 Action Efficiency

This metric measures the efficiency overhead required to execute a desired action, such as picking up an object lying on the floor across the room. The metric is measured in a Monte Carlo fashion exactly analogous to that described for Path Efficiency, and computes a very similar ratio:

$$AE_a = \frac{pathLength_a}{minPathLength} \tag{5.11}$$

where a is the action type in question, $minPathLength$ is the length of the reference path computed exactly as per the path efficiency metric, and $pathLength_A$ is the length of the shortest path through the embedded graph that ends in a clip which executes an instance of action a in the end location.

High values of this metric as compared to the Path Efficiency value for the same $(start, end)$ pair typically represent actions which are inflexibly linked into the motion graph, such as a ducking motion which can only be accessed after running straight for several meters, and which are coupled with or consist of motions which do not fit well into regions of the environment (such as a region requiring twisty paths between obstacles).

A* Planner

Our A* planner uses the following as a conservative distance estimate:

$$D_{est} = D_{S \to X} + ||X - E|| \tag{5.12}$$

where D_{est} is the heuristic estimate of distance from the start point S to the end point E, $D_{S \to X}$ is the length of the path currently being examined, and $||X - E||$ is the linear distance between X, the end of the current path, and E.

In practice, rapidly culling paths which stray too far from the straight line between the start and end locations is critical for efficient path planning. We first look for paths which are at most 10% longer than the optimal distance, which creates a very narrow oval of paths being explored and hence can be searched very efficiently. If no such path exists, the oval is progressively widened to allow examination of paths which stray further and further from the straight line between the start and end points. We use four passes, examining paths with a maximum length of 110%, 150%, 250%, and 600% of the ideal path length.

Note that our A* path planner returns the optimal path between the specified start and end locations; applications which use a different style of planner which sometimes returns sub-optimal paths will have commensurately-worse path-based metric results. In the more demanding environments, some of the optimal paths initially move a substantial distance away from the end location, so a path planner with a limited horizon would generate paths that were longer or even substantially longer than optimal, which would result in a worse score on both the Path Efficiency and the Action Efficiency metrics.

5.3.5 Local Maneuverability

Inspired by the idea of local controllability[Luenberger, 1979], the local maneuverability metric is designed to evaluate the responsiveness of the character to interactive user control. In many interactive applications, a highly responsive character is critical; if a user playing an interactive game is unable to evade traps because the animated character responds too sluggishly to user control, the game will be rendered almost unplayable. A key

requirement for an animated character being controlled interactively by a user is that the character must be able to rapidly and visibly respond to user controls.

The instantaneous local maneuverability of a character is simply the mean amount of time required for that character to perform any other action which is currently valid. At a particular moment, the instantaneous local maneuverability of the character with regard to action k is:

$$LM_k(t) = (1 - \alpha(t)) * D_{c_0} + MDP_k \tag{5.13}$$

where c_0 is the currently-playing clip, $\alpha(t)$ is the fraction of that clip already played at time t, D_c is the duration in seconds of clip c, and MDP_k is the shortest possible time to reach an instance of motion type k from the end of the current path while following paths from the motion graph. For example, if the character is 0.3s from the end of a running clip and the minimum-time path from the end of that clip to an instance of punching is 1s, then the character's Local Maneuverability with respect to punching is 1.3s.

The character's overall Local Maneuverability is the weighted sum of its per-action Local Maneuverabilities:

$$LM(t) = \frac{1}{||K||} \sum_{k \in K, k \neq T_{c_0}} (w_k * LM_k) \tag{5.14}$$

where K is the set of actions which are in the motion graph and currently valid, and w_k is the weight for action type k. This gives an overall measure of the character's ability to respond to external control, taking into account the different reactivity needs of the different motion types (i.e., evasive actions such as ducking may need to be available much more rapidly than actions such as picking up an object).

There are two ways to measure expected overall local maneuverability. The first is *Theoretical* Local Maneuverability, which is measured in the original motion graph:

$$LM_{T,k} = \frac{1}{||C||} \sum_{c \in C} (0.5 * D_c + MDMP_{c,k}) \tag{5.15}$$

where C is the set of all clips in the motion graph and $MDMP_{c,k}$ is the duration in seconds of the minimum duration path through the motion graph from the end of clip c to

any instance of motion type k. This gives a baseline for what instantaneous local maneuverability a character can be expected to have at any time under ideal conditions (i.e., the environment does not invalidate any of the minimum-duration paths). Poor theoretical local maneuverability values typically indicate poor connectivity between motion types in the motion graph.

By contrast, in-practice or *Practical* Local Maneuverability computes the expected instantaneous local maneuverability from each node of the *embedded* graph:

$$LM_P = \frac{1}{||G||} \sum_{n \in G} (0.5 * D_{c_n} + MDEP_{c_n,k}) \tag{5.16}$$

where G is the set of nodes in the embedded graph, c_n is the clip associated with embedded graph node n, and $MDEP_{c,k}$ is the duration in seconds of the minimum duration path through the embedded graph from the end of clip c to an instance of motion type k. This gives an expectation for what instantaneous local maneuverability a character can be expected to have at any time when navigating through the environment in question. Comparing this to its theoretical equivalent can provide information about the restrictions the environment places on responsiveness to user control.

Note that computing local maneuverability for every node is computationally expensive; in practice, we use a subset of nodes selected uniformly at random (i.e., 1% sampling means any particular node was used in the calculation with probability 0.01). The stability of the result at different sampling densities is examined in Section 5.4.6.

Finally, note that local maneuverability can be computed for many slices of the data; of note are:

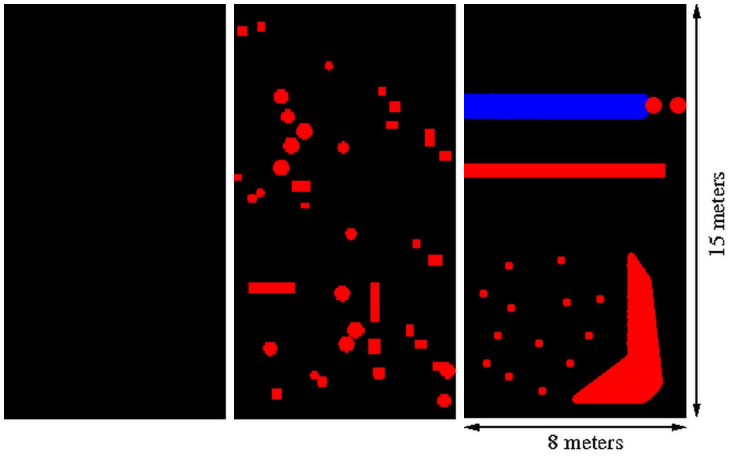
- Computing the local maneuverability for a single action which needs to be rapidly accessible at all times (such as evasive ducking).
- Computing the local maneuverability from one subset to another, such as from locomotions to all evasive maneuvers.
- Computing the local maneuverability from locomotions to same-type locomotions with the character's facing turned more than N degrees clockwise from the starting

facing, in order to gain a local measure of the character's ability to respond rapidly to a user's navigational directions (such as "run into that alcove to the left to avoid the boulder rolling at you").

- Computing statistical information on any of the above by examining the distribution of local maneuverability values at each node. Information about extremal values, such as how often it would take greater than 3 seconds to reach an evasive motion, can be particularly informative.

5.4 Results

5.4.1 Example Scenarios



(a) Empty (b) Random (c) Baseline

Figure 5.13: Evaluation environments of increasing complexity. The upper obstacle (blue) in environment (c) is annotated as a chasm or similar obstacle, meaning it can be jumped over.

The two components of a test scenario are the motion graph used and the environment

in which it is embedded. Our primary test environment was a large room with varying levels of clutter (Figure 5.13(c)). This room measures 15m by 8m, and our primary testing resolution used a grid spacing of 20cm by 20cm for the character's position and 20 degrees for the character's facing angle. We include results from other environments, including tests on randomly-generated environments of varying sizes and configurations, and other discretization grid resolutions. To take into account obstacles, we do collision detection with a cylinder of radius 0.25m around the character's root; hence, all obstacles can be described as 2D shapes with annotations.

All motion graphs used were computed using motions downloaded from the CMU motion capture database (mocap.cs.cmu.edu). Our primary dataset consisted of 50 motion clips comprising slightly over seven minutes of captured motion, with significant amounts of unstructured locomotion (running, sneaking, and idling), several examples each of several actions (jumping, ducking, picking up an object from the floor, punching, and kicking an object on the floor), and transitions between the locomotions and some of the (locomotion,action) pairs (sneak+duck, run+duck, run+punch, run+kick, sneak+pickup, etc.). Each source motion was labeled with the types of motions it contained as well as the start, center, and ending times of actions. For example, a clip consisting of running, ducking, and then running again would be labeled as running until the character visibly started transitioning to ducking, would be labeled as ducking until it was undergoing normal running movement again, and would be labeled as running motion after that point. In addition, the "duck" action would be labeled as starting when the character was fully ducking, ending when the character started to rise, and centered at the midpoint between those two times.

The per-pair values of adding a transition between any pair of frames were computed using the technique of Lee et al. [2002]. The resulting matrix of transition costs was processed in a globally-greedy manner to identify local maxima (see Section 5.4.2) and to enforce user-specified minimum and maximum clip lengths. Our primary motion graph consisted of 98 nodes (clips) connected by 288 edges, representing 1.5 minutes of motion capture data with eight distinct types of motion.

A secondary dataset consisted of 23 walking and stepping-over motions downloaded from the same repository. This secondary database consisted of 77 seconds of motion, and

was used exclusively with the step-obstacle environment (Figure 5.1).

Our embedding algorithm was implemented in Java and used a simple hash table of fixed size to cache nodes' edge lists during metric evaluation, with hash table collisions being resolved by a simple hit-vs.-miss heuristic. Running on a 3.2GHz Xeon computer, finding the final (augmented) embedded graph required about 6.7 minutes and produced an embedded graph with 218K nodes and 1.88M edges.

When playing back motion through the embedded graph, transitioning from the end of motion clip A to the start of clip B was done by warping the first frame of clip B to match the last frame of clip A, and then using quaternion splines to smoothly remove that warping over the following 0.5s. Motion editing to warp clips to grid centers was done using displacement splines on the root translation and facing angle. This editing technique of course creates footsliding artifacts, which should be cleaned up in post-processing.

5.4.2 Transition Selection

To create the motion graphs used in our experiments, we used a globally-greedy algorithm that attempted to coalesce nearby edges into hubs of high-quality transitions.

First, we find the frame-to-frame similarity matrix in a manner similar to Lee et al. [2002]. From this data, we consider only the local maxima (each maxima dominates a 5-frame radius in the matrix), and consider only maxima above a pre-set threshold. This defines a set of *candidate transitions*, and could be used directly to form a motion graph. The resulting motion graph would have no guarantees on minimum or maximum clip length, however, whereas we wished to enforce a minimum length of 0.5s and a maximum of 1.5s.

Enforcing the minimum length is accomplished by *coalescing* nearby candidate transitions; i.e., deleting all candidate transitions within a window of frames and replacing them with transitions to a single frame within that window. The algorithm for this is given in detail below; in brief, the approach is to examine each window of frames (i.e., every set of sequential frame of length 0.5s) and select the optimal frame within that set to reroute all of the transitions in that set through; this frame is given a score equal to the sum of the

quality of all acceptable (i.e., above threshold) transitions which will be re-routed through it. Once this value is calculated for all windows of frames, the globally-highest value is selected, the transitions in the corresponding window are coalesced, and the process iterates until no window contains more than one transition (i.e., all clips are at least 0.5s long). A maximum clip length of 1.5s is enforced as a post-processing step by splitting long clips.

Although $O(n^4)$ in a naive implementation, the coalescing step can be implemented to incrementally update the window values based only on changes since the last step, making the process run in $O(n^2)$ and, in practice, only a few hours even for our largest motion graphs.

Coalescing Candidate Transitions

Repeat

- Find window w with $count(w) > 1$, frame $c \in w$ that maximizes $value(w, c)$
- $coalesce(w, c)$

Until $count(w) \leq 1 \; \forall$ windows w

window(s, f, m): the set of m frames in source file s ranging from f to $f + m - 1$. Here, m is always the minimum clip size, and w will always refer to a window of this sort.

count(w): the number of frames $f \in w$ s.t. $\exists$ frame $g \in DB$ s.t. $equivalency(f, g) > 0$.

DB: the set of frames of the input source motion files

value(w,c): $\sum_g equivalency(c, g) \; \forall g$ s.t. $\exists f \in w$ s.t. $equivalency(f, g) > 0$. i.e., the value of window w if all transitions must go through frame c.

equivalency(f,g): the similarity of frames f and g, if that value is greater than the acceptance threshold, else 0.

coalesce(w,c): $\forall g \in DB$ s.t. $\exists f \in w$ s.t. $equivalency(f, g) > 0$ recompute $equivalency(c, g)$ and set $equivalency(f, g)$ to 0. i.e., force all transitions inside window w to use frame c or no frame at all.

Motion Graph		
Clips	**Transitions**	**Motion**
98	288	87s
190	904	163s
282	1,712	248s
389	3,198	350s

Table 5.1: Clips, transitions, and total amount of motion contained by the different motion graphs created for our main evaluations. The full motion database contained 439 seconds of motion.

Crafting Representative Motion Graphs

The frame-to-frame similarity matrix computed above will depend strongly on the precise details of the input motion capture clips. In practice, even a relatively small change in the transition qualities between an input file and other files in the database can result in a substantially different motion graph. If one running motion is replaced by another, for example, or even if the same running motion is used but with a different transition-selection threshold, a very different motion graph can result due to the discrete nature of taking local maxima in the similarity matrix, coalescing them to enforce minimum clip lengths, and taking the largest strongly connected component of the result.

In order to ensure that high-quality and representative motion graphs were used for our tests, each of the 50 input files was assigned a "desirability" weight which directly modified the relative quality of any transitions to or from that motion. These weights were manually adjusted until an acceptable motion graph was obtained. Our baseline motion graph was crafted via these weights to have a representative mix of all available motion types, with an emphasis on the two main locomotions (running and sneaking), a smaller amount of idling motion, and one to four examples of each of the actions (jumping, picking-up, ducking, punching, kicking) in the database. In addition, we ensured that each instance of an action was directly accessible from locomotion (i.e., it was not necessary to pass through a kicking motion to reach a ducking motion, for example), and that all three

types of locomotion (including idling) could be reached from each other without passing through an action clip (i.e., the motion graph consisting of just the three locomotions was also a strongly connected component).

Three larger motion graphs were created from the primary motion database in the manner detailed above (see Table 5.1).

5.4.3 Baseline

Table 5.2 shows evaluation results from our basic test scenarios. XZ Coverage is the fraction of collision-free (X,Z) grid cells in the environment which contain at least one node of the embedded graph; XZA Coverage is the analogous quantity taking into account character facing (see Section 5.3.1). Local Maneuverability (Section 5.3.5) is the minimum-duration path to an instance of the "pick" action in either the original motion graph (Theoretical LM) or the embedded graph (Practical LM). Path Efficiency is the ratio of the distances of the shortest point-to-point path in the embedded graph vs. an ideal reference path (Section 5.3.3), and Action Efficiency is the ratio of the distances of the shortest path ending in a "pick" motion vs. the shortest path ending in any motion (Section 5.3.4). All of these paths are optimal for their particular start and end locations, so the Median Path Efficiency, for example, is the efficiency of the *optimal* path for a typical pair of start and end points. Accordingly, any scenario with poor Path Efficiency has a poor value for that metric in the optimal case, so a path at "90% Path Efficiency" means the *optimal* path for that particular start and end location was less efficient than the optimal paths for 90% of the other $(start, end)$ pairs chosen.

Results for the environment with no obstacles are excellent, suggesting that the process of discretization and analysis does not unduly affect the capabilities of the motion graph. Note, however, how Practical Local Maneuverability and Action Efficiency show mobility is slightly affected, largely by the walls surrounding the environment removing some path options.

The environment with randomly-placed obstacles is relatively open, but represents a much more realistic environment, with more interesting evaluation results. XZ Coverage is

still high (94.3%), but XZA Coverage is much lower (66.5%), reflecting congested regions which the character can only traverse along a single axis. Median Path Efficiency is still very close to the optimum, but the mean path is over 20% longer than the reference path, meaning that some regions of the environment are difficult for navigation and require very long paths. This is reflected more strongly in the sharply higher values for Practical Local Maneuverability and especially for Action Efficiency. The latter is especially interesting; even the median paths were much (85%) longer than the reference paths, suggesting that the motion graph requires a sizeable open region to efficiently set up and use motions such as "pick"s.

The performance of the default motion graph in the Baseline Environment – the most obstacle-dense of the three – is quite poor, especially with respect to having the character use specific actions in specific parts of the environment. The high XZ Coverage value (95.7%) indicates that the character can reach almost any point in the environment; however, the lower XZA Coverage value (60.6%) – reflecting restrictions on the character's facing at many points in the environment – indicates the character may have a restricted ability to maneuver at many points. We see that most point-to-point paths through the embedded graph are still relatively efficient (median was 11% longer than reference), although interactions with highly congested regions of the environment cause a significant number of exceptions (Figure 5.14).

By contrast, the mean time from any frame to the nearest "pick" action in the embedded graph is almost doubled (3.6s to 6.6s) from its already-high value in the original motion graph, and is substantially worse than the equivalent measurement in the relatively-cluttered Random Environment. Accordingly, we observe substantial difficulty in creating efficient point-to-point paths which end in a specific action, such as a "pick" motion; even typical paths (Figure 5.15) are about 160% longer than the ideal reference paths.

Of the discrete actions available in our motion graph (i.e., pick, duck, kick, punch), the range of areas in which each could be used varied widely (see Figure 5.16). In addition, a slight modification to the Baseline Environment radically changed the coverage pattern (see Figure 5.17).

Section 5.4.4 examines several potential approaches for improving the performance of

Environ	Coverage(%)		Local Maneuv		Path Efficiency		Action Efficiency	
	XZ	XZA	Theory	Prac	Mean	Median	Mean	Median
Empty	99.8	94.1	3.6s	4.4s	1.00	1.00	1.25	1.11
Random	94.3	66.5	3.6s	5.6s	1.21	1.03	1.93	1.85
Baseline	95.7	60.6	3.6s	6.6s	1.87	1.11	2.85	2.59

Table 5.2: Evaluation results for the three basic test scenarios. Capability steadily worsens as the environment become more congested with obstacles.

motion graphs in this environment. Section 5.4.5 explores how this evaluation approach scales with increasing size and complexity of scenarios, as well as the effects of our on-demand edge computation algorithm. Finally, specifying a set of metrics required making several assumptions, the validity of which are examined in Section 5.4.6.

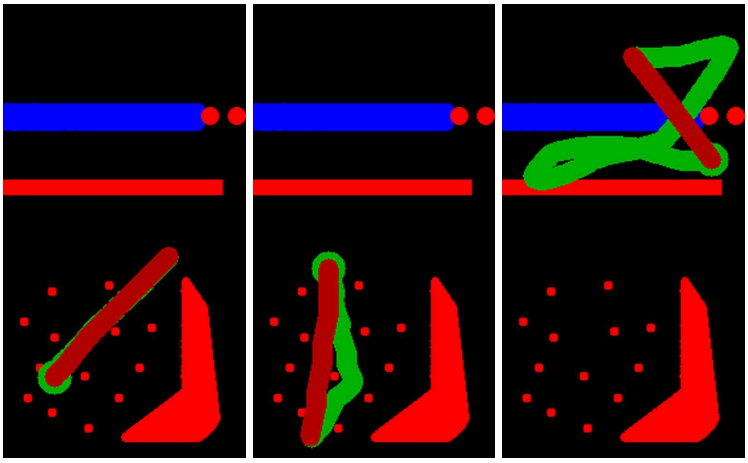
5.4.4 Improving Motion Graphs

We examine the effects of three common techniques which can be used to improve the flexibility of a character's performance when using a motion graph:

1. Adding more motion data
2. Allowing more motion editing
3. Increasing inter-action connectivity

Larger Motion Graphs

Table 5.3 shows the evaluation results for motion graphs of approximately 200%, 300%, and 400% the size (in terms of number of clips) of the baseline motion graph (first row). Larger motion graphs were created from the original dataset by allowing progressively lower-quality transitions to be included in the motion graph. Larger motion graphs allow more flexibility in creating motions, and improve the metrics across the board; however,

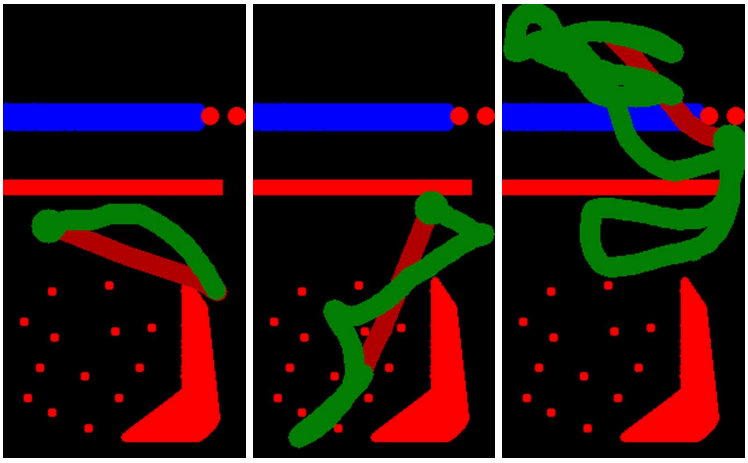


(a) Good path (10th percentile)

(b) Median path (50th percentile)

(c) Bad path (90th percentile)

Figure 5.14: Representative character navigation paths in the baseline scenario. Actual paths are in green, ideal reference paths are in dark red. The starting point of the actual path is drawn slightly widened.

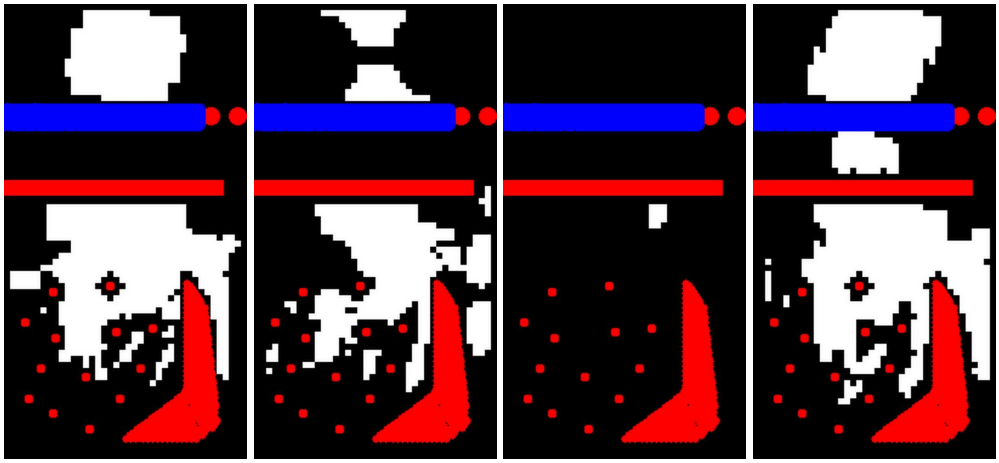


(a) Good path (10th percentile)

(b) Median path (50th percentile)

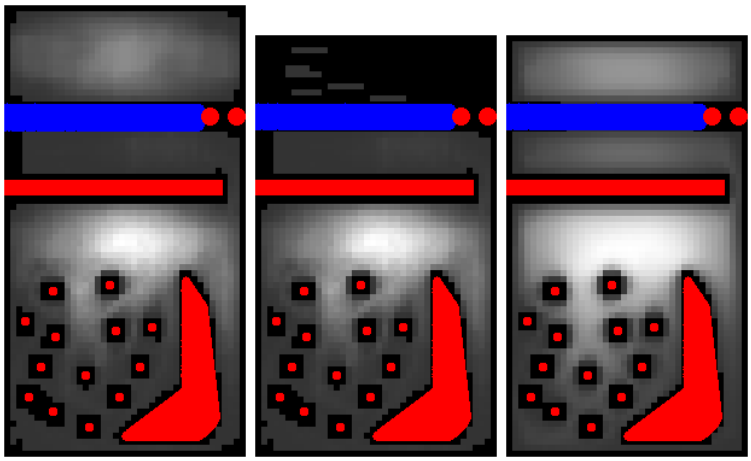
(c) Bad path (90th percentile)

Figure 5.15: Representative character pick paths in the baseline scenario. Paths ending in a pick action are in dark green and ideal reference paths are in dark red. The starting point of the actual path is drawn slightly widened.



(a) Duck coverage (b) Kick coverage (c) Punch coverage (d) Pick coverage

Figure 5.16: XZ Coverage locations for ducking, kicking, punching, and picking-up in the baseline scenario. Section 5.5.2 examines the poor coverage of the punching action.



(a) Baseline (b) Shortened (c) Larger motion graph

Figure 5.17: (a) XZ Coverage for the baseline scenario. (b) XZ Coverage for a version of the baseline scenario with the environment shortened from 15m to 14m. (c) XZ Coverage for the shortened environment with a larger motion graph.

	Coverage(%)		Local Maneuv		Path Efficiency		Action Efficiency	
Clips	**XZ**	**XZA**	**Theory**	**Prac**	**Mean**	**Median**	**Mean**	**Median**
98	95.7	60.6	3.6s	6.6s	1.87	1.11	2.85	2.59
190	98.3	90.3	3.3s	4.6s	1.11	1.01	2.36	2.10
282	98.3	91.5	3.6s	3.9s	1.06	1.01	1.47	1.37
389	98.3	95.5	2.6s	3.0s	1.02	1.00	1.20	1.11

Table 5.3: Evaluation results by size of motion graph.

no substantial improvement to the efficiency of paths ending in a "pick" action was noted until the size of the motion graph had tripled from the original, and adequate performance was not obtained until a motion graph of quadruple size was used.

In addition, even the motion graph of 200% baseline size allowed sufficient flexibility to navigate efficiently within the Baseline Environment, and to restore the capability lost in the shortened environment (see Figure 5.17).

Unfortunately, our primary method of generating larger motion graphs – by lowering the threshold for acceptable transitions when creating the motion graph – tended to lower the overall quality of motions generated from that motion graph, due to the larger amount of editing needed to smooth over the lower-quality transitions. In practice, some of the paths generated from the largest motion graph were very efficient and looked good at a global level, but contained poor-quality motion and looked poor at a more local level, suggesting that increased capability from lower transition thresholds should be balanced against this potential quality loss.

Restricted Motion Databases

We examine the character's ability to navigate in the secondary test environment (Figure 5.1) with the secondary test motion graph (23 walking motions with a stepping-over motion) for test motion sets with different types of motions removed from the secondary test database.

We begin by comparing different motion data sets. How much motion is really needed to allow the character to perform reasonably in the environment? To explore the value of having duplicate motions and different types of turns, we examined results from 5 different motion sets, starting from the original set of 23 motions containing straight walks, gradual turns, sharp turns, and stepping:

1. the original dataset: 23 motions
2. half of the motions of each type removed: 12 motions
3. one motion of each type retained: 6 motions
4. all sharp turns removed: 15 motions
5. all gradual turns removed: 15 motions

The number of frames in the resulting motion graphs is shown in Table 5.4. Note that the number of frames in these motion sets does not vary as much as might be expected. Because we enforce a minimum clip length of 0.333 seconds when forming the motion graph, doubling the number of motions does not always lead to doubling the number of frames in the motion graph. Through experimentation, we found that when enforcing a minimum clip length, short source motions (2–3s) tended to provide far fewer usable clips as compared to longer source motions (4–5s) than their relative sizes would suggest. The *Half Motions* and *No Duplicates* sets of source motions do not contain these shorter motions, accounting for their more efficient utilization of the available frames.

Table 5.4 also shows the differences in coverage obtained from different motion sets. Coverage was quite high for all examples, and only pruning the motion set all the way down to 6 clips had a sizable effect.

Table 5.5 shows the differences in Path Efficiency in the secondary test environment with different types of motions knocked out of the secondary test database. In all cases, the median motion is quite good. Motions with no gradual turns show the poorest performance, especially in the number of paths more than 25% longer than the minimum (E_P (Path Efficiency) > 1.25). Motions with no sharp turns produce a relatively large motion

Motion Set	Source Frames	MoGraph Frames	Coverage Fraction	
			XZ	XZA
Full Set	2304	1177	0.951	0.904
Half Motions	1424	1010	0.955	0.905
No Duplicates	733	537	0.922	0.879
No Sharp Turns	1551	948	0.942	0.893
No Gradual Turns	1543	682	0.958	0.896

Table 5.4: Frames of source motion, frame size of the motion graph, and coverage comparison for the different motion datasets. XZ coverage indicates the fraction of the collision-free ground plane that can be reached at some orientation. XZA coverage is the fraction of all collision-free 3D configurations (x, z, θ) that can be reached.

graph and good coverage (Table 5.4), but produce many poor paths due to the inability to take sharp corners. (This problem is especially apparent in the $E_P > 1.1$ column.) All of the motion sets tested have some combinations of start and end positions that produce poor results.

Increasing Allowed Editing

Table 5.6 shows the evaluation results for the baseline motion graph with varying assumptions about the maximum allowable amount of motion editing. The results show that a minimum level of editing is necessary to form a well-connected embedded graph. After that point, however, capability improves at a decreasing rate. Even the highest level of editing does not adequately resolve the problems with poor Practical Local Maneuverability and Action Efficiency, with transitioning from running to a "pick" action taking a mean of almost 5 seconds, and paths ending in a "pick" being over 60% longer than the ideal reference paths.

While allowing more motion editing permits greater coverage of the environment, in practice there are limits on the extent to which a motion can be edited while remaining of high enough quality to meet the requirements of the application. When editing size is

Motion Set	**Path Efficiency**		**% Poor Efficiency**	
	Median	95%	$E_P > 1.1$	$E_P > 1.25$
Full Set	1.0066	1.124	7.0	2.0
Half Motions	1.0063	1.131	6.6	2.8
No Duplicates	1.0083	1.140	8.2	2.2
No Sharp Turns	1.0066	1.133	9.0	2.2
No Gradual Turns	1.0082	1.180	8.8	3.8

Table 5.5: Path Efficiency comparison for the different motion datasets. The median column lists median path length as a fraction of the minimum path length. The 95% column lists 95th percentile values for E_P. The "$E_P > 1.1$" column lists the percent of paths tested having a path length ratio greater than 1.1, and the "$E_P > 1.25$" column lists the percent of paths with path length ratios greater than 1.25.

Edit	**Coverage(%)**		**Local Maneuv**		**Path Efficiency**		**Action Efficiency**	
Size	**XZ**	**XZA**	**Theory**	**Prac**	**Mean**	**Median**	**Mean**	**Median**
75%	40.5	17.2	3.6s	10.3s	2.67	2.67	5.12	5.39
88%	51.5	28.9	3.6s	9.0s	1.54	1.24	3.29	3.31
100%	95.7	60.6	3.6s	6.6s	1.87	1.11	2.85	2.59
112%	96.8	69.4	3.6s	5.9s	1.55	1.05	2.34	1.89
125%	97.2	75.9	3.6s	5.7s	1.25	1.02	1.98	1.48
150%	98.3	84.0	3.6s	4.9s	1.18	1.02	1.61	1.19

Table 5.6: Evaluation results for the baseline scenario with different sizes of editing footprint.

Editing Size	**Coverage Fraction(%)**			
	Always connect		**Respect edit bounds**	
	XZ	XZA	XZ	XZA
5	55.7	25.6	0.0	0.0
10	77.2	53.8	24.1	14.5
15	88.2	73.7	62.4	40.5
20	93.8	87.5	89.7	82.1
25	95.1	90.4	94.9	89.6
30	96.9	94.3	96.9	94.2
35	97.4	95.4	97.4	95.4

Table 5.7: Effect of allowable motion editing on coverage for a simple walking dataset in a small environment (Figure 5.1). Editing size is the value of r_x and r_z in $cm/meter$ and also the value of r_θ in $degrees/$m (see Equation 5.1; α was zero for this dataset, so clip duration was ignored). "Always connect" means to always connect to at least one grid point. "Respect edit bounds" means only connect to grid points within the editing footprint. Results depend on grid spacing, which was 20cm for x and z and 20 degrees for θ.

Motion Graph	Coverage(%)		Local Maneuv		Path Efficiency		Action Efficiency	
	XZ	XZA	Theory	Prac	Mean	Median	Mean	Median
Baseline	95.7	60.6	3.5s	6.6s	1.87	1.11	2.85	2.59
Tuned	93.0	40.0	2.9s	5.2s	1.51	1.17	2.48	2.39

Table 5.8: Evaluation results for the Baseline Environment with either the baseline motion graph or a smaller, hub-based one.

larger than 125% of baseline, we observe unacceptable footsliding artifacts in the resulting motion. Although our current estimate of a reasonable amount of motion editing is subjective, it is our hope that it will be possible to develop perceptually based algorithms to automatically compute such estimates.

Note that low levels of allowable editing with short motion clips can potentially result in clips whose editing footprints sometimes contain no grid points at all. While it is possible to increase the size of each footprint until it contains at least one grid point, doing so would compromise the conservative nature of the embedding (Section 5.2.7). As Table 5.7 shows, however, such a relaxation of the embedding algorithm's guarantees is not necessary in scenarios where the discretization size is reasonable compared to the amount of editing allowed, and in practice was never necessary in our test scenarios.

User-Modified Motion Graphs

Table 5.8 compares the baseline motion graph with a small but densely-connected hub-based motion graph which was carefully edited to have highly-interconnected motion types in a manner similar to that of Lau and Kuffner [2005].

While the hub-based motion graph has superior Local Maneuverability, its Path Efficiencies are only slightly better than that of the baseline motion graph. In general, the hub-based motion graph did not offer as much improvement as expected, some reasons for which are examined in Section 5.5.2.

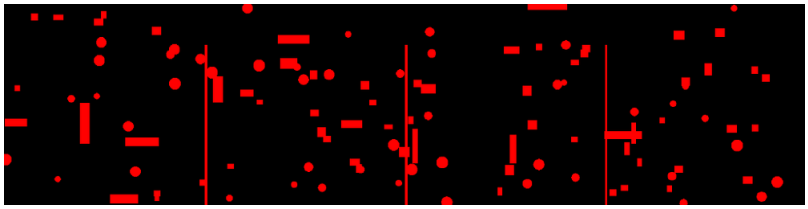


Figure 5.18: 40m by 10m environment partitioned into rooms by 8m-long walls. 10m by 10m, 20m by 10m, and 80m by 10m environments of the same format were also used to evaluate scaling behavior of the algorithm in equivalently-dense environments.

5.4.5 Scaling Behavior of the Methods

Realistic scenarios often include large environments with complex motion graphs, and a practical method for evaluating motion graphs must be able to scale to meet these demands. We examined the scaling behavior of the evaluation method in terms of increasing demands of several different types.

Scaling with Increasing Area

Table 5.9 shows the time and memory requirements for evaluating environments of different sizes. Embedding Time is the total computation time required to obtain the strongly connected component of the final embedded graph. Metrics time is the computation time required to compute all metrics. Base memory is the memory use reported by the Java Virtual Machine (JVM) after the graph embedding. Peak memory is the maximum memory use reported by the JVM; path search is the main contributor to the difference between peak and base memory, with some effect from dynamic edge caching. Each environment was populated with randomly-selected and randomly-placed obstacles in the manner of the Random Environment (Figure 5.13(b)) of a size appropriate to the environment (i.e., larger environments tended to have both more and larger obstacles). Graph embedding dominated the running time of larger environments, increasing from 28% of processing time for the smallest environment to 66% for the largest.

	Time(s)			Edges		Memory Used	
Area(m^2)	**Total**	**Embedding**	**Metrics**	**/sec**	**/**m^2	**Base**	**Peak**
56	612	163	470	3.5k	10k	182MB	204MB
120	1,769	742	1,010	5.8k	36k	182MB	283MB
240	4,709	2,137	2,550	8.2k	73k	183MB	382MB
1,000	24,337	15,667	8,653	8.3k	130k	190MB	361MB
5,000	417,075	274,052	142,955	10.2k	560k	222MB	542MB

Table 5.9: Time and memory requirements for evaluation of environments of different sizes. "Edges/sec" is the number of edges in the embedded graph divided by the total embedding time. For comparative purposes, an NBA basketball court is approximately $430m^2$, and an NFL football field is approximately $5,300m^2$.

Size(m)	**Embedding Time(s)**	**Time(s)** $/m^2$	**Edges**	
			/sec	**/**m^2
10x10	1,007	10.5	3.7k	38k
20x10	2,190	11.0	3.5k	39k
40x10	4,076	10.2	3.8k	39k
80x10	9,571	12.0	3.7k	44k

Table 5.10: Time to compute the embedded graph is approximately linear with area across environments with equivalent obstacle densities. (Additional "fake obstacles" (which had no effect on edge validity) were added to the smaller environments to equalize collision-detection overhead between environments.)

In this experiment, runtime scaled approximately quadratically with area, but actually decreased on a per-edge basis ("Edges/sec" column in Table 5.9). The apparent reason for this quadratic scaling with area is that the size of each obstacle increased with the size of the environment, making successive environments closer and closer approximations of an unconstrained environment; accordingly, the size of the embedded graph (and hence the computation time) increased faster than the size of the environment ("Edges/m^2" column).

To factor out the effects of different obstacle densities, we examined a set of environments designed to have approximately equal obstacle density (see Figure 5.18). All environments in the set were 10m long, with widths of 10m, 20m, 40m, and 80m. Each obstacle was partitioned into 10m by 10m sections by a series of narrow 8m-long walls (so movement between sections was possible only through the 2m gaps beside the walls); these partitions made the larger environments approximate a series of the smallest 10m by 10m environment, rather than having large regions away from the constraining influence of side walls. Each environment was populated with an equal density of random obstacles drawn from the same population (i.e., comparable sizes). Finally, each environment contained enough "fake" obstacles (requiring collision tests but not affecting edge validity) to bring the total obstacle count up to the number of obstacles present in the largest environment. These fake obstacles made per-edge collision detection costs comparable across the environments.

By contrast with the largely unconstrained environments seen previously, embedding time for these environments divided into "rooms" scaled close to linearly with area (Table 5.10). The partitioned environments maintain approximately the same embedded graph density at all sizes, and hence their computational requirements scaled linearly with area, as the theoretical analysis suggested. We note, however, that there was a small upwards trend in embedded graph density and per-m^2 embedding time (about 15% over an 8x size increase), suggesting that even the small 2m openings between adjacent rooms may have slightly changed the obstacle density of the environments. We conclude that different configurations of obstacles – such as long, thin walls partitioning the interior of a building vs. small, scattered bushes or tree trunks in a field – may have qualitatively different effects on the capability of an animated character.

Motion Graph			Time(s)			Memory Used	
Clips	**Edges**	**Motion**	**Total**	**Embedding**	**Metrics**	**Base**	**Peak**
98	288	87s	1,613	399	1,201	182MB	200MB
190	904	163s	10,616	1,602	9,001	352MB	594MB
282	1,712	248s	19,214	4,014	15,185	523MB	995MB
389	3,198	350s	29,519	8,204	21,303	741MB	1,246MB

Table 5.11: Time and memory requirements for embedding different sized motion graphs in the Baseline Environment.

Finally, we note that the embedded graph in the largest environment evaluated (100m by 50m) contained 122,368,278 nodes and 2,798,837,985 edges. Explicitly storing this embedded graph would have required approximately 11GB of memory, whereas computing it via the one-step algorithm would have required approximately 50GB, as compared to the 0.5GB used by the space-efficient method.

Scaling with Increasing Motion Graph Size

Table 5.11 shows the time and memory requirements for evaluating the Baseline Environment with motion graphs of various sizes. Larger motion graphs were formed from the same database of motions by progressively lowering the threshold for acceptable transitions. As with the baseline motion graph, per-file transition weights were manually adjusted to give reasonable and representative motion graphs.

Total running times ranged from under half an hour to over six hours, with metric evaluation taking approximately 75% of the overall time. Much of the heightened requirements were due to the path searches through the large motion graphs required for the Efficiency metrics, as A* search is exponential in the branching factor, which tends to be higher in larger motion graphs.

On-Demand Edge Computation Overhead

Table 5.12 shows the time and memory requirements for evaluating environments using different levels of edge caching. Default caching places edge-lists into a very simple hash table of fixed size, using usage-frequency information to determine which list to keep in the event of a collision. Full caching stores all edges prior to metric computation, so stored edges are included in base memory. Explicit caching computes all candidate edges and uses Depth-First Search to find the embedded graph as in Reitsma and Pollard [2004].

Explicit computation of the embedded graph rapidly becomes intractable due to the memory requirements. Moreover, the minimum possible time such an algorithm could take – conservatively estimated by adding together the time to find the edges, find the largest SCC (without augmentations), and compute the metrics with fully-cached edges, but ignoring any other necessary operations – is at best lower than the flood-based algorithm used in this paper by a small constant factor, due to the smaller number of nodes expanded by the flood-based algorithm (see Section 5.2.6).

For the flood-based algorithm, the three caching regimes tested offer a tradeoff between storage space and computation time. Caching all edges in the embedded graph before computing the metrics is only a small amount of memory overhead for small scenarios and results in almost a 50% reduction in computation time, but memory requirements become increasingly costly as environments or motion graphs grow larger, for increasingly smaller gains in computation time (only 30% for the larger environment). Caching frequently used edges (the default) provides a tradeoff that maintains most of the speed benefits for only a fraction of the memory.

5.4.6 Validity of the Evaluations

While the results of these metrics seem intuitively reasonable, one of the key goals of this work was to reduce the reliance on animator intuition by providing objective and verifiable evaluations.

Cache	Area(m^2)	Time(s)		Memory Used	
		Total	Metrics	Base	Peak
None	56	992	822	182MB	200MB
Default	56	612	437	182MB	204MB
Full	56	504	324	191MB	209MB
Explicit	56	499+	324	572MB	590MB
None	120	2,791	2,118	182MB	258MB
Default	120	1,736	1,068	182MB	289MB
Full	120	1,412	690	235MB	306MB
Explicit	120	1,181+	690	1,064MB	1,135MB
None	240	6,934	4,076	183MB	325MB
Default	240	5,496	2,639	183MB	382MB
Full	240	4,307	1,302	357MB	498MB
Explicit	240	3,192+	1,302	2,070MB	2,211MB
None	1,000	41,780	15,117	189MB	274MB
Default	1,000	30,027	9,870	189MB	361MB
Full	1,000	27,859	5,954	1,305MB	1,390MB
Explicit	1,000	13,747+	5,954	9,832MB	9,917MB

Table 5.12: Time and memory requirements for different edge-caching schemes. All times are in seconds. Explicit caching refers to the algorithm of Reitsma and Pollard 2004. The ”+” notation is used as we replicated only the first parts of their algorithm for comparative purposes; some additional computation beyond the amount timed is required by their algorithm.

Grid Cells per		Coverage(%)		Local	Path Efficiency		Action Efficiency	
Meter	2π **Radians**	**XZ**	**XZA**	**Maneuv**	**Mean**	**Median**	**Mean**	**Median**
4	15	47.1	25.0	7.5s	1.78	1.45	3.88	3.62
5	18	95.7	60.6	6.6s	1.87	1.11	2.85	2.59
6	20	96.0	65.0	6.7s	1.55	1.04	2.70	2.10
7	24	96.3	74.1	6.8s	1.59	1.02	2.35	1.80

Table 5.13: Evaluation results at different discretization levels.

Finer Discretization Resolution

Past a minimum threshold resolution, the metrics are relatively stable, with evaluation results steadily but slowly improving with increasingly fine resolution; however, the problems identified in the default resolution remain in the finer resolution analyses (see Table 5.13). This trend suggests that the baseline evaluation at which our initial tests were performed should be a reasonable (if pessimistic) estimate of the metric values that would be obtained in the limit as grid sizes went to zero. While drastic improvements in the metric values do not occur, such improvements are possible, making this approach a conservative estimate.

In addition, while Local Maneuverability results are only given for the running-to-pick-up action transition and Action Efficiency results are only given for paths ending in a picking-up motion, several other actions were tested (run-to-duck, sneak-to-duck, paths ending in a duck, etc.), with substantially similar results to those reported for picks. Similarly, the Path Efficiency and Action Efficiency metrics gave essentially the same results regardless of whether path-planning was unbounded or limited to a generous horizon. For efficiency, both metrics were run with a horizon of six times the length of the reference path; failure to find a path within that distance resulted in an effective path length equal to the horizon distance for metric purposes.

Sampling Density	Compute Time(s)	Local Maneuverability			
		Mean	Median	StdDev	StdErr
Baseline	6	6.6s	5.3s	4.7s	0.14s
0.03%	1	7.3s	6.2s	5.4s	1.33s
0.1%	1	8.5s	7.1s	6.1s	0.87s
0.3%	2	6.6s	5.3s	4.6s	0.36s
1.0%	3	7.0s	5.9s	4.8s	0.21s
3.0%	6	6.7s	5.3s	4.7s	0.12s
10.0%	13	6.7s	5.3s	4.9s	0.07s
30.0%	38	6.8s	5.3s	4.8s	0.03s
100.0%	122	6.7s	5.3s	4.8s	0.02s

Table 5.14: Practical Local Maneuverability results for the baseline environment when each possible sample in the environment has the given chance of being evaluated.

Effect of Metric Sampling Density

Practical Local Maneuverability was tested at sampling densities ranging from 100% (i.e., the true value) down to 0.03% (see Table 5.14). The computed values were highly stable past approximately 1% sampling density, validating our probabilistic sampling approach. The baseline at which our evaluations were performed was 10 samples per square meter of the environment, or about 2.3% sampling density for the Baseline Environment.

Path Efficiency and Action Efficiency metrics were run with between 50 and 500 samples, as compared to the baseline of 150 sample paths (Table 5.15). Metric results were stable across different numbers of samples, suggesting that the default number of samples evaluated should provide a reasonable estimate of the true metric values.

Other Aspects of Path Sampling

The paragraphs above discuss the effect of varying the sampling density for different metrics; however, Path Efficiency is never exhaustively evaluated. While such an exhaus-

Samples			Path Efficiency		Pick Efficiency	
Path	**Pick**	**Time(s)**	**Mean**	**Median**	**Mean**	**Median**
50	17	386	1.92	1.09	2.95	2.59
100	33	696	1.86	1.09	2.93	2.59
150	50	1,109	1.87	1.11	2.85	2.59
200	67	1,447	1.86	1.10	3.04	2.68
300	100	2,022	1.92	1.10	2.96	2.61
500	167	3,036	1.96	1.12	2.95	2.39

Table 5.15: Metric results for different numbers of path efficiency and pick efficiency tests run in the baseline environment. Baseline number was 150/50.

tive evaluation would make an excellent basis for validation of the Monte Carlo sampling method we use, we note that such evaluation would not be computationally feasible.

Consider, for example, the Baseline Environment detailed in Section 5.4.1. At the default discretization resolution, the environment has roughly 2,000 valid start location and a slightly lower number of valid end locations, resulting in approximately 3,000,000 possible $(start, end)$ pairs on which to run the Path Efficiency metric. At approximately 10 seconds per path evaluated, an exhaustive evaluation of those 3,000,000 pairs would require just under a year to compute.

While a stochastic sampling approach is necessary to evaluate metrics such as Path Efficiency, we note that sampling schemes other than the ones we chose are possible. For example, one could imagine using a constant sampling density per square metre of the environment (as is used to estimate Practical Local Maneuverability) to estimate Path Efficiency, rather than the constant number of samples per environment as was actually used. Such an approach might be more accurate for large and non-homogeneous environments, as more paths would be sampled more evenly over the environment; however, the observed stability of our current sampling method suggests there would not be a substantial change in metric values for our test environments.

A second type of alternative approach would be to evaluate the Path Efficiency metric

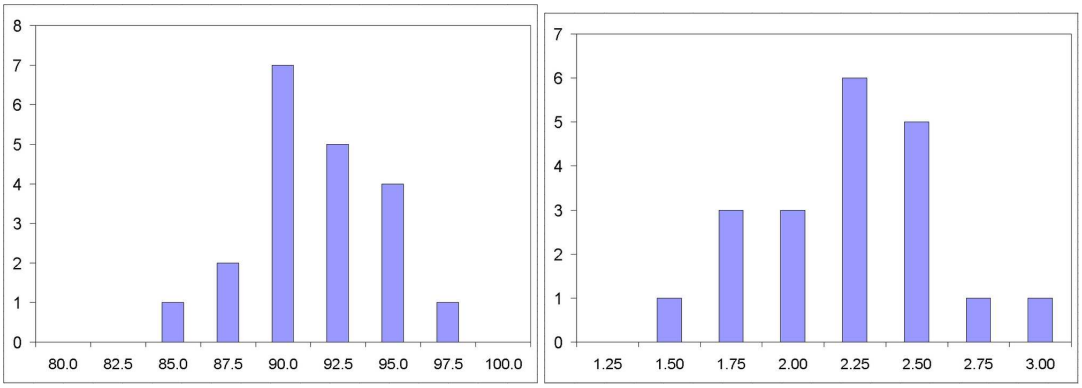
Total Times(s)		XZ Cvgs(%)		Prac LM		Path Efficiencies		Action Efficiencies	
Mean	**StdDev**	**Mean**	**StdDev**	**Mean**	**StdDev**	**Mean**	**StdDev**	**Mean**	**StdDev**
1,535	260	90.3	3.1	5.8s	0.55s	1.25	0.11	2.11	0.36

Table 5.16: Mean values and standard deviations for evaluation results of randomly-generated environments.

without discretization; i.e., simply unroll the motion graph from the selected start location until a path is found which comes within a pre-selected tolerance of the end location. As noted in Section 5.2.1, dispensing entirely with discretization is infeasible, due to the manner in which possible paths increase exponentially with distance. For example, evaluating one instance of Path Efficiency in a situation where the optimal path is at least 20m (which occurred several times in the Baseline Environment with the baseline motion graph) by using no discretization while unrolling a motion graph with mean branching factor of 3 and mean per-clip distance of 1m (similar to the baseline motion graph) would result in over 3 billion paths to be examined – even before considering the effects of motion editing or dead-end checking – which could pose difficulties for computational time as well as memory requirements. By contrast, finding and evaluating a 20m optimal path in the embedded graph could be done rapidly and efficiently, typically requiring the evaluation of a few hundred thousand path segments (equivalent to about 10,000-20,000 paths) and could be accomplished with minimal memory overhead in 10-30s. Moreover, as the embedded graph is required anyway for the computation of the different Coverage metrics, using it for evaluating Path or Pick Efficiency incurs no additional overhead.

Measurement Stability Across Different Environments

Table 5.16 gives the distributions of evaluation results for the baseline motion graph run in 20 randomly-generated environments. The area ($120m^2$) of the environments and types of obstacles placed were the same as those in the Random Environment used for initial testing (see Figure 5.13(b)), although configuration details (height/width of environment



(a) XZ Coverage

(b) Action Efficiency

Figure 5.19: Distributions of a representative pair of metric values for the random environments evaluated. Y axis is count of number of environments in each bin.

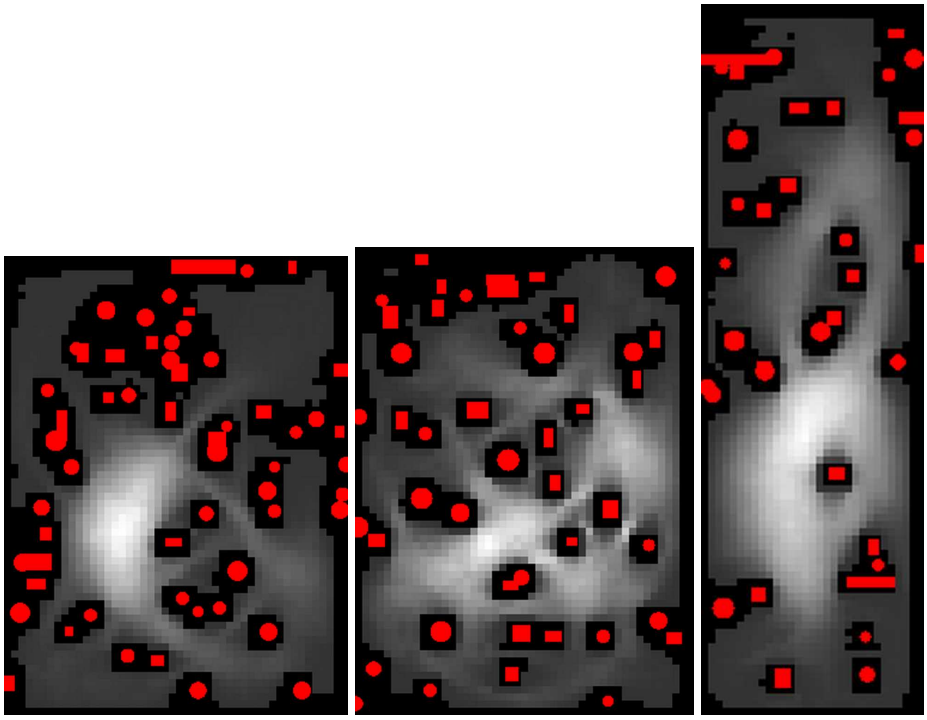
as well as number/size/shape/placement of obstacles) were determined randomly. Figure 5.20 shows the XZ Coverage overlaid on three representative environments.

In contrast to the drastic changes seen with the shortened Baseline Environment (Figure 5.17), results were relatively stable across different environment configurations, with no drastic changes in the capability of the motion graph. As Figure 5.19 shows, the distribution of values for metrics evaluated on the randomly-generated environments was approximately normally distributed. This result suggests motion graph capability will tend to be strongly correlated across many classes of similar environments.

5.5 Discussion

5.5.1 Findings

When using regular motion graphs on general environments, our experiments identified several problems.



(a) Difficult environment (b) Typical environment (c) Easy environment

Figure 5.20: Representative examples of the random environments created for testing.

Effects of Complexity

One fundamental result of our experiments is the observation that when working with a motion graph data structure, *easy tasks are easy; more complex tasks are hard*. For a simple task, such as basic locomotion in an obstacle-free environment, all motion graphs we tested performed well, with extremely good coverage and average paths under 1% longer than ideal reference paths.

Making either the task or the environment more complex, however, had a substantial effect on the capabilities of the motion graph. More complex tasks, such as transitioning to a particular action type or performing a particular action type at a designated location in the environment, were more difficult to perform in even the obstacle-free environment, with Local Maneuverability and Action Efficiency values both higher by about 20% (Table 5.2). Moreover, both measures degraded rapidly even in the relatively-open random environments, and especially in the highly-congested Baseline Environment. Similarly, even basic locomotion suffered in the relatively-open random environments, going from near-perfect efficiency to an average path length 25% longer than the ideal reference, and locomotion in the Baseline Environment was extremely inefficient, with average paths generated by the baseline motion graph being over 85% longer than the reference paths.

We note that one approach to handling this complexity has been pioneered by Lee and his colleagues [Lee et al., 2002, 2006]). Briefly, their approach is to collect substantial quantities of motion data interacting with each obstacle or cluster of obstacles in close proximity in the environment, in a sense forming a specialized motion graph for each unique region in the environment. As this technique relies on re-using motions captured for the specific parameters of each object cluster, however, it is unclear how the technique might apply to more general environments.

While generic motion graphs are effective for simple animations in simple environments, we note that the ability to deal with increased complexity of both task and environment is necessary to make motion graphs a more general tool. In particular, simple tasks in environments with few or no obstacles may not provide an adequate test of a motion generation algorithm's abilities.

Reactivity

Even the Theoretical Local Maneuverability of a typical motion graph is quite poor, and embedding the motion graph in even a relatively open environment degrades this measure of reactivity substantially (Table 5.2). For example, changing from a running motion to an evasive action (ducking) in the baseline motion graph took an average of 3.6s even in theory, increasing to an average of 5.8s in the random environments (Table 5.16) and 6.6s in the Baseline Environment. Considering that interactive environments such as games and training scenarios will tend to operate at timescales driven by users' recognition and choice reaction times (approximately 0.5s, rising to 1s for unexpected stimuli; see Green [2000a,b], Kosinski [2005]), substantial improvements in the ability of motion graphs to react to interactive control are necessary.

Currently, this is often handled by immediately transitioning to the target behavior, regardless of the detrimental effects on motion quality. While we would like to avoid this drastic measure when using motion graphs, the unfortunate fact is that most motion graphs give the character insufficient ability to react quickly, as evidenced by their poor Theoretical Local Maneuverability and even worse Practical Local Maneuverability scores. Ikemoto et al. [2006] examine one possible approach to this problem.

Embedding

We note the importance of embedding a motion graph into its target environment in order to evaluate its capability. Our experiments with variants of the Baseline Environment (Figure 5.17) show that minor changes to an environment can potentially cause drastic changes in the effective capability of one motion graph – in this case making the upper portion of the environment largely inaccessible – while causing virtually no changes in the effective capability of another motion graph. However, our experiments with randomly-generated environments (Table 5.16) demonstrate that there is a strong correlation between the capability of a motion graph in environments with similar types and densities of small obstacles. The capability of a motion graph in one of our randomly-generated environments was distributed approximately normally (Figure 5.19). Based on the results with

the Shortened Baseline Environment and the set of partitioned environments, however, (Figure 5.17 and Table 5.10, respectively) it appears that smaller numbers of longer or wider obstacles will lead to less predictable capability than the larger numbers of smaller obstacles used in the randomly-generated environments.

5.5.2 Identified Causes

Our experiments suggested three main problems with the motion graphs tested.

First, many clips, especially those of actions such as picking-up, ducking, or jumping, were effectively "set pieces" – i.e., they were linked into the motion graph in such a way that substantial amounts of specific motion was unavoidable both before and after the action itself. Both instances of punching actions, for example, required about *eight meters* of mostly straight-line movement to perform, resulting in the almost complete inability to perform punches in the Baseline Environment (Figure 5.16(c)). By contrast, other actions were embedded in linear paths through the motion graph consisting of about 2-4m of movement, which was much easier to place within the environment (Figure 5.16). Lengthy "set pieces" of motion such as these are not only easily disrupted by obstacles, reducing the coverage of the action in the environment, but also drastically worsen Local Maneuverability.

Similarly, actions of these sorts tend to have only a few instances in the motion graph in order to keep the overall size of the graph down as the variety of actions it contains increases. Unfortunately, the paths corresponding to the instances of these actions are often poorly linked into the motion graph, with the start of the branchless portion of the path often being accessible from only two other clips in the motion graph. In a congested environment, however, the ability to take any particular transition from the character's current location can easily be blocked by obstacles, meaning that the small number of direct paths through the motion graph to the desired action can easily be rendered unavailable. This makes the character's Local Maneuverability unacceptably fragile and variable, as well as contributing to the inefficiency of paths.

Finally, many obstacles in close proximity, such as the lower-left region of the Baseline

Environment (Figure 5.13(c)), create a need for paths with specific patterns of left-and-right turning in order to wend between the obstacles. Motion graphs, due to their heavy reliance on the source data, have trouble substantially changing the curvature patterns of their constituent paths, although some of this is possible during clip transitions.

We examined three common approaches to resolving these problems: adding more data to the basic motion graph; allowing more extensive motion editing at clip transitions; and using a small but highly-connected, hub-based motion graph.

Generality from Motion Graph Size

We note that in our experiments the most effective method for improving the capability of a motion graph was simply to add more data, increasing the number of nodes while keeping the density of the graph roughly constant. The apparent reason for this improvement was the way the increased variety of actions allowed increased flexibility in navigating constrained areas (i.e., higher or lower curvature of turns allows threading between obstacles), as well as the larger number of target nodes in the motion graph for paths constrained to end in a specific actions. Both of these factors made it more likely that an appropriate clip of motion would be available for any given point in the environment.

In particular, this had an especially large effect on the ability of the motion graph to efficiently reach specific actions. For the largest motion graph, Practical Local Maneuverability improved sharply to only about 15% higher than Theoretical Local Maneuverability, and Action Efficiency for "pick" actions was only 20% above the ideal (Table 5.3). Reducing the effect of "set piece" actions is only part of this improvement, however, as the shortest of the available "pick" action sets was still 3.0m, vs. 3.7m in the baseline motion graph.

This was also, however, one of the most computationally expensive methods, especially for tasks using the motion graph, such as path-planning. Accordingly, to scale to very large motion databases will require either separating out distinct behaviors as suggested by Kovar et al. [2002a] or pursuing an approach that involves clustering and/or multiple levels of resolution, as in the work of Arikan et al. [2003] or Lau and Kuffner

Figure 5.21: An environment which could be hard to navigate with semantically-invariant editing methods.

[2005].

Generality from Motion Editing

Increasing the amount of motion editing permitted had a significant effect on the capability of the motion graph, but did not resolve the problems identified, especially for tasks requiring specific actions: even when allowing 50% more motion editing – a level at which we observed substantial visible artifacts in the animations – Practical Local Maneuverability for "pick" actions was still nearly five seconds, with average paths required to end in a "pick" action being over 60% longer than the ideal reference (Table 5.6).

We note that our motion editing model assumes editing is largely *semantically-invariant*; i.e., edits are conducted purely at a local level, using incremental changes to produce acceptable variations of a source motion. Editing approaches of this type, while commonly used, have limited flexibility. Consider Figure 5.21; only source motions with the correct number of turns in nearly the correct places can easily navigate this environment when only local edits are permitted.

By contrast, editing techniques which have some notion of what a *natural path* is for human movement allow much more flexibility in the set of valid paths that can be generated from an input motion. Global edits at the level of a whole motion path rather than at the local level of a few frames of motion would allow much more sophisticated interaction between environments and motion paths, potentially allowing adequate capability to be obtained with substantially smaller datasets. In addition, such an explicit model of

what is or is not natural at a larger, path-level scale would complement lower-level naturalness metrics such as those of Reitsma and Pollard [2003] and Ren et al. [2005], and would help prevent locally-natural but globally-unnatural paths that semantically-invariant editing techniques can create. Some promising work in this direction has been done by Warren and his colleagues [Warren et al., 2001, Fajen and Warren, 2003] in the cognitive science community, and by Vieilledent, Bertoz, and their colleagues [S. et al., 2001, H. et al., 2005] in the neuroscience community.

Generality from Motion Graph Connectivity

We note that the densely-connected, hub-based motion graph we used has slightly better performance than the larger automatically-generated motion graph, but that the difference is surprisingly small; both produced poor paths in the baseline and random environments, with poor Local Maneuverability.

One reason for the smaller-than-expected improvement is that the lengths of the available pieces of motion are not altered by this increased connectivity, and the character must often travel a significant distance before returning to a hub pose and becoming able to transition to a new clip of motion. For example, performing a "pick" action required playing approximately 3.7m of motion in the baseline motion graph before a choice of which clip to transition to next was available, vs. 3.5m in the hub-based motion graph. Due to this, the immediate-horizon ability of the character to navigate through a congested environment is not greatly improved.

Accordingly, a highly-connected, hub-based motion graph which does not address the problem of long "set piece" chunks of motion does not appear to adequately address the deficiencies observed.

One potential solution would be to increase the degree of connectivity not only at the ends of clips, but *inside* clips as well. This approach would allow the character to move between actions such as locomotion with different radius of curvature more freely, permitting a greater ability to navigate in highly-congested environments. Unfortunately, this approach would substantially increase the size and density of the motion graph. The result-

ing graph would be significantly harder to reason about, either intuitively or analytically, and would be much more expensive to use for operations such as path planning. Both of these drawbacks undermine some of the important benefits of motion graphs, suggesting the importance of a careful and well-informed tradeoff.

5.5.3 Scaling Behavior

One-Step vs. Space-Efficient Embedding

In our experiments, the backwards footprint (f' in Equation 5.6) covered more nodes than the (forwards) editing footprint by about a factor of three, suggesting the one-step embedding algorithm takes approximately one-fourth the time that the space-efficient algorithm takes to embed a node. In practice, however, the one-step algorithm expands every node, whereas the space-efficient algorithm expands only those reachable from the seed node. Since a typical size for a strongly connected component is 10-20% of the total number of nodes in an environment, the flood-based algorithm can be competitive or even faster than the one-step algorithm in some cases, since it will usually expand significantly fewer nodes. In typical environments, the runtime of the algorithms will differ by at most a small constant factor.

This big-Theta time equality also holds for evaluating the metrics. The reason for this is that running large numbers of probabilistically-sampled tests (such as searching for minimum paths between two randomly-chosen points) typically requires a search through the graph, in which case time is spent for each edge of each node processed (for example, putting the distance of a path through that edge into a heap), so the overhead to compute those edges on the fly is only a constant factor. In our tests, a search on the embedded graph took approximately three times as long when edges were computed on the fly as opposed to being retrieved from a lookup table.

In practice, other computational aspects of the evaluation process, such as the A* search to find the minimal reference path for Path Efficiency, tend to reduce the overall difference between the methods; experimentally, the slowdown for using the flood-based

algorithm is approximately a factor of two. A tradeoff also exists between storage space and computation time. Caching the edge lists of frequently-accessed nodes can also provide a significant speedup, with the speed of this on-the-fly algorithm approaching the speed of the one-step algorithm as the amount of memory allocated to caching approaches the amount required to store all edges of the graph in the cache (see Section 5.4.5).

Memory

As noted in the theoretical analysis of memory requirements (Section 5.2.5), base memory required increases linearly with the size of the target environment Only the storage required for computing the embedded graph – about four bits per node – increases linearly, but is not a significant limitation (with the default motion graph and discretization, 1GB can store the information necessary to compute the embedded graph for about $50,000m^2$, or ten football fields).

Base memory is overwhelmingly due to the caching of pre-rotated motion clips, and increases linearly with the amount of motion data in the motion graph; an optimized implementation could reduce the base memory requirement by a significant constant factor.[5]

For comparative purposes, and considering only memory to store the embedded graph, the one-step algorithm can store at most an embedding consisting of approximately ten million nodes (for example, the 7m by 8m environment in Reitsma and Pollard [2004] examined at discretization 10cm by 10cm by $\frac{\pi}{15}$ radians by 64 clips, or the 20m by 12m environment in Section 5.4.5 examined at discretization 20cm by 20cm by $\frac{\pi}{9}$ radians by 98 clips). For the flood-based algorithm, either of these environments requires less than 7MB of memory to store the embedded graph. Storing the embedded graph data for a 10m by 10m environment discretized at 2cm by 2cm by 5 degrees by 100 clips would require 1GB for the flood-based algorithm, but approximately 5 terabytes of memory for the one-step

[5]Our implementation includes substantial display-related data structures; initial experiments suggest that a computation-only implementation could reduce the memory required to store a clip by at least a factor of five, making even our largest motion graph comfortably fit within 200MB. Without display-related data structures, our largest motion graph (about 6 minutes of motion) at our highest resolution (12cm by 12cm by 12 degrees) with a basketball court environment ($430m^2$) would require about 350MB of base memory.

algorithm.

Peak memory requirements are driven by the A* search technique used to evaluate the ability of the motion graph to produce efficient paths in the target environment. Due to the exponential nature of A*, this value is highly sensitive to parameters such as the planning horizon, and in practice memory limitations can be significantly lessened by appropriate parameter settings and heuristics. Moreover, applications using motion graphs will already have a path-planning method in place to plan motions created from the motion graph, and that method will be the most appropriate one to use when evaluating scenarios for that application. As such, more detail on the peak memory requirements can be found in the path planning and search references.

Computation Time

For larger and more complex scenarios, computation time is typically a stronger concern than memory requirements. For larger environments, computation time is increasingly dominated by the time required to embed the graph in the environment (increasing from under 30% for the smallest environment to over 60% for the largest). In practice, larger environments are often qualitatively different from smaller environments (see Section 5.4.5), and their greater openness leads to denser embedded graphs. The embedded graph is asymptotically linear in the size of the environment, although in practice it tends to be sublinear for smaller environments, due to the mobility-restricting effect of the walls around the environment.

By contrast, larger motion graphs continue to have the large part of their runtime (around 70%) come from path search. As with memory requirements, computational requirements for path search and path planning have been a subject of much research, and whatever method is intended for use in the application employing the motion graph can be used efficiently in the embedded graph.

5.5.4 Improving Runtime

Two options we have examined for reducing the runtime requirements of the evaluation process are *parallelization* and *tiling*.

Parallelization

We note that all major steps in the evaluation process are highly parallelizable, allowing a tradeoff between computation time and required hardware.

All aspects of the analysis consist of many disjoint computations (and hence are immediately parallelizable) other than the flooding algorithm used to mark the base and augmented strongly connected components (representing 75% of the time required for embedding). This algorithm, however, can be parallelized simply by splitting up the set of frontier nodes to expand in each pass. Since the result of expanding a node is setting the "active" bit of that node to 0 and status bits of the node's unvisited children to 1, results from disjoint computation can be merged simply by doing an AND operation or operating into shared memory. Accordingly, the analysis should parallelize extremely efficiently, potentially reducing computation times to a matter of hours even for very large and complex scenarios.

Tiling

A second option for reducing runtime is to divide the environment into segments, either examining only those segments required (such as in the case of localized changes to obstacles) or forming the environment out of a number of repeatable tiles (see Figure 5.22), such as the work of Lee et al. [2006].

With the first form of segmentation, the environment is simply broken up into overlapping chunks, each of which is analyzed as if it were a self-contained environment. This is possible with the current system; however, it only provides an approximation of the true SCC within the region, since paths which exit the region and then return are not captured. To minimize this error, each chunk consists of a central core surrounded by buffer zones.

These buffer zones capture the effect of paths which leave the central core and later return; hence, the depth of the buffer zone allows a tradeoff between computational cost and accuracy of estimation. Also, we note that paths which stray too far from the central core will generally be undesirable paths for a character to take anyway. Computing the values of the metric in this central core provides an accurate estimate of the values of the metrics for that region as computed in the full environment; accordingly, tiling the environment with such central cores and combining the results allows the evaluation of environments of arbitrary size.

5.5.5 Validity

We note that metric results typically improve broadly with increasing resolution, but that the identified problems with the capabilities of the baseline motion graph in the Baseline Environment largely remain. This suggests that not only can a lower-resolution analysis be used to find problems that will be present in the continuous analysis, will tend to do so *conservatively*; i.e., since any path in a discretized analysis must also be present in the continuous analysis, a motion graph with acceptable capability in any discretized analysis is guaranteed to have acceptable capability in the continuous analysis. (Note that this is a probabilistic guarantee, due to the use of Monte Carlo methods in the metrics.)

One benefit of this one-sided error is that it opens up the possibility of automated quality-assurance of a motion graph. Once the tasks and requirements for a target scenario have been specified, motion graphs can be created and vetted without human intervention, with a passing motion graph being guaranteed to (with high probability) offer acceptable capability within the target environment. Such an analysis and guarantee will remove much of the risk of deploying motion graphs in demanding, risk-averse animation applications such as interactive games and training scenarios, since it will be known that an acceptable motion always exists.

Some care must still be taken while using the motion graph in the non-discretized environment; an acceptable discretized analysis guarantees an expectation of efficient paths between any two *grid cells*, and not necessarily between any two points; however, any

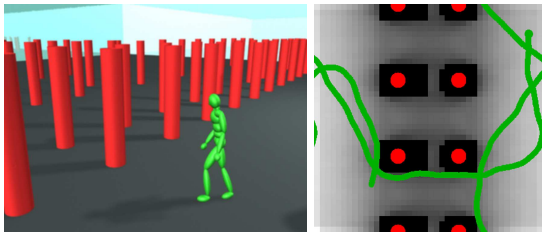


Figure 5.22: (Left) The character is following a random path through a motion graph that has been embedded into an environment tiled with a repeating pattern. (Right) A random path wraps around from left to right, bottom to top, then right to left.

point covered by the discretized analysis is always within a grid cell width – and within the clip's acceptable editing distance – of a point which was a grid cell center, and hence is covered by the guarantee.

5.6 Conclusions

The results presented in this chapter, along with the earlier version in Reitsma and Pollard [2004], are to our knowledge the first attempts to evaluate global properties of a motion graph data structure such as ability to efficiently reach all points in an environment and ability to quickly respond to user control in a complex environment.

Embedding a motion graph into the environment and assessing global properties of this data structure allows us to compare motion datasets and identify weak points in these motion datasets and problematic areas in the environments.

We also provide a method for determining a reasonable answer to the standard question of "how much motion is enough?" Our analysis techniques, coupled with task-based metrics representing the application at hand, can be used to evaluate the capabilities of a motion graph, allowing one to be selected that is neither too big for efficient searching and planning nor too small for effective task performance.

Our experiments highlighted several important considerations regarding generating an-

imations with motion graphs.

First, capability degrades surprisingly rapidly with increasing complexity. Tasks more complex than basic navigation and environments with obstacles in close proximity sharply narrow the options available when path-planning with a motion graph. Accordingly, testing an animation system with only simple tasks or relatively unconstrained environments may not provide the full picture of the system's abilities.

Additionally, reactivity for characters animated by motion graphs is often poor even in theory, and is easily degraded further by complex tasks or environments. The ability to react rapidly to user control, however, is crucial to interactive applications, and an inability to move efficiently between motion types lessens the capability of motion graphs with regards to more complex tasks.

Furthermore, while several approaches can help alleviate these problems (e.g., denser motion graphs (perhaps with shorter clips or edges transitioning from the inside of clips), path-level editing, and adding more data to motion graphs), these approaches tend to offer performance tradeoffs, however, increasing capability while increasing the computational cost of using a motion graph.

Our hope is that analysis methods such as the one in this chapter will be useful in managing tradeoffs such as these. In addition, such methods potentially allow a motion graph to be certified as sufficient for a particular environment and set of tasks, or even certified as suitable for a class of environments (e.g., environments whose obstacles do not cluster more tightly than a certain threshold). Additionally, a conservative estimation of capability may allow deemed-acceptable motion graphs to be used with confidence in their target environments.

In summary, the techniques shown here provide a way to evaluate a character's capabilities that is more sound than the trial and error approach commonly used. Evaluation techniques such as these can help a user to compare alternative data structures and provide detailed information on performance to aid the user in refining their motion graph or motion generation algorithm.

Chapter 6

Future Work

Dynamic obstacles are an important part of many scenarios (Section 6.1). An application we are eager to explore is using our evaluations in a feedback loop for automatic optimization of motion graphs or motion capture sessions (Section 6.2). A deeper understanding of how graph-theoretic properties of embedded graphs correspond to their evaluation results could allow the use of machinery from graph theory to improve our ability to evaluate and synthesize effective motion graphs (Section 6.3). Finally, in Section 6.4 we end with some speculative, long-range goals which motivate our future directions for this research.

6.1 Dynamic Task Domains

The algorithms presented here are appropriate for the portion of the task domain that is static. Moving obstacles are more complicated, as some notion of time is required to fully take into account their effect on the capabilities of the motion graph in the environment.

We have determined how to extend this approach to environments containing simple combinations of *reactive obstacles* or *cyclic obstacles*. Reactive obstacles allow events to be keyed to the behaviour of characters traversing the environment, and are often used to build tension or to present a hazard which must be responded to rapidly. Cyclic obstacles, by contrast, allow the environment to be dynamic in ways unconnected to the character's

behaviour, and are often used to make environments feel more “alive”, as well as to present a hazard which must be responded to carefully. These types of obstacles significantly extend the range of annotations usable in the scenario to be evaluated, allowing evaluation of scenarios containing many of the moving obstacles used in applications such as interactive games.

The augmented SCCs defined in Section 5.2.9 can be extended to take into account the passage of time since triggering a reactive obstacle. It can also be shown that, for embedded graphs encountered in practice, a cyclic obstacle which can *ever* be navigated can *always* be navigated, regardless of where in its cycle the obstacle was initialized; accordingly, the evaluation of environments with cyclic obstacles can be accomplished using the same type of augmented SCCs as reactive obstacles. More details can be found in Appendix A.

6.2 Optimizing Motion Graphs

A logical next step, now that we have a method for evaluating motion graphs, is to use that information to automatically optimize motion graphs. The result would be motion graphs that not only use the available source motions more effectively to improve both the quality and capability of the animations they can generate, but also have a known level of quality, and hence can be relied upon in a wider range of applications.

We envision three main requirements for automatically creating high-quality motion graphs:

1. Optimizing motion quality

2. Optimizing motion capability

3. Optimizing motion capture

6.2.1 Optimizing Motion Quality

To adequately optimize the motion graph for motion quality, it will be necessary to broaden the range of motions for which we have perceptually-derived guidelines; one possible approach to keeping the number of required studies down to a manageable number would be to examine the space of motions in terms of external forces and torques applied by the environment.

These improved metrics will replace the heuristic editing model currently used for evaluation, while also allowing an effective online motion filter for additional applications.

6.2.2 Optimizing Motion Capability

We are considering several approaches for this optimizing process, including a bottom-up construction which progressively adds the motions or connectivity which most improve the motion graph or a top-down approach which culls away the least important motions or connectivity. We theorize that the former may be well-suited to making small, efficient motion graphs, whereas perhaps the latter is more suited to motion graphs where high capability and motion quality is more important. More detail on these approaches is presented in Appendix B.

6.2.3 Optimizing Motion Capture

An adequate motion graph cannot be made if adequate source motions are not available. If the system cannot simultaneously meet the requirements of the user in terms of quality, capability, and efficiency, it should be able to calculate the motions which would most improve the motion graph if they were available. Provided the runtime of the system was fast enough, these requested motions could be used to drive a motion capture shoot, resulting in an iterative process whereby the system requests a motion, a similar motion is captured, and that motion is added to the database and evaluated, resulting in either an adequate motion graph or a new motion to capture. This process would have the benefit

of capturing a set of motions which efficiently spanned the space of required actions, potentially resulting in highly efficient motion graphs.

6.3 Graph Property Measurement

Based on our initial motion graph evaluation work, our intuition suggests that embedded graphs which embody good capability in a scenario will tend to share some similar graph properties. Our Local Maneuverability measure examines this to some extent, but we are interested in applying the machinery of graph theory to further understand the graph properties which indicate good capability, and potentially using that to derive and synthesize graph properties which indicate motion graphs which will result in good capability when embedded.

A general and more task-independent method of evaluation such as measuring the second eigenvalue of the embedded graph (Fiedler's Eigenvalue [Fiedler, 1973], a general measure of connectivity) is only useful if the method truly does correspond to our other, more task-specific measurements of interest. To determine the extent to which measuring such global graph properties accurately evaluates the capabilities of a motion graph or embedded graph, we will examine the correlation between those properties and the motion graph's score on other, more task-dependent, measures, as well as the predictive power of the general graph properties. If those general properties effectively encapsulate the results of the direct tests of required tasks, we can have confidence that our general measures are effective.

Graph-theoretic properties also offer promise for task-specific metrics. For example, a slight modification on the Max Flow/Min Cut algorithm could be used to evaluate the degree of redundancy present in the connectivity of the embedded graph. Areas of the embedded graph grid with low flow capacity will be regions of the environment in which the character is restricted to a small number of possible paths between the source and sink regions of the environment, regardless of the local number of actions available. This metric would offer information on the variability of the motion graph's coverage of the task

space – i.e., the degree to which a few motion paths will be relied upon, possibly to the extent of making the generated motions look too repetitive – and would also give valuable information regarding the robustness of the motion graph's coverage in the environment. In general, we theorize that a motion graph whose embedded graph has a large min cut between regions of interest value will tend to be relatively densely connected, and hence will be robust to small changes in the configuration of the environment (e.g., changes in the shape or position of the obstacles); by contrast, an embedded graph with a small min cut between important regions is likely to be much more fragile; the connectivity between the two regions is due to a small number of motion paths, suggesting that even small changes to the environment could have large effects on the capability of the motion graph, similar to that seen with the Shortened Environment (Section 5.4.6). Computing the static probability distribution over the embedded graph (using some assumption for probability distribution over edge choices, such as a uniform random walk) could provide a different view on this information.

6.4 Motivating Goals

Taking a broad view of the area of evaluation of human animation, we see several highly-speculative but enticing possible applications as motivation for our continuing research.

In the long term, we are excited by the potential for understanding human motion that animation offers. Psychophysical studies can take advantage of the unique opportunity offered by animation to carefully manage and control all aspects of the motions seen by subjects. While substantial work remains to be done uncovering how different rendering styles and characters will influence the results, we are already aware of psychologists and cognitive scientists who are keenly interested in these possibilities.

An additional possibility springing from a greater understanding of animated human motion is the ability to synthesize stylized *non-realistic* motion. Japanese anime and Chinese martial arts movies are two well-known examples of stylized non-realistic motion that is nevertheless internally consistent. The ability to deviate from realism in well-understood

and reliable ways may open up a wealth of artistic opportunities, from recreating well-known styles without the need for laborious hand-drawing of animations to a whole new range of styles to explore.

One very interesting application of an understanding of the overall capability of a humanoid character is to apply that understanding back to the real world and use it to inform the design of *usable spaces*. Applying this type of analysis to the capabilities of people with movement restrictions – whether due to age or injury – could potentially be especially valuable. Not only could the evaluation detect simple problems, such as the inability of a person in a wheelchair to make the tight turns required to utilize the designed wheelchair ramp – a process that now relies on a human poring over the designs and asking the right questions – the evaluation could also assess the design for response time (in high-performance work stations), for ergonomics (in repetitive factory positions), or for simple comfort and ease of use (in a retirement community). As the developed world undergoes an unprecedented demographic shift towards an older population over the coming decades, design of spaces that will allow elderly residents to maximally care for their own needs may be particularly appreciated and important in our future.

Chapter 7

Conclusions

Our overall research approach is to measure and evaluate important and under-explored areas of animation, and to derive practical results from the data obtained. The key research problems addressed by our work are summarized below, and followed by a brief listing of our contributions:

1. How does one measure the subjective quality of a motion clip?
    - How can those measurements be used to derive practical animation guidelines?
    - Do differences in character type affect a user's sensitivity to errors in that motion?
2. How does one measure the capability of a motion graph to fulfill a scenario's requirements?
    - What is a functional and actionable definition of capability?
    - How can this evaluation be efficiently applied to the large scenarios practical use demands?
    - What specific deficiencies in capability can be identified from the resulting measurements?
    - How well do common techniques resolve these deficiencies?

7.1 Measuring Motion Quality

Given a target set of motions, an environment in which they will be used, and a target category of perceptual errors in the motions, we showed how to measure the sensitivity of viewers to those errors. Using this approach, viewer sensitivity can be determined for a wide variety of scenarios.

We have demonstrated a technique for using this information to derive practical guidelines for animators to take into account. Given a set of measurements of user sensitivity to errors in animated human motion, we showed how to derive acceptability thresholds on measurable properties of those errors.

Our experiments uncovered a variety of interesting perceptual effects, including the systematic differences between types of errors (e.g., acceleration vs. deceleration) and between types of characters (e.g., gravity errors are more noticeable in a realistic human than in a simple cannonball).

7.2 Measuring Motion Graph Capability

Our first contribution with regard to motion graph capability was simply to show that not only is it an entity which can be sensibly defined, it can be measured and evaluated to obtain useful results.

Given a motion graph and a model of how motion editing can be applied to the results of that graph, we showed how to evaluate global properties of the motion graph, such as task-space coverage and local maneuverability. Our algorithms are sufficiently efficient to be used on large enough scenarios to be practical.

Our experiments demonstrate how to use quantitative measurements to gain insight into the particular deficiencies present in a motion graph intended for use in a given scenario, and we demonstrate how common, brute-force techniques for improving motion graphs address these deficiencies partially or inefficiently, leaving substantial room for well-informed improvement. In addition, our evaluation is conservative, allowing ade-

quate motion graphs – if the method deems them acceptable – to be used with substantially more confidence than was available previously.

7.3 Ensemble

Together, these techniques provide new tools for understanding animation algorithms and data structures, and also understanding the perceptual issues related to them. It is our hope that the new insights and applications produced by continuing research on animation, perception, and human motion will offer substantial benefits, both in terms of pure knowledge and in terms of applied solutions.

Appendix A

Dynamic Obstacles

We have determined how to extend our evaluation approach to environments containing simple combinations of reactive obstacles or cyclic obstacles. These types of obstacles significantly extend the range of annotations usable in the scenario to be evaluated, including annotations such as:

- Obstacle can be traversed by any motion sequence in which the character is ducking during the period $[2.2s, ..., 2, 9s]$ after entering the obstacle (pressure-plate-activated wall blade).

- Obstacle can be traversed by any motion sequence in which the character is ducking during the $0.4s$ danger phase of a three-second cycle (swinging bladed pendulum with 3-second period).

While a large portion of environments found in games and similar applications fall into this category, accommodating truly general moving obstacles is an area of future work. We are interested in examining whether techniques related to *occupancy grids* ([Elfes, 1989]) might help provide a way to expand the range of dynamic environments our system can evaluate.

A.1 Reactive Obstacles

Reactive obstacles—such as a blade which scythes out of the wall when the character steps on a pressure plate, a laser which blasts down a corridor when a motion sensor is tripped, or a crushing block which trembles and smashes down when someone passes under it—are common in action games and can play an important role in other applications, such as dangerous-environment simulators. Their defining characteristic is that the environment reacts to the character's movements, typically triggering a hazard shortly after the character reaches a certain location (such as a pressure-sensitive floor region), and can only be avoided by taking quick evasive action of a few types (such as diving out of the way or ducking under the attack). Once the triggered threat has occurred, the danger is over until the character leaves the triggering region, at which point the hazard may reset. These interactive hazards add excitement and interactivity to an environment.

One way to augment the basic system described above to include these obstacles is to compute augmented SCCs for evasive actions, in the same manner as is done for selective actions (see Section 5.2.9). Given a region R annotated to allow only paths which are doing action D during time period $[t_1, ..., t_2]$ after entering the region, we can compute the augmented SCC for D:

- Compute a maximal SCC M_D, which is computed as per the regular SCC but with locomotion and D permitted within R.
- Keep track of the edges of M_D which enter R; search forward from these edges through M_D, throwing away nodes inside R which are not reachable via a valid path (i.e., a path which exits R before t_1 or which performs D for exactly one interval which must contain $[t_1, ..., t_2]$). The result is an intermediate graph I_D which may not be strongly connected.
- The largest SCC in I_D is S_D, the augmented SCC for D, and takes into account the character's ability to traverse region R.

Note that I_D can be computed efficiently from M_D only if the region R is not too large, since path computation is more complex than the reachability computation used to

determine SCCs. In that case, S_D can also be computed efficiently from I_D, since the difference outside R is likely to be minimal.

A.2 Cyclic Obstacles

Cyclic obstacles—such as a floor which electrifies for 2s after being off for 2s, a spiked log which swings across the corridor in a pendulum motion, or a series of crushing blocks which smash down in different but linked cycles—are perhaps the most common type of obstacle in adventure games. Like reactive obstacles, they add excitement and interest to an environment, but with a stronger element of planning, as a player carefully times their run at the obstacle.

Due to this planning, a path traverses a cyclic obstacle if it successfully traverses the obstacle at any point in its cycle (i.e., a path that works when starting at the beginning of the obstacle's cycle may well not work when started 1s into the cycle, and vice versa). Accordingly, cyclic obstacles are processed as for reactive obstacles (see above), but with their more-general path-selection criteria.

Note that this assumes any entry point to the cyclic obstacle region can be accessed by the character at any point in the cycle. We note that this is true if the cycles in the base SCC form a *generating set* for the obstacle's cycle; i.e., the available cycles can be repeated zero or more times in combination to allow any edge entering the cyclic obstacle region to be traversed starting at any frame in the obstacle's cycle. For two cycles, the relevant objects are:

- A cyclic obstacle O, with period P (an integral number of frames).
- The first cycle, with period c_1.
- The second cycle, with period c_2.
- The motion sequence transitioning from the end of the first cycle to the beginning of the second cycle, with length l_1.

- The motion sequence transitioning from the end of the second cycle to the beginning of the first, with length l_2.

- The motion sequence transitioning from the end of the first cycle to the entry point to the cyclic region, with length l_3.

Length l_3 is irrelevant, since it is applied only once and hence simply relabels the cycle. Since for two integers A and B which are relatively prime, A is a generating element for the cyclic group of order B and vice versa, one way to be sure we have a generating set of cycles is to find a solution to $d = n_1c_1 + n_2(l_1 + l_2) + (n_2 + n_3)c_2$ with $n_i \in \mathcal{Z}$ and $gcd(d, P) = 1$. Given this, $\{n_1, n_2, n_3\}$ define set of traversals of the available cycles which will allow the entry point of O to be accessed at any point in O's period, since applying the composite cycle associated with d $1, 2, 3, ...P$ times is guaranteed to put the character at the entry point to O at a different point of O's cycle each time.

Of course, such a solution is not always possible; however, any significant portion of an SCC will tend in practice to have many cycles, making the lack of an appropriate pair vanishingly unlikely[1]. Perhaps the easiest way to ensure compliance with this assumption that we can enter the obstacle at any point in its cycle is to give obstacles cycles with a period which is a prime number of frames of motion; the visual difference between a 3s cycle and a 2.97s cycle (89 frames at 30fps) is minimal, but the existence of *any* cycle in the embedded graph reaching this obstacle's entry points with *any* period that is not an integral multiple of 89 guarantees our assumption is correct. The existence of such a cycle is virtually certain if any cycles exist.

Finally, we note that there are obstacle configurations where no cycles will exist; for example, if all access to an obstacle with period P_1 is blocked by an obstacle with period P_2 with no space in between the two obstacles, it is likely that no cycles of motion will exist in the space occupied by the second obstacle, foiling this approach. Such combinations of obstacles can be treated as a single composite obstacle (i.e., only paths which success-

[1] The probability of any two randomly-chosen integers being relatively prime to each other is $\frac{6}{\pi^2}$[Weisstein]; accordingly, in practice the presence of thousands of cycles in any reasonable embedded graph will virtually ensure an appropriate pair of cycles.

fully navigate both obstacles are considered), although each such type of obstacle must be implemented separately in our current system. Fortunately, the existence of idling-in-place cycles in most realistic datasets prevents such composite obstacles from being necessary too often, as any safe space between the two obstacles is likely to contain a generating set.

Appendix B

Motion Graph Optimization

Using our evaluation system itself as well as insight gained from it into what makes for a good motion graph, we will undertake to create a method for generating motion graphs that automatically fulfill many of the criteria for which we evaluate motion graphs. In particular, we intend to create motion graphs that have the property of local maneuverability, following our earlier hypothesis that this property will be a good predictor across a wide range of scenarios of motion graph appropriateness for that scenario.

The ultimate goal for this line of research is to understand how to automatically generate motion graphs which possess specified properties, such as guarantees of local maneuverability. Accomplishing this goal would make motion graphs significantly faster and easier to use; an animator would only need to specify a few requirements (such as local maneuverability and coverage over the given environment) before enough information was available to automatically construct a motion graph suited to the task, rather than engage in laborious manual crafting involving only a fraction of the available data. Making a few assumptions about user requirements in common scenarios would allow user requirements to be specified through a straightforward set of choices, allowing even a naive user to produce graphs well-suited to their needs. Allowing users to simply create motion graphs in this way would allow realistic, capable, interactive characters to be used much more widely than is now possible.

Several daunting challenges must be overcome before this goal can be achieved; in particular:

- Determine the requirements necessary for well-suited motion graphs
- Define these requirements in terms amenable to motion graph construction or optimization
- Create a method to automatically construct motion graphs fulfilling these requirements

B.1 Graph Requirements

What features of a motion graph are necessary in order to ensure it provides the motions needed for a particular application? While we expect that being able to evaluate the suitability of a motion graph for an application and environment will provide guidance for our examination of this question, our intuition suggests that local maneuverability—having very rapid access to the different types of motions—will be a very important characteristic of good motion graphs.

Our initial hypothesis is that a motion graph with local maneuverability will prove highly suitable for a wide variety of applications.

B.2 Construction Approaches

First, the set of actions available to the character needs to be identified. We intend to identify the actions in the original data set, either by hand or using automated techniques [Barbic et al., 2004] as a pre-processing step.

Maximizing this objective function (equation 5.14) via global search on the space of motion graphs is likely to be extremely computationally expensive, even with a good model of how much motion editing is permissible cutting down the number of possible

transitions. Accordingly, the search requires further restrictions and tactics, and we intend to look at several approaches:

1. Constrained optimization
2. Local search from a feasible solution
3. Divide-and-conquer then graph culling

B.2.1 Constrained Optimization

The requirements formulation specified in section 5.3.5 can be approached from a constrained optimization framework. The two main constraints we will place on the system are:

1. *Reuse*: constrain the number of times a frame in the original motion database can be used in a clip in the motion graph (usually to 1).
2. *Size*: constrain the overall number of frames, clips, or edges in the motion graph.

Initially, we intend to investigate a slight simplification of the requirements as an optimization problem:

- Given a set of *actions* $a \in A$
- Given initial motion database *DB* with actions in *A* marked
- Given yes/no frame-to-frame transition matrix based on motion editing model
- Given minimum clip length *L*
- Given nodes n in motion graph N
- Minimize $V = \sum_{n \in N} \sum_{a \in A} time(n, a, m)^2$

- Over the set of valid motion graphs $m \in M$ formable from *DB*

- Where *time(n,a,m)* is the time (in seconds) required to reach any instance of action label *a* from node *n* in motion graph *m*

The initial motion database is divided into a set *T* of potential transition points. Each of these transition points belongs to one or more *exclusion sets*, which are simply the list of transition points which are within the minimum clip distance of each other, and hence cannot both be included in the same valid motion graph. For example, if the minimum clip length was 10 frames, viable transitions at frames 20 and 25 would be in the exclusion set centered at frame 23's transition, but viable transitions at frame 33 and frame 13 would not be in that exclusion set. Selecting a set of transitions to include defines the motion graph defined from the motion database, and also defines the exclusion sets selected. Accordingly, the goal is to cover a subset of exclusion sets (i.e., ensure every possible transition is within at most one exclusion set of a selected transition) while performing the above minimization.

One simple approach to this problem is selecting transitions in a greedy manner. To do this, we select the still-valid (i.e. not in an exclusion set of any previously-selected transition) transition from the original motion database that most lowers the evaluation sum V. (For this purpose, we consider a result of "cannot reach action a" to have a large but finite cost.)

B.2.2 Local Search

Mutation of an existing viable solution is another possible approach. Local moves within the search space would consist of small changes in the set of transition points used to create the motion graph. For example, we could remove all transition points between frames i and j of the motion database from the motion graph, selecting new transition points to take their place. A second type of local move would be to relax the constraint of non-reuse of motion, allowing one or more frames of the original motion database to be reused; this type of move would typically put heavier reliance on the size constraint for the motion

graph.

Searches could be directed using standard techniques such as genetic algorithms or simulated annealing, using changes in the value of the evaluation sum V to determine the value of moves within the search space.

B.2.3 Divide and Cull

The idea with this approach is to build a large, overlapping motion graph by individually linking together actions, combining the result, and then pruning out unnecessary motions.

First For each pair of actions $a, b \in A$, connect each instance of a to any instance of b using the best combination of clips and transitions in the original motion database. This will form a large (possibly disconnected) motion graph.

Second Greedily add clips from the original motion database to turn this motion graph into a strongly connected component.

Third Iteratively prune the clip that adds the least to the evaluation sum V until the motion graph is within the reuse and size constraints specified.

B.3 Requirements Satisfaction Evaluation

While our goal is automatic and complete satisfaction of user requirements with the constructed motion graph, some runs of the algorithm may not result in a perfect motion graph. For many applications, especially those requiring online motion generation, it is crucial to have confidence in the quality of the motion graph before using it. Accordingly, we will use our embedding framework to evaluate the extent to which a constructed motion graph satisfies the user's requirements, providing both quantitative and qualitative information on the extent to which the user can rely on the motion graph for their application, as well as information on how to improve the motion graph should it not be sufficient.

Bibliography

O. Arikan and D. Forsyth. Interactive motion generation from examples. In *Proceedings of ACM SIGGRAPH 2002*, 2002. 3.2.1

O. Arikan, D. Forsyth, and J. O'Brien. Motion synthesis from annotations. In *Proceedings of ACM SIGGRAPH 2003*, 2003. 3.2.1, 5.5.2

Okan Arikan, David A. Forsyth, and James F. O'Brien. Pushing people around. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 59–66, New York, NY, USA, 2005. ACM Press. ISBN 1-7695-2270-X. 3.1.3

Jean-Pierre Aubin. A survey of viability theory. *Society for Industrial and Applied Mathematics Journal of Control and Optimization, 28(4), p.749-788*, 1990. 5.3.1

Jernej Barbic, Alla Safonova, Jia-Yu Pan, Christos Faloutsos, Jessica K. Hodings, and Nancy S. Pollard. Segmenting motion capture data into distinct behaviors. In *Proceedings of Graphics Interface 2004*, 2004. B.2

G. P. Bingham. Kinematic form and scaling: Further investigations on the visual perception of lifted weight. *Journal of Experimental Psychology: Human Perception and Performance*, 13(2):155–177, 1987. 3.1.1

Brooks, Fay, Heller, Moncror, Yeung, and Schell. *Project Biohazard*. http://www.etc.cmu.edu/projects/biohazard/spring03/BIOHAZARD2about.htm, 2003. 1.1

M. G. Choi, J. Lee, and S. Y. Shin. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics*, 22(2):182–203, 2003. 3.2.3

R. L. Cohen. Problems in motion perception. *Uppsals, Sweden: Lundequistska*, 1964. 3.1.2, 4.7

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001. 5.2.3

B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *Journal of the ACM*, 40(5):1048–1066, 1993. 3.2.3

M. J. Matarić E. Drumwright, O. C. Jenkins. Exemplar-based primitives for humanoid movement classification and control. In *Proc. IEEE Intl. Conference on Robotics and Automation*, 2004. 3.2.1

Alberto Elfes. *Occupancy grids: a probabilistic framework for robot perception and navigation*. PhD thesis, Carnegie-Mellon University, 1989. A

B. R. Fajen and W. H. Warren. The behavioral dynamics of steering, obstacle avoidance, and route selection. In *Journal of Experimental Psychology: Human Perception and Performance*, volume 29(2), pages 343–362, 2003. 5.5.2

P. Faloutsos, M. van de Panne, and D. Terzopoulos. Composable controllers for physics-based character animation. In *SIGGRAPH 01 Proceedings*, Annual Conference Series. ACM SIGGRAPH, ACM Press, August 2001. 3.2.1

A. C. Fang and N. S. Pollard. Efficient synthesis of physically valid human motion. In *ACM Transactions on Graphics 2, 3*, 2003. 3.2.1

M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Math. Journal 23 (1973), pp. 298–305*, 1973. 6.3

M. Gleicher. Motion editing with spacetime constraints. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pages 139–148, Providence, RI, April 1997. 3.2.1

Michael Gleicher, Hyon Joon Shin, Lucas Kovar, and Andrew Japsen. Snap-together motion: Assembling run-time animations. In *Proceedings of 2003 Symposium on Interactive 3D Graphics*, 2003. 3.2.2

Marc Green. Driver reaction time. *http://www.visualexpert.com/Resources/reactiontime.html*, 2000a. 5.5.1

Marc Green. How long does it take to stop? methodological analysis of driver perception-brake times. *Transportation Human Factors, 2(3), 195-216*, 2000b. 5.5.1

E. Grossman, M. Donnelly, R. Price, D. Pickens, V. Morgan, G. Neighbor, and R. Blake. Brain areas involved in perception of biological motion. *Journal of Cognitive Neuroscience, 12:5, pp. 711-720*, 2000. 3.1.1

Hicheur H., Vieilledent S., Richardson M.J.E., Flash T., and Berthoz A. Velocity and curvature in human locomotion along complex curved paths: A comparison with hand movements. In *Experimental Brain Research, vol 162, pp. 145-154*, number 162, pages 145–154, 2005. 5.5.2

Jason Harrison, Ronald A. Rensink, and Michiel van de Panne. Obscuring length changes during animated motion. In *ACM Transactions on Graphics 23, 3*, 2004. 3.1.3, 1, 5.1.2

H. Hecht and M. Bertamini. Understanding projectile acceleration. *Journal of Experimental Psychology: Human Perception and Performance*, 26(2):730–746, 2000. 3.1.2, 4.7

Rachel Heck and Michael Gleicher. Parametric motion graphs. *In proceedings of Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, 2006. 3.2.2

R.W. Jr. Hill, J. Gratch, S. Marsella, J. Rickel, W. Swartout, and D.R. Traum. Virtual humans in the mission rehearsal exercise system. In *KI special issue on Embodied Conversational Agents*, 2003. 1.1, 1.1.1

J. K. Hodgins, J. F. O'Brien, and J. Tumblin. Perception of human motion with different geometric models. *IEEE Transactions on Visualization and Computer Graphics*, 4(4): 307–316, October 1998. 3.1.3, 4.1, 4.7

D. Hsu, R. Kindel, J.C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. In *Int. J. of Robotics Research, 21(3):233-255*, 2002. 3.2.1

Leslie Kanani Michiko Ikemoto, Okan Arikan, and David Forsyth. Quick motion transitions with cached multi-way blends. Technical Report UCB/EECS-2006-14, EECS Department, University of California, Berkeley, February 13 2006. URL `http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-14.html`. 3.1.3, 3.2.2, 5.5.1

Thierry Simeon Julien Pettre Jean-Paul Laumond. A 2-stages locomotion planner for digital actors. In *Proceedings of ACM Symposium on Computer Animation 03*, 2003. 3.2.1

M. K. Kaiser, D. R. Proffitt, S. M. Whelan, and H. Hecht. Influence of animation on dynamical judgments. *Journal of Experimental Psychology: Human Perception and Performance*, 18(3):669–690, 1992. 3.1.2

M. Kalisiak and M. van de Panne. A grasp-based motion planning algorithm for character animation. *Journal of Visualization and Computer Animation*, 12(3):117–129, 2001. 3.2.1

L. E. Kavraki and J.-C. Latombe. Probabilistic roadmaps for robot path planning. In K. Gupta and A. del Pobil, editors, *Practical Motion Planning In Robotics: Current Approaches and Future Directions*, pages 33–53. John Wiley, 1998. 3.2.3

T. Kim, S. I. Park, and S. Y. Shin. Rhythmic-motion synthesis based on motion-beat analysis. In *ACM Transactions on Graphics 22, 3*, 2003. 3.2.1

Robert J. Kosinski. A literature review on reaction time. *http://biae.clemson.edu/bpc/bp/Lab/110/reaction.htm*, 2005. 5.5.1

L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *Proceedings of ACM SIGGRAPH 2002*, 2002a. 3.2.1, 5.5.2

L. Kovar, M. Gleicher, and J. Schreiner. Footskate cleanup for motion capture editing. In *Proceedings of ACM Symposium on Computer Animation 2002*, 2002b. 5.1.2

Lucas Kovar and Michael Gleicher. Automated extraction and parameterization of motions in large data sets. In *ACM Transactions on Graphics 23, 3*, 2004. 3.2.1

L. T. Kozlowski and J. E. Cutting. Recognizing the sex of a walker from a dynamic point-light display. *Perception & Psychophysics*, 21(6):575–580, 1977. 1.1.1

Taesoo Kwon and Sung Yong Shin. Motion modeling for on-line locomotion synthesis. In *Proceedings of ACM Symposium on Computer Animation 2005*, 2005. 3.2.1

J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991. 3.2.3

Manfred Lau and James Kuffner. Behavior planning for character animation. In *Proceedings of ACM Symposium on Computer Animation 2005*, 2005. 3.2.2, 5.4.4, 5.5.2

Manfred Lau and James J. Kuffner. Behavior planning for character animation. In *Proceedings of ACM Symposium on Computer Animation 2006*, 2006. 3.2.2

J. Lee, J. Chai, P. Reitsma, J. Hodgins, and N. Pollard. Interactive control of avatars animated with human motion data. In *Proceedings of ACM SIGGRAPH 2002*, 2002. 3.2.1, 3.2.3, 5.4.1, 5.4.2, 5.5.1

Kang Hoon Lee, Myung Geol Choi, and Jehee Lee. Motion patches: Building blocks for virtual environments annotated with motion data. In *Proceedings of SIGGRAPH 2006*, 2006. 3.2.1, 3.2.3, 5.5.1, 5.5.4

J. Lengyel, M. Reichert, B. R. Donald, and D. P. Greenberg. Real-time robot motion planning using rasterizing computer graphics hardware. *Proceedings of ACM SIGGRAPH 90*, 24(4):327–335, 1990. 5.3.1

Y. Li, T. Wang, and H.-Y. Shum. Motion texture: a two-level statistical model for character motion synthesis. In *Proceedings of SIGGRAPH 2002*, pages 465–472, 2002. 3.2.1

C. Karen Liu and Zoran Popović. Synthesis of complex dynamic character motion from simple animations. *ACM Transactions on Graphics*, 21(3):408–416, July 2002. 3.2.1

J. Lo and D. Metaxas. Recursive dynamics and optimal control techniques for human motion planning. In *Proceedings of Computer Animation '99*, pages 220–234, 1999. 3.2.1

T. Lozano-Pérez and P. A. O'Donnell. Parallel robot motion planning. In *Proc. IEEE Intl. Conference on Robotics and Automation*, 1991. 3.2.3

D. G. Luenberger. *Introduction to Dynamic Systems: Theory, Models, and Applications*. John Wiley & Sons, 1979. 5.3.5

N. A. Macmillan and C. D. Creelman. *Detection Theory: A User's Guide*. Cambridge University Press, New York, 1991. 4.5, 4.5

R. McDonnell, S. Dobbyn, S. Collins, and C. O'Sullivan. Perceptual evaluation of lod clothing for virtual humans. *Proceedings of Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, 2006. 3.1.3

C. F. Michaels and M. M. de Vries. Higher order and lower order variables in the visual perception of relative pulling force. *Journal of Experimental Psychology: Human Perception and Performance*, 24(2):526–546, 1998. 3.1.1

A. Michotte. The perception of causality. *Metheun, London*, 1963. 3.1.2, 4.7

M. Mizuguchi, J. Buchanan, and T. Calvert. Data driven motion transitions for interactive games. In *Short Presentation, Eurographics 2001*, 2001. 1.2.1, 3.2.2

Luis Molina-Tanco and Adrian Hilton. Realistic synthesis of novel human movements from a database of motion capture examples. In *In proceedings of IEEE Workshop on Human Motion 2000*, 2000. 3.2.1

Tomohiko Mukai and Shigeru Kuriyama. Geostatistical motion interpolation. *ACM Trans. Graph.*, 24(3):1062–1070, 2005. ISSN 0730-0301. 3.2.1, 5.1.2

S. Muthukrishnan and G. Pandurangan. The bin-covering technique for thresholding random geometric graph properties. In *Proceedings of SODA 2005*, 2005. 1

M. Oesker, H. Hecht, and B. Jung. Psychological evidence for unconscious processing of detail in real-time animation of multiple characters. *Journal of Visualization and Computer Animation*, 11:105–112, 2000. 3.1.3, 4.7

C. O'Sullivan and J. Dingliana. Collisions and perception. *ACM Transactions on Graphics*, 20(3):151–168, July 2001. 3.1.2, 4.7

Kevin A. Pelphrey, Teresa V. Mitchell, Martin J. McKeown, Jeremy Goldstein, Truett Allison, and Gregory McCarthy. Brain activity evoked by the perception of human walking: Controlling for meaningful coherent motion. *The Journal of Neuroscience, 23(17):6819-6825*, 2003. 3.1.1

Zoran Popović and Andrew P. Witkin. Physically based motion transformation. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 11–20, August 1999. 3.2.1

D. R. Proffitt. Naive physics. In R. Wilson and F. Keil, editors, *The MIT Encyclopedia of the Cognitive Sciences*, pages 577–579. MIT Press, 1999. 3.1.2

Paul S. A. Reitsma and Nancy S. Pollard. Perceptual metrics for character animation: sensitivity to errors in ballistic motion. In *ACM Transactions on Graphics 2*, *3*, 2003. 3.2.1, 1, 2, 5.1.2, 5.5.2

Paul S. A. Reitsma and Nancy S. Pollard. Evaluating motion graphs for character navigation. In *Proceedings of ACM Symposium on Computer Animation 2004*, 2004. 3, 5.4.5, 5.5.3, 5.6

Liu Ren, Alton Patrick, Alexei Efros, Jessica Hodgins, and James Rehg. A data-driven approach to quantifying natural human motion. *ACM Trans. Graph.*, 24(3), 2005. 3.1.3, 5.5.2

C. F. Rose, M. F. Cohen, and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, September/October: 32–40, 1998. 3.2.1

Charles F. Rose., Peter-Pike J. Sloan, and Michael F. Cohen. Animation: Artist-directed inverse-kinematics using radial basis function interpolation. In *Computer Graphics Forum 20, 3*, 2001. 3.2.1

S. Runeson and G. Frykholm. Visual perception of lifted weight. *Journal of Experimental Psychology: Human Perception and Performance*, 7(4):733–740, 1981. 3.1.1

Sverker Runeson. Constant velocity – not perceived as such. *Psychological Research, 37(1), pp. 3-23*, 1974. 3.1.2, 4.7

Vieilledent S., Kerlirzin Y., Dalbera S., and Berthoz A. Relationship between velocity and curvature of a human locomotor trajectory. In *Neuroscience Letters*, volume 305, pages 65–69, 2001. 5.5.2

A. Safonova, J. K. Hodgins, and N. S. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. In *ACM Transactions on Graphics 23, 3*, 2004. 3.2.1

Hyun Joon Shin and Hyun Seok Oh. Fat graphs: Constructing an interactive character with continuous controls. *In proceedings of Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, 2006. 3.2.2

Peter-Pike J. Sloan, Charles F. Rose, and Michael F. Cohen. Shape by example. In *Proceedings of the 2001 ACM Symposium on Interactive 3D Graphics*, 2001. 3.2.1

P. J. Stappers and P. E. Waller. Using the free fall of objects under gravity for visual depth estimation. *Bulletin of the Psychonomic Society*, 31(2):125–127, 1993. 3.1.2, 4.7

Jordan Strawn, Ronald A. metoyer, and Aaron Schnabel. Perceptual thresholds for foot slipping in animated characters. Technical report, Oregon State University, 2006. 4.7

Gita Suthankar, Michael Mandel, Katia Sycara, and Jessica K. Hodgins. Modeling physical capabilities of humanoid agents using motion capture. In *AAMAS 2004 Proceedings*, 2004. 3.2.3

J. Wang and B. Bodenheimer. An evaluation of a cost metric for selecting transitions between motion segments. In *Proceedings of ACM Symposium on Computer Animation 2003*, 2003. 3.1.3

J. Wang and B. Bodenheimer. Computing the duration of motion transitions: An empirical approach. In *Proceedings of ACM Symposium on Computer Animation 2004*, 2004. 3.1.3

W. Warren, B. Fajen, and D. Belcher. Behavioral dynamics of steering, obstacle avoidance, and route selection. In *Journal of Vision*, 2001. 5.5.2

B. Watson, A. Friedman, and A. McGaffey. Measuring and predicting visual fidelity. In *SIGGRAPH 01 Proceedings*, 2001. 4.7

Eric W. Weisstein. *Relatively Prime. From MathWorld–A Wolfram Web Resource*. http://mathworld.wolfram.com/RelativelyPrime.html. 1

D. J. Wiley and J. K. Hahn. Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics and Applications*, Nov/Dec:39–45, 1997. 3.2.1

Andrew Witkin and Michael Kass. Spacetime constraints. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, volume 22, pages 159–168, August 1988. 3.2.1

W. L. Wooten and J. K. Hodgins. Simulating leaping, tumbling, landing, and balancing humans. In *Proc. IEEE Intl. Conference on Robotics and Automation*, 2000. 3.2.1

K. Yamane, J. J. Kuffner, and J.K. Hodgins. Synthesizing animations of human manipulation tasks. In *ACM Transactions on Graphics 23, 3*, 2004. 3.2.1

V. B. Zordan and J. K. Hodgins. Motion capture-driven simulations that hit and react. In *Proceedings of ACM Symposium on Computer Animation 2002*, 2002. 3.2.1

Victor B. Zordan, Anna Majkowska, Bill Chiu, and Matthew Fast. Dynamic response for motion capture animation. *ACM Transaction on Graphics*, 24(3):697–701, 2005. 3.2.1