Motion Doodles

A Sketch-based Interface for Character Animation

by

Matthew E. Thorne

B.Math, The University of Waterloo, 2001

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Department of Computer Science)

We accept this thesis as conforming
to the required standard

. . . .

. . . .

THE UNIVERSITY OF BRITISH COLUMBIA

September 30, 2003

Abstract

This thesis presents a novel sketch-based interface for animating an articulated human character quickly and with a limited amount of training. The character is animated by sketching a sequence of simple arcs, loops and lines which are interpreted as an appropriate sequence of steps, jump, flips and other motions. The motion synthesis extracts key features from the sketch such as height, distance and timing information allowing a user to create a variety of styles of motion. The details of the sketch interpretation process are discussed for both a 2D and 3D version of the system.

Matthew Thorne.

mthorne@cs.ubc.ca

Contents

List of Tables

List of Figures

Acknowledgements

First, I would like to thank and acknowledge NSERC for providing the funding for this research for the past two years.

There are, however, others who also deserve acknowledgement of one form or another. For one, I would like to thank my supervisor, Michiel, for steering me onto this topic. It's been, overall, a fun project to work on and hasn't been overly frustrating, for the most part. And I still have too much fun playing with the animation system. Maybe I'm just easily amused.

I'd also like to thank my parents for their support and encouragement not just in this endeavour, but in many things over the years. They did *something* to make me value an education — I'm still not sure what it was, but regardless, I am grateful.

−− Matthew Thorne

Chapter 1

Introduction

Traditional hand-drawn animation and 3D computer animation is a challenging task requiring artistic talent and extensive training to create even simple animations. Further, when creating animation there are a myriad of styles and goals to be considered. Sometimes the goal is to create physically-realistic and natural motions, while at other times highly exaggerated, physically implausible or stylized motions are desired. Indeed, it is often necessary to exaggerate realistic motions to make them seem alive and to avoid a sense of stiffness and rigidity. Additionally, varying levels of control over the resulting animation is required depending on the application. Whatever the needs or goals, however, they all have one thing in common: animation is time consuming and challenging, limiting not only how animation is created, but also who is capable of animating.

This thesis presents a novel technique for creating and controlling animation using a sketch-based interface. Our sketching system, called "Motion Doodles", allows for "quick and dirty" animations and is designed to make animation accessible to people of all ages, skill levels and experience. Motion Doodles enables the user to draw and animate a 2D articulated figure in tens of seconds, without the need for the complex rigging and manipulations typical of standard computer animation packages. Increasing the speed at which one can create animations also potentially changes the workflow used in creating an animation. With the ability to rapidly create animation, it becomes much easier to experiment with different motions, their sequencing, and to experiment with the features of a motion such as the timing, height, or distance of a jump. While the principal system we describe is targetted at 2D animation, a prototype 3D animation system has been created which demonstrates the ability of sketch-based animation control to extend to more complex 3D animation.

The speed and ease of use of the system stems from the concept of motion sketching. A user, be they a novice or otherwise, creates a motion by using a stylus or mouse to sketch a sequence of "motion doodles" which simultaneously convey the sequence and attributes of motions in a desired animation sequence.

1.1 A Simple Example

A motion doodle is a simple curve such as an arc, loop or line which is interpreted by our system and used to synthesize a motion such as a walk, jump or flip.

Figure 1.1 shows a sample motion doodle and the resulting sequence of motions. Key features are extracted from each motion doodle gesture which partially specify the character's motion and serves as a basis for synthesizing the complete motion through the use of a keyframe database and inverse kinematics. While many extracted features are directly evident from the input sketch such as the length of a jump or the location of footfalls, the timing information associated with the sketch is not apparant in Figure 1.1, but it is an important feature of the input for our system. The speed at which the sketch is drawn directly affects the speed of the resulting motion. A quickly drawn scribble will produce an equally quick motion while a slowly drawn sketch results in an apparantly slow motion.

Motion doodles are drawn in one continuous stroke (the motion sketch). As illustrated in Figure 1.1, animating with motion doodles is a simple process. Motion doodles are easy to learn and require that the user be able to draw no

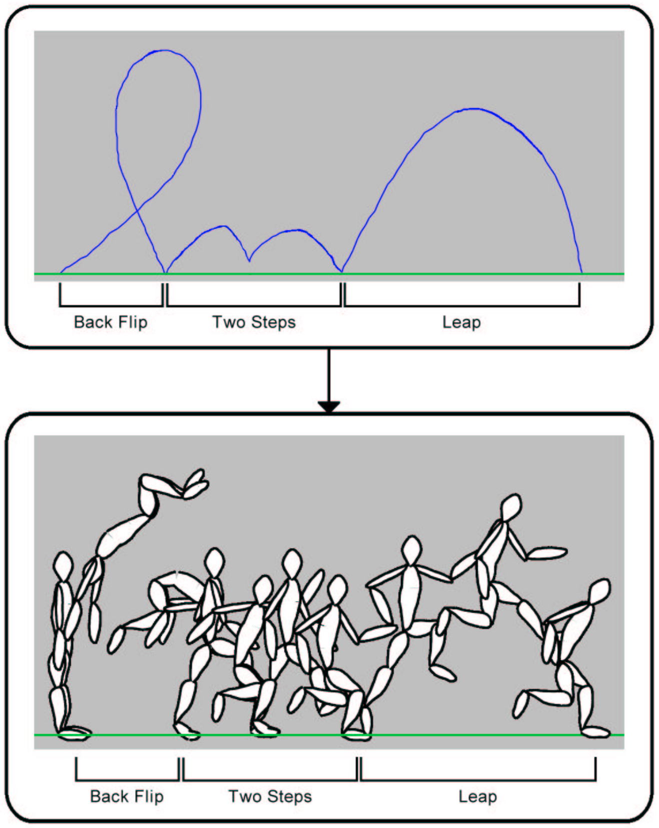

Figure 1.1: In our system a “motion doodle” (top, with parabolic trajectories fit to sketched motion also shown) is converted into a sequence of motions (bottom).

more than a sequence of relatively simple curves. The goal is to make animation accessible to virtually everyone, including children, unskilled novices and seasoned veterans who just want to experiment with an idea in a quick and straightforward manner.

Our animation system is not intended to replace established animation techniques but rather to address the particular niche of quick and easy-to-use prototyping tools.

1.2 A Gesture Vocabulary for Animation

Animations provide animators with the means to realize motions that initially exist only in the mind of the animator. Determining an appropriate vocabulary to describe motions has long been a challenge, one that has led to numerous motion annotation systems, such as Labanotation [Hut1987], among others. The vocabulary supported by the tool is an important design decision.

The first step involved in creating our sketch-based animation system was to design a "language" of gestures which represent different motions and, when strung together in an input sketch, are converted into an animation sequence. We have used three main considerations in designing the gestures used by our animation system.

Before discussing how our gestures have been designed it is necessary to be precise about what is meant by a "gesture". The term gesture refers to motions such as hand gestures as in [Hon2000] and has also been used to refer to simple glyphs drawn with an input device as in [Rub1991]. In our system, a gesture is defined as *the smallest unit of a sketch which can be translated into a motion* such as a single arc or loop. This is similar to the glyphs of [Rub1991] except that several gestures are typically drawn with a single continous motion of the input device.

1.2.1 Quick to Learn

The gestures representing each motion need to be chosen such that the correspondence between gesture and motion seems natural, allowing a novice user to quickly and easily learn the gestural vocabulary of the system. This means that a gesture should be "self-explanatory" and "evocative" of the desired motion. An arc, for example, naturally lends itself to a jumping or stepping motion since the arc itself suggests the trajectory of a jump in its flight phase or the path followed by a foot during a step. Determining appropriate mappings is, arguably, one of the most difficult aspects of designing a gesture-based animation system, especially as the vocabulary becomes richer.

For our animation system, one of the sources of inspiration for gestures has been the "motion lines" often found in hand drawings, as shown in Figure 1.2. For example, loops often represent tumbling, falling, spinning or some similar motion and arcs often represent the flight of objects through the air.

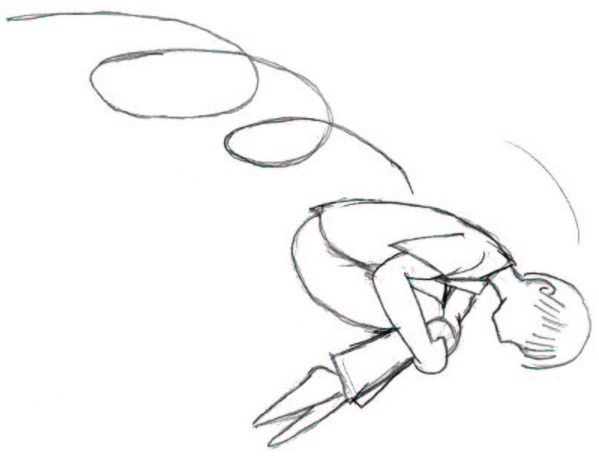

Figure 1.2: An example of "motion lines" illustrating a tumbling motion.

1.2.2 Simplicity

The second of our design goals presents two considerations. First, as the goals of our animation system are to empower novice users, including children to create and experiment with animation and to increase the speed at which animation may be created, we have chosen to keep each gesture as simple to draw as possible. Complex shapes and patterns are more difficult to learn and master (conflicting with our first design goal) and may also serve to discourage the less artistically inclined.

Second, requiring that gestures be as simple as possible presents a trade-off. Simple gestures limit the complexity of motions that can be represented and the number of degrees of freedom of a character that can be directly controlled. In our system, a gesture provides a partial specification of a motion, such as the approximate trajectory during the flight of a jump or the location of foot contact with the ground. A keyframe database then serves to fill in the remaining details. Databases of motion capture data could also be used. If more complexity or a greater degree of control is desired, the most common actions should be kept as simple as possible while less common actions could be assigned to more complex gestures.

1.2.3 Content

The last step in designing the gestures for our system is to determine the specific details of how a gesture is mapped to a motion. There are two types of information that can be encoded in a gesture: literal information such as height, distance, or timing and semantic information such as the number of loops in a flip or other stylistic information.

The way in which a gesture's literal information is used depends on the desired style of animation. In our system, we want to allow for exaggerated cartoon-like actions such as leaping a tall building in a single bound, thus the literal information extracted from a gesture directly specifies the exact height, distance, timing and foot placement associated with a motion. This has the added advantage of allowing the user to sketch directly within the scene in which the animated character exists, thereby providing immediate feedback on how motions will interact with the world. If more realistic, physically-based motions were desired then the interpretation of the gesture's literal information would likely need to be relaxed.

Semantic information is much more flexible (and more subjective). The main examples of how this type of information is encoded in our system are the number of spins in a flip and the mapping from gestures to specific types of motion. Flips are specified by loop gestures and the number of loops strung together specify the number of spins that should be included in a flip, but do not directly influence the height or distance of a flip. Similarly, stylistic information is extracted from arcs making up a walking motion to allow tiptoeing and stomping in addition to a regular walk, but these stylistic modifications are encoded in such a way that they do not impact on the literal information extracted from the gestures.

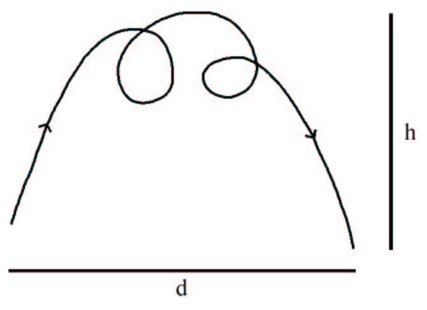

Figure 1.3: Literal vs. Semantic gesture content.

An example of a double loop gesture, representing a double forward flip, is shown in Figure 1.3. The length, d, and maximum height, h, of the gesture directly specify the height and length of the flip action. The two loops in the gesture indicate that two forward flips should be performed while in the air.

1.3 Organization

The organization of the remainder of this thesis is as follows. Previous and related work is discussed in Chapter 2. An overview of the system architecture is provided in Chapter 3. The details of transforming an input sketch to a sequence of motions are described in Chapter 4. The 3D extensions to the system are detailed in Chapter 5. The results for both the 2D and 3D systems are shown in Chapter 6. We conclude with a summary of benefits, limitations and possible future directions in Chapter 7. Finally, Appendix A provides additional details on the implementation of the various motion controllers.

Chapter 2

Previous Work

Animation has a wide range of applications including film, games, simulation, choreography and many others. Each of these different applications have differing requirements in terms of animator involvement and hence the controls presented to the end user. Games, for example, typically trigger a pre-scripted animation sequence in response to buttons pressed by the player. On the other hand, production animation for films requires a fine level of control over all aspects of the animation by the animator but provides no control to the audience. Table 2.1 summarizes and compares some common animation and motion planning techniques with our system.

The tools used for offline animation production typically use successive stages to allow for various levels of detail. This often begins with storyboards and animatics[1] and eventually proceeds to keyframes for the detailed design of the desired final animation. Going from the early animatic-based concepts to well designed animated motions is still a significant task that would benefit from a system allowing the most relevant features of a desired motion to be used as input with remaining details being filled in automatically.

There have been two broad classes of techniques used to approach this problem. The first class consists of algorithmic approaches involving either simulations or trajectory optimizations using a mixture of hard and soft constraints, in-

[1] *Animatics* are still frames or rough animations containing key poses used to help determine camera positions and timing of the animation

cluding those imposed by physics. The second class of approaches uses databases of motion capture data or other pre-existing motion data by modifying, interpolating or stitching together segments of motions. In our system we utilize an underlying set of keyframes and so it is related to the second class of techniques. However, we focus specificly on providing interactive control to the user over the properties and sequence of resulting animation. In the remainder of the chapter, techniques in graphics and animation that are most closely related to our system are examined.

	Control	Realism	Interactivity
Storyboards and animatics	Sparse overview of desired animation	Low; concerned with global positioning, timing and camera positioning	Not real-time
Keyframing	Detailed, low-level	As desired	Not real-time
Performance Animation, Motion Capture	Directly control DOFs	High	Real-time, may require offline post-processing
Dance choreography, Labanotation	Manual specification of motion	Not directly applicable	Time consuming
Spacetime constraints, Optimization	High-level control by specifying constraints	Variable	Not real-time
Physics-based control and simulation	User may control some DOFs	Variable	Real-time
Games	Button presses trigger prescripted animation	Varies from game to game	Real-time
Motion Doodles	Sketch-based interface provides high-level control	cartoon-like	Real-time, with delay

Table 2.1: A comparison of animation and motion planning techniques.

2.1 Sketch-based Modelling

Sketch-based interfaces have been used for a variety of geometric modeling systems starting with the SketchPad system [Sut1963]. More recent sketch-based modeling systems include SKETCH [Zel1996] which allows for 3D modelling using simple gestures or sequences of gestures to create, place and manipulate geometric primitives; and Teddy [Iga1999] in which a user draws freehand curves, combined with simple gestures for special modeling operations, to create 3D models which resemble stuffed animals. While none of these systems is directly related to our animation sketching system, they do nevertheless serve as inspiration and show the potential that sketch-based interfaces provide for quick-and-dirty design. Teddy especially shows the potential these types of interfaces have for making normally complex tasks accessible to novice users, including children.

2.2 Performance Animation and Animation Sketching

The idea of sketching an animation can be thought of as a type of performance animation or puppetry in which an animator's gestures are transformed into the actions of a character [Stu1998]. Mappings from the performer to the character range from direct joint-to-joint mappings such as is standard in motion capture to more abstract mappings such as using a dataglove in conjunction with a phoneme synthesizer [Fel1995].

Other research in performance animation and digital puppetry has explored the use of more abstract mappings between user input and output animation. [Oor2002] used two 6-DOF tracking devices to provide interactive, bimanual control over stepping and walking motions of a character. The work of [Las2000] provides a series of interfaces for controlling characters within a dynamical simulation using the mouse and keyboard for input. [Don2003] presents a system for character animation which uses motion-capture to map an animator's actions onto an animated character using several passes, or layers. Our system also belongs to this more abstract end of the spectrum in that it maps high-level information extracted from a 2D input sketch onto an 18-DOF character.

There exist some published techniques for using sketching to directly specify animated motions. The work of [Bal1995] investigates the sketching of 3D animation paths of a single 3D point, which are then automatically fit by spline curves. Several systems allow sketching of a desired walking path for a character, from which the specification of foot location and timing of foot placements can be estimated and then used to create animation [Gir1987, van1997]. Work has also been done in sketching the desired trajectory for a single rigid body and then using optimization techniques to find the physically-based motion that best fits the sketched trajectory [Pop2001].

2.3 Sketch and Gesture Recognition

There are two main areas of research in gesture recognition. One body of work aims at supporting 2D sketching applications by taking drawn input and either interpreting shapes drawn as gestures or geometric primitives. The second body of work looks at gesture recognition for hand and arm movements as captured using video. It is the first, sketch-based class of work which which we are concerned and on which we focus our discussion.

Any system utilizing a sketch or gesture-based interface needs to be able to analyze and make some sense of the sketch drawn by the user. The work of [Rub1991] presents a trainable gesture recognizer that treats each individual input stroke as a single gesture, which is different from the requirements of our system, in which an entire sketch is itself a single stroke which typically consists of several gestures strung together. More recently, [Sez2001] presents an algorithm for finding corner points in a sketch based on speed and curvature data, and uses this data to clean up sketches in a CAD-like system for engineering diagrams. [Cal2002] also segments input sketches by finding corner points but uses a trainable recognizer based on semantic networks. Fuzzy sets have also been used to recognize shapes by [Fon2000]. Our current system does not incorporate a trainable recognizer and, instead, like in [Cal2002], it relies on segmenting the input sketch based on identifying corner points, as will be explained in detail in Chapter 4. Before getting to such details, the next chapter first outlines our system.

Chapter 3

System Overview

Our animation system is divided into a number of functional blocks: sketch analysis, the motion controllers which then synthesize the motion data to drive the animated character, and the system manager. The role of each component is elaborated in this chapter.

3.1 System Organization

The organization of our animation system and the path of user input through the system is illustrated in Figure 3.1. A system manager (not shown) coordinates the activities of all components. It handles all communication between components and maintains and coordinates the list of known controllers.

The functional blocks of our system serve to support two main tasks: sketch analysis and motion parsing. Sketch analysis takes a sketch which consists of a sequential list of time-stamped input points generated by the user and converts it into a sequence of gestures such as an arc or a loop. Motion parsing then interprets these gestures and transforms them into a sequence of motions which are synthesized by the motion controllers. These two tasks are discussed further in the following sections.

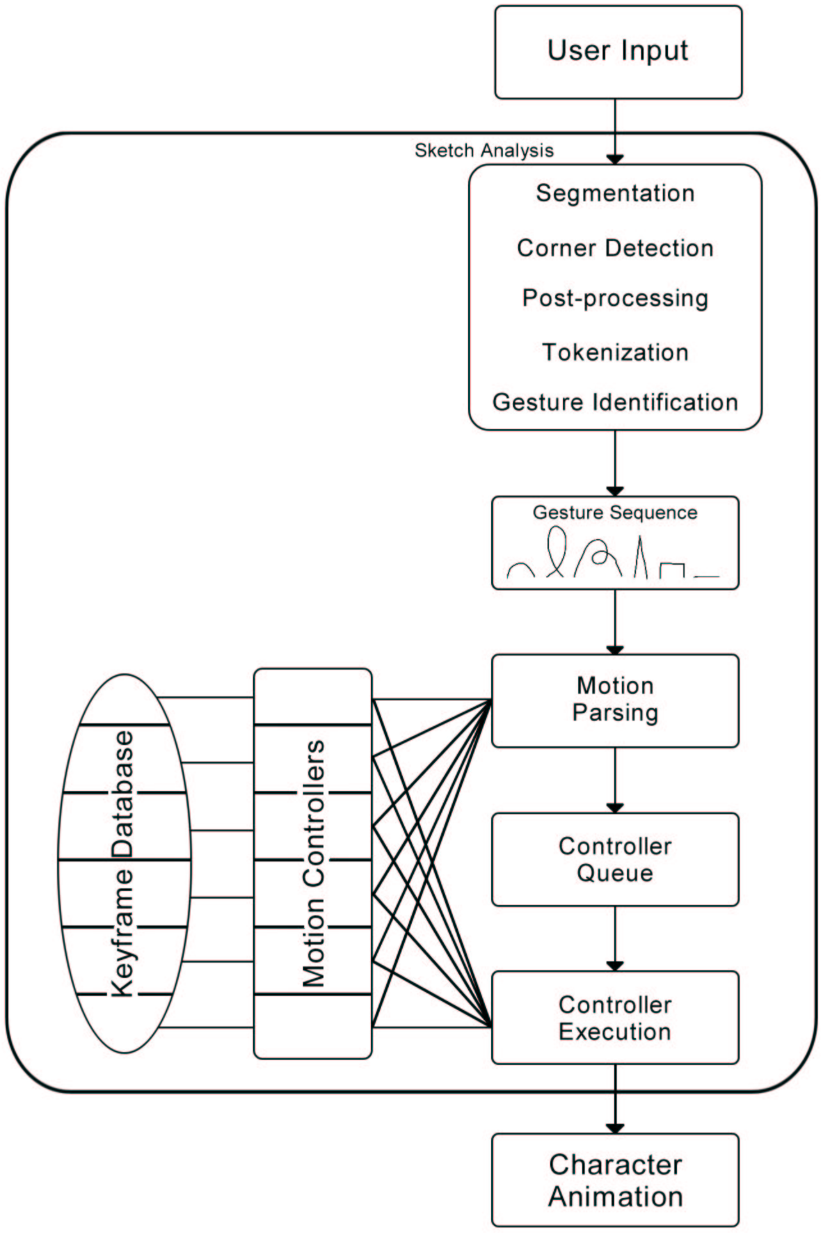

Figure 3.1: An overview of the Motion Doodles system architecture.

3.1.1 Sketch Analysis

The sketch analysis subsystem receives a sequence of time-stamped points, the sketch, as input. The input points are grouped into segments which are delimited by changes in the direction of motion of the input device. These segments are then analyzed further, broken into smaller segments at sharp corners and converted into tokens which are used to identify the gesture primitives recognized by the system. Finally, a sequence of identified gestures is passed on to the motion parsing stage. The sketch analysis process is described in further detail in Chapter 4.

3.1.2 Motion Controllers and Motion Parsing

Each motion supported by our animation system is implemented by a *motion controller*. Internally, the motion controllers are hard-coded finite state machines with each state associated with a single keyframe from the underlying keyframe database. Each controller in our system has between four and eight states, most of them proceeding in a linear sequence from one state to the next. Transitions from one state to the next are triggered either by contact between the ground and some part of the foot or based upon timing information derived from the input sketch and the controller definition. The workings of the motion controllers are discussed in detail in section 4.3 and Appendix A

The sketch analysis component of our system produces a sequence of gestures as its output. The mapping between gestures and motions is not a one-to-one as some gestures may be used to represent more than one motion and gestures may also have a different meaning depending on the gestures immediately preceeding or following them in the sequence. For example an arc may normally indicate one motion, but when occuring in the sequence arc-line-arc it becomes part of a larger gesture. For these reasons the gesture sequence must be parsed by the motion controllers. As the gesture sequence is parsed, the controllers that are associated with a particular gesture or sequence of gestures are placed on a queue of controllers awaiting execution. The process of parsing the gestures is described in more detail in the next chapter.

3.2 The Animated Character

Figure 3.2 shows examples of the characters rendered in both the 2D and 3D versions of the system. The 2D character is shown rendered in a 3D world.

The animated character is assumed to consist of sixteen links, arranged hierarchically as shown in Figure 3.3. There are three links in the torso, one for the head, two for each arm, two for each leg and two in each foot. The extra links in the torso and feet give added flexibility to the character and making the animation seem less rigid.

The same hierarchies used in both the 2D and 3D versions of our system, except that the 3D character has extra joints at the hips and shoulders to offest the arms and legs from the torso, and extra joints at the ankles. These extra joints are illustrated as shaded nodes in the hierarchy and are not present in the 2D versions.

Character Sketching

A system that allows a user to quickly sketch a desired character of the type shown in Figure 3.2 (bottom) has been implemented as described in [Tho2003]. The front-end allows a user to sketch a custom character for use with the animation system. A summary of the character sketching system is presented here.

The user sketches a character by sketching seven links, one for the head, body, upper and lower arm, upper and lower leg and foot. Each of these links

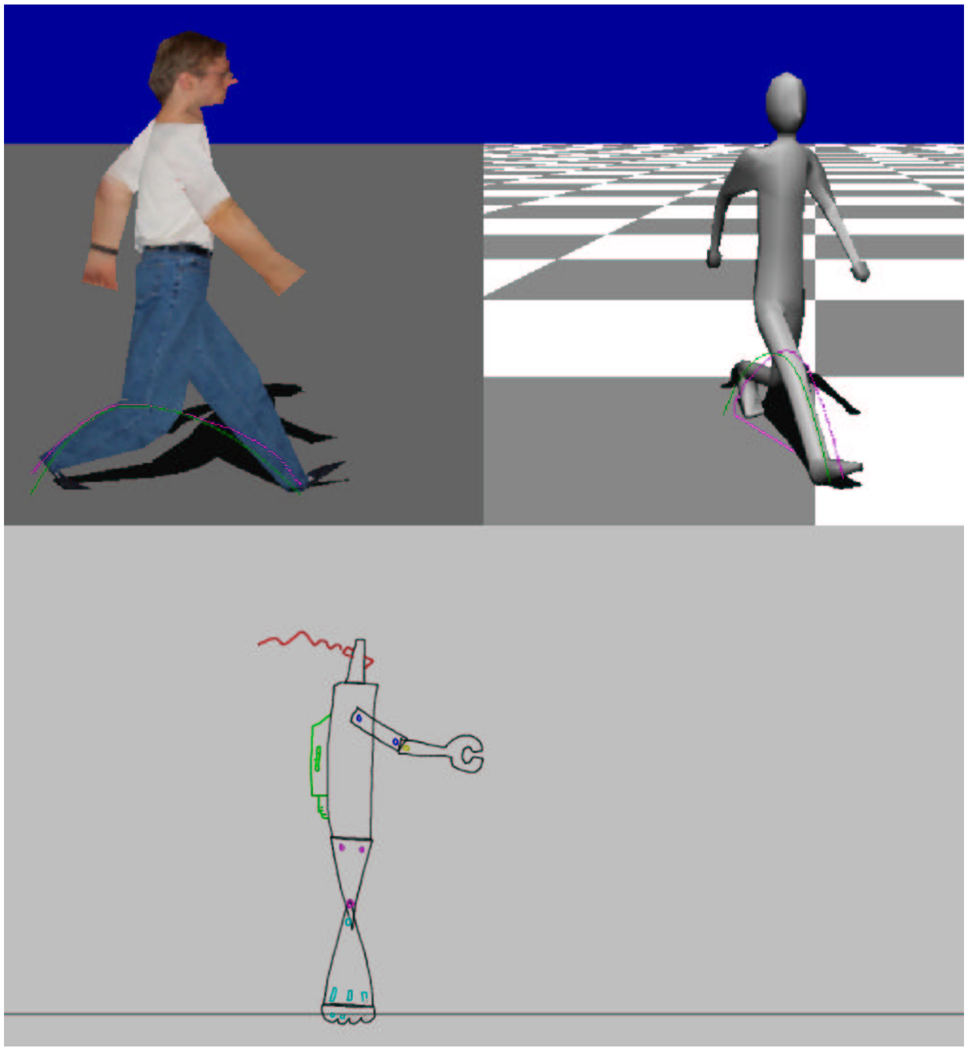

Figure 3.2: Rendered versions of the 2D Character (top left), 3D Character (top right) and a sketched character (bottom).

is drawn as a single continuous stroke and may be a simple shape such as an ellipse or may be more complex, containing surface details such as noses or fingers. The links can be drawn in any order and in any of a wide range of initial poses. Some poses are prone to problems and should be avoided, such as poses with the arms too close to the head or with the arms hanging straight down at the sides of the character. The body is automatically subdivided into three separate links and the foot is also automatically subdivided into heel and toes. This is done according to simple predefined proportions, such as assuming that the toes are 15% of the length of the foot. Links belonging to the arm or leg are mirrored to create the second arm and leg, giving the full sixteen links needed for the animation system's character hierarchy.

Once the character has been drawn, the character sketching system automatically infers the locations of joints and determines the connectivity between links. Joint locations are found using the major-axis of oriented bounding boxes fit to each link. The link connectivity is easily determined using the fact that we are working with a humanoid character of known topology. These two steps are described in detail in [Tho2003].

Even though our animation system requires a fixed character hierarchy, animated characters having a wide variety of limb sizes may be animated through the use of the character sketching front-end. The system is able to successfully handle characters with a wide variety of proportions through the use of inverse kinematics when ground interactions are involved. The details of this will be elaborated in Chapter 4 and Appendix A.

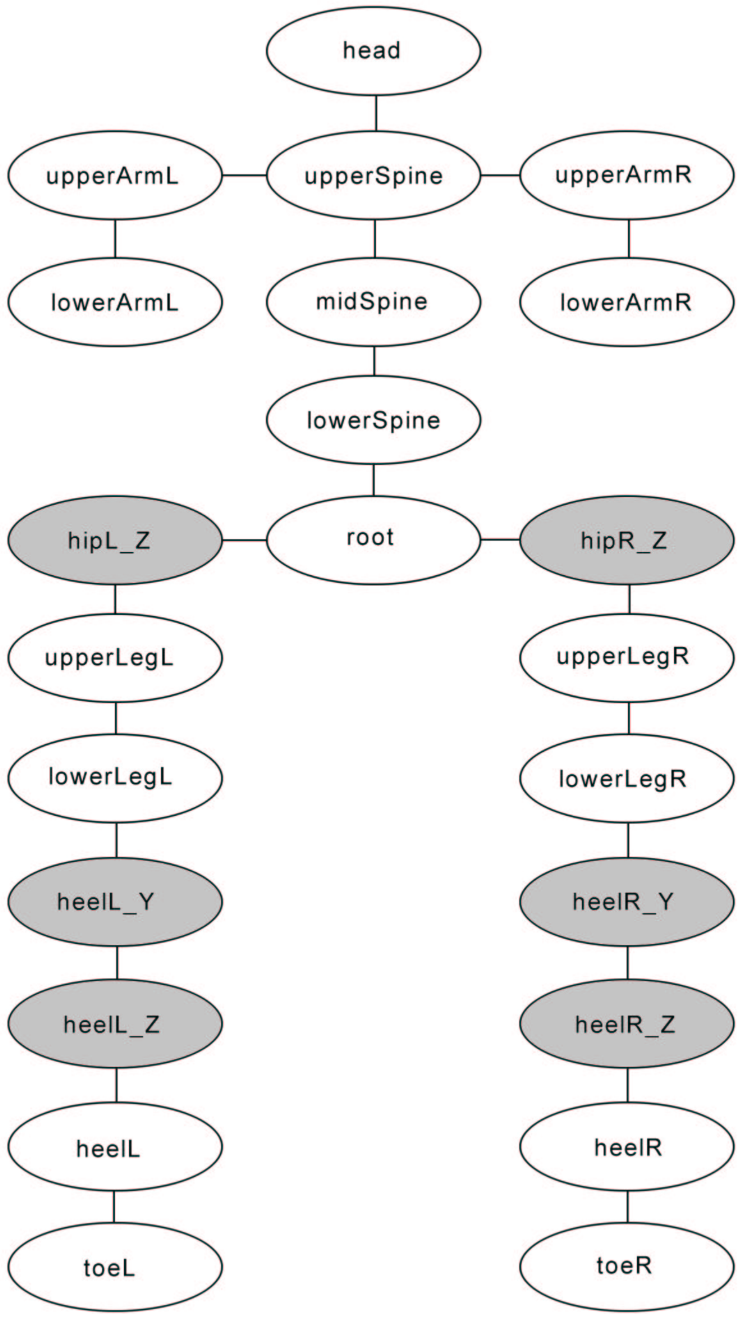

Figure 3.3: Each link in the character hierarchy has a length and one rotational degree of freedom. The shaded links are only present in the 3D character and are of zero length, allowing for additional degrees of freedom at the hips and ankles.

Chapter 4

2D Motion Sketching

The previous chapter discussed the organization of the system, how its components interact with each other and gave an overview of the process of taking an input sketch and transforming it into an animation sequence. There are several phases in interpreting an input sketch, or "motion doodle" as drawn by a user and converting it into a form that can be used for motion synthesis. This chapter discusses this transformation process in detail and how the motion controllers produce the final motions.

4.1 Motion Sketching Definitions

The terms used in discussing the successive stages of the sketch analysis and motion synthesis process are defined below:

Point A point is a single 2D position sampled from an input device.

Segment A segment is a contiguous group of points in the input sketch whose end points mark important feature points in the sketch such as corners or the transition from ascending to descending y-coordinates of input points.

Token A token is an abstract label applied to a segment based on shape (curved versus linear) and slope.

Gesture A gesture is the smallest unit of a sketch which is translated into a type of motion, and they are classified according to curve type – arc, loop, line, for example, which are shown in Figure 4.6. Gestures are built out of multiple tokens but no token may contribute to more than one gesture.

Controller Each motion that the animated character can perform, such as a jump or step, is handled by a controller. A controller is identified by a sequence of one or more gestures.

These definitions are used throughout the remainder of this chapter.

4.2 Sketch Analysis

In our animation system, a sketch is a sequence of time-stamped points, $p_i = (x_i, y_i, t_i)$, $i = 0...N-1$ where N is the number of points, collected between a single stylus[1] down event and a stylus up event. Input points are first filtered to discard incoming points that are less than a predetermined distance from the previous input point. This helps to smooth out noise that may result from jittery movement of the stylus.

4.2.1 Analysis Overview

The sketch analysis process itself consists of several sequential steps as illustrated in Figure 4.1: rough segmentation, corner detection, corner post-processing, merging of (nearly) colinear segments and identifying stylus pauses in linear segments. Sketch analysis is performed on-the-fly. That is, as each input point is received, it is examined to see if a new rough segmentation point can be created and, if so, further analysis is performed immediately on the new segment.

As segments are identified they are classified as tokens. The types of tokens used in our system are illustrated in Figure 4.2. The sequence of segments shown

[1]In this thesis we assume that 'stylus' can refer to a mouse, tablet stylus, electronic whiteboard or other similar input device.

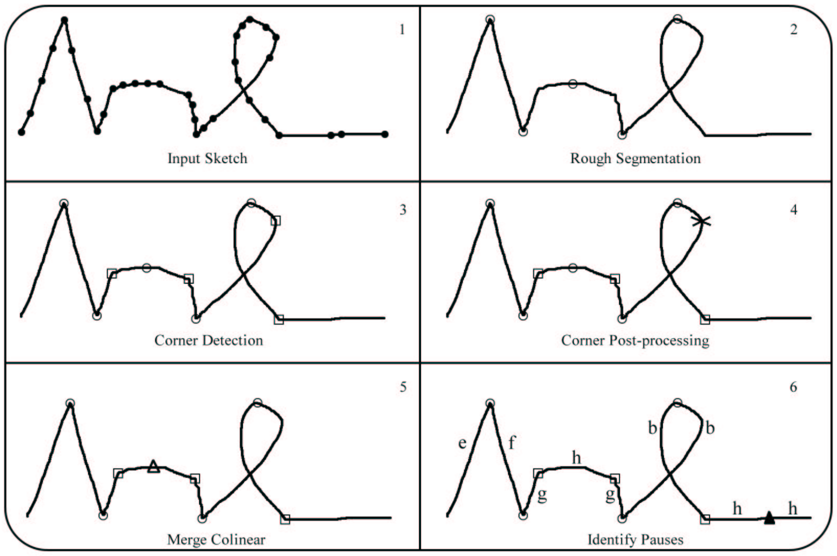

Figure 4.1: A summary of the sketch analysis phases. 1. The input sketch with sampled stylus points marked by solid circles. 2. Hollow circles mark boundaries in the rough segmentation. 3. Hollow squares mark corners detected by curvature analysis. 4. The corner at the 'x' is discarded in corner post-processing. 5. The hollow triangle where two neighbouring, nearly colinear segments are merged. 6. The solid triangle marks where a linear segment is split due to a pause in stylus motion. Box 6 shows the final segments with each boundary point marked.

in Figure 4.1 ultimately produces the token sequence: `efghgbbhh`, assuming the sketch is drawn from left-to-right.

We now describe each phase of the analysis process in greater detail.

4.2.2 Rough Segmentation

The first step in analyzing the input sketch is to segment it based on changes in stylus motion as shown in Figure 4.3. Since the gesture primitives in our animation system can be viewed as leaving the ground plane (upwards motion) and then, eventually, returning to the ground plane (downwards motion) an

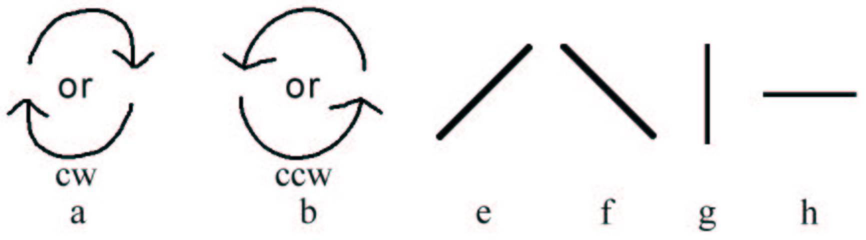

Figure 4.2: Tokens are shown labelled with their character encoding. Note that tokens for curved segments have an associated direction (clockwise or counter-clockwise) but tokens for linear segments do not.

initial rough segmentation is created by segmenting the sketch at points where the stylus motion changes direction from downwards to upwards or vice-versa.

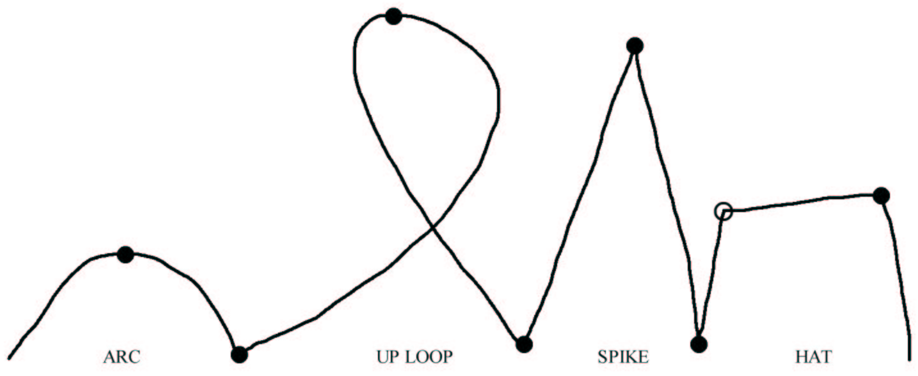

Figure 4.3: Example of the rough segmentation scheme. Solid circles mark segment boundaries determined by changes between upwards and downwards stylus motion. The hollow circle marks a corner point which is missed when using only the rough segmentation scheme.

As each initial segment is identified by this rough segmentation scheme, it is passed to a corner detector for further analysis. Additional analysis is necessary because the rough segmentation may miss features in the sketch which are important for identifying gestures. For example, Figure 4.3 shows a sketch with a sequence of four gestures – arc, loop, spike and hat from left to right. These and other gestures recognized by the system are shown separately in Figure 4.6. The hat gesture is broken into two segments, the first (leftmost) of which contains a corner point that is not identified by the rough segmentation step (refer to the marked point in Figure 4.3). Without further analysis to detect this corner the gesture would be misinterpreted by the system.

4.2.3 Corner Detection

To catch important features such as the above mentioned corner point missed by the rough segmentation phase, each segment found is passed through a corner detection algorithm based on the one given by [Che1999] and explained below. The basic idea of the algorithm is to identify points along a curve where it is possible to inscribe a sufficiently narrow angle, as shown in Figure 4.4.

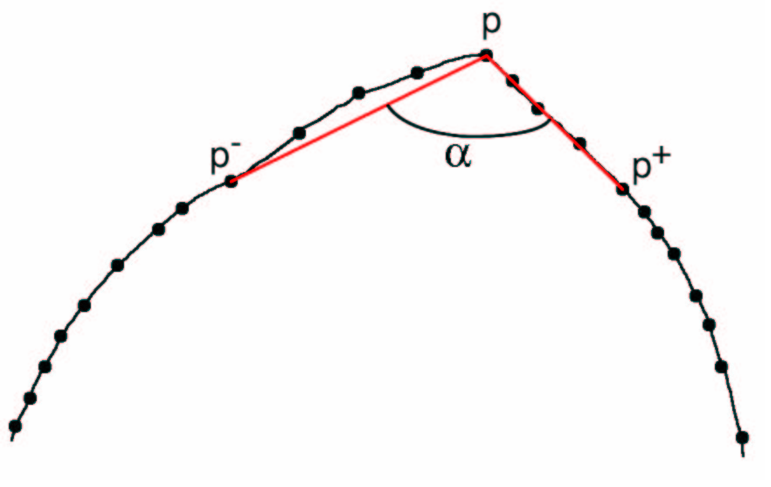

Figure 4.4: Corner Detection. Points at which it is possible to inscribe sufficiently small angles are identified as corner candidates.

Corner Detection Algorithm

At each curve point, p_i, we attempt to inscribe a triangle (p^-, p_i, p^+) which satisfies the following constraints:

$$d_{min}^2 \leq |p_i - p^+|^2 \leq d_{max}^2$$

$$d_{min}^2 \leq |p_i - p^-|^2 \leq d_{max}^2$$

$$\alpha \leq \alpha_{max}$$

where d_{min} and d_{max} are parameters which determine the region of the sketch around point p_i which affect the detection of a corner at that point. Smaller values of d_{min} make the algorithm more susceptible to flagging jitters in the sketch as corners, while larger values of d_{max} make it more likely to detect corners in areas of high curvature. α_{max} is the maximum angle considered to be a corner.

For each input point p_i, we search outward for potential triangles that satisfy the above constraints. The search stops either at segment boundaries (as determined by the rough segmentation step) or when one of the constraints is violated. If at least one triangle was found that satisfied the constraints, p_i is recorded as a corner candidate, otherwise it is ignored.

Corner Post-processing

By itself, this algorithm still tends to identify too many false positives and also some false negatives. Sections of the sketch intended to be smoothly curved generate false positives due to the noisy nature of the input sketch and some areas perceived as corners to the human eye are missed because they are too rounded to be identified as corners by the detection algorithm.

In order to alleviate the problem of false positives, potential corners found by the detection algorithm are verified using knowledge about the speed at which the curve segments were drawn. Since drawing speed tends to decrease when drawing a corner, corner points at which the stylus speed is greater than some threshold are discarded as being unlikely corner candidates. The default threshold is about 30% of the average stylus velocity along the segment.

The original sketch segment is divided into smaller segments at each of the identified corner points. These refined segments passed along for classification and testing for two special cases.

Special Segmentation Steps

After corner detection is complete there are two special steps that occur before segments can be converted into tokens. First, segments are classified as either curved or linear. The method for doing this classification is based on [Sez2001]. The arclength along the sketch segment is compared to the Euclidean distance between the endpoints of the segment. For a linear segment, this ratio will be close to 1 (in our system the range [1, 1.025] for this ratio indicates a linear segment), otherwise the segment is curved. Curved segments are passed on without further modification to the tokenization step.

Linear segments are compared to neighbouring segments. Pairs of linear segments which are nearly colinear are merged into a single segment. This step is necessary since jitters in the up/down motion of the input device may cause portions of the input sketch that are intended to be horizontal (or nearly horizontal) to be split in the rough segmentation phase rather than leaving them as a single segment. It is possible that, after merging several nearly colinear segments, we may end up with an arc rather than a linear segment although in practice this does not appear to happen.

Further, linear segments are subdivided at points where the stylus comes to a momentary stop in order to provide a mechanism to allow multiple sequential straight line segments that are nearly colinear.

4.2.4 Tokenization

In order to facilitate the identification step, sketch segments are converted into a sequence of tokens which encode some high-level information about the segment that they represent. The types of tokens are summarized in Figure 4.2 and the information encoded in a token is summarized in Table 4.1.

Figure 4.2 shows each token along with its character encoding. Tokens `a` and `b` represent curved segments drawn in a clockwise or counterclockwise direction respectively. The linear tokens are based only on the slope of a segment, not the direction it is drawn. The ranges of slopes used to classify each linear segment are illustrated in Figure 4.5.

type	a single character uniquely identifying a token
angle	the net angle traversed along the length of a segment (linear segments have angle 0); for curved segments this value is calculated by summing the angle formed by successive pairs of points along the length of the segment
vDir	binary value which encodes the net vertical direction of the segment underlying the token (UP/DOWN)
sPt, ePt	pointers to the starting and ending points of the segment in the input sketch

Table 4.1: Summary of information encoded in a token.

4.2.5 Identification

The final step in sketch analysis is to parse the token stream into recognized gestures. The gestures currently recognized by our animation system are shown in Figure 4.6.

Gesture identification uses a combination of information from the sequence of tokens. The identifying features, namely the acceptable grammar expressions, of each gesture type are summarized below. If more than one gesture matches a

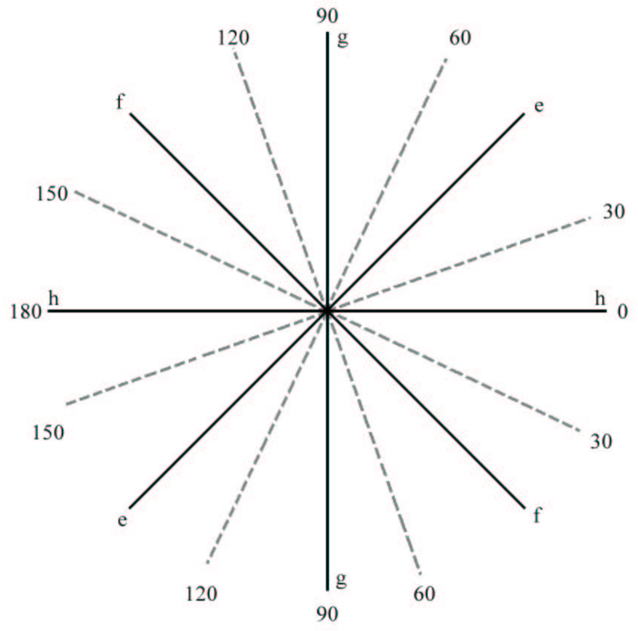

Figure 4.5: The "bins" used for classifying linear tokens. The dark, solid lines represent the actual tokens (and are labelled with the token they represent) and the lighter, dashed lines mark boundaries between bins.

portion of the token sequence then the gesture containing more tokens is taken or, if they are the same length, then the gesture with higher priority is used. Gesture priority is defined implicitly by the order in which they are added to the system.

Arc

An arc gesture consists of two tokens: an upward token followed by a downward token, one of which must be curved. Accepted token sequences (refering to Figure 4.2) are, when drawing left-to-right: `aa`, `af`, `ag`, `ah`, `ea`, `ga`, `ha`. When drawing right-to-left these become: `bb`, `be`, `bg`, `bh`, `fb`, `gb`, `hb`. Note also, that

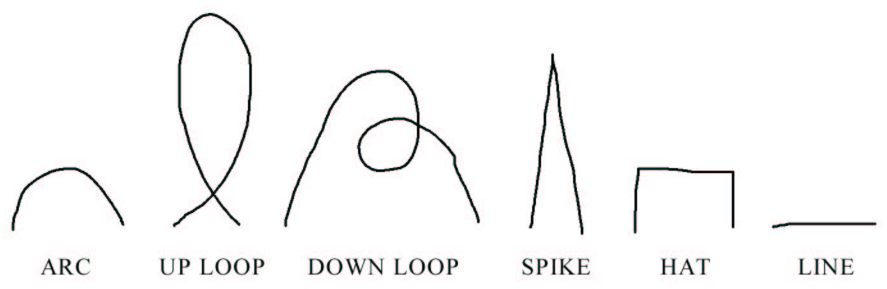

Figure 4.6: Gestures recognized in the animation system. Gestures may be drawn left-to-right or right-to-left.

the overall direction of rotation of the tangent to the sketch must be the same over both tokens – that is, both tokens must be clockwise or both must be counter clockwise. For our purposes, linear segments are considered to have the same rotation as both clockwise and counter-clockwise curved segments since linear segments introduce no net rotation. This means that the only token combinations that are not considered arcs are `ab`, `ba` or any pair of linear segments.

Up Loop

Like the arc, the up loop is an upward token followed by a downward token, one of which must be curved. However, the direction of rotation of the tangent is opposite to that of the arc and the up loop is also expected to intersect itself. To be considered an up loop, either a self-intersection must be detected or there must be a net rotation of at least 245° in the sketch. Acceptible token sequences (drawn left-to-right) are `bb`, `gb`, `fb`, `eb`, `bg`, `bf`, and `be`. From right-to-left these become `aa`, `ga`, `fa`, `ea`, `ag`, `af` and `ae`.

Down Loop

The down loop is the most difficult gesture to recognize since it is the most complex and the first half of a down loop can easily be mistaken for an arc. The direction of rotation of the tangent is the same for a down loop as for an arc, but like an up loop we expect to find a self-intersection somewhere along the length of the loop. The ambiguities between arcs and down loops do, however, result in delaying in the recognition of arcs until a down loop can be either discarded, in which case one or more arcs may be recognized, or a down-loop is positively identified. The acceptible token sequence, drawn left-to-right, is `a(aa)`*`a` and drawn right-to-left is `b(bb)`*`b`.

Spike

A spike is similar to an arc in that it consists of an upward token followed by a downward token. Unlike an arc, both tokens must represent straight lines. Accepted token sequences are: `ef`, `eg`, `ee`, `gf`, `gg`, `gf`, `ff`, `fe`, `fg`.

Hat

A hat is defined as a sequence of three straight lines: one upward, one horizontal and one downward. The corresponding possible sequences of tokens include: `(f|e|g|a|b)hh`*`(f|g|e|a|b)`. Note that curved segments are allowed in the upwards and downwards strokes since, especially when drawing quickly, these can sometimes appear curved to the system, even if this is not intended by the user.

Line

The line is the simplest gesture, being any straight line token that is not recognized as part of another gesture.

4.3 Motion Controllers

Each motion controller is responsible for examining the sequence of gestures that is output by the sketch analysis subsystem and identifying the gesture (or gestures) that apply to it, and secondly, is responsible for synthesizing the resulting output motion.

4.3.1 Controller Recognition

Each individual motion controller communicates with the system as described in Chapter 3.1 and, indirectly, with each other. A controller must be registered with the system before the system can make use of it. As an input sketch is drawn and analyzed, the gesture sequence produced by the sketch analysis component of the system is presented to each registered controller in the order that they were registered. Thus the order in which controllers are registered defines an implicit priority among the controllers, although there is currently one exception to this case in our system.

The gesture sequence resulting from sketch analysis is presented to each controller in the order they were registered with the system. If more than one controller recognizes a portion of the gesture sequence, then the longest sequence is always given priority, otherwise the first controller to recognize a gesture sequence is used. For example, the walking controller recognizes a sequence of one or more `ARC` gestures and the moonwalk controller recognizes the gesture sequence `ARC-LINE-ARC`. By selecting the controller that recognizes the longer sequence, in this case the moonwalk, we ensure that the initial gesture will not be interpreted as belonging to a different motion regardless of priority.

It is also possible for a single type of gesture to be associated with more than one controller such as is the case with the `ARC` which by itself can be interpreted as either a jump or a step, or the `SPIKE` which can be either a jumping stomp or stiff-legged step. A pair of system parameters are used to deal with these cases. The *jumpThreshold* and *maxStepLength* parameters determine the cut-off between walking and jumping motions. In general, a gesture with maximum height below *jumpThreshold* and horizontal distance between the starting and ending points less than *maxStepLength* are considered to be step-like motions, otherwise they are jump-like motions. In our system *jumpThreshold* is an arbitrary distance above the ground height and *maxStepLength* is set to the distance between the character's heels with the legs forming a 70° angle. These two parameters are illustrated in Figure 4.7

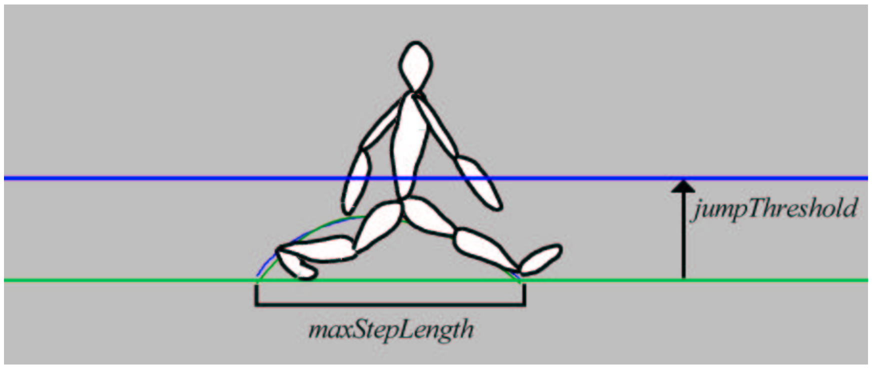

Figure 4.7: The two system parameters *maxStepLength* and *jumpThreshold* are used to differentiate between walking and jumping.

Table 4.2 summarizes the motion controllers, the sequence of gestures associated with each controller and any special considerations for that motion type.

Dealing with this sort of ambiguity means that the analysis and recognition process will lag behind the input sketch. Also, a new controller may be recognized before the current one has completed its animation, especially if, say, the `ARC-LINE-ARC` sequence turns out not to be a moonwalk after all but a step, shuffle, step or step, shuffle, leap instead, for example. Therefore, the manager maintains a queue of identified controllers which are processed in order.

Gesture(s)	**Motion**	**Special**
ARC	Jump	Maximum height above *jumpThreshold* or total distance longer than *maxStepLength*
ARC	Leap	As with Jump, but requires having an initial state having one foot ahead of the other (such as occurs after a step)
ARC	Step	Height of arc below *jumpThreshold* and horizontal distance shorter than *maxStepLength*
ARC-LINE-ARC	Moonwalk	Both ARCs need to satisfy the step criteria, and the LINE must have an opposite horizontal direction to the ARCs
HAT	Hop	None
LINE	Shuffle	Horizontal distance covered by line must be shorter than *maxStepLength*
LINE	Slide	Horizontal distance covered by line must be larger than (or equal to) *maxStepLength*
LOOP UP	Back Flip/Front Flip	Back Flip when drawn left to right, otherwise Front Flip
LOOP DOWN	Front Flip/Back Flip	Front Flip when drawn right to left, otherwise Back Flip
SPIKE	Jump-stomp	Spike peak higher than *jumpThreshold* or horizontal distance longer than *maxStepLength*
SPIKE	Stiff-legged Step	Spike peak below *jumpThreshold* and horizontal distance shorter than *maxStepLength*

Table 4.2: Controllers and their gesture sequences.

4.3.2 Motion Synthesis

Once the appropriate motion controller has been identified, the last step is for the motion controller to synthesize the motion. Internally, a controller is divided into a fixed number of stages and implemented as a finite state machine. The states for each controller in our system are summarized in Table 4.3. Each controller may have a different number of states. Additionally, each state applies a number of tools: a keyframe database, a keyframe interpolator, an inverse-kinematics solver and the ability to place the character's centre-of-mass at a specified point.

Controller	**States**
Jump, Leap, Hop	ANTICIPATE, LAUNCH, ASCENT, DESCENT, LANDING, FOLLOWTHROUGH
Front Flip, Back Flip	CROUCH, LAUNCH, ASCENT, CURL, SPIN, UN-CURL, LANDING, FOLLOWTHROUGH
Walk, Tip-toe, Stomp	CONTACT LEFT, DOWN LEFT, PASSING LEFT, UP LEFT (mirrored for right leg)
Moonwalk, Shuffle	LIFT, DRAG
Slide	PREP, PUSH, GLIDE

Table 4.3: Sequential controller states for each motion primitive.

Controllers directly or indirectly extract a number of parameters from the input sketch. These include the timing of a motion, height, distance, number of loops and so forth. Details on the controller internals and the information they extract from the sketch are summarized here for the jump controller; the full details for all controllers are given in Appendix A.

Jump Controller Example

The jump motion is broken down into six states: `ANTICIPATE`, `LAUNCH`, `ASCENT`, `DESCENT`, `LANDING` and `FOLLOWTHROUGH`, shown in Figure 4.8. Interally, the jump controller functions as a finite state machine, as do all others. In general, there are two types of events which may trigger a state change: contact between feet and the ground and timing information. For the jump controller, only the transition from `DESCENT` to `LANDING` is triggered by ground contact. All other transitions are triggered by timing information. For time-based states, the duration of that state is defined as a fixed percentage of the total time for the entire motion.

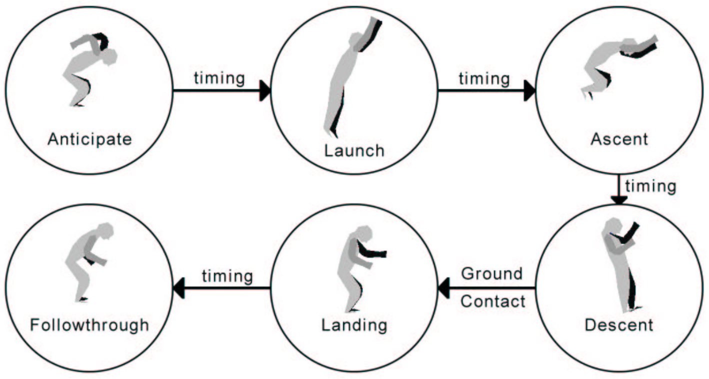

Figure 4.8: Jump controller state transition.

For a jump, three main pieces of information are extracted from a sketched arc: the arc start point, the arc end point, and the arc apex and the timing information for these three points. Two half parabolae are fit to these points as shown in Figure 4.9, one for ascent, the other for descent, and the slope of these parabolae at the start and end points are used to set the overall angle of the character for `LAUNCH` and `LANDING` respectively.

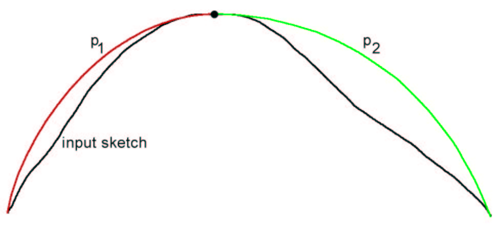

Figure 4.9: Fitting two half parabolae to an arc gesture.

Because the start and end points of the arc specify the foot location, and the apex indicates the location of the character's centre-of-mass at the peak of the jump, the two parabolae fit to these points are adjusted during the course of the jump. At the transition point between `LAUNCH` and `ASCENT`, the current centre-of-mass location becomes the start of the ascending parabola and at the transition between `ASCENT` and `DESCENT`, the end point of the descending parabola is adjusted so that it gives the position for the centre-of-mass such that the character's feet will land in the spot indicated by the sketch.

Keyframe Database

The input sketch is only a partial specification of the motion of the character at any one time. As well, different parts of a single gesture may specify different constraints on the character. This can be seen in the jump example where we have a specification of the location of the feet at the start and end of the jump, and the location of the centre-of-mass at the peak of the jump. This leaves many details of the motion unspecified. The keyframe database helps to fill in these missing details. A single keyframe is associated with each state of each controller, resulting in a fairly sparse collection of keyframes. Many of the keyframes used in the Motion Doodles system were inspired by [Wil2001] (walking, tiptoeing and jumping) and [Muy1955] (front and back flips).

The keyframe for each controller state is defined in terms of a set of joint angles and defines the target pose for the character at the *end* of the state. Catmull-Rom interpolation is used to interpolate between successive keyframes, treating the keyframes as forming a loop. For any state in which one of the character's feet touches the ground, inverse-kinematics is used to pose the leg, overriding the interpolated angles for the affected joints. Since the character's legs are composed of two segments and three joints (for simplicity we allow the character's feet to follow the keyframes) the cosine law is used to perform inverse-kinematics although other IK algorithms such as the Cyclic Coordinate Descent (CCD) method [Wel1993] could easily be used to handle more complex characters or to extend IK manipulation to other parts of the character.

If desired, other back-ends for generating motion data may be substituted for the keyframes depending on the needs of the application, such as a motion capture database. However, methods that result in more realistic motion synthesis would require either a less literal interpretation of the motion sketch or some alternate interpretation for phyically implausible motions such as a leap that no mere mortal could accomplish.

Our 2D sketch-based animation system, as described in detail in this chapter, allows a user to experiment with a wide variety of motions. One of the most obvious ways to extend our system is to allow for motion sketching in a 3D environment. The next chapter presents an initial 3D implementation of our system and, as shall be seen, the basic mechanics of the 2D system can be adapted to 3D.

Chapter 5

3D Motion Sketching

While the 2D Motion Doodles system provides an interesting starting point for exploring sketch-based animation, an obvious question to ask is whether it can be exteded to 3D. Is it possible to extend the basic mechanisms used in the 2D system to work in a 3D setting? The answer is "yes", but with some reservations. As shall be seen, the sketching paradigm can be carried over into a 3D environment, although the addition of an extra dimension introduces ambiguities into the interpretation of the 2D input sketch. Figure 5.1 shows an example of our 3D system in action.

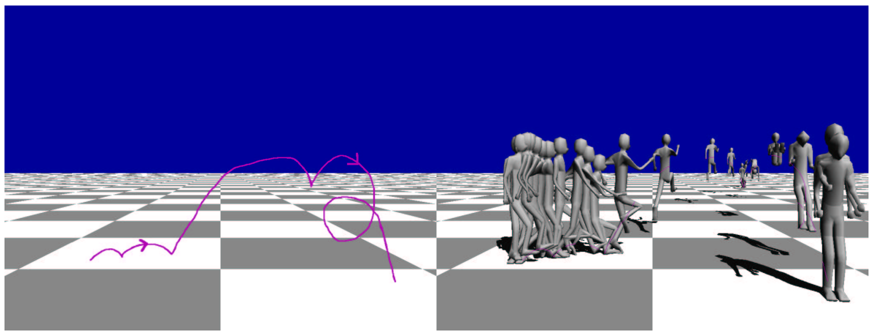

Figure 5.1: 3D Example: We wish to be able to draw a sketch (left) as with our 2D system and transform it into an animation sequence in a 3D environment (right).

One solution to the ambiguities inherent in interpreting 2D input in a 3D environment is to use 3D input devices such as a Polhemus tracker. While this option has not been explored for the purposes of this thesis, it is suggested as an avenue for future work as discussed briefly in Chapter 7.

5.1 2D Input to 3D Sketch

In extending the 2D sketch-based system to a 3D system, the main goal is to keep the same style of interaction as was used for the 2D system. That is, we want to take a 2D input sketch and convert it into motion in three-dimensional space. Preserving the 2D-sketching paradigm introduces two related issues to overcome:

- Mapping 2D points to 3D points
- Resolving ambiguities in the interpretation of the input

5.1.1 Mapping from 2D to 3D

The first challenge in extending our 2D sketch-based animation system to 3D is mapping input from a 2D input device to allow interactions with a 3D environment. To do this, we start by identifying the local minima of the sketch in screen space (points where the stylus motion switches from downward to upward motion) which define the start and end of a gesture and map those points to corresponding 3D points on the ground plane or height field. The 3D points are obtained by taking a ray from the eye point that passes through the minima point on the image plane and intersecting that ray with the ground plane as shown in Figure 5.2.

Once the start and end points of a gesture have been identified and have been mapped to the 3D world, the rest of the gesture must also be transformed into 3D space. This process is illustrated in Figure 5.3. The start and end points of the gesture (sPt and ePt) define a plane, $\mathcal{Q}$, perpendicular to the ground. Each point, p, on the gesture is mapped to the near and far clipping planes as

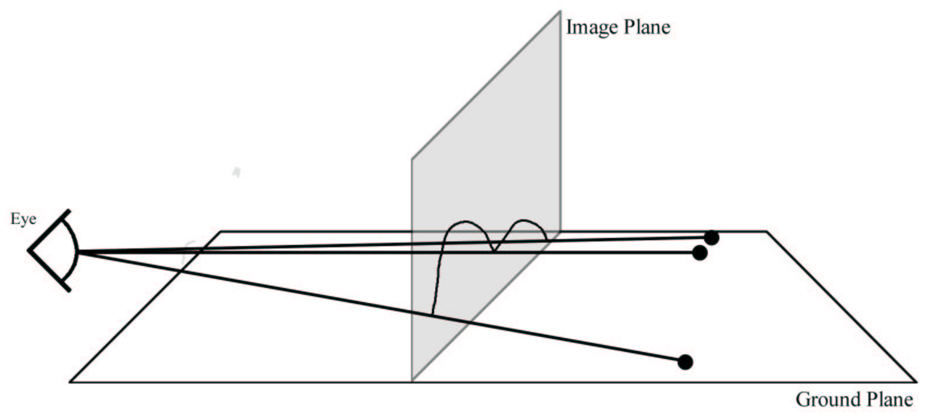

Figure 5.2: Sketch local minima projected to the ground plane.

p_n and p_f respectively. The line joining p_n and p_f is intersected with plane $\mathcal{Q}$ and this intersection point gives the location of pt in the 3D world.

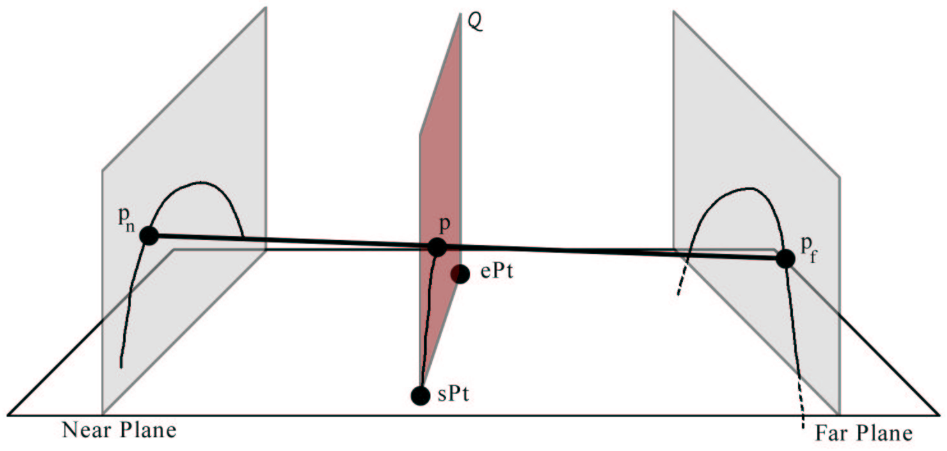

Figure 5.3: Mapping a 2D sketch to 3D. Start and end points of a gesture are projected onto the ground plane in order to establish the location of a vertical plane in the 3D world that then embeds the rest of the gesture.

5.1.2 Resolving Ambiguities

The transition from 2D to 3D introduces two further types of ambiguities that do not exist in the 2D system. These are gesture ambiguities and direction ambiguities. We now describe what these are and how we cope with them.

Gesture Ambiguities

The limitations inherent in using a 2D input device to specify 3D motions means that not all gestures can be easily distinguished from one another. As a simple example, the input shown in Figure 5.4 can be interpreted as a spike producing either a stiff-legged walk or a stomping jump, *or* it could be interpreted as two line gestures, which would produce a long slide back away from the camera followed by a turn and a slide towards the camera. An ideal system should allow for both possibilities since a user may intend for either case in different instances.

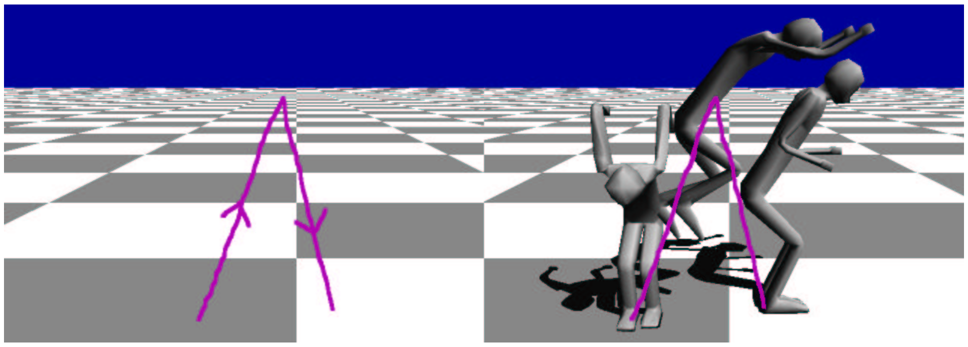

Figure 5.4: Gesture Ambiguities: The input sketch (left) can be interpreted as a single spike gesture in a plane parallel to the impage plane, or as two separate line gestures along the 3D ground plane. Our system (right) interprets this as a single spike gesture, producing a jumping action.

One potential solutions to resolve this situation is to assume one case or the other, or perhaps select between cases using a threshold value. The user might then override this behaviour in a post-processing phase, if desired. An alternate solution is to add an auxilliary gesture or event, such as a keypress to indicate a break between gestures.

Direction Ambiguities

The extra degrees of movement afforded by a 3D environment also add ambiguity in terms of the direction of movement. As illustrated in Figure 5.5, an arc in one direction followed by an arc in the opposite direction could be interpreted as: (1) a forward step (facing right) followed by a backward step; (2) as two forward steps involving a sharp, nearly 180° turn; or (3) as a backward step (facing left) followed by a forward step. This ambiguity occurs because while the direction of travel is specified by the direction in which the gesture is drawn, the current facing direction is unknown.

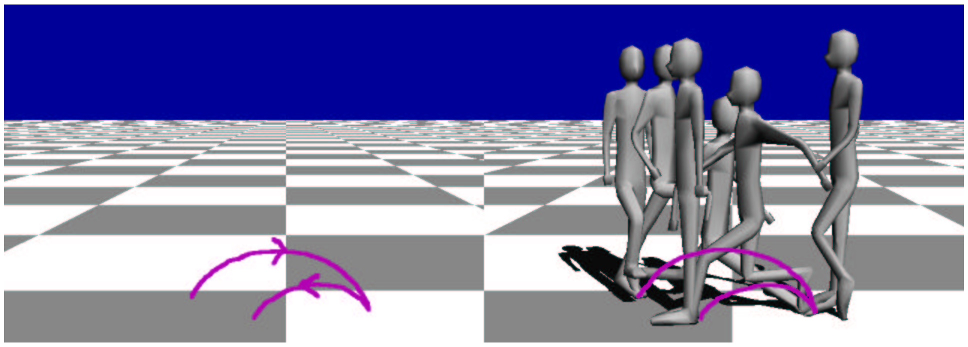

Figure 5.5: Direction Ambiguities: Two successive arcs drawn in opposite directions (left) can be interpreted in several ways. Our system (right) interprets this as two forward facing steps with a sharp turn between them.

Techniques such as a keypress to disambiguate the facing direction could also be used in this case. As another option, a two-pass system could be used in which, the user first draws a general path of travel through the scene, followed by a sketch of the motion along that path as they would do in the 2D system.

For the implemented 3D prototype, motion is always forward; the character may turn sharply at times, but it never walks backwards.

5.2 3D Motion Synthesis

Synthesizing motion in 3D occurs much as it does in the 2D case. The same features are extracted from the sketch as with the 2D system and a keyframe database serves as a back-end to fill in details of the motion that are not provided directly from the sketch or through inverse kinematics. For our prototype 3D system we use the same set of keyframes as in the 2D system.

There are two main points in which 3D motion synthesis differs from its 2D counterpart: foot placement and direction smoothing.

5.2.1 Foot Placement

In the 2D system, the start and end point of each gesture marks where the foot (or feet) strikes the ground. This has been relaxed slightly in the 3D system. Instead of falling exactly on the points taken from the sketch, the feet are offset from the vertical plane embedding the sketch to allow for separate left and right foot placements. This is illustrated in Figure 5.6 for the case of a single jump.

5.2.2 Direction Smoothing

The direction that the character is facing does not change in the 2D system but it may do so in the 3D system. A vector from the start point to the end point of a gesture is used to define the character's direction of travel. However, simply setting the character's heading to this direction vector results in sharp discontinuities in the character's direction as it snaps to the new heading for each gesture which is an undesirable artifact. Some form of smoothing is required.

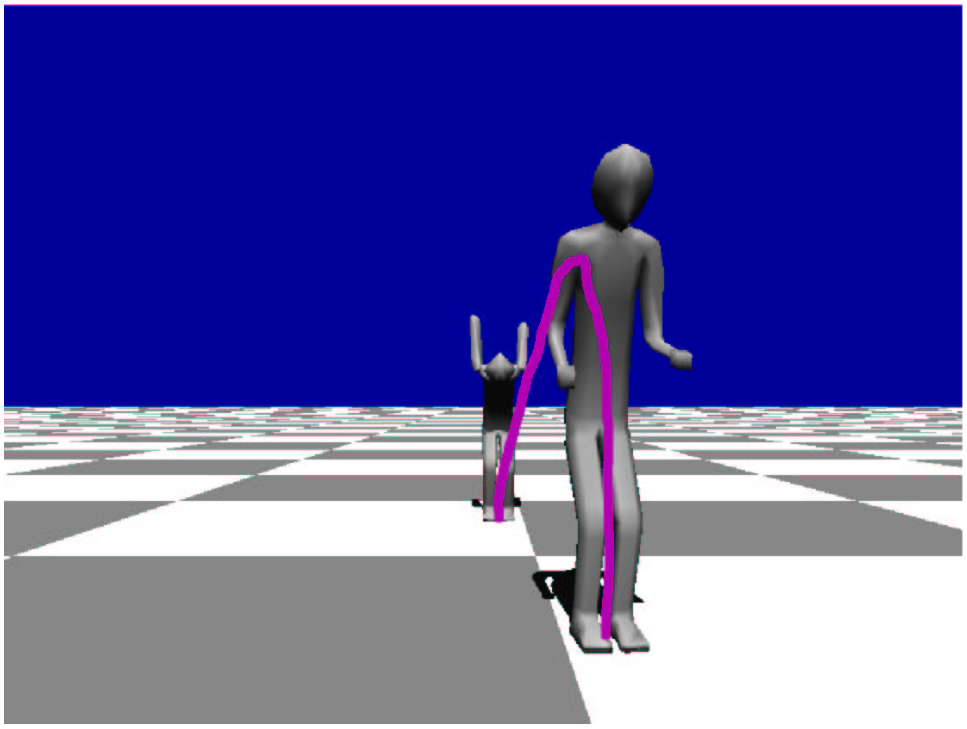

Figure 5.6: Foot placement in 3D.

There are several possible ways to approach this, such as carefully selecting a foot orientation to suit both the present motion and the upcoming motion. The method implemented in our prototype is to simply interpolate the heading over a small portion of a motion. This gives the character the appearance of rotating on its heel.

As a side effect of the foot placement and interpolating of the character's heading, the heading may not be exactly the direction specified by the direction of the gesture during walking motions. Figure 5.7 illustrates the situation with a sequence of three steps. Figure 5.7.a shows what happens if we adhere to the gesture's direction vector. The foot position drifts and is no longer centred around the gesture's start and end points – this is particularly noticable after a sharp turn. Figure 5.7.b shows how the target heading is modified to ensure that foot placement occurs as desired.

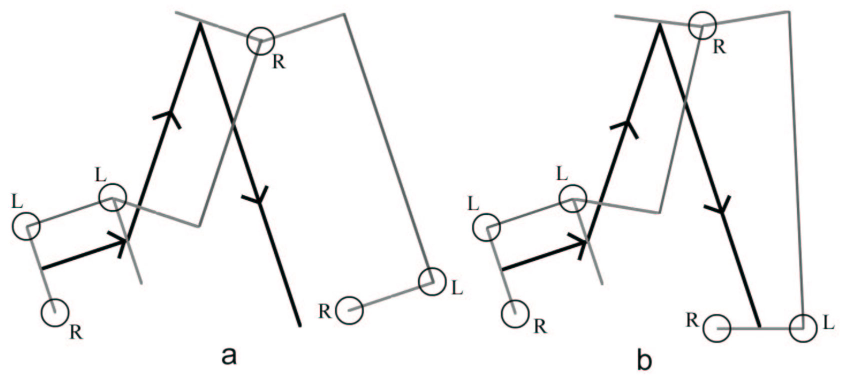

Figure 5.7: A sequence of three steps shown from an overhead view with foot positions marked by circles. In (a) the character's heading adheres to that defined by the gesture but as a result the foot location exhibits drift from the desired position. In (b) the character's heading is modified so that the feet are always placed in the correct position.

Having discussed the details of our system, we now present the results in Chapter 6.

Chapter 6

Results

This section presents a selection of results generated with our animation system, showing each of the individual motions, their variations, as well as longer compound motion sequences containing several types of motions.

Time is an important input dimension that cannot be seen in the figures presented in this chapter. Our system produces animations that mimic the timing of the input sketch so that a sketch may be drawn with different timings to produce different animated motions.

The figures in this chapter show character instances that are not equally spaced in time in order to maximize the clarity of the figures.

6.1 2D System

The motions available in the 2D animation system are illustrated below and are rendered using the character sketching front-end described in [Tho2003].

6.1.1 Jumping and Variations

Figures 6.1 through 6.4 illustrate the basic two-footed jump, a leap, an angry jumping stomp, and a hop.

The leap motion in Figure 6.2 is preceeded by a single step since in order to recognize an arc as a leap rather than a jump, the character's feet must be separated when starting the jump.

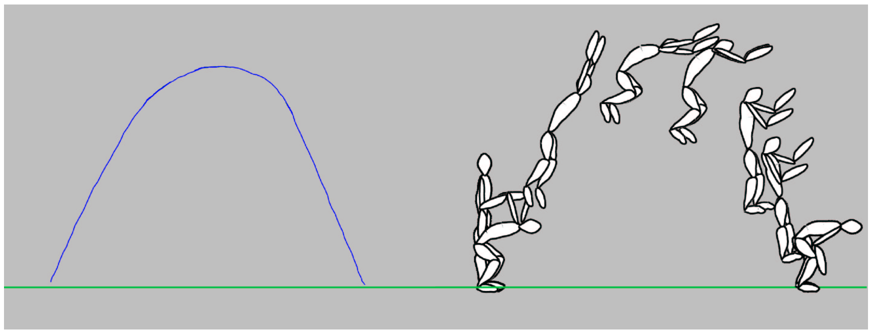

Figure 6.1: The Jump: Arc gesture (left) and resulting jump motion (right).

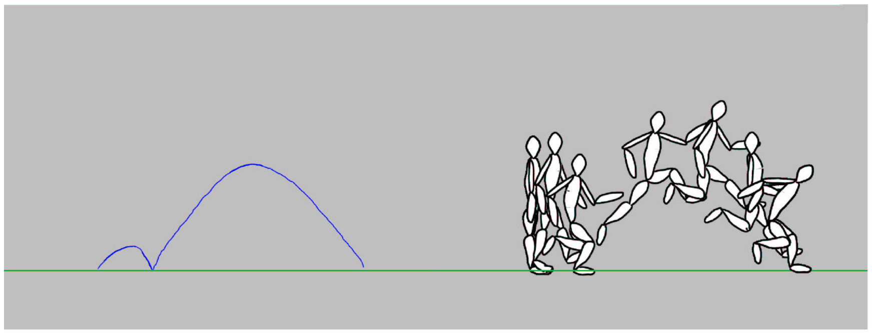

Figure 6.2: The Leap: Arc gesture of a small step and then a leap (left) and the resulting step and leap motion (right).

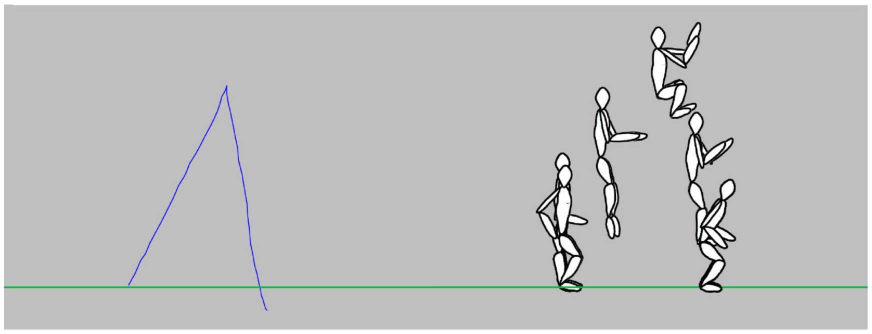

Figure 6.3: The Jumping Stomp: Spike gesture (left) and the resulting jumping stomp motion (right)

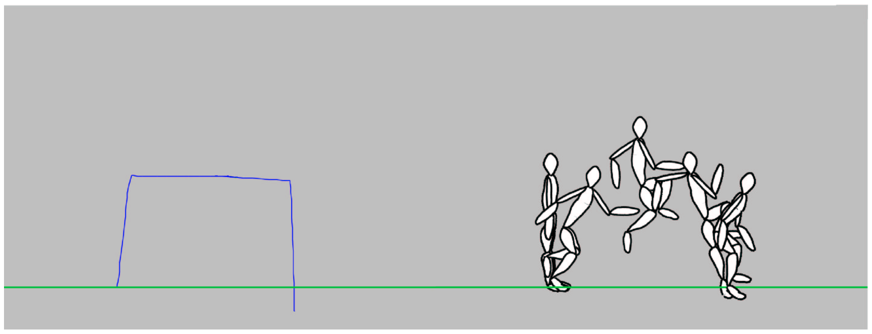

Figure 6.4: The One-legged Hop: Hat gesture (left) and resulting hopping motion (right)

6.1.2 Front and Back Flips

Figures 6.5 through 6.8 illustrate the front and back flips.

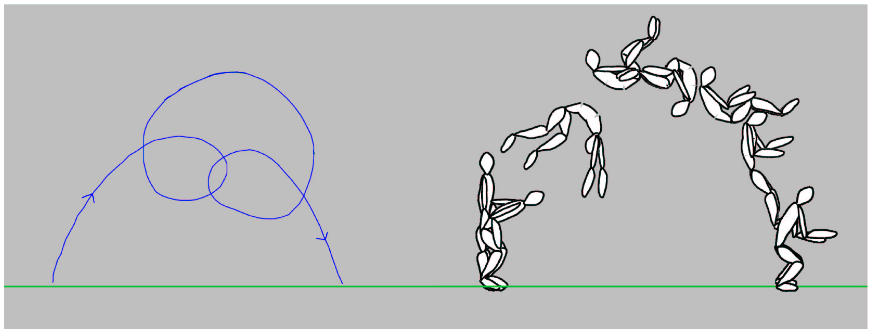

Figure 6.5: The Forward-traveling Front Flip: A down loop gesture containing two loops (left) and the resulting double front flip (right)

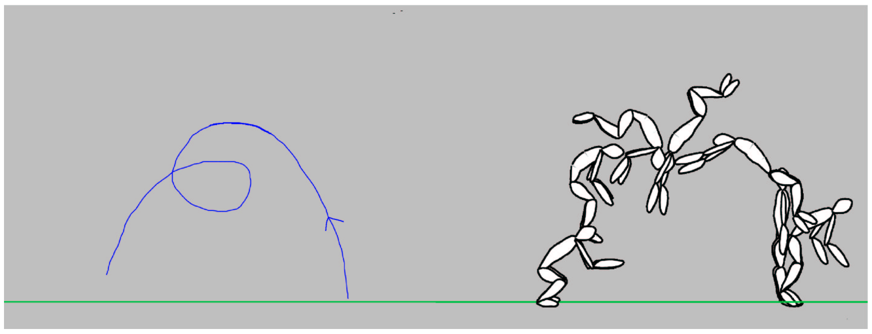

Figure 6.6: The Backwards-traveling Back Flip: A down loop gesture drawn right to left (left) and the resulting backwards back flip (right)

When drawn left-to-right (Figure 6.5) the down loop translates into the front flip action but if drawn right-to-left (Figure 6.6) it produces a back flip.

A forward-traveling back flip and a backwards-traveling front flip have gestures which preserve the desired mapping from the direction of rotation of the motion doodle to the rotation of the body during the flip. These motions are illustrated in Figure 6.7 and Figure 6.8.

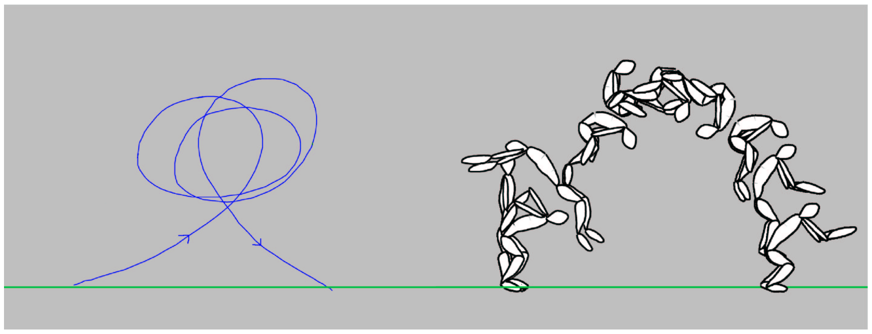

Figure 6.7: The Forward-traveling Back Flip: An up loop gesture containing three loops (left) and the resulting triple back flip (right).

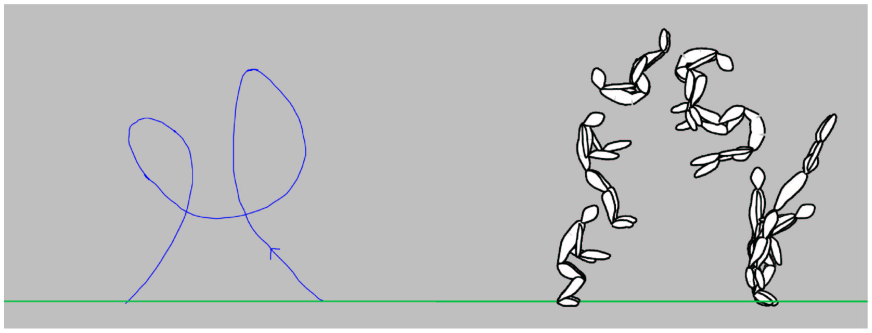

Figure 6.8: The Backwards-traveling Front Flip: Up loop gesture drawn right to left (left) and the resulting backwards back flip (right).

6.1.3 Walking and Variations

Figures 6.9 through 6.12 illustrate the basic walking motion and its variations. The first three types of walking – normal, tiptoeing and stomping – are all identified by the arc gesture. The location of the peak of the arc discriminates between the three styles. For a tiptoe, the arc is shifted towards the beginning of the step and for a stomp it is shifted towards the end of a step. For a normal walk it should be placed near the middle.

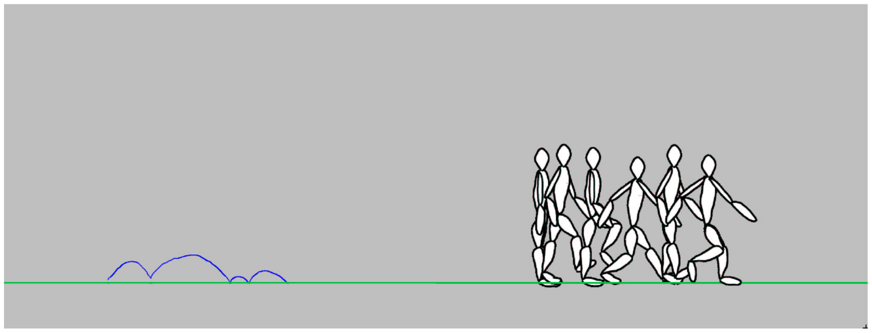

Figure 6.9: A Walk: A sequence of arc gestures (left) and the resulting walking motion (right).

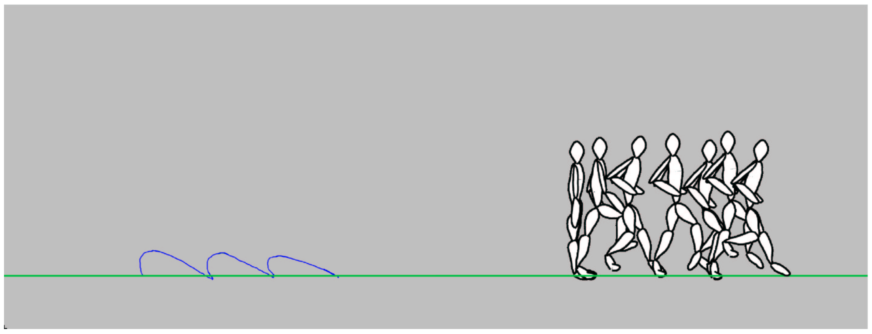

Figure 6.10: A Tiptoe: A sequence of arc gestures (left) and the resulting tiptoe motion (right).

As seen in Figure 6.12, the stiff-legged walk is specified by the spike gesture.

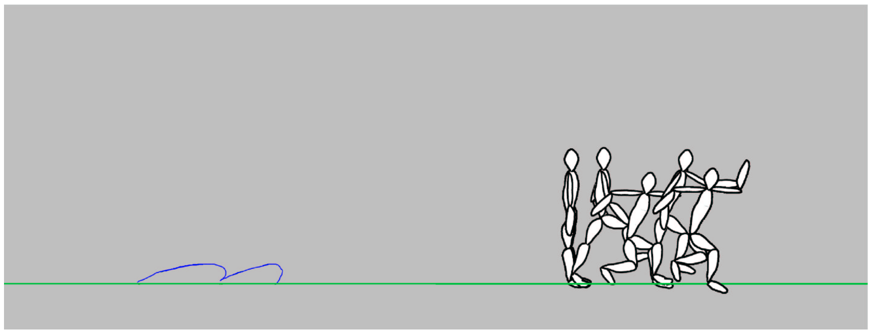
Figure 6.11: A Stomp: A sequence of arc gestures (left) and the resulting stomping motion (right).

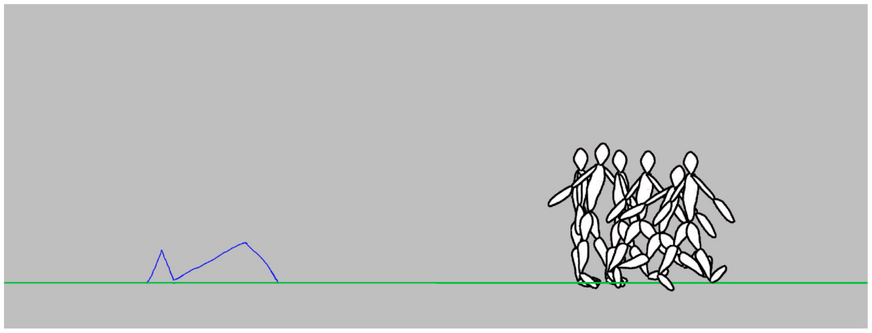
Figure 6.12: A Stiff-legged Walk: Sequence of spike gestures (left) and the resulting stiff-legged walking motion (right).

6.1.4 Moonwalking and Shuffling

These two motions are similar in that they both involve dragging the feet along the ground. Figure 6.13 through 6.15 illustrates the moonwalk, shuffling and sliding, respectively.

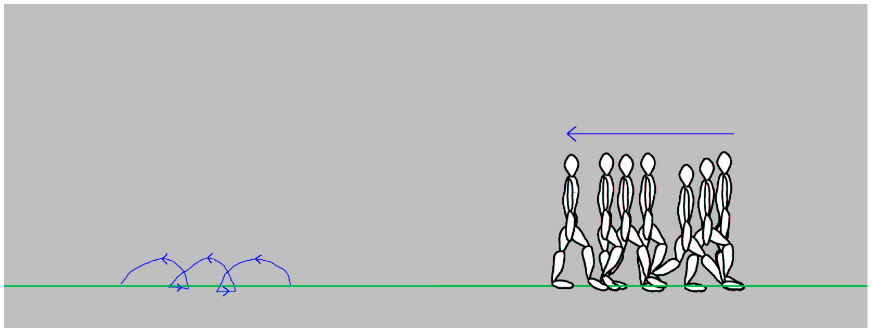

Figure 6.13: The Moonwalk: Arc-line-arc gesture sequence (left) and the resulting moonwalk motion (right).

The moonwalk is unique from other motions in that it can only be performed moving backwards. It is specified by an arc-line-arc sequence as shown above.

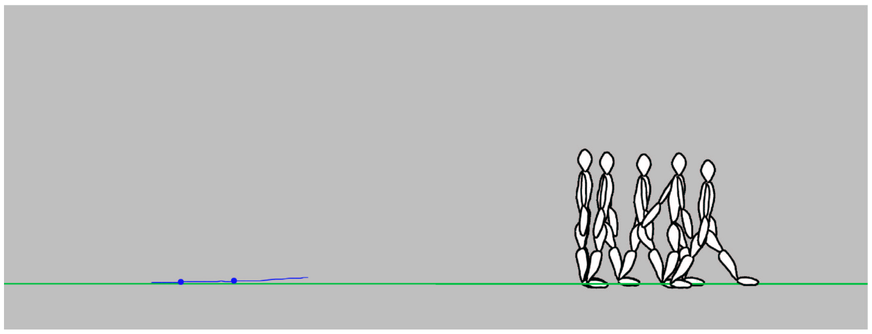

Figure 6.14: The Two-foot Shuffle: Line gestures (left) and the resulting shuffling motion (right). The two dark dots indicate points at which the stylus motion has paused.

Both shuffling and sliding are indicated by horizontal line gestures. Pauses in mouse motion delimit different shuffling or sliding steps. A slide is distinguished

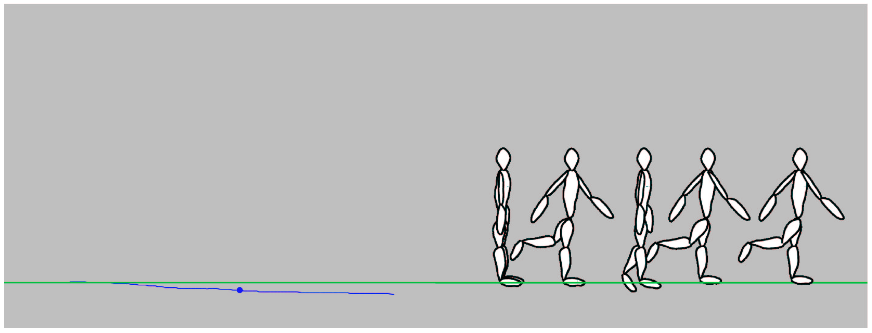

Figure 6.15: Sliding or Skating: Line gestures (left) and the resulting sliding motion (right). The dark dot marks a point at which the stylus motion has paused.

from a shuffle by its length; a line longer than the character's maximum stride length becomes a slide, otherwise it is a shuffle.

6.1.5 Combined Motions

Figures 6.16 through 6.19 show a series of combined motions. Figure 6.16 shows a combination normal steps and shuffling and Figure 6.17 shows a combination of normal steps interspersed with stiff-legged steps.

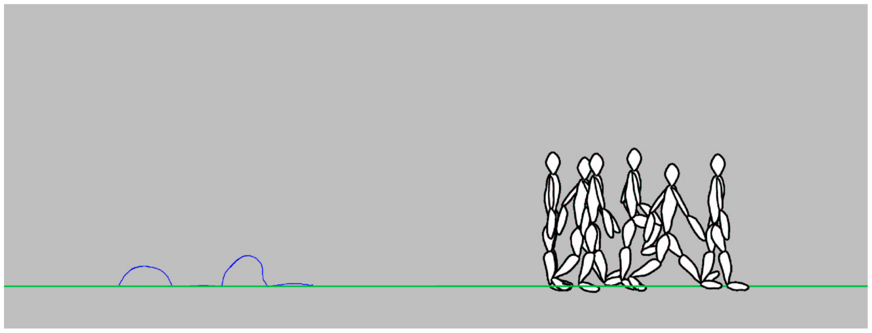

Figure 6.16: Shuffle-step: Arc-line gesture sequence (left) and the resulting motion (right).

Figure 6.18 shows a sequence of step, leap, flip, shufle, jump and Figure 6.19

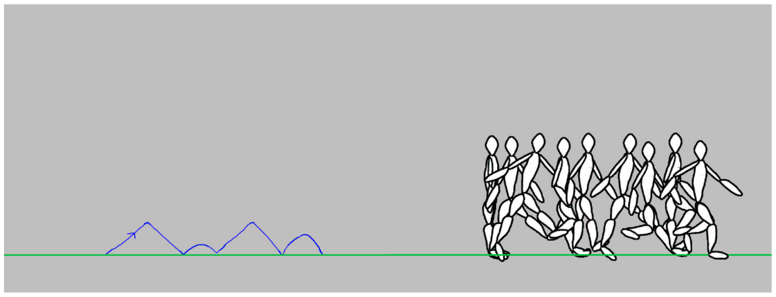

Figure 6.17: Stiff-leg: Spike-arc gesture sequence (left) and the resulting motion (right).

shows a sequence (right-to-left) of stiff-legged step, normal step, moonwalk, flip.

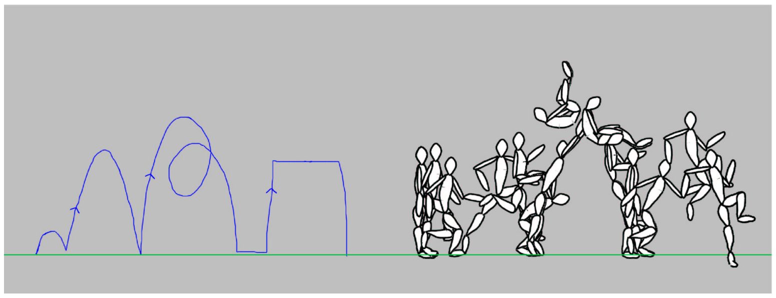

Figure 6.18: Step, leap, front-flip shuffle, and hop: Sequence of gestures (left) and the resulting motion (right).

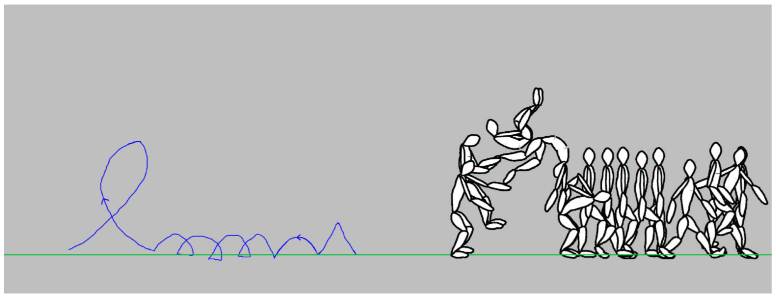

Figure 6.19: Two backwards steps, moonwalk, backwards-traveling front-flip: Sequence of gestures (left) and the resulting motion (right).

6.2 3D System

The motions available in the 3D system are illustrated below.

6.2.1 Jumping and Leaping

Jumping and leaping in 3D are shown in Figures 6.20 and 6.21 respectively. As in the 2D system, a leap must be preceded by a step.

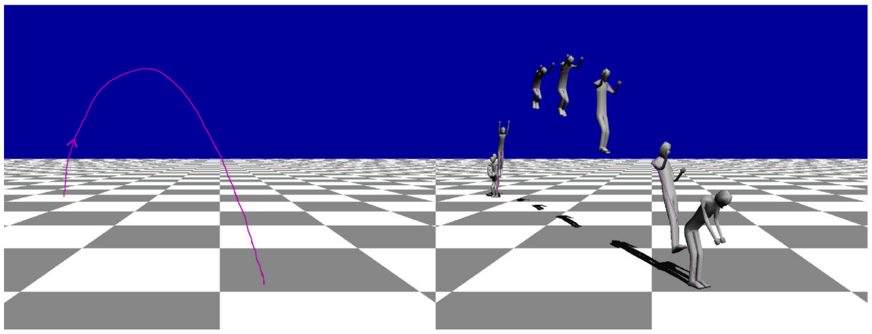

Figure 6.20: 3D Jump: Arc gesture (left) and the resulting jump motion (right).

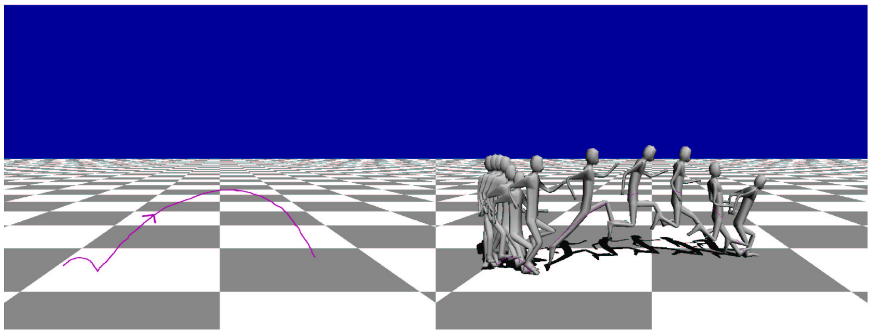

Figure 6.21: 3D Leap: Arc gestures (left) and the resulting step and leap motion (right).

6.2.2 Front and Back Flips

The front and back flips are illustrated in figures 6.22 and 6.23 respectively. Unlike in the 2D system, the direction in which the gestures are drawn does not affect its interpretation – the flips are assumed to always be traveling forward.

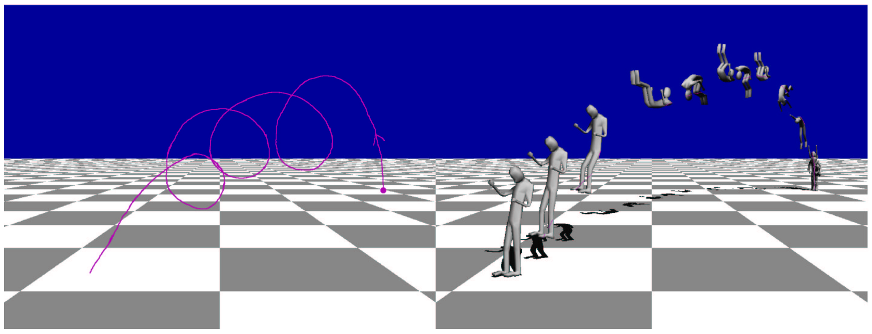

Figure 6.22: 3D Front Flip: Input sketch (left) containing three loops and the resulting triple front flip (right)

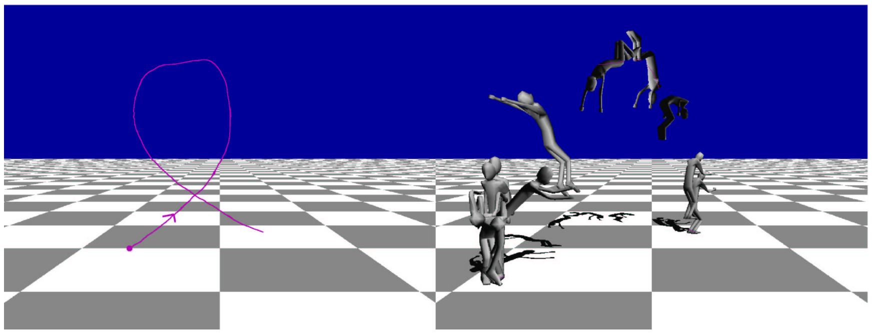

Figure 6.23: 3D Back Flip: Input sketch (left) and the resulting back flip (right)

6.2.3 Walking

Figure 6.24 illustrates two walking sequences, each containing three steps. The sequence on the left shows a more gradually curved path. The sequence on the right shows a path with sharper turns.

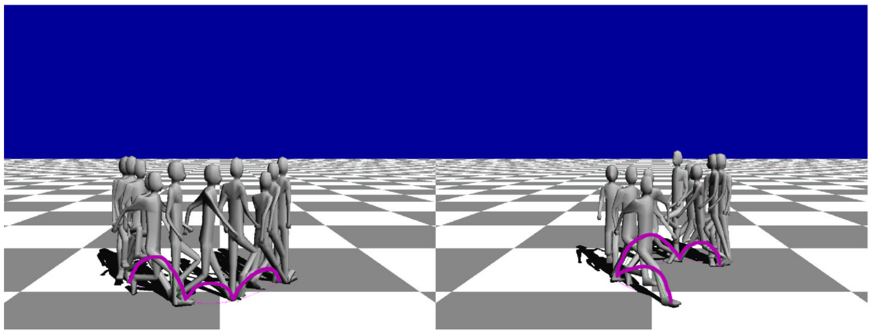

Figure 6.24: 3D Walking: Two three-step sequences.

6.2.4 Combined Motions

Figures 6.25 and 6.26 illustrate sequences of combined motions in 3D.

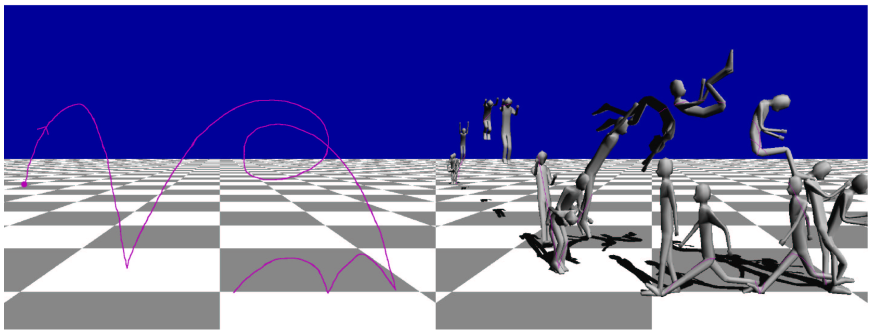

Figure 6.25: Jump, front-flip and two steps: Sequence of gestures (left) and resulting motion (right).

Figure 6.25 illustrates a sequence consisting of a jump, front flip and a couple of steps. Figure 6.26 illustrates a double back flip, jump, two steps and a leap.

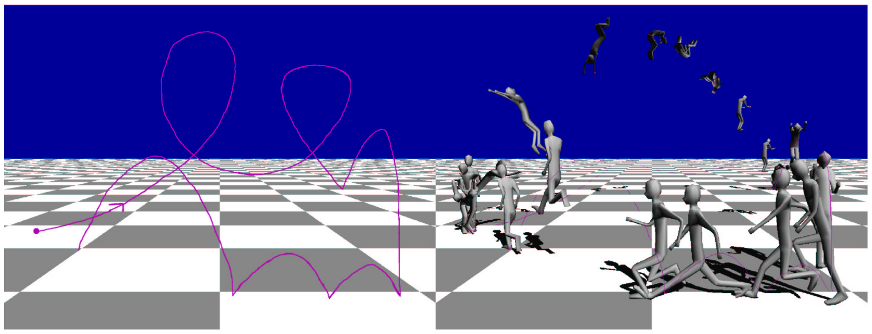

Figure 6.26: Double back-flip, jump, two steps and a leap: Sequence of gestures (left) and the resulting motion (right).

6.3 Discussion

The animation system allows for animations to be produced very quickly by both experienced and novice users alike, opening up animation to a new audience of potential animators. A variety of motions and variations can be produced as has been illustrated in this chapter. Animations can also be produced in both 2D and 3D using the same interface, although due to limitations inherent in using a 2D input device in a 3D environment, the 3D version of the system has a smaller repertoire of motions.

Although the results presented in this chapter show a normally proportioned character, we can also create and animate characters having strange proportions through the use of the character sketching front-end [Tho2003]. Figure 6.27 shows an example of a character with strangely proportioned legs. The animation produced is still reasonable.

The main weakness of the system as currently implemented lies in the sketch analysis algorithm. Corner detection is still less reliable than it should be which leads to some problems in correctly identifying hat gestures or the moonwalk gesture sequence, especially when drawn quickly.

The system expects to receive an input sketch composed of a sequence of

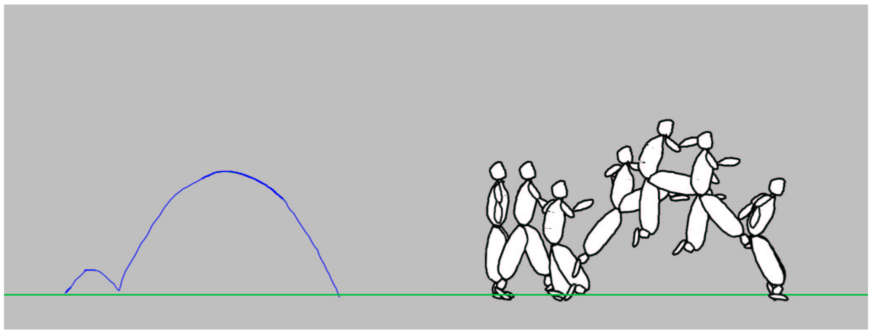

Figure 6.27: Strange Leg Proportions: An example of a character with very long upper legs and very short lower legs. The sketch (left) produces the animation on the right.

recognizable gestures. If it is given a random scribble as input the system does its best to make some sense out of it, an example of which is shown in Figure 6.28. Generally this results in some random sequence of jumps and flips. The system does occassionally break when faced with nonsensical input.

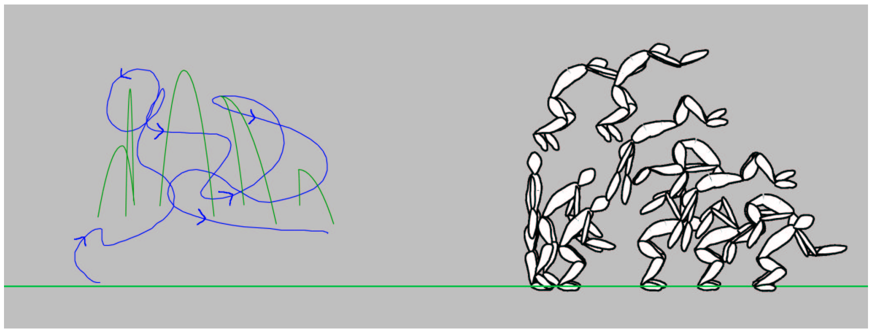

Figure 6.28: Random Scribbles: A random scribble as input (left) with the parabolae that the system has attempted to fit to the sketch. The results (right) showing three jumps and two flips. Note that the first flip changes direction in mid-air.

Chapter 7

Conclusions

The current implementation of our animation system provides a starting point for further exploration of sketch-based animation systems and strives to make animation accessible to a wider audience of potential animators. As with the first iteration of any concept, ours system has a number of limitations and areas for further work.

7.1 Limitations

The current repertoire of motions in the system and the level of control provided over those motions is limited. While adding new gestures and motions is a fairly straightforward process, it does require a significant amount of hand-tuning, particularly in the control scheme used by new motion controllers. Much of this could likely be simplified through the use of scripting for controllers and other techniques for gesture recognition.

While the sketch and gesture analysis has been made as flexible as possible to facilitate the addition of new gesture and motion types, there is still more work that can be done in this area, particularly in integrating a statistically-based gesture recognition algorithm. The gesture recognition technique described in this thesis relies heavily on hand tuned values, most notably those used in detecting corners in the input sketch as this is where the current algorithm is most likely to fail. The use of a statistical based algorithm could avoid much of this hand tuning, although selecting an appropriate set of features to reliably distinguish between gestures such as required by [Rub1991] becomes an issue. Further, such schemes interpret single strokes or a group of strokes (such as in [Cal2002]) as a single gesture or object, whereas our system extracts multiple gestures from a single stroke of the stylus and thus a method for dividing a sketch into its component gestures is still required.

The 3D prototype currently only implements a subset of the motions available in the 2D system. The limitations of the 3D system are due in large part to the ambiguities inherent in using a 2D input device to control 3D motion.

Both the 2D and 3D systems are presently limited to bipedal human characters. It would be interesting to extend our system to quadrupeds such as cats and horses or other types of bipeds such as birds or bipedal robots in the "mech warrior" or Star Wars styles.

7.2 Future Work

Most of the suggestions for future work attempt to address the limitations and shortcomings of the system. An immediate way to extend the system is through the addition of new motion types. There are many opportunites to extend the repertoire of motions in Motion Doodles. Actions from everyday life, or more specialized areas such as gymnastics, dancing, skating, karate, trampolining and other types of choreography provide many examples that could be used to enrich the vocabulary of the system. An example of such a motion is a twist or lateral spin which occurs in gymnastic routines.

We have built the system on a single set of keyframes for each motion. The ability to easily edit new sets of keyframes and use them in place of the default set would provide the ability to change the style of motion produced by the system.

Presently, the environment is static, with flat terrain and serves only as a backdrop on which a character moves around. The ability to manipulate objects – such as picking up and moving objects or opening doors – in an environment is another important area of future work.

In connection to environment interactions, the ability to interact with rugged, non-flat terrain would be an interesting extension of the current system.

The 3D prototype could be extended in many of the same ways as the 2D system, though the ambiguities discussed in Chapter 5 would need to be addressed. Testing with users to determine which assumptions are best and making use of domain dependent assumptions – limiting the system to a particular class of motions such as those found in gymnastic routines – would help to resolve some of these issues.

As discussed in Chapter 5, using a 2D input device to control animation in a 3D environment introduces several ambiguities. Using a 3D input device for 3D sketching, such as a Phantom or Polhemus tracker could solve some of the ambiguities in the current 3D system, although using expensive or exotic input devices would reduce the potential user base. Future camera-based tracking devices would also provide more complex 3D input geared towards a performance animation based version of the system.

The current system only handles bipedal human characters but there is potential in extending the system to other non-human types of characters such as quadrupeds or other types of bipeds such as birds.

Finally, the ability to control and synchronize multiple characters would allow a user to choreograph much more complex scenes.

References

[Bal1995] Balaguer, J. and Gobbetti, E. 1995. Sketching 3D animations. *Computer Graphics Forum 14*, 3, 241-258.

[Cal2002] Calhoun, C., Stahovich, T. F., Kurtoglu, T., and Kara, L. B. 2002. Recognizing Multi-Stroke Symbols.

[Che1999] Chetverikov, D., and Szabó, Z. 1999. *Detection of High Curvature Points in Planar Curves.* http://www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/ CHETVERIKOV2/cornerweb.html

[Don2003] Dontcheva, M., Yngve, G. and Popović, Z. 2003. Layered Acting For Character Animation. In *Proceedings of SIGGRAPH 2003*, ACM SIGGRAPH.

[Fel1995] Fels, S., and Hinton, G. 1995. Glove-talk ii: An adaptive gesture-to-format interface. In *Proceedings of CHI'95: Human Factors in Computing Systems*, ACM Press, 456-463.

[Fon2000] Fonseca, M. J., and Jorge, J. A. 2000. Using Fuzzy Logic to Recognize Geometric Shapes Interactively. In *Proceedings of the 9th Int. Conference on Fuzzy Systems*, FUZZ-IEEE'00, San Antonia, USA, 191-196.

[Gir1987] Girard, M. 1987. Interactive design of computer-animated legged animal motion. *IEEE Computer Graphics and Applications 7*, 6, 39-51.

[Hon2000] Hong, P., Turk, M. and Huang, T. S. 2000. Gesture Modeling and Recognition Using Finite State Machines. In *Proceedings of IEEE Conference on Face and Gesture Recognition*, Grenoble, France.

[Hut1987] Hutchinson, A. *Labanotation: The System of Analyzing and Recording Movement*. Routledge, 1987.

[Iga1999] Igarashi, T. Matsuoka, S., and Tanaka, H. 1999. Teddy: A sketching interface for 3d freeform design. In *Proceedings of SIGGRAPH 99*, ACM SIGGRAPH / Addison Wesley Longman, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, 409-416.

[Las2000] Laszlo, J., van de Panne, M. and Fiume, E. L. 2000. Interactive control for physically-based animation. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, Computer Graphics Proceedings, Annual Conference Series, 201-208.

[Muy1955] Muybridge, E. *The Human Figure in Motion* Dover Publications, Inc., 1955.

[Oor2002] Oore, S., Terzopoulos, D., and Hinton, G. 2002. A Desktop Input Device and Interface for Interactive 3D Character Animation. In *Proc. Graphics Interface*, 133-140.

[Pop2001] Popović, J. 2001. *Interactive Design of Rigid-Body Simulations for Computer Animation*. PhD thesis, Carnegie Mellon University.

[Rub1991] Rubine, R. 1991. Specifying Gestures by Example. *Computer Graphics 25*, 4 (July), 329-337.

[Sez2001] Sezgin, T. M., Stahovich, T., and Davis, R. 2001. Sketch Based Interfaces: Early Processing for Sketch Understanding.

[Stu1998] Sturman, D. J. 1998. Computer Puppetry. *IEEE Computer Graphics and Applications 18*, 1 (Jan-Feb), 38-45.

[Sut1963] Sutherland, I. E. 1963. Sketchpad: A man-machine graphical communication system. In *Proceedings AFIPS Spring Joint Computer Conference*, vol. 23, 329-346.

[Tho2003] Thorne, M., Burke, D., and van de Panne, M. 2003. Motion Doodles: A Sketching Interface for Character Animation. Technical Report, TR-2003-09, University of British Columbia.

[van1997] van de Panne, M. 1997. From footprints to animation. *Computer Graphics Forum 16*, 4, 211-224.

[Wel1993] Welman, C. *Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation.* Masters Thesis, Simon Fraser University.

[Wil2001] Williams, R. *The Animator's Survival Kit.* Faber and Faber.

[Zel1996] Zeleznik, R. C., Herndon, K. P., and Hughes, J. F. 1996. Sketch: An interface for sketching 3d scenes. In *Proceedings of SIGGRAPH 96*, ACM SIGGRAPH / Addison Wesley, New Orleans, Louisiana, Computer Graphics Proceedings, Annual Conference Series, 163-170.

Appendix A

Controller Details

Each motion is broken down into a series of states as outlined in table 4.3. Internally, each controller functions as a finite state machine.

In general, there are two mechanisms which trigger a state change: (1) contact between a foot and the ground, and (2) timing data. During any state in which contact between the foot and ground occur, inverse kinematics is used on the leg or legs involved in the contact. The joint angles of the underlying keyframe database is used to drive all joints of the character that are not under the control of inverse kinematics.

A.1 Jump, Leap, Hop

The jumping motion and its variations (leaping, hopping and the more energetic stomping jump) are all governed by the same controller. The state transition diagram for this controller is shown in Figure A.1.

The jump and leap actions are both specified using an arc input gesture while the hop action corresponds to the square wave-like hat gesture and the stomping jump is a spike (see Figure 4.6). While the leap and jump both share the same input gesture they are distinguished by looking at the separation of the character's feet at the start of the action. If the character's feet are together then a jump is performed, but if the feet are separated such as would occur after a step, a leap occurs instead. Regardless of the action, the same information is

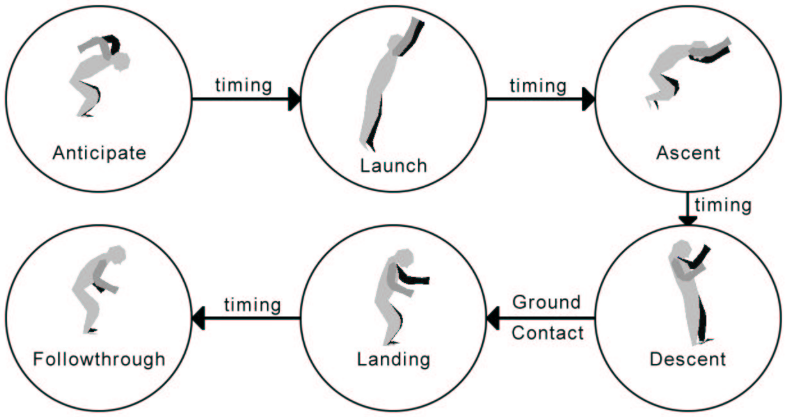

Figure A.1: Jump controller state transition.

taken from each of the arc, spike and hat gestures: the start and end points, and the maximum y-value within the gesture. If the maximum point does not lie between the start and end points (when projected onto the ground plane) it is repositioned to be half-way between the other two points.

Once extracted, two half parabolae are fit to the start, maximum (or apex) and end points, one for ascent and one for descent, as documented in Figure 4.9. The slope at the start and end of these two half parabolae give the angle of the character's body at the end of the `LAUNCH` and `DESCENT` states respectively.

We derive a separate centre-of-mass curve from the fit parabolae which shares the same maximum point. This is because the sketch indicates where the foot (or feet) is to be placed at the start and end of the jump, but it is the centre-of-mass which follows the parabolic flight path, not the feet. At the end of the `LAUNCH` state we create the CoM curve such that the start point is at the location of the centre-of-mass at the point of time where the character is just about to take off of the ground. At this point we also check the apex point to make sure that it is above the new starting point, otherwise the character would appear to

State	Transition	Notes
ANTICIPATE	Timing	Legs controlled by IK
LAUNCH	Timing	No special processing
ASCENT	Timing	Centre-of-mass (CoM) follows parabolic arc from its position at the start of the state to the apex of the sketched arc at the end of the state
DESCENT	Ground contact	Project where CoM must be to put feet at end of arc at ground contact and have CoM follow parabolic arc from apex to that point
LANDING	Timing	Legs controlled by IK as CoM continues downward, maintaining ground contact velocity; degree of crouch modified according to height/distance of jump
FOLLOWTHROUGH	Timing	Same as LANDING

Table A.1: Jump Controller states.

jump in the wrong vertical direction. If the apex lies below the starting height of the CoM then the target apex is readjusted upwards so that it will be above the CoM's starting height. At the beginning of the `DESCENT` state we must also determine where the centre-of-mass needs to be at the end of `DESCENT` so that the foot (or feet) will land at the point indicated in the sketch.

Table A.1 summarizes the details of each state and the special processing that occurs in each state and on state transitions.

A.2 Flipping

The front and back flips were implemented as two separate controllers, although the structure of both is similar. The state transition diagram is shown in Figure A.2.

The flip controller is similar to the jump controller. Indeed, jumping could be viewed as a flip with no loops. The loops, or spins, that the character performs is

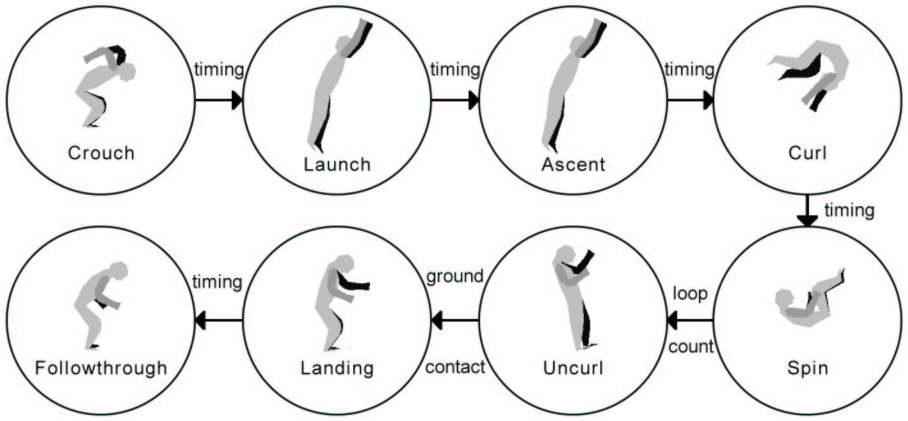

Figure A.2: Flip controller state transition.

what differentiates a flip from a normal jump. The time taken for a character to do one flip is the average time, per loop, taken to draw all the loops in the loop gesture. This is done to prevent abrupt changes in the angular velocity of the character. Also because of the loops, unlike jumps, flips have no specific `ASCENT` or `DESCENT` states since travelling along the upward and downward half of the arcs is divided among the `ASCENT`, `CURL`, `SPIN` and `UNCURL` states. Whether the character is ascending or descending depends on which half of the overall motion the character is currently being performed. During the first half the character ascends and during the second half it descends.

The two half parabolae tracked by the character's centre-of-mass are created in different ways for the flips and for the jump. Because a single flip may use several loop gestures we find the maximum point over all the loops and use that as the maximum for the parabolae. The first point of the first loop and last point of the last loop provide the other two points for forming the parabolae. Figure A.3 show some examples. As with the jump, the start and end point of the fit parabolae indicate the placement of the feet and the centre-of-mass curve is created from these parabolae as described for the jump.

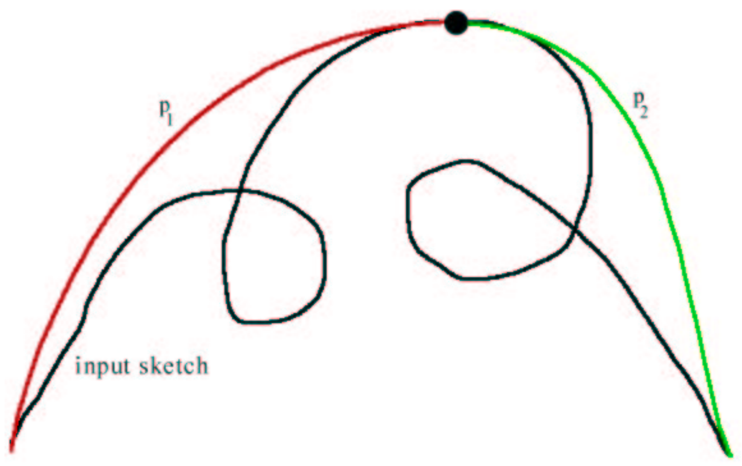

Figure A.3: Example of flip parabolae. The two half parabolae, p_1 and p_2, are labelled and the apex point is marked by a circle.

Table A.2 summarizes the details of each state and the special processing that occurs in each state and on state transitions.

A.3 Walking

The walking controller is responsible for walking, tiptoeing and stomping. The stiff-legged walk was implemented using a separate controller which has a similar structure. Figure A.4 shows the state transition diagram.

The controller has eight states, consisting of two sets of four states which have left-right symmetry. While the keyframes for each leg in our system are mirror images of those for the opposite leg, this need not be the case if an asymmetrical gait were desired. Under the current implementation, asymmetrical gaits can be produced by mixing the different walk styles with each other and with the

State	Transition	Notes
CROUCH	Timing	Same as for jumping
LAUNCH	Timing	Same as for jumping
ASCENT	Timing	Same as for jumping, but only traverses part of the upward parabola
CURL	Timing	CoM continues traversing upward arc as character curls into a ball
SPIN	Number of Loops	CoM finishes traversing upward arc and switches to downward arc; does same projection for landing as with jumping (at the point where the switch between the upward and downward arcs occur); remains in SPIN state until all loops indicated in the sketch have been performed
UNCURL	Ground contact	CoM finished traversing the downward arc as the character straightens out
LANDING	Timing	Same as for jumping
FOLLOWTHROUGH	Timing	Same as for jumping

Table A.2: Flip Controller states.

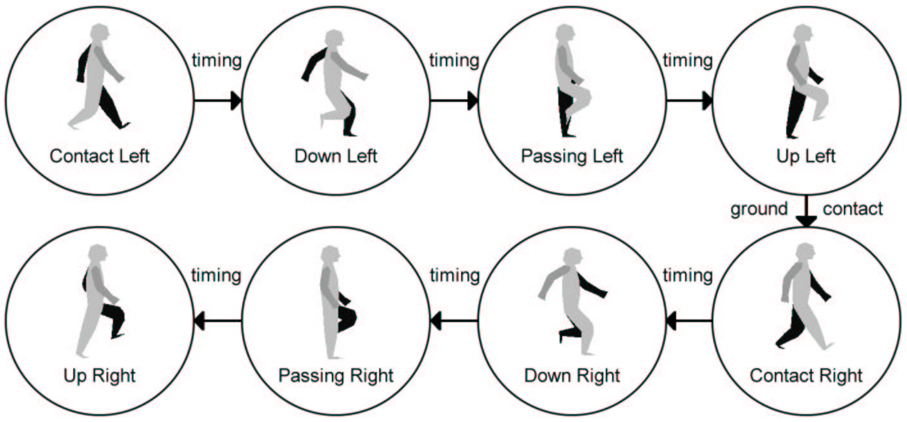

Figure A.4: Walk controller state transition.

shuffling motion, as described in section A.4. Examples of such mixed gaits are illustrated in section 6.1.5.

A sequence of arcs produces a series of steps/tiptoes/stomps. The placement of the apex of each arc determines which style is used for a given step. An apex occuring close to the start of a step indicates a tiptoe. If the apex occurs close to the end of the step it indicates a stomp and otherwise it is a normal step. These differences are illustrated in figure A.5

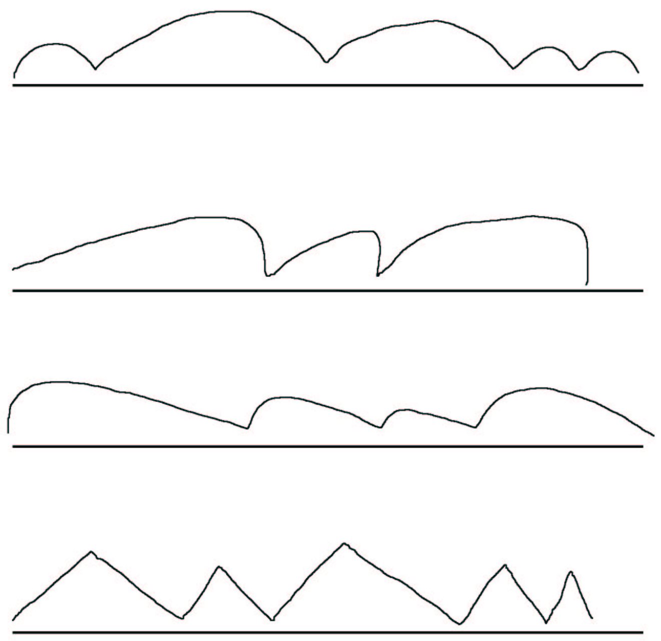

Figure A.5: Variations in walk styles: Gestures (drawn left-to-right) for normal walk (top), stomping (2nd line), tiptoeing (3rd line) and the stiff-legged walk (bottom).

The end point of an arc indicates where the swing foot (the foot being controlled by that arc) should be placed at the end of the step. The height of the arc indicates how high the foot should be lifted off of the ground at mid-stance. The starting point of an arc does not normally coincide with the location of the swing foot at the start of the step so the arc as drawn is, in essence, stretched so that the positions will match and it is this stretched version of the arc which is used to guide the swing foot during a step. This is illustrated by Figure A.6.

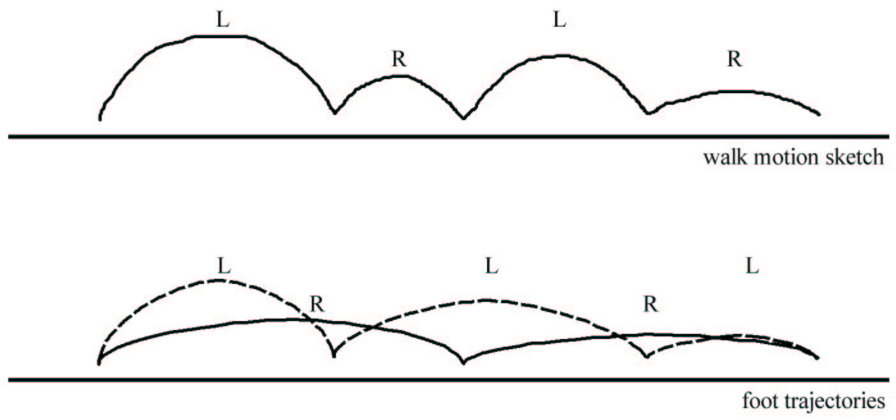

Figure A.6: Foot Trajectories: the top shows a sequence of four arcs representing four walking steps. The bottom shows how the drawn arcs are converted to foot trajectories. Arcs are labelled as corresponding to the left (L) or right (R) foot. The character is assumed to start at rest and end at rest after taking the four steps.

The positions of both feet are specified during a step: the support foot does not move and the swing foot is guided by an arc derived from the input sketch. The hips, however, still needs to be placed. In our system, the hips are placed to be above the midpoint of the two feet. The vertical placement is then constrained by the support leg which has its pose determined by the walk keyframes.

The stiff-legged walk is specified by a sequence of spike gestures. As both legs are kept straight in this version of the walk, the height of a spike does not indicate how high the swing foot should be lifted off of the ground, although

State	**Transition**	**Notes**
CONTACT LEFT/RIGHT	Ground Contact	Support leg kept straight (if possible) with IK done on each leg as needed; swap support and swing legs at end of state
DOWN LEFT/RIGHT	Timing	Support leg allowed to follow keyframes, swing leg controlled by IK
PASSING LEFT/RIGHT	Timing	Same as with DOWN LEFT/RIGHT
UP LEFT/RIGHT	Timing	Same as with CONTACT LEFT/RIGHT

Table A.3: Walk Controller states.

this could be done to yield some highly exaggerated motions. Instead, the swing leg is made to lie on the ray from the character's hip to the point where the foot would be for one of the other walk variations.

Table A.3 summarizes the details of each state and special processing the occurs in each state and on state transitions. The names for these states were taken from [Wil2001].

A.4 Moonwalking and Shuffling

The moonwalk and shuffle use separate controllers but are structured similarly. The state transition diagram for these controllers a shown in Figure A.7.

Both the moonwalk and shuffling motion involve sliding both feet along the ground so the only points that need to be extracted from the input sketch are the start and end points of the gesture or gestures marking a single moonwalk step or shuffle. The moonwalk is a highly stylized type of motion and is only done moving backwards. A moonwalk consists of the following actions:

1. Right leg bent, heel up with toe on ground, left foot flat on ground

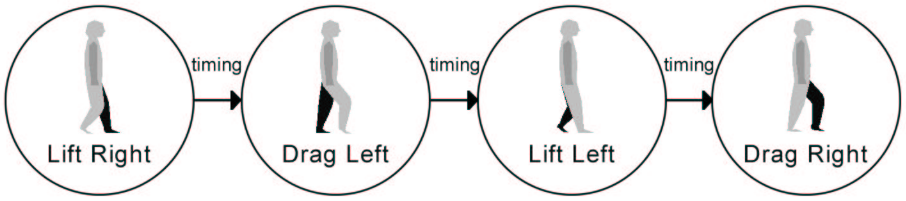

Figure A.7: Moonwalk/shuffle controller state transition. In the lift left and lift right states the heel is lifted of the ground and in the drag left and drag right states the left or right leg respectively is dragged backwards along the ground.

2. Drag left foot backwards, right foot fixed
3. Straighten the right leg, putting heel on floor, put left leg in the right leg's former pose
4. Drag right foot backwards, left foot fixed
5. Repeat from 1

The `LIFT` states correspond to steps 1 and 3 where we position each leg using IK. Neither foot moves at this time. Steps 2 and 4 are handled by the `DRAG` states where the leg being dragged is kept straight and the other leg is controlled through IK.

Shuffling is handled similarly to moonwalking except that neither heel is lifted from the ground. Instead, foot sliding occurs in both `LIFT` and `DRAG` states. Also, shuffling may proceed either forwards or backwards.

Table A.4 summarizes the details of each state and special processing that occurs in each state and on state transitions.

State	Transition	Notes
LIFT LEFT/RIGHT	Timing	Heel of support leg is lifted off the ground (moonwalk only); IK applied to both legs
DRAG LEFT/RIGHT	Timing	Leg being dragged is kept unbent with support leg being controlled by IK

Table A.4: Moonwalking and Shuffling Controller states.

A.5 Sliding

The sliding motion is similar to shuffling, but different enough to warrant its own controller. The state transition diagram is shown in Figure A.8.

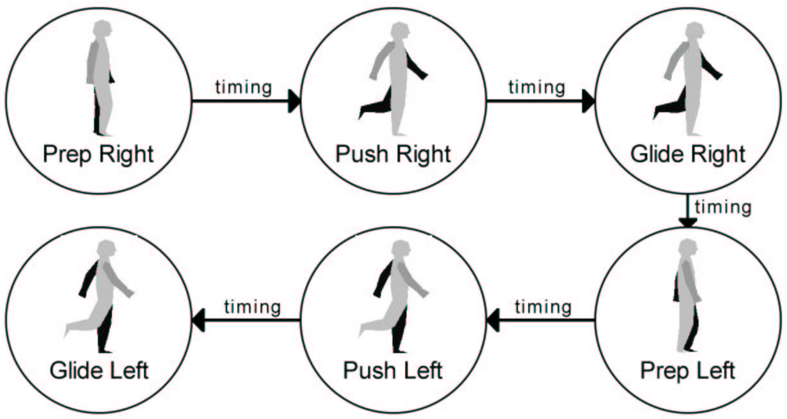

Figure A.8: Slide controller state transition.

Sliding resembles a skating style of action where one glides along on one foot with the other foot raised behind the body. The left and right `PREP` states (short for preparation) allow for the support to shift from one leg to the other. The left and right `PUSH` states are where the character pushes off and starts to glide, raising the non-support leg to the bent, gliding position. Finally, during the `GLIDE` states the character maintains its pose, sliding along the ground for the remainder of the action.

Table A.5 summarizes the details of each state and special processing that occurs in each state and on state transitions.

State	**Transition**	**Notes**
PREP LEFT/RIGHT	Timing	Switch support from one leg to the other
PUSH LEFT/RIGHT	Timing	Begin sliding as non-support leg is being raised
GLIDE LEFT/RIGHT	Timing	Maintain pose while sliding

Table A.5: Slide Controller states.