Chapter 5

Registration Curves

Chapter 4 showed how new motions can be synthesized by building graph structures that facilitate the rearrangement and attachment of short segments of captured motion. While this is a natural strategy for controlling the sequencing of different actions, it breaks down when one wants individual actions that are not present in the original data set. Imagine, for example, that one has collected a data set containing several motions of a character reaching to different positions on a shelf, with the goal of being able to animate new reaching motions. Clearly, graph-based methods are of no avail, since there is no sensible way of rearranging captured reaches to form new ones. Similarly, while one can edit a captured reach to reposition the hand (e.g., using inverse kinematics methods such as the one described in Chapter 3), subtle but important correlations between the movement of different parts of the body may be lost if the adjustment is too large — even if the character's hand ends up in the right place, the motion may look awkward or unnatural. More generally, existing motion editing methods [15, 98, 31, 54] provide no simple way of synthesizing the detail present in captured motions; indeed, they are specifically designed to adjust motions *without* changing the finer details.

An alternative to rearranging or editing captured motions is to blend them [96, 76]. Motion blending combines captured examples according to weight functions that specify how much influence each example has at each point in time. Different weight functions yield different synthesis operations; see Figure 5.1. A transition between two motions can be created by initially placing all of the weight on one motion and then smoothly shifting it to the other motion; this technique

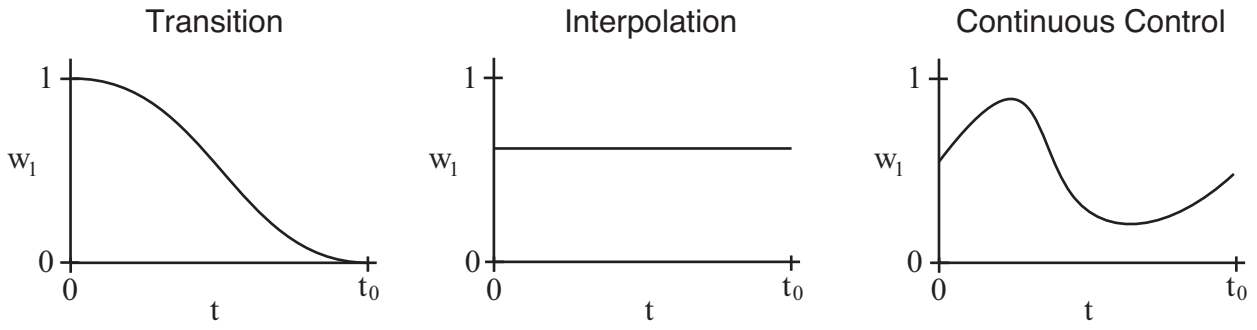

Figure 5.1: Different weight functions yield different blending operations. These graphs depict weight functions $\mathbf{w}(t) = (w_1(t), w_2(t))$ that might be used for transitioning, interpolation, and continuous control. In each case $w_2(t) = 1 - w_1(t)$.

was used in Chapter 4. An interpolation of the input motions can be created by holding the blend weights constant. This is useful for creating intermediary actions, such as a reach that has a target location in between those of the captured examples. Finally, continuous control can be gained by continually varying the blend weights. For example, if the inputs are various kinds of locomotion, then one could adjust the blend weights to interactively control the speed and direction at which a character travels. Blending operations such as these have considerable practical importance and have proven useful in commercial applications like video games [62, 88].

While blending is a powerful technique, it clearly will not produce realistic results on arbitrary sets of input motions. Instead, these inputs must be chosen with some care, and the range of motions that can be successfully blended depends on how much information is given to the blending algorithm. In this dissertation we are interested in *automatic* motion blending, where nothing is known about the input motions other than the root position, joint orientations, and constraint annotations at each frame. This chapter shows how many of the limitations of existing automatic blending algorithms can be avoided by building *registration curves*, which are data structures that encapsulate relationships involving the timing, local coordinate frame, and constraint states of an arbitrary number of input motions. While some existing blending algorithms use similar timing and constraint (but not coordinate frame) relationships [76, 68], they require a user to supply this information directly through special annotations added to the input motions. Registration curves

can be built and used without user intervention, both automating these manual algorithms and expanding the range of motion that can be blended successfully.

The remainder of this chapter describes how to construct and use registration curves. First, Section 5.1 provides an overview of registration curves. Sections 5.2 and 5.3 then explain how to build registration curves and use them to create blends. Finally, Section 5.4 demonstrate registration curves in common blending applications, and Section 5.5 concludes with a discussion of the advantages and limitations of our methods.

5.1 Overview

A blend $\mathbf{B}(t)$ is a motion constructed from n input motions $\mathbf{M_1}, \ldots, \mathbf{M_n}$ and an n-dimensional weight function $\mathbf{w}(t)$. While the specific sequence of skeletal poses in $\mathbf{B}(t)$ depends on the blending algorithm, if $w_i = 1$ at t_0 and all other weights are 0, then $\mathbf{B}(t_0)$ is identical to a frame from $\mathbf{M_i}$. Blend weights usually sum to 1 and are often assumed to be non-negative, but neither of these properties are required [76].

When the only available information is the input motions themselves, a standard blending method is *linear blending*. The i^{th} frame of a linear blend is a weighted average of the skeletal parameters at the i^{th} frame of each input motion. The blended root position is calculated by averaging either the input root positions or the input root velocities, and blended joint angles can be calculated in a variety of ways, including averaging Euler angles [76] and computing the exponential map of averaged logarithmic maps [68]. Constraints are not handled by linear blending.

Linear blending produces reasonable results when it is used to create short blends of similar motions. Indeed, in Chapter 4 brief transition motions were created for motion graphs by applying a modification of linear blending that inferred constraints from the input motions. However, linear blending suffers from several problems that make it of limited use in more general circumstances. We propose an alternative blending algorithm that, as with linear blending, combines frames via averaging, but also automatically extracts information from the input motions to help decide which frames to combine, how to position and orient them prior to averaging, and what constraints should

exist on the result. This information is used to create a registration curve, which is a composite data structure consisting of a *timewarp curve*, an *alignment curve*, and a set of *constraint matches*. The remainder of this section provides an overview of what these elements are and why they are needed.

5.1.1 Timing

Logically similar motions can have different timing in the sense that corresponding events may occur at different absolute times. For example, successive heel strikes of a walk are spaced further apart in time than those of a run, and a jab will reach its apex sooner than a stronger, more committed punch. If linear blending is used to combine motions such as these, then frames from unrelated portions of the motions will be combined, which can create pronounced and disturbing artifacts (Figure 5.2). A solution is to timewarp the input motions so corresponding events occur simultaneously. This involves determining a timewarp curve $\mathbf{S}(u)$ where each point provides the indices of corresponding frames, one from each input motion (Figure 5.3). These frame indices may specify times in between data samples; when this occurs, the captured skeletal poses are interpolated by linearly interpolating root position data and applying spherical linear interpolation to joint orientation data. During blending, only corresponding frames are every combined — that is, the i^{th} blend frame is a composition of the input frames from some point $\mathbf{S}(u_i)$ on the timewarp curve.

The timewarp curve is assumed to be continuous and strictly increasing, so if $u_1 > u_0$, then $S_i(u_1) > S_i(u_0)$. As a result, given a frame at any time t in motion $\mathbf{M_i}$, a unique position $u_{S_i}(t)$ on the timewarp curve can be determined. The parameter u may be thought of as defining a reference time system: each input motion can be viewed as a realization of some canonical motion, and when this canonical motion is at frame u then $\mathbf{M_i}$ is at frame $S_i(u)$. Note that this timewarping may make sense *locally* even if the input motions themselves are quite different overall. For example, say that in $\mathbf{M_1}$ a character walks straight forward and opens a door, and in $\mathbf{M_2}$ it stumbles a few steps and falls into an open manhole. The walking portion of $\mathbf{M_1}$ and the stumbling portion of $\mathbf{M_2}$ may have

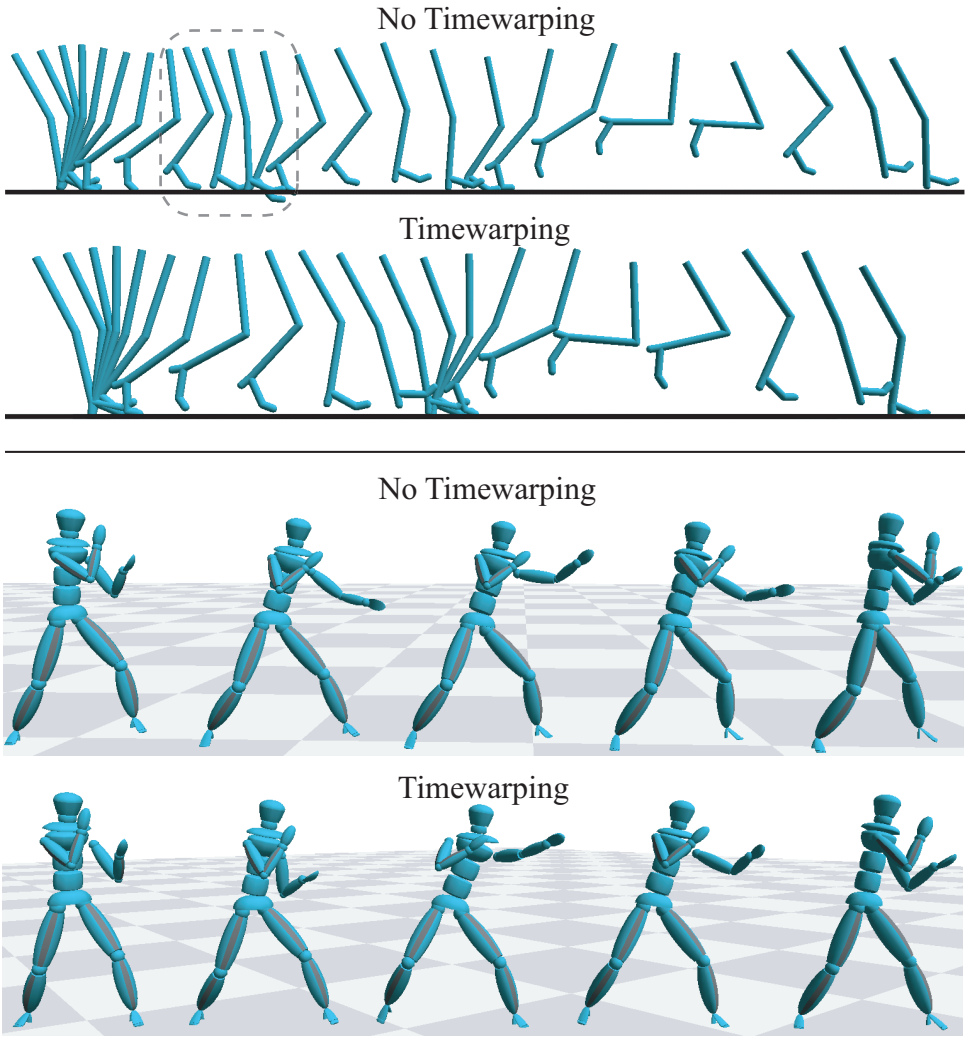

Figure 5.2: Top: A transition between walking and jogging that spans two locomotion cycles. For clarity, only the right leg is shown. Without timewarping, out-of-phase frames are combined and the character's leg stutters as if an extra step is being taken (see circled region). **Bottom:** An interpolation of a jab and a stronger, more committed punch. Without timewarping, the character's arm wobbles.

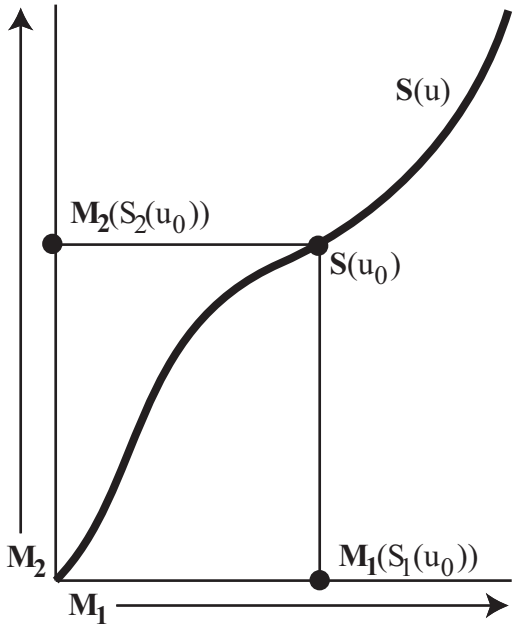

Figure 5.3: An example timewarp curve for two motions.

a perfectly reasonable timewarp curve even if the entire motions do not. Timewarping is therefore valid for inherently local blending operations like transitioning.

Since no information is present other than the motion data, $\mathbf{S}(u)$ is created using the heuristic that the more similar the skeletal poses are for a set of frames, the more likely it is that these frames correspond. This heuristic complements the way frames are combined during blending: it is more reasonable to average skeletal parameters if they are already similar, and thus motions are aligned in time so corresponding skeletal poses are as similar as possible. Section 5.2.1 provides details on how timewarp curves are constructed.

5.1.2 Coordinate Frame Alignment

Linear blending can fail even when frames that occur at the same time have very similar skeletal poses. Imagine using linear blending to create a halfway interpolation of two walking motions, one curving to the left and the other to the right. While one would intuitively expect the character to walk straight forward, the blended root path collapses because the input root positions are combined through linear interpolation (Figure 5.4). Also, the character suddenly flips around near

Figure 5.4: A halfway interpolation of two walking motions that curve in opposite directions. The lines show the projection of the root trajectory onto the ground at each frame, and the triangles show the root's position and orientation at selected frames. Linear blending causes the root trajectory to collapse. Note the sudden change in root orientation at the end of the motion, when the accumulated angle between the roots exceeds $180°$.

the end of the motion. This is because standard orientation averaging schemes like spherical linear interpolation have discontinuities when the input angles change by more than $180°$.

To solve this problem, we use the fact that motions are fundamentally unchanged by rotations about the vertical axis and translations in the floor plane (see Section 4.1.1.1 of Chapter 4). In particular, if specific frames $\mathbf{M_1}(t_1), \ldots, \mathbf{M_n}(t_n)$ are to be combined, then the motions can be aligned so they are as similar as possible at these frames, a process we refer to as *coordinate frame alignment*[1] (Figure 5.5). If the character's body were a single point located at the character's root, then coordinate frame alignment would be the same as finding transformations that place each root in the same global position and orientation. Since we instead represent the body as a skeleton, the methods of Chapter 4 are used to find transformations that optimally align these skeletal postures.

Coordinate frame alignment allows us to construct an alignment curve $\mathbf{A}(u)$, which gives a set of transformations that align the frames at each point $\mathbf{S}(u)$ on the timewarp curve. By analyzing how these transformations change during a blend, we can incrementally blend root parameters in a way that avoids the artifacts of traditional methods (Figure 5.4). Section 5.2.2 explains how to construct an alignment curve and Section 5.3.2 provides details on using it in blending.

[1] Whenever the word "frame" is used in isolation, it refers to a frame of motion, and should not be confused with a coordinate frame.

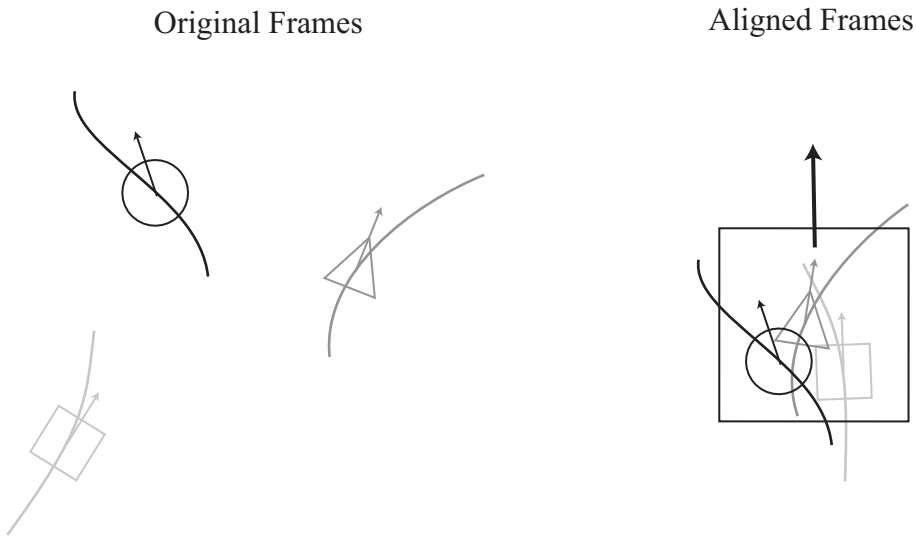

Figure 5.5: Left: Before alignment, a set of corresponding frames is scattered within the global coordinate system. The circle, triangle, and square represent root configurations of three frames, with the arrows indicating orientations in the ground plane. The curved lines depict the root trajectories of the surrounding segments of motion. **Right:** The same set of frames after alignment. Aligning transformations are based on the entire skeletal posture at each frame, and hence in general the roots do not end up in the same global position and orientation.

5.1.3 Constraint Matching

Once a position and orientation is determined for the root, the rest of the blended skeletal posture is created simply by averaging joint parameters. As a result, kinematic constraints on the blended motion may be violated. While it is straightforward to adjust the blend so constraints are satisfied (using, for example, the methods of Chapter 3), these constraints must first be explicitly labelled. Even if the constraints happen to already be satisfied, it is desirable for the blend to have explicit constraint annotations in the event that further changes are made to the motion, such as if it is used as an input for another blend. However, determining constraints can take some care because the input motions may have fundamentally different constraint states. For example, while a walking motion always has at least one foot planted, a running motion contains flight stages where no constraints exist, and the intervals over which constraints are active will not match even if the motions are timewarped. Also, the user might want to blend motions with different numbers of

constraints. For instance, the user might want to create different turning motions by interpolating a motion where a character stands still with one where it turns in place (and hence picks up its feet).

If the interval of each constraint is represented in terms of the global time parameter u, then it is reasonable to expect that constraints existing over nearby intervals have the same structural meaning, even if they do not span identical regions of time. For example, in both walking and running a foot is planted whenever it is supporting the character's weight, although the duration of this plant relative to the locomotion cycle varies. To infer the constraints on a blend, the global times at which these related constraints start and end may be averaged just like ordinary skeletal parameters. Section 5.2.3 presents an algorithm for automatically identifying sets of related constraints, which we call constraint matches. To handle input motions with different numbers of constraints, this algorithm allows individual constraints to be discarded or split into smaller pieces.

5.2 Building Registration Curves

Given n motions $\mathbf{M_1}, \ldots, \mathbf{M_n}$, a registration curve is created in three stages:

1. Build a timewarp curve $\mathbf{S}(u)$ representing sets of corresponding frames.
2. Build an alignment curve $\mathbf{A}(u)$ that specifies rigid $2D$ transformations that align the frames at each point on $\mathbf{S}(u)$.
3. Map constraints from the input motions into the global time frame defined by $\mathbf{S}(u)$ and identify constraint matches.

This section explains each of these stages in detail.

5.2.1 Timewarp Curves

A timewarp curve is built around a dense collection of *frame correspondences*, which are sets of n frame indices, one from each input motion, that occur at structurally similar points in the motions. The entire collection of frame correspondences is called a *time alignment*. A time alignment is effectively a discrete approximation of a timewarp curve, and a proper timewarp curve can be

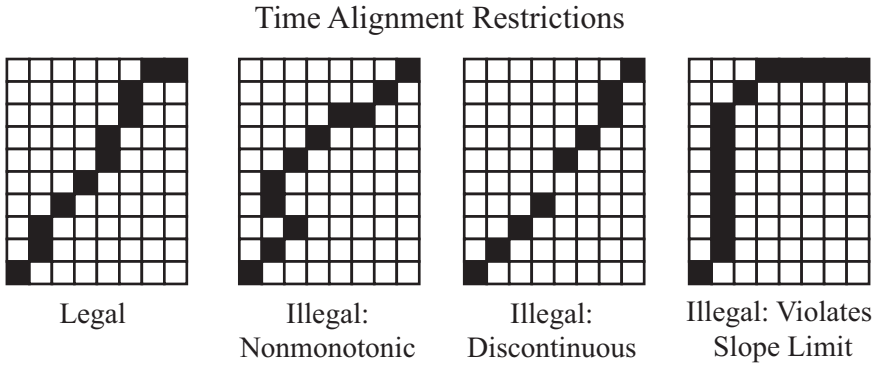

Figure 5.6: Legal and illegal time alignments ($W = 2$).

generated by fitting a strictly increasing spline to it. We first discuss the construction of timewarp curves when there are just two input motions, and then generalize the procedure to an arbitrary number of input motions.

5.2.1.1 Timewarp Curves for Two Motions

Let $\mathbf{M_1}$ have r_1 frames and $\mathbf{M_2}$ have r_2 frames, and imagine an r_1 by r_2 grid where cell (i, j) represents the frame pair $(\mathbf{M_1}(t_i), \mathbf{M_2}(t_j))$. A time alignment corresponds to a sequence of cells on this grid. To be a reasonable approximation of a timewarp curve, the time alignment is required to obey the following properties, which are illustrated in Figure 5.6.

1. **Continuity.** Each cell on the path must share a corner or edge with another cell on the path.

2. **Monotonicity.** Paths must always travel up and/or to the right.

3. **Slope Limit.** At most W consecutive steps may be taken in the same direction (horizontally or vertically).

The slope limit restriction prevents "degenerate" time alignments (and hence degenerate timewarp curves) where a large section of one motion maps to a small section of another. While such a mapping can be logically reasonable — e.g., if one motion contains a pause that is not present in the other — in practice it often introduces undesirable distortions into blends. Intuitively, this is because timewarping may be viewed as a process that trades off error in body posture (combining

frames with different skeletal poses) with error in timing (allowing time to flow at different rates for different motions), and neither of these errors can be too large if blending is too succeed. In our implementation we set W to 2 or 3.

There are many time alignments that obey the above restrictions, and we therefore construct one that is, under some measure, optimal. We use the heuristic that corresponding frames should appear more similar than frames that do not correspond, where similarity is measured with the distance metric defined in Equation 4.1 of Chapter 4. Section 5.5 discusses some of the limitations of this approach; for now we simply note that it is reasonable given that no information is present beyond the captured body postures. In our implementation the point cloud windows used in the distance computation were three frames wide ($L = 1$, or about a tenth of a second), although we have found that the optimal time alignment changes little with shorter or longer windows (between 0s and $\frac{1}{3}$s).

A continuous, monotonic, and slope-limited time alignment that minimizes the sum of the distances between corresponding frames can be found with dynamic programming, which is also called dynamic timewarping in the speech recognition literature [73]. Assume that a starting cell c (i.e., an initial frame correspondence) is known. For every cell c' that can be connected to c with at least one valid path, dynamic timewarping efficiently finds the optimal path by exploiting the fact that if the optimal path from c to c' passes through c'', then the subpaths from c to c'' and c'' to c' must also be optimal. Let $D_{i,j}$ be the frame distance associated with cell (i, j), and let $v_{i,j}$ be the total cost of the optimal path from c to cell (i, j). As long as $D_{i,j}$ is strictly positive, the cost of taking a horizontal step on the grid followed by a vertical step (or vice-versa) is always more costly than taking a single diagonal step. In light of this and the continuity, monotonicity, and slope limit restrictions, the end of a partial time alignment can only be extended in the $2W - 1$ ways shown in Figure 5.7, and therefore when calculating the optimal path to cell (i, j) it is sufficient to consider only the $2W - 1$ cells that connect to it through one of these extensions. Hence $v_{i,j}$ can be computed recursively:

$$v_{i,j} = \min \left\{ \min_{r \in [1,W]} \left\{ v_{i-r,j-1} + \sum_{k=0}^{r-1} D_{i-k,j} \right\}, \min_{r \in [1,W]} \left\{ v_{i-1,j-r} + \sum_{k=0}^{r-1} D_{i,j-k} \right\} \right\}, \qquad (5.1)$$

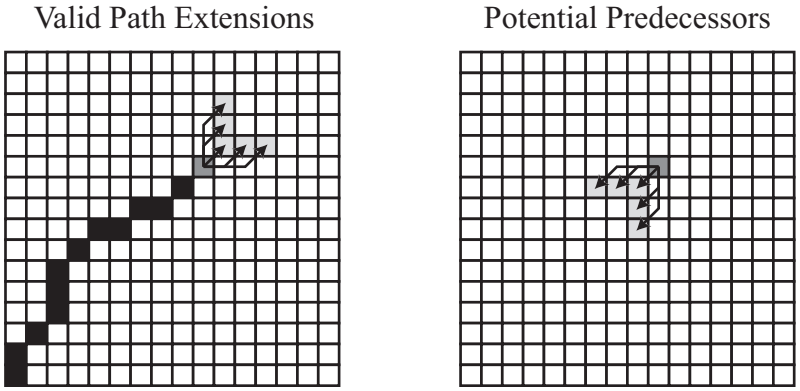

Figure 5.7: Let the slope limit be $W = 3$. **Left:** A valid path can only be extended in $2W - 1$ ways. **Right:** When computing the optimal path that terminates at a given cell, one only needs to consider the $2W - 1$ cells from which these extensions could originate.

where the first expression considers horizontal path sections that end at cell (i, j) and the second expressions considers vertical path sections. The total cost of the optimal path to each cell may now be determined by evaluating Equation 5.1 for all cells that are either 1 row above or 1 column to the right of c, then all cells either 2 rows above or 2 columns to the right of c, and so on. Processing cells in this order ensures that each v on the right hand side has a known value. Each cell in the grid is either marked as unreachable or is annotated with both its optimal cost $v_{i,j}$ and the cell that minimized Equation 5.1, which is called the *predecessor*. If a reachable cell $\tilde{c}$ is selected to be the end of the path, then the rest of the (optimal) path can be reconstructed by following the chain of predecessors back to c.

All that remains to be determined are the initial cell c and the final cell $\tilde{c}$. Typically c will be the lower left corner of the grid and $\tilde{c}$ will be the upper right corner, which is necessary to ensure that the entirety of the the input motions are spanned by the time alignment. However, for a transition it may be desirable to set c at the transition center, which will be in the interior of the grid. In this case, a time alignment can be constructed by separately computing "forward" and "backward" time alignments, where the former is computed as described above and the latter simply replaces up/right with down/left. Also, $\tilde{c}$ may have no preferred location, in which case our algorithm selects the cell on the grid boundary whose optimal path has the minimum average cell

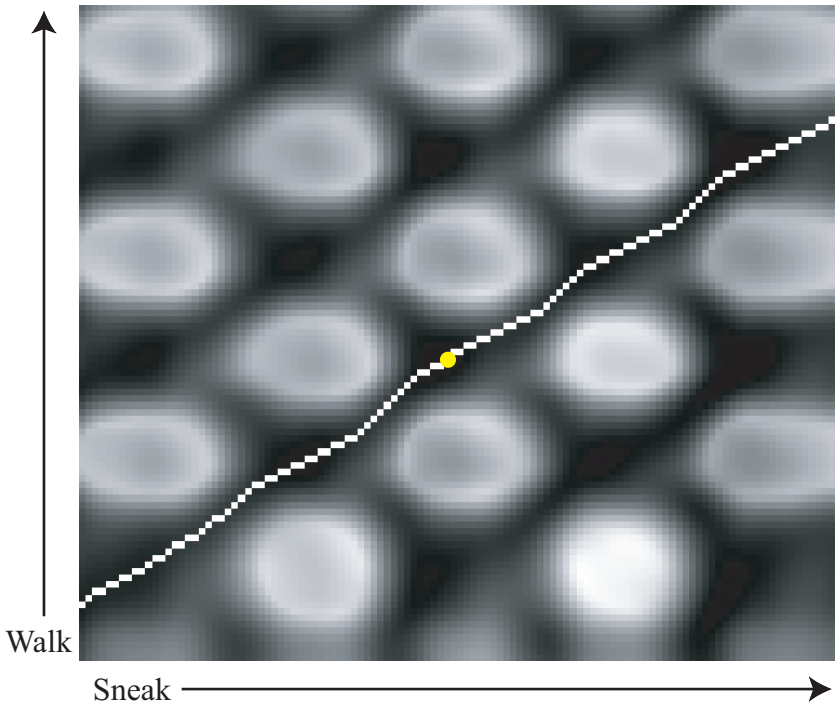

Figure 5.8: A time alignment for a walk and a sneak. The background image shows frame distances; smaller distances correspond to darker cells. The central yellow circle was chosen as the starting point, and optimal paths (shown in white) were separately generated forwards and backwards. The local slope variations arise because the character pauses slightly at each footstep in the sneaking motion but travels at a steady pace in the walking motion.

cost. To expedite this calculation, it is convenient to store not just v and the predecessor at each cell, but also the number of cells on the optimal path. Figure 5.8 shows an example time alignment where the starting cell was near the center of the grid and no preferred value was given for $\tilde{c}$.

Since the grid is r_1 by r_2, the total cost of using dynamic programming is $O(r_1 r_2)$, even if cells that are outside of the slope limits are not processed (Figure 5.9). A more efficient variant of the algorithm is to find the optimal path on a smaller k by k grid whose lower left corner is at c, retain the first k' cells, and iterate the process with cell $k'+1$ serving as the lower left corner of the new grid. This algorithm is $O\left(\frac{k^2}{k'}\left(r_1+r_2\right)\right)$, since $O\left(\frac{r_1+r_2}{k'}\right)$ grids must be processed with $O(k^2)$ work per grid. In practice we have found that relatively small values of k (15 to 20 frames; $k'=10$) yield results qualitatively similar to computing over the full grid. Indeed, in some cases

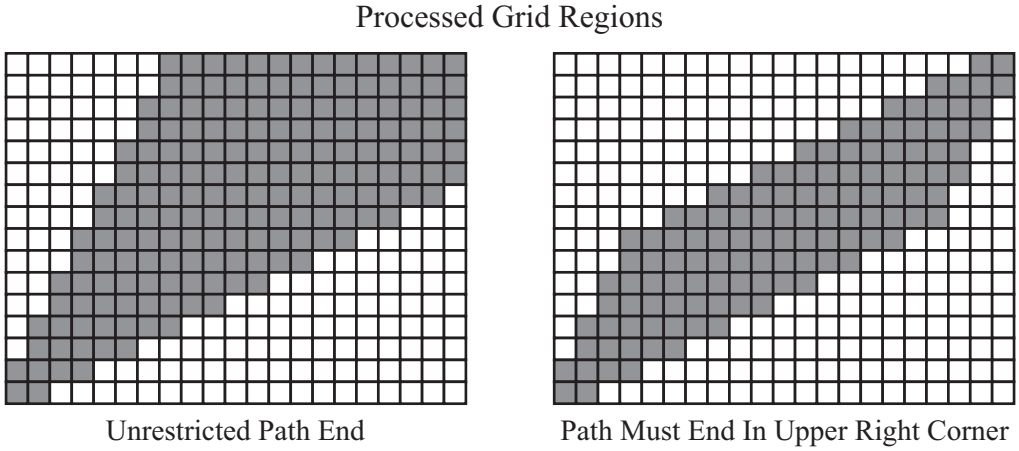

Figure 5.9: Only a subset of the grid needs to be processed, since the slope limits render some cells inaccessible (W=2). **Left:** Processed region with no restrictions on the last cell of the path. **Right:** Processed region when the last cell is required to be (r_1, r_2)

this process can produce superior results. For example, if the goal is to make a transition between two motions that are very dissimilar away from the transition point, then a globally optimal time alignment may be less desirable than one that is only optimal within the vicinity of the transition, since the globally optimal result will be forced to match unrelated frames while the incremental result can effectively ignore frames that are not included in the blend.

The frame correspondences generated by dynamic timewarping could be linearly interpolated to form a continuous mapping between the frames of $\mathbf{M_1}$ and $\mathbf{M_2}$, but the result would not be a proper timewarp curve because it is not in general strictly increasing — multiple frames of one motion may be matched to a single frame of the other. Moreover, while the slope limits place global restrictions on how steep or shallow this pseudo-timewarp curve can be, they do not ensure any smoothness properties. Our algorithm instead creates a timewarp curve by computing a strictly increasing two-dimensional quadratic B-spline with uniformly spaced knots that is minimally distant from the time alignment in a least-squares sense.

Let $p_{i,j}$ be the j^{th} dimension of the i^{th} control point of the desired B-spline. Then for any three consecutive control points $\mathbf{p_i}$, $\mathbf{p_{i+1}}$, and $\mathbf{p_{i+2}}$, the derivative $\frac{dS_j}{du}$ in the associated spline interval is always between $(p_{i+1,j} - p_{i,j})$ and $(p_{i+2,j} - p_{i+1,j})$, and the spline can therefore be forced to be

strictly increasing by requiring $(p_{i+1,j} - p_{i,j}) > 0$ for all i and j. Since a quadratic objective is to be minimized subject to linear inequality constraints, computing the optimal spline is a quadratic programming problem. However, in light of the causality and slope limit restrictions, it is likely that a spline fit without any constraints will be *approximately* strictly increasing. Our strategy is hence to fit a uniform quadratic B-spline as an unconstrained optimization (which is a linear least squares problem) and then adjust the knots. For each j, the knot adjustment algorithm starts at $i = 1$ and iteratively computes

$$\Delta_{i,j} = (p_{i+1,j} - p_{i,j}) - \epsilon, \tag{5.2}$$

where ϵ is a small positive number. If $\Delta_{i,j} \geq 0$, nothing is done. Otherwise, define

$$\lambda = \max\left(\frac{1}{2}, \frac{\Delta_{i-1,j}}{\Delta_{i,j}}\right), \tag{5.3}$$

where $\Delta_{i-1,j}$ is computed using the value of $p_{i-1,j}$ at the beginning of the current iteration. The value of $p_{i,j}$ is reduced by $\lambda\Delta_{i,j}$ and the value of $p_{i+1,j}$ is increased by $(1-\lambda)\,\Delta_{i,j}$. At this point $(p_{k+1,j} - p_{k,j}) \geq \epsilon$ for all k between 1 and i. Equation 5.3 chooses λ so the necessary change is split as evenly as possible between $p_{i,j}$ and $p_{i+1,j}$.

The user may want the first point on the timewarp curve to be exactly $(1, 1)$ and the last point to be exactly (r_1, r_2), so the entire frame ranges of $\mathbf{M_1}$ and $\mathbf{M_2}$ are spanned. If the timewarp curve has k control points, then this can be accomplished by requiring $\frac{1}{2}(\mathbf{p_1} + \mathbf{p_2}) = (1, 1)$ and $\frac{1}{2}(\mathbf{p_{k-1}} + \mathbf{p_k}) = (r_1, r_2)$, which can be enforced by adjusting the control points in a manner similar to what was described in the previous paragraph.

5.2.1.2 Timewarp Curves For More Than Two Motions

Directly applying the methods of Section 5.2.1.1 to arbitrarily many motions is computationally intractable, because generalizing the dynamic programming algorithm to n input motions involves processing an n-dimensional grid, which requires $O(n^r)$ work if each motion has on average r frames. On the other hand, a simple and efficient algorithm can be devised by combining multiple timewarp curves for pairs of motions. Consider building a timewarp curve for three motions $\mathbf{M_1}$, $\mathbf{M_2}$, and $\mathbf{M_3}$, and assume that timewarp curves $\mathbf{S^{1\leftrightarrow 2}}$, $\mathbf{S^{1\leftrightarrow 3}}$, and $\mathbf{S^{2\leftrightarrow 3}}$ have already been

constructed for each pair. If $S_1^{1\leftrightarrow 2}(t_1) = t_2$ and $S_1^{1\leftrightarrow 3}(t_1) = t_3$, then is reasonable to expect that $\{\mathbf{M_1}(t_1), \mathbf{M_2}(t_2), \mathbf{M_3}(t_3)\}$ will be an accurate frame correspondence, which implies that $S_1^{2\leftrightarrow 3}(t_2)$ is approximately t_3 and $S_2^{2\leftrightarrow 3}(t_3)$ is approximately t_2 . Hence $\mathbf{S^{2\leftrightarrow 3}}$ might be discarded and $\mathbf{S^{1\leftrightarrow 2}}$ and $\mathbf{S^{1\leftrightarrow 3}}$ could be merged into a single timewarp curve.

More generally, given a set of motions $\mathbf{M_1}, \ldots, \mathbf{M_n}$, a motion $\mathbf{M_{ref}}$ is selected to serve as a reference. In our implementation, $\mathbf{M_{ref}}$ was chosen to minimize the average distance to the other motions, where the distance between $\mathbf{M_i}$ and $\mathbf{M_j}$ is defined as the average distance between each frame pair of the time alignment. Let $\mathbf{S_i}(u) = (S_{i,1}(u), S_{i,2}(u))$ be the timewarp curve between $\mathbf{M_{ref}}$ and $\mathbf{M_i}$. For convenience and without loss of generality, we assume $S_{i,1}(u)$ returns frame indices of $\mathbf{M_{ref}}$ and $S_{i,2}(u)$ returns corresponding frame indices from $\mathbf{M_i}$. We sample the frame indices of $\mathbf{M_{ref}}$ and for each sample use the inverse functions $u_{S_{i,1}}(t)$ to determine the corresponding frames in each other motion. Specifically, the frame of $\mathbf{M_i}$ that corresponds to $\mathbf{M_{ref}}(t_0)$ is at time $S_{i,2}(u_{S_{i,1}}(t_0))$. The result of this sampling is a set of frame correspondences

$$\left\{\mathbf{M_{ref}}(t_i), \mathbf{M_1}\left(S_{1,2}(u_{S_{1,1}}(t_i))\right), \ldots, \mathbf{M_n}\left(S_{n,2}(u_{S_{n,1}}(t_i))\right)\right\}, \tag{5.4}$$

and the final timewarp curve is formed by fitting to these frame correspondences a strictly increasing n-dimensional quadratic spline with uniformly spaced knots, as in the two-dimensional case.

5.2.2 Alignment Curves

Given a timewarp curve $\mathbf{S}(u)$, an alignment curve $\mathbf{A}(u)$ is constructed by finding rigid $2D$ coordinate transformations that mutually align the frames at each point on $\mathbf{S}(u)$. We first consider the case where there are just two input motions, $\mathbf{M_1}$ and $\mathbf{M_2}$. For the i^{th} frame correspondence $\{\mathbf{M_1}(t_{1_i}), \mathbf{M_2}(t_{2_i})\}$ generated through dynamic programming, Equations 4.2–4.4 of Chapter 4 specify a rigid $2D$ transformation $\{\theta_i, x_{0_i}, z_{0_i}\}$ that, when applied to $\mathbf{M_2}(t_{2_i})$, aligns it with $\mathbf{M_1}(t_{1_i})$. A $3D$ quadratic spline may be fit to these transformations, yielding an alignment curve $\mathbf{A}(u) = (\mathbf{A}_1(u), \mathbf{A}_2(u))$ where every $\mathbf{A}_1(u)$ is the identity transformation and $\mathbf{A}_2(u) = \{\theta(u), x_0(u), z_0(u)\}$ is the result of the spline fit. To avoid angle discontinuities, prior to fitting the spline the sampled rotation parameters should be adjusted so $|\theta_i - \theta_{i-1}| \leq \pi$. Also, there are

sometimes small spikes in the transformation parameters when the dynamic timewarping algorithm shifts from one kind of step (vertical, horizontal, or diagonal) to another. We have found it useful to treat these spikes as impulse noise and remove them with a short median filter (kernel width of 3 to 5 frames) prior to fitting the spline. Finally, it is important that the knot spacing on the spline be sufficiently small that it remains close to the (filtered) samples; we have found that a knot for every 3 to 5 samples yields good results.

The case of n input motions is handled in direct analogy to Section 5.2.1.2. It is assumed that a timewarp curve has been constructed for the entire set of inputs and that pairwise alignment curves have been created between the reference motion $\mathbf{M_{ref}}$ and each other input motion $\mathbf{M_i}$. Frames are sampled from $\mathbf{M_{ref}}$, and for each sample $\mathbf{M_{ref}}(t_j)$ the corresponding frame of each other motion is determined through the timewarp curve. The alignment curves are then used to find transformations $\mathbf{A_i}$ that align the frame of each $\mathbf{M_i}$ with $\mathbf{M_{ref}}(t_j)$. Note that this assumes that $\mathbf{A_k^{-1}A_i}$ is a good approximation of the transformation that directly aligns the frame of $\mathbf{M_i}$ with the frame of $\mathbf{M_k}$. The result of this procedure is a collection of sets of n transformations; at least one transformation of each set is always the identity since it must align a frame of $\mathbf{M_{ref}}$ with itself. Finally, a $3n$-dimensional quadratic spline is fit to these sampled transformations as in the two-motion case.

We note that this method yields smooth alignment curves even when the root paths of the input motions have sharp turns or when they effectively degenerate into single points (such as when the characters are standing in place); see Figure 5.16. This is because the coordinate transformations are determined by the shape of the whole body over a small time window. Clearly, this shape cannot collapse to a point, since the body always has a nonzero extent. Similarly, while the root path may geometrically have a sharp turn, the body shape itself will nonetheless change smoothly as that turn is executed.

5.2.3 Constraint Matches

The final step in building a registration curve is to identify constraint matches. Each kind of constraint is treated independently — for example, left heelplants might be considered first, then

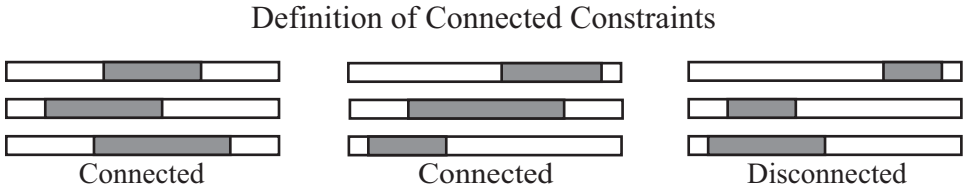

Figure 5.10: Examples of connected and disconnected constraints. Note that a set of constraints can be considered connected even if individual pairs do not overlap, such as in the central figure.

right wristplants, and so on. Also, the time intervals of each kind of constraint are assumed to be disjoint, so if two left heelplants occur respectively in the intervals $[t_1^s, t_1^e]$ and $[t_2^s, t_2^e]$, then $t_1^e < t_2^s$. The algorithm starts by mapping the duration of each constraint into the global time frame via the timewarp curve. Constraints are then grouped into constraint matches based on the heuristic that corresponding constraints should occur at similar points in (global) time. Specifically, there are two guidelines:

1. Each constraint match must contain exactly one constraint from each motion.

2. The elements of each constraint match must be "connected" in the sense that the union of all constraint intervals must form a single continuous interval (Figure 5.10).

Note that some constraints may not belong to any constraint match (for example, a constraint may overlap with no other constraints). In this case that constraint is eliminated. Intuitively, this is because a constraint which is not shared by all of the input motions may be viewed as incidental to the motion that has it, rather than a structural property of the kind of action that is taking place. To illustrate, imagine blending a motion where a character stands on its left leg with a motion where it stands on both legs. One expects the blend to consist of a character standing on its left leg and varying the height of its right foot according to the blend weights. If all of the weight happens to be on the second motion, then both legs will happen to be in contact with the ground, but except for this special case the left foot is logically planted and the right foot is not. It is therefore reasonable to eliminate the plant constraints on the right foot. Using similar reasoning, one or more gaps may be added to a constraint of one motion if it overlaps with multiple constraints of the other motions, thereby splitting it into shorter constraints that can be separately matched.

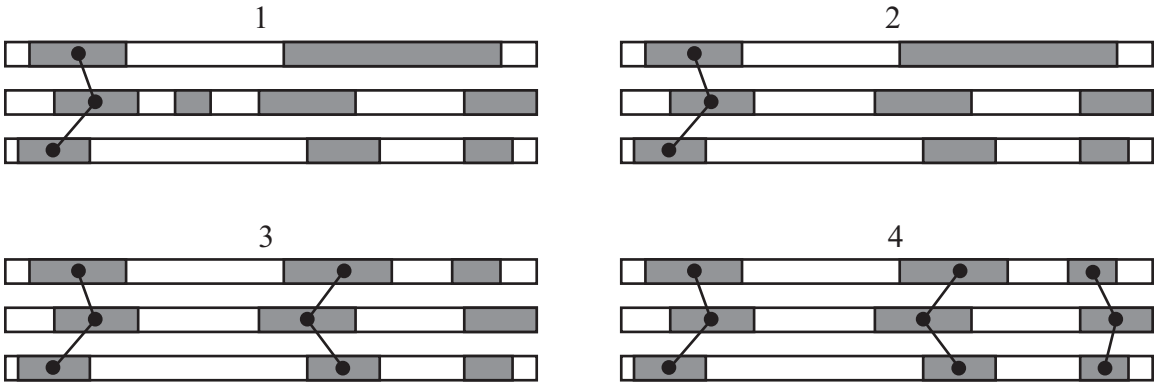

Figure 5.11: The steps taken by the constraint matching algorithm on a sample input. Note that a constraint is removed from the second motion and split in the first motion.

Our constraint matching algorithm proceeds as follows. Let $C_{i,j}$ be the j^{th} constraint of the i^{th} motion. Our algorithm processes the constraints sequentially: $C_{i,j}$ is either eliminated or added to a constraint match before $C_{i,j+1}$ is considered. Each iteration starts by checking whether the earliest unprocessed constraint of each motion are all connected in the sense of Figure 5.10. If not, the constraint which starts the earliest cannot be part of any constraint match, so we discard it and proceed to the next iteration. Otherwise a constraint match can be formed from these constraints. The algorithm first decides whether any of them should be split, and then a constraint match is built from the constraints that were not split and the first portion of the constraints that were split. Figure 5.11 shows the operation of our constraint matching algorithm on a sample input.

The only remaining details are how to determine whether a particular constraint should be split and how to actually perform such a split. Without loss of generality, we limit our discussion to the case where the set of candidate constraints for a constraint match are exactly the first constraint $C_{i,1}$ of each motion. The decision to split is based on *subsumption*: $C_{i,1}$ subsumes $C_{j,1}$ if

1. $C_{i,1}$ overlaps $C_{j,1}$ and $C_{j,2}$ ($C_{j,2}$ must exist)

2. $C_{i,2}$ does not overlap $C_{j,2}$ (possibly because $C_{i,2}$ does not exist)

Figure 5.12 illustrates subsumption. Note that if $C_{i,1}$ subsumes $C_{j,1}$, then $C_{j,1}$ does not subsume $C_{i,1}$. The constraints are partitioned into two sets $\mathcal{S}_1$ and $\mathcal{S}_2$ such that every element of $\mathcal{S}_1$ subsumes

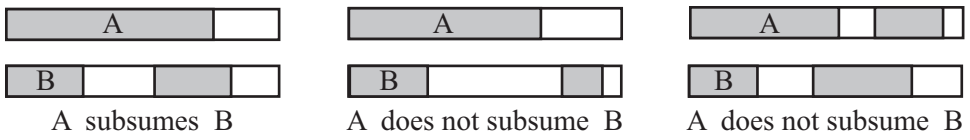

Figure 5.12: Examples of subsumption. In the rightmost case, there would be two separate constraint matches, one containing the first constraint of each motion and the other containing the second constraint of each motion.

every element of $\mathcal{S}_2$; the constraints that will be split are in $\mathcal{S}_1$ and the constraints that will remain intact are in $\mathcal{S}_2$. A candidate partition can be created by first placing a particular constraint $C_{i,1}$ into $\mathcal{S}_1$ and the remaining constraints into $\mathcal{S}_2$. Each constraint in $\mathcal{S}_2$ is next checked to see if it is subsumed by all constraints currently in $\mathcal{S}_1$, and if not then it is switched to $\mathcal{S}_1$. This process iterates until no further changes can be made. If $\mathcal{S}_2$ ends up being empty, nothing should be split, and hence all the constraints are transferred to $\mathcal{S}_2$. The sets $\mathcal{S}_1$ and $\mathcal{S}_2$ are generated in this fashion for each constraint, and our algorithm keeps the partition where the "effective" constraint intervals are as similar as possible. More exactly, if the interval of $C_{i,j}$ in global time is $[u^s_{i,j}, u^e_{i,j}]$, then let I_i be $[u^s_{i,1}, u^e_{i,1}]$ if $C_{i,1} \in \mathcal{S}_1$ and $I_i = [u^s_{i,1}, u^e_{i,2}]$ if $C_{i,1} \in \mathcal{S}_2$. That is, if the constraint is in $\mathcal{S}_2$, then pretend that it terminates at the end of constraint that immediately follows it, since each constraint in $\mathcal{S}_1$ maps to *pairs* of constraints implicitly defined in $\mathcal{S}_2$. The quality of the partition σ is then defined as

$$\sigma = \sum_i \sum_j (I_i \cap I_j), \tag{5.5}$$

where the operator $\cap$ returns the size of the intersection of the two intervals.

Each constraint in $\mathcal{S}_1$ is now split into two shorter constraints separated by a gap; refer to Figure 5.13. Let $C_{i,1} \in \mathcal{S}_1$ be the current constraint that is to be split. Each constraint $C_{j,1} \in \mathcal{S}_2$ votes on where the start and the end of the gap in $C_{i,1}$ should be. First, the end of $C_{j,2}$ is mapped to a point p on $C_{i,1}$: if $C_{j,3}$ exists and $u^s_{j,3} < u^e_{i,1}$, then $p = u^e_{j,2}$, and otherwise $p = u^e_{i,1}$. The vote for the start of the gap is then

$$u^s_{i,1} + \frac{p - u^s_{i,1}}{u^e_{j,1} - u^s_{j,1}}(u^e_{j,1} - u^s_{j,1}) \tag{5.6}$$

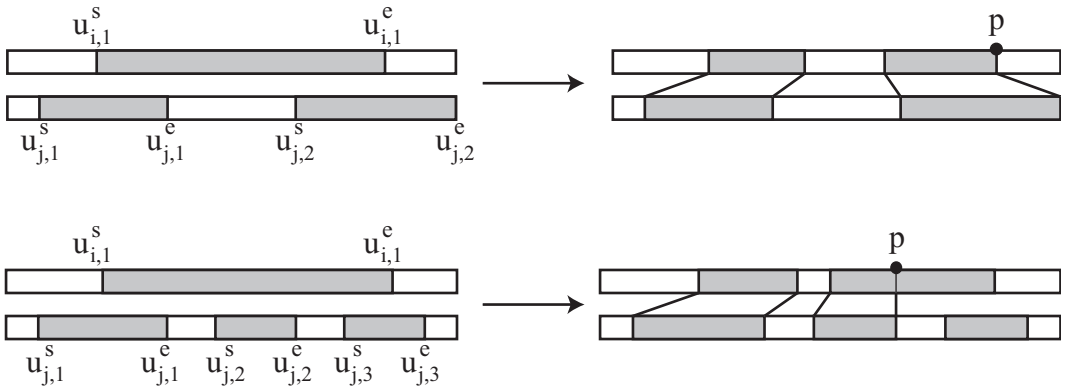

Figure 5.13: Adding a gap to a constraint.

and the vote for the end of the gap is similarly defined. The actual gap in $C_{i,1}$ is determined by averaging the votes.

5.3 Blending with Registration Curves

Creating a single frame $\mathbf{B}(t_i)$ of a blend involves four steps:

1. An appropriate point $\mathbf{S}(u_i)$ on the timewarp curve is calculated.
2. The frames at $\mathbf{S}(u_i)$ are positioned and oriented according to the corresponding point on the alignment curve, $\mathbf{A}(u_i)$.
3. These adjusted frames are combined using the blend weights $\mathbf{w}(t_i)$.
4. Constraints are determined for the blended frame.

We assume that for some blend frame $\mathbf{B}(t_0)$, u_0 is already known. This frame is created first and the other frames are generated in chronological order: first the forward frames ($\mathbf{B}(t_1)$, $\mathbf{B}(t_2)$, ...) and then (if $\mathbf{B}(t_0)$ is not the first blend frame) the backward frames ($\mathbf{B}(t_{-1})$, $\mathbf{B}(t_{-2})$, ...). We require $u(t)$ to be strictly increasing, so advancing forward in blend time t will result in advancing forward in global time u.

The remainder of this section explains how to create a single blend frame. The discussion is limited to the case of generating frames in forward order; moving backward in time is handled similarly.

5.3.1 Advancing Along the Timewarp Curve

If the first blend frame $\mathbf{B}(t_0)$ is being created, then this step is skipped, since u_0 is assumed to be known in advance. Otherwise, the previous blend frame combined input frames at some point $\mathbf{S}(u_{i-1})$ on the timewarp curve, and the goal is now to obtain a new set of input frames by moving to an updated location $\mathbf{S}(u_i)$. To understand how this might be done, imagine that for the remainder of the blend $w_j(t) = 1$ and the remaining blend weights are 0. In this case the rest of $\mathbf{B}$ should be identical to a portion of $\mathbf{M_j}$, and hence the position on the timewarp curve should change such that $\mathbf{M_j}$ is played at its natural rate. Specifically, to advance $\Delta t = t_i - t_{i-1}$ units of time, $\Delta u = u_i - u_{i-1}$ should be chosen such that

$$S_j(u_{i-1} + \Delta u) - S_j(u_{i-1}) = \Delta t \tag{5.7}$$

Taking the limit as $\Delta t \to 0$ and $\Delta u \to 0$,

$$\frac{dS_j}{dt} = \frac{dS_j}{du}\frac{du}{dt} = 1$$

$$\frac{du}{dt} = \left(\frac{dS_j}{du}\right)^{-1} \tag{5.8}$$

For general $\mathbf{w}$(t), we extend this equation to

$$\frac{du}{dt} = \sum_{j=1}^{k} w_j(t) \left(\frac{dS_j}{du}\right)^{-1}. \tag{5.9}$$

Intuitively, each motion votes on how fast the global time parameter u should flow, and these votes are combined according to the blend weights. Note that $\left(\frac{dS_j}{du}\right)^{-1}$ can be expressed analytically, since $S_j(u)$ is a quadratic function of u. In general this differential equation must be solved numerically, but since the time step between frames is small and $\mathbf{w}$ and $\mathbf{S}$ are smooth, we have found

that a single Euler step can be taken to advance one frame in the blend:

$$\Delta u = \left(\sum_{j=1}^{k} w_j(t_{i-1}) \left(\left. \frac{dS_j}{du} \right|_{u=u_{i-1}} \right)^{-1} \right) \Delta t \tag{5.10}$$

A similar strategy was used in [68]. For convex weights, which are commonly viewed as natural choices for blending operations, Equation 5.10 will always yield $\Delta u > 0$ and hence time will always flow forward. If arbitrary blend weights are allowed, then $w_j(t_{i-1})$ can be replaced with $\frac{|w_j(t_{i-1})|}{\sum_r |w_r(t_{i-1})|}$ in Equation 5.10, ensuring $\Delta u > 0$.

5.3.2 Positioning and Orienting Frames

Once u_i has been determined, the frames $\mathbf{M}_j(S_j(u_i))$ are extracted from the input motions, and the root configuration of each frame is transformed by $\mathbf{A}_j(u_i)$. This yields a group of mutually aligned frames that, just as with a single frame of motion, may be translated and rotated in the ground plane by a transformation $\mathbf{T}(t_i)$ (Figure 5.14), in which case the total transformation applied to the root of $\mathbf{M}_j(S_j(u_i))$ is $\mathbf{T}(t_i)\mathbf{A}_j(u_i)$. For the first frame of the blend, $\mathbf{T}(t_0)$ may be chosen arbitrarily. For the other frames, $\mathbf{T}(t_i)$ must be chosen so the position and orientation of $\mathbf{B}(t_i)$ is consistent with the preceding frame $\mathbf{B}(t_{i-1})$. To see how this can be done, assume temporarily that $w_j(t)$ is 1 and the other blend weights are all 0 for $t > t_{i-1}$. The remainder of the blend should then simply be a copy of a portion of $\mathbf{M}_j$, transformed rigidly by $\mathbf{T}(t_{i-1})\mathbf{A}_j(u_{i-1})$ so it connects seamlessly with $\mathbf{B}(t_{i-1})$. If $\Delta\mathbf{T}_j(t_i)$ is defined as

$$\Delta\mathbf{T}_j(t_i) = \mathbf{T}(t_{i-1})\mathbf{A}_j(u_{i-1}){\mathbf{A}_j}^{-1}(u_i), \tag{5.11}$$

then setting $\mathbf{T}(t_i) = \Delta\mathbf{T}_j(t_i)$ yields this result.

More generally, each motion $\mathbf{M_j}$ votes for the $\mathbf{T}(t_i)$ that leaves its local coordinate system unchanged (namely, $\Delta\mathbf{T}_j(t_i)$), and these votes are averaged according to the blend weights, as depicted in Figure 5.14. The first step in this process is to choose appropriate parameters to represent $\Delta\mathbf{T}_j(t_i)$. To locate an origin that is near the center of the transformed frames, each $\mathbf{M}_j(S_j(u_i))$ is transformed by $\Delta\mathbf{T}_j(t_i)\mathbf{A}_j(u_i)$, and the new root positions are then projected onto the ground

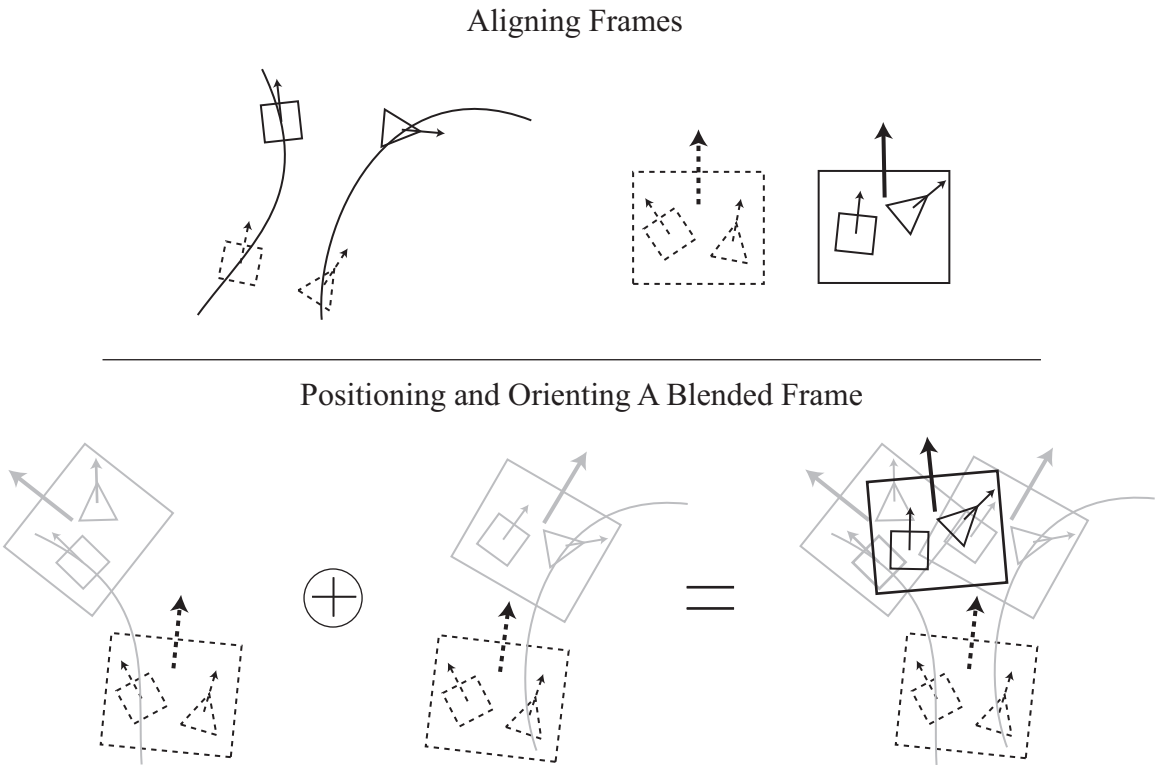

Figure 5.14: Top: Given any set of corresponding frames, the alignment curve specifies rigid $2D$ transformations that mutually align them. The curved lines on the left show the root trajectory of two motions, and the root configurations are indicated for two pairs of corresponding frames: the solid square and solid triangle, and the dashed square and dashed triangle. The right shows what these root configurations might look like when the frames are aligned. Each group of aligned frames can itself be rigidly transformed. **Bottom:** Based on the position and orientation of the previous blend frame, each motion votes on a new position and orientation for the current blend frame. These votes are combined according to the blend weights; here we show the case $w_1 = w_2 = 0.5$.

and averaged. $\Delta\mathbf{T}_j(t_i)$ is then represented by the parameter set $\{\phi_j, x_j, z_j\}$, which corresponds to a rotation by ϕ_j about this origin followed by a translation (x_j, z_j). $\mathbf{T}(t_i)$ is then defined as

$$\mathbf{T}(t_i) = \left\{ \sum_j w_j \phi_j, \sum_j w_j x_j, \sum_j w_j z_j \right\}. \tag{5.12}$$

5.3.3 Creating the Blended Frame

Having extracted frames from the input motions and placed them in the appropriate positions and orientations, the blended skeletal pose is formed by using the current blend weights to compute a weighted average of the input joint parameters. To average orientations, we originally adopted the algorithm advocated by Park et al [68]. Their method first computes a reference orientation $\mathbf{q_{ref}}$ that minimizes a distance measure to the input orientations $\mathbf{q_i}$. Each $\mathbf{q_i}$ is then transformed to the local coordinate frame of $\mathbf{q_{ref}}$ and converted to a logarithmic map representation, and these logarithmic maps are averaged. Finally, the average orientation $\overline{\mathbf{q}}$ is found by taking the exponential map and transforming back to the global coordinate frame:

$$\overline{\mathbf{q}} = \mathbf{q_{ref}}\mathbf{exp}\left(\sum_i w_i \mathbf{log}\left(\mathbf{q_{ref}^{-1}}\mathbf{q_i}\right)\right) \tag{5.13}$$

However, we subsequently determined that nearly identical results could be obtained in practice by placing each orientation on the same hemisphere of the quaternion sphere (i.e., replacing $\mathbf{q_i}$ with its antipode if $\mathbf{q_i} \cdot \mathbf{q_1} < 0$), averaging their coordinates like ordinary 4-vectors, and normalizing the result. Using this simpler operation reduced the total time needed to make a blend by a factor of three.

The final task is to determine the constraints on the new blend frame. For each constraint match $\mathcal{M}$, an interval $I_{\mathcal{M}}$ for the corresponding constraint is determined based on the current blend weights. Specifically, if the j^{th} constraint of $\mathcal{M}$ is active over the interval $[u_j^s, u_j^e]$, then

$$I_{\mathcal{M}} = \left[\sum_j w_j(t_i)u_j^s,\ \ \sum_j w_j(t_i)u_j^e\right] \tag{5.14}$$

If u_i is inside this interval, then the new blend frame is annotated with the appropriate constraint. Actually enforcing the constraints is up to the implementation; we used the method described in Chapter 3.

5.4 Results and Applications

Building a registration curve for n motions that are m frames each takes $O(m^2n^2)$ time, since timewarp and alignment curves must be generated for each pair of motions, and constructing the former requires solving an $O(m^2)$ dynamic programming problem (unless one uses the approximate algorithm discussed in Section 5.2.1.1, in which case it only takes $O(m)$ time). The largest registration curve we constructed in our experiments was composed of 18 motions with an average of 300 frames each (about 10s at 30Hz, or 3 minutes of data total). Using the $O(m^2)$ dynamic programming algorithm, it took 3.3s to construct this registration curve on a machine with a 1.3GHz Athlon processor. This time should be viewed as a preprocessing step, since a registration curve can be computed once and stored for use in multiple blending operations. The amount of space needed for a registration curve is $O(mn)$ (i.e., proportional to the total number of frames), since all that must be stored are the control points of the timewarp curve, the control points of the alignment curve, and the constraint intervals of each constraint match. On average the size of this information on disk was 1%–2% of the original data. The time needed to create a blend is also $O(mn)$, and computing a full 300-frame blend of the 18 motions mentioned earlier took about 0.3s.

Registration curves can be used as a back end for common blending applications. The rest of this section discusses using registration curves for transitions, interpolations, and continuous motion control. Videos are available at http://www.cs.wisc.edu/Gallery/Kovar/RegistrationCurves/.

5.4.1 Transitions

To create a transition, our algorithm must know its location and duration. The location can be conveniently specified through the central frames $\mathbf{M}_1(t_i)$ and $\mathbf{M}_2(t_j)$ — that is, if the transition were created simply by cutting from $\mathbf{M}_1$ to $\mathbf{M}_2$, then $\mathbf{M}_1(t_i)$ would immediately precede $\mathbf{M}_2(t_j)$. The duration of the transition can be given through its half-width h, in which case it spans a total of $2h + 1$ frames and the central frame is an equally weighted combination of $\mathbf{M}_1(t_i)$ and $\mathbf{M}_2(t_j)$.

When creating the registration curve, the grid cell $(\mathbf{M}_1(t_i), \mathbf{M}_2(t_j))$ serves as the starting point for the dynamic timewarping algorithm, and the timewarp curve is generated forwards and

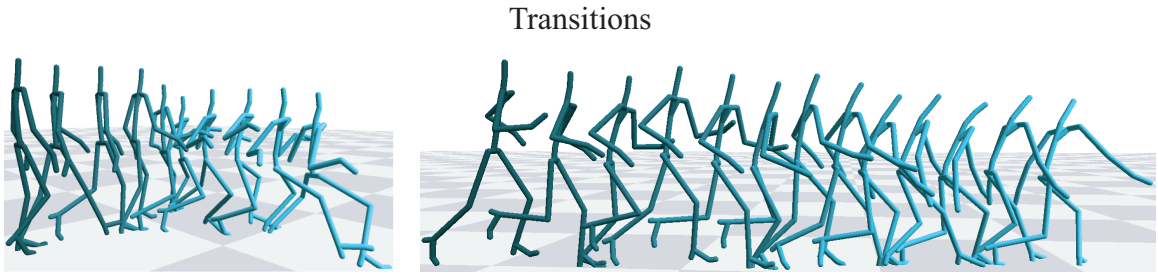

Figure 5.15: Transitions between different kinds of locomotion. Both transitions span two locomotion cycles. **Left:** Normal walking to a stylized walk. **Right:** Jogging to sneaking.

backwards from this point. When creating the blend, the starting position $\mathbf{S}(u_0)$ on the time-warp curve is found by averaging the u values that correspond to $\mathbf{M}_1(t_i)$ and $\mathbf{M}_2(t_j)$: $u_0 = \frac{1}{2}(u_{S_1}(t_i) + u_{S_2}(t_j))$. Once this central frame is created, the remainder of the blend is formed by generating h frames forward in time and then h frames backward in time. Blend weights that smoothly change from $(1, 0)$ to $(0, 1)$ are generated through the function defined in Equation 3.5 of Chapter 3. Figure 5.15 shows example transitions generated by our system.

In general, the first and last frames of the transition will not occur at integer times, and so $\mathbf{M}_1$ and $\mathbf{M}_2$ must be resampled if they are to connect smoothly to the transition boundaries. If this is undesirable, the boundaries can be forced to occur at integer times by generating $u_{-h}, \ldots, u_h$ as described in Section 5.3.1, applying a smooth displacement map that rounds u_{-h} and u_h to the nearest integers, and using these new values in place of the originals.

Registration curves could readily be used to generate transitions in the motion graph construction algorithm presented in Chapter 4. However, the transition times in this case are so short (0.5s, compared with 1-2s in the examples in this chapter) that we have found linear blending to suffice — indeed, we were unable to find a specific case where registration curves yielded clearly superior results. Intuitively, this is because in short transitions there is not enough time for motions to get sufficiently out of phase or for their root trajectories to diverge sufficiently that registration curves are necessary. On the other hand, registration curves could potentially be used to automatically build motion graphs where transition durations span a wide range of values instead of being fixed.

5.4.2 Interpolations

A set of input motions can be interpolated by using fixed blend weights. Our algorithm assumes that the entirety of each input motion is to be blended, and therefore the timewarp curve (and each pairwise time alignment) is forced to pass through the first frame of each motion and the last frame of each motion. The blend itself is created by setting u_0 to be the first point on the timewarp curve and stepping forward until the end of the timewarp curve is reached. Figures 5.16 and 5.17 show interpolations of several sets of input motions.

One of the principle applications of motion interpolation is the construction of parameterized motions. Parameterized motions are considered in more detail in Chapter 6, where methods are presented for automatically identifying and extracting variants of a given action from a data set (that is, obtaining the initial examples) and for building accurate maps between blend weights and relevant motion features.

Figure 5.16: Interpolations of three kicks are randomly sampled to create a large number of similar but distinct kicks.

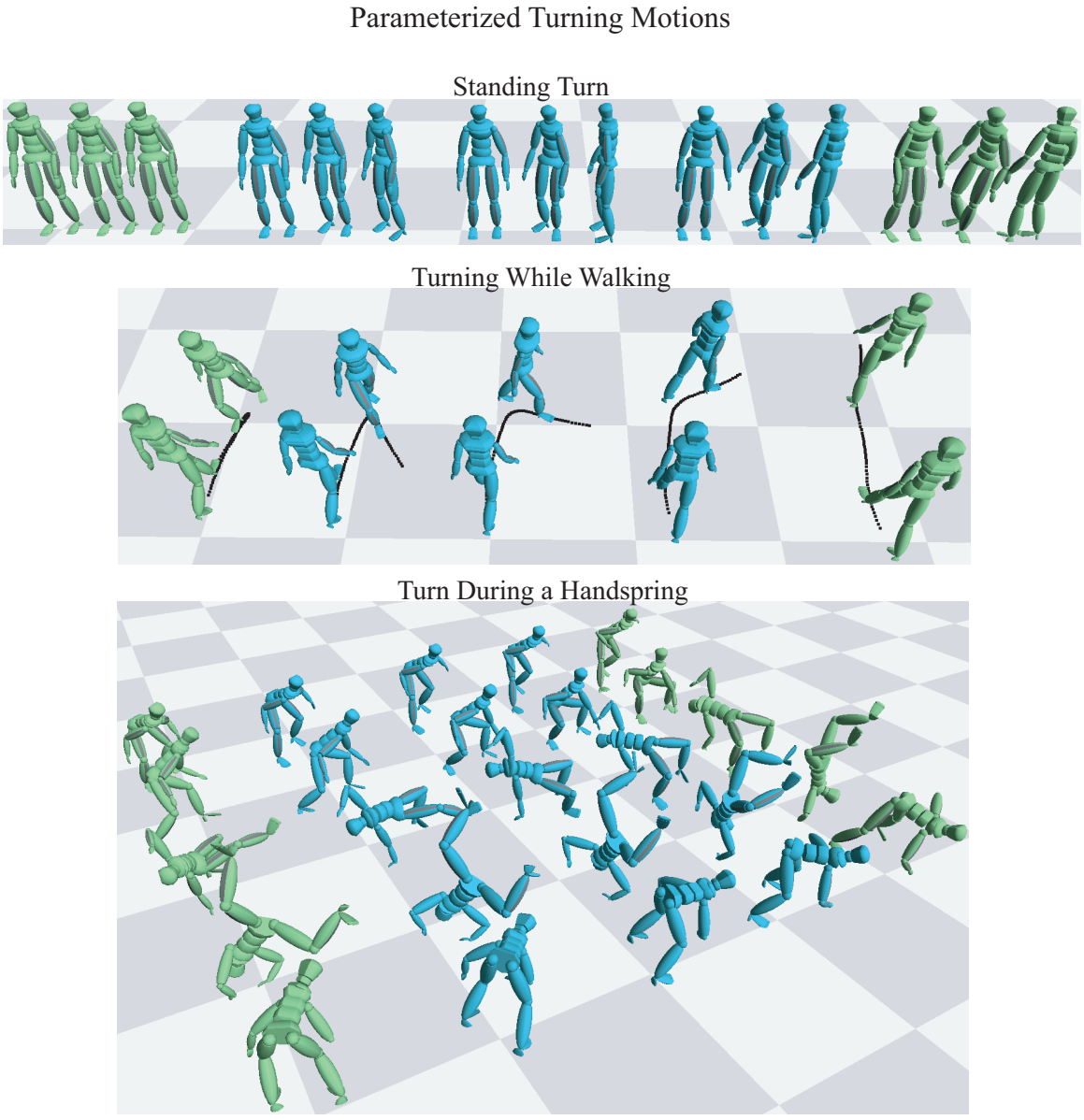

Figure 5.17: Parameterized motions created through interpolation. Original motions are green and interpolations are blue. In each case the parameterization is on the amount the character turns. Blends such as the center one are not possible with existing blending algorithms (even manual ones), because these algorithms can not handle input motions with sharply varying root trajectories.

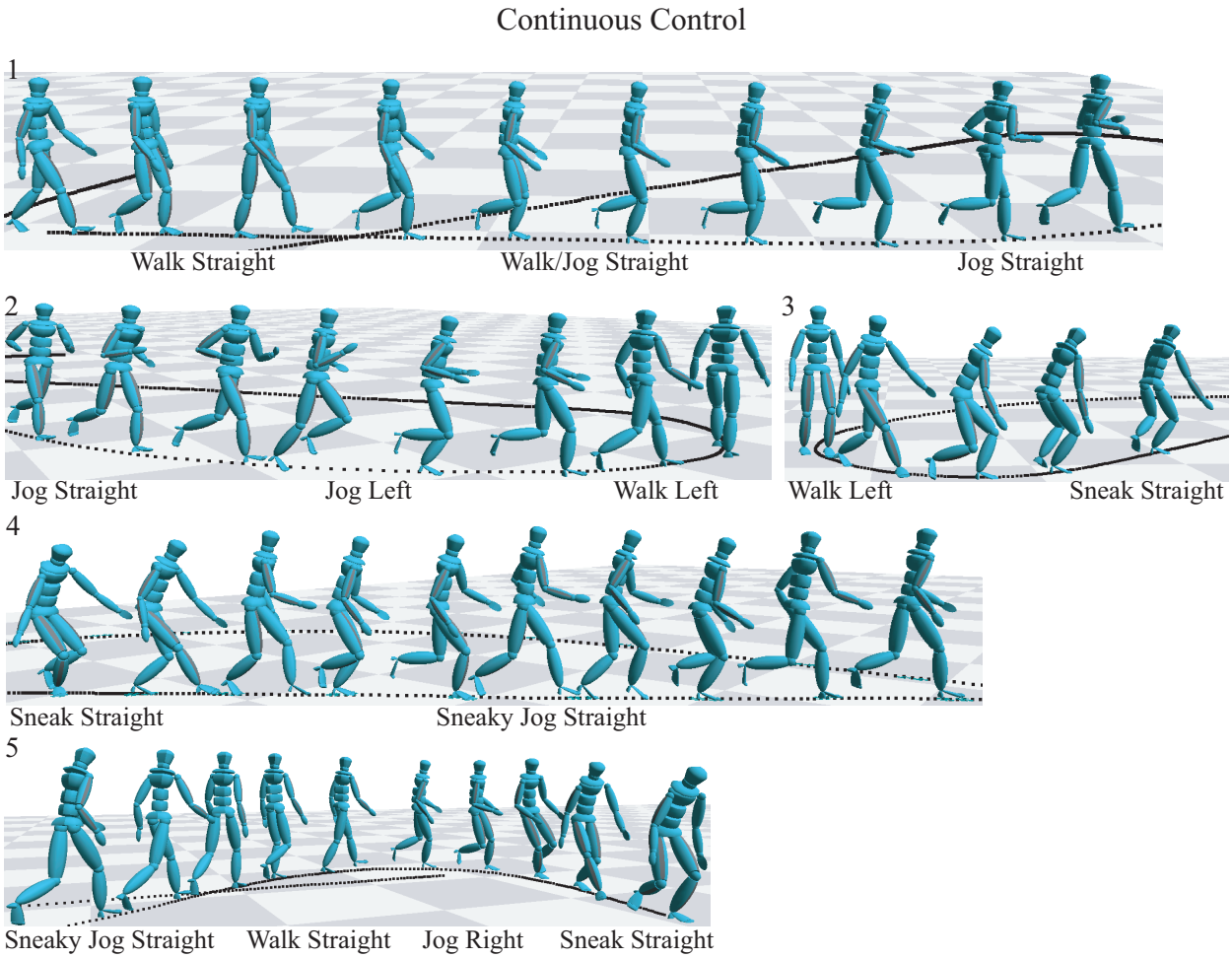

Figure 5.18: Starting with nine variations of locomotion, a registration curve was built and the speed, direction, and style of a character's movement was controlled by continuously varying blend weights. The synthesized motion has been split across several images, as it was too long to be displayed in a single picture.

5.4.3 Continuous Motion Control

For certain sets of input of motions, it makes sense to use blend weights that vary continuously. This can be implemented by generalizing interpolation to use an arbitrary user-specified weight function, rather than a constant function. To demonstrate this application, we obtained a set of nine walking, jogging, and sneaking motions; for each locomotion style there was a motion that travelled straight ahead, one that curved to the left, and one that curved to the right. A registration

curve was constructed and used to continuously control the speed, curvature, and "sneakiness" of a character. An example motion is shown in Figure 5.18.

5.5 Discussion

This chapter has introduced registration curves, a data structure that can be used to automatically generate blends of an arbitrary number of input motions. Registration curves significantly expand the range of motions that can be blended with automatic methods by encapsulating relationships between the timing, root trajectory, and constraint state of the input motions. Moreover, registration curves can seamlessly serve as a back end for common blending operations such as transitioning, interpolation, and continuous control.

As with any blending method, our algorithm is not guaranteed to create realistic blends for arbitrary inputs, and it is therefore important to understand when it is likely to succeed and when it will probably fail. Registration curves assume that structurally related parts of motions look more similar than unrelated parts. In many cases this assumption is valid. For example, corresponding skeletal poses of a locomotion cycle do in fact look more similar than skeletal poses which are out of phase. Similarly, registration curves are appropriate for motions which perform the same action in different styles, such as casually picking up a glass versus forcefully snatching it. On the other hand, in some cases logically corresponding parts of motions have the most *dissimilar* poses. Imagine two motions where a character picks up an object, one where it must stand on its tiptoes to reach to an upper shelf and another where it must bend down to reach near the ground. While the apexes of these reaches are logically identical, these are also the most dissimilar poses in the two motions, and the timewarp curve generated by our method would explicitly avoid matching these frames. However, this does not imply that our framework is unable to handle motions like reaching. If the set of desired motions is sampled sufficiently densely, then for reaches that target nearby locations it will in fact be true that poses at corresponding parts of the reach look the most similar, and correspondences between more distant motions could be inferred using intermediary motions. This idea will be explored in greater depth in Chapter 6.

Although arbitrary blend weights may be specified, some care should be exercised to ensure that synthesized motions retain the quality of the original examples. In our experience convex blend weights are the safest, although modest deviations from convexity can add some extra flexibility without unduly affecting motion quality. It is also important that blend weights not be changed too fast (i.e., during a transition or continuous control), since this effectively introduces a discontinuity into the blend. Acceptable rates of change must ultimately be determined by the application, since one might be willing to accept lower motion quality in order to improve responsiveness.

While blending is more likely to succeed when the input motions are similar (or, at least, when they sample the intended range of variation with reasonable density), ultimately there is no way to be certain without actually computing and evaluating specific blends. One of the biggest advantages of registration curves is that they simplify this process by allowing a user to almost immediately begin experimenting with different blend weights — even for comparatively large input sets, a registration curve can be generated in just a few seconds (Section 5.4), and entire blends can then be computed at interactive rates.