Eurographics/SIGGRAPH Symposium on Computer Animation (2003)
D. Breen, M. Lin (Editors)

Flexible Automatic Motion Blending with Registration Curves

Lucas Kovar[1] and Michael Gleicher[1]

[1] University of Wisconsin, Madison

Abstract

Many motion editing algorithms, including transitioning and multitarget interpolation, can be represented as instances of a more general operation called motion blending. We introduce a novel data structure called a registration curve *that expands the class of motions that can be successfully blended without manual input. Registration curves achieve this by automatically determining relationships involving the timing, local coordinate frame, and constraints of the input motions. We show how registration curves improve upon existing automatic blending methods and demonstrate their use in common blending operations.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Animation

1. Introduction

Motion capture has been a successful technique for creating realistic animations of humans, in large part because of the availability of motion editing methods. Since capture is fundamentally limited to producing a finite set of clips, by itself it offers little flexibility. However, through motion editing these clips can be adapted to meet the demands of a particular situation. Some of the most successful editing methods are based on the idea of *motion blending*, which produces new motions (*blends*) by combining multiple clips according to time-varying weights. Motion blending has several applications. For example, blending can be used to create seamless transitions between motions, allowing one to build lengthy, complicated motions out of simpler actions. Another application is interpolation, or creating motions "in-between" the initial set to produce a parameterized space of motions. Blending operations such as these have significant practical importance and have proven useful in commercial applications like video games [13, 21].

While blending is a powerful tool, it will not produce realistic results unless the input motions are chosen with some care. The range of motions that can be successfully blended depends heavily on how much information about the motions is given to the algorithm. If *nothing* is known about the input other than the raw motion parameters (e.g., the root position and joint angles at each frame), then existing blending algorithms are reliable only if the motions are quite similar.

Our goal is to expand the range of motions that can be successfully blended without requiring manual intervention. Toward this end we introduce *registration curves*. A registration curve is an automatically constructed data structure that encapsulates relationships involving the timing, local coordinate frame, and constraint states of an arbitrary number of input motions. These relationships are used to improve the quality of blended motion and allow blends that were previously beyond the reach of automatic methods.

In the remainder of this paper we describe how to construct and use registration curves. The next section starts with an overview of registration curves. Section 3 reviews related work. Section 4 explains how to construct registration curves, and Section 5 presents a blending algorithm that exploits the information contained in registration curves. Finally, Section 6 demonstrates our technique in common applications and Section 7 concludes with a discussion of the advantages and limitations of registration curves.

2. Overview of Registration Curves

We represent motions in the standard skeletal format. Formally, a motion is defined as a continuous function $\mathbf{M}(f) = (\mathbf{p}_R(f), \mathbf{q}_{1\ldots n}(f))$, where $\mathbf{p}_R$ is the global position of the root and $\mathbf{q}_i$ is the orientation of the i^{th} joint in its parent's coordinate system. Each parameter vector $\mathbf{M}(f)$ is called a *frame* and f is the *frame index*. Motions are assumed to be annotated with constraint information, e.g., that the left heel

should be planted over some range of frames. Constraints may be generated automatically using techniques as in [4]; in our experiments we used automated methods and tuned the results by hand. While motions are continuous functions, in practice one has access only to discrete samples. We map these samples to integer frames and generate skeletal poses at non-integer frames through interpolation.

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 form of $\mathbf{B}(t)$ depends on the blending algorithm, if $w_i = 1$ at t_0 and all the other weights are 0, then $\mathbf{B}(t_0)$ is identical to a frame from $\mathbf{M}_i$. In this framework, a *transition* involves two motions and a weight function that starts at $(1,0)$ and smoothly changes to $(0,1)$, and an *interpolation* combines an arbitrary number of motions according to a constant weight function.

When the only available information is the input motions themselves, the 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. Specifically, the blended root position is calculated either by averaging root positions or velocities, and blended joint angles can be calculated by averaging Euler angles or computing the exponential map of averaged logarithmic maps [14]. Constraints are not handled directly by linear blending, although often they are enforced as a postprocess.

Linear blending produces reasonable results when the input motions are sufficiently similar. Our strategy is thus to still combine frames via averaging, but to automatically extract 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 *timewarp curve*, a *coordinate alignment curve*, and a set of *constraint matches*, which together form a registration curve. The rest of this section provides an overview of these data structures and how they are merged into a registration curve.

2.1. Timing

Linear blending can fail when motions have different timing, that is, when corresponding events occur at different absolute times. See Figure 1 for examples and Bruderlin and Williams [5] for a discussion. A solution is to *timewarp* the input motions so corresponding events occur simultaneously. This involves determining a timewarp curve $\mathbf{S}(u)$ that returns sets of frame indices, one from each input motion, such that the corresponding frames are all logically related. For example, given a walking motion and a jogging motion, $\mathbf{S}(u)$ would be constructed so $S_1(u)$ and $S_2(u)$ are at the same point in the locomotion cycle. It is also useful if the timewarp curve is continuous and strictly increasing (that is, if $u_1 > u_0$ then $S_i(u_1) > S_i(u_0)$). If these properties hold, then the inverse functions $u(S_i)$ are well-defined, and given a frame of an input motion we can uniquely determine the related point

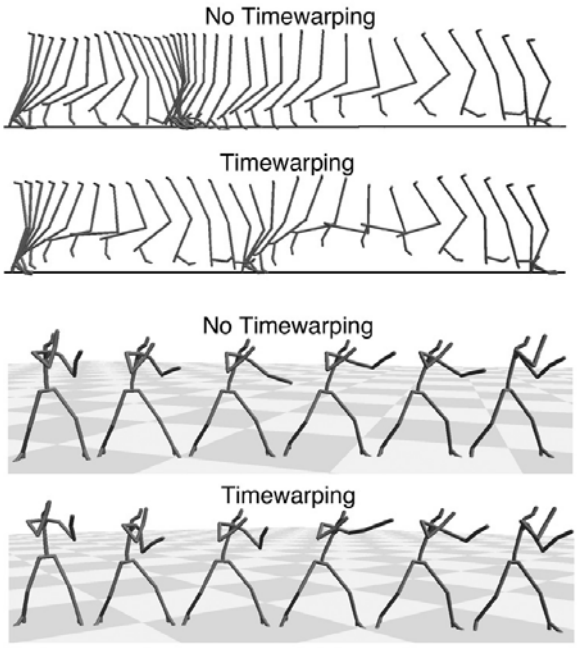

Figure 1: ***Top:*** *A transition between walking and jogging that spans two locomotion cycles. For clarity, only the right leg is shown. Without timewarping, out-of-phase frames are combined and the character floats above the ground with its legs nearly straight.* ***Bottom:*** *an interpolation between a jab and a slower, stronger punch. Without timewarping, the character's arm wobbles.*

on the timewarp curve and vice-versa. The parameter u then defines a reference time system: each input motion may be thought of as a realization of some canonical motion, and when this canonical motion is at frame u then $\mathbf{M}_i$ is at frame $S_i(u)$ and vice-versa.

A registration curve is built around a continuous, strictly increasing timewarp curve $\mathbf{S}(u)$. This timewarp curve is generated automatically for an arbitrary number of motions; details are in Section 4.2. During blending, the set of frames combined at any given time always corresponds to some point on $\mathbf{S}(u)$. Since we have no information other than the input data to work with, to create the timewarp curve we use the heuristic that frames are more likely to correspond if they are similar. This complements the way we actually combine frames: since it is more reliable to average skeletal parameters when poses are similar, we time-align motions so corresponding frames are as similar as possible.

2.2. Coordinate Frame Alignment

Linear blending can fail even when frames occurring at the same time are quite similar. Imagine that we have two walking motions that are in phase, one curving to the left and the other to the right (Figure 2). Since the root position is linearly interpolated, the blended root path collapses. Also, the

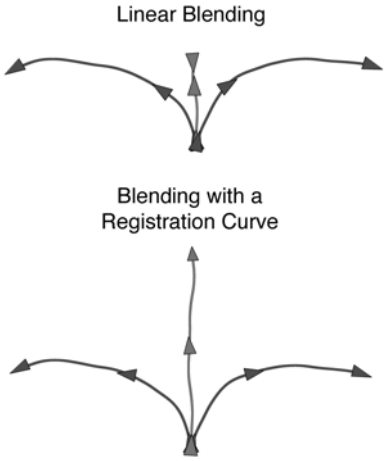

Figure 2: *An interpolation of two walking motions that curve in different directions. The black paths show the projection of the root onto the ground, and the triangles indicate the position and orientation of the root at selected frames. Linear blending causes the root trajectory to collapse. Note that the root orientation changes suddenly at the end of the motion, when the accumulated angle between the roots exceeds* 180°.

character suddenly flips around near the end of the motion; this is because standard orientation averaging schemes like spherical linear interpolation have discontinuities when the angles change by more than 180°.

To solve this problem, we observe that motions are fundamentally unchanged by rotations about the vertical axis and translations in the floor plane [7, 9, 11]. For example, a walking motion looks the same whether it starts at the origin and heads east or starts ten feet to the side and heads southwest. This implies that we are free to apply rigid $2D$ transformations to a motion, effectively changing its local coordinate frame. In particular, we can choose transformations that make motions as similar as possible at specified frames. We refer to this process as *coordinate frame alignment*.

Coordinate frame alignment can successfully blend motions with nontrivially distinct paths. Say we are blending the frames $\mathbf{M}_1(f_1),\ldots,\mathbf{M}_n(f_n)$. These frames are aligned to form a single, abstract frame $\mathcal{F}$ which may itself be rigidly transformed. Each $\mathbf{M}_i$ votes on the position and orientation of $\mathcal{F}$, and these votes are combined according to the blend weights. This allows the blended root trajectory to be determined incrementally. Once $\mathcal{F}$ is situated, the final blended frame is a weighted average of the input frames. Registration curves implement this idea by containing alignment curves. An alignment curve is a function $\mathbf{A}(u)$ that gives a set of transformations which align the frames at each point $\mathbf{S}(u)$ on the timewarp curve. Section 4.3 explains how to construct an alignment curve and Section 5.2 provides details on using them in blending. Figure 2 shows that blending with an alignment curve correctly handles the example at the start of this section.

2.3. Constraint Matches

Since we combine frames via averaging, constraints on the blend may not be satisfied. However, if we can simply determine what these constraints are, then a variety of methods exist for adjusting the motion to enforce them [6, 12, 10]. Even if the blend looks fine, it is still desirable to know its constraints since it may be further edited.

Determining constraints on the blend takes 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. Even if the motions are timewarped, the intervals over which constraints are active will not match. Also, we may be interested in blending motions that have different numbers of constraints. For instance, to create a space of parameterized turning motions we might interpolate a motion where a character stands still with one where it turns in place (and hence picks up its feet).

If we represent the interval of a constraint in terms of the global time parameter u, then it is reasonable to presume that constraints which are nearby "mean" roughly the same thing, even if they do not span identical regions. Registration curves employ this idea to create constraint matches. Each constraint match contains a set of related constraints, one from each motion. The start and end of these constraints in global time are treated as parameters which may themselves be blended. Our algorithm for finding constraint matches is discussed in detail in Section 4.4.

3. Related Work

Motion blending has been an integral part of several research efforts. One of the earliest was due to Perlin [15], who presented a real-time system based on procedurally-generated motions. After a user manually constructed base motions, blending operations were used to create new motions and transition between existing ones. The language for constructing motions contained built-in timing and constraint models, simplifying the related blending issues. Several systems have leveraged multitarget blending to create intuitively parameterized spaces of motions [23, 18, 20]. Multiple research groups have used general blending (that is, blending with arbitrary weight functions) to provide continuous control of locomotion [8, 2, 14]. Blend-based transitions have been incorporated into various systems for graph-based motion synthesis [18, 9].

Techniques for certain blending operations have been developed outside of the framework presented in Section 2. Instead of averaging skeletal parameters directly, Unuma et al. [22] used weighted averages in the Fourier domain. Rose et al. [19] used spacetime optimization to create transitions that minimize joint torque. Other researchers [11, 1, 16] have gener-

ated transitions by concatenating motions and eliminating discontinuities with displacement mapping techniques [24].

Timewarping has been an important part of previous work on motion blending. Some systems required the user to mark sparse "key times" in the input motions, which were linearly interpolated to form a timewarp curve [18, 14]. Our technique requires no user intervention. Ashraf and Wong [2] semi-automatically labelled motions with sparse timing information, such as zero-crossings of the second derivative of user-specified joint angles. Timewarp curves were generated through a greedy algorithm that matched identical labels. Our algorithm generates dense correspondences and uses a dynamic programming technique with stronger optimality properties. The method of Bruderlin and Williams [5] is the most similar to our own, as it also is based on dynamic programming. However, their algorithm will not in general yield a strictly increasing timewarp curve, and it is only designed to handle two motions. We extend this method to produce strictly increasing timewarp curves for an arbitrary number of motions.

Most previous motion blending efforts used linear interpolation to compute root parameters. A notable exception is the work of Park et al. [14], where the root was positioned and oriented according to a user-specified path (Gleicher [7] used a similar method to interactively edit individual motions). Blending was then used to determine a skeletal pose appropriate for the local speed and curvature of the root trajectory. We blend root parameters directly, so there is no need to specify the root path. Also, Park et al.'s method was based on the assumption that the input motions were sufficiently smooth that the root path could be well approximated by a circular arc. Our method is applicable to motions with sharply varying path curvatures (see Figure 8).

Bindiganavale and Badler [4] developed a method for automatically detecting constraints satisfied by a motion. However, for blends we are interested in the constraints that *should* be satisfied, not the ones that *are* satisfied. Rose et al [18] manually specified the starting and ending times of corresponding constraints in the input motions and blended the interval bounds. Our method is similar, except corresponding constraints are identified automatically. Ashraf and Wong [3] and Kovar et al. [9] copied the constraints from the input motion with the currently highest blending weight. This approach works well for the special case of transitions, but can produce artifacts with other blending operations such as interpolation. In particular, when the blend weights are nearly equal a small change in the weights may produce large changes in the constraints. Our technique ensures that constraints change smoothly if the blend weights change smoothly.

4. Constructing Registration Curves

To build a registration curve, we start by constructing a timewarp curve $\mathbf{S}(u)$. An alignment curve $\mathbf{A}(u)$ is then built around $\mathbf{S}(u)$, and constraint matches are identified by using $\mathbf{S}(u)$ to map constraint intervals into a standard time frame. Fundamental to our approach is a function $\mathbf{D}(\mathbf{F}_1, \mathbf{F}_2)$ which simultaneously determines the "distance" between two frames of motion and an aligning rigid $2D$ transformation. We now describe $\mathbf{D}$ and discuss the creation of the timewarp curve, the alignment curve, and the constraint matches.

4.1. A Coordinate-Invariant Distance Function

We use the same distance function as Kovar et al. [9]; see this paper for a discussion of its motivation. Briefly, computing $\mathbf{D}(\mathbf{F}_1, \mathbf{F}_2)$ involves three steps (Figure 3). First, small windows of nearby frames are extracted around $\mathbf{F}_1$ and $\mathbf{F}_2$. We used neighborhoods of five frames, which has the effect of incorporating velocity and acceleration information. Next, each frame is converted to a point cloud by attaching markers to the joints of the skeleton, forming two larger point clouds. Finally, the optimal sum of squared distances between corresponding points is computed over all rigid $2D$ transformations of the second point cloud. That is, we compute

$$\mathbf{D}(\mathbf{F}_1, \mathbf{F}_2) = \min_{\theta, x_0, z_0} \sum_{i=1}^{n} w_i \|\mathbf{p}_i - \mathbf{T}_{\theta, x_0, z_0} \mathbf{p}'_i\|^2 \qquad (1)$$

where $\mathbf{p}_i$ and $\mathbf{p}'_i$ are respectively the i^{th} point of the first and second point clouds, $\mathbf{T}_{\theta, x_0, z_0}$ is a rotation by θ about the y (vertical) axis followed by a translation in the floor plane by (x_0, z_0), and w_i may be used to preferentially weight different joints. In our experiments the w_i were equal and each point cloud had 200 points (40 points per frame for five frames).

Assuming the w_i sum to unity, the optimal solution to Equation 1 is achieved under the following transformation:

$$\theta = \tan^{-1}\left(\frac{\sum_i w_i(x_i z'_i - x'_i z_i) - (\overline{x}\overline{z'} - \overline{x'}\overline{z})}{\sum_i w_i(x_i x'_i + z_i z'_i) - (\overline{x}\overline{x'} + \overline{z}\overline{z'})}\right) \qquad (2)$$

$$x_0 = \overline{x} - \overline{x'_i}\cos\theta - \overline{z'_i}\sin\theta \qquad (3)$$

$$z_0 = \overline{z_i} + \overline{x'_i}\sin\theta - \overline{z'_i}\cos\theta \qquad (4)$$

where we use the shorthand notation $\overline{\alpha} = \sum_{i=1}^{n} w_i \alpha_i$.

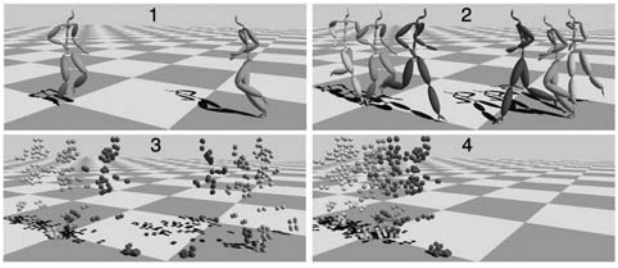

Figure 3: *Computing* $\mathbf{D}(\mathbf{F}_1, \mathbf{F}_2)$.

4.2. Creating the Timewarp Curve

The timewarp curve is generated by creating a dense set of frame correspondences, fitting a spline to these frame correspondences, and adjusting the spline so it is strictly increasing. We first consider the case of just two motions and then generalize to more motions.

4.2.1. Timewarp Curves for Two Motions

Using the distance function in Equation 1, we can create a grid where columns correspond to frames from the first motion, rows correspond to frames from the second motion, and each cell contains the distance between the corresponding pair of frames. Given two cells in this grid, we can use the *dynamic timewarping* algorithm to find a minimal-cost connecting path (Figure 4), where the cost of a path is found by summing its cells. This path corresponds to an optimal time-alignment that starts and ends at the bounding cells. We only consider paths having three properties:

1. **Continuity.** Each cell on the path must share a corner or edge with another cell on the path.
2. **Causality.** Paths must not reverse direction; they should either go entirely forward or entirely backward in time.
3. **Slope Limit.** At most L consecutive horizontal steps or consecutive vertical steps may be taken.

These properties are illustrated in Figure 4. Slope limits are useful because very large slopes typically indicate that the motions are (at least locally) so different that there are no good frame matches. In this case it is preferable to limit the timewarping and accept the fact that corresponding poses will not be particularly similar. We have found a slope limit of 2 or 3 to produce good results.

Dynamic timewarping is based on well-known dynamic programming methods; refer to the literature [17, 5] for details. From a given starting point, dynamic timewarping can be used to find optimal paths leading to all points on the boundary of the grid. The only question is which boundary point to select. In some cases, this will be known *a priori*. For example, if the entire motions are to be time-aligned — that is, both the first frames and the last frames must correspond — then the path must pass through the lower-left and upper-right corners of the grid. In many cases, however, there is no preferred boundary point. In this case we select the one that minimizes the average cell cost.

The frame correspondences generated by dynamic timewarping could be used directly to form a piecewise-linear timewarp curve. However, though monotonic, this curve would not be 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 timewarp curve can be, they do not ensure any smoothness properties. For these reasons we instead create the timewarp curve by fitting a smooth, strictly increasing function to the frame correspondences. If we only cared about smoothness, then a simple solution would be

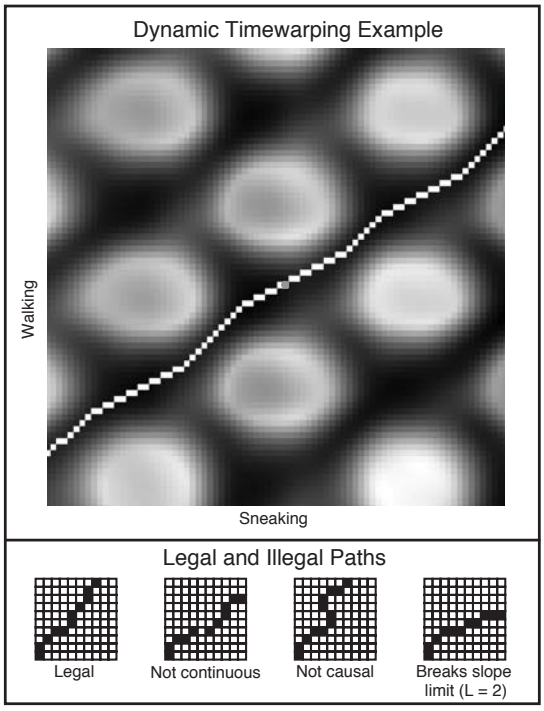

Figure 4: ***Left:*** *Dynamic timewarping applied to a walking motion and a sneaking motion. The background image shows frame distances; larger distances correspond to whiter cells. The central grey circle is the starting point and the generated path is in bright white. The local slope variations arise because the character pauses slightly at each footstep in the sneaking motion but travels at a steady pace in the walking motion.* ***Right:*** *Legal and illegal dynamic timewarping paths ($L = 2$).*

to fit a spline, which only requires solving a linear system. Adding in the constraint that the spline be strictly increasing produces a significantly more expensive quadratic programming problem. However, given the causality and slope limit restrictions, it is likely that a spline fit without any constraints will be *approximately* strictly increasing.

In light of this, our strategy is to fit a uniform quadratic B-spline as an unconstrained optimization and then adjust the knots. The result is the final timewarp curve $\mathbf{S}(u)$. Due to the convex hull property of B-splines, we need only ensure that the knots $\mathbf{p}_1, \ldots, \mathbf{p}_n$ are strictly increasing. We do this by selecting a "base" motion $\mathbf{M}_{base}$ and for the other motion $\mathbf{M}_j$ enforcing the property

$$\varepsilon \leq \frac{p_{i,j} - p_{i-1,j}}{p_{i,base} - p_{i-1,base}} \leq \frac{1}{\varepsilon}, \quad 1 < i \leq n \tag{5}$$

for some $\varepsilon > 0$. It is easy to show that this implies $\varepsilon^2 \leq \frac{dS_i}{dS_j} \leq \frac{1}{\varepsilon^2}$. Appendix A gives an algorithm for enforcing Equation 5. $\mathbf{M}_{base}$ is selected implicitly by running this algorithm with

each motion as the base and keeping the result with the smallest adjustment.

4.2.2. Timewarp Curves For More Than Two Motions

Directly applying the methods of Section 4.2.1 to more than two motions is computationally expensive. The generalization of Equation 1 yields a nonlinear optimization for which there is no closed-form solution, and since dynamic timewarping requires filling a k-dimensional grid for k motions, it scales exponentially. However, we can design a simple and efficient algorithm by combining multiple timewarp curves relating pairs of motions. Say we have three motions $\mathbf{M}_1$, $\mathbf{M}_2$, and $\mathbf{M}_3$ and timewarp curves $\mathbf{S}_{1\to2}$, $\mathbf{S}_{1\to3}$, and $\mathbf{S}_{2\to3}$, one for each pair. Then if $\mathbf{S}_{1\to2}(f_1) = f_2$ and $\mathbf{S}_{1\to3}(f_1) = f_3$, it is likely that $\mathbf{S}_{2\to3}(f_2)$ is also approximately f_3. Hence we might simply remove $\mathbf{S}_{2\to3}$ and combine $\mathbf{S}_{1\to2}$ and $\mathbf{S}_{1\to3}$ into a single timewarp curve.

More generally, given a set of motions $\mathbf{M}_1,\ldots,\mathbf{M}_k$, a motion $\mathbf{M}_{ref}$ is selected to serve as a reference. $\mathbf{M}_{ref}$ may be 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 the corresponding frames generated by dynamic timewarping. 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 $u(S_{i,1})$ to determine the corresponding frames in each other motion. Specifically, the i^{th} motion's frame is $S_{i,2}(u(S_{i,1}))$. To form the final timewarp curve, we fit a k-dimensional quadratic spline to these samples and adjust the knots to enforce Equation 5 (see Appendix A), just as in the two-dimensional case.

4.3. Creating the Alignment Curve

Given the timewarp curve, the alignment curve is simple to construct. Consider first the case of two motions. For each frame correspondence generated via dynamic timewarping, Equation 1 specifies a rigid $2D$ transformation $\{\theta_i, x_{0_i}, z_{0_i}\}$ that aligns the second frame with the first. A $3D$ quadratic spline is 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 θ_i 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 and then back again. We have found it useful to treat these spikes as high-frequency noise and remove them with a short median filter (kernel width of 3 to 5) 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.

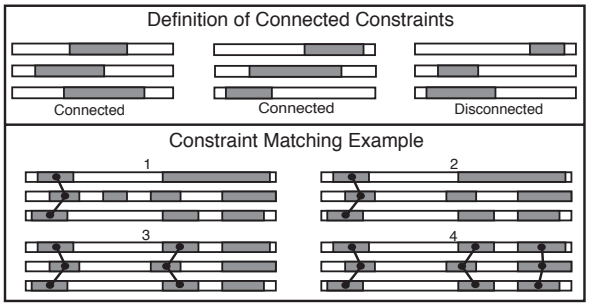

Figure 5: ***Top:*** *Examples of connected and disconnected constraints.* ***Bottom:*** *The steps taken by the constraint matching algorithm on a sample input. Note that a constraint is removed from the second motion and split in the first motion.*

The case of more than two motions is handled in direct analogy to Section 4.2.2. We start with a timewarp curve for the entire set of motions and a collection of alignment curves relating pairs of motions. A reference motion is selected and frames are sampled from it. For each sample, the timewarp curve is used to create a set of corresponding frames, and from the alignment curves we find a set of mutually aligning coordinate transformations. A spline is then fit to these samples as in the two-motion case.

4.4. Identifying Constraint Matches

The final step in building a registration curve is to identify constraint matches. Each kind of constraint is treated independently (e.g., we might first consider left heelplants, then right toeplants, and so on). For each motion, the intervals of a given kind of constraint are assumed to be disjoint. The algorithm starts by mapping the duration of each constraint into a standard time frame via the timewarp curve. It then groups constraints into constraint matches based on two guidelines. First, each constraint match must contain exactly one constraint from each motion. Second, the elements of a constraint match must be *connected* in the sense that the union of all the constraint intervals must form a single continuous interval (Figure 5). This is based on the heuristic that corresponding constraints ought to occur at similar points in the motions.

Some constraints may not be able to belong to any constraint match; for example, a constraint may overlap with no other constraints. In this case that constraint is eliminated. That is, if a constrained region is logically associated with an unconstrained region, that constraint is broken. To illustrate, say we blend a motion where a character stands on its left leg with a motion where it stands on both legs. We expect the character to stand on its left leg and vary the height of his right foot according to the blend parameters — that is, the constraint on the right foot from the second motion

should be broken. For similar reasons, if a constraint of one motion overlaps two or more constraints of another motion, we consider adding a gap to it, thereby splitting it into two shorter constraints.

Let $C_{i,j}$ be the j^{th} constraint of 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 constraints of each motion are connected. 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 we can form a constraint match from these constraints. We first decide whether any of them should be split (see Appendix B), and then we build a constraint match from the constraints that were not split and the first portion of the constraints that were split. Figure 5 shows our constraint matching algorithm on a sample input.

5. Blending With Registration Curves

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

1. Determine a position $\mathbf{S}(u_i)$ on the timewarp curve
2. Position and orient the frames at $\mathbf{S}(u_i)$.
3. Combine the frames based on the blending weights $\mathbf{w}(t_i)$.
4. Determine the constraints on the resulting frame

We assume that for some frame $\mathbf{B}(t_0)$, u_0 is known. If this happens to be the first frame of the blend, then we generate in order $\mathbf{B}(t_1)$, then $\mathbf{B}(t_2)$, and so on. If $\mathbf{B}(t_0)$ is *not* the first frame, we first generate these forward frames and then create $\mathbf{B}(t_{-1}), \mathbf{B}(t_{-2}), \ldots$. We require that $u_i > u_{i-1}$, i.e., moving forward in time is equivalent to moving forward in u.

We now discuss how to create an individual frame of the blend. To simplify matters, we only discuss the case of moving forward in u; moving backward is handled similarly.

5.1. Advancing Along the Timewarp Curve

For $\mathbf{B}(t_0)$, this step is skipped, since u_0 is given to us. Otherwise, we are at some point $\mathbf{S}(u_{i-1})$ on the timewarp curve and want to move to a new location $\mathbf{S}(u_i)$. Consider the situation where $w_j(t) = 1$ and the other weights are always 0. In this case $\mathbf{B}$ should be identical to $\mathbf{M}_j$, and so time should flow such that $\mathbf{M}_j$ is played at its normal rate. So, for example, if we wanted to advance Δt units of time, then we should pick $\Delta u = u_i - u_{i-1}$ such that $S_j(u_{i-1} + \Delta u) - S_j(u_{i-1}) = \Delta t$. Taking the limit as $(\Delta t, \Delta u) \to 0$ and exploiting the fact that $S_j(u)$ and $u(t)$ are both strictly increasing, this yields $\frac{du}{dt} = \frac{du}{dS_j}$.

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

$$\frac{du}{dt} = \sum_{j=1}^{k} w_j(t) \frac{du}{dS_j} \tag{6}$$

Intuitively, each motion votes on how fast the global time parameter u should flow, and these votes are combined based upon the blend weights. In general this differential equation must be solved numerically. However, since the time step between frames is small and $\mathbf{w}$ and $\mathbf{S}$ are smooth, we have found that we can advance a frame in a single Euler step:

$$\Delta u = \left(\sum_{j=1}^{k} w_j(t_{i-1}) \frac{du}{dS_j} \right) \Delta t \tag{7}$$

A similar strategy was used in [14]. For convex weights, which are commonly viewed as natural choices for blending operations, Equation 7 will always yield $\Delta u > 0$ and hence time will always flow forward. Systems which do not limit themselves to arbitrary blend weights can replace $w_j(t_{i-1})$ with $\frac{|w_j(t_{i-1})|}{\sum_r |w_r(t_{i-1})|}$ in Equation 7, ensuring $\Delta u > 0$.

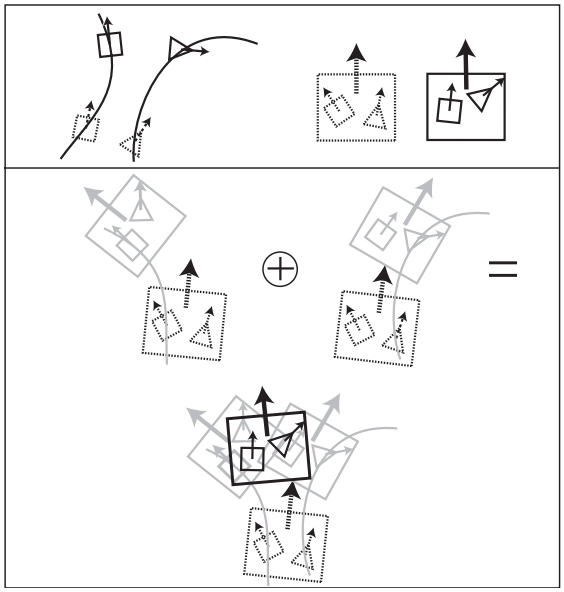

Figure 6: ***Top:*** *Given any set of corresponding frames, the coordinate alignment curve specifies rigid 2D transformations that mutually align them. This group of aligned frames can itself be rigidly transformed.* ***Bottom:*** *Based on the position and orientation of the previous blended frame, each motion votes on a new position and orientation for the current blended frame. These votes are combined according to the blend weights; here we show the case* $w_1 = w_2 = 0.5$.

5.2. Positioning and Orienting Frames

Now that the value of u_i is known, we extract the frame $\mathbf{M}_j(S_j(u_i))$ from the j^{th} motion and transform it by $\mathbf{A}_j(u_i)$. This yields a set of mutually aligned frames that, just as with a single frame of motion, we are free to position and orient on the ground plane (Figure 6). Our task is to find an appropriate rigid 2D transformation $\mathbf{T}(t_i)$ to apply to this group of frames, making $\mathbf{T}(t_i)\mathbf{A}_j(u_i)$ the total transformation applied to $\mathbf{M}_j(S_j(u_i))$. For the first frame of the blend, $\mathbf{T}(t_0)$ may be

chosen arbitrarily. For the other frames, we need to choose $\mathbf{T}(t_i)$ so the position and orientation of $\mathbf{B}(t_i)$ is consistent with the preceding frame. To see how to do this, consider the case where $w_j(t)$ is 1 and the other blending weights are 0 for all $t > t_{i-1}$. In this case the remainder of the blend should 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 we define

$$\Delta\mathbf{T}_j(t_i) = \mathbf{T}(t_{i-1})\mathbf{A}_j(u_{i-1})\mathbf{A}_j{}^{-1}(u_i) \tag{8}$$

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

More generally, we compute $\mathbf{T}(t_i)$ by averaging the coordinate transformations $\Delta\mathbf{T}_j(t_i)$ according to the $w_j(t_i)$; see Figure 6. Each motion votes for the $\mathbf{T}(t_i)$ that leaves its coordinate system unchanged, and we average these votes according to the blend weights. To perform this averaging, we need to choose parameters to represent $\Delta\mathbf{T}_j(t_i)$. We start by choosing an origin that is near the transformed frames. This is done by transforming each $\mathbf{M}_j(S_j(u_i))$ by $\Delta\mathbf{T}_j(t_i)\mathbf{A}_j(u_{i-1})$, projecting the root position onto the ground, and averaging over all the motions. $\Delta\mathbf{T}_j(t_i)$ 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). Finally, $\mathbf{T}(t_i)$ is:

$$\mathbf{T}(t_i) = \left\{ \sum_j w_j\phi_j, \left(\sum_j w_j x_j, \sum_j w_j z_j \right) \right\} \tag{9}$$

5.3. Making the Blended Frame

Using these transformed frames, we compute the skeletal pose of the blend frame by computing a weighted average of the root positions and averaging the joint orientations using the method presented by Park et al [14]. Our final task is to determine the constraints on this new frame. For each constraint match $\mathcal{M}$ we determine an interval $I_{\mathcal{M}}$ based on the current blend weights. If the i^{th} constraint C_i of $\mathcal{M}$ is active over the interval $[C_i^s, C_I^e]$, then $I_{\mathcal{M}} = [\sum w_i C_i^s,\ \sum w_i C_i^e]$. If u_i is inside this interval, then the corresponding constraint is applied to the blend frame. Actually enforcing the constraints is up to the implementation; we used the method of Kovar et al [10].

6. Results and Applications

The most expensive part of constructing a registration curve is filling in the dynamic timewarping grid. Computing the full grid for two 20 second motions (600 frames each and 360,000 cell evaluations) takes 3.43 seconds on a 1.3 gHz Athlon processor using point clouds with two hundred points. The remainder of the construction process takes 0.03 seconds. A registration curve is computed once and stored for future blending operations. A full 5 second output motion (150 frames) can be generated at interactive rates.

Registration curves fit seamlessly into common blending applications. The rest of this section discusses using registration curves for transitioning, interpolating, and continuous motion control.

6.1. Transitions

To create a transition, a user specifies its length and where it takes place. Our algorithm thus takes as input two frames $\mathbf{M}_1(f_1)$ and $\mathbf{M}_2(f_2)$ defining the center of the transition plus the half-width h of the transition (the entire width is $2h+1$ frames). $\mathbf{M}_1(f_1)$ and $\mathbf{M}_2(f_2)$ serve as the starting point for the dynamic timewarping algorithm (Section 4.2.1).

To find the starting position $\mathbf{S}(u_0)$ of the blend on the timewarp curve, we average the u values that correspond to $\mathbf{M}_1(f_1)$ and $\mathbf{M}_2(f_2)$: $u_0 = \frac{1}{2}(u(S_1(f_1)) + u(S_2(f_2)))$. We then generate h frames going both forward in time and backward in time, using a weight function that smoothly changes from $(0,1)$ to $(1,0)$. An example transition is shown in Figure 7.

In general, the first and last frames of the transition will not occur at integer times, and so $\mathbf{M}_1$ and $\mathbf{M}_2$ have to be resampled if they are to connect smoothly to the transition boundaries. If this is undesirable, then we can force the boundaries to occur at integer times by generating $u_{-h},\ldots,u_h$ as described in Section 5.1, 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.

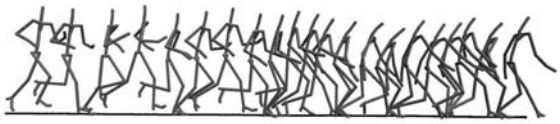

Figure 7: *A transition between jogging and a significantly slower sneaking motion.*

6.2. Interpolations

An interpolation can be created by specifying a set of motions and a fixed set of blend weights. It is presumed that the entirety of each input motion is to be blended, which means that when we execute the dynamic timewarping algorithm we force it to match the first frames and last frames together. Once a registration curve for these motions is built, we create the blend 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.

We have used interpolations to generate several parameterized clips (see Figure 8), including ones that allow control of the angle of a turn (both in place and while walking), the direction and force of a punch, and the "snap" and follow-through of a lunging front kick.

6.3. Continuous Motion Control

For certain kinds of motions, it makes sense to use blend weights that vary continuously. This is a simple generalization of interpolation; we merely replace the constant weight function with a user-specified weight function. To demonstrate this application, we obtained a set of 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. We generated a registration curve out of these nine motions and used it to continuously control the speed, curvature, and "sneakiness" of a character. An example motion is shown in Figure 9.

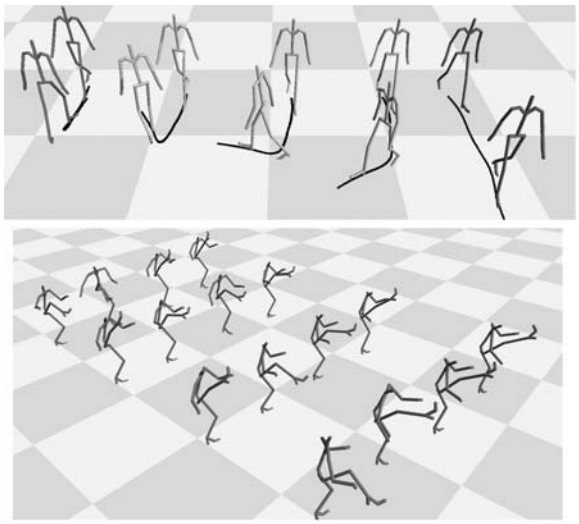

Figure 8: ***Top:*** *a parameterized clip built from a straight walk (far right) and one with a sharp 180 degree turn (far left) is used to create sharp path changes in a continuum of directions.* ***Bottom:*** *a parameterized clip built from three kicking motions is sampled to create a large number of similar but distinct kicks.*

7. Discussion

In this paper we have introduced registration curves, a tool for automatically generating blends between an arbitrary number of motions. Registration curves improve upon existing automatic methods by explicitly addressing differences in timing, path, and constraint state, expanding the range of motions that can be successfully blended. At the same time, registration curves offer a simple interface for common blending operations like transitions, interpolations, and continuous control.

While we have presented registration curves as a tool for manipulating motion capture data, they work equally well with other motion sources, such as keyframed or physically modelled animation. Moreover, our method is readily adapted to data that is not in skeletal format, since the time-warping and coordinate frame alignment algorithms only require point data.

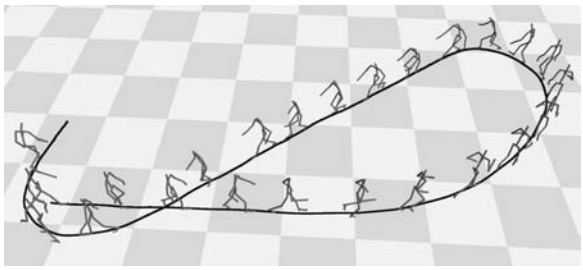

Figure 9: *A sample motion from a registration curve built automatically for nine motions. These input motions were of a character walking, jogging, and sneaking at different curvatures. By continuously varying blends weights, a user was able to control the speed, curvature, and sneakiness of the character.*

No blending method is perfect, and so it is 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. For a large class of motions, this assumption is valid. Corresponding points in a locomotion cycle, for example, do in fact look more similar than frames 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. Say we have two motions where a character reaches for a glass, but in the first it reaches above its head and in the second it bends down to the ground. While the apexes of each reach are logically identical, they are also the most dissimilar poses in the two motions. In this case the timewarp curve our method generates would not be nearly as accurate as manually labelling correspondences. 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 motions which reach to nearby locations do in fact look most similar at corresponding parts of the reach. Using multiple registration curves, the entire space could then still be spanned.

Our method currently does not enforce any physical constraints like balance. However, we expect these physical requirements could be enforced as a post-process once a blend is generated. Incorporating physical constraints into our methods is left for future work.

Acknowledgements

We thank House of Moves, Demian Gordon, and The Ohio State University for donating motion capture data. This work was supported in part by NSF grants CCR-9984506 and

CCR-0204372, along with equipment donations from Intel. Lucas Kovar is supported by an Intel Foundation Fellowship.

References

1. O. Arikan and D. A. Forsyth. Interactive motion generation from examples. In *Proceedings of ACM SIGGRAPH 2002*, Annual Conference Series. ACM SIGGRAPH, July 2002.
2. G. Ashraf and K. C. Wong. Generating consistent motion transition via decoupled framespace interpolation. *Computer Graphics Forum*, 2000.
3. G. Ashraf and K. C. Wong. Constrained framespace interpolation. In *Computer Animation 2001*, 2001.
4. Rama Bindiganavale and Norman Badler. Motion abstraction and mapping with spatial constraints. In *Modelling and Motion Capture Techniques for Virtual Environments, CAPTECH'98*, pages 70–82, November 1998.
5. A. Bruderlin and L. Williams. Motion signal processing. In *Proceedings of ACM SIGGRAPH 95*, Annual Conference Series, pages 97–104. ACM SIGGRAPH, August 1995.
6. M. Gleicher. Retargeting motion to new characters. In *Proceedings 0f ACM SIGGRAPH 98*, Annual Conference Series, pages 33–42. ACM SIGGRAPH, July 1998.
7. M. Gleicher. Motion path editing. In *Proceedings 2001 ACM Symposium on Interactive 3D Graphics*. ACM, March 2001.
8. S. Guo and J. Roberge. A high-level control mechanism for human locomotion based on parametric frame space interpolation. In *Proceedings of Eurographics Workshop on Computer Animation and Simulation '96*, 1996.
9. L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *Proceedings of ACM SIGGRAPH 2002*, Annual Conference Series. ACM SIGGRAPH, July 2002.
10. L. Kovar, J. Schreiner, and M. Gleicher. Footskate cleanup for motion capture editing. In *Proceedings of ACM SIGGRAPH Symposium on Computer Animation 2002*. ACM SIGGRAPH, July 2002.
11. J. Lee, J. Chai, P. Reitsma, J. Hodgins, and N. Pollard. Interactive control of avatars animated with human motion data. In *Proceedings of ACM SIGGRAPH 2002*, Annual Conference Series. ACM SIGGRAPH, July 2002.
12. J. Lee and S. Y. Shin. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of ACM SIGGRAPH 99*, Annual Conference Series, pages 39–48. ACM SIGGRAPH, August 1999.
13. M. Mizuguchi, J. Buchanan, and T. Calvert. Data driven motion transitions for interactive games. In *Eurographics 2001 Short Presentations*, September 2001.
14. S. I. Park, H. J. Shin, and S. Y. Shin. On-line locomotion generation based on motion blending. In *Proceedings of ACM SIGGRAPH Symposium on Computer Animation 2002*. ACM SIGGRAPH, July 2002.
15. K. Perlin. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):5–15, March 1995.
16. Katherine Pullen and Christoph Bregler. Motion capture assisted animation: Texturing and synthesis. In *Proceedings of ACM SIGGRAPH 2002*, Annual Conference Series. ACM SIGGRAPH, July 2002.
17. L. Rabiner and B.H. Juang. *Fundamentals of Speech Recognition*. Prentice Hall, Englewood Cliffs, NJ 07632, 1993.
18. C. Rose, M. Cohen, and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Application*, 18(5):32–40, 1998.
19. C. Rose, B. Guenter, B. Bodenheimer, and M. Cohen. Efficient generation of motion transitions using spacetime constraints. In *Proceedings of ACM SIGGRAPH 1996*, Annual Conference Series, pages 147–154. ACM SIGGRAPH, August 1996.
20. C. Rose, P. Sloan, and M. Cohen. Artist-directed inverse-kinematics using radial basis function interpolation. *Proceedings of Eurographics 2001*, 20(3), 2001.
21. S. Theodore. Understanding animation blending. *Game Developer*, pages 30–35, May 2002.
22. M. Unuma, K. Anjyo, and T. Tekeuchi. Fourier principles for emotion-based human figure animation. In *Proceedings of ACM SIGGRAPH 95*, Annual Conference Series, pages 91–96. ACM SIGGRAPH, 1995.
23. D. Wiley and J. Hahn. Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics and Application*, 17(6):39–45, 1997.
24. A. Witkin and Z. Popović. Motion warping. In *Proceedings of ACM SIGGRAPH 95*, Annual Conference Series, pages 105–108. ACM SIGGRAPH, August 1995.

Appendix A

This appendix describes how to adjust a set of n-dimensional vectors $\mathbf{p}_1, \ldots, \mathbf{p}_k$ such that Equation 5 holds. Without loss of generality, we assume $\mathbf{M}_{base} = \mathbf{M}_1$.

Refer to Figure 10. We process each knot in order from $\mathbf{p}_2$ to $\mathbf{p}_k$, adjusting it if Equation 5 is violated. For $j \in [2,n]$, let $\mathbf{c}_j$ be defined such that $c_{j1} = -1$, $c_{jj} = \varepsilon$, and every other component is 0; also let $\mathbf{c}'_j$ be defined such that $c'_{j1} = \varepsilon$, $c'_{jj} = -1$, and every other component is 0. If we define $\mathbf{d}_i = \mathbf{p}_i - \mathbf{p}_{i-1}$, then Equation 5 holds if and only if $\mathbf{d}_i \cdot \mathbf{c}_j \leq 0$ and $\mathbf{d}_i \cdot \mathbf{c}'_j \leq 0$ for $2 \leq j \leq n$.

Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be sets of indices. For $2 \leq j \leq n$, we put j into $\mathcal{S}_1$ if $\mathbf{d}_i \cdot \mathbf{c}_j > 0$ and into $\mathcal{S}_2$ if $\mathbf{d}_i \cdot \mathbf{c}'_j > 0$ [†]. If $\mathcal{S}_1$ and $\mathcal{S}_2$ are empty, no adjustment needs to be made to $\mathbf{p}_i$. Otherwise we project $\mathbf{d}_i$ onto the linear subspace defined by the equations $\mathbf{x} \cdot \mathbf{c}_j = 0$ and $\mathbf{x} \cdot \mathbf{c}'_k = 0$, where $j \in \mathcal{S}_1$ and $k \in \mathcal{S}_2$. To do this, we create an orthogonal basis $\mathbf{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$ for this space. Define $\mathbf{v}'_1$ such that $v'_{11} = 1$, $v'_{1j} = \frac{1}{\varepsilon}$ if $j \in \mathcal{S}_1$, $v'_{1j} = \varepsilon$ if $j \in \mathcal{S}_2$, and $v'_{1j} = 0$ otherwise. The desired basis

[†] At least one of the conditions will hold for reasonable knot sequences. For example, in Figure 10, $\mathbf{p}_i$ would be in the bottom-left quadrant if both conditions were violated.

has $\mathbf{v}_1 = \frac{\mathbf{v}_1'}{\|\mathbf{v}_1'\|}$ and the other $\mathbf{v}_i$ set to the coordinate axis vectors corresponding to the indices that are *not* in $\mathcal{S}_1$ or $\mathcal{S}_2$. The projection of $\mathbf{d}_i$ onto this subspace yields a new vector $\mathbf{d}_i + \Delta\mathbf{d}_i$, where $\Delta\mathbf{d}_i = (I - VV^T)d_i$.

To minimize the overall disturbance, we would like to split the change $\Delta\mathbf{d}_i$ across $\mathbf{p}_i$ and $\mathbf{p}_{i-1}$ (Figure 10). We replace $\mathbf{p}_i$ with $\mathbf{p}_i + (1-\lambda)\Delta d_i$ and $\mathbf{p}_{i-1}$ with $\mathbf{p}_i - \lambda\Delta d_i$. Here λ is a positive number no larger than 0.5 and as large as possible such that replacing $\mathbf{p}_{i-1}$ with $\mathbf{p}_i - \lambda\Delta d_i$ does not lead to a violation of Equation 5.

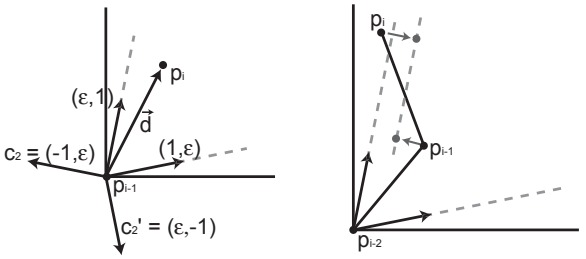

Figure 10: ***Right****: a diagram showing the relationship between* $\mathbf{p}_i$, $\mathbf{c}$, $\mathbf{c}'$*, and the slope bounds in two dimensions.* ***Left****:* $\mathbf{p}_i$ *and* $\mathbf{p}_{i-1}$ *are adjusted to satisfy Equation 5*

Appendix B

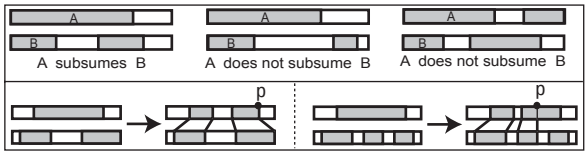

Figure 11: ***Top:*** *Examples of subsumption.* ***Bottom:*** *Adding a gap to a constraint*

This appendix discusses when and how constraints are split for constraint matches (Section 4.4). Let $C_{i,j}$ be the j^{th} constraint of the i^{th} motion, active over the interval $[C^s_{i,j}, C^e_{i,j}]$. To simplify the discussion, we assume without loss of generality that we are handling 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 $C^e_{i,1} > C^s_{j,2}$ and $C^s_{i,2} > C^e_{j,2}$ (Figure 11). Note that if $C_{i,1}$ subsumes $C_{j,1}$, then $C_{j,1}$ does not subsume $C_{i,1}$. This means that based on any single constraint, we can partition all the constraints into two sets $\mathcal{S}_1$ and $\mathcal{S}_2$ such that every element of $\mathcal{S}_1$ subsumes every element of $\mathcal{S}_2$. For example, if we put $C_{i,1}$ into $\mathcal{S}_1$, then every constraint it does *not* subsume must also be placed in $\mathcal{S}_1$. The remaining constraints are placed in $\mathcal{S}_2$. Each constraint in $\mathcal{S}_2$ is then checked to see if it is subsumed by every element in $\mathcal{S}_1$ and, if not, 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 so we transfer all the constraints to $\mathcal{S}_2$. The quality of the final partition is determined through the "total overlap" $\sum_i \sum_j (I_i \cup I_j)$, where $I_i = [C^s_{i,1}, C^e_{i,1}]$ if $C_{i,1} \in \mathcal{S}_1$ and $I_i = [C^s_{i,1}, C^e_{i,2}]$ if $C_{i,1} \in \mathcal{S}_2$. The operator $\cup$ returns the size of the intersection of the two intervals. We generate $\mathcal{S}_1$ and $\mathcal{S}_2$ for each constraint and keep the pair that has the highest total overlap.

Each constraint in $\mathcal{S}_1$ is split into two shorter constraints separated by a gap; refer to Figure 11. Say we are splitting $C_{i,1} \in \mathcal{S}_1$. For each constraint $C_{j,1} \in \mathcal{S}_2$, we obtain a vote on where the start and the end of the gap in $C_{i,1}$ should be. First $C^e_{j,2}$ is identified with a point p on C_{i_1}: if $C_{j,3}$ exists and $C^s_{j,3} < C^e_{i_1}$, then $p = C^e_{j,2}$, and otherwise $p = C^e_{i,1}$. The vote for the start of the gap is then

$$C^s_{i,1} + \frac{p - C^s_{i,1}}{C^e_{j,1} - C^s_{j,1}} (C^e_{j,1} - C^s_{j,1}) \tag{10}$$

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.