A Hybrid Interpolation Scheme for Footprint-driven Walking Synthesis*

B.J.H. van Basten† S. A. Stüvel‡ A. Egges§

Games and Virtual Worlds Group
Utrecht University, The Netherlands

ABSTRACT

In many constrained environments, precise control of foot placement during character locomotion is crucial to avoid collisions and to ensure a natural locomotion. In this paper, we present a new exact motion synthesis technique that generates planar parameterized stepping motions based on a combination of rotational and Cartesian interpolation. Existing stepping motions are blended in a linearized representation to guarantee exact control of foot placement. By concatenating these parameterized steps, we can generate highly-constrained stepping animations in real-time. Furthermore, because of a novel blend candidates selection strategy, soft constraints such as stance duration and foot orientation are also taken into account. We will show that our technique can generate a variety of different stepping animations efficiently, even though we impose many constraints on the animation.

Index Terms: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

1 INTRODUCTION

Locomotion plays an important role in games and simulations. Game characters constantly walk around the environment while trying to avoid obstacles. Therefore, generating animations of human walking has received a lot of attention during the past decades. Various techniques have been developed that generate human locomotion, each having its own advantages and disadvantages.

Many locomotion systems are driven by a *path*. Given an input path, the locomotion system generates an animation that tries to follow this path through the environment as closely as possible. However, simply providing the path that a character should follow is not always sufficient. In environments containing narrow corridors or lots of obstacles exact foot placement and different locomotion styles such as side-stepping and walking backwards can be required. In such case, a global path is an underspecification of the motion we want the character to perform. Instead, a control system based on *foot placement* may be more appropriate. By providing the desired foot positions instead of a path, it is possible to control virtual characters more precisely. However, it is not trivial to produce an animation that is realistic and that adheres to a set of foot placement constraints. Furthermore, many games and simulations require such a system to be real-time.

This animation problem is also known as the *stepping stone* problem. Given a set of query foot placements, called a *foot plan*, that contains temporal and spatial constraints, generate an animation that adheres to these constraints. In our problem setting, we consider the foot positions as *hard constraints* and foot orientation and temporal constraints, such as swing duration, as *soft constraints*. Footprint-driven animation is already supported by several 3D modeling packages, such as 3D Studio Max [15], showing that foot prints provide an intuitive means for creating an animation. Often, these commercial footprint-driven systems are not very versatile: foot plans that exhibit other motions than regular walking may result in self-intersection or unpredictable behavior [5].

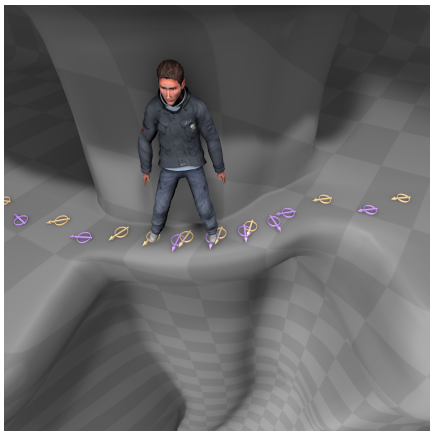


Figure 1: In some environments, exact foot placement is very important. Our technique generates animations that exactly follow a desired foot plan.

There are several advantages in footprint-driven techniques over path-driven techniques. One problem is that the pelvis (root) paths can be fairly complicated and hard to plan in cases that involve a manipulation task, such as opening a door or picking up an object. Also, foot placement over many steps is the key determinant of dynamic stability [18]. Foot placement will provide us with upper body balancing information which is useful in case we want to modify the upper body motion. Finally, decoupling step planning from animation will reduce the dimensionality of the generic motion planning problem. Instead of planning all degrees of freedom of an articulated body in a high-dimensional configuration space, a planner, such as [5], only needs to plan foot placements, after which the animation of the character following the footsteps can be computed.

1.1 Motion Parameterization

In this paper we present a new motion parameterization technique for planar foot steps based on motion interpolation. In many systems, the user desires control over specific motion properties, such as the end location of a punch or the curvature of walking. Blending techniques can be used to generate such *parameterized* motions given these high-level parameters.

Motion parameterization can be formally defined as follows: given a set of n example motions $\mathbf{M}$, each corresponding to a parameter value $\mathbf{p} \in \mathbf{P}$ where $\mathbf{P}$ is a d-dimensional parameter space, and a query parameter value $\mathbf{p}_q$, blend a set of *blend candidates*

*This research has been supported by the GATE project, funded by the Netherlands Organization for Scientific Research (NWO) and the Netherlands ICT Research and Innovation Authority (ICT Regie).

†e-mail: basten@cs.uu.nl
‡e-mail: sastuvel@cs.uu.nl
§e-mail:egges@cs.uu.nl

$\mathbf{B} \subseteq \mathbf{M}$ such that the resulting motion corresponds to $\mathbf{p}_q$. Clearly the resulting motion depends on the blending scheme. We define a *parameter mapping* $f : \mathbf{M} \rightarrow \mathbf{P}$ as the function that retrieves the parameter value $\mathbf{p}$ from a motion M. Note that such a mapping is many-to-one, as many motions can correspond to the same abstract parameter. In case of motion parameterization, we are looking for the inverse function $f^{-1} : \mathbf{P} \rightarrow \mathbf{M}$. Given a query parameter value $\mathbf{p}_q$, find (one motion from) the set of motions $\mathbf{M}_p$ that corresponds to $\mathbf{p}_q$. The parameter mapping does not need to be bijective, for many motions can correspond to the same parameter and this inverse mapping is one-to-many. In Section 2 we will elaborate on existing motion parameterization techniques.

1.2 Our contribution

Our technique allows for real-time synthesis of concatenated stepping motions that *exactly* step on the desired positions, while taking into account soft constraints such as foot orientation and swing duration. Our technique is comprised by two novel concepts.

Hybrid interpolation scheme
To achieve exact foot placement, we present a novel exact motion parameterization technique that interpolates between a number of example stepping animations using the (partially linearized) skeleton representation presented by Kulpa *et al.* [12]. We present a hybrid interpolation scheme that uses both Cartesian (positional) and rotational interpolation of joints. This results in foot placement parameters that are linear in the blend weights, yielding exact positioning results. So, in contrast to standard blending techniques, the accuracy of our technique is not dependent on the resolution of the example motions in the parameter space.

Blend candidate selection strategy for soft constraints
Where standard techniques often just take the nearest examples in parameter space as blend candidates, our exact interpolation scheme allows us to consider examples that are further away in parameter space. A blend candidate selection scheme evaluates examples in a higher, more descriptive, parameter space. Soft constraints are then also taken into account. Our technique is fully automatic: no manual annotation of animations is needed. Also, no foot trajectory modification is required because we guarantee exact foot placement. This is in contrast to standard parameterization techniques, which generally cannot guarantee exact positioning.

An overview of our system is provided in Section 3. In Section 4 we will elaborate on the construction of the high-dimensional and low-dimensional parameter spaces. In Section 5 we will present our technique to query these spaces for suitable blend candidates and blend the animations to construct realistic motion in real-time with exact foot placement. In Section 6 we will discuss experimental results and Section 7 concludes the paper. Before we detail our method, we will first discuss some related work.

2 Related Work

A lot of work has been done on generating human locomotion. In general, most techniques can be classified in three categories. *Procedural* techniques calculate locomotion from scratch based on empirical and biomechanical concepts. *Physics-based* techniques simulate locomotion by applying realistic torques on the joints. A third class is comprised by *example-based* approaches. Existing motions are reused to generate new motions. Basically, there are two main classes of example-based techniques. *Motion concatenation* techniques stitch clips of motion together [10, 13]. *Motion parameterization* techniques interpolate between two or more existing motions to generate new motions corresponding to a specific high-level parameter such as end-effector position or locomotion curvature. Motion concatenation generally yields more natural motion, while motion parameterization offers a higher level of control. Combinations of these two techniques have also been investigated [8].

Not many techniques offer exact foot placement control. One of the earliest techniques that generate animations according to an explicit foot plan is from Van de Panne [23]. A space-time optimizer determines the COM trajectory after which inverse kinematics is used to determine the leg motion. Chung and Hahn [2] present a procedural hierarchical system that generates forward walking over foot prints laid over uneven terrain based on biomechanical principles. Coros *et al.* [3] and Wu and Zordan [25] present physics-based controllers that are aware of a foot plan and try to follow it as closely as possible. Choi *et al.* [1] build up a roadmap by sampling valid stances of the biped figure. Two stances are connected if they can be connected with an adapted (portion of a) motion clip. If the required transformation is too large or the resulting motion results in collisions the connection fails. Hierarchical displacement mapping is then used to retarget the input motion to the target footprints. Unfortunately, because a roadmap has to be built in a preprocessing step, this technique only works in fixed environments. Van Basten *et al.* [22] present a technique that concatenates individual foot steps. To reduce the foot placement error, a small pivoting motion over the stance foot is allowed. a closed-form inverse kinematics technique is used to resolve the remaining error, possibly introducing balancing artifacts. Safonova and Hodgins [21] use a large (semi-parametric) motion graph that is created by discretely blending all pairs of motions from a standard motion graph. This graph is than pruned and searched using a global algorithm. This technique provides greater accuracy than standard motion graphs because it allows for the interpolation of two paths (or: motions) in a motion graph. Like our approach, they are able to generate motion over a set of constraints (like footplants). However, this technique is computationally expensive (6-7 minutes for an animation of 15 seconds according to the authors) and is ideally applied in a fixed environment, making it better suited for offline purposes.

Because we strive for control while retaining naturalness, we employ an *example-based motion parameterization* technique. A common approach is to linearly determine the blend weights in the abstract parameter space $\mathbf{P}$ such as Wiley and Hahn [24]. They create a densely sampled parameter space to generate parameterized motions. In order to retrieve the blend candidates $\mathbf{B}$ efficiently, they resample motions into a grid using a brute force technique. Only example motions bounding the grid cell are taken into account. Guo *et al.* [7] present a similar technique, albeit on keyframed motions. In their application, they do not create a grid, but let 4 example motions span an entire parameter space. Note that, just like Wiley and Hahn [24], $O(2^d)$ reference motions are required where d represents the dimensionality of the parameter space. There are also techniques that allows one to take more (or all) examples into account. Rose *et al.* [19] present the *verb and adverb* system. Multiscattered data interpolation is done by determining the weights using combinations of radial basis functions and hyperplanes. This interpolation scheme requires $O(d)$ examples to approximate the baseline approximation (and $O(n^3)$ to compute the result, where n is the number of example motions). This interpolation scheme does map parameters corresponding to example motions on the example motions themselves. This technique has later been improved by Rose *et al.* [20] using cardinal basis functions. Mukai and Kuriyama [16] employ geostatistics-based interpolation techniques to further improve the accuracy of the resulting motion. All the previous methods assume that the example motions are semantically identical (for example, all motions are single reach motions) and are annotated with key events (such as foot downs). Kovar *et al.* [9] propose a k-nearest neighbor interpolation scheme. Given a set of motion capture data, they automatically retrieve sets of "similar",

time-aligned motions that form a parameter space.

Unfortunately, because of the non-linearity of the orientation domain and the constraints defined by an articulated skeleton consisting of joints and bones, motion blending will not yield a linear parameterization of the space. In other words, the resulting motion will **not** exactly correspond to the desired parameter $\mathbf{p}_q$. This holds for all motion parameterization techniques described above. The error of these techniques depends on the resolution of the examples in the parameter space. This error is acceptable in some parameter domains, such as "happiness", but not with spatial parameters such as end-effector positions. Several techniques solve this problem by explicitly modifying the resulting motion. For example, Park *et al.* [17] also use radial basis functions like Rose *et al.* and force the trajectory of the root to follow the desired curvature and apply retargeting to solve foot skating. Grassia [6] transforms the motion using inverse kinematics to meet the exact constraints. Rose *et al.* [20] propose to iteratively adjust the desired position in the direction of the error vector. Still, exact parameterization cannot be guaranteed and the precision depends on the resolution of the created parameter space. Other techniques perform *resampling* to reduce the error. For example, Kovar *et al.* [9] randomly generate (nearly convex) weights and create pseudo examples by blending nearby examples with these weights. This technique is also used in the parametric motion graphs from Heck and Gleicher [8]. Because we also concatenate parameterized steps, our technique resembles a parametric motion graph. However, our technique guarantees that the resulting foot positions will be exactly at the desired position, in contrast to the standard parametric motion graph and other motion parameterization techniques. Therefore there is no need for end-effector trajectory modification.

3 Overview

Our system can be divided into an offline and online phase. During the offline phase, we create the data structures that allow for fast motion synthesis. During the online phase, these structures are queried and the resulting motion is rendered to the character. In this section we provide an overview of our technique.
The offline phase (Section 4) consists of the following steps:

1. We automatically segment a corpus of motion data consisting of locomotion animations into clips of individual steps.
2. We represent these steps in a 10-dimensional parameter space S_{high} (see Figure 3).
3. We also represent the same steps in a 3-dimensional parameter space S_{low} (see Figure 4) using a Delaunay tetrahedralization with cross-pointers to corresponding steps in S_{high}.
4. We augment every tetrahedron T in the 3-dimensional tetrahedralization with a list of possible blend candidates $\mathbf{B}(T)$.

During the online phase (Section 5) the user or footstep planner supplies a query foot plan. Then, for each query step in this foot plan:

1. The query step is transformed to the lower-dimensional parameter representation.
2. We utilize S_{low} to rapidly find the set of possible blend candidates $\mathbf{B}$.
3. We select the final blend candidates by evaluating the soft constraints of the blend candidates in the higher-dimensional, more descriptive, parameter space S_{high}.
4. The final blend candidates are blended in the lower-dimensional space S_{low}, using a novel interpolation scheme that yields exact foot placement.
5. The generated step is aligned and fitted to the previous step.

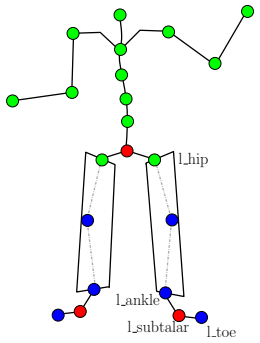


Figure 2: We transform the lower body to the morphology-independent representation of Kulpa *et al.*

4 Creating the Step Spaces

In this section, we will elaborate on the offline construction of the high-dimensional and low-dimensional parameter spaces and how these are automatically filled given a corpus of motion capture data. From now on, we will call these parameter spaces *step spaces*. First, we extract individual steps from the motion capture data. In order to extract individual steps, we need to determine the moments the feet are planted. The moments are detected by evaluating the height and velocity of the feet [5]. This footstep detector is sufficient for walking. We consider one step to be the displacement of one foot and we will elaborate on the exact step definition in the next section.

We represent the lower-body of the stepping motions by the morphology independent representation of Kulpa *et al.* [12] (see Figure 2). This representation is independent of the anthropometric properties of the character. The lower body is partially linearized: instead of storing a leg limb as an upper and lower leg segment, the half plane containing the triangle Δ(hip, knee, ankle) is stored and hence, the knee is not explicitly present. We define the *swivel angle* as the rotation of this half plane around the vector **l** from hip to ankle. The upper body is represented by a standard skeleton representation.

4.1 High-Dimensional Step Space

When the individual steps are segmented, we can retrieve the step parameters that comprise a 10-dimensional step space S_{high} [22]. Figure 3 shows an overview of the parameters. In this example, a left step, we define the left foot as the *swing foot* and the right foot as the *supporting foot*. In our (planar) representation we consider 6 spatial parameters and 4 temporal parameters.

The spatial parameters are expressed in a coordinate frame aligned with the supporting foot. The origin of this coordinate frame coincides with the *subtalar* of the foot and the y-axis is parallel to the vector from the subtalar to the toe. Every foot step consists of 2 foot placements f_1 and f_2. We represent both foot placements by a local position and orientation (x_1, y_1, θ_1) and (x_2, y_2, θ_2). We denote the world position of the foot placements of the swing foot as $W(f_1)$ and $W(f_2)$ and the world position of the supporting foot as W_{sup}. The orientation θ of a foot placement is the angle between the subtalar-toe vector of that placement and the y-axis. Because a foot placement spans multiple frames, we take the average of these parameters over the entire stance.

The temporal parameters are the *stance durations* t_{stance_1} and t_{stance_2} of both placements, the *swing duration* t_{swing} and the stance duration of the supporting foot t_{sup}. Note that the 4 double-headed arrows corresponding to the temporal parameters depicted in Figure 3 are only for clarification, but are not situated in the coordinate system. Now, a step i can be represented in this high-dimensional space as: $F_i^{high} = (x_1, y_1, x_2, y_2, \theta_1, \theta_2, t_{stance_1}, t_{stance_2}, t_{swing}, t_{sup})$.

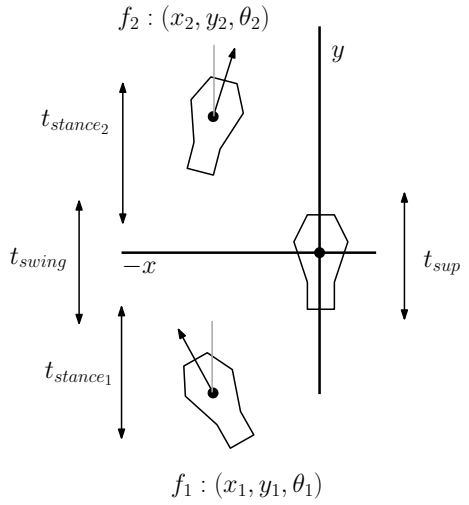


Figure 3: We represent a foot step in S_{high} using 10 parameters.

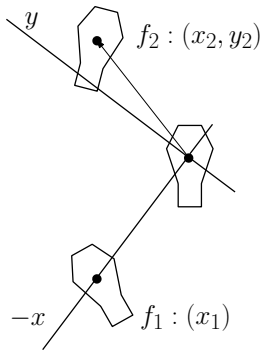


Figure 4: We express the spatial properties of a step in S_{low} using 3 parameters.

Note that not all parameters are independent. For example, there is a dependency between the swing duration and the stance duration of the supporting foot, although they are generally not the same. Next to that, this space is, in a sense, over-defined: In case of a left step s_1 followed by a right step s_2, the foot placement f_2 of s_1 corresponds to the supporting foot placement of s_2. Our assumption is that creating two consecutive steps s_1 and s_2 where the overlapping foot placements resemble each other allows for a good transition.

4.2 Low-Dimensional Step Space

We blend animations in a lower-dimensional space (see Figure 4). In this low-dimensional step space S_{low} we only consider the position of the feet (subtalar). In our problem setting, the positions of the subtalars are the *hard constraints*. Again, the positions are expressed in the coordinate frame centered on the subtalar of the supporting foot. In case of a left step, the (negative) x-axis is aligned with the vector from the subtalar of the supporting foot to the subtalar of the first foot placement f_1. In case of a right step, we align the positive x-axis instead of the negative. The y-axis remains orthogonal to the x-axis such that the coordinate system is right-handed. Now, as can be seen in Figure 4, we can represent the spatial property of one step using only 3 parameters: $F_i^{low} = (x_1, x_2, y_2)$

We transform the steps from S_{high} to this representation and keep track which step in S_{low} corresponds to S_{high} and vice versa. The 3D representations can then be stored in a Delaunay tetrahedralization,

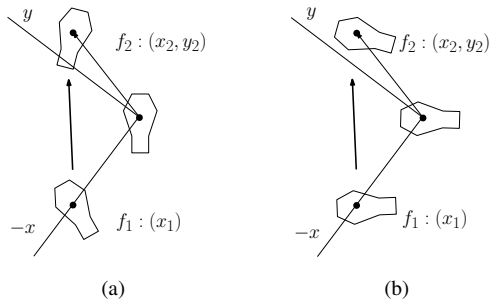


Figure 5: A normal step and a cross-step can be identical in S_{low}.

for which we refer the reader to De Berg *et al.* [4]. We create 2 tetrahedralizations: one for left steps and one for right steps. This avoids blending left and right steps when the desired parameters are close to the origin.

4.3 Augmenting the Low-Dimensional Step Space with blend candidates

Because of our hybrid blend scheme, on which we will elaborate in Section 5, we can guarantee that the positional (hard) constraints are always exactly met, as long as the query motion is inside the convex hull of the corresponding tetrahedralization in S_{low}. This allows us to consider other blend candidates than the vertices (or: example motions) of the tetrahedron $T_{contains}$ that contains the query motion, and take soft constraints into account.

Simply taking the vertices of $T_{contains}$ as blend candidates means that we are only considering foot positions. S_{low} is underdefined, as there is an ambiguous mapping from steps in S_{low} to the actual motion. A cross step such as the one depicted in Figure 5(b) has the same values in S_{low} as a normal step depicted in Figure 5(a). Therefore we would like to consider tetrahedra whose vertices (or: example motions) correspond more to the soft constraints.

The foot positions of these alternative blend candidates might not be so close to the desired values of the query foot step F_q^{low}, but the other parameters (such as orientation and stance durations) might correspond more to the soft constraints in F_q^{high} (the query foot step represented in S_{high}). So, we consider alternative blend candidates by evaluating their (weighted) distance to F_q^{high} in S_{high}, on which we will elaborate in Section 5.2.2. The only problem that arises is that the new set of blend candidates need to span a tetrahedron that actually contains F_q^{low}, otherwise one needs to extrapolate. This problem can be reformulated in finding the set $\mathbf{B}_{total}(F_q^{low})$ of *all* tetrahedra of a point set P consisting of n points that contain the query step F_q^{low}. A brute-force approach to determine all $\binom{n}{4}$ tetrahedra in a point set would take $O(n^4)$. But even with a more advanced algorithm this will not be suitable for real-time purposes. Therefore, during preprocessing, we approximate the possible set of blend candidates *per tetrahedron* in S_{low}. A 2D example is shown in Figure 6. For each tetrahedron T, we determine its centroid $\bar{c}$. From this centroid, we determine the k-nearest neighbors within a spherical range r. From these nearest vertices, we determine the set of tetrahedra $\mathbf{B}(T)$ that fully contain T. This set is then stored with the tetrahedron. We also explicitly add T itself, as this, in theory, does not need to be part of $\mathbf{B}(T)$. We define a tetrahedron $T_{cand} \in \mathbf{B}(T)$ as *blend tetrahedron* and its motions $\mathbf{s}(T_{cand})$ as blend candidates.

It is easily shown that the resulting set $\mathbf{B}(T)$ will be a subset of the total set of blend candidates for a query point inside T. Nev-

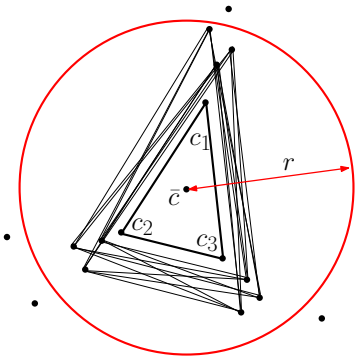


Figure 6: Possible blend candidates are determined in the vicinity of $\bar{c}$.

ertheless, a large percentage of possible blend candidates is captured using this heuristic. For efficiency, only tetrahedra that are fully within a distance of r to the center $\bar{c}$ of T are taken into account. This is not a problem, as at least one of the blend candidate vertices will be distinct from the query point F_q^{low} and will probably not be useful. A Delaunay tetrahedralization can contain up to $O(n^{\lceil 3/2 \rceil})$ tetrahedra [4], each storing a list of possible blend tetrahedra. Clearly, the size of $\mathbf{B}(T)$ can be reduced by decreasing k.

Our space of example motions contains several styles of motions and it can be possible that several styles of motions are blended. In order to avoid blending of forward and backward motions, we neglect tetrahedra whose vertices correspond to both forward and backward stepping. We can detect such cases automatically by evaluating the angle between trunk direction and global root direction of the blend candidates.

Furthermore, in case our motion database contains walking animations with diverse upper body motions we need to evaluate the resemblance between these upper body motions. In case the upper body motions are totally different (for example, a hitting motion versus a clapping motion) we would also like to ignore this tetrahedron. The upper body error d_u for a tetrahedron T containing motions $\mathbf{s}(T) = s_a, s_b, s_c, s_d$ is determined by a distance metric from Lee *et al.* [13]. We determine the total upper-body posture distance over all frames between every pair of motions. The upper body error for a tetrahedron T is defined as follows:

$$d_u(T) = \sum_{s_a,s_b \in \mathbf{s}(T) \times \mathbf{s}(T)} \left(\sum_{m=1}^{M} d_u(s_a, s_b, m) \right) \quad (1)$$

$$d_u(s_a, s_b, m) = \sum_{j \in \mathbf{J}} w_k \left\| \log(q_{s_b,m,j}^{-1} \cdot q_{s_a,m,j}) \right\|^2 \quad (2)$$

where $\mathbb{J}$ is the set of upper body joints, w_k is a (in our case uniform) weight set and $q_{s,m,k}$ is the j'th upper body joint orientation of frame m of motion s.

5 Step Synthesis

In this section we will present our technique for online step synthesis. The main outline is that we rapidly find a set of blend candidates in S_{low}, select the 4 motions to blend by evaluating them in the more descriptive parameter space S_{high} and determine the blend weights in S_{low}. The stepping motions are then blended using the blending scheme explained in Sections 5.2.1 and 5.2.2. Each generated stepping motion is aligned to the previously generated motion. For the remainder of this section, we assume the existence of a query foot plan F_q consisting of steps represented in the high-dimensional representation depicted in Figure 3 and explained in Section 4.1. Such a foot plan can come from a foot step planner [5], interactively set by an animator or extracted from motion capture. In Section 6 we will show examples of foot plans based on all three.

5.1 Selecting the blend candidates

During online motion synthesis, we generate the steps sequentially. First, we express each query step in F_q in its lower-dimensional representation F_q^{low}. We determine the *initial blend tetrahedron* $T_{contains}$ by doing a point location query in the corresponding Delaunay tetrahedralization D.

From tetrahedron $T_{contains}$, we retrieve the preprocessed list of blend tetrahedra $\mathbf{B}$. For each tetrahedron $T_{cand} \in \mathbf{B}(T_{contains})$, we evaluate the sum of the distances from its blend candidates ($s_1^{T_{cand}}$ to $s_4^{T_{cand}}$) to F_q in the higher-dimensional, more informed, space S_{high}. From $\mathbf{B}$, we select T_{blend} as follows:

$$T_{blend} = \underset{T_{cand} \in \mathbf{B}(T)}{\operatorname{argmin}} \left(\sum_{i=1}^{4} d_w(s_i^{T_{cand}}, F_q^{high}) + w_u d_u(T_{cand}) \right) \quad (3)$$

where d_w is a 10D weighted Euclidean distance function in S_{high}. In case the user gives a high weight to the foot position parameters, the blend tetrahedron T_{blend} might be the containing tetrahedron T itself. However, using these techniques, the soft constraints are taken into account. In case the database contains diverse upper-body motions, the upper body error can be taken into account via w_u. After the blend tetrahedron, and hence, the 4 example motions, are selected, we blend them as described in the next section.

5.2 Blending the steps

After we have determined the 4 blend candidates (which we will denote as $\mathbf{B} = \{b_1, \ldots, b_4\}$ from now on), we can derive the (convex) blend weights $\mathbf{w}_i = \{w_1, \ldots, w_4\}$ in S_{low} such that $\sum_{i=1}^{4} w_i b_i = F_q^{low}$, $0 \leq w_i \leq 1$ and $\sum_{i=1}^{4} w_i = 1$.

Because all steps have a similar structure (double stance, swing, double stance) it is trivial to construct a piecewise linear mapping between the blend candidates and the resulting motion. The resulting durations of the two double stance and swing periods are determined by linearly interpolating the durations of the blend candidates.

To generate the final stepping animation, we employ a hybrid blending scheme. For the upper body we use a (standard) interpolation scheme that interpolates the orientations of the joints. For the lower body we use a novel interpolation scheme that combines Cartesian (positional) and rotational interpolation. Figure 2 illustrates the techniques used for the individual joints. Red joints are determined by Cartesian interpolation, green joints by rotational interpolation and blue joints are derived from by a combination of both. Note that this hybrid blend scheme is applied on all frames of the blend candidates, resulting in a new stepping motion.

5.2.1 Blending Upper Body Motion

To determine the new upper body joints (including hip and root), we blend all the *orientations*. The *logarithmic map* transforms all quaternions $\mathbf{q}$ to a vector representation $\mathbf{v}$. We can then determine the weighted average of each orientation of joint j: $\mathbf{v_j^{result}} = \sum^{b_i \in \mathbf{B}} w_i \cdot \mathbf{v}_{j,i}$ and transform the logarithmic map back to the quaternion representation. For a more elaborate explanation on blending in the logarithmic map, we refer the reader to Park *et al.* [17]. The position of the root is linearly interpolated.

5.2.2 Blending Lower Body Motion

We use *Cartesian* interpolation for the subtalar joints: we blend the *positions* of the subtalars instead of joint orientations. Blending is done in the coordinate system of the supporting foot, so that footskating on the supporting foot is not possible. This will result

in subtalar trajectories whose start and end positions exactly correspond to F_q^{low}.

For the remaining joints of the feet (ankle and toe), we cannot perform a positional interpolation because this will introduce bone stretching. Therefore we estimate the positions of the ankle and toe based on rotational interpolation. We first interpolate the orientations in the feet, this will result in foot positions that do not coincide with F_q^{low}. Let $\mathbf{p}_{ankle}^{rot}$, $\mathbf{p}_{toe}^{rot}$, $\mathbf{p}_{subtalar}^{rot}$ be the resulting positions of the ankle, toe and subtalar after (standard) rotational interpolation of the leg joints as described in the previous section. We then determine the offset vector $\mathbf{v_{error}} = \mathbf{p}_{subtalar} - \mathbf{p}_{subtalar}^{rot}$ where $\mathbf{p}_{subtalar}$ is the position of the subtalar derived by Cartesian interpolation. We then determine the positions of the ankle and toe by translating their positions with $\mathbf{v_{error}}$. This technique attempts to keep the heading of the feet similar to the result of rotational interpolation. Note that one can only linearize a posture to a certain degree: interpolating the positions of the ankle, subtalar and toe independently might introduce bone stretching/shrinking in the foot.

After we have derived the positions of the ankles, subtalars and toes we can derive the position of the knees. Using the law of the cosines, one can derive the knee angle. We determine the *swivel* angle (see Section 4) of the new stepping motion by linear interpolation of the swivel angles of the blend candidates. Using the knee and swivel angles, it is trivial to derive the knee position. Now, all joints positions and orientations are determined and the result is a stepping motion whose start and end position of the subtalars exactly coincide with the desired positions.

5.3 Step Concatenation

After the step motion is generated, we align the resulting step motion s_i such that $W(f_1)$ (world position of the first foot placement of the swing foot) and W_{sup} (world position of the supporting foot) of s_i coincide with respectively W_{sup} and $W(f_2)$ of the previously generated step s_{i-1}. In case the previous step is a similar sided step, we align $W(f_1)$ and W_{sup} of s_i with respectively $W(f_2)$ and W_{sup} of the previously generated step s_{i-1}. Because the soft constraints might not have fully been satisfied, it could be possible that the foot orientation between consecutive steps does not fully match. Therefore we determine the new orientation of the supporting foot of s_i as the average of the final orientation of the swing foot of s_{i-1} and the supporting foot of s_i. The desired orientation of the foot is then eased-in during the swing of s_{i-1}.

We also need to ensure that the ankle positions are reachable from the current root position *without* fully stretching the legs, to avoid *knee popping* [11], which occurs in particular when the leg approaches full extension. So, we set the allowed length of the hip-ankle vector to $\alpha(|hip \rightarrow knee| + |knee \rightarrow ankle|)$. We have found that a damping factor of $\alpha = 0.98$ yields good results. Finally, because we generate steps independently the positional interpolation might introduce a discontinuity in the root trajectory between steps. Therefore we apply filtering on the root trajectory. We fit a Bézier curve to smoothen the root trajectory in a window centered at double stances. After we have concatenated the steps, we can transform the motion back to the standard skeleton representation and render it on the character.

We concatenate the generated steps during the periods of double support. It is tempting to think that one can actually transition from the current generated step to the next generated step during the swing, while rooting the foot to the floor, as is done in [5]. Ideally, the previous step s_1 is blended out and the previous step of the next step s_2 is blended in. Unfortunately, there does not need to be a previous step of s_2, as s_2 is generated by parameterization. One could also try to generate the previous step of s_2 by parameterization, but various left and right steps (or no steps at all) might be blended together. Due to noise in the data the periods of double support can be as short as a single keyframe. Such a small blending window can result in jerky transition on the upper body. In order to solve this, we filter the orientations of the upper body using the efficient filtering technique of Lee *et al.* [14].

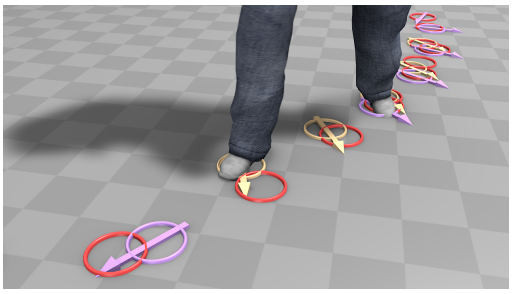


Figure 7: Standard parameterization techniques yield a high positional error.

6 Results

In this section we will present some results of our technique. Our database S_{high} consisted of 400 steps sampled at 30 Hz (both the left and right-sided S_{low} contained 200 steps). This includes the mirrored versions of the recorded steps. All recorded motions are walking motions and the database (5 minutes of motion) consists of 54% forward, 28% backwards and 18% side and cross stepping motions. For determining alternative blend candidates using d_w, we use a weight of 1.0 for the spatial parameters (x_1, x_2, y_1, y_2), 0.25 for the angular parameters (θ_1, θ_2) and 0.02 for the temporal parameters [5]. For upper-body filtering, we use a low-pass filter with mask $[\frac{1}{16}, \frac{4}{16}, \frac{6}{16}, \frac{4}{16}, \frac{1}{16}]$ as suggested by [14]. Furthermore, w_u is set to 0, as our database contains only pure walking animations and k (the number of nearest neighbors) is set to 50. Preprocessing $\mathbf{B}(T)$ took around 1 to 2 hours. All experiments were executed on an Intel Pentium 3 GHz (dual-core) with 3 GB RAM. We tested on several foot plans. In the first example, in Figure 8, we synthesized a stepping animation over a foot plan determined by the planner from Egges and Van Basten [5]. Based on the clearance along a path, the planner determines when to start side stepping. Our character correctly walks and side steps through the environment containing several narrow corridors. This foot plan consists of 24 steps and results in 14 seconds of animation. In the second example, Figure 9, we extracted a foot plan of 12 steps from real recorded motion. In 9(a) we see the resulting animation. Our system allows for interactive editing of foot plans (see Figure 9(b) and 9(c)). The duration of the resulting motion was 8.2 seconds. The third example (Figure 10) is an animation of 9.0 seconds over a foot plan set by the user. This foot plan consisted of 12 steps comprising a transition from forward-to-backward-to-forward walking. More examples can be seen in the accompanying video.

The application of standard motion parameterization techniques (linearly determine blend weights and blend orientations accordingly) to the 3-dimensional S_{low} will result in large discrepancies, as can be seen in Figure 7. In this figure, the foot placements resulting from the standard parameterization technique are shown as red rings and clearly show the spatial discrepancy introduced by this technique. Our method, on the contrary, provides exact foot placement. On average, online generation of a single step (excluding rendering, including concatenation to previous step) takes 0.028 seconds (σ = 0.02 seconds, N = 72 steps). The average duration of a generated step was 0.59 seconds (σ = 0.05). This makes this technique suitable for real-time applications. With the parameter set listed above, in 15% of blend candidate selection, an alternative tetrahedron was choosen. We performed a leave-one-out cross-validation to validate the effect of selecting alternative blend candidates on the soft constraints. We removed one step q from the

Table 1: Leave-one-out cross-validation (error in radians)

Method	Containing tetra	Alternative blend cand.
av. error θ_1	0.3043	0.2924
av. error θ_2	0.3200	0.2947

database and generated a step g with the same parameter values using only the remaining steps in the database. We compared the foot orientations from the original step q to the generated step g. The average orientation errors are depicted in Table 1. Here we see that the orientations of the resulting step g are closer to the original when considering alternative blend candidates (instead of using blend candidates corresponding to the containing tetrahedron). The total average orientation error ($\theta_1 + \theta_2$) decreases 6% when using alternative blend candidates, which is a significant improvement (paired T-test, $p < 0.02$).

7 Conclusion and Future Work

We have presented an efficient motion synthesis technique that generates parameterized stepping motions with exact foot placement using a hybrid interpolation scheme. The accuracy of our technique is not dependent on the resolution of the example motions in the parameter space. Therefore we can evaluate other examples than those in the near vicinity of the query in the parameter space and take soft constraints, such as timing and foot orientation, into account. This is not possible in standard techniques where blend candidates need to be close to the desired parameter value in parameter space. Our method does not require manual annotation of animations and no extensive post-processing and end-effector trajectory modification is needed. This allows for fully automatic generation of highly-constrained locomotion in real-time.

At this point, our approach can be used for walking motions. However, we see possibilities for running too, although our local approach might not be sufficient and one must consider a sequence of steps. One cannot drastically change speed, curvature or foot placements in a single step of running. Furthermore, it is interesting to extend this technique for non-planar locomotion (which would require a collision check of the foot trajectory with the walking surface) or upper body manipulation. Currently, when a step is outside the convex hull of the parameter space, the user is just informed that the step cannot be synthesized. Optionally, the nearest example is selected from the database and inverse kinematics is applied. A more advanced constraint editor that provides feedback when a desired step is outside the convex hull of the parameter space would be beneficial. Furthermore, instead of evaluating each blend candidate individually with respect to the query step, it might be interesting to evaluate the resulting motion of blending the candidates instead. This might give an even better metric for blend candidate selection. Also, we have observed that *limited* extrapolation can result in natural motions. Further research is needed to determine to what extent extrapolation is possible. It would be interesting to compare the results of our technique to other footprint-driven techniques, such as [5] or [1]. A comparison in terms of both perceived motion quality (user study) as well as computation time will be useful contributions.

Although the linearization of the skeletal structure is limited to specific limbs and can lead to bone stretching/shrinking when not properly applied, it still is interesting to investigate other, even more simplified, representations to reduce and simplify the posture space. We are currently looking into using a linearized representation for other applications, such as reaching. We believe that geometric posture representations combined with a hybrid Cartesian interpolation scheme can be a very powerful tool for efficient synthesis of various kinds of motion.

References

[1] M. G. Choi, J. Lee, and S. Y. Shin. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Trans. Graph.*, 22(2):182–203, 2003.

[2] S. Chung and J. Hahn. Animation of human walking in virtual environments. In *Proc. of Computer Animation*, volume 99, pages 4–15. Society Press, 1999.

[3] S. Coros, P. Beaudoin, K. Yin, and M. van de Panne. Synthesis of constrained walking skills. In *ACM SIGGRAPH Asia 2008 papers*, page 113. ACM, 2008.

[4] M. De Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. Computational geometry: Algorithms and applications. 2000.

[5] A. Egges and B. van Basten. One step at a time: animating virtual characters based on foot placement. *The Vis. Comp.*, 26(6):497–503, 2010.

[6] F. S. Grassia. *Believable automatically synthesized motion by knowledge-enhanced motion transformation*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2000.

[7] S. Guo, J. Roberge, and T. Grace. Controlling the movement of an articulated figure using parametric frame-space interpolation. In *Proc. of SPIE*, volume 2644, page 766, 1996.

[8] R. Heck and M. Gleicher. Parametric motion graphs. In *Proc. of the symp. on Inter. 3D graph. and games*, pages 129–136, New York, NY, USA, 2007. ACM Press.

[9] L. Kovar and M. Gleicher. Automated extraction and parameterization of motions in large data sets. *ACM Trans. Graph.*, 23(3):559–568, 2004.

[10] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *SIGGRAPH*, pages 473–482, New York, NY, USA, 2002. ACM Press.

[11] L. Kovar, J. Schreiner, and M. Gleicher. Footskate cleanup for motion capture editing. In *Proc. of the sym. on Comp. animation*, pages 97–104, New York, NY, USA, 2002. ACM.

[12] R. Kulpa, F. Multon, and B. Arnaldi. Morphology-independent representation of motions for interactive human-like animation. *Comp. Graphics Forum*, 24(3):343–352, 2005.

[13] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard. Interactive control of avatars animated with human motion data. In *SIGGRAPH*, pages 491–500, New York, NY, USA, 2002. ACM Press.

[14] J. Lee and S. Shin. General construction of time-domain filters for orientation data. *IEEE Trans. on Vis. and Comp. Graphics*, pages 119–128, 2002.

[15] 3D Studio Max. accessed 20 august 2010. http://usa.autodesk.com/.

[16] T. Mukai and S. Kuriyama. Geostatistical motion interpolation. *ACM Trans. Graph.*, 24(3):1062–1070, 2005.

[17] S. I. Park, H. J. Shin, and S. Y. Shin. On-line locomotion generation based on motion blending. In *Proc. of the sym. on Comp. animation*, pages 105–111, New York, NY, USA, 2002. ACM.

[18] A. Patla. Strategies for dynamic stability during adaptive human locomotion. *IEEE Eng. in Medicine and Biology Magazine*, 22(2):48–52, 2003.

[19] C. F. Rose, M. F. Cohen, and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Comput. Graph. Appl.*, 18(5):32–40, 1998.

[20] C. F. Rose, P.-P. J. Sloan, and M. F. Cohen. Artist-directed inverse-kinematics using radial basis function interpolation. *Comp. Graphics Forum*, 20(3):239–250(12), September 2001.

[21] A. Safonova and J. K. Hodgins. Construction and optimal search of interpolated motion graphs. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 106, New York, NY, USA, 2007. ACM.

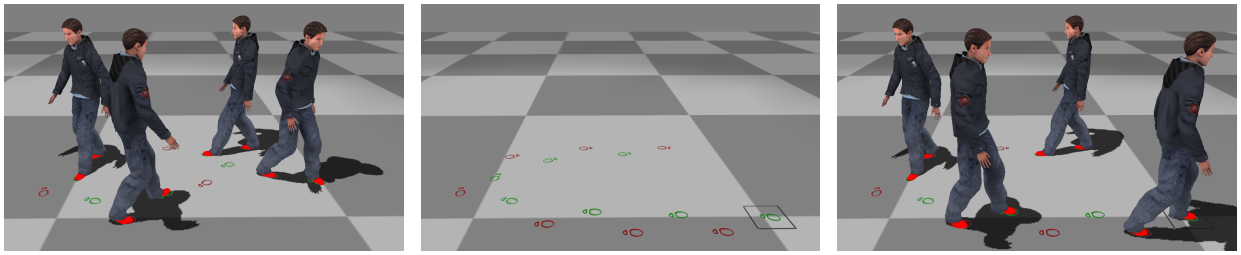
[22] B. J. H. van Basten, P. Peeters, and A. Egges. The step space: example-based footprint-driven motion synthesis. *Comp. Anim. and Virt. Worlds*, 21(3):433–441, 2010.

[23] M. van de Panne. From footprints to animation. *Comput. Graph. Forum*, 16(4):211–224, 1997.

[24] D. J. Wiley and J. K. Hahn. Interpolation synthesis of articulated figure motion. *IEEE Comput. Graph. Appl.*, 17(6):39–45, 1997.

[25] C. Wu and V. Zordan. Goal-directed stepping with momentum control. In *Proc. of the sym. on Comp. animation*, New York, NY, USA, 2010.

Figure 8: The character walks over a planned footprint avoiding the red obstacles.



(a) (b) (c)

Figure 9: Our technique allows for interactive editing.

Figure 10: The character changes from forward to backwards to forwards walking.