Footstep Parameterized Motion Blending using Barycentric Coordinates

Alejandro Beacco[1], Nuria Pelechano[1], Mubbasir Kapadia[2,3] & Norman I. Badler[3]

[1]Universitat Politècnica de Catalunya
[2]Rutgers University
[3]University of Pennsylvania

Abstract

This paper presents a real-time animation system for fully-embodied virtual humans that satisfies accurate foot placement constraints for different human walking and running styles. Our method offers a fine balance between motion fidelity and character control, and can efficiently animate over sixty agents in real time (25 FPS) and over a hundred characters at 13 FPS. Given a point cloud of reachable support foot configurations extracted from the set of available animation clips, we compute the Delaunay triangulation. At runtime, the triangulation is queried to obtain the simplex containing the next footstep, which is used to compute the barycentric blending weights of the animation clips. Our method synthesizes animations to accurately follow footsteps, and a simple IK solver adjusts small offsets, foot orientation, and handles uneven terrain. To incorporate root velocity fidelity, the method is further extended to include the parametric space of root movement and combine it with footstep based interpolation. The presented method is evaluated on a variety of test cases and error measurements are calculated to offer a quantitative analysis of the results achieved.

Keywords:
Character animation, Crowd simulation, Footsteps controller

1. Introduction

Crowd simulation research has matured in recent years with important applications in training, building design, psychological studies, and video-games. All these applications benefit from having fully-embodied virtual human characters animated in real-time while accurately satisfying control objectives without any noticeable artifacts.

Algorithms that generate center of mass (COM) trajectories [1, 2, 3, 4] lead to ambiguities when trying to superimpose a fully articulated virtual human to follow them, thus producing foot-sliding artifacts when no suitable animation is found, or when the root orientation and the displacement vector of the animation do not match. Different animations can be blended by tweaking some of the upper body joints [5] to minimize artifacts, at the expense of constant updates to account for the decoupling between the crowd simulation and the animation system.

Footstep-based control systems [6, 7] output a list of space-time foot-plants to define a fine-grained trajectory with fewer ambiguities that can solve more complex scenarios (e.g., complex manipulation tasks requiring careful control of the lower body, or collaborative tasks, such as careful sidestepping to make way for another agent in a narrow corridor). To realistically represent such simulations, we need a method to synthesize animations that accurately follow the output trajectory, i.e., accurate placement of feet with space-time constraints. This problem is traditionally known as the *stepping stone* problem. Moreover, the output trajectory can be modified by external perturbations such as uneven terrain.

We present an online animation synthesis technique for fully embodied virtual humans that satisfies foot placement constraints for a large variety of locomotion speeds and styles (see Fig. 1). Given a database of motion clips, we precompute multiple parametric spaces based on the motion of the root and the feet. A root parametric space is used to compute a weight for each available animation based on root velocity. Two foot parametric spaces are based on a Delaunay triangulation of the graph of possible foot landing positions. For each foot parametric space, blending weights are calculated as the barycentric coordinates of the next footstep position for the triangle in the graph that contains it. These weights are used for synthesizing animations that accurately follow the footstep trajectory while respecting the singularities of the different walking styles captured.

Blending weights calculated as barycentric coordinates are used to reach the desired foot landing by interpolating between several proximal animations, and IK is used to adjust the final position of the support foot to correct for minor offsets, foot step orientation and the angle of the underlying floor.

Since foot parametric space only considers final landing positions of the feet without taking into account root velocity, this may lead to the selection of animations that satisfy position constraints but introduce discontinuities in root velocity. To incorporate root velocity fidelity we present a method that can integrate both foot positioning and root velocity fidelity. Our method also allows the system to recover nicely when the input

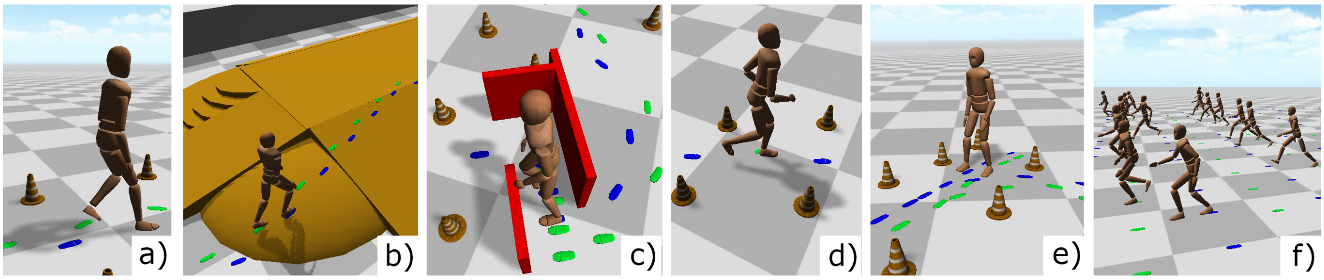

Figure 1: An autonomous virtual human navigating a challenging obstacle course (a), walking over a slope (b), exercising careful foot placement constraints including side-stepping (c), speed variations (d), and stepping back (e). The system can handle multiple agents in real time (f).

foot trajectory contains steps that are not possible to perform with the given set on animations (for example, due to extreme distance between steps).

The presented method is evaluated on a variety of test cases and error measurements are calculated to offer a quantitative analysis of the results achieved. Our framework can efficiently animate over sixty agents in real time (25 FPS) and over a hundred characters at 13 FPS, without compromising motion fidelity or character control, and can be easily integrated into existing crowd simulation packages. We also provide the user with control over the trade-off between footstep accuracy and root velocity.

2. Related Work

Locomotion synthesis can be tackled from different points of view depending on how the character is being controlled. If a user controls the character with a 3rd person controller, it is common to work on a root velocity basis, because the user wants to move the character around in an agile way. In such cases, like video-games, real-time response is critical and artifacts such as foot skating can be ignored. Optimization based approaches [8] are able to synthesize animations that conform to velocity and orientation constraints. However, they need a very large database and their computational time does not allow many characters in real-time. Semi-procedural animation systems [9] work with a small set of animations and use inverse kinematics only over the legs to ensure ground contact and to adapt the feet to possible slopes of the terrain, but they are unable to follow footstep trajectories.

Animation systems for autonomous agents must be computationally efficient to animate a multitude of characters in real-time, and need to follow different control trajectories, depending on the controller used. Controllers that account for animation constraints while computing control decisions such as motion graphs [10, 11, 12, 13] or precomputed search trees [14] can simply playback the animation sequence. These approaches try to reach the goal by connecting series of motion[15], which sometimes limits the movements of the agents. The main issues with motion graphs are that they require a very large amount of animation clips (over 400) and have a high computational cost which makes them not suitable for large groups of agents in real-time. Precomputed search trees can handle groups, but work with a few animation clips and are unable to synthesize new animations.

Approaches that ignore animation constraints produce center of mass trajectories for the animation system to follow. Different models include social forces [2], rule-based approaches [1], flow tiles [16], roadmaps [17], continuum dynamics [3], and force models parametrized by psychological and geometrical rules [4]. These techniques can easily simulate hundreds and thousands of characters in real-time, but do not account for locomotion constraints, thus producing artifacts such as foot-sliding which require correction and simulation updates [5].

Considering the root velocity as the input parameter for character control, numerous approaches can synthesize smooth, versatile and more plausible locomotion animations [18, 9]. Some approaches have also used the idea of selecting animations from a Delaunay triangulation of all the available animation clips [19, 20]. But all of these approaches are restricted to the root for performing character control.

There has been a recent surge in approaches that produce footstep trajectories for character control. They can be physically based but generated off-line [21], be generated online from an input path computed by a path planner [6], or use simplified control dynamics to produce bio-mechanically plausible footstep trajectories for crowds [7]. These approaches often show their animation results off-line using tools such as 3D Max [22].

Footstep-driven animation systems [23] produce unnatural results using procedural methods. The work in [24] uses a statistical dynamic model learned from motion capture data in addition to user-defined space-time constraints (such as footsteps) to solve a trajectory optimization problem. In [25] random samples of footsteps make a roadmap going from one point to another which is used to find a minimum-cost sequence of motions matching it and then retarget to the exact foot placements. The work in [26, 27] performs a global optimization over an extracted center of mass trajectory to maximize the physical plausibility and perceived comfort of the motion, in order to satisfy the footprint constraints. Recent solutions [6, 28, 29] adopt a greedy nearest-neighbor approach over larger motion databases. To ensure spatial constraints, the character is properly aligned with the footsteps and reinforced with inverse kinematics, while temporal constraints are satisfied using time warping. These techniques achieve highly accurate results in terms of foot positioning, but their computational cost makes them unsuitable for real-time animation of large groups of agents.

Comparison to Prior Work. Our method produces visually appealing results with foot placement constraints, using only a handful of motion clips, and can seamlessly follow footstep-based control trajectories while preserving the global appearance of the motion. Compared to [9], we exploit the combination of multiple parameter spaces for footstep-precision control. This reduces the dimensionality of the problem, compared to [29]. Unlike previous work in the literature, our method can synthesize animations for a large number of characters in real time, following footstep trajectories for different walking styles and even running motions with a small flying phase.

3. Framework Overview

Animating characters in real time animations has different requirements depending on the application. In many applications, the user only wants to control the direction of movement and speed of the root, but there are other situations where a finer control of the foot positioning is required. For example, the user may want to respect different walking gaits depending on the terrain, to make the character step over stones to cross a river, or walk through some space full of holes whilst avoiding falling. For this purpose we have developed a framework to animate virtual characters following footstep trajectories, while still being able to follow trajectories based on the movement of the COM when necessary.

Online locomotion systems [9] traditionally produce synthesized motions that follow a COM trajectory, with procedural corrections for uneven terrain. These methods can nicely follow COM trajectories, but they lack control over the style of walking and the kind of steps. For instance, we cannot control whether in order to walk fast, the character will move with large distances between steps or with a fast sequence of short steps. This is the main issue we address in our work: to provide an animation system that is able to accurately follow footstep trajectories while meeting real-time constraints, and that can scale to handle large groups of animated characters .

For this purpose, we introduce two parametric spaces based on the position of each foot: Ω_{f_L} and Ω_{f_R}, and switch between the two depending on the swing foot, as well as a parametric space based on the root movement Ω_{f_R}. Our technique takes into account both displacement (from Ω_{f_L} and Ω_{f_R}) and speed (from Ω_r) to ensure the satisfaction of both spatial and temporal constraints. Our system provides the user with the flexibility to choose between different control granularities ranging from exact foot positioning to exact root velocity trajectories. Fig. 2 shows our framework.

4. Footstep-based Locomotion

The main goal of the Footstep-based Locomotion Controller is to accurately follow a footstep trajectory, i.e., to animate a fully articulated virtual human to step over a series of footplants with space and velocity constraints. The system must meet real-time constraints for a group of characters, should be robust enough to handle sparse motion clips, and needs to produce synthesized results that are void of artifacts such as foot sliding and collisions.

4.1. Motion Clip Analysis

From a collection of cyclic motion clips[1], we need to extract individual footsteps. Each motion clip contains two steps, one starting with the left foot on the floor, and one starting with the right foot. A step is defined as the action where one foot of the character starts to lift-off the ground, moves in the air and finishes when it is again planted on the floor. We say that a footstep corresponds to one foot when that foot is the one performing the action previously described. The foot that stays in contact with the floor for most of the duration of the footstep is called the supporting foot, since it supports the weight of the body. This applies even for running motions, where the support foot goes into fly mode for a short phase of the footstep, but it is still the one supporting the weight during most of the footstep.

During an offline analysis, each motion clip m_i is annotated with the following information: (1) $\mathbf{v}_i^r$: Root velocity vector. (2) $\mathbf{d}_i^L$: Displacement vector of the left foot. (3) $\mathbf{d}_i^R$: Displacement vector of the right foot.

Similar to [9], animations are analyzed in place, that is, we ignore the original root forward displacement, but keep the vertical and lateral deviations of the motion. This allows an automatic detection of foot events, such as lifting, landing or planting, from which we can deduce the displacement vector of each foot. For example, the displacement vector of the left foot $\mathbf{d}_i^L$ is obtained by subtracting the right foot position at the instant of time when the left foot lands, from the right foot position at the instant of time when the left foot is lifting off. These displacements will be later used to move the whole character, eliminating any foot sliding. By adding $\mathbf{d}_i^L$ to $\mathbf{d}_i^R$ and knowing the time duration of the clip, we can calculate the average root velocity vector $\mathbf{v}_i^r$ of the clip m_i.

This average velocity is used to classify and identify animations, by providing an example point which is the input for the polar gradient band interpolator (where each example point represents a velocity in a 2D parametric space). Gradient band interpolation specifies an influence function associated with each example, which creates gradient bands between the example point and each of the other example points. These influence functions are normalized to get the weight functions associated with each example. However the standard gradient band interpolation is not well suited for interpolation of examples based on velocities. The polar gradient band interpolation method is based on reasoning that in order to get more desirable behavior for the weight functions of example points that represent velocities, the space in which the interpolation takes place should take on some of the properties of a polar coordinate system. It allows for dealing with differences in direction

[1] Although cyclic animations are not strictly required by our method, they help find smoother transitions between consecutive footsteps and are preferred by most standard animation systems [9].

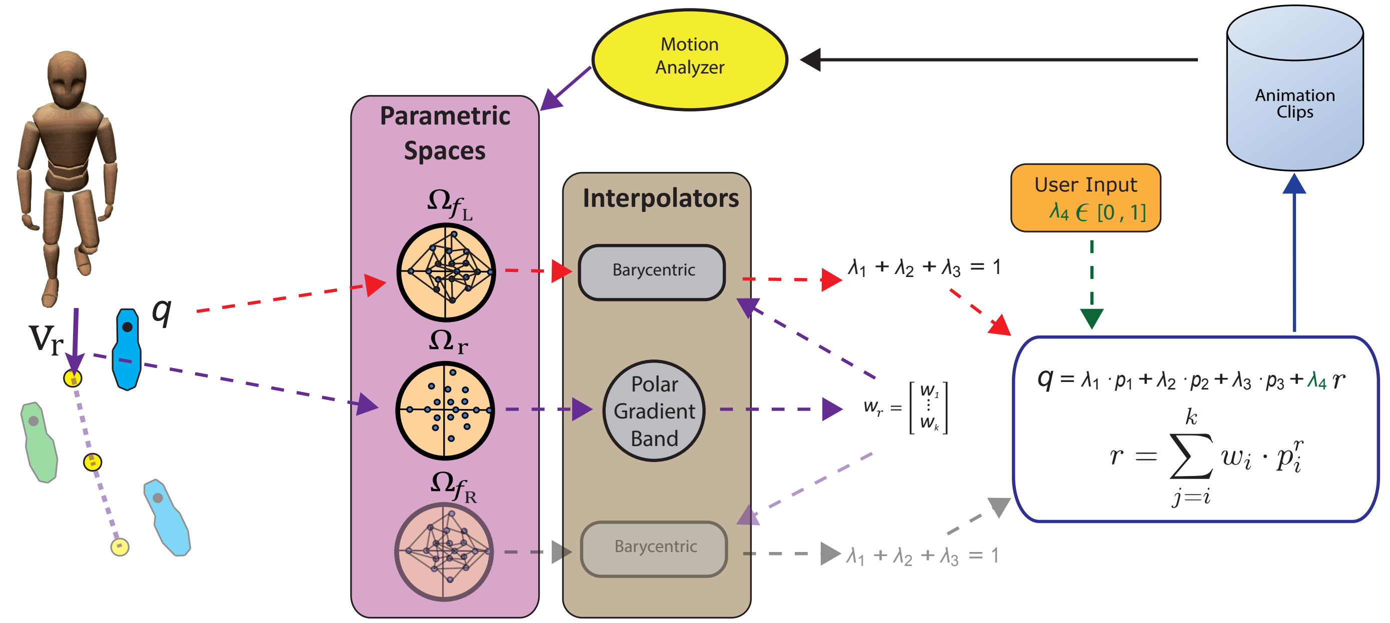

Figure 2: Online selection of the blend weights to accurately follow a footstep trajectory. Ω_r uses a gradient band polar based interpolator [9] to give a set of weights w_j, which are then used by the barycentric coordinates interpolator to tradeoff between footstep and COM accuracy.

and magnitude rather than differences in the Cartesian vector coordinate components. For more details we refer the reader to [9].

Each motion clip is then split into two animation steps A_i^L for the left foot and A_i^R for the right foot. For each foot, we need to calculate all the possible positions that can be reached based on the set of animation steps available. Since the same analysis is performed for both feet separately, from now on we will not differentiate between left and right for the ease of exposition. For each individual step animation A_i and given an initial root position, we want to extract the foot landing position p_i, if the corresponding section of its original clip was played. This is calculated by summing the root displacement during the section of the animation with the distance vector between the root projection over the floor and the foot position in the last frame.

The set $\{p_i | \forall i \in [1, n]\}$ where n is the number of step animations, provides a point cloud. Fig. 3 shows the Delaunay triangulation that is calculated for the point cloud of landing positions. This triangulation is queried in real time to determine the simplex that contains the next footstep in the input trajectory. Once the triangle is selected, we will use its three vertices p_1, p_2 and p_3 to compute the blending weights for each of the corresponding animations A_1, A_2 and A_3.

4.2. *Footstep and Root Trajectories*

Our system can work with both footstep trajectories and COM trajectories. A footstep trajectory will be given as an ordered list of space-time positions with orientations, whether it is precomputed or generated on-the-fly.

The input footstep trajectory may be accompanied by its associated root trajectory (a space-time curve, rather than a list of points, and an orientation curve), or else we can automatically compute it from the input footsteps by interpolation. This is

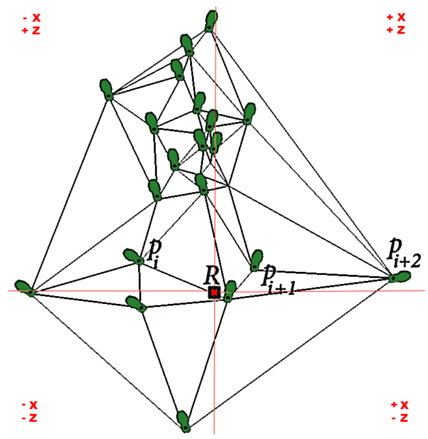

Figure 3: Delaunay triangulation for the vertices representing the landing positions (p_i, p_{i+1}, p_{i+2},...) of the left foot when the root, R is kept in place.

achieved by computing the projection of the root on the ground plane, as the midpoint of the line segment joining two consecutive footsteps. The root orientation is then computed as the average between the orientation vectors of each set of consecutive steps. This provides us with a sequence of root positions and orientations which can be interpolated to approximate the motion of the root over the course of the footstep trajectory.

4.3. *Online Selection*

During run time, the system animates the character towards the current target footstep. If the target is reached, the next footstep along the trajectory is chosen as the next target. For each footstep q_j in the input trajectory $\{q_1, q_2, q_3, ..., q_m\}$ we need to align the Delaunay triangulation graph with the current root position and orientation. Then the triangle containing the next foot position is selected as the best match to calculate the weights required to nicely blend between the three animations in order to achieve a footstep that will land as close as possible to the desired destination position q_j (Fig. 4). Notice that these weights are applied equally to all the joints in the skeleton, which means that at this stage we cannot accurately adjust the specific foot orientation required by each footstep in the input trajectory.

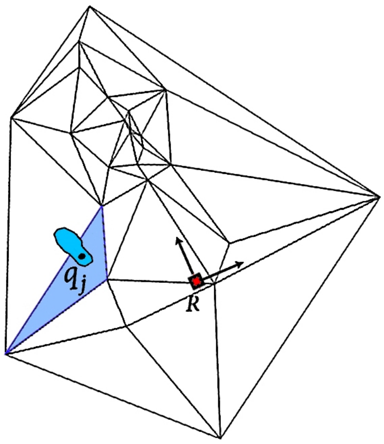

Figure 4: By matching root position and orientation, we can determine the triangle containing the destination position for the landing position q_j.

4.4. Interpolation

Footstep parameters change between successive footplants, remaining constant during the course of a single footstep (several frames of motion). Therefore we need to compute the best interpolation for each footstep, blend smoothly between consecutive steps, and apply the right transformation to the root in order to avoid foot-sliding or intersections with the ground.

To meet these requirements, we use a barycentric coordinates based interpolator in Ω_{f_L} and Ω_{f_R}, and constrain the solution based on the weights computed in Ω_r. This allows us to animate a character at the granularity of footsteps, while simultaneously accounting for the global motion of the full body.

If we only consider the footstep parametric space, then the vertices of the selected triangle are those that can provide the best match for the desired foot position. The barycentric coordinates of the desired footstep are calculated for the selected triangle as the coordinates that satisfy:

$$q_j = \lambda_1 \cdot p_1 + \lambda_2 \cdot p_2 + \lambda_3 \cdot p_3, \qquad (1)$$
$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

where p_1, p_2 and p_3 are the positions of the foot landing if we run animation steps A_1, A_2 and A_3 respectively. The calculated barycentric coordinates are then used as weights for the blending between animations. A nice property of the barycentric coordinates is that the sum equals 1, which is a requirement for our blending. Finally in order to move the character towards the next position, we need to displace the root of the character adequately to avoid foot sliding. The final root displacement vector, $\mathbf{d}^r_j$ is calculated as the weighed sum of the root's displacement of the three selected animation steps (Eq. 2), and changes in orientation of the input root trajectory are applied as rotations over the ball of the supporting foot.

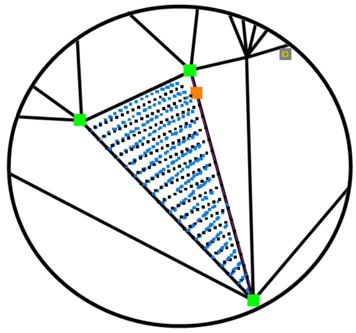

Figure 5: Offsets for different landing positions in a triangle, between barycentric coordinates interpolation (black dots) and blending the whole skeleton using SLERP (blue dots).

$$\mathbf{d}^r_j = \lambda_1 \cdot \mathbf{d}^r_1 + \lambda_2 \cdot \mathbf{d}^r_2 + \lambda_3 \cdot \mathbf{d}^r_3 \qquad (2)$$

This provides a final root displacement that is the result of interpolating between the three root displacements in order to avoid any foot sliding. It is important to notice that the barycentric coordinates provide the linear interpolation required between three points in 2D space to obtain the position q_j. This is an approximation of the real landing position that our character will reach, as the result of blending the different poses of the three animation clips, using spherical linear interpolation (SLERP) with a simple iterative approach as described in [30].

Therefore there will be an offset between the desired position q_j and the position reached after interpolating the three animations. To illustrate this offset, Fig. 5 shows the points sampled to compute barycentric coordinates in black, and in blue the real landing positions achieved after applying the barycentric weights to the animation engine and performing blending using SLERP. In order to correct this small offset at the same time that we adjust the feet to the elevation of the terrain and orient the footstep correctly, we incorporate a fast and simple IK solver.

4.5. Inverse Kinematics

An analytical IK solver modifies the leg joints in order to reach the desired position at the right time with a pose as close as possible to the original motion capture data. For footstep-based control, the desired foot position is already encoded in the footstep trajectory, and for COM trajectories the final position is calculated by projecting the current position of the foot over the terrain. The controller feeds the IK system with the end position and orientation for each footstep. This allows the system to handle footsteps on uneven terrain.

5. Incorporating Root Movement Fidelity

In some scenarios the user may be more interested in following root velocities than in placing the feet at exact footsteps or with specific walking styles. We present a solution to include root movement based interpolation in our current barycentric coordinates based interpolator through a user controlled parameter λ_4.

For this purpose, we incorporate the locomotion system presented by Johansen [9] to produce synthesized motions that follow a COM trajectory with correction for uneven terrain. During offline analysis, a parametric space is defined using all the root velocity vectors extracted from the clips in the motion database. For example, a walk forward clip at 1.5 m/s, and a left step clip at 0.5 m/s produces a parametric space using the root velocity vectors going from the forward direction to the 90^o direction, and with speeds from 0.5 m/s to 1.5 m/s.

Given a desired root velocity we define a parametric space Ω_r, and a gradient band interpolator in polar space [9] is created to compute the weights for each animation clip to produce the final blended result. The gradient band interpolator does not ensure accuracy of the produced parameter values but it does ensure smooth interpolation under dynamically and continuously changing parameter values, as with a player-controlled character. Once the different clips are blended with the computed weights, the system predicts the support foot position at the end of the cycle and projects it on the ground to find the exact position where it should land.

The root movement based interpolator will select a set of k animations A^r_1 to A^r_k with their corresponding weights: $w_1, ..., w_k$. Each of those animations provides a landing position $p^r_1, ..., p^r_k$, and if we only interpolated these animations we would obtain the landing point r.

In order to incorporate the output of the polar gradient band interpolator in the barycentric coordinates based interpolator we proceed as indicated in Algorithm 1.

The algorithm first checks whether a vertex of the current triangle $\langle p_1, p_2, p_3\rangle$ can be replaced by any of the three vertices with highest weights selected by the polar band interpolator, p^r_j, $j \in [1, k]$ (lines 1-13 in the algorithm). This replacement takes place if the distance between the two landing positions p_i and p^r_j is within a user input threshold ϵ (line 7), and the resulting triangle still contains the desired landing position q_j (function *IsInTriangle* returns true if q_j is inside the new triangle). This means that there is another animation that also provides a valid triangle and has a root velocity that is closer to the input root velocity.

Next, function *CalculateRootLanding* computes the landing position reached after blending the animations given by the root movement interpolator (Eq. 3).

$$r = \sum_{i=1}^{k} w_i \cdot p^r_i \tag{3}$$

Finally, *ComputeWeights* calculates the three λ_i for the next footstep q_j by incorporating a user provided λ_4 and the result of the polar band interpolator r (Eq. 4).

$$q_j = \lambda_1 \cdot p_1 + \lambda_2 \cdot p_2 + \lambda_3 \cdot p_3 + \lambda_4 \cdot r \tag{4}$$

Algorithm 1 Incorporating root movement fidelity

Input:
- The target position q_j,
- The current triangle $\langle p_1, p_2, p_3\rangle$,
- Root landing positions $\langle p^r_1, ..., p^r_k\rangle$,
- Animation weights $\langle w_1, ..., w_k\rangle \,|\, w_1 \geq ... \geq w_k$,
- A user input threshold ϵ,
- A user input weight parameter λ_4

Output: $\lambda_1, \lambda_2, \lambda_3$

1: **for** $i = 1$ **to** 3 **do**
2:   $u \leftarrow (i + 1) \mod 3$
3:   $v \leftarrow (i + 2) \mod 3$
4:   $j \leftarrow 1$
5:   *replaced* $\leftarrow$ false
6:   **while** $j \leq 3 \wedge \neg$*replaced* **do**
7:     **if** $\left\| p_i - p^r_j \right\| \leq \epsilon \;\wedge\; IsInTriangle\left(q_j, \left\langle p^r_j, p_u, p_v\right\rangle\right)$ **then**
8:       $p_i \leftarrow p^r_j$
9:       *replaced* $\leftarrow$ true
10:     **end if**
11:     $j \leftarrow j + 1$
12:   **end while**
13: **end for**
14: $r \leftarrow CalculateRootLanding\left(\left\langle p^r_1, ..., p^r_k\right\rangle, \langle w_1, ..., w_k\rangle\right)$
15: $\langle \lambda_1, \lambda_2, \lambda_3\rangle \leftarrow ComputeWeights\left(\langle p_1, p_2, p_3\rangle, \lambda_4, r\right)$

and λ_i are defined using the following relationship:

$$\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 1 \tag{5}$$

Since w_i and p^r_i are known $\forall i \in \{1, ..., k\}$, and λ_4 is a user input, we have a linear system, where λ_4 determines the trade-off between following footsteps accurately (if $\lambda_4 = 0$), and simply following root movement (if $\lambda_4 = 1$).

As the user increases λ_4 there will be a value $\beta \in [0, 1]$ for which λ_1, λ_2 or λ_3 will be negative, when solving the system of equations formed by eq.4 and eq.5. In order to avoid animation artifacts it is necessary to deal only with positive weights, therefore we guarantee that the system will only reproduce q_j accurately as long as $\lambda_4 < \beta$. If we further increase λ_4 beyond the value β then the algorithm will provide the blending values that correspond to a new point q' which is the result of a linear interpolation between q_j and point r. When $\lambda_4 = 1$ the resulting blending will be exclusively the one provided by the root movement trajectory since $\lambda_1 = \lambda_2 = \lambda_3 = 0$. Fig. 6 illustrates this situation.

Time Warping. Incorporating root velocity in the interpolation, does not always guarantee that the time constraints assigned per footstep will be satisfied. Therefore once we have the final set of animations to interpolate between, with their corresponding weights λ_i, $i \in \{1, 2, 3\}$ and w_j, $j \in [1, k]$, we need to apply time warping. Each input footstep f_m has a time stamp τ_m indicating the time at which position q_m should be reached (where $m \in [1, M]$ and M is the number of footsteps in the input trajectory). The total time of the current motion, T can

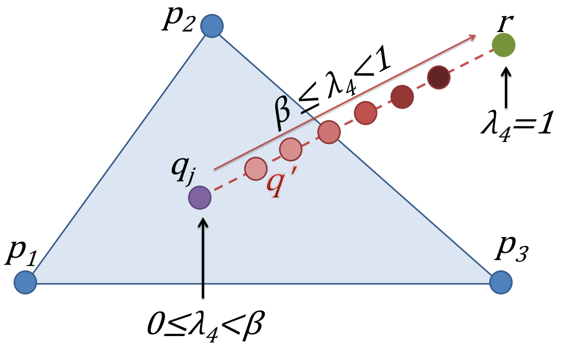

Figure 6: When solving the system of equations given by eq.4 and eq.5, the value of either λ_1, λ_2 or λ_3 will be negative when $\lambda_4 \geq \beta$. Therefore we need to calculate the barycentric coordinates for a new point q' which moves linearly from q_j to r as the user increases the value of λ_4 from β to 1. This means solving the system of equations for q' instead of q_j, as it is the closest point to the desired landing position which guarantees that all weights in eq. 5 will be positive.

be calculated as the weighted sum of the time of the animation steps being interpolated: $T = \sum_{i=1}^{3}(\lambda_i \cdot t(A_i)) + \sum_{j=1}^{k}(w_j \cdot t(A_j))$. Therefore the time warping factor that needs to be applied can be calculated as: $warp_m = (\tau_m - \tau_{m-1})/T$.

Outside the Convex-Hull. The footstep parametric space defines a convex-hull delimiting the area where our character can land its feet. When our target footstep position falls inside this area, clips can be interpolated to reach that desired position. But if it falls outside this convex-hull we still want the system to consider and try to reach it. Our solution to handle this problem consists of projecting orthogonally the input landing position q over the convex-hull to a new position q_{proj}. Our system then gives the blending weights for q_{proj} and applies IK to adjust the final position. We include a parameter to define a maximum distance for the IK to set an upper limit on the correction of the landing position. It is important to notice that even if the input trajectory has some footsteps that are unreachable with the current data base of animation clips, our system will provide a synthesized animation that will follow the input trajectory as closely as possible, until it recovers and catches up with future steps in the input trajectory. This situation is similar to the scenarios where the user increases λ_4 and then reduces it again.

6. Results

The animation system described in the paper is implemented in C# using the Unity 3D Engine [31]. The footstep trajectories used to animate the characters are generated using the method described in [7] or are created by the user. Some difficult scenarios, exercising careful footstep selection, are shown in Fig. 1 and Fig. 7. Agents carefully plant their feet over pillars (Fig. 7-a) or use stepping stones to avoid falling into the water (Fig. 7-b). We show our ability to handle over a hundred agents at 13 FPS (Fig. 7-c and Fig. 9). The supplementary video demonstrates additional results (high resolution video[2], low resolution video[3]).

Obstacle Course. We exercise the locomotion dexterity of a single animated character in an obstacle course. The character follows a footstep trajectory with different walking gaits , alternating running and walking phases (Fig. 1-a,b), and including sidesteps (Fig. 1-c) and backward motion (Fig. 1-e).

Stepping Stone Problem. Stepping stone problems (Fig. 7-b) require careful footstep level precision where constraints require the character to place their feet exactly on top of the stones in order to successfully navigate the environment. Our framework can be coupled with footstep-based controllers to solve these challenging benchmarks.

Integration with Crowd Simulator. We integrate our animation system with footstep-based simulators [7]; our character follows the simulated trajectories without compromising its motion fidelity while scaling to handle large crowds of characters (Fig. 7-c).

It is important to mention that the quality of the results depends strongly on the quality of the clips available from the motion capture library. As can be seen in the video, the least precise movements in our results are side steps and back steps. This is due to two reasons: (1) we had a small number of animations compared to other walking gaits, and thus triangles covering that space have larger areas, and (2) interpolation artifacts appear when blending between animations that move in opposite directions (for example a backwards step with a forward step). We believe that having a better and denser sampling in these areas will improve the results. For steps falling in triangles of smaller areas, and with all the vertices in the same quartile we have obtained results of high quality even for difficult animations such as running or performing small jumps.

6.1. Foot Placement Accuracy

The presented barycentric coordinates interpolator assumes a small offset between the results of linearly interpolating landing positions from the set of animations being blended, and the actual landing position when calculating spherical linear interpolation over the set of quaternions. This small offset depends on the area of the triangle, so as we incorporate more animations into our data base, we obtain a denser sampling of landing positions and thus reduce both the area of the triangles and the offset. We believe this is a convenient trade off since such a small offset can be eliminated with a simple analytical solver but the efficiency of computing barycentric coordinates offers great performance. It is also important to notice that if exact foot location is not necessary, and the user only needs to indicate small areas for stepping as in the water scenario, then it is not necessary to apply the IK correction. Fig. 8 shows the offset between the landing position and the footstep. The magnitude of the error is illustrated as the height of the red cylinders that are located at the exact location where the foot first strikes.

[2] https://www.dropbox.com/s/o1b9w73qd45fmip/videoCAG.mp4?dl=0
[3] https://www.dropbox.com/s/ptdz788f2k9ad3g/videoCAGlowRes.mp4?dl=0

(a) (b) (c)

Figure 7: (a) Agents accurately following a footstep trajectory and avoiding falls by carefully stepping over pillars. (b) The stepping stone problem is solved with characters avoiding falls into the water. (c) A crowd of over 100 agents simulated at interactive rates.

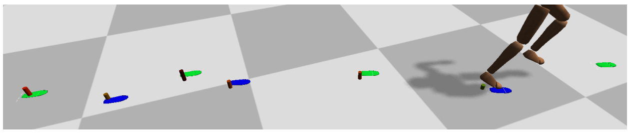

Figure 8: The red columns show the small offset between landing position and the footstep when the IK corrections are not being applied.

6.2. Performance

Fig. 9 shows the frame rate we obtain as we double the number of agents. It is important to notice that increasing the number of animations would enhance the quality and accuracy of the results, with just a small overhead on the performance.

The average time of the locomotion controller is 0.43ms, this process includes blending animations, IK, the polar band interpolator and our barycentric coordinates based interpolator. The computational cost of our footstep interpolator is 0.2 ms, which is amortized over several frames as the interpolation in Ω_{f_L} or Ω_{f_R} only need to be performed once per footstep. This time is divided between computing the root movement polar band interpolator which takes 0.155ms and our barycentric coordinates interpolator which takes 0.045ms. Performance results were measured on an Intel Core i7-2600k CPU at 3.40GHz with 16GB RAM.

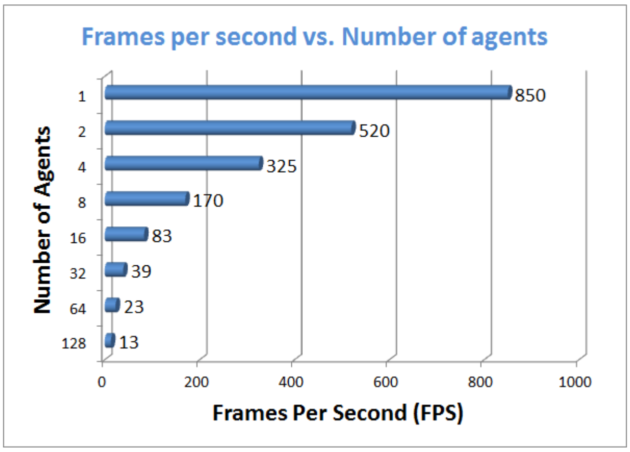

Figure 9: Performance of the Footstep Locomotion System in frames per second as the number of agents increases.

7. Conclusions and Future Work

We have presented a system that uses multiple parameter spaces to animate fully embodied virtual humans to accurately follow a footstep trajectory respecting root velocities, using a relatively small number of animation clips (24 in our examples). Our method is fast enough to be used with tens of characters in real time (25 FPS) and over a hundred characters at 13 FPS. The method can handle uneven terrain, and can be easily extended to introduce additional locomotion behaviors by grouping new sets of animation clips and generating different parametric spaces. For example, walking and running motions can be blended together, but if we wanted to add crawling motions or jumping motions, it would be better to separate them in different parametric spaces for each style. This will avoid unnatural interpolations that can appear when blending between very different styles. Having different parametric spaces requires some sort of classification, which could initially be done manually but it could also be based on the characteristics of the motion, such as changes in acceleration, maximum heights of the root, length of fly phase, etc. Assuming we can extract the parametric spaces for different animation types, it would also be necessary in some cases to have additional transition clips to switch between very different locomotion types, i.e. crawling and walking.

We do not run physical or biomechanical simulations, and use interpolation and blending between motion capture animations. Our method accuracy depends on the variety of animation clips, while its quality and efficiency depends on the number of clips. A trade-off between efficiency and accuracy is therefore necessary, for which we have found a good equilibrium.

Limitations. In order to reduce the dimensionality of the problem, we have not included in our parametric space the orientation of the previous footstep. Ignoring the final orientation of the character at the end of the previous step can induce some discontinuities between footsteps. We mitigate this effect by blending between footsteps automatically for a small amount of time (about 0.2 seconds) at the advantage of reducing the computational time and thus making our method suitable for large groups of agents in real time. Regarding the selection of animation at the end of each footstep, notice that in our database, left and right animation steps are extracted from complete animation cycles that are usually consistent in parameters such as velocity, acceleration and walking gait. Therefore for a given sequence of steps, the most likely animation steps to be chosen will be those extracted from the same set of animation cycles, thus resulting in smooth and natural transitions between very similar steps. When the characteristics of the steps change drastically, then our method needs to blend between steps from very different animation cycles. So in general, alternating left/right steps results in natural transitions with smooth continuity when blending animations, and only when the input step trajectory changes drastically between each pair of steps, we may observe transitions between animations that feel unnatural. This can happen if the step trajectory is done manually with artifacts due to the user's lack of experience creating footstep trajectories, or for example when the input trajectory forces the character to walk over artificially located steps, like crossing a river by stepping over stones. We would like to emphasize that this situation would also look awkward in the real world and thus the result of our synthesized animation may be the desired one.

Future Work. For future work we would like to extend our barycentric coordinates interpolator to 3D space with the third coordinate being the root velocity. This will free our system from the polar band interpolator which not only takes longer to compute but also selects too many animations which results in slower blending. One thing to explore could be to interleave the execution of the Footstep-based Locomotion Controller from different characters in different frames, ensuring we do not execute it for all the agents in the crowd.

Acknowledgements

This work has been partially funded by the Spanish Ministry of Science and Innovation under Grant TIN2010-20590-C02-01. A. Beacco is also supported by the grant FPUAP2009-2195 (Spanish Ministry of Education). The research reported in this document/presentation was also performed in connection with Contract Number W911NF-10-2-0016 with the U.S. Army Research Laboratory. The views and conclusions contained in this document are those of the authors and should not be interpreted as presenting the official policies or position, either expressed or implied, of the U.S. Army Research Laboratory, or the U.S. Government. Citation of manufacturers or trade names does not constitute an official endorsement or approval of the use thereof. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] C. W. Reynolds, Flocks, herds and schools: A distributed behavioral model, in: ACM SIGGRAPH, Vol. 21, 1987, pp. 25–34.

[2] D. Helbing, I. Farkas, T. Vicsek, Simulating dynamical features of escape panic, Nature 407 (2000) 487–490.

[3] A. Treuille, S. Cooper, Z. Popović, Continuum crowds, in: ACM Transactions On Graphics, Vol. 25, 2006, pp. 1160–1168.

[4] N. Pelechano, J. M. Allbeck, N. I. Badler, Controlling individual agents in high-density crowd simulation, in: ACM Symposium on Computer Animation, 2007, pp. 99–108.

[5] N. Pelechano, B. Spanglang, A. Beacco, Avatar Locomotion in Crowd Simulation, International Journal of Virtual Reality 10 (2011) 13–19.

[6] A. Egges, B. van Basten, One Step at a Time: Animating Virtual Characters Based on Foot Placement, The Visual Computer 26 (6-8) (2010) 497–503.

[7] S. Singh, M. Kapadia, G. Reinman, P. Faloutsos, Footstep Navigation for Dynamic Crowds, Computer Animation and Virtual Worlds 22 (2011) 151–158.

[8] A. Treuille, Y. Lee, Z. Popović, Near-Optimal Character Animation with Continuous Control, ACM Transactions on Graphics 26 (3) (2007) 7.

[9] R. S. Johansen, Automated Semi-Procedural Animation, Master Thesis. URL `http://runevision.com/thesis/`

[10] L. Kovar, M. Gleicher, F. Pighin, Motion graphs, in: ACM SIGGRAPH, 2002, pp. 473–482.

[11] L. Zhao, A. Safonova, Achieving Good Connectivity in Motion Graphs, Graphical Models 71 (4) (2009) 139–152.

[12] C. Ren, L. Zhao, A. Safonova, Human Motion Synthesis with Optimization-Based Graphs, Computer Graphics Forum 29 (2).

[13] J. Min, J. Chai, Motion Graphs++, ACM Transactions On Graphics 31 (6) (2012) 1.

[14] M. Lau, J. J. Kuffner, Precomputed search trees: planning for interactive goal-driven animation, in: ACM Symposium on Computer Animation, 2006, pp. 299–308.

[15] A. Witkin, Z. Popovic, Motion warping, in: ACM SIGGRAPH, 1995, pp. 105–108.

[16] S. Chenney, Flow tiles, in: ACM Symposium on Computer Animation, 2004, pp. 233–242.

[17] A. Sud, E. Andersen, S. Curtis, M. Lin, D. Manocha, Real-time path planning for virtual agents in dynamic environments, in: IEEE Virtual Reality, 2007, pp. 91–98.

[18] P. Glardon, R. Boulic, D. Thalmann, Robust on-line adaptive footplant detection and enforcement for locomotion, The Visual Computer 22 (3) (2006) 194–209.

[19] J. Pettré, J.-P. Laumond, T. Siméon, A 2-stages locomotion planner for digital actors, in: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '03, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2003, pp. 258–264.

[20] J. Pettre, J.-P. Laumond, A motion capture-based control-space approach for walking mannequins: Research articles, Computer Animation and Virtual Worlds 17 (2) (2006) 109–126.

[21] M. Felis, K. Mombaur, Using Optimal Control Methods to Generate Human Walking Motions, Motion in Games (2012) 197–207.

[22] Autodesk, 3d studio max, www.autodesk.com/products/autodesk-3ds-max/overview (2014).

[23] M. Girard, A. A. Maciejewski, Computational modeling for the computer animation of legged figures, in: SIGGRAPH, ACM, 1985, pp. 263–270.

[24] J. Chai, J. K. Hodgins, Constraint-Based Motion Optimization Using A Statistical Dynamic Model, ACM SIGGRAPH.

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

[26] H. Ko, N. I. Badler, Animating human locomotion with inverse dynamics, IEEE Computer Graphics & Applications 16 (2) (1996) 50–59.

[27] M. van de Panne, From Footprints to Animation, Computer Graphics Forum 16 (4) (1997) 211–223.

[28] B. van Basten, P. W. A. M. Peeters, A. Egges, The Step Space : Example-Based Footprint-Driven Motion Synthesis, Computer Animation and Virtual Worlds 21 (May) (2010) 433–441.

[29] B. van Basten, S. Stüvel, A. Egges, A Hybrid Interpolation Scheme for Footprint-Driven Walking Synthesis, Graphics Interface (2011) 9–16.

[30] A. Shoulson, N. Marshak, M. Kapadia, N. I. Badler, ADAPT : The Agent Development and Prototyping Testbed, ACM SIGGRAPH I3D.

[31] Unity, Unity - game engine (2014). URL `http://unity3d.com/`