Visual Comput (2006) 22: 399–414
DOI 10.1007/s00371-006-0017-3

ORIGINAL ARTICLE

Pascal Glardon
Ronan Boulic
Daniel Thalmann

Dynamic obstacle avoidance for real-time character animation

Published online: 17 May 2006
© Springer-Verlag 2006

P. Glardon · R. Boulic (✉) · D. Thalmann
Ecole Polytechnique Fédérale
de Lausanne (EPFL), Virtual Reality Lab,
CH-1015 Lausanne, Switzerland
ronan.boulic@epfl.ch

Abstract This paper proposes a novel method to control virtual characters in dynamic environments. A virtual character is animated by a locomotion and jumping engine, enabling production of continuous parameterized motions. At any time during runtime, flat obstacles (e.g. a puddle of water) can be created and placed in front of a character. The method first decides whether the character is able to get around or jump over the obstacle. Then the motion parameters are accordingly modified. The transition from locomotion to jump is performed with an improved motion blending technique. While traditional blending approaches let the user choose the transition time and duration manually, our approach automatically controls transitions between motion patterns whose parameters are not known in advance. In addition, according to the animation context, blending operations are executed during a precise period of time to preserve specific physical properties. This ensures coherent movements over the parameter space of the original input motions. The initial locomotion type and speed are smoothly varied with respect to the required jump type and length. This variation is carefully computed in order to place the take-off foot as close to the created obstacle as possible.

Keywords Human body simulation · Real-time motion blending · Motion planning · Obstacle avoidance

1 Introduction

In computer graphics, many methods and techniques have been developed to animate virtual humans, with the same principal objectives: believability, efficiency and adaptability. Motion capture [26] produces incontestably realistic human animations, but needs post-processing work to filter and adapt the raw recorded data. Editing methods have, therefore, been developed, such as motion interpolation or blending, producing new motions by combining multiple clips according to time-varying weights. This provides effective ways of adapting motion clips to the desired constraints specified by animators. However, these are sometimes complex to formulate, notably for real-time animation in dynamic environments.

In this paper, we present a method that determines these constraints automatically so as to animate an autonomous character in a dynamic environment. As opposed to traditional methods that consider the objects (or obstacles) of a scene as static [9, 14, 32] or as described with pre-defined trajectories [16, 23], we aim at handling obstacles that are not known in advance, generated on-the-fly during the animation process. Based on a locomotion engine, the character moves freely in a virtual environment. As soon as an obstacle is visible for the character, the method decides whether to get around or to jump over

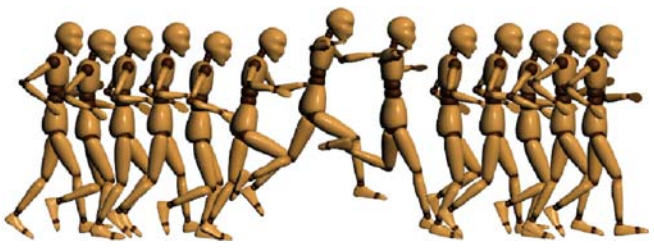
Fig. 1. A transition from locomotion to jump generated by our blending method

it, and then constructs the appropriate animation on-the-fly. This mechanism is composed of three phases.

The first phase aims to determine if the jump is physically feasible with respect to the current animation parameters such as the locomotion speed, obstacle length, distance between the character and the obstacle. According to the resulting decision, the second phase establishes a path that the character has to follow. A trajectory results from this path, providing the necessary linear and/or angular speed variation, either to avoid the obstacle or to place the take-off foot as close to the obstacle as possible. Finally, the last phase consists in generating the animation with respect to the parameter variation by applying the animation engine presented in [11], which produces the exact desired motion. In the case of jumping over the obstacle, we elaborate an on-line motion blending method that ensures a smooth transition from a locomotion pattern to a jump sequence without knowledge of their parameters in advance.

To produce realistic animation results, the motion blending method needs the implementation of time warping techniques, where correspondences between the motions are identified and then aligned. Instead of applying time warping methods where all input motions have to be pre-processed [19, 34], we opt for an approach similar to [27] that allows dynamic blending of motions not known in advance. However, its disadvantage is that the correspondences of motion structures have to be done manually, by annotating the time constraints of the feet, referred to as the support phases. We therefore propose an automatic method based on a subset of pre-labeled motions. Owing to a relation between these motions and their support phases, the time constraints of a new generated motion are detected on-the-fly.

All of the above mentioned methods perform blending as soon as it is requested by an animator. As a consequence, the resulting produced motion is sometimes not coherent: blending two walking motions that are out-of-phase, for example. This stems from either an incorrect chosen transition time and/or duration, or from a violation of dynamic motion properties. Hence, our method controls the blending from locomotion to jump by choosing the appropriate transition time and duration. In addition, the locomotion speed is adapted to the required jump for the given obstacle so as to minimize dynamic incoherence.

The first contribution of this paper concerns the elaboration of a blending technique for virtual characters, allowing the transition from locomotion to jump and ensuring coherence of motion properties (see Fig. 1). Using this technique, the second contribution consists in real-time one speed variation, which places the take-off foot as close to an obstacle as possible for a given final run-up speed. This obstacle “appears” dynamically in front of the moving character, at a distance not known in advance. As opposed to a previous technique [23] based on a search technique for the best appropriate motion into a motion clip database, we generate on-the-fly the exact motion corresponding to the requirements.

The remainder of this paper is organized as follows. Related work is reviewed in the next section. Section 3 presents an overview of our method, while the following section describes the control of a blending operation. Then we explain the path generation to get around or jump over an obstacle. Finally, we conclude in Section 6 with results and a discussion.

2 Related work

The method presented in this paper is related to two main research directions. The first one concerns the generation of transitions with continuous parameter variation. We use two animation engines, one for the locomotion and the other for the jump, explained in [11]. However, we still have to generate the transition between them. We focus, therefore, on motion blending techniques in the first part of this related work section. The second research direction encompasses the motion planning methods allowing the modification of a character’s action according to a goal, in order to avoid obstacles in the environment.

2.1 Motion blending

Nowadays it is common to use motion capture data to generate parameterized animations either by interpolating, assembling or blending original motion clips.

Two or more motions can be interpolated to produce a new parameterized one, by applying different techniques. The bi-linear [15] or multivariate interpolation [34] method generates new motions over a multi-dimensional parameter space. Recently, Mukai and Kuriyama [28] improved the interpolation function construction described in [34] by defining a specific kernel function for each input motion according to its characteristics. Other approaches based on statistical methods using PCA (principal component analysis) [10, 11] produce parameterized motion in a very efficient way, by computing them in a low-dimensional space.

Concerning assembling animation sequences, a large motion capture database can be organized in a motion graph [2, 20, 25], where edges represent pieces of original motions or generated transitions, and node choice points where motions join seamlessly. These points are automatically computed by comparing pairs of motions, frame by frame. A metric function measures the distance between two frames: a result smaller than a specified threshold indicates a transition between these two frames.

However, this transition threshold has to be manually quantified, as different kinds of motions have different fidelity requirements. In addition, some metric functions contain non-intuitive weight values, which are difficult to parameterize. For example, the proposed set of weights used for the cost metric function in [25] is compared to an optimal set in [38]. Even if the presented user study — applied on one performer without highly dynamic motions — reveals better results for the optimal set of weights proposed in [38], this latter method remains difficult to evaluate under various conditions.

Another problem of the motion graph technique concerns the transition duration. Actually, even if its starting point is found automatically, its duration has to be determined manually by giving the number of transition frames around the starting point. Moreover, adding a new motion on-the-fly is impossible, as all input motions have to be compared pairwise to construct the directed graph.

Motion blending is a basic way either to create new motions by interpolation or to generate transition between clips. Precursor works treat the motion as a time-varying signal. Signal processing techniques have, therefore, been developed to perform blending between motions by varying the frequency bands of the signal [7], or the Fourier coefficients [37]. However, disadvantages appear: the transition method in [37] is not invertible (a transition from run to walk is impossible), support phases of the feet are not considered in [7], and transition time and duration have to be specified by hand.

In [19, 29] blending is controlled by changing blend weights of input motions, during a period of time given by the user. Unlike [20], the time constraints (or support phases) change smoothly according to these weight values. However, this solution does not allow one to add a new input motion dynamically. It needs to be compared to the existing ones. In contrast, Menardais et al. in [27] propose a synchronization method to perform dynamic blending. If a requested transition is currently unfeasible due to incompatible support phases, the blending is shifted until it becomes possible. Another approach [3] proposes a decouple blending to resolve diverse coordination configurations between upper and lower body-halves, restricted to cyclic motions. Relevant events of input sequences are manually labeled and correspondences between them are established. Therefore, all motions have to be known in advance. In addition, the blending is performed at an interactive frame rate.

Even if all these blending techniques are widely accepted and used, transitions are not totally controlled and can produce incoherent results. In the study of [38], sequences with fixed or variable transition durations have been evaluated by some volunteers. As a result, the users preferred transition durations that are adapted to the context.

Finally, approaches taking dynamics into account, like [35, 39], can control valid transitions by modifying physical parameters such as angular velocity or the center of mass. However, the parameter setting is not straightforward and the motion generation is an off-line process. Recently, a momentum-based method was developed in [1]. Starting from a single motion capture sequence, a space of new physically plausible motions is pre-computed without parameterization effort. Although on-line blending can be performed between these motions, the method is limited to high dynamic movements. Furthermore, the transition time and duration are constant.

This short survey in transition generation reinforces our motivation to develop a methodology able to compute automatically not only transition duration, but also transition time, and to adapt on-the-fly motion parameters to ensure the coherence of the blended motions.

2.2 Motion planning

Motion planning increases the autonomy of virtual characters. Generally, given initial conditions and a goal, a path planner method provides a path to follow. Concerning the special case of locomotion tasks, the planner provides the path taking into account the obstacle avoidance problem. Additionally, this path is transformed into a trajectory that returns the adequate locomotion parameters at each time.

The probabilistic roadmap methods (PRM) sample the configuration space randomly as preprocessing, to construct an accessible point graph called a roadmap. This latter is then used to search for a path during the planning stage [17]. These methods have demonstrated good performance empirically for difficult problems. Moreover, they are generic, applicable to navigating car-like robots with non-holonomic constraints, humanoid robots with many degrees of freedom, and also molecule movements in biochemistry.

Pettre et al. [31, 32] apply PRM to get a collision-free 2D path in a 3D environment. This path is represented by Bezier curves and generates a trajectory encapsulating linear and angular speed variations. Finally, a walking animation is generated according these variations. Another approach based on PRM is presented in [9]. The roadmap is constructed by randomly sampling foot positions and orientations to form the graph nodes. The connection between two nodes is possible if a subsequence searched in a motion capture database approximates the given foot configuration well. The final animation from a given start and goal position is obtained by traversing the graph. The input motion clips are adapted to the foot configuration described in the graph nodes. However, these techniques are only designed for static environment and are not adapted to real-time context.

Other approaches based on A* algorithm allow to search a path into a tree structure at interactive or even real-time performance. Kuffner [21] proposes to subdivide the 2D scene projection into a regular grid of cells. Each cell is labeled as free or occupied by an object. A collision-free path is then searched using a dynamic programming algorithm such as A*. The final animation is generated using only a single walking cycle played at varying frequency according to a proportional derivative controller. However, due to the 2D projection, the obstacles are considered as a wall and, therefore, cannot be stepped over.

Recently, finite state machines (FSM) have been used to abstract different behaviors [8, 23]. From a current behavior state, FSM proposes a set of possible next behavior states to be reached. The optimal one is then chosen by applying an A* search algorithm so as to find the solution that is less costly. Hence, the results are highly dependent on the database size. In addition, the approach in [23] allows one to jump over obstacles. However, they have to be assigned to specific motion clips. It is therefore not possible to add an obstacle dynamically in the scene without having its corresponding jumping motion in the database.

The previous approaches are not well-adapted for dynamic scenes where the number and position of the obstacles are not known in advance. In [30], the method proposes to modify the computed path jammed by a moving obstacle. The solution is based on the work of Khatib et al. [18] using repulsive force assigned to the obstacle. However, this technique is off-line and was applied only to the hand movements. Kathib et al. [6] introduce a method that represents the path as a deformable elastic strip. From a previously planned motion, the path can be modified in real-time according to moving obstacles intruding it. Originally designed for robot motions, this method can be applied to virtual humans. However, the skeleton is controlled dynamically, which therefore limits the approach to motion patterns for which one masters such control techniques (e.g. skiing in [18]).

Finally, the approach in [14] allows the steering of a 3D point into a dynamic environment by selecting the best trajectory among predefined ones. However, this technique is well-adapted for 3D point animation such as a spacecraft, but still difficult to adapt for virtual humans whose motion constraints are much more complex. In fact, the pre-computed trajectories have a constant linear speed.

3 Method overview

Our method is based on a state machine model composed of ten states (see Fig. 2). The neutral state corresponds to the animation of a character that is moved freely in an environment by a locomotion engine based on motion captured data introduced in [11]. Its attached parameter vector $\boldsymbol{w}$ allows the real-time variation of a stylistic weight vector $\boldsymbol{w}_{subj}$ composed of v_{tot} captured subject weights, a locomotion weight w_{loco} from walking ($w_{loco} = 0$) to running ($w_{loco} = 1$) and the speed s. Jumping motions are generated analogously, by substituting the parameter jump length l for the speed s. The locomotion weight w_{loco} is either equal to 0 for the walking jump or 1 for the running jump.

This state machine is composed of a sub-part responsible for the transition generation from locomotion to a jump sequence, as described in [12]. The underlying idea is to adapt the current locomotion in order to be coherent with the requested jump. It results in a speed and locomotion type variation performed in state 1. Once this variation is finished, the blending method starts the transition as

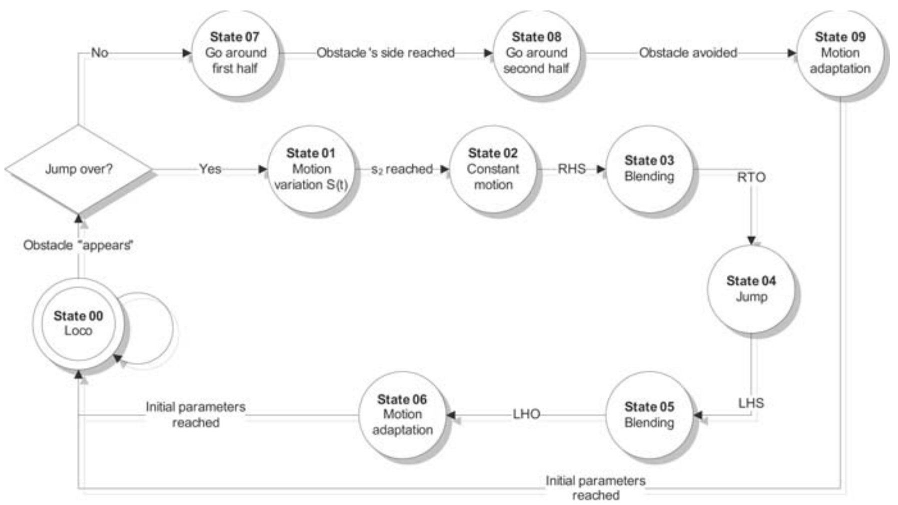

Fig. 2. The state machine allowing either to go around or to jump over an obstacle

soon as the support phases of both motions contain similarities (state 3). A blending operation is then performed, from a walking (or running) motion to a jumping motion, during the support phase. After that, the jumping motion is played (state 4) until another compatible support phase is detected. During this phase, the transition back to walking (or running) is executed (state 5). Finally, the locomotion engine returns to its neutral state, by modifying the motion parameters to their initial values (state 6).

The remaining states of the state machine are specific to the motion planning problem. As soon as an obstacle "appears" dynamically in front of the moving character, the choice whether or not to clear an obstacle is determined by performing different tests. The results depend on the current motion parameters, the obstacle dimension and its distance from the character. For the situation where the obstacle is jumped over, a speed variation (state 1) is computed so as to place the take-off foot as close as possible to the obstacle, with a specific run-up speed determined by our blending method. The alternative situation consists in getting around the obstacle. It involves a linear and angular speed variation so as to follow the path that goes to one obstacle side. This stage is determined using a steering method divided into two phases (state 7 and 8).

In the next section, we explain in detail first the blending mechanism and second our motion planning algorithm.

4 Controlled motion blending

We describe the method to control a blending operation from locomotion to a jump, composed of three stages. First the locomotion is adapted to generate a run-up phase compatible with the requested jump, ensuring a coherent transition. The second stage determines the transition time and duration automatically. Finally, the transition itself is performed by aligning in time the two blended motions. Without loss of generality, we consider in this paper only jumps executed with the right take-off foot.

4.1 Coherent motion adaptation

Imagine that a jump, described with a parameter $\boldsymbol{w}_{jump}$, is requested at time t_1, while the character is animated by a locomotion cycle parameterized with $\boldsymbol{w}_1$ whose speed is s_1. Two conditions have to be satisfied before effectively operating the transition: the final run-up speed s_2 to be reached before the jump execution at time t_2, and the run-up duration $t_{runup} = t_2 - t_1$.

Final run-up speed. In order to preserve the dynamic coherence, a jump of a specific length l can only be performed for a locomotion speed range. To determine this range, we analyzed jumps from a motion capture database to establish a relationship between the jump length and the run-up speed. Actually, the actors were asked to select their most natural run-up speed to execute a jump of a given length. This limits high dynamic variations between the run-up phase and the jump. Therefore, the speed s_2 measured just before the jump execution can be approximated by the mean speed of this jump. We extract the mean speed of all jumps in the database. This is computed by dividing the traveled distance of the character's root by the jump duration. Figures 3 and 4 illustrate the resulting relation between the jump length l and the mean speed s for walking and running jumps, respectively. Clearly, the best approximation function of these data is obtained by performing a linear least squares fit. Two functions are then constructed, one for each type of jump (f_W and f_R, for walking and running, respectively), returning the mean speed that corresponds to the run-up speed s_2 of a given jump length:

$$f_W(l) = s_2 = 0.97l - 0.04 \pm 0.1\,, \tag{1}$$

$$f_R(l) = s_2 = 1.16l - 0.13 \pm 0.15\,. \tag{2}$$

Note that we add a tolerance to these functions. Actually, depending on the skill level of the performer, a given jump length matches several acceptable mean speeds. This

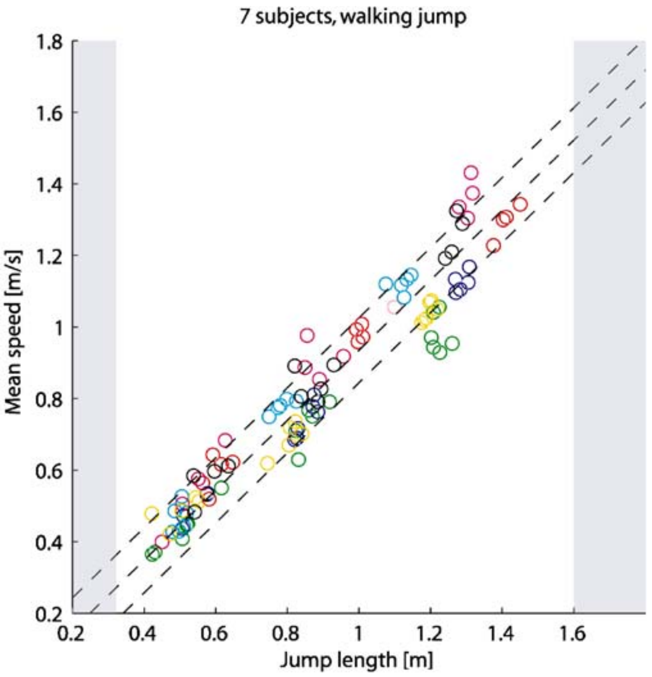

Fig. 3. Walking jumps: relation between the mean speed and jump length. Given a jump length, the stripe delimited by the linear approximation (middle dashed line) and the lower and upper tolerances defines allowed speed values just before the jump. The gray zone indicates not commonly feasible jumps

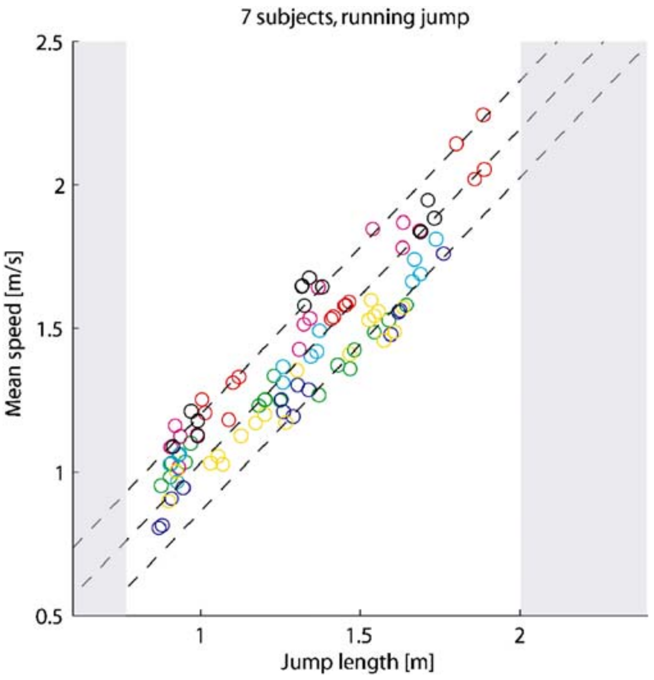

Fig. 4. Running jumps: relation between the mean speed and jump length. Given a jump length, the stripe delimited by the linear approximation (middle dashed line) and the lower and upper tolerances defines allowed speed values just before the jump. The gray zone indicates not commonly feasible jumps

speed range is illustrated in Figs. 3 and 4 by the stripe built by the positive and negative tolerances of Eqs. 1 and 2.

Therefore, given a jump length, a range of speed values is determined. If the current locomotion speed s_1 is outside of this range, we propose two motion adaptation scenarios:

- ▶ The priority is given to the jump execution. A new jump is generated, having the appropriate length and type which correspond to the current locomotion parameter. This solution is well adapted to game environment where the user's actions need to be quickly executed.
- ▶ The priority is given to the jump parameters. The current motion is adapted by a continuous speed variation until an acceptable speed that the jump requires is reached. Additionally and simultaneously, the current locomotion weight w_{loco} is adapted to the one of the jump. When the jump is finished, the locomotion parameters are gradually restored to their initial values. This solution is interesting when a character has to clear accurately an obstacle having a specific dimension. This problem is addressed later in this paper.

Run-up duration. The determination of the run-up phase duration t_{runup} is the second property ensuring a coherent transition. We first describe an approach presented in [12], which is based on a constant speed variation. According to the linear speed variation $\Delta s = s_2 - s_1$, the duration t_{runup} has to be chosen so that the induced acceleration is less than a threshold ε_a:

$$a = \frac{\Delta s}{t_{runup}} \leq \varepsilon_a \,. \tag{3}$$

To compute t_{runup}, we consider the next possible time when the transition to a jump is possible. In fact, a transition starts only at a specific foot event that occurs once during a locomotion cycle. To facilitate the foot event identification, we use a generic time referred to as the locomotion phase φ_i and defined in the interval $[0 \ldots 1]$. The phases $\varphi_i = 0$ and $\varphi_i = 1$ correspond, respectively, to the beginning and the end of the cycle (or jump sequence).

According to the elapsed time Δt between two consecutive frames, the current phase φ_i is updated as follows:

$$\varphi_i = \varphi_{i-1} + \Delta t F(\boldsymbol{w}) \,, \tag{4}$$

where $F(\boldsymbol{w})$ is the frequency function defined in [11]. This function returns the inverse duration of the locomotion cycle (or jumping sequence) characterized by the parameter vector $\boldsymbol{w}$.

For the transition from locomotion to jump, we define φ_1 the phase at time t_1 and φ_2 the phase at time t_2. The run-up duration can, therefore, be approximated using Eq. 4:

$$t_{runup} = t_2 - t_1 = (\varphi_2 - \varphi_1) \frac{2}{F(\boldsymbol{w}_1) + F(\boldsymbol{w}_1)} \,, \tag{5}$$

where the frequency variation is approximated by considering it as constant.

In order to determine the necessary number of locomotion cycle before performing the jump, Eq. 5 is evaluated by setting $\varphi_2 = 1$. This operation is repeated by adding 1 to φ_2 (i.e. an additional cycle) as long as the returned t_{runup} does not satisfy the acceleration condition described in Eq. 3.

However, this approach involves a constant speed variation during this run-up phase and can generate several locomotion cycles before executing the jump. Hence the take-off foot position is not fixed and depends on the current motion and jump parameters. We address this problem in the Sect. 5.

4.2 Transition time and duration

The locomotion parameters being compatible with the jump ones, the transition time and duration has to be determined.

First, we define eight foot events (Table 1) that interact with the floor. These are needed to identify the different support phase types described in Fig. 5. A support phase is defined as a period of time during which one feet (single support) or two feet (double support) touch the ground.

Table 1. Foot events interacting with the floor

Event name	Abbreviation
Right heel strike	RHS
Right heel off	RHO
Right toe strike	RTS
Right toe off	RTO
Left heel strike	LHS
Left heel off	LHO
Left toe strike	LTS
Left toe off	LTO

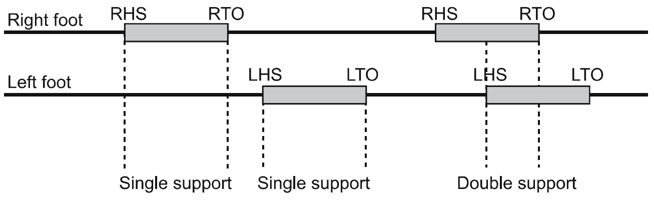

Fig. 5. Description of the support phase types

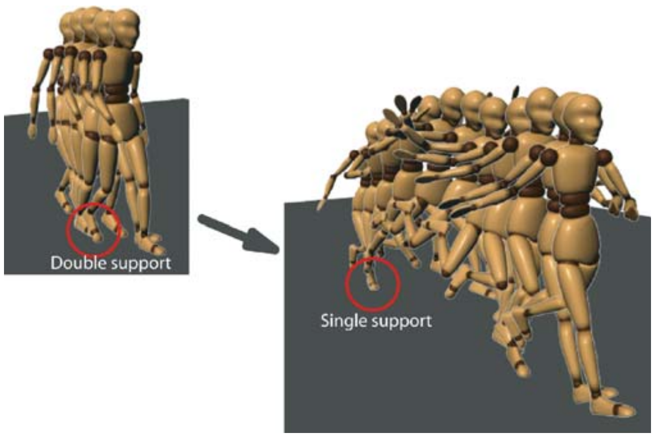

Fig. 6. Incompatible support phases to blend a walk with a jump motion

We simplify this definition by assuming that the heel always touches the floor before the toe, and inversely when the foot leaves the floor.

Many works [19, 27, 34] have demonstrated the necessity to take into account the support phases when blending is performed. In fact, a specific support phase (e.g. a right hopping with a left hopping) cannot be blended as they violate the motion properties. The support phases also allow one to blend several motions in a synchronous way. The algebraic relation proposed in [27] determines dynamically the transition time by defining and checking the support phases' compatibility of the blended motions. For example, a double support is always compatible with a single support.

However, this latter method is not general enough. For example, the double support defined from LHS to RTO in a walking motion (Figs. 5 and 6) is incompatible with the single support from RHS to RTO for a jump starting with the right take-off foot. Actually, the right leg is in front of the character trunk at the double support and back at the single support. Our method avoids this drawback by guaranteeing that the blending from locomotion to jump occurs during the single support phase defined between RHS and RTO, and similarly from jump to locomotion between LHS and LTO. In that way, those support phases determine not only the transition time, but also the transition duration.

Still, the foot events must be detected from the blended motions. As our method is foreseen to work dynamically, without knowing those motions in advance, these events have to be detected in real-time. Therefore, the motions have to be labeled, in order to know which type of support phase happens at which generic time.

In our situation, only a few walking and running motions generated by the engine have been labeled ($\sim 14\%$ of the input data), by selecting specific speeds and by considering the eight foot events described in the previous section. We observe that, given a subject and a type of locomotion, the different events occur almost independently of the speed normalized by the leg length of the character, for values between 0.8 and 2.2 [ms^{-1}]. Analogously, a subset of jumping motions ($\sim 25\%$) have been labeled, leading to the conclusion that, given a subject, the event timing is linearly dependent on the jump length, as shown in Fig. 7.

From those observations, the feet constraints of any walking, running or jumping motion generated by the engines can be instantaneously and automatically computed.

4.3 Time-aligned blending

The support phases of two blended motions may have different durations, depending on the motion properties. For a similar jump, one subject can put the take-off foot longer on the floor longer than another one. Hence, it is necessary to align in time the support phases that are blended.

Let $\boldsymbol{A}$ be a motion where frames $\mathcal{F}_{a1}$ and $\mathcal{F}_{a2}$ delimit a support phase that has to be blended with the motion $\boldsymbol{B}$ having a compatible support phase from frames $\mathcal{F}_{b1}$ to $\mathcal{F}_{b2}$. According to the elapsed time Δt between two consecutive frames, we have to compute the time aligned frames $\mathcal{F}_a$ and $\mathcal{F}_b$, as illustrated in Fig. 8.

While the motions are generated in a generic time and have an identical total frame number f_{max}, the frequencies F_a and F_b associated to the motion $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively, are used to update the current motion phase φ_i. During the blending, we define the frequency F, which varies from F_a to F_b, and a phase variable φ_i. For each Δt, the phase is updated as described in Eq. 4. Hence, the current frame $\mathcal{F}_a$ of the motion $\boldsymbol{A}$ is computed as follows:

$$\mathcal{F}_a = \varphi_i f_{max} \,. \tag{6}$$

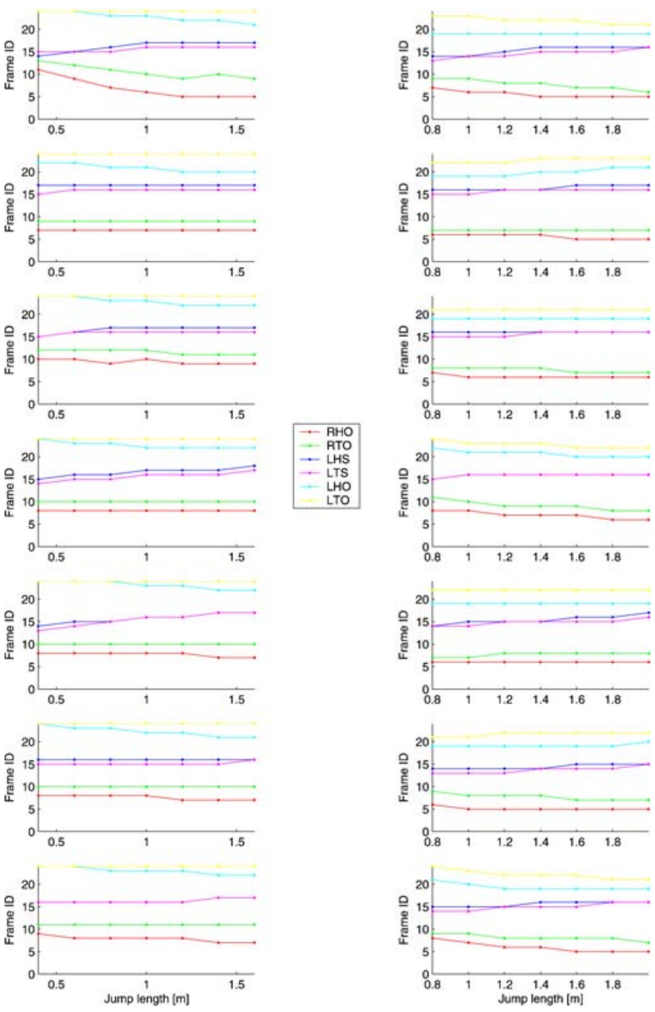

Fig. 7. Manual labeling of the foot events, for the seven captured subjects performing various jump lengths. *Left:* walking jumps with a length variation from 0.4 to 1.6 [m]. *Right:* running jumps with a length variation from 0.8 to 2.0 [m]

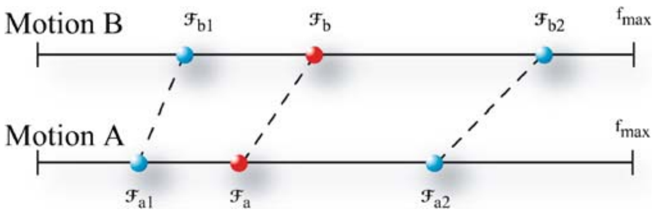

Fig. 8. Frame alignment between compatible support phases of two motions. Motion A and motion B support phases are defined from frame $\mathcal{F}_{a1}$ to $\mathcal{F}_{a2}$ from frame $\mathcal{F}_{b1}$ to $\mathcal{F}_{b2}$, respectively

To find the corresponding time-aligned frame f_b in the motion $\boldsymbol{B}$, we define a blending weight parameter p, which varies from 0 at the blending start, to 1 at the blending end. This parameter indicates the normalized progression of frames in both motions and is computed as follows:

$$p = \frac{\mathcal{F}_a - \mathcal{F}_{a1}}{\mathcal{F}_{a2} - \mathcal{F}_{a1}} = \frac{\mathcal{F}_b - \mathcal{F}_{b1}}{\mathcal{F}_{b2} - \mathcal{F}_{b1}} . \qquad (7)$$

From this latter equation, $\mathcal{F}_b$ is determined and then interpolated with $\mathcal{F}_a$ to compute the blended frame $\mathcal{F}$. As the frames hold a posture expressed with quaternions, each joint's rotation of $\mathcal{F}$ is computed by the spherical interpolation method [36] using the parameter p. The root global position is similarly computed by a linear interpolation between the jump's and the locomotion's root position. For this latter, as the pattern is originally generated on a treadmill, we add the displacement corresponding to the current speed. Additionally, the frequency F is computed by performing a linear interpolation from the frequencies F_a to F_b, parameterized with the weight p.

5 Motion planning for dynamic obstacles

The blending technique presented in the previous section is included in our method that controls a character for handling obstacles dynamically. When an obstacle "appears" in the scene, our method decides on a trajectory according to two solutions. If the obstacle is to be got around, two way-points are created in order to construct the trajectory. Otherwise, the obstacle is jumped over and the trajectory is constructed so as to place the take-off foot as close as possible to the obstacle.

An obstacle *obs* is defined by a flat box (e.g. a puddle of water) with a center position $\boldsymbol{C}_{obs}$, length l_{obs} and width w_{obs}. We assume that the obstacle always faces a character perpendicularly (see Fig. 9) and has a null height. Let t_1 be the time when an obstacle is dynamically created in front of a virtual human. Its current position is $\boldsymbol{p}_r(t_1)$ with a $\boldsymbol{h}(t_1)$ normalized heading direction. The distance $\boldsymbol{d}$ from the character position to the front of the obstacle is computed as follows:

$$\boldsymbol{d} = \boldsymbol{C}_{obs} - \left(\frac{l_{obs}}{2}\boldsymbol{h}(t_1)\right) - \boldsymbol{p}_r(t_1) . \qquad (8)$$

The position just in front of the obstacle is defined as $\boldsymbol{P}_{start} = \boldsymbol{p}_r(t_1) + \boldsymbol{d}$. In addition, at time t_1 the character's motion is defined with a linear speed s_1, an angular speed $\omega_1 = 0$ and a locomotion weight w_{loco_1}.

With this environment configuration in mind, the choice to clear an obstacle or not is determined by performing different tests. The first one checks if the obstacle length l_{obs} is included in the jump length range that a human is able to perform. Two ranges are defined, from $l_{w_{\min}}$ to $l_{w_{\max}}$ and from $l_{r_{\min}}$ to $l_{r_{\max}}$, for walking and running jumps, respectively. By default, the current locomotion weight w_{loco_1} determines the type of jump. However, a requested length between $l_{w_{\max}}$ and $l_{r_{\max}}$ compels a walking motion to be smoothly modified towards a running one.

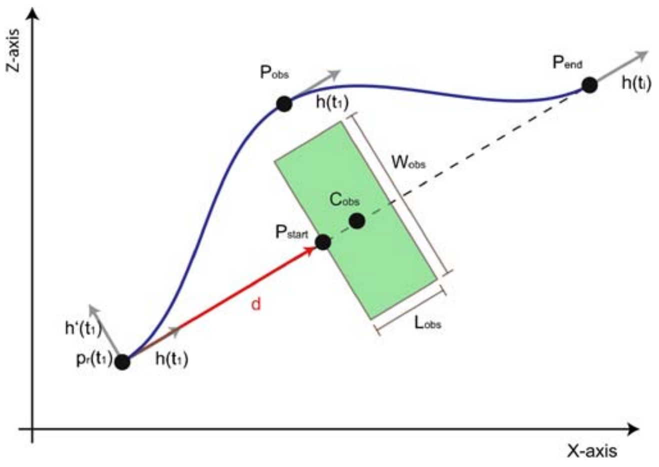

Fig. 9. Top-view of the environment configuration at time t_1 when an obstacle is generated dynamically. According to the decision planning, either the red or the blue path is determined

Assuming that a given jump length l_j lies between the minimal and maximal boundaries, we compute the required speed s_2 to execute this jump. According to its type and length (i.e. the obstacle length l_{obs}), l_j is inserted into either Eq. 1 or Eq. 2 to determine the corresponding s_2. In the case where $s_2 < s_1$, we avoid decelerating before executing the jump by setting $s_2 := s_1$. As a consequence, the jump length l_j is adjusted by inserting s_2 either into Eq. 1 or Eq. 2.

The next test evaluates the mean acceleration $\bar{a}$ computed between the time t_1 and the jump start, as described in Eq. 9, where $\bar{s}$ represents the mean between the s_1 and s_2 speeds. To avoid too abrupt speed modifications, an acceleration threshold ε_a is fixed that corresponds to the maximal acceleration a non-sportsman can achieve. Beyond this value, the obstacle has to be got around.

$$\bar{a} = \frac{s_2 - s_1}{\frac{d}{\bar{s}}} \leq \varepsilon_a \,. \qquad (9)$$

The presented rules allow an autonomous character to decide either to get around the obstacle or to jump over it. In both cases, the motion planning method determines a trajectory that the character has to follow. In the next subsections, we present the methodology for constructing the trajectory for getting around and for jumping over the obstacle.

5.1 Getting around the obstacle

When an obstacle cannot be jumped over, the character has to get around it by passing either by its right or left side. We choose arbitrarily the left side and we define a point $\boldsymbol{P}_{obs}$ to go through, depicted in Fig. 9 and computed as described in Eq. 10, where $\boldsymbol{h}'(t_1)$ is orthogonal to $\boldsymbol{h}(t_1)$ (see Fig. 9).

$$\boldsymbol{P}_{obs} = \boldsymbol{C}_{obs} + \boldsymbol{h}'(t_1)\left(\frac{w_{obs}}{2} + \delta l_{obs}\right) . \qquad (10)$$

The underlying idea is to move this point away from the obstacle regarding its length, in order to avoid a possible collision. We therefore fix a value $\delta \in [0 \ldots 1]$ in order to determine the influence of the obstacle length. In addition, the character's original trajectory can be joined back by defining a second point $\boldsymbol{P}_{end}$ to go through, by a symmetry of the point $\boldsymbol{p}_r(t_1)$ according to the axis traversing the obstacle, perpendicular to $\boldsymbol{d}$.

The path that avoids the obstacle is then determined by those two points P_{obs} and P_{end}, with their directions similar to $\boldsymbol{h}(t_1)$. Bezier curves can be applied to construct a smooth path [31]. It must then be transformed into a trajectory respecting velocity and acceleration constraints, a classical problem in robotics [22]. However, we prefer to adopt the steering model proposed in [5]. This latter allows one to go through a desired position (like [33]), with a desired heading direction at this position. This steering method ensures the reach of the target, thanks to a process that anticipates either the success or failure of the reach, and adjusts the desired speed accordingly. In addition, precalculated data greatly reduces the computational cost, which is appreciable for our real-time constraint.

Hence, the getting around the obstacle is performed in two stages. The first one is initialized by setting the target position $\boldsymbol{P}_{obs}$ and direction $\boldsymbol{h}(t_1)$ to the steering model. While the target is not reached, the model updates at each animation time step the character's linear and angular speed, according to the character's previous position as well as its linear speed and locomotion phase. As soon as the first target is reached, the second phase can be started if necessary by setting the new target position $\boldsymbol{P}_{end}$ and direction $\boldsymbol{h}(t_1)$.

5.2 Run-up footprint computation

When the jump is feasible, its run-up phase should be generated so that the right take-off foot is placed as close to the obstacle as possible and so that the locomotion speed corresponds to the required jump length. To solve this problem, the algorithms presented in [9] may be applied. It consists in generating randomly a series of footprints between the start and the end of the run-up motion, and then in adapting an appropriate motion capture clip. Another method that provides less precise results looks for a suitable motion capture clip into an enhanced motion graph structure [23]. However, as these methods are only interactive and do not ensure the speed constraint, we propose an alternate approach. This allows the on-the-fly motion sequence generation that corresponds to the obstacle constraints.

The fundamental idea of our method is to compute a footprint combination matching the following constraint formulation: "position the right take-off foot at $\boldsymbol{P}_{start}$ with a locomotion speed s_2". In other words, we have to determine a speed variation function (or speed profile) $S(t)$, defined from the speed s_1 at time t_1, i.e. at the obstacle generation, to the speed s_2 at time t_2, when the character is located at $\boldsymbol{P}_{start}$ (note that $s_1 \leq s_2$). The locomotion phase φ (mod 1) is used to ensure the correct take-off foot position. Let us recall that $\varphi = 0$ indicates the start of the locomotion cycle and coincides with the RHS event.

To begin with and for simplification, our method considers that the locomotion type is identical during the run-up phase. The speed variation model $S(t)$ that we introduce is based on a quadratic function:

$$S(t) = at^2 + bt + c\,. \tag{11}$$

The three parameters a, b and c determine the shape of the function in order to match the goal position, final speed and locomotion phase conditions (Fig. 10). In addition to these unknowns, the variation duration that corresponds to the run-up duration $t_{runup} = t_2 - t_1$ has to be computed. We determine four equations: the first two match the initial and final conditions of the speed profile, assuming that $t_1 = 0$:

$$\begin{aligned} S(0) &= c = s_1\,, \\ S(t_2) &= a\,t_2^2 + bt_2 + c = s_2\,, \end{aligned} \tag{12}$$

Then the traveled distance $d = ||\boldsymbol{d}||$ between $\boldsymbol{p}_r(t_1)$ and $\boldsymbol{P}_{start}$ is expressed as a function of time and speed:

$$d = \int_0^{t_2} (at^2 + bt + c)\,\mathrm{d}t = \frac{1}{3}at^3 + \frac{1}{2}bt^2 + ct\,. \tag{13}$$

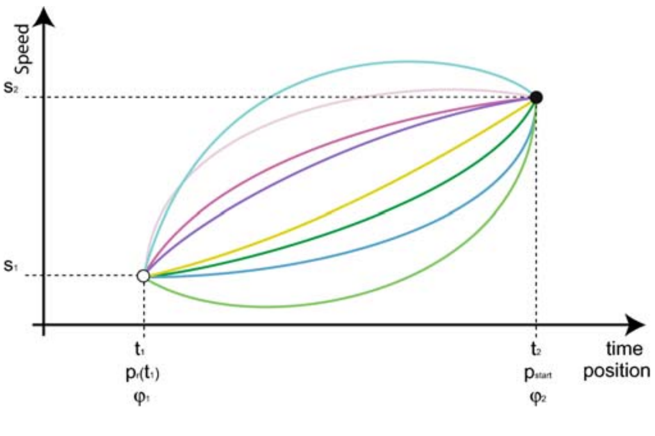

Fig. 10. Quadratic speed variation from s_1 at position $\mathbf{p}_r(t_1)$ to s_2 at $\mathbf{P}_{start}$. Over the curve family, one has to be chosen to match optimally the right take-off foot constraint at position $\mathbf{P}_{start}$

Finally, the last equation expresses the constraint for the locomotion phase φ_2:

$$\begin{aligned} \varphi_2 &= \varphi_1 + \Delta\varphi \\ &= \varphi_1 + \left[\sum_{i=1}^{n} F(S(\frac{i}{n}t_2))\frac{t_2}{n}\right] \\ &= \varphi_1 + \int_0^{t_2} F(S(t))\,\mathrm{d}t\,. \end{aligned} \tag{14}$$

As the frequency function F is only dependent on the speed s, it can be defined as described in [11]:

$$F(s) = gs^h\,, \tag{15}$$

where g and h are constant parameters. Hence, the integral in Eq. 14 is:

$$\int_0^{t_2} F(S(t))\,\mathrm{d}t = \int_0^{t} g(at^2 + bt + c)^h\,\mathrm{d}t\,. \tag{16}$$

However, the solution of this integral uses hypergeometric functions and its analytic formulation is therefore very complex to compute and inappropriate for a real-time application. In addition, this integral does not consider the case where the locomotion weight has to vary from 0 to 1, inducing the modification of the frequency function F. To solve this problem, the frequency function is expressed as a linear interpolation between the two frequency functions F_1 and F_2 associated to locomotion weight at time t_1 and t_2, respectively. Hence, we approximate Eq. 14 with Eq. 17 by assuming that the frequency function is linear between $S(0)$ and $S(t_2)$.

$$\begin{aligned} \varphi_2 &= \varphi_1 + \int_0^{t_2} F(S(t))\,\mathrm{d}t \\ &\approx \varphi_1 + \frac{F_1\left(S\left(t_2\right)\right) + F_2\left(S(t_2)\right)}{2}t_2\,. \end{aligned} \tag{17}$$

Finally, the four unknowns a, b, c and t_2 are computed by solving the system composed of the four described equations (Eq. 12, Eq. 13 and Eq. 17).

However, the value of φ_2 that actually corresponds to the performed cycle number before the jump remains to be determined. For given parameters s_1, s_2, φ_1 and d, a too big φ_2 involves a negative speed during a time interval (Fig. 11, left). In practice, this means that the number of cycles is so important that the character has to slow down, and even walk backwards when the speed is negative, which is not the intention of the method. Conversely, a too small φ_2 (i.e. a small number of cycles) entails an abrupt speed variation not physically feasible by the character

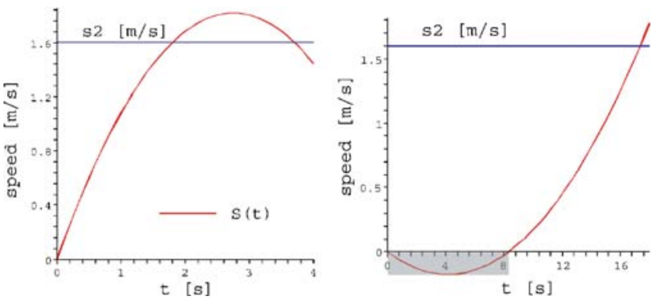

Fig. 11. *Left:* φ_2 is too big and violates the condition $S(t) \geq 0$. *Right:* φ_2 is too small and entails a too strong acceleration

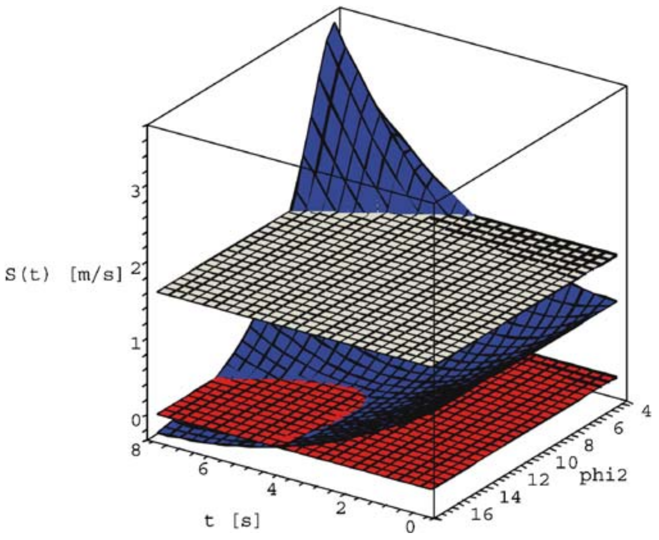

Fig. 12. The function $S(t)$ (in blue) for different φ_2, given $s_1 = 1$, $s_2 = 1.6$, $\varphi_1 = 0.1$ and $d = 5$. The red plane corresponds to $S(t) = 0$ and helps to visualize when $S(t) < 0$. The white plane represents the final speed ($S(t) = s_2$)

(Fig. 11, right). Figure 12 illustrates the speed variation $S(t)$ (blue surface) for a continuous φ_2 variation. Observe the negative speed variation for some φ_2 values.

Hence, we determine φ_2 that generates the variation closest to the linear one. Actually, this linear variation entails the smallest acceleration difference, which has to be below the acceleration threshold ε_a defined in Eq. 9. The algorithm is composed of two phases.

Firstly, φ_2 is initialized by assuming that the speed variation is linear. Using the approximation of Eq. 16, the final locomotion phase is rounded as follows:

$$\varphi_2 = \left\lfloor \varphi_1 + \frac{F(s_1) + F(s_2)}{s_1 + s_2} d + 0.5 \right\rfloor . \tag{18}$$

Secondly, this approximated phase value and the given input parameters s_1, s_2, φ_1 and d allow the unknowns a, b, c and t_2 to be computed in order to generate the function $S(t)$. From its derivative

$$\frac{dS(t)}{dt} = A(t) = 2at + b \tag{19}$$

we compute the slope $m = 2a$.

A negative m implies that either the extremum of $S(t)$ is a maximum inside $[0 \ldots t_2]$ or that the speed always decreases from s_1 to s_2, as illustrated in Fig. 13. In both cases, the target phase φ_2 may be increased in order to reduce the acceleration variation. Hence, this phase is iteratively incremented by 1. At each i-th iteration, the new parameters a_i, b_i, c_i and t_{2i} are used to compute the acceleration difference Δa_i defined as follows:

$$\Delta a_i = |A(t_{2i}) - A(0)| = |2a_i t_{2i}| . \tag{20}$$

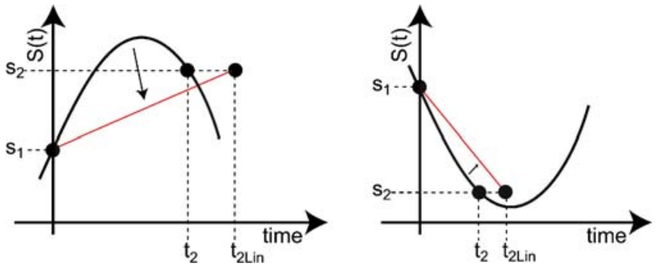

Fig. 13. Speed variation $S(t)$ (black curve) where the acceleration slope is negative. The arrow indicates that by increasing φ_2, $S(t)$ approaches the red line illustrating the linear speed variation, from $t = 0$ to $t = t_{2Lin}$. *Left:* the maximum is between $t = 0$ and $t = t_2$. *Right:* the extremum is outside $[0 \ldots t_2]$

The algorithm stops when $\Delta a_i \geq \Delta a_{i-1}$ or when the speed variation provides negative speed values, which is easily tested using the extremum of $S(t)$.

On the contrary, a positive m indicates that φ_2 may be decreased (Fig. 14). The algorithm proceeds iteratively by decrementing the phase, and stops similarly to the case where m is negative. In addition, it is ensured that the final $\varphi_2 \geq 1$.

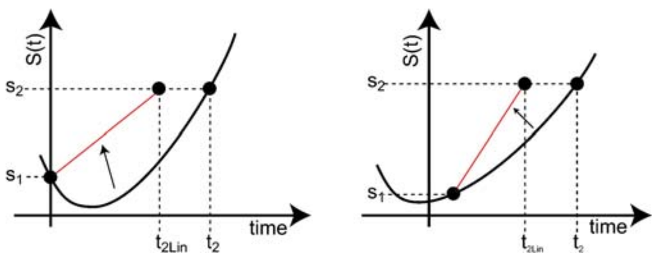

Fig. 14. Speed variation $S(t)$ (black curve) where the acceleration slope is positive. The arrow indicates that by decreasing φ_2, $S(t)$ approaches the red line illustrating the linear speed variation, from $t = 0$ until $t = t_{2Lin}$. *Left:* the minimum is between $t = 0$ and $t = t_2$. *Right:* the extremum is outside $[0 \ldots t_2]$

The motion planning is therefore completed first by determining either to get around or to jump over an obstacle and secondly by providing the adequate motion parameter variations to follow the path in the environment.

6 Results

We have integrated our state machine model into a 3D application where an animator can steer a virtual human by changing the speed, locomotion weight and the personification vector. We present the results of our blending method and our motion planning technique separately.

6.1 Transition between locomotion and jump (without obstacle)

A jump can be parameterized with a desired jump length, a personification vector and jump type. At any time, the animator can trigger a jump. The blending model adapts on-the-fly the current motion automatically to preserve a speed coherence based on observations made on 105 walking and 103 running captured jumps. Therefore, the speed and locomotion weight is smoothly modified during a specific period of time (Fig. 15, bottom). After some experiments, we choose ε_a as an acceleration value, normalized by the leg length of the character, of 0.8 $[s^{-2}]$.

Additionally, the method decides when and how long to perform the blending operation, by taking into account the motion support phases. The generated transition is seamless and performed in real-time, with a duration time varying from 0.2 to 0.5 seconds.

The top image of Fig. 15 shows a transition from walking to a running jump, using a traditional blending method such as [29]. One can observe that the jump length is large regarding the step length, leading to a dynamic incoherence. The bottom image shows the same situation by applying our technique. The motion is adapted by modifying the speed and locomotion weight according to the requested jump. For a better perception of the improvements proposed by our method, the reader can also view the movies attached to this paper.

Some artifacts may nevertheless occur due to the approximation used to detect the feet constraints. Therefore, like many other blending methods, we add IK control on the feet to prevent sliding effects, by extending the prioritized IK described in [4, 24] to the automatic generation of constraints in a dynamically evolving context [13]. This correction is efficient enough to not alter the real-time performance of the animation.

6.2 Motion planning for obstacles

In our developed real-time application, a jump can be generated at any time by the animator. The jump dimension and its distance to the current character's position are generated randomly over a predefined range of values. The distance is selected between 2 and 10 [m], the obstacle length between 0.4 and 2.6 [m], the and width between 1 and 6 [m]. The obstacle height is constant and set to 0.1 [m].

We present the results when the character gets around the obstacle. Figure 16 illustrates the situation where the obstacle cannot be jumped over. In fact, the final run-up speed necessary for this jump length entails a too important acceleration starting from the initial character locomotion speed. The path goes through the first way-point placed on the obstacle side and the second one placed in order to recover the path followed if the obstacle were jumped over. In the resulting generated animation, we notice that the linear speed is significantly decreased near to the two way-points. This is due to the steering method that we used. In fact, this method is highly dependent on a database of pre-computed trajectories. At any time, for a given locomotion parameter situation, the method searches for a correspondence in this database. If it fails, the current parameters are modified in order to find a correspondence. In our case, the speed is decreased to match a trajectory in the database [5].

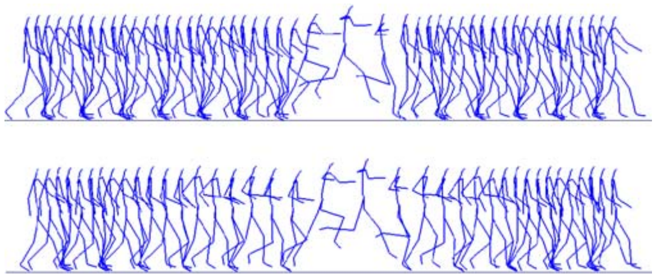
Fig. 15. Comparison between two transition methods from a walking motion with running jump. *Top:* without motion adaptation. *Bottom:* with locomotion speed and type adaptation

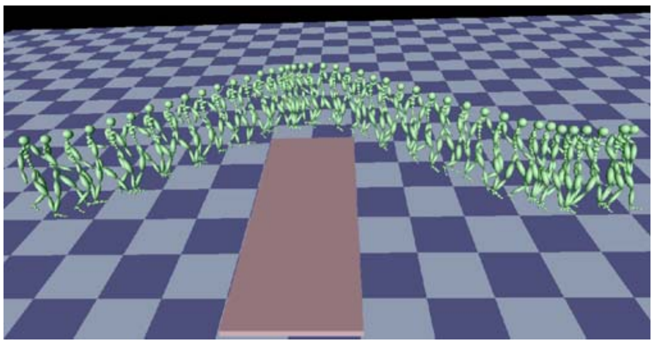
Fig. 16. Motion sequence when the character gets around the obstacle

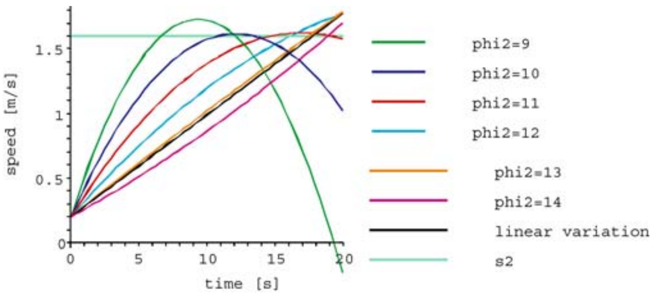

Fig. 17. Increase of the run-up cycle number. Starting from $\varphi_2 = 13$, $\varphi_2 = 14$ is selected among different speed variations $S(t)$ for a given $\varphi_1 = 0.3$, $s_1 = 0.2$ [m/s], $s_2 = 1.6$ [m/s] and $d = 16$ [m]

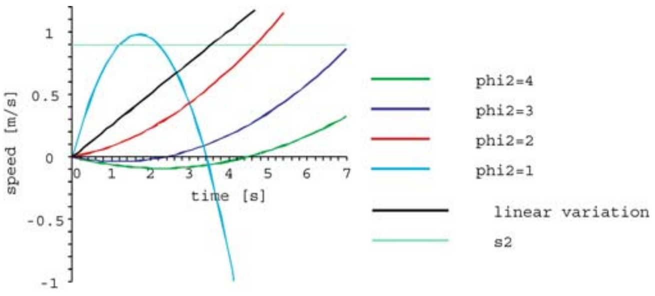

Fig. 18. Decrease of the run-up cycle number. Starting from $\varphi_2 = 4$, $\varphi_2 = 1$ is selected among different speed variations $S(t)$ for a given $\varphi_1 = 0.1$, $s_1 = 0$ [m/s], $s_2 = 0.9$ [m/s] and $d = 1.6$ [m]

In situations where the character jumps over the obstacle, we illustrate two speed variation results, explaining the optimal φ_2 computation described in Sect. 5.2. In the first example (Fig. 17), the approximated $\varphi_2 = 13$ returns a negative slope m. The phase is augmented until reaching the variation closest to the linear one. As the curve defined with $\varphi_2 = 14$ leads to a Δa greater than the previous one, $\varphi_2 = 13$ is selected. For pedagogical purposes, the graph in Fig. 17 depicts curves from $\varphi_2 = 9$ to $\varphi_2 = 14$.

The second example (Fig. 18) illustrates the situation where the phase φ_2 is iteratively decremented until the optimal value $\varphi_2 = 1$. The corresponding speed variation shows two t_2 solutions to reach s_2. However, the solution is unique ($t_2 = 2.09$) due to the φ_2 constraint. One can observe that the variation with $\varphi_2 = 2$ provides in absolute a less accelerated solution. However, our method is based on the relative acceleration difference between the start and end of the variation. This is the reason why $\varphi_2 = 1$ is preferred, as this solution offers less difference, despite that the absolute accelerations at $t = 0$ and $t = t_2$ are high.

In Figs. 19, 20 and 21 we illustrate scenarios where the current locomotion speed and type are modified on-the-fly in order to clear the obstacle. According to the obstacle dimension, the performed jump needs a fast final run-up speed, executed by running. Our method computes the

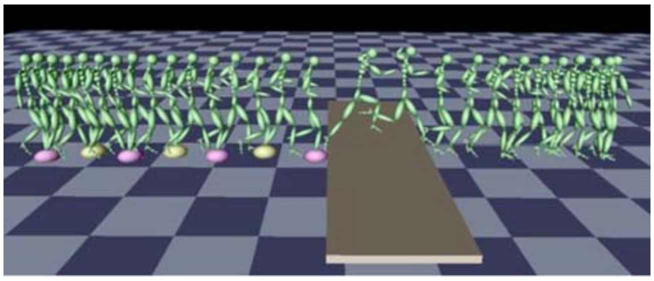

Fig. 19. Motion sequence when the obstacle is jumped over. From slow walking to fast running, the method computes the footprint positions (right foot in purple, left in yellow) in order to place the right take-off foot as close as possible to the obstacle

footprint positions (i.e. the speed variation) so as to place the right take-off foot as close as possible to the obstacle, with the required speed and type of locomotion. After jumping over the obstacle, the motion parameters are restored to the initial ones.

Our model constrains the current locomotion phase to be null (RHS event) when the 3D root position $\boldsymbol{p}_r$ is in front of the obstacle. However, the right foot at this phase is forward positioned regarding $\boldsymbol{p}_r$, as the legs are apart in order to perform the next step. We therefore reduce the distance d by the resulting step length at speed s_2. In that way, the right foot is exactly placed in front of the obstacle.

Thanks to the performance of our method (it takes about 4μ sec on average to compute the speed variation $S(t)$), we are able, not only to generate the required jump, but to compute the speed variation at any time during the animation. It is, therefore, possible to animate an autonomous character that handles obstacles appearing dynamically, as illustrated in the videos attached to this paper.

We have also evaluated the quality of our method. As our original motion capture jump sequences do not contain the run-up phase, we propose another evaluation method. We ran our method in a dynamic environment, where obstacles are generated on-the-fly by randomly selecting their dimensions and their positions to the moving character. The error E is computed by subtracting from the start obstacle's position $\boldsymbol{C}_{obs} - \frac{1}{2}\boldsymbol{l}_{obs}$ the effective position of the right toe $\boldsymbol{P}_{toe}$ at the jump beginning. Hence, a quantitative measurement of the method precision is performed. The histograms in Figs. 22 and 23 illustrate the experimental results, by counting the jump number with respect to the precision of the take-off foot position. The error is computed for 353 obstacles jumped over with a constant initial walking speed $s_1 = 1.1$ [m/s] for 334 obstacles with a running speed of $s_1 = 1.5$ [m/s]. The foot is correctly placed within an error of 5 [cm] (for a character's leg length of 0.88 [cm]) in more than 70% of the jumps. In addition, we try to quantify the influence of the frequency function approximation (Eq. 17) on E regarding

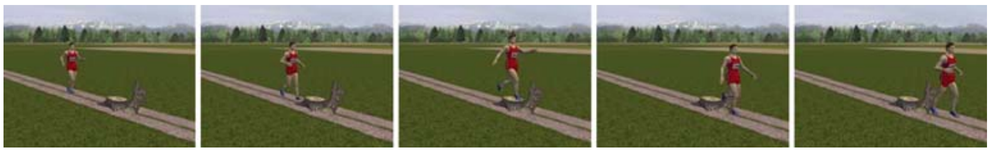

Fig. 20. Sequence of motion when the obstacle is jumped over

Fig. 21. Different virtual characters performing jumps differing in length and type

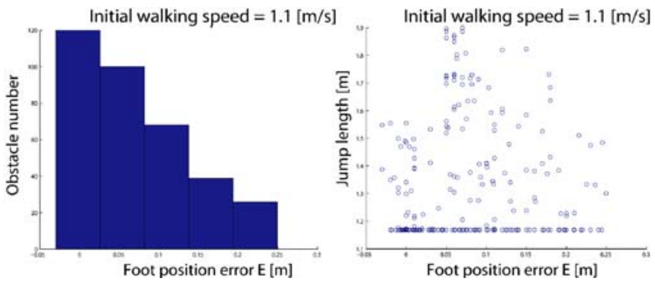

Fig. 22. Results of 353 obstacles jumped over with an initial walking speed $s_1 = 1.1$ [m/s] *Left:* histogram representing the distribution of the measured error E. *Right:* comparison between the measured error E and the jump length to jump over an obstacle

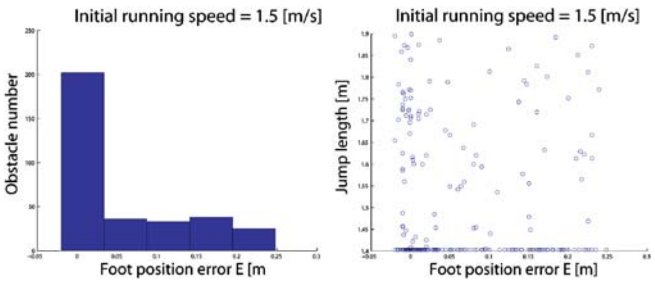

Fig. 23. Results of 334 obstacles jumped over with an initial running speed $s_1 = 1.5$ [m/s] *Left:* histogram representing the distribution of the measured error E. *Right:* comparison between the measured error E and the jump length to jump over an obstacle

the speed difference between s_1 and s_2. At a first thought, we can intuitively induce that the longer the jump, the bigger the error. The graphs in Figs. 22 (right) and 23 (right) depict each jump (represented by a circle) according to its length and take-off foot position error. We conclude that E is not directly influenced by big length jumps (i.e. big final speed s_2).

7 Conclusion

Compared to earlier approaches [2, 20, 25] to finding the possible transition time over a large motion capture database, our motion blending method presents advantages. First, we do not need to perform a pre-processing step involving all the clips of the database. This allows us to generate dynamically parameterized motions. Then our methodology determines the transition time and duration automatically, in contrast to [20] where transition thresholds have to be specified by hand. Finally, we adapt the current motion by modifying its parameters to enforce the requested blending, unlike previous methods that either do not allow the transition or perform it without adaptation.

In addition, we have presented a new motion planning model for obstacle handling in dynamic environments. When an obstacle of a specific length suddenly appears in front of a moving autonomous character, the method chooses first whether to go around or jump over the obstacle. For the first choice, the linear and angular speeds vary to generate a path that avoids the obstacle. For the second choice, a quadratic speed variation is computed. This latter ensures that the right run-up foot is placed just before the obstacle and that the character has an appropriate speed allowing the required jump length to be performed. The method is fully dynamic as the motions are generated on-the-fly, without knowledge about the environment topology. In addition, the method performance (4μ sec) allows one to maintain real-time capability of the animation.

Compared to similar previous work [23], our method is based on a continuous parameter animation. The motion result precision is, therefore, not dependent on the database size as the exact desired motions are always available thanks to our animation engines (locomotion and jump). Furthermore, our method does not need to know in advance the obstacle location over time. In the case of moving obstacles, we can iteratively apply our motion planning method to correct the speed variation.

The steering method used to get around the obstacle decreases the linear speed near the way-points. This problem can be solved by placing two way-points near the obstacle side to subdivide the problem complexity. Another idea is to place two way-points to recover the original path, one with the position and the other for the direction.

Our speed variation method is adapted also for other scenarios. Imagine that the character has to jump over a low wall from a standing position (without run-up). The speed variation is computed by setting to zero the speed to be reached at the obstacle beginning. Another interesting problem solved by our method concerns the locomotion combined with stair climbing. It is important to be sure that either the right or the left foot is placed just in front of the first stair. In this case, the method computes a speed variation ensuring that the final phase equals either 0 (right foot) or 0.5 (left foot). However, this final phase can take any other value according to the desired task.

During the run-up phase, the model estimates the locomotion frequency variation by a linear function. This approximation allows a drastic computational cost reduction without decreasing the result quality. Actually, even when the speed variation is important, the foot is placed within a tolerance of 10 cm. We can nevertheless ensure that the foot never touches the obstacle by reducing the distance to the obstacle and increasing the length of the jump. The precision of the take-off foot position can be improved by re-computing our speed variation during the run-phase, at a specific sampling rate.

The choice of a quadratic variation model is motivated by its efficiency and flexibility, allowing one to handle special cases such as identical initial and final speeds. However, our method induces an acceleration discontinuity, even though we limit the acceleration difference during the variation. Other polynomial variation models of higher degree can be elaborated, in order to add a constraint for ensuring acceleration continuity. Still, our opinion is that those models increase the computational costs and do not guarantee a significant improvement in the animation believability. Actually, we think that the human eye is not extremely sensitive to acceleration discontinuities as our results already provide pleasant animations.

Finally, we do not consider the obstacle height. However, our method can be easily adapted for such situation. At the strategic planning layer level, the parameter height may lead to other decisions like a different final run-up speed or a stop in front of the obstacle. In this latter case, our method provides a speed variation given the final speed equal to zero. Nevertheless, our method can hardly be applied to compute the run-up speed variation for obstacles having a considerable height. In this case, the dynamics of the motion has to be taken into account in order to compute the speed variation.

Acknowledgement The authors would like to thank Rodrigue Oeuvray for his helpful advice and fruitful discussions. The present research was funded by the Swiss National Science Foundation. The Maya licenses were granted by Alias through their research donation program.

References

1. Abe, Y., Liu, C., Popović, Z.: Momentum-based parameterization of dynamic character motion. In: Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation (2004)
2. Arikan, O., Forsyth, D.: Interactive motion generation from examples. In: Proceedings of ACM SIGGRAPH, Annual Conference Series (2002)
3. Ashraf, G., Wong, K.: Generating consistent motion transition via decoupled framespace interpolation. Comput. Graph. Forum **19**(3), 447–456 (2000)
4. Baerlocher, P., Boulic, R.: An inverse kinematic architecture enforcing an arbitrary number of strict priority levels. Visual Comput. **20**(6), 402–417 (2004)
5. Boulic, R.: Proactive steering toward oriented targets. In: Proceedings of Eurographics, short presentation (2005)
6. Brock, O., Khatib, O.: Elastic strips: a framework for motion generation in human environments. Int. J. Robot. Res. **21**(12), 1031–1052 (2002)
7. Bruderlin, A., Williams, L.: Motion signal processing. In: Proceedings of ACM SIGGRAPH, Annual Conference Series, pp. 97–104 (1995)
8. Chestnutt, J., Lau, M., Cheung, G., Kuffner, J., Hodgins, J., Kanade, T.: Footstep planning for the Honda ASIMO humanoid. In: Proceedings of IEEE International Conference on Robotics and Automation (2005)
9. Choi, M., Lee, J., Shin, S.: Planning biped locomotion using motion capture data and probabilistic roadmaps. ACM Trans. on Graph. (2003)
10. Egges, A., Molet, T., Magnenat-Thalmann, N.: Personalised real-time idle motion synthesis. In: Proceedings of Pacific Graphics, pp. 121–130. Seoul, Korea (2004)
11. Glardon, P., Boulic, R., Thalmann, D.: A coherent locomotion engine extrapolating beyond experimental data. In: Proceedings of Computer Animation and Social Agent, pp. 73–83 (2004)
12. Glardon, P., Boulic, R., Thalmann, D.: On-line adapted transition between locomotion and jump. In: Proceedings of Computer Graphics International, pp. 44–49 (2005)
13. Glardon, P., Boulic, R., Thalmann, D.: Robust on-line adaptive footplant detection and enforcement for locomotion. Visual Comput. (2006, to appear)
14. Go, J., Vu, T., Kuffner, J.: Autonomous behaviors for interactive vehicle animations. In: Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation (2004)
15. Guo, S., Robergé, J.: A high-level control mechanism for human locomotion based on parametric frame space interpolation. In: Proceedings of Eurographics Workshop on Computer Animation and Simulation, pp. 95–107 (1996)
16. Hsu, D., Kindel, R., Latombe, J., Rock, S.: Randomized kinodynamic motion planning with moving. Int. J. Robot. Res. **21**(3), 233–255 (2002)
17. Kavraki, L., Svestka, P., Latombe, J.C., Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Trans. Robot. Auto. **12**(4), 566–580 (1996)
18. Khatib, O.: Real-time obstacle avoidance for manipulators and mobile robots. Int. J. Robot. Res. **5**(1), 90–98 (1986)
19. Kovar, L., Gleicher, M.: Flexible automatic motion blending with registration curves. In: Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 214–224 (2003)
20. Kovar, L., Gleicher, M., Pighin, F.: Motion graphs. In: Proceedings of ACM SIGGRAPH, Annual Conference Series, pp. 473–482 (2002)
21. Kuffner, J.: Goal-directed navigation for animated characters using real-time path planning and control. In: Proceedings of Workshop on Modelling and Motion Capture Techniques for Virtual Environments, CAPTECH'98, LNAI 1537, pp. 171–186. Springer, Berlin, Heidelberg (1998)
22. Lamiraux, F., Laumond, J.P.: From paths to trajectories for multi-body mobile robots. In: Proceedings of International Symposium on Experimental Robotics, LNCIS 250, pp. 237–245. Springer (1997)

23. Lau, M., Kuffner, J.: Behavior planning for character animation. In: Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation (2005)
24. Le Callennec, B., Boulic, R.: Interactive motion deformation with prioritized constraints. In: Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation (2004)
25. Lee, J., Chai, J., Reitsma, P., Hodgins, J., Pollard, N.: Interactive control of avatars animated with human motion data. In: Proceedings of ACM SIGGRAPH, Annual Conference Series (2002)
26. Menache, A.: Understanding Motion Capture for Computer Animation and Video Games. Academic Press, San Diego (2000)
27. Menardais, S., Kulpa, R., Arnaldi, B.: Synchronisation for dynamic blending of motions. In: Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation (2004)
28. Mukai, T., Kuriyama, S.: Geostatistical motion interpolation. In: Proceedings of ACM SIGGRAPH, Annual Conference Series, pp. 1062–1070 (2005)
29. Park, S., Shin, H., Shin, S.: On-line locomotion generation based on motion blending. In: Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation (2002)
30. Park, W., Chaffin, D., Martin, B.: Modifying motions for avoiding obstacles. SAE Trans. **110**(6), 2250–2256 (2002)
31. Pettré, J., Laumond, J.P., Siméon, T.: 3D collision avoidance for digital actors locomotion. In: Proceedings of International Conference on Intelligent Robots and Systems (2003)
32. Pettré, J., Laumond, J.P., Siméon, T.: A 2-stages locomotion planner for digital actors. In: Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation (2003)
33. Reynolds, C.: Steering behaviors for autonomous characters. In: Proceedings of Game Developers Conference, pp. 763–782. Miller Freeman Game Group, San Francisco (1999)
34. Rose, C., Cohen, M., Bodenheimer, B.: Verbs and adverbs: Multidimensional motion interpolation. IEEE Comput. Graph. Appl. **18**(5), 32–41 (1998)
35. Rose, C., Guenter, B., Bodenheimer, B., Cohen, M.: Efficient generation of motion transitions using spacetime constraints. In: Proceedings of ACM SIGGRAPH, Annual Conference Series, pp. 147–154 (1996)
36. Shoemake, K.: Animating rotation with quaternion curves. In: Proceedings of ACM SIGGRAPH, Annual Conference Series, pp. 245–254 (1985)
37. Unuma, M., Anjyo, K., Takeuchi, R.: Fourier principles for emotion-based human figure. In: Proceedings of ACM SIGGRAPH, Annual Conference Series, pp. 91–96 (1995)
38. Wang, J., Bodenheimer, B.: Computing the duration of motion transitions: An empirical approach. In: Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 335–344 (2004)
39. Wooten, W., Hodgins, J.: Simulating leaping, tumbling, landing and balancing humans. In: Proceedings of IEEE International Conference on Robotics and Automation (2000)

PASCAL GLARDON is a research assistant and received his PhD in the Virtual Reality Laboratory at the Swiss Federal Institute of Technology (EPFL). His research interests include the generation and the control of virtual character animations, as well as their interaction with virtual environments. He received his Master's degree in Computer Science in 2000 at the Swiss Federal Institute of Technology, Zurich (ETHZ). In 2001 he worked as a software engineer at a company specialized in the security document printing.

RONAN BOULIC is a Senior Researcher, Lecturer and PhD Director at the Swiss Federal Institute of Technology, Lausanne (EPFL).
He is working in the Virtual Reality Lab and his research interests include realistic motion synthesis for virtual humans and robot. He received the PhD degree in Computer Science in 1986 from the University of Rennes, France, at the INRIA-IRISA research institute. He received the Habilitation degree from the University of Grenoble, France, in March 1995. Ronan Boulic is co-author of 83 research papers, among which 21 have appeared in international peer-reviewed journals. He served as Paper Chair of the 2004 SIGGRAPH-Eurographics Symposium on Computer Animation. He is a senior member of IEEE and a member of ACM and Eurographics.

DANIEL THALMANN is Professor and Director of The Virtual Reality Lab (VRlab) at EPFL, Switzerland. He is a pioneer in research on virtual humans. His current research interests include real-time virtual humans in virtual reality, networked virtual environments, artificial life, and multimedia. Daniel Thalmann has been Professor at The University of Montreal and Visiting Professor/Researcher at CERN, University of Nebraska, University of Tokyo, and Institute of System Science in Singapore.
He is coeditor-in-chief of Computer Animation and Virtual Worlds (formerly Journal of Visualization and Computer Animation), and a member of the editorial board of the Visual Computer and four other journals. Daniel Thalmann has been a member of numerous program committees, the program chair of several conferences and the chair of the Computer Graphics International '93, Pacific Graphics '95, ACM VRST '97, and MMM '98 conferences. He was Program Cochair of IEEE VR 2000. He has also organized five courses at SIGGRAPH on human animation and crowd simulation.
Daniel Thalmann has published more than 400 papers on graphics, animation, and virtual reality. He is coeditor of 30 books, including the recent "Handbook of Virtual Humans", published by John Wiley and Sons, and coauthor of several books. He was also codirector of several computer-generated films with synthetic actors including a synthetic Marilyn shown on numerous TV channels all over the world.
He received his PhD in Computer Science in 1977 from the University of Geneva and an Honorary Doctorate (Honoris Causa) from University Paul-Sabatier in Toulouse, France, in 2003.