A Motion Capture based Planner for Virtual Characters Navigating in 3D Environments

Juan Carlos Arenas-Mena[1], Jean-Bernard Hayet[1], and Claudia Esteves[2]

[1]Centro de Investigación en Matemáticas, Guanajuato, Gto., Mexico

[2]Departamento de Matemáticas, Universidad de Guanajuato, Gto., Mexico

{jcarenas, jbhayet, cesteves}@cimat.mx

Abstract. In this work, a strategy to automatically generate eye-believable motions for a virtual character that navigates in a 3D environment is presented. The overall approach consists of four components as follows. (1) A state-of-the-art path planner that computes a collision-free reference path for the character's center of mass (COM). For this planner, a simplified model that bounds the character's geometry is proposed. (2) A segmentation algorithm that divides the path into behaviors. (3) A classifier that compares each behavior with the corresponding motion capture segments previously analyzed and stored in a database. (4) A whole-body motion generator that synthesizes the appropriate behavior determined by the classifier. The main contribution of this work is to produce a sampling-based global motion planner that generates different behaviors (in addition to locomotion) issued from environmental constraints. Several results of our algorithm in different environments are shown and its current limitations are discussed.

Keywords. I.3.7 computing methodologies, computer graphics, three-dimensional graphics and realism, motion planning, character animation, motion-capture classification.

Un planificador basado en capturas de movimiento para personajes virtuales desplazándose en ambientes 3D

Resumen. En este trabajo se presenta una estrategia para generar automáticamente movimientos visualmente creíbles para un personaje virtual que navega en un ambiente 3D. Esta estrategia consta de 4 componentes: (1) Un planificador de movimientos que calcula un camino sin colisiones para el centro de masa (COM) del personaje. Para esto, se propone un modelo simplificado que envuelve la geometría del personaje. (2) Un algoritmo de segmentación que divide el camino en comportamientos. (3) Un clasificador que compara cada comportamiento con segmentos de captura de movimiento para identificar el tipo de comportamiento correspondiente. (4) Un controlador local de movimientos para todas las articulaciones del personaje que genera los comportamientos determinados por el clasificador. La contribución principal de este trabajo es producir un planificador de movimientos global basado en muestreos que genera diferentes comportamientos (además de locomoción) a partir de las restricciones del ambiente. Se muestran algunos resultados de aplicar esta estrategia en varios ambientes de prueba de para el personaje virtual y se discuten las limitantes del trabajo.

Palabras clave: I.3.7 metodologías computacionales, gráficas por computadora, gráficas tridimensionales y realismo, planificación de movimientos, animación de personajes, clasificación de comportamientos.

1 Introduction

Generating eye-believable motions for virtual characters has gained an increasing interest from both Computer Graphics and Robotics communities. This interest is mainly due to such demanding applications as video games, simulation environments, product design, architectural design, education, etc. As an example, the video game community would greatly benefit from algorithms that would make the evil characters motions somewhat plausible, i.e., taking the shortest possible paths in order to avoid obstacles, while simultaneously maintaining a "natural" gait. Eye-believable human-like motion synthesis is particularly challenging for two main reasons:

1. As we humans are very familiar with looking at human motions, even the smallest artifacts on the computer generated motions may strike us as odd or unnatural.
2. The high dimensionality of the representation of a humanoid skeleton makes the specification of every configuration highly redundant relative to almost any task.

To deal with the first problem, a now standard solution is to use clips recorded from real actors and stitch or edit them to generate new motions. These clips are referred to as *motion capture (mocap) clips*. The second issue has been tackled from several angles; one among them is the use of simplified models of the characters, which can greatly reduce the dimensionality of the problem.

In this work we propose a motion planner that computes a collision-free, eye-believable trajectory for a character in a cluttered environment. The planner takes as inputs the initial and final positions and orientations of the character as well as a set of prerecorded motion capture clips of several different behaviors (walking, running, jumping, bending and crawling). These clips are used in two stages:

1. An *offline motion analysis stage*, where the captured examples are processed and stored in a compact and convenient structure, useful to compare with the movements generated by the planner.
2. An *online motion synthesis stage*, where new collision-free paths are generated using a reduced model of the character. These paths are then segmented into homogeneous parts and compared with the motion database to find the type of action best suited to follow the path.

The contribution of this work is two-fold:

- We provide an original simplified model that reduces the dimensionality of the virtual character skeleton while keeping enough degrees of freedom (DOFs) to plan non-trivial motions. We use it for both planning and motion classification.
- We propose a compact PCA-based structure to store motion-capture clips and adequately generate new motions from a planned path.

These contributions allow us to consider more behaviors in addition to locomotion such as jumping or crawling when needed to avoid obstacles. The generated combined behaviors are more complex than those generated from locomotion only.

The rest of this work is structured as follows. In Section 2 we review most relevant work related to our problem. In Section 3, an overview of our complete strategy is presented. Section 4 describes the models of a virtual character which we use for planning, motion classification and motion generation. Section 5 describes a method for building a compact motion capture database from the original motion capture clips. In Section 6, our algorithm for generating collision-free whole-body motions is described. Section 7 presents some of the results obtained using the proposed strategy, and finally, in Section 8, conclusions and future work are discussed.

2 Related Work

Among the methods proposed in the literature to synthesize human-like motions for character navigation using motion capture clips, one of the most popular has been the *Motion Graphs* (e.g., [2, 11]) which stores a set of captured clips and automatically constructs transitions between them when these transitions are pertinent. Clips and transitions are stored in a directed graph, the *motion graph*, which is searched when new animation sequences are needed. A graph representation has the advantage that it preserves the realism of the original clips and that new animation sequences can be synthesized only by performing graph searches. However, as the clips are only stitched together, it is not easy to obtain a fine control and the variability necessary to accurately follow a reference path. Hence, other works, mostly within the Computer Graphics community, have proposed controllers based on a combination of captured clips. Pettré *et al*. [15] propose a locomotion controller for the navigation of a virtual character where reference linear and angular velocities of the character's center of mass (CoM) are provided by the user as input to the controller. Within a previously constructed database of examples, the algorithm

looks for the three clips with the closest velocities to the input ones and interpolates them linearly to generate a new motion. In our work, we use Pettré's controller together with controllers for different behaviors that consider obstacle height (jumping or crawling). This makes our approach go further than this previous work by using, in addition to locomotion, other behaviors in a single planning scheme. In the context of motion planning, several planners for virtual characters navigating in cluttered environments have also been proposed in the literature [5, 6, 13, 16, 18]. Most of these planners take as input motion capture clips to generate eye-believable motions. These methods can be divided into single-query and multiple-query approaches, depending on which motion planning strategy is used. Within single-query methods, the authors of [13] propose a planner based on a finite-state machine, in which states are motion capture clips of the same type of behavior (running, walking, jumping, etc.) and edges are transitions between them. The environment is represented as a 2D height map annotated with types of motion which could be used near certain obstacles, and a Rapidly-Exploring Random Tree (RRT) [12] is created using the finite state machine as a control to compute a collision-free path in a given environment. Within multiple-query approaches, among which our work can be classified, two-stage methods have been proposed. In the first stage of these methods, a collision-free path is found for a reduced model of a character, and in the second stage, the path is followed using motion capture clips. The authors of [18] propose a multi-layered grid, with each layer consisting of a single posture of a character. These postures represent a characteristic configuration of a type of movement or behavior such as walking, jumping or crawling. A collision-free path is found in this grid by giving some postures throughout the path. The postures are interpolated and dynamically validated to obtain a collision-free path for the whole body of the character. In [5], the authors plan a feasible, collision-free path for the footprints of a character. These footprints are the nodes of a graph and are linked with motion capture clips. Retargeting methods are used to satisfy the constraints imposed by each footprint position and orientation. In [16], the authors propose a two-stage method to synthesize new motions to avoid obstacles using existing motion capture examples. First, they compute a collision-free path for a box bounding a character's geometry. The resulting path is converted to linear and angular velocity references and given as input to the locomotion controller presented in [8, 15]. The authors take a similar approach but use a functional decomposition of a character. This decomposition divides the model of the character in three groups according to their function: locomotion, manipulation and pose. The motion planning stage is performed only for the box bounding the locomotion DOFs (the lower part of the character's body). The manipulation DOFs are computed using an inverse kinematics algorithm appropriate for closed kinematic chains. In the final stage, residual collisions are eliminated using a local correction of the pose kinematic chain. Our work is a two-stage multiple-query method following the same idea as [8, 16].

Our contribution relative to these works is two-fold: we added more behaviors into the planner to handle different types of obstacles (by jumping or crawling), and we propose a more efficient reduced model for the character which allows to plan a wider range of initial collision-free paths. More recent works [6] have used sweeps of motion capture clips as deformable models which are fitted inside constrained environments. The original captures are deformed in an equivalent manner to obtain a collision-free motion following by going as near as possible as the original motion.

3 Our Approach

The goal of our approach is to obtain a collision-free eye-believable path from input motion capture clips. Fig.1 presents the overall approach. The algorithm can be divided in two stages: (1) a motion analysis stage and (2) a motion synthesis stage. The first step of the analysis stage is to use a reduced model of a character to reduce the dimensionality of the motion capture data. The reduced model is a mesh which deforms as the character moves (Section 4.1). The values at some of the vertices

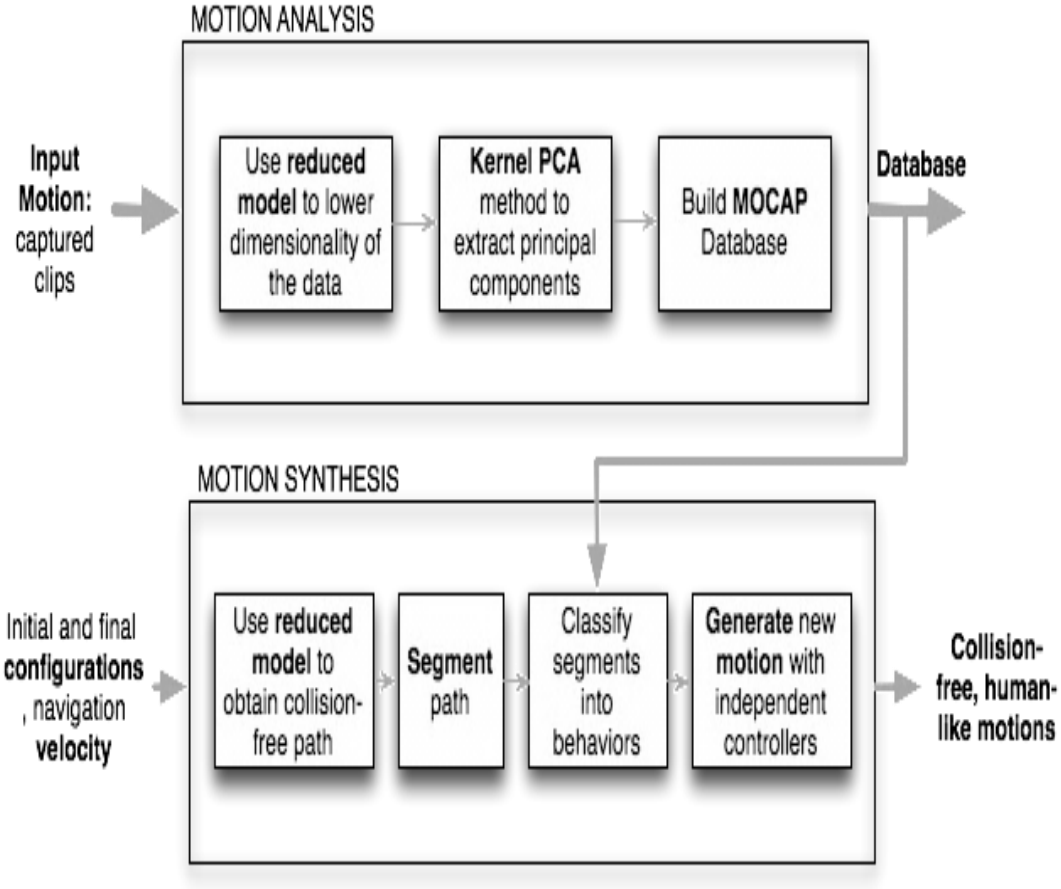


Fig. 1. Overview of our overall strategy

of this mesh are extracted and taken as the new values to be analyzed by our algorithm. The second step is to further reduce the dimensionality of the data by extracting its principal components (Section 5.1) and projecting the points of the mesh of each clip on these components producing clusters of behaviors on a new space (Section 5.3). The result of the analysis stage is thus a database of clustered behaviors in a low-dimensional space (two or three dimensions).

In the motion synthesis stage, we first produce a path using a reduced model of a character. This model is a set of five boxes attached to retracting joints making them shrink or expand on the vertical position (Section 4.2). This allows the simplified model to avoid obstacles below or above the default height of the character.

The second step is to segment the obtained path to identify the different behaviors needed to avoid the obstacles and to classify them using the database (Section 6.2) constructed during the motion analysis stage. Depending on the behavior determined by the classifier, a different motion controller is issued to generate locomotion, crawling or jumping motions to follow the computed path. This motion generation strategy is summarized in Algorithm 1 (Section 6.3).

In the following sections we further describe each step of our algorithm.

4 Character Model

A virtual character is usually represented as a series of linkages and joints rooted on the character's pelvis (see Fig.2 (a)). Every joint in this whole-body model is spherical, i.e., each joint can be completely specified by three angles if an Euler angles representation for orientations is chosen. Each degree of freedom (DOF) of every joint is bounded according to its anatomical limits. The limits to the knee and elbow joints are set to values such that only the degree of freedom allowing extension and flexion is allowed to move. The root is a free floating object in the 3D space, the configuration of which can be described using three angles to represent its rotation and three scalars to represent its translation. This model has usually around 57 or more DOFs for a realistic virtual character.

Even though modern sampling-based motion planning methods can perfectly handle models of dozens of degrees of freedom, human motion has specific patterns which must be respected to produce believable motions (e.g., locomotion). Hence, a whole-body model is not adequate for planning with sampling-based techniques: human motions are living in very tight sub-manifolds of $\mathbb{R}^{57}$. In this work we therefore propose a reduced model of the system, useful for planning and helpful to reduce the dimensionality when analyzing and storing the motion database. It is not until the last stage of our algorithm that we use the whole-body model again, when we have to compute the required angle for each degree of freedom. The motivation to determine the correct reduced model is to find a common space into which both motion analysis and synthesis can be modeled and eventually compared.

4.1 Reduced Model for Motion Analysis

Motion capture data is frequently used when eye-believable motions are desired as output. The usual procedure to deal with this data is to divide the recorded motion into small clips according to the behavior the actor performed. The clips can be segmented by hand or automatically (e.g., as in [3] or more recently in [19]).

In our application, input data consists of clips of several hundreds of frames, where the complete pose of the character as well as its root's position and orientation is specified. Here, all motion clips have been resized to contain the same number of frames which will be referred to as n_f. Hence, the data we keep from each clip can be considered as a matrix $\boldsymbol{C}_l$ in $\mathbb{R}^{3n_j x n_f}$, where n_j is the number of joints. Each column of this matrix corresponds to one frame in the clip. Typically, the number of frames in one clip is around 500.

To reduce the dimensionality of this input data in the motion analysis procedure, a deformable polygonal mesh which bounds the character is used as illustrated in Fig. 2(b). Polygonal meshes are of common use in the area of Computer Graphics to handle skin or clothes of virtual characters.

The displacement of any control point p_k of the mesh (any vertex on Fig. (b)) is associated to the

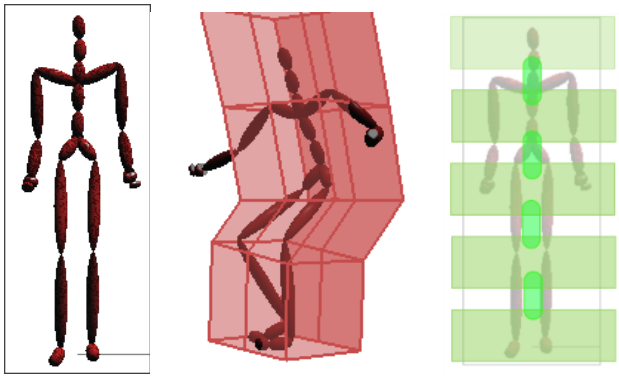


Fig. 2. (a) Whole-body model of the character, (b) Polygonal mesh used for motion analysis, (c) Linked boxes used for motion synthesis

motion of a joint or a set of joints of the skeleton through a weight $w_{i,k}$, which empirically represents the influence of the joint *i* on the motion of the vertex point p_k of the mesh. For example, a weight $w_{i,k} = 0.5$ means that the vertex p_k moves half the amount and in the same direction as the center of the joint *i*. The displacement of any given joint is measured

relative to the initial posture, usually called the bind pose. Hence, between two consecutive frames *t* and *t+1*, the displacement of a vertex on the mesh expressed on the same reference frame is given by Eq. 1,

$$P_k^{(t+1)} = P_k^{(t)} + \sum_{i=1}^{n} \omega_{i,k} \left[a_i\left(c^{(t+1)}\right) - a_i\left(c^{(t)}\right) \right]$$
$$with\ 0 \leq \sum_{i=1}^{n} \omega_{i,k} \leq 1 \quad (1)$$

where $\mathbf{a_i(c)}$ is an operator that returns the 3D position of the center of a joint *i* by applying a $3n_j \times 1$ vector of joint parameters $\boldsymbol{c}$ to the skeleton hierarchy (i.e., direct kinematics). The matrix $\boldsymbol{W} = \omega_{i,k}$ is a sparse matrix determined empirically. Our reduced model is constructed by taking the **height** of the points $\boldsymbol{p}_k$of the polygonal mesh at every frame n_f computed as in Eq. 1.

The idea is that, when constructing our motion database, we do not use all joints but only some of the vertices $p_k$on the polygonal mesh (typically, five points on the mesh are sufficient here) which (1) greatly reduces the problem dimensionality and (2) can be directly compared to the model used in planning as explained in the next paragraphs. Moreover, to cluster motions by types, we will see that the heights of a few points $\boldsymbol{p}_k$ are sufficient.

4.2 A Model for Path Generation

For planning purposes, the idea is also to use a reduced configuration space, not only for simplifying the complexity of the problem, but also to be able to compare segments of the computed paths with the motion capture data clips described above. To this end, we use another geometrical model represented in Fig. 2(c), which consists of a series of boxes linked through vertical and limited translating joints. The number of boxes is the same as the number of control points (n_c) chosen to represent the polygonal mesh on the previous section (five in this case). The group of boxes has a size such that, when the translating joints are at their default value, they bound the character in a neutral position (Fig.3 (a)), when they are at their upper limits, they bound a jumping character (Fig.3 (b)), and when they are at their lower limits, they bound a crawling or bending character (Fig.3 (c)).

(a) (b) (c)

Fig. 3. Simplified model of the character used for motion synthesis: (a) neutral configuration, (b) configuration for avoiding lower obstacles issuing jumping behaviors, (c) configuration for avoiding tall obstacles making the character crawl

Our aim is to use such 2.5D model to plan paths not only on the 2D floor plane on the workspace but also (up to certain limits) in the 3D workspace, as we can generate configuration which *jump* over short obstacles or *crawl* under tall ones. Since our motion capture database has motion clips for these types of motion, we would like to generate paths which could be reproduced with these behaviors, where a normal 2D planner would simply avoid all the obstacles. Then, a configuration for the character will be fully specified by (1) a 2D position and orientation vector on the ground plane $\boldsymbol{v} = (x, y, \theta)^T$ and (2) a vector $\boldsymbol{h} \in \mathbb{R}^{n_c}$ storing the heights of the n_c different boxes which are not treated at the same level as it will be detailed in Section 6.

The way to generate valid configurations for this box and linkage model is by applying the *3D Chainmail Algorithm* used for volume deformation [9]. When an element of the chain is moved, its displacement affects only the position of its neighboring elements allowing fast propagation of the deformation throughout the system. When the box moves up or down, the chain linking the boxes absorbs the motion and is stretched up to some limit; when this limit is reached, the motion is transmitted to the neighbors. Hence, small displacements have only local effect while large displacements affect the whole system. For

example, consider two consecutive boxes a and b among the n_c boxes in the reduced model with their respective vertical positions $h_a < h_b$. These boxes are linked through a translation joint with maximal displacement ${\delta h_{ab}}^+$ between them. Any vertical displacement δh imposed on a (e.g., due to a collision on the box a) induces new heights ${h'}_a$ and ${h'}_b$ which will satisfy Eq. 2.

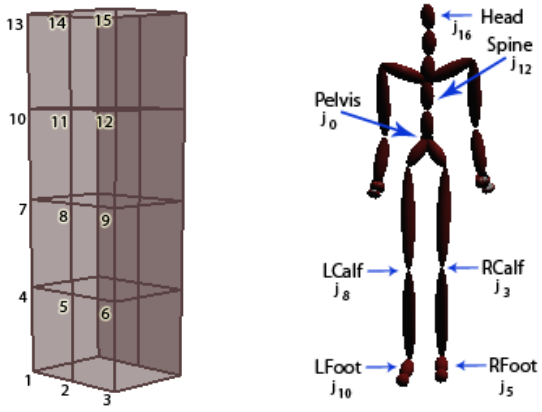
$$
\begin{aligned}
h'_a &= \begin{cases} h_a + \delta h & \text{if } \delta h < \delta h^+_{ab} \\ h_a + \delta h^+_{ab} & \text{if } \delta h \geq \delta h^+_{ab} \end{cases} \\
h'_b &= \begin{cases} h_b & \text{if } \delta h < \delta h^+_{ab} \\ h_b + (\delta_h - \delta h^+_{ab}) & \text{if } \delta h \geq \delta h^+_{ab} \end{cases},
\end{aligned}
\tag{2}
$$

The latter means that the displacement may be propagated (or not) to b. Similarly, the propagated displacement at b could also be propagated to the next level and so on. This scheme is at the core of the planning algorithm: the basic sampling is done in 3D (x, y, θ) space, but in the case of a collision with the lowest (or the highest) box, the previous propagation scheme is applied to eventually infer a collision-free configuration.

5 Clustering Mocap Databases

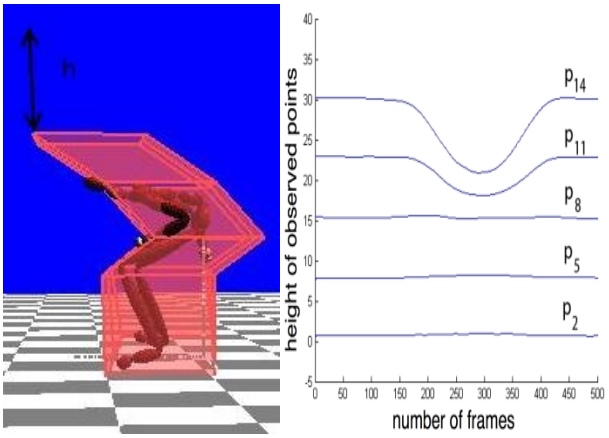
In their raw format, it can be difficult to retrieve or compare behaviors in a motion capture database due to a large dimensionality of the data ($3n_j \times n_f$). In Section 4.1, we have described a more compact representation for each clip which relies on the heights of a set of points distributed on a regular polygonal mesh bounding the character (Fig. 4 (a)). The choice of these points (vertices 14, 11, 8, 5 and 2 in Fig.4 (a)) has been made by observation to discriminate the behaviors that we have chosen to use here: running/walking, jumping and bending/crawling, but more behaviors could be included without losing the generality of the proposed model.

For now, the five control points we consider are the ones that are closer to the head, the spine, the pelvis, the midpoint between the left and right knees and the midpoint between the left and right feet (Fig. 4(b)).



a) b)

Fig. 4. (a) Polygonal mesh used to reduce dimensionality of the input mocap clips. The numbered vertices p_k are shown. (b) Most weighted joints for the considered behaviors



a) b)

Fig. 5. (a) A frame from a bending motion bounded by the polygonal mesh. (b) Reduced input vector for the bending motion showing the height of the five observed vertices on the mesh

As an example, the vertices 13, 14 and 15 on the top of the box all include only the contribution of the head joint J_{16}, while the vertex 2 is considered with equal weights that have the value of 0.5; the contribution of the left foot joint J_{10} and that of the right foot joint joint J_5.

After this process, each input capture clip with initial dimensionality of $3n_j \times n_f$(57x500) is reduced to a $n_c \times n_f$ (5x500) vector of heights along frames. Fig. 5(a) shows one configuration of a bending motion bounded by the polygonal mesh and Fig. 5(b) shows an example of the time evolution of the five heights of the observed control points during the same bending motion. Fig. 5(b) clearly demonstrates that the head and shoulders of the character lower while the feet, knees and waist remain at the same height, which is characteristic of any bending behavior.

5.1 PCA on Motion Capture Data

At this point, our data is represented by the stacked entries of matrices $\boldsymbol{H}_l$, i.e., a vector $\boldsymbol{h}_l$ of dimension $n_h \stackrel{\text{def}}{=} n_c \times n_f = 2500$, which is still very large for behavior classification. In order to further reduce the dimensionality, we turn to *Principal Component Analysis* (PCA) which has the additional advantage of reducing noisy data components and keeping only the most relevant parts of the data for classification.

In addition to the problem of dimensionality, we have as input data many more variables (heights of the mesh vertices in all frames in a clip) than observations (number of clips in the database) and therefore standard PCA cannot capture the relationships between variables. Instead, we use *Kernel-PCA* with a linear kernel [4]. Our PCA-based process for generating a compact representation of the database consists in the following steps:

1. Collect a simplified representation as explained before, of each of the N clips, from a $n_j \times n_f$ matrix $\boldsymbol{C}_l$ to a $n_h \times 1$ vector $\boldsymbol{h}_l$.
2. Compute the vectors mean $\overline{\boldsymbol{h}}$ and the resulting centered vectors: $\widetilde{\boldsymbol{h}_l} = \boldsymbol{h}_l - \overline{\boldsymbol{h}}$.
3. Set the $N \times n_h$ data matrix.

$$\boldsymbol{M} = \begin{pmatrix} \widetilde{\boldsymbol{h}_1}^T \\ \ldots \\ \widetilde{\boldsymbol{h}_N}^T \end{pmatrix}.$$

4. Set the $\mathrm{N} \times \mathrm{N}$-gram matrix $\mathrm{K} = \mathrm{MM}^{\mathrm{T}}$.
5. Apply SVD on $\mathrm{K} = \mathrm{USV}^{\mathrm{T}}$.

Once the SVD has been computed, any vector $\boldsymbol{h}$ can be projected on the base formed by the eigenvectors of $\boldsymbol{K}$ (columns of $\boldsymbol{V}$), and its

$$(\mathbf{h} - \hat{\mathbf{h}})^T \mathbf{M}^T \mathbf{V} \sqrt{\mathbf{S}^{-1}}. \quad (3)$$

coordinates are shown in Eq. 3.

Now note that, among the *N* eigenvectors, some are much more significant than others. Hence, we will simply consider the first $n_r < N$ eigenvectors with highest eigenvalues.

Each height vector will be represented by its coordinates on this base. An example of such a projection is given in Fig. 6 and Fig. 7. In Fig. 6, a set of 16 motion capture clips $\boldsymbol{h}_l$ in the format of heights vectors is shown.

In Fig. 7, their coordinates onto the $n_r = 3$ first eigenvectors of $\boldsymbol{K}$ are shown altogether with their

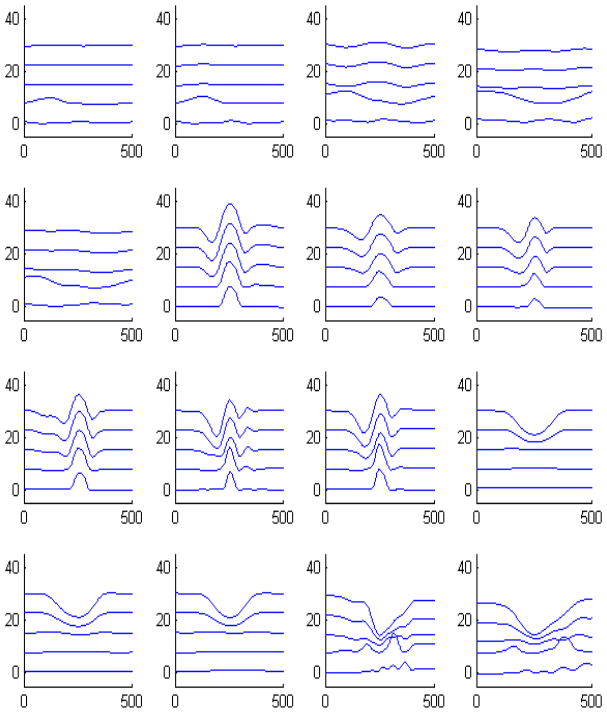


Fig. 6. Sequences of heights (vectors $\boldsymbol{h}_l$) for different motion capture clips forming our database. Walking, bending and jumping behaviors are noticeable. The vertical axis is the height of the observed points in the mesh and horizontal axis is the number of frames

annotation in terms of behavior. It is interesting to note that with this scheme, the main behaviors in which we are interested in this work, namely, walking/running, jumping and bending/crawling are well separated, enough to be able to classify new height vectors as explained hereafter.

5.2 Time-Centering of Height Vectors

For our whole process to capture the variations of the motions in the database, special care has to be put on the clips to avoid adding bias in the data because of time shifts. As an example, Fig. 8(a) shows one of the heights of a vector $\boldsymbol{h}$ extracted from the CMU motion database [7].

As the jump in this case is at the end of the sequence, this vector would be considered quite different from the other vectors of the same type, where the jump occurs sooner in the sequence. As a consequence, vectors are first centered before being inserted in the PCA process as

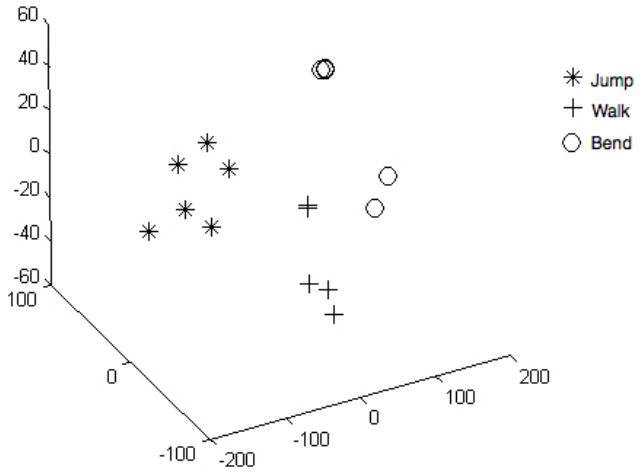
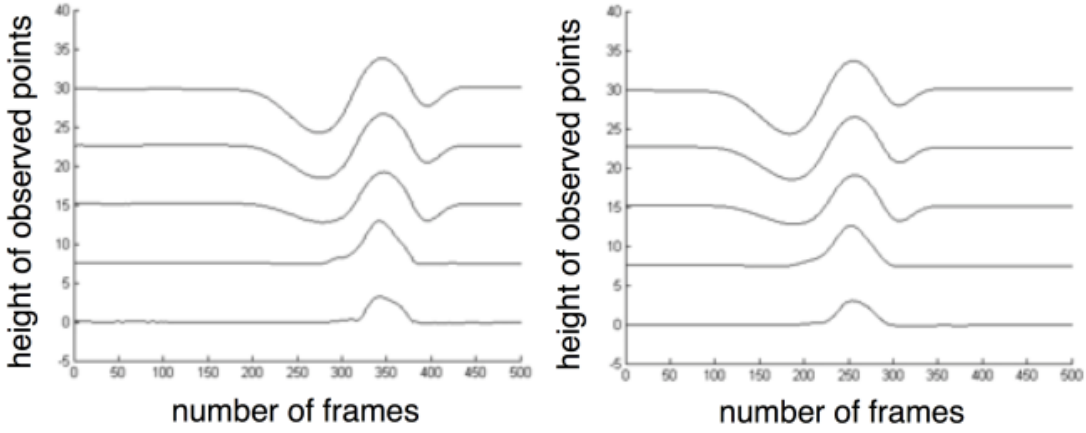


Fig. 7. Projections of the previous height vectors, with annotations of behaviors. The three axes are the three main components issued from the PCA process



(a) (b)

Fig. 8. (a) A non-centered jumping motion from the CMU database. (b) The same jumping motion, time-centered

described before, and this centering consists simply in identifying the highest peak in the height vectors and setting it at the center of this time series. Fig. 8(b) illustrates the centering process applied on the height vector of Fig. 8(a).

5.3 Classification of Height Vectors

An important stage of our work is to segment paths generated by a path planner and convert them into a height vector $\boldsymbol{h}$ as the ones we described above. By doing so, we can then extract from the database of motion capture clips the most relevant clips to perform the required path segment in a natural way. For selecting these "closest" clips, we used a clustered database, such as the one illustrated in Fig. 7: clips are associated to a behavior, and a new height vector has to be assigned to one of these behaviors in order to choose an adequate strategy to execute the motion, i.e., it has to be *classified*. Here, we use a k-nearest neighbor algorithm in $\mathbb{R}^{n_r}$ which is the space of coordinates on the base of the eigenvectors.

6 Planning and Synthesizing Motions

After the motion database is constructed using the input clips, a motion generation stage is needed to synthesize new motions which can adapt to the environment. The second stage of our method is therefore a motion planner that computes collision-free motions for a virtual character. For this, three main steps are followed:

1. A collision-free path is computed using the reduced model from Fig. 2(c).
2. The computed path is segmented into homogenous parts by detecting large changes in the path height coordinates.
3. Each segment is compared with the elements in the database to find the closest behavior and to determine the adequate controller to generate the whole-body motion which can follow the computed path.

Each step is further described in the following paragraphs.

6.1 Path Planner

In order to obtain a collision-free path, any sampling-based method can be used. The main idea of these techniques is to capture the topology of the character's collision-free configurations C_{free} into a roadmap without computing the graph explicitly (i.e., by randomly sampling C_{free}). Once the roadmap is computed, it is used to find a path connecting the initial and final configurations. In this work, we have chosen to use a variant of the *Probabilistic Roadmap* (PRM) algorithm [10], a multiple-query sampling-based method. Multiple-query methods are divided in two stages: a *learning phase* and a *query phase*.

In the *learning phase*, feasible random configurations are drawn in the character's configuration in space C. If a random configuration is collision-free, it is connected to the nearest sample using an edge only if this edge also lies inside C_{free}. The form of these edges, also known as *local paths*, depends on the kinematic constraints of the system for which the path is being computed. In this work, based on the results from the area of Movement Neuroscience [1] suggesting that most humans exhibit nonholonomic constraints when navigating in large open environments, we add differential constraints to the reduced model for planning. Bézier curves of the third degree are used as in [17] to ensure these differential constraints with the additional advantage of obtaining smooth paths.

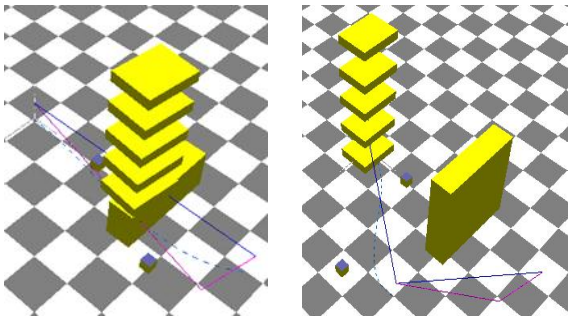
The aim of the *query phase* is then to find a path in the roadmap constructed during the learning phase. For this, the initial and final configurations are added as new nodes and connected with local paths with a node of the existing roadmap. Then, a graph search is performed to find a path between the start and the goal configurations. If such a path is found, then it can be smoothened to remove useless detours.

The novelty of our planning algorithm is on the leaning phase. As it was mentioned in Section 4, all the DOFs of the reduced system are not treated in the same way by the planning algorithm. First, feasible (within DOF limits) random configurations are drawn for the 2D

position and orientation *only*, $(x, y, \theta)^T$, of the reduced system for planning (shown in Fig. 2(c)). When colliding configurations are detected, they are not immediately rejected from the roadmap, but instead, the 3D Chainmail algorithm (see Section 4) is triggered. Fig. 9 shows this process for two boxes, one of which is avoided using the Chainmail algorithm shrinking the boxes upwards (Fig. 9(a)), a classic PRM algorithm would have discarded this configuration. The second box cannot be avoided with this technique; the obstacle has to be avoided by going around it (Fig. 9(b)). Here, the Chainmail algorithm looks for a configuration that can avoid this collision by displacing the boxes of the reduced model up or down until the joint limits are attained or the collision is avoided. If the collision cannot be avoided, the sample is discarded, but if it can be avoided, it is stored in the graph.

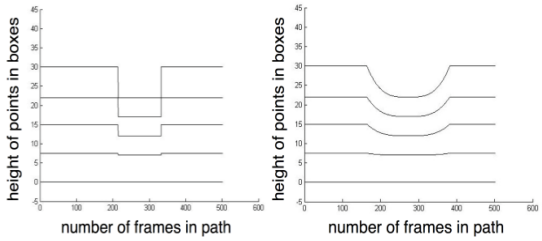
The resulting path computed by the planner is a sequence of configurations specified with the eight DOFs of the reduced model: three for the character's position and orientation in space $(x, y, \theta)^T$ and five for the heights of each of the translation joints as described in Section 4.

In Fig. 9(a) and Fig. 9(b), three paths are shown on the plane. The path in light gray (magenta in the color version) shows the path



a) b)

Fig. 9. Two different samples in the learning phase of our planning algorithm. (a) A colliding configuration is avoided using the Chainmail algorithm displacing some boxes upward. (b) No collision-free configuration was found using the Chainmail algorithm. The obstacle was avoided by going around it



(a) (b)

Fig. 10. Example of a sagittal view of the trajectory output from the planning algorithm. Here, an obstacle was avoided using the Chainmail algorithm by lowering the upper elements of the reduced model. (a) Original trajectory. (b) Smoothened trajectory

resulting from the algorithm before any optimization or smoothing. This path contains generally several sub-paths which are straight line segments even for very small paths. The dark gray (blue on the color version) is a path optimized recursively to avoid useless detours. Finally, the dotted path (green on the color version) is the smoothed path using Bézier curves of third degree.

Fig. 10 shows the trajectory for the heights of the boxes in the sagittal plane. Fig. 10(a) is the output trajectory using the Chainmail algorithm for a big obstacle on the top of the environment (the reduced model has to shrink in the direction of the floor) and Fig. 10(b) shows the same trajectory after being smoothened to simplify the comparison with the motion capture database.

6.2 Path Segmentation and Segment Classification

Once a path is computed, a segmentation process needs to be applied on it in order to extract n_s sections of motion that can be compared to the motions in the mocap database. Ideally, each segment S_i would have a unique type of motion, regardless to its particularities and duration. Each segment will be the input to a simple nearest-neighbor classifier within the samples in the mocap database so that they can be labeled with the type of motion they correspond to (see Section 5.3). Our segmentation process is very simple: we detect

strong changes in the heights of the boxes of the reduced model. The cut of each segment S_i is set (experimentally) to a number k of frames before and after a strong change is detected to account for the transition between motions. Fig.11 shows an example of a path segmented into five pieces.

When all the segments are obtained, they have to be classified using the procedure from Section 5.3 in order to obtain the type of motion that should be generated for each S_i. To be able to compare each segment with the data in the mocap database, it is resampled to have a length of 500 frames, so as to form a vector $\boldsymbol{h}$ living in the same space as the aforementioned reduced motion capture clips. Each segment has therefore a dimension of $n_c \times n_f = 5 \times 500$, the same as the PCA-transformed mocap data inside the database.

Finally, the classifier projects the segment into the coordinate system provided by the training data principal components and the nearest neighbors found (see Fig. 12) give the adequate type of movement to follow the path.

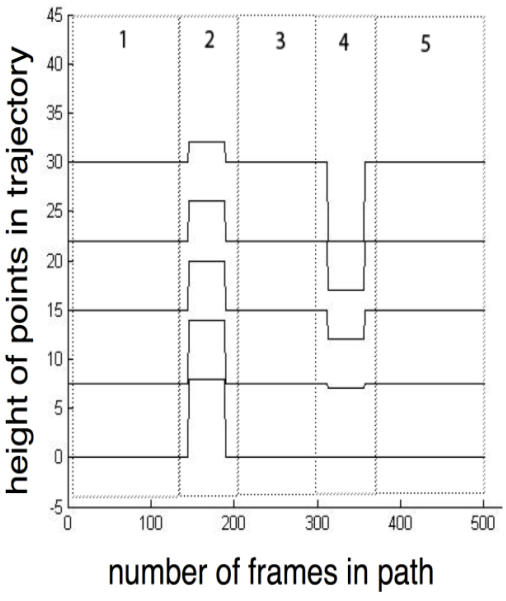


Fig. 11. Segments are divided when a strong change in their height is detected

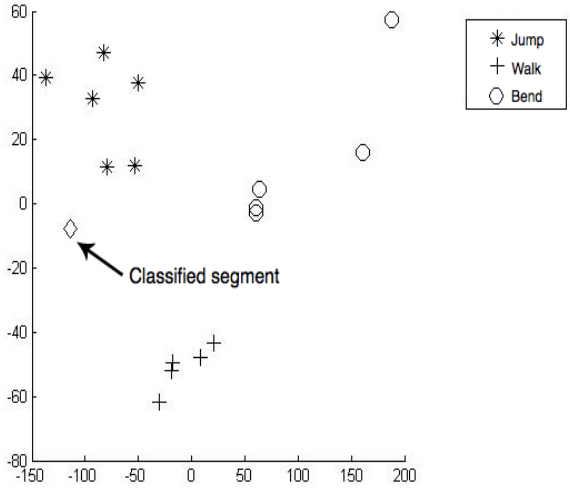


Fig. 12. A segment is classified by projecting it to the coordinate system defined by the two principal components of the training data and then finding its nearest neighbor. In this example the segment is classified as a jumping motion

6.3 Whole-Body Motion Synthesis

Once the type of motion has been identified from the classification process, a specific local controller for each type of motion (walking, jumping or bending) is used to generate the whole-body motions needed to follow the path. Each controller needs a different set of input parameters, one of them being the linear velocity provided by the user. Here, we only use three types of motion and therefore three local controllers. The first one is the locomotion generator presented in [15] which uses as input the linear and angular velocities of the desired walking pattern (extracted from the path) and produces a locomotion sequence by interpolating the three closest captures from the mocap database. This controller is used when the segment is classified as a walking/running motion. The second controller chooses an appropriate jumping motion in the mocap database by using the linear velocity and the jumping height extracted from the path segment. The third

controller, for bending motions, is very similar to the jumping controller except that it takes as an input the amount of displacement of the highest point of the reduced model to obtain a bending or crawling motion capture that avoids the obstacle. After the whole-body motions are obtained from each segment, they are interpolated with the previous segments to get the complete trajectory. Algorithm 1 sums up the complete motion generation method.

7 Simulations

In this section, we present some representative results obtained by applying our algorithm to a virtual mannequin asked to navigate in different simple worlds. All the results have been obtained with the motion capture database of CMU [7] which is the largest public motion capture database available online. Since it contains a lot of motion not relevant to our application of navigation of a character in virtual worlds (e.g., baseball motions are not useful to wander in polygonal environments), we have selected a very restricted subset of the data, namely, 16 motions represented in Fig. 6. These motions include walking/running, jumping or bending/crawling patterns. To construct the database in such a way that a comparison with planned paths is possible, the "*skinning*" process (see Section 4.1) is applied to all 16 input motions, and the resulting $n_c \times n_f$ dimensional vectors are projected onto the coordinate system specified by the first principal components obtained after applying the Kernel-PCA method (see Section 5.1).

Algorithm 1 Whole-body motion generation algorithm for virtual characters

Require: Constructed database { see Section 5 }
1: Compute path p { see Section 6.1 }
2: Segment path $p = \cup \text{Segment}_i$ { see Section6.2 }
3: **for** $i = 1$ to n_s **do** {n_s is the number of segments in the path}
4: Label_i = nearest motion clip to Segment_i
5: $\text{Sequence}_i = \text{subcontroller}(\text{Label}_i, \text{Segment}_i)$
6: **end for**
7: **for** $i = 1$ to n_s **do**
8: Trajectory = Trajectory + Sequence_i
9: **end for**
10: **return** Trajectory

Interestingly, very few principal components have to be handled to capture the database variations. As an illustration, Fig. 13 shows the reconstruction of one of the particular motions (not from the database) which can be done with an increasing number of principal components. As it can be seen, the original motion (corresponding to the one obtained with 16 components, on the down-right corner) is already quite fairly reconstructed for $n_r = 3$ components. Due to this fact, we considered $n_r = 3$ in our experiments. We checked that these three main components contain at least 90% of the information from the original motion data.

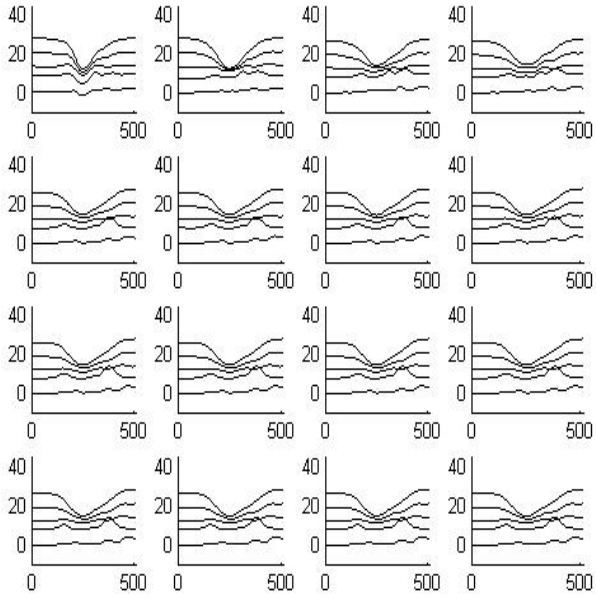


Fig. 13. Reconstruction of a particular motion with increasing numbers of principal components (from 1, on the top-left corner to 16 on the down-right corner reading them by rows). The vertical axes of each graph are the heights of the observed points and the horizontal axes are the number of frames

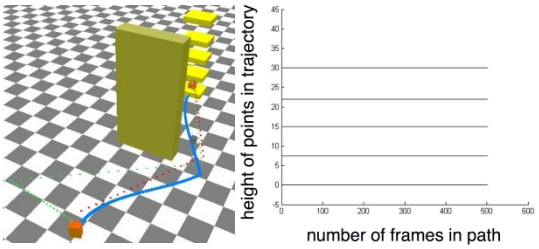
The software implementing our algorithm has been entirely written in C++, and the authors of [15] have kindly lent us their code for the motion controller of walking patterns. The scenes shown in our examples are quite simple but we eliminate the clutter to make our point and show that obstacles of different heights can be avoided either by bending, jumping over them or walking around them.

The first example of a synthesized motion is a walking motion illustrated in Figures 14, 15 and 16. As it can be seen, it contains a parallelepiped obstacle which cannot be avoided by jumping or crawling. Our algorithm first computes a collision-free path by using the PRM-Chainmail method (Section 6.1) which generated a path avoiding the obstacle by passing around it. This path is shown in Fig. 14(a). The final smoothened path is displayed with a thick curve (blue on the color version). A graph of the heights of the centers of the boxes is shown in Fig. 14(b). As there are no strong height changes, only one segment is extracted by segmentation and classified (Section 6.2) as a walking motion in Fig. 15(a). Fig. 15(d) demonstrates the motion generated by the locomotion controller following the computed path.

The computational time for this simple example on a standard PC was 7.02s for the motion planner (PRM + Chainmail) plus 0.0026s for path segmentation plus 0.018s for path classification. One must keep in mind that the PRM construction (the main source of complexity for this algorithm) has to be done only once, and that every new query for motions to be generated use the same graph generated by the PRM initial run. This is the main advantage of multiple-query planning algorithms.

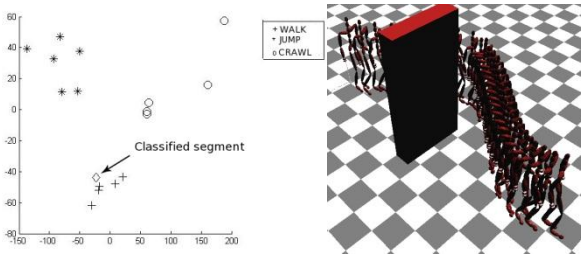
The second example is a *walk-jump-walk* motion shown in Figures 16 and 17. As it can be seen, there is a wall separating the initial and final configurations. The planner could have avoided it as in the first example, but thanks to the Chainmail algorithm, it was able to compute a path that goes *over* the obstacle.

Fig. 16(a) shows the position and orientation of the computed path on the plane. Fig. 16(b) shows a graph of the heights of the joints of the reduced model. Here, three segments are extracted using our segmentation procedure.



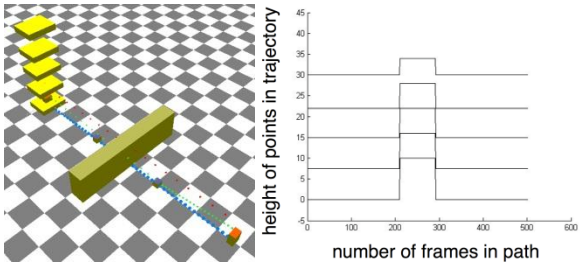
(a) (b)

Fig. 14. Walking trajectory. (a) Planned path using PRM and Chainmail. (b) Segmentation as a height vector



(a) (b)

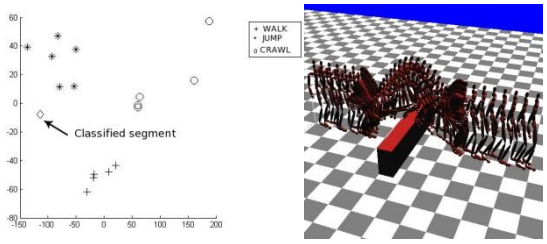
Fig. 15. (a) Classification of the height vector of a walking behavior. (b) Whole-body locomotion synthesis



(a) (b)

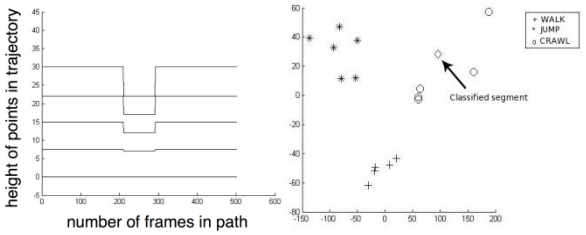
Fig. 16. Jumping trajectory: (a) Projection on the plane of the path of the reduced model's CoM. (b) Segmentation of the path in function of the height of the boxes of the reduced model

The three extracted segments are classified, the first as a walking motion, the second as a jumping behavior, and the third as another walking behavior. The classification of the middle segment is shown in Fig. 17(a). The third example (Figures 18 and 19) is similar to the previous one. Here, the scene has also one obstacle but the



(a) (b)

Fig. 17. (a) Classification of the height vector. (b) Whole-motion walk-jump-walk synthesis



(a) (b)

Fig. 18. Crawling/bending trajectory (a) Segmentation as walking-bending-walking behaviors in function of box heights. (b) Classification of the segments

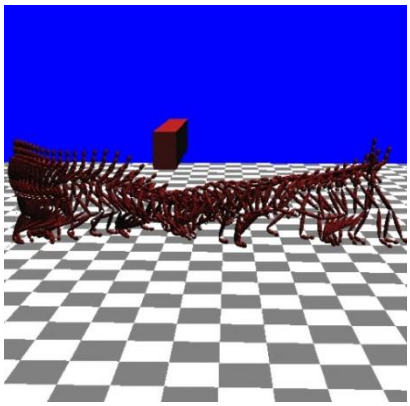


Fig. 19. Whole-motion synthesis for walking-bending-walking behaviors

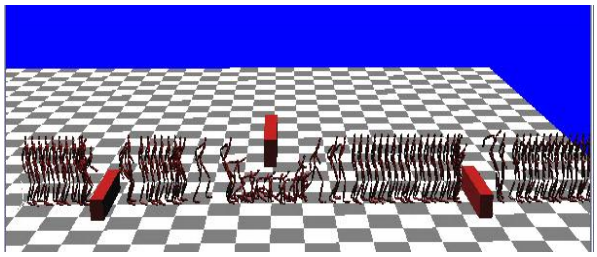


Fig. 20. Whole-body motion synthesis for a composite behavior trajectory

point is to show the algorithm avoiding the obstacle by finding a bending motion.

The planner generates a path that goes under the obstacle, and the corresponding height vector (Fig.18(a)) is segmented into three distinctive parts corresponding to two walking and one bending segments (classification of the third segment is shown in Fig.18(b)).

The final synthesized motion is depicted in Fig.19. This trajectory involves a bending motion to avoid the obstacle. A hand is used naturally for stability on the floor because it was recorded on the closest clip.

The last example presents a more complex scene (also not a cluttered scene but with obstacles of variable heights) with obstacles that can be avoided by jumping or crawling under them. The planner managed to generate paths avoiding the obstacles. After a segmentation stage where seven segments where obtained, three types of behaviors were obtained in the following sequence: walk-jump-walk-bend-walk-jump-walk. For this example the computational time was 3.56s for planning, 6.26s for segmentation and 0.018*7 for segment classification. The resulting whole-body motion can be seen in Fig. 20.

8 Conclusions

In this paper, we have described an algorithm that simultaneously plans collision-free paths and synthesize eye-believable motions for virtual characters that evolve in cluttered environments. By using two adequately chosen reduced models of the system, one for the motion capture data, one for the configuration space of the planner, our method is able to produce human-like motions chosen among the examples stored in a motion database while ensuring that the generated paths are feasible for the character. We have shown examples of trajectories generated with our method in challenging environments with obstacles which can be jumped or passed under, and we think that such an algorithm could be

particularly useful in applications like video games.

Currently, our method is limited by the fact that the motion needed to avoid obstacles (e.g., bending or jumping) may have an amplitude which is not available in the motion database, and these motions, for the moment, cannot be transformed directly to include this kind of parameter. The problem can be solved firstly by including more motions in the database. Another solution would be to make the local controllers use a generalized inverse kinematics method to locally avoid the obstacle. For example, if a bending motion has been generated, residual collisions, which may remain between the path and the obstacle, could be handled within a few frames by actuating on the torso parameters. Also, it may happen that the planner provides a path with two consecutive segments, and that, when synthesizing the motions, the first motion ends further than the point where the second motion has to start. This can be solved by interpolating the consecutive segments according to the obstacles or by adapting the motion length.

As future and ongoing work we are planning new motion controllers for different types of motion, such as jumping or bending in order to produce new motions from existing ones. Through this parameterization, we hope to satisfy the constraints imposed on the character by the computed path, e.g., to control jumps by their widths and heights. We expect to do this by integrating physically-based motion controllers, which are currently being actively researched within the Computer Graphics community, and which we would use to edit the chosen capture clips from the database. Moreover, we intend to improve the transitions between the synthesized segments which at the moment are interpolated but which we intend to assemble in a more natural way [14]. Lastly, we are also working on improving the segmentation phase, which is critical for our algorithm.

References

1. **Arechavaleta, G., Laumond, J-P., Hicheur, H. & Berthoz, A. (2008)**. On the nonholonomic nature of human locomotion. *Autonomous Robots*, 25(1), 25-35.
2. **Arikan, O. & Forsyth, D.A. (2002).** Interactive motion generation from examples. *ACM Transactions on Graphics*, 21(3), 483-490.
3. **Barbic, J., Safonova, A., Pan, J., Faloutsos, C., Hodgins, J. & Pollard, N. (2004).** Segmenting motion capture data into distinct behaviors. *Graphics Interface*, 1,185-194.
4. **Bishop, C.M., (2007).** Pattern Recognition and Machine Learning. Springer.
5. **Choi, M.G., Lee, J., & Shin, S.Y. (2003).** Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics*, 22(2), 182-203.
6. **Choi, M.G., Kim, M., Hyun, K., & Lee, J. (2011).** Deformable motion: squeezing into cluttered environments. *Computer Graphics Forum (Eurographics),* 30(2), 445-453.
7. CMU's Motion capture database. (2003). Retrieved from http://mocap.cs.cmu.edu.
8. **Esteves C., Arechavaleta, G., Pettré, J., & Laumond, J-P (2006).** Animation planning for virtual characters cooperation. *ACM Transactions on Graphics*, 25(2), 319-339.
9. **Gibson, S. (1997).** 3D Chainmail: a fast algorithm for deforming volumetric objects. *International Symposium on Interactive 3D Graphics*. 149-154.
10. **Kavraki, L., Svetska, P., Latombe, J-C. & Overmars, M. (1996).** Probabilistic roadmaps for path planning in high dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4), 566-580.
11. **Kovar, L., Gleicher, M. & Pighin, F. (2002).** Motion graphs. *ACM Transactions on Graphics*. 21(3), 473-482.
12. **Kuffner, J.J. & LaValle, S. (2000).** RRT-Connect: an efficient approach to single-query path planning. IEEE *International Conference on Robotics and Automation*. 995-1001.
13. **Lau, M. & Kuffner, J.J. (2005).** Behavior planning for character animation. *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 271-280.
14. **Li, L., McCann, J., Faloutsos, C. & Pollard, N. (2008).** Laziness is a virtue: motion stitching using effort minimization. *Short Papers Proceedings of Eurographics*.
15. **Pettré, J. & Laumond, J-P. (2006).** A motion capture-based control-space approach for walking mannequins. *Computer Animation and Virtual Worlds.* 17(2), 109-126.

16. **Pettré, J.**, Laumond, J-P. & Siméon, T. A 2-stages locomotion planner for digital actors. *ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* 258-264.
17. **Pettré, J., Siméon, T. & Laumond, J-P. (2002).** Planning human walk in virtual environments. *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 3048-3053.
18. **Schiller, Z., Yamane, K. & Nakamura, Y. (2001)**. Planning motion patterns of human figures using a multi-layered grid and the dynamics filter. *IEEE International Conference on Robotics and Automation*. 1-8.
19. **Zhou, F., De la Torre, F. & Hodgins, J.K. (2011).** Hierarchical aligned cluster analysis for temporal clustering of human motion. *Accepted for publication at IEEE PAMI 2012*.

Juan Carlos Arenas-Mena obtained his M.S. degree at the Center of Research in Mathematics (CIMAT) in Guanajuato, Mexico, in April 2010. His research interests are in Computer Graphics and Software Engineering.

Jean-Bernard Hayet graduated from ENSTA (Paris), University Paris 6 and got his Ph.D. from the University of Toulouse (2003) where he worked at CNRS-LAAS. He had been a postdoctoral fellow at the University of Liege from 2003 to 2007. Since 2007, he has worked at the Center of Research in Mathematics (CIMAT) in Guanajuato, Mexico. He teaches Computer Science and his main research interests are in Landmark-based Navigation, Motion Planning with perception and visual tracking.

Claudia Esteves has been an Associate Professor at the Department of Mathematics of the University of Guanajuato, Mexico, since September 2007. She received her Ph.D. in Informatic Systems in 2007 from the University of Toulouse, France, where she worked at CNRS-LAAS on Motion Planning for humanoid robots and virtual characters. In 2006, she made a short stay with the Joint French-Japanese Robotics Laboratory (JRL) in Tsukuba, Japan, to work on the implementation of Motion Planning algorithms. Her current research interests are in Motion and Task Planning of anthropomorphic mechanisms and Motion Planning with perception constraints.

Article received on 09/02/2011; accepted on 03/11/2011.