Abstract

With the increasing performances of the graphics hardware, virtual reality applications are reaching a new level of realism. It is now possible to animate entire crowds of virtual avatars, or to display in real-time highly realistic deformable characters. This new type of applications requires that characters are deformed, either to generate a large number of individuals for creating a crowd, or to match a particular shape. To make the animation fit to the animated character and to prevent self-collisions, adapting the motion is mandatory. Unlike previous approaches which relied on skeletal models, this paper presents a method considering the actual shape of the character’s body to perform the adaptation. Our approach uses spacetime optimization to remove the self-penetration, and finally re-establishes the balance of the motion. We also introduce an interpolation scheme based on radial basis functions that can blend pre-calculated adaptations, and thus achieve real-time performances.
Keywords: Computer Graphics, Animation, Virtual Reality
Introduction

Deformable characters allow to customize the shape of a body to reproduce the specific characteristics of a real person. Various approaches were proposed [1] and several companies from the clothing industry developed their own in-house solution [2]. Until now, none of these commercially available packages were able to provide body animation along with the deformations. The reason for this is twofold. First, the cloth simulations provided by these software packages are of high quality and thus would require a big amount of computation time. Second, a given body animation only matches the body for which it was created. Therefore, when a body is deformed the animation should be adapted accordingly. Moreover, crowd applications [3] are demanding in terms of avatars: to be realistic, a crowd should feature a wide variety of characters and animations. This variety involves that a motion clip is modified for each character being part of the crowd, otherwise all of them would move the same way.

Deformable characters introduce a new kind of self-penetration which we call shape penetration. Unlike other types of self-penetrations, the shape penetrations last for a longer period of time, often during the entire animation. Even though self-penetrations created by the movements of the end-effectors were already addressed by previous works [4, 5], these methods based on per-frame IK are unusable here. Indeed, they would introduce discontinuities in the final animations, as explained on figure 1.

Our adaptation process has two goals: remove the self penetrations and keep the phys-
ical properties of the original motion. Our approach adapts the movements of each limb separately, and then corrects the balance of the character. This strategy has the advantage to divide the problem, making the adaptation easier to calculate.

The remainder of this paper is organized as follows. We review the previous works in next section. We then detail the creation of skeletal models from the actual body shape. Next we will explain the adaptation process and the real-time interpolation method. Eventually, we present the results we obtained and conclude this paper.

**Previous Works**

Early works [6] tackled the problem of editing an existing animation clip by using interpolation and wavelet decomposition. Gleicher [7] proposed an algorithm based on spacetime optimization [8] for the same purpose. He was the first one to introduce the notion of adaptation - or retargeting - for mapping an existing motion to a new character [9]. His method consists of defining characteristics of the motion that the user wants to keep as a set of constraints to be solved by a spacetime optimization algorithm. The optimization algorithm computes an adapted motion that re-establishes the constraints while preserving the characteristics of the original motion.

Popovic [10] again used spacetime optimization for correcting the physical behavior of a motion clip. His approach used a simplified version of the character to reduce the complexity of the problem. Tak et Al. [11] used a slightly different approach to address the same
issue. A Kalman filter estimates the state of the character at a given frame, and the pose is then optimized according to constraints defined by the designer. As the optimization is performed on a single frame instead of the whole sequence, it performs much faster than the previous approaches. Abe et Al. [12] revisited the spacetime optimization approach. No simplification of the skeleton is required, and an interpolation scheme enables to achieve real-time performances. Shin et Al. [13] used closed-form method and hierarchical displacement maps to influence the physical properties of a motion.

Choi and Ko [14] used inverse rate control to perform a retargeting in real-time. Their algorithm is able to enforce several constraints while adapting a motion to a new character. Lee and Shin [15] introduced the use of Inverse Kinematics (IK) for editing motion while Tolani et Al. [16] investigated how to reach real-time performances for moving the arm or leg of a virtual character through an analytical solution for a limb with seven Degrees of Freedom (DoF). Baerlocher and Boulic [17] extended the classical IK algorithms to include the concept of priorities among the constraints. The higher priority constraint is enforced first, and the search space is then projected onto the subspace that satisfies the first constraint.

Zhao [4], Jeong [18] and Peinado [5] used numerical IK for removing self collisions. The body shape is approximated using collision volumes, and potential fields are used to detect the occurrence of a collision. Eventually, numerical IK adapts the posture of the body so that no more collision remains. However, these approaches modify a motion frame per frame and are not suitable if collisions occur during a long amount of time. More recently, Chai and Hodgins [19] used a statistical model learned from example motions to bind the mo-

5
tion adaptation process. By learning what is possible or not, the system thus only generates natural looking motions.

**skeleton design**

We used the Biped hierarchy from Character Studio with thirty joints and twenty nine bones. From this skeleton we generate a collection of cylinders to be used in the collision detection process. We use cylinders in place of a more accurate algorithm because it allows a faster adaptation. Moreover, the skin seldom deforms in a physical way but rather following interpolation schemes [20]. Thus it does not make sense to strictly comply with the deformation, as an additional physical layer should be added to account for the soft tissue deformations. Last but not least, the use of cylinders enables our algorithm to work even if the skin deformation is of poor quality.

The cylindrical model provides two quantities: a radius for each limb, to be used for removing the self collisions, and an approximated volume and center of mass (CoM) to be used for correcting the character’s balance.

Each radius is estimated by first calculating the covariance matrix $\Sigma_i$ for each limb, and then calculate the associated eigenvectors and eigenvalues, very much like a PCA analysis. We find which eigenvalue corresponds to the axis of the cylinder by comparing the eigenvectors with the actual axis of the bone (for this we retain the greatest dot product). Eventually, the radius of a cylinder is taken as the average of the two remaining eigenvalues.
An example of the generated cylinders can be seen on figures 2.

The vertex/cylinder allocation is done by using the skinning data: we keep, for each vertex, the bone with the greatest influence. The next section details how we used this cylindrical model coupled with a non-linear optimization framework to perform the motion adaptation.

**Limbs adaptation**

We want to change the animation of the character so that no self-penetration remains, while preserving other aspect of the motion. For this we use spacetime optimization as it allows to optimize complex phenomena such as human movements while complying with higher level requirements, e.g. minimize the energy consumption. In a general SQP framework [21] it is possible to assign constraints over the variables. If the starting point does not satisfy with the constraints, then the optimizer must generate a new one. This can sometimes turn out to be very difficult to do and thus we use constraints only for imposing boundaries on the parameter values (to prevent the limbs from reaching impossible configurations). The non-penetration constraints are transformed into an objective function, yielding to the following minimization:

\[
\min_{x \in \mathbb{R}^n} \ p(x) + \sum_{M} \begin{cases} 
  f_i(x) & \text{if } f_i(x) > 0 \\
  0 & \text{otherwise}
\end{cases}
\]  

(1)
With $p(x)$ a function that limits the magnitude of the parameter values and $f(x)$ the objective function. $M$ is the set of frames taken into account for the adaptation (usually the whole animation range) and $f_i(x)$ is the contribution to the objective function of frame $i$. The dimensionality $n$ of the optimization space depends on the number of frames $m$ of the animation and the number of DoF considered for each optimization. Similarly to [9], we do not act directly on the values of the parameter curves, but rather on the control points of spline curves. This extra layer ensures that the corrections do not add high frequencies to the motion, and it also reduces the dimensionality of the search space. The maximum interval between two control points appeared to be half a walk cycle, but placing points every two to five frames works well. The dimensionality of the search space is thus $n = \frac{m}{k}.l$ with $k$ the interval in frames between two control points, and $l$ the number of DoF considered. In the remainder of this paper, when speaking about a joint’s rotation, we will actually refer to the value of the control points used to calculate the corrections applied to this joint.

Even though well defined, large systems are difficult to optimize. Numerous valleys with local minima might trap the search, while peaks may prevent the algorithm from covering the entire solution space. There exists a way to break down a complex optimization problem into smaller pieces called block coordinates descent [22]. This approach assigns sets of variables that are fixed, while the other variables are being optimized. It works well if the variables are disjoint from each other, but it might have difficulties in finding the true minimum if the variables are correlated. Fortunately, the motions of a character’s limbs are well separated. Thus, an approach similar to block coordinate descent was adopted and each
limb is adapted separately, as shown on figure 3.

First the arms and legs penetrations are removed while a third step of collision removal considers the other possible penetrations, such as hands/legs, legs/torso, hands/torso, and so on. These processes slightly differ in the sense that the arms and legs adaptation correct the penetrations due to the shape of the character. For instance, if the legs are too big they must be taken apart from each other. The general purpose collision removal corrects the penetrations due to the character’s movements, for instance if the character puts its hand on its waist. The foot plant detection and enforcement can be done using one of the existing approaches [23, 24]. We picked the one from Lyard as it seems to be the easiest one to implement. We extended it so that it can handle flying phases: in such case we simply keep the original root trajectory of the skeleton.

**Arms adaptation**

We adapt the motion of the arms using two sets of variables: first the penetration itself is removed, then a second step takes place to better match the original clip.

The first step consists of rotating the shoulder joint to place the elbow at a threshold distance from the torso. We check the collision occurrence by monitoring the distance between the collision volumes, as proposed by Kuffner et Al. [25].

We want to obtain a motion free of any self penetration, but also to change the angles of the joints as little as possible to keep the resulting motion close to the original. Hence, \( p \) and \( f \)
from equation 1 become:

\[ p(x) = \alpha x^2 \]

\[ f_i(x) = \beta (d_{min}^2 - d_i^2) \]  

(2)

Here \( d_i \) is the distance at frame \( i \) between the \textit{elbow} and \textit{spine} joints and \( d_{min} \) is the minimum acceptable distance between these two joints. We took \( d_{min} \) as the sum of the radius of the two appropriate cylinders. The variable \( x \) represents the rotation of the shoulder joint in the coronal plane, i.e. the plane defined by the vertical axis and the line between the left and right thigh joints. The remaining DoF of the shoulder are exploited in the last collision removal stage, in case some penetrations remain. \( \alpha \) and \( \beta \) are picked so that the optimizer first removes the penetrations and then minimizes the corrections. In our implementation, \( x \) was expressed in radians and the distances in centimeters, and thus both \( \alpha \) and \( \beta \) could be set to 1.

Making the motion of the arms self collision free has altered the visual look of the animation. To correct this we bring back the forearm towards its initial orientation. During the first step we rotated the shoulder around one axis only. Thus we still can make the upper arm rotate around itself and bend the elbow. For this second step, \( p \) remains the same as in equation 2 while \( f \) from equation 1 becomes:

\[ f(x) = -\beta \sum_m V_i . v_i \]  

(3)

Here \( V_i \) is the vector between the original hand and elbow locations at frame \( i \), while \( v_i \) is the same vector after the adaptation. The ratio between \( \alpha \) and \( \beta \) determines how far the forearm
is brought back towards its initial orientation. A ratio of 1 brings the arm back close to its original location while setting $\alpha$ to 0 makes the adaptation complete.

**Legs adaptation**

We address the legs penetration in the same fashion as we did for the arms: we keep a minimal distance between the calf joints by adapting the legs posture. For doing so, we modify the thigh and hip joints configuration, as shown on figure 4. Three DoF (left thigh, right thigh and hip) are available to handle this penetration. Among these three only two are kept because opposite correction values should be applied to each leg. $p$ and $f_i$ from equation 1 become:

$$p(x) = \sum_{n} \begin{cases} \alpha x_i^2 & \text{if } i < \frac{n}{2} \\ \beta x_i^2 & \text{otherwise} \end{cases}$$

$$f_i(x) = d_{\text{min}}^2 - (M.V_i)^2$$

This time $V_i$ is the vector between the left and right knees at frame $i$. The first half of the vector $x$ represent the hip joint rotation values, the second half the thigh rotation. Setting $\alpha = 1$ and $\beta = 100$ produces a fair balance between the hip and thigh rotations. $M$ is a matrix projecting $V_i$ onto the coronal plane. We used this matrix to make the legs’ separation independent from the current posture. Indeed, when the knee joints are far apart - during a walk cycle for instance - penetration might still take place at the thighs. $M$ brings back the considered vector in the coronal plane and thus ensures a meaningful value of the
penetration regardless of the legs posture.

**General purpose collisions removal**

The arms and legs adaptation removes the penetrations related to the body shape. The remaining penetrations occur during a shorter amount of time, when the character puts its hand on its waist for instance. It is thus possible to use existing IK approaches such as [5, 18] to correct this. However, as we have the optimizer handy, we propose to use it for this purpose. The principle remains the same, i.e. modify a given set of DoF so that no more penetration remains. $p(x)$ remains the same as in equation 2 while $f_i(x)$ becomes:

$$f_i(x) = \beta \sum_{j} r_j^2 - d_{ij}^2$$ (5)

With $x$ gathering the corrections of the DoF chosen for the adaptation, $J$ the set of cylinders taken into account, $r_j$ the radius of the $j^{th}$ cylinder and $d_{ij}$ the distance at frame $i$ between the colliding joint and the line supporting cylinder $j$. Better than a joint, it is straightforward to define one or several vertices of the body mesh that should not penetrate the body. Each vertex becomes an offset from its master joint, thus preventing a full deformation of the skin to calculate the value of $f_i$.

We implemented this algorithm and took into account the legs/torso, hands/torso, hands/head, hands/shoulder and hands/legs penetrations. We allowed the character to bend its elbow, rotate its shoulder and bend its torso depending on the kind of penetration considered. We
placed control points every two frames because these collisions can occur within a short time.

**Balance correction**

After the penetration removal, the motion of the character has slightly changed and its balance is not maintained any longer. The final step ensures that the movement of the character are at least as balanced as in the original animation.

The Zero Momentum Point (ZMP) is the dynamic equivalent to the CoM. It is of great importance to assess the balance of a motion as it is the point on the ground where the torque created by gravity compensates the inertia of the body motion. In other words, it is the point where the character should push to remain balanced. It can be quite difficult to calculate, but fortunately there exists a closed form solution [11]:

\[
ZMP = \left( \begin{array}{c}
\frac{\sum_i m_i (\ddot{y}_i + g) x_i - \sum_i m_i \ddot{\tilde{x}}_i y_i}{\sum_i m_i (\ddot{y}_i + g)} \\
\frac{\sum_i m_i (\ddot{y}_i + g) z_i - \sum_i m_i \ddot{\tilde{z}}_i y_i}{\sum_i m_i (\ddot{y}_i + g)}
\end{array} \right)
\] (6)

The summations are done over the rigid bodies composing the figure, \( m_i \) is the mass of the \( i^{th} \) rigid body. Due to the second derivative terms, this calculation involves several consecutive frames, thus a spacetime approach seems well suited.

Before acting on the physical behavior of the character, the weights of the individual limbs must be estimated. It is very difficult, if not impossible, to accurately measure the mass directly on a real subject. Because this issue arises as soon as one tries to deal with
the physical properties of a movement, several studies were conducted in the past to get a meaningful estimate of the density of the limbs. For instance, the US army conducted series of measurements on corpses [26] and produced density measures for each limb.

Previous works used global optimization techniques to estimate the body mass [13]. They calculate the optimal mass distribution so that the original motion clip would appear to be balanced throughout its duration. However, we noticed that this optimization easily fails to accurately estimate the mass, and often falls into local minima with an unrealistic mass distribution. Instead we decided to calculate this data from the cylinders we previously fitted to the subject along with the limbs densities from [26]. This enables the direct calculation of the ZMP of the character from a given animated clip.

During a walk, the ZMP should remain within the supporting area of the character. However, due to the successive approximations made during body modeling, weights estimation and motion data processing, it often goes away from the supporting polygon. Thus, instead of bringing the ZMP back within this polygon, we calculate it for the original motion and body shape. This gives one distance $z_{mp_i}$ between the ZMP and supporting area for each frame $i$. We use this distance as a per-frame threshold for the balance correction.

**Balance Optimization**

To maintain his/her balance, a person can shift his/her CoM using the ankle and thigh articulations, or lean the torso (figure 5). We allow the adaptation process to use both these
strategies by defining four sets of free variables: two for the legs and two for the torso. Each pair of variables makes the CoM move sideways or forward/backward. The torso variables can directly be applied to the spine joint of the skeleton, while the legs variables are shared by the ankle and thigh joints with the appropriate sign value. The adaptation is achieved by reusing the optimization strategy defined in section . Here the $p$ and $f$ functions become:

$$ h(x) = \alpha \sum_n x_i $$

$$ f_i(x) = \beta (zmp_i^2 - V_i^2) \quad (7) $$

$V_i$ is the vector going from the closest point of the supporting area to the ZMP at frame $i$ and $zmp_i$ is the maximum distance from the supporting area allowed for the ZMP. This time we picked $\alpha = 10\beta$ to produce natural looking adaptations.

**Runtime system**

To fully adapt a motion to a specific character takes several minutes. This prevents a user to use it interactively for animating his/her character, or to process hundreds of thousands of models to generate a complete crowd. To accelerate the process we devised an interpolation scheme that takes pre-calculated examples as an input and interpolates the appropriate corrections for a specific body shape.

Radial Basis Functions are a very common way to interpolate scattered data. An even better
approach called normalized radial basis function (NRBF) is defined as follow:

\[
\varphi(X) = \frac{\sum_{i=1}^{N} a_i \rho(\| X - C_i \|)}{\sum_{i=1}^{N} \rho(\| X - C_i \|)}
\]

(8)

with \( N \) the number of example data points, \( C_i \) the center vector of example data \( i \); \( a_i \) is the weight associate with the example data \( i \) and \( \rho \) an interpolation function.

A theoretical justification of this formulation can be found in [27]. Intuitively, RBFs simply are a way to calculate a data sample by a weighted interpolation of existing data. If the sum of the interpolated data is not equal to 1, then the resulting data does not properly reflect the examples and should thus be normalized, hence the NRBF formula.

If we deform a body so that we choose where to take the example data point, then it is possible to accurately re-synthesize the example data. For doing so we picked \( \rho \) as the Hanning function, defined as follow:

\[
h(t) = \begin{cases} 
0.5 - 0.5 \cos(2\pi \frac{t}{T}) & t \in [0, T] \\
0 & \text{otherwise}
\end{cases}
\]

(9)

This function is commonly used in signal processing and has a shape similar to a Gaussian. Its value is truly zero outside \([0, T]\) while a Gaussian only decreases to zero when tending towards infinity. Using this function smoothly interpolates the sample data points and completely limit the influence of a point to a given distance.

The Euclidian norm is usually used for calculating the distances \( \| \ldots \| \). However, its equidistant lines are circular and thus the influence of a particular example point cannot strictly be bound between square areas. Instead, we use the infinity norm (also known as
Chebyshev distance): \[d_{\text{Cheb}}(p, q) = \max_i(|p_i - q_i|)\] with \(p\) and \(q\) \(n\)-dimensional vectors.

This distance much better corresponds to our needs because equidistant lines draw hypercubes in the parameter space instead of hyperspheres.

This way, if the example data samples are regularly spaced in all the interpolation dimensions, and if all the \(a_i\) are set to 1, then we can guarantee that the calculated data points accurately reflect the example data set. Using NRBFs for our interpolation better serve our goals than multidimensional linear interpolation because it ensures that the data is smoothly interpolated between the samples, with no discontinuities around the example data points.

**Results**

We implemented the algorithms described above in C++ using the RFSQP optimizer from AEM Design [21]. The motion adaptation itself took between 0.5 and 1 second per frame on a pentium IV 3.0 Ghz processor.

We tried our approach on various body shapes and animations and we successfully eliminated the self penetrations while maintaining a high level of realism. Figure 6A shows a the landing of a jump sequence. On figure 6B the posture was adapted to maintain the balance. Figure 6C shows the result of a penetration removal, while figure 6D is an full adaptation of a scanned body. For these examples the full adaptation has been applied, with noticeable differences emerging only when necessary. On the left, in red are the original postures, in the middle in green are the adapted ones and on the right both postures are superimposed.
The run-time system was tested by making each of the limbs successively grow and shrink in diameter while keeping the size of the other limbs fixed. We kept five dimensions for the interpolation space (legs, arms and trunk) and thus the number of optimizations to perform was \(3^5 = 243\), which took approximately one days for an height hundred frames animation. Snapshots of inputs to the system can be seen on figure 6E, while figure 7 shows an interpolated animation.

**Conclusion**

In this paper we presented a method that adapts a given animation clip so that the motion fits to any kind of body. The proposed approach uses spacetime optimization to prevent any self penetration to occur, while maintaining the balance of the character. We also proposed a variant of the NRBF interpolation to make the corrections computable in real-time.

We did not apply the forearm orientation correction to the hand because it generated too much correction, producing unrealistic postures. When the legs motion is adapted, the hip rotation makes the entire upper body rotate as well. In case one would like to keep the original head orientation (for controlling the gaze for instance), then the head joint can be rotated with the inverse of the hip rotation. Similarly, when adapt the posture of a character, the forearm orientation is also changed. This can be adjusted by applying the opposite rotation value from the spine to the shoulder. However we recommend being careful with this as it may introduce new penetrations which did not exist before.
In the future we would like to extend our approach to incorporate more physics in the adaptation to make the actual strength of the character influence the final result. We would also like to introduce physical deformations of the body in order to produce a more accurate adaptation and skin deformation. Eventually, we would like to implement the footskating removal method from [23] because the one from [24] sometimes introduced small artifacts in the final animations.

Acknowledgements

This work has been developed under the European Project LEAPFROG IP (FP6-NMP-515810). We like to thank Marlene Arevalo, Christiane Luible and Nedjma Cadi for their assistance.

References


Figure 1: Illustration of the issue related to the per-frame approaches for collision removal.

On the left is the original trajectory: The elbow joint is moving from the rear of the body towards the front, the circle representing a collision volume. On the middle a per-frame approach would apply a correction only when needed, thus discarding the natural movement. On the right is the effect we intend to produce: the motion was changed globally so that no more penetration remains, while keeping the resulting movement close to the original.
Figure 2: A virtual character (left) and its cylinders counterpart (right).
Figure 3: Conceptual representation of the motion adaptation.
Figure 4: Conceptual view of the legs configuration adaptation.
Figure 5: Conceptual view of the balance adaptation. The legs move the CoM towards the left, while the torso is bent so that the CoM moves in the opposite direction.
Figure 6: Various Examples obtained using the proposed approach.
Figure 7: Snapshots of a walking sequence of a deformable character growing along time. The adaptation data was precalculated and interpolated at runtime to adapt the animation according to the growth of the body. The cloth was animated using the method proposed in [28].