Vol. [VOL], No. [ISS]: 1–20

Real-time Animation using a Mix of Physical Simulation and Kinematics

Herwin van Welbergen, Job Zwiers, and Zsofia M. Ruttkay
Human Media Interaction, University of Twente, Enschede, The Netherlands

Abstract. Expressive animation (such as gesturing or conducting) is typically generated using procedural animation techniques. These techniques offer precision in both timing and limb placement, but they lack physical realism. On the other hand, physical simulation offers physical realism, but does not provide the precision offered by procedural motion. We introduce a new animation technique that combines the advantages of procedural animation and physical simulation. The technique is based upon simultaneously using kinematics and physics, applied to different regions of the human body, combined with a torque feedback method that couples the two. It is efficient and easy to implement as a plugin for a wide range of physical simulation engines. Source code is available online.

1. Introduction

Synthesis of expressive motion, such as conducting or gesturing, for virtual humans in interactive, real-time applications is a challenging task. Such motion is often tightly synchronized to other internal output modalities, such as speech, or external output modalities such as user input or music. *Motion (capture) editing*[1] methods are not flexible enough to deal with the many control parameters and the tight synchronization to other modalities, that is needed for such expressive motion [Thalmann 08, Gleicher 08]. *Physi-*

[1] Or editing of keyframe animation

1086-7651/06 $0.50 per page

cally simulated animation steers the body of a virtual human using muscle forces, taking gravity and inertia into account. While such motion is physically realistic, precise *timing* and limb positioning is still an open problem in real-time physical simulation [van Welbergen et al. 09]. Therefore, timed expressive motion, as used in talking and gesturing virtual humans, is typically the domain of *procedural* motion techniques [Perlin 95, Chi et al. 00, Howe et al. 05, van Welbergen et al. 06, Neff et al. 08]. However, procedural animation does not explicitly model physical integrity. As a result, the generated motion can look unnatural, as it does not seem to respond to gravity or inertia [Magnenat-Thalmann and Thalmann 96]. The inset 'on procedural animation and physical simulation' provides some background information on the animation techniques used throughout this article. We refer the interested reader to [van Welbergen et al. 09] for a more elaborate discussion on the trade-offs between naturalness, control and calculation time of different animation techniques.

Our system builds on the notion that the requirements of physical integrity and tight synchronization are often of different importance for different body parts. For example, for a gesturing avatar, tight synchronization with speech is primarily important on the arm and head movement. At the same time, a physically valid balancing motion of the whole body could be achieved by moving only the lower body, where precise timing is less important.

Our *mixed dynamics* system can apply kinematic motion (including procedural motion) on certain selected body parts, and combine this with physical simulation on the remaining body parts. [Isaacs and Cohen 87] show how a combination of inverse and forward dynamics can be used to animate an articulated body in a physically coherent manner, if either the joint acceleration or the joint torque is known for each joint in the body, at each time frame. A similar approach is commonly used in biomechanics to visualize the biomechanical movement model of interest on some body parts (using joint torques), enhanced with known motion on other body parts (using kinematic motion) [Otten 03]. Our system builds upon the ideas in [Isaacs and Cohen 87] in an interactive application, using physical motion controllers and procedural kinematic motion. Implementing the system described in [Isaacs and Cohen 87] entails implementing a full physical simulator. Implementing such a new physical simulator from the ground up is a daunting task. Nowadays, many physical simulators are available that handle the movement of articulated bodies, collision detection and friction (among others: [Smith 08, Coumans 08, Sherman and Rosenthal 01, Nvidia 08, Havok 08]). We introduce a simplification in Isaac's simulation model, which allows us to use efficient iterative techniques to calculate the torques exerted by the kinematically steered joints. Using this simplification, our system can be used as plugin for existing physical simulators.

The mixed dynamics system is integrated in our open source gesture/speech

synthesis system Elckerlyc [van Welbergen et al.].[2] One of the applications of Elckerlyc is a virtual conductor [Reidsma et al. 08]. The virtual conductor can conduct human musicians in a live performance interactively. Using knowledge of the musical score (tempo, volume, the different voices, etc.) combined with real-time analysis of the music currently played by the orchestra, he leads the musicians through the piece and corrects them when certain types of mistakes occur. Another application is our reactive virtual trainer [Ruttkay and van Welbergen 08]. She exercises along with a user, continuously monitoring the user's state (exercise tempo and performance, tiredness, etc.). During the exercise she gives him feedback and encouragement and, if needed, she adapts the tempo or difficulty of the exercise. These application requires real-time motion adaptation as well as precise control of the timing of motion. Similar timing and control requirements apply to other interactive talking virtual human applications that generate a wide range of parameterized gestures, which are aligned to the timing of speech [Chi et al. 00, Howe et al. 05, van Welbergen et al. 06, Neff et al. 08].

While our focus is on combining procedural motion with physical simulation, our system can, in principle, combine all kinds of kinematically specified motion with physical simulation. For instance, a combination of motion capture editing techniques and physical simulation can be useful to prevent physical anomalies, such as foot skate and lack of balance, that can arise when using motion editing. We have demonstrated that combining motion capture with physical simulation also allows the user-evaluation of physical balance controllers that act on a selected set of joints [Jansen and van Welbergen 09]. This is done in motion Turing tests: subjects are asked to select the natural moving character from a pair of characters: one is moved by combined motion capture and physically simulated motion (with the physical controller that is to be evaluated) and the other shows the original motion capture motion.

Our technique is demonstrated on a virtual humans by combining a physical controller for lower body balancing with kinematic animation for the upper body movements. We show how our algorithm is used with different types of kinematic arm and head animations, including parameterized procedural animation (for example, conducting motions or speech-accompanying gestures) and motion captured animation. This paper will discuss the implementation of our system in detail, providing enough information for a robust implementation.

Throughout the paper we make use of Featherstone's concise notation of the equations of motion using 'spatial' 6-vectors. The transformation from such spatial vectors to the traditional 3-vectors is shown in the appendix. For a more thorough overview of spatial algebra, we refer to [Featherstone 07].

Source code of the Featherstone inverse dynamics-algorithm, the mocap

[2] http://hmi.ewi.utwente.nl/showcase/Elckerlyc

filter and the numerical differentiation is available online at the address listed at the end of this paper.

On procedural animation and physical simulation
Procedural animation is a kinematic technique that uses mathematical formulas for motion control, given motion time and a set of movement parameters. This can be used to directly control the rotation of joints [Perlin 95]. A typical application is at a slightly higher level: the movement path of hands through space is defined mathematically to generate gestures [Chi et al. 00, Neff et al. 08, Howe et al. 05]. This approach offers precise timing and limb positioning. However, authoring physically natural full body motion using mathematical formulas is a difficult task, as physical correctness has to be explicitly addressed for all possible parameter instances.
In *physical simulation*, the virtual human is controlled by applying torques on the joints. A physical simulator then moves the virtual human's body using Newtonian dynamics, taking friction, gravity and collisions into account. A physical controller can provide motion control in real time [Wooten and Hodgins 00]. The input to such a controller is the desired value of the virtual human's state, for example desired joint rotations or the desired position of the virtual human's center of mass. The output is a set of joint torques that, when applied to the virtual human, should guide the virtual human's variables toward the desired state. A simple to implement and often used controller is the Proportional Derivative (PD) controller. The output torque of the PD-controller is proportional to the difference in position and velocity between the desired state and the actual state:

$$\tau = k_p(x_d - x) + k_d(\dot{x}_d - \dot{x})$$

in which x_d is the desired state, x is the actual state and k_p and k_d are the proportional and derivative gains. Such a system reacts like a spring-damper system, with spring gain k_p and damper gain k_d. The goal of the system is to minimize the discrepancy between the actual and desired state. It can, to a certain extent, cope with external perturbation, in the form of external forces or torques exerted on the body. However, precise limb positioning and timing is an open problem in real-time physical simulation, as in general, you can not predict when the desired state will be reached, or even if it will be reached at all.

2. Our approach

In our approach (see figure 1), motion is executed by a kinematic model (which can consist of motion editing method(s) and/or procedural motion model(s)) and by physical controller(s). The motion can be adapted in real-time by changing the parameters and timing of the kinematic motion or the desired state of the physical controller. The kinematic model directly rotates

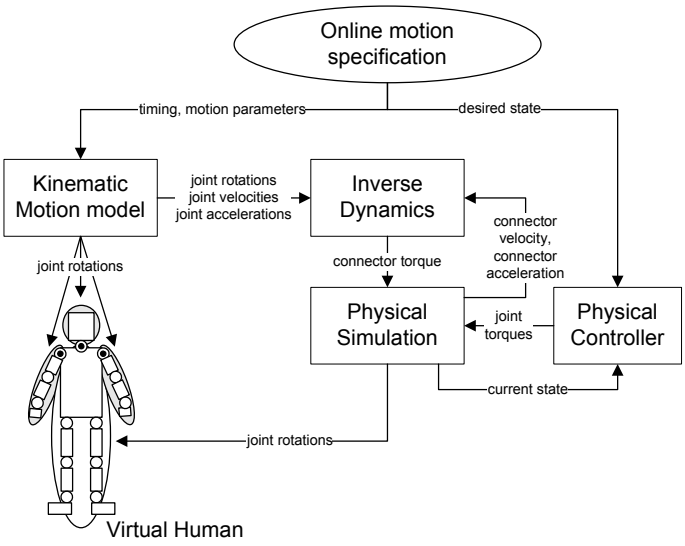


Figure 1. Architecture

the joints in the virtual human. We use inverse dynamics to calculate the torque applied by the kinematically steered body parts onto the physically steered body part, based on their rotations, angular velocities and angular accelerations. The physical controller calculates the joint torques that reduce the discrepancy between a desired physical state and the current physical state. An existing physical simulation engine is then used to calculate the joint rotations on the physically steered joints.

The body of the virtual human is divided in one physically steered part and one or more kinematically steered parts. Each part consists of joints, connected by rigid bodies. We denote the set of joints on the physically steered part by P. In our example, these joints are located on the lower body. Groups of kinematically steered joints are denoted by $K_1, ..., K_n$. The joints in each K_j are required to be connected to each other in a tree. P and all K_j's are mutually disjoint, that is, if a joint is steered, it is either steered by the kinematic model or by a physical controller. The groups are set up in such a way that each K_j connects to P at a single connector location C_j. C_j is located on the position of the root joint of K_j, in the rigid body in P that connects to this joint. See Figure 2 for an example structure.

To realistically model the effect that kinematic motion has on the physically steered body, we transfer the force exerted by each K_j to P via C_j. This force is calculated using inverse dynamics. The inverse dynamics algorithm needs the position, velocity and acceleration of each joint in K_j and the velocity and acceleration of C_j.

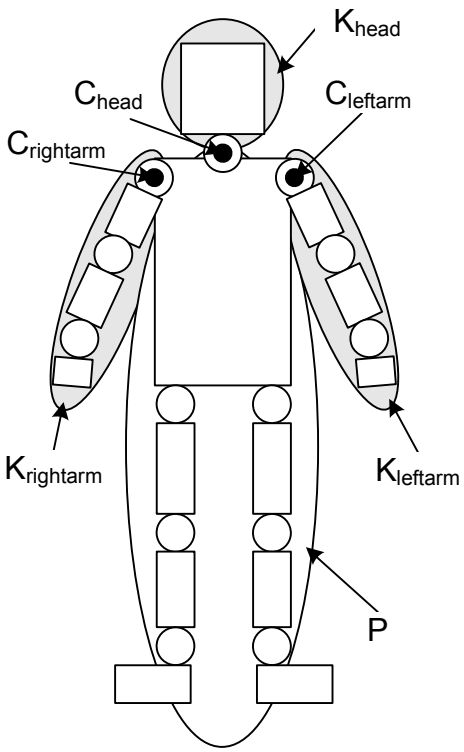


Figure 2. A body divided into kinematic parts that steer the arms and head and a physical part that steers the lower body and trunk.

The velocity and acceleration of C_j is dependent on the movement of *all* joints in the body, and can only be calculated accurately by an algorithm that takes the accelerations $\ddot{\mathbf{q}}_\mathbf{k}$ (of all joints in $K_1, ..., K_n$) and torques τ_p (of all joints in P) into account simultaneously. The equation of motion then has the form

$$\begin{bmatrix} \tau_k \\ \tau_p \end{bmatrix} = \mathbf{H}(\mathbf{q}) \begin{bmatrix} \ddot{\mathbf{q}}_k \\ \ddot{\mathbf{q}}_p \end{bmatrix} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{f}^x), \tag{1}$$

where $\mathbf{q}$, $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ are vectors of generalized joint position, velocity and acceleration, $\mathbf{f}^x$ is a vector of external forces (including gravity), $\mathbf{H}$ is the joint-space inertia matrix and $\mathbf{C}$ is the joint-space bias force. $\mathbf{q}$, $\dot{\mathbf{q}}$, $\mathbf{f}^x$, $\mathbf{H}$ and $\mathbf{C}$ are considered inputs for the algorithm. Intuitively, $\ddot{\mathbf{q}}_\mathbf{p}$ and τ_k can be calculated if $\ddot{\mathbf{q}}_\mathbf{k}$ and τ_p are known.

Currently no real-time physics engine exists that solves the equation of motion for such a hybrid specification of joint torque and acceleration. Furthermore, solving the equation of motion given both forces and accelerations can not be done using efficient iterative approaches such as the recursive New-

Table 1. Terms used in the Featherstone recursive Newton Euler approach

$\mu(i)$	set of children of body i
$\lambda(i)$	parent of body i
N_B	number of rigid bodies
$\mathbf{a}_g$	spatial gravitational acceleration
$\mathbf{v}_{C_j}$	spatial velocity of connector j
$\mathbf{a}_{C_j}'$	spatial acceleration of connector j at the previous frame
$\mathbf{v}_i$	spatial velocity of body i
$\mathbf{a}_i$	spatial acceleration of body i
$\mathbf{q}_i$	generalized DoF value vector of joint i
$\mathbf{S}_i(\mathbf{q}_i)$	matrix that maps generalized joint velocities on the DoF to spatial joint velocity
$\mathbf{I}_i$	spatial inertia tensor of body i
$\mathbf{f}_i^B$	spatial net force on body i
$\mathbf{f}_i^x$	spatial external force on body i
$\mathbf{f}_i$	spatial force transmitted across joint i
τ_i	torque exerted on joint i

ton Euler approach [Otten 03]. The recursive Newton Euler approach has a complexity of $O(n)$, where n denotes the number of joints. Typically, in a hybrid system, the equations of motion are solved using a Lagrangian approach, which has a complexity of $O(n^3)$.

Because of this, we opted to sacrifice a slight amount of accuracy to gain calculation efficiency and allow our hybrid method to be used with current real-time physics engines. Rather than calculating the acceleration $\mathbf{a}_{C_j}$ of C_j, at the *current* frame, we use $\mathbf{a}_{C_j}'$, the acceleration of C_j at the *previous* frame to calculate the forces that each K_j exerts on P. Using this simplification, we can model the movement of the K_j's as movement of isolated systems, connected to a moving base C_j, that moves with acceleration $\mathbf{a}_{C_j}'$. The torque of each joint in each K_j can then efficiently be calculated using the recursive Newton Euler approach. The reactive torque of the parent joint in K_j is then applied to the rigid body in P that is connected to this parent joint. We make use of Featherstone's formulation of recursive Newton Euler approach, using 'spatial' 6-vectors [Featherstone 07]. The transformation from such spatial vectors to the traditional 3-vectors is shown in the appendix. Table 1 summarizes the terms used in Featherstone's formulation of the recursive Newton Euler approach.

For the sake of clarity (and in our practical applications) we model each K_j as a chain of joints. This is not a limitation of our system, as the recursive Newton Euler approach can easily deal with a branching tree of joints. K_j contains a chain of N_B rigid bodies, connected by $N_B - 1$ joints. An additional

joint (joint 1) connects the chain to P at its connector location. The bodies are sequentially numbered $1..N_B$, starting with body 1, which is connected at the connector location C_j by means of joint 1. The remaining joints connect the rigid bodies in the chain: joint $i \in 2..N_B$ connects body $i-1$ with body i. Figure 3 illustrates the numbering convention used.

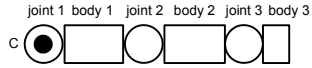


Figure 3. Numbering convention for joints and rigid bodies.

The spatial velocity of body i can be calculated as the sum of the spatial velocity of its parent and the spatial velocity across the joint connecting it to its parent:

$$\mathbf{v}_i = \mathbf{v}_{\lambda(i)} + \mathbf{S}_i \dot{\mathbf{q}}_i \qquad (\mathbf{v}_0 = \mathbf{v}_{C_j}), \tag{2}$$

where $\lambda(i)$ denotes the number of the parent of joint i. $\dot{\mathbf{q}}_i$ is the n-dimensional vector of generalized joint velocity, in which n is the number of degrees of freedom of the joint. $\mathbf{S}_i$ is a 6×n matrix that maps $\dot{\mathbf{q}}_i$ to spatial joint velocity. The spatial acceleration of body i can be calculated by differentiating equation 2:

$$\mathbf{a}_i = \mathbf{a}_{\lambda(i)} + \mathbf{S}_i \ddot{\mathbf{q}}_i + \dot{\mathbf{S}}_i \dot{\mathbf{q}}_i \qquad (\mathbf{a}_0 = \mathbf{a}'_{C_j} + \mathbf{a}_g) \tag{3}$$

The net force acting on body i is given by the equation of motion

$$\mathbf{f}_i^B = \mathbf{I}_i \mathbf{a}_i + \mathbf{v}_i \times^* \mathbf{I}_i \mathbf{v}_i \tag{4}$$

in which $\mathbf{I}_i$ is the 6×6 spatial inertia tensor. $\times^*$ is the spatial cross product of force and velocity operator (see equation 22 in the Appendix). Successive iteration of equations 2, 3 and 4 with i ranging from 1 to N_B provides the net forces acting on all bodies in the chain.

The spatial force transmitted from body $\lambda(i)$ to body i, across joint i is given by:

$$\mathbf{f}_i = \mathbf{f}_i^B - \mathbf{f}_i^x + \sum_{k \in \mu(i)} \mathbf{f}_k, \tag{5}$$

in which $\mu(i)$ is the set of children of a body. For a chain of bodies

$$\mu(i) = \begin{cases} \emptyset & \text{if } i = N_B \\ \{i+1\} & \text{if } i < N_B \end{cases} \tag{6}$$

$\mathbf{f}_i^x$ is the net external spatial force acting on body i. The values of such external forces are assumed to be known. For instance, gravity can be modeled as an external spatial force [3]. Figure 4 illustrates equation 5 for a chain of rigid bodies.

[3]However, it is more efficient to model a uniform gravitational field as a fictitious spatial acceleration of C_j, as we did using the gravitational acceleration vector $\mathbf{a}_g$ in equation 3.

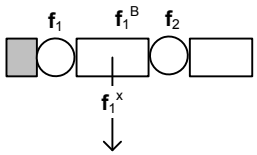


Figure 4. Spatial forces acting on rigid body 1. $\mathbf{f}_1^B = \mathbf{f}_1 + \mathbf{f}_1^x - \mathbf{f}_2$, in which $-\mathbf{f}_2$ is the reactive force of joint 2 on body 1.

Successive iterations of equation 5 with i ranging from N_B down to 1 will calculate the spatial forces acting on all joints in the chain.

Finally, the torque at joint i is given by:

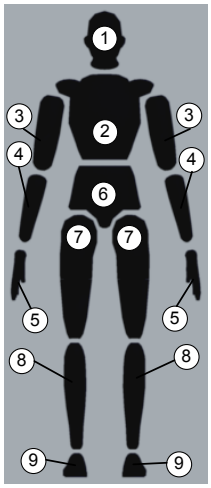
$$\tau_i = \mathbf{S}_i^T \mathbf{f}_i, \tag{7}$$

The reactive torque $-\tau_1$ is the torque excerted by K_j on a physicical body consisting of P and K_j. We assume that the inertia of K_j is small compared to the inertia of P and apply a reactive torque $-k\tau_1$ directly to the rigid body from P that is connected to K_j. Alternatively, one can augment the inertia of P with the current combined inertia of the rigid bodies in K_j, by modifying the inertia tensor of rigid body from P that is connected to K_j on each simulation frame if the physical simulator allows one to do this. If this is not the case, one could use the articulated body method ([Featherstone 07], chapter 7) to calculate the spatial acceleration of a physical body P augmented with K_j resulting from applying $-\tau_1$. We can then calculate the torque to be applied on a physical body consisting solely of P (as used in the simulator) to achieve this desired spatial acceleration. In practice the assumpution holds for kinematic gesture motion on the arms and neck combined with physical motion on the lower body and physically convincing motion is generated without requiring such computations.

For a value of $k = 1$, the exact torque generated by the kinematic chain is applied to the rigid body in P. Values of k in the range $0 < k < 1$ can be used to increase the stability of the physical simulation. This can be seen as a crude way to model an increase in muscle tension to dampen the effect of large movements. Values of $k > 1$ can be used to exaggerated the effect of the joint torques of in the kinematic motion.

3. Illustration

We illustrate our mixed kinematic/physical simulation approach by combining a physical balancing model for the lower body with kinematic motion: a procedural arm swing, conducting arm gesture, a speech-accompanying gesture or a motion capture recording. Videos of these animations are available



	body part	ρ kg/l	mass kg	center of mass (x,y,z) in m		
1	head&neck	1.11	4.80	0.00	1.63	0.01
2	upper-body	1.03	25.20	0.00	1.28	0.02
3	upper arm	1.07	2.95	±0.31	1.44	-0.01
4	lower arm	1.13	1.59	±0.32	1.17	-0.03
5	hand	1.16	0.70	±0.27	0.88	-0.01
6	sacroiliac	1.03	10.56	0.00	1.00	0.01
7	upper leg	1.05	8.79	±0.09	0.77	0.01
8	lower leg	1.09	3.73	±0.10	0.34	-0.03
9	foot	1.10	1.41	±0.10	0.04	0.02
	Total		94.21			

Figure 5. Segmentation of the virtual human into rigid bodies and the inertial properties of the bodies

on the web address listed at the end of this paper.

3.1. Physical model of our virtual human

The physical model of our virtual human consists of 15 rigid bodies, connected by 14 joints (see Figure 5). Meshes of these rigid bodies were constructed by segmenting the mesh of the original virtual human, adapting it to be skin-tight, and closing the gaps in the resulting segments. We assume that the rigid bodies have a uniform density ρ. This density can be measured directly, from cadavers for instance, or using scanning systems that produce the cross-sectional image at many intervals across the segments [Winter 04]. We use the density table from [Winter 04], which provides densities for all our segments but the sacroiliac, where we use the density given in [Dempster and Gaughran 67]. Given the uniform density of each body and its closed polyhedral shape, we can determine its mass, its center of mass and its inertia tensor, using [Mirtich 96]. The results are shown in Figure 5.

We base the joint rotation limits for the physically steered joints on data from male US air force personal [Woodson et al. 92](see table 2).

Precise collision shapes are not crucial in our applications, and collision detection is fast when simple bounding shapes, such as boxes, capsules and spheres are used. We set the collision shape of the rigid bodies to the bounding box of their mesh. If mores precise collision detection is needed, the actual

Table 2. Joint rotation limits, in degrees, in a right-handed coordinate system, with the y-axis pointing up and the virtual human facing the positive z direction. Rotation limits for the shoulder around the y-axis are not provided in [Woodson et al. 92].

joint	x_{min}	x_{max}	y_{min}	y_{max}	z_{min}	z_{max}
left wrist	-27	47	-	-	-90	81
right wrist	-27	47	-	-	-81	90
left forearm	-	-	-103	113	-	-
right forearm	-	-	-113	103	-	-
left elbow	-142	0	-	-	-	-
right elbow	-142	0	-	-	-	-
left shoulder	-188	61	?	?	-48	134
right shoulder	-188	61	?	?	-134	48
neck	-60	61	-79	79	-41	41
left ankle	-38	35	-	-	-24	23
right ankle	-38	35	-	-	-23	24
left lower leg	-	-	-43	35	-	-
right lower leg	-	-	-35	43	-	-
left knee	0	113	-	-	-	-
right knee	0	113	-	-	-	-
left hip	-113	0	-31	30	-31	53
right hip	-113	0	30	31	-53	31

mesh of the rigid body can be used as a collision shape, or the collision shape can be approximated with a combination of simple bounding shapes.

3.2. Obtaining joint velocity and acceleration

The joint velocities and accelerations for each joint are calculated from its rotation data at time t, $t-h$ and $t+h$. We define $\mathbf{p}(t)$ as the rotation of a joint at time t. If the simulation rate is set to step size h, it is possible to reuse the $\mathbf{p}(t+h)$ and $\mathbf{p}(t)$ values from the previous simulation step, so that in each step only $\mathbf{p}(t+h)$ needs to be calculated. In our examples, h is 3 ms.

3.2.1. Reparameterization

The rotation of the joints is represented by quaternions. The quaternions $\mathbf{p}$ and $-\mathbf{p}$ represent the same rotation. For a sequence of quaternions, representing the rotation of a joint, switches between these alternate representation cause large differences between the quaternion components of quaternions that actually present (nearly) the same rotation. This is undesired for our signal

analysis techniques that work on quaternion components, like filtering and numerical differentiation. Therefore we reparameterize $\mathbf{p}(t)$ and $\mathbf{p}(t+h)$ so that the distance between the quaternion components of $\mathbf{p}(t-h)$ and $\mathbf{p}(t)$ and between $\mathbf{p}(t)$ and $\mathbf{p}(t+h)$ is minimized:

$$\tilde{\mathbf{p}}(t) = \begin{cases} -\mathbf{p}(t) & \text{if} \quad \mathbf{p}(t-h)\cdot\mathbf{p}(t) < 0 \\ \mathbf{p}(t) & \text{otherwise} \end{cases} \tag{8}$$

$$\tilde{\mathbf{p}}(t+h) = \begin{cases} -\mathbf{p}(t+h) & \text{if} \quad \tilde{\mathbf{p}}(t)\cdot\mathbf{p}(t+h) < 0 \\ \mathbf{p}(t+h) & \text{otherwise} \end{cases} \tag{9}$$

where $\tilde{\mathbf{p}}(\mathbf{t})$ is a reparameterized quaternion rotation at time t and $\mathbf{p}(t)$ is the original rotation at time t.

3.2.2. Filtering

Motion capture data contains high frequency noise. This noise gets amplified with time differentiation [Winter 04]. Noise will dominate the signal after double differentiation. To prevent this, we make use of the 2-pass Butterworth low pass filter proposed in [Winter 04] to cut off high frequency noise before differentiating. The filter is described by:

$$\tilde{x}_i = \begin{cases} x_i & \text{if} \quad i < 2 \\ a_0 x_i + a_1 x_{i-1} + a_2 x_{i-2} + b_1 \tilde{x}_{i-1} + b_2 \tilde{x}_{i-2} & \text{otherwise} \end{cases} \tag{10}$$

where $\tilde{x}_i$ is the filtered data at frame i and x_i is the raw data at frame i. In a Butterworth filter, the filter coefficients a_0, a_1, a_2, b_1 and b_2 are calculated as follows:

$$\begin{aligned} \omega_c &= \frac{\tan(\pi f_c / f_s)}{C} \\ a_0 &= \frac{K_2}{1 + K_1 + K_2}, \quad \text{with} \quad K_1 = \sqrt{2\omega_c}, \quad K_2 = \omega_c^2 \\ a_1 &= 2a_0 \\ a_2 &= a_0 \\ b_1 &= -2a_0 + K_3, \quad \text{with} \quad K_3 = \frac{2a_0}{K_2} \\ b_2 &= 1 - 2a_0 - K_3 \end{aligned} \tag{11}$$

where f_c is the desired cut-off frequency, f_s is the sample frequency. The digital filter introduces a phase shift in the output signal relative to the input signal. To cancel out this phase shift, the once-filtered data is filtered again in the reverse direction of time [Winter 04]. C is a correction factor for each additional pass of the Butterworth filter:

$$C = (2^{\frac{1}{n}} - 1)^{0.25} \tag{12}$$

n is the number of filter passes. In our case, $C = 0.802$. The exact value of f_c is not very critical, values of around 15-25 Hz work well in our applications. We filter the s, x, y and z components of the quaternions in the keyframe data separately and re-normalize the quaternions after filtering. If the quaternions are reparameterized according to equation 8, the renormalization only slightly adjusts the filtered quaternions in our experience. Procedural motion is typically already smooth by design and does not need filtering.

3.2.3. Calculating angular velocity and angular acceleration

For ease of calculation, we model all joints driven by kinematic motion as ball joints, i.e. with three rotational degrees of freedom. If we choose a joint's angular velocity vector ω in the joint's own coordinate system as its velocity variable $\mathbf{q}_i$, $\mathbf{S}$ reduces to

$$\mathbf{S} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{13}$$

[Featherstone 07]. ω and $\dot{\omega}$ can be determined from the quaternion rotation $\mathbf{p}(t)$ and its derivatives:

$$\begin{bmatrix} 0 \\ \omega \end{bmatrix} = 2\dot{\mathbf{p}}(t)\mathbf{p}(t)^{-1} \tag{14}$$

$$\begin{bmatrix} s \\ \dot{\omega} \end{bmatrix} = 2\ddot{\mathbf{p}}(t)\mathbf{p}(t)^{-1} \tag{15}$$

$\dot{\mathbf{p}}(t)$ and $\ddot{\mathbf{p}}(t)$ are determined using numerical differentiation of the reparameterized and optionally filtered joint rotations $\mathbf{p}(t-h)$, $\mathbf{p}(t)$ and $\mathbf{p}(t+h)$:

$$\dot{\mathbf{p}}(t) = \frac{\mathbf{p}(t+h) - \mathbf{p}(t-h)}{2h} \tag{16}$$

$$\ddot{\mathbf{p}}(t) = \frac{\mathbf{p}(t+h) - 2\mathbf{p}(t) + \mathbf{p}(t-h)}{h^2} \tag{17}$$

3.3. Simulation details

The Open Dynamics Engine (ODE) [Smith 08] is used to generate the motion of the lower body, based on the joint torques provided by the balance controller and the torques calculated by inverse dynamics. It also handles the collision detection and contact of the physical model of the lower body with

the floor. Friction of the feet with the floor is handled using ODE's simplification of Coulomb friction. In very long simulations (longer than 1 hour), small foot-lifts and accumulated simulation errors can slightly move the feet over time. If extra stability or calculation speed is needed, friction handling can be omitted by setting foot constraints that effectively 'glue' the feet to the floor, preventing them from moving completely. To take advantage of multi-processor systems, the physical simulation runs in a separate thread.

3.4. Our balancing controller

We use the balancing controller described in [Wooten and Hodgins 00]. Our controller dampens the velocity of the center of mass and steers it toward its desired position, specified by a predefined hip height and a horizontal balance location which lies in between the feet. The output of the controller are the torques, to be applied to hips, knees and ankles. To adapt to a body with different inertial properties, a single stiffness multiplier is used on all spring gains in the PD-controllers used in the balance controller. An estimation of the value of this stiffness multiplier can be calculated (see [Hodgins and Pollard 97]), but in practice its easier to tweak it manually. A video of the balance controller, using virtual humans with different physical properties (fat vs thin) is shown on the web address listed at the end of this paper.

3.5. Results

Figure 6 shows a series of captured frames of animation generated with our system, using a combination of our physical balance model with a procedurally generated large arm swing. The motion enhancement created by our system is subtle for smaller kinematic motions and therefore hard to capture on a series of images. We refer the interested viewer to the videos on the web address listed at the end of this paper to see the system in action with more subtle kinematic motions, including several procedural conducting and other gestures and motion captured arm and head movements. We also reproduce one of the motions described in [Isaacs and Cohen 87]: a physical swing is put into motion with a kinematically moving body.

3.6. Performance

In a performance test, our system animated up to 30 conducting virtual humans in real-time on a desktop computer (2.83 GHz, Quad core, 4Gb ram, Nvidia GeForce 8800 GTS videocard). Each conductor is animated by its

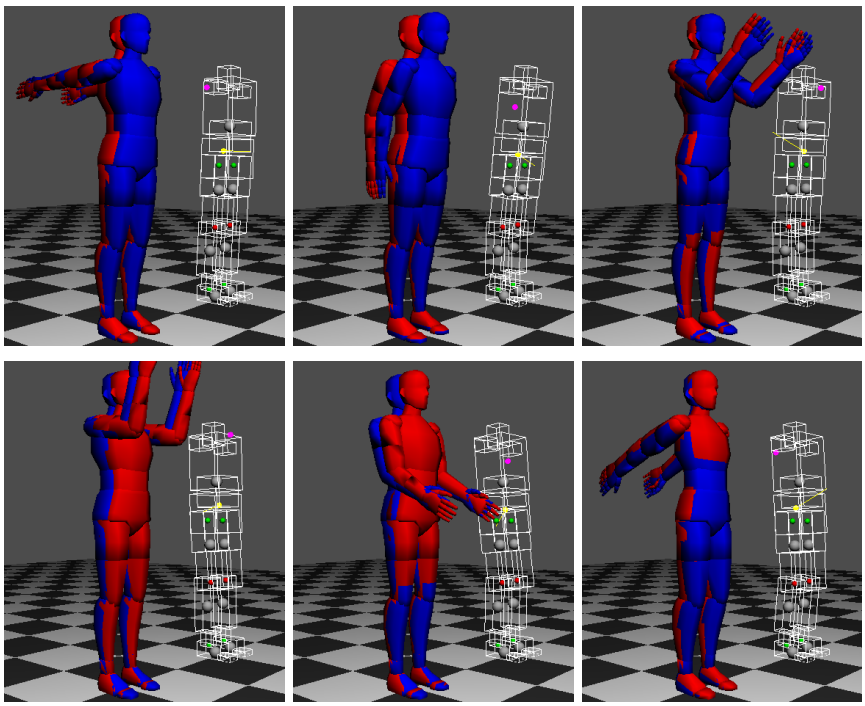


Figure 6. Mixing a kinematic arm swing with physical balancing. The blue virtual human is animated with physical simulation and kinematic arm motion, the red virtual human is animated solely with kinematic motion. The wireframe on the right side of each picture shows the visualization of the physical model of the lower body of the virtual human.

own procedural animation model and physical balance model. The physical simulation frame rate is set to 200 fps and the visual framerate is around 50 fps. Roughly half of the simulation time is spent in the procedural animation, the other half in physical simulation. See the web address listed at the end of this paper for a video of this performance test.

Like our method, *motion tracking* can be used with any existing physics engine. Motion tracking (see [van Welbergen et al. 09] for an overview of techniques) uses physical simulation on the whole body. A tracking controller is used to compute the torque on each joint. The desired state of this controller is the desired rotation of the joint, as specified in motion capture or other kinematic data. Motion capture noise, tracking errors and environmental changes can easily disturb the balance of a character whose body is animated using a tracking controller. Therefore an explicit physical balance controller is still needed. Because tracking makes use of physical controllers, the motion

generated by tracking has a time-lag relative to the kinematically specified motion and it is not guaranteed that the joint rotations specified in the mocap data are actually achieved. This makes tracking unsuitable for applications where precise timing and limb placement is needed.

Our method potentially preserves the characteristics of the kinematic motion better than tracking methods. Furthermore, our method is far more efficient than tracking methods, not only because solving the equations of motion in our hybrid system is more efficient ($O((n-k)^3+k)$ vs $O(n^3)$ using ODE), but also because it avoids the expensive double integration of acceleration for the kinematically steered joints and does not need to do collision detection on those joints. A tracking method would be preferred over our method if realistic collision detection and response on kinematically steered joints is needed and precise timing and limb placement is less crucial.

Unlike methods that model the physical balancing solely through the displacement and velocity of the center of mass [Neff 05, Oore et al. 02], our method also models the force transference from the arms to the trunk. This results in a more natural 'sharper', less smooth movement of the lower body when large accelerations occur in arm and head movement. The videos on the web address listed at the end of this paper illustrate this with a clapping motion and several conducting motions.

4. Discussion

We developed a system that can combine kinematic motion with physical simulation in a physically correct way. What we did not model yet is the fact that human balancing is not a purely *reactive* process. Our system lacks the notion of anticipation. For example, it does not move backward *in advance* to anticipate a large arm swing forward, like real humans might do. We aim to develop a new physical balancing system that can take this into account.

Other hybrid physical simulation/kinematic systems [Shapiro et al. 03, Zordan et al. 05, Zordan et al. 07] have been designed to switch between full-body kinematic motion and full-body physical simulation, depending on the current situations' needs. Such systems can show realistic interaction with the environment (e.g. falling) when needed. Rather than doing full body switches, we allow switching to a different mix of physically and kinematically steered joints in real-time. One of the usage scenarios for this is the conductor. A conductor typically conducts with his right hand and uses the left hand only for expressive cues. If the left hand is not needed, it should hang down loosely. We modeled this loose movement using a simple PD pose controller (see movie on the website mentioned at the end of the paper). The desired state for the controller is the desired rotation of the shoulder and elbow joints. The animation needed to create the expressive left hand cues require tight syn-

chronization to the music and is therefore generated by procedural motion. Switching from loosely hanging arm movement to expressive left hand conducting gesture and back occurs in real time and requires switching between different mixes of physically and kinematically steered joints. A switch from kinematical to physical control on K_j is implemented by augmenting P with the rigid body representation of K_j and applying the current joint velocity and rotation to the matching joints in the new physical representation. This will obviously result a similar torque being executed on P. Therefore such a switch results in smooth movement. A switch from the physical to kinematic control removes the physical representation of a body part from P and inserts a new kinematic chain K_j. To ensure that no sudden torques occur on the new physical body, the movement on K_j directly after the switch must be similar to the movement in its former physical representation. We refer the interested reader to [van Welbergen et al.] for a more thorough discussion on our switching mechanisms.

The inverse dynamics analysis of kinematic movement does not only yield the reactive torque, but also the torque on all other kinematically steered joints. This type of information can potentially be used in the motion planning stage, for example to drop a load if it is too heavy, or to show an angry facial expression when some motion costs more effort than anticipated.

Acknowledgments. This research has been supported by the GATE project, funded by the Netherlands Organization for Scientific Research (NWO) and the Netherlands ICT Research and Innovation Authority (ICT Regie). The authors would like to thank Dennis Reidsma for his valuable comments on the draft version of this paper.

References

[Chi et al. 00] Diane M. Chi, Monica Costa, Liwei Zhao, and Norman I. Badler. "The EMOTE model for effort and shape." In *SIGGRAPH*, pp. 173–182. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000.

[Coumans 08] Erwin Coumans. "Bullet Open Source Physics Library.", 2008. Http://www.bulletphysics.com/.

[Dempster and Gaughran 67] Wilfrid Taylor Dempster and George R. L. Gaughran. "Properties of Body Segments Based on Size and Weight." *American Journal of Anatomy* 120:1 (1967), 33–54.

[Featherstone 07] Roy Featherstone. *Rigid Body Dynamics Algorithms*. Springer, 2007.

[Gleicher 08] Michael Gleicher. "More Motion Capture in Games - Can We Make Example-Based Approaches Scale?" In *Motion in Games*, Lecture Notes in

Computer Science, 5277, Lecture Notes in Computer Science, 5277, pp. 82–93. Springer Berlin / Heidelberg, 2008.

[Havok 08] Havok. "Havok Physics.", 2008. Http://www.havok.com/.

[Hodgins and Pollard 97] Jessica K. Hodgins and Nancy S. Pollard. "Adapting simulated behaviors for new characters." In *SIGGRAPH*, pp. 153–162. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997.

[Howe et al. 05] Nicholas R. Howe, Björn Hartmann, Michael E. Leventon, Maurizio Mancini, William T. Freeman, and Catherine Pelachaud. "Implementing Expressive Gesture Synthesis for Embodied Conversational Agents." In *Gesture Workshop*, Lecture Notes in Computer Science, 3881, edited by Sylvie Gibet, Nicolas Courty, and Jean-François Kamp, Lecture Notes in Computer Science, 3881, pp. 188–199. Springer, 2005.

[Isaacs and Cohen 87] Paul M. Isaacs and Michael F. Cohen. "Controlling dynamic simulation with kinematic constraints." In *SIGGRAPH*, pp. 215–224. New York, NY, USA: ACM Press, 1987.

[Jansen and van Welbergen 09] S.E.M. Jansen and H. van Welbergen. "Methodologies for the User Evaluation of the Motion of Virtual Humans." In *Intelligent Virtual Agents*, Lecture Notes in Computer Science, 5573, Lecture Notes in Computer Science, 5573, pp. 125–131. Berlin: Springer Berlin / Heidelberg, 2009.

[Magnenat-Thalmann and Thalmann 96] Nadia Magnenat-Thalmann and Daniel Thalmann. "Computer animation." *ACM Computing Surveys* 28:1 (1996), 161–163.

[Mirtich 96] Brian Mirtich. "Fast and accurate computation of polyhedral mass properties." *Journal of Graphics Tools* 1:2 (1996), 31–50.

[Neff et al. 08] Michael Neff, Michael Kipp, Irene Albrecht, and Hans-Peter Seidel. "Gesture modeling and animation based on a probabilistic re-creation of speaker style." *ACM Transactions on Graphics* 27:1 (2008), 1–24.

[Neff 05] Michael Neff. "Aesthetic Exploration and Refinement: A Computational Framework for Expressive Character Animation." Ph.D. thesis, Department of Computer Science, University of Toronto, 2005.

[Nvidia 08] Nvidia. "Nvidia PhysX.", 2008. Http://www.nvidia.com/.

[Oore et al. 02] Sageev Oore, Demetri Terzopoulos, and Geoffrey E. Hinton. "Local Physical Models for Interactive Character Animation." *Computer Graphics Forum* 21:3.

[Otten 03] E. Otten. "Inverse and forward dynamics: models of multi-body systems." *Philosophical Transactions of the Royal Society* 358:1437 (2003), 1493–1500.

[Perlin 95] Ken Perlin. "Real Time Responsive Animation with Personality." *IEEE Transactions on Visualization and Computer Graphics* 1:1 (1995), 5–15.

[Reidsma et al. 08] D. Reidsma, A. Nijholt, and P. Bos. "Temporal Interaction Between an Artificial Orchestra Conductor and Human Musicians." *Computers in Entertainment* 6:4 (2008), 1–22.

[Ruttkay and van Welbergen 08] Z.M. Ruttkay and H. van Welbergen. "Elbows higher! Performing, observing and correcting exercises by a Virtual Trainer." In *Proceedings of the Eight International Conference on Intelligent Virtual Agents*, Lecture Notes in Artificial Intelligence, 5208, Lecture Notes in Artificial Intelligence, 5208, pp. 409–416. London: Springer Verlag, 2008.

[Shapiro et al. 03] Ari Shapiro, Frederic H. Pighin, and Petros Faloutsos. "Hybrid Control for Interactive Character Animation." In *Pacific Graphics*, pp. 455–461. Washington, DC, USA: IEEE Computer Society, 2003.

[Sherman and Rosenthal 01] Michael Sherman and Dan Rosenthal. "SD/FAST.", 2001. Http://www.sdfast.com/.

[Smith 08] Russell Smith. "Open Dynamics Engine.", 2008. Http://www.ode.org/.

[Thalmann 08] Daniel Thalmann. "Motion Modeling: Can We Get Rid of Motion Capture?" In *Motion in Games*, Lecture Notes in Computer Science, 5277, Lecture Notes in Computer Science, 5277, pp. 121–131. Springer Berlin / Heidelberg, 2008.

[van Welbergen et al.] H. van Welbergen, D. Reidsma, Zs. M. Ruttkay, and J. Zwiers. "Elckerlyc: A BML Realizer for continuous, multimodal interaction with a virtual human." *Submitted to Journal on Multimodal User Interfaces.*

[van Welbergen et al. 06] H. van Welbergen, A. Nijholt, D. Reidsma, and J. Zwiers. "Presenting in Virtual Worlds: Towards an Architecture for a 3D Presenter explaining 2D-Presented Information." *IEEE Intelligent Systems* 21:5 (2006), 47–99.

[van Welbergen et al. 09] H. van Welbergen, B. J. H. van Basten, A. Egges, Zs. Ruttkay, and M. H. Overmars. "Real Time Animation of Virtual Humans: A Trade-off Between Naturalness and Control." In *Eurographics 2009 - State of the Art Reports*, edited by Mark Pauly and Guenther Greiner, pp. 45–72. Munich, Germany: Eurographics Association, 2009.

[Winter 04] David A. Winter. *Biomechanics and Motor Control of Human Movement.* Wiley, 2004.

[Woodson et al. 92] Wesley E. Woodson, Barry Tillman, and Peggy Tillman. *Human Factors Design Handbook.* McGraw-Hill, 1992.

[Wooten and Hodgins 00] Wayne L. Wooten and Jessica K. Hodgins. "Simulating Leaping, Tumbling, Landing, and Balancing Humans." In *Proceedings of the International Conference on Robotics and Animation*, pp. 656–662. IEEE, 2000.

[Zordan et al. 05] Victor Brian Zordan, Anna Majkowska, Bill Chiu, and Matthew Fast. "Dynamic response for motion capture animation." In *SIGGRAPH*, pp. 697–701. ACM Press, 2005.

[Zordan et al. 07] Victor Brian Zordan, Adriano Macchietto, Jose Medina, Marc Soriano, and Chun-Chih Wu. "Interactive dynamic response for games." In *Sandbox: Proceedings of the ACM SIGGRAPH symposium on Video games*, pp. 9–14. New York, NY, USA: ACM Press, 2007.

Web Information:

Videos of animation generated by our system and the source code of the Featherstone inverse dynamics algorithm, the mocap filter and the numerical differentiation are provided at:
http://www.herwinvanwelbergen.nl/phd/mixed/mixed.html

H. van Welbergen, Human Media Interaction, University of Twente, P.O. Box 217, 7500 AE, Enschede, The Netherlands,
(welberge@ewi.utwente.nl)

Appendix: Conversion of Featherstone's 6D-vectors to traditional 3D vectors

Spatial velocity $\hat{\mathbf{v}}$ from velocity $\mathbf{v}$ and angular velocity ω:

$$\hat{\mathbf{v}} = \begin{bmatrix} \omega \\ \mathbf{v} \end{bmatrix} \tag{18}$$

Spatial acceleration $\hat{\mathbf{a}}$:

$$\hat{\mathbf{a}} = \begin{bmatrix} \dot{\omega} \\ \dot{\mathbf{v}} - \omega \times \mathbf{v} \end{bmatrix} \tag{19}$$

Spatial force $\hat{\mathbf{f}}$ from torque $\mathbf{n}$ and force $\mathbf{f}$:

$$\hat{\mathbf{f}} = \begin{bmatrix} \mathbf{n} \\ \mathbf{f} \end{bmatrix} \tag{20}$$

The spatial inertia tensor $\hat{\mathbf{I}}$ of a rigid body is a 6×6 vector, constructed from the inertial tensor at the CoM $\mathbf{I}^{cm}$, the mass m and the offset $\mathbf{c}$ of the CoM from the bodies origin.

$$\hat{\mathbf{I}} = \begin{bmatrix} \mathbf{I}^{cm} - m\mathbf{c} \times \mathbf{c}\times & m\mathbf{c}\times \\ -m\mathbf{c} & m\mathbf{1} \end{bmatrix} \qquad \mathbf{c}\times = \begin{bmatrix} 0 & -c_z & c_y \\ c_z & 0 & -c_x \\ -c_y & c_x & 0 \end{bmatrix} \tag{21}$$

Spatial cross product of force and velocity:

$$\hat{\mathbf{v}} \times^* \hat{\mathbf{f}} = \begin{bmatrix} \omega \times \mathbf{n} + \mathbf{v} \times \mathbf{f} \\ \omega \times \mathbf{f} \end{bmatrix} \tag{22}$$

Received [DATE]; accepted [DATE].