Volume 19 (2000), number 4 pp. 231–242

COMPUTER GRAPHICS *forum*

Versatile Tuning of Humanoid Agent Activity

Luc Emering, Ronan Boulic, Tom Molet and Daniel Thalmann

{emering, boulic, molet, thalmann}@lig.di.epfl.ch
Computer Graphics Lab. (LIG)
Swiss Federal Institute of Technology (EPFL)
CH-1015 Lausanne, Switzerland

Abstract

In this paper, we present an integration framework for heterogeneous motion generators. The objective is to outline issues that are currently easily solved in professional post-processing systems used in film and game production but which cannot be transposed as is to real-time systems with autonomous agents. We summarise our approach for articulated agent-modelling and their animation by combining heterogeneous motion generators, such as real-time motion capturing, key-framing, inverse kinematics, procedural walking. We propose an agent/action-oriented framework. Activity properties such as action simultaneity and motion blending, spatial coherence, motion-flow update schemes, agent attachments, and location corrections, are the main topics handled by our generic animation framework. Numerous examples throughout the paper illustrate our approach and outline encountered problems and solutions or open research directions.

1. Introduction

Real-time synthetic character animation as needed in virtual reality applications, is still a challenge for computer graphics scientists. Numerous animation algorithms can be found in literature, but they all suffer more or less from being specifically designed for one type of tasks, such as locomotion[11], inverse kinematics or dynamics. Although they generally produce high-quality animations, we state that they are of limited use if they cannot be combined with other motion generators, especially if real-time applications are targeted. Combining motion generators[8,10] is not trivial, because each motion generator has specific requirements concerning initial, run-time and termination states[13], such as posture, geometry, support planes, inter-agent links. Additionally the composition of motions itself generates a whole new set of animation artefacts, especially transition[15], location and collision problems. Ref. [14] describes a system that integrates motions and behaviours but the main objective is the script language allowing authors to control the animation.

In this paper, we briefly present our approach to real-time character modelling and then focus on issues resulting from the need of a generic and open plug-in architecture for motion generators. The first section presents our modelling approach for articulated characters. Section 3 presents an overview of the animation framework and introduces agent and action concepts. Section 4 focuses on problems resulting from motion mixing especially at the end-effector level. The paper terminates with a conclusion on the presented and ongoing work.

2. Modelling Humans

In order to preserve the mobility of a human skeleton in a body model, we have to handle a large number of degrees of freedom (DOF), around 40–80, excluding hands and feet. Currently, our model has 68 DOF for the main skeleton (Figure 1). It is a hierarchical structure of nodes. A node is either instantiated with a one-DOF joint, a six-DOF joint, a geometric entity or a neutral type[4].

Each node corresponds to a local reference frame that is dependent on its parent-node transformations. For example, increasing the shoulder joint's flexion DOF implies that subsequent child nodes (the arm, the forearm and the hand) move accordingly, i.e. the arm is lifted to the front. The special order of the nodes in-

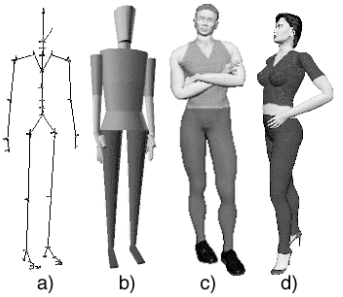


Figure 1: *Samples of humanoid body representations.*

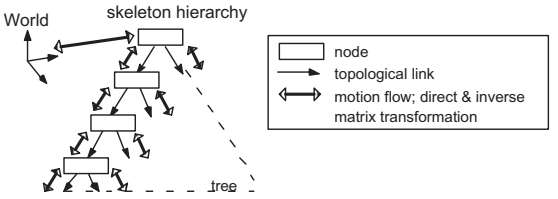


Figure 2: *Node topology and motion-flow scheme of a skeleton model hierarchy.*

side the skeleton hierarchy pre-defines the motion flow. By default, the motion flow is evaluated in a top-down approach, similarly to the topological tree structure: the parent nodes are evaluated prior to their child nodes (Figure 2).

In Section 4.2 we discuss issues of changing the motion flow in order to satisfy body support constraints more easily.

At the terminal nodes of the skeleton structure, we attach geometric primitives such as spheres, cubes, cylinders, or polygonal objects. They serve either for display purposes (Figure 1), or for computation facilities such as collision detection. Neutral type nodes serve for hierarchy organisation and inserting static translation and/or rotation offsets between a root node and a sub-tree.

3. Animating Humans

3.1. Motion Generators

We define a motion generator as an algorithm that, at each time step, generates a set of DOF values. Many different types of motion generators are available in our system: keyframing, live motion capturing, procedural walking, heuristic grasping, physics based, inverse kinematics and inverse kinetics. The reason for creating highly specialised motion generators is that most motions require specific parameterisations in order to produce realistic and believable animations. For example, 'walking' is best done with a procedural approach due to the numerous customisation possibilities. Real-time human animation for virtual reality applications requires accurate motion capture[12]. Object grasping asks for inverse kinematics or inverse kinetics[7] if the body has to maintain balance.

However, creating a set of specific motion generators raises a software integration problem. As each motion generator has its proprietary interface and its own set of adjustable parameters, it is difficult to combine the generated motions. We propose a single meta-controller capable of motion mixing, sequencing and synchronisation – without any dependency to the motion generators' specific interfaces. Another objective is an easy extensibility of the motion repertoire by programmers that ignore the framework's kernel implementation and that are not allowed to recompile the core library.

3.2. Agents and Actions

We encapsulate each motion generator as a Specialised Action within a generic model interface (Figure 3). This is a key issue for clearly delimiting the system core from the actual motion generators. Basically, we define three entities: *Agent, Action and Specialised Action.* We also distinguish four types of users: *Action Provider (AP), Core Developer (CD), Application Creator (AC) and End User (EU).* In the following section, we briefly summarise those concepts; for details on action transitions and priorities, please refer to[6].

First of all, we define an agent as an encapsulation of the character's articulated skeleton model (2). As our research interests mainly focus on human animation, we most often use humanoid agents. However, an agent can be any arbitrary rigid or articulated object. Agents are animated by the means of actions. An action can be understood as a generic virtual motion generator -'virtual' because it does not generate any motion information: it only imports the general functionality, such as 'Start', 'Execute' and 'Stop' commands, from a specific motion generator. Each agent instantiates its own set of actions. Some actions are specifically designed for humanoid agents such as walking, others apply to object agents only, and still others can be applied indifferently to humanoid and object agents, such as 'falling down' due to gravity. The agent's animation basically consists in a sequence of start-execute-stop action commands. An action is linked to a specific motion generator via a specialised action. A specialised action imports high-level tuning functions for the underlying motion generator. Specialised actions are also responsible for standardising the access to the agent structure. Indeed, a major objective of the specialised action is to prevent direct access to the agent's DOF by motion generators (Section 3.3).

Action Providers propose motion generators to be plugged into the animation library. The motion genera-

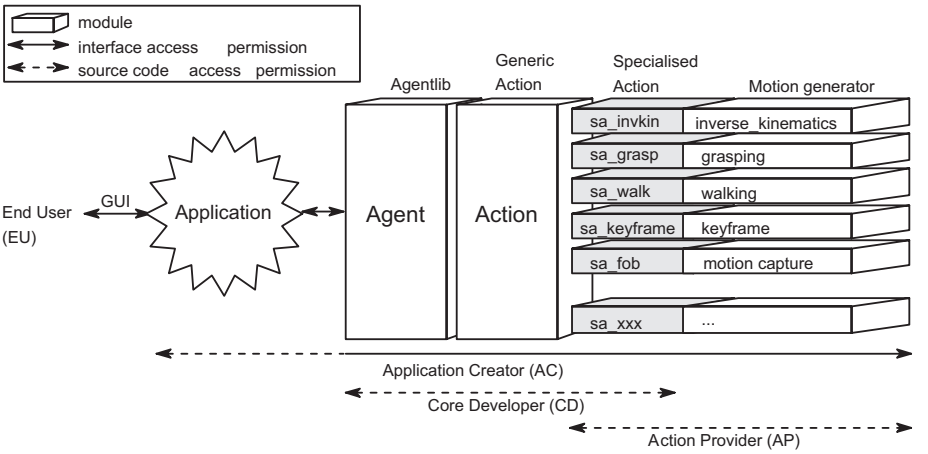


Figure 3: *Agentlib framework and user categories.*

tor can be developed independently but the associated specialised action has to be library-compliant in order to function properly with existing actions[6]. Then all features of the library's motion controller unit become instantly available, without any core compilation. Note that simple motion generators can be directly implemented inside a specialised action: for example 'nodding the head' or 'spinning around an axis' requires only small software developments. Only APs are allowed to modify the source code of the motion generator and the associated specialised action. Library Core Developers have access to the source code of the agent, action and specialised action modules. As the animation library is based on an action plug-and-play paradigm, CDs can afford ignoring the motion generators' implementation. Application Creators' access is limited to the module interfaces. They program at a conceptually higher level such as behaviours, interaction and autonomy. The last category of users, are the End Users. They interact with the application either via a graphical and/or text interface or by more sophisticated means such as virtual reality equipment[9].

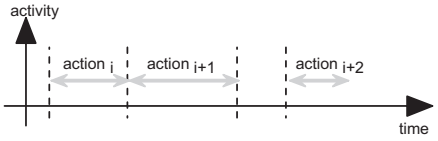


Figure 4: *Sequential action performances.*

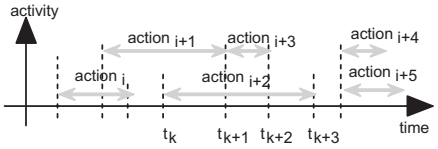


Figure 5: *Parallel action performances.*

3.3. Action Simultaneity

Realistic character animation requires more than the performance of an action sequence (Figure 4). Human actions most often perform in parallel or, at least, overlap in time (Figure 5), for example giving a phone call with a mobile phone while walking. Simultaneous actions are not necessarily synchronized. They have their own life cycle, from activation to de-activation, depending on the Application Creator's desired semantic: the phone call might finish much earlier than the walk or vice versa.

Unpredictable activation and de-activation requests for an arbitrary number of actions prevent from predefining a set of motion combinations as done in most of the games. During the DOF-conflict resolution we need a blending algorithm. Ref. [15] describes a system for creating transitions of motion units in a semi-automatic way, over the entire transition time interval. However, the employed optimization process appears to be incompatible with real-time constraints and unsuitable for free body motion. Other blending algorithms use DOF frequency analyses[3,16] or motion warping techniques[17]. The objective of these approaches are the tranformation or creation of motions from an available set of unit motions. Frequency analyses generally require the knowledge of the entire DOF functions, but in a dynamic real-time context the future of DOF evolutions is not known a priori. Motion warping is rather an interactive off-line process where an animator creates new motions by adjusting some parameters or defining constraints. In our context, we are primarly interested in automatic transition generations rather than semi-automatic mo-

Body parts	Number of DOF
BODY_HEAD	3
BODY_NECK	6
BODY_THORAX	5
BODY_ABDOMEN	5
BODY_PELVIS	3
BODY_L_THIGH	6
BODY_R_THIGH	6
BODY_L_LEG	2
BODY_R_LEG	2
BODY_L_FOOT	4
BODY_R_FOOT	4
BODY_L_U_ARM	7
BODY_R_U_ARM	7
BODY_L_L_ARM	2
BODY_R_L_ARM	2
BODY_L_FIST	2
BODY_R_FIST	2
BODY_L_HAND	30
BODY_R_HAND	30

Table 1: *Body parts.*

tion generation. We choose a generic meta-motion controller which solves simultaneous DOF-update conflicts with a weight and priority mechanism.

Testing for DOF-update conflicts can be speeded up if each action defines a minimal set of required DOF which can be quickly compared for intersections. We divide the body model into nineteen body parts (Table 1).

At creation, each action fixes its minimal set of required body parts. During action-activation, the action's body parts are compared with the body parts already in use by competing actions. In case of intersections, we either inhibit the activation (this is the default mode) or we give a higher priority to the new action and generate a motion transition through the meta-motion controller (Section 3.4) on the intersecting body parts.

3.4. Action Mixing

The action blending is performed according to Equation 1. (Table 2) shows some results of blending two actions' DOF-contributions.

Equation 1 is basically composed of two expressions representing the following two action mixing modes: blended and added mode. The action weight can be any function of time whose value range is normalised to [0.0, 1.0]. We currently use cubic-step functions because of symmetry and continuity issues (Table 2).

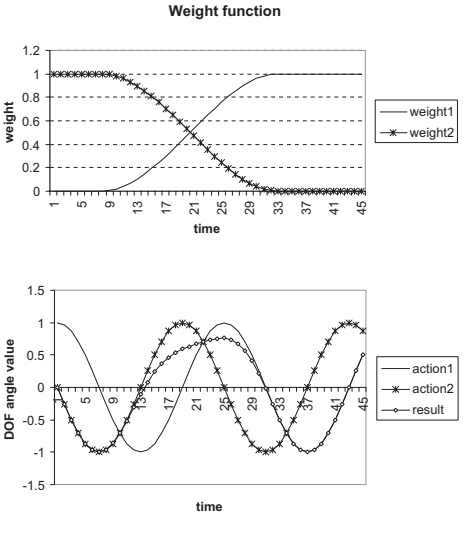


Table 2: *a) Weight functions of two action-contributions for a same DOF. b) Sinusoid action-contributions and blended result for a same DOF.*

$$\theta^{k+1} = \theta^{k} + \sum_{i=1}^{I} (\theta_i^k - \theta^k) \cdot \omega_i^k + \sum_{j=1}^{J} \theta_j^k$$

Equation 1: Motion mixing for one DOF, where

- I number of actions that perform in blended mode (I ¿= 0)
- J number of actions that perform in added mode (J ¿= 0)
- k is a time index
- θ is the current DOF value at time k
- θ_p^k is the DOF value of the pth contributing action at time k
- ω_p^k is the DOF weighting function of the pth contributing action.

- *Blended Mode*
  In the default mode, the Blended Mode, the current motion is smoothly blended with the newly activated action. For each DOF, we compute a weighted sum of the actions' contributing DOF values. We associate a weight to each action-controlled DOF. A weight is a value ranging from zero to one. It modulates the magnitude of the DOF contribution in the final motion. When an action is activated, a desired weight and duration is specified. The actions' effective weights evolve according to a cubic function, until the desired weights are reached (Figure 6).
  When several actions try to simultaneously update a

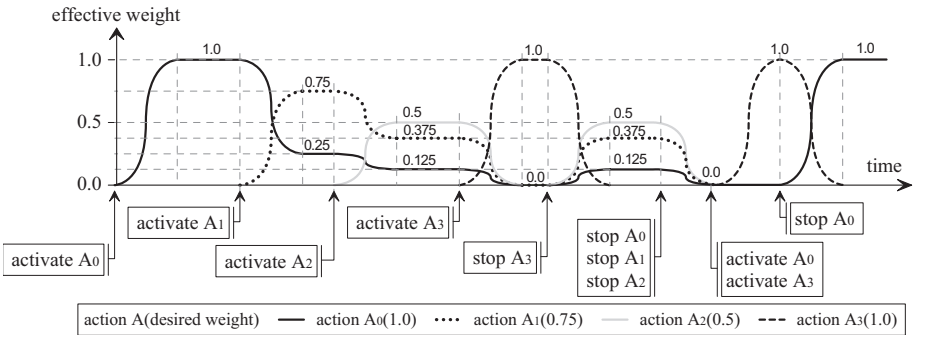


Figure 6: *Some actions' DOF-weight evolution.*

same DOF, all associated effective weights evolve such that their sum constantly equals 1.0.
However, there are two exceptions to this constraint: first, when one, or several actions are activated while no other actions are active, and second, when all active actions are de-activated. In these situations there is a transition phase during which the sum of the actions' effective DOF weights tend towards one or zero respectively.
Note that the blended mode expression of Equation 1 can be simplified when at least one action is already fully active:

$$\sum_{i=1}^{I} \omega_i^k = 1.0 \Rightarrow \theta^{k+1} = \sum_{i=1}^{I} \theta_i^k \omega_i^k + \sum_{j=1}^{J} \theta_j^k$$

Equation 2: DOF-blending for at least one fully active action.

We stress here that the weight mechanism is exclusively dedicated to blending purposes, despite its impact on the contributing action amplifications. In other terms, if an AC wants to modulate the magnitude of an action, the AP is invited to foresee such a feature at the specialised action level. For illustration, consider a 'walk' action: although the action weight would make an agent walk more or less fast, it is essential to include a speed regulator in the algorithm instead of alienating the blending weights.

- *Added Mode*
  In this mode, the contributions of newly activated actions are added to the current motion. This mode is required for actions such as breathing or fidgeting.

- *Matrix Blending*
  For the global positioning and orientation we need to blend homogenous matrices. In our human models, the global location matrix is situated at the root node of the hierarchy, the Global Motion (GM) node. An homogenous matrix represents a frame location relative to a given reference frame (Figure 7). First, we compute the delta transformations between the current matrix and the matrix of the ith contributing motion generator (Figure 7). The rotation matrix is converted to an equivalent axis-vector representation: a vector representing the rotation axis and whose norm is the rotation angle around this axis. By accounting previous angle values, we extend the angle range from $[0, 2\pi]$ to new boundaries $[-\infty, +\infty]$ in order to avoid discontinuity problems. The rotation angle is scaled by the current weight and the axis-vector is converted back to a matrix. All contributing matrices are multiplied together in order to produce the final location matrix. As matrix multiplications are not commutative, the order of the multiplications must always be the same.

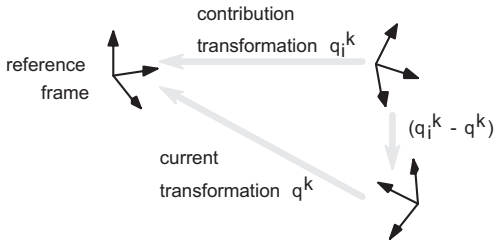


Figure 7: *Motion blending of homogenous matrices.*

- *Scope*
  The blending mechanism works correctly only if the actions do not directly update the agent's skeleton posture. The posture updates have to be co-ordinated according to Equation 1: an animation control unit attached to the agent is responsible for it. To ensure that the actions' underlying motion generators do not have write permission on the skeleton, we introduce the concept of scope. A scope is a special buffer that

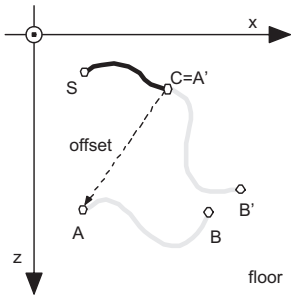


Figure 8: *Top view showing the desired agent motion continuity. It shows the reference frame, the environment floor and the displacements of one agent. The agent starts moving at point S, for example by using a walk action. At a given time, it arrives at point C and activates a keyframe action. However the keyframe action has been recorded with absolute location values. If we naïvely apply the action blending equation, the agent will slide from point C to point A and continues its displacements in direction of point B. In general, this doesn't correspond to the desired motion effect. We would rather like the agent to start playing the keyframe at point C and continue moving in the direction of point B'.*

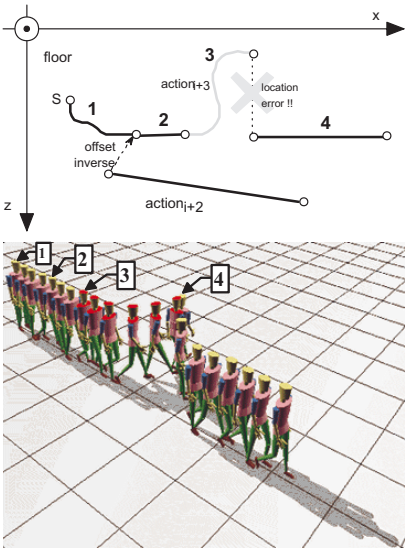


Figure 9: *Location Offset_Adjust mode. 1) Activate $action_{i+2}$ and compute offset inverse transformation for the location correction. 2) Perform $action_{i+2}$ and apply the location correction. 3) Activate and perform $action_{i+3}$, which modifies the agent location. $Action_{i+3}$ looses the agent location control because $action_{i+3}$ has the higher priority. 4) $Action_{i+3}$ terminates and $action_{i+2}$ regains control of the agent location. However the agent's location has been changed according to the influence of actioni+3. The initial offset-inverse matrix is invalid now, therefore the agent slides in order to re-integrate the agent position and location as adopted in 2).*

offers memory space for each DOF that an action desires to update. The actions' scope is declared in the action interface. The definition of each scope is the duty of the specialised action, alone to know the body parts and/or DOF used by the underlying motion generator (Figure 3). The scope also maintains the effective and desired weights associated to each DOF.

4. Animation Fine-Tuning

4.1. Spatial Coherence

An action produces a skeleton posture at each animation frame. A posture consists either of absolute or incremental DOF angle values. We choose absolute angle values and absolute agent-positioning at the specialised-action level, as reflected in Equation 1. Whenever two actions have DOF-update conflicts, the blending mechansim computes weighted relative contributions. Concerning agent positioning, we have to distinguish several cases because we cannot simply combine them in order to generate a final agent location. In the Default mode, an agent will slide from its current location to the action's absolute location-contribution. The problem is illustrated in Figure 8.

A first proposal is called the Offset_Adjust mode (Figure 9). It consists in computing an initial correction matrix in order to annihilate the offset between the agent's current location and the action's location-contribution. All subsequent action location-contributions are adjusted with this correction matrix. The Offset_Adjust mode is straightforward and requires only one additional matrix multiplication per iteration.

However, it fails in the frequent case where an action B is activated after an action A and B terminates before the end of action A (Figure 10): $action_{i+2}$ and $action_{i+3}$. At time t_k we compute a location correction matrix for $action_{i+2}$. At time t_{k+1}, $action_{i+3}$ is activated and modifies the agent's location. Therefore, when $action_{i+3}$ terminates at time t_{k+2}, the location correction matrix of $action_{i+2}$ is not up to date any more. The motion results again in an undesired location drift. Of course, one can detect all action activation and termination phases and re-compute the correction matrices, but we propose a uniform approach, called the Dynamic_Adjust mode (Figure 10). We explicitly transform the absolute location data of the action's contribution into relative location data. If subsequently applied to all actions, this solution works fine for any action activation configurations. The price to pay is a matrix inversion and two matrix multiplication per animation frame.

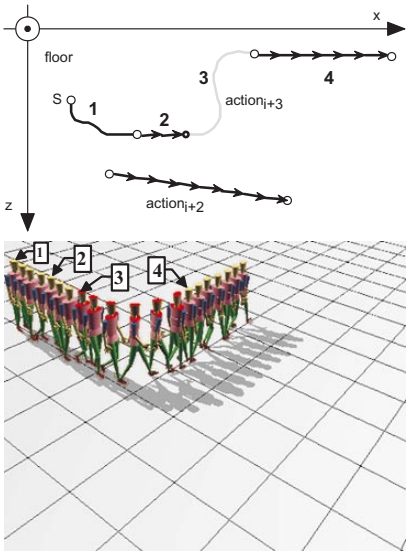
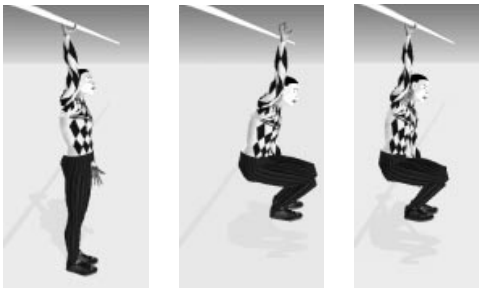


Figure 10: *Location Dynamic_Adjust mode. 1) Activate action$_{i+2}$. 2) At each iteration: compute relative location data for action$_{i+2}$ and apply it. 3) Activate and perform action$_{i+3}$, which modifies the agent location. 4) Action$_{i+3}$ terminates and action$_{i+2}$ regains control of the agent location. As we compute incremental location data for action$_{i+2}$, the agent's location is coherent.*



a) b) c)

Table 3: *a) The agent is waiting in its initial position: it is ready to start its gymnastics exercise. The gymnastics animation is a pre-recorded sequence read from a file and played back in real-time. The sequence has been recorded with a magnetic motion capture device with 12 sensors. b) The agent performs its gymnastics. The spine base is the motion flow root node, therefore it is not guaranteed that the hand stays fixed on the horizontal bar. c) The agent performs its gymnastics again. This time, the right hand is the motion flow root node: the hand necessarily remains fixed on the horizontal bar.*

Spatial coherence enforcement as presented with the Offset and the Dynamic modes, guarantees a continuous animation without agent sliding. However, these modes impact on the agent controllability. For example consider a 'restaurant' scenario where a waiter-agent has a set of pre-defined absolute motions such that table collisions are successfully avoided. In the context of the Offset and the Dynamic modes, the pre-defined motions can be seamlessly concatenated, but the dynamic transformation of the absolute motions into relative motions might violate the table-collision avoidance constraint. Due to the transformation, the agent's absolute position is not easily controlled any more. In contrast, the Default mode allows to play the pre-defined motions just as they were designed. However, the motion-concatenation might induce agent sliding. Agent-sliding can be avoided if the pre-defined motions are specifically designed: the paradigm is comparable to 'tileable texture patterns': the final phase of a first motion can be seamlessly appended to the intial phase of a second motion. The benefit is an exact absolute agent location, known at any time.

4.2. Motion Flow Scheme

Each node of the skeleton model maintains 'direct' and 'inverse' transformations relative to its parent node and relative to the global reference frame. 'Direct' and 'inverse' transformation matrices are grouped inside a single structure, the DIT (Direct-Inverse-Transformation) (Figure 1).

The update procedure of each node's global DIT depends on the topology of the skeleton model. By default, we use a top-down update approach: the skeleton's tree root node is updated first, then its children, their subsequent children and so on (Default Motion Flow Update Scheme). The entire DIT update procedure is called the Motion Flow Scheme (MFS). The MFS has always a root node, in our human body model it is the spine base node. By default, the spine base node is both the topological and MFS root.

As the spine base node spawns all body limbs, the positioning of the spine base in 3D space is equivalent to positioning the whole body in space. We position the body by updating one of the spine base's parent-node, the 6-DOF 'Global-Motion' (GM) node. This method is reasonable for most actions such as walking or jumping. However, sometimes it is preferable to position the body by an end effector - hands, feet and head are considered end effectors.

An example is an agent doing its daily gymnastics exercises with a horizontal bar: the right hand is positioned at the bar and all body limbs move relative to that hand. Doing this, requires switching from the default MFS root (Figure 3) to the hand node. A similar formulation to the MFS problem is known as re-rooting transformation as described in[1].

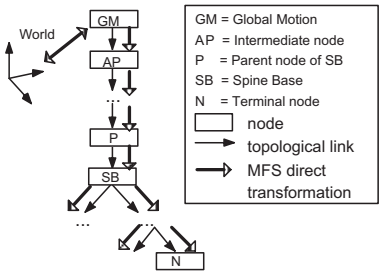


Figure 11: *Default Motion Flow Update Scheme.*

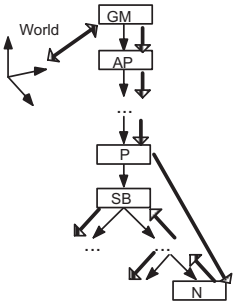


Figure 12: *Modified Motion update Flow Scheme, with an end effector as motion root.*

From a conceptual point of view, we reassign the DIT update flow in the hierarchy: we cut the flow after the GM node and redirect it through the end effector node. The end effector node, which previously was a terminal node in the MFS, now becomes a motion source node. The update flow proceeds to the topological parent node of the end effector's node, to its child nodes and so on (Figure 11) and (Figure 12).

Changing the default MFS requires the creation of an additional 6-DOF node in the hierarchy. This node stores the DIT between the new MFS root node and its parent, which necessarily differs from the topological parent. This additional node also allows the specification of a local DIT in order to position the new MFS root relative to its MFS parent. We call this approach, the topological MFS change method (tMFS) because it modifies the model hierarchy.

In the context of our motion integration framework, the tMFS approach is inadequate for the following two reasons. First, actions are not allowed to modify the skeleton hierarchy, especially the creation of an additional node in the hierarchy tree is not tolerated (Section 3.3). Second, and this is far more critical, we cannot handle the situation where two concurrently performing actions define each a proprietary MFS. Due to the topological impact of the MFS modification, it is impossible

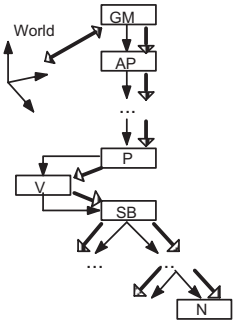


Figure 13: *Simulation of a modified Motion Flow Scheme by introducing a virtual node and reporting its effects up to the Global Motion node.*

to define a reasonable compromise. For instance, suppose that an action A defines a MFS with the left foot as motion root node: the right foot motion is relative to the left foot. Now, an action B defines an MFS with the right foot as motion root. Action B is activated while action A is still performing. The result is a circular dependency in the DIT update procedure: the right foot motion depends on the left foot motion and vice versa.

The question then is how to achieve a MFS modification without impact on the skeleton model? The answer is based on one major observation: changing the MFS does not modify the skeleton posture. For a given set of DOF values, the MFS only alters the global position and orientation of the body as illustrated in (Table 3). Both sequences only differ by a global rotation/translation transformation applied to the body. Therefore, we can simulate the MFS (sMFS) concept by computing compensating matrix-contributions for the GM node. The following equations briefly outline the sMFS matrix computations. We temporarily introduce a virtual node V at the place where the motion-flow link is cut, and then report the virtual node's value up to the GM node (Algorithm 1).

Note that the sMFS approach successfully avoids the topological modifications of the agent's skeleton model, but it does not solve the problem of concurrent actions asking for distinct MFS. In such a case, a priority has to be given to one of the actions. We suggest that the most recently activated action controls the MFS choice. If the new MFS differs from the current one, the motion meta-controller switches in an abrupt way to the new MFS. A smooth transition between MFS is technically feasible, by a fade-in fade-out paradigm of the GM delta contributions, however real-life situations corresponding to such a transition seem to be less frequent. Indeed MFS changes are mostly used when an agent moves relative to an end effector or the spine base.

Step 1	We start with a default body skeleton model (Figure 11). The current motion flow starts at GM, goes through the nodes P and SB and ends at node N (Figure 3b). We update the hierarchy with all DOF-value contributions.
Step 2	We want to redirect the motion flow from the node P through node N and propagate it upwards in the tree (Figure 12) (Figure 3c). This invalidates the current DIT between node P and SB. We temporarily introduce a virtual node V as illustrated in (Figure 13). We compute the new matrix transformation, M_V, between node P and SB, where W is the world reference frame and ${}_aM_b^V$ is the matrix transformation from a to b in the hierarchy containing V: $M_V^V = {}_P M_N^V \cdot {}_N M_{SB}^V$
Step 3	Now we report the V-node's transformation as a delta contribution to the GM node (Figure 11). The equations are based on the following equivalence: ${}_WM_{SB}$ with V = ${}_WM_{SB}$ without V but GM scaled by Δ $\Leftrightarrow {}_WM_{SB}^V = {}_W M_{SB}$ $\Leftrightarrow {}_WM_{GM}^V \cdot {}_{GM}M_P^V \cdot M_V^V = {}_W M_{GM} \cdot \Delta \cdot {}_{AP} M_{SB}$ $\Leftrightarrow {}_{GM}M_P^V \cdot M_V^V = \Delta \cdot {}_{AP} M_{SB}$ $\Leftrightarrow \Delta = {}_{GM} M_P^V \cdot M_V^V \cdot {}_{SB} M_{AP}$ as V is temporary, ${}_{SB}M_{AP}$ is equal to ${}_PM_{AP}^V$: $\Delta = {}_{GM} M_P^V \cdot M_V^V \cdot {}_P M_{AP}^V$ **Note that, if the GM-node is a direct parent node of the SB-node, the transformation matrix Δ is simply equal to M_V^V.**

Algorithm 1: Computation of simulated Motion Flow Scheme.

Finally, we mention a functional difference between the tMFS and the sMFS when we choose an arbitrary internal node as MFS root in the model hierarchy, for example the right shoulder twist node as MFS root as shown in (Figure 4c) and (Figure 4d). For the tMFS, this may result in skeleton dislocations (Figure 4c) because the hierarchy is split into two independently updated sub-trees. For the sMFS approach skeleton, dislocations are impossible (Figure 4d) because of the absence of topological impacts. However the computed GM corrections equally affect both sub-trees of such an sMFS, that is the body moves in order to stay connected to the arm model sub-tree (Figure 4d). Note that overlapping motions with two separate roots could possibly be handled as multi-point constraints with a root/effector approach such as inverse kinematics.

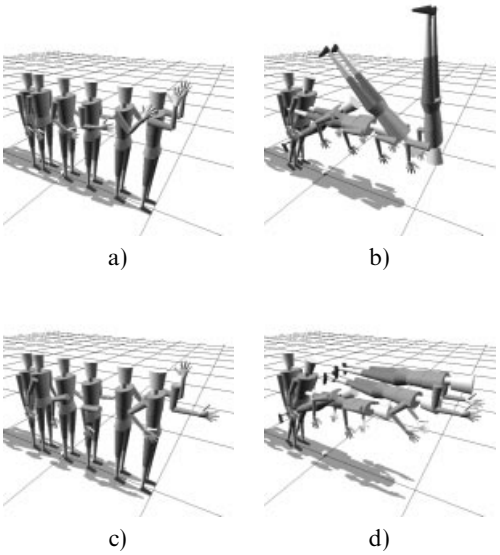
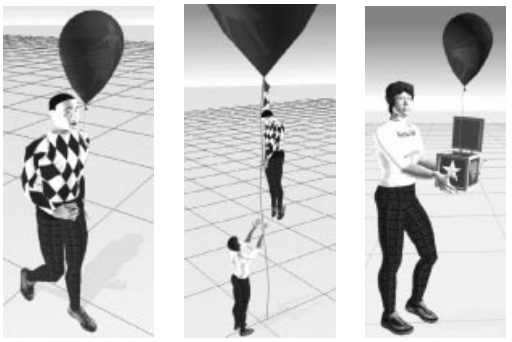


Table 4: *a) The test animation sequence with the default MFS. b) The left hand as tMFS (or sMFS) effector, the spine base node as MFS root: both motions are equivalent. c) The right hand as tMFS effector, right shoulder twist node as tMFS root: the arm disconnects from the body. d) The right hand as sMFS effector; right shoulder twist node as sMFS root: the arm stays connected, but the body moves.*

4.3. Agent Attachment

Agents can be either complex articulated structures such as humanoids, or simple objects. Behavioural animation often requires agents to be attached to other agents. The attached agent, or passive agent, looses the control of its spatial location, independently of its currently performing actions. The active agent is the one which decides of attaching another agent to itself.

For example, a human agent grasps a balloon agent and walks away with it (Figure 5a). As soon as the humanoid reaches the balloon, possibly by applying an inverse kinematics or a grasping action, we attach the balloon to the hand. From here on, the distance and orientation of the balloon remains constant relative to the agent. In order to model the relative mobility of the balloon to the agent, the balloon model can have extra animation nodes. We can reverse the role of active and passive agent (Figure 5b). The choice of the active and passive agents depends on which agent is supposed to control the other one - is the human carrying the balloon or is the balloon lifting the human into the



a) b) c)

Table 5: *a) A balloon attached to the humanoid's hand. b) A human agent attached to a balloon. c) A balloon attached to a box that is attached to the humanoid's hands.*

Figure 14: *Floor intersections, due to sequential activation of walk sequences.*

air? Attachments and detachments can be dynamically specified. They can be sequenced such that the roles of the active and passive agents are reversed, for example an active human agent carries a passive balloon agent (Figure 5a), but soon the balloon becomes an active agent and lifts, the now passive human agent, into the air (Figure 5b). By extension, any agent can be active and passive at the same time, e.g. a box attaches a balloon and is attached to a humanoid (Figure 5c).

4.4. Position Corrections

As we apply the motion blending mechanism to the body's position DOF, we sometimes get unwanted animation artefacts, such as a body lifting of the floor or, on contrary, sinking below (Figure 14).

Indeed the blending equation ignores any motion context parameters such as velocity, acceleration, gravity, amplitude, period or phase. In consequence, the continuity of these parameters is not guaranteed during the animation. There are three solutions:

First, one can use an automatic action-initialisation procedure in order to ensure motion parameter continuity. During the action initialisation phase, the action senses the agent's current state and adjusts its motion parameters accordingly. This solution, although conceptually attractive, requires a general-purpose method for parameter extraction of arbitrary motions, which is non-trivial. The concept of such parameter extraction could be inspired by approaches as presented in[16].

The second solution for floor contact enforcement consists in activating a kind of 'reset' action that re-adjusts the agent position. This action is activated whenever floor contacts are likely to appear, typically when a lot of body-displacements happen during animation. If potential floor contacts are difficult to predict, the reset action can be consequently activated prior to each new action. However, inserting an action with the objective of merely correcting the visual aspect, is generally not satisfactory: action sequences should exclusively depend on the animator's requirements, not on technical problems. Besides the conceptual problems, the 'reset' method suffers from severe side effects relative to currently active actions: the final animation is altered due to the presence of the reset action. During the motion-blending phase, the reset action receives the control of the location matrix whereas currently performing actions loose it. Thus the presence of the reset actions influences the global animation.

The third solution to the floor contact enforcement consists in working at the Agent level rather than the Action level. It is performed at the end of the motion blending process. This way the correction procedure limits its impact to state-dependent actions. State-dependent actions refer to the agent's current state in order to compute the state for the next time step. Due to their incremental nature, they will notice the correction previously applied and intrinsically take it into account in their own motion generation. Possible approaches are: correcting only the body location, correcting the location of effectors violating the floor constraint with inverse kinematics as in[5], or a combination of both. We present here the first approach because it is produces reasonably good results while being computationally interesting.

The location correction works as follows: the body position is adjusted such that at any time, one foot at least touches the floor and none of the feet ever drop below the floor. The correction consists in adding a vertical translation to the agent's location matrix.

In order to compute the translation value, we evaluate foot constraints at each iteration of the agent animation, after the action-blending step. We define a virtual sensor at strategic foot regions such as the toes, the heel or the foot centre and compute their distance relative to the floor. Increasing the number of virtual sensors improves the accuracy of foot/floor intersections but it also complicates the computation of the agent position corrections. For example, with a single sensor at the middle of the foot, floor intersections such as e) to h), cannot be detected and are therefore not corrected.

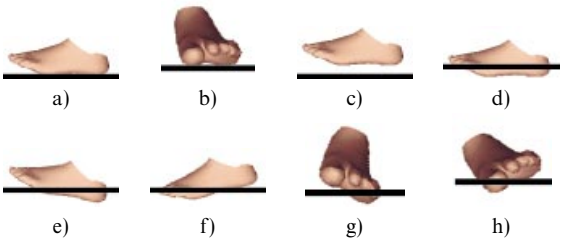


Table 6: *Easily detectable foot position configurations relative to the floor. a) Lateral foot position is correct. b) Frontal foot position is correct. c) Foot floats above the floor. d) Foot intersects with the floor. e) Heel intersects with the floor. f) Toes intersect with the floor. g) Lateral foot/floor intersection. h) Lateral foot/floor intersection.*

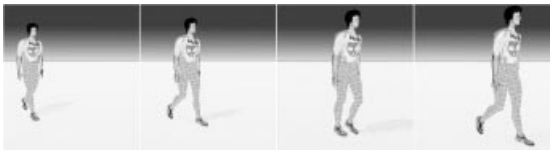


Figure 15: *Walking with vertical agent location corrections for floor contact enforcement.*

Especially when a highly flexible foot model is used, we have to use more sensors in order to get sufficiently accurate floor intersections. Currently, we use two sensors per foot, at the heel and at the toes. So the situations illustrated in Table 6g) and Table 6h) are not corrected. The floor is specified as a plane defined by a reference point and a normal vector. As the foot model varies for various humanoid models and because accurate sensor locations are not hardwired inside the model, the agent ideally includes a static floor offset-correction.

4.5. Action sequencing artefact

Sometimes a given sequence of action activation shows up awkward side effects, such as self-collisions, although the actions themselves produce a flawless animation when performed separately. (Figure 16) illustrates such a self-collision example with two sequentially performed actions. First, the humanoid is in a rest position (Figure 16a). The first action is activated: the humanoid adopts an attentive attitude by putting the hands to the back while leaning forward and starring to the camera (Figure 16b). Next, we activate a 'What's the time?' action that brings the left hand to the front of the body (Figure 16d): during the performance, the left hand collides with the humanoid's body (Figure 16c).

The self-collision is due to the humanoid's posture at the moment when the second action is activated (Figure 16b). Indeed both actions are key-frame actions,

Figure 16: *Self-collisions during the sequential performance of two actions: a) initial rest posture, a 'Be attentive!' key-frame action is performed, b) a 'What's the time?' key-frame action is activated, c) final body posture; all actions are completed.*

meaning that they play back a sequence of postures relative to the current body configuration. In particular, the artefact (Figure 16b) can be avoided by inserting a third action (similar to the 'buffer action' described in[14]) such that the hands avoid the main body. In the simplest case, this action simply resets the body to the rest posture. Alternatively this buffer action can be merged within the other actions: we insert a 'rest posture' frame at the end of the first (or at the beginning of the second) action's key-frame sequence. More elaborate schemes with automatic activation of various buffer actions are currently under study. Potentially such actions can be posture-reset actions, inverse kinematics actions with end-effector constraints, or specific collision-avoidance actions such as described in[19] or inspired by the finger/object collision avoidance approach used in automatic grasping actions.

5. Conclusions

In this paper, we presented a flexible framework implementation for integrating various kinds of motion generators. With a generic Action and Agent concept, we are able to plug in new motion generators that can then be freely mixed with existing actions. We also discussed new possibilities in animation such as action parallelism and sequencing, body support planes, Agent attachments, automatic Agent position corrections. Finally we pointed out some animation artefacts that may occur due to the generic character of our framework and proposed solutions.

The animation library is currently used for all kinds of applications in our laboratory and has proven to be a robust and easily extensible solution for integrating a large variety of animation techniques. The encouraging feedback led us to extend the framework with new features, especially in the domain of perception and object management[2].

Acknowledgements

The authors thank P. Bécheiraz as a major contributor to Agentlib, A. Aubel for the deformable human bodies, M. Clavien for the body texture design, C. Delon for the collaboration on motion flow schemes, and all members of LIG who either contributed to or used the work presented in this paper. The research was partly supported by the Swiss National Foundation for Scientific Research.

References

1. N. I. Badler, C. B. Philips and B. L. Webber. *Simulating Humans*. New York, Oxford University Press, ISBN 0-19-507359-2 (1993).
2. C. Bordeux, R. Boulic and D. Thalmann. "An efficient and flexible perception pipeline for autonomous agents", in *Proc. Eurographics '99, Milano, Italy* (1999).
3. A. Bruderlin and L. Williams. "Motion signal processing", in *Proc. SIGGRAPH'95*, pp. 97–104 (1995).
4. R. Boulic and O. Renault. "3D hierarchies for animation", in *New Trends in Animation and Visualization*, John Wiley and Sons, Chichester, UK, pp. 59–78 (1991).
5. R. Boulic and Thalmann D. "Combined direct and inverse kinematic control for articulated figure motion editing", *Computer Graphics Forum*, **2**(4) (October 1992).
6. R. Boulic, P. Bécheiraz, L. Emering and D. Thalmann. "Integration of motion control techniques for virtual human and avatar real-time animation", in *Proc. VRST'97*, pp. 111–118, ACM Press (September 1997).
7. R. Boulic, R. Mas and D. Thalmann. "Complex character positioning based on a compatible flow model of multiple supports", *IEEE Transactions in Visualization and Computer Graphics*, **3**(3) pp. 245–261 (1997).
8. D. E. Dow and S. K. Semwal. "Fast techniques for mixing and control of motion units for human animation", in *Proc. Pacific Graphics '94*, pp. 229–242, Beijing (1994).
9. L. Emering, R. Boulic and D. Thalmann. "Conferring human action recognition skills to life-like agents", *Applied Artificial Intelligence Journal*, Special Issue on 'Animated Interface Agents: Making them Intelligent' (1999).
10. J. P. Granieri, J. Crabtree and N. I. Badler. "Production and playback of human figure motion for 3D virtual environments", in *Proc. VRAIS '95*, pp. 127–135, IEEE (1995).
11. H. Ko and N. Badler. "Animating human locomotion in real-time using inverse dynamics, balance and comfort control", *IEEE Computer Graphics and Applications* **16**(2) pp. 50–59 (March 1996).
12. T. Molet, R. Boulic and D. Thalmann. *Human Motion Capture Driven by Orientation Measurements*. Presence, MIT Press, Vol. 8 (1999).
13. Franck Multon and Bruno Arnaldi. "Human motion coordination: example of a juggler", in *Proc. Computer Animation'99*, Geneva, Switzerland, pp. 148–159, ISBN 0-7695-0167-2 (1999).
14. K. Perlin and A. Goldberg. "Improv: a system for scripting interactive agents in virtual worlds", in *Proc. SIGGRAPH'96*, pp. 205–216 (1996).
15. C. Rose, B. Guenter, B. Bodenheimer and M. F. Cohen. "Efficient generation of motion transitions using spacetime constraints", in *Proc. SIGGRAPH'96*, pp. 147–154 (1996).
16. M. Unuma, K. Anjyo and R. Takeuchi. "Fourier principles for emotion-based human figure animation", in *Proc. SIGGRAPH'95*, pp. 91–108 (1995).
17. A. Witkin and Z. Popovic. "Motion warping", in *Proc. SIGGRAPH'95*, pp. 105–108 (1995).
18. G. Zachmann. "A language for describing behavior of and interaction with virtual worlds", in *Proc. VRST'96*, ISBN 0-89791-825-8, pp. 143–150, ACM Press (July 1996).
19. Xinmin Zhao and N. I. Badler. "Interactive body awareness", in *Computer Aided Design* **26**(12) pp. 861–867 (1994).