Scripting Interactive Physically-Based Motions with Relative Paths and Synchronization

Alexis Lamouret, Marie-Paule Cani

Scripting Interactive Physically-Based Motions with Relative Paths and Synchronization

Alexis Lamouret, Marie-Paule Gascuel
email: Alexis.Lamouret@imag.fr, Marie-Paule.Gascuel@imag.fr
iMAGIS[†]/ IMAG – INRIA
BP 53, F-38041 Grenoble cedex 09, France

Abstract

This paper presents a novel approach for facilitating the use of physically based models by animators. The idea is to let the user guide motion at a high level of control by giving approximate desired trajectories and synchronization constraints between the objects over time, while a simulation module computes the final motion, dealing with collision detection and response, and enhancing realism.

The objects, which are either isolated or components of an articulated structure, are guided through the specification of key-position and orientations, defined in a referential that can be fixed or relative to another object. The animation sequence is scripted by specifying a graph of synchronization constraints between objects over time. During the animation, objects automatically regulate their speed in order to meet these constraints.

Résumé

Cet article présente une nouvelle approche pour faciliter l'utilisation de modèles physiques par les graphistes. L'idée est de laisser l'utilisateur guider l'animation à un haut niveau de contrôle en donnant une approximation des trajectoires désirées et des contraintes de synchronisation temporelles entre les objets. Un module de simulation calcule ensuite le mouvement final, gérant les détections et réponses aux collisions, tout en améliorant le réalisme.

Les objets, isolés ou composants d'une structure articulée, sont guidés par la spécification de positions et orientations-clés, définies dans un repère fixe ou relatif à un autre objet. La séquence d'animation est définie par la spécification d'un graphe de contraintes de synchronisation entre les objets au cours du temps. Pendant l'animation, les objets régulent automatiquement leur vitesse pour satisfaire ces contraintes.

Keywords: animation, simulation, motion control, collisions.

[†]iMAGIS is a joint project between CNRS, INRIA, Institut National Polytechnique de Grenoble and Université Joseph Fourier.

1 Introduction

Finding a good compromise between simulation and control has became one of the main challenges in current Computer Animation research. Traditional animation systems offer precise control on motions and on deformations through key-frames or key-shapes, but provide no help at all for improving the realism of the resulting animation. In particular, there is no automatic process for detecting physically infeasible paths, avoiding inter-penetrations, computing deformations during contacts, or suggesting coherent speed variations. Conversely, pure physically-based simulators do not seem adequate to be used for production, due mainly to their lack of control. Generating motions by directly specifying applied forces and torques with time can be sufficient for simulating inanimate objects, but seems impossible when several interacting characters need to be synchronized according to a given script.

Could physically-based simulation become a tool for animators ? Probably yes if we can preserve a user-friendly interface, while offering the benefits of a simulation for enhancing realism and reducing the animator's work.

This paper presents a method for scripting and synchronizing physically-based motions. The animator guides an object by giving a set of successive key-positions and key-orientations, each of them being defined either in world-fixed coordinates or with respect to other moving objects. The final motion is generated during a forward simulation process, thus correcting the predefined path according to the object's physical parameters and to the events such as collisions detected during motion. Speed variations along the paths are computed by the simulation, which ensures their coherence with the path complexity and with the collisions, frictions, and contacts occurring during motion. The user can synchronize different objects (or different components of a complex structure) by specifying a graph of temporal events which links together some of the key-positions defined for each of them. This enables the design of complex scripts that include appointments (the objects

will adapt their speed if one of them is delayed), and that control the synchronization of the different motions.

The remainder of this paper is structured as follows: Section 2 briefly reviews related work. Section 3 describes a method, to appear in [13], for introducing some trajectory control in the physically-based animation of a given object. Some extensions to this original method are then presented. The method introduced in Section 4 allows to define some of the key-positions of an object relative to other moving solids. Section 5 develops an algorithm for adding synchronization constraints to link the motions of different objects. It ensures that specific key-positions on the different objects trajectories will be reached at the same time, even when one of the objects has been delayed by a collision, or by a more complex path. Section 6 explains how to use these temporal constraints for scripting and synchronizing complex motions in a full animation sequence. We conclude and discuss work in progress in Section 7.

2 Related Work

Constraints methods [2, 16, 9] and inverse dynamic techniques [12, 6, 16, 17] enable the specification of trajectories for some of the components of a complex structure, while leaving the system to animate the other degrees of freedom. They do not offer any help for improving the realism of the specified motions. In particular automatic collision processing (that would possibly include deviations from the desired motion) cannot take place for the components for which either position or orientation is specified.

Optimization techniques [3, 10, 22, 4, 15] provide a convenient interface for the user, who defines the animation through a set of key-frames. Physical models are used for finding correct interpolations between key-positions during a minimization process (the criteria most frequently used results in the minimization of the amount of energy needed to perform the motion). These methods however either require several forwards and backwards simulations over time, or the use of a global optimization process. Consequently, they do not produce animations at interactive rates. Moreover, most of these methods cannot be associated with automatic collision detection and response: contacts must be predefined by the user as extra constraints holding during specified time intervals.

Methods based on controllers have the advantage of being compatible with forward simulation techniques, which facilitate automatic collision detection and response. Both specialized controllers [18, 21, 5] and techniques for automatically generating an adequate controller [20, 14] have been previously presented. These approaches are very promising, but they are not aimed at producing complex motions that are synchronized according to a user-defined script.

We have recently proposed a method for using a generalized controller to keep an object close to a desired path, while providing some automatic collision processing [13]. The remainder of this paper first reviews this method and extends it to enable the definition of relative paths. Then, we present a new synchronization algorithm, and describe the way we use it for scripting an animation.

3 Combining Simulation and Trajectory Control

This section explains how to guide the trajectory of a single physically-based object (that can be a component of an articulated structure) while using the simulation for improving the user-defined path. The final trajectory will be smoothed according to the object mass and inertia, corrected with respect to the eventual collisions and contacts detected during motions, while deformations and adequate speed variations of the object will be produced. A more detailed description of this method can be found in [13].

The central idea is to let the user define a rough trajectory for the object, either for translation or for orientation (or both), by setting some key-frames. The user also specifies a *base distance*, d_{base}, that gives the precision with which the desired trajectory should be followed[1]. The system ensures that any point of the predefined trajectory will be reached with this precision. An actuator, either in translation or in rotation, is associated with the object. It will be used for generating a force (or a torque) pulling the object near the predefined path. The next section explains how we use a generalized Proportional-Derivative controller (PD-controller) for computing the actuator action.

3.1 Basic trajectory control

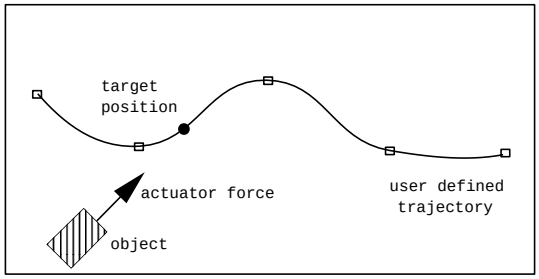

Figure 1: Actuated object guided along a path

Let us first consider an isolated object (with no articulation and that is not submitted to lasting external forces). The method for computing the actuator action is to perform a

[1] We use the quaternion representation of orientations to enable a simple definition of a "distance" between two orientations.

race between the object and a *target position* following the pre-defined path (see Figure 1). At each animation step:

1. The target moves forwards onto the path, trying to maintain the distance d_{base} with the object.
2. The actuator produces a force computed for covering a fixed portion of the distance to the target ;
3. This force is integrated in the object's equations of motion together with eventual reaction forces due to collisions. The object is moved to its next position.

The benefits of this algorithm are that the system object/target reaches a constant speed (controlled by the user) if nothing occurs, while a collision between the object and an obstacle produces some delay, without introducing any extra loss of precision in the final trajectory: if the object has been deviated by a collision, the target just waits for it.

We now detail the two first steps of this algorithm:

Computing the target motion

As mentioned before, the desired trajectory followed by the translation or orientation target has been pre-specified by the user through either a set of key-positions or key-orientations. Keys are interpolated by a spline curve (defined in quaternion space for orientations). The target position, an increasing parameter along this curve, is recomputed by binary search in order to maintain the base distance with this object. If the distance is already too high, the target just stays at the same location.

Computing the actuator action

Let α be the proportion of the distance between object and target that we want the object to cover during a fixed relaxation time[2] Δt. These two parameters control the ideal speed the system will reach when nothing happens.

The actuator force $\vec{F}$ or torque $\vec{\mathcal{T}}$ enabling the object to perform the desired motion is computed via a generalized version of a PD-controller:

$$\vec{F}(t) = 2m\,\frac{\alpha\left(\overrightarrow{x_{\text{target}}} - \vec{x}(t)\right)}{\Delta t^2} - 2m\frac{\beta\vec{v}(t)}{\Delta t} \qquad (1)$$

$$\vec{\mathcal{T}}(t) = \frac{2.J\left(\alpha\vec{W} - \beta\vec{\omega}(t)\right)}{\Delta t} - \vec{\omega}(t) \wedge J\vec{\omega}(t) \qquad (2)$$

$$\text{with } \tilde{W} = \frac{R_{\text{target}}R_{\text{object}}^{-1} - I}{\Delta t} \text{ and } \forall \vec{a}\ \tilde{W}\vec{a} = \vec{W} \wedge \vec{a}$$

where m is the object mass, J its inertia tensor, $\vec{x}$ its position, R its orientation and $\vec{v}$ and ω its linear and rotation speed vectors, respectively.

[2] We use here a fixed relaxation time because our animation system handles an adaptive time step. The way we compute the actuator action must not be perturbed by time-step changes due to other processes (such as the collision processing module).

Detailed comments on these two formulas can be find in [13]. We can stress that the control parameter β sets the smoothness/damping of the motion compared to the given trajectory:

- if $\beta = 0$, the object may oscillate indefinitely from one side to the other of the desired trajectory.
- if $\beta = 1$, there are no more oscillations, but the inertia is not taken into account in the generation of motion.
- In-between, we can tune β to obtain a more or less damped motion (see Figure 2).
- $\beta_{critical} = \sqrt{2\alpha}$ (obtained from the control theory) appears to be a good value for the quickest convergence.

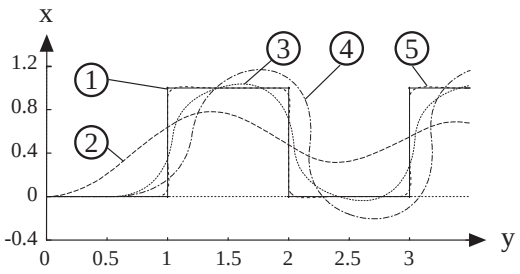

Figure 2: Tuning the actuator and target parameters. This figure shows various motions for a trajectory physically impossible (curve 1). Changing the base distance (curves 2 and 5) enables to choose how much we want to smooth the trajectory. Changing α within a certain range, and with the same β affects only the speed of the motion. Changing β (curves 2, 3 and 4) affects the damping of motion.

3.2 Articulated objects / Lasting external forces

The actuator force (or torque) described above is sufficient to deal with external forces applied only during a few time-steps, such as response to collision. Figure 3 shows how an object comes back near the desired path after a deviation due to an obstacle (our system uses the algorithm of [8] for detecting and responding to collisions).

However, the effect of lasting forces such as weight or connection forces between neighboring components in an articulated structure should not affect the precision with which the desired path is followed. To deal with these lasting external forces, we add a correction term to the force or torque computed in equations (1) or (2). This term is computed from an estimation of the sum of external actions applied on the object during the previous time-step (we observe the difference between expected position and reached position during the last time interval). We use a filter which smoothes the corrections terms over time, and produces the delay needed for maintaining an adequate deviation in the case of a sudden external action.

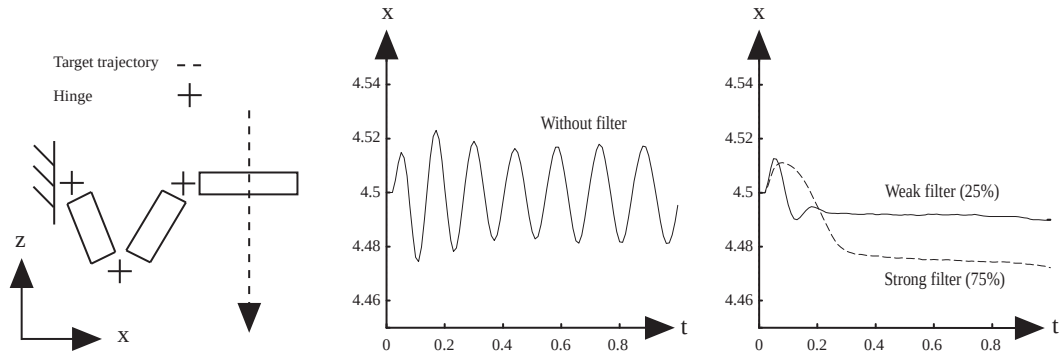

Figure 4: Filter effect on an articulated object. (a) motion without filtering the actuator forces. (b) motion with a filter.

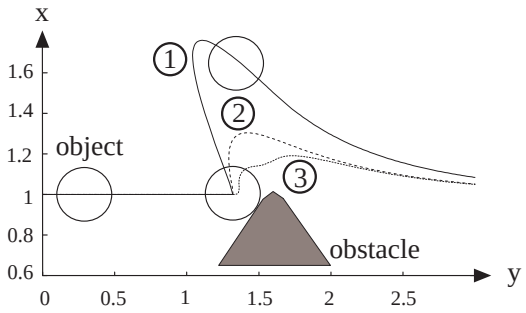

Figure 3: Deviation from the user-defined path due to a collision. In this figure an actuated object collides with a rigid obstacle. The trajectory given by the animator is a simple horizontal line passing through the obstacle. We see various behaviors depending on the mass (1 and 2) and the stiffness of the object (1 and 3). In (3) the object deforms and slides around the obstacle, while in (1) and (2) it bounces more or less violently.

Detailed equations for computing correction terms and filtering them are given in [13]. Figure 4 shows the results we obtain for a three-link articulated arm whose extremity is expected to move down on a vertical line (the method we use for animating articulated bodies is described in [9]). Filtering avoids the horizontal oscillations due to the action of neighboring components during the downward motion.

4 Defining Paths relative to another object

Specifying motion relative to other objects may be more useful for some applications than giving a desired location in world fixed coordinates. For instance, if two characters must shake hands, the precise location where they do it is not really important. This section proposes a first contribution to our original method which enables the user to define each key-positions or orientations in local coordinates with respect to another moving object.

The principle is the following. When the user specifies the set of keys defining the desired path for an actuated object, he associates each of them with either the world fixed coordinate system or with another object's local coordinate system.

The algorithm for computing the target motion (see Section 3.1) is modified as follows:

1. We select the control points (the keys) influencing the current target parameter on the spline curve (each curve segment depends on four control points). We add the control point that immediately follows, as the target may pass through a joint.
2. We convert the coordinates of these control points (if needed) into world fixed coordinates, taking the current locations and orientations of local coordinate systems into account.
   For most applications, the trajectory should not depend on the motion of a control point the target has already passed through, so a control point is "pinned" to its position in world-fixed coordinates as soon as the target reaches it (see Figure 5).
3. Then, we have the desired spline segments in world fixed coordinates, and we use the usual binary search method for finding the new target parameter.

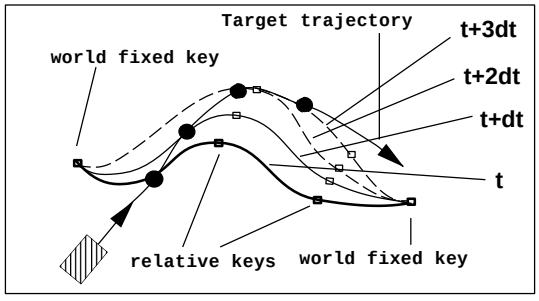

Figure 5: Target motion for a relative path

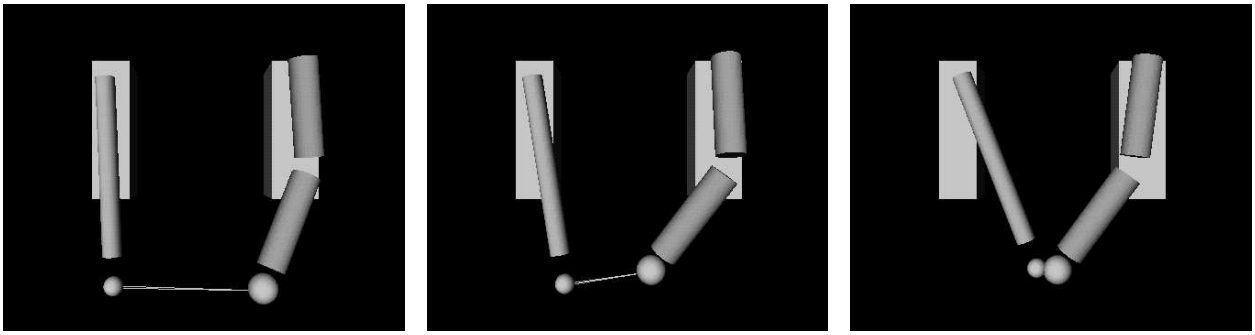

Figure 6: Two articulated arms shaking hands. The final position for each hand is defined in local coordinates relative to the other hand. The location where the shaking takes place is computed by the simulation, and depends on the strength, mass, inertia and articulations of each arm.

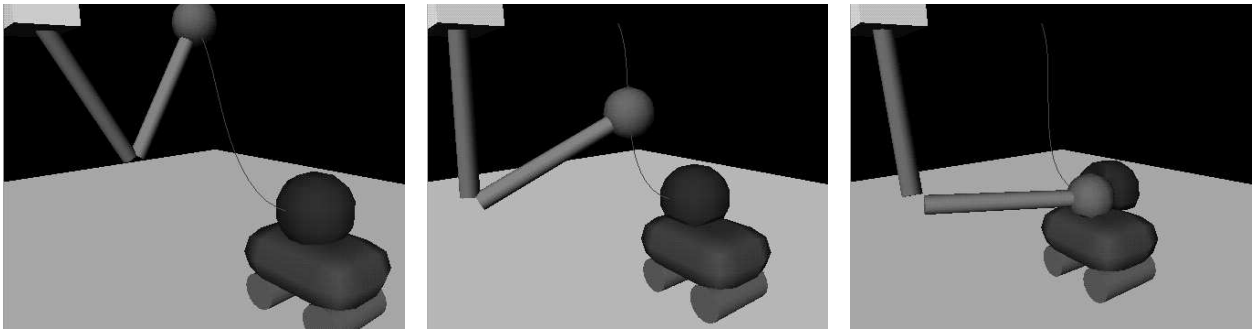

Figure 7: A three links articulated arm has to kick down a ball lying on a moving wagon. The desired final trajectory for the hand (position to reach and tangent to the spline curve) are defined in local coordinates relative to the ball. The contact between the ball and the wagon and the collision with the hand are automatically detected and treated by the simulation.

Since each object does not move too much during a single animation step (this condition is required for any system that performs step by step animation), the target trajectory stays smooth. Results are shown in Figures 6 and 7.

5 Synchronizing motions

In the method described above, we have chosen to compute the timing of motion during the simulation process rather than pre-specifying it. Indeed, the speed variations must depend on the path complexity, and on the events (collisions, contacts) occurring during motion. However, most animation sequences produced in Computer Graphics require careful motion synchronization between the different objects of the scene, or even between the different components of an articulated character. For instance, modeling walking requires careful synchronization of legs and arms motions.

Our second contribution to the original method consists in adding synchronization constraints which link the respective motions of various actuated objects. This section defines the method we use for fulfilling a single synchronization constraint between any number of objects. The application to scripting an animation sequence will be presented next.

A synchronization constraint is defined by selecting a set of positions (or orientations) on the different user-desired trajectories of the objects, and imposing the condition that all these positions will be reached at the same time by the object's respective targets. For instance, an appointment between different moving characters in a specific space location can be modeled this way. Another example is a character that must bend his knees while raising his arms. The synchronization is then realized between keys some of which are in translation, and others in rotation.

Performing motions under a synchronization constraint is done by regulating the average speed of the different targets in an auto-adaptive way.

Regulating Target Speed

Since constraints are defined between target positions, we are looking for a method for adjusting target speeds. Targets have to adapt their speed according to the events (ob-

jects delayed by collisions) that happen during the simulation, so we do not want to predefine these speeds using spline reparametrization methods [19, 1, 11]. We are rather looking for an adaptative way of regulating them.

The easiest way to synchronize a set of objects is to regulate all the speeds according to the slowest one. As the motion of the target depends upon the associated object, the slowest target will correspond to the slowest object, and we will definitely be able to slow down the others, while it would be uncertain to try to accelerate the slowest. Another point is that we want the target to recover its original ideal speed after the synchronization has been achieved (targets that have been slowed down should not stay slow, otherwise the current speed of each object would depend of every perturbation occurred since the beginning of the animation!). So, when no reduction of a target speed has been needed during several consecutive time intervals, or when the synchronization constraint has been reached, we increase the speed again.

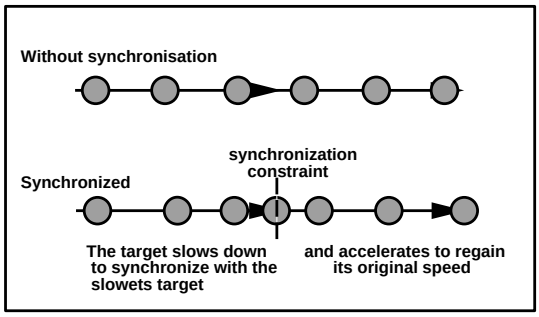

Figure 8: Target Speed Variations due to synchronization

Choice of the parameter to control: If there is no external event (no collision, etc), the average speed of an object and hence of its target, is $\alpha.d_{base}/\Delta t$ (see Section 3.1). To modify the speed of a target, we must change one of these parameters. Moreover, apart from the different speeds of objects relative to each other, we do not want to change any characteristic of the movement: neither the damping of the trajectory (based on the coefficient α/β^2), nor the precision with which the object will follow it (given by d_{base}). Modifications of the target speed will thus be achieved by regulating α while preserving α/β^2.

Synchronization algorithm: We are looking for a criterion that tells us which object is late or not with respect to a given synchronization constraint. This constraint states that the objects' targets should reach specified goal positions on their trajectories at the same time. In consequence, we use the relative distances between current target positions and goal positions along spline curves to define the relative "proximity" of objects with respect to the constraint. Since target speeds must be taken into account, we compare the distance the target covered in one time step to the total distance between current and goal positions.

In our implementation, we get direct computation of the *curvilinear abscissa s* (defined as the length of the spline between the first and the current point) by using reparametrized splines curves, for which the spline parameter equals an approximation of the abscissa[3].

Thus the time for a target τ to reach its goal is estimated as: $t_{estimated}(\tau) = dt.(s_{goal}(\tau) - s_{current}(\tau))/ds(\tau)$, where ds is the length covered in the previous timestep dt.

At each time step, the module that regulates the target speed operates as follows:

- Check if each target has reduced its speed recently. If not, it is allowed to accelerate by a certain percentage of its current speed.
- Evaluate the time each target would spend at its current speed to reach its desired goal, and keep the maximum as the desired time: $t_{desired} = max_\tau(t_{estimated}(\tau))$
- Reduce each target speed so that it will reach its goal in a time $t_{desired}$: $\alpha_{new}(\tau) = (t_{estimated}(\tau)/t_{desired}).\alpha_{old}(\tau)$.

Then we apply the method of Section 3.1 to move targets, compute actuators forces or torques and simulate all the objects. An example is shown in Figure 9.

6 Scripting an Animation Sequence

The method we have just presented for synchronizing motions can be used in a larger scale, in order to script an entire animation sequence.

In the previous section, we demonstrated how to organize coordination of several targets for a single synchronization constraint. More generally, if the user is animating *n* actuated objects, he/she may need to define several synchronization events to occur over time. Each of these events may link together several of the targets, and some succession order may need to be specified between some of them, but not necessarily between all.

The sequencing of synchronization constraints can then be defined by the user through a Petri network, as depicted in Figure 10. This network defines succession relationship between some of the constraints. At a given time, only a limited number of these constraints are activated, i.e. only those whose parents have already been reached (for instance constraint C_4 in the figure will be activated only when C_2 and C_3 have been reached).

[3]These re-parametrized splines are described in [7], and are computed by storing an array of values $ds = \sqrt{(dx/du)^2 + (dy/du)^2 + (dz/du)^2 du}$ (the new parametrization) between sample points distributed along the curve.

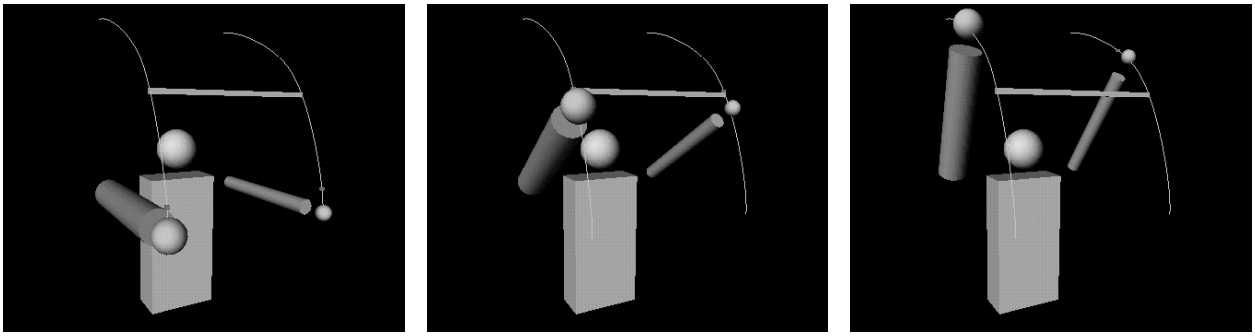

Figure 9: Two articulated arms, with different weights and inertia, must fulfill a synchronization constraint while going up: the intermediate key-positions are to be met at the same time. The results shows how the strongest arm decelerates to wait for the slowest, and how it comes back to its ideal speed after the appointment.

To implement this process, we store the synchronization constraints in a structure where each of them successively reaches three states:

- *Waiting*, which means that no effort is made to take this constraint into account, so targets linked only to it are not constrained yet.
- *Active*, which means that targets are trying to synchronize according to this constraint.
- *Done*, which means nothing is done anymore, so that the targets depending only on it return to their original state (unconstrained).

A "waiting" constraint becomes "active" as soon as all its parents are "done".

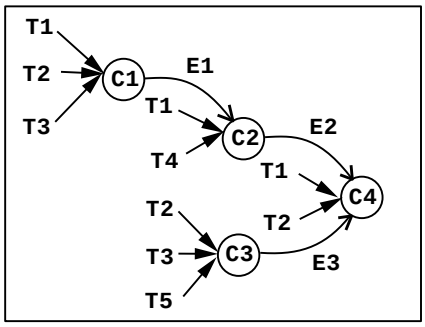

Figure 10: The graph of synchronization constraints. The C_i $(i = 1..3)$ represent the constraints, the T_j $(j = 1..5)$ are the targets involved for each of them, and the E_k $(i = 1..3)$ are the oriented edges, defined the user, which specified the desired sequencing between events (an edge from C_i to C_j means that the synchronization constraint C_i must take place before C_j).

Other algorithms can be chosen for activating constraints. For instance, it may be useful to take them into account earlier, by specifying that they will be considered as soon as a specified number of their parents are done. See Figure 11.

7 Conclusion

Designing an appealing key-frame animation requires a great amount of specialized knowledge from the animator, who spends a lot of time specifying motions and deformations. This very precise control is certainly desirable for animating the foreground, but reducing the time the animator spends on secondary motions would be good too.

In this paper a method was presented for scripting secondary motions with a high level of control. The user gives some key-positions and key-orientations for those of the objects (or components of an articulated structure) whose trajectory needs to be guided. These keys are either defined in world-fixed coordinates or relative to other objects. Spending a lot of time adjusting them is not necessary: inter-penetration avoidance (with subsequent deviation from the predefined path), deformations due to contacts, and adequate speed variations are automatically generated by the simulation. In addition, the key-features defined by the user can be linked together by temporal constraints. This enables the specification of a script that controls synchronization between the various elementary motions. For instance, appointments can be achieved even in a scene where objects are delayed by unexpected collisions.

The method consists of associating the objects with actuators capable of pulling them towards a target position that follows the desired path. The target auto-regulates its speed according to the object motion and to the synchronization constraints to be met. As the algorithm works during a single forward simulation over time, animations are generated at interactive rates. The resulting motion is as realistic or as unrealistic as the animator wants since it stays close to the predefined path, but not exactly on it if the path can be improved.

Work in progress includes attempts to combine our approach with a technique for computing muscle action for character animation. In this case, using muscles for gener-

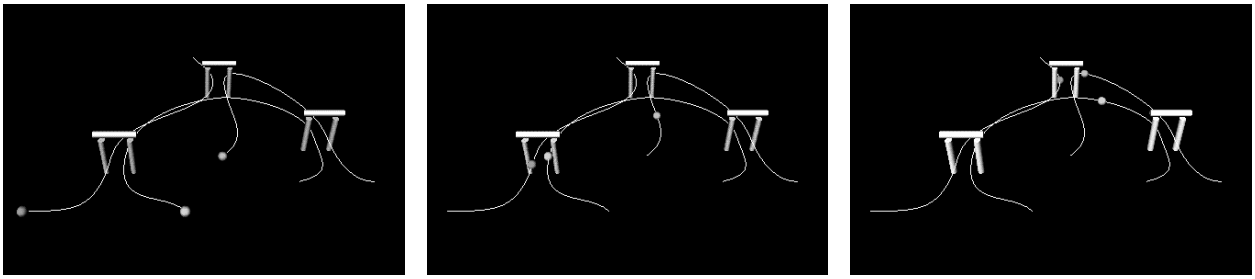

Figure 11: Three spheres representing simplified characters are synchronizing themselves according to a user-defined script: couples of spheres must pass together under each of the three arches. The automatic speed regulation makes the spheres accelerate or decelerate according to the next constraint and to the others motion.

ating torques at hinges as much as possible would be better than simply pulling objects as marionettes. Unfortunately, the exclusive use of muscles and reaction forces is an unsolved problem for complex aperiodic motions. We think that combining actuators with muscles would lead to an easier solution in this case.

Acknowledgements

Many thanks to Michiel Van de Panne for very interesting discussions, to Jean-Dominique Gascuel for his assistance, and to George Drettakis for carefully re-reading this paper.

References

[1] R. Bartels and I. Hardtke. Speed adjustement for key-frame interpolation. *Graphics Interface '89*, pages 14–19, May 1989.

[2] R. Barzel and A. Barr. A modeling system based on dynamic constraints. *Computer Graphics*, 22(4):179–188, August 1988.

[3] L. Shapiro Brotman and A. N. Netravali. Motion interpolation by optimal control. *Computer Graphics*, 22(4):309–315, August 1988.

[4] M. Cohen. Interactive spacetime control for animation. *Computer Graphics*, 26(2):293–302, July 1992.

[5] Y. Delnondedieu, A. Luciani, and C. Cadoz. Physical elementary component for modeling the sensori-motricity: the primary muscle. *Fourth Eurographics Animation and Simulation Workshop*, September 1993.

[6] G. Dumont. Animation de scènes tridimensionnelles : la mécanique des solides comme modèle de synthèse du mouvement. *Thèse de Doctorat*, Université de Rennes I, May 1990.

[7] J.D. Gascuel and C. Lyon. A new set of tools to describe and tune trajectories. In *Computer Animation*, May 1995.

[8] M.P Gascuel. An implicit formulation for precise contact modeling between flexible solids. *Computer Graphics*, pages 313–320, August 1993. Proceedings of SIGGRAPH'93.

[9] M.P. Gascuel and J.D. Gascuel. Displacement constraints for interactive modeling and animation of articulated structures. *The Visual Computer*, 10(4):191–204, March 1994.

[10] G. Hanotaux. Techniques de contrôle du mouvement pour l'animation. *Thèse de doctorat*, École Nationale Supérieure des Mines de Saint-Étienne, Université de Saint-Étienne, April 1993.

[11] G. Hanotaux and B. Peroche. Interactive control of interpolation for animation and modeling. *Graphics Interface '93*, pages 201–208, May 1993.

[12] P.M. Isaacs and M.F. Cohen. Controlling dynamic simulation with kinematic constraints, behavior functions and inverse dynamics. *Computer Graphics*, 21(4):215–224, July 1987.

[13] A. Lamouret, M.P. Gascuel, and J.D. Gascuel. Combining physically-based simulation of colliding objects with trajectory control. To appear in *The Journal of Visualization and Computer Animation*, 5:1–20, 1995.

[14] J.T Ngo and J. Marks. Spacetime constraints revisited. *Computer Graphics*, August 1993. Proceedings of SIGGRAPH'93.

[15] J.L Nougaret, B. Arnaldi, and R. Cozot. Optimal motion control using a wavelet network as a tunable deformation controller. In *5th Eurographics Workshop on Animation and Simulation*, September 1994.

[16] C. Van Overveld. An iterative approach to dynamic simulation of 3-D rigid-body motions for real-time interactive computer animation. *The Visual Computer*, 7:29–38, 1991.

[17] C. Van Overveld. Building blocks for goal-directed motion. *The Journal of Visualization and Computer Animation*, 4:233–250, 1993.

[18] M. Raibert and J. Hodgins. Animation of dynamic legged locomotion. *Computer Graphics*, 25(4):349–358, July 1991.

[19] S. Skeletee and N. Badler. Parametric keyframe interpolation incorporating kinetic adjustement and phrasing control. *Computer Graphics*, 19(3):255–262, July 1985.

[20] M. van de Panne and E. Fiume. Sensor-actuator networks. *Computer Graphics*, August 1993. Proceedings of SIGGRAPH'93.

[21] M. van de Panne, E. Fiume, and Z.G. Vranesic. Control techniques for physically-based animation. In *Third Eurographics Worshop on Animation and Simulation*, Cambridge, England, September 1992.

[22] A. Witkin and M. Kass. Spacetime constraints. *Computer Graphics*, 22(4):159–168, August 1988.