Combining Physically-Based Simulation of Colliding Objects with Trajectory Control

Alexis Lamouret
Marie-Paule Gascuel
Jean-Dominique Gascuel

iMAGIS */ IMAG
BP 53, F-38041 Grenoble cedex 09, France
email: Alexis.Lamouret@imag.fr,

July 6, 1995

Abstract

This paper describes a method that facilitates the use of physically based models by animators. The main point is to give the animator a familiar interface, while providing a simulation module which detects collisions thus enhancing realism.

The user gives a set of key-frames to guide motion, but does not have to address problems such as interpenetration avoidance, deformations due to collisions, or realism of motion. The simulator will *correct* the trajectories and compute deformations according to each object's physical properties (such as mass, inertia, stiffness) as well as the collisions and contacts automatically detected during motion. To achieve this, objects are provided with actuators capable of generating forces and torques computed via generalized Proportional-Derivative controllers. When deflected by external actions, actuated objects try to return to their initial path. Speed variations over time are computed during the simulation, and depend on the complexity of the paths, on the objects' models, and on the events such as collisions occurring during motion. In addition simulations are generated at interactive rates, even in the case of complex articulated objects. This facilitates the fine tuning of an animation sequence.

Keywords: animation, simulation, motion control, articulated objects, collisions.

1 Introduction

Over the past few years, physically-based models have demonstrated their usefulness for the automatic generation of realistic motion, including situations where bodies collide or remain in contact. However, these techniques are not yet widely used in industrial animation softwares. The main reason is the kind of interface they offer: the user of a simulation module has to provide a model for each object (with parameters such as mass, inertia, stiffness, etc), initial conditions, and a set of externally applied forces. This

*iMAGIS is a joint project of CNRS, INRIA, Institut National Polytechnique de Grenoble and Université Joseph Fourier.

interface is clearly inappropriate for an animator, who has a precise idea of the script he wants to follow, but has no way of finding out which forces should be applied to the models over time.

Perhaps we should address the problem in another way : what would an animator want ? It is desirable to take advantage of the help physically-based animation can offer, in particular for automatic collision detection and response, deformation of the objects, and realism of motion. And at the same time, keep the most frequently-used and most convenient interface : key-frames. In addition, throughout the process, the animator should be able to see the results of any parameter change in real time.

This paper presents an approach for combining the physical simulation of objects, including those in collision or contact situations, with trajectory control. The user still defines key-frames to guide motion, but does not have to consider the realism of motion, nor the collisions that will take place. The system *automatically corrects* this rough motion during an interactive simulation process: the final trajectory, the speed variations and the deformations of the objects over time will depend on the physical models and on events such as collisions and contacts detected during motion.

1.1 Related Work

Motion control has become a very important issue in physically-based animation. We briefly review the main approaches.

Constraint methods [BB88, Ove91b, GG94] and *inverse dynamic techniques* [IC87, Dum90, Ove91a, Ove93] offer direct control on some of the objects degrees of freedom. To animate a complex structure composed of different solids connected by hinges, the user can specify trajectories for some of the components, and let the system compute the other component's motions during a physically-based simulation process. Unfortunately, this is not sufficient for attaining our goal: The user has no help for improving the realism of the "leader components" motions, and in particular there is no automatic collision detection and response for these components.

Optimization techniques provide a convenient interface for the user, who defines a set of key-positions for the objects. Physical models are used for finding correct interpolations between these 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).

In [BN88], the method is restricted to motions expressed by linear differential equations, and to interpolation between an initial and a final configuration. The resolution requires two steps: backwards integration over time of a Ricatti system expressing the final conditions, and then forwards simulation from this result. An attempt for generalizing the method to non-linear systems is described in [Han93], but it requires iterating the process of forwards and backwards integrations over time until it converges. The approach is therefore compute-intensive, and very difficult to handle in complex situations.

In [WK88], discrete objects trajectories are improved during a series of iterations, under a set of space-time constraints defined by the user. Dynamic laws are included as extra constraints between position, velocity and acceleration parameters over time. In spite of the improvement to the method presented in [Coh92] – use of space-time windows to independently recompute parts of the motion without affecting everything – this technique does not produce animation sequences at interactive rates. In addition, the user has to pre-define interactions between objects as extra constraints that hold during specified time intervals. This can be time consuming (imagine, for instance, the complexity of defining constraints for a deformable wheel rolling and jumping on a bumpy floor). Moreover, the contact characteristics – deformations of the solids, contact duration – are then dictated by the user-defined constraints rather than computed from the physical parameters. This can adversely affect the realism of motion.

Methods based on controllers have the advantage of being compatible with forward simulation techniques, which facilitate the automatic collision detection and response. In many applications in Computer Graphics, a control module which senses some variables as input and outputs a set of actuator forces and torques is combined with a classical physically-based simulator.

Specialized controllers in [RH91, vdPFV92] are dedicated to legged locomotion, and address specific problems such as organizing leg actions and maintaining balance. [JLR93] uses controllers to maintain speed constraints on land vehicles interacting with various landscapes. [DLC93] uses "primary muscles" to generate motion of objects either controlled in real time by an operator, or moving autonomously, for a purpose of path planning.

Adequately tuning a controller for each model in the scene represents a great deal of work, and requires specialized knowledge. Recently, some approaches have been developed for automatically generating a controller for a given model, according to a criteria such as "cover the longest distance in a given time interval". In [vdPF93], a suitable controller is selected during a random generation process. A genetic algorithm is used to perform the same goal in [NM93]. These approaches are promising for enabling a system to automatically generate different kinds of locomotion, whatever the object may be. But they are not aimed at producing a specific motion that is close to a script or to key-frames defined by an animator.

1.2 Overview

This paper describes a general method for controlling physically-based models according to a predefined script, while preserving their ability to automatically detect and respond to collisions. The animator specifies key-frames to guide motion, but does not have to address problems such as realism of motion or interpenetration avoidance. The system corrects this rough motion during a forward simulation over time, performed at interactive rates.

To achieve this, objects are provided with actuators of limited strength, representing motors or muscles. At each animation step, actuator forces and torques are computed from generalized Proportional-Derivative (PD) controllers that use the difference between a target position and the current object position as their main input. Targets are located on interpolation curves defined from the animator's key-positions (see Figure 1). Their motion is closely related to the associated object motion. When an object is deflected by a collision, its target position may remain unchanged during a few time steps in order to avoid a loss of precision in the trajectory control.

Section 2 explains the basic principles of the approach by addressing the case of an isolated solid guided by key positions and orientations defined by an animator. Section 3 deals with more complex situations. The controlled objects can be connected to others

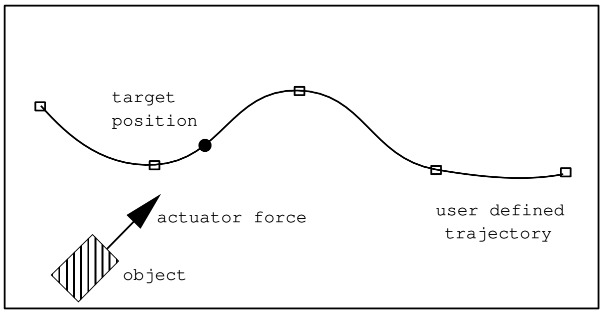


Figure 1: Actuated object guided along a path

by various kinds of hinges. Each object may be subject to a set of external actions, which may or may-not last over time. Section 4 describes results, and shows that the use of our method greatly simplifies the animator's task. Section 5 presents our conclusions and discusses works in progress.

2 Basic Trajectory Control

In the remainder of this paper we use the following notation, where rotation speed matrix and angular speed vector are linked together by: $\forall \vec{a} \in I\!R^3 \quad \tilde{\omega}.\vec{a} \ = \vec{\omega} \wedge \vec{a}$

Symbol	Description
m	mass
J	inertia tensor
$\vec{F}$	force
$\vec{\mathcal{T}}$	torque w.r.t. the center of mass
$\vec{v}$	linear speed vector
$\vec{\omega}$	angular speed vector
$\tilde{\omega}$	rotation speed matrix
$\vec{x}$	current translation vector
R	current orientation matrix

This section studies the case of a single isolated object, which is guided by animator-specified key-frames. More complex cases such as objects subject to lasting external forces and components of articulated structures will be treated in Section 3. According to the desired type of control, the object is provided with a translation actuator, a rotation actuator, or both. Each actuator is characterized by a limited strength: the force or torque it can generate is limited to a specified threshold.

The simulation method looks like a game of cat and mouse. At each time step, the object tries to move towards a target position located on the user-defined path. But this target moves, and its speed is closely related to the object motion. The next two sections detail the key points of the method: how to compute the actuator action from the current target position, and how to modify this position at each time step.

2.1 Computing Actuator Forces and Torques

Two kinds of actuators are available in our system. Translation actuators limited in strength generate a translation force $\vec{F}$ applied to the object's center of mass, verifying: $||\vec{F}|| < F_{\max}$. Rotation actuators generate a torque $\vec{\mathcal{T}}$ verifying: $||\vec{\mathcal{T}}|| < \mathcal{T}_{\max}$. The principle of the actuator action is to generate forces or torques to move the object towards an associated target position $\overrightarrow{x_{\text{target}}}$ or orientation R_{target}.

Suppose that there is no external action. According to the the Newton integration scheme described in Appendix A, the force to apply in order to reach the position $\vec{x}_{\text{target}}$ in one time step dt is:

$$\vec{F}(t) = 2m\,\frac{\overrightarrow{x_{\text{target}}} - \vec{x}(t)}{dt^2} - 2m\,\frac{\vec{v}(t)}{dt} \tag{1}$$

Unfortunately, this force is not suitable in our case, for many reasons. First of all, the velocity needed to reach the target in one time step would propel the object far past the target in subsequent time steps. Next, entirely suppressing the effect of the previous speed-vector would effectively suppress the inertia, as long as the maximum strength of the actuator is not reached. Finally, most simulation systems use an adaptive time step to regulate the simulation when problems such as overly deep interpenetrations between colliding objects occur. Using the current time step for computing the actuator action would cause unexpected acceleration or deceleration for each variation of dt.

Our approach is different. We compute a force so that the object will comply a given portion α of the distance to the target in a fixed relaxation time Δt (to avoid any problems with the adaptive time step). A parameter β is used for taking the speed of the object into account, without entirely compensating for it. This leads to the formula:

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

- If $\beta = 0$, the speed of the object is ignored during the computations. The object will then oscillate indefinitely around the target position, since there is no absorption. In practice, this limit motion is obtained only for a very small simulation time-step. Due to integration errors, the trajectory quickly diverges with usual time steps.
- $\beta = 1$ means we entirely correct effect of speed, hence suppressing inertia.
- In between we can tune β to obtain a more or less damped trajectory, as shown in Figure 2. The value of β therefore affects the rate of convergence to the target position. An analogy with the control theory, which will be further developped in Section 2.3, gives a theoretic value for β's critical damping value:

  $$\beta_{\text{critical}} = \sqrt{2\alpha}$$

  During our experiments, we have verified that this heuristic-value for β_{critical} is good for sufficiently small time steps.

In practice, an animator may not want to directly give a value for β, but rather to tune the system between more or less damped motions. The animator thus gives an "elasticity coefficient" $B \in [0, 1/\beta_{\text{critical}}]$, with default value at 1, and we use $\beta = B.\beta_{\text{critical}}$. When a high convergence speed is desired, the choice of a B lightly smaller than 1, and depending on the convergence criterion is recommanded. For instance, experiments show that if we want to converge quickly while staying within one tenth of the initial distance from the trajectory, a good choice is $B = 0.7$.

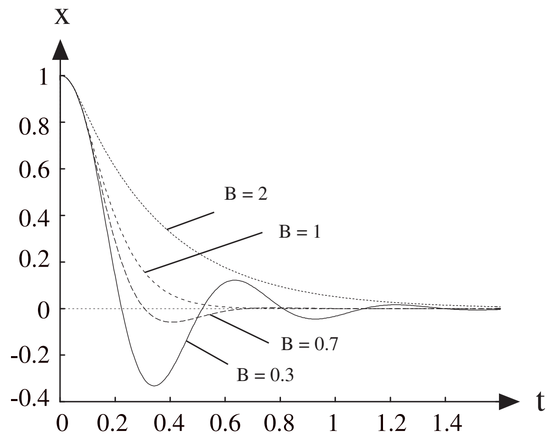


Figure 2: Overdamped, underdamped and critical curves as a function of B, for a dynamic point trying to reach a stationary target position, with $\alpha = 0.1$.

The computations for a rotation actuator are similar (see Appendix A for the Newton integration scheme for rotations). We compute the torque which would rotate the object by a percentage α towards a target orientation during the relaxation time Δt, with a compensation of rotation speed regulated by a parameter β:

$$\vec{\mathcal{T}}(t) = \frac{2.J\left(\alpha\vec{W} - \beta\vec{\omega}(t)\right)}{\Delta t} - \vec{\omega}(t) \wedge J\vec{\omega}(t), \tag{3}$$

$$\text{where} \quad \tilde{W} = \frac{R_{\text{target}} R_{\text{object}}^{-1} - I}{\Delta t}. \tag{4}$$

Let us now explain how target positions and target orientations are computed from the user-defined key-frames and from the associated object motion.

2.2 Computing a New Target Position

As emphasized in the introduction, the animator roughly specifies the trajectory of actuated objects through a set of key positions and/or orientations. This data is first interpolated to get a continuous trajectory for the target. Methods for interpolating between positions and orientations (converted into quaternions) can be found in [Duf86]. Target values for computing both forces or torques applied by actuators at each time step are represented by current scalar parameters along spline curves.

Moving the target position at a constant speed along the curve would not be a good solution. Indeed, the actuated object can be slowed down by a collision, and will not follow the predefined path with sufficient precision if the target has run too far ahead.

Moreover, the actuator callback force (or torque) would become too large in such a case, and therefore produce unwanted speed variations for the object.

We instead compute the next target position from a *base distance* specified by the user. The system ensures that any point of the predefined path will be at most at the base distance to the object final trajectory.

The idea consists in trying to always keep the target at the base distance from the object, with the constraint that the target never moves backwards along its path. Then, if an object is pushed backwards, its target position remains the same during a few time steps. If the object goes fast, so will the target.

To achieve this, we define a distance function to be used between the object and the target. For translation, it is simply the metric distance, wereas for orientations we use quaternions [Sho85] as follows: The distance between two quaternions q_a and q_b is defined by:

$$d(q_a, q_b) = angle(q_a.q_b^{-1}) = 2acos(Re(q_a.q_b^{-1}))$$

At each time step, we consider the last position of the target and then look for the first higher parameter value for which the distance is reached (this is done by binary search).

2.3 Comparison with Proportional Derivative (PD) Controllers

The computations used in Section 2.1 for the actuator forces and torques are quite similar to the action of a Proportional Derivative controller [Sev89].

For instance in the case of forces, a PD controller sensing an error $e = \overrightarrow{x_{\text{target}}} - \vec{x}(t)$ and its derivative $\dot{e} = \overrightarrow{v_{\text{target}}} - \vec{v}(t)$ would produce a force $\vec{F} = \alpha e + \beta \dot{e}$ aimed at minimizing e. This force, which differs from the force of equation (2) by addition of the term $\beta \overrightarrow{v_{\text{target}}}$, would tend to make the object reach the target and regulate its speed based on the target speed.

This formula would not work for our application. Our aim is to make the target regulate its speed on the object motion, rather than the contrary. We do this by always leaving the target at the same distance from the object, with the result that the object-target system will reach a constant speed on a straight line when no other force is observed. Conversly, if we used the force formula given by the PD controller, the object would accelerate at each time step to try to reach the target, so the target would also accelerate at each time step, to keep its distance.

Consequently, our method can be seen as a generalized version of a PD controller, adequately adapted to our specific goal.

3 Generalization to Complex Situations

The method just described enables the combination of physically-based simulation of an isolated solid with trajectory control. The actuator force described above is still sufficient if an external force is applied to the object during a few time steps. For instance, when a collision with another body is detected, the controlled object is deflected from its trajectory, but comes back closer to the predefined path after a while, due to the action of its actuators.

However, most of the objects used in Computer Animation are subject to continuous external actions such as gravity. Some external actions may be very complex, such as the interactions between neighboring components in an articulated structure. Hence, the set of externally applied forces and torques must be adequately taken into account while controlling the motion.

This Section first describes a method for animating articulated structures that is well suited to our approach for control. Then, the control algorithm presented earlier is adapted to general situations, where various kinds of external actions may be applied on each solid.

3.1 Animation of Complex Structures

We are looking for a convenient method for animating articulated bodies, and more generally constrained structures. This method must be well suited to the approach for trajectory control we have developed. In particular, the user must be free to independently associate actuators to only some of the component of a structure, the other ones being animated by another algorithm. This forbids the use of approaches based on the expression of laws of dynamic in the parameter space of articulated objects. Another important criteria is the efficiency of the whole animation system. As we said previously, the simulations should be computed at interactive rates. To achieve this, we use the "*displacement constraints*" method detailed in [GG94]. We present here only a brief review of the technique.

Complex articulated structures are built from independent solid components connected together by geometric constraints. The user is free to choose the number of degrees of freedom in rotation and in translation at hinges, and to specify angular or linear constraints on motion. The graph of constrained objects may contain any number of closed loops. A technique for the automatic construction of valid initial positions is provided together with the animation method.

At each computation step, the solids move first as if they were independent according to Newton integration scheme detailed in Appendix A. Then, constraints are met through *iterative tunings in displacements.*

Displacements associated with a single constraint

Let us first present the method in the case of a single constraint between two solids. We will explain how to combine the action of multiple constraints later on.

A "point to point" constraint between two solids S_1 and S_2 creates a joint with 3 degrees of freedom in rotation by making two points, P_1 of S_1 and P_2 of S_2, coincide throughout movement. See Figure 3.

Meeting a "point to point" constraint by translating a solid without any rotation is always possible, but would most of the time produce unrealistic behaviors. To find the adequate proportion between rotation and translation we make an analogy with the action of some hypothetical rubber-band that would have maintained the constraint during a time step.

This leads to the following computation scheme, where the proportion between rotation and translation, and the relative displacements of the two solids are consistent with their parameters of mass and inertia :

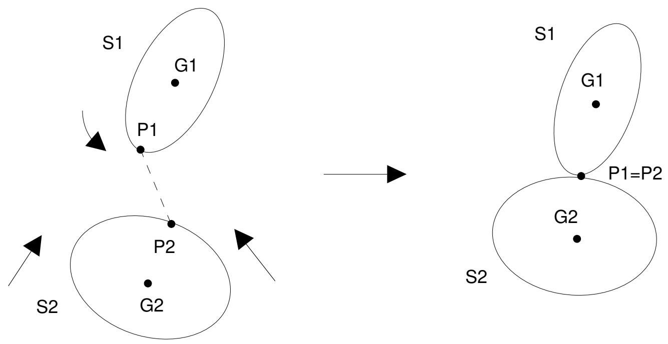


Figure 3: A "point to point" constraint between two solids.

1. Apply respectively to S_1 and S_2 the small rotations given by the rotation vectors:
$$\begin{cases} \overrightarrow{\Delta R}_1 = \frac{m_1 m_2}{m_1+m_2} J_1^{-1} \left(\overrightarrow{G_1 P_1} \wedge \overrightarrow{P_1 P_2}\right) \\ \overrightarrow{\Delta R}_2 = \frac{m_1 m_2}{m_1+m_2} J_2^{-1} \left(\overrightarrow{G_2 P_2} \wedge \overrightarrow{P_2 P_1}\right) \end{cases} \quad (5)$$
where m_i are the masses and J_i the inertia tensors of the two solids.
2. Then, from the positions P_1', P_2' of the points *after rotation*, compute and apply the small translations exactly satisfying the constraint:
$$\begin{cases} \overrightarrow{\Delta x}_1 = \frac{m_2}{m_1+m_2} \overrightarrow{P_1' P_2'} \\ \overrightarrow{\Delta x}_2 = \frac{m_1}{m_1+m_2} \overrightarrow{P_2' P_1'} \end{cases} \quad (6)$$

The method can be extended in order to offer some translational degrees of freedom at hinges, possibly limited in scope. To constrain the movement of P_1 in any sub-area defined w.r.t S_2, P_2 is replaced in equations (5) and (6) by the point P_2' of the chosen domain which is the closest from P_1. "Point to segment", "point to curve", "point to surface", "point in sphere" are examples of simple constraint concepts that may be useful.

In the method presented so far, each constraint leaves 3 degrees of freedom in rotation between the two solids. Restricting rotations at hinges may be useful, and can be done by the same type of approach. Once a parametrization (such as quaternions) has been chosen for orientations, one can compute the smallest rotation to bring the angular "distance" between two objects back to some allowed freedom space. This rotation is split between the solids according to their respective inertia for this particular axis.

Combining displacements due to individual constraints

In practice, several constraints can simultaneously be applied to a solid, so rotations and translations due to each constraint must be combined together. A sum of the displacements computed for each individual constraint would conserve first order momenta, but would lead to divergences in some particular cases, as shown in [GG94].

This problem can be solved by weighting the displacements affected to the solids before summing them. In order to obtain the conservation of first order momenta, the same weighting factor must be used for couples of displacements due to the same constraint. If S_i

and S_j are two objects linked by a given constraint, we weight the associated displacements by: $\frac{1}{\max(n_i, n_j)}$, where n_i and n_j are the numbers of constraints respectively applied on each solids.

A series of iterations is needed to fulfill the constraints as soon as more than one constraint per solid are applied. This is done through an iterative process which stops when all the resulting corrections are smaller than a specified threshold, or when a maximal number of iterations is reached. Limiting the number of iterations avoids deadlocks when the system is over constrained. The equations used lead to very fast convergence rates. Usually a few iterations are sufficient, even when the graph of constrained objects contains closed loops (see [GG94]).

Correcting the kinematic behavior of constrained solids

The corrections due to constraints must be taken into account in the kinematics of movement, as if we had added constraint forces. Once a position of the solids that meets the constraints have been found, we adjust their linear and angular speeds by considering the positions and orientations they have effectively reached during a time step:

$$\begin{aligned} \vec{v}(t+dt) &= 2(\vec{x}(t+dt) - \vec{x}(t))/dt - \vec{v}(t) \\ \tilde{\omega}(t+dt) &= 2\left(R(t+dt)R^{-1}(t) - I\right)/dt - \tilde{\omega}(t) \end{aligned}$$

In conclusion, this animation algorithm for complex structures is simple and efficient. It basically gives the same effect than the computation of a set of constraint forces at hinges[1]. However, tuning displacements is more direct, as there is no need of integrating constraint forces during the series of iterations needed for fulfilling constraints. Quite general constrained structures can be built and animated with this method, with no restriction on the particular algorithm used for computing the independent motions of each object. Therefore, this approach seems very well suited to our trajectory control module.

3.2 Control of Objects Submitted to Any External Action

As emphasized earlier, the study of isolated objects is very restrictive. Most objects are subject to various kinds of external actions during an animation sequence. Some of these actions result in continuous external forces such as gravity or fluid friction, while others only last for a few time steps such as most collision forces. Objects which are components of articulated structures are subject to highly varying small interactions maintaining constraints with their neighbors.

To keep the associated object close to the predefined-path, the control module must take the effect of external actions into account in the computation of actuator forces and torques. Typically, an object subject to a constant gravity force must compensate for it with its actuator action, otherwise it will remain far below the goal trajectory, as shown in Figure 4.

[1] although it is not proved that the methods are equivalent

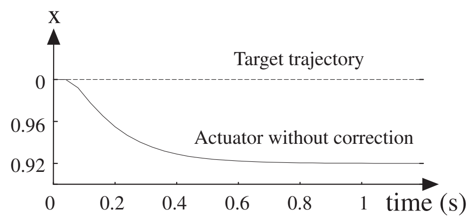


Figure 4: Effect of gravity for a ball following a target on a straight line ($\alpha = 0.1, \beta = 0.6$)

However, when an animator introduces animation effects such as an object bouncing on obstacles due to gravity and to response forces, he does not want the controle module to instantaneously hide these effects. An object should not react too quickly to an externally applied force, but still be deflected by sudden collisions. After a reasonable period of adaptation, the object should tend to compensate for an externally applied force which remains constant over time. In particular, an object subject to gravity should be controllable.

First of all, we need to provide the control module with an adequate sensor, able to apprehend the effect of external actions. Directly sensing the value of external forces would not be a good solution. Due to our approach for animating complex structures, some of the external actions may be directly expressed by displacements rather than by forces. So we use the object current position, orientation and speed, as entries for the controller.

At each time step, the controller needs to approximate the sum of external actions during the last time interval. This can be done by comparing the object current location with the "predicted position" it was trying to reach at the last time step. In practice, we prefer to use speeds rather than positions and orientations for approximating the sum of external actions. This reduces numerical errors, as we avoid the approximation resulting from the conversion of a rotation vector into a matrix. The evaluated external forces and torques are:

$$\vec{F}_{\text{ext}}(t-dt) \;=\; m\frac{\vec{v}(t)-\vec{v}(t-dt)}{dt} - \vec{F}_{\text{actuator}}(t-dt) \tag{7}$$

and similarly for torques:

$$\vec{\mathcal{T}}_{\text{ext}}(t-dt) \;=\; J\left(\frac{\vec{\omega}(t)-\vec{\omega}(t-dt)}{dt}\right) + \vec{\omega}(t) \wedge J\vec{\omega}(t) - \vec{\mathcal{T}}_{\text{actuator}}(t-dt) \tag{8}$$

Overcoming the action of these external forces and torques could be done by adding the opposite of $\vec{F}_{\text{ext}}$ or $\vec{\mathcal{T}}_{\text{ext}}$ to the actuator action. But this would not be a good idea for several reasons. As said before, we do not want to correct too quickly the action of external forces. Another point is that articulated objects are often subject to highly varying small displacements for maintaining constraints, due to very complex interaction forces between neighboring components. Using – with a delay – the opposite of these forces can produce an amplification phenomenon which leads to oscillations around the goal trajectory. Such a situation is depicted in Figure 5 (a), which represent the horizontal oscillations generated during the controlled motion of a three-link articulated arm whose extremity should move down on a vertical line.

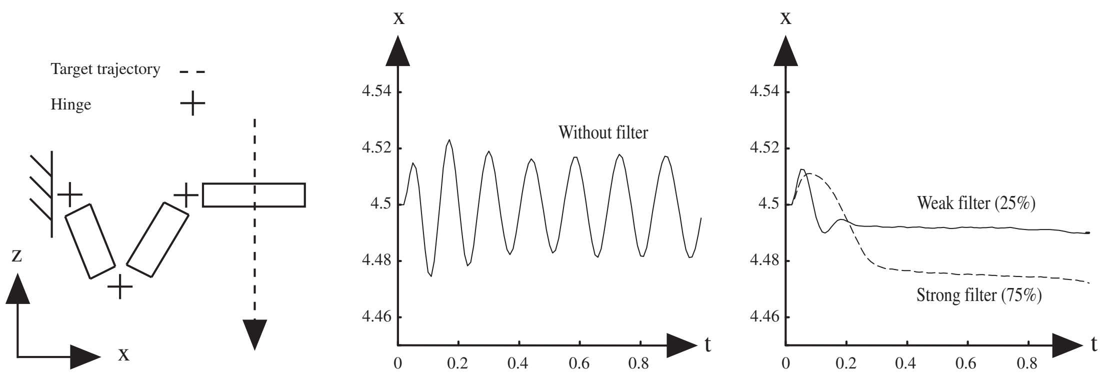


Figure 5: Filter effect on an articulated object.
(a) motion without filtering the actuator forces.
(b) motion with a filter.

The solution consists in using a filter for computing actuator forces and torques from approximated external actions. The first benefits of this filter is the production of softly varying actuator forces (Figure 5 (b) shows that the oscillations disappear). It also provides a convenient parameter for regulating the delay between external action and response of the control module.

In order to offer an amount of filtering independent of the adaptive time steps used during the simulation process, the user specifies a parameter γ representing the amount of filtering per second. The actuator correction force or torque is then computed from the approximated external action and the previous values of corrections at $t - dt$ by using the filter parameter γdt:

$$\vec{F}_{\text{corr}}(t) = -\gamma dt \vec{F}_{\text{ext}}(t - dt) + (1 - \gamma dt)\vec{F}_{\text{corr}}(t - dt) \quad (9)$$

$$\vec{\mathcal{T}}_{\text{corr}}(t) = -\gamma dt \vec{\mathcal{T}}_{\text{ext}}(t - dt) + (1 - \gamma dt)\vec{\mathcal{T}}_{\text{corr}}(t - dt) \quad (10)$$

Finally, the complete algorithm for computing actuator forces and torques is:

1. Compute the main component of the actuator action from equations (2) and (4).

2. Compensate from observed external actions by adding the correction terms given by equations (9) and (10), where $\vec{F}_{\text{ext}}$ and $\vec{\mathcal{T}}_{\text{ext}}$ are computed from (7) and (8).

3. If necessary, truncate the actuator forces and torques according to the maximal available strength.

4 Results

4.1 Animation Background

We have implemented the trajectory control method within the framework of the animation system described in [Gas93, GG94]. Each solid, either rigid or elastic, is structured in:

- A rigid component, which can be stationary, follow a predefined path, or be dynamically animated from rigid body equations of motion of Appendix A. When the object is composed of elastic material, the mass and inertia tensor of the rigid layer are computed from the object rest shape.

- A coating layer at rest with respect to the rigid component, which represents the material constituting the object. Coating layers give the geometry of the object surface. Our model for elastic coatings, based on an implicit formulation, generates exact contact surfaces between colliding objects and provides precise evaluation of response forces.

The animation algorithm is composed of the control module previously described, and a simulation module consisting of three steps: Integration of the individual equations of motion for each solid rigid component, treatment of constraints through iterative tunings in displacement, and collision detection and response computed from the coating layers. The whole simulation process is based on an adaptive time step: if the time step used is too large with respect to the objects motion, divergences could appear in the constraint processing and/or in the collision module. When this problem is detected, the system goes back in time with a smaller time step as depicted in Figure 6. When no more exception message are generated during a fixed number of simulation steps, the system increases dt, if it does not exceed the display rate.

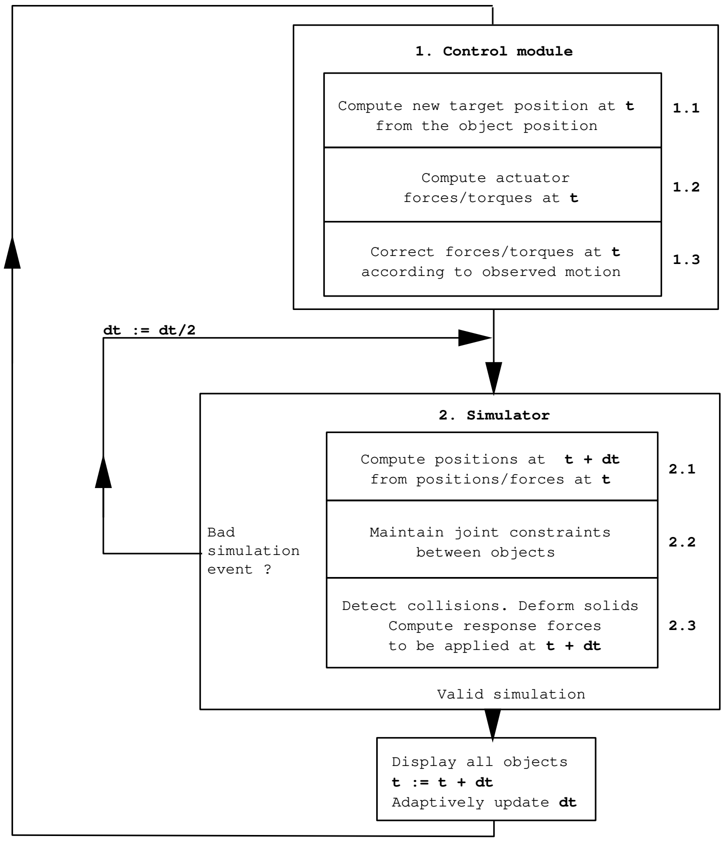
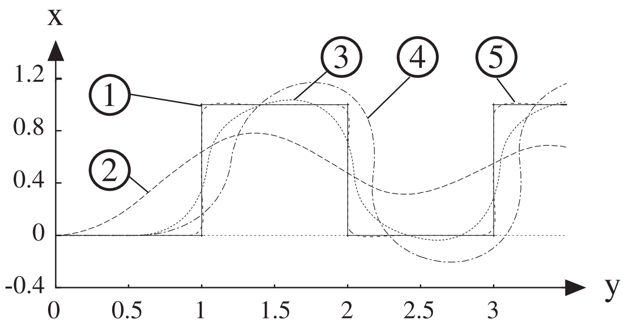


Figure 6: The animation algorithm

4.2 Examples in 2-D

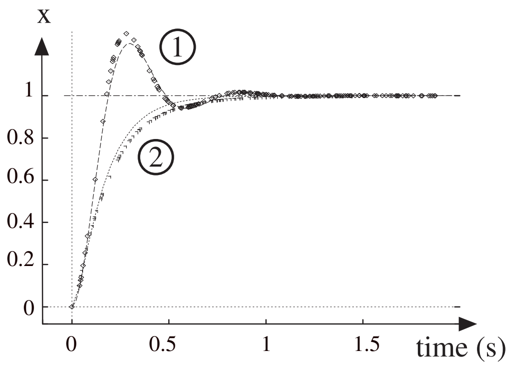
Before showing animation sequences with a complex environment, let us present some results in 2-D to show how the tuning of the different parameters enables to obtain various motions.

Smoothing a rough trajectory



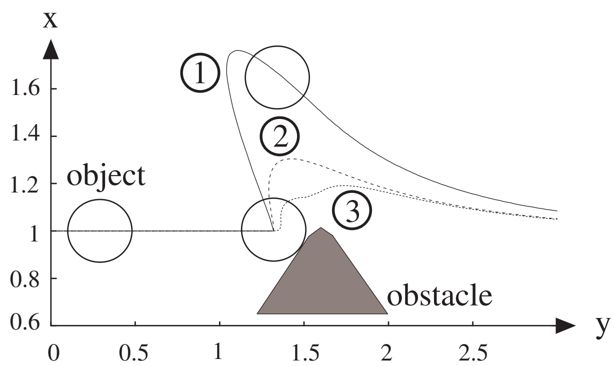
This figure shows various motions for a trajectory physically impossible (curve 1). Changing the base distance (curves 2 and 5) enables to choose how much we want to smooth the trajectory. Changing α within a certain range, and with the same B affects only the speed of the motion. Changing B (curves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

Validation of the method with random time step



The method will be used in an animation system with an adaptative time step. It is therefore important to have a good tolerance to random variations of the time step. This figure shows the motion of the object for a stationary target position, for a constant time step (curves) and for a random time step (dots). We see that the result is quite the same in both cases ($\alpha = 0.1$: $B = 0.5$ in (1), $B = 1.2$ in (2)).

Samples of collisions



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 behaviours 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.

4.3 Simply Implicit

The animation "Simply Implicit" shows the results of our trajectory control method in the case of *a simple object controlled in translation*: an elastic ball subject to gravity and to interactions (that include elastic response and friction) with other solids. The animation was designed for the presentation of the implicit elastic material developed in [Gas93].

The script drawn by an animator (Figure 7) shows the ball bouncing into a nursery, and colliding with a set of flexible toys until it succeeds in knocking them off a shelf.

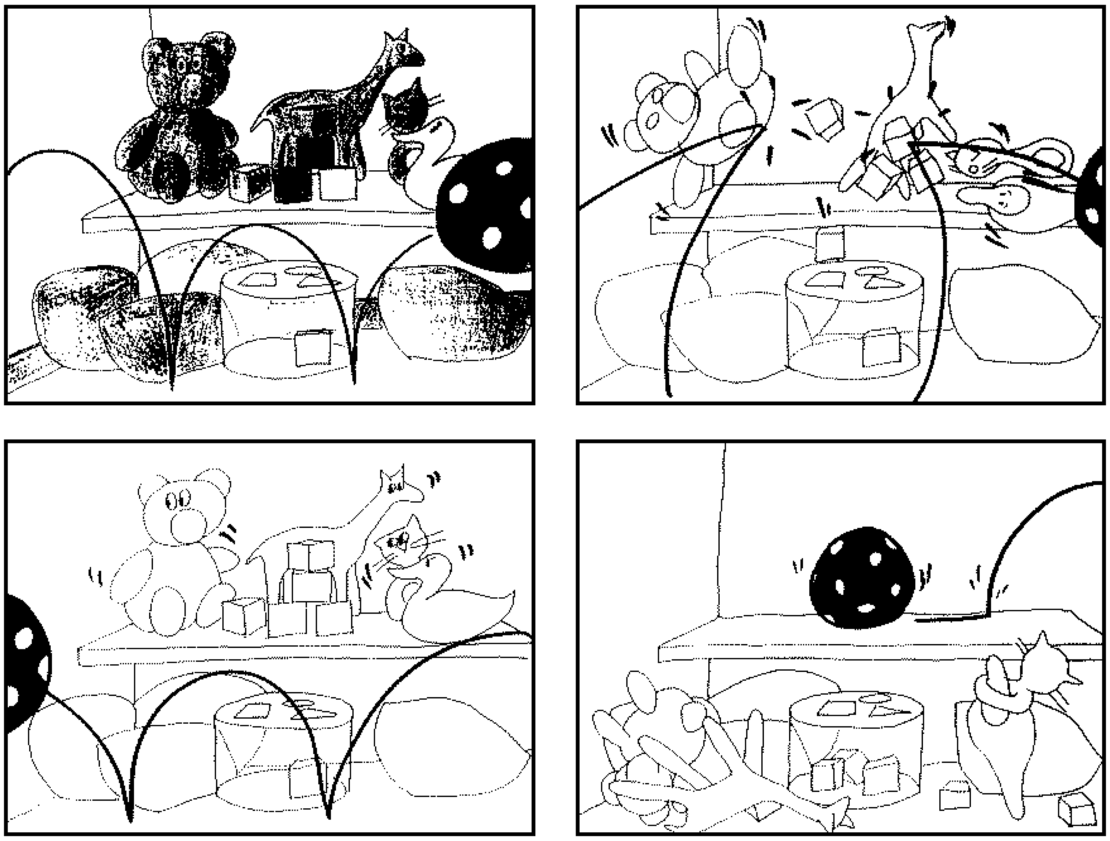
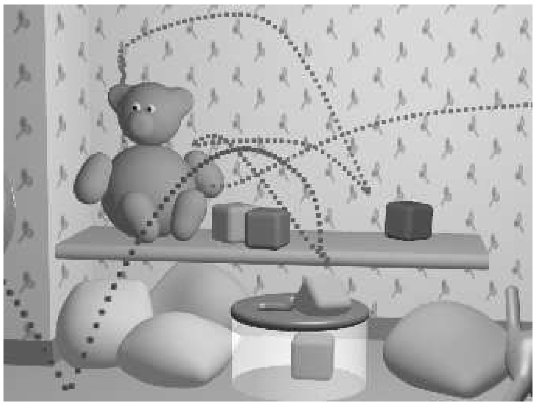
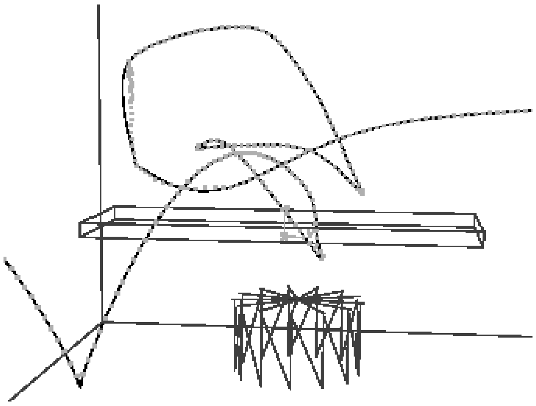


Figure 7: Script drawn by an animator

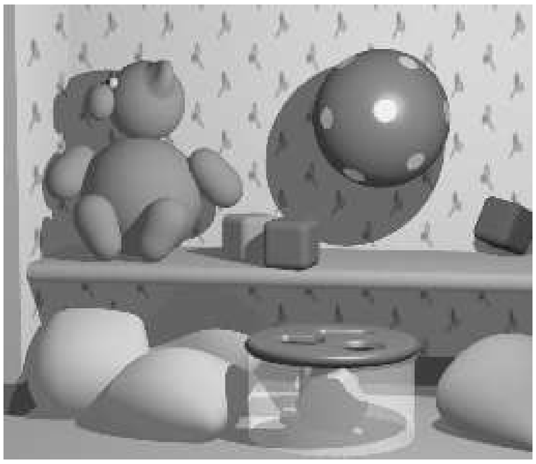
To obtain the final animation, the ball model is provided with a translation actuator. No trajectory control is applied to the other objects, composed of elastic implicit material, and animated by direct use of the simulation module. The animator defines a rough trajectory for the ball by giving some key positions. During this process, he does not have to consider at all the physical properties of the ball (mass, inertia tensor, stiffness coefficient), nor the numerous collisions and contacts that will take place during motion. The system automatically *corrects* the user trajectory during the simulation, according to all these parameters (see Figure 8). In particular, the rotation of the ball is entirely due to the action of friction forces during collision processing.



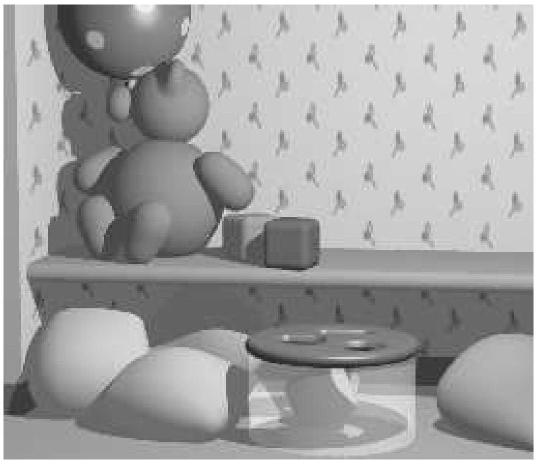
a: Trajectory of the ball in the complete environment.



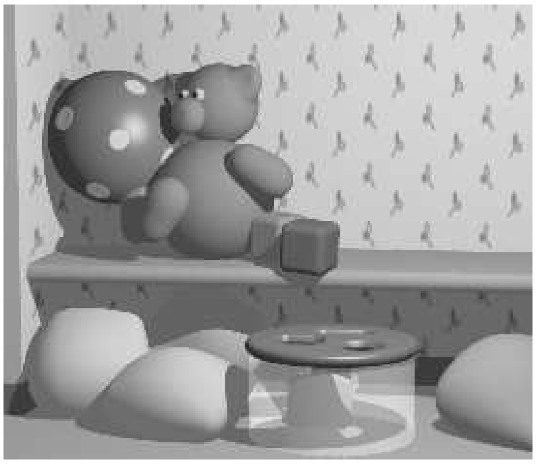
b: Trajectories of the ball and its target. (1), (2) and (3) show corrections on the trajectory due to collisions.



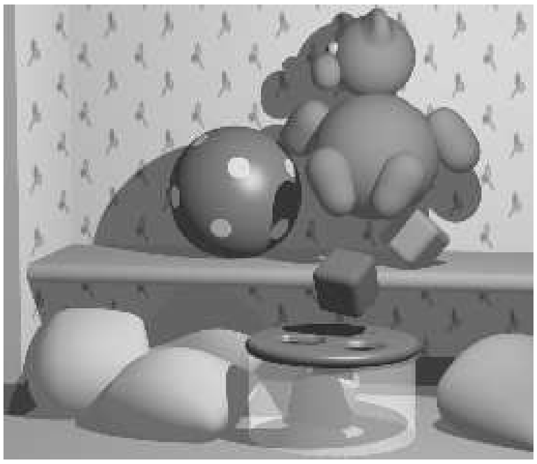
c: The ball enters the scene...



d: ...collides with the wall and the bear...



e: ...pushes the bear off the wall..



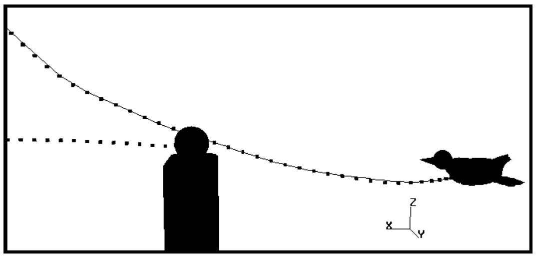
f: ...and starts rolling on the shelf.

Figure 8: Simply Implicit

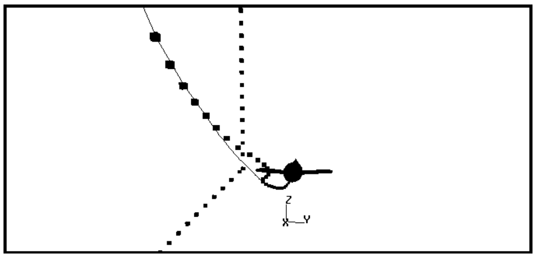
4.4 Bird Flight

Figure 9 shows experiments for the motion of *an articulated object controlled with four actuators*. The object (a bird) is composed of a rigid body and of two rigid wings, connected to the body by hinges with angle constraints. The body is provided with translation and rotation actuators, and the wings with rotation actuators only. Thanks to the "displacement-constraints" module maintaining joints constraints between the body and the wings, no translation actuator is needed for the wings. Up and down key-rotations are given for the wing orientations in order to simulate the bird flight.

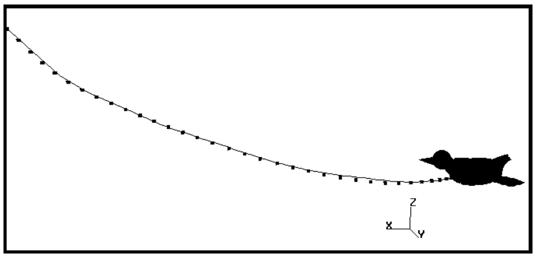
The different experiments show how the bird trajectory is deflected by various collisions (The figure shows only the trajectory of the body in translation).



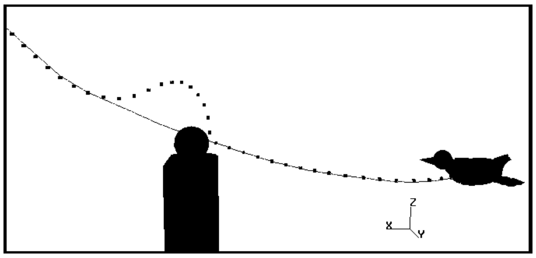
3.1 The bird alone: it is just slightly deviated because of inertia.



3.2 The bird and a light ball: only the ball is affected by the impact.



3.3 The ball and a very heavy ball: the bird is deviated while the ball remains still.



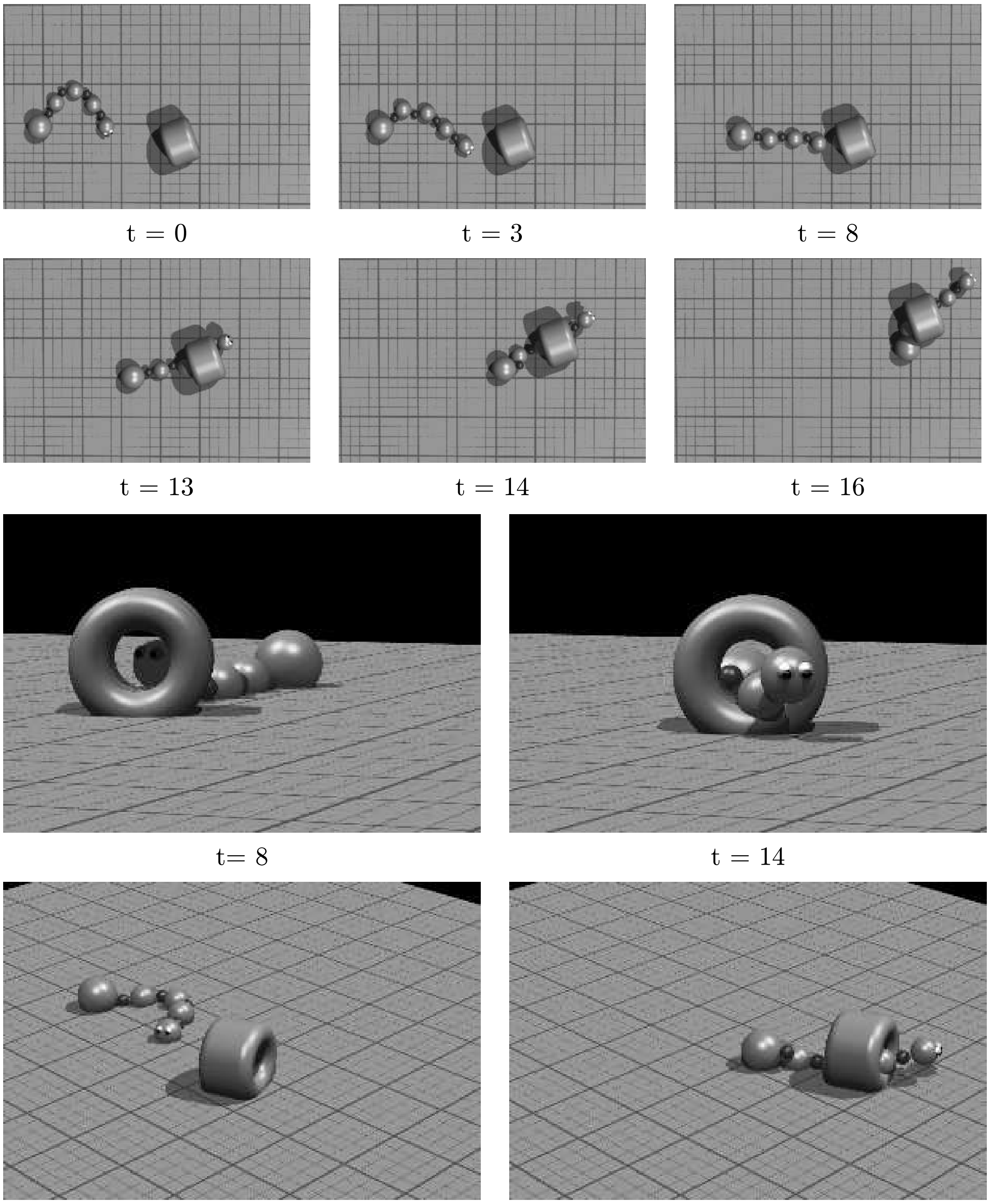
3.4 The bird and a falling ball: both are deviated.

Squares: positions of the bird and the ball at each time step
Lines: interpolation spline between key positions

Figure 9: Bird flight variations according to different external actions

4.5 Snake

Figure 10 shows a snake composed of nine deformable links connected by hinge constraints. A translation trajectory has been specified for an actuator in translation in the head of the snake, passing through a deformable cylinder. We see the body of the snake unfolding from its initial position and then colliding the cylinder, thus slowing down while dragging it.



t = 0 t = 3 t = 8

t = 13 t = 14 t = 16

t= 8 t = 14

t= 0 t = 14

Figure 10: A snake interacting with floor and a deformable cylinder

5 Conclusion

The method developed here should greatly simplify the use of physically-based animation, offering important help for the design of realistic motions and deformations when precise scripts are specified. The aim here is not to compute some muscle action to perform a purely dynamic motion, but rather to increase the realism of a trajectory that may be totally unfeasible without external help. Our method enables objects to follow a pre-defined script which can be as far as true dynamics as the animator wants it to, while adding dynamic quality wherever possible.

A convenient interface is offered to the animator, who controls motion by defining key frames. During this process, there is no need to spend time for carefully tuning the trajectories in order to improve realism, nor for problems such as interpenetration avoidance, or deformations of colliding objects. During an interactive simulation over time, the system will *automatically correct* the trajectories according to the physical parameters and to the collisions and contacts detected during motion. Adequate deformations of colliding objects will be generated during the same process.

The method works by associating translation or rotation actuators (or both) to the objects to control. The control module used for computing the actuator action from the user-defined key-frames derives from a generalized version of PD controllers. Objects can take observed external actions into account while regulating their motion. Deflected by sudden collisions, objects tend to compensate for continuous external actions after a period for adaptation specified by the user. The control method still works in complex situations where objects are components of articulated structures. In addition, trajectory control can be applied to only some of the objects in the scene, pure dynamic simulation or any other algorithm being used for the others. This should help the animator to only focus on the important motions. An object may be controlled in translation but not in orientation (or the opposite), leaving the simulator to generate realistic rotations according to friction forces during collisions and contacts with other objects.

Work in progress

In the method presented here, the velocity of the objects is adjusted during the simulation according to parameters such as mass and inertia, strength of actuators, path complexity, and events such as collisions which can accelerate the motion or slow it down. This greatly improves the realism of motion. However, we are currently studying an extension enabling the user to specify, if needed, preferencial speeds at some of the key-positions. The system will enforce an object to slow down or accelerate during motion, according to the animator specification.

In the current version of the system, target trajectories are independently defined for each solid component, so there is no synchronization between the different actuators acting on an articulated body, nor between different body's motions. Work in progress includes attempts to use a finite-state graph layer to add synchronization constraints to the system. The method will be based on the first extension, since acceleration or deceleration commands are needed to synchronize different motions.

Acknowledgements

We would like to thank Morgane Furio, from the Ecole Nationale Supérieure des Arts Décoratifs (Paris, France) for the design and scenario of the animation "Simply Implicit". Many thanks to Kevin Novins and to George Drettakis for re-reading the paper.

References

[BB88] R. Barzel and A. Barr. A modeling system based on dynamic constraints. *Computer Graphics*, 22(4):179–188, August 1988.

[BN88] L. Shapiro Brotman and A. N. Netravali. Motion interpolation by optimal control. *Computer Graphics*, 22(4):309–315, August 1988.

[Coh92] M. Cohen. Interactive spacetime control for animation. *Computer Graphics*, 26(2):293–302, July 1992.

[DLC93] 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.

[Duf86] T. Duff. Splines in animation and modeling. *State of the Art in Image Synthesis (SIGGRAPH'86 course notes Number 15, Dallas, TX)*, 1986.

[Dum90] 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.

[Gas93] M.P Gascuel. An implicit formulation for precise contact modeling between flexible solids. *Computer Graphics*, pages 313–320, August 1993. Proceedings of SIGGRAPH'93.

[GG94] 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.

[Han93] 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.

[IC87] 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.

[JLR93] S. Jimenez, A. Luciani, and O. Raoult. Physical simulation of land vehicles with obstacle avoidance and various terrain interactions. *The Journal of Visualisation and Computer Animation*, 4:79–94, 1993.

[NM93] J.T Ngo and J. Marks. Spacetime constraints revisited. *Computer Graphics*, August 1993. Proceedings of SIGGRAPH'93.

[Ove91a] C. Van Overveld. The generalized display processor as an approach to real-time interactive 3-D computer animation. *The Journal of Visualization and Computer Animation*, 2:16–25, 1991.

[Ove91b] 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.

[Ove93] C. Van Overveld. Building blocks for goal-directed motion. *The Journal of Visualization and Computer Animation*, 4:233–250, 1993.

[RH91] M. Raibert and J. Hodgins. Animation of dynamic legged locomotion. *Computer Graphics*, 25(4):349–358, July 1991.

[Sev89] Y. Sevely. *Systèmes et asservissements linéaires échantillonnés.* Dunod Univertité, Bordas, Paris, France, 1989.

[Sho85] K. Shoemake. Animating rotation with quaternion curves. *Computer Graphics*, 19(3):245–254, July 1985.

[vdPF93] M. van de Panne and E. Fiume. Sensor-actuator networks. *Computer Graphics*, August 1993. Proceedings of SIGGRAPH'93.

[vdPFV92] 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.

[WK88] A. Witkin and M. Kass. Spacetime constraints. *Computer Graphics*, 22(4):159–168, August 1988.

A Newton Integration Scheme for Equations of Motion

Let $\vec{x}(t)$ be the current object position, and $\vec{v}(t)$ its speed. Under a set of external forces $\sum \vec{F}$, we used the "Newton" scheme for integrating the equations of motion during a time interval dt:

$$\vec{v}(t+dt) = \vec{v}(t) + \frac{\sum \vec{F}(t)}{m}\, dt \tag{11}$$

$$\vec{x}(t+dt) = \vec{x}(t) + \vec{v}(t)\, dt + \frac{1}{2}\frac{\sum \vec{F}(t)}{m}\, dt^2 = \vec{x}(t) + \frac{1}{2}(\vec{v}(t) + \vec{v}(t+dt))dt \tag{12}$$

Similarly, the second Newton equation of motion for a solid of inertia tensor J and of current orientation matrix $R(t)$ gives:

$$\vec{\omega}(t+dt) = \vec{\omega}(t) + J^{-1}\left(\vec{\mathcal{T}}(t) - \vec{\omega}(t) \wedge J\vec{\omega}(t)\right)\, dt \tag{13}$$

$$R(t+dt) = \left(I + \frac{1}{2}(\tilde{\omega}(t) + \tilde{\omega}(t+dt))\, dt\right)\, R(t) \tag{14}$$

where $\tilde{\omega}$ is computed from $\vec{\omega}$ in such a way that: $\forall \vec{a}\;\; \tilde{\omega}\vec{a} = \vec{\omega} \wedge \vec{a}$.

Newton scheme computes an exact solution for constant forces applied during the time interval dt. It produces a better result in practice than Euler integration scheme used in [GG94].