Asynchronous Interactive Physical Simulation

Jérémie Dequidt, Laurent Grisoni, Christophe Chaillou

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Asynchronous Interactive Physical Simulation

Jérémie Dequidt — Laurent Grisoni — Christophe Chaillou

N° 5338

Octobre 2004

Thème COG

ISSN 0249-6399 ISRN INRIA/RR--5338--FR+ENG

Asynchronous Interactive Physical Simulation

Jérémie Dequidt , Laurent Grisoni , Christophe Chaillou

Thème COG — Systèmes cognitifs
Projet Alcove

Rapport de recherche n° 5338 — Octobre 2004 — 21 pages

Abstract: This document introduces a multi-agent framework for real-time physical simulation. Unlike classical physical simulators, no shared discrete time-line is imposed to simulated objects. Each simulated object is an autonomous agent that can maintain its simulation state, and possibly modify its behavior by checking its environment. Decision schemes are proposed that enable objects to locally adapt their computation time, in order to maintain approximate global synchronization. Results and measures are presented, especially regarding time management and simulation synchronization in regard of classical simulation methods. Examples are shown, outlining some of the practical advantages such a framework provides.

Key-words: interactive physical simulation, autonomous agents, simulation framework, loose synchronization.

Unité de recherche INRIA Futurs
Parc Club Orsay Université, ZAC des Vignes,
4, rue Jacques Monod, 91893 ORSAY Cedex (France)
Téléphone : +33 1 72 92 59 00 —Télécopie : +33 1 72 92 59 ??

Simulation physique interactive asynchrone

Résumé : Ce document introduit une architecture multi-agents pour la simulation physique temps-réél. Contrairement aux simulateurs physiques classiques, aucune ligne de temps partagée n'est imposée aux objets simulés. Chaque objet simulé est un agent autonome capable de déterminer son état et de modifier son comportement en interrogeant son environnement. Des méthodes de décisions sont proposées pour permettre aux objets d'adapter localement leur temps de calcul afin de maintenir une synchronisation globale. Des résultats et mesures sont présentés, notamment concernant la gestion du temps et la synchronisation de la simulation par rapport aux méthodes de simulation classiques. Des exemples soulignant les avantages pratiques de notre architecture sont donnés.

Mots-clés : simulation physique interactive, agents autonomes, architecture de simulation, synchronisation lâche.

1 Introduction

In modern physical simulations, objects tend to become more and more complex, either saw from the type of geometry they can handle (e.g. cloth [VTJT96, CYMTT92, BFA02], human organs [DDCB01], etc...), or their numerical flexibility (e.g. adaptive simulations [GKS02, CGC+02]). Adaptive objects appear to be very appealing due to their potential ability to change their behavior in order to match some constraints (limited system resources, precision...). However, current simulators are often built for a specific kind of modelling and can hardly handle adaptive or multi-resolution objects. Indeed, building some sophisticated simulation where many objects co-exist, each of them having its own schemes of resolution, is for now pure imagination. We think the main reason of this is the fact that sophisticated simulation often requires ad-hoc context. For example, some mechanical multi-resolution framework results [CGC+02] mention that for the simulation to be adaptive, and real-time to be kept, then implicit differential equation resolution [Cas75] can be used, along with adaptive time-step. This argument is correct, but demands that, if such objects would be integrated along with other objects in a larger simulation, the other objects can handle adaptive time-step too, and are simulated using implicit integration, which is not always natural for very simple simulations. Hence, a simulation involving many objects, among which such adaptive objects, demands specific development and algorithm, even for very simple objects: such a constraint is not natural, and should be able to be overcome.

These past years, within the software development community, Agent-based systems have known a growing success [Woo02]. Such a technique already benefits from extensive success on many Virtual Reality applications [HEV02, Tha00, NNI+03]. Agent based architectures allow for complete encapsulation of treatments and decisions of any kind. Such an approach already gives to virtual humans powerful features, easily providing systems of connection and communication between virtual entities, by providing models for trades, co-actions and collaborations. It nevertheless remains a new research field, and a lot remains to be done before such entity communication can be fully modelled. In particular, the physical simulation field has only been, to our knowledge, poorly studied so far. If such simulation could take advantage of multi-agent based architecture, then all the benefits of agents systems could be included within physical simulators, including the example mentioned, i.e. co-existence on the same simulation of simple objects, and multi-resolution ones.

This paper studies the application of multi-agent principles to physical simulation, and addresses the issues involved. In our model, each simulated object can handle motion resolution, collision detection and control of geometric and mechanical models. Such an object may be easily integrated in a simulation as most of computation is done by the object itself and not by another process. Using this approach, different models may coexist in the same simulation. Because of the autonomy of agents, the simulation is non-synchronous, and no global discrete timeline is any longer available, which is in our opinion the first required step for handling motion adaptivity in complex simulations. Consistency of the simulation is obviously the main question about such a structure: this paper pays special attention to tests and measures regarding simulation synchronization (i.e studies the question to know

if objects, following our scheduling technique, make the simulation consistent, even if no strong synchronization is globally imposed).

The remainder of the paper is organized as follow : section 2 presents some related works on physical simulation, section 3 overviews our framework. Section 4 copes with collision detection for autonomous objects, section 5 deals with autonomous agents. In section 6, tests and results are exposed: measures are provided that show the plausibility of simulation, and few interesting applications are outlined, in order to demonstrate all the potential advantages such a framework provides.

2 Related works

Simulations are usually non-dynamic frameworks: as long as it is intended to develop some complex interaction involving several simulated objects, the physical bodies have common behaviour rules, the integration step is usually the same for every physical body, collision detection uses a common model, etc... Moreover some stages of the simulation loop are, in most architectures, centralized. Thus traditionnal simulation framework cannot be applied with our simulated objects. However the following works propose a more open simulation framework.

In [Mir00], each physical body is a process which receives events for collisions or persistent contacts. Processes have also non synchronized local clock that measures *Local Virtual time*. The synchronization protocol is optimistic and was proved to be correct: the events have a *timestamp* and when a body receives an event, the body is integrated to the timestamp of the event and then the event is processed. Yet, a simulated object A may be rolled back to a previous state if it received events whose timestamp is prior to the body local virtual time. Other objects that are interacting with A received events to let them know they have to roll-back. Due to the roll-back mechanism, this may not be used for interactive simulation.

OpenMASK [Mar01] is a synchronous distributed simulation in which animated objects can work at different frequencies. The approach is mainly parallel: a centralized scheduler computes the simulation frequency, keeps synchronization between the objects and schedules tasks on all hosts. Such a framework is also not fully designed for physical simulation.

The BREVE system [Kle02] provides multi-agent framework for non-centralized systems, possibly handling physical simulation. In the breve system, the collision process involves global, classically centralized process: possible collision entails event production, hence providing immediate treatment of the response. This closely relates to classical, centralized, simulation. By comparison, we tried to relax such a condition, by providing each agent the possibility to decide of its own simulation rythm, and define logical adaptation rules, included to each agent.

Chenney [Che99] proposed an event-based simulation on rigid bodies, in order to combine both the need for simulation consistency, and the need to compute only needed simulations. Such is only applicable to rigid bodies, as deformable bodies usually demand more integration computation for plausible simulation. In addition to that, the work presented in this paper, by contrast, tries to separate simulation time from actually detected collision

time. Simulation frequencies are changed by the agents themselves, according to respective delay of simulation between agents, instead of using event-based programmation, that would super-impose computation frequencies to agents.

Some other works study problems related to the issue we address here. [BW97] discusses the possibility, for heterogenous simulation systems, to collaborate in order to achieve one global simulation. Constraints handling between systems are discussed. Our approach goes one step beyond this work, by setting each objet (not sets of objects of the same simulation kind) as an autonomous simulation, without any global discrete time-line superimposed to objects (by comparison [BW97] provides a common discrete time-line for all the simulations).

3 Technique overview

Our framework may be seen as a multi-agent system: each dynamic object is an agent that, given some internal data, computes its next state. In classical simulation, forces and torques are computed, then the motion is resolved, using some integration numerical tool, to provide new state of the system: spatial position and orientation. Communication between entities are restricted to collisions and constraints as they can change the motion of an entity (because they create new forces and torques). In the case of rigid bodies [Mir00], only a force and a torque is required. However for deformable bodies, many forces may be created due to collisions.

To ensure correctness of interactions and realism of simulation (further called *simulation consistency*), bodies must be at *somewhat* the same simulation time. Indeed a collision detected between 2 bodies A and B, which are too distant in the simulation timeline, is non-sense. Explicit synchronization is often used to grant simulation consistency: after each dynamic object has done its iteration, data are collected by another process to compute forces due to interactions and so on, this leads to synchronous simulation. However explicit synchronization does not fit with the independency of the multi-agent system paradigm. Such an independency has two main consequences:

- First, the collision detection framework should be designed to work with autonomous objects. Actually most of existing architectures involve a global treatment, which does not go in the direction of object independence.
- Second, one needs to make sure the simulation remains consistent, even when the synchronization is relaxed, and no strong synchronization is superimposed to objects: every simulated object should be more or less at the same simulation time. Autonomous agents take into account some information about the environment to give consistency to the simulation. They can dynamically decide to increase / decrease their simulation or modify their resolution scheme whether they are to slow or not.

In the following sections we detail these two issues in detail. In section 4, we present a collision detection framework that insure autonomy of the objects and in section 5 the global framework is exposed explaining how the simulation works and how it remains consistent.

4 Collision detection aspects

Our collision detection and treatment to collisions should insure, as much as possible, the autonomy of simulated objects. It means that a simulated body has to detect its collisions with other bodies and has to adapt its dynamics accordingly. Moreover, the collision and treatment process should be effective so as to work in an interactive simulation.

Given two colliding objects and an estimation of their intersection, penalty forces is a simple way to separate the objects: a force in proportion with the intersection estimation is computed according to the spring-damper model. The estimation of intersection may be the penetration depth which is the shortest vector that bring two colliding objects in touching contact.

Therefore, our goal is to propose a framework which detects collisions, computes penetration depth when collisions are detected and both of these stages should be done by simulated objects.

4.1 Rigid convex polyhedra

Lots of works have been produced on collision detection for rigid convex polyhedra (see [LG98, JTT01] for surveys). Algorithms based on the *GJK* [GJK88] are among the fastest and most robust algorithms. *ISA-GJK* [Ber98] is an incremental algorithm designed for convex polyhedra that quickly determine if two polyhedra are colliding or not (boolean answer). The computation of force response has, so far, been a bit less studied to our knowledge. Yet, the use of penetration depth for force response evaluation has so far proved to be quite a good compromise [KOLM02, KLM02]. We choose to use the *Expanding Polytope Algorithm* [Ber01], also based on the GJK, that uses *ISA-GJK* as a start point to get the penetration depth between two polyhedra.

These two stages can be done for autonomous objects: knowing the position of all the polyhedra in the scene, an object can check the collision and compute penetration depth if necessary. Moreover using two GJK-algorithms is no concurrence: initialization of the *Expanding Polytope Algorithm* is done by the last iteration of *ISA-GJK*, reducing the total computation time.

4.2 Deformable Objects

Collision resolution is far more expensive for deformable models, thus most frameworks use approximate methods to estimate penetration depth. Many recent works are available on deformable models [LM01], but using spheres for collision on deformable bodies is interesting [DJK97] because computing the penetration depth between two colliding spheres or between a sphere and a polyhedron is simple: for two colliding spheres, the penetration depth is $r_1 + r_2 - d$ where r_i are the radius and d the distance between the radii of the two spheres. Moreover, the Expanding Polytope Algorithm allows to determine penetration depth between a sphere colliding a polyhedron. Spheres clearly have the drawback to allow only for poor collision precision, especially regarding deformable objects such as those used

in section 6 (i.e. tissue): yet, they provide very fast collision tests, and can be adapted at will for precision increase [GNRZ02].

The main issue is to handle collisions between a high number of colliding spheres: consider a single sphere A that potentially collides a deformable object B (where B is composed of hundreds of spheres). It is obviously time-consuming to check collisions between A and each sphere that composes B, whereas most of spheres of B are far from A.

To accelerate the collision process, an internal voxel grid is built [DJK97] (figure 1). Each rectangular voxel contains spheres from the decomposition. So to test collision between A and B, A is placed in the voxel grid and collision are computed between A and the spheres that belong to the voxels in which A takes place (darkened voxels). Thus only spheres that are close to A need collision detection (blackened spheres).

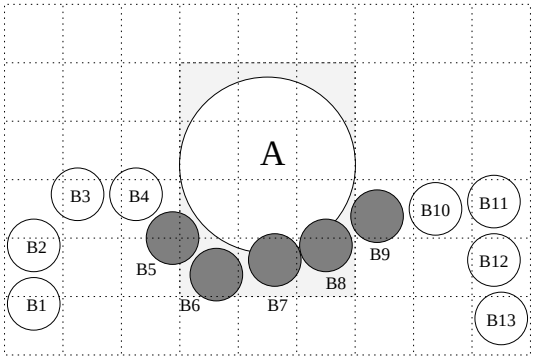

Figure 1: Internal voxel grid for deformable objects : deformable object B is in collision with the sphere A. Collisions are computed only between A and the blackened spheres.

4.3 Common pipeline

It is time-consuming for an object to know the position and to check the collision of all the other objects. Thus we build a pipeline [Zac01] for collision detection, in order to diminish its cost. We divided the scene space into regular zones in which a *zone agent* is defined. Agents get bounding volumes of every object in the zone and apply *Sweep And Prune* algorithm [CLMP95] to quickly determine potential colliding pairs. When a potential collision is found, the agent informs both objects that are concerned and then they send their position to each other and then apply the final stage (*ISA-GJK* and *Expanding Polytope Algorithm* for polyhedra, or algorithm based on sphere decomposition for deformable models, using voxel grid optimization mentionned above). We choose to use k-DOP [Zac98, KHM+98] as bounding volumes : they are a good approximation for most objects and are easy to construct and to update due to *GJK*.

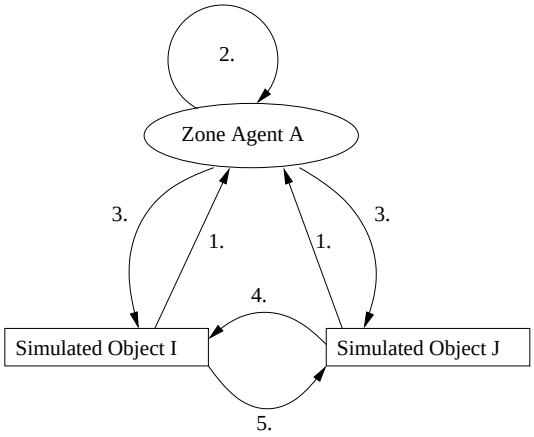

Figure 2: Example of two colliding polyhedra (I and J) in the zone A.

To make things clearer, consider figure 2. First each polyhedron send its k-DOP to a zone agent (arrows 1). Then the zone agent finds potential colliding pairs (arrow 2) and informs each polyhedron of its potential collisions (arrows 3). As I and J may probably collide, J sends its position to I (arrow 4). I detects a collision with J,then computes the penetration depth and sends it to J (arrow 5).

All of these added stages reduce the computational cost of collision detection in order to use it in interactive simulations. Moreover most of computations (especially costly computations) are done by the simulated objects, insuring autonomy for the agents.

4.4 Discussion

The collision approach presented in this section aimed at providing simulated agents with collision process that would be as "agent-like" as possible. In that way, we tried to minimize the global processing part of the collision detection, and optimize the collision pipeline. It is to note that, clearly, such an approach has the main properties:

- First, classical collision problems of multiple contacts between rigid objects are not addressed by our approach, as they are out of the scope of the problem addressed in this article. About this problem, the approach presented here does not bring anything new on this, as it is based on existing results about geometrical collision detection. Techniques like global optimizations [MS01], especially the question to know is such approaches could be linked to agent-based techniques for simulation, are indeed interesting, but are far beyond the scope of this paper.

- We use penalty method for force response calculus: as a result, we do not try to handle exact contact between objects: this presents the immediate drawback that simulated

objects can potentially geometrically intersect each other during the simulation, but also has the advantage that the relaxed synchronization of the simulation can actually be done.

Finally, it should also be noted that the notion of zone agent mentioned in this section does not refer to the same type of *agent* than in the other sections of this article: we named this likewise in order to remain within the same terminology. Yet, they can be seen as practical ways, or available tools for animated agents, for easily finding the objects potentially colliding them. Zone agents are not scheduled (see section 5), like other agents composing the framework.

5 Non-synchronous simulation time handling

As explained in section 3, our framework is an agent-like system that needs consistency to make the global simulation realistic. Consistency cannot be achieved with explicit synchronization within our context. Instead of imposing constraints to agents, we choose to control the behavior of an agent by defining local decision rules, that let each agent acts on its simulation. In order to produce efficient, and reactive enough system, we chose to built our own scheduler that coordinates tasks (dynamic objects). This scheduler (that is build in a quite classical manner) insures autonomy of agents: when an agent is provided with execution time, the agent animation loop iteration is called, and such a call is not interrupted until the function call terminates. Within this call, each agent has the ability first to calculate its next simulation state, but also to send to the scheduler its priority change wish.

5.1 The scheduler

Our scheduler is a priority based, FIFO scheduler that schedules tasks (i.e. autonomous agents) based on their priorities. A set of $N+1$ integer priorities is defined within the simulation: each agent a is assigned some initial priority $p(a) \in \{0, 1, \cdots, N\}$ (high priority value corresponds, in our notation, to intensive computation objects). The way our scheduler works is close from classical system scheduler: a queue Q_i is associated with each priority i, a task a is included in queue $Q_{p(a)}$. Each time an agent is activated, it can dynamically change its priority level, in which case it is then queued in its new priority queue (see section 5.2.1 for details). One additional array, S, stores objects that temporarily dropped off and do not need system resources for a moment (see section 5.2.2 for details about this). Each queue Q_i, $i \in \{0, \cdots, N\}$ is associated with an execution integer n_i, that can never exceed some pre-defined value max_exec: the n_i are used to control the execution frequency, and avoid execution starving problems. In pseudo-code description, the scheduler algorithm could be described as follows:

init(S, Q_0, $\cdots$, Q_N);
$prioCurrent$=N;

do forever:
  $n_{prioCurrent} = n_{prioCurrent} + 1$;
  if ($n_{prioCurrent}$==max_exec) then
    $n_{prioCurrent} = 0$;
    $prioCurrent = (prioCurrent - 1) mod (N + 1)$;
  else
    a=*getQueueHead*($Q_{prioCurrent}$);
    callAnimationStep(a);
    queue(a, $Q_{p(a)}$);
    $prioCurrent = N$;

It is important to understand that *callAnimationStep* on agent a can change its priority, according to rules defined in section 5.2.1, and hence, that the queue storage might not store agent a in the queue it was initially inside. The *queue* function can either store agent a in its new treatment queue, or store it in array S if necessary (see section 5.2.2). The proposed scheduling algorithm, simple to implement, ensures that agent of priority k are statistically activated for animation max_exec times more often than agents of priority $k - 1$: as a result, frequency call, as a function of priority, is an exponential rule.

Tasks stored in S are stored along with a requested wake-up time: S is constantly sorted according to this wake-up time, in order to keep efficient treatment. Agents are woken up using event propagation are reinserted in ad-hoc priority queue when requested. Each autonomous agent may completely stop its simulation for a period of time with this queue: yet, an agent never stops taking decisions such as priority change, or sleep time duration modification. This ensures that no artificial, non-recoverable decay would be inserted within simulation, without knowledge of the concerned agents.

This scheduler is designed so that a consistent simulation can only be achieved with the cooperation of every agent, so the main thing for the agent is that all the dynamic objects controlled by the agents should be at the same simulation time (this is the consistency criterion for the global simulation): each agent has, apart from the task to achieve its own simulation, to dynamically adapt its computation intensity to the global simulation speed. The following subsection describes the proposed technique.

5.2 Simulation control

All the control is given to agents themselves about their simulation. As explained above, the scheduler allows each agent to change the way it is computed, through two means: priority handling and sleep. In order to grant simulation consistency, efficient decision schemes within agents are needed on these two aspects, for defining the behavior of an agent. For such a purpose, an agent obviously needs to get information about its environment. As a result, the *callAnimationStep* function, used in the above algorithm, also transmits some information to agent a: the simulation time of the fastest agent and the simulation time of the slowest one. It is important to note that, as no common discrete time-line is imposed to objects, the only common time-line is the continuous simulation time. Depending on what

we will call from now on the current *simulation time window*, rules are used by agents for priority change decision, or, in case of important decay, sleep for a given simulation duration.

5.2.1 Priority modification

The algorithm used for priority modification is very simple: when an agent is on minimal (respectively maximal) time simulation (given by the scheduler at activation), agent priority is increased by 1 (respectively decreased by 1), for next activation, taken into account by the scheduler according to algorithm describe in section 5.1. By dynamically changing priorities, the gap between the fastest agent and the slowest one tends to decrease. As every agent acts in the same manner, agents tend to converge to the same simulation "speed" (on the simulation time-line). In order to make the system flexible, little delay between the simulation time of the fastest agent and the slowest one is permitted. Actually, temporal simulation coherency has among other consequences that, when simulation time distance between two objects interacting together is small, then the introduced errors, in comparison to classical, synchronized simulation, are negligible, provided that movements are of reasonable magnitude. Agents do not modify their priorities unless the delay between the extremal agents is greater than a pre-set *threshold*: this way, priorities changes are only achieved when needed.

5.2.2 Simulation sleep

Another way for an agent to control its simulation is to interrupt simulation computation for a while, by asking the scheduler to be called back only after a given amount of time. The idea is to dynamically evaluate how much CPU time a simulated object needs. The principle of this scheme is actually pretty simple: evaluation of simulation speed, simply by comparing evolution of simulation time, related to simulation window (i.e. maximal and minimal simulation time for all agents), easily provides some evaluation factor of CPU consumption accuracy for the considered agent, relatively to others. Once an object has evaluated the computation time it needs and placed it in regard to other simulated objects, it can interrupt itself after an iteration (and ask the scheduler not to be called before some given simulation time), allowing the other objects to do their own iteration. Then when it resumes, all objects are supposedly at the same simulation time, which ensures consistency of the global simulation.

modifyBehavior(curTime,minTime,maxTime)
$midTime$=($maxTime$+$minTime$)/2
$delta$=($midTime$-$locTime$)/($maxTime$-$minTime$)
$ratio$=$ratio$+$c1$*$delta$
$wake$=$curTime$+$lastCompTime$*(1-$ratio$)/$ratio$

$ratio$ is the fraction of CPU time the object is ideally consuming. Computation time of an object is bound to be irregular throughout time, because of collision computation, or numerical integration (implicit integration requires sometimes several iterations). Ac-

cording to the new *ratio*, it delays its next simulation call, allowing other objects to do their simulation: In the above algorithm description, $curTime$ represents the time (continuous simulation time) when the scheduler calls the function $modifiyBehavior$, $wake$ is the estimated time when the object will wake up and $locTime$, $minTime$, $maxTime$ are respectively the local simulation time, the minimal simulation time, the maximal simulation time. The modification of *ratio* is based on a value *delta* and a constant $c1$. The *delta* is a signed value included in $[-1/2, 1/2]$ which represents the distance between an extremal simulation time and the center of the simulation window. The closer $locTime$ is to the middle simulation time, the closer will be *delta* to 0, which will lead to a slight modification of *ratio*. Conversely, if $locTime$ is close to $maxTime$ (resp. $minTime$) then *delta* will be close to $-1/2$ (resp. $1/2$), which will create an important decrease (resp. increase) of *ratio*.

In other words, simulation interruption artificially delays computation of some objects. It can work in combination with, or independently from priority modification, in order to bring consistency to the simulation. When combined with priority modification functionality, simulation interruption adds another mean for simulated objects to control their behavior (for example by trying to respect some pre-set simulation frequency).

5.3 Comparison to centralized simulation system

Apart from the question to know if objects can actually remain, under some given tolerance, all together at the same simulation time, the most sensible question is the cost overhead point: actually, the scheduling task does not appear in some classical simulation architecture, and potentially demands additional computation time, while classical simulation is already quite computationally expensive. The task switching cost can be ignored in our test platform, as it reduces to simple function call (no multi-thread or multi-process structure is used). Looking at the scheduling algorithm, one can see that no complex computation is involved, and the computation overhead is only $O(1)$ per simulated agent.

Another point is that our approach has the important advantage that it can quite easily take advantage of multiple-CPU architecture, without specific additional efforts. As all memory writings are only achieved by the agent the data belong to, the potential concurrent access on data would be read-only, and the management of such access could be reduced to its most simple expression. By comparison, classical simulation, in order to provide the same properties, would need to get to more classical parallel numerical algorithms, which would demand a full re-writing of the resolution kernel.

6 Tests and applications

First, we present some tests aiming at measuring the efficiency of our framework : especially tests measuring the consistency of the simulation, i.e. the simulation time possible decay between objects. Then, as our agent-like framework offers the possibility to mix simulated objects of different kind, we describe practical examples that illustrate the potential benefits physical simulation can take from agent-based approaches.

6.1 Framework Validation

Framework validation, and/or comparison to classical, synchronous simulation, is indeed a difficult question. The tests we achieved try to show that in some "simple" case, our framework behaves in the same manner a classical global simulation would do. To test our framework, we have run simulations on Intel PIV 2.6 Ghz with 512 MB Ram. The test scene is composed of about ten non-deformable spheres falling on a plane. The integration step is constant, and set to 1 millisecond, using explicit Euler integration. Each simulated object has, at the beginning of the simulation, a random priority.

The main thing in our framework is to insure simulation consistency: this is linked to the delay between the fastest object and the slowest one. In our test scene, we measured the delay between the maximal simulation time (*fastest* object) and the minimal simulation time (*slowest* object). Figure 3 exposes the results. 10 000 instant measures where taken during simulation and all delays were lower than 16 msec. About 10% of the samples are below 6 msec, and most samples are between 8 msec and 11 msec which is confirmed by the average delay : 8,5 ms.

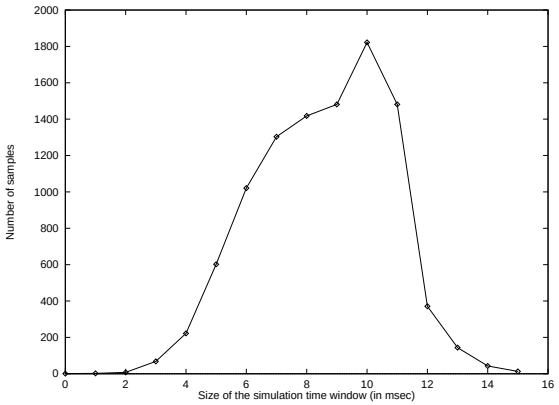

Figure 3: Repartition of delays between the maximal and minimal simulation time.

The average simulation time delay is about 8 milliseconds when we consider simulated objects whose integration step is 1 millisecond. Thus for the objects with extremal simulation time, the delay is about 8 simulation iteration. Because of temporal coherency that is, most of the time, available in interactive simulations, we can consider that the delay is small enough for providing a realistic simulation. Moreover, as we only record the delay between the fastest object and the slowest one, we assume that the delays between common objects (e.g. apart from the two extremal objects) are smaller.

Another possible test is to monitor the priority evolution during a simulation. We consider the same test application as previous, where the initial priority is randomly set for each simulated object. As all the objects are of the same nature, they should have more or

less the same computation time, and one would expect them to tend to the same priority. We snapshot objects priority at regular temporal step and put the results on figure 4. The curves have been smoothed for clarity, which leads to non-integer priorities. Actually, all the objects priority tend to converge to the same value, which matches the intuitive behavior one would expect from each autonomous agent.

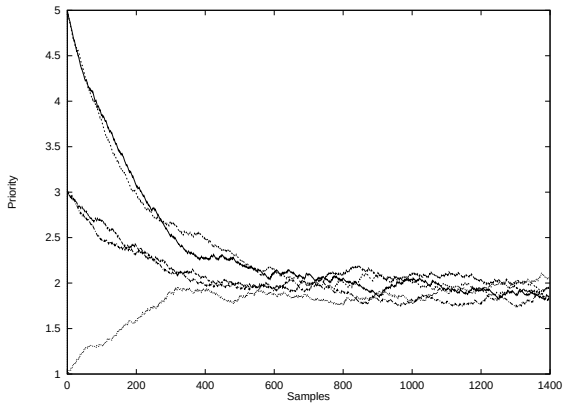

Figure 4: Priority evolution for 5 simulated objects (curves are smoothed for better readability, which creates non-integer priorities).

6.2 Sample applications

Apart from framework validation, it is natural to expect from agent-like framework to provide the same result as classical simulation frameworks, but also to propose extensions to it, in regard of simulation possibilities. We first present two example applications, that show that non-synchronous framework provides the same kind of functionalities than classical ones: first, complex simulation on rigid, convex bodies, and second, interaction on deformable model. We also provide here examples that show agent-based structure flexibility and advantages: first, simulation involving different integration schemes for objects (without loss of generality, mix between explicit and implicit numerical integration is provided as an example). Second, we provide example of complex interaction on several deformable models, where advantage is taken of implicit integration numerical stability in order to adapt time-step of each deformable model, depending on its interaction context.

6.2.1 Rigid convex objects simulation

The simplest application of our framework is rigid body simulation: no complex numerical integration is needed nor collision computation (our objects are spheres). Figure 5 exposes hundreds of rigid spheres falling on fixed bricks.

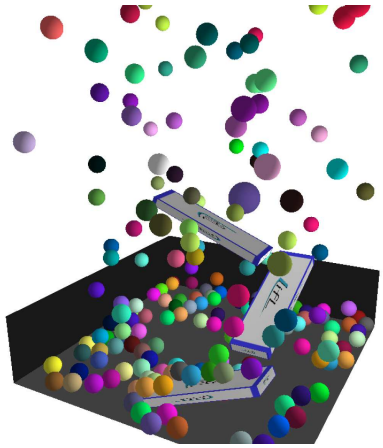

Figure 5: Falling spheres on fixed bricks.

6.2.2 Interaction on deformable objects

This example (figure 6) combines a rigid object and a deformable one. A deformable cloth is hanged by its four corners and a solid sphere, manipulated by the user, interacts with the cloth.

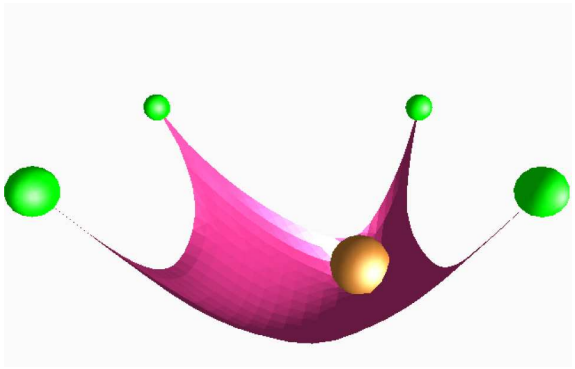

Figure 6: A Solid sphere interacting with a deformable cloth.

6.2.3 Heterogeneous integration

Because of the agent-based structure, no condition on the used integration numerical technique is super-imposed. It is actually straightforward, using the framework presented here, to combine objects with explicit integration (which is known to provide non robust, yet very fast, integration that can provide satisfying results on simple PDE) and objects with implicit integration (more robust, but more complex integration process), which allows to take benefits of both schemes. Figure 7 includes rigid spheres which use Euler explicit integration (for rapidity) and a deformable cloth which uses Euler implicit integration (for stability and precision). Thus computation time is not wasted to compute the movement of rigid spheres (as explicit integration is not too time-consuming). A companion video illustrates such property: a set of spheres (that all use simple explicit integration scheme) fall under the action of gravity within a tissue (that uses implicit integration, for better realism and stability). These spheres are then interactively manipulated by user, using some interaction sphere.

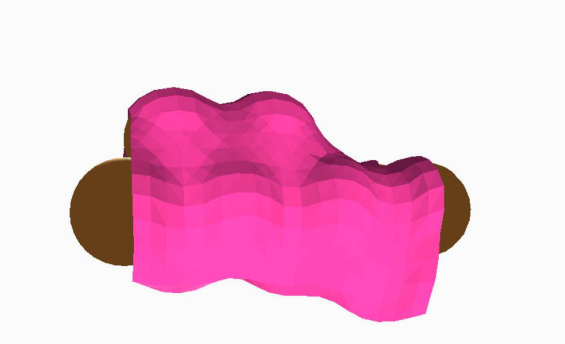

Figure 7: Example of heterogeneous simulation mixing explicit (on tissue) and implicit integration (on rigid spheres).

6.2.4 Adaptive time-step

As our framework provides consistency and important freedom for simulated objects design, one can control the simulation more precisely. When an object is close to stability (e.g. its movement does not change significantly), it can reduce its computation time: a possible way to achieve this is to modify the integration step dynamically. Some works [BW98] introduce adaptive integration step for implicit integration to meet some stability criteria. The method we used is the following: when the percentage of movement (i.e. euclidean distance between two consecutive state vectors for the considered object) for an object is below a fixed threshold for some time, this means that the object is close to stability. As a result, integration step may be increased, in order to save computation time (or possibly,

simulation may even be stopped). Conversely, interaction on the object, or collision with some other object entails automatic and event-based reconfiguration of the sleeping agent for reset of simulation time-step, as well as sleep time shortening (i.e. sleep time changing, see section 5). Figure 8 shows an example of such application, where 3 clothes are interactively

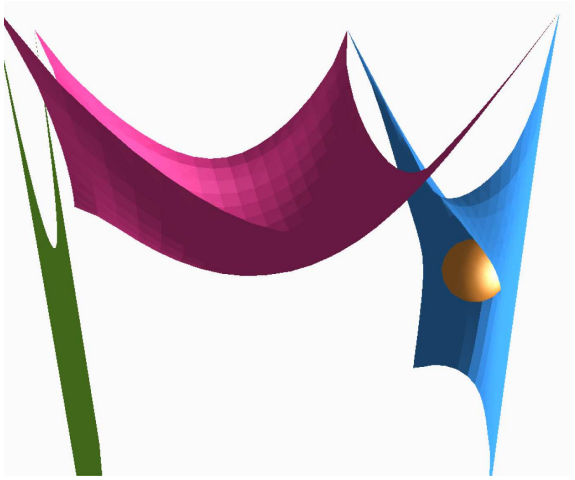

Figure 8: Application with adaptive time-step: the manipulated cloth has a smaller integration step than the other clothes that are in a stable state.

manipulated by user. A companion video illustrates real-time interaction on such models: on this video, where a set of 3 tissue is manipulated in real-time by user using some simple sphere, one can see that each tissue reacts in a satisfying manner, and that tissue interaction can be done at will by user, in spite of possible time-step modifications. This simulation takes full advantage of the fact that during the interaction, always at least two of the clothes are left hanging, while the third one is being manipulated. Timestep adaptivity allows for optimal CPU use, and better interaction than non-adaptive integration: in the non-adaptive case, too small a timestep makes a slow simulation, and too big a timestep provides poor numerical accuracy and weakens interaction with a model. Adaptive timestep simulation, where timestep is adapted for each object would be quite tricky to provide in a global simulation and is, by comparison, very straightforward to design in the agent-based framework presented here.

We think that the main advantage of such approach lies within the independence that each agent is provided with. The ability, for some object, to dynamically adapt its resolution time-step has no consequences on the algorithms involved for the other objects. On the simple illustrated example of Figure 8, it is clear that some centralized system, provided with explicit synchronization and timestep adaptivity, would somewhat provide the same features. Such centralized framework would demand complex numerical handling of the

global simulation resolution system, and ad-hoc data structure for such. By comparison, in the presented approach, the features added to some object simulation have no consequences at all on the design complexity of the other involved objects: each object remains responsible of its own complexity, both in terms of algorithm cost, and software design. Adaptive timestep is only one practical advantage that simulated objects can benefit of, within agent-based approach: no kernel adaptation was needed for our test, neither was it for use of heterogeneous integration techniques.

7 Conclusion

Our framework provides multi-agent system-like for physical simulation. Rigid bodies, as well as deformable objects, can be handled. This framework allows for hybrid method of integration, and may potentially be used on quite different types of objects (adaptive ones or very simple model). Most steps of the simulation are done by the agents themselves to insure autonomy and independency.

A lot remains to be done before all the advantages of agent-based framework for simulation would be completely perceived. Constraints between objects especially become a sensitive point in this framework: is a multi-body [Mir00] still a set of objects? What should be done, and how, when a set of objects is dynamically linked in order to create a multi-body?

We think this framework would be a good tool for complex, adaptive, physical interactive simulation, where many objects would interact together, each having its own sophisticated physical model. Adaptive time-step for implicit integration within a complex environment is only a small step toward really flexible practical simulation; providing simulators with stable dynamic regulation rules, that would automatically adapt computation cost to interaction situation, would also be of high usefulness.

References

[Ber98] G. Bergen. A fast and robust GJK implementation for collision detection of convex objects. *Journal of Graphic Tools*, 4(2):7–25, 1998.

[Ber01] G. Bergen. Proximity queries and penetration depth computation on 3D game object. *Game Developers Conference*, 2001.

[BFA02] R. Bridson, R. Fedkiw, and J. Anderson. Robust treatment of collisions, contact and friction for cloth animation. *Proc. Siggraph 2002*, pages 594–603, 2002.

[BW97] D. Baraff and A. Witkin. Partitioned dynamics. Tech. Report CMU-RI-TR-97-33, The Robotics Institute, Canregie Mellon University, 1997.

[BW98] D. Baraff and A. Witkin. Large steps in cloth simulation. In *Proceedings of ACM SIGGRAPH'98*, pages 43–54, 1998.

[Cas75] J. R. Cash. A class of implicit runge-kutta methods for the numerical integration of stiff ordinary differential equations. *Journal of the ACM (JACM)*, 22(4):504–511, 1975.

[CGC+02] S. Capell, S. Green, B. Curless, T. Duchamp, and Z. Popovic. A multiresolution framework for dynamic deformations. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 41–47. ACM Press, 2002.

[Che99] S. Chenney. Asynchronous, adaptive, rigid body simulation. In *Proc. SIGGRAPH 99, technical abstracts and applications*, July 1999.

[CLMP95] J. Cohen, M.C. Lin, D. Manocha, and M.K. Ponamgi. I-Collide : An interactive and exact collision detection system for large-scale environments. *ACM interactive 3D Graphics Conference*, pages 189–196, 1995.

[CYMTT92] M. Carignan, Y. Yang, N. Magnenat-Thalmann, and D. Thalmann. Dressing animated synthetic actors with complex clothes. *Proc. Siggraph 1992, Computer Graphics*, 26(2):99–104, 1992.

[DDCB01] G. Debunne, M. Desbrun, M.-P. Cani, and A. H. Barr. Dynamic real-time deformations using space and time adaptive sampling. In *Computer Graphics Proceedings*, Annual Conference Series. ACM Press / ACM SIGGRAPH, Aug 2001. Proceedings of SIGGRAPH'01.

[DJK97] Sung Yong Shin Dong Jin Kim, Leonidas J. Guibas. Fast collision detection among multiple moving spheres. In *13th annual symposium on Computationnal geometry*, August 1997.

[GJK88] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, RA-4:193–203, 1988.

[GKS02] E. Grinspun, P. Krysl, and P. Schröder. Charms: a simple framework for adaptive simulation. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 281–290. ACM Press, 2002.

[GNRZ02] L. Guibas, A. Nguyen, D. Russel, and L. Zhang. Collision detection for deforming necklaces. In *Proceedings of the eighteenth annual symposium on Computational geometry*, pages 33–42. ACM Press, 2002.

[HEV02] Z. Huang, A. Eliëns, and C. Visser. 3d agent-based virtual communities. In *Proceeding of the seventh international conference on 3D Web technology*, pages 137–143. ACM Press, 2002.

[JTT01] P. Jiménez, F. Thomas, and C. Torras. 3D collision detection: A survey. In *Computer Graphics*, volume 25, pages 269–285, 2001.

[KHM+98] J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *T-VCG(4)*, pages 21–36, 1998.

[Kle02] J. Klein. breve: a 3d environment for the simulation of decentralized systems and artificial life. In *Proc. of 8th international conference on artificial life*, December 2002.

[KLM02] Y. Kim, M. Lin, and D. Manocha. Fast penetration depth estimation using rasterization hardware and hierarchical refinement. In *Workshop on Algorithmic Foundations of Robotics*, December 2002.

[KOLM02] Y. Kim, M. Otaduy, M. Lin, and D. Manocha. Fast penetration depth computation for physically-based animation. In *ACM Symposium on Computer Animation*, July 2002.

[LG98] M.C. Lin and S. Gottschalk. Collision detection between geometric models : A survey. In *IMA Conference on Mathematics of Surfaces*, 1998.

[LM01] T. Larsson and A. Moller. Collision detection for continuously deforming bodies. In *EACG Conference on Eurographics*, 2001.

[Mar01] D. Margery. *Environnement logiciel temps-réel distribué pour la simulation sur réseau de PC.* PhD thesis, Université de Rennes 1, 2001.

[Mir00] B. Mirtich. Timewarp Rigid Body Simulation. *ACM SIGGRAPH*, pages 193–200, July 2000.

[MS01] V. J. Milenkovic and H. Schmidt. Optimization-based animation. In *SIGGRAPH*, pages 37–46, 2001.

[NNI+03] H. Nakanishi, S. Nakazawa, T. Ishida, K. Takanashi, and K. Isbister. Can software agents influence human relations?: balance theory in agent-mediated communities. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 717–724. ACM Press, 2003.

[Tha00] D. Thalmann. The virtual human as a multimodal interface. In *Proceedings of the working conference on Advanced visual interfaces*, pages 14–20. ACM Press, 2000.

[VTJT96] P. Volino, N. Magnenat Thalmann, S. Jianhua, and D. Thalmann. An evolving system for simulating clothes on virtual actors. *IEEE Computer Graphics and Applications*, 16(5):42–51, 1996.

[Woo02] M. Wooldridge. *An Introduction to Multiagent Systems.* Wiley & Sons, 2002.

[Zac98] G. Zachmann. Rapid collision detection by dynamically aligned DOp-trees. *Proceedings of IEEE, VRAIS*, 1998.

[Zac01] G. Zachmann. Optimizing the collision detection pipeline. *First International Game Technology Conference (GTEC)*, 2001.

Unité de recherche INRIA Futurs
Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399