Eurographics/SIGGRAPH Symposium on Computer Animation (2003)
D. Breen, M. Lin (Editors)

A Practical Dynamics System

Zoran Kačić-Alesić zoran@ilm.com
Marcus Nordenstam mnorden@ilm.com
David Bullock bullock@ilm.com

Industrial Light + Magic

Abstract

We present an effective production-proven dynamics system. It uses an explicit time differencing method that is efficient, reasonably accurate, conditionally stable, and above all simple to implement. We describe issues related to integration of physically based simulation techniques into an interactive animation system, present a high level description of the architecture of the system, report on techniques that work, and provide observations that may seem obvious, but only in retrospect. Applications include rigid and deformable body dynamics, particle dynamics, and at a basic level, hair and cloth simulation.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and RealismAnimation; I.3.5 [Computer Graphics]: Computational Geometry and Object ModelingPhysically based modeling; G.1.7 [Numerical Analysis]: Ordinary Differential EquationsConvergence and stability, Stiff equations

1. Introduction

Our goal is to produce computer animation that is both visually compelling and physically believable. Examples include common everyday situations – characters and objects that move under the influence of gravity and other external forces, interact with each other and with the environment, and possibly deform or break apart in the process.

Commonly available computer animation tools based on forward and inverse kinematics may be sufficient for a skilled animator to create an animation that appears to obey the laws of physics. But doing it for a large number of objects or at a high level of detail is at best tedious. Numerical simulations are often employed to either create physically based animation of dynamic systems or to augment the animation conceived by the animator in a way that adds a layer of realism.

Dynamic systems often exhibit chaotic behavior – a tiny change in the initial state can cause a large change in the outcome. Of all possible physically correct outcomes, we have a strong preference for those that are visually interesting to the animator or some other subjective observer. Rolling dice can be both fun and exciting but an animator's job often requires that "seven" be rolled any number of consecutive times without raising suspicions, at least not visually.

The problems we face are as much about control as they are about physically and numerically sound algorithms. Animators need tools to take the animation in any desired direction without destroying the illusion of physical correctness. When on occasion they are asked to do the physically impossible, it is a toolmaker's job as much as an artist's to make the impossible look good.

Many choices need to be made when implementing a dynamics system. We hope our exposition is of equal interest to developers of physically based animation tools and researchers looking to create a playground for testing new ideas. Our system is based on years of film production experience, a considerable degree of frustration, and a number of successes. We emphasize techniques that work and have been useful to our users. We prefer solutions that are simple and flexible. In particular, we would like to draw attention to the advantages of using simple explicit integration methods, handling everything uniformly as springs, and trading off stiffness for computation speed. Hopefully, many concepts we describe will seem obvious to an experienced practitioner.

Most of our examples come from rigid and deformable

Figure 1: *Rigid body mayhem: fallen Battle Droids from Star Wars Episode II: Attack of the Clones*

body dynamics, but the core principles are applicable to hair and cloth simulation, and to particle dynamics in general.

2. Related Work

A well written introduction and overview of physically based modeling can be found in [32]. Several important concepts related to the implementation of rigid body algorithms are described there in more detail than in our paper. For an introduction to numerical methods for integration of ordinary differential equations (ODEs) see [28] and [22]. A more in-depth treatment can be found in [16] and [14].

Much of the recent research in the area of dynamics has concentrated on cloth. The choice of integration method is often based on cloth simulation needs (see [4], [9], [15], and [31]). Choi and Ko [8] recently propose a cloth bending model that decreases fictitious damping of implicit methods.

The choice of integration method in many of these papers seems to be based on the premise that dynamics problems are numerically stiff and that implicit integration methods must be used to achieve stability with reasonably sized time steps. Absolute stability of implicit methods can indeed provide computation savings by allowing very large integration time steps, but the cost of taking large time steps is a much decreased accuracy and a loss of high frequency response of the system. A common result is an animation that appears overly damped and sluggish. We believe that preservation of high frequences is absolutely critical for natural looking cloth, hair, and rigid bodies.

It is, of course, possible to decrease the time step in an implicit method so that it approaches the size required for stability of explicit methods and for preservation of high frequency response, but this defeats the primary reason for using implicit methods. The computational cost of building and solving a large sparse system, the trouble of maintaining sparseness and positive definiteness (stability) for over-constrained and highly damped systems, and the increased complexity of implementation make implicit methods less attractive.

We have had very good success over the years with explicit integration methods. In our experience, significant benefits include better control over damping, simplicity of implementation, and ease of extension.

Bridson at al [5] demonstrate that complex cloth can be simulated using an explicit central differencing method that is substantially similar to the method we use. They also address cloth collision issues that are beyond the scope of our paper. Other recent treatments of collisions include [12] and [3]. More recently, Bridson et al [6] propose a new mixed explicit/implicit integration scheme and an accurate bending model for cloth.

We have found no publications documenting existing dynamics systems used for film, animation, or game production at a level that would allow a technically informed comparison. Of the commercially available animation systems, Maya [1] supports physically based simulations of rigid and soft bodies, as well as cloth. Its cloth dynamics has been used successfully on several film productions, including Shrek [25]. Maya's dynamics engine is, as far as we know, the only one based on published algorithms (see [4] and [15]). These publications do not describe a system, but we can learn that Maya's solver uses an implicit integration method, in contrast to our solver, which uses an explicit method.

For certain rigid body configurations, for example long chains made of many rigid links, Maya is faster than our system. On the other hand, animators who have equal access to Maya and our system, use ours almost exclusively for complex rigid body simulations. They attribute this primarily to the robustness of our software in dealing with large over-constrained systems (like those in figures 1, 5 and 6, which for reasons unknown to us tend to be unstable and unreliable in Maya), to its speed, and to having a complete control over every aspect of simulation [2].

Commercial dynamics libraries used in the games industry, like Havoc and MathEngine, are implemented at a level similar to our solver. Their internals are not publicly documented and they also do not provide a system in the sense described in our paper.

3. Overview of the System

Our system is built on a spring-mass model. At the lowest level, the *solver* sees a world consisting of a potentially large number of point masses, or centers of mass for rigid bodies, connected with springs in an arbitrary configuration. While

this may not be the most accurate model of the world, it is simple and it works surprisingly well for many applications.

At the application level, the dynamics system is a module in a general 3D modeling and animation system. At this level the world, or the *scene*, as we often call it, consists of many complex polygonal or B-Spline surface models animated using a combination of keyframe animation, procedural methods, deformations, simulation, and motion capture. The user has complete control over every attribute that describes the objects in the scene, their motion, and interaction – there is a graphical user interface for interactive control and a scripting language that can achieve the same programmatically.

There is a layer between these two levels that translates scene description to and from a format suitable for simulation. This separation allows us to write simulation code that is compact, efficient, reusable, and easy to understand and maintain – at the expense of duplicating some of the scene description data.

Here are several important observations about our system:

- Every parameter that controls dynamic behavior, including the fact that an object is being simulated at any given time, can be animated by the user.
- There is only one type of constraint at the solver level: a spring. Complex user level constraints become arrangements of springs.
- There are two types of springs: those with stiffness controlled by the user and those that are always as stiff as possible. Much of the section 4.2 is devoted to figuring out how stiff the second type can be.
- During simulation, the primary tradeoff is between the stiffness of constraints and computation speed. None of the constraints are absolutely rigid, but they can become arbitrarily stiff if we are willing to wait for the results. In the majority of cases we do not need to wait very long since...
- Most real life examples are not numerically stiff.
- Since all the constraints are somewhat soft, the system can be severely overconstrained without adverse effects on stability. When multiple conflicting constraints are imposed, the constraints are not completely satisfied, but they are met as well as possible.
- Objects that interact in any way need to be solved by the same solver at the same time. Conversely, while objects do not interact with each other they do not need to be solved by the same solver, at the same time, or using the same simulation parameters. The concept of animation *layering* came from this simple observation and is described later in the paper (section 6.
- Transitions between keyframe animation and dynamic simulation (figures 5 and 7), as well as interactions between keyframe animated objects and dynamic objects (figure 4) must be flawless. This is one of the most used features of our system and is primarily a matter of correctly computing the state variables from the keyframed motion. For rigid bodies, understanding the relationship between the angular velocity and the angular momentum (see [32] and [11]) is important for correct computation of angular momenta from animation.
- Handling collisions is by far the most computationally expensive part of many simulations. While efficiency of the time discretization scheme is important, the optimization effort is much better spent on detecting and resolving collisions.
- Double precision floating point arithmetic provides significant improvements in terms of accuracy and stability over single precision arithmetic. The improvements are well worth the cost of increased memory usage.

4. The Solver

The solver is responsible for advancing the state of the spring-mass system over time. Important properties are stability, accuracy, and efficiency – a good solver is one that balances these three properties well. Critical choices are that of a time discretization technique (numerical integration method) and the size of the time step.

Other important aspects of the solver are simplicity of implementation and extensibility. For that reason, as well as for accuracy, we prefer explicit integration methods.

We have had very good results with the *Verlet* integration method [30]. It is widely used in molecular and particle dynamics, and lately in the games industry, as well as in other areas where substantially similar or equivalent methods are known under different names: *Störmer method* in astronomy, *leap-frog method* in the context of partial differential equations and in plasma physics [29], and *Newmark central differencing* (a special case of the general Newmark scheme, see [17]) in structural dynamics. In several of these areas it has become the most widely used integration scheme, and is often regarded as having the optimal balance of stability, accuracy, and efficiency.

Although only marginally more stable than the explicit (forward) *Euler scheme*, the Verlet method achieves second order accuracy at a computational cost of a first order method by introducing a velocity approximation at the midpoint, as described below (for that reason this scheme is often called *velocity Verlet*). It has another property, probably less critical for our application than, for example, in astronomy, but still nice: it falls into the category of *symplectic* methods which preserve the numerical flow of the underlying system. In contrast, both explicit (forward) and implicit (backward) Euler, as well as higher order *Runge-Kutta* methods exhibit numerical drift away from the flow of the system. For an in-depth treatment of symplectic structure preserving algorithms for ODEs see [13].

4.1. Equations of Motion and the Verlet Scheme

Implementing an explicit first or second order integration method is not very difficult but it still needs to be done correctly. We feel it is important to outline here how the state of the system is advanced over time. This is a bit involved – in the first pass the reader might want to skip to section 4.4.

The state of a spring-mass system at a given time t is defined by positions $\mathbf{p}^t$ and velocities $\mathbf{v}^t$ of the particles. This is known as a *Lagrangian* formulation as opposed to a position and momentum based *Hamiltonian* formulation.

If there are rigid bodies in the scene, all the steps that apply to particles also apply to the centers of mass of rigid bodies and/or spring attachment points – a spring is allowed to attach to an arbitrary point f on the rigid body, not just the center of mass, and the position p_f is stored with the spring. The state of rigid bodies additionally includes the orientation $\mathbf{R}^t$ and angular momentum $\mathbf{L}^t$ around the center of mass.

In addition to the above time-varying state variables, other variables are needed to fully describe the system: particle masses $\mathbf{m}$, spring stiffness coefficients $\mathbf{k}$, spring damping coefficients $\mathbf{b}$, spring rest lenghts $\mathbf{l}_r$, and the connectivity information (particles a and b) for each spring. For rigid bodies add the inertia tensor in the local object space $\mathbf{I}_{body}$ (see [32] for details). Although these extra variables can be animated by the user, they are allowed to change only at certain times and can be considered constant for the duration of any time step.

More temporary variables are needed to store partial results of computation: total forces $\mathbf{F}^t$ exerted on particles and the resulting accelerations $\mathbf{a}^t$. For rigid bodies add torques $\mathbf{T}^t$ and angular velocities ω^t.

From an initial state $(\mathbf{p}^0, \mathbf{v}^0)$ and optionally $(\mathbf{R}^0, \mathbf{L}^0)$ at time t = 0 we first compute the following for each particle and/or point of attachment for rigid bodies:

$$\mathbf{F}^0 = \mathbf{F}^0_{external} + \sum_{springs\ i} -k_i x_i^0 \quad (1)$$

$$\mathbf{a}^0 = \frac{\mathbf{F}^0}{\mathbf{m}} \quad (2)$$

where $x_i^0 = l_i^0 - l_{ir}$, $l_i^0 = p_{ai}^0 - p_{bi}^0$, and $\mathbf{F}_{external}$ are the combined forces from all external non-spring sources such as force fields.

Things are slightly more complicated for rigid bodies. Forces $\mathbf{F}^0$ computed in step (1) are divided into two components. The component in the direction of the center of mass is applied like a regular force and causes linear acceleration of the rigid body. The component perpendicular to the line between the attachment point p_f and the center of mass $\mathbf{p}$ results in a torque that contributes to the angular momentum of the rigid body:

$$\mathbf{T}^0 = \sum_{points\ f} (p_f^0 - \mathbf{p}^0) \times \mathbf{F}_f^0 \quad (rigid\ only) \quad (3)$$

We then proceed taking a number of time steps Δt using the Verlet method:

$$\mathbf{v}^{n+\frac{1}{2}} = \mathbf{v}^n + \frac{\Delta t}{2}\mathbf{a}^n \quad (4)$$

$$\mathbf{L}^{n+\frac{1}{2}} = \mathbf{L}^n + \frac{\Delta t}{2}\mathbf{T}^n \quad (rigid\ only) \quad (5)$$

$$\mathbf{p}^{n+1} = \mathbf{p}^n + \Delta t \mathbf{v}^{n+\frac{1}{2}} \quad (6)$$

$$\omega^{n+1} = (\mathbf{R}^n \mathbf{I}_{body} \mathbf{R}_n^T)^{-1} \mathbf{L}^{n+\frac{1}{2}} \quad (rigid\ only) \quad (7)$$

$$\mathbf{R}^{n+1} = \mathbf{R}^n + \Delta t \omega^{n+1} \quad (rigid\ only) \quad (8)$$

$$\mathbf{F}^{n+1} = \mathbf{F}^{n+1}_{external} + \sum_{springs\ i} -k_i x_i^{n+1} - b_i \mathbf{v}_{li}^{n+\frac{1}{2}} \quad (9)$$

$$\mathbf{T}^{n+1} = \sum_{points\ f} (p_f^{n+1} - \mathbf{p}^{n+1}) \times \mathbf{F}_f^{n+1} \quad (rigid\ only) \quad (10)$$

$$\mathbf{a}^{n+1} = \frac{\mathbf{F}^{n+1}}{m} \quad (11)$$

$$\mathbf{v}^{n+1} = \mathbf{v}^{n+\frac{1}{2}} + \frac{\Delta t}{2}\mathbf{a}^{n+1} \quad (12)$$

$$\mathbf{L}^{n+1} = \mathbf{L}^{n+\frac{1}{2}} + \frac{\Delta t}{2}\mathbf{T}^{n+1} \quad (rigid\ only) \quad (13)$$

Note that the velocities $\mathbf{v}_l$ used in step (9) to compute the damping forces are not the velocities of the particles at the end of the springs, but rather the rate at which the springs change their length.

4.2. Stability and the Size of the Time Step

For a thorough treatment of linear stability theory for ODEs see [16] and [14]. Similar to other explicit methods (Euler and Runge-Kutta), the Verlet scheme is conditionally stable based on the size of the time step (this is the limitation that inspired the use of implicit methods) – it is stable only if the following condition (sometimes called the *Courant* stability limit) is met:

$$\Delta t \leq \frac{2}{\omega_d} \quad (14)$$

where ω_d is the highest modal natural frequency of the system. For an undamped single spring this frequency is (see [26]):

$$\omega_{d0} = \sqrt{\frac{k}{m_h}} \quad (15)$$

and for a damped spring it is

$$\omega_d = \sqrt{\omega_{d0}^2 - \left(\frac{b}{2m_h}\right)^2} \tag{16}$$

where the harmonic mass m_h of masses m_a and m_b at the two ends of the spring is defined as:

$$m_h = \frac{1}{\frac{1}{m_a} + \frac{1}{m_b}} \tag{17}$$

Solving for $\omega_d = 0$, we can calculate critical damping b_c, the damping factor that results in no oscillations:

$$b_c = 2\sqrt{km_h} \tag{18}$$

For a spring system with n springs the worst possible scenario is when all the springs are connected in parallel between two particles. The combined spring coefficient of all springs is $K = \sum^n k_i$ and the integration is stable if

$$\Delta t \leq 2\sqrt{\frac{m_{min}}{K}} \tag{19}$$

where m_{min} is the smallest harmonic mass of any spring in the system.

While this criterion might seem useful in adaptive time stepping, it is never used in practice in favor of a less restrictive and more general "10% rule", described in [5] and [23]. This effective rule of thumb for computing the size of the time step is based on the premise that no spring should ever change its length by more than 10% in a single time step. It is applied after $\mathbf{v}^{n+\frac{1}{2}}$ are computed in step (4) and before the final values of $\mathbf{p}^{n+1}$ are computed in step (6). Note that there is nothing magical about the 10% value – we encourage the reader to experiment with other values.

The stability problem can be posed another way: given a fixed time step, how much force can the springs exert without destabilizing the system? Using the same worst-case scenario and assuming for the moment that all n springs have the same coefficient k_{max} and therefore $\sum^n k_i = nk_{max}$, we can solve for k_{max} and conclude that the integration will be stable if

$$k_{max} \leq \frac{4m_{min}}{n\Delta t^2} \tag{20}$$

This is a very conservative upper bound on the maximum spring stiffness. In practice, values larger by one to two orders of magnitude can be used safely. But k_{max} does provide a starting point for springs that are not explicitly controlled by the user and that need to be as stiff as possible (e.g. collision springs described in section 4.4), or for clipping the stiffness of springs in order to prevent the time step from getting smaller.

If the solver is allowed (by the user) to decrease the time step adaptively, the 10% rule on existing springs is very good at accounting for external and damping forces. If time step is not allowed to decrease, which is typical in interactive use, the stiffness of existing springs is clamped in an attempt to maintain stability. Of course, this may fail if stability is compromised by excessive overdamping or external (field) forces. Eventually, one of the state variables may overflow causing a floating point exception, which is handled by the solver in a way that causes no loss of work. At that point the user is informed about the exact source of the problem, e.g. a particular constraint or a rigid body, and instructed to decrease the time step or tame the offending force field, before resuming simulation.

Most of the time we let the animator determine the size of the time step. Its value can be animated by the user but the system allows it to change only at the beginning of a frame of animation. In the early blocking stages of animation, very large time steps are often used. Simulations run fast but the resulting motion is soft and imprecise since spring stiffness values are being clipped. The animator can visually judge the result and tighten the time steps only where needed. This tradeoff between speed and softness is somewhat intuitive and has become widely accepted by the animators. It is even used as a tool to achieve different moods of animation.

4.3. Damping and Ether Drag

It is easy to critically damp any single spring but, in general, it is not possible to critically damp an entire spring system, except for perfectly symmetric cases. If critical damping is important for the whole system (it rarely is in practice), most springs in the system will need to be over-damped.

While deriving equations (19) and (20) we assumed that spring damping cannot increase the natural frequency of the system. That may seem logical if one looks at the equation (16) for a single damped spring but, unfortunately, is not true. Due to numerical nature of integration, when highly damped stiff springs compress or expand at high rates, large damping forces can overcompensate and destabilize the system. To our knowledge this has not been a problem for rigid body simulations, and we do not have a clear indication that it has adversely affected hair or cloth simulations in production. But it is easy to demonstrate that a highly damped hair or cloth model requires smaller time steps for stability than a less damped but otherwise identical model.

A possible solution is to use an implicit method only for damping forces while keeping the explicit method for the elastic forces, as described in [6]. The usually objectionable "artificial" damping associated with implicit methods is

not an issue since high frequencies are preserved for elastic forces. And it is possible to modify our implementation (steps 4-13) in a way that avoids much of the complexity and overhead of fully implicit methods.

Besides damping the springs our system also allows user controlled damping of particles. We refer to this as *ether drag*, since it simulates the viscosity of the medium through which the particles travel. It is very useful in removing unwanted high frequency oscillations in the system, particularly oscillations perpendicular to the spring directions since those are not easily controlled by spring damping.

4.4. Collisions

Collisions are a critical component of any physically based dynamic system. Rigid bodies collide with each other and their environment; hair collides with the body, cloth and other hair; cloth collides with the body, other objects in the environment, other cloth and itself, and so on. A large portion of all forces in a system at a given time may be due to collisions. Besides damping, collisions are the predominant way for a system to dissipate energy.

Collision detection is a large topic that is well covered by published research and still attracts considerable interest (see [5] and [12]). Our solver treats collision detection as a black box - it does not depend on any particular algorithm. We can easily swap one method for another or use multiple methods simultaneously. We use several collision detection techniques, none of which is perfect or novel: we collide points of one object against either polygons or a signed distance volume of another object. We use both analytical/procedural and discretized distance volumes (distance fields or level sets, see [10] and [19]). Point-polygon collision detection is optimized by a straightforward use of space partitioning data structures such as bounding box trees and k-d trees.

We use a penalty-based collision response model. Handling of collisions between geometries of two objects is broken into detection and resolution. When a collision is detected, zero-length springs of maximum possible stiffness are dynamically created between points of one object and the nearest points on the surface of the other object. These springs then work like any other spring over a number of time steps and when the collision is resolved the collision springs are deleted.

Elasticity of collisions is simulated by the damping force of the collision spring. A perfectly undamped collision spring should invert the incoming point's normal velocity; a critically damped collision spring simulates a perfectly inelastic collision. A user controlled scale factor e_s, $0 \leq e_s \leq 1$, is used to scale between the two. Note that the scale factor is not linear in the amount of absorbed velocity or energy.

Elastic collision response works well for rigid and soft bodies, as well as for particles, but may not be the best solution for hair and cloth, which tend not to bounce and can benefit from more direct and computationally faster techniques (section 4.6), particularly for self-collision. Details of such techniques are beyond the scope of this paper.

Collision handling is by far the most expensive part of any simulation and compromises are often necessary to achieve reasonable simulation times. We already talked about the softness vs. computation time tradeoff. Other choices include using collision geometry that is of much lower resolution than the models rendered in the final animation. Also, collision detection is not done at every iteration – a typical ratio in our rigid body solver is 100 regular time steps for every collision detection step; the actual ratio is controlled by the user.

In some cases, particularly for closed models that do not change shape during simulation, volumetric collision objects based on signed distance fields can provide great savings and also improve the quality of collisions. Even faster are collision objects based on (a collection of) procedural or analytical distance fields, such as spheres, ellipsoids, cylinders, and cubes.

4.5. Force Fields

Our solver provides a simple callback plugin architecture that allows technical users to implement their own force field functions, which get evaluated at every iteration during step (9) for every particle or polygon. In spite of this flexibility our users accomplish almost everything with just three types of fields: gravity, air (wind), and water (buoyancy). Gravity is an acceleration field with user controlled magnitude and direction. Air is a linear or radial flow field that exerts force on polygons based on their orientation, area, and relative speed. Air attributes include direction and magnitude of flow, linear and spin drag, radius or cone of action and falloff. Water is a "soft collision" half-space with density and viscosity attributes.

There can be any number of fields affecting any or all dynamic objects at any time, and everything can be animated. For example, air fields are used to a great extent in rigid body dynamics to provide sudden blasts of force and initiate explosions. Some of the fields may be active for only one or two animation frames.

The callback plugin architecture can be used for other external forces, not just force fields. For example, the cloth bending model described in [6] can be added to the system using this architecture.

The solver performs no stability analysis for the force fields. While this is rarely a problem, excessive sudden forces can destabilize the solver. If this happens, it is important that the solver recovers gently from such an exception and provides the user with as much useful information about the source of instability as possible. The recourse is to either decrease the time step or to tame the force field.

4.6. The Not-So-Physically-Based Aspects of the System (the Controls)

While every attempt is made to use the spring-mass model where possible, there are occasions when great speed improvements, both in terms of simulation and user time, can be achieved by simpler mechanisms. These mechanisms have one thing in common: they modify the state variables (positions and velocities) in a non-physical manner at every time step.

A typical scenario arises from the following question: "Why can't the simulated object behave just like this animated object"? Upon further examination a more precise statement of the problem may turn out to be: "I really like the way my character tumbles down the stairs in this simulation but it needs to continue waving the right hand just like in the animation. And by the way, I need this done by 5pm".

This is a control problem. And often, physically based solutions are at best tedious. Animators are skilled in keyframe animation and can express themselves most easily using traditional animation user interfaces and techniques. Motion capture data is often available and can be used together with animation and simulation.

In the above case a much simpler and yet effective solution is to allow the simulated hand to inherit some of the motion (relative to the arm) of the animated hand. More precisely, some of the localized velocity of the animated hand is added to the velocity of the simulated hand. This is balanced against velocities generated by physical means (spring forces) to a degree controlled by the animator.

For another example we turn to published research. In their treatment of hair-body collisions Chang et al [7] just move penetrating hairs to the nearest valid location outside the body and set the velocity of the hair to be the same as the velocity of the body. This is simple, fast, and effective.

While such techniques are needed infrequently in rigid body dynamics, they are still essential for hair and cloth simulation [24]. Since the immediate benefits are considerable, it is tempting to add one-of-a-kind non-physical state override mechanism to the solver whenever a need for a particular type of control arises. Our cloth system has several dozen such controls. But there are considerable costs: stability analysis becomes more difficult, and in the longer term, such techniques eventually lead to a solver that is difficult to understand and maintain, and a system that is difficult to use. Although we cannot recommend the use of such techniques, in practice we find them difficult to resist.

5. The Dynamics Module

Our dynamics module is a plugin that provides the interface between our general animation system and the solver. It also provides a graphical user interface and scriptable commands for creating dynamic objects, constraints, and force fields, rigging dynamic systems, and controlling the simulation parameters.

A description of a general animation system is well beyond the scope of this paper. We assume that it provides a scene representation and a file format, a robust scene dependency graph, a graphical user interface for display, manipulation, and control of 3D objects, a scripting command language, and a plugin architecture (a software API) that provides complete and efficient programming access to all of the above. Commercially available systems like [1] and [27] loosely fit this description. We use our own proprietary animation system.

When the dynamics module is loaded by the application, it inserts operators (nodes) into the dependency graph for every dynamic object in the scene. It also introduces new types of objects specific to simulation into the scene, such as dynamic constraints, force fields, and objects containing solver attributes.

When the system wants to draw the scene at a given time (figure 5) the following happens:

1. The drawing method asks each object in the scene for its position and shape. This triggers the evaluation of the scene dependency graph.
2. One of the dynamics operators eventually gets evaluated. This causes evaluation of all other dynamics operators solved by the same solver.
3. Each dynamics operator evaluates the current state of its dynamic object. It converts the shape of the object into an appropriate spring-mass configuration and passes it to the solver, along with the collision geometry and values of all intersting attributes of the dynamic object.
4. The solver receives the data from all dynamic objects, constraints, and force fields, and it starts integrating the system up to the current animation time using some number of time steps.
5. Upon completion, the resulting state is retrieved by the dynamics operators, which cache the data for efficiency, and then pass it further down the dependency graph.
6. The drawing method eventually gets its data and draws the objects in their simulated position and shape.

This process is repeated for every frame of animation. There are two important observations about this architecture:

- The animation systems knows nothing about dynamics.
- The solver knows nothing about the animation system or the dynamics module. The whole interaction with the solver is handled entirely by the dynamics module through the public API of the solver.

This separation allows us to write code at each level that is as compact, efficient, reusable, and easy to understand as possible.

6. Discussion and Results

Implementation of the above architecture is not a small endeavor. While it may take a skilled programmer several days to implement a toy solver, or several months to implement a full-featured solver with a a well-structured API and a dynamics module, the implementation of a modern general animation system is probably beyond the means of any single programmer.

But it is not necessary to have a full-fledged animation system to have fun with dynamics. To support this claim we implemented a very simple standalone application based on the core techniques from our solver and added a trivial Open GL (GLUT) interface. Figures 2 and 3 are examples from this simple application.

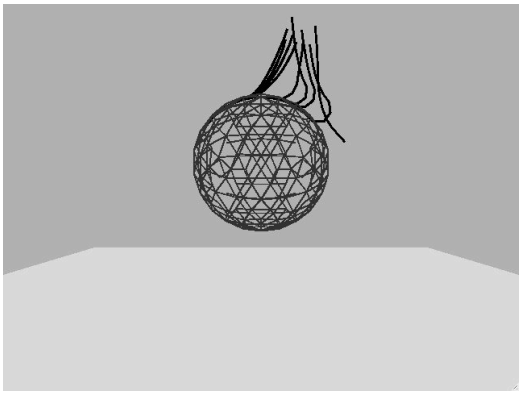

Figure 2: *Interactive manipulation of 10 strands of spaghetti as they collide against a spherical analytical volume. Each strand consists of 13 segments connected by 23 springs. All springs have stiffness of 2000, damping of 0.7 (where 0 means no damping and 1 is critical damping), and there is also ether drag applied to all points. 100 time steps are computed per frame at 33 frames per second on a commodity 1.8 GHz Pentium 4 system.*

There are two important points about the cube example in figure 3. The undamped animation demonstrates preservation of high frequences. The damped animation shows that we can get a rigid behavior from very stiff highly damped spring systems. For all practical purposes the damped cube is equivalent to a rigid body simulation. But building complex structures from springs alone may be unnecessarily expensive both in terms of computation and modeling time, so rigid body dynamics still has a future. In a sense, simulation of rigid bodies can be viewed as an optimization technique.

Another commonly used optimization is *clustering* of rigid bodies (figure 6). A cluster is a collection of rigid bodies that is treated as a single rigid body. Clusters can be created and broken at any time under the control of the user. While the same is achievable using animated rigid constraints between rigid bodies, clusters provide a much more effective interface for the user.

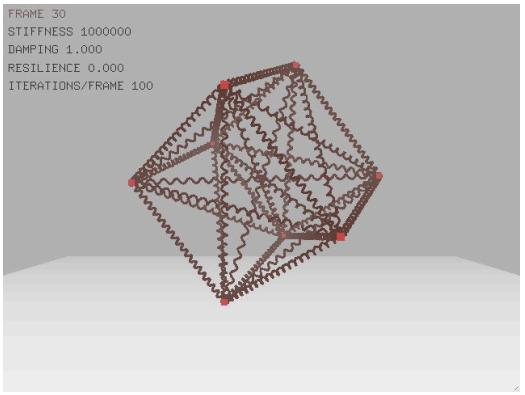

Figure 3: *A 3D structure (cube) made of 8 point masses (weighing 29 kg each) and 28 very stiff springs falling under the influence of gravity and colliding with the floor. In the video we show a highly damped (stiffness = 1,000,000; spring damping = 1; 100 iterations/frame at 256 frames/second) and an undamped configuration (stiffness = 200,000; spring damping = 0; 1000 iterations/frame at 29 frames/second), ran on a commodity 1.8 GHz Pentium 4 system.*

The concept of *animation layering* is not a part of the design of the system. It is how the system is used. In the example in figure 6, the spaceship is very large and it has a low "natural frequency". Its overall motion in response to explosions is slow and wobbly, not rigid at all. This part of animation is done first, using large time steps and soft constraints. Simulations run at nearly interactive speeds and provide very quick feedback to the animator. Once this part of animation is done, the simulation results are *baked*, that is, converted into keyframe animation, and no longer actively simulated. As parts of the ship break off, smaller more rigid pieces are simulated separately, either in small groups (if they interact with each other) or individually, using smaller time steps and stiffer constraints. Since fewer objects are simulated, the system still runs at nearly interactive speeds and provides a tight feedback loop. The animator reports that at no time did the system take more than 5 seconds per frame, and averaged under 2 seconds per frame on an ancient 300 Mhz R10000 SGI O2.

Finally, in figure 7 we demonstrate a concept of deformable "rigid" bodies. It is an inexpensive way of denting the geometry of the rigid body during collisions. A user specified fraction of the collision force is used to displace the colliding points. The rest of the force is still applied rigidly on the body, or is dissipated through friction and damping. The "softness" map that controls the the ease of deformation can the painted on the geometry of the rigid body. It is

important that after each deformation the solver recomputes the center of mass and the inertia tensor of the rigid body.

The most recent application of our system that combines traditional key framed character animation, motion capture, and rigid body dynamics to create extremely complex, convincing performances for "The Hulk" is demonstrated in [18].

7. Conclusions and Future Work

Dynamic simulations are at the core of many serious production tools, and they are also fascinating to play with. Developing dynamics systems is a complex task, but robust, effective, and simple dynamics solvers should be within reach of most developers and researchers.

No animator has yet complained that dynamics simulations run too fast or are too easy to control. As hardware becomes faster, it will only open the room for more complex and more frequently used simulations. Making simulation techniques intuitive, fast, easy to control, and easy to implement will remain the goal for the foreseeable future, as will breaking the boundaries between different types of simulation.

Acknowledgements

Many thanks to the entire R&D software team at ILM, particularly Jim Hourihan and John Anderson who worked on the original version of our rigid body dynamics system, Ron Fedkiw who kept us on track with numerical algorithms, and Jeff Yost and Florian Kainz who led the design of our animation system.

We would also like to thank the technical animators and the ILM's rigid body simulation team who for years have been pushing the development of the system from the users' side, particularly James Tooley, Scott Benza, and Hiromi Ono.

Steve Sauers was the technical animator for the droid slicing shot (figure 1) and the Jeep example (figure 7), Juan Luis Sanchez for the Geonosian arena shot (figure 4), Michael Balog for the destroyer droid shot (figure 5), and Paul Kavanagh for the Naboo cruiser explosion shot (figure 6),

References

1. Alias|Wavefront. *Maya 1.0, Using Maya*. Toronto, Ontario, Canada, 1998. www.aliaswavefront.com.
2. ILM's Technical Animators. Private conversations, 2002.
3. David Baraff, Andrew Witkin, and Michael Kass. Untangling cloth. *ACM Transactions on Graphics (SIGGRAPH Proceedings)*, 2003. To appear.
4. David Baraff and Andrew P. Witkin. Large steps in cloth simulation. In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, pages 43–54, Orlando, Florida, July 1998. ACM SIGGRAPH / Addison Wesley. ISBN 0-89791-999-8.
5. Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. *ACM Transactions on Graphics (SIGGRAPH Proceedings)*, 21:594–603, 2002.
6. Robert Bridson, Sebastian Marino, and Ronald Fedkiw. Simulation of clothing with folds and wrinkles. In *Eurographics/SIGGRAPH Symposium on Computer Animation*, San Diego, California, 2003.
7. Johnny T. Chang, Jingyi Jin, and Yizhou Yu. A practical model for hair mutual interactions. In *Proc. ACM SIGGRAPH Symposium on Computer Animation*, pages 77–80, 2002.
8. Kwang-Jin Choi and Hyeong-Seok Ko. Stable but responsive cloth. *ACM Transactions on Graphics (SIGGRAPH Proceedings)*, 21:604–611, 2002.
9. Mathieu Desbrun, Peter Schröder, and Alan Barr. Interactive animation of structured deformable objects. In *Graphics Interface*, pages 1–8, 1999.
10. Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. Adaptively sampled distance fields: a general representation of shape for computer graphics. *Computer Graphics (SIGGRAPH Proc.)*, pages 249–254, 2000.
11. Herbert Goldstein, Charles Poole, and John Safko. *Classical Mechanics*. Addison-Wesley, 2002. (3rd edition).
12. Eran Guendelman, Robert Bridson, and Ronald Fedkiw. Nonconvex rigid bodies with stacking. *ACM Transactions on Graphics (SIGGRAPH Proceedings)*, 2003. To appear.
13. Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration, Structure-Preserving Algorithms for Ordinary Differential Equations*. Springer, 2002.
14. Ernst Hairer and Gerhard Wanner. *Solving Ordinary Differential Equations II, Stiff and Differential-Algebraic Problems*. Springer, 2002. (2nd edition).
15. Donald H. House and David E. Breen, editors. *Cloth modeling and animation*. A. K. Peters, 2000.
16. J. D. Lambert. *Numerical Methods for Ordinary Differential Systems, The Initial Value Problem*. John Wiley and Sons, 1991.
17. N. M. Newmark. A method of computation for structural dynamics. *ASCE J.of the Engineering Mechanics Division*, 85:67–94, 1959.

18. Hiromi Ono, Scott Benza, and Zoran Kačić-Alesić. Bringing digital crash dummies to life for 'the hulk'. In *SIGGRAPH Sketches and Applications*, San Diego, California, 2003. ACM SIGGRAPH.

19. Stanley Osher and Ronald Fedkiw. *Level set methods and dynamic implicit surfaces*. Springer-Verlag, 2002. New York, NY.

20. Ronald N. Perry and Sarah F. Frisken. Kizamu: a system for sculpting digital characters. *Computer Graphics (SIGGRAPH Proc.)*, pages 47–56, 2001.

21. Jovan Popović, Steven M. Seitz, Michael Erdmann, Zoran Popović, and Andrew Witkin. Interactive manipulation of rigid body simulations. *Computer Graphics (Proc. SIGGRAPH)*, pages 209–217, 2000.

22. William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C, The Art of Scientific Computing, 2nd Edition*. Cambridge University Press, 1992.

23. Xavier Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Graphics Interface*, pages 147–154, May 1995.

24. Ari Rapkin. How to dress like a jedi: techniques for digital clothing. In *SIGGRAPH Conference Abstracts and Applications*, Computer Graphics , Annual Conference Series, page 196, San Antonio, Texas, 2002. ACM SIGGRAPH.

25. Bill Seneshen and Luca Prasso. Clothing simulations in shrek. In *SIGGRAPH Conference Abstracts and Applications*, Computer Graphics , Annual Conference Series, page 190, New Orleans, Louisiana, 2000. ACM SIGGRAPH.

26. Raymond A. Serway. *Physics For Scientists and Engineers*. Saunders College Publishing, 1990. (3rd ed.).

27. Softimage. *XSI*. Montreal, Canada, 2002. www.softimage.com.

28. Gilbert Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, 1986.

29. Toshiko Tajima. *Computational Plasma Physics: With Applications to Fusion and Astrophysics*. Addison-Wesley, 1989.

30. Loup Verlet. Computer experiments on classical fluids, thermodynamical properties of lennard-jones molecules. *Physical Review*, 159:98–103, 1967.

31. Pascal Volino and Nadia Magnenat-Thalmann. Comparing efficiency of integration methods for cloth simulation. In *Proc. Computer Graphics International*, pages 265–274, 2001.

32. Andrew Witkin, David Baraff, and Michael Kass. Physically based modeling. *Siggraph Course Notes #25*, 2001.

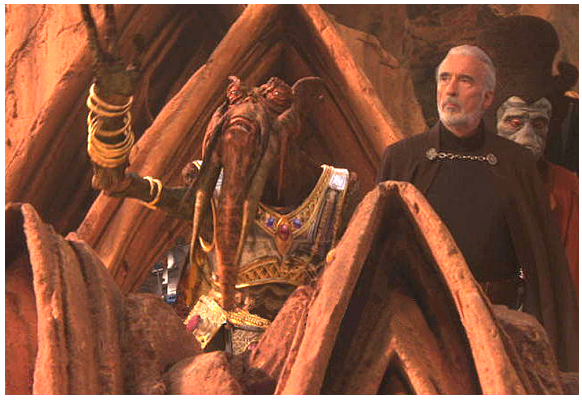

Figure 4: *A simulation of bracelets on a keyframe animated arm from the Star Wars Episode II: Attack of the Clones. Only collision forces and gravity affect the bracelets, as well as the simulated medallion and the chain on the armor.*

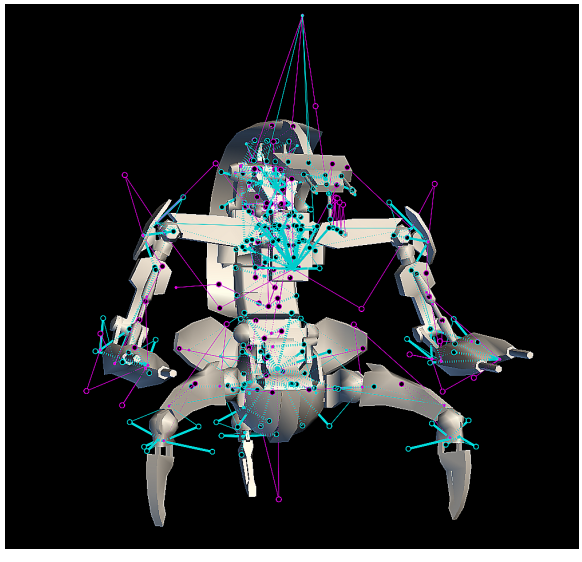

Figure 5: *A complex rigid body rig (Destroyer Droid from Star Wars Episode II) as it appears in a 3D view of the dynamics module. The geometry of the rigid bodies is shaded, constraints are drawn in cyan and purple, the red sphere (in the video only) is an air field. Simulation starts after 15 frames of keyframe animation.*

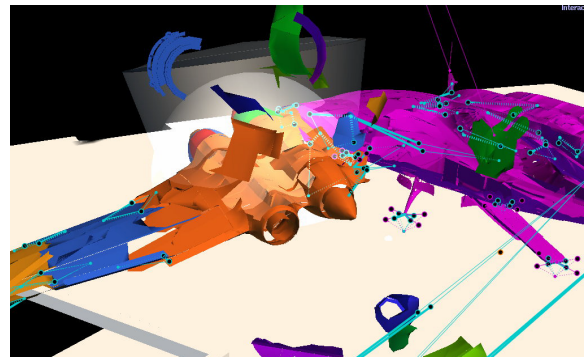

Figure 6: *A complex rigid body simulation of the spaceship explosion (Naboo Royal Cruiser from the opening sequence of the Star Wars Episode II). The explosion was choreographed by the art director well before any simulation started. The animator had to match the choreographed progression of explosions and the gradual ship breakup. This illustrates the use of air fields (translucent white cone and sphere), clusters (differently colored parts of the ship), animated constraints (purple and cyan lines and dots), and the concept of animation layering.*

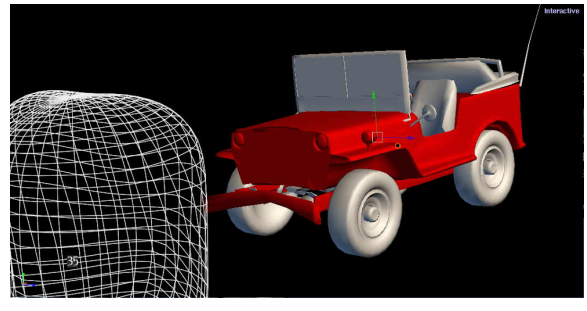

Figure 7: *Deformable "rigid" bodies (an early test from "The Hulk" production). The initial velocity of the vehicle comes from the keyframe animation. The simulation starts just before it hits the "rock". Everything is simulated automatically from that point on, including the dent in the front of the vehicle.*