Dynamic Real-Time Deformations using Space & Time Adaptive Sampling

Gilles Debunne * iMAGIS-GRAVIR
Mathieu Desbrun U. of So. Cal.
Marie-Paule Cani iMAGIS-GRAVIR
Alan H. Barr Caltech

Abstract

This paper presents a robust, adaptive method for animating dynamic visco-elastic deformable objects that provides a guaranteed frame rate. Our approach uses a novel automatic *space and time adaptive* level of detail technique, in combination with a large-displacement (Green) strain tensor formulation. The body is partitioned in a non-nested multiresolution hierarchy of tetrahedral meshes. The local resolution is determined by a *quality condition* that indicates where and when the resolution is too coarse. As the object moves and deforms, the sampling is refined to concentrate the computational load into the regions that deform the most. Our model consists of a continuous differential equation that is solved using a local explicit finite element method. We demonstrate that our adaptive Green strain tensor formulation suppresses unwanted artifacts in the dynamic behavior, compared to adaptive mass-spring and other adaptive approaches. In particular, damped elastic vibration modes are shown to be nearly unchanged for several levels of refinement. Results are presented in the context of a virtual reality system. The user interacts in real-time with the dynamic object through the control of a rigid tool, attached to a haptic device driven with forces derived from the method.

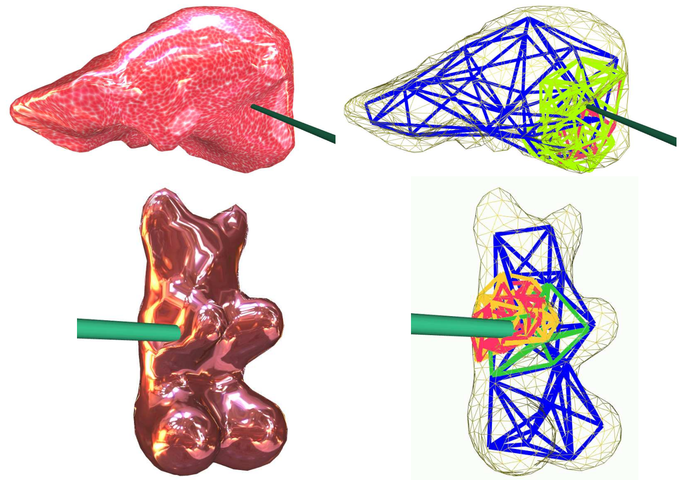


Figure 1: *Our real-time multiresolution model allows for a wide range of applications, from virtual surgery simulators (top: liver laparoscopic operation) to other immersive simulations (bottom: toy example). The approach makes use of a local refinement technique to ensure high physical fidelity while bounding the global computation load to guarantee real-time animations.*

1 Introduction

Animating deformable objects in real-time is essential to many interactive virtual reality applications, such as surgery simulators or video-games. An important point for a successful immersion is the liveliness of objects: deformations should be *dynamic* (oscillations should appear after a deformation for example), and not just a succession of static postures. Another essential point is to make a strict guarantee for real-time. Merely satisfying visual and tactile fusion frequencies (30 images, 1000 force samples per second) is not sufficient to prevent lag or slow motion in the animation. In addition, the simulation time must *always* be synchronized with the physical time, regardless of computational platform.

Although quasi-static interactive simulators have been proposed in the last few years [3, 20], computing accurate dynamic deformations in real-time is still a challenge. This paper proposes a solution to this problem, using an adaptive physically-based model which is locally animated at different levels of detail.

1.1 Related work

Background on deformable models

Many approaches have been developed for animating deformable objects (see [17] for a thorough overview), but only few models can be used when aiming at real-time performance.

Global deformation models have been designed for interactive animation, but they restrict deformations to the combination of a given set of vibration modes, or of a specific class of global deformations [23, 26]. They are less useful, however, when realistic deformations are called for.

Particle and mass-spring systems are based on a local description of the material. These systems allow for large deformations and displacements, and are fairly easy to implement. The equations of motion are integrated independently for each particle, which generally leads to fast calculations. When the spring stiffness is large, *implicit* integration can be used to animate hundreds of mass points efficiently [1], and even in real-time with some approximations [10].

Deformable models with greater physical accuracy have been derived from elasticity theory [12] using finite element methods (FEM). One of the most widely used solutions computes an equilibrium shape of the object from a set of limit conditions [25, 18]. This method requires solving a large sparse system; as a result, it is less compatible with real-time applications. Preinverting the matrix allows for real-time [3], but the object has to be deformed using penalty forces, which often fails to model accurate contact between the object and the tool. The inverse matrix can also be *updated* depending on which nodes are moved by the user, allowing a quasi-static real-time animation of a hundred points [20]. One can also precompute the effects of the displacement of each external point in the three axis directions and then use a linear combination of these deformations during animation [6]. However, superposition problems appear with this technique when several points are moved at the same time.

The second main approach using elasticity theory is sometimes called *explicit* finite elements [9, 22] (the term "explicit" here refers to a *spatial* discretization, and must not be mistaken with the more common notion of *time* explicit integration). The way the animation is performed is quite similar to the mass-spring case, since each node of the FEM mesh dynamically integrates its motion from the positions of neighboring nodes. These methods give more accurate results than mass-spring systems. However, computing local deformations at a visually good discretization level requires a fine sampling that prevents real-time performances, even if interactive

*{debunne|cani}@imag.fr,{mathieu|barr}@cs.caltech.edu

rates can be reached for soft bodies (20 simulation steps per second can be performed for around 4000 mass samples in [9]).

In summary, all the above dynamic techniques simulate a given number of 'mass samples' in the material, at a time frequency that increases with the material stiffness (easily 1000Hz for common materials). A true real-time simulation requires that each simulation step must be computed in less than the time step duration (*e.g.* 1/1000th of a second) to avoid animations with unwanted slow motion or lag, even if only 30 frames a second have to be displayed for visual feedback. Unfortunately, even implicit integration techniques, while allowing larger time steps, cannot currently lead to real-time simulations as soon as the number of mass points exceeds a few hundred points.

However, an attractive way to optimize the computations, maximizing the overall realism while guaranteeing frame rate, is to use a *space and time adaptive technique* as we present in this paper. Similar to level-of-detail (LOD) or hierarchical methods in rendering, the idea is to locally adapt the sampling (hence the accuracy) at which the simulation is computed, depending on the current local deformations. This idea, while extensively used in other domains, has received little attention in physically-based animation until now.

Adaptive animation in physically-based animation

Applying LOD to computer animation consists of using simplified animation techniques when objects are far away, moving rapidly, or out of sight [2, 5]. For instance, a dynamic simulation becomes cinematic, or even a simple particle motion, as an articulated object recedes from the viewer [4]. Observe that in this example the coherence between LODs is limited.

In the case of deformable objects, the same idea can be used to change the number of nodes inside an object, depending on the desired accuracy. Some previous work uses this idea to locally refine a model in regions of interest: a piece of draped cloth represented by masses and springs is refined in regions of high curvature [19]; a volumetric mass-spring network refines near user-controlled cutting lines [13]; a tetrahedral mesh for an explicit finite element simulation of a stiff solid is refined along fracture lines [22] or in predetermined regions [27].

Another step is to design models that allow both for refinement when and where a large deformation occurs, *and* for simplification in stable regions. In [15], this idea is applied to a particle system that simulates a mud flow, using division and clustering of particles that obey a global behavior specified by a SPH-based model. A visco-elastic object was simulated at different local resolutions using finite differences applied over an octree representation in [7]. Our previous work [8] achieved more efficiency by using a finite volume/finite element method over tetrahedral meshes, based on discrete differential operators [11].

For all these models, the main problem consists of ensuring that the deformable model stays self-consistent despite the change of resolution. This is hard to do using standard mass-spring or particle systems, for there is no underlying physical model to refer to in order to find what changes in parameters (stiffness, density, etc) will simulate the same behavior at another resolution. Hutchinson [19] increases the mass each time a refinement is performed. Ganovelli [13] proposes a more convincing solution, where new spring parameters are computed as the closest to an elastic continuous representation, using Van Gelder's stiffness equation [16]. However, we demonstrate in section 3.1 that this solution is not sufficient for avoiding artifacts when several resolutions are combined.

In theory, locally adapting the resolution should work better using a continuous model such as those in [22, 15, 7], since all resolutions are approximations of the same specified behavior. However, we show in section 3 that these models may introduce diverging instabilities when different resolutions are combined. Our results demonstrate that the regions simulated with different discretization rates tend to adopt different vibration modes, which immediately yields instability except when a very strong damping is applied. This analysis is key to understanding our solution to the problem.

1.2 Overview

This paper presents the first robust, adaptive method for animating *dynamic deformations* of a visco-elastic object in *real-time*, with a guaranteed frame-rate. Our model belongs to the physics-based continuous models and is solved using an explicit finite element method. It uses the Green deformation tensor to allow for very large displacements.

To achieve real-time performance, our system automatically adapts the local resolution at which a region of the object is simulated, in order to concentrate the computational load in regions that deform the most. Contrary to previous approaches, we demonstrate that such an automatic sampling adaptivity does not create visible unwanted artifacts in the physical behavior of the object. Lastly, we propose visual and haptic interfaces that make the adaptation of the model transparent to the user. Results are shown in the context of a virtual reality system where the user interacts in real-time with the dynamic object through the control of a rigid tool.

Section 2 describes our method for locally adapting the resolution. Section 3 analyzes the different possible deformable models in the context of adaptive simulation, and concludes with a solution that ensures robustness. Section 4 presents our visual and haptic interfaces. Section 5 analyzes the results by comparing the gain in efficiency the adaptive algorithm achieves during the simulation of different deformable bodies. We finally conclude in Section 6.

2 Local adaptive refinement

We inferred in the previous section that the current best approach towards real-time dynamic deformation is to use an accurate, local computation method. We will thus adopt such an approach: in this section, we only suppose that the motion of a node can be computed from values (positions or displacements) at neighboring nodes. We will describe the formal computations, i.e. the explicit finite element method, in Section 3.

This section presents our method for providing several levels of resolution of an object. We explain how to locally switch between these LODs, and how two neighboring regions simulated at different resolutions can coexist without creating instability.

2.1 Non-nested tetrahedral meshes

We use tetrahedral meshes to represent the deformable body, since they can accurately fit arbitrary geometries. Recursive subdivision of such 3D meshes cannot be used for defining different resolutions: regardless of the subdivision method used, the quality of the initial mesh in terms of angles and/or aspect ratio will be lost after few subdivisions. Relaxation and/or re-meshing processes are incompatible with real-time applications. Some algorithms (usually those based on finite element techniques) require the preservation of a conformal mesh which makes the subdivision process even harder. Moreover, the refinement process may not be invertible and the quality of the mesh may be altered when it comes back to a coarser level.

To avoid these drawbacks, our method relies on arbitrary, independently defined meshes for representing the different levels of detail. Each of the meshes is a quasi-uniform sampling of the 3D shape representing the deformable body at a given resolution. These meshes, although representing the same object, can be completely independent (no vertex needs to be shared), which leaves complete freedom for their generation. In practice, we decimate the surface of the object at different resolutions [14], and use a volume mesh generator to provide a good tetrahedralization of the interior of the simplified meshes. In the remainder of this paper, the term "parent mesh" (*respectively* "child mesh") is used for the meshes that represents the deformable body at the immediately coarser (*resp.* finer) scale.

2.2 Dynamic adaptivity

The idea of our algorithm is to refine or coarsen the geometric model depending on the local approximation quality of the simulation provided by the deformable model. Suppose that this quality criterion indicates that the resolution is too coarse near a given active[1] node P. We then deactivate this mass node, and activate the nodes of the child mesh which sample the same region (Fig 2). Those "children" nodes are naturally chosen as the one closer to P than to any other node from this mesh (*i.e.* inside the Voronoï region of P). As Voronoï regions form a partition of space, all the nodes of the finer level will have a parent, and two parents will not share a child, thus leading to a coherent refinement.

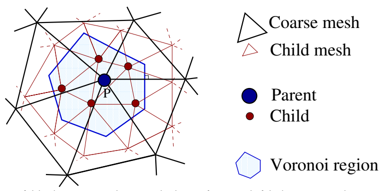


Figure 2: P will be replaced by its children when sampling becomes too coarse in this region.

Similarly, a group of children nodes will be replaced by their parent when the quality criterion indicates that a coarser sampling would be sufficient in this region. The child-parent relationship between the different meshes is precomputed and stored, so that no expensive tests of inclusion in Voronoï regions have to be performed during simulation.

2.3 Interface between different simulated LODs

Since the adaptation of the level of refinement is local, some neighboring nodes of a given active node may not be active themselves (we call them "interface nodes"). This is a problem since, according to the deformable model we use, each active node uses either the position or displacement of the neighboring nodes to compute its motion. A simple interpolation can be used to solve this problem: the information at an interface node is interpolated from the information of the currently active tetrahedron in which it is located. This tetrahedron may either belong to a coarser or to a finer refinement level (see Fig. 3). This idea is somewhat similar to the domain decomposition used in FEM [24].

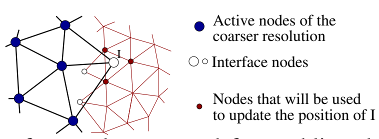


Figure 3: Interface nodes are used for enabling the joint use of different resolutions during simulations (2D view, coarse mesh is black, finer mesh is red).

We currently use linear interpolation, based on the barycentric coordinates of the interface node in the tetrahedron to update the values at interface nodes. We will see in the next section that this interpolation is coherent with our model. Since such a computation is negligible compared to a force computation, our interface between different resolutions does not impede the performance of our method, even when interpolation is performed on a large number of interface nodes.

3 Space and time adaptive model

This section describes our choice of deformable model. As stated earlier, we want to compute the dynamics of a visco-elastic object, possibly undergoing a large deformation. To be compatible with the adaptive refinement method we have just described, the model should be sufficiently accurate so that computations remain compatible in neighboring regions of different resolutions. It is not sufficient that the different resolutions approximate the same static behavior: if regions simulated at different resolutions tend to vibrate at different frequencies and amplitudes, interference at the interfaces between these regions will quickly lead to numerical instability.

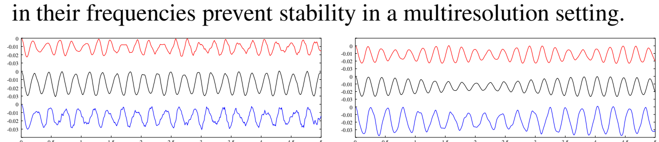
The first part of this section compares and discusses the way the previous deformable models handled our adaptive simulation algorithm. We then describe the chosen model and explain how it was implemented in the adaptive animation framework.

3.1 Comparative analysis of deformable models

Modus Operandi of the comparative study Before deciding on a model, we implemented and tested multiple adaptive deformable models on the same testbed. We studied the results they give on a simple example: the oscillations of a cube of elastic material under gravity, one of its sides being attached to a vertical wall. At time zero, the cube is released from its horizontal position. For each model, we performed the cube simulation at three different resolutions (27, 57 and 137 nodes, plotted in red, black and blue respectively). The curves presented in the remainder of this section plot the vertical oscillations of one of the cube corner over time. In this test example, neither internal nor external damping forces are used. This allows us to easily compare the vibration modes resulting from each of the resolutions.

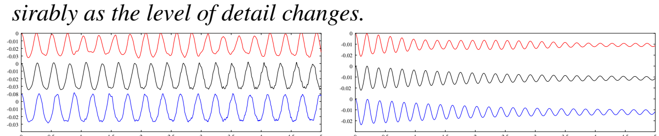
Masses and springs Springs are located along the edges of the tetrahedral mesh. Their stiffness is computed as in [16, 13]. Fig. 4a depicts the results and shows that mass-spring systems clearly fail to ensure the same amplitude and frequency of oscillations at different resolutions.

Cauchy strain tensor with explicit linear FE The Cauchy strain tensor was used by [9, 3, 7, 8] and provides a fast computation of each node force. Due to linear approximations, this method has a very simple expression, but is not very robust against large displacements. This is especially true for rotations since it then creates extra forces that deform the object. As shown in Fig. 4b, different vibration modes appear in the simulation. The differences in their frequencies prevent stability in a multiresolution setting.



a b

Figure 4: *Multiresolution test: vertical position of an oscillating cube over time, using mass-springs (a) or Cauchy FE (b). The cube is sampled at different resolutions, using 27 nodes (red), 57 nodes (black) or 135 nodes (blue). Note that the behavior changes undesirably as the level of detail changes.*



a b

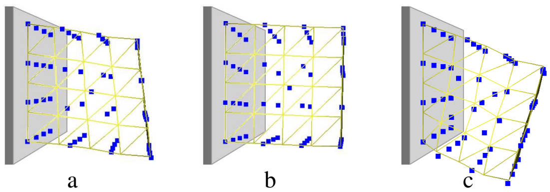
Figure 5: *Multiresolution test with 27, 57, and 135 nodes (as in Fig. 4 and 6) using Green strain tensor. Note the closer resemblance in behavior between the different levels of detail (a), even when damping is applied (b).*

Our previous work [8] used a simplified version of this tensor which had better multiresolution properties at the expenses of simplified dynamic behavior (see Fig. 6).

Green strain tensor with explicit FE The Cauchy tensor is a first order approximation of the Green strain tensor that was used in [25, 22]. Computationally more complex, this method handles large displacements, including global rotations. Its dynamic behavior is more interesting, exhibiting complex motions (Fig. 6).

[1] We call a node *active* if it computes and integrates a force according to its neighbors' positions

Fig. 5 demonstrates the good multiresolution behavior of this method, which justifies its choice for our simulations. Similar multiresolution simulations are achieved when damping is added (Fig. 5b), which is needed for realistic simulations.



a b c

Figure 6: *Deformed shapes for the cube obtained using Cauchy (a), simplified Cauchy (b) and Green (c).*

3.2 The Green deformable model

Strain, stress and deformation law

At each point of the material, local deformations are modeled using a *strain tensor* ϵ. The Green strain tensor is expressed by a 3×3 matrix. Its (i,j) coefficient is:

$$(\epsilon)_{ij} = \left(\frac{\partial \mathbf{x}}{\partial u_i} \cdot \frac{\partial \mathbf{x}}{\partial u_j} \right) - \delta_{ij} \tag{1}$$

where δ_{ij} equals 1 if $i = j$ and equals 0 otherwise ("Kronecker delta"). $\mathbf{u}$ is an $\Re^3$ vector which denotes a location in the material coordinates. $\mathbf{x}(\mathbf{u})$ represents the point's position in world coordinates as a function of its material coordinates. Those two coordinate systems match when the object is not deformed, resulting in a null strain tensor. $\mathbf{x}(\mathbf{u})$ is continuous and evolves according to the object's deformations. The material coordinates $\mathbf{u}$ of a point mass are fixed, but its position in space $\mathbf{x}$ will vary over time.

The *stress tensor* σ represents the force distribution inside the material. It can also be represented with a first order approximation as a 3×3 matrix. The force $\mathbf{f}$ acting on an elementary surface dS, with an outward unit normal $\mathbf{n}$ is: $\mathbf{f} = \sigma\, \mathbf{n}\ dS$.

Linear elasticity assumes that stress and strain are linearly linked (as for a spring). If we assume that our material is isotropic, symmetry considerations state that only two independent coefficients describe the material behavior:

$$\sigma \;=\; \lambda\, \mathrm{tr}\,(\epsilon)\, \mathcal{I}_3 \;+\; 2\,\mu\, \epsilon \tag{2}$$

where $\mathcal{I}_3$ being the identity 3×3 matrix and $\mathrm{tr}\,(\epsilon)$ is the trace of ϵ. μ and λ are the Lamé coefficients: μ represents the rigidity of the material while λ measures its "ability" to preserve volume (incompressible materials should have an infinite λ).

Damping forces

Damping is very simply modeled, and coherent with the previous tensors. The *strain rate tensor* ν is the time derivative of ϵ (Eq. 1):

$$\nu = \frac{\partial \epsilon}{\partial t}, \qquad (\nu)_{ij} = \left(\frac{\partial \mathbf{v}}{\partial u_i} \cdot \frac{\partial \mathbf{x}}{\partial u_j} \right) + \left(\frac{\partial \mathbf{x}}{\partial u_i} \cdot \frac{\partial \mathbf{v}}{\partial u_j} \right) \tag{3}$$

where $\mathbf{v} = \frac{\partial \mathbf{x}}{\partial t}$ is the velocity of a point.

The total stress tensor σ is then updated using a similar equation as for ϵ:

$$\sigma \mathrel{+}= \phi\, \mathrm{tr}\,(\nu)\, \mathcal{I}_3 \;+\; 2\,\psi\, \nu \tag{4}$$

where ϕ and ψ control how fast the material looses kinetic energy. Rigid motions of the object will not be damped by this formulation ($\nu \equiv 0_3$), which only dissipates internal vibrations.

Explicit Finite Elements

Finite Element techniques partition the object into elements on which the physical equations are expressed . Instead of merging all these equations in a large matrix system, *explicit* FEM solve each element independently through a *local* approximation, dramatically reducing computational time. We choose to use tetrahedral elements with linear basis functions as they seem to offer the best trade-off between speed and efficiency. For each element $\mathcal{E}$, the linear basis function $a^i x + b^i y + c^i z + d^i$ for a node i can be represented [2] as a vector $\mathbf{L^i} = (a^i\, b^i\, c^i\, d^i)$. Applying $\mathbf{L^i}$ to a point (its world coordinates) inside the tetrahedral element $\mathcal{E}$ gives this point's i^{th} barycentric weight. The weight of a node i equals 1 for node i and 0 for the other nodes:

$$\mathbf{L^i} \cdot \begin{pmatrix} \mathbf{u^j} \\ 1 \end{pmatrix} = \delta_{ij} \quad \forall\, i, j \in 1..4 \tag{5}$$

For each i, these four equations ($j \in 1..4$) form a linear system which can be inverted (since all the tetrahedra used in the rest position are not degenerated). The resulting interpolation functions $\mathbf{L^i}$ will allow us to compute material derivatives.

Any vector field $\mathbf{c}$ can be linearly interpolated over the element using the barycentric coordinates. For an interior point with material coordinates $\mathbf{u}$, the value of $\mathbf{c}(\mathbf{u})$ is:

$$\mathbf{c}(\mathbf{u}) = \sum_{i=1}^{4} \mathbf{c^i}\ \ \mathbf{L^i} \cdot \begin{pmatrix} \mathbf{u} \\ 1 \end{pmatrix} \tag{6}$$

The material derivatives are then expressed by:

$$\frac{\partial \mathbf{c}}{\partial u_j} = \sum_{i=1}^{4} \mathbf{c^i}\ \ (\mathbf{L^i})_j \tag{7}$$

Applying this equation to the position or speed vector allows us to compute the ϵ, ν and σ tensors (Eq. 1 & 3).

The force on node i that results from the deformation of element $\mathcal{E}$ is expressed by (see [22]):

$$\mathbf{f^{\,i}} = -\frac{vol^{\,\mathcal{E}}}{2} \sum_{j=1}^{4} \mathbf{x^{\,j}} \sum_{k=1}^{3} \sum_{l=1}^{3} \ (\mathbf{L^i})_k\ (\mathbf{L^j})_l\ (\sigma)_{kl} \tag{8}$$

where $vol^{\,\mathcal{E}}$ is the volume of $\mathcal{E}$. Each mesh node then simply sums the contributions of all the tetrahedra it belongs to.

3.3 Adaptive resolution with our model

Adapting space resolution The basics of the multiresolution algorithm were described in Section 2. As we use linear basis functions for our FE model, we choose to define a *quality criterion* which represents the adequacy of these piecewise linear functions to the current deformation state.

The maximum difference between the linear and the quadratic term of the displacement field is indicated by $\Delta\mathbf{d}\ h^2$, where h is the minimum distance between a node and its neighbors and $\Delta\mathbf{d}$ is the laplacian of $\mathbf{d}$. This value is our discontinuity criterion γ, indicating how far $\mathbf{d}$ is from a linear function: we will then adjust the number of local samples to ensure a good linear approximation of the deformations.

One can show [7] that the force $\mathbf{f}$ per volume V is directly linked to the laplacian $\Delta\mathbf{d}$ of the displacement field for almost incompressible objects: $\mathbf{f} \approx V\, \mu\, \Delta\mathbf{d}$. Hence:

$$\gamma = \Delta\mathbf{d}\ h^2 \approx \frac{\mathbf{f}\, h^2}{\mu V} = \frac{\mathbf{f}}{\mu h} \tag{9}$$

Computed at each active node, γ gives a fast and accurate measure of the inadequacy of the node. Two thresholds, γ_{min} and γ_{max}, control splitting (needed when the discontinuity is too high) and merging (if all the children are varying linearly enough), ensuring an adequate linear approximation everywhere. The use of two different values instead of a single threshold creates an hysteresis preventing repetitive oscillations between two resolutions.

[2] Superscript denote the value at a given node ($\mathbf{c}^{\,\mathbf{i}}, \mathbf{u}^{\mathbf{j}}$) while subscripts denote vector component ($u_1 = u_x, u_2 = u_y, u_3 = u_z$).

Time stepping The Courant condition ensures a proper simulation of a wave propagating inside the object. It provides the maximum time step allowable for a given particle:

$$dt < h \sqrt{\frac{\rho_0}{\lambda + 2\,\mu}} \tag{10}$$

where ρ_0 is the material's rest density. In practice we use time steps that are inverse powers of 2 of the display frequency: $dt^i = dt/2^i$ to ensure the synchronization of simulation and display. Each particle will use the largest time step satisfying this criterion. A particle can also momentarily reduce its time step if its speed increase is too high, to allow for a better numerical integration.

A simple modified Euler integration scheme gives very good results on our examples. However, the local nature of our explicit FE scheme permits us to make the scheme *semi-implicit*. Similarly to [10], we can precompute the force jacobian matrix $\mathcal{J} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}$. It represents a first order approximation of the force applied on a node when it moves. We compute this matrix for each node in the object's rest position, assuming that its neighbors are fixed. As in [10], if we assume that this matrix is constant during the simulation, we can filter the integration scheme at almost no cost. Semi-implicit integration simply consists in the multiplication of a precomputed 3×3 matrix with the speed vector $\mathbf{v}$, resulting in the following integration scheme:

$$\mathbf{v} \mathrel{+}= \frac{\mathbf{f}}{m}\,dt \qquad \mathbf{v} \mathrel{*}= (\mathcal{I}_3 - \frac{dt^2}{m}\mathcal{J})^{-1} \qquad \mathbf{x} \mathrel{+}= \mathbf{v}\,dt \tag{11}$$

This *semi-implicit* integration scheme allows us to multiply the time step by a factor 2. It also adds artificial damping, which can be partially compensated (if not desired) by reducing ϕ and ψ (see Eq. 4).

```
for each frame                                   // infinite loop, 30Hz
    for each simulation step                     // 2^{nb_dt_level} steps, ~1000Hz
        for each particle whose dt_level matches the simulation step
            ComputeForce                         // Section 3.2
            IntegrateForce                       // Equation (11)
            UpdateInterfaceNodes                 // Section 2.3
            PossibleSplitMerge                   // Section 3.3
        ReturnFeedbackForce                      // Section 4.2
    CollisionProcessing                          // Section 4.2
    DisplayScene                                 // Section 4.1
```

Figure 7: *Pseudo-code of our algorithm.*

Guaranteed fame rate For immersive applications like surgery simulations, real-time animation must take precedence over precision. With our multiresolution framework, we can tune the cost/precision ratio, and guarantee a true real-time simulation. The machine load, updated at each display, conditions the addition of new sample points. When the limits of the machine are reached, the new samples will not be created until some merging has occurred. As there are no iterative methods or convergence criteria used in our technique, we therefore guarantee real-time computations.

4 Visual and haptic interfaces

Providing convincing interfaces is important in virtual reality applications, where the user can see, but also feel the virtual object (s)he is interacting with. The adaptation of the deformable model has to be made totally transparent to the user.

4.1 Animation of an external surface

Rather than displaying a triangulation of the external active nodes, which would produce popping artifacts when the resolution changes, we prefer always displaying the same surface, at a given fixed resolution. It can either be the surface of one of the meshes, or another completely independent surface.

To animate the surface whatever the underlying resolution, we have to attach surface points to the animated mesh nodes. This is done by precomputing, at each level of resolution r, the offset between each of the surface points and the closest point $\mathcal{C}^r$ on the closest external triangle formed by the inside FE nodes. During animation, only one given resolution is active near a surface node, and the surface is then positioned from the most appropriate active triangle, $\mathcal{C}^r$'s position being computed from its barycentric coordinates in the triangle.

Our tests and video sequences show that this does not create visual artifact: the surface motion is visibly continuous, and the adaptive refinements occuring inside are totally transparent to the user.

4.2 Force feedback

Our deformable model runs in a force-feedback framework, where the user interacts by moving a rigid tool. The haptic interface we use is a Phantom desktop. A device of this kind requires response forces to be computed at a high frequency (above 400 Hz), otherwise the effect will not be natural for the user. This is not a problem with our adaptive method, since a high resolution mesh, simulated with small time steps, is always used near the rigid tool when there is a collision (see Section 3.3).

We detect collisions using the graphics hardware [21]: the tool, of a simple shape, is modeled as a viewing frustum with which the surface of the deformable object intersects. To save time, collision detection is only performed before each display, ensuring a coherent visual collision.

Computing penalty forces as in [3, 22] would fail to model a surface of contact between the tool and the deformable object. Instead, our approach consists of computing the object deformation from surface displacements imposed by the motion of the tool. After the collision detection process builds the list of surface polygons intersecting the tool, the affected surface points are moved along their normal direction away from the tool. Since we do not detect collisions at each simulation step, we apply this motion gradually, so that the surface gets out of the way of the tool just before the next frame. This prevents discontinuities in the response force, due to surface oscillations between the inside and the outside of the tool.

While the surface points move, we compute a correct response force from the inner physical model: at each simulation step, the active mesh nodes that are linked to an affected surface point are also moved. This is easily achieved using the offsets and the weights we used to define their respective position in Section 4.1. Then, rather than integrating the force applied by the rest of the deformable object, each affected mesh node *transmits* this force to the surface point it is linked to. Every surface point sums these forces from its associated mesh points. Finally, the force on a surface point is weighted by the area that it samples in the collision surface; the weighted forces are then summed and returned to the tool.

5 Results

As all our tests and video sequences demonstrate, our method allows true real-time manipulations of deformable objects. We animate a few hundred points in real-time on an 800MHz PC at frequencies varying between 300 and 5000Hz, depending on the object's stiffness and on the particle dt level. The faster the CPU, the more our model will refine in deformed regions. The behavior of the objects can be tuned using the physical coefficients (see Eq. 2 and 4). They exhibit a realistic response to the user's action, with intuitive deformations and nice oscillations when the tool is removed. Typical values for μ and λ range from 10 to 10^6 in our tests. Fig. 8 shows some snapshots of our animations, using the method described in this paper.

The speedup factor between our multiresolution simulation and a classical animation technique using a fixed fine discretization ranges between 5 and 20 on our simple examples. Our multiresolution scheme takes advantage of the best available discretization rate offered by fine meshes while providing a real-time application that would have been impossible otherwise.

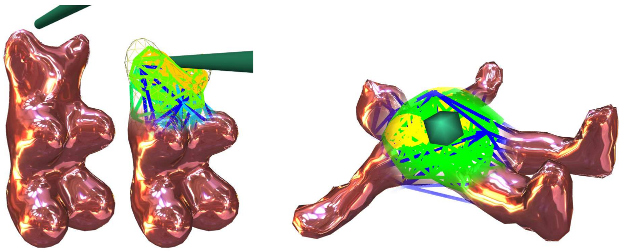


Figure 8: *Changing the material's physical coefficients allows the simulation of a wide scope of material behaviors. Blue, green and yellow indicate increasing levels of detail.*

6 Conclusion and discussion

We have developed a robust, real-time, adaptive method for dynamic simulations. This method handles large displacements, is still stable even when no damping is applied (when most of the previous work on adaptive methods added a lot of viscosity to avoid numerical instabilities), and thus allows us to simulate many different kinds of materials. We maintain a guaranteed frame rate, which ensures true real-time animation. Our new approach relies on adaptive simulation to reach such performances: the space and time sampling is automatically adapted locally to concentrate the computational effort where and when it is needed the most. The benefits of our approach are illustrated by its use in a virtual reality application, where the user interacts with a deformable body through a rigid tool. Our results show that the adaptation of the model resolution can be made totally transparent to the user.

Previous interactive methods [3, 6, 20], in addition to being restricted to quasi-static deformations, were not able to handle topology changes (fractures, cuts): their intensive precomputations are based on a fixed mesh topology. Our method belongs to the class of local approaches: fractures and cuts could be computed by locally refining our model in the regions of interest, as in [22, 9, 13]. Connections between some of the mesh nodes would then be suppressed, to model the cut. However, implementing such topological changes in an adaptive framework is more difficult than with a single resolution: some of the tetrahedra in the cut region will only partially be cut, which has to be modeled correctly. A simple, but probably not entirely satisfying solution would be to no longer activate the coarse meshes in the cut region. Another problem is the surface interface. Here, the best solution would probably be to display the surface of the finest mesh in the cut region. Future work also includes rapid detection of self-collision, which is relatively expensive in cut areas. Finally, other adaptive methods, based on Catmull-Clark-like subdivision for instance, should be investigated.

Acknowledgment

The author would like to thank Eitan Grispun, Zoë Wood, and Mark Meyer for proof-reading and improving this paper. This work has been partially supported by the Rhône-Alpes region (EURODOC grant), the National Science Foundation (ASC-89-20219), as part of the STC for Computer Graphics and Scientific Visualization, and by the Integrated Media Systems Center, a NSF Engineering Research Center, Cooperative Agrement No. EEC-9529152.

References

[1] D. Baraff and A. Witkin. Large steps in cloth simulation. In *SIGGRAPH'98 Conference Proceedings*, Annual Conference Series, pages 43–54. ACM SIGGRAPH, Addison Wesley, July 1998.

[2] R. Berka. Reduction of computations in physics-based animation using level of detail. In *Proceedings of the 1997 Spring Conference on Computer Graphics*, pages 69–76, 1997.

[3] M. Bro-Nielsen and S. Cotin. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. In *Eurographics*, pages 21–30, 1996.

[4] D. Carlson and J. Hodgins. Simulation levels of detail for real-time animation. In *Proceedings of Graphic Interface*, pages 1–8, 1997.

[5] S. Chenney and D. Forsyth. View-dependent culling of dynamics systems in virtual environments. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pages 55–58, 1997.

[6] S. Cotin, H. Delingette, and N. Ayache. Real-time elastic deformations of soft tissues for surgery simulation. *IEEE Transactions On Visualization and Computer Graphics*, 5(1):62–73, Jan-Mar 1999.

[7] G. Debunne, M. Desbrun, A. Barr, and M.-P. Cani. Interactive multiresolution animation of deformable models. In *10th Eurographics Workshop on Computer Animation and Simulation, Milano*, 1999.

[8] G. Debunne, M. Desbrun, M.-P. Cani, and A. Barr. Adaptive simulation of soft bodies in real-time. In *Computer Animation 2000, Philadelphia, USA*, May 2000.

[9] H. Delingette, S. Cotin, and N. Ayache. A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation. *Computer Animation*, May 26-28 1999.

[10] M. Desbrun, P. Schröder, and A. Barr. Interactive animation of structured deformable objects. In I. Scott MacKenzie and James Stewart, editors, *Graphics Interface '99*, June 1999.

[11] Mathieu Desbrun, Mark Meyer, Peter Schrder, and Alan H Barr. Discrete differential-geometry operators in 2d and 3d. Submitted, 2000. http://www-grail.usc.edu/pubs/.

[12] Y.C. Fung. *Foundations of Solids Mechanics*. Prentice-Hall, 1965.

[13] F. Ganovelli, P. Cignoni, C. Montani, and R. Scopigno. A multiresolution model for soft objects supporting interactive cuts and lacerations. In *Proceedings of Eurographics*, 2000.

[14] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In Turner Whitted, editor, *SIGGRAPH'97 Conference Proceedings*, Annual Conference Series, pages 209–216. Addison Wesley, August 1997. ISBN 0-89791-896-7, http://www.cs.cmu.edu/g̃arland/multires/my-work.html.

[15] J.-D. Gascuel, M.-P. Cani, M. Desbrun, E. Leroi, and C. Mirgon. Simulating landslides for natural disaster prevention. In *Proceedings of the 8th Eurographics Workshop on Animation and Simulation*, 1998.

[16] A. Van Gelder. Approximate simulation of elastic membranes by triangulated spring meshes. *Jour. of Graphics Tools*, 3(2):21–42, 1998.

[17] S. F. Gibson and B. Mirtich. A survey of deformable modeling in computer graphics. Technical report, MERL, November 1997. TR-97-19, http://www.merl.com.

[18] J.-P. Gourret, N. Magnenat-Thalmann, and D. Thalmann. Simulation of object and human skin deformation in a grasping task. *Proceedings of SIGGRAPH'89 (Boston, MA)*, pages 21–30, July 1989.

[19] D. Hutchinson, M. Preston, and T. Hewitt. Adaptative refinement for mass/spring simulations. In *Proc. of the Eurographics Workshop on Computer Animation and Simulation*, pages 31–45, Poitiers, sep 1996.

[20] D. James and D. Pai. Art defo - accurate real time deformable objects. In *Proc. of SIGGRAPH*, Computer Graphics Proceedings, SIGGRAPH, pages 65–72. ACM SIGGRAPH, ACM Press, August 1999.

[21] J.-C. Lombardo, M.-P. Cani, and F. Neyret. Real-time collision detection for virtual surgery. In *Proc. of Computer Animation*, May 1999.

[22] J. O'Brien and J. Hodgins. Graphical models and animation of brittle fracture. In *Proceedings of SIGGRAPH'99*, pages 137–146, 1999.

[23] A. Pentland and J. Williams. Good vibrations: modal dynamics for graphics and animation. *Proc. SIGGRAPH*, 23(3):215–222, July 1989.

[24] B. Smith, P. Bjrstad, and W. Gropp. *Domain Decomposition : Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge Univ Pr, May 1996.

[25] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. *Proc. of SIGGRAPH*, 21:205–214, July 1987.

[26] A. Witkin and W. Welch. Fast animation and control for non-rigid structures. *Proceedings of SIGGRAPH*, 24(4):243–252, August 1990.

[27] Yan Zhuang and John Canny. Real-time and physically realistic simulation of global deformation. In *SIGGRAPH99 Sketches and Applications*, Los Angeles, California, August 1999.