Adaptive Simulation of Soft Bodies in Real-Time

Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, Alan H. Barr

Adaptive Simulation of Soft Bodies in Real-Time

Gilles Debunne[†] *Mathieu Desbrun*[‡] *Marie-Paule Cani*[†] *Alan Barr*[§]
[†] iMAGIS[*] - [‡] USC - [§] Caltech

Abstract

This paper presents an adaptive technique to animate deformable bodies in real-time. In contrast to most previous work, we introduce a multi-resolution model that locally refines or simplifies the simulated object over time in order to optimize the computational effort. We use the mixed Finite-Volume/Finite-Element method to derive fast, local discrete differential operators over irregular grids with tight error bounds. The linear elasticity equations can be simulated using an arbitrary non-nested hierarchy of volumetric meshes, allowing the computation load to be automatically concentrated where and when needed. Real-time simulation, with a guaranteed frame rate, can be achieved as demonstrated through a series of examples on our video.
Keywords: Animation, Deformable bodies, Multiresolution, Space-time adaptivity, Adaptive sampling.

1 Introduction

Deformable models for computer animation has long been a heavily researched topic. Different approaches, ranging from to computational physics methods to more ad-hoc techniques, have been proposed over the last fourteen years, resulting recently in real-time techniques for moderately complex objects. Research has been primarily focussed on reducing the computational complexity, mainly using very simple physical models and/or using fast, stable integration techniques. However, very little work has been done in the direction of adaptive models, where different levels of resolution are provided in order to reduce the overall complexity by only simulating relevant levels of detail. Many adaptive models already exist in Computer Graphics. For example, the radiosity method for example has gained tremendous efficiency and reliability with hierarchical algorithms that automatically subdivide or cluster surface patches to ensure a given accuracy at a lower cost. Nevertheless, similar approaches in animation are harder to develop, since we are dealing with a dynamic system.

1.1 Prior work

The first model in Computer Graphics to animate deformable bodies was introduced by Terzopoulos *et al.* [24], using finite differences for the integration of energy-based Lagrange equations. This initial model, based on Hooke's law for perfectly elastic objects, has been improved subsequently to handle plasticity and fractures [25, 23]. Finite element techniques have also been proposed [13, 20], including real-time simulations methods for linear elastic bodies [4, 16, 5, 7]. However, these methods use static models, thus loosing the dynamic behavior (once a pressure exerted upon a deformable objects is released, the model pops back directly to its original, undeformed shape). As these physically-based methods are computationally intensive, alternate approaches have been derived, allowing fast animation of simple dynamic objects by taking into account only some possible deformations or vibration modes [21, 26, 19]. Unfortunately, such restrictions on the behavior considerably affects the realism of the animation. Other approaches developed robust implicit integration schemes to allow for larger time steps, thus dramatically reducing the necessary computational time per second of animation [1, 11].

As mentioned above, all these techniques use a fixed space discretization rate. Most of them also use a fixed time discretization rate. However, a model using adaptive resolution has been developed for the simulation of hanging clothes [14]. The mass-spring network modeling the piece of cloth refines locally as soon as two adjacent springs form an angle exceeding a given threshold, to provide a more accurate shape description. This approach allows the model to converge towards the static equilibrium faster by limiting the number of masses used during the calculation. Unfortunately, such a simple model cannot guarantee an identical global behavior during the animation: incoherences will take place when a refinement occurs. Even if collisions with obstacles are handled correctly, the cloth weight changes as new masses appear. This prevents any adequate simulation if the cloth is pulled for instance.

Another model, introduced for highly deformable materials like dough or mud, proposes a space and time adaptive physics-based technique based on SPH[2] [9, 12, 8]. This time, a state equation which represents the object's behavior (such as stiffness) is defined by the user. Particles discretize the material, and they subdivide or merge according to a local energy criterion while deriving appropriate interaction forces to ensure the same global behavior defined by the state equation. However, simulating structured objects with this method, like soft bodies or human organs, is inappropriate. From a theoretical point of view, the SPH formalism is really only adequate for a large number of particles (for a given accuracy) since it is based on a Monte-Carlo integration. Boundary conditions are also not handled correctly, therefore limiting the applications to fluid-like materials.

Recently, we proposed a new approach which allowed multiresolution animation, based on a local discretization of the linear elasticity laws [6]. The main drawback of this method is the empirical computation of the discrete operators involved in the equations. Although they give good results when applied to nearly uniformely sampled points, trying to use several levels at the same time jeopardizes the accuracy of the result since no clear error bound exists for irregular grids. Therefore, the dynamic behavior of the material is not guaranteed to be similar when several sampling rates coexist.

1.2 Contributions and Overview

We propose two major contributions in this paper, leading to a complete, simple adaptive animation technique for deformable bodies. First, we derive new differential operators to ensure guaranteed error bounds. These operators will be proven equivalent to the finite element formalism, while keeping all the computations local for efficiency purposes. Second, we introduce a simple, general adaptive technique to handle adaptivity over a general, non-nested hierarchy of meshes describing an object. Since these meshes will be well-conditioned, the hierarchy will ensure an optimized accuracy. Dif-

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

[2] Smooth Particle Hydrodynamics

ferent levels of the mesh hierarchy will "communicate" through a simple "ghost node" technique, similar to that used in the domain decomposition method.
Section 2 will detail the dynamic model we use and the way we derive and compute accurate differential operators. Section 3 will explain how several volumetric meshes can be simulated simultaneously, and section 4 explains how we choose which level is appropriate in a given region, providing a complete adaptive simulation technique. We show results in Section 5, and give conclusions in Section 6.

2 Dynamic Model for Simulation

In this section, we present our basic dynamic model for a given, fixed discretization. It involves the choice of a physical model for the material, and a numerical technique to integrate the model over time.

2.1 Physical Model

To ensure an efficient simulation, we picked the conventional linear elasticity theory as it results in a simple, yet general physical model. The deformation of the object is measured by the *displacement field* $\mathbf{d}$ which is simply the difference vector between a point's current position and its rest position. We chose to express this model using the Lamé formulation as explained in our previous work [6] (bold characters represent vectors):

$$\rho\ \mathbf{a} = \lambda\Delta\mathbf{d} + (\lambda+\mu)\nabla(\nabla\cdot\mathbf{d}) \qquad (1)$$

This equation states that the acceleration $\mathbf{a}$ of a point, times its density ρ, is the weighted sum of two terms: $\Delta\mathbf{d}$ is the *laplacian* vector of the displacement field, and $\nabla(\nabla\cdot\mathbf{d})$ is the gradient of the divergence of the same field, often called gradient of the volume expansion in fluid mechanics[3]. λ and μ are the *Lamé* coefficients and determine the material's behavior.
We can interpret the linear elasticity model in another way: it is a deformation propagation (due to the $\Delta\mathbf{d}$ term), with more or less compressibility (due to the $\nabla(\nabla\cdot\mathbf{d})$ term) depending on the coefficients. This equation has the major advantage of encapsulating the Euclidean strain and stress tensors in a unique partial derivative equation. Deformation in a material is then expressed as a displacement field added to the original shape.
As this formulation only depends on second order derivatives, the resulting acceleration will be null if a rigid displacement (translation or rotation) is applied to the entire object. Animating the object simply amounts to integrating Equ. (1) over time to compute the evolution of this displacement field in the object. To do so, one needs to estimate accurately the various differential quantities involved.

2.2 Accurate Discrete Operators

Although discrete operators for both the Laplacian and the volume expansion were derived in [6], they were no defined error bounds due to the empirical foundations they relied on. In this section, we propose a new, sound derivation for these two discrete operators which will guarantee an improved accuracy as tests will confirm.

2.2.1 Mixed Finite-Volume/Finite-Element Method

Historically, fluid dynamics and continuum mechanics were built upon local conservation laws [2]. The idea was that the partial differential equation (PDE) defining the behavior is valid *everywhere* within the material, so numerical methods must enforce a local respect of the PDE. Methods such as Finite Differences (FD) or Finite Volumes (FV) basically implement this approach. With the development of the calculus of variations, variational techniques like the Finite Element technique has been recently overwhelmingly preferred, since these methods were versatile enough to handle cases where FD were unsuited (arbitrary boundaries for instance).
In this paper, we propose an alternative to these different techniques. We use the mixed FV/FE formulation [18] to derive good local estimates of differential quantities with guaranteed error bounds. We will derive both the Laplacian and the Gradient of Divergence hereafter, first in 2D for simplicity, but the extension to 3D will be straightforward.
The key behind the mixed FV/FE method is Gauss's theorem, which turns a local integral over a region of a derivative into a line integral over the boundary of this region:

$$\int_{\mathcal{V}} \frac{\partial}{\partial i}\mathbf{X}\ dV = \int_{\partial\mathcal{V}} \mathbf{X}\cdot\mathbf{n_i}\ dl \quad ,\ \forall i \qquad (2)$$

$\frac{\partial}{\partial i}\mathbf{X}$ is the derivative with respect to coordinate i of the field $\mathbf{X}$, and $\mathbf{n_i}$ is the i^{th} component of the normal to the region. Each sample point (or particle) is assigned a specific region, such as these regions tile the whole object. We chose to define a point's specific volume as its *Voronoi region*, since the interior of this region is closer to this particle than to any other one[4]. We therefore use Gauss's theorem to ensure that the PDE is satisfied by all these tiling regions. We use a piecewise bilinear (resp. trilinear) interpolation of the field (FE) within each triangle (resp. tetrahedron) of the object's mesh to finally derive our operators, as detailed below.

2.2.2 Laplacian on a triangulated domain

The Laplacian of the displacement is $\Delta\mathbf{d} = \mathbf{div}\ \mathbf{grad}\ \mathbf{d} = \nabla\cdot(\nabla\mathbf{d})$. Applying, Gauss's formula (Equ 2), for each of the component d of the vector $\mathbf{d}$ ($\mathbf{X}$ being replaced by $\nabla\ d$), gives:

$$\int_{\mathcal{V}} \Delta d \ = \ \int_{\mathcal{V}} \nabla\cdot(\nabla\ d) \ = \ \int_{\partial\mathcal{V}} (\nabla\ d)\cdot\mathbf{n}\ \ dl$$

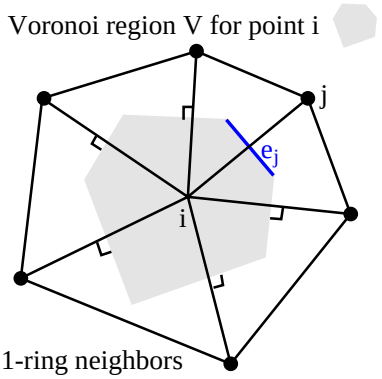


Figure 1: *Point i samples its Voronoi region. The field is piecewise linear over each triangle.*

Since we assume that the displacement field is linear over each triangle, its gradient is constant on a triangle. Let (i,j,k) be one of the triangles of the Voronoi region around node i (Fig 2(a)). On this triangle, the gradient can be expressed as $\nabla d = \frac{1}{2\mathcal{A}_{ijk}}[(d_j -$

[3]The nabla vector ∇ coordinates are defined as $(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})$. It is a convenient way of writing first derivatives as $\mathbf{grad}\ \mathbf{X} = \nabla\mathbf{X}$ and $div\ \mathbf{X} = \nabla\cdot\mathbf{X}$.

[4]This choice can be related to Natural Element Method [3]. We will explore the similarities in another paper

$d_i)\vec{\mathbf{ki}}^{\perp} + (d_k - d_i)\vec{\mathbf{ij}}^{\perp}]$ where $\mathcal{A}_{ijk}$ is the area of the triangle and $\mathbf{X}^{\perp}$ is a vector orthogonal to $\mathbf{X}$ of the same length.
The integration on the boundary of $\mathcal{V}$ can be rewritten as

$$\begin{aligned}\int_{\partial\mathcal{V}} (\nabla\, d)\cdot\mathbf{n}\; dl &= \sum_{edges\ \mathbf{j'k'}} (\nabla\, d)\cdot\mathbf{j'}\vec{\mathbf{k}'}^{\perp} \\ &= \sum_{edges\ \mathbf{jk}} (\nabla\, d)\cdot\frac{1}{2}\mathbf{j}\vec{\mathbf{k}}^{\perp} \\ &= \sum_{edges\ \mathbf{jk}} \frac{1}{4\mathcal{A}_{ijk}}(d_j - d_i)\vec{\mathbf{ki}}\cdot\mathbf{j}\vec{\mathbf{k}} + (d_k - d_i)\vec{\mathbf{ij}}\cdot\mathbf{j}\vec{\mathbf{k}}\end{aligned}$$

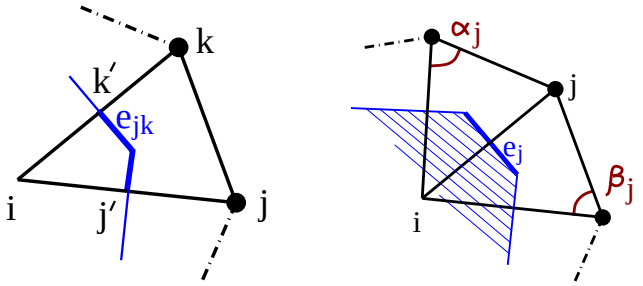


Figure 2: *(a) The Laplacian operator uses an integral over the boundary of its Voronoi region, and involves (b) the cotg of the edge opposite angles*

The dot products and the area can be simplified, making the cotangent of the angle appear. Reorganizing terms by edge contribution, we obtain : (see Fig 2(b))

$$\int_{\partial\mathcal{V}} (\nabla\, d)\cdot\mathbf{n}\; dl \;=\; \frac{1}{2}\sum_{edges\ \mathbf{jk}} (cot\ \alpha_j + cot\ \beta_j)\,(d_j - d_i)$$

The Laplacian is assumed smooth enough to be sampled by point i over the whole Voronoi region[5], thus allowing us to separate the volume integral. The previous equation being valid for each of the component d of vector $\mathbf{d}$. We can write for the vector Laplacian:

$$\boxed{\Delta\mathbf{d} \;=\; \frac{1}{2 * Volume(\mathcal{V})}\sum_{edges\ \mathbf{jk}} (cot\ \alpha_j + cot\ \beta_j)\,(\mathbf{d}_j - \mathbf{d}_i)} \quad (3)$$

For an edge e_j, the weighting coefficient $(cot\ \alpha_j + cot\ \beta_j)$ only depends on the mesh geometry and must be computed using the mesh rest's position (in the finite element small deformation theory). It can be precomputed and stored for an efficient computation of the Laplacian.
This formulation satisfies the action-reaction law: the force exerted by i on j is the opposite of the one exerted by j on i. The volume term indeed vanishes when accelerations are converted into forces and multiplied by the mass over the rest density. No assumption on a constant or uniform density is made as it cancels out from equation 1 when we compute the force.
Some geometrical considerations can prove that this is a first order approximation of the Laplacian. However, if angles α_j and β_j are equal, a second order computation can be achieved. This condition, enforced on each edge, results in 2D in a 'perfect' mesh made of equilateral triangles. In 3D, however such a regular mesh is not possible, but this constraint can govern the mesh optimization preprocess in order to have well conditioned simulation.

[5]Although we have a linear interpolation over the triangle, which could lead to a null second order derivative, the fact that we consider the field as being piecewise linear *over each triangle* will create a non null Laplacian.

2.2.3 Gradient of Divergence operator

We compute the $\nabla(\nabla\cdot\mathbf{d})$ term using the same methodology. Let's first estimate the divergence of $\mathbf{d}$ on one mesh triangle. It is a constant as $\mathbf{d}$ is linear over the triangle. Using the FE basis function W_i[6], we can write for an interior point x, $\mathbf{d}(x) = \sum_{\alpha\ \in\ i,j,k} \mathbf{d}_\alpha\ W_\alpha(x)$ and the gradient of $\mathbf{d}$ satisfies: $\nabla\mathbf{d} = \sum_{\alpha\ \in\ i,j,k} \mathbf{d}_\alpha\ \nabla W_\alpha$ ($\mathbf{d}_i$ is the value at point i).
The ∇W_i vector is collinear to the height h_i of the triangle which passes through point i and satisfies (see figure below)
$\nabla W_i = \frac{1}{h_i}\ \frac{\vec{\mathbf{jk}}^{\perp}}{||\vec{\mathbf{jk}}||}$

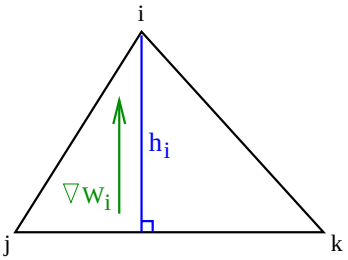


Figure 3: *We use the triangle basis functions W_i to compute the divergence.*

Let's consider a triangle (i,j,k) (Fig 3). Using $\sum_{1..3} W_i(x) = 1$ ($\sum_{1..3}\nabla W_i = 0$), and the area $\mathcal{A}_{ijk}$ of the triangle ($2\mathcal{A}_{ijk} = h_i\ \|\vec{jk}\| = h_j\ \|\vec{ki}\| = h_k\ \|\vec{ij}\|$) , we can write

$$\nabla\mathbf{d} = \frac{1}{2\mathcal{A}_{ijk}}(\mathbf{d}_j - \mathbf{d}_i)\vec{\mathbf{ki}}^{\perp} \;+\; (\mathbf{d}_k - \mathbf{d}_i)\vec{\mathbf{ij}}^{\perp}$$

The divergence of a field is the sum of its derivatives with respect to all the coordinates:

$$(\nabla\cdot\mathbf{d})_{ijk} = \frac{1}{2\mathcal{A}_{ijk}}\left\|\vec{\mathbf{ij}}\times(\mathbf{d}_k - \mathbf{d}_i) \;+\; \vec{\mathbf{ki}}\times(\mathbf{d}_j - \mathbf{d}_i)\right\|$$

where $\times$ is the cross product operator. We now apply once again the Gauss's theorem (Equ 2), coordinate by coordinate, to compute the gradient of the divergence. Using the same notations, we have:

$$\boxed{\nabla(\nabla\cdot\mathbf{d})_i \;=\; \frac{1}{Volume(\mathcal{V})}\sum_{edges\ jk} (\nabla\cdot\mathbf{d})_{ijk}\,\vec{\mathbf{jk}}^{\perp}} \quad (4)$$

Once again, this term is a weighted sum of the displacement difference between a point and its neighbors. The weights are purely geometric and can be computed before simulation starts.

2.3 Extension to 3D

In this section, we simply directly give the results of a straightforward generalization to 3D of the previously described operators. In 3D, we deal with tetrahedron. Just like before, the heights of each element will determine the operators' coefficients.

[6]W_i is a linear function, which is null at two of the three points of the triangle, and is 1 at the i^{th} point.

The formulation is slightly different: instead of computing the operators for each edge, we give the contribution of an entire element to one of its point, say i. This contribution is once again a geometrically weighted sum of the displacements of all the elements' points. Note that point i itself is to be included in the following sums which range over the 4 points of the elements.
Let's call $\alpha_{\mathbf{i}}$ the vector which is orthogonal to the face opposed to point i, and which length is equal to $\frac{1}{h_i}$ (h_i is still the height of the element at point i).
The Laplacian operator, is then simply written as

$$\boxed{\Delta \mathbf{d}_i \;=\; - \sum_{node\ j \in\ 1..4} (\alpha_{\mathbf{i}} \cdot \alpha_{\mathbf{j}})\ \mathbf{d}_j} \tag{5}$$

The $\nabla(\nabla \cdot \mathbf{d})$ operator is represented as a 3 by 3 matrix as follows

$$\boxed{\nabla(\nabla \cdot \mathbf{d})_i \;=\; - \sum_{node\ j \in\ 1..4} ({\alpha_{\mathbf{i}}}^T \cdot \alpha_{\mathbf{j}})\ \mathbf{d}_j} \tag{6}$$

Although they are not written in the same manner (point i's displacement is separated from the neighbors' one), these equations give exactly the previous ones when they are computed in 2D.

2.4 Comparison with Finite Elements

The finite element formalism usually computes its coefficients on a *per element* basis. For each element (triangles or tetrahedra), an *elementary stiffness matrix* is computed, its coefficients being based on the heights of the element. These matrices are then regrouped in a global stiffness matrix $\mathcal{K}$ which links internal forces and displacements through $\hat{\mathbf{f}} = \mathcal{K}\ \hat{\mathbf{d}}$, where $\hat{\mathbf{f}}$ and $\hat{\mathbf{d}}$ are vectors containing all the points' respective displacements and applied forces.
Depending on boundary conditions, (free displacement (resp. force) and imposed force (resp. displacement) on each point), the system unknowns change, and the matrix has to be inverted (or intelligently updated as in [16]) each time boundary conditions change.
Instead of inverting the *global* system matrix, we only compute *local* forces on each point using its neighbors' information (namely displacement) and equations 5 and 6. On the one hand, we loose the benefits of a global resolution of the system, which guarantees a coherent state after each time step. But on the other hand, the computational time is tremendously reduced at each time step and moreover, this allows us to implement the adaptive approach described in the next section. Other approaches [20, 7] avoided the costly matrix inversion. However, none of them achieved a true real-time simulation.

If one develops the elementary stiffness matrix of an element and isolates the influences of point i over point j in that element, they will find back equations 3 and 4 (separating the two terms using the λ and μ coefficients). Each of the two operators are represented as a 3 by 3 matrix for each edge.
Although the $\nabla(\nabla \cdot \mathbf{d})$ gives the same result when computed using FE or with the method described in previous section, the $\Delta \mathbf{d}$ is not the same. Finite Elements represent the Laplacian as a 3 by 3 antisymmetric matrix. The diagonal terms are all equal and their value is exactly to cotg based one computed before (Equ 3). What about the non-diagonal terms ? We found out that those terms were actually, once divided by the volume, simply plus and minus ones. When a point computes the Laplacian of the displacement using the finite element method, it adds (or subtracts) some of the vector coordinates of its neighbors' displacement. These computations actually cancel out.
An edge being shared by two elements, we've found that the plus and minus ones will always cancel out when the contributions of the two elements are summed. Classic finite elements techniques hence introduce extra computations (the single diagonal value is replaced by 9 non null coefficients) and instabilities in the simulation as the terms will never exactly cancel out when they are processed by a computer.
For boundary elements however, these extra terms don't vanish as no opposite element is present. Nevertheless, the ghost particle principle, which consists in adding a virtual symmetric point on the other side of the boundary[7], states that the orthogonal component of the gradient of the displacement field must be null on the boundary. These extra terms can hence be simply skipped.

2.5 Adding Rayleigh Damping

Adding damping to our simulation is necessary in order to increase its realism since objects never oscillate indefinitively in reality. Our damping force also comes from the finite element theory. Just like in [20], we add to our strain tensor the contribution of the *strain rate tensor*, which measures the rate at which the strain is changing inside the material. The equations are similar to the one described before (1), except that with now use the first derivative with respect to time of the displacement (namely the point's speed). This is classically known as Rayleigh damping which introduces a first time derivative term $\dot{x}$ in the $\mathcal{K}x = f = m.a = m\ddot{x}$ basic equation. The added acceleration is given by

$$\rho\ \mathbf{a}_{damp} = \phi \Delta \mathbf{v} + (\phi + \psi) \nabla(\nabla \cdot \mathbf{v}) \tag{7}$$

where $\mathbf{v}$ represents the velocity vector. ϕ and ψ will control the internal kinetic energy dissipation. Just like in the previous equation, rigid motions will not be damped by this equation which will only reduce internal vibrations. The coefficients that were computed before in (5) and (6) are simply applied to the speed field to compute this damping force.

2.6 Simulation of Fixed Mesh

Now we have both the physical model and the way to accurately estimate differential operators, simulating an object given by a fixed tetrahedralization is simple. At each time step,

1. Compute $\Delta \mathbf{d}$, $\Delta \mathbf{v}$, $\nabla(\nabla \cdot \mathbf{d})$ and $\nabla(\nabla \cdot \mathbf{v})$ for each mesh node using the operators described in Section 2.2.

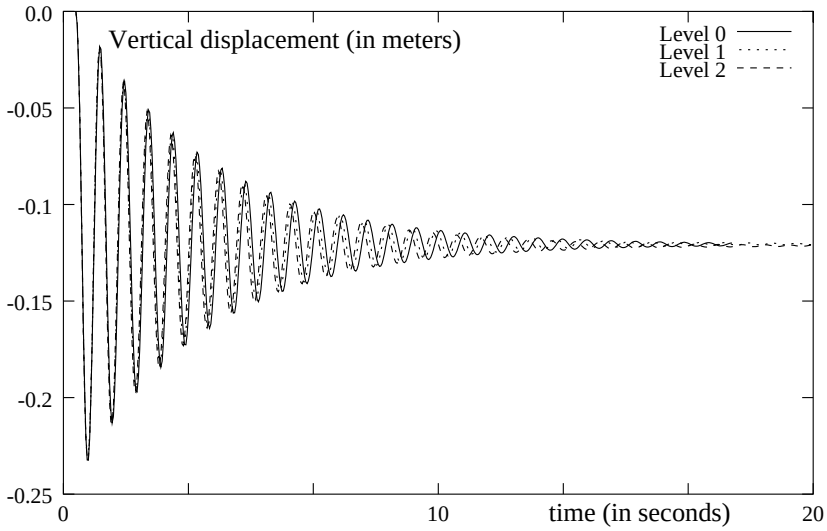
2. From Equ. (1) and Equ. (7), deduce the acceleration at each node.

3. Integrate the acceleration, and update position and velocity accordingly (explicit Euler integration).

4. Go to the next time step.

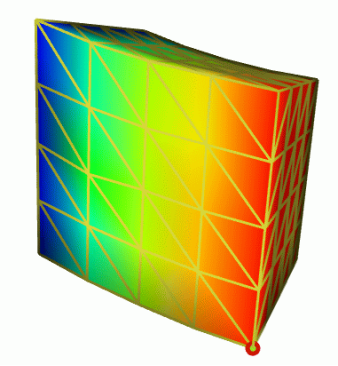
Figure 2.6a shows the result of the animation of a cube under the action of gravity. One of its face is fixed and we measure the vertical displacement of one of its corner (circled in Fig 2.6b). The simulation was done at three different spatial resolutions (27, 57 and 135 points). Note that the physical coefficients were the same in all the simulations, resulting in similar results.
This algorithm is as simple as in the case of a mass-spring system. The overall complexity is comparable, but the finite element formalism used here provides some guarantees on the result. Its independency from the mesh resolution will especially allow us to combine several mesh resolutions during the same simulation as detailed in the next section.

[7] this principle is widely used in the SPH theory to model free boundaries



(a)

Vertical displacement of the corner of a 1 meter cube under the action of gravity with damping. Three different resolutions are plotted. Physical coefficients are $\lambda = 40,000$ and $\mu = 100,000$.



(b)

The cube in its final rest position. Plotted point is circled.

3 Levels of Detail

Most simulation methods compute deformations at a pre-defined resolution, using a given mesh. As stated earlier, this can be very inefficient: a high resolution is needed in highly deformed areas, while a coarser level of detail would be sufficient in little deformed areas. The aim of this paper is to allow a real-time adaptation of the level of detail being used to compute deformations. This adaptation is local in space and in time, in order to concentrate the computational load only where and when needed.
This section explains how we adapt the simulation method introduced in section 2 in order to enable the joint use of different levels of details for representing different regions of the deformable body.

3.1 LOD representation with non-nested meshes

Most previous works on adaptive resolution for the animation of deformable bodies[14, 6, 20] has used recursive subdivision of an initial mesh for providing the different levels of detail needed. The advantage of this formulation is that subdivided and not-subdivided parts of the mesh can interact through common nodes and edges.
Most methods rely on tetrahedrical meshes for sampling the deformable body, since they provide a good sampling of arbitrary shapes (however this was not the case with our previous method [6] with was based on a hierarchy on cubic octrees). With such meshes, using recursive subdivision for defining levels of detail is not a good idea: whatever the subdivision method, the quality of the initial mesh in terms of angles and/or aspect ratio will be lost after several subdivisions. Relaxation steps or complete re-meshing of the object are possible, but incompatible with a real-time application. 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 they represent the same object, can be completely independent (no vertex needs to be shared), which leaves complete freedom for their generation. In the remainder of this paper, the terms ”parent mesh” and ”child mesh” are used for the meshes that represents the deformable body at the immediately coarser (respectively finer) scale with respect to a given mesh.

3.2 Interface between LOD

Suppose that different regions of the deformable body are sampled using different LOD during a simulation. Meshes representing these LOD should be able to “cooperate” at the interface between the regions. To achieve this, we use a simplified version of the domain decomposition method [8]: the different levels of detail slightly overlap at their interface, and some points, that we call *ghost nodes*, will transmit the information between the meshes.
At a given time of the simulation, different meshes, representing different LOD, will be “active” in different regions of the deformable body. Let P be a node near the interface between two LOD (Figure 4).

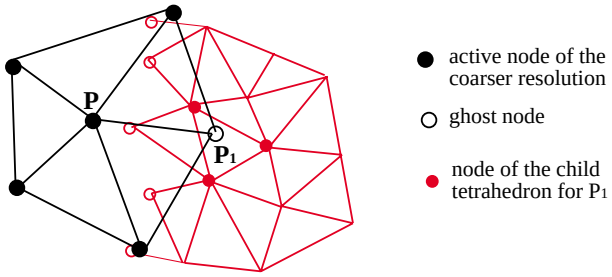


Figure 4: *Ghost nodes are used for enabling the joint use of different LODs during simulations (2D view, coarse mesh is black, finer mesh is grey).*

[8] See `http://www.ddm.org/` for pointers on this method.

To perform computations at node P (Equations (3) and (4)), displacement field values stored at the neighboring nodes such as P_1 are needed. However, no simulation is performed at P_1 since the latter belongs to a region which is simulated at another LOD. P_1 is then a "ghost node" (as opposed to "active"), and we approximate the displacement field value at P_1 by interpolating values computed at the current resolution in this region (Figure 4). "Active" nodes do actually compute a force and integrate it over time, where as "ghost" one simply interpolate field values.

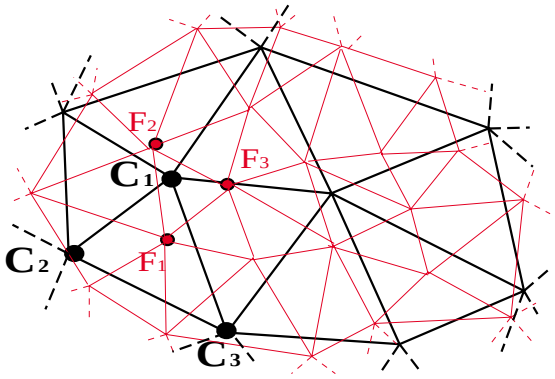


Figure 5: *Linking the LOD meshes (2D drawing) : node C_1 will pull information from finer level using its child element (F_1, F_2, F_3). Information can also be pushed to node F_1 from its parent element (C_1, C_2, C_3), depending on which level is actually simulated.*

We use a simple linear interpolation to compute the displacement value of a ghost node. This fast solution is consistent with the fact that we use linear finite elements. In practice, each node stores its barycentric coordinates with respect to its 'parent' and 'child' tetrahedron (The parent (resp. child) tetrahedron of a node N is the tetrahedron of the parent (resp. child) mesh which spatially contains N. A node at the surface of the object may not be *inside* a coarser level tetrahedron, but in that case we choose the closest tetrahedron — one of the barycentric coordinates thus being negative). In a preprocess, each node finds its parent (resp. child) tetrahedron in the coarser (resp. finer) mesh, and stores its barycentric coordinates. When a ghost node needs to compute its displacement value, it uses these barycentric coordinates to interpolate from the coarser or finer mesh.
A ghost node may need to interpolate from nodes which are ghost too. However, this is not a specific case, and the node will simply use the displacement value stored in the ghost node. Our algorithm guarantees that the value stored in this node was recursively pulled from the finest simulated level (see section 4.3).
The method we have just described could easily be applied to the simulation of a body represented at a predefined, non-uniform spatial resolution. In this paper, we dynamically adapt, during the simulation, the local space and time sampling to the needs of the simulation. This results in stable and reliable computations in areas experimenting high deformation, and increased efficiency in stable areas.

4 Adaptive Simulation

4.1 Building a hierarchy of levels of detail

Mesh resolution and time steps are closely related: to offer stable computations with explicit integration, time steps should satisfy the Courant criteria, which involves the size of tetrahedra within the mesh and the speed at which deformations propagate, computed from the lamé constant of the simulated material (see [6]):

$$dt < h \sqrt{\frac{\rho_0}{\lambda + 2\ \mu}} \tag{8}$$

h is the minimum distance between two adjacent nodes, ρ_0 is the material's density, the square root term representing the speed of sound in the material. In practice, the different time steps associated with the meshes representing the different LOD are chosen as inverse powers of two subdivisions of the display frame rate: $dt_{mesh} = dt_{framerate}/2^n$. This means that the average edge length should be divided by two between two consecutive meshes. This condition is not necessary, but it allows an optimized simulation and leads to intuitive LOD representation.
Our meshes were generated using the commercial software GHS3D [22], which generates good quality 3D meshes from a triangulation of a closed surface. The meshes could be optimized using a relaxation technique : according to the criteria described in section 2, length of edges and dihedral angles between two adjacent tetrahedra could be set to be as equal as possible. The relaxation process is only applied to the internal nodes of each mesh, thus preserving the surface appearance.
The precision (and stability) of a finite element simulation is directly related to the aspect ratio of the elements it uses. As our method never changes the shape of the elements (operators are computed on the undeformed mesh, and this mesh is never changed during the simulation), the preprocessing step can create appropriate meshes for an efficient and accurate simulation.

4.2 On-the-fly adaptivity

Our aim is to allow local adaptations of the LOD during simulation. The nodes of a mesh represent a given region of space (their associated Voronoi region), and their values being the average of the local material's properties within this region. When sampling becomes too coarse in a region, we switch to a more detailed level of detail, adding sample points in that region to enhance the local description of the material. We do this on a per-node basis, a coarse node being replaced by its "children" from the child mesh (the children are the nodes from the finer mesh which lie inside the Voronoi region of their parent). As Voronoi regions form a partition of space, all the nodes of the finer level will have a parent, and two parents won't share a child, thus leading to a hierarchical tree structure (see Figure 6).

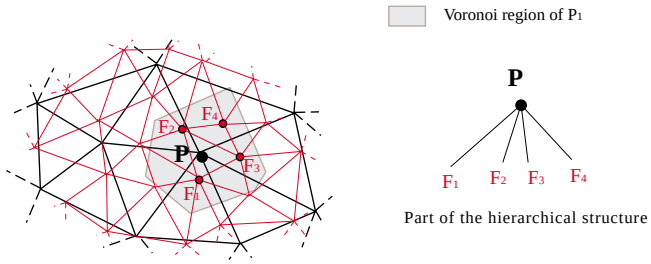


Figure 6: *A hierarchical structure is built by connecting each node of a given LOD to the nodes of the next LOD that are included into its Voronoi region. These children will replace P_1 if sampling becomes too coarse.*

As stated in Section 2, the stress-strain tensors assume local linear deformations. We thus need a refined sampling in regions where the

current linear approximation of the displacement field is not sufficient. To ensure coherence with the rest of our approach, we use a criterion based on the spatial second derivatives of the displacement field:

$$\delta_i = \hat{h}_i^2 ||(\Delta \mathbf{d})_i|| \quad (9)$$

where $\hat{h}_i$ is the average distance between the sample point i and its 1-ring neighbors. This discontinuity δ_i therefore measures how far away from a linear field we locally are. Two thresholds control splitting (when discontinuity is too high) and merging (if all the children are continuous enough) of nodes, ensuring an adequate linear approximation everywhere. The use of two different values for splitting and merging threshold ensures that a region won't come back and forth between the divided and simplified states continuously.

4.3 Guaranteeing real-time

Let us first recall the animation algorithm: at each simulation step, for each LOD that needs to be simulated[9]:

1. Compute displacement field for ghost points of that level that are located in regions simulated at a finer scale.
2. Simulate active nodes
3. Compute displacement fields at ghost nodes of the finer level, using the new displacement values.
4. Split or merge points if needed.

With this algorithm, the transmission from finer to coarser level (step 1 of the algorithm) is not performed each time a finer node moves. As the parent is simulated only every other step, this transmission only has to be done before the parent level is simulated.
The local computation (no global matrix inversion) used in this paper provides fast force computation, and a linear computing time with respect to the number of active nodes. In order to achieve a given frame rate, hence ensuring a realistic visual result in real-time, we simply have to make sure that the number of active nodes (multiplied by the number of times they are updated per second) doesn't exceed a machine-dependent threshold.
The display frame rate (20-50Hz) is usually lower than the simulation time step (10^{-2} to 10^{-5} seconds = 100-10000Hz) and many simulation steps are done between each display. The time needed by the CPU to compute those steps has to be smaller than the display frame rate to ensure a true real-time computation. When these computations are done, the algorithm *waits* for the synchronization with the display before it starts a new time step. Computations for the next step cannot be started before since we are waiting for a new tool position, defined by the user, and that must be read at constant intervals.
In practice, before each display, the program measures the computation time per frame that was really needed by the simulation. When it exceeds 95% of the period between to frame displays, the splitting of nodes is forbidden, only merging can occur thus limiting the computational load, and a warning is sent to the user. This simple method results in an almost constant frame rate.

5 Results

In the examples below (see also the joined video tape), collisions between the deformable body and a user-controlled tool are detected using the method based on graphics hardware described in [17]. Collision detection is only performed once per frame, just before display. To avoid inter-penetrations, surfaces points are pulled out of the tool. The same translation is applied to the closest active node. Then, simulation is applied and results in a larger scale deformation of the object and into a feedback force that can be applied to the tool or displayed.

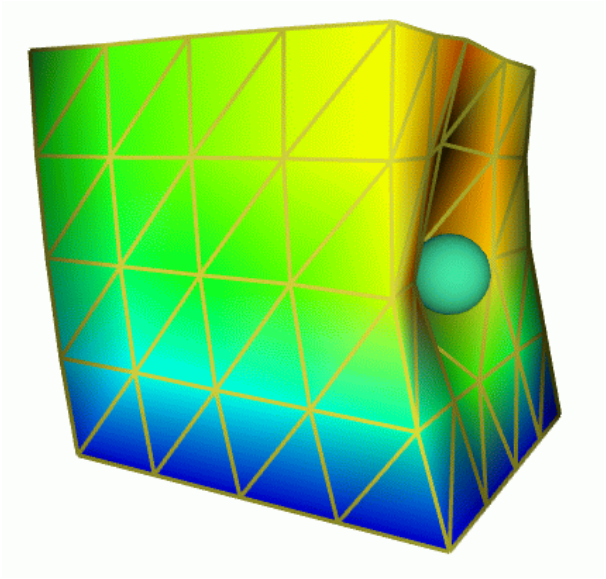


Figure 7: *The cube's down face is fixed. With $\mu = 10^6$, the volume preservation creates an intuitive deformation when the sphere is pulled inside the cube.*

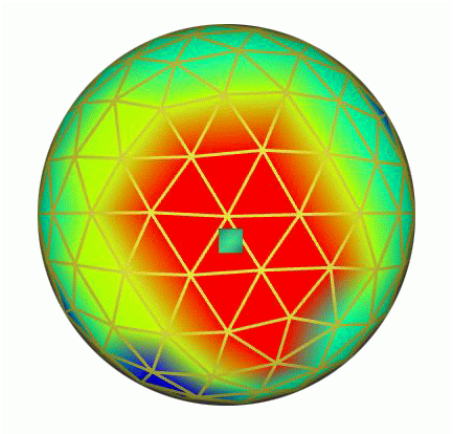


Figure 8: *Color represents the object's inner discretization level. The cubic tool (near the center of the sphere) created an intuitive sampling of the material when it was pulled against the surface*

6 Conclusion

This paper has presented a new method for computing dynamic simulations of elastic bodies in real-time. We have introduced a new multiresolution technique, using several (non-nested) 3D meshes representing different levels of detail to describe the object to animate. The method has strong connections to the adaptive finite element method, although we don't solve for a global system matrix: the force computation of each node is local and only involves

[9] As mentioned in Section 4.1, the levels of detail all have different time steps that are powers of two subdivisions of the frame rate.

the mesh neighbors of the node. On-the-fly local switches between levels of detail are therefore easier, and it allows us to concentrate the available computational time only where needed. Compared to previous approaches, this paper offers two main contributions: we developed robust, accurate discrete operators, and we proposed an adaptive simulation relying on a general non-nested hierarchy of meshes.

Future research includes interfacing the simulator with a force feedback device. We are also adding an underlying rigid object behavior to our soft objects. User's interactions will result in surface (small) deformations as well as in global rigid translations and rotations (this is a layered model in the sense of [25]). Parallelization is also easier thanks to our local computation scheme and should broaden the scope of our applications. We are also looking at implicit integration techniques to guarantee larger time steps. Finally, non-linear elasticity can be thought of, as the differential operators could be slightly modified to introduce non-linearity [10].

As a last remark, we stress the point that applying topological changes to the deformable body during the simulation would be very useful in applications such as surgery simulators[10]. An advantage of our approach among others such as [16] is that modeling cuts by locally suppressing some connections between nodes should be relatively easy. We plan to do this as our next future work.

Acknowledgments: The authors would like to express their most virulent thanks to Mark "Sooo Cool" Meyer for proof-reading and continuous insights.
The second author has been funded in part by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, Cooperative Agreement No. EEC-9529152.

References

[1] David Baraff and Andrew Witkin. Large steps in cloth simulation. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 43–54. ACM SIGGRAPH, Addison Wesley, July 1998.

[2] G. K. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge University Press, 1973.

[3] Jean Braun and Malcolm Sambridge. A numerical method for solving partial differential equations on highly irregular evolving grids. *Nature*, 376:655–660, August 1995.

[4] Stéphane Cotin and Morten Bro-Nielsen. Real-time deformable models for surgery simulation using finite-elements and condensation. *Eurographics proceedings*, 1996:21–30, 1996.

[5] Stéphane Cotin, Hervé Delingette, and Nicolas Ayache. Real-time elastic deformations of soft tissues for surgery simulation. *IEEE Transactions On Visualization and Computer Graphics*, 5(1):62–73, January-March 1999.

[6] Gilles Debunne, Mathieu Desbrun, Alan Barr, and Marie-Paule Cani. Interactive multiresolution animation of deformable models. In *10th Eurographics Workshop on Computer Animation and Simulation (CAS'99)*. Available at:`http://www-imagis.imag.fr/Publications/debunne/Multi`, September 1999.

[7] Hervé Delingette, Stéphane Cotin, and Nicolas Ayache. A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation. In *Computer Animation*, Geneva Switzerland, May 26-28 1999.

[8] Mathieu Desbrun and Marie-Paule Cani. Space-time adaptive simulation of highly deformable substances. *INRIA Technical Report*, RR-3829 (`http://www.inria.fr/RRRT/RR-3829.html`), December 1999.

[9] Mathieu Desbrun and Marie-Paule Cani-Gascuel. Smoothed particles: A new approach for animating highly deformable bodies. In Springer Computer Science, editor, *7th Eurographics Workshop on Animation and Simulation*, pages 61–76, Poitiers, France, September 1996.

[10] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH 99 Conference Proceedings*, 1999. Los Angeles, CA.

[11] Mathieu Desbrun, Peter Schröder, and Alan Barr. Interactive animation of structured deformable objects. In *Graphics Interface'99 proceedings*, 1999.

[10] see [15] for instance

[12] Jean-Dominique Gascuel, Marie-Paule Cani, Mathieu Desbrun, Eric Leroi, and Carola Mirgon. Simulating landslides for natural disaster prevention. In *8th Eurographics Workshop on Animation and Simulation'98*, 1998.

[13] Jean-Paul Gourret, Nadia Magnenat Thalmann, and Daniel Thalmann. Simulation of object and human skin deformations in a grasping task. *Computer Graphics*, 23(3):21–29, July 1989. Proceedings of SIGGRAPH'89 (Boston, MA, July 1989).

[14] Dave Hutchinson, Martin Preston, and Terry Hewitt. Adaptive refinement for mass/spring simulation. In *7th Eurographics Workshop on Animation and Simulation*, pages 31–45, Poitiers, France, September 1996.

[15] INRIA. Aisim. http://www-sop.inria.fr/epidaure/AISIM/.

[16] Doug James and Dinesh Pai. Art defo - accurate real time deformable objects. In *Proceedings of SIGGRAPH '99 (Los Angeles, California, August 8–13, 1999)*, Computer Graphics Proceedings, Annual Conference Series, pages 65–72. ACM SIGGRAPH, ACM Press, August 1999.

[17] Jean-Christophe Lombardo, Marie-Paule Cani, and Fabrice Neyret. Real-time collision detection for virtual surgery. In *Computer Animation'99*, May 1999.

[18] S. F. McCormick. *Multilevel Adaptive Methods for Partial Differential Equations. Chapter 2: The Finite Volume Method*. Vol. 6, SIAM, Philadelphia, 1989.

[19] Dimitri Metaxas and Demetri Terzopoulos. Dynamic deformation of solid primitives with constraints. *Computer Graphics*, 26(2):309–312, July 1992.

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

[21] Alex Pentland and John Williams. Good vibrations: Modal dynamics for graphics and animation. *Computer Graphics*, 23(3):215–222, July 1989. Proceedings of SIGGRAPH'89 (Boston, MA, July 1989).

[22] SIMULOG. Ghs3d. http://www.simulog.fr/itetmeshf.htm.

[23] D. Terzopoulos and K. Fleischer. Deformable models. *The Visual Computer*, 1988(4):306–331, 1988.

[24] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. *Computer Graphics*, 21(4):205–214, July 1987. Proceedings of SIGGRAPH'87 (Anaheim, California).

[25] Demetri Terzopoulos and Andrew Witkin. Physically based model with rigid and deformable components. *IEEE Computer Graphics and Applications*, pages 41–51, December 1988.

[26] Andrew Witkin and William Welch. Fast animation and control for non-rigid structures. *Computer Graphics*, 24(4):243–252, August 1990. Proceedings of SIGGRAPH'90 (Dallas, Texas, August 1990).