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. Our method relies on mixed Finite-Volume/Finite-Element method applied to an arbitrary non-nested hierarchy of volumetric meshes. We achieve a garanteed frame rate thanks to a novative multi-resolution algorithm that locally refines or simplifies the simulated object in order to concentrate computation load where and when needed.

1. Introduction

Many deformable models have been proposed for computer animation over the last fourteen years. Research has focussed on reducing the computational complexity by using simplified physical models and/or stable integration techniques, resulting recently in real-time models for moderately complex objects. 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 Computer Graphics model to animate deformable bodies relied on finite differences for integrating energy-based Lagrange equations [21]. It was improved using an hybrid model with rigid and deformable components to be more efficient and to handle plasticity and fractures [22, 20]. Finite element methods have also been used [11, 17], yielding real-time simulations for linear elastic bodies in the static case [3, 14, 4, 6]. Unfortunately, using static models can be too limitative: once a pressure exerted upon a deformable objects is released, the model pops back directly to its original, undeformed shape. Alternate approaches have achieved fast animation of simple dynamic objects by taking into account only some possible deformations or vibration modes [18, 23, 16]. However, such restrictions on the behavior considerably affects the realism of the animation. Lastly, some approaches developed robust implicit integration schemes to allow for larger time steps, thus dramatically reducing the necessary computational time per second of animation [1, 10].

As mentioned above, all these techniques used a fixed space discretization rate. A first model based on adaptive space resolution has been developed for the simulation of hanging clothes [12]. The mass-spring network modeling the piece of cloth refines locally when the angle between two adjacent springs exceeded a threshold. This model provides a more accurate shape description, but does not guarantee a constant dynamic behavior. The cloth weight changes over time, preventing any adequate simulation if the cloth was 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 (Smooth Particle Hydrodynamics) [8, 7]. This time, a state equation which represents the object's macroscopic behavior is specified. Particles discretizing the material subdivide or merge during an animation. While doing so, they still approximate the same desired behavior, from which they derive their interaction forces. However, simulating structured objects, like soft bodies or human organs, is inappropriate with this method.

Recently, we proposed a new multiresolution animation method, based on a local discretization of linear elasticity laws [5]. The main drawback of this method is the empirical computation of the discrete operators involved in the equations. No error bounds were found for these operators when applied to irregular grids. Therefore, the dynamic behavior of the material is not guaranteed to be similar to the original one when several sampling rates coexist.

1.2. Contributions and Overview

This paper includes two major contributions, leading to a real-time adaptive animation technique for deformable bodies. First, we derive new local differential operators to ensure guaranteed error bounds. These operators will be proven equivalent to the finite element formalism. Second, we introduce a simple, general adaptive technique to handle adaptivity over a general, non-nested hierarchy of well-conditionned meshes describing an object. Different levels communicate through a simple "ghost node" technique,

1087-4844/00 $10.00 © 2000 IEEE

similar to those used in domain decomposition methods. The resulting model ensures an optimized accuracy.

Section 2 details the dynamic model and the way accurate differential operators are derived. Section 3 explains how several volumetric meshes can be simulated simultaneously, and section 4 describes the adaptive simulation technique, which automatically activates the appropriate mesh in a given region. Sections 5 and 6 present our results and conclude.

2. Dynamic Model for Simulation

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 its *displacement field* **d**, the difference between current and rest positions (bold characters represent vectors). We use the Lamé formulation [5]:

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

This equation states that the acceleration **a** of a point, times its density ρ, is the weighted sum of two terms: $\Delta\mathbf{d}$, the *laplacian* vector of the displacement field, and $\nabla(\nabla \cdot \mathbf{d})$, the gradient of the divergence of the same field, called gradient of the volume expansion in fluid mechanics[1]. λ and μ are the *Lamé* coefficients determining the material's behavior. 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.

2.1. Accurate Discrete Operators

A set of partial differential equation (PDE) defining the behavior and valid *everywhere* within the material defines its behaviour. Numerical methods, such as Finite Differences (FD) or Finite Volumes (FV), try to enforce a local respect of the PDE. With the development of the calculus of variations, variational techniques like the Finite Element technique has been preferred, since they are versatile enough to handle cases, such as arbitrary boundaries, where FD are unsuited.

This paper uses a mixed FV/FE formulation [15] 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. The key behind the mixed FV/FE method is Gauss' theorem, which turns a local integral over a region of a derivative into a line integral over the boundary of this region:

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

$\frac{\partial}{\partial i}\mathbf{X}$ is the derivative with respect to coordinate i of the field **X**, and $\mathbf{n}_i$ is the i^{th} component of the normal to the region. The object is tiled by assigning each sample point (or particle) a specific volume, chosen to be its *Voronoi region* (similarly to Natural Element Method [2].). We use Gauss' theorem to ensure that the PDE is satisfied by all these tiling regions. Operators are derived using piecewise bilinear (resp. trilinear) interpolation of the field (FE) within each triangle (resp. tetrahedron) of the object's mesh.

[1]The nabla vector ∇ coordinates are defined as $(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})$. First derivatives can be expressed as $\mathbf{grad}\ \mathbf{X} = \nabla\mathbf{X}$ and $div\ \mathbf{X} = \nabla \cdot \mathbf{X}$.

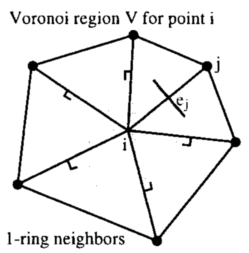


Figure 1. *Point i samples its Voronoi region. The field is piecewise linear over each triangle.*

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' formula (2) to each component d of the **d** (**X** being replaced by $\nabla\ d$) gives:

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

We assume a linear displacement field, and thus a constant field gradient over each triangle. Let (i, j, k) be a triangle of the Voronoi region around node i (Fig 2(a)). The gradient there is $\nabla d = \frac{1}{2\mathcal{A}_{ijk}}[(d_j - d_i)\vec{\mathbf{ki}}^{\perp} + (d_k - d_i)\vec{\mathbf{ij}}^{\perp}]$ where $\mathcal{A}_{ijk}$ is the triangle area and $\mathbf{X}^{\perp}$ is orthogonal to **X**, and of same length. Integration on the boundary of $\mathcal{V}$ gives:

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

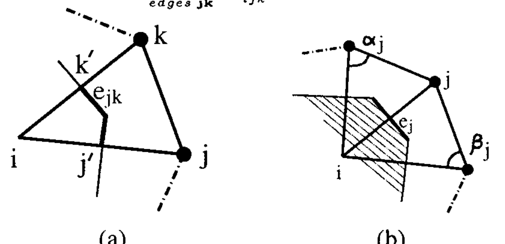


Figure 2. *(a) The Laplacian operator uses an integral over the boundary of its Voronoi region, and involves (b) the cotg of the edge opposite angles*

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

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

The Laplacian is assumed smooth enough to be sampled by point i over the whole Voronoi region, thus we can separate the volume integral. The previous equation yields:

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

The weighting coefficient $(cot\ \alpha_j + cot\ \beta_j)$ for edge e_j only depends on the mesh geometry at rest. It is precomputed and stored for increasing efficiency. Some geometrical considerations show that this is a first order approximation of the Laplacian. Moreover, if $\alpha_j = \beta_j$, the formula becomes a second order approximation. In 2D, this condition holds everywhere for a mesh made of equilateral triangles. In 3D, using such a regular mesh is not possible, but the angle constraint can govern a mesh optimization pre-process yielding well conditioned simulation.

Gradient of Divergence operator

We compute $\nabla(\nabla \cdot \mathbf{d})$ 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 (linear function, which is null at two of the three points of the triangle, and is 1 at the i^{th} point.), 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). ∇W_i is collinear to the height h_i of the triangle which passes through point i and satisfies:

$\nabla W_i = \frac{1}{h_i} \frac{\mathbf{jk}^\perp}{||\mathbf{jk}||}$

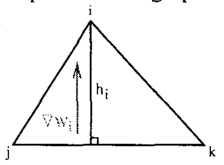


Figure 3. *Divergence is computed from the triangle basis functions W_i.*

Using $\sum_{1..3} W_i(x) = 1$ ($\sum_{1..3} \nabla W_i = 0$) and the area $\mathcal{A}_{ijk}$ of triangle (i, j, k) (Fig 3), we have:

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

The divergence of the field is :

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

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

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

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

2.2. Extension to 3D

We use slightly different formulation in 3D: instead of expressing the operators for each edge of a tetrahedron, 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 each element.

Let α_i be a vector orthogonal to the face opposed to point i, of length $\frac{1}{h_i}$ (h_i being the element height at i). Then:

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

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

The weight is a scalar in $\Delta \mathbf{d}_i$ and a 3x3 matrix for $\nabla(\nabla \cdot \mathbf{d})$.

2.3. Comparison with Finite Elements

The finite element formalism usually computes coefficients on a *per element* basis. For each element, an *elementary stiffness matrix* is computed, coefficients being based on the heights of the element. These matrices are then grouped into 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 just updated as in [14]) 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 [17, 6] avoided the costly matrix inversion. However, none of them achieved a garanteed real-time simulation.

If one develops the elementary stiffness matrix of an element and isolates the influences of i over j in that element, they will find back equations 3 and 4 (separating the two terms using λ and μ). Each operator is represented as a 3X3 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 anti-symmetric 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, states that the orthogonal component of the gradient of the displacement field must be null on the boundary. Extra terms can hence be skipped.

2.4. Adding Rayleigh Damping

Adding damping is necessary in order to increase realism since real objects never oscillate indefinitively. We use the damping force in [17]: we add to the strain tensor the contribution of a *strain rate tensor*, which measures the rate at which the strain is changing inside the material. The equations are similar to (1), except that we now use the first time derivative of the displacement (i.e. the point's speed). This is known as Rayleigh damping. The added acceleration is:

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

where $\mathbf{v}$ represents velocity vector. ϕ and ψ control the internal kinetic energy dissipation. This equation reduce internal vibrations without damping rigid motion.

2.5. Simulation of a Given Mesh

At each time step:

1. Compute $\Delta\mathbf{d}$, $\Delta\mathbf{v}$, $\nabla(\nabla\cdot\mathbf{d})$ and $\nabla(\nabla\cdot\mathbf{v})$ node.
2. Compute the acceleration at each note from (1) and (7).
3. Integrate acceleration, update position and velocity.

Figure 4a compares the animation of a cube under gravity at three spatial resolutions (27, 57 and 135 points). A lateral face is fixed and we measure the vertical displacement of the corner circled in Fig 4b. The physical coefficients were the same in the 3 simulations, and we observe similar results.

To conclude, our new algorithm is as simple as a mass-spring system, with a comparable complexity. But the finite element formalism used here guarantees accurate results. Moreover, its independency from mesh resolution will allow us to combine several resolutions at the same time step, as explained next.

3. Levels of Detail

As stated earlier, compute deformations at a pre-defined resolution can be very inefficient: a high resolution is needed in highly deformed areas, while a coarser level of detail is sufficient in stable areas. This paper proposes an automatic adaptation of the level of detail used to compute deformations, unabling to concentrate computational load where and when needed.

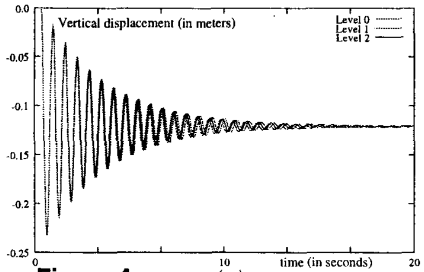
3.1. LOD representation with non-nested meshes

Previous works on adaptive resolution[12, 5, 17] have used recursive subdivision of an initial mesh for providing the LODs needed. The advantage of this formulation is that subdivided and not-subdivided parts of the mesh can interact through common nodes and edges. However, using recursive subdivision for defining LODs of a tetrahedral mesh 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.

We avoid these drawbacks by using arbitrary, independently defined meshes for representing the LODs. Each mesh is a quasi-uniform sampling of the deformable body at a given resolution. No vertex needs to be shared. In the remainder of this paper, the terms "parent mesh" and "child mesh" are used for the meshes that represent the deformable body at the immediately coarser (respectively finer) scale with respect to the current mesh.

3.2. Interface between LOD

When different regions of the deformable body are sampled using different LODs, meshes must "cooperate" at the interface between regions. This is done using a simplified domain decomposition method (see `http://www.ddm.org/`): LODs slightly overlap at their interface, and some points, called *ghost nodes*, transmit the information between meshes. To perform computations at a node P near the interface between two LOD (Figure 4(c)), displacement field values stored at neighboring nodes such as P_1 are needed. However, no simulation is performed at P_1 since the latter belongs to a region simulated at another LOD: P_1 is a "ghost node" as opposed to "active". We then approximate P_1's displacement by linearly interpolating values computed at the current resolution in this region (this is consistent with the fact that we use linear finite elements). In practice, a node preprocesses and stores the barycentric coordinates with respect to its 'parent' and 'child' tetrahedron needed for future interpolation. If a node at the surface of the object is not *inside* the coarser mesh, we choose the closest tetrahedron — a coordinate being negative. Note that a ghost node may interpolate from



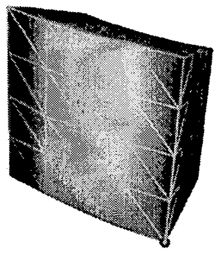
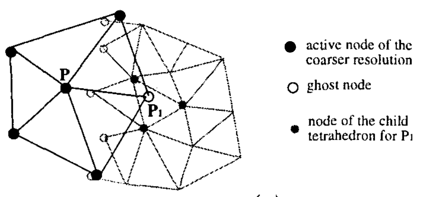


P
P1
active node of the coarser resolution
ghost node
node of the child tetrahedron for P1

Figure 4. *(a)* *Vertical displacement of the corner of a cube, with gravity and damping. Three different. resolutions are plotted.* *(b)* *The cube in its final rest position. Plotted point is circled.* *(c)* *Ghost nodes are used as an interface between the different LODs. (coarse mesh is black, finer mesh is grey*

nodes which are ghosts too: displacement value are recursively pulled through ghost nodes from the finest simulated level (see section 4.2).

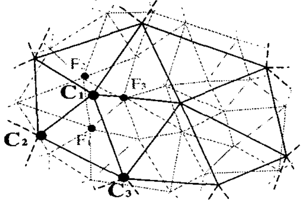


Figure 5. *Linking LODs: C_1 may pull information from its child (F_1, F_2, F_3), or F_1 may push it from its parent (C_1, C_2, C_3).*

4. Adaptive Simulation

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[5]:

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

h is the minimum distance between adjacent nodes, ρ_0 is the material's density, the square root term is the speed of sound in the material. In practice, we chose the different time steps associated with the LODs as inverse powers of two subdivisions of the display frame rate: $dt_{mesh} = dt_{framerate}/2^n$ (then the average edge length should be halved between two consecutive meshes). This condition optimizes the simulation and leads to an intuitive LOD representation.

Our meshes are generated from a triangulation of a closed surface using GHS3D [19]. They could be optimized using a relaxation technique : according to the criteria in section 2, length of edges and dihedral angles between two adjacent tetrahedra should be set to be as equal as possible. This relaxation process would only be applied to the internal nodes of each mesh, thus preserving surface appearance.

4.1. On-the-fly adaptivity

Our aim is to generate automatic local adaptation of the LOD during simulation. The nodes of a mesh represent regions, their values being the average of the local material's properties within this region. When sampling becomes too coarse, 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 stated in Section 2, the stress-strain tensors assume local linear deformations. We thus need to refine regions where a this approximation is not sufficient. So we use a criterion based on the second spatial derivatives of the displacement field: $\delta_i = \hat{h}_i^2 ||(\Delta \mathbf{d})_i||$ where $\hat{h}_i$ is the average distance between the sample point i and its 1-ring neighbors. 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.

4.2. Guaranteeing real-time

Our algorithm is: at each time step, for each LOD,

1. Compute displacement field for ghost points 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.

The transmission from finer to coarser level (step 1) is not performed each time a finer node moves. As the parent is simulated only every other step, it only has to be done before the parent level is simulated. Computational time is linear with respect to the number of active nodes. To achieve a given frame rate, we make sure that this number of 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 rate (100-10000Hz). The time needed to compute the simulation 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, especially when we are waiting for user interaction at constant time intervals). In practice, the program measures before each display the computation time that was needed by the simulation steps. When it exceeds 95% of the display step, splitting is forbidden, and a warning is sent to the user. This results in almost constant frame rates.

5. Results

Collision detection is 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.

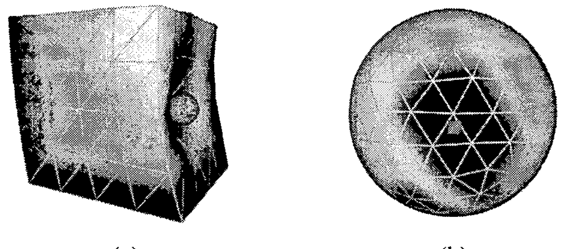


Figure 6. *(a) With $\mu = 10^6$, volume is almost preserved. (b) Intuitive sampling created by the cubic (pseudo-colored display).*

6. Conclusion

This paper has presented a new method for computing dynamic simulations of elastic bodies in real-time. We have introduced a multiresolution technique based on several (non-nested) 3D meshes representing different LODs. The method has strong connections to finite element method, but force computation at each node is local, and thus efficient. On-the-fly local switches between levels of detail are allowed to concentrate computational time only where needed.

Future research includes interfacing the simulator with a force feedback device. Parallelization 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 [9]. As a last remark, applying topological changes to the deformable body during the simulation would be very useful in applications such as surgery simulators (see [13]). An advantage of our approach among others [14] 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: Mathieu Desbrun 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] D. Baraff and A. Witkin. Large steps in cloth simulation. *SIGGRAPH 98 Conference Proceedings*, , pages 43–54. July 1998.

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

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

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

[5] G. Debunne, M. Desbrun, A. Barr, and M.-P. Cani. Interactive multiresolution animation of deformable models. In *10th Eurographics Workshop on Comp. Anim. and Sim.* www-imagis.imag.fr/Publications/1999/DDBC99, 1999.

[6] H. Delingette, St. Cotin, and N. Ayache. A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation. In *Computer Animation*, Geneva Switzerland, May 1999.

[7] M. Desbrun and M.-P. Cani. Space-time adaptive simulation of highly deformable substances. *INRIA Technical Report*, RR-3829 (www.inria.fr/RRRT/RR-3829.html), Dec. 1999.

[8] M. Desbrun and M.-P. Cani-Gascuel. Smoothed particles: A new approach for animating highly deformable bodies. *7th Eurographics Workshop on Animation and Simulation*, pages 61–76, Sept. 1996.

[9] M. Desbrun, M. Meyer, P. Schröder, and A. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH 99 Conference Proceedings*, 1999. Los Angeles, CA.

[10] M. Desbrun, P. Schröder, and A. Barr. Interactive animation of structured deformable objects. In *Graphics Interface'99 proceedings*.

[11] J.-P. Gourret, N. Magnenat Thalmann, and D. Thalmann. Simulation of object and human skin deformations in a grasping task. *Computer Graphics*, 23(3):pages 21–29, 1989. *SIGGRAPH'89 Conf. Proc.*.

[12] D. Hutchinson, M. Preston, and T. Hewitt. Adaptive refinement for mass/spring simulation. In *7th Eurographics Workshop on Comp. Anim. and Sim.*, pages 31–45, September 1996.

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

[14] D. James and D. Pai. Art defo - accurate real time deformable objects. In *Proceedings of SIGGRAPH '99* August 1999, Computer Graphics Proceedings, pages 65–72.

[15] S. F. McCormick. *Multilevel Adaptive Methods for PDE. Chap. 2: The Finite Volume Method.* Vol. 6, SIAM, Philadelphia, 1989.

[16] D. Metaxas and D. Terzopoulos. Dynamic deformation of solid primitives with constraints. *Comp. Graphics*, 26(2):pages 309–312, 1992.

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

[18] A. Pentland and J. Williams. Good vibrations: Modal dynamics for graphics and animation. *Computer Graphics*, 23(3):pages 215–222, July 1989. Proceedings of SIGGRAPH'89, July 1989.

[19] SIMULOG. Ghs3d. www.simulog.fr/itetmeshf.htm

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

[21] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. *Computer Graphics*, 21(4):pages 205–214, July 1987. Proceedings of SIGGRAPH'87.

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

[23] A. Witkin and W. Welch. Fast animation and control for non-rigid structures. *Computer Graphics*, 24(4):pages 243–252, August 1990. Proceedings of SIGGRAPH'90 (August 1990).