Interactive multiresolution animation of deformable models

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

Abstract

This paper presents an approach to animate elastic deformable materials at interactive rates using space-time adaptive resolution. We propose a new computational model, based on the conventional Hooke's law, that uses a discrete approximation of differential operators on irregular grid. It allows local refinement or simplification of the computational model based on local error measurement. We in effect minimize calculations while ensuring a realistic and scale-independent behavior within a given accuracy threshold. We demonstrate this technique on a real-time virtual liver surgery application.

1 Introduction

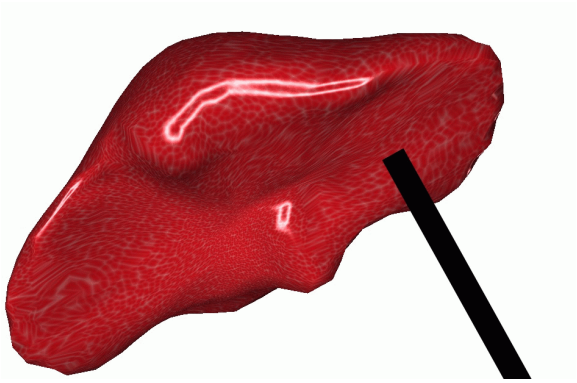
Although simple interactive animation techniques exist and are used in virtual reality systems for instance, they mainly simulate rigid bodies. Using simplified solid mechanics laws, they focus on issues like collision detection and contact modeling, where naive approaches are computationally intensive [Bar96, Fau98]. A smaller amount of effort has been put into deformable object simulation. The majority of the existing techniques have to be performed off-line, and cannot be used in a virtual environment with real-time display.
However, recent work has demonstrated unrivaled low computation times for deformable objects or surfaces [BW98, DSB99]. These approaches rely on implicit integration to advance the simulation in time with no or few concerns about stability even at large time steps. We may soon observe a number of VR applications, like interactive animation of deformable tissues for surgery training for instance.
The current state-of-the-art animation techniques use constant spatial discretization (fixed number of mass elements). Yet, we should be able to save even more computation in adapting discretization accordingly to the complexity of the occurring motion. A body undergoing significant local deformation should be refined in this region only, to ensure both a precise geometrical description of the deformation and a prescribed accuracy. Large amount of computation may be saved using such an adaptive technique, similarly to what is now widely used in simulation of lighting by radiosity techniques. Unfortunately, ensuring a same global behavior for the object, whatever the discretization rate is, remains challenging.

1.1 Prior work

The first model in Computer Graphics to animate deformable bodies was introduced by Terzopoulos *et al.* [TPBF87], using finite differences or finite elements 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 [TW88, TF88]. Finite element techniques have also been proposed [GMTT89], including a real-time simulation of elastic bodies [BNC96, DCA99], but using quasi-static models, thus loosing the dynamic behavior. As these physically-based methods are computationally intensive, other approaches appeared, allowing fast animation of simple dynamic objects by taking into account only some possible deformations or vibration modes [PW89, WW90, MT92]. Unfortunately, such restrictions on the behavior considerably affect the realism of the animation.
As mentioned earlier, all these techniques use a fixed space discretization rate, and also usually a fixed time discretization rate. Recently, a model using adaptive resolution has been developed for the simulation of hanging clothing [HPH96].

[*]iMAGIS is a joint project of CNRS, INRIA, INPG & UJF.
{debunne,mpc}@imag.fr {mathieu,barr}@cs.caltech.edu

This article was published in the proceedings of the Eurographics Workshop on Computer Animation and Simulation'99, Springer-Verlag Editor.



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 idea 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 a global and identical behavior during the animation: the dynamic behavior of the simulated object will change incoherently when a refinement occurs. Additionally, mass increases with subdivision. Even if collisions with obstacles are handled correctly, the cloth weight changes and prevent 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 [DCG96, GCD$^+$98]. This time, a state equation which represents the object's behavior (like stiffness) is defined by the user. The particles discretizing the material subdivide and merge according to a local energy criterion, and derive appropriate interaction forces from the state equation to ensure the same global behavior. Simulating structured objects like human organs with this method is, however, inappropriate. From a theoretical point of view, the SPH formalism is really adequate for a large number of particles if a desired accuracy is called for. In practice, the model as proposed simulates viscous fluids, and is not well conditioned for structured objects. We use the same "philosophical" approach in this paper, as we propose to adaptively simulate a given equation of motion. Yet a large amount of particles is no longer needed to obtain convincing results.

1.2 Approach

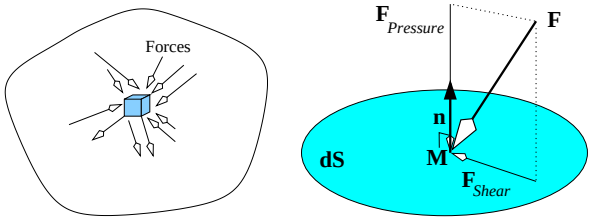
In this paper, we propose to improve upon the amount of sample points needed to animate deformable objects. Using a general elasticity model, we derive a partial derivative equation guaranteeing a global behavior for the object in Section 2. We then propose in Section 3 simple differential operators to integrate this latter PDE even on moderately irregular grids. Once an error criterion is defined, we demonstrate in Section 4 that a space-time adaptive integration, using local refinement/simplification of the mass and time discretization, is easily handled. We describe our implementation in Section 5, detailing data structures and surface display. Finally, we show a virtual surgery simulation resulting from our approach in Section 6, and conclude in Section 7.

2 A general physical model

In this paper, we basically simulate the same model as in [TPBF87]. Nevertheless, we distinguish from this approach in our mathematical development. This section reviews the standard physics used by our method, detailed in [TG70] for instance.

2.1 Notation

We use a slightly nonstandard notation for the sake of simplicity throughout this paper. Vectors will be indicated in bold: $\mathbf{u} = \begin{pmatrix} u_x & u_y & u_z \end{pmatrix}^T$, matrices in calligraphy: $A = \mathbf{u}\mathbf{u}^T$. We also make use of compact notation for derivatives. For instance, $u_{x,y} = \partial u_x / \partial y$, or $u_{z,xx} = \partial^2 u_z / \partial x^2$.



(a) (b)

Figure 1: *(a) Forces acting around a mass element. (b) Force acting on a given surface element dS, centered on a point M and defined by its normal* **n**.

2.2 Stress tensor

A small element **M** of matter receives forces from all around (Figure 1(a)). One way to describe these peripheral forces acting locally is to evaluate the surface force (called stress) acting on a given surface element centered on **M** with a normal **n**. This force **F** will have a component along **n**, analogous to a pressure, and an orthogonal component, creating shearing (Figure 1(b)). The *stress tensor* then defines a *linear* application between all normals and their associated stresses. This 3×3 symmetric matrix, usually noted σ, actually gives the applied stress force **F** for a given surface element with normal **n**:

$$\sigma \mathbf{n} = \mathbf{F}.$$

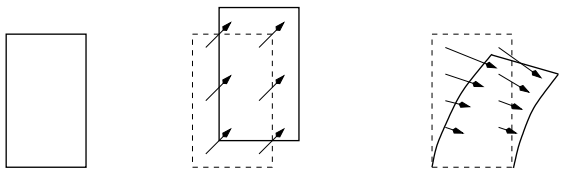
From this tensor, we deduce the *resulting force per volume* unit acting on the matter element as being the divergence of σ[1]. Then, if ρ is the mass density of the considered element, **g** the gravity acceleration, and **a** the acceleration of this element, we can use the fundamental principle of mechanics to write:

$$\rho \mathbf{a} = Div\, \sigma + \rho \mathbf{g}. \quad (1)$$

In order to compute the stress tensor, we need to know the current state of deformation undergone in the material to derive local forces from it. We next define the strain tensor for that matter.

2.3 Strain tensor

We call **d** the *displacement* of an element of matter from its initial position. This defines a vector field in the body. A pure translation of an object will create a constant displacement field, while complex deformation can create an arbitrary field as sketched in Figure 2.



Rest position Translation Deformation

Figure 2: *An object and some possible displacement fields defining the current shape.*

By definition, the gradient of this vector field is:

$$A = \overline{grad}(\mathbf{d}) = \begin{pmatrix} d_{x,x} & d_{x,y} & d_{x,z} \\ d_{y,x} & d_{y,y} & d_{y,z} \\ d_{z,x} & d_{z,y} & d_{z,z} \end{pmatrix} \quad (2)$$

The antisymmetric part of this matrix represents only the rotational part of the displacements, while the symmetric part, called *strain rate tensor* and noted ε, expresses the intrinsic deformation rate acting on an element of matter:

$$\varepsilon = \frac{1}{2}(A + A^T) = \frac{1}{2}\begin{pmatrix} 2d_{x,x} & d_{x,y}+d_{y,x} & d_{x,z}+d_{z,x} \\ d_{x,y}+d_{y,x} & 2d_{y,y} & d_{y,z}+d_{z,y} \\ d_{x,z}+d_{z,x} & d_{y,z}+d_{z,y} & 2d_{z,z} \end{pmatrix} \quad (3)$$

As for the stress tensor, we emphasize that this tensor captures only first-order deformation rates, and must be seen as a linear approximation of the local deformations.

2.4 Deformation law

A physical model for an object defines how this object deforms accordingly to applied forces, and vice-versa. Thus, we have to define a relation between the stress tensor and the strain tensor. We choose the *Hooke's law* as it is one of the simplest, yet it describes precisely enough a large range of common materials. This law stipulates (with I_3 being the 3×3 identity matrix):

$$\sigma = 2\mu\varepsilon + \lambda\, trace(\varepsilon)\, I_3 \quad (4)$$

Given this hypothesis, we can now deduce displacements from forces, or forces from displacements.

2.5 Lamé equation

If we use the above deformation law, the global equation of motion can be rewritten, omitting gravity for simplicity, as (see [TG70]):

$$\rho \mathbf{a} = \mu \Delta \mathbf{d} + (\lambda + \mu) \nabla (div\, \mathbf{d}) \quad (5)$$

by just substituting Hooke's law in the fundamental equation of motion $\rho\mathbf{a} = Div\, \sigma$ and expanding the different derivative terms. This formulation, initially due to Lamé, encapsulates the strain/stress law in a partial derivative equation that offers another interpretation of the Hooke's law. We note that such a physical model is the composition of a wave propagation and of a volume preservation constraint. Since the acceleration is the second time derivative of the displacement **d** and $\Delta\mathbf{d}$ is the sum of the second spatial derivatives, the first part of the latter equation $\rho\mathbf{a} = \mu\Delta\mathbf{d}$ is, indeed, a hyperbolic partial derivative equation, also called *wave equation*. The velocity of propagation in this case is: $c = \sqrt{\mu/\rho}$. The other part, $\rho\mathbf{a} = (\lambda+\mu)\nabla(div\, \mathbf{d})$, represents a volume preservation term. Since $div\, \mathbf{d}$ is the *volume expansion*, following the gradient of the volume expansion will tend to restore the initial volume. According to the values of λ and μ, we can interpret Hooke's law as a deformation wave with more or less compressibility [2]. This interpretation will help us to design our numerical simulation.

3 Simulation at a fixed resolution

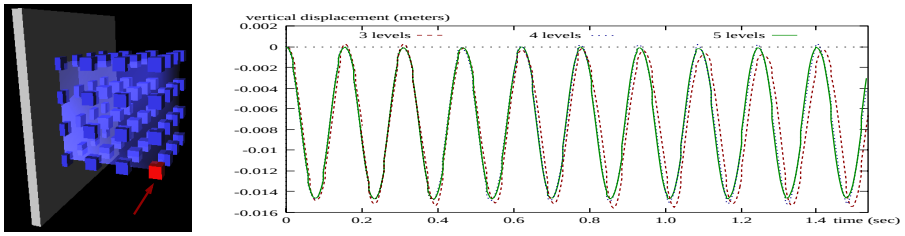
In this section, we present how we implement the physical model introduced above for a fixed, given resolution. Although we basically simulate the same model than in [TPBF87], our numerical algorithm is completely different. In particular, we will show that we can extend it to handle adaptive resolution easily.

3.1 Principle

Since we need to simulate a given material at different levels of resolution, we have to rely on a behavior equation (which can be called *state equation*, or *differential equation of motion*) defined regardless of any time or space discretization. As mentioned in the previous section, we decide to adopt the general Hooke's law to describe our physical model. In order to spare as much computation as possible, we will use the *Lamé equation* (Equ. (5)) as no strain and stress tensors need to be used. It encapsulates the physical model into a simple partial differential equation, hiding the use of strain and stress, thus sparing memory requirement. Animating an object requires a mass discretization, followed by an integration of the PDE over the sample point-masses, and over time. As the object moves and deforms,

[1] The divergence operator *Div* for a matrix is the vector formed of the divergence *div* of each line, with $div(\mathbf{u}) = u_{x,x} + u_{y,y} + u_{z,z}$.

[2] Although it can not be perfectly achieved with this formulation, the preservation of the object's volume is usually considered as good when $\lambda > 100\mu$, which we use in our examples.



(a) (b)

Figure 3: *A 10 cm cube oscillating with gravity, one of its face being fixed. We measure the vertical displacement of one of its corner (arrow) at different spatial resolutions. Levels are made of 64, 512 and 4096 particles. Note that no damping at all was used for this simulation.*

the discretization may end up being an almost arbitrary grid. We thus need to define a way to efficiently integrate this PDE over an irregular sampling, while ensuring a good accuracy threshold.

3.2 Discrete approximation of operators

To be able to integrate Equ. (5), we just need to approximate two operators: the Laplacian of the displacement field, $\Delta\mathbf{d}$, and the gradient of the divergence of the same field: $\nabla(div\,\mathbf{d})$. Although Finite Difference formulas exist on regular grids, they do not apply here: we must assume an arbitrary complex grid since the object continuously deforms, and also because the discretization itself may change. However, approximating such operators on irregular grids is not an easy task, and has been extensively analyzed in physics and mathematics.

Laplacian operator

Milne in his thesis shows how sensitive the approximation of a second derivative can be in 1D, creating noise source problem as soon as neighboring samples are not centered [Mil95]. Fortunately, the extension of Finite Differences in 1D proposed by Fornberg [For88] solves this noise problem. It consists in fitting a quadratic function between a sample and its two closest neighbors. For three samples of a function f spaced at respectively Δ and δ from the central point, we obtain:

$$f_i'' = \frac{2}{\delta+\Delta}\left(\frac{f_{i-1}-f_i}{\delta}+\frac{f_{i+1}-f_i}{\Delta}\right)$$

A straightforward generalization of this formula to 3D gives the scale-dependent umbrella operator [DMSB99, Fuj95] (where $l_{ij} = ||\mathbf{d}_i - \mathbf{d}_j||$ is the distance between sample points i and j.):

$$\Delta\mathbf{d}_i = \frac{2}{\sum_j l_{ij}} \sum_{j \text{ neighbors}} \frac{\mathbf{d}_j - \mathbf{d}_i}{l_{ij}} \tag{6}$$

This simple formulation, recently introduced in Computer Graphics for mesh smoothing through diffusion for its better properties compared to the uniform umbrella operator, allows us to have a good approximation of the Laplacian whose accuracy depends little on the neighboring distribution of particles. We delay the quantitative results to Section 3.3.

Gradient-of-divergence operator

Now that we have a robust Laplacian operator, we must derive a gradient-of-divergence operator in a coherent way, to provide a stable pair of operators for our simulations.
By expanding again the second derivatives involved in the Laplacian operator, we find the following relation:

$$\Delta\,\mathbf{d} = \nabla(div\,\mathbf{d}) - \nabla\times(\nabla\times\mathbf{d}) \tag{7}$$

where $\times$ is the cross product operator.

We know that the divergence of $\mathbf{d}$ is a measure of the volume expansion as mentioned in Section 2.5. As shear strains don't create any volume change, only normal (also called radial) strains affect the volume. Therefore, we propose to decompose the Laplacian into a radial and a rotational component. Remembering that for any vector

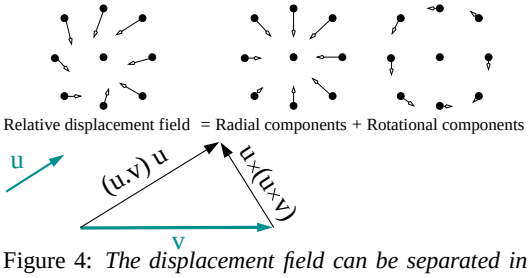


Figure 4: *The displacement field can be separated in two components: the radial component (created by pressure forces) and the rotational component (created by shear forces).*

$\mathbf{v}$ and unit vector $\mathbf{u}$, we decompose $\mathbf{v}$ along $\mathbf{u}$ and a vector normal to $\mathbf{u}$ through the relation: $\mathbf{v} = (\mathbf{v}\cdot\mathbf{u})\mathbf{u} - \mathbf{u}\times(\mathbf{u}\times\mathbf{v})$ *with* $\mathbf{u} = \mathbf{l}_{ij}/l_{ij}$ (see Fig. 4), we can expand Equ. (6) into:

$$\Delta\mathbf{d}_i = \frac{2}{\sum_j l_{ij}} \sum_{j \text{ neighbors}} \frac{\mathbf{d}_j - \mathbf{d}_i}{l_{ij}}$$

$$= \frac{2}{\sum_j l_{ij}} \sum_{j \text{ neighbors}} \frac{\left[(\mathbf{d}_j - \mathbf{d}_i)\cdot\frac{\mathbf{l}_{ij}}{l_{ij}}\right]\frac{\mathbf{l}_{ij}}{l_{ij}} - \frac{\mathbf{l}_{ij}}{l_{ij}}\times\left[\frac{\mathbf{l}_{ij}}{l_{ij}}\times(\mathbf{d}_j - \mathbf{d}_i)\right]}{l_{ij}}$$

$$= \frac{2}{\sum_j l_{ij}} \sum_{j \text{ neighbors}} \frac{[(\mathbf{d}_j - \mathbf{d}_i)\cdot\frac{\mathbf{l}_{ij}}{l_{ij}}]\frac{\mathbf{l}_{ij}}{l_{ij}}}{l_{ij}} - \frac{2}{\sum_j l_{ij}} \sum_{j \text{ neighbors}} \frac{\frac{\mathbf{l}_{ij}}{l_{ij}}\times(\frac{\mathbf{l}_{ij}}{l_{ij}}\times(\mathbf{d}_j - \mathbf{d}_i))}{l_{ij}} \tag{8}$$

Therefore, comparing Equ. (7) and (8), we decide to identify the radial component as being the gradient of divergence:

$$\nabla(div\,\mathbf{d})_i = \frac{2}{\sum_j l_{ij}} \sum_{j \text{ neighbors}} \frac{[(\mathbf{d}_j - \mathbf{d}_i)\cdot\frac{\mathbf{l}_{ij}}{l_{ij}}]\frac{\mathbf{l}_{ij}}{l_{ij}}}{l_{ij}} \tag{9}$$

We will see in the next section that these two operators provide good results in practice.

3.3 Algorithm and Validity

The overall algorithm to animate our deformable model is straightforward. At each time step:

- On each particle, evaluate the internal forces using the Laplacian and the gradient-of-divergence operator,
- Deduce the acceleration using Equ. 5,
- Integrate the acceleration over a time step dt to update positions and velocities.

The Laplacian and gradient-of-divergence operators behave very well at different resolutions. Figure 3 demonstrates that even in three different resolutions, an object undergoes the same deformation in time. It validates our scale-dependent umbrella operator, and we can move forward to adapt discretization.

4 Space-time adaptive simulation

Conventional models usually discretize matter at fixed resolution in space, and often use a fixed time resolution too. These discretization rates have to be defined by the user *a priori*. As the number of mass elements and the size of the time step often affect the overall result of the animation, the user has to go through a series of trials and corrections before obtaining what (s)he wanted. Moreover, if a shock happens during the animation, the time step of the whole sequence (resp. the number of particles) has to be taken small enough (resp. large enough) to avoid divergence, resulting in unbearable computation times.
Numerous CG techniques use adaptive time step, but we propose to optimize the sampling rate of our deformable model both in space and time. Similar techniques already exist in computational physics, as the adaptive finite element method for instance. In this paper, we propose a simpler model that results in a relatively straightforward implementation and a reduced computational time. Our model is designed to offer an *automatic adaptive resolution* both in time and in space that concentrates computations where and when required. This section reviews this technique, which will offer the user a tunable trade-off between precision and efficiency by concentrating computations where and when required.

4.1 Space adaptivity

Since the basic model developed in the previous sections ensures a same behavior at any spatial resolution, we can adapt the spatial discretization rate during the simulation. If the user uses a tool to manipulate the object, adding new sample points with refined time step near the tool to increase the quality of the simulation in this area is particularly convenient. We will obtain a more accurate feedback force, along with enhanced visual complexity. When the user moves the tool to another place, the previous area can be simplified back, so that computations are now mainly dedicated to the new area of interest. Therefore, two criteria have to be defined: a criterion controlling if refinement has to be performed, and another criterion allowing simplification. The first criterion is important for accuracy, while the other one is capital to save computations.

Adaptivity criteria

Our physical model relies on approximations of second derivatives of the deformation field, indicating the rate of displacement variation. As mentioned earlier, the stress/strain tensors assume local linear deformation. As a consequence, we need a refined sampling whenever the displacement field varies too suddenly, i.e., when the local "frequency" is too high for the current discretization rate. A linear approximation does not fit such cases anymore. This can be tested using the Laplacian operator (which measures the variation from linearity), already available, through the following relation:

$$h^2 \, ||\Delta \mathbf{d}|| > \varepsilon_{max} \tag{10}$$

where h represents the shortest distance between this particle and its neighbors. We found this coarse, yet fast estimation adequate in practice: in our tests, spatial refinements appear where and when we intuitively thought it should.
Similarly, we use an opposite criterion for the simplification. A particle and its siblings can be replaced by a single coarser particle if, for each of them:

$$h^2 \, ||\Delta \mathbf{d}|| < \varepsilon_{min} \tag{11}$$

Once again, despite the simplicity of this criterion, we obtain adequate results as simplifications appear in "calm" areas as desired.

4.2 Time adaptivity

Once the space discretization has been adapted accordingly, time discretization has also to be adapted to prevent instabilities. Keeping a too large time step can introduce severe inaccuracies due to the approximation of constant acceleration during this time step. Nevertheless, too little a time step would lead to consequent loss of efficiency. We thus have to adapt the time step carefully. Moreover, we want to adapt the time step locally, so that regions undergoing no or little stress are not updated as often as regions with high stresses.

Time stepping

We use Courant's condition [DCG96] for the Lamé equation:

$$dt < h \sqrt{\frac{\rho_0}{\lambda + 2\,\mu}} \tag{12}$$

where h represents the smallest distance between this particle and one of its neighbor, and ρ_0 the material's rest density. This time step is the maximum allowable one for this particle, but we also ensure that the time step is sufficiently small to handle sudden and fast deformations. This can be done for instance by constraining the time step to satisfy:

$$||\mathbf{a}\, dt|| = ||\Delta \mathbf{v}|| < \Delta \mathbf{v}_{max}$$

This way, we will never get fast velocity changes, plausible cause of divergence. We will pick the biggest time step satisfying the two above criteria, ensuring an optimized time step for each particle. Other criteria can be added depending on the application to make sure no instabilities can arise during the animation.

4.3 Validation

Implementing these criteria for space-time adaptivity exhibits stable and adequate behaviors. As demonstrated in Figure 5, an object at its coarsest resolution at rest will subdivide in highly stressed regions, i.e., where the user acts and in regions where the deformation propagates. The time steps are adapted automatically to ensure a stable result. Here, the scale-dependent umbrella operator is critically needed, as we deal with an irregular resolution [DMSB99].

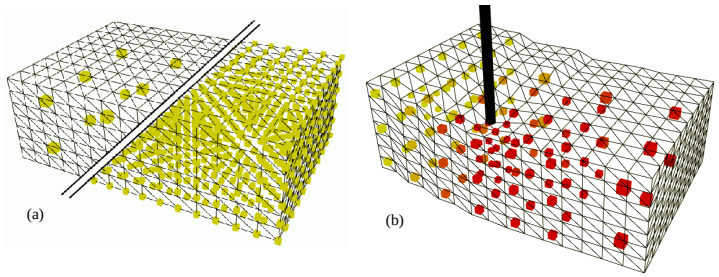


Figure 5: *A parallelepiped undergoing a tool's deformation. Figure (a) shows the coarser (24 particles) and higher (1056 particles) resolutions. Figure (b) shows the intuitive sampling occurring during simulation (the particle's color is proportional to its displacement).*

5 Implementation

In this section, we detail our implementation, presenting data structures that efficiently handle the multiresolution properties of our method. We also describe a damping implementation as well as the surface representation we used for visualization.

5.1 Spatial data structures

This implementation is not as general as the model described in previous sections: we optimized it for limited deformation in order to improve the efficiency of our simulation. We have chosen to limit the range of applications of our simulator to soft materials (such as a human organ). If we assume that during the simulation, the material will not be deformed too much, we can assume that the topology will not change. In practice, this means that the neighborhood of a given point of matter will remain the same, thus allowing us to precompute and store it, saving a lot of computational time. It also allows us to build a *hierarchical* representation of the material.

Topological octree All the particles are sorted in an octree, the 8 children (resp. the parent) of a given particle being the particles which will sample the same zone of space if this particle splits (resp. merges). The octree structure is well suited for a hierarchical representation of space, since a cube recursively divided in 8 new cubes offers the only uniform sampling of space at each level of subdivision. The octree is constructed in a bottom-up fashion as follows (see Figure 6):

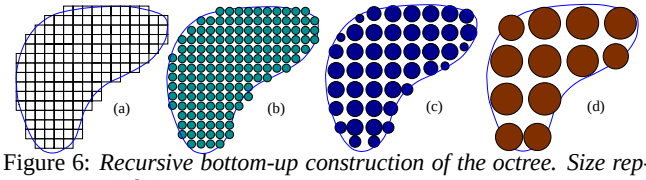


Figure 6: *Recursive bottom-up construction of the octree. Size represents particle's mass*

– (a) The cubic 3D grid is fitted on the simulated object bounding box and is then recursively divided up to its higher level.
– (b) We then determine which of these small cubes are inside the object and simply skip the other ones. The remaining particles form our octree's higher level, their mass is computed as being the object total mass divided by their number (for a uniform material).
– (c)&(d) The previous levels are then recursively built from this one, regrouping each group of 8 (or less on the boundary) particles in a new one, using

$$m = \sum m_i \ , \ \mathbf{p} = \frac{\sum m_i \mathbf{p_i}}{m} \tag{13}$$

where m is the mass, and $\mathbf{p}$ is the position of the parent particle. This average guarantees mass preservation and offers a good spatial sampling of the object.
Note that this is a *topological* octree and not a *spatial* octree. It stores the particles' child/parent relationships, but does not represent a classical octree division of space, as it will be deformed during the animation when particles move.

Neighbor structure At a given level of hierarchy, we define a particle's neighborhood as the set of all the particles that are adjacent in the topological octree, by a face, an edge or a vertice. Figure 7 shows this in 2D.

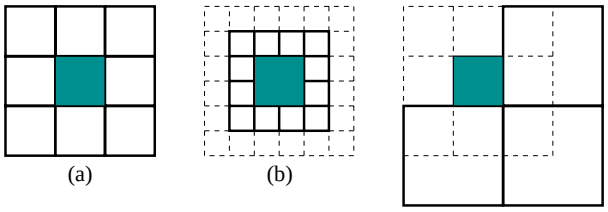


Figure 7: *Definition of the neighbors (in bold lines) of a given particle (filled), in 2D. (a) same level, (b) higher level and (c) lower level. In 3D, each particle has 89 potential neighbors.*

In order to restrict the number of possible neighbors, and for sampling quality reasons, we will ensure that our octree remains restricted [VB87]. During the animation, two neighboring *active* particles can only differ by one level in the hierarchy (see Figure 8(a)). It limits the number of possible neighbors of a given particle, and still assuming that the object is not too deformed, allows us to precompute and store the entire neighborhood of each particle. In practice, in 3D, the total number of potential neighbors is 89 for each particle, but as they cannot be all active at the same time, the effective maximum number of active ones is 56 (if all the neighbors are split), and the minimum is 14 (all the neighbors are merged).

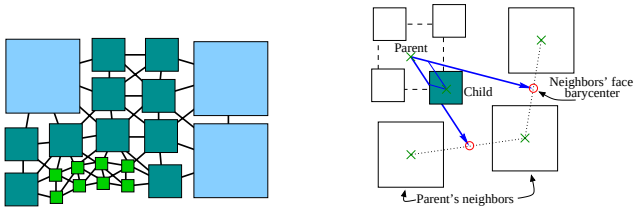


Figure 8: *(a) During a simulation, particles' neighbors can only belong to the same level of resolution, to the level above, or the level below. (b) The local coordinate system (bold arrows), defined by the parent's neighbors ensures a good spatial sampling, even when the object is deformed.*

Structure update The restricted octree imposes some constraints for splitting/merging particles. At each time step, splitting and merging lists are computed using the criterion described in section 4.1. These lists are then parsed, favoring splitting for the sake of stability:
– A particle can merge if and only if all its neighbors have an equal or lower level in the hierarchy.
– A particle can split if and only if all its neighbors have an equal or higher level in the hierarchy. If it's not the case, the lower level neighbors are inserted in the splitting list, so that this particle can split later (perhaps not at the next time step as the neighbor may also have to wait before splitting : this is a recursive process).
Each particle has pointers to all its potential neighbors, and keeps a list of all the active (i.e. really simulated) particles among them. When a particle merges or splits, precomputed tables allow us to update its active neighbor list with no computations, as well as the active neighbor lists of its neighbors.

Position When some particles merge, their parent averages their different values (position, speed, displacement...) to update its own values. The problem is more complex when it comes to the splitting of a particle. As we want our sampling of space to be as uniform as possible, the positions of the new particles have to be carefully computed. We chose to define the particle's position in a frame centered on its parent position (See Figure 8(b)).
The axis of this frame are defined by the parent's neighbors (through the center of mass of these neighbors for each direction). This local coordinate system provides adequate new particle positions, even if the splitting occurs when the object is deformed. A special treatment is done for boundary particles which local frame is defined by the mirrored parents' neighbors. During the simulation, if the parent's neighbors are not currently active, their positions are computed from their children using (13). When merging, each particle stores its new local coordinates, so that it appears at the same place when its parent splits again.

5.2 Temporal data structure

As each particle samples a volume eight times smaller than its parent, *Courant's condition* (see §4.2) stipulates that its time step should be at least twice as small as the one its parent uses. As a consequence, and in order to synchronize the time steps easily, we choose the time steps to be a power of two multiple of the minimum time step (those of the smallest particles), which is determined from the material's stiffness using equation (12).

In practice, all the active particles are sorted in lists corresponding to their time step, which are parsed when needed. Changing a particle's time step simply means to transfer it to another list. Determining at each time step which lists have to be simulated can be very quickly done using a binary operation on the value of the current number of iterations done.

5.3 Internal damping

Adding damping forces in this model allows us to handle a larger set of materials' behaviors. Damping adds realism in the animation which would elsewhere oscillate endlessly. Note that these oscillations, which are perfectly normal for an undamped elastic system, are indeed obtained with our simulation which presents a great stability during the simulation. We add to each particle a first damping force which is negatively proportional to its speed ($\mathbf{F}_d = -k_d.\mathbf{speed}$).
We also add an artificial "viscosity" in our model, by adding a force which represents the effects of a particle neighbors' motions on the particle itself. In other words, this force will try to make a particle follow its neighbors' average motion, adding internal coherence in the material. Inspired by SPH formulations [Mon92, DCG96], we have chosen to use

$$\mathbf{F}_v = \frac{k_v}{\sum_j m_j} \sum_{neighbors\, j} m_j(\mathbf{v}_j - \mathbf{v}_i) \tag{14}$$

This formulation can be seen as an extension of those used with a uniform sampling of space. As our octree remains restricted during the animation, it gives good coherent results in practice.

The influence of a neighbor has been chosen to be proportional to its mass (intuitively linked to the contact surface). This force was restricted to its damping action: it cannot accelerate a particle, nor can it inverse the direction of its speed.

Note that damping is very special, as we cannot guarantee a same behavior whatever the resolution. Our strategy, close to similar work in physics [Mon92], is to use the previous formulation as it minimizes the observed behavior difference at each level. However, we only slightly use this force in our examples, mainly to create a more rigid material. As the user only sees the multiresolution result, and as the result is visibly plausible, we stick to this method.

5.4 Surface management

We describe here how this model is embedded in a graphics environment: the discrete particles are linked with a surface representing the current shape of the object. The displayed surface is a visual interface, exhibiting the deformations that take place in the material and hiding the granularity of the model, so that the user is not even aware of the adaptive granularity. It is also used to detect collisions, and to transmit external forces to the internal physical model.

Surface representation The surface display has to be real-time, and a triangulated surface seems appropriate as most graphics engines use triangles as a primitive. The resolution of the mesh depends only on the desired visual quality, and can be completely separated from the internal discretization.

The motion of the mesh nodes is defined with respect to the motion of the inner particles. We *link* each mesh node with some of the inner particles, chosen to be the m-th closest ones within a given radius from the node in the rest position. For each of these links, a constant offset defined as the vector joining the particle's rest position and the mesh node is precomputed. During the animation, the position $\mathbf{n}$ of the node, is determined as a weighted average of its linked particles' positions $\mathbf{p}_i$ plus their respective offset $\mathbf{o}_i$. The links' weights w_i are chosen to be inversely proportional to the length of the $\mathbf{o}_i$ vector:

$$\mathbf{n} = \frac{1}{\sum_{links\ i} w_i} \sum_{links\ i} w_i(\mathbf{p}_i + \mathbf{o}_i) \qquad w_i = \frac{1}{||\mathbf{o}_i||}$$

Practically, we noticed that a very small number of particles are sufficient to produce convincing animations, which is useful for real-time applications (we use $m = 4$ currently). This surface motion's smoothing is intended to hide the underlying possible coarse resolution. The smoothing can be controlled by adjusting the maximum number of linked particles m, as well as the maximum offset length (radius of influence of the nodes, hence of the particles).
Although this model, as the offset is a constant translation, should be limited to rigid bodies motion or small deformations, it gives convincing results in practice. We hence avoid the need to use finer methods, such as the computation at each time step of a local reference frame for each particle, based on its neighbors positions, and in which the offset could be defined.

Handling multiresolution In our case, the problem is slightly more difficult since particles may appear or disappear during the simulation. We chose to create a hierarchy of links, a parent particle being linked with all its children's linked nodes. For each of these nodes, the link's offset is also precomputed from the rest position, and its weight is the sum of its children links' weights to this node.
During the simulation, active particles are parsed, and each of them gives its weighted position contribution to the nodes it's linked with. The way we computed the links ensures that the surface nodes will not be affected by a split or a merge of the inner particles as long as it takes place when they are at their rest positions. In practice, the possible surface popping effect can be controlled by the split and merge thresholds which indirectly determine how far from its rest position a particle can split/merge, thus determining the maximum surface visible shift. As a result, surface popping is almost not noticeable in our tests.

Collision detection We use the hardware-based collision detection described by [LCN99]. An offscreen rendering returns a list of the triangles that intersect the tool. Their nodes are then pulled out of the tool, and these displacements are transmitted to the inner linked particles (if a particle is linked with several moving nodes, it averages their imposed displacements). A simulation step is then normally performed and the accelerations these particles compute are then summed and transmitted to the user's force feedback device.

6 Results

We tested our implementation on different basic examples, and on a concrete medical simulation. We describe these tests and discuss results in the next two sections.

6.1 Proof-of-concept example

We chose the "classical" rod example to begin our tests. A rod is attached to a wall at one end, and bends under gravity, in a damped media. Figure 9a shows the initial and equilibrium positions. The Lamé coefficient for this rod are $\mu = 5000$ and $\lambda = 1000000$.

level	2	3	4	adaptive
nb of particles	4	32	256	4-88
simulation time (cpu units)	0.87	4.29	38.90	5.27

This table shows averaged simulation times for this simulation, using different space resolutions. As expected, the adaptive simulation offers a good trade-off, giving a computational time close to the one we have with 32 particles (3 levels), while taking advantage of the potential 256 simulated particles of level 4. The number of particles really simulated varies between 4 (beginning of the simulation) and 88 (before stabilization), stabilizing at 32 (as with level 3) when the rod is its rest position (see Figure 9b for reference snapshots of the animation)

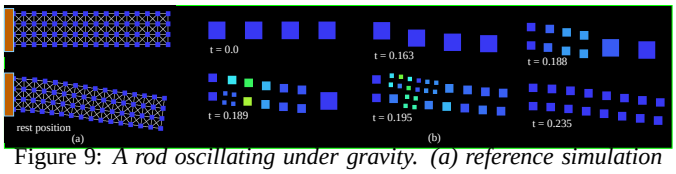


Figure 9: *A rod oscillating under gravity. (a) reference simulation made with 256 particles. (b) adaptive simulation snapsnots.*

6.2 A real-time application

Providing simulators for surgeon-apprentices has several significant advantages, both ethical and financial: it substitutes for corpses or animals, and improves the training as the surgeons can practice as much as they want for the same cost. In the context of laparoscopic surgery [3], the liver operation is a perfect case to study, being one of the most common operation with this medical technique.
We use the simulation technique developed in this paper to implement a laparoscopic surgery simulator. Using volume and surface data from a typical human liver, we provide a real-time liver simulator, that reacts to surgery tools as displayed on Figure 10. The multiresolution nature of our simulation is the key in this context: simulation at fixed fine resolution would be overkill. Focusing computations where and when needed using our space-time adaptive technique is vital for efficiency. We achieve a 30 Hz simulation using an R10K SGI Onyx2, simulating an average of 100 active particles. The frame-rate is constant and guaranteed by a limitation of the computational time, which varies linearly with the number of particles, weighted by their respective time step level.

7 Conclusions and future work

We proposed in this paper a multiresolution animation technique to animate deformable structured objects. This new computational model benefits from an adaptive sampling of both space and time to minimize calculations. Based on Hooke's law, it makes use of discrete scale-dependent approximation of derivative operators on

[3] Laparoscopic surgery is a minimally invasive technique, where surgeons operate using tools that are introduced into the patient's body.

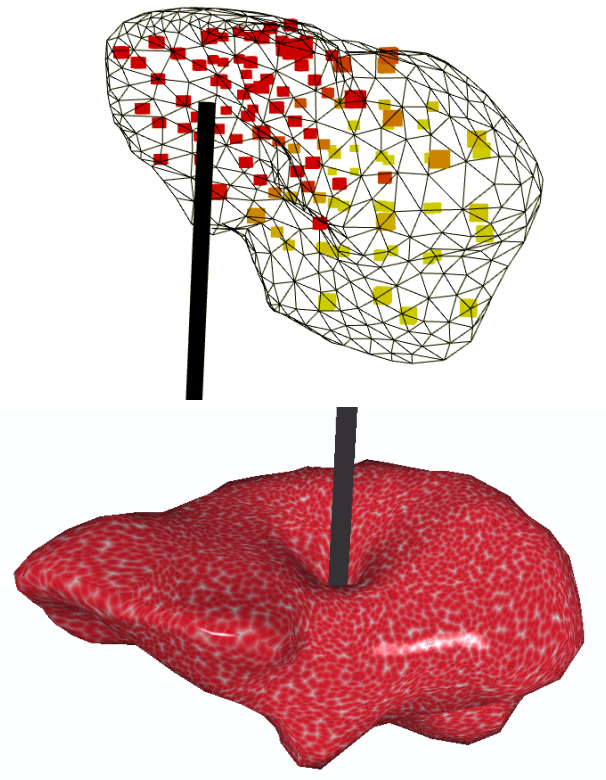


Figure 10: *Although the imposed strain sometimes highly exceeds the small deformations formalism, our model still presents good response to the user's stress. Adding textures highly increase the realism of the simulator.*

irregular meshes. Using intuitive criteria, we refine or coarsen automatically our mass sampling to guarantee an adequate error tolerance.

This approach is very general, and can be easily enhanced. Implicit integration for instance could suppress all the adaptive time steps while still guaranteeing stability for applications where little accuracy suffices. Better thresholds to control the errors would also be very useful for more accurate applications. The new discrete differential operators introduced in this article behave well on a not too deformed grid and could be enhanced to handle a larger range of simulations. Multiresolution animation would then become a powerful and efficient tool, like radiosity is, after many improvements over the last decade.

References

[Bar96] David Baraff. Linear-time dynamics using lagrange multipliers. In *SIGGRAPH 96 Conference Proceedings*, Computer Graphics Proceedings, Annual Conference Series, pages 137–146. ACM SIGGRAPH, Addison Wesley, August 1996. ISBN 0-201-94800-1.

[BNC96] Morten Bro-Nielsen and Stéphane Cotin. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. In *Eurographics*, pages 21–30, 1996.

[BW98] 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.

[DCA99] 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 '99*, pages 70–81, may 1999.

[DCG96] 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.

[DMSB99] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of arbitrary meshes using diffusion and curvature flow. In *SIGGRAPH 99 Conference Proceedings*, Annual Conference Series. ACM SIGGRAPH, Addison Wesley, 1999.

[DSB99] Mathieu Desbrun, Peter Schröder, and Alan H. Barr. Interactive animation of structured deformable objects. *To appear in Graphics Interface '99*, 1999.

[Fau98] François Faure. Interactive solid animation using linearized displacement constraints. *9th Eurographics Workshop on Computer Animation and Simulation*, September 1998.

[For88] Bengt Fornberg. Generation of finite difference formulas on arbitrarily spaced grids. *Math. Comput.*, 51:699–706, 1988.

[Fuj95] Koji Fujiwara. Eigenvalues of laplacians on a closed riemannian manifold and its nets. In *Proceedings of the AMS 123*, pages 2585–2594, 1995.

[GCD+98] J.D. Gascuel, M.P. Cani, M. Desbrun, E. Leroy, and C. Mirgon. Simulating landslides for natural disaster prevention. In *9th Eurographics Workshop on Computer Animation and Simulation (EGCAS'98)*, September 1998.

[GMTT89] 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).

[HPH96] 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.

[LCN99] Jean-Christophe Lombardo, Marie-Paule Cani, and Fabrice Neyret. Real-time collision detection for virtual surgery. In *Computer Animation'99*, May 1999.

[Mil95] Roger B. Milne. An adaptive level-set method. *Ph.D. Thesis*, University of California, Berkeley, December 1995.

[Mon92] J. J. Monaghan. Smoothed Particle Hydrodynamics. *Annu. Rev. Astron. Astrophys.*, 30:543, 1992.

[MT92] Dimitri Metaxas and Demetri Terzopoulos. Dynamic deformation of solid primitives with constraints. *Computer Graphics*, 26(2):309–312, July 1992. Proceedings of SIGGRAPH'92 (Chicago, Illinois, July 1992).

[PW89] 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).

[TF88] Demetri Terzopoulos and Kurt Fleischer. Modeling inelastic deformations: Viscoelasticity, plasticity, fracture. *Computer Graphics*, 22(4):269–278, August 1988. Proceedings of SIGGRAPH'88 (Atlanta, Georgia).

[TG70] S.P. Timoshenko and J.N. Goodier. *Theory of Elasticity*. McGraw-Hill, 1970.

[TPBF87] 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).

[TW88] Demetri Terzopoulos and Andrew Witkin. Physically based model with rigid and deformable components. *IEEE Computer Graphics and Applications*, pages 41–51, December 1988.

[VB87] Brian Von Herzen and Alan H. Barr. Accurate triangulations of deformed, intersecting surfaces. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 103–110, July 1987.

[WW90] 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).