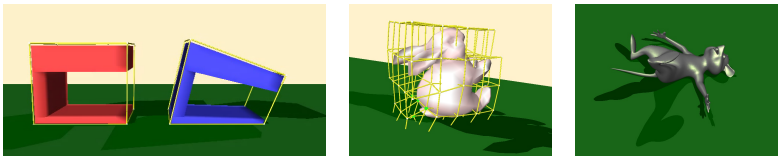
3rd Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS" (2006)
C. Mendoza, I. Navazo (Editors)

Animating Shapes at Arbitrary Resolution with Non-Uniform Stiffness

Matthieu Nesme[1,2], Yohan Payan[2] and François Faure[1]
[1]GRAVIR/IMAG-INRIA [2]TIMC/IMAG - Grenoble, France



Abstract

We present a new method for physically animating deformable shapes using finite element models (FEM). Contrary to commonly used methods based on tetrahedra, our finite elements are the bounding voxels of a given shape at arbitrary resolution. This alleviates the complexities and limitations of tetrahedral volume meshing and results in regular, well-conditionned meshes. We show how to build the voxels and how to set the masses and stiffnesses in order to model the physical properties as accurately as possible at any given resolution. Additionally, we extend a fast and robust tetrahedron-FEM approach to the case of hexahedral elements. This permits simulation of arbitrarily complex shapes at interactive rates in a manner that takes into account the distribution of material within the elements.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Physically based modeling I.3.7 [Computer Graphics]: Animation

Keywords: physically based animation, finite elements, deformable modeling, real time simulation

1. Introduction

Physically-based animation has generated a growing interest over the last twenty years, because it allows a virtual object to realistically react to external input rather than replay pre-computed animations. However, it is not yet widely used in real-time applications such as video games, because physical models are difficult to tune and their animation is computationally expensive.

In this paper, we propose a novel approach to tackle these difficulties in the case of viscoelastic deformable bodies. Viscoelastic deformation can be used to model a wide range of real-world objects, including biological tissues, cloth, and a variety of deformable manufactured objects. They are modeled as a continuous material subject to physical laws relating local strain to local stress, and discretized over a finite number of degrees of freedom to allow numerical simulation. This has been deeply studied in the domain of mechanical engineering [Bat82], and a wide variety of material laws and discretization methods has been proposed. In the computer graphics community, the seminal paper of Terzopoulos [TPBF87] showed how to produce complex and visually realistic animations, at a high computational price. Since then, a lot of work has been done to reduce the complexity using alternative material laws, spatial discretization, and time integration.

Our approach can be summarized as follows. We first build a high-resolution voxelization of the geometrical object (if not already available).We then recursively merge the voxels up to an arbitrarily coarser mechanical resolution. The merged voxels are then used as hexahedral finite elements embedding the detailed geometrical shape and animated using fast implicit time integration.At each level, the mass and stiffness of a merged voxel are deduced from its eight children, automatically taking into account the non-uniform distribution of material. The objects can simultaneously include volumetric parts, surface parts such as wings or ears, and one-dimensional parts such as tails. The animation is robust against degenerate configurations such as element inversion. We believe that these features put together make our ap-

proach suitable for interactive virtual environments such as video games.

Our specific contributions are: the automatic voxelization of a surface model, the automatic tuning of the FEM parameters based on the distribution of material in each cell, and the robust animation of hexahedral elements.

The remainder of this paper is organized as follows. Related work is briefly discussed in section 2. In Section 3, we show how to construct the voxel mesh and embed the surface in the deformable voxels. In Section 4, we present a new method to apply fast dynamic FEM to the voxels and detail how to interact with the embedded surface. In Section 5, we explain how to tune the mass and stiffness of the voxels. We discuss results in Section 6 and conclude in Section 7.

2. Related Work

The complexity of the material laws have a high impact on computation time. T he most simple viscoelastic law is the spring model [CEO*93, BHW94]. However, it has been shown that it can not accurately model 3D elasticity, and recent work focus on 3D finite elements [Hau04]. Important gains in computation time have been obtained by replacing the complex Green-Lagrange rotationally invariant strain tensor by the product of a rotation with the linearized Cauchy strain tensor [MDM*02, EK03, MG04]. This method is commonly referred as *stiffness warping* or *co-rotational elements*. Robustness in degenerate configurations such as flat or inverted elements has then been improved [ITF04, NPF05].

Spatial discretization directly impacts the number of degrees of freedom, and thus, the complexity of the system. Traditional FEM analysis requires an accurate mesh of the entire volume of the object. This is hardly applicable to visually pleasing detailed shapes. Moreover, most available computer graphics models are surface meshes, and it is hard to mesh the interior of such models with a controllable number of tetrahedra without creating nearly singular elements which result in unstable simulations, especially for small parts like the ears of the bunny or the tail of the mouse presented on the teaser. Moreover, it is difficult to animate surfaces (like clothes or dragon's wings) using a volume mesh. Some approaches try to separate rendering detail from the (possibly hierarchical) mechanical model, using an external [DDCB01] or embedded [CGC*02b, CGC*02a, MG04] rendering layer. Nevertheless, the tetrahedrization stage remains far from trivial. To avoid this volume meshing stage and control the number and shape of elements, some methods build automatically an optimized mechanical mesh using a 3D grid [MTG04, JBT04] or an octree [DMG05, NFP06]. However, the resulting mechanical properties are simplified and the meshes have to be very fine to model the objects accurately. Alternatively, it is possible to reduce the number of degrees of freedom of detailed shapes using modal analysis [BJ05] or global shape matching [MHTG05], however these methods fail to capture local deformation. Recently, meshless methods have been proposed [DC95, MKN*04, PKA*05] in order to more easily model fracture and tearing, however they are quite slow and not the best adequate for real-time animation of non-pasty soft bodies.

Time integration is an important issue when dealing with stiff objects, which need very small time steps to avoid instabilities when explicit schemes are used [PTVF92]. Baraff and Witkin [BW98] showed how to efficiently apply implicit time integration which allows large time steps. Collision detection and response is also a difficult topic related to time integration. A lot of detection methods have been proposed [TKH*05] and response strategies have been discussed [BMF03].

3. Deformable Bounding Voxels

Contrary to the traditional finite elements, in our approach, the volume mesh does not fit exactly the object, and all the nodes are not exactly under the surface, like in [CGC*02b, CGC*02a, MG04]. Some elements include the surface. Surface points are linearly interpolated within the cells. In the following, this cells are called *bounding elements*.
The first step consists in building the mesh and computing the mechanical properties. Then the mechanical simulation can be performed. The global algorithm is presented in Algorithm 1.

Algorithm 1 PREPROCESSING

in: (surface, Young modulus, Poisson ratio, max depth, mechanical depth)
out: (elements,masses,stiffnesses,interpolation weights)
Build the octree at maximal resolution
Detect boundary/outside/inside cells
Compute the mass and stiffness of the leaves
Pop up the mass and stiffness at desired mechanical level

3.1. Building the Voxels

The first step consists of a voxelization of the object. Starting from the bounding box of the object, an octree decomposition is employed up to a given fine maximal resolution (see Figure 1). Only non-empty cells are considered. To detect the outside/inside/boundary cells, the bounding box is inflated a little in order to ensure that its vertices and edges are outside the object (in our implementation, an arbitrary inflation of 1% is used). All the boundary cells contain surface points or intersect surface polygons. All the non-boundary cells on the borders are outside. This outside state is propagated to all their neighbours until a boundary cell is met. All others cells are inside. This algorithm (illustrated in Figure 2) works well for volumes without holes inside, but the animation can be nevertheless interesting for them. It could be easily extended to more complex geometries, and generalized to non-hexahedral cells, like in [CGC*02b]. Note that

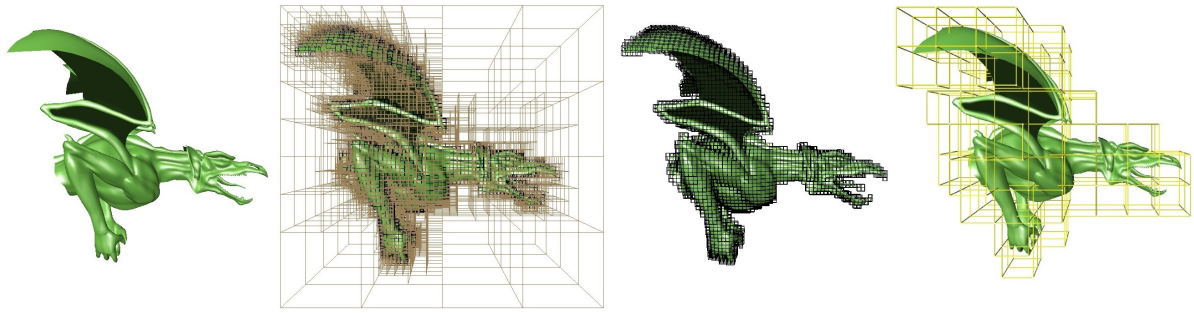


Figure 1: *The original shape. The entire maximal depth octree. All finest non-empty cells used to precompute mechanical properties. The animated mechanical depth.*

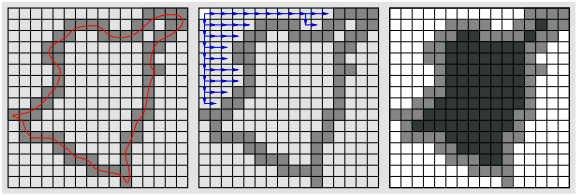


Figure 2: *Algorithm to detect outside/inside/boundary cells. First step consists of detecting boundary cell (grey)* i.e., *containing surface points and/or intersecting surface triangles (red). Then, outside border cells (white) are propagated to their neighborhood (blue arrows). Remaining cells are inside (black).*

it also automatically handles open surfaces, seen as object boundaries and animated like non-full cells.

3.2. Embedding the Shape

In order to link the surface with bounding elements, surfaces vertices $\mathbf{u}(p)$ are trilinearly interpolated using the eight values $\mathbf{u}(q)$ defined at the nodes of the bounding element as illustrated in 2D in Figure 3:

$$\mathbf{u}(p) = \mathbf{H}\mathbf{u}(q) \tag{1}$$

where the (3×24) interpolation matrix $\mathbf{H}$ of the considering element is the concatenation of the influences of the element vertices on a given vertex:

$$\mathbf{H}_i = \begin{bmatrix} h_i & 0 & 0 \\ 0 & h_i & 0 \\ 0 & 0 & h_i \end{bmatrix}, \; h_i(r,s,t) = \frac{1}{8}(1 \pm r)(1 \pm s)(1 \pm t)$$

To simplify the computations, the interpolations are done in natural coordinates in a local frame $(r,s,t) \in [-1,1]^3$. The Jacobian operator $\mathbf{J}$ relating the world coordinates to the local coordinate is needed for consistency: $\frac{\partial h}{\partial r} = \mathbf{J}\frac{\partial h}{\partial x}$.

In regular elements,

$$\mathbf{J} = \begin{bmatrix} \frac{2}{length} & 0 & 0 \\ 0 & \frac{2}{width} & 0 \\ 0 & 0 & \frac{2}{height} \end{bmatrix}$$

Large triangles remain flat even if they belong to several deformable voxels, so they have to be splitted where they cross voxel faces.

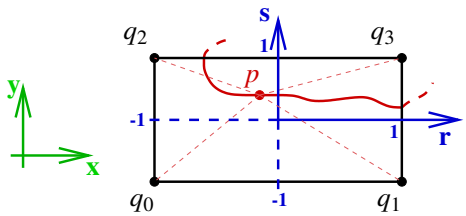


Figure 3: *Interpolation of a surface vertex p inside a boundary element.*

4. Mechanical Animation

When the collection of cells is built, mechanical laws can be applied to bounding elements in order to animate cells and the interpolated shape.

4.1. Force Computation

The linear Finite Element Method on hexahedral elements is used to compute internal forces [Bat82]. In this paper, we consider only isotropic linear elastic materials. According to Hooke's law, the material properties are only defined by the Young's Modulus and the Poisson's ratio. As explained in [MDM*02], a great advantage of the linear formulation is that all stiffness matrices can be precomputed, because they do not evolve too much during the animation.

Since the standard linear approach is inaccurate for large rotation of the elements, stiffness warping has to be employed

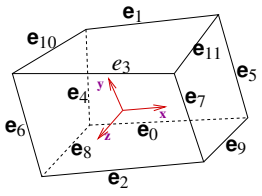
to avoid artificial inflating. The main idea is to compute the forces in an element's local rotated frame by decomposing the displacement into a rigid rotation combined with a deformation. The force applied by a deformed element to its sampling points is given then by

$$\mathbf{f} = \mathbf{R}^T \mathbf{K}(\mathbf{R}\mathbf{x} - \mathbf{x}^0)$$

where $\mathbf{K}$ is the stiffness matrix, $\mathbf{x}$ and $\mathbf{x}^0$ are the current and the initial positions, and matrix $\mathbf{R}$, which encodes the rotation of a local frame with respect to its initial orientation, is updated at each frame.

Since the first approach proposed [MDM*02], several methods have been proposed to compute $\mathbf{R}$. Methods using eigenvectors and eigenvalues [EKS03, MG04, ITF04] give the smallest deformations for most accurate results. A significantly faster method has been proposed in [NPF05]. However, the latter introduces some vertex ordering-dependent anisotropy and the evaluated strain is a slightly higher. However, in our case, a perfect application of the laws of physics is not necessary that is why we prefer the speed of this approach than the accuracy. An another important point of this approach is its robustness in degenerate configurations such as flat or inverted elements. An elegant treatment has been presented in [ITF04] but it is not aimed at real-time applications. Using [NPF05], the inversion of an element is detected for free, and modeled as a high compression. Stability is maintained and the elements can recover their initial shape without visible artifact. This is the essential and the desired effect in the case of non-physical situations, which can occur in real applications.

We extend to hexahedra the method [NPF05] initially designed for tetrahedra. For each hexahedron, three arbitrary edges could be selected in order to extract the rotation the same way as done for a tetrahedra. However, it is preferable to involve all the vertices in the computation to obtain a rotation that results in smaller measured deformations. To do this, we compute the average of four edges in the three directions, as illustrated Figure 4. Extreme element twisting can result unrealistic stable configurations, however this is unlikely to occur when several elements are used, and it does not impact stability.



$$\begin{aligned} \mathbf{x} &= \tfrac{1}{4}(\mathbf{e}_0 + \mathbf{e}_1 + \mathbf{e}_2 + \mathbf{e}_3) \\ \mathbf{y} &= \tfrac{1}{4}(\mathbf{e}_4 + \mathbf{e}_5 + \mathbf{e}_6 + \mathbf{e}_7) \\ \mathbf{z} &= \tfrac{1}{4}(\mathbf{e}_8 + \mathbf{e}_9 + \mathbf{e}_{10} + \mathbf{e}_{11}) \end{aligned}$$

Figure 4: *Deformed local frame of an hexahedron.*

4.2. Time integration

The stable Euler implicit solver is employed to solve the ODE. The filtered conjugate gradient presented in [BW98] gives the advantage to not necessary need an assembling stiffness matrix, since it only requires a product matrix/vector. It is well adapted to the stiffness warping formulation for which the assembling matrix should vary at each time step because of rotations. It is possible to process the elements one after another. Implicit integration is very well-suited to our approach, where all elements are regular, with the same size. The corresponding equation system is well-conditioned and can be solved iteratively. As usual, a large number of iterations can be necessary to accurately model materials with high stiffness. However, a reduced number of iterations (near 5 to 10) is generally acceptable. This allows trade-offs between accuracy and computation speed.

4.3. Boundary Conditions

Applying classic boundary conditions to the cell nodes is trivial but is not always sufficient. To directly manipulate the surface, or to accurately handle contact constraints, the boundary conditions must be applied to the surface, and dispatched over the lattice vertices.

Penalty forces

In case of penalty forces, forces $\mathbf{f}(p)$ are applied to points of the surface. We dispatch these forces over the lattice vertices. The principle of virtual work implies that

$$\mathbf{f}(q) = \mathbf{H}^T \mathbf{f}(p)$$

where H is the interpolation matrix at point p.

Hard constraints

When a displacement is imposed at the surface, the corresponding lattice displacements have to be computed. We compute the smallest displacements of the control points in least-squares sense, as in direct manipulation of FFD [HHK92].
A displacement constraint is written $\mathbf{u}(p) = \mathbf{c} \Leftrightarrow \mathbf{H}\mathbf{u}(q) = \mathbf{c}$ where $\mathbf{c}$ the constraint value. When several constraints are applied, we build a system containing all constrained surface points $\mathbf{p}$ and all influenced control points $\mathbf{q}$. The resulting system can be solved using a pseudo-inverse of the assembled matrix $\mathbf{H}$:

$$\mathbf{u}(\mathbf{q}) = \mathbf{H}^+ \mathbf{c}$$

$$\text{with} \begin{cases} \mathbf{H}^+ = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T & , dim(\mathbf{p}) \leq dim(\mathbf{q}) \\ \mathbf{H}^+ = \mathbf{H}^T(\mathbf{H}\mathbf{H}^T)^{-1} & , dim(\mathbf{p}) > dim(\mathbf{q}) \end{cases}.$$

When there are more constraints than control points, the best compromise (in the least-square sense) is computed. Note that in the specific case of at most one constraint applied to each lattice vertex, $\mathbf{H}_i^T \mathbf{H}_i$ is a scalar, and the solution of the system is efficient.

5. Accurate Physical Parameters

To adapt the stiffness of a cell according to its content, it is necessary to refine the octree mesh more precisely than desired for the animation. The main idea is to pop up the information from fine cells to coarser cells.

5.1. Weighted Stiffness

In order to improve an approximation of the stiffness of an element, the amount of matter contained in a cell is taken into account, as a material stiffness factor:

$$\mathbf{D}_c = \alpha_c \mathbf{D}$$

α_c is the filling ratio of the mechanical cell c and D is the stress-strain matrix relating the material properties. The finest depth is considered close enough to the surface to chose a filling rate equal to 50% for boundary cells. Inside cells have a filling ratio equal to 100% and outside cells to 0%. Using the octree hierarchy, these ratios are averaged until the desired mechanical level, such as $\alpha_{parent} = \frac{1}{8}\sum_{i=0}^{7} \alpha_{child_i}$. Thus, almost empty cells are softer than full cells. Figure 5 shows the poping up of filling ratios, from the finest depth to the mechanical level.

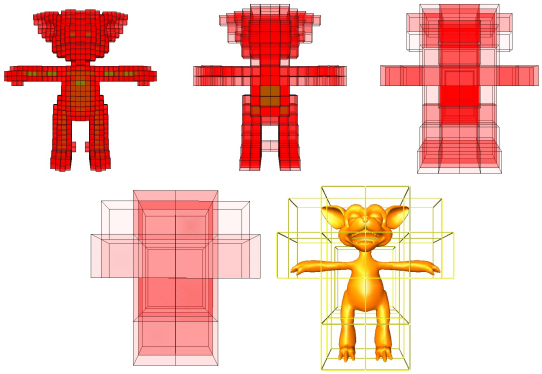


Figure 5: *Red cells are boundary and green are inside. Filling rates are shown in red transparency. From the maximal depth to the mechanical resolution*

5.2. Weighted Masses

The given total mass of the object m_{total} is distributed to each cell. To do this, the masses of the finest leaves m_{leaf} are computed by subdividing the total mass by the number of leaves, taking into account the filling ratio.

$$m_{leaf} = \frac{m_{total}}{2 \times \#insideLeaves + \#boundaryLeaves}$$

The factor 2 comes from the filling ratio, *i.e.,* an internal cell is twice heavier, so $m_{internal} = 2 \times m_{leaf}$ and $m_{boundary} = m_{leaf}$. The masses are then summed up from the finest level to the mechanical level.

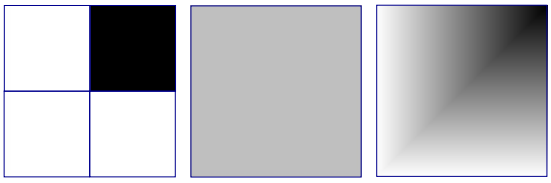


Figure 6: *2D stiffness in level of gray. From left to right: a) four original cells, three empty, one full. b) the corresponding cell with uniform weighted behavior presented in section 5.1. c) the corresponding non-uniform cell presented in section 5.3, stiffness varies along the axes*

5.3. Precomputed Non-Uniform Stiffness

Let us consider the child cells as interpolations of their parent cell. In this context, it is possible to deduce the stiffness influence of a child cell $\mathbf{K}_{child}$ on its parent $\mathbf{K}_{parent}$, and to deduce the stiffness of each parent cell based on its eight children. In this section, we take into account the distribution of the material over the child cells. Differences with the uniform stiffness are illustrated in Figure 6.

If only the large cells are considered, that returns to remove degrees of freedom to child nodes. To some extent child nodes are dependent from their parents and can be deduced as an interpolation (child nodes are constrained to stay "in the middle"). In this case, we can define eight matrices $\mathbf{L}_{child}$ which represent the interpolated child nodes $\mathbf{u}_{child}$ based on their parent cell nodes $\mathbf{u}_{parent}$: $\mathbf{u}_{child} = \mathbf{L}_{child}\mathbf{u}_{parent}$. Reciprocally, forces applied to child nodes can be popped up to the parent nodes using the transpose of the interpolation: $\mathbf{f}_{parent} = \mathbf{L}^T_{child}\mathbf{f}_{child}$. However $\mathbf{f}_{child} = \mathbf{K}_{child}\mathbf{u}_{child}$, so $\mathbf{f}_{parent} = \mathbf{L}^T_{child}\mathbf{K}_{child}\mathbf{L}_{child}\mathbf{u}_{parent}$. Summing the influence of the eight children of a parent cell, we obtain:

$$\mathbf{K}_{parent} = \sum_{i=0}^{7} \mathbf{L}_i^T \mathbf{K}_i \mathbf{L}_i$$

We do the same for the masses: $\mathbf{M}_{parent} = \sum_{i=0}^{7} \mathbf{L}_i^T \mathbf{M}_i \mathbf{L}_i$, where $\mathbf{M}$ is the mass matrix of a cell. At the mechanical level, we finally lump the mass matrix to obtain a diagonal matrix, which allows faster matrix products and easy weight computation.

Matrices $\mathbf{L}_{child}$ only depend on the shape of the elements, so they can be defined once for all. Matrices for hexahedral subdivision are given in Appendix A.

6. Results

6.1. Non-Uniform Stiffness

Taking into account the contents of bounding elements improves animation quality without adding complexity. This allows us to perform plausible animations using a reduced number of elements. An extreme case is presented

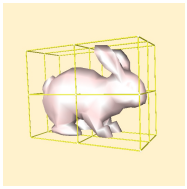
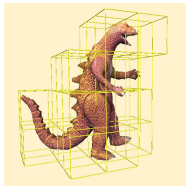
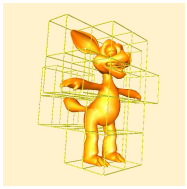
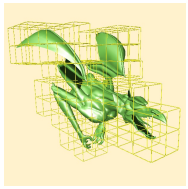
				
model	bunny	dino	fox	dragon
# points	**453**	**2151**	**8564**	**21507**
# triangles	901	4299	16777	42391
# elements	**8**	**33**	**30**	**125**
# particles	27	87	79	292
resolution max	5	5	5	6
animated resolution	1	2	2	3
preprocess time (s)	1.5	2	3	7
animation time (ms)	1	2	3	7
update time (ms)	1	2	7	9
FPS	**270**	**60**	**18**	**7**

Table 1: *Results.*

in Figure 7 where an object in form of 'c' is animated using one single boundary element. As expected, using a classical uniform stiffness, both parts of the object have the same properties, and the empty part is as stiff as the full part. Another example is presented Figure 8 for a more complex object. In contrast, using our precomputed non-uniform law, stiffness takes into account where the matter is, resulting in more realistic behaviors.

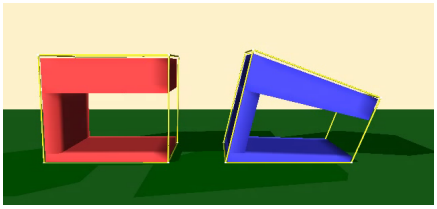


Figure 7: *Two similar objects are simulated by one single cell and subject to gravity. The left one is simulated using a basic uniform law and the right one, using the precomputed non-uniform law.*

Figure 8: *Two similar bunnies with the same mechanical mesh (left) are subject to gravity. The middle one is animated using a uniform stiffness, while the right one uses a non-uniform stiffness. Note the difference of ears behavior.*

6.2. Performance

Table 1 gives results for several simulations that were run on a laptop Pentium M 2 GHz with 2 GB of RAM and a nVidia Quadro FX Go1400. Performances depend linearly on the numbers of surface points and the number of bounding elements since number of iterations is fixed. It is a great advantage, since it is possible to keep a fast frame rate even when a high detailed mesh is animated, by reducing the number of cells.

In our implementation, a large time is used to update surface vertices. It is possible to accelerate this by performing the interpolations on the GPU using a vertex shader. Preliminary results show that a very simple shader that only compute final interpolations for rendering, without reading back the results for collision management, accelerates the dragon animation by a factor of two.

6.3. Robustness

Thanks to the used Finite Element Method, our simulator is robust face to degenerated configuration (Figure 9).

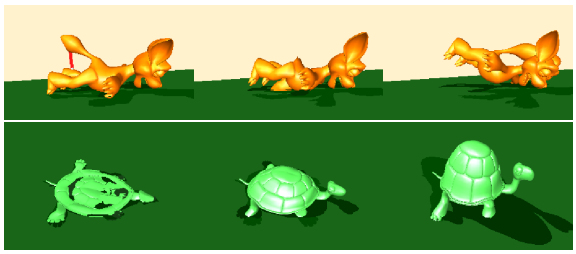


Figure 9: *Robustness face to degenerated configurations (top: element inversions, bottom: flat elements recover their rest form by increasing their stiffness).*

6.4. Surfaces and Lines

An important contribution of our approach is to handle surfaces, like the dragon's wings and T-shirt (Figure 10), as well as lines, the same way as volume. A nice feature of our method applied to this kind of objects is that their rest shape can be straightforwardly chosen as the most familiar form. For example, the T-shirt tends to recover the shape of a worn cloth.

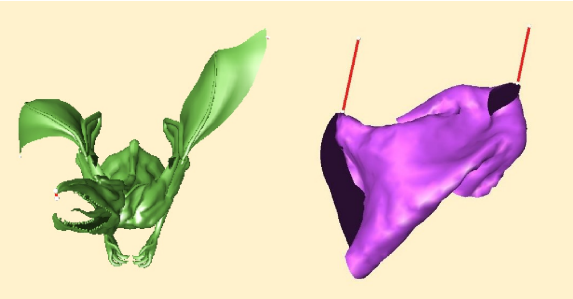


Figure 10: *Animate surfaces like volumes.*

7. Conclusion

Our approach disconnects mechanical complexity from geometrical detail. The mechanical resolution is independent from the rendering model, allowing to adjust the mechanical precision in order to interactively animate very detailed objects. Additional data computed in a preprocessing stage (grid points, interpolation weights and stiffness matrices) allows to apply fast and robust finite element dynamics. It does not add complexity compared with traditional FEM.

Moreover, we have presented novel hexahedral elements with stiffness warping and robust to inversion.

A nice feature of our method is its ability to animate objects including both volumes (closed surfaces) and surfaces, since voxels can be built automatically inside the object and around the surfaces.

Our approach is adaptable to more classic finite element meshes, like tetrahedra. It only needs to build meshes at different scales and the interpolation schemes to convert from one level to the other, which is available in all subdivision schemes.

Currently, a linear interpolation of the surface inside cell is performed, which can introduce discontinuities of normals in case of large deformations. More continuous interpolation would be an improvement, for example by using Berstein polynomials. Moreover, in order to improve computational speed, interpolation could be performed by the GPU using a vertex shader.

All our examples are limited to a single mechanical level, but the precomputed non-uniform stiffness idea could be extended to meshes with non-uniform resolution, such as deformable octrees.

References

[Bat82] BATHE K.: *Finite Element Procedures in Engineering Analysis*. Prentice-Hall, 1982. 17, 19

[BHW94] BREEN D. E., HOUSE D. H., WOZNY M. J.: Predicting the drape of woven cloth using interacting particles. In *Proc SIGGRAPH'94* (New York, NY, USA, 1994), ACM Press, pp. 365–372. 18

[BJ05] BARBIČ J., JAMES D. L.: Real-time subspace integration for st.venant-kirchhoff deformable models. *ACM Transactions on Graphics (SIGGRAPH 2005) 24*, 3 (Aug. 2005). 18

[BMF03] BRIDSON R., MARINO S., FEDKIW R.: Simulation of clothing with folds and wrinkles. In *SCA* (2003). 18

[BW98] BARAFF D., WITKIN A.: Large steps in cloth simulation. In *SIGGRAPH '98* (1998). 18, 20

[CEO*93] COVER S., EZQUERRA N., O'BRIEN J., ROWE R., GADACZ T., PALM E.: Interactively deformable models for surgery simulation. *IEEE Comput. Graph. Appl. 13*, 6 (1993), 68–75. 18

[CGC*02a] CAPELL S., GREEN S., CURLESS B., DUCHAMP T., POPOVIĆ Z.: Interactive skeleton-driven dynamic deformations. In *Proc SIGGRAPH'02* (2002). 18

[CGC*02b] CAPELL S., GREEN S., CURLESS B., DUCHAMP T., POPOVIĆ Z.: A multiresolution framework for dynamic deformations. In *SCA '02* (2002). 18

[DC95] DESBRUN M., CANI M.-P.: Animating soft substances with implicit surfaces. In *Proc SIGGRAPH'95* (1995), pp. 287–290. 18

[DDCB01] DEBUNNE G., DESBRUN M., CANI M.-P., BARR A. H.: Dynamic real-time deformations using space and time adaptive sampling. In *SIGGRAPH '01* (2001). 18

[DMG05] DEQUIDT J., MARCHAL D., GRISONI L.: Time critical animation of deformable solids. *Journal of Computer Animation and Virtual Worlds 16* (2005), 177–187. 18

[EK03] ETZMUSS O., KECKEISEN M.: *A Linearised Finite Element Model for Cloth Animation*. Technical Report WSI-2003-2, Universität Tübingen, 2003. 18

[EKS03] ETZMUSS O., KECKEISEN M., STRASSER W.: A Fast Finite Element Solution for Cloth Modelling. *Proc Pacific Graphics* (2003). 20

[Hau04] HAUTH M.: *Visual Simulation of Deformable Models*. Phd thesis, Wilhelm-Schickard-Institut für Informatik, University of Tübingen, Germany, July 2004. 18

[HHK92] HSU W. M., HUGHES J. F., KAUFMAN H.: Direct manipulation of free-form deformations. In *Proc of SIGGRAPH '92* (New York, NY, USA, 1992), ACM Press, pp. 177–184. 20

[ITF04] IRVING G., TERAN J., FEDKIW R.: Invertible finite elements for robust simulation of large deformation. In *Proc SCA* (2004). 18, 20

[JBT04] JAMES D. L., BARBIČ J., TWIGG C. D.: Squashing cubes: Automating deformable model construction for graphics. In *Proc SIGGRAPH'04 Conference on Sketches & Applications* (2004). 18

[MDM*02] MÜLLER M., DORSEY J., MCMILLAN L., JAGNOW R., CUTLER B.: Stable real-time deformations. In *Proc SCA* (2002). 18, 19, 20

[MG04] MÜLLER M., GROSS M.: Interactive virtual materials. In *Proc Graphics Interface* (2004). 18, 20

[MHTG05] MÜLLER M., HEIDELBERGER B., TESCHNER M., GROSS M.: Meshless deformations based on shape matching. In *Proc SIGGRAPH'05* (july 2005), pp. 471–478. 18

[MKN*04] MÜLLER M., KEISER R., NEALEN A., PAULY M., GROSS M., ALEXA M.: Point based animation of elastic, plastic and melting objects. In *Proc SCA* (2004), pp. 141–151. 18

[MTG04] MÜLLER M., TESCHNER M., GROSS M.: Physically based simulation of objects represented by surface meshes. In *Proc SCA* (2004). 18

[NFP06] NESME M., FAURE F., PAYAN Y.: Hierarchical multi-resolution finite element model for soft body simulation. In *Symposium on Biomedical Simulation* (2006). 18

[NPF05] NESME M., PAYAN Y., FAURE F.: Efficient, physically plausible finite elements. In *Eurographics (short papers)* (august 2005), pp. 77–80. 18, 20

[PKA*05] PAULY M., KEISER R., ADAMS B., DUTRÉ; P., GROSS M., GUIBAS L. J.: Meshless animation of fracturing solids. *ACM Trans. Graph. 24*, 3 (2005), 957–964. 18

[PTVF92] PRESS, TEUKOLSKI, VETTERLING, FLANNERY: *Numerical Recipes in C*. Cambridge University Press, 1992. 18

[TKH*05] TESCHNER M., KIMMERLE S., HEIDELBERGER B., ZACHMANN G., RAGHUPATHI L., FUHRMANN A., CANI M.-P., FAURE F., MAGNETAT-THALMANN N., STRASSER W., VOLINO P.: Collision detection for deformable objects. *Computer Graphics Forum 24*, 1 (March 2005), 61–81. 18

[TPBF87] TERZOPOULOS D., PLATT J., BARR A., FLEISCHER K.: Elastically deformable models. In *SIGGRAPH '87* (1987). 17

Appendix A: Interpolation Matrices for Hexahedral Subdivision

The following presents the eight interpolation matrices L_c that give values of child cells c nodes from their parent cell nodes, according with indices presented in figure 11. They are very easy to compute. If $a_{i\to c_j}$ is the influence of the i^{th} parent point on the j^{th} point of its c^{th} child, then

$$\mathbf{L}_c = \begin{bmatrix} a_{c_0 0} & \dots & a_{c_0 7} \\ & \dots & \\ a_{c_7 0} & \dots & a_{c_7 7} \end{bmatrix}$$

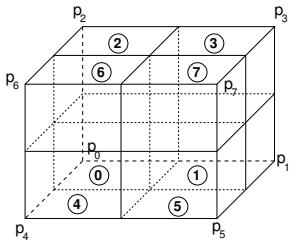


Figure 11: *Indices.*

$$\mathbf{L}_0 = \frac{1}{8}\begin{bmatrix} 8&0&0&0&0&0&0&0\\ 4&4&0&0&0&0&0&0\\ 4&0&4&0&0&0&0&0\\ 2&2&2&2&0&0&0&0\\ 4&0&0&0&4&0&0&0\\ 2&2&0&0&2&2&0&0\\ 2&0&2&0&2&0&2&0\\ 1&1&1&1&1&1&1&1 \end{bmatrix}$$

$$\mathbf{L}_1 = \frac{1}{8}\begin{bmatrix} 4&4&0&0&0&0&0&0\\ 0&8&0&0&0&0&0&0\\ 2&2&2&2&0&0&0&0\\ 0&4&0&4&0&0&0&0\\ 2&2&0&0&2&2&0&0\\ 0&4&0&0&0&4&0&0\\ 1&1&1&1&1&1&1&1\\ 0&2&0&2&0&2&0&2 \end{bmatrix}$$

$$\mathbf{L}_2 = \frac{1}{8}\begin{bmatrix} 4&0&4&0&0&0&0&0\\ 2&2&2&2&0&0&0&0\\ 0&0&8&0&0&0&0&0\\ 0&0&4&4&0&0&0&0\\ 2&0&2&0&2&0&2&0\\ 1&1&1&1&1&1&1&1\\ 0&0&4&0&0&0&4&0\\ 0&0&2&2&0&0&2&2 \end{bmatrix}$$

$$\mathbf{L}_3 = \frac{1}{8}\begin{bmatrix} 2&2&2&2&0&0&0&0\\ 0&4&0&4&0&0&0&0\\ 0&0&4&4&0&0&0&0\\ 0&0&0&8&0&0&0&0\\ 1&1&1&1&1&1&1&1\\ 0&2&0&2&0&2&0&2\\ 0&0&2&2&0&0&2&2\\ 0&0&0&4&0&0&0&4 \end{bmatrix}$$

$$\mathbf{L}_4 = \frac{1}{8}\begin{bmatrix} 4&0&0&0&4&0&0&0\\ 2&2&0&0&2&2&0&0\\ 2&0&2&0&2&0&2&0\\ 1&1&1&1&1&1&1&1\\ 0&0&0&0&8&0&0&0\\ 0&0&0&0&4&4&0&0\\ 0&0&0&0&4&0&4&0\\ 0&0&0&0&2&2&2&2 \end{bmatrix}$$

$$\mathbf{L}_5 = \frac{1}{8}\begin{bmatrix} 2&2&0&0&2&2&0&0\\ 0&4&0&0&0&4&0&0\\ 1&1&1&1&1&1&1&1\\ 0&2&0&2&0&2&0&2\\ 0&0&0&0&4&4&0&0\\ 0&0&0&0&0&8&0&0\\ 0&0&0&0&2&2&2&2\\ 0&0&0&0&0&4&0&4 \end{bmatrix}$$

$$\mathbf{L}_6 = \frac{1}{8}\begin{bmatrix} 2&0&2&0&2&0&2&0\\ 1&1&1&1&1&1&1&1\\ 0&0&4&0&0&0&4&0\\ 0&0&2&2&0&0&2&2\\ 0&0&0&0&4&0&4&0\\ 0&0&0&0&2&2&2&2\\ 0&0&0&0&0&0&8&0\\ 0&0&0&0&0&0&4&4 \end{bmatrix}$$

$$\mathbf{L}_7 = \frac{1}{8}\begin{bmatrix} 1&1&1&1&1&1&1&1\\ 0&2&0&2&0&2&0&2\\ 0&0&2&2&0&0&2&2\\ 0&0&0&4&0&0&0&4\\ 0&0&0&0&2&2&2&2\\ 0&0&0&0&0&4&0&4\\ 0&0&0&0&0&0&4&4\\ 0&0&0&0&0&0&0&8 \end{bmatrix}$$