# Dynamic Triangulation of Implicit Surfaces: towards the handling of topology changes

Matthieu Nesme, Antoine Bouthors

▶ **To cite this version:**

Matthieu Nesme, Antoine Bouthors. Dynamic Triangulation of Implicit Surfaces: towards the handling of topology changes. [Research Report] RR-6128, INRIA. 2006. inria-00132537v2

**HAL Id: inria-00132537**
**https://inria.hal.science/inria-00132537v2**

Submitted on 22 Feb 2007

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

# *Dynamic Triangulation of Implicit Surfaces: towards the handling of topology changes*

Matthieu Nesme — Antoine Bouthors

**N° 6128**

Novembre 2006

Thème COG

*Rapport de recherche*

ISSN 0249-6399 ISRN INRIA/RR--6128--FR+ENG

# Dynamic Triangulation of Implicit Surfaces: towards the handling of topology changes

Matthieu Nesme , Antoine Bouthors

Thème COG — Systèmes cognitifs

Projet EVASION

Rapport de recherche n° 6128 — Novembre 2006 — 24 pages

**Abstract:** In this paper, we introduce a new approach to mesh an animated implicit surface for rendering purposes. This approach is based on a double triangulation of the surface. In the first triangulation, the vertices are the nodes of a finite element model. The aim of this model is to uniformly and dynamically sample the surface. It is robust, efficient and prevents the inversion of triangles. The second triangulation is dynamically created from the first one at each frame and provides details in regions of high curvature. Lastly, we present a mechanism to deal with both the topological merging and splitting of surfaces. As shown in our results, our approach provides robust, quality, interactive rendering of animated implicit surfaces, including those with certain kinds of topology changes.

**Key-words:** implicit surfaces, real-time rendering, triangulation, interactive modeling, animation, particle system, finite element

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

# Triangulation dynamique de surfaces implicites : vers une gestion des changements de topologie

**Résumé :** Dans ce rapport, nous introduisons une nouvelle approche pour mailler une surface implicite animée dans le but d'en faire le rendu. Cette approche est basée sur une double triangulation de la surface. Dans la première, les sommets sont les nœuds d'un modèle éléments finis. Le but de ce modèle est d'échantillonner uniformément et dynamiquement la surface. Il est robuste, performant, et empêche l'inversion des triangles. La seconde triangulation est dynamiquement créée à partir de la première à chaque image et fournit des détails dans les régions de forte courbure. Enfin, nous présentons un mécanisme permettant de gérer la fusion et la séparation topologiques des surfaces. Comme le montrent nos résultats, notre approche permet un rendu de qualité, robuste et interactif de surfaces implicites animés, y compris celles comportant certains changements de topologie.

**Mots-clés :** surfaces implicites, rendu temps-réel, triangulation, modélisation interactive, animation, système de particules, éléments finis

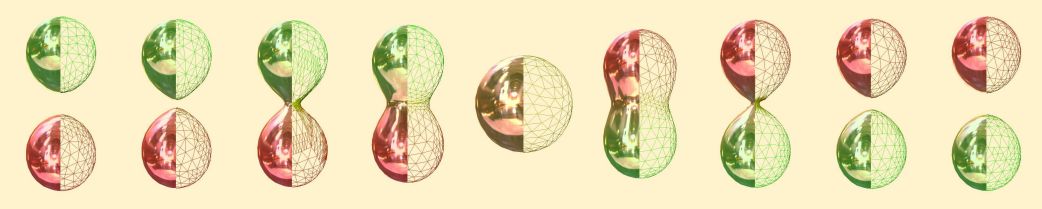


Figure 1: An animated surface defined by two skeleton elements crossing each other.

# 1 Introduction and Previous Work

Implicit surfaces are a powerful tool for modeling and animating deformable objects such as organic structures or fluids (for more details, see the study of [8]). The advantages of this representation lie in the fact that the surface is defined no explicitly, but implicitly, via a *field function* defined on the whole space. In the remainder of this paper, the implicit surface is defined as the zero level-set $\{\mathbf{x} \in \mathbb{R}^3, F(\mathbf{x}) = 0\}$ of a field function $F : \mathbb{R}^3 \rightarrow \mathbb{R}$, which is positive inside the objects.
Unfortunately, an interactive rendering of implicit surfaces is difficult to set up. Ray-tracing is well adapted for implicit surfaces rendering because it has no topology constraints, but it is extremely expensive. Some approaches have been proposed to make it interactive, such as ray-casting [2, 15]. Today's graphics hardware rely on polygons in order to achieve real-time frame rates. The rendering time of a triangulation of an implicit surface with such hardware depends almost only on the time spent to build the triangles. As a result, finding a fast triangulation method of a time-varying implicit surface for rendering purposes has become a challenging issue.

This work presents an approach that generates a manifold mesh describing a dynamic implicit surface. The latter is updated in real time to reflect the changes of the surface.

## 1.1 Triangulation Methods for Dynamic Implicit Surfaces

Methods for triangulating implicit surfaces can be divided into two families: tessellation techniques and particle-based techniques.

The most famous tessellation method is the marching cubes algorithm which has seen much evolution since [20, 5]. However, the main drawback of this kind of technique is that a new triangulation has to be rebuilt from scratch as soon as the surface moves and warps. An improvement has been proposed by [13] who recompute the triangulation only in modified areas. However, this is only relevant for static surfaces that undergo localized deformations.

Particles-based techniques have been introduced by [11] to sample implicit surfaces. Since then,

several methods have been presented. The main idea is to constrain self-repulsing or self-attracting particles on the surface and let them populate the surface. [27] extend this approach to permit animation and control of the implicit surface. In addition, particles are created or removed in regions where they are too sparse or too dense, respectively. [9] are the first authors to take into account the curvature of the surface in the particle system.

Starting from a particle-based method, [19] have proposed to display implicit surfaces using point rendering. The main drawback is the convergence time, which probably does not permit animated surfaces in real-time. Further more, the overlapping of surfels (surface elements) can be an obstacle for rendering textured or transparent surfaces.

Particle-based methods work well to sample and follow dynamic implicit surfaces, but obtaining a triangulation from a set of points is far from trivial and is time-consuming. To obtain a triangular mesh, [11] polygonize particles with a Delaunay triangulation at each step which is expensive. A faster triangulation method was developed in [10]. It takes into account neighboring information given by the particle system to accelerate computations.

More time can be saved by making the triangulation evolve with the particle system. [24] introduced the first method for dynamic triangulation of a skeleton-based implicit surface. The idea is to start from an initial triangulation, and then convert each vertex into a particle and each edge into a spring linking the two particles. As the particles evolve under the forces of the springs, the vertices of the triangulation are moved to follow the particles. In addition, triangles are inserted or removed accordingly when particles and springs are added or removed. The rest lengths of the springs are modified depending on the local curvature, which allows more triangles in high curvature areas. This makes the system far less stable and increases computations. When a surface splits, the triangles are cut to create several surfaces. However, this method does not handle other kinds of topology changes, like blending. Finally, some triangles can be inverted as the system is updated, leading to an invalid triangulation, and this method does not provide any means to recover from such an invalid state.

[25] introduced a method to guarantee the topology of a dynamic implicit surface by searching for and tracking *critical points* [21] in the field function and modifying the polygonization when necessary. However, a set of critical points is built during the preprocessing stage and if new critical points appear during animation of the surface, they are not detected, and the topology change is not handled. Therefore, this method supports any kind of topology change, but not in every configuration.

## 1.2 Our Contribution

The idea of binding the mesh geometry to a mechanical system that evolves as the surface is animated, such as [24], is good because it avoids rebuilding a new triangulation at each frame. However,

the criteria for an ideal geometry and an ideal mechanical system are quite different. In order for the mechanical system to be as stable as possible the triangles should all have the same size and the same shape (as close as possible to an equilateral triangle). In addition, changing the rest shape of the mechanical elements through time, such as adapting their sizes to curvature, makes the system unstable. That is why the rest state of the mechanical elements should stay constant. On the contrary, in order for the geometry to best represent the surface, it should be finer along directions of high curvature and coarser along directions of low curvature. Since the surface can be animated, this criteria yields triangles whose size and aspect ratio vary in both space and time.

This problem leads us to consider two separate representations of the surface. The first one, that we call the "mechanical mesh", is used to follow dynamically the animated surface. The second one, called "geometric mesh", is generated from the "mechanical mesh" and used for rendering purposes. The main criterion for the design of the mechanical mesh is robustness. It acts like an elastic "skin" that "slips" on the implicit surface. The elasticity forces tend to give all the triangles a common isotropic shape, and make them sample the surface uniformly. To solve this elasticity simulation, contrary to [24] who use a mass-spring system, we use a Finite Element Method (FEM), where all the elements share as much as possible the same size, aspect ratio, and stiffness. Each mechanical triangle corresponds to a 2D finite element, and all of its points are constrained to lie on the surface. Our mechanical system uses a quasi-static solving approach. The geometric mesh is generated by refining the mechanical mesh in areas of high curvature. As a result, each mechanical triangle can correspond to one or more geometric triangles.

In addition, we present a method dealing with two cases of topology change: splitting and blending. Topology changes are detected on the geometric mesh. When a topology change is detected, the mechanical mesh is modified through erasing, collapsing, and welding operations.

Our approach works on field function whose derivative is continuous and nonzero near the surface, defined by a skeleton or by discrete values stored on a grid.

In section 2, we describe our mechanical system. In section 3, we explain how we build the geometric mesh based on the mechanical one. In section 4, the management of topology changes is presented. We discuss results in section 5.

## 2 Mechanical System

In this section, the mesh, triangles, vertices, edges, and particles terms are all related to the mechanical mesh.

As in [24], we start with an initial mesh that can be constructed by any method. We use a marching cube algorithm [20], but more sophisticated methods like [1] or [16] can be used. Each vertex of this initial mesh becomes a particle on which forces are applied. By obeying the mechanical laws presented in the following, the final mesh tends to be well formed as illustrated in figure 2, and evolves following the movements and deformations of the surface.

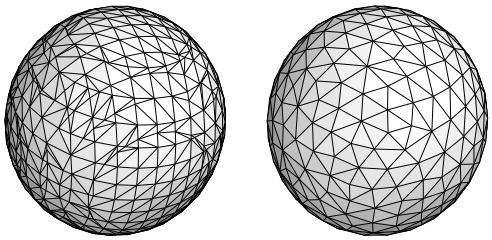


Figure 2: *Left:* the initial state of the mechanical model, corresponding to the result of a marching cubes algorithm. *Right:* at equilibrium, the mechanical model has evolved to a regular mesh.

## 2.1 Quasi Static Time Integration

Due to the application, we are not interested in animating the dynamic evolution of the particle system, but only in a succession of equilibrium states. So we prefer to use a quasi static solving approach rather than a dynamic resolution. Since each step is independent and previous errors are not accumulated, this integration is fast and stable. At each frame $t$, we search for the ideal positions $\mathbf{x}_t$ of the particles. For a time step length $dt$ the new positions are given by $\mathbf{x}_{t+dt} = \mathbf{x}_t + \triangle\mathbf{x}_t$. The ideal positions can be found by solving the non-linear system $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ respecting the force equilibrium. To find a solution, we modify the implicit solver [4] without taking into account mass and velocities resulting in:

$$\mathbf{I} - dt\frac{\partial \mathbf{f}}{\partial \mathbf{x}} \triangle \mathbf{x} = dt\mathbf{f}(\mathbf{x}) \tag{1}$$

## 2.2 Surface Constraint

In order to keep the particles on the implicit surface, [27, 24] project the velocity of each particle along the surface gradient at each time step. With this method, particles are always moving following the surface, but at the end of a step, their positions are not guaranteed to be exactly on the surface. This constraint is applied using Lagrange multipliers, augmenting the linear equations system (equation (1)). As explained in [4], this can degrade the conditioning of the system and can bring numerical instabilities ; moreover, they point out that the system is no longer guaranteed as positive definite, which is a problem in case of iterative methods such as the conjugate gradient we use. For these reasons, we prefer to use a filtered conjugate gradient in the same way as [4]. The idea is to project initial residual and successive iterative solutions along the surface tangent plane during the conjugate gradient resolution.

We correct the particle positions directly by projecting them to the surface after the conjugate gradient resolution. The projection of a point $\mathbf{p}$ on the implicit surface corresponds to solving $f_{\mathbf{p}}(x) = 0$, $f_{\mathbf{p}} : \mathbb{R} \to \mathbb{R}, x \to F(\mathbf{p} + x\dot{F}(\mathbf{p}))$. This new position is computed using an iterative optimization based

on the Newton-Raphson method. At each iteration $k$, the minimized point $\mathbf{p}$ moves along the surface gradient $\dot{F}(\mathbf{p})$:

$$\mathbf{p}_{k+1} = \mathbf{p}_k - F(\mathbf{p}_k)\frac{\dot{F}(\mathbf{p}_k)}{\left\|\dot{F}(\mathbf{p}_k)\right\|^2} \tag{2}$$

This method is very efficient and works well for any field functions whose derivative is continuous and nonzero in the neighborhood of the solution. In these cases, very few steps are necessary to find a good projection. The maximum number of iterations is fixed for efficiency. In certain cases (*e.g.*, when there is no solution), even though the point is close to the surface, this Newton-Raphson projection method does not converge (see figure 3). We call these points "quasi-critical" points, because in our context they are intuitively near critical points. Such points are not projected and stay at their initial position. We record the point it happened to and from which triangle it was issued. As we will see in section 4.2, this record is used for tracking topological splitting events.

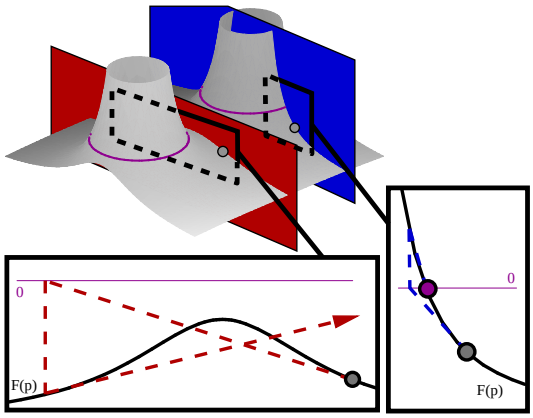



Figure 3: Two examples of the projection method in 2D. The grey surface represents the field function. The purple circles on the surface represent the isosurface. For two points (grey dots), the Newton-Raphson projection method (eqn. 2) is applied. On the blue example (bottom, right), a solution is found. On the red example (bottom, left), the method diverges, no solution is found. Such a non-projectable point is called "quasi-critical".

## 2.3 A FEM-based Approach

Several methods for modeling forces and force derivatives exist. The two most popular in computer graphics are the mass-spring system and the finite element method. Mass-spring systems are very fast but have many known drawbacks. Particularly, they cannot precisely represent triangles because resulting forces do not take into account all the surface. Because in our case, a perfect application of the laws of physics is not necessary, the efficient FEM-based method presented in [23] has been used, resulting in equation 3. In order to speed up computations significantly, stiffness matrices can

$$\mathbf{R}_j^T\mathbf{x}-\mathbf{x}_0 = \begin{bmatrix} 0 \\ 0 \\ v^x - v_0^x \\ 0 \\ w^x - w_0^x \\ -w^y - w_0^y \end{bmatrix}$$

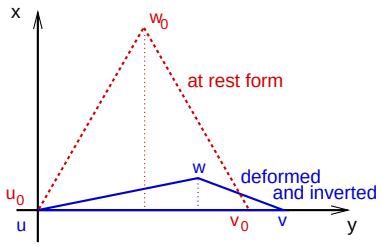



Figure 4: An element known as inverted and its rest form in their local frame

be precomputed, as explain in [22]. [23] have shown that this precomputation can introduce "ghost torques". However, in our approach based on a quasi static integration, the system does not conserve the velocities of the particles, so the effect of ghost torques are limited. In all ways, we do not require perfect mechanical simulation.

$$\mathbf{f}_j(\mathbf{x}) = \mathbf{R}_j\mathbf{K}_j(\mathbf{R}_j^{\mathrm{T}}\mathbf{x}-\mathbf{x}_0) \tag{3}$$

where $\mathbf{K}_j$ is the stiffness matrix of the element, $\mathbf{x}$ and $\mathbf{x}_0$ the current and the initial positions of the sampling points, respectively. $(\mathbf{x}-\mathbf{x}_0)$ is called the displacement. Matrix $\mathbf{R}_j$, which encodes the rotation of a local frame attached to the element with respect to its initial orientation, is updated at each frame.

**Recovering from Inversions** Forces of inverted elements are modified so that they fall back to an uninverted state, mixing the elegant approach of [17] and the fast approach of [23]. The local frame presented in [23] is chosen such that its third vertex (assimilated as the inverted one) has the smallest height as in [17] (see figure 4). The displacements of the inverted element $j$ then become:

where $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$ are the coordinates of the first, second and third vertex of the element $j$ in the local frame given by the rotation $\mathbf{R}_j$, respectively. With these displacements, forces are computed classically as in formula (3), so they do not require more computations than in case of a non-inverted element. Only the inversion detection, presented in the section 2.5, needs one more evaluation of the vertex gradients.

## 2.4 Well Shaped Mechanical Mesh

We enforce the aspect ratio of all the geometry. In consequence, we set a fixed size, shape and material for all the mechanical elements. As a result, we obtain only one initial shape $\mathbf{x_0}$ and one stiffness matrix $\mathbf{K}$ for all the elements, which can be precomputed. So for an element $j$, forces applied to its vertices are $\mathbf{f}_j(\mathbf{x}) = \mathbf{R}_j\mathbf{K}(\mathbf{R}_j^{\mathrm{T}}\mathbf{x}-\mathbf{x}_0)$, where $\mathbf{K}$ and $\mathbf{x}_0$ are the same for all elements. The common rest shape is computed as an equilateral triangle with edge lengths $\hat{e}$ given by the user.

When the surface is deformed, stretching of elements can pass over a threshold $e_{max}$, so new triangles must appear, and some triangles must disappear when elements are too compressed (threshold $e_{min}$). The mesh is modified using edge collapse and edge subdivision, as shown in figure 5. Criteria for mesh changes are based on geometrical information (that is, edge lengths $\| e \|$), because this information is faster to compute than information based on mechanics (such as deformation directions). For this, we loop on edges and check their length $\| e \|$, as explained in figure 5. As in [24], we collapse or subdivide an edge if $\| e \| > e_{max} \times \hat{e}$ *and* $\| e_t \| - \| e_{t-1} \| > \varepsilon_e \times \hat{e}$ or $\| e \| < e_{min} \times \hat{e}$ *and* $\| e_t \| - \| e_{t-1} \| < \varepsilon_e \times \hat{e}$, respectively. At each collapse or subdivision, all modified edges (marked with circles in figure 5) are flagged as unmodifiable for the next time step. This avoids oscillation during the search of equilibrium state with the new created mesh. In certain cases, collapsing can generate a non-manifold mesh [12], where an edge is shared by more than two triangles (see the example in figure 6). In these cases, collapsing is forbidden, to ensure the mesh is always manifold. This mesh alteration, which loops on edges and subdivides or collapses them if necessary, is called before the quasi-static integration step. It is referred in the rest of the paper as the ALTERMESH procedure.

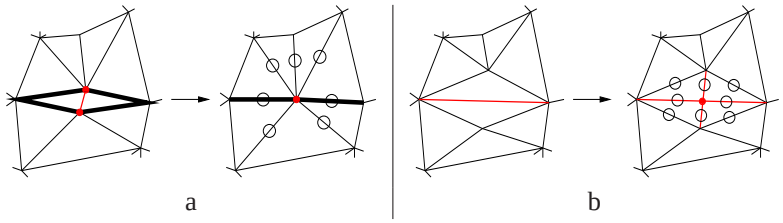



Figure 5: *(a)* Collapse: when an edge is too compressed (in red) and has not stretched since the last time step, the two vertices at the ends of this edge are joined in its middle (red point). It results in the removal of two triangles (in bold) and the welding of two edges. *(b)* Subdivide: when an edge is too stretched (in red) and has not been compressed since the last time step, it is subdivided into two parts by inserting a new vertex in its center (red point). It results in the creation of two new triangles and two new edges (in red).

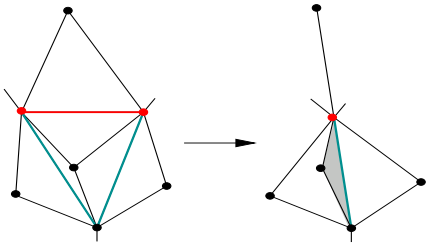


Figure 6: Example of collapsing creating a non-manifold mesh: the red edge is collapsed, and the two green edges become identical, yielding two superimposed triangles (grey). The new green edge has four incident triangles.

## 2.5 Inversion Detection and Preventing

### 2.5.1 Origin of Inversions

An inverted element is a triangle facing the inside of the object instead of the outside. We consider that a triangle $(\mathbf{a},\mathbf{b},\mathbf{c})$ of normal $\mathbf{n}$ is inverted if

$$\mathbf{n}\cdot\dot{F}(\mathbf{a})>0 \textit{ and } \mathbf{n}\cdot\dot{F}(\mathbf{b})>0 \textit{ and } \mathbf{n}\cdot\dot{F}(\mathbf{c})>0$$

We found several cases that yield such inverted triangles:

1. discrete integration can let points cross an edge and create a fold, as shown in figure 7*a*,
2. collapsing an edge may invert a neighboring triangle (see figure 7*b*),
3. the constraint may invert the triangles when projecting the points to the surface (see figure 7*c*),
4. subdividing an edge belonging to an inverted triangle inherently inserts new inverted elements,
5. user interaction or surface animation, as well as topology changes, may modify particles positions too fast for the mechanical system to follow them.

If the mechanical system allows such inversions as a rest state, as is the case with a mass-spring system, they will never be removed, and may even expand when the subdivision criterion is satisfied. A solution to this problem was very briefly introduced in [24], but was based on collapsing edges, which can create additional inverted elements. We address this problem by two means: trying to have as few inversions as possible, and remove them when they appear.

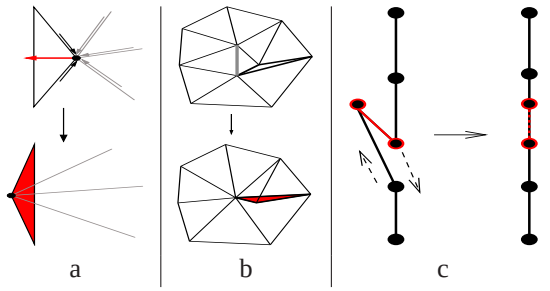



Figure 7: *(a)* Inversion resulting from a discrete integration. At time $t$, the gray forces are stronger than the black ones and push the point inside the triangle. At time $t+dt$, the point is pushed too far, resulting in an inversion. *(b)* Inversion resulting from collapsing the gray edge: the bold triangle becomes inverted (red). *(c)* Forces can create wrinkles that give inverted triangles when projected to the surface (red).

### 2.5.2 Preventing Inversions

Our FEM is better at avoiding inversions than a mass-spring system, since the forces increase much more when a triangle is compressed and approaches an inversion state (case 1, figure 7*a*). Moreover, we arrange so that elements are stretched most of the time, by simply setting the subdivide and collapse criteria $e_{max}$ and $e_{min}$ to 2 and 1, respectively. This means the "skin" is always stretched and never makes wrinkles that could lead to inversions (case 3, figure 7*c*). The result mimics in fact a particle system approach in which particles attract each other instead of repulsing each other. We avoid collapsing an edge if doing so creates an inversion (case 4), and we forbid subdivision of edges belonging to an inverted triangle (case 2, figure 7*b*). Despite this measures, case 5 can still create inversions which can be handled with method explain in section 2.3.

# 3 Building the Geometric Mesh

After each update of the mechanical system, we create an initial geometric mesh by copying the mechanical one. The recursive REFINE($\tau$) procedure is then applied to each geometric triangle $\tau$ to refine the geometry. Its recursive depth is limited to avoid infinite recursion in degenerate areas. It proceeds as follow. A triangle is refined if at least one of its edge needs to be refined. An edge needs to be refined if the angle between the normals of its two points is higher than a user-defined threshold value $\alpha_r$. For each edge that needs to be refined, we create a new point in its center and project its position onto the implicit surface using the Newton-Raphson technique described in section 2.2.

The case where two edges of the triangle need to be refined is special: in this case, we also refine the third edge in order to create a proper triangulation, but we do not project the new point, so that this edge matches that of the neighboring triangle (figure 8*a*). This particular point is called a "ghost" point. Doing so allows us to subdivide locally without having to consider the triangle neighbors.

Depending on how many edges should be refined, REFINE($\tau$) replaces $\tau$ by either two or four smaller triangles as shown in figure 8*a*. It then calls itself on these new triangles. Note that if $\alpha_r$ is set to the minimal value (that is $-\pi$), the geometric mesh is an exact copy of the mechanical mesh. Example results can be seen on figures 10 and 9.

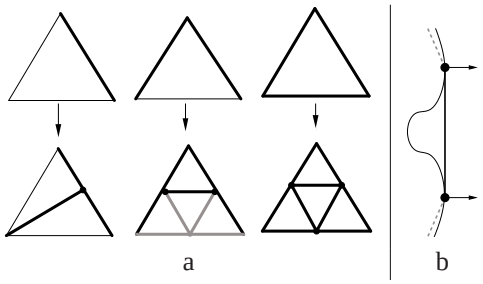



Figure 8: *(a)* Refinement scheme. Bold lines on the top show what edges need to be refined. Bold lines on the bottom show what edges were modified. Gray lines and gray points indicate that new triangles were inserted but the position of the added point is not projected onto the surface, so that it matches the neighboring triangle. *(b)* Limitation of using normals as refinement criterion: the bold edge will not refine to sample the gap.

This method has two main limitations:

- The refinement criterion may miss areas were refinement should happen, *e.g.,* figure 8*b*. If one does not want to miss these regions, either a finer mechanical resolution must be chosen or a different criterion must be used, such as one based on the Hessian of the field function.
- The refinement scheme inserts ghost points when two edges of a triangle need to be refined. These are T-vertices, that could be noticeable when zooming very close on the surface. If these T-vertices are undesirable, one should use another refinement scheme, such as red-green refinement [6, 3].

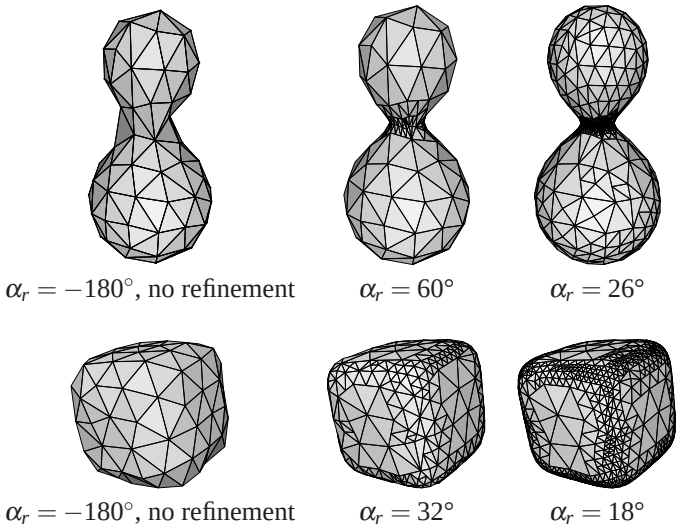



Figure 9: Results of geometry refinements with varying $\alpha_r$.

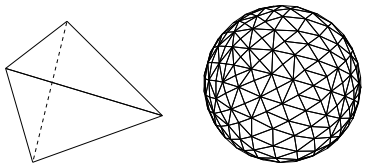


Figure 10: Geometric refinement on an extremely coarse mesh: although the metaball is mechanically represented by only four triangles (*left*), the refinement with $\alpha_r = 20°$ gives a well detailed mesh (*right)*.

# 4 Topology Changes

The upcoming section deals with the treatment of blending and splitting of surfaces. Both operations share the same principle: topology changes are detected on the geometric mesh. When such changes are detected, modifications are applied on the mechanical mesh to account for the new topology. Lastly, since the mechanical mesh has changed, the geometry is re-generated. Both operations use several common procedures, all performed on the mechanical mesh:

- The ERASE($\tau$) function simply removes triangle $\tau$ from the triangulation
- The FINDBORDERS procedure searches the triangulation for border edges, that is, edges that belong to only one triangle. Neighboring border edges are put together in a linked list. The

result is a set of such linked lists, each list representing one hole in the triangulation. These holes are the result of a prior call to ERASE. In the following algorithms, only two holes are created each time, which means only two holes (that is, two linked lists of border edges) are returned by FINDBORDERS. Note that this search does not have to be performed on the whole triangulation, but only among the edges of the triangles that have been removed.

- The COLLAPSE$(E, \mathbf{p})$ function collapses a set of edges $E$ and places the resulting point at position $\mathbf{p}$

## 4.1 Topological Blending

**Overview** The blending detection is based on the fact that two disconnected meshes sampling the same surface will overlap. As a result, if a scene was composed of two disconnected surfaces, each sampled by two different meshes, and the two surfaces blend together to form one, there should arise an interpenetration between the two meshes. It is then enough to remove the colliding parts and weld the two meshes together. That is what the BLENDING procedure does, as summarized in algorithm 1 and figure 11. The result of the operation is shown in the first steps of figure 1.

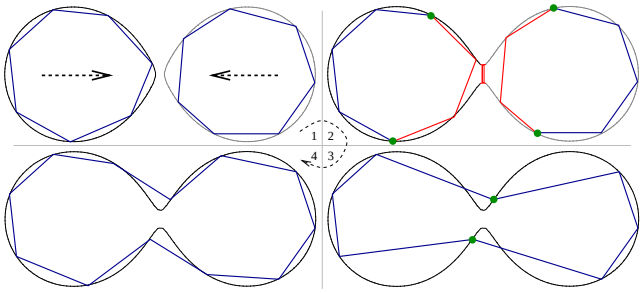



Figure 11: Summary of blending algorithm. *Stage 1:* At time $t_{i-1}$, the two surfaces are separated. *Stage 2:* At time $t_i$, the two geometric meshes collide (red). The corresponding mechanical triangles and their neighbors are marked for deletion (red). *Stage 3:* Marked triangles are deleted, and border points (green) are welded together. The geometric mesh is re-generated. *Stage 4:* At time $t_{i+1}$, the ALTERMESH() procedure subdivides the stretched edges.

**Algorithm 1** BLENDING

$T \Leftarrow$ DETECTCOLLIDINGTRIANGLES()
**for each** $\tau \in T$ **do**
  ERASE(NEIGHBORS($\tau$))
  ERASE($\tau$)
**end for**
$(B_1, B_2) \Leftarrow$ FINDBORDERS()
$(B_1, B_2) \Leftarrow$ ADJUST$(B_1, B_2)$
MERGE$(B_1, B_2)$

**Collision Detection** The collision detection is performed on the geometric triangles, so that we take benefits of the best precision available. As a result, the resolution of the coarse mechanical mesh does not matter in topology management, only the chosen geometric resolution does. When a collision arises, the mechanical triangles from which the colliding geometric triangles where issued are removed, as well as their neighbors.
In our implementation, we used a basic naive $O(n^2)$ collision detection algorithm. However, many algorithms and libraries are now available to handle very fast collision detection of thousands of triangles. For real-time approaches, see [26, 18, 14].

**Welding** After removing some mechanical triangles, each of the mechanical meshes has a hole. We find the borders of these holes using the FINDBORDERS procedure. If both borders do not have the same number of edges, we collapse edges from the biggest edge list so that both lists have the same number of edges (procedure ADJUST).
We then weld each point of $B_1$ with a point of $B_2$ in the appropriate order. We first select a pair of starting points to weld, then walk both lists in the same rotating order and weld the rest of the points. To choose the pair of starting points, we choose the closest points in the projection of the lists on a plane of normal $\mathbf{u} = \mathbf{b}_1 - \mathbf{b}_2$ where $\mathbf{b}_i = \text{barycenter}(B_i)$. When welding two points, the resulting point is placed at the barycenter or the two original points, then projected to the surface. The corresponding edges of the welded points are also welded together.

**Multiple Blending** The extension of this approach to blending in several locations at the same time is straightforward. The FINDBORDERS would then return more than two border lists. These lists can easily be paired using the collision information. The welding process is then done on each pair.

**Limitations** There are some limitations to this method. In particular, it is important to observe that the accuracy of the blending detection is dependent of the resolution of the geometric mesh, and not directly on the field function itself.
Also, note that this blending method cannot handle all kinds of blending. For example, the particular

case of three objects blending at the exact same location is a problem. After cutting the colliding triangles, we would have to weld three border lists together, which is not trivial. A solution may be to weld only two of them together and close the hole of the third, then re-perform the blending operation.

## 4.2 Topological Splitting

**Overview** Splitting is managed in three steps. First, we detect the need for splitting and its location using "quasi-critical" points. Then, we cut the mechanical mesh according to a splitting plane, which gives two separate meshes. Finally, we close the holes resulting from the cutting in each of the two new meshes. That is what the SPLITTING procedure does, as summarized in algorithm 2 and figure 12. The result of the operation is shown in the last steps of figure 1.

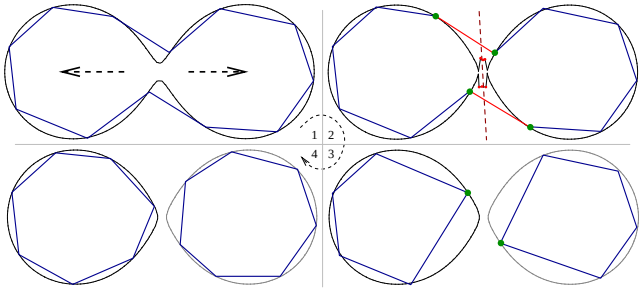



Figure 12: Summary of splitting algorithm. *Stage 1:* At time $t_{i-1}$, the surface is about to be split. *Stage 2:* At time $t_i$, several subdivision points (red) cannot be projected onto the surface. A cutting plane (brown) is computed. The triangles forming a ring crossing the plane are marked for deletion (red). *Stage 3:* Marked triangles are deleted, and border edges (green) are collapsed. The geometric mesh is re-generated. *Stage 4:* At time $t_{i+1}$, the ALTERMESH() procedure subdivides the stretched edges.

**Splitting Detection** As mentioned in section 2.2, we record when a geometric point could not be projected onto the implicit surface. Let us call such a point a "quasi-critical" point, and the mechanical triangle it belongs to a "quasi-critical" triangle. When a quasi-critical point is found, it means that this point is located near a plane dividing two implicit surfaces. Let us call this plane the splitting plane. As explained in [25], the ideal splitting plane should pass through a critical point, and have as normal one of the eigenvectors of the Hessian matrix of the field function at this critical point. Since searching for critical points is expensive and that this technique limits us to $C^2$ continuous functions, we decide to find an approximation of this ideal splitting plane using the mechanical and geometric meshes we already have.
Note, however, that in the case of implicit surfaces defined by skeletons, a solution for the splitting plane may lie in the skeleton itself. Knowing the position and the field function of the skeleton

elements, it seems relatively easy to detect the exact location and orientation of this plane. Nevertheless, since we want our method to withstand any kind of implicit surface, we choose an algorithm based only on the triangulations. The splitting of the surface is done only when a certain number of quasi-critical points $min_{split}$ has been reached. We choose the center of the splitting plane as the barycenter of all the quasi-critical points found. The normal of the plane is then chosen as the average of the cross product between the normals of all the quasi-critical triangles.

**Algorithm 2** SPLITTING

Let $n$ be the number of quasi-critical points
**if** $n \geq min_{split}$ **then**
  Let $\mathbf{p}_1 \dots \mathbf{p}_n$ be the quasi-critical points
  Let $m$ be the number of quasi-critical triangles
  Let $\tau_1 \dots \tau_m$ be the quasi-critical triangles
  Let $\mathbf{n}_1 \dots \mathbf{n}_m$ be the normals of $\tau_1 \dots \tau_m$
  $\mathbf{p} \Leftarrow \text{barycenter}(\mathbf{p}_{1\dots n})$
  $\mathbf{n} \Leftarrow (0,0,0)$
  **for each** $(i,j), 1 \leq i < j \leq m$ **do**
    $\mathbf{n}_{ij} \Leftarrow \text{normalize}(\mathbf{n}_i \times \mathbf{n}_j)$
    $\mathbf{n} \Leftarrow \mathbf{n} + \text{sign}(\mathbf{n} \cdot \mathbf{n}_{ij})\mathbf{n}_{ij}$
  **end for**
  $\mathbf{n} \Leftarrow \text{normalize}(\mathbf{n})$
  Let $P$ be the plane of normal $\mathbf{n}$ passing through $\mathbf{p}$
  $R \Leftarrow$ RING $(P, t_1 \dots t_m)$
  ERASE$(R)$
  $(B_1, B_2) \Leftarrow$ FINDBORDERS$()$
  $\mathbf{b}_1 \Leftarrow$ PROJECTPOSITION(barycenter$(B_1)$)
  $\mathbf{b}_2 \Leftarrow$ PROJECTPOSITION(barycenter$(B_2)$)
  COLLAPSE$(B_1, \mathbf{b}_1)$
  COLLAPSE$(B_2, \mathbf{b}_2)$
**end if**

**Cutting** We then search for a ring of mechanical triangles that cross the splitting plane and that contains at least one quasi-critical triangle (procedure RING). When such a ring is found, its triangles are deleted, which results in creating two holes.

**Closing** The bounds of these holes are retrieved using procedure FINDBORDERS, which returns two lists of border edges. Each border is closed by collapsing all its edges to a single point. This point is placed at the barycenter of the list and projected onto the surface.

**Multiple Splitting** The extension of this approach to splitting in several locations at the same time is harder than for blending. The problem is that we have to find which quasi-critical points correspond to which splitting location. The solution to this problem lies in the field of cluster analysis, and may be found using hierarchical tree clustering or k-means clustering with v-fold cross validation [7]. We did not try to solve this special case.

**Limitations** It is important to note that this technique could not be used without geometric refinement. Actually, when the mechanical points are moved, both their position and their displacement are constrained to the surface, so that they are compelled to stay on it. When the implicit surface splits, these mechanical points stay on one of the two new surfaces, and only some triangles linking the two surfaces cross the splitting plane. Therefore, it is the sampling of these mechanical triangles through geometric refinement that allows us to find the splitting plane. This means that $\alpha_r$ has to be high enough so that refinement appears in splitting areas. It also means that the mechanical mesh should not be too fine.
In addition, the detection of quasi-critical points strongly depends on the projection method. The Newton-Raphson approach we used showed good results on our examples, however we cannot guarantee that it works for every class of field function. In these cases, a better suited projection method has to be found.

# 5 Results

| Constant | Meaning | Value |
|---|---|---|
| E | Young's modulus (for finite element method) | $[100;3000]$ |
| $\nu$ | Poisson's ratio (for finite element method) | 0.3 |
| $k$ | Springs stiffness (for mass-springs comparison) | 100 |
| $e_{min}$ | Maximal edge compression (section 2.4) | 1 |
| $e_{max}$ | Maximal edge stretching (section 2.4) | 2 |
| $\hat{e}$ | Desired mechanical edge length (section 2.4) | user-defined |
| $\varepsilon_e$ | Limit for test on edge length variation (section 2.4) | 0.1 |
| $\alpha_r$ | Max. angle between normals of geometric points (sec. 3) | user-defined |
| $min_{split}$ | Min. # of quasi-critical points triggering a splitting (alg. 2) | 4 |

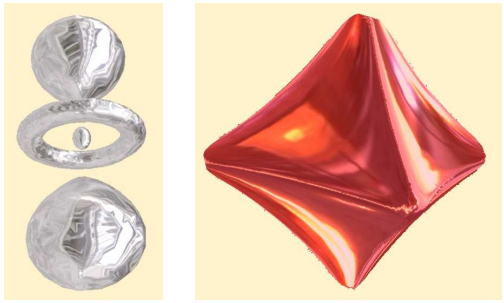
Table 1: Used constants

Results were run on a Pentium M 2GHz with 2GB of RAM and a nVidia Quadro FX Go1400, using the constants of table 1. Note that only two parameters must be user-defined. We tested our method on several classes of implicit surfaces. Figures 1, 2, *right*, 9, 10, 10, *right* and 14 show skeleton-based implicit surfaces with $\frac{1}{d^2}$-like field functions, $d$ being the distance to the skeleton element.

We used different norms for $d$, like $\|\|\|_4$ (figure 9, *bottom*), $\|\|\|_{0.6}$ (figure 13*b*). Figure 13*a* shows our method on a volume data set.

We can see that the surfaces are well sampled and the geometric refinement adds details to areas of high curvature. The mechanical mesh is well-shaped, and manages to follow fast movements of the surfaces. The very rare inversions are only created by fast manipulation or animation and are usually removed in one or two time steps.

For $C^0$ continuous field functions, we observe some artifacts (figure 13), due to the fact that on these examples the potential is not differentiable everywhere. As a result, some points are not well constrained. However, the mechanical system remains stable and manages to find a solution, including on data containing border edges. Nevertheless, our topology handling mechanism does not work well because the projection method is not adapted to $C^0$ continuous field functions.



a b

Figure 13: *(a)* Method applied to a $C^0$ continuous volume data. Although the constraint method is not adapted, the system finds a solution. *(b)* Method applied on a skeleton-based $\|\|\|_{0.6}$-ball. The system finds a solution even though there are discontinuities in the derivative. Notice the undesired jaggy sharp features.

Results presented in table 2 correspond to computations of ALTERMESH (section 2.4) and QUASI-STATIC. Screenshots of examples are shown in figure 14. We see that our optimized FEM-based approach is very close to a mass-spring system in terms of speed. Only the inversion treatment, presented in section 2.3, which needs one more call to the field function, makes the FEM approach a little more dependent on the field function, as shown on example 3. The cost of the latter being not insignificant, it can make a noticeable difference for large models. On the other hand, the improved robustness of the FEM approach compared to the mass-spring system allows us to save computation time by modifying the mesh less often, as we can see for example 4. Moreover, the FEM system is more stable with a large time step $dt$.

Speed results for different geometric refinement criteria are presented in table 3. The speed measurements include notably the handling of the mechanical system, the creation of the geometry and the rendering.

| | FEM | | MS | |
|---|---|---|---|---|
| Number of iterations | 2 | 10 | 2 | 10 |
| example 1 ($dt$=40ms) | 9 | 16 | 10 | 20 |
| example 2 ($dt$=40ms) | 15 | 16 | 12 | 14 |
| example 3 ($dt$=40ms) | 197 | 215 | 148 | 183 |
| example 4 ($dt$=40ms) | 90 | 130 | not stable | not stable |
| example 4 ($dt$=10ms) | 90 | 130 | 110 | 190 |

Table 2: Speed comparisons (in ms per step) between our FEM approach and a mass-spring system for the examples presented in figure 14 using the presented quasi-static integration with various numbers of iterations for the conjugate gradient.

| $\alpha_r$ | 37° | 26° | 18° | 8° |
|---|---|---|---|---|
| Number of geometric triangles | 832 | 1847 | 3111 | 12475 |
| Frames per second | 58 | 38 | 25 | 7 |

Table 3: Rendering speeds (in frames per second) for the first example of figure 14 with several values for $\alpha_r$.

# 6 Conclusion and Future Work

In this paper, we have presented a method based on two dynamic triangulations of an implicit surface, a *mechanical* one and a *geometric* one. Thanks to this decoupling, we can address the needs for a stable mechanical system and a detail-friendly geometry at the same time.
The mechanical resolution is robust and fast thanks to several features. First, it uses a finite element method where all elements have a common and constant ideal rest state. Secondly, the quasi-static integration avoids accumulated errors. Finally, the treatment of inverted elements allows the handling of degenerate configurations.
By subdividing the geometric triangulation in areas of high curvature, we generate a mesh that is detailed while keeping the number of triangles low.
In addition, the geometric details allow us to detect precisely the blending and the splitting of surfaces and to treat these topology changes, providing more topological correctness than [24].
As a result, this method provides robust, interactive and detailed rendering of animated or manipulated implicit surfaces, including surfaces that undergo splitting or blending.

There are areas in which our method could be improved. A different projection method has to be found for $C^0$ continuous functions, in order to handle sharp features and topology changes. We

believe that the criteria for subdivision, collapsing and refinement can be improved. Also, the idea of [13] to modify the meshes in the places where the values are affected by user manipulation could be used here.

Future work should concentrate on treating more cases of topology changes, such as the closing and opening of holes. Good results may also be obtained by connecting this work with the treatment of topology changes through the tracking of critical points [25].

# References


[1] C. Andújar, P. Brunet, A. Chica, I. Navazo, J. Rossignac, and A. Vinacua. Optimizing the topological and combinational complexity of isosurfaces. In *Computer-Aided Design*, 2005.

[2] R. S. Avila and L. M. Sobierajski. A haptic interaction method for volume visualization. In *Proceedings of the 7th conference on Visualization*, 1996.

[3] R. E. Bank, A. H. Sherman, and A. Weiser. *Refinement Algorithms and Data Structures for Regular Local Mesh Refinement*. 1983.

[4] D. Baraff and A. P. Witkin. Large steps in cloth simulation. In *Proceedings of SIGGRAPH 98*, 1998.

[5] J. Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 1988.

[6] F. Bornemann and R. Erdmann, B.and Kornhuber. Adaptive multilevel-methods in three space dimensions. *Intl. J. for Numer. Meth. in Eng.*,, 1993.

[7] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA, 1984.

[8] M.-P. Cani. Implicit representations in computer animation : a compared study. In *Proceedings EUROGRAPHICS Workshop on Implicit Surfaces'99*, 1999.

[9] P. Crossno and E. Angel. Isosurface extraction using particle systems. In *Proceedings of IEEE Visualization '97*, 1997.

[10] P. J. Crossno and E. S. Angel. Spiraling edge: Fast surface reconstruction from partially organized sample points. In *Proceedings of IEEE Visualization '99*, 1999.

[11] L. H. de Figueiredo, J. de Miranda Gomes, D. Terzopoulos, and L. Velho. Physically-based methods for polygonization of implicit surfaces. In *Proceedings of the conference on Graphics interface '92*, 1992.

[12] T. K. Dey, H. Edelsbrunner, S. Guha, and D. V. Nekhayev. Topology preserving edge contraction. *Publications de l' Institut Mathematique (Beograd)*, 1999.

[13] E. Ferley, M.-P. Cani, and J.-D. Gascuel. Practical volumetric sculpting. *The Visual Computer*, 2000.

[14] N. K. Govindaraju, M. Lin, and D. Manocha. Quick-cullide: Fast inter- and intra-object collision culling using graphics hardware. In *Proceedings of IEEE Virtual Reality 2005*, 2005.

[15] M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, and M. Gross. Real-time ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum*, 2005.

[16] C.-C. Ho, F.-C. Wu, B.-Y. Chen, Y.-Y. Chuang, , and M. Ouhyoung. Cubical marching squares: Adaptive feature preserving surface extraction from volume data. In *Proceedings of EUROGRAPHICS 2005*, 2005.

[17] G. Irving, J. Teran, and R. Fedkiw. Invertible finite elements for robust simulation of large deformation. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2004.

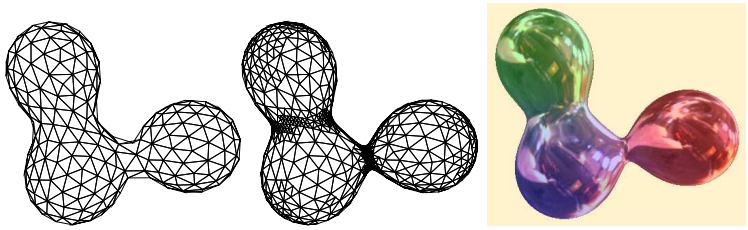
[18] N. Jain, I. Kabul, N. K. Govindaraju, D. Manocha, and M. Lin. Multi-resolution collision handling for cloth-like simulations. *Computer Animation and Virtual Worlds*, 2005.

[19] F. Levet, J. Hadim, P. Reuter, and C. Schlick. Anisotropic sampling for differential point rendering of implicit surfaces. In *Proceedings of the Winter School of Computer Graphics'05*, 2005.

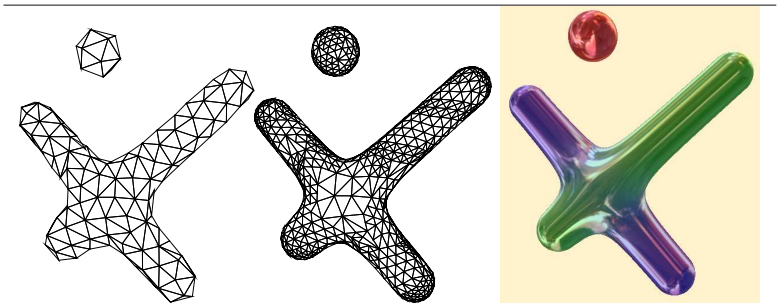
[20] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, 1987.

[21] J. W. Milnor. *Topology from the Differentiable Viewpoint*. University Press of Virginia, 1965.

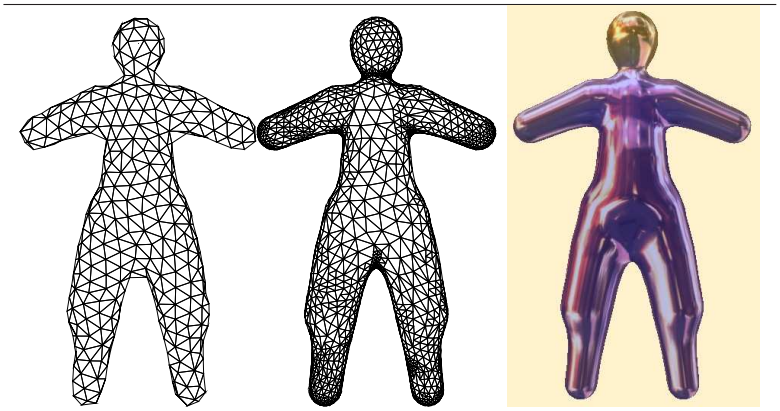
[22] M. Müller, J. Dorsey, L. McMillan, R. Jagnow, and B. Cutler. Stable real-time deformations. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2002.
[23] M. Nesme, Y. Payan, and F. Faure. Efficient, physically plausible finite elements. In *Eurographics (short papers)*, 2005.
[24] H.-C. Rodrian and H. Moock. Dynamic triangulation of animated skeleton-based implicit surfaces. In *Proceedings of the EUROGRAPHICS Workshop on Implicit Surfaces'96*, 1996.
[25] B. T. Stander and J. C. Hart. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In *Proceedings of SIGGRAPH 97*, 1997.
[26] G. Van Den Bergen. Efficient collision detection of complex deformable models using AABB trees. In *Journal of Graphics Tools*, 1997.
[27] A. P. Witkin and P. S. Heckbert. Using particles to sample and control implicit surfaces. In *Proceedings of SIGGRAPH 94*, 1994.



*≈ 350 particles, ≈ 690 mechanical triangles, 3 skel. elements,* $\alpha_r = 26^\circ$



*≈ 180 particles, ≈ 350 mechanical triangles, 3 skel. elements,* $\alpha_r = 32^\circ$*)*



*≈ 530 particles, ≈ 1050 mechanical triangles, 12 skel. elements,* $\alpha_r = 37^\circ$

Figure 14: Four skeleton-based examples, with a $\frac{1}{d^2}$-like field function. The first column shows the mechanical mesh, the second is the geometric mesh and the last is a textured rendering.

# Contents

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399