Simulation and Visualization of topology-changing plastic material

Marc Gissler[*]

Computer Graphics
University of Freiburg
Germany

Abstract

We present a dynamic simulation framework for topology-changing deformable objects. The objects are represented using tetrahedral meshes and deformations are governed by a corotational finite element approach for linear elasticity and plasticity. Geometric constraints are employed to efficiently handle topology changes in a unified way. Topology changes comprise fracturing and merging of deformable objects. Fracturing is realized by breaking existing constraints, while merging is implemented by generating new constraints. Thus, geometric constraints significantly reduce the complexity of data structure updates in the context of topology changes. This improves the computational efficiency and reduces the implementation effort. Experiments illustrate the versatility and the efficiency of our approach.

Keywords: Physically Based Animation, Finite Element Method, Constraints, Topology Changes, Fracturing, Merging

1 Introduction

The simulation of physical systems at interactive rates is essential for a variety of applications, ranging from medical training systems to computer animation for games and feature films. In order to implement an interactive system, all components ranging from fluid and solid modeling to collision detection and contact handling have to be considered.

Efficient data structures are an important aspect in terms of interactivity. While the processing of data structures is less demanding for deformable solids, the update of these representations poses additional challenges in the context of topology changes. Therefore, we propose to employ an efficient and accurate approach to geometric constraints in order to simplify and to accelerate these updates.

Our contribution. We present a dynamic framework for topology-changing deformable tetrahedral meshes. Elasto-mechanical properties are simulated with a corotational finite element approach for linear elasticity and a strain-state based plasticity model. Topology changes such as breaking and merging are efficiently processed in a unified way with minimized complexity in terms of data structure updates. In a pre-processing step, objects are decomposed into single tetrahedrons and efficient and accurate constraints [7] are employed to re-assemble the object to its original geometry. During the dynamic simulation, fracturing and merging are realized by removing existing constraints and by generating new constraints, respectively. Information on internal strain can be considered to control fracturing, while collision information can be used to induce a merging step. In each simulation step, a consistent high-resolution surface mesh is maintained using subdivision surfaces.

[*]gisslerm@informatik.uni-freiburg.de

2 Related Work

We give a short overview over the three main areas of research we have incorporated into our scheme.

FEM: Twenty years ago, Terzopolous et al. [16] proposed the use of physically-based models for the animation of elastic and plastic objects. They used surfaces to represent the objects and solved the governing partial derivatives using finite difference schemes. Modeling three-dimensional volumes using a finite element method is based on techniques presented in mechanical engineering and computer graphics [3] [16] and widely used [13] [11] [1].

Constraints: There exist two main strategies to solve constraints numerically. The first technique computes forces or impulses using maximal coordinates. In this context, Lagrange multipliers or propagation methods have been proposed. As an example, [21] uses Lagrange multipliers to compute constraints on linear deformations. In contrast, the second technique reduces the number of coordinates to represent the system state. The remaining coordinates are so-called reduced coordinates or generalized coordinates. Reduced coordinates are preferred in global approaches, if the number of constraints is large compared to the number of degrees of freedom. As an example, [8] presents an approach for contact constraints of deformable bodies. Recently, a constraint approach for articulated rigid bodies has been presented in [20]. This approach employs information on the underlying numerical integration scheme. [7] also considers the underlying integration scheme but proposes a local, non-iterative solution for deformable objects.

Topology changes: Shortly after physically based models were proposed for animating elastic models, [17] described how to also simulate fracture effects.O'Brien et al. presented a finite element technique to simulate brittle [13] and ductile [12] fracture in combination with elasto-plastic materials. By analyzing the stress tensors computed with FEM, their algorithm determines where cracks should initiate and in what directions they should propagate. Recently Müller et al. [11] proposed a method to animate and fracture a detailed surface mesh along with the underlying hexahedral mesh. Up until now, only little attention was given to methods that allow for the merging of objects. Simulating virtual clay could be an application for such methods. [4] and [14] use volumetric models that define the object as an iso-surface of a three-dimensional scalar field function stored in a grid. This representation allows for the addition and removal of material. Theoretically, the merging of objects would therefore be possible with this methods but is not explicitly mentioned. To our knowledge there exists no work dealing with merging of objects using tetrahedral meshes as volumetric representation.

3 Deformable solids

Our simulation framework allows for the simulation of both elastically and plastically deformable solids. We use the corotational finite element approach as proposed by Müller et al. [11] and Becker [1]. We describe the main steps of the two models in the next two sections.

3.1 Elasticity model

The Finite Element Method requires an object to be discretized into a finite set of elements. Therefore, we approximate a deformable solid by generating a coarse tetrahedral mesh as was proposed by Spillmann et al. [15], yielding a set of tetrahedrons and mass points (see Figure 1). At every mass point we define a displacement vector $\mathbf{u}$, which gives the difference between the position of the mass point in the undeformed and the deformed state. Every point $\mathbf{x}$ within a tetrahedral element can then be linearly interpolated using the displacement vectors at the four mass points of the tetrahedron, given by $\mathbf{u}(\mathbf{x}) = N(\mathbf{x})\tilde{\mathbf{u}}$, with $\mathbf{u}(\mathbf{x})$ being the displacement field over the whole mesh, N($\mathbf{x}$) being a shape function i.e. the barycentric coordinates of $\mathbf{x}$, and $\tilde{\mathbf{u}} = [\mathbf{u}_1\ \mathbf{u}_2\ \mathbf{u}_3\ \mathbf{u}_4]^T$ being a collection of the four displacement vectors.

In [1] Becker shows how the Cauchy linear strain tensor within every tetrahedral element e can be computed using the displacement field, yielding

$$\varepsilon = \mathbf{B}_e\tilde{\mathbf{u}}, \tag{1}$$

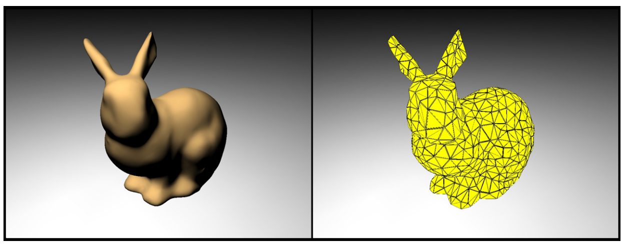

Figure 1: Tetrahedralization: The triangulated surface of the Stanford bunny serves as the input model (left). A volumetric mesh consisting of 974 points and 4053 tetrahedrons is generated (right).

where $\mathbf{B}_e$ is a constant matrix and can be pre-computed. Furthermore, the relation between the stress ε and the strain σ is given by the constitutive equation $\sigma = \tilde{\mathbf{C}}\varepsilon$, where $\mathbf{C}$ is a symmetric matrix defining the mechanical properties of the material. In order to compute the dynamic behavior of the objects we derive forces $\mathbf{f}_e$ for every tetrahedral element from the potential energy of the element. It turns out, that the forces are linearly dependent on the displacements $\tilde{\mathbf{u}}$ defined at the four mass points of the element:

$$\mathbf{f}_e = \mathbf{K}_e\mathbf{u}, \tag{2}$$

with $\mathbf{K}_e = V_e\mathbf{B}_e^T\tilde{\mathbf{C}}_e\mathbf{B}_e$ being the stiffness matrix and V_e the volume of the element.

3.1.1 Corotational stiffness

The linear Finite Element Method is not rotationally invariant. Thus, objects grow in size under large deformation. In [11] Müller et. al propose the corotational stiffness algorithm to overcome this shortage.

If $\mathbf{p}_1, ..., \mathbf{p}_4 \in \mathbb{R}^3$ and $\mathbf{q}_1, ..., \mathbf{q}_4 \in \mathbb{R}^3$ denote the points of a tetrahedron in the undeformed and deformed state respectively, then the transformation of a tetrahedron is given by:

$$\mathbf{A} = \mathbf{H}\mathbf{P}^{-1} = \begin{pmatrix} & \mathbf{B} & & \mathbf{t} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3}$$

with $\mathbf{H} \in \mathbb{R}^{2x4}$ and $\mathbf{P} \in \mathbb{R}^{2x4}$ containing the points $\mathbf{p}_i$ and $\mathbf{q}_i$ respectively, $\mathbf{t} \in \mathbb{R}^3$ denoting the translational and $\mathbf{B} \in \mathbb{R}^{3x3}$ denoting the rotational and stretching part of the transformation. A polar decomposition extracts the rotational part $\mathbf{R} \in \mathbb{R}^{3x3}$ from $\mathbf{B}$. Employing $\mathbf{R}$ to the force computation yields

$$\mathbf{f}_e = \mathbf{R}_e\mathbf{K}_e(\mathbf{R}_e^{-1}\bar{\mathbf{t}} - \bar{\mathbf{p}}), \tag{4}$$

with $\bar{\mathbf{t}} = [\mathbf{q}_1\,\mathbf{q}_2\,\mathbf{q}_3\,\mathbf{q}_4]^T$, $\bar{\mathbf{p}} = [\mathbf{p}_1\,\mathbf{p}_2\,\mathbf{p}_3\,\mathbf{p}_4]^T$ and $\mathbf{R}_e \in \mathbb{R}^{12\times12}$ having the rotation matrix $\mathbf{R}$ four times on its diagonal.

3.1.2 Dynamic deformation

Using the derived forces, we now can compute the dynamics of the system

$$\mathbf{M}\frac{d^2\mathbf{X}(t)}{dt^2}+\mathbf{D}\frac{d\mathbf{X}(t)}{dt}=\mathbf{F}^{ext}(\mathbf{X},t)-\mathbf{KU}, \qquad (5)$$

where the coordinate vector $\mathbf{X}$ is made a function of time $\mathbf{X}(t)$, $\mathbf{M}$ is the mass matrix, $\mathbf{D}$ the damping matrix, $\mathbf{K}$ an assembly of all stiffness matrices $\mathbf{K}_e$ and $\mathbf{U}$ the displacement matrix. For numerical integration we use the Verlet scheme, because it is an explicit integration scheme that can be computed very efficiently and allows for comparatively large time steps [18].

3.2 Plasticity model

This section shows how to add elasto-plastic properties to the simulation model. This technique was proposed by [12] and adopted to linear corotational FEM by [11]. As opposed to elastic materials, elasto-plastic materials store part of the deformation. They move to a resting state between the deformed and undeformed state after removing the external forces (see Figure 2). The resting state depends on the elastic limit of the material and the creep. The creep is the permanent deformation resulting from prolonged application of stress below the elastic limit. It is influenced by the magnitude of the load and the time the load is applied.

Using equation 1 and considering the rotation-free displacement of an element yields the total strain

$$\varepsilon_{total}=\mathbf{B}_e\tilde{\mathbf{q}}=\mathbf{B}_e(\mathbf{R}_e^{-1}\bar{\mathbf{t}}-\bar{\mathbf{p}}). \qquad (6)$$

resulting in a deformation of an object . To store some of the deformation in the elasto-plastic material we remove a portion of the strain - the plastic strain $\varepsilon_{plastic}$. Therefore, the elastic strain is $\varepsilon_{elastic}=\varepsilon_{total}-\varepsilon_{plastic}$.

Three scalar parameters control the plasticity of a material: c_{yield}, c_{creep} and c_{max}. c_{yield} defines the elastic limit. If the 2-norm of the elastic strain exceeds the elastic limit, the plastic strain absorbs part of it: $\varepsilon_{plastic} \mathrel{+}= \triangle t * c_{creep} * \varepsilon_{elastic}$. c_{creep} defines the creep of the material. It gives the amount of strain the material can absorb per time step. If $c_{creep}\in[0...1/\delta t]$ is $1/\delta t$, all the elastic strain is absorbed in one time step. c_{max} defines the maximum plastic strain that can be stored in the material. If the 2-norm of the plastic strain exceeds this threshold, it is adjusted respectively: $\varepsilon_{plastic} \mathrel{*}= c_{max}/||\varepsilon_{plastic}||_2$. We have to integrate the plastic strain into the system of equations. Therefore, we compute corresponding plastic forces $f_{plastic}$ using the definition of the stiffness matrix in section 3.1:

$$\begin{aligned}\mathbf{f}_{plastic} &= \mathbf{R}_e\mathbf{K}_e\tilde{\mathbf{q}}_{plastic}\\ &= \mathbf{R}_e\mathbf{K}_e\mathbf{B}_e^{-1}*\varepsilon_{plastic}\\ &= \mathbf{R}_e(V_e\mathbf{B}_e^T\tilde{\mathbf{C}}\mathbf{B}_e)\mathbf{B}_e^{-1}*\varepsilon_{plastic}\\ &= \mathbf{R}_eV_e\mathbf{B}_e^T\tilde{\mathbf{C}}\cdot\varepsilon_{plastic}\\ &= \mathbf{R}_e\mathbf{P}_e\cdot\varepsilon_{plastic}.\end{aligned}$$

As can be seen the plasticity matrix $\mathbf{P}_e=V_e\mathbf{B}_e^T\tilde{\mathbf{C}}$ maps the plastic strain to plastic forces. An advantage of this technique is the possibility to pre-compute this matrix for each element. Furthermore, only a force vector is added to equation 5. Thus, the plasticity does not change the condition of the linear system that needs to be solved:

$$\mathbf{F}^{ext}(\mathbf{X},t)=\mathbf{M}\frac{d^2\mathbf{X}(t)}{dt^2}+\mathbf{D}\frac{d\mathbf{X}(t)}{dt}+(\mathbf{f}^{elastic}-\mathbf{f}^{plastic})(\mathbf{X},t). \qquad (7)$$

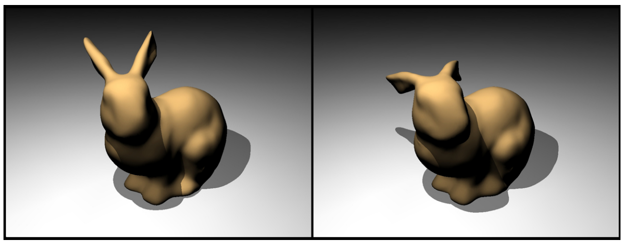
Figure 2: The Stanford bunny in its resting position after it was deformed elastically (left). The bunny in its resting position after it was deformed plastically (right).

4 Constraints

By now, we are able to simulate objects with elastic and plastic properties. Now we want to add additional properties which allow for changes in the topology of the objects. Therefore, we introduce local constraints. Two objects merge with each other by applying constraints, and objects fracture by dissolving constraints. This section describes the constraints for merging and fracturing objects. We use the local constraints method as presented in [7]. The following sections recapitulate the main steps of the method.

4.1 Considering the integration scheme

The local constraint method employs information on the underlying integration scheme of the simulation. The update from time t to $t+h$ of positions $\mathbf{x}_i^t$ and velocities $\mathbf{v}_i^t$ of mass points m_i can generally be written as

$$\begin{pmatrix}\mathbf{x}_i^{t+h}\\ \mathbf{v}_i^{t+h}\end{pmatrix}=\begin{pmatrix}\mathbf{A}_i\\ \mathbf{B}_i\end{pmatrix}\mathbf{s}_i^t+\begin{pmatrix}c_i\,\mathbf{F}_i^t\\ d_i\,\mathbf{F}_i^t\end{pmatrix} \qquad (8)$$

with system matrices $\mathbf{A}_i$ and $\mathbf{B}_i\in\mathbb{R}^{3\times k}$, state vector $\mathbf{s}_i^t\in\mathbb{R}^k$ and scalars c_i and $d_i\in\mathbb{R}$.

As we use the Verlet integration scheme, positions and velocities are updated with

$$
\begin{aligned}
\mathbf{x}_i^{t+h} &= 2\mathbf{x}_i^t - \mathbf{x}_i^{t-h} + \frac{h^2}{m_i}\mathbf{F}_i^t \\
\mathbf{v}_i^{t+h} &= \frac{1}{2h}\left(\mathbf{x}_i^{t+h} - \mathbf{x}_i^{t-h}\right)
\end{aligned}
\tag{9}
$$

where the velocity update can be rewritten as

$$
\mathbf{v}_i^{t+h} = \frac{1}{h}\left(\mathbf{x}_i^t - \mathbf{x}_i^{t-h} + \frac{h^2}{2m_i}\mathbf{F}_i^t\right). \tag{10}
$$

By incorporating these equations into equation 8 we get

$$
\begin{aligned}
\mathbf{A}_i &= \begin{pmatrix} 2\mathbf{I}_3 & -\mathbf{I}_3 \end{pmatrix} & c_i &= \frac{h^2}{m_i} \\
\mathbf{B}_i &= \begin{pmatrix} \frac{1}{h}\mathbf{I}_3 & -\frac{1}{h}\mathbf{I}_3 \end{pmatrix} & d_i &= \frac{h}{2m_i}
\end{aligned}\ , \tag{11}
$$

and for the state vector we get

$$
\mathbf{s}_i^t = \begin{pmatrix} \mathbf{x}_i^t \\ \mathbf{x}_i^{t-h} \end{pmatrix}. \tag{12}
$$

4.2 Constraint forces

The goal is to compute a constraint force $\tilde{\mathbf{F}}_i^t$ to meet an implicitly or explicitly defined constraint on a position or velocity which are denoted with $\tilde{\mathbf{x}}_i^{t+h}$ and $\tilde{\mathbf{v}}_i^{t+h}$, respectively. Adding this constraint force to equation 8 yields

$$
\begin{pmatrix} \tilde{\mathbf{x}}_i^{t+h} \\ \tilde{\mathbf{v}}_i^{t+h} \end{pmatrix} = \begin{pmatrix} \mathbf{A}_i \\ \mathbf{B}_i \end{pmatrix} \mathbf{s}_i^t + \begin{pmatrix} c_i \left(\mathbf{F}_i^t + \tilde{\mathbf{F}}_i^t\right) \\ d_i \left(\mathbf{F}_i^t + \tilde{\mathbf{F}}_i^t\right) \end{pmatrix}. \tag{13}
$$

4.2.1 Points-to-point forces

We are specifically interested in constraint forces that hold n points $\mathbf{x}_i^t$ at a jointed position x_{goal}. This jointed position is not user-defined but instead the result of the n equations computing the n constraint forces $\mathbf{F}_i^t$. Adding the equation preserving the momentum in the mass point system yields

$$
\begin{aligned}
\mathbf{x}_{goal} = \tilde{\mathbf{x}}_i^{t+h} &= 2\mathbf{x}_i^t - \mathbf{x}_i^{t-h} + \frac{h^2}{m_i}\left(\mathbf{F}_i^t + \tilde{\mathbf{F}}_i^t\right) \\
&= \frac{1}{\sum_j m_j}\sum_j m_j \mathbf{x}_j^{t+h}.
\end{aligned}
\tag{14}
$$

This is the center of mass of the n positions $\mathbf{x}_j^{t+h}$ of the points involved, as was shown in [7]. Figure 3 illustrates this procedure for three points. Solving this equation for $\tilde{\mathbf{F}}_i^t$, we compute the constraint forces for a points-to-point constraint by

$$
\begin{aligned}
\tilde{\mathbf{F}}_i^t &= -\frac{m_i}{h^2}\left(\mathbf{x}_i^{t+h} - \mathbf{x}_0^{t+h}\right) + \frac{m_i}{\sum_j m_j}\sum_j \frac{m_j}{h^2}\left(\mathbf{x}_j^{t+h} - \mathbf{x}_0^{t+h}\right) \\
&= \frac{m_i}{h^2}\left(\frac{1}{\sum_j m_j}\sum_j m_j \mathbf{x}_j^{t+h} - \mathbf{x}_i^{t+h}\right).
\end{aligned}
\tag{15}
$$

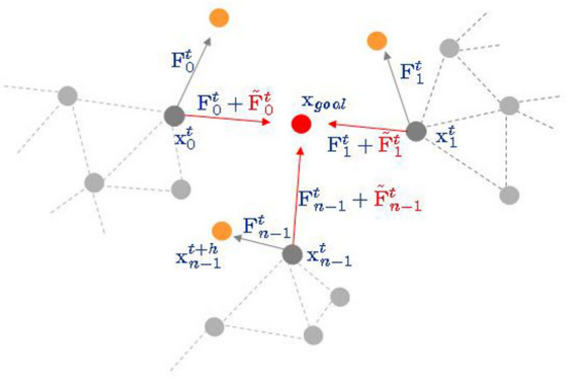

Figure 3: Three points (gray dots) are constrained together. Orange dots indicate the "would-be" positions of the three points in $t+h$. However, actual force (gray arrow) and constraint force together (red arrow) move the point to the center of gravity of the three points (red dot).

The positions $\mathbf{x}_i^{t+h}$ are computed using Verlet integration as well. Thus, the constraint forces can easily be computed with an additional integration step for all affected points. However, the more points are involved in the constraints, the more integrations have to be executed per constraint.

5 Topology changes

This section describes how to use the constraints described in the previous section for modeling the merging and breaking and therefore the topology changes of objects. If two objects collide with each other constraints are computed to keep them stuck together. Conversely, constraints are dissolved, if an object should break at at a specific position.

5.1 Merging

An object is able to merge with another object, if they collide with with each other. The collision detection is accomplished by the spatial hashing approach as presented by Teschner et al. [19] and its extension for consistent penetration depth estimation by Heidelberger et. al [9]. The algorithm in [9] processes three steps that are used in our merging algorithm: First, it detects all colliding points. Second it labels all colliding points adjacent to one or more non-colliding points as border points. Third, it detects all intersecting edges that contain one non-colliding point and one border point.

Instead of computing a collision response we use the information from the collision detection to constrain the two objects. Therefore, we loop through all the intersecting edges and perform the following steps:

- Find a border point that is also a surface point.

- Use one of the intersecting edges this border point is part of to find the nearest permissible adjacent surface point on the penetrated object.
- Constrain the two points found in the previous steps. Insert points into possibly existing constraints at one of the points.

We demand several quality criteria to be fulfilled while looping through these three steps: In step one we search for a border point also being a surface point because we want both points to lie on the surface of their tetrahedral meshes. This is a precondition for the visualization algorithm described in section 6. Step two demands a permissible surface point. A point is not permissible if constraining it would yield a tetrahedron with zero ore negative volume.

The method can handle the self-collisions an therefore the merging of objects with itself. In fact, the algorithm only knows about points, tetrahderons and constraints and does not distinguish between objects as such.

5.2 Fracturing

We assume a tetrahedron being the smallest part of an object. In a pre-processing step we divide an object into its individual tetrahedrons it was built of. Additionally we store the connectivity between the tetrahedrons using the constraints. Points duplicated from the same initial point form a points-to-point constraint. This yields an object built up with individual tetrahedrons and held together by constraint forces.

This special data structure gives us the opportunity to fracture objects very fast. The only thing we have to do is removing points from constraints. This yields the loss of connectivity between tetrahedrons and therefore fractures the object. In contrast to other approaches we do not have to duplicate points during run-time since this was done in a pre-processing step. This may be considered as an overhead as long as no fracture occurs. On the other hand we simulate the worst case, the fracture of a tetrahedral mesh into all its simplices, right from the start. Thus, constant simulation times are guaranteed during all possible events.

A point is removed from the constraint, if the 2-norm of the constraint force computed to satisfy the constraint exceeds a certain scalar threshold. This is done recursively:

1. compute the center of mass of the constraint
2. loop through all points of the constraint and compute their constraint forces
3. if the 2-norm of the constraint force of a point exceeds the specified threshold, remove it from the current constraint and insert it into a new constraint group else leave the point in the current constraint
4. if if no points were removed from the current constraint, return
5. repeat from step 1 with the current constraint
6. repeat from step 1 with the new constraint

Employing this method allows for the removal of individual points from the constraint instead of dissolving the whole constraint (see Figure 4).

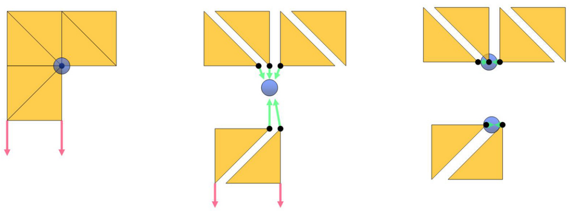

Figure 4: External forces (red) act upon points of the object (left). The object was moved unconstrained to $\mathbf{x}^{t+h}$. Constraint forces (green arrows) are computed based on the center of mass (blue circle) of the points. The constraint forces of the lower two points exceed a threshold. (middle). Two new constraints were built. Now, all the constraint forces lie below the given threshold (right).

6 Surface animation

In the following subsections we describe how we generate the surfaces of our objects and how we adopt changes in topology from the simulation level to the visualization level. The algorithm both handles the merging as well as the fracturing of objects in the same way. Furthermore, the algorithm keeps an initially watertight surface mesh watertight throughout the simulation.

6.1 Generation

In this section we describe how to generate the surface mesh of the objects in a scene. We first describe how the algorithm works in general and then explain the modifications to adapt the algorithm to topological changes.

Given a tetrahedral mesh we search for the surface of this mesh. Therefore we loop through all the tetrahedrons and visit their faces. Faces visited twice are lying inside the mesh. Faces only visited once lie on the surface of the mesh. They build the set surface faces. Now, we consider more than one tetrahedral mesh and a set of constraints applied onto this mesh. We assign a unique ID to every constraint representing all the points participating in a constraint. We loop through all the tetrahedrons of all meshes as described above, but this time we consider the constraints and the unique IDs assigned to them. Thus, faces having their three faces constrained in the same constraints are considered as faces lying adjacent to each other and therefore inside an object. As a result, we get all the surface faces of all tetrahedral meshes in the scene with respect to the constraints in the scene. As can be seen the algorithm does not distinguish between tetrahedral meshes but between points constrained or not.

6.2 Smoothing

We use surface subdivision to smooth the resulting surface mesh generated in the previous section. Subdivision schemes defining smooth surfaces have been introduced by Catmull and Clark [2], Doo and Sabin [5], and Loop [10]. In our implementation we use the Loop scheme [10] and a modified version of the Butterfly subdivision scheme [6] [22]. Both can be applied to triangular meshes. Furthermore, they are local, stationary and uniform.

The Loop scheme is an approximating scheme where the vertices of the control mesh do not need to lie on the limit surface. This perfectly smoothes the control mesh making the control mesh disappear completely. However, if it is applied to a control mesh with high curvature such as a single tetrahedron, the resulting mesh might be smaller in size as the control mesh and does not cover the tetrahedral mesh anymore. On the other hand the butterfly is an interpolating scheme leaving the vertices of the control mesh on the limit surface. This prevents the mesh to shrink even in cases of high curvature. However, the low resolution of the control mesh can be identified in the limit surface due to interpolation. Depending on the scenario we use the scheme that suits the best.

6.3 Mesh coupling

We use mesh coupling as proposed by Müller et al. [11] to couple the low resolution tetrahedral mesh with the high resolution subdivision surface mesh. For every vertex in the high resolution mesh we search for the closest tetrahedron in the tetrahedral mesh and store the barycentric coordinates of the vertex with respect to this tetrahedron. In each visualization step the new vertex position is computed by interpolating the positions of the tetrahedron using the stored barycentric coordinates.

Finding the closest tetrahedron for every vertex can be costly. It lies in $O(nm)$ with n being the number of tetrahedrons of the tetrahedral mesh and m being the number of vertices of the surface mesh. Since we know over which edge a vertex is created during subdivision we also know the nearest tetrahedron immediately. This reduces the problem to $O(n)$.

6.4 Fracturing

In Section 5 we described how the tetrahedral mesh is fractured when points are removed from constraints due to their constraint forces exceeding a threshold. Changing or dissolving a constraint triggers the surface mesh update, because it possibly separates two tetrahedrons t_1 and t_2 and their common face f gets exposed. A closing surface has to be computed to cover the two new surface faces.

6.5 Merging

The second event that triggers a surface mesh update is the merging of objects, because constraints are created or altered. If the three points of a surface face are constrained to the three points of another face using the algorithm described in section 5.1, it becomes an inner face and does not contribute to the surface anymore. This poses some conditions on the properties of a surface face. Ideally, the size of the surface faces should not differ greatly, otherwise big surface faces might be constrained onto several small faces. This would not be considered as a merged object in the visualization algorithm since the faces do not match onto each other.

In conclusion, the surface fracturing and the surface merging both trigger the generation of a new surface as soon as changes in the constraints occur.

7 Results

We tested our implementation in various scenarios, ranging from off-line computations to interactive animations. All experiments have been performed on an Intel Core Duo 2.13 GHz PC using an ATI Radeon X1300 graphics card.

The first scenario shows the versatility of the plasticity model. In Figure 5 a hand falls on a plastically deformable plate. For the plasticity parameters, we set c_{yield} to zero. Thus, plastic strain always absorbs part of the elastic strain. c_{creep} was set to one half of the time step, so plastic strain absorbs half of the elastic strain per time step. The scene consists of 10000 mass points and 40000 tetrahedrons and the simulation runs at 4 frames per second in average, including deformation and collision handling.
The second scenario demonstrates the fracturing of a model. Figures 6 and 7 show a teddy with a volumetric mesh composed of 4292 tetrahedrons and 1540 surface faces for which the subdivision algorithm computes a high resolution surface mesh of about 23500 triangles. Removing the arm from the body of the teddy corresponds to dissolving 25 constraints. The closing surface that is generated at the crack adds an additional 1000 triangles to the surface mesh. The simulation runs at an average of 20 frames per second. Currently, visualizing the frame following a topological change takes 200 ms since the surface subdivision algorithm works globally and generates all the 23500 triangles from scratch. A local refinement only in areas where topology changes occur will greatly improve this performance in the near future.
The last scenario shows the merging of some clay cubes (see Figure 8). They stick to each other as soon as they collide with each other. If the properties of an object do not allow the merging with other objects, collision response forces are computed to separate the objects. This is demonstrated by the sphere hitting the pile of cubes.

The cubes consist of 1080 tetrahedrons, the sphere of 1200 tetrahedrons.

8 Conclusion

We have presented a novel approach to handle topology changes. It handles both merging and fracturing of objects in a unified way. It creates constraints to merge objects and dissolves constraints to fracture them. Finding the surface of an object after a modification of the constraints is executed only on the knowledge about the constraints. We do not need to know how many objects are present in the scene. We use surface subdivision to smooth the resulting surface. The algorithm guarantees watertight meshes that surround the object at all times. Our implemented deformation model allows the application of our approach on elastic as well as plastic materials. Currently, we are working on an extension of the proposed method to consider internal stresses in the objects. Computing splitting planes from the stresses allows for crack propagation.

References

[1] M. Becker and M. Teschner. Robust and efficient estimation of elasticity parameters using the linear finite element method. In *Proc. Simulation and Visualization, 2007*, to appear.

[2] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. In *Computer Aided Design 10, 6*, pages 350–355, 1978.

[3] R.D. Cook, D.S. Malkus, and M.E. Plesha. *Applications of Finite Element Analysis*. John Wiley & Sons, New York, third edition, 1987.

[4] G. Dewaele and M.-P. Cani. Virtual clay for direct hand manipulation. In *Eurographics (short papers)*, 2004.

[5] D. Doo and M. Sabin. Analysis of the behaviour of recursive division surfaces near extraordinary points. In *Computer Aided Design 10, 6*, pages 356–360, 1978.

[6] N. Dyn, S. Hed, and D. Levin. Subdivision schemes for surface interpolation. In *Workshop in Computational Geometry , A.C. et. al., Ed., World Scientific*, pages 97–118, 1993.

[7] M. Gissler, M. Becker, and M. Teschner. Local constraint methods for deformable objects. In *Proc. Virtual Reality Interactions and Physical Simulations VriPhys*, pages 25–32, 2006.

[8] K.K. Hauser, C. Shen, and J.F. O'Brien. Interactive deformation using modal analysis with constraints. In *Graphics Interface*, pages 247–256, 2003.

[9] B. Heidelberger, M. Teschner, R. Keiser, M. Mueller, and M. Gross. Consistent penetration depth estimation for deformable collision response. In *Proc. Vision, Modeling, Visualization*, pages 339–346, 2004.

[10] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, Utah University, USA, 1987.

[11] M. Müller and M. Gross. Interactive virtual materials. In *Proc. Graphics Interface*, pages 239–246, 2004.

[12] J.F. O'Brien, A.W. Bargteil, and J.K. Hodgins. Graphical modeling and animation of ductile fracture. In *Proc. ACM SIGGRAPH*, pages 291–294, 2002.

[13] J.F. O'Brien and J.K. Hodgins. Graphical modeling and animation of brittle fracture. In *Proc. ACM SIGGRAPH*, pages 287–296, 1999.

[14] R.N. Perry and S.F. Frisken. Kizamu: A system for sculpting digital characters. In *Proc. ACM SIGGRAPH*, pages 47–56, 2001.

[15] J. Spillmann, M. Wagner, and M. Teschner. Robust tetrahedral meshing of triangle soups. In *Proc. Vision, Modeling, Visualization VMV*, pages 9–16, 2006.

[16] D. Terzopolous, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. In *Proc. ACM SIGGRAPH*, pages 205–214, 1987.

[17] D. Terzopoulos and K. Fleischer. Modeling inelastic deformation: viscolelasticity, plasticity, fracture. In *Proc. Computer graphics and interactive techniques*, pages 269–278, 1988.

[18] M. Teschner, B. Heidelberger, M. Müller, and M. Gross. A versatile and robust model for geometrically complex deformable solids. In *Proc. Computer Graphics International*, pages 312–319, 2004.

[19] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets, and M. Gross. Optimized spatial hashing for collision detection of deformable objects. In *Proc. Vision, Modeling, Visualization*, pages 47–54, 2003.

[20] R. Weinstein, J. Teran, and R. Fedkiw. Dynamic Simulation of Articulated Rigid Bodies with Contact and Collision. *IEEE TVCG*, 12(3), 2006.

[21] A. Witkin, M. Gleicher, and W. Welch. Interactive dynamics. In *Proc. Symposium on Interactive 3D graphics*, pages 11–21, 1990.

[22] D. Zorin, P. Schröder, and W. Swelders. Interpolating subdivision for meshes with arbitrary topology. In *Proc. ACM SIGGRAPH*, pages 189–192, 1996.

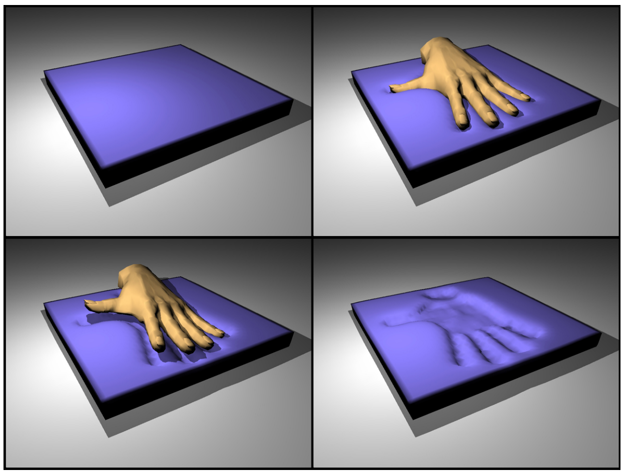

Figure 5: To test the plasticity properties of the deformation method, a hand is stamped into a plastically deformable plate. After the hand is removed an imprint is left in the plate.

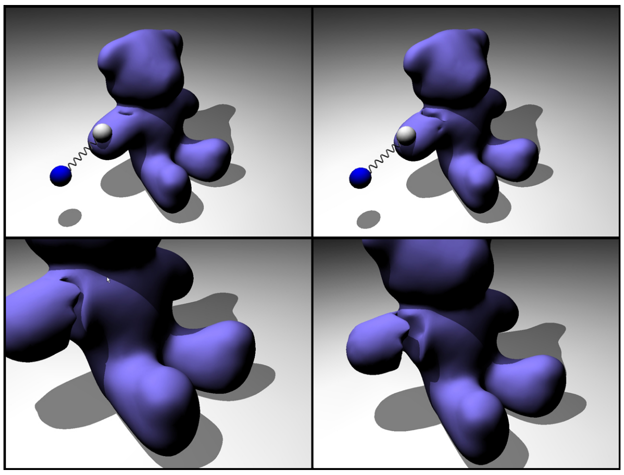

Figure 7: Consistent and watertight fracturing is guaranteed during the whole fracture process. In the top row fracturing starts on top of the arm whereas in the bottom row it starts at the lower front.

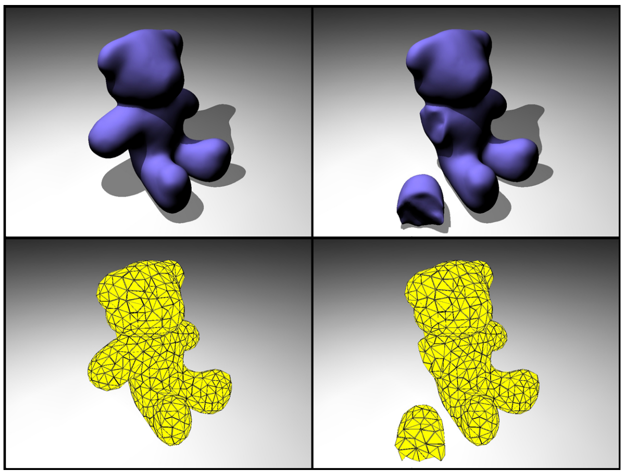

Figure 6: Upper row: A teddy with a smooth and watertight surface mesh (left). The arm was removed from the body and a closed surface was generated at the exposed areas. (right). Lower row: The volumetric mesh of the teddy. Constraints were dissolved at the fracture spot.

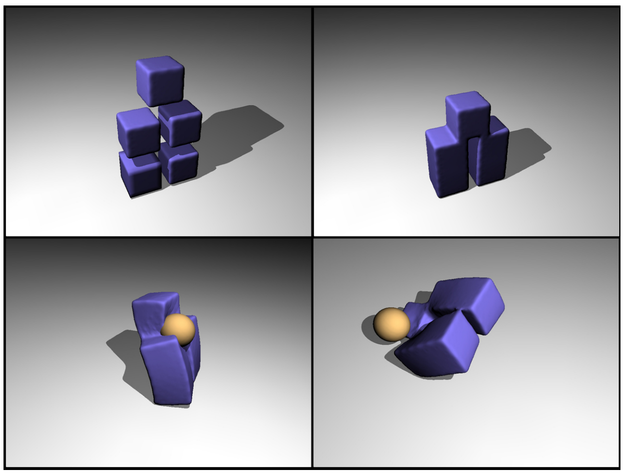

Figure 8: Clay cubes. 5 cubes are are merged together (top row). A ball hits the resulting object and deforms it plastically (bottom row).