Eurographics Symposium on Point-Based Graphics (2005, to appear)
M. Pauly, M. Zwicker, (Editors)

A Unified Lagrangian Approach to Solid-Fluid Animation

Richard Keiser[1] Bart Adams[2] Dominique Gasser[1] Paolo Bazzi[1] Philip Dutré[2] Markus Gross[1]

[1] ETH Zürich [2] Katholieke Universiteit Leuven

Abstract

We present a framework for physics-based animation of deforming solids and fluids. By merging the equations of solid mechanics with the Navier-Stokes equations using a particle-based Lagrangian approach, we are able to employ a unified method to animate both solids and fluids as well as phase transitions. Central to our framework is a hybrid implicit-explicit surface generation approach which is capable of representing fine surface detail as well as handling topological changes in interactive time for moderately complex objects. The generated surface is represented by oriented point samples which adapt to the new position of the particles by minimizing the potential energy of the surface subject to geometric constraints. We illustrate our algorithm on a variety of examples ranging from stiff elastic and plasto-elastic materials to fluids with variable viscosity.

1. Introduction

Realistic animation of physical phenomena has gained increasing importance in many fields of computer graphics, including the game and special effects industries. Typical examples include the animation of deformable solids and fluids. The phase transition between solids and fluids, i.e., melting and freezing, also plays an important role in special effects. A prominent example is the terminator sequence from the well-known feature film, where the metallic terminator is shattered, after which the individual pieces melt and fuse, before retaining the old shape and freezing to a solid.

A variety of methods exist for physically based animation. Mesh- and grid-based approaches such as mass-spring systems, Finite Element, Finite Difference or Finite Volume methods are still the most widespread, but mesh-free methods have recently become increasingly popular for the animation of elasto-plastic materials and fluids. Mesh-free methods or particle systems allow for easy implementation and have demonstrated to be computationally efficient [MCG03, MKN*04], which is especially important for games and interactive applications, but also for prototyping of animations. Furthermore, mesh-free methods can handle topological changes, which inherently occur for fluid-like materials, without the need for remeshing.

Mesh-free methods require the definition or extraction of an implicit or explicit surface for rendering. In the context of a unifying framework for solid-fluid animations the surface must be able to fulfill various requirements. Solid surfaces are often very detailed, while fluid surfaces are smooth due to surface tension. For phase transitions from solids to fluids the detail should disappear, while from fluids to solids the existing detail has to be preserved. Additionally, the surface should be temporally smooth, i.e. temporal aliasing such as popping artifacts should be avoided. Finally, topological changes such as splashes must be handled by the surface. We will show how to fulfill these requirements by dynamically maintaining a point-sampled surface wrapped around the particles.

1.1. Our Contributions

In this paper we propose a unified Lagrangian approach for both physics and surface animation of solids and fluids as well as phase transitions. The main challenges are the handling of locally different physical behavior during melting and freezing and the extraction of a surface which is able to represent detailed and smooth surfaces as well as handling topological changes. The main contributions are:

- By **combining the equations of solid mechanics with the Navier-Stokes equations using a Lagrangian approach**, we are able to employ a unified method to animate both fluids and solids as well as phase transitions.
- We present a **hybrid implicit-explicit surface generation approach** which dynamically constructs a point-sampled surface wrapped around the particles. Potentials are defined which guide the surface deformation. The generated surface fulfills aforementioned requirements. Most importantly, we are able to represent the fine surface detail required for solids, as well as the smooth surfaces of fluids.

- **Topological changes** are incorporated in a lightweight and efficient manner. Furthermore, we avoid the blending artifacts typically arising when handling topological changes using implicit functions.

1.2. Outline

We give an overview of related work in the next section. Sec. 3 explains the physics-based simulation framework. Next, we show how the surface is generated and animated in Sec. 4. A variety of results are given in Sec. 5. We conclude the paper with a discussion of results and an outlook on future work.

2. Background

Exhaustive work has been done in physically based animation. We provide an overview of the existing work that is most relevant to this paper.

2.1. Deformable Solids & Fluids

Terzopoulos and co-workers pioneered the field of physically based animation in their seminal paper [TPBF87]. They proposed to compute the dynamics of deformable models from the potential energy stored in the elastically deformed body using finite difference discretization. They extended their work to model plasticity and fracture [TF88]. In [TPF89], Terzopoulos et al. used a mass-spring system to model heating and melting of deformable objects, achieving a (local) phase transition from solids to fluids by simply varying the spring constant and finally removing a spring. Similar work was presented by Tonnesen [Ton91], who applied the heat equation to a particle system. Desbrun and Canni [DC96] introduced Smoothed Particle Hydrodynamics (SPH) [Mon92] to the graphics community. SPH was used by Müller et al. to derive an interactive fluid simulation algorithm from the Navier-Stokes equations [MCG03]. Stora et al. [SAC*99] simulated the flow of lava using SPH by coupling viscosity with temperature. Stam achieved a significant performance improvement for fluid animations using a semi-Lagrangian method for fluid convection and an implicit integrator for diffusion [Sta99]. Carlson and coworkers used an Eulerian grid-based fluid simulation method for melting, flowing and solidifying of objects [CMVT02]. Their method is capable of modelling different materials, ranging from rigid solids to fluids, by varying the viscosity. Recently, Carlson et al. presented an approach to simulate fluids as rigid bodies by adding an extra term to the Navier-Stokes equations. This way the interplay between rigid bodies and fluid could be computed efficiently using the same simulation solver for both [CMT04]. Goktekin et al. [GBO04] added elastic terms to the Navier-Stokes equations solving them using Eulerian methods, thus obtaining viscoelastic fluids that can model a variety of materials such as clay and pudding. Our work is similar to theirs in that we also combine fluid and solid characteristics. Merging of the solid mechanics with the Navier-Stokes equations allows us to animate materials from highly stiff elastic and plasto-elastic materials to fluids with low viscosity.

2.2. Surface Generation & Visualization

Desbrun and Canni model soft inelastic objects which split and merge by coating a set of skeletons using an implicit representation [DC95]. They extend their work by introducing active implicit surfaces [DCG98], which move according to a velocity field. The velocity field is chosen such that the surface is attracted to a geometric coating, but other terms such as surface tension and volume conservation are also applied.

State-of-the art methods in fluid simulation use level sets, introduced by Osher and Sethian [OS88], to render the fluid [FF01]. Level sets start with an implicit function which is evolved over time using a velocity field. This allows temporally smooth surface animation. However, level set evolution can suffer from severe volume loss, especially near detailed features such as splashes. As a solution, Enright et al. propose to combine level sets with surface particles [EMF02, ELF04].

The techniques discussed so far animate the surface by solving a PDE on a grid, followed by iso-surface extraction for rendering. Witkin and Heckbert defined constraints to keep surface particles on a moving implicit surface [WH94]. Surface particles adaptively sample the surface using a splitting and merging scheme. Szeliski and Tonnesen introduced oriented particles for surface modeling [ST92]. Additionally to long-range attraction and short-range repulsion forces, they define potentials which favor locally planar or locally spherical arrangements. Their particle system can handle geometric surfaces with arbitrary topology. Similar to them, we use surfels as oriented particles. Our explicit surface representation deforms by internal and external forces and geometric constraints, similar to active implicit surfaces [DCG98]. We derive the forces by minimizing a potential energy term which depends on both the surface and the physical particles. Besides gaining efficiency in computation and memory, we can exploit all advantages of explicit surfaces such as the simple representation of fine surface details.

2.3. Point-based Animation

Point-sampled surfaces have become popular the last few years for editing [ZPKG02], modelling [AD03, PKKG03] and rendering [ZPvG01, BK03, AA04]. Recently, mesh-free physics has been combined with point-sampled surfaces in so-called point-based animations [pba04]. Müller et al. introduced to the computer graphics community a mesh-free, continuum-mechanics-based model for animating elasto-plastic objects [MKN*04]. They showed how to embed a point-sampled surface into the simulation nodes. Furthermore, they propose a multi-representation approach, consisting of an implicit and a detailed representation. At places where topological changes occur, a blending between the detailed and the implicit representation is performed. Pauly

et al. [PKA*05] extended their work to account for fracture by introducing surface and volume sampling methods. Our work is based on the physics framework of [MKN*04]. While they are able to simulate highly viscous materials by increasing the plasticity, we show how to introduce the Navier-Stokes equations into their framework. This allows us to simulate fluids with varying viscosity and the transition from stiff elastic and plasto-elastic materials to fluids and vice versa. A surface generation algorithm which is suitable for both solids and fluids and supports melting and freezing completes our framework.

3. Physics Framework

From a high-level point of view, the mechanics for solids and fluids are quite similar. The main difference is that solids have restoring forces due to stresses, while an ideal (Newtonian) fluid stores no deformation energy [TF88]. We refer to [GBO04] for a broader discussion.

3.1. Governing Equations

Following [MSHG04], we can write the equations for an elastic solid as

$$\rho\frac{\partial^2\mathbf{u}}{\partial t^2} = \nabla\sigma^s(\mathbf{u}) + \mathbf{f}, \tag{1}$$

and for an incompressible Newtonian fluid as

$$\rho\frac{D\mathbf{v}}{Dt} = \nabla\sigma^f(\mathbf{v}) + \mathbf{f}. \tag{2}$$

Both Eq. 1 and Eq. 2 describe the change in momentum which is equal to internal force density fields due to stresses and body force density fields $\mathbf{f}$, where ρ denotes the density, $\mathbf{u}$ the displacement from the material coordinates, $\mathbf{v}$ the velocity and σ the stress tensor. $\frac{D\mathbf{v}}{Dt}$ is the material derivative of the velocity field. Conservation of mass is represented as

$$\frac{\partial\rho}{\partial t} + \nabla(\rho\mathbf{v}) = 0. \tag{3}$$

These equations can be simplified when using a particle-based Lagrangian approach. By assigning each particle p_i a constant mass m_{p_i}, mass conservation is guaranteed and Eq. 3 can be omitted. Furthermore, because the particles move with the fluid, the material derivative $\frac{D\mathbf{v}}{Dt}$ of the velocity field is equal to the time derivative of the velocity of the particles [MSHG04]. For the same reason the partial derivative of the displacement field is equal to the time derivative of the displacement. Using that $\mathbf{v} = \frac{d\mathbf{u}}{dt}$, Eq. 1 and 2 can be written as

$$\rho\frac{d^2\mathbf{u}}{dt^2} = \nabla\sigma^s(\mathbf{u}) + \mathbf{f}, \tag{4}$$

$$\rho\frac{d^2\mathbf{u}}{dt^2} = \nabla\sigma^f(\mathbf{v}) + \mathbf{f}. \tag{5}$$

Finally, we merge Eq. 4 and Eq. 5 as follows

$$\rho\frac{d^2\mathbf{u}}{dt^2} = \nabla\sigma(\mathbf{u},\mathbf{v}) + \mathbf{f}, \tag{6}$$

where $\sigma(\mathbf{u},\mathbf{v}) = \sigma^s(\mathbf{u}) + \sigma^f(\mathbf{v})$ is the sum of the elastic, viscous and pressure stress.

Note that although $\mathbf{f}$ is a force density field (force per unit volume), we will denote it as force in the following subsections for simplicity.

3.2. Internal Forces

We will briefly summarize how to solve Eq. 6 using a point collocation scheme. We refer to [MCG03] and [MKN*04] for details.

Elastic Force. As shown in [MKN*04], the elastic force can be computed via the strain energy density

$$U = \frac{1}{2}(\varepsilon^s \cdot \sigma^s),$$

where ε^s is the strain and σ^s the elastic stress. We use a linear stress-strain relationship (Hooke's law), i.e., $\sigma^s = \mathbf{C}\varepsilon^s$. For an isotropic material, $\mathbf{C}$ depends only on Young's Modulus E and Poisson's Ratio υ. The strain is measured using the quadratic Green-Saint-Venant strain tensor

$$\varepsilon^s = \frac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^T + \nabla\mathbf{u}\nabla\mathbf{u}^T),$$

where $\nabla\mathbf{u}$ is approximated using Moving Least Squares (MLS). We refer the reader to [LS81] for more details about MLS.

To model plasticity, we follow [OBH02] and store for each particle a plastic strain tensor $\varepsilon^{plast}_{p_i}$, which is updated in every time step depending on the plastic yield and plastic creep material constants γ^{yield} and γ^{creep}, respectively. The difference between the measured strain $\varepsilon^s_{p_i}$ and the plastic strain, i.e., $\widetilde{\varepsilon}^s_{p_i} = \varepsilon^s_{p_i} - \varepsilon^{plast}_{p_i}$, is used to compute the elastic force $\mathbf{f}^{elastic}_{p_i}$ acting on a particle p_i as the directional derivative of the strain energy density

$$\mathbf{f}^{elastic}_{p_i} = -\nabla_{\mathbf{u}_i}U = -\frac{1}{2}\nabla_{\mathbf{u}_i}(\widetilde{\varepsilon}^s_{p_i}\cdot\mathbf{C}\widetilde{\varepsilon}^s_{p_i}) = -\sigma^s_{p_i}\nabla_{\mathbf{u}_i}\widetilde{\varepsilon}^s_{p_i}.$$

Viscosity and Pressure Force. The fluid stress tensor $\sigma^f(\mathbf{v})$ is composed of the viscosity stress and the pressure stress

$$\sigma^f(\mathbf{v}) = \mu\nabla\mathbf{v} - \varphi\mathbf{I},$$

where μ is a viscosity constant and φ the scalar pressure. φ is computed as $\varphi = k^{gas}(\rho - \rho_0)$, where ρ_0 is the rest density and k^{gas} is a gas constant [DC96]. Note that for melting and freezing, μ can locally differ and is subject to change.

The viscosity and pressure force are derived using SPH, similar to [MCG03]. For an introduction to SPH we refer to [Mon92] and [DC96]. Application of the SPH rule yields the following forces acting on a particle p_i

$$\mathbf{f}^{visc}_{p_i} = \sum_{p_j} m_{p_j}\frac{\mu_{p_i}+\mu_{p_j}}{2}\frac{\mathbf{v}_{p_j}-\mathbf{v}_{p_i}}{\rho_{p_j}}\nabla^2 W^{visc}(\mathbf{x}_{p_i}-\mathbf{x}_{p_j},h_p)$$

$$\mathbf{f}^{pressure}_{p_i} = -\sum_{p_j} m_{p_j}\frac{\varphi_{p_i}+\varphi_{p_j}}{2\rho_{p_j}}\nabla W^{pressure}(\mathbf{x}_{p_i}-\mathbf{x}_{p_j},h_p)$$

where h_p is the support radius of a particle and p_j are the neighboring particles of p_i within h_p. We use the same kernel functions W^{visc} and $W^{pressure}$ as in [MCG03].

Surface Tension Force. Additionally to the above mentioned forces, cohesive forces between liquid molecules keep the fluid together. We follow [MCG03] and define a smoothed color field as

$$c(\mathbf{x}) = \sum_{p_j} m_{p_j} \frac{1}{\rho_{p_j}} W(\mathbf{x} - \mathbf{x}_{p_j}, h_p).$$

The gradient is denoted as $\mathbf{n}_c(\mathbf{x}) = \nabla c(\mathbf{x})$. The surface tension force acting near the surface is computed as

$$\mathbf{f}_{p_i}^{tension} = -k^{tension} \nabla^2 c(\mathbf{x}_{p_i}) \frac{\mathbf{n}_c(\mathbf{x}_{p_i})}{\|\mathbf{n}_c(\mathbf{x}_{p_i})\|},$$

where $k^{tension}$ is a material specific constant.

3.3. Melting & Freezing

In our setting, a material can be defined by the following main properties: Stiffness (Young's Modulus E), compressibility (Poisson's Ratio υ), plasticity (γ^{yield} and γ^{creep}), viscosity (μ) and cohesion between (surface) particles (surface tension $k^{tension}$). As long as these values are chosen in a (physically) reasonable range, they can be arbitrarily combined. Therefore, also materials which do not exist in reality can be created, as e.g. elastic fluids.

For melting and freezing, the material parameters need to change together with the changing aggregation state. To allow local changes, each particle stores its own material parameters. If the material melts from solid to fluid, the stiffness and viscosity decrease, while cohesion at the surface and plasticity increase, and vice versa for freezing. The user can set the parameters described above for two materials. We assign to each particle p_i a scalar value T_{p_i} which is used to interpolate between the two materials. We call T_{p_i} the temperature of p_i. Assuming a scalar material parameter a was set by the user to a^{min} and a^{max} for T^{min} and T^{max}, respectively (see also Sec. 5), a_{p_i} is computed using linear interpolation.

Heat transfer between particles is modelled by solving the heat equation $\frac{dT}{dt} = k^{heat} \nabla^2 T$ using SPH, similar to [SAC*99]

$$\nabla^2 T_{p_i} = \sum_{p_j} m_j \frac{T_{p_j} - T_{p_i}}{\rho_{p_j}} \nabla^2 W^{visc}(\mathbf{x}_{p_i} - \mathbf{x}_{p_j}, h_p),$$

where we use the same kernel as for computing the viscosity.

3.4. Particle Animation

Depending on the application, we either sample the volume of an object or use a source that creates a stream of particles. Mass m_{p_i}, density ρ_{p_i} and volume v_{p_i} are initialized as is done in SPH, we refer to [DC96] and [MKN*04] for details.

In each iteration, we first compute for each particle p_j its particle neighborhood using a hash grid as a search data structure [THM*03]. The force density fields are then computed as described in the previous sections, yielding the final force density field $\mathbf{f}_{p_i} = \mathbf{f}_{p_i}^{elastic} + \mathbf{f}_{p_i}^{visc} + \mathbf{f}_{p_i}^{pressure} + \mathbf{f}_{p_i}^{tension}$ for a particle p_i. Finally, the acceleration $\mathbf{a}_{p_i}$ is

$$\mathbf{a}_{p_i} = \frac{d^2 \mathbf{u}_{p_i}}{dt^2} = \frac{\mathbf{f}_{p_i}}{\rho_{p_i}}.$$

For integration we use the Leap-Frog scheme, which showed to be both efficient and stable for our animations. After an iteration step, we update the material coordinates as described in [MKN*04].

4. Surface Animation

The surface of the animated object is explicitly represented using oriented point samples, called *surfels*. To be able to animate and deform the surface, each surfel keeps a set of neighboring particles and a set of neighboring surfels (Sec. 4.1). After performing an animation step of the particles, we get an estimation of the new surface by carrying the surfels along with the neighboring particles (Sec. 4.2.1). Similar to active contour models [KWT88], the surface adapts to the new position of the particles by minimizing a potential energy term (Sec. 4.2.2) while fulfilling geometric constraints (Sec. 4.2.3). The surface resolution is adapted using a simple resampling scheme that ensures a hole-free surface (Sec. 4.3). Disjoint components of the surface are identified and the particles and surfels are separated accordingly (Sec. 4.4.1). We detect intersections of separated surfaces and merge the intersecting surface parts (Sec. 4.4.2). Finally, we show how the surface can be blended smoothly between detailed solid and smooth fluid surfaces (Sec. 4.5).

4.1. Surfel Neighborhoods

Each surfel s_i caches a set P_i of neighboring particles and a set S_i of neighboring surfels of the same object (Fig. 1). The neighboring particles are used to estimate the displacement of the surfels after an animation step. From the surfel neighborhood, forces are computed to update the surface as described below.

Particle Neighborhood. The neighboring particles are computed in a pre-animation step. We reuse the hash grid data structure (Sec. 3.4) of the particles to determine the particles p_j which are within the (particle) support radius h_s^p of a surfel s_i (we choose $h_s^p = 2h_p$, where h_p is the support radius of the particles, see Sec. 3.2). Furthermore, we store a weight $\omega^{h_s^p}_{\mathbf{x}_{s_i}, \mathbf{x}_{p_j}}$ for each neighbor particle which is computed using a smoothly decaying weight function depending on the distance to s_i. We use the following compactly supported polynomial function with support radius h

$$\omega^h_{\mathbf{x},\mathbf{y}} = \omega(\mathbf{r}) = \begin{cases} (1-\mathbf{r}^2)^3 & \|\mathbf{r}\| \leq 1 \\ 0 & \|\mathbf{r}\| > 1, \end{cases} \tag{7}$$

where $\mathbf{r} = (\mathbf{x} - \mathbf{y})/h$.

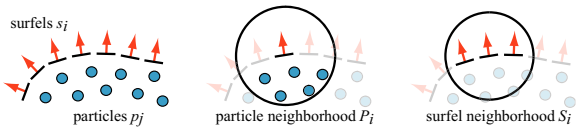

Figure 1: *Left: The surfels s_i are wrapped around the particles p_j. Middle and Right: Each surfel s_i stores a particle neighborhood P_i and a surfel neighborhood S_i.*

Surfel Neighborhood. Additional to the particle neighborhood, a surfel s_i also caches the neighboring surfels within the (surfel) support radius h_s^s (we choose $h_s^s = h_p$). We use again a hash grid as a search data structure. This neighborhood is recomputed after the surface has been carried along with the particles (see Sec. 4.2.1). Next, during surface update (see Sec. 4.2.2), the position of the surfels change frequently and therefore this surfel neighborhood needs to be updated quite often. Instead of doing an update of the search data structure and expensive recomputation of the neighbors each time the position of a surfel changes, we use the following update scheme.

Assume the surfel s_i changed its position. Let the new neighborhood be the (initially empty) set S_i'. First, all surfels in S_i with an Euclidean distance smaller than h_s^s are added to S_i'. Next, we iterate through all neighbors of the neighbors in S_i and add them to S_i' if their Euclidean distance is smaller than h_s^s. By tagging neighbors which were already visited, this update procedure can be performed very fast. Note that not necessarily all neighbors are found, however, as the surfel position does not change significantly during the surface deformation, this update procedure showed to be sufficient. Some of the following algorithms will only use the k nearest neighbors from the neighbor set S_i (we use $k = 10$). We denote this subset as S_i^{sub}.

4.2. Surface Deformation

Surface animation is performed after animating the particle system. The deformation is applied in two steps. First, the surfels are carried along with the particles (Sec. 4.2.1). Next, they are updated to reflect the new particle positions by minimizing the surface energy, where constraints restrict the possible movements (Sec. 4.2.2).

4.2.1. Surface Displacement

The displaced surfel position $\mathbf{x}_{s_i}^{t+1}$ is computed using a first order accurate approximation of the displacements $\mathbf{u}_{p_j}$ of the neighboring particles p_j as follows [MKN*04]

$$\mathbf{x}_{s_i}^{t+1} = \mathbf{x}_{s_i}^{t} + \sum_{p_j \in P_i} \overline{\omega}_{\mathbf{x}_{s_i},\mathbf{x}_{p_j}}^{h_s^p} (\mathbf{u}_{p_j} + \nabla_{\mathbf{u}}^T \mathbf{u}_{p_j} (\mathbf{x}_{p_j}^t - \mathbf{x}_{s_i}^t)),$$

where $\overline{\omega}_{\mathbf{x},\mathbf{y}}^{h_s^p} = \omega_{\mathbf{x},\mathbf{y}}^{h_s^p} / \sum_{\mathbf{y}} \omega_{\mathbf{x},\mathbf{y}}^{h_s^p}$, and $\omega_{\mathbf{x},\mathbf{y}}^{h_s^p}$ is the weighting function defined in Eq. 7. We reuse the MLS approximation of $\nabla_{\mathbf{u}} \mathbf{u}_{p_j}$, which is computed for deriving the elastic force as described in Sec. 3.2. Similar to [PKKG03], both the surfel center and its tangent axes are deformed, yielding the deformed surfel normal $\mathbf{n}_{s_i}^{t+1}$. For brevity we will omit the index t hereafter.

4.2.2. Surface Update

After carrying the surface along with the particles, it is deformed under the action of surface forces, similar to balloons [Coh91]. The forces are derived by minimizing the potential energy of the surface. The potential energy is composed of external potentials which depend on the particles and internal potentials which depend on the surfels. We derive an implicit and an attracting potential such that the energy is minimized when the surfels are attracted to an implicit surface and to the particles, respectively. Minimizing the internal potentials, consisting of the smoothing potential and the repulsion potential, yields a locally smooth and uniformly sampled surface. From the potential energy we derive forces acting on the surfels. While the derived forces from the implicit, attracting and smoothing potential act in normal direction, the repulsion force is applied in tangential direction. Note that these forces are defined for deforming the surface only, and are therefore independent of the physics framework described in Sec. 3.

Implicit Potential. Similar to [DCG98], we define a purely geometric implicit coating of the particles that attracks our explicit surface. Each particle p_j defines a field function. A potential field is defined by computing the weighted sum of the field functions at an arbitrary position in space [Bli82]. We use the color field $c(\mathbf{x})$ described in Sec. 3.2 as a potential field. We define the implicit potential $\phi_{s_i}^{impl}$ as the squared distance from the position $\mathbf{x}_{s_i}$ of a surfel s_i to its projected position $\mathbf{x}_{s_i}^{impl}$ on an iso-level I of the potential field, i.e.,

$$\phi_{s_i}^{impl} = \frac{1}{2} (\mathbf{x}_{s_i}^{impl} - \mathbf{x}_{s_i})^2.$$

$\mathbf{x}_{s_i}^{impl}$ is computed similar to [MKN*04]. The normal $\mathbf{n}_{s_i}^{impl}$ at $\mathbf{x}_{s_i}^{impl}$ is equal to the gradient of the color field , i.e. $\mathbf{n}_{s_i}^{impl} = \nabla c(\mathbf{x}_{s_i}^{impl})$.

Attracting Potential. Generally, we want the surface to coat the particles as tight as possible. Therefore we define a potential such that the surfels are attracted to the particles. We define the attracting potential ϕ^{attr} as the sum of weighted squared distances from a surfel s_i to its neighboring particles $p_j \in P_i$, i.e.,

$$\phi^{attr} = \frac{1}{2} \sum_{p_j \in P_i} (\mathbf{x}_{p_j} - \mathbf{x}_{s_i})^2 \omega_{\mathbf{x}_{s_i},\mathbf{x}_{p_j}}^{h_s^p},$$

where $\omega_{\mathbf{x}_{s_i},\mathbf{x}_{p_j}}^{h_s^p}$ is the weighting function defined in Eq. 7.

Smoothing Potential. Minimization of the implicit and attracting potential yields the well-known blob artifacts due to the discretization of the volume with particles (see Fig. 2). Therefore, we derive a potential ϕ^{smooth} which yields a smooth surface. We compute a weighted Least Squares (LS)

plane through the neighbors $s_j \in S_i$ of a surfel s_i using Prinicpal Component Analysis (see e.g. [PGK02]). We restrict the neighborhood to neighbor surfels whose normals have an angle to the normal of s_i smaller than a threshold (we choose $\pi/4$). The LS plane is given by the weighted mean $\mathbf{x}_{s_i}^{mean}$ and the LS plane normal $\mathbf{n}_{s_i}^{smooth}$. The projection of $\mathbf{x}_{s_i}$ onto the LS plane gives the position $\mathbf{x}_{s_i}^{proj} = \mathbf{x}_{s_i} + (\mathbf{n}_{s_i}^{smooth} \cdot (\mathbf{x}_{s_i}^{mean} - \mathbf{x}_{s_i}))\mathbf{n}_{s_i}^{smooth}$. Finally, we define the smoothing potential $\phi_{s_i}^{smooth}$ as the squared distance from s_i to the LS plane, i.e.,

$$\phi_{s_i}^{smooth} = \frac{1}{2}(\mathbf{x}_{s_i}^{proj} - \mathbf{x}_{s_i})^2.$$

Repulsion Potential. To achieve a locally uniform distribution, we define a repulsion potential $\phi_{s_i}^{rep}$ similar to [PGK02]. It is minimal when the neighboring particles $s_j \in S_i^{sub}$ of a surfel s_i have a distance h_s^s to s_i:

$$\phi_{s_i}^{rep} = \frac{1}{2} \sum_{s_j \in S_i^{sub}} (h_s^s - \|\mathbf{d}_{s_i,s_j}\|)^2,$$

where $\mathbf{d}_{s_i,s_j} = \mathbf{x}_{s_i} - \mathbf{x}_{s_j}$.

Minimizing Forces. The potential energy of the surfels s_i is minimized by applying forces which are derived from the energy fields:

$$\begin{aligned}
\mathbf{f}_{s_i}^{impl} &= -\nabla_{\mathbf{x}_{s_i}} \phi_{s_i}^{impl} = \mathbf{x}_{s_i}^{impl} - \mathbf{x}_{s_i}, \\
\mathbf{f}_{s_i}^{attr} &= -\nabla_{\mathbf{x}_{s_i}} \phi_{s_i}^{attr} = \sum_{p_j \in P_i} (\omega(\mathbf{r})\mathbf{d}_{p_j,s_i} - \frac{1}{2}\omega(\mathbf{r})'\mathbf{d}_{p_j,s_i}^2), \\
\mathbf{f}_{s_i}^{smooth} &= -\nabla_{\mathbf{x}_{s_i}} \phi_{s_i}^{smooth} = \mathbf{x}_{s_i}^{proj} - \mathbf{x}_{s_i}, \\
\mathbf{f}_{s_i}^{rep} &= -\nabla_{\mathbf{x}_{s_i}} \phi_{s_i}^{rep} = \sum_{s_j \in S_i^{sub}} \frac{(h_s^s - \|\mathbf{d}_{s_i,s_j}\|)}{\|\mathbf{d}_{s_i,s_j}\|} \mathbf{d}_{s_i,s_j},
\end{aligned}$$

where $\mathbf{d}_{p_j,s_i} = \mathbf{x}_{p_j} - \mathbf{x}_{s_i}$ and $\mathbf{r} = \mathbf{d}_{p_j,s_i}/h_s^p$.

We restrict the repulsion force to only act in tangential direction and the other forces to only act in normal direction. We do this by computing a new surfel normal $\mathbf{n}'_{s_i}$ as the average of the implicit gradient $\mathbf{n}_{s_i}^{impl}$ and the smoothed normal $\mathbf{n}_{s_i}^{smooth}$, i.e., $\mathbf{n}'_{s_i} = (k^{impl}\mathbf{n}_{s_i}^{impl} + k^{smooth}\mathbf{n}_{s_i}^{smooth})/\|k^{impl}\mathbf{n}_{s_i}^{impl} + k^{smooth}\mathbf{n}_{s_i}^{smooth}\|$. The tangential force $\mathbf{f}_{s_i}^{tan}$ is then defined as the projection of the repulsion force $\mathbf{f}_{s_i}^{rep}$ on the tangent plane defined by the new surfel normal $\mathbf{n}'_{s_i}$:

$$\mathbf{f}_{s_i}^{tan} = \mathbf{f}_{s_i}^{rep} - (\mathbf{n}'_{s_i} \cdot \mathbf{f}_{s_i}^{rep})\mathbf{n}'_{s_i}.$$

The force in normal direction $\mathbf{f}_{s_i}^{normal}$ is computed as the projection of the implicit, attracting, smoothing and optional external forces:

$$\mathbf{f}_{s_i}^{normal} = (\mathbf{n}'_{s_i} \cdot \mathbf{f}_{s_i}^{sum})\mathbf{n}'_{s_i},$$

where $\mathbf{f}_{s_i}^{sum} = k^{impl}\mathbf{f}_{s_i}^{impl} + k^{attr}\mathbf{f}_{s_i}^{attr} + k^{smooth}\mathbf{f}_{s_i}^{smooth} + k^{ext}\mathbf{f}_{s_i}^{ext}$. The weights are user defined parameters, where there is always a tradeoff between smoothness and closeness of the surface to the particles, see Fig. 2.

Figure 2: *Illustration of the impact of the implicit, smoothing and attracting force. Left: Implicit force only ($k^{implicit} = 0.2$). Middle: Implicit and smoothing force ($k^{implicit} = 0.2$, $k^{smooth} = 0.6$). Right: Implicit, smoothing and attracting force ($k^{implicit} = 0.2$, $k^{smooth} = 0.6$, $k^{attr} = 0.1$).*

Integration. Finally, we get the new surfel position using Euler integration, i.e. $\mathbf{x}'_{s_i} = \mathbf{x}_{s_i} + \alpha(\mathbf{f}_{s_i}^{normal} + \mathbf{f}_{s_i}^{tan})$, where $0 < \alpha \leq 1$ is a scaling factor. Note that applying the forces along the new surfel normal vector $\mathbf{n}'_{s_i}$ can be seen as a semi-explicit Euler integration, yielding a very stable integration if all weights are smaller than one, as $\mathbf{n}'_{s_i}$ is smooth (assuming that the iso-value I is chosen such that the iso-surface is smooth). To avoid oscillations, we damp the system by multiplying α at each iteration with a damping constant. The integration is repeated until the maximal displaced distance of all surfels of a surface component is below a threshold.

4.2.3. Constraints

Constraints restrict the position and movement of the surface. We propose to use two constraints which are applied in the following order:

Implicit Constraint. Optionally, we restrict a surfel to be within a minimal iso-level (see the implicit force description in the previous section), which is useful for splitting of solid objects as will be shown in Sec. 4.4.1. If the color field value $c(\mathbf{x}_{s_i})$ of a surfel s_i is smaller than a user defined minimal iso-value I^{min}, it is projected onto I^{min}. This can also be seen as a maximum allowed distance to the particles.

External Constraints. External constraints are used in our case to restrict the surface to a certain area, for example when doing collision detection with an obstacle (e.g. a glass). During collision detection we only consider the particles. Therefore it can happen that surfels still penetrate the obstacle although the particles do not. Surfels which penetrate the obstacle are projected back onto the colliding object. This projection is performed (if necessary) after each surface deformation iteration. Because of the smoothing potential the surface remains smooth at the border of the projected surfels.

4.3. Resampling

Resampling is important to adapt the number of surfels when the surface is stretched or compressed. Each time before a force (Sec. 4.2.2) is computed, we test if the number of neighbors $|S_i|$ of a surfel s_i is smaller or larger than a minimum or maximum threshold, respectively. In the former case the number of missing neighbors are randomly distributed around s_i, where the new surfels inherit the neighbors of s_i. A neighbor update is then performed for the new surfels and

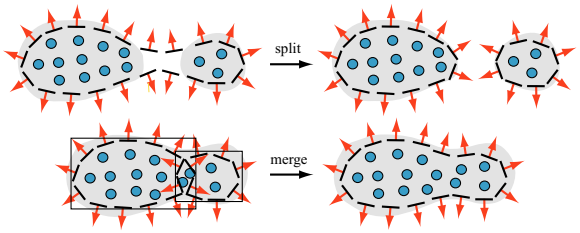

Figure 3: *Top row: When a surfel is outside the minimal isovalue it is projected onto it. Flood filling over all the surfels results in the construction of separated components. Bottom row: Merging is performed by detecting and removing colliding surfels from different components.*

the neighbors of s_i. If the number of neighbors is too large, we delete the surfel and remove it from all its neighbors. We make the neighbor thresholds dependent on the radius of a surfel. Assume that all surfels have the same radius r_s and that the surfels are distributed uniformly, i.e. they lie on a hexagonal grid. Then a distance between two neighbors on this grid equal or smaller than r_s guarantees a hole-free surface. Therefore, we choose $6h_s^s/r_s$ as the minimum threshold and $9h_s^s/r_s$ as the maximum threshold.

This resampling scheme is also used to create an initial surface enclosing the particles, e.g. when using a source. At each particle position we create a surfel. These surfels are projected onto the implicit surface by setting the surface constants $k^{impl} = 1$ and $k^{attr} = k^{smooth} = 0$. Performing a few steps of surface deformation with resampling yields one or more closed surface components. Finally, the surface constants are reset to the user values. Applying the surface deformation scheme yields the final surface.

4.4. Topological Changes

By recomputing the neighborhood and using the forces and constraints described above, the surface implicitly handles topological changes such as disjoint components and merging (see Fig. 3). The implicit constraint ensures that a (locally) stretched surface is always split even if the implicit force weight is chosen to be small. Furthermore, two intersecting surface components are merged automatically by recomputing the surfel neighborhood and the applied forces. A well known problem of handling topological changes implicitly is that two surface components will blend rather than collide, i.e., they are merged before they intersect which results in considerable artifacts. Therefore, we suggest a method to detect disjoint surface components similar to the blending graph described in [DC95]. These are then handled as separated objects. We show how intersecting separated objects are merged.

4.4.1. Splitting

To detect two disjoint components of a surface, we perform a flood-fill over all surfels using restricted surfel neighborhoods S_i^{rest} after having deformed the surface. A neighbor $s_j \in S_i^{sub}$ is added to S_i^{rest} if the angle between its normal and the normal of s_i is smaller than a threshold (we choose $\pi/4$). Starting with an arbitrary surfel s_i, we add s_i and the surfels in S_i^{rest} to a set S^{sep}. The neighbors of the restricted neighborhood of the surfels in S_i^{rest} are then added recursively to S^{sep}, until no neighbors are left. This procedure is repeated (with new sets) as long as there are surfels which do not belong to a set yet. By tagging surfels which belong to a set already, the detection can be done in linear time to the number of surfels assuming a constant maximum number of neighbors.

After separating the surfels we assign the particles to the appropriate set by performing an inside/outside test similar to [PKKG03]. A particle is added to a set if it is inside the surface represented by the surfels. Each set then builds a separated surface component. The surfel neighbors S_i and the particle neighbors P_i of a surfel s_i are always computed from the surfel and particle set of its separated component.

4.4.2. Merging

When two disjoint components intersect they either need to be merged or contact handling has to separate them. For merging we require that not only the surfaces intersect, but also that at least one particle is inside the other surface. This guarantees that the surfaces are merged smoothly.

We first compute a bounding box for each object surface part (see Fig. 3). From two intersecting bounding boxes, the colliding particles can be efficiently computed [KMH*04]. If we find a set of colliding particles, we also compute the colliding surfels which are deleted before we merge the two separated objects again. The surfel neighborhoods are then recomputed and the separated objects merge smoothly through the acting surface forces. Note that this way we avoid the unnatural blending typically arising when using the implicit function for merging, i.e., we avoid the blending of two separated surface parts before they intersect.

When merging is not appropriate, collision response forces could be applied to separate the colliding surface components, similar to [KMH*04].

4.5. Blending Between Solids and Fluids

While solids might have a very detailed surface, fluid surfaces are usually rather smooth. The particles account for this by surface tension. However, to smooth the surface this is not sufficient. Assume we start with a highly detailed solid which melts. In this case we expect the detail to disappear. On the other hand if we freeze a fluid, the existing detail should be preserved.

If we only apply the surfel displacement according to the particles as described in Sec. 4.2.1, all the detail is preserved, but if we additionaly update the surface using the potential fields as described in Sec. 4.2.2, the detail vanishes and the surface huddles against the particles. To get a smooth transition between solids and fluids, we perform both approaches and blend between them, similar to [MKN*04].

Assume that after particle animation, a surfel s_i is displaced to the position $\mathbf{x}_{s_i}$. Let $\mathbf{x}'_{s_i}$ be the positions afte applying the surface deformation forces and constraints. We get the blended position $\mathbf{x}_{s_i}^{blend}$ and normal $\mathbf{n}_{s_i}^{blend}$ by simple interpolation: $\mathbf{x}_{s_i}^{blend} = (1-\beta_i)\mathbf{x}_{s_i} + \beta_i \mathbf{x}'_{s_i}$ and $\mathbf{n}_{s_i}^{blend} = ((1-\beta_i)\mathbf{n}_{s_i} + \beta_i \mathbf{n}'_{s_i})/\|(1-\beta_i)\mathbf{n}_{s_i} + \beta_i \mathbf{n}'_{s_i}\|$.

For melting and freezing, we use the temperature T_{s_i} as a blending factor (see Sec. 3.3). The temperature of a surfel is approximated from the neighboring particles, i.e., $T_{s_i} = \sum_{p_j \in P_i} \overline{\omega}^{h_s^p}_{\mathbf{x}_{s_i},\mathbf{x}_{p_j}} T_{p_j}$. The normalized temperature is then used as a transition factor, i.e., $\beta_i = (T_{s_i} - T^{min})/(T^{max} - T^{min})$, where T^{max} and T^{min} are the temperature thresholds described in Sec. 3.3.

5. Results

We tested our physics framework and the surface generation with a variety of examples. Fig. 4 shows the simulation of a pure fluid, i.e., Young's Modulus E is set to 0 and viscosity μ to 5. The surface is smooth and encloses the particles tightly. As also shown in the accompanying video, our surface is able to handle all topological changes like splitting, merging and self-intersections of the surface. Merging of disjoint surface components prevents from blending two components before they intersect (see Sec. 4.4.2), resulting in less artifacts compared to implicit merging.

The example shown in Fig. 5 exploits the possibilities of our physics framework with the incorporated Navier-Stokes equations. It shows a quicksilver-like fluid which is poured into a glass. While it is flowing, we decrease the temperature of all particles within 10 iterations from T^{max} to T^{min}. Therefore, the fluid freezes to an elastic solid with stiffness $E = 5e5Nm^{-2}$. At the same time we remove the glass. The freezed fluid elastically bounces onto the ground where it splits.

Finally, different melting behaviors are shown in Fig. 6 and 7 (images taken at corresponding timesteps). We start with an elastic bust of the Nefertitis (57k surfels) which is dropped onto a heated box. When a particle collides with the box, its temperature increases with a constant value and diffuses to the other particles. The temperature locally affects the stiffness, plasticity, viscosity and surface tension of the object as described in Sec. 3.3. Therefore, upon collision the model starts melting. Depending on the heating transfer parameters, the temperature of the heating box and the maximal viscosity, the model shows different melting behaviors. In Fig. 6, the box has a temperature of $8000°C$ and we choose $k^{heat} = 0.3$. The particles heat up very quickly, resulting in a splashing fluid with low viscosity $\mu = 5$. In the animation of Fig. 7 the box's temperature is set to $4000°C$ and k^{heat} is set to 0.04. The model first bounces elastically and then slowly melts to a viscous fluid ($\mu = 10$). Note that the surface detail is still preserved even though large parts of the model are already liquid. The phase state of the Nefertitis is color coded from blue for solid to red for fluid.

Animation	Particles/Surfels	Physics	Surface	fps
Fluid	3k/3.4k	0.13 s	1.3 s	0.7
Freezing	2.4k/3.4k	0.4 s	1.2 s	0.5
Melting 1	3.9k/60k	3.2 s	22 s	0.03
Melting 2	3.9k/56k	3.1 s	20 s	0.03

Table 1: *Average timings of our system running on a 3 GHz Pentium 4 with a GeForce FX GPU. Timings are shown for one physics animation step and one surface deformation iteration step, followed by the resulting frame rate for all the sequences presented in this paper.*

In our examples, the physics animation runs in interactive time with 2400 to 3900 particles, see Table 1. The performance of the surface generation algorithm depends on the surface resolution and the surfel neighborhood radius h_s^s. For the fluid example shown in Fig. 4 (average number of 3.4k surfels) our algorithm needs on average 1.3s per iteration step. For the same example with smaller neighborhood (average number of 1.2k surfels) the animation runs with 1.5 fps. We use a low resolution surface for interactive animations and prototyping and increase the resolution for making production animations. Interactive rendering is achieved using surface splatting on the GPU [BK03]. The pictures in the paper and the video were created with the open-source renderer POV-Ray (http://www.povray.org), which we modified for raytracing point-sampled objects [AA04].

6. Conclusion & Future Work

Merging the continuum mechanics approach for solids presented in [MKN*04] with the Navier-Stokes equations proved to be very stable, and indeed did not affect the stability of the pure solid mechanics approach. It allows to model a variety of materials which could not be simulated before, like elastic and freezing fluids, and solids which melt to splashing fluids. Melting and freezing are modelled by simply changing the material parameters.

Our surface generation approach fits our needs to represent both highly detailed solids and smooth fluids with rapidly changing topology. Surfels showed to be suitable as an explicit representation because no mesh needs to be maintained. However, there are limitations and possible extensions which we plan to address in future work:

- We are able to avoid blending artifacts by separating disjoint surface components and merging them when two components intersect. However, these artifacts still occur when surface parts come close which have not been split. Avoiding this is a very difficult problem, that includes the detection of self-intersections.
- To represent small drops in a strongly splashing fluid a prohibitive high number of surfels with small radii are needed. In the future we want to adapt the density of the surfels depending on whether they lie on a flat or on a highly curved surface.

- Volume preservation is a well-known problem in SPH based methods. Furthermore, also our surface generation approach is not volume preserving. This could be improved by adding an additional force which lets the volume shrink or grow, similar to the volume preservation constraint suggested in [DCG98].
- So far we use a global termination criterion for the iterative integration of the surface forces for surface deformation (see Sec. 4.2.2). Because the forces are defined locally, efficiency could be improved significantly by computing and integrating the forces only for surfels s_i whose position changed and for the neighbors of s_i.
- In the future we want to allow particles to adaptively split and merge similar to [DC99]. This poses a challenge for the surface animation as the surface should not be affected by the particle resampling.
- The implicit constraint ensures that largely stretched surfaces of solids split, resulting in a smoothly closed surface. For stiff materials, a fracturing approach as suggested in [PKA*05] would produce more realistic looking results.

Acknowledgements. Bart Adams is funded as a Research Assistant by the Fund for Scientific Research - Flanders, Belgium (F.W.O.-Vlaanderen). Thanks to Miguel Otaduy for helpful comments.

References

[AA04] ADAMSON A., ALEXA M.: Approximating bounded, non-orientable surfaces from points. In *Proc. of Shape Modeling International* (2004). 2, 8

[AD03] ADAMS B., DUTRE P.: Interactive boolean operations on surfel-bounded solids. *ACM Trans. Graph. 22*, 3 (2003), 651–656. 2

[BK03] BOTSCH M., KOBBELT L.: High-quality point-based rendering on modern gpus. In *Proc. of Pacific Graphics 2003* (2003), pp. 335–343. 2, 8

[Bli82] BLINN J. F.: A generalization of algebraic surface drawing. *ACM Trans. Graph. 1*, 3 (1982), 235–256. 5

[CMT04] CARLSON M., MUCHA P. J., TURK G.: Rigid fluid: animating the interplay between rigid bodies and fluid. *ACM Trans. Graph. 23*, 3 (2004), 377–384. 2

[CMVT02] CARLSON M., MUCHA P., VAN HORN III B., TURK G.: Melting and flowing. In *Proc. of the ACM SIGGRAPH Symposium on Computer Animation* (2002). 2

[Coh91] COHEN L. D.: On active contour models and balloons. *CVGIP: Image Underst. 53*, 2 (1991), 211–218. 5

[DC95] DESBRUN M., CANI M.-P.: Animating soft substances with implicit surfaces. In *Computer Graphics Proceedings* (1995), ACM SIGGRAPH, pp. 287–290. 2, 7

[DC96] DESBRUN M., CANI M.-P.: Smoothed particles: A new paradigm for animating highly deformable bodies. In *6th Eurographics Workshop on Computer Animation and Simulation '96* (1996), pp. 61–76. 2, 3, 4

[DC99] DESBRUN M., CANI M.-P.: *Space-Time Adaptive Simulation of Highly Deformable Substances*. Tech. rep., INRIA Nr. 3829, 1999. 9

[DCG98] DESBRUN M., CANI-GASCUEL M.-P.: Active implicit surface for animation. In *Proc. of Graphics Interface* (1998), pp. 143–150. 2, 5, 9

[ELF04] ENRIGHT D., LOSASSO F., FEDKIW R.: A fast and accurate semi-lagrangian particle level set. 479–490. 2

[EMF02] ENRIGHT D., MARSCHNER S., FEDKIW R.: Animation and rendering of complex water surfaces. In *SIGGRAPH: Proc. of the 29th annual conference on Computer graphics and interactive techniques* (2002), pp. 736–744. 2

[FF01] FOSTER N., FEDKIW R.: Practical animation of liquids. In *SIGGRAPH '01: Proc. of the 28th annual conference on Computer graphics and interactive techniques* (2001), pp. 23–30. 2

[GBO04] GOKTEKIN T. G., BARGTEIL A. W., O'BRIEN J. F.: A method for animating viscoelastic fluids. In *Proc. of ACM SIGGRAPH* (2004), vol. 23, pp. 463–468. 2, 3

[KMH*04] KEISER R., MÜLLER M., HEIDELBERGER B., TESCHNER M., GROSS M.: Contact handling for deformable point-based objects. In *Proc. of Vision, Modeling, Visualization VMV* (Nov 2004), pp. 339–347. 7

[KWT88] KASS M., WITKIN A., TERZOPOULOS D.: Snakes: active contour models. *International Journal of Computer Vision 1*, 4 (1988), 321–331. 4

[LS81] LANCASTER P., SALKAUSKAS K.: Surfaces generated by moving least squares methods. *Mathematics of Computation 87* (1981), 141–158. 3

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation* (2003), pp. 154–159. 1, 2, 3, 4

[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. of the ACM SIGGRAPH/Eurographics symposium on Computer animation* (2004), pp. 141–151. 1, 2, 3, 4, 5, 7, 8

[Mon92] MONAGHAN J.: Smoothed particle hydrodynamics. *Annu. Rev. Astron. and Astrophysics 30* (1992), 543. 2, 3

[MSHG04] MÜLLER M., SCHIRM S., HEIDELBERGER B., GROSS M.: Interaction of fluids with deformable solids. In *Computer Animation and Virtual Worlds (CAVW)* (2004), pp. 159–171. 3

[OBH02] O'BRIEN J. F., BARGTEIL A. W., HODGINS J. K.: Graphical modeling and animation of ductile fracture. In *Proc. of SIGGRAPH* (2002), pp. 291–294. 3

[OS88] OSHER S., SETHIAN J. A.: Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations. *J. Comput. Phys. 79*, 1 (1988), 12–49. 2

[pba04] Point-based animation. http://www.pointbasedanimation.org, 2004. 2

[PGK02] PAULY M., GROSS M., KOBBELT L. P.: Efficient simplification of point-sampled surfaces. In *Proc. of the conference on Visualization* (2002), IEEE Computer Society, pp. 163–170. 6

[PKA*05] PAULY M., KEISER R., ADAMS B., DUTRÉ P., GROSS M., GUIBAS L. J.: Meshless animation of fracturing solids. In *Proc. of ACM Siggraph* (2005). To appear. 3, 9

[PKKG03] PAULY M., KEISER R., KOBBELT L. P., GROSS M.: Shape modeling with point-sampled geometry. In *Proc. of ACM Siggraph* (2003), pp. 641–650. 2, 5, 7

[SAC*99] STORA D., AGLIATI P.-O., CANI M.-P., NEYRET F., GASCUEL J.-D.: Animating lava flows. In *Proc. of Graphics Interface* (1999), pp. 203–210. 2, 4

[ST92] SZELISKI R., TONNESEN D.: Surface modeling with oriented particle systems. In *SIGGRAPH '92: Proc. of the 19th annual conference on Computer graphics and interactive techniques* (1992), pp. 185–194. 2

[Sta99] STAM J.: Stable fluids. In *SIGGRAPH: Proc. of the 26th annual conference on Computer graphics and interactive techniques* (1999), pp. 121–128. 2

[TF88] TERZOPOULOS D., FLEISCHER K.: Modeling inelastic deformation: viscolelasticity, plasticity, fracture. In *Proc. of the 15th annual conference on Computer graphics and interactive techniques* (1988), ACM Press, pp. 269–278. 2, 3

[THM*03] TESCHNER M., HEIDELBERGER B., MÜLLER M., POMERANERTS D., GROSS M.: Optimized spatial hashing for collision detection of deformable objects. In *Proc. Vision, Modeling, Visualization VMV* (2003), pp. 47–54. 4

[Ton91] TONNESEN D.: Modeling liquids and solids using thermal particles. In *Graphics Interface* (June 1991), pp. 255–262. 2

[TPBF87] TERZOPOULOS D., PLATT J., BARR A., FLEISCHER K.: Elastically deformable models. In *Computer Graphics Proceedings* (July 1987), Annual Conference Series, ACM SIGGRAPH 87, pp. 205–214. 2

[TPF89] TERZOPOULOS D., PLATT J., FLEISCHER K.: Heating and melting deformable models (from goop to glop). In *Graphics Interface* (1989), pp. 219–226. 2

[WH94] WITKIN A. P., HECKBERT P. S.: Using particles to sample and control implicit surfaces. In *Computer Graphics Proceedings* (1994), ACM SIGGRAPH, pp. 269–277. 2

[ZPKG02] ZWICKER M., PAULY M., KNOLL O., GROSS M.: Pointshop 3d: an interactive system for point-based surface editing. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), pp. 322–329. 2

[ZPvG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), pp. 371–378. 2

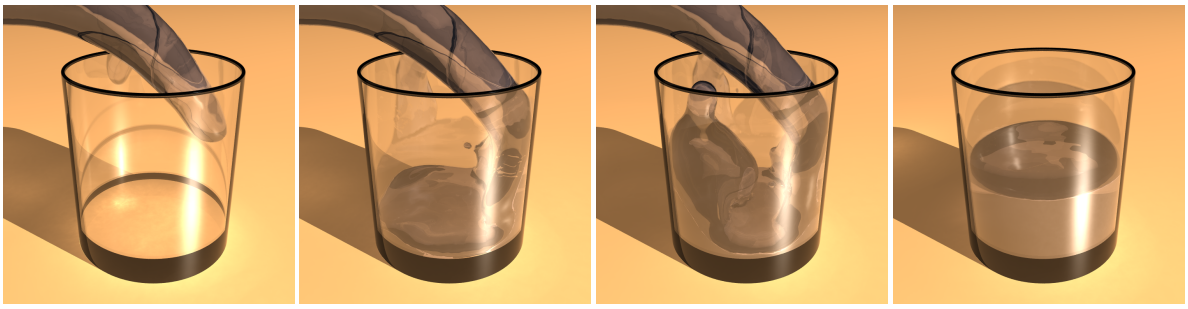

Figure 4: *Pouring a pure fluid into a glass.*

Figure 5: *Freezing a quicksilver fluid which is poured into a glass. After removing the glass, the elastic solid bounces onto the ground and fractures.*

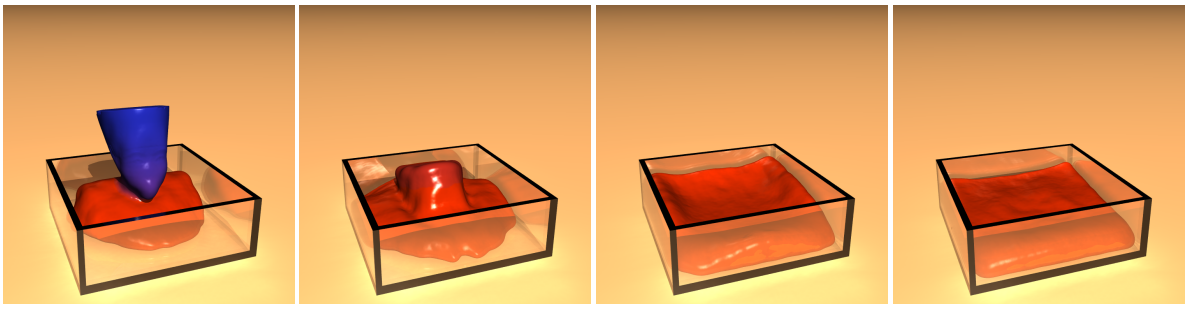

Figure 6: *An elastic solid is dropped onto a heated box and melts to a splashing fluid due to the quick heat transfer.*

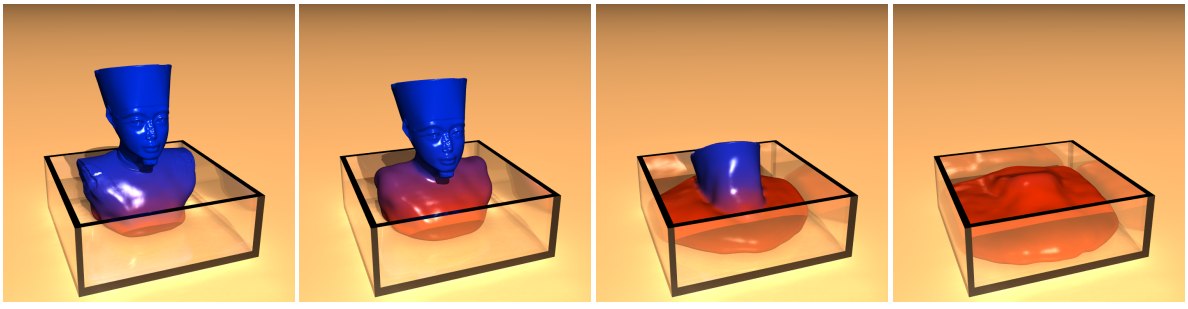

Figure 7: *An elastic solid is dropped onto a heated box and slowly melts to a viscous fluid.*