Eurographics Symposium on Point-Based Graphics (2006)
M. Botsch, B. Chen (Editors)

# Versatile Virtual Materials Using Implicit Connectivity

Martin Wicke[†] Philipp Hatt[‡] Mark Pauly[*] Matthias Müller[†] Markus Gross[*]

[*]ETH Zurich [†]Ageia Inc.

**Abstract**

*We propose a new method for strain computation in mesh-free simulations. Without storing connectivity information, we compute strain using local rest states that are implicitly defined by the current system configuration. Particles in the simulation are subject to restoring forces arranging them in a locally defined lattice. The orientation of the lattice is found using local shape matching techniques. The strain state of each particle can then be computed by comparing the actual positions of the neighboring particles to their assigned lattice positions. All necessary information needed to compute strains is contained in the current state of the simulation, no rest state or connectivity information is stored. Since no time integration is used to compute the strain state, errors cannot accumulate, and the method is well-suited for stiff materials.*

*In order to simulate phase transitions, the strain computation can be integrated into an existing particle-based fluid simulation framework. Implementing phase transitions between liquid and solid states becomes simple and elegant, since no transfer of material between different representations is needed. Using the current neighborhood relationships, the model provides penalty-based inter-object and self-collision handling at no additional computational cost.*

## 1. Introduction

Physical simulations are widely used in computer animation. As more and more computing power is available on commodity hardware, simulations have also started to replace or enhance scripted animations in computer games. A broad range of materials, from smoke and fluids to elastic and rigid solids is simulated in order to avoid tedious manual animation. Naturally, each class of materials is simulated using custom data structures that have proven their utility for the problem at hand. For instance, Eulerian grids have evolved as the preferred simulation domain for fluids and smoke, while Lagrangian meshes are usually used for simulation of deformable or rigid solids.

As more and more different simulated materials are part of one animation, questions of interaction naturally arise. Two-way interaction between solids and fluids has been thoroughly researched, for example in [Ben92, MST*04, GSLF05]. Another difficult area is the treatment of materials which do not clearly belong to any category, such as highly viscous or viscoelastic fluids. [GBO04, KAG*05, CBP05] have extended Eulerian and Lagrangian fluid simulations to include elastic stresses.

The problem of phase transitions is even more involved, since adding mass to one representation and removing it from another poses problems for the respective simulations. [LIGF05] treat the case of melting or burning, where a solid dissolves and mass is inserted into the fluid simulation. The different data structures for surface, fluid simulation and deformable or rigid body simulation have to be synchronised in order to allow mass transfer between the coupled simulations.

In this paper, we present a virtual material that is highly versatile. The possible material properties range from those of a stiff elastic, brittle solid to those of a low-viscous fluid. Starting from a particle-based fluid simulation, we add elastic forces to the simulation by forcing neighboring particles to positions on a locally defined lattice. Inspired by crystallography, the lattice is a hexagonal grid in two dimensions, and a cubic closest packed structure in three dimensions. Its orientation is determined using local shape matching similar

† {grossm,pauly,wicke}@inf.ethz.ch
‡ {hattp,mmueller}@ageia.com

to [MHTG05]. In no part of the simulation, explicit information on connectivity is needed. The spatial neighborhood relationships between particles in the current simulation state implicitly define a connectivity that is used for strain computation. As will be shown, the absence of a simulation mesh or rest state greatly simplifies the simulation of melting and freezing processes. Since our model can handle fluids as well as solids, material does not have to be transferred between representations. The neighborhood information computed during the simulation can be used to provide a simple penalty-based collision handling scheme at no additional computational cost.

The remainder of this paper is structured as follows: We first discuss related work before presenting our simulation model in Section 3. The materials simulated using our technique have inherent properties that are described in Section 4. One of the greatest advantages of our method is the simple integration of phase transitions, as detailed in Section 5. Section 6 treats surface reconstruction, before we show some results in Section 7. Section 8 discusses strengths and limitations of our approach and gives an outlook on future research directions.

## 2. Related Work

Recent work has greatly extended the range of materials that can be simulated using Eulerian fluid simulation methods. [CMHT02] use extreme viscosity in model plastic, nonelastic material. [CMT04] constrain parts of the fluid to rigid motions and can thus simulate rigid bodies within a fluid simulation. Coupling Eulerian fluid simulations with other simulation types is challenging. [LIGF05] simulate melting and burning by transferring material between a Lagrangian simulation grid and the simulation grid of the fluid simulation. [GSLF05] treat the interaction of thin shells or cloth with a Eulerian fluid simulation.

Elastic or visco-elastic materials require the computation of strain. [GBO04] achieve this for Eulerian fluid simulations by integrating strain rates. The integration errors limit the stiffness of the simulated materials.

Particle-based methods are always Lagrangian methods, hence the difference between solid and fluid simulation is less pronounced. [MKN*04] propose a point-based elasticity model. Using extreme plasticity, the behaviour of a viscous fluid can be modeled. [KAG*05] start from a smoothed particle hydrodynamics (SPH) simulation and measure strain by storing and modifying a rest state, including particle connectivity. In [CBP05], dynamically generated springs are used to model elasticity in an SPH simulation. A similar method was first introduced by Terzopolous et al. [TPF89] who enhanced a model inspired by molecular dynamics with explicit connectivity in order to model elasticity.

The method that is conceptually closest to our approach is simulation of elastic materials and viscous fluids using particles and Lennard-Jones potentials [Ton98]. Similar to our method, no stored connectivity between particles is needed. The rest state of each particle is implicitly given as the nearest minimum of the potential function defined as the superposition of the particle potentials. In contrast, our approach uses shape matching to determine the rest state. In practice, this is much more stable than reliying on the potential function alone. It also allows for larger deformations and larger integration timesteps. Müller et al. [MHTG05] also present a deformation framework based on shape matching, however relying on stored node connectivity.

## 3. Simulation Model

Our aim is to simulate materials ranging from fluid to solid. Similar to [GBO04, KAG*05], we will use a fluid simulation and enhance it with the necessary additional forces. Since our simulation is particle-based, we use a variant of XSPH [Mon89]. For a good introduction to SPH methods, we refer the reader to [Mon05].

### 3.1. Implicit Rest State

In order to introduce elastic restoring forces into a fluid simulation, we need to compute strain. Assuming an initially regular sampling, we can reconstruct the appropriate rest state from the current simulation state. The rest state is implicit to the system configuration at any point of the simulation. We extract the local rest state information using shape matching. Using the rest state, we can easily compute the strain tensor, which in turn can be used in any standard elasticity model.

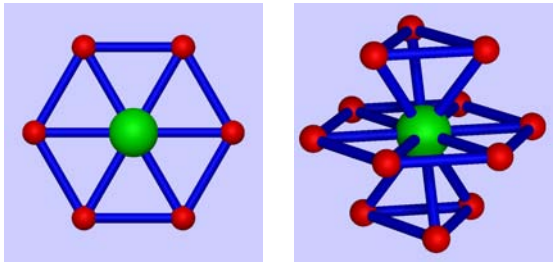
In a crystal lattice, the rest state of the neighbors of any atom is determined by the orientation of the lattice and the lattice type. We use a closest sphere packing for our lattice. In two dimensions, this is a hexagonal grid. In three dimensions, we use the cubic closest packed structure, since it has a higher symmetry than the hexagonally closest packed lattice. An important property of closest sphere packings is that they are a stable state of particle-based fluid simulations.

A cubic closest packed lattice around the origin consists of the points in the set

$$L = \left\{ D\left[ l\hat{\mathbf{x}} + m\left(\tfrac{1}{2}\hat{\mathbf{x}} + \sqrt{\tfrac{3}{4}}\hat{\mathbf{y}}\right) + n\left(\tfrac{1}{2}\hat{\mathbf{x}} + \sqrt{\tfrac{1}{12}}\hat{\mathbf{y}} + \sqrt{\tfrac{2}{3}}\hat{\mathbf{z}}\right)\right] \right\}, \tag{1}$$

with integers $l,m,n \in \mathbb{Z}$. The vectors $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$ are an orthonormal basis of $\mathbb{R}^3$. $D$ denotes the inter-particle distance in the lattice. See Figure 1 for an illustration.

In our simulation, elastic forces only depend on the current neighborhood of any particle. Each particle in the simulation will exert elastic forces on the particles that are its neighbors. This set of neighbors is determined by the SPH weight function, which is nonzero only within a certain radius around each point.



(a) (b)

**Figure 1:** *Closest lattice points to a particle. (a) 2D lattice. (b) 3D lattice. Shown are the center particle (green) as well as the lattice positions of its immediate neighbors (red).*

### 3.2. Finding the Rest State

One simulation step can be summarized as follows: We first assign lattice points to particles in the neighborhood. Then, a linear tranformation $\mathbf{A}$ is computed such that the transformed grid best matches the particle locations. Applying only the rigid part of $\mathbf{A}$ to the grid yields a rest state for the neighboring particles.

We use the grid transformation from last timestep, $\mathbf{A}^{(t-1)}$, to assign lattice points to particles. For each particle $\mathbf{p}_j$ in the neighborhood of $\mathbf{p}_i$, we assign the lattice point $\mathbf{l}_{ij}$ which is closest to $\mathbf{r}_{ij} = (\mathbf{p}_j - \mathbf{p}_i)$ in the untransformed lattice.

$$\mathbf{l}_{ij} = \underset{\mathbf{l}\in L}{\operatorname{argmin}} \left\| \mathbf{l} - \left(\mathbf{A}_i^{(t-1)}\right)^{-1} \mathbf{r}_{ij} \right\|^2 \tag{2}$$

Due to the structure of the grid, this minimization can be easily solved by enumerating the nearest points in $L$. Note that it is possible that several neighboring particles are assigned to the same lattice point. In such a case, we only assign the particle that is closest to the lattice point. The forces generated by this neighborhood will not be applied to the free particle.

After all points are assigned, we compute the transformation $A$ such that the transformed grid best matches the actual particle positions in a least squares sense.

$$\mathbf{A}_i = \underset{\mathbf{A}\in\mathbb{R}^{3\times 3}}{\operatorname{argmin}} \sum_j w_i(\mathbf{p}_j) \|\mathbf{A}^{-1}\mathbf{r}_{ij} - \mathbf{l}_{ij}\|^2 \tag{3}$$

Here, $w_{ij}$ determines the influence of $\mathbf{p}_j$ on the matching. The weights should be a smooth function with local support. We use the SPH kernel functions as weight functions: $w_{ij} = k(\|\mathbf{p}_i - \mathbf{p}_j\|)$.

As pointed out in [MHTG05], the solution to this minimization is

$$\mathbf{A}_i = \left(\sum_j w_{ij}\mathbf{r}_{ij}\mathbf{l}_{ij}^T\right)\left(\sum_j w_{ij}\mathbf{l}_{ij}\mathbf{l}_{ij}^T\right)^{-1} \tag{4}$$

In order to obtain the rigid part of the transformation, we compute a polar decomposition of $\mathbf{A}_i = \mathbf{R}_i\mathbf{S}_i$. The rotational part $\mathbf{R}_i$ represents the rigid motion of the lattice. The rest state $\mathbf{g}_{ij}$ of a particle $\mathbf{p}_j$ with respect to $\mathbf{p}_i$ is then

$$\mathbf{g}_{ij} = \mathbf{p}_i + \mathbf{R}_i\mathbf{l}_{ij}. \tag{5}$$

### 3.3. Computing Strain

Having computed rest states for all neighboring particles, we can compute an estimate for the strain tensor at a particle $p_i$. The linear strain tensor is defined as $\varepsilon = \frac{1}{2}\left(\nabla\mathbf{u}^T + (\nabla\mathbf{u}^T)^T\right)$, where $\mathbf{u}$ is the displacement of the material. We will discretize the partial derivatives of the displacement using one-sided differences. For the $y$ component of the gradient of $u_x$ at $\mathbf{p}_i$, a particle $\mathbf{p}_j$ contributes

$$\left(\frac{du_x}{dy}\right)_{ij} = \frac{(\mathbf{r}_{ij} - \mathbf{g}_{ij})\cdot\hat{\mathbf{x}}}{\mathbf{g}_{ij}\cdot\hat{\mathbf{y}}}, \tag{6}$$

the other derivatives are computed accordingly. Since several particles influence the strain state of $\mathbf{p}_i$, we weight the contributions of the particles:

$$\left(\frac{du_x}{dy}\right)_i = \frac{\sum_j w_{ij}\mathbf{g}_{ij}\cdot\hat{\mathbf{y}}\left(\frac{du_x}{dy}\right)_{ij}}{\sum_j w_{ij}\mathbf{g}_{ij}\cdot\hat{\mathbf{y}}} = \frac{\sum_j w_{ij}(\mathbf{r}_{ij} - \mathbf{g}_{ij})\cdot\hat{\mathbf{x}}}{\sum_j w_{ij}\mathbf{g}_{ij}\cdot\hat{\mathbf{y}}} \tag{7}$$

Thus, the strain can be computed from the knowledge of rest state and current particle positions. Note that in a regular setting, (7) yields central differences. Using the same idea, nonlinear strain can be approximated. The strain can then be used to apply any standard elasticity model. In the next section, we describe a simplified model that directly uses the implicit rest state.

### 3.4. Elastic Restoring Forces

For each particle $\mathbf{p}_i$, we compute elastic forces for all neighbors $\mathbf{p}_j$ that pull $\mathbf{p}_j$ closer to its rest state position $\mathbf{g}_{ij}$.

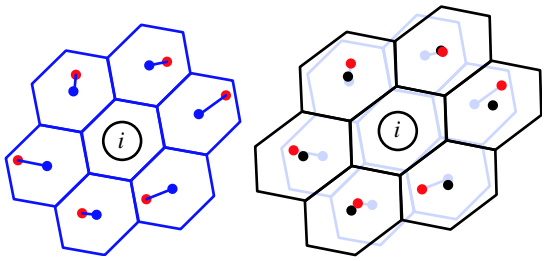
$$\mathbf{F}_{ij} = w(\mathbf{l}_{ij})k(\mathbf{g}_{ij} - \mathbf{r}_{ij}) \tag{8}$$

Here, $k$ is a stiffness constant and $w(\mathbf{l}_{ij})$ a smoothly decaying weight function, representing the diminishing influence of $\mathbf{p}_i$ on points that are not direct neighbors. In order to enforce preservation of linear momentum, we apply $\frac{1}{2}\mathbf{F}_{ij}$ to $\mathbf{p}_j$ and $-\frac{1}{2}\mathbf{F}_{ij}$ to $\mathbf{p}_i$.

This introduces a torque $\tau_i = \frac{1}{2}\sum_j(\mathbf{p}_j - \mathbf{c}_i)\times\mathbf{F}_{ij} - (\mathbf{p}_i - \mathbf{c}_i)\times\mathbf{F}_{ij}$, measured around the center of mass $\mathbf{c}_i$ of $\mathbf{p}_i$ and its neighbors $\mathbf{p}_j$. As this torque would violate the preservation of angular momentum, we redistribute it onto the $\mathbf{p}_j$ by adding a torque correction force to $\mathbf{p}_j$.

$$\mathbf{F}_{ij}^{\tau} = \frac{w_{ij}}{\sum_j w_{ij}\|\mathbf{p}_j - \mathbf{c}_i\|}\tau_i \times \frac{\mathbf{p}_j - \mathbf{c}_i}{\|\mathbf{p}_j - \mathbf{c}_i\|} \tag{9}$$

Again, the weighting ensures that the influence of a particle smoothly decays to zero.




(a) (b)

**Figure 2:** *Assigning lattice points and shape matching. (a) Neighboring particles (red) are assigned to lattice points (blue) using the local grid transformation* $\mathbf{A}_i^{t-1}$. *(b) Shape matching:* $\mathbf{A}_i$ *is computed such that the grid points (black) best match the particle positions.*

### 3.5. Damping

For systems with high stiffnesses, damping is essential. While implicit integration methods include numerical damping by construction, explicit integration requires an explicit damping model even if an undamped system is to be simulated. Particle fluid simulations are damped using viscosity, which is not sufficient when elastic forces are simulated. While damping should remove high-frequency oscillations from the system, it should not affect rigid body motion. We therefore apply damping only to those components of the particle velocities that do not correspond to locally rigid motions.

Consider a particle $\mathbf{p}_i$ and its neighboring particles $\mathbf{p}_j$. We first compute the average velocity $\bar{\mathbf{v}}_i$ at the center of mass $\mathbf{c}_i$ and the angular velocity $\omega_i$ around $\mathbf{c}_i$. Using the SPH interpolation framework, we find

$$\bar{\mathbf{v}}_i = \sum_j w(\|\mathbf{c}_i - \mathbf{p}_j\|)\frac{m_j}{\rho_j}\mathbf{v}_j \tag{10}$$

$$\omega_i = \sum_j w(\|\mathbf{c}_i - \mathbf{p}_j\|)(\mathbf{v}_j - \bar{\mathbf{v}}_i) \times (\mathbf{p}_j - \mathbf{c}_i) \tag{11}$$

Here, $w(\cdot)$ denotes the SPH weight function used in the fluid simulation, and $\rho_j$ is the density at the positon $\mathbf{p}_j$. The density is computed during the fluid simulation.

We can now split the velocity of each of the particles $\mathbf{p}_j$ into a rigid and a nonrigid part.

$$\mathbf{v}_j = \mathbf{v}_{ij}^n + \mathbf{v}_{ij}^r = \mathbf{v}_{ij}^n + \bar{\mathbf{v}}_i + \omega_i \times (\mathbf{p}_j - \mathbf{c}_i) \tag{12}$$

where $\mathbf{v}_{ij}^r$ is the locally rigid part of $\mathbf{v}_j$ and $\mathbf{v}_{ij}^n$ denotes the particle's individual nonrigid movement, each with respect to the average angular and linear velocities of the neighborhood $i$.

For any particle $\mathbf{p}_j$, we only want to damp the locally nonrigid modements $\mathbf{v}_{ij}^n$ for all reference systems $i$ that $\mathbf{p}_j$ is influenced by. We thus use the SPH average of the nonrigid velocities in each of the neighborhoods and obtain a damping force

$$\mathbf{F}_j^d = -\eta\mathbf{v}_j^n = -\eta\sum_i w_{ij}\frac{m_i}{\rho_i}\mathbf{v}_{ij}^n, \tag{13}$$

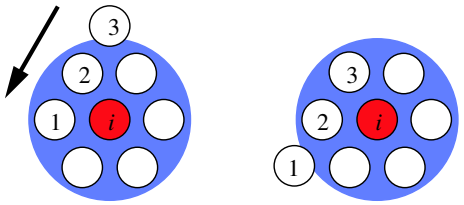
where $\eta$ is a damping constant. In a simulation with timestep $\Delta t$, $\eta$ should not exceed $\frac{m_i}{\Delta t}$, where $m_i$ is the mass of particle $i$. If $\eta$ is greater than the above value, not only are the nonrigid velocities completely damped out, new nonrigid movement is introduced in the subsequent integration step.

## 4. Inherent Material Properties

The virtual material as described above exhibits a wide range of material properties, depending on the parameter settings. However, some properties are inherent to the approach chosen, and shall be discussed here.

### 4.1. Plasticity

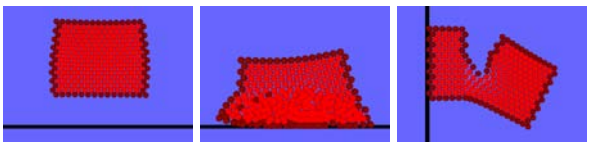
Since no rest state information is stored, the rest state of the material has to be inferred from the current state. All information on the rest state is thus contained in the positions of the current neighbors of a particle. Since the particle does not store which particles were its initial neighbors, the method has no way of knowing if the particles in the neighborhood have changed. In that case, there are no restoring forces for the original particles. Instead, they are integrated into their new neighborhoods. See Figure 3 for an illustration.




(a) (b)

**Figure 3:** *Inherent plasticity: Since we do not store connectivity, particle i has no way of distinguishing between the situations shown above. Its neighborhood is limited to the blue region. There will be no restoring forces for particles 1, 2 and 3, the deformation is plastic.*

This also means that our virtual material cannot be stretched arbitrarily. Consider a pair of neighboring particles $p_i$ and $p_j$. A permanent plastic deformation occurs if $p_j$ is displaced far enough such that different lattice point in is closest to its current position, or even leaves the neighborhood of $p_i$ altogether. Since our material has no memory and does not know the "true" rest state of $p_j$, these changes are not counteracted by restoring forces. However, since the particles are assigned to lattice points using the local deformation from the last timestep, this situation can only occur if $p_j$ changes its position radically within one timestep, or if it leaves the neighborhood of $p_i$ due to large deformations. Figure 4 (b) shows plastic deformation in a 2D simulation.



(a) (b) (c)

**Figure 4:** *Particles in 2D simulations of different material properties. (a) An elastic cube bounces off the groud plane. (b) Rest state of a plastic cube after falling. (c) Fracture. The cube is fixed to the Wall and fractures under the influence of gravity. Darker particles have a docking rate $\lambda_d = 0$ for some of their lattice points. They form the boundary of the solid.*

### 4.2. Fracture

For a fluid, topological changes are temporary. As soon as different parts of the fluid rejoin, they behave as one. The situation is different for solids. Fracture permanently breaks the material, even if the different parts of the solid come into contact, they will not reunify.

In our basic model, however, as soon as a particle enters a neighborhood of another particle, it will be integrated into the lattice. Thus, cracks once formed will close again when the edges come into contact.

To avoid this behaviour, each particle stores which of its lattice points are allowed to be occupied by other particles. We use a probabilistic model to account for fault and weaknesses in the material. In order to be able to model probabilistic properties independent of the simulation timestep, we store a *docking rate* $\lambda_d$ for each lattice point. If a particle is available in one timestep, the probability that it is assigned to the lattice point is given by a Poisson process: $P_d = 1 - e^{-\lambda_d \Delta t}$. If a lattice point is not assigned to a particle during a timestep, the docking rate of this lattice point is reduced by $\Delta\lambda_d$. If it is assigned, the docking rate is increased by $\Delta\lambda_d$. A stress criterion can be applied to additionally modify $\lambda_d$ or $P_d$. Figure 4 (c) shows fracturing in a 2D simulation.

### 4.3. Multiple Objects

We can easily extend the algorithm to handle multiple distinct objects. Each particle carries an object ID, and the shape matching as well as the force computation are confined to particles with the same object ID.

If we restrict forces acting between different objects to repulsive forces instead of disallowing them altogether, the result is a simple penalty-based collision detection scheme. If $p_i$ and $p_j$ are particles from different objects, we apply a modified interaction force $F'_{ij}$

$$\mathbf{F}'_{ij} = \begin{cases} \frac{\mathbf{p}_j - \mathbf{p}_i}{\|\mathbf{p}_j - \mathbf{p}_i\|} \cdot \mathbf{F}_{ij} \frac{\mathbf{p}_j - \mathbf{p}_i}{\|\mathbf{p}_j - \mathbf{p}_i\|} & (\mathbf{p}_j - \mathbf{p}_i) \cdot \mathbf{F}_{ij} > 0 \\ 0 & \text{otherwise} \end{cases} \quad (14)$$

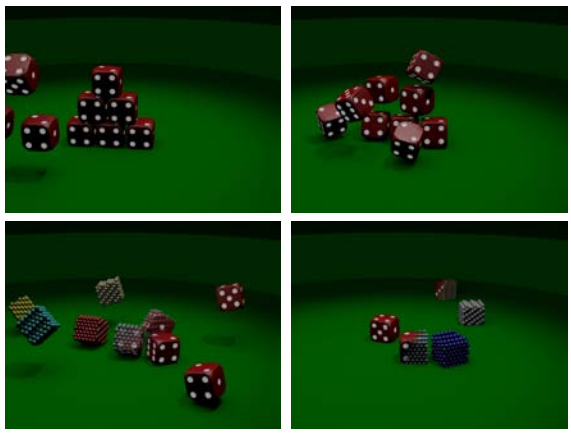


**Figure 5:** *Several colliding stiff elastic objects. Inter-object collisions are handled as described in Section 4.3, no additional collision handling is necessary.*

Of course, more elaborate collision handling schemes can be implemented for objects within our framework, but the implicit collision handling provided comes at no additional cost and has proven sufficient in most situations. All scenes shown in this paper were computed using only the inherent collision handling to resolve inter-object collisions and solid-fluid interaction.

## 5. Phase Transitions

One of the greatest advantages of our simulation method is the simple integration of phase transitions. Since both fluids and elastic solids can be handled within our framework, phase transitions can be implemented without switching representations. Since no rest states are stored, no artificial rest state information has to be generated in the case of freezing. Remeshing, mesh erosion and other implementational hurdles are avoided entirely.

Note that we do not aim at a correct simulation of the physical process of phase transitions. Our goal is to provide a simple means to change the behaviour of a material between liquid to solid.

The only difference between solids and fluids in our framework is the fact that particles in a solid are subject to elastic restoring forces while particles in the fluid phase are not. Thus, melting can be achieved simply by reducing the elastic forces as well as the associated damping until in the completely fluid phase, the particle is not influenced by lattice-induced forces any more.

An arbitrary criterion can be used to determine when a phase transition should occur. Useful criteria are based on position in space, the simulation time, or physical properties like temperature. Heat conduction and other transport phenomena can be simulated using the already available SPH

framework [Mon05]. We use a probabilistic approach similar to the one described in Section 4.2. In our framework, the *melting rate* is computed from temperature alone:

$$\lambda_m = \max(0, \lambda_m^0 (T - T_m)), \tag{15}$$

where $T$ is the temperature of a particle and $T_m$ is the melting point of the material.

The inverse process of freezing works similarly. However, it is advisable to further restrict when the phase transition may occur. If a fluid particle is close to a solid, it has a probability to freeze and thus integrate into the lattice of the solid. This probability is dependent on the relative velocity of the particle to its solid neighbors (if any), and the distance of the fluid particle to the next available lattice point, as well as other environmental factors as described for melting. This statistical approach mimicks the freezing process without the overhead of a full-blown physical simulation [KL03, KHL04].

To compute the *freezing rate* for a particle $p_i$ with neighbors $p_j$, we use a criterion based on temperature $T$, number of solid neighbors $n$, their average velocity $\bar{v}$, and the accumulated lattice forces.

$$\begin{aligned}\lambda_f = \max(0, \lambda_f^0 (T_f - T)(n - n_0) \cdot \\ \min(1, \tfrac{v_{max}}{\|\mathbf{v}_i - \bar{\mathbf{v}}\|}) \min(1, \tfrac{F_{max}}{\|\sum_j \mathbf{F}_{ij}\|}))\end{aligned} \tag{16}$$

$T_f$ is the freezing temperature of the fluid, which we usually set to zero. $n_0$ denotes the minimum number of solid neighbors. If this parameter is set to zero, particles can freeze spontaneously, given that the other parameters allow it. Otherwise, particles can only freeze to already solid material. The last term diminishes the freezing rate of a particle if the accumulated lattice forces exceed a given maximum. This is a simple measure of how good the current position of the particle fits into the lattice of a neighboring solid. Also, if the average velocity of the surrounding solid particles differs too much from the particles' velocity, it is unlikely to freeze.

## 6. Surface Representation

If the simulated material is liquid or solid, a surface needs to be extracted from the simulation data. An easy way to do so is using marching cubes [LK87] to extract an implicit surface from a potential field that is the superposition of potential fields attached to each of the particles. The blobbies [Bli82] approach or variants thereof [ZB05] yield good surfaces provided that the sampling density is high enough.

As the name suggests, the implicit functions defined as above are blobby. Although this is not a problem if the sampling with particles is dense enough, sharp features created by fracture cannot be reproduced faithfully in low-resolution simulations. In these situations, explicit surface representations, such as sampled surfaces [KAG*05], or semi-implicit approaches such as particle level sets [EFFM02] could lead to crisper surfaces retaining sharp features. As all necessary information is stored with the particles, it is not a problem to use these surface reconstruction methods together with our simulation model.

For solid objects that are known not to fracture or undergo plastic deformation, we can use a simple skinning technique to deform the surface. Similar to [WSG05], each particle $\mathbf{p}_i$ stores the a local position of nearby surface points $\mathbf{v}_j$ or mesh vertices in its local coordinate system. During initialization the local coordinates of $v_j$ with respect to $\mathbf{p}_i$ are computed as

$$\mathbf{v}_j^i = \mathbf{v}_j^0 - \mathbf{p}_i^0. \tag{17}$$

When the object deforms, the current grid transformation matrix $\mathbf{A}_i$ of the particle is applied to the stored local vertex positions. This yields deformed local surface points $\mathbf{v}_i^j$.

$$\mathbf{v}_j'^i = A_i \mathbf{v}_j^i \tag{18}$$

The deformed position of the surface point is a weighted sum of the different local surface points. In our implementation, we use the SPH weight function.

$$\mathbf{v}_j' = \sum_i w(\mathbf{p}_i - \mathbf{v}_j) \left( \mathbf{v}_j'^i + \mathbf{p}_i \right) \tag{19}$$

This method yields a smooth surface for deforming objects. Foldovers and self-intersections that plaque skeleton-based skinning approaches are not a problem in this setting unless the particle sampling is extremely coarse.

## 7. Results

The examples in this paper were computed on a P4 3GHz, and rendered using POV-Ray. Simulation times given exclude rendering.

Figure 5 shows nine stiff-elastic dice. Each die is sampled with 246 particles. The simulation time is approximately 19 seconds per frame. The dice are rendered using a textured surface mesh which is moved along with the particles using the skinning method described in Section 6.

In Figure 6, a viscous fluid freezes to a cooled rod. Fluid, ice and rod are modeled using the presented method. The surface of the cylinder is attached to its particles using skinning, the surfaces for fluid and ice are computed using marching cubes. The cylinder is sampled with 1606 particles, fluid and ice use up to 6000 particles. Average simulation time in this example is approximately 18 seconds per frame.

Figure 7 shows an elastic bunny being dropped and melting on the ground. The model is sampled using 9871 particles. The surface was reconstructed using marching cubes. Average simulation time is around 17 seconds per frame. Heat transfer is simulated as a diffusion process.

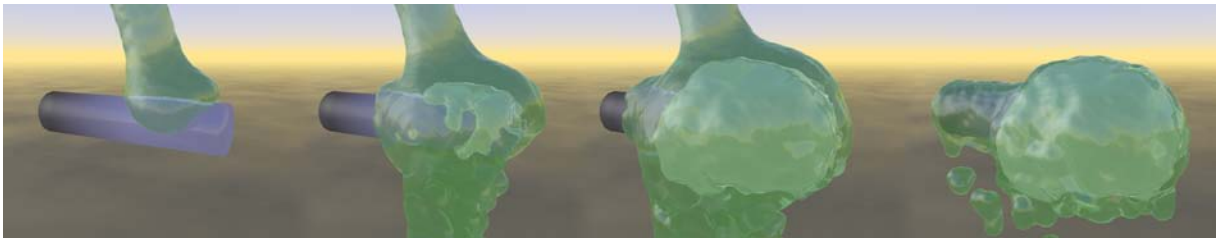


**Figure 6:** *Alien goo freezing on a cooled rod. The rod as well as the viscous fluid are modeled using our method. The freezing criterion from Equation 16 is used to trigger phase transitions of individual particles. The temperature of the rod is fixed, heat transport is modeled as a diffusion process.*

## 8. Discussion and Future Work

We have presented a method for computing elastic strain without storing rest states or connectivity. The rest state that is needed to compute a strain tensor can be inferred from the current state of the simulation. This method offers several advantages: Since only one representation is used for fluids as well as solids, no transfer of material between different representations is needed. Freezing can be implemented without making up a rest state for the newly frozen material.

Thus, a broad range of material properties from stiff elastic solids to fluids can be modeled. All of these materials exhibit the material properties described in Section 4. In particular, this means that materials simulated using our model deform plastically under large or abrupt deformations as described in Section 4.1, and fracture if stretched too much. The fracture process does not pose any problem for the simulation model, however, the surface reconstruction involving fracture is challenging. Since no explicit fracture events are generated by the model and no connectivity information is available, established techniques such as [MBF04] or [PKA*05] cannot be easily adapted. One future reseach direction will be to generate such events ex post, and apply that knowledge to a surface reconstruction technique that gracefully handles fracture in the underlying material.

Since the inherent material properties described in Section 4 are present in all materials simulated using the proposed approach, our method is limitated in that it cannot handle materials that do not exhibit these properties, most prominently extremely deformable yet non-plastic materials (e.g. chewing gum, certain types of rubber).

The assumed regular sampling does introduce discretisation artifacts when sampling boundaries. Since a high-resolution mesh is used for rendering, this is not noticeable visually. However, for tactile feedback or other applications that need high-resolution collision detection, this limitation can pose problems. A possible solution is to develop a multiresolution version of the presented algorithms. This would also greatly help for a more realistic simulation of low viscosity fluids.

Stiff materials impose increasingly severe limitations on timesteps, and simulating large scenes with quasi-rigid objects can become unfeasible. Although we found that very stiff materials such as seen in Figure 5 can be simulated using explicit integration, we plan to research the possibilities of implicit integration for this method. This would greatly alleviate timestep restictions.

## References

[Ben92] BENSON D. J.: Computational Methods in Lagrangian and Eulerian Hydrocodes. *Comput. Methods Appl. Mech. Eng. 99*, 2-3 (1992), 235–394. 1

[Bli82] BLINN J. F.: A Generalization of Algebraic Surface Drawing. pp. 235–256. 6

[CBP05] CLAVET S., BEAUDOIN P., POULIN P.: Particle-based Viscoelastic Fluid Simulation. In *Proceedings of the Symposium on Computer Animation '05* (2005), pp. 219–228. 1, 2

[CMHT02] CARLSON M., MUCHA P. J., HORN R. B. V., TURK G.: Melting and Flowing. In *Proceedings of the Symposium on Computer Animation '02* (2002), pp. 167–174. 2

[CMT04] CARLSON M., MUCHA P. J., TURK G.: Rigid Fluid: Animating the Interplay Between Rigid Bodies and Fluid. In *Proceedings of SIGGRAPH '04* (2004), pp. 377–384. 2

[EFFM02] ENRIGHT D., FEDKIW R., FERZIGER J., MITCHELL I.: A Hybrid Particle Level Set Method for Improved Interface Capturing. *Journal of Computational Physics 183* (2002), 83–116. 6

[GBO04] GOKTEKIN T. G., BARGTEIL A. W., O'BRIEN J. F.: A Method for Animating Viscoelastic Fluids. In *Proceedings of SIGGRAPH '04* (2004), pp. 463–468. 1, 2

[GSLF05] GUENDELMAN E., SELLE A., LOSASSO F., FEDKIW R.: Coupling Water and Smoke to Thin Deformable and Rigid Shells. In *Proceedings of SIGGRAPH '05* (2005), pp. 973–981. 1, 2

[KAG*05] KEISER R., ADAMS B., GASSER D., BAZZI P., DUTRÉ P., GROSS M.: A Unified Lagrangian Approach to Solid-Fluid Animation. In *Proceedings of the Symposium on Point-Based Graphics '05* (2005), pp. 125–133. 1, 2, 6

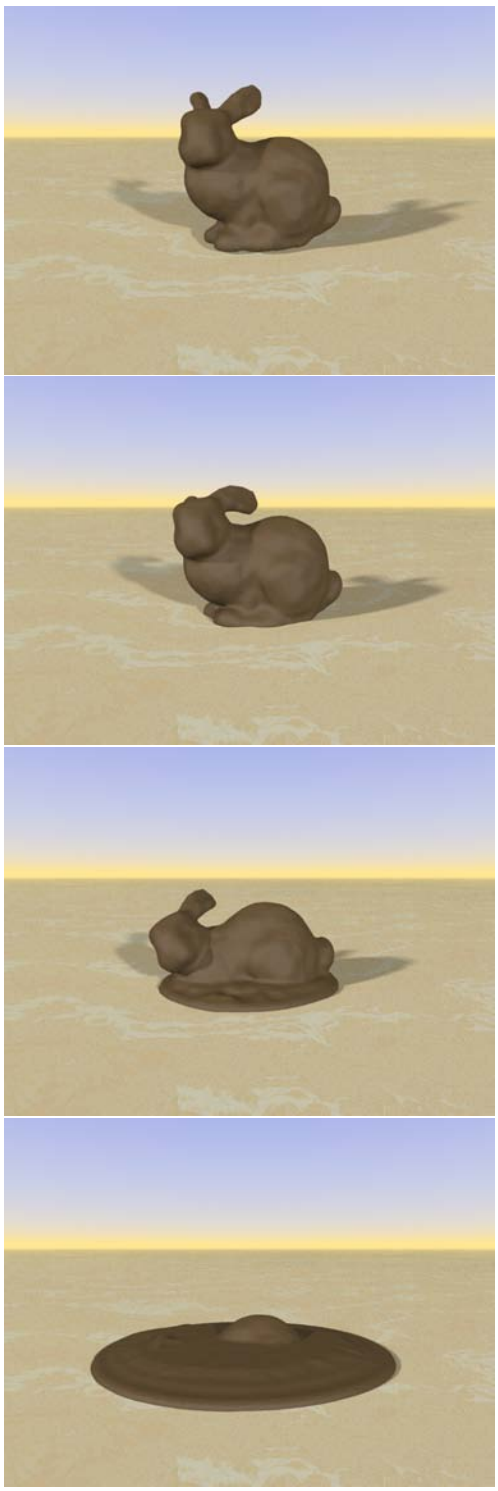


**Figure 7:** *A chocolate bunny falls, then melts on the hot desert sand.*


[KHL04] KIM T., HENSON M., LIN M. C.: A Hybrid Algorithm for Modeling Ice Formation. In *Proceedings of the Symposium on Computer Animation '04* (2004), pp. 305–314. 6

[KL03] KIM T., LIN M. C.: Visual Simulation of Ice Crystal Growth. In *Proceedings of the Symposium on Computer Animation '03* (2003), pp. 86–97. 6

[LIGF05] LOSASSO F., IRVING G., GUENDELMAN E., FEDKIW R.: Melting and Burning of Solids into Liquids and Gases. *IEEE Trans. Visualization and Computer Graphics in press* (2005). 1, 2

[LK87] LORENSEN W., KLINE H. E.: Marching Cubes: A HighResolution 3D Surface Construction Algorithm. In *Proceedings of SIGGRAPH '87* (1987), pp. 163–170. 6

[MBF04] MOLINO N., BAO Z., FEDKIW R.: A Virtual Node Algorithm for Changing Mesh Topology During Simulation. In *Proceedings of SIGGRAPH '04* (2004), pp. 385–392. 7

[MHTG05] MÜLLER M., HEIDELBERGER B., TESCHNER M., GROSS M.: Meshless Deformations Based on Shape Matching. In *Proceedings of SIGGRAPH '05* (2005), pp. 471–478. 2, 3

[MKN*04] MÜLLER M., KEISER R., NEALEN A., PAULY M., GROSS M., ALEXA M.: Point Based Animation of Elastic, Plastic and Melting Objects. In *Proceedings of the Symposium on Computer Animation '04* (2004), pp. 141–151. 2

[Mon89] MONAGHAN J. J.: On the Problem of Penetration in Particle Methods. *Journal of Computational Physics 82* (1989), 1–15. 2

[Mon05] MONAGHAN J. J.: Smoothed Particle Hydrodynamics. *Reports on Progress in Physics 68* (2005), 1703–1759. 2, 6

[MST*04] MÜLLER M., SCHIRM S., TESCHNER M., HEIDELBERGER B., GROSS M.: Interaction of Fluids with Deformable Solids. *Journal of Computer Animation and Virtual Worlds 15*, 3-4 (July 2004), 159–171. 1

[PKA*05] PAULY M., KEISER R., ADAMS B., DUTRÉ P., GROSS M., GUIBAS L. J.: Meshless Animation of Fracturing Solids. In *Proceedings of SIGGRAPH '05* (2005), pp. 957–964. 7

[Ton98] TONNESEN D.: *Dynamically Coupled Particle Systems for Geometric Modeling, Reconstruction, and Animation*. PhD thesis, University of Toronto, 1998. 2

[TPF89] TERZOPOLOUS D., PLATT J., FLEISCHER K.: Heating and Melting Deformable Models (From Goop to Glop). In *Proceedings of Graphics Interface '89* (1989), pp. 219–226. 2

[WSG05] WICKE M., STEINEMANN D., GROSS M.: Efficient Animation of Point-Based Thin Shells. In *Proceedings of Eurographics '05* (2005), pp. 667–676. 6

[ZB05] ZHU Y., BRIDSON R.: Animating Sand as a Fluid. In *Proceedings of SIGGRAPH '05* (2005), pp. 965–972. 6