Particle-based viscoelastic fluid simulation

Conference Paper · July 2005

DOI: 10.1145/1073368.1073400 · Source: DBLP

3 authors:

Simon Clavet
Université de Montréal
2 PUBLICATIONS **123** CITATIONS
SEE PROFILE

Philippe Beaudoin
Google Inc.
13 PUBLICATIONS **477** CITATIONS
SEE PROFILE

Pierre Poulin
Université de Montréal
92 PUBLICATIONS **1,952** CITATIONS
SEE PROFILE

Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2005)
K. Anjyo, P. Faloutsos (Editors)

Particle-based Viscoelastic Fluid Simulation

Simon Clavet, Philippe Beaudoin, and Pierre Poulin [†]

LIGUM, Dept. IRO, Université de Montréal

Abstract

We present a new particle-based method for viscoelastic fluid simulation. We achieve realistic small-scale behavior of substances such as paint or mud as they splash on moving objects. Incompressibility and particle anti-clustering are enforced with a double density relaxation *procedure which updates particle positions according to two opposing pressure terms. From this process surface tension effects emerge, enabling drop and filament formation. Elastic and non-linear plastic effects are obtained by adding springs with varying rest length between particles. We also extend the technique to handle interaction between fluid and dynamic objects. Various simulation scenarios are presented including rain drops, fountains, clay manipulation, and floating objects. The method is robust and stable, and can animate splashing behavior at interactive framerates.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling – Physically Based Modeling; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – Animation and Virtual Reality

1. Introduction

The field of computational fluid dynamics has introduced many different algorithms to simulate the complex behavior of fluids. Since liquids and gases play an important role in everyday life, several fluid simulation methods have been successfully used for computer graphics applications.

Two main categories of simulation methods exist: Eulerian grids and Lagrangian particles. Eulerian methods discretize the problem using a subdivision of the spatial domain and control fluid flow in each cell. Such techniques have been able to achieve among the most realistic simulations of liquid types ranging from simple flows to highly viscous fluids, with plastic and elastic behaviors. They easily enforce the incompressibility condition, but do not guarantee mass conservation for small features.

Lagrangian methods discretize the fluid mass using particles. By directly tracking chunks of matter as they travel through space, particle methods trivially guarantee mass conservation and provide a conceptually simple and versatile simulation framework.

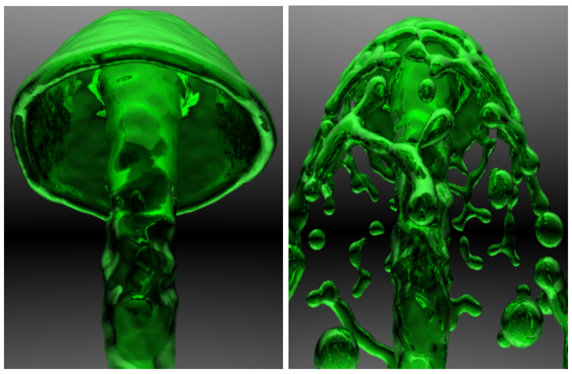


Figure 1: *Fountain simulation illustrating sheet, filament, and drop formation.*

Particles also alleviate the dependence on a specific reference frame. In an Eulerian approach, the distance a fluid feature can travel during a simulation step is limited by the grid resolution. Therefore, for very rapid and detailed flows, tracking changes as they occur at a fixed point in space appears less natural than tracking changes which occur along a particle trajectory. We revisit particle-based simula-

[†] email: clavetsi | beaudoin | poulin@iro.umontreal.ca

tion and propose extensions and modifications that increase their flexibility and robustness. Our formulation is simple and intuitive, and it can be implemented within days.

An important disadvantage of particle simulations is the difficulty to represent a smooth surface with particles. Robustly handling surface tension effects for small features is also recognized as a difficult task. Surface tension forces exist at the surface of the fluid, and depend on its curvature. The problem comes from the fact that particle simulations avoid modeling the surface explicitly, and thus robustly computing its curvature can be cumbersome.

We propose a novel procedure for incompressibility and particle anti-clustering. We call this procedure *double density relaxation*. Loosely speaking, double density relaxation computes two different particle densities: one quantifying the *number* of neighbors, and another quantifying the *number* of close neighbors. The force between two particles depends on these two context-dependent measures, instead of depending only on the distance between them. Liquids animated with this method display a smooth particle surface, and surface tension effects such as those shown in Figure 1 emerge naturally. No curvature has to be evaluated, and the procedure is robust even in presence of small surface details.

Numerical instabilities are often an issue with any physically-based simulations, requiring the use of prohibitively small timesteps. We use a conceptually intuitive prediction-relaxation scheme that remains stable even for large timesteps and fast splashing effects.

We also present a simple method to simulate viscoelastic substances by insertion and removal of springs between pairs of particles. Various material behaviors can be obtained with simple spring rest length update rules. The intuitive parameters governing such a scheme, combined with our fast simulation, allow an artist to interactively experiment with the fluid properties to obtain the desired results.

Finally, we describe a simple way to integrate the fluid solver with a rigid-body simulator, enabling two-way coupling between liquids and objects. We also propose a simple solution to the stickiness of liquids on objects.

2. Previous Work

2.1. Eulerian Simulation

Grid-based methods have been quite popular for computer graphics applications. Foster and Metaxas [FM96, FM97] were the first to propose solving the full 3D Navier-Stokes equations on a regular grid in order to re-create the visual properties of a dynamic fluid. Stam [Sta99] produced dynamic gases using a semi-Lagrangian integration scheme that achieves unconditional stability at the expense of artificial viscosity and rotational damping. Foster and Fedkiw [FF01] extended the technique to liquids, tracking the surface using both a level-set method and particles inside the liquid. Enright *et al.* [EMF02] added particles outside the fluid volume to improve surface tracking. They also proposed an extrapolation technique to assign velocities to air cells just outside the liquid surface, resulting in a smoother surface motion on a coarse grid.

A number of researchers have developed methods for modeling highly viscous or viscoelastic fluids. Carlson *et al.* [CMHT02] developed an Eulerian solver for very high viscosity liquids and were able to simulate melting objects. Fluids simulated by Goktekin *et al.* [GBO04] exhibit not only viscous but also elastic and plastic behaviors by integrating and advecting strain-rate throughout the fluid.

2.2. Lagrangian Simulation

Smoothed particle hydrodynamics (SPH) is an alternative approach, first developed by Lucy [Luc77] and by Gingold and Monaghan [GM77], to tackle astrophysical problems. In SPH, space is non-uniformly sampled using particles. These particles maintain various field properties, such as mass density or velocity, and are tracked during the simulation. The field quantities can be evaluated anywhere in space using radially symmetric smoothing kernels.

Reeves [Ree83] introduced particle systems to computer graphics. They quickly became a popular tool for portraying various effects such as fire or waterfalls. Miller and Pearce [MP89] borrowed ideas from molecular dynamics to add basic particle interactions, resulting in limited simulations of liquids and melting solids. Terzopoulos *et al.* [TPF89] modeled melting thermoelastic materials using particles that apply various forces to their neighbors. In a solid material, the particles are connected with springs, which weaken and eventually disappear as the material melts. This model does not handle plasticity.

Desbrun and Gascuel [DG96] applied SPH concepts to computer graphics in order to simulate highly deformable bodies. Recently, Müller *et al.* [MCG03] implemented interactive liquid simulation and rendering using SPH. They simulate surface tension using ideas from Morris [Mor00], implicitly defining an interface with the particles and applying a force proportional to curvature, computed as the divergence of the normal field. For surface features represented by a small number of particles, such curvature evaluation can cause numerical problems.

Premože *et al.* [PTB*03] also obtained realistic looking fluids using a Lagrangian method. They solve the Navier-Stokes equations using the *Moving-Particle Semi-Implicit* (MPS) method [KO96], which ensures a greater level of incompressibility than standard SPH.

Müller *et al.* [MKN*04] proposed a particle-based method for animating volumetric objects with material properties ranging from stiff elastic to highly plastic. Their technique is derived from continuum mechanics and therefore allows for direct specification of well-defined physical properties. Although they do not try to simulate a flowing liquid, they obtain very realistic results for the material types supported.

Steele *et al.* [SCED04] presented a particle-based method for simulating viscous liquids. They define the adhesion properties of different types of particles using general distance-dependent forces between particles. Their iterative relaxation scheme to conserve volume is similar to ours, but the linear density kernel and the hard anti-penetration constraints limit their method to highly viscous fluids.

3. Simulation Step

Our fluid is represented by particles evolving through space and time. A typical particle simulation goes through the following steps. First, various forces are computed and accumulated for each particle. These forces then modify the velocities, which are finally used to update particle positions. Such an explicit scheme tends to be unstable unless very small timesteps are used.

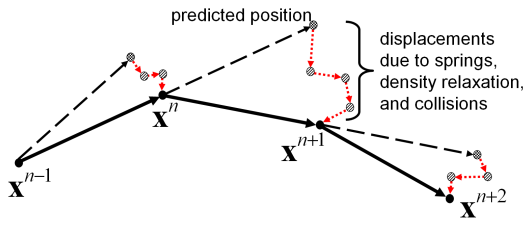


Figure 2: *Prediction-relaxation scheme.*

Our method avoids explicit force integration by using a prediction-relaxation approach. It is similar in concept to a more involved implicit scheme, but it is fast and straightforward to implement. Particles are moved according to their velocities and then their positions are relaxed, subject to positional constraints (Figure 2). At the end of the timestep, velocities are recomputed by subtracting previous positions from relaxed position. Because of this velocity recomputation, the relaxation displacements are equivalent to impulses applied to the velocity at the beginning of the timestep. These impulses being computed near the end position (on a path between the predicted position and the end position), the prediction-relaxation scheme bears some similarities with an implicit scheme in which the force exists at the unknown end configuration. Intuitively, using forces that exist further in time prevents instabilities by predicting difficult situations and reacting before they actually occur. Mathematical considerations regarding the prediction-relaxation scheme can be found in the Appendix.

Our simulation step is detailed in Algorithm 1. First we update particle velocities according to gravity and viscosity (lines 1 to 5). Then we save the previous positions and move

Algorithm 1: Simulation step.

```
1.  foreach particle i
2.      // apply gravity
3.      $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t \mathbf{g}$
4.  // modify velocities with pairwise viscosity impulses
5.  applyViscosity                        // (Section 5.3)
6.  foreach particle i
7.      // save previous position
8.      $\mathbf{x}_i^{\text{prev}} \leftarrow \mathbf{x}_i$
9.      // advance to predicted position
10.     $\mathbf{x}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
11. // add and remove springs, change rest lengths
12. adjustSprings                         // (Section 5.2)
13. // modify positions according to springs,
14. // double density relaxation, and collisions
15. applySpringDisplacements              // (Section 5.1)
16. doubleDensityRelaxation               // (Section 4)
17. resolveCollisions                     // (Section 6)
18. foreach particle i
19.     // use previous position to compute next velocity
20.     $\mathbf{v}_i \leftarrow (\mathbf{x}_i - \mathbf{x}_i^{\text{prev}})/\Delta t$
```

the particles according to their velocities. Lines 12 to 15 modify spring rest lengths and apply spring forces as particle displacements. Volume conservation, anti-clustering, and surface tension are then enforced (line 16). Finally, collisions between particles and static/dynamic bodies are resolved, and particle velocities are recomputed.

4. Double Density Relaxation

Our double density relaxation is a simplified and extended formulation of the SPH paradigm. The impulses exchanged between particles depend on two different measures of their neighbor density.

4.1. Density and Pressure

In an SPH framework, the global goal of minimizing compressibility translates into a local constraint to maintain constant density. The density at particle i is approximated by summing weighted contributions from each neighbor j. We choose the density at particle i to be

$$\rho_i = \sum_{j \in N(i)} (1 - r_{ij}/h)^2 \tag{1}$$

where $r_{ij} = |\mathbf{r}_{ij}|$, $\mathbf{r}_{ij} = \mathbf{x}_j - \mathbf{x}_i$, and $N(i)$ denotes the set of neighboring particles that are closer than the interaction radius h. This form of density is not a true physical property; it is simply a number quantifying how the particle relates to its neighbors. We tried various other density kernel shapes, such as the unbounded $(r/h)^{-2}$ or the bell shaped polynomial $(1-(r/h)^2)^3$, but we obtained our best results with the quadratic spike $(1-r/h)^2$.

We define a pseudo-pressure P_i proportional to the difference between the current density ρ_i and a rest density ρ_0

$$P_i = k(\rho_i - \rho_0) \tag{2}$$

where k is a stiffness parameter. To keep the formulation as simple as possible, we combined the normalizing constant, that usually scales the density in the SPH literature, with the parameters k and ρ_0. We also consider that all the particles have the same mass. This way, mass cancels out and does not have to be included in the equations.

4.2. Incompressibility Relaxation

The incompressibility relaxation is implemented as a global loop over each particle i. An iteration of this loop is composed of two passes on each neighbor j of particle i. The first pass estimates the local density at particle i by summing the weighted contributions of its neighbors (Equation 1). If this density is higher than the rest density ρ_0, neighbors will be pushed away. If it is lower, they will be pulled closer. Pushing and pulling neighbors is done in the second pass on the neighbors of particle i, in which each pair ij exchanges a displacement.

The density relaxation displacement between two particles is proportional to the pseudo-pressure, weighted by the linear kernel function:

$$\mathbf{D}_{ij} = \Delta t^2 P_i (1 - r_{ij}/h)\hat{\mathbf{r}}_{ij} \tag{3}$$

where $\hat{\mathbf{r}}_{ij}$ is the unit vector from particle i to particle j. Since force is integrated twice in time to get displacement, a factor Δt^2 is included in Equation 3 to preserve timestep duration independence (see the Appendix). We apply this displacement directly by modifying the predicted position of particle j. An equal and opposite displacement is applied to particle i, enforcing the action-reaction principle and thus linear momentum conservation. The displacement being directed along $\hat{\mathbf{r}}_{ij}$, angular momentum is conserved. The linear ramp $(1 - r/h)$ scales the magnitude of the displacement depending on the distance between particles. Note that this pressure kernel is proportional to the derivative of the associated density kernel. This particular relation provides a consistent algorithm, as it is theoretically justified in the SPH literature (for example in [MCG03]).

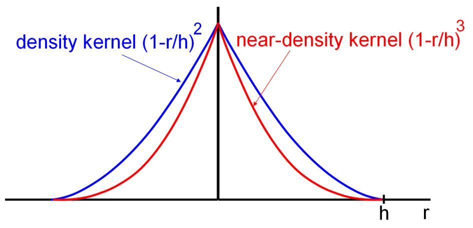
4.3. Near-Density and Near-Pressure

As described, this relaxation procedure does not prevent particle clustering. A particle can reach rest density by strongly pulling a small number of neighbors. The fluid then separates into a collection of independent clusters, exhibiting no coherency. In preliminary tests, we followed an approach similar to Steele *et al.* [SCED04] and tried to prevent clustering by adding a simple distance-dependent repulsion force, but it caused visible artifacts when simulating low viscosity fluid.

We solve the clustering problem by adding instead a second context-dependent pressure term. This new force does not simply depend on the distance between particles, but depends on a new density – the near-density – computed with a different kernel function. We define near-density as

$$\rho_i^{\text{near}} = \sum_{j \in N(i)} (1 - r_{ij}/h)^3 \tag{4}$$

which uses a kernel with a sharper spike than our simple density. Again, we tried various other spike shapes, but this simple cubic gave good results in our tests.



In order to make the corresponding force exclusively repulsive, we define near-pressure as

$$P_i^{\text{near}} = k^{\text{near}} \rho_i^{\text{near}} \tag{5}$$

which is analogous to Equation 2, but with vanishing rest density.

As for simple pressure, near-pressure produces a displacement for each particle pair in the second pass of the relaxation. Augmented with this new term, Equation 3 becomes

$$\mathbf{D}_{ij} = \Delta t^2 \left(P_i (1 - r_{ij}/h) + P_i^{\text{near}} (1 - r_{ij}/h)^2 \right) \hat{\mathbf{r}}_{ij} \tag{6}$$

where the quadratic kernel defines how near-repulsion is applied to neighbors. Again, the near-pressure kernel is proportional to the derivative of the near-density kernel.

Algorithm 2 summarizes our approximate volume conservation and anti-clustering method. For each particle, density and near-density are computed (Equations 1 and 4). Pressure and near-pressure are then evaluated (Equations 2 and 5). Finally, the second loop on neighbors applies the displacements to the particles (Equation 6). Neighbor particles j are immediately moved, but we sum the displacements of particle i and apply them only at the end of its relaxation step. This prevents any bias that could result from a fixed neighbor processing order. The order in which particles i are relaxed is randomized but is not modified throughout the simulation.

The result of double density relaxation is a coherent fluid representation in which particles tend to be at equal distance from all immediate neighbors. However, directly enforcing a constant immediate neighbor distance with a Lennard-Jones-like force can lead to undesirable artifacts due to particles aligning in a rigid pattern. Here, near-density minimization smoothly restricts how the target density can be approached, and no rigid patterns appear.

Algorithm 2: Double density relaxation.

```
1.  foreach particle $i$
2.      $\rho \leftarrow 0$
3.      $\rho^{\text{near}} \leftarrow 0$
4.      // compute density and near-density
5.      foreach particle $j \in$ neighbors( $i$ )
6.          $q \leftarrow r_{ij}/h$
7.          if $q < 1$
8.              $\rho \leftarrow \rho + (1-q)^2$
9.              $\rho^{\text{near}} \leftarrow \rho^{\text{near}} + (1-q)^3$
10.     // compute pressure and near-pressure
11.     $P \leftarrow k(\rho - \rho_0)$
12.     $P^{\text{near}} \leftarrow k^{\text{near}}\rho^{\text{near}}$
13.     $\mathbf{dx} \leftarrow 0$
14.     foreach particle $j \in$ neighbors( $i$ )
15.         $q \leftarrow r_{ij}/h$
16.         if $q < 1$
17.             // apply displacements
18.             $\mathbf{D} \leftarrow \Delta t^2 (P(1-q) + P^{\text{near}}(1-q)^2)\hat{\mathbf{r}}_{ij}$
19.             $\mathbf{x}_j \leftarrow \mathbf{x}_j + \mathbf{D}/2$
20.             $\mathbf{dx} \leftarrow \mathbf{dx} - \mathbf{D}/2$
21.     $\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{dx}$
```

4.4. Surface Tension

We observed that besides reducing particle clustering, our method provides another important fluid behavior: surface tension effects. As illustrated in Figure 3 and in the accompanying video [VID], particles group into structures such as sheets, filaments, and drops.

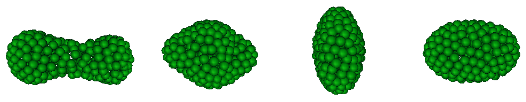


Figure 3: *Oscillating drop.*

Surface tension is physically caused by attraction between molecules. Inside the fluid, this attraction cancels out, but for molecules near the surface, asymmetry in neighbor distribution results in a non-zero net force towards the fluid. Furthermore, this asymmetry changes depending on surface curvature. In view of these physical considerations, surface tension is usually considered to be an external force oriented towards the negative surface normal and with magnitude proportional to the surface curvature.

We can visualize how the combined effect of pressure and near-pressure can produce surface tension. Suppose that density at a particle position is smaller than rest density. Neighboring particles will be pulled by pressure with an impulse proportional to the linear kernel, and then pushed by near-pressure with an impulse proportional to the sharp-spiked quadratic kernel. Neighbors that are further away will tend to move more since they are not affected much by near-pressure. The smooth repulsion of the nearest particles causes an indirect long-distance attraction. This context-aware attraction leads to smooth and stable particle structures.

5. Viscoelasticity

Viscoelastic behavior is introduced in our model through three sub-processes: elasticity, plasticity, and viscosity. Elasticity is obtained by inserting springs between particles, plasticity comes from the modification of the spring rest lengths, and viscosity is introduced by exchanging radial impulses determined by particle velocity differences.

5.1. Elasticity

To simulate elastic behavior, we add springs between pairs of neighboring particles. Springs create displacements on the two attached particles. The displacement magnitude is proportional to $L - r$, where r is the distance between the particles and L is the spring rest length. It is also scaled with the factor $1 - L/h$, which gradually reduces to zero the force exerted by long springs. The process is detailed in Algorithm 3.

Algorithm 3: Spring displacements.

```
1.  foreach spring $ij$
2.      $\mathbf{D} \leftarrow \Delta t^2 k^{\text{spring}} (1 - L_{ij}/h)(L_{ij} - r_{ij})\hat{\mathbf{r}}_{ij}$
3.      $\mathbf{x}_i \leftarrow \mathbf{x}_i - \mathbf{D}/2$
4.      $\mathbf{x}_j \leftarrow \mathbf{x}_j + \mathbf{D}/2$
```

5.2. Plasticity

A perfectly elastic substance always remembers its fixed rest shape, and fights external forces to recover it. On the other hand, a perfectly plastic substance considers its current shape as its rest shape. In general, plasticity can be thought of as the rate at which a substance forgets its past.

This intuitive view of elasto-plasticity leads to our dynamic rest length spring scheme. At each timestep, springs are added or removed, and their rest lengths change depending on their current lengths.

The rate of rest length change of a linearly plastic spring is proportional to its deformation:

$$\Delta L = \Delta t \; \alpha \; (r - L) \tag{7}$$

where α is the plasticity constant.

A linearly plastic material slowly flows until all forces reach equilibrium. To model substances such as clay, which change shape under the significant pressure given by one's fingers but resists small forces such as gravity, a non-linear plasticity model is needed.

Algorithm 4: Spring adjustment.

```
1.  foreach neighbor pair $ij$, $(i < j)$
2.      $q \leftarrow r_{ij}/h$
3.      if $q < 1$
4.          if there is no spring $ij$
5.              add spring $ij$ with rest length $h$
6.          // tolerable deformation = yield ratio * rest length
7.          $d \leftarrow \gamma L_{ij}$
8.          if $r_{ij} > L + d$            // stretch
9.              $L_{ij} \leftarrow L_{ij} + \Delta t\, \alpha\, (r_{ij} - L - d)$
10.         else if $r_{ij} < L - d$       // compress
11.             $L_{ij} \leftarrow L_{ij} - \Delta t\, \alpha\, (L - d - r_{ij})$
12. foreach spring $ij$
13.     if $L_{ij} > h$
14.         remove spring $ij$
```

Our plastic spring model is inspired by the *von Mises* condition [Fun65], which states that plastic flow should occur only if the deformation is large enough. Translated into the spring system, this means that L should be changed only if $|r-L|$ is larger than some fraction of L (Figure 4). This fraction is specified by the yield ratio, denoted γ, for which we typically choose a value between 0 and 0.2. The rest length increment can then be written

$$\Delta L = \Delta t\ \alpha\ \mathrm{sign}(r-L)\ \max(0, |r-L| - \gamma L) \qquad (8)$$

which becomes identical to Equation 7 when $\gamma = 0$.

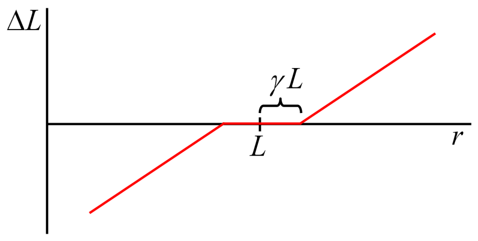


Figure 4: *Rest length change as a function of current length.*

As the fluid moves, springs must be added and removed. A spring is added between two particles if their distance becomes smaller than the radius of interaction h, and is later removed if its rest length becomes larger than h. Pseudocode for our complete spring adjusting procedure is given in Algorithm 4. Additional control on viscoelastic behavior can be gained by using separate values of α and γ for compression and stretching. For example, a stickier material can be simulated by setting γ^{compress} to zero while letting γ^{stretch} take some non-zero value.

5.3. Viscosity

Viscosity has the effect of smoothing the velocity field. It is applied as radial pairwise impulses between neighboring

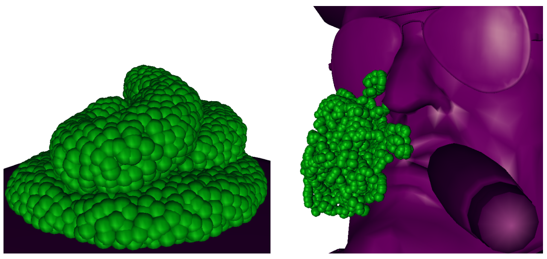


Figure 5: *Various plastic behaviors.*

particles. These impulses modify particle velocities at the beginning of the timestep, before moving them to their predicted positions.

Algorithm 5: Viscosity impulses.

```
1.  foreach neighbor pair $ij$, $(i < j)$
2.      $q \leftarrow r_{ij}/h$
3.      if $q < 1$
4.          // inward radial velocity
5.          $u \leftarrow (\mathbf{v}_i - \mathbf{v}_j) \cdot \hat{\mathbf{r}}_{ij}$
6.          if $u > 0$
7.              // linear and quadratic impulses
8.              $\mathbf{I} \leftarrow \Delta t(1-q)(\sigma u + \beta u^2)\hat{\mathbf{r}}_{ij}$
9.              $\mathbf{v}_i \leftarrow \mathbf{v}_i - \mathbf{I}/2$
10.             $\mathbf{v}_j \leftarrow \mathbf{v}_j + \mathbf{I}/2$
```

Algorithm 5 shows how viscosity changes particle velocities. We measure how fast particle j is moving towards particle i by projecting the velocity difference on $\hat{\mathbf{r}}_{ij}$ (line 5). For non-viscous fluid, viscosity is only used to handle collisions, and we therefore apply impulses only if particles are running into each other.

The impulse dependence on distance is captured by the linear kernel $(1 - r_{ij}/h)$, and the factor $(\sigma u + \beta u^2)$ controls the viscosity's linear and quadratic dependences on velocity. This formulation is inspired by usual SPH techniques [DG96].

If a highly viscous behavior is desired, σ can be increased. For less viscous fluids, only β should be set to a non-zero value. The quadratic term prevents particle interpenetration by removing high inward velocity, but avoids smoothing the interesting features of the velocity field.

Viscosity impulses are applied sequentially to particle pairs. The ordering of particle pairs can theoretically bias the solution, but no visible artifacts were observed in our tests.

6. Interactions with Objects

We have integrated our fluid simulation into a rigid-body system, enabling interesting simulation scenarios such as floating objects and liquid sticking on surfaces. During the collision stage, the fluid is considered to be an assembly of rigid spheres exchanging impulses with surrounding objects.

6.1. Collisions

For each object, a signed distance field is sampled and stored in a grid structure. For each particle we obtain the interpolated distance value d. A collision is identified when d is smaller than the particle collision radius R. For the colliding particle we obtain the object normal $\hat{\mathbf{n}}$ using the distance field gradient.

Our rigid-body solver is impulse-based, as in [GBF03]. In this kind of framework, object penetrations are resolved by sequentially applying impulses between bodies. This method for particle-body collisions leads to instabilities and makes it impossible to simulate floating objects. We therefore propose the three step method illustrated in Figure 6.

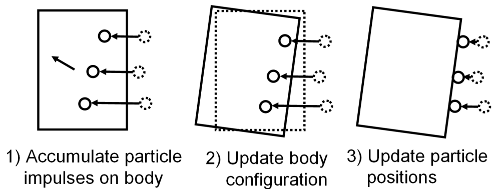

1) Accumulate particle impulses on body
2) Update body configuration
3) Update particle positions

Figure 6: *Particle-body interactions.*

In the first step, corresponding to lines 1 to 7 of Algorithm 6, bodies are advanced and impulses due to penetrating particles are accumulated into force and torque buffers. In the second step (lines 8 to 11), the velocity $\mathbf{V}$ and angular velocity ω of each body is updated using the accumulated force and torque. The bodies are advanced again from the initial configuration according to these new velocities, but the positions of the colliding particles are not yet modified. The usual body-body collisions and contacts are then resolved, and the bodies reach their final positions and velocities. The third step (lines 12 to 15) updates the particle positions to remove penetration velocities. In this final step, the bodies are considered to have infinite inertia.

Impulses due to particle-body collisions are based on a zero-restitution collision model with wet friction. This model requires the current particle velocity $\mathbf{v}_i$, computed using the difference between the current and previous particle positions. We then compute the particle relative velocity $\bar{\mathbf{v}} = \mathbf{v}_i - \mathbf{v}_p$, where $\mathbf{v}_p$ is the body velocity at the contact point. This velocity is separated into normal and tangential components:

$$\bar{\mathbf{v}}^{\text{normal}} = (\bar{\mathbf{v}} \cdot \hat{\mathbf{n}})\hat{\mathbf{n}} \tag{9}$$

$$\bar{\mathbf{v}}^{\text{tangent}} = \bar{\mathbf{v}} - \bar{\mathbf{v}}^{\text{normal}}. \tag{10}$$

The impulse computed at lines 6 and 13 cancels the normal velocity and removes a fraction of the tangential velocity:

$$\mathbf{I} = \bar{\mathbf{v}}^{\text{normal}} - \mu\, \bar{\mathbf{v}}^{\text{tangent}} \tag{11}$$

Algorithm 6: Particle-body interactions.

```
1.  foreach body
2.      save original body position and orientation
3.      advance body using $\mathbf{V}$ and $\omega$
4.      clear force and torque buffers
5.      foreach particle inside the body
6.          compute collision impulse $\mathbf{I}$
7.          add $\mathbf{I}$ contribution to force and torque buffers
8.  foreach body
9.      modify $\mathbf{V}$ with force and $\omega$ with torque
10.     advance from original position using $\mathbf{V}$ and $\omega$
11. resolve collisions and contacts between bodies
12. foreach particle inside a body
13.     compute collision impulse $\mathbf{I}$
14.     apply $\mathbf{I}$ to the particle
15.     extract the particle if still inside the body
```

where μ is a friction parameter enabling slip ($\mu = 0$) or no-slip ($\mu = 1$) boundary conditions.

In some difficult situations, such as when a particle is already inside an object at the beginning of the timestep, removing the penetration velocity is not sufficient. We solve this issue by using the distance field to extract the offending particles at the end of the collision stage (line 15).

As it is common with any distance field-based collision detection, collisions can be missed if the object is too thin with respect to the velocity of the particles.

6.2. Stickiness

As they are now described, particles will detach and fall from objects. We implemented a simple method to make them stick, even underneath a horizontal surface.

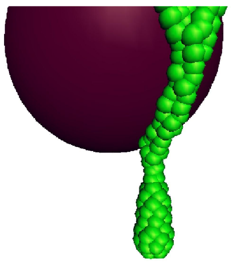


Figure 7: *Particles sliding underneath a sphere, before forming a drop.*

The idea is to modify the impulse computed at Equation 11 by adding an attraction term for particles that are close to an object. Let d_i be the distance between the object and the particle collision surface. An attraction impulse occurs when d_i is smaller than a fixed distance d^{stick}. To avoid artifacts such as particles jumping to the surface, d^{stick}

should stay small enough with respect to the particle interaction range h. We use the following formulation:

$$\mathbf{I}^{\text{stick}} = -\Delta t\, k^{\text{stick}} d_i \left(1 - \frac{d_i}{d^{\text{stick}}}\right) \hat{\mathbf{n}} \qquad (12)$$

which is maximal at distance $d^{\text{stick}}/2$. Lines 5 and 12 of Algorithm 7 must now consider all particles in the attraction range.

7. Implementation and Results

7.1. Neighbor Search

Finding all particles that are within the interaction radius of a particle is a frequent task in any particle-based method. Since the interaction radius h is constant and identical for each particle, it makes sense to use a simple regular spatial hashing grid with cube size h. Each cube stores a list of enclosed particles, which is updated each time a particle moves from one cube to another. All neighbors reside in the particle's cube and its 26 neighboring cubes. Furthermore, we avoid using a fixed cube lattice by only instantiating cubes that contain particles. The cubes are stored in a hash table, indexed with their 3D index. The animation scenario can thus take place in a virtually infinite domain, and no information on the future location of the fluid is needed at the beginning of the simulation. This spatial hashing method is similar to Teschner *et al.* [THM*03].

7.2. Rendering

For fast previsualization, particles are simply rendered as polygonized spheres. High quality rendering uses a surface mesh extracted with the marching cube algorithm [LC87].

The marching cube extracts an isosurface of a scalar function defined by the particles. We use the function

$$\phi(\mathbf{x}) = \left(\sum_j (1 - r/h)^2\right)^{\frac{1}{2}} \qquad (13)$$

where $r = |\mathbf{x} - \mathbf{x}_j|$. The square root ensures that the function behaves like a distance function, *i.e.,* has an almost constant slope around the extracted isosurface. This is important for moving isosurfaces because the location of the surface in a cube is linearly interpolated. If the function had a changing slope around the isovalue, temporal artifacts would appear as particles move. Simply using a sum of cones could achieve this goal, but the result would not be smooth enough.

As in the case of neighbor search, the cube grid used for isosurface extraction is sparse and dynamic. The cube size h^{surface} is independent from the particle interaction radius h, and is typically smaller if a smooth surface mesh is desired. A spatial hash table is used to store the non-zero grid points as particles add their contributions to the implicit function.

7.3. Results

The results, shown in the accompanying video sequences [VID] and in Figures 8 to 12, took an average of 2 seconds per frame to simulate, with typically 20000 particles. A 10 fps interactive session can simulate 1000 particles, which is enough to create relatively complex splashing effects.

These timings include surface extraction with the marching cube algorithm and *OpenGL* display. Surface extraction usually takes from 10% to 60% of the computation time. High quality renderings were generated offline with the raytracer *Pixie* [PIX].

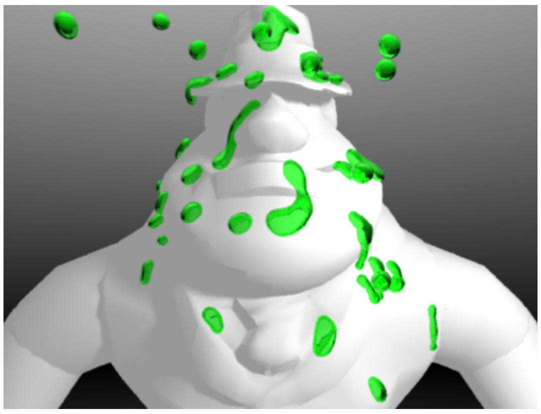


Figure 8: *Viscous rain on a character.*

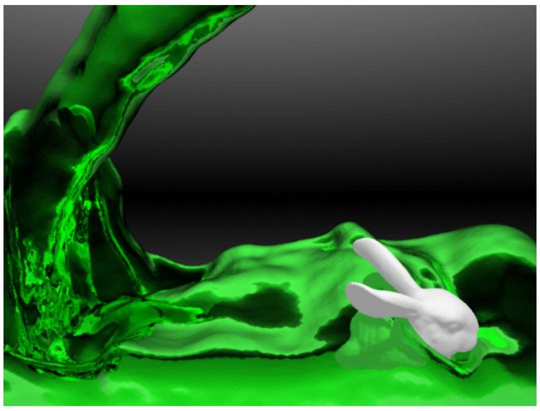


Figure 9: *Pouring liquid in a tank containing a heavy object.*

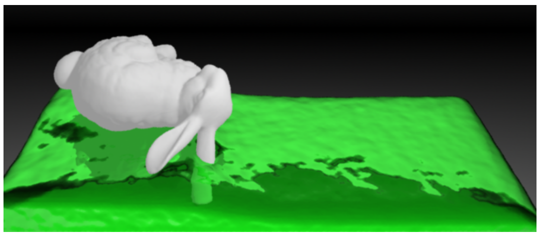


Figure 10: *A floating object.*

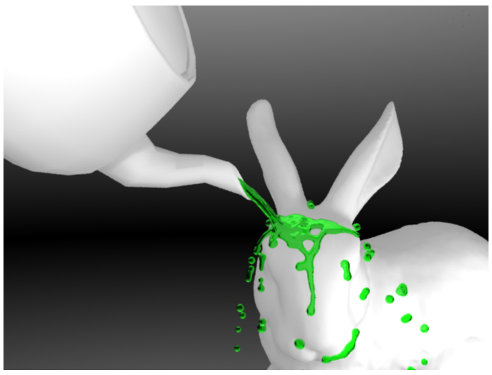


Figure 11: *Pouring liquid on the Stanford Bunny.*

Computing many timesteps per frame can greatly improve volume conservation and overall simulation quality. We used up to 10 timesteps per frame for very fast moving liquid or when volume conservation was critical, but most of the time, only one timestep per frame was computed.

Despite the fact that many parameters control the simulation, tweaking fluid behavior is intuitive and easy. Our typical parameter values for the most important parameters are $\Delta t = 1$ (where the time unit is $1/30$ second), $\rho_0 = 10$, $k = 0.004$, $k^{\text{near}} = 0.01$, $k^{\text{spring}} = 0.3$, and $\alpha = 0.3$. The interaction range h can be specified independently to set the fluid resolution.

8. Conclusion

We have presented a particle-based method to interactively simulate the complex behavior of viscoelastic fluids.

The main contributions of our work are :

1. A simple and flexible method to simulate viscoelasticity with varying rest length springs. Inserting, removing, and modifying the rest length of springs between particles provide an intuitive control of viscoelasticity. Complex non-linear plastic effects such as clay or gel manipulation can be achieved by simple rest length update rules.
2. A new scheme for the robust simulation of surface tension in particle-based systems, without requiring computation of curvature over the liquid surface. In contrast to typical particle systems, it can produce a smooth surface, yet enables the formation of coherent features such as liquid drops, filaments, and sheets.
3. A stable method for long timesteps. Numerical instabilities that often plague physically-based simulations are significantly reduced, and fast animations can be obtained by computing a single timestep per frame.

The application of our method was geared mostly towards interactive fluid jets, but we demonstrated its flexibility in a number of different situations, such as plastic and elastic deformable 3D models and rain drops. The integration of our fluid solver into a rigid-body system has enabled floating objects, and a simple solution to the stickiness of liquids on objects has further extended the simulation possibilities.

Our method is specifically designed for fast simulation of rapidly moving fluids, but is not as well adapted for simulation of low viscosity liquids such as water. We could adapt the method to simulate water by using an explicit high order accurate integration scheme, and simulating numerous timesteps per frame. The increased computation times would, however, contradict our initial goals of interactivity.

Several other avenues for future work can be identified. Volume conservation could be improved by coupling our method with a more sophisticated technique such as the *Moving-Particle Semi-Implicit* method [KO96]. We would also like to consider several types of particles to animate the interaction between different substances. Simulating air would significantly increase animation realism by enabling bubble formation.

We believe that the simplicity, stability, speed, and versatility of our method should prove very useful for many applications.

Acknowledgements

The authors thank all the LIGUM members for their support. Luc Leblanc deserves a special mention for his help with the renderings. We thank the reviewers for their helpful comments. Financial support from NSERC is acknowledged.

References

[CMHT02] Carlson M., Mucha P. J., Horn R. B. V., Turk G.: Melting and flowing. In *SIGGRAPH/Eurographics Symposium on Computer Animation* (2002), pp. 167–174.

[DG96] Desbrun M., Gascuel M.-P.: Smoothed particles: A new paradigm for animating highly deformable bodies. In *Computer Animation and Simulation '96* (1996), pp. 61–76.

[EMF02] Enright D., Marschner S., Fedkiw R.: Animation and rendering of complex water surfaces. In *SIGGRAPH* (2002), pp. 736–744.

[FF01] Foster N., Fedkiw R.: Practical animations of liquids. In *SIGGRAPH* (2001), pp. 23–30.

[FM96] Foster N., Metaxas D.: Realistic animation of liquids. *Graphical Models and Image Processing 58*, 5 (1996), 471–483.

[FM97] Foster N., Metaxas D.: Modeling the motion of a hot, turbulent gas. In *SIGGRAPH* (1997), pp. 181–188.

[Fun65] Fung Y. C.: *Foundations of Solid Mechanics*. Prentice-Hall, 1965.

[GBF03] GUENDELMAN E., BRIDSON R., FEDKIW R.: Nonconvex rigid bodies with stacking. In *SIGGRAPH* (2003), pp. 871–878.

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

[GM77] GINGOLD R., MONAGHAN J.: Smoothed particle hydrodynamics – theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society 181* (1977), 375.

[KO96] KOSHIZUKA S., OKA Y.: Moving-particle semi-implicit method for fragmentation of incompressible fluid. *Nuclear Science Engineering 123* (July 1996), 421–434.

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH* (1987), pp. 163–169.

[Luc77] LUCY L.: A numerical approach to the testing of the fission hypothesis. *Astronomical Journal 82* (1977), 1013.

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *SIGGRAPH/Eurographics Symposium on Computer Animation* (2003), pp. 154–159.

[MKN*04] MÜLLER M., KEISER R., NEALEN A., PAULY M., GROSS M., ALEXA M.: Point based animation of elastic, plastic and melting objects. In *SIGGRAPH/Eurographics Symposium on Computer Animation* (2004), pp. 141–151.

[Mor00] MORRIS J. P.: Simulating surface tension with smoothed particle hydrodynamics. *International Journal for Numerical Methods in Fluids 33*, 3 (2000), 333–353.

[MP89] MILLER G., PEARCE A.: Globular dynamics: A connected particle system for animating viscous fluids. *Computers & Graphics 13*, 3 (1989), 305–309.

[PIX] sourceforge.net/projects/pixie

[PTB*03] PREMOŽE S., TASDIZEN T., BIGLER J., LEFOHN A., WHITAKER R. T.: Particle-based simulation of fluids. *Computer Graphics Forum 22*, 3 (2003), 401–410.

[Ree83] REEVES W. T.: Particle systems – a technique for modelling a class of fuzzy objects. In *SIGGRAPH* (1983), pp. 359–376.

[SCED04] STEELE K., CLINE D., EGBERT P. K., DINERSTEIN J.: Modeling and rendering viscous liquids. *Journal of Computer Animation and Virtual Worlds 15*, 3-4 (2004), 183–192.

[Sta99] STAM J.: Stable fluids. In *SIGGRAPH* (1999), pp. 121–128.

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

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

[VID] www.iro.umontreal.ca/labs/infographie/papers

Appendix – Prediction-Relaxation Scheme

The integration scheme we use can be written as

$$\begin{aligned}
\mathbf{x}_* &= \mathbf{x}_n + \Delta t\, \mathbf{v}_{n-\frac{1}{2}} \\
\mathbf{x}_* &\leftarrow \mathbf{x}_* + \Delta t^2\, \mathbf{F}^1(\mathbf{x}_*)/m \\
\mathbf{x}_* &\leftarrow \mathbf{x}_* + \Delta t^2\, \mathbf{F}^2(\mathbf{x}_*)/m \\
\mathbf{x}_* &\leftarrow \mathbf{x}_* + \Delta t^2\, \mathbf{F}^3(\mathbf{x}_*)/m \\
&\ldots \\
\mathbf{x}_{n+1} &= \mathbf{x}_* \\
\mathbf{v}_{n+\frac{1}{2}} &= (\mathbf{x}_{n+1} - \mathbf{x}_n)/\Delta t.
\end{aligned}$$

This sequential application of force components is analogous to operator splitting often used in Eulerian simulations [Sta99]. Each stage of force application uses the previously updated position, which can lead to greater stability. For example, it is intuitively correct to compute collisions using positions that have already been modified by other constraints. Allowing a particular constraint to adjust to what happened in previous stages often prevents overshoots and erroneous constraint responses.

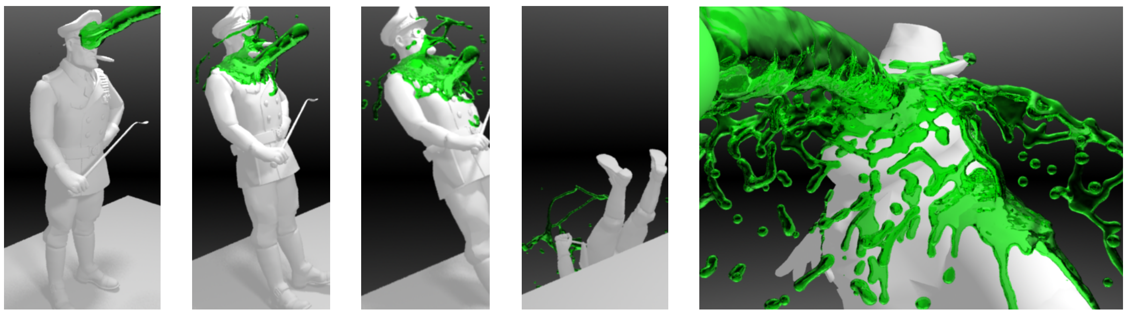
If only one force is used, the integration scheme can be simplified to

$$\begin{aligned}
\mathbf{v}_{n+\frac{1}{2}} &= \mathbf{v}_{n-\frac{1}{2}} + \Delta t\, \mathbf{F}(\mathbf{x}_n + \Delta t\, v_{n-\frac{1}{2}})/m \\
\mathbf{x}_{n+1} &= \mathbf{x}_n + \Delta t\, \mathbf{v}_{n+\frac{1}{2}}
\end{aligned}$$

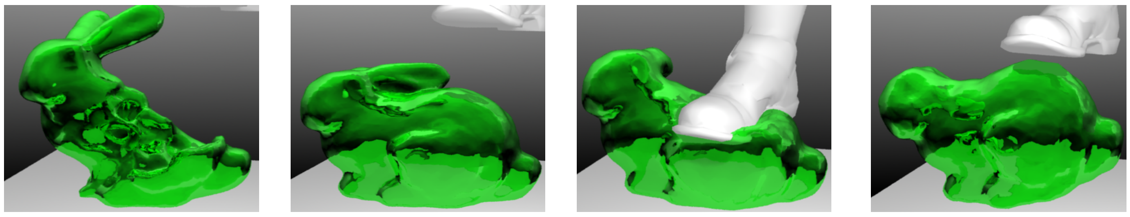
which is similar to the usual leap-frog scheme, but with the force computed at the predicted position. Testing the canonical example $\mathbf{F} = -k\mathbf{x}$ leads to

$$\mathbf{v}_{n+\frac{1}{2}} = \mathbf{v}_{n-\frac{1}{2}} + \Delta t\, (-k\mathbf{x}_n - k\Delta t\, \mathbf{v}_{n-\frac{1}{2}})/m$$

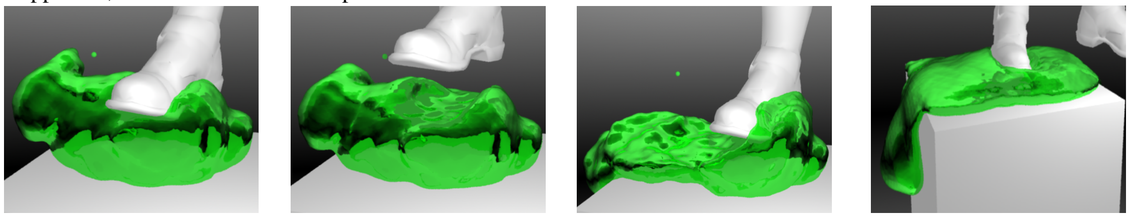
in which the actual force being computed is augmented with the term $-k\Delta t\, \mathbf{v}_{n-\frac{1}{2}}$, an artificial viscosity.



Liquid jet on a dynamic character. Fast detailed splash.



A polygonal mold is filled with liquid, springs are added, and then the mold is removed. The resulting elastic bunny is stepped on, but recovers its rest shape.



Then plasticity is increased and the foot leaves its print. Finally, springs are deleted and the bunny flows away.

Figure 12: *Various illustrations of our particle-based fast fluid simulation.*