Implicit Formulation for SPH-based Viscous Fluids

Tetsuya Takahashi et al.
EUROGRAPHICS 2015

Presented by MyungJin Choi
2015.06.02

Copyright of figures and other materials in the paper belongs to original authors.
1. Introduction

• The first SPH method that uses implicit integration for the full form of viscosity

• The first method that extracts matrix coefficients contributed by second-ring neighbors

• Our method offers the Following advantages:
  ▪ It is efficient
  ▪ It is robust and stable
  ▪ It can generate coiling and buckling phenomena and handle variable viscosity
2. Related Work (1/4)

**Melting and flowing**
[Mark Carlson et al. / 2002 SIGGRAPH]

First enabled stable simulation of high viscous fluid

**Directable Photorealistic Liquids**
[RASMUSSEN N. et al. / 2004 SCA]

Implicit-explicit scheme for the full form of viscosity to correctly handle variable viscosity
2. Related Work (2/4)

Accurate viscous free surfaces for buckling, coiling, and rotating liquids
[BATTY C. et al. / 2008 SIGGRAPH]

It possible to take larger time steps, handle variable viscosity, and generate coiling and buckling

A simple finite volume method for adaptive viscous liquids
[BATTY C. et al. / 2011 SIGGRAPH]

It is for an adaptive tetrahedral fluid simulator
2. Related Work (3/4)

Simulating Liquids and Solid-Liquid Interactions with Lagrangian Meshes
[CLAUSEN P. et al. / 2013 TOG]

A Lagrangian FEM that can handle elastic, plastic, and fluid materials in a unified manner

Discrete viscous sheets
[BATTY C. et al. / 2012 TOG]

Dimensionally reduced discrete methods and generated coiling and buckling
2. Related Work (4/4)

Fast Simulation of Viscous Fluids with Elasticity and Thermal Conductivity Using Position-Based Dynamics
[Takahashi T. et al. / 2014 C&G]

For unified framework of Position-based dynamics

Deformation embedding for point-based elastoplastic simulation
[Jones B. et al. / 2014 TOG]

A deformation-based method to handle varying mass materials
3. Fundamentals for Simulating Viscous Fluids Formulations

- The Navier-Stokes equations for particle \( i \) can be described as

\[
\rho_i \frac{d\mathbf{u}_i}{dt} = -\nabla p_i + \nabla \cdot \mathbf{s}_i + \frac{\rho_i}{m} \mathbf{F}^\text{ext}_i, \tag{1}
\]

\[
\mathbf{s}_i = \mu_i \left( \nabla \mathbf{u}_i + (\nabla \mathbf{u}_i)^T \right), \tag{2}
\]

- \( \rho_i \): density of particle \( i \)
- \( t \): time
- \( \mathbf{u}_i \): \( [u_i, v_i, w_i]^T \) (velocity)
- \( \mathbf{S}_i \): viscous stress tensor
- \( m \): mass
- \( \mathbf{F}^\text{ext}_i \): external force
- \( \mu_i \): dynamic viscosity
3. Algorithm (1/2)

Algorithm 1 Procedure of our method

1: // $j$: neighbor particle of $i$
2: // $W_{ij}$: kernel with a kernel radius $h$
3: for all particle $i$ do
4: find neighbor particles
5: for all particle $i$ do
6: apply external force $\mathbf{u}_i^* = \mathbf{u}_i^t + \Delta t F_{i}^{\text{ext}} / m$
7: for all particle $i$ do
8: solve viscosity using Eqs. (3) and (4) // § 4
9: for all particle $i$ do
10: compute $p_i$ using a particle-based fluid solver
11: for all particle $i$ do
12: compute $F_{i}^P = -m^2 \sum_j \left( \frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W_{ij}$
13: for all particle $i$ do
14: integrate particle velocity $\mathbf{u}_{i}^{t+1} = \mathbf{u}_{i}^{**} + \Delta t F_{i}^P / m$
15: integrate particle position $\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \Delta t \mathbf{u}_i^{t+1}$
3. Algorithm (2/2)

- More details of Eq.(2)

\[ s_i = \mu_i \left( \nabla u_i + (\nabla u_i)^T \right), \]  
\[ u_i^{**} = u_i^* + \frac{\Delta t}{\rho_i} \nabla \cdot s_i^{**}, \]
\[ s_i^{**} = \mu_i \left( \nabla u_i^{**} + (\nabla u_i^{**})^T \right). \]

- \( u_i^* \): first intermediate velocity
- \( u_i^{**} \): second intermediate velocity
- \( S_i^{**} \): intermediate viscous stress tensor
- \( \mu_i \): dynamic viscosity
Discretization of Eq.(3) and (4) using implicit integration in SPH framework

\[ \mathbf{u}_i = \mathbf{u}_i^* + m\Delta t \sum_j \left( \frac{s_i}{\rho_i^2} + \frac{s_j}{\rho_j^2} \right) \nabla W_{ij}, \tag{5} \]

\[ s_i = \mu_i \sum_j V_j \left( (\mathbf{u}_j - \mathbf{u}_i) \nabla W_{ij}^T + \nabla W_{ij} (\mathbf{u}_j - \mathbf{u}_i)^T \right). \tag{6} \]
4.1 Implicit Integration for Full Form of Viscosity (2/3)

By substituting $s_i$ in Eq. (6) into Eq. (5) and arranging the terms in these equations, we obtain an implicit formulation:

$$
\mathbf{u}_i + \hat{m} \sum_j (\hat{\mu}_i \mathbf{Q}_{ij} + \hat{\mu}_j \mathbf{Q}_{jk}) \nabla W_{ij} = \mathbf{u}_i^*, \quad (7)
$$

$$
\mathbf{Q}_{ij} = \begin{bmatrix}
2 \sum_j a_{ij,x} u_{ij} & q_{ij,xy} & q_{ij,xz} \\
q_{ij,xy} & 2 \sum_j a_{ij,y} v_{ij} & q_{ij,yz} \\
q_{ij,xz} & q_{ij,yz} & 2 \sum_j a_{ij,z} w_{ij}
\end{bmatrix}, \quad (8)
$$

$$
q_{ij,xy} = \sum_j (a_{ij,y} u_{ij} + a_{ij,x} v_{ij}),
q_{ij,xz} = \sum_j (a_{ij,z} u_{ij} + a_{ij,x} w_{ij}),
q_{ij,yz} = \sum_j (a_{ij,z} v_{ij} + a_{ij,y} w_{ij}),
$$

$\hat{m} : m \Delta t$

$\hat{\mu}_i : \mu_i / \rho_i^2$

$k : \text{neighbor particle of } j$

$a_{ij} : [a_{ij,x}, a_{ij,y}, a_{ij,z}]^T = V_j \nabla W_{ij} = V_j [\nabla W_{ij,x}, \nabla W_{ij,y}, \nabla W_{ij,z}]^T$

$u_{ij} : u_i - u_j$

$v_{ij} : v_i - v_j$

$w_{ij} : w_i - w_j$
4.1 Implicit Integration for Full Form of Viscosity (3/3)

- This implicit formulation Eq. (7) is a linear system and can be rewritten in a matrix form as

$$
{\bf C}U = {\bf U}^*
$$

$${\bf C} : \text{coefficient matrix (}3N \times 3N, \ N \text{ is number of particles)}$$

$${\bf U} : [..., u_i, v_i, w_i, ...]^T \ (3N \times 1, \ N \text{ is number of particles)}$$
4.2 Sparsity of Coefficient Matrix

- Sparsity of Coefficient Matrix
  - $i$ has radius $h$ and 30 ~ 40 neighbors
  - Minkowski sum $M_i$ has radius $2h$ and 240 ~ 320 neighbors
  - Non-zero values for each velocity component can be 960
4.3 Solver and Coefficient Extraction (1/4)

- By substituting $Q_{ij}$ in Eq. (8), we can rewrite Eq. (7) for $x$ component of $u_i, u_i$ as

\[
 u_i + \hat{m} \sum_j \left( \hat{\mu}_i \left( 2 \nabla W_{ij,x} \sum_j a_{ij,x} u_{ij} + \right. \right. \\
 \left. \left. \left. \nabla W_{ij,y} \sum_j (a_{ij,y} u_{ij} + a_{ij,x} v_{ij}) + \nabla W_{ij,z} \sum_j (a_{ij,z} u_{ij} + a_{ij,x} w_{ij}) \right) \right) \\
 + \hat{\mu}_j \left( 2 \nabla W_{ij,x} \sum_k a_{jk,x} u_{jk} + \nabla W_{ij,y} \sum_k (a_{jk,y} u_{jk} + a_{jk,x} v_{jk}) \right. \right. \\
 \left. \left. + \nabla W_{ij,z} \sum_k (a_{jk,z} u_{jk} + a_{jk,x} w_{jk}) \right) \right) = u_i^*. (9)
\]
4.3 Solver and Coefficient Extraction (2/4)

- we further convert Eq. (9) into the following equation to straightforwardly extract coefficients

\[ c_{u_i u_i}, c_{v_i u_i}, c_{w_i u_i}, c_{u_j u_i}, c_{v_j u_i}, c_{w_j u_i}, c_{u_k u_i}, c_{v_k u_i}, c_{w_k u_i} : \]

\[
\begin{bmatrix}
  c_{u_i u_i} \\
  c_{v_i u_i} \\
  c_{w_i u_i}
\end{bmatrix}
\begin{bmatrix}
  u_i \\
  v_i \\
  w_i
\end{bmatrix}
+ \sum_j
\begin{bmatrix}
  c_{u_j u_i} \\
  c_{v_j u_i} \\
  c_{w_j u_i}
\end{bmatrix}
\begin{bmatrix}
  u_j \\
  v_j \\
  w_j
\end{bmatrix}
+ \sum_k
\begin{bmatrix}
  c_{u_k u_i} \\
  c_{v_k u_i} \\
  c_{w_k u_i}
\end{bmatrix}
\begin{bmatrix}
  u_k \\
  v_k \\
  w_k
\end{bmatrix}
= u_i^*,
\]
4.3 Solver and Coefficient Extraction (3/4)

\[ c_{ui,ui} = 1 + \hat{m} \hat{u}_i \left( 2 \omega_{ij,x} \alpha_{ij,x} + \omega_{ij,y} \alpha_{ij,y} + \omega_{ij,z} \alpha_{ij,z} \right), \]
\[ c_{vi,ui} = \hat{m} \hat{u}_i \omega_{ij,y} \alpha_{ij,x}, \]
\[ c_{wi,ui} = \hat{m} \hat{u}_i \omega_{ij,z} \alpha_{ij,x}, \]
\[ c_{uj,ui} = \hat{m} \left( -\hat{u}_i \left( 2 a_{ij,x} \omega_{ij,x} + a_{ij,y} \omega_{ij,y} + a_{ij,z} \omega_{ij,z} \right) + \hat{u}_j \left( 2 \nabla W_{ij,x} \alpha_{jk,x} + \nabla W_{ij,y} \alpha_{jk,y} + \nabla W_{ij,z} \alpha_{jk,z} \right) \right), \]
\[ c_{vj,ui} = \hat{m} \left( -\hat{u}_i a_{ij,x} \omega_{ij,y} + \hat{u}_j \nabla W_{ij,y} \alpha_{jk,x} \right), \]
\[ c_{wj,ui} = \hat{m} \left( -\hat{u}_i a_{ij,x} \omega_{ij,z} + \hat{u}_j \nabla W_{ij,z} \alpha_{jk,x} \right), \]
\[ c_{uk,ui} = -\hat{m} \sum_j \hat{u}_j \left( 2 \nabla W_{ij,x} a_{jk,x} + \nabla W_{ij,y} a_{jk,y} + \nabla W_{ij,z} a_{jk,z} \right), \]
\[ \alpha_{ij} : \left[ \alpha_{ij,x}, \alpha_{ij,y}, \alpha_{ij,z} \right]^T = \sum_j a_{ij} \]
\[ \omega_{ij} : \left[ \omega_{ij,x}, \omega_{ij,y}, \omega_{ij,z} \right]^T = \sum_j \nabla w_{ij} \]

\[ (10) \]
\[ (11) \]
\[ (12) \]
4.3 Solver and Coefficient Extraction (4/4)

Algorithm 1 Procedure of our method

1: // j: neighbor particle of i
2: // \( W_{ij} \): kernel with a kernel radius \( h \)
3: for all particle i do
4:  find neighbor particles
5: for all particle i do
6:  apply external force \( \mathbf{u}_i^t = \mathbf{u}_i^t + \Delta t F_{i}^{\text{ext}} / m \)
7: for all particle i do
8:  solve viscosity using Eqs. (3) and (4) // § 4
9: for all particle i do
10:  compute \( p_i \) using a particle-based fluid solver
11: for all particle i do
12:  compute \( F_{i}^{p} = -m \sum_j \left( \frac{p_i}{\rho_i^j} + \frac{p_j}{\rho_j^i} \right) \nabla W_{ij} \)
13: for all particle i do
14:  integrate particle velocity \( \mathbf{u}_i^{t+1} = \mathbf{u}_i^{t*} + \Delta t F_{i}^{p} / m \)
15: integrate particle position \( \mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \Delta t \mathbf{u}_i^{t+1} \)

Algorithm 3 Algorithm for coefficient extraction

1: initialize a matrix
2: for all fluid particle i do
3:  compute \( \hat{\mathbf{u}}_i \), \( \omega_{ij} \) and \( \sigma_{ij} \)
4:  compute \( \hat{m} \)
5: for all fluid particle i do
6:  initialize storage for \( u_k \), \( v_k \), and \( w_k \)
7:  add \( c_{u_k, \mathbf{u}_i}, c_{v_k, \mathbf{u}_i}, c_{w_k, \mathbf{u}_i}, c_{u_k, \omega_{ij}}, c_{v_k, \omega_{ij}}, c_{w_k, \omega_{ij}}, c_{u_k, \sigma_{ij}}, \) and \( c_{w_k, \sigma_{ij}} \) to the matrix
8: for all fluid particle j do
9:  compute \( \nabla W_{ij} \) and \( a_{ij} \)
10: add \( c_{u_k, \mathbf{u}_j}, c_{v_k, \mathbf{u}_j}, c_{w_k, \mathbf{u}_j}, c_{u_k, \omega_{ij}}, c_{v_k, \omega_{ij}}, c_{w_k, \omega_{ij}}, c_{u_k, \sigma_{ij}}, \) and \( c_{w_k, \sigma_{ij}} \) to the matrix
11: for all fluid particle k do
12:  compute \( a_{jk} \)
13:  add \( c_{u_k, \mathbf{u}_i}, c_{v_k, \mathbf{u}_i}, c_{w_k, \mathbf{u}_i}, c_{u_k, \omega_{ij}}, c_{v_k, \omega_{ij}}, c_{w_k, \omega_{ij}}, c_{u_k, \sigma_{ij}}, \) and \( c_{w_k, \sigma_{ij}} \) to the matrix using the storage
14: for all \( i \)'s storage do
15:  add \( c_{u_k, \mathbf{u}_i}, c_{v_k, \mathbf{u}_i}, c_{w_k, \mathbf{u}_i}, c_{u_k, \omega_{ij}}, c_{v_k, \omega_{ij}}, c_{w_k, \omega_{ij}}, c_{u_k, \sigma_{ij}}, \) and \( c_{w_k, \sigma_{ij}} \) to the matrix
4.4 Implementation Details and Algorithm

- When fluid particles collide with solid particles, we use explicit viscosity integration for fluid particles with low viscosity while using Dirichlet boundary condition
  - namely setting averaged solid particle velocities $\mathbf{u}_{\text{solid}}$ to fluid particles if viscosity of the fluid particles is higher than a criterion $\mu_{\text{Dirichlet}}$

**Algorithm 2 Algorithm for solving viscosity**

1: assemble the matrix // see Appendix A
2: solve the linear system with CG
3: for all fluid particle $i$ do
   4: if $\mu_{\text{Dirichlet}} < \mu_i \land$ neighbor solid particle exists then
   5: enforce solid boundary condition $\mathbf{u}_i = \mathbf{u}_{\text{solid}}$
5. Result

- Implementation
  - C++ and Open MP 2.0
  - IISPH as an incompressible fluid solver
  - z-index neighbor search method

- Setting
  - Intel Core i7 3.40 GHz CPU and RAM 16.0 GB
  - Physically-based renderer Mitsuba.
5.1 Numerical Stability

- Our implicit method successfully simulates the bunny with a large time step and high viscosity
  - SPH fluids for viscous jet buckling
  - [ANDRADE LUIZ F. D. S. et al. / 2014 SIBGRAPI]
5.2 Performance

- We can take a 260.0 times larger time step than the method of Andrade *et al.* and more fast
5.3 Variable Viscosity

- An example of a dragon consisting of particles with different viscosities from 0.0 (light green) to 800.0 kg/(ms) (dark green)
5.4 Buckling and Coiling (1/2)

- Buckling

![Laplacian form vs Full form diagram]
5.4 Buckling and Coiling (2/2)

- Coiling

![Diagram showing low and high viscosity](image)
6. Discussions and Limitations

• Robustness
  ▪ More robust and allows large step
  ▪ But, Our method may not generate plausible fluid behaviors
    • Very large step, very high viscosity and resolution

• Solver
  ▪ Jacobi method
    • It is able with small time step, low viscosity and low resolution
  ▪ MICCG
    • More fast than Jacobi method but slow than CG method
6. Discussions and Limitations

• Performance
  ▪ Solving our viscosity formulation generally occupies more than 90% of the whole computational time
    • It can be improved by using precomputation

• Memory
  ▪ Preserving a coefficient matrix requires a large memory
    • e.g. 12 GB memory for 500k particles, due to 1k of 8 byte double values for 3 velocity components of 500k particles

• Scalability
  ▪ The size of a matrix grows proportionally to the number of particles
7. Conclusion and Future Work

• We proposed a new SPH-based implicit formulation for the full form of viscosity.
  ▪ efficient
  ▪ stable viscous fluid simulations
    • Larger time steps
    • Higher viscosities
    • Resolutions

• We additionally presented a novel coefficient extraction method for a sparse matrix that involves second-ring neighbors to efficiently solve a linear system with a CG solver