Implicit Formulation for SPH-based Viscous Fluids

Tetsuya Takahashi *et al.*
EUROGRAPHICS 2015

Presented by MyungJin Choi

2015.06.02

Computer Graphics @ Korea University

KOREA UNIVERSITY

KUCG

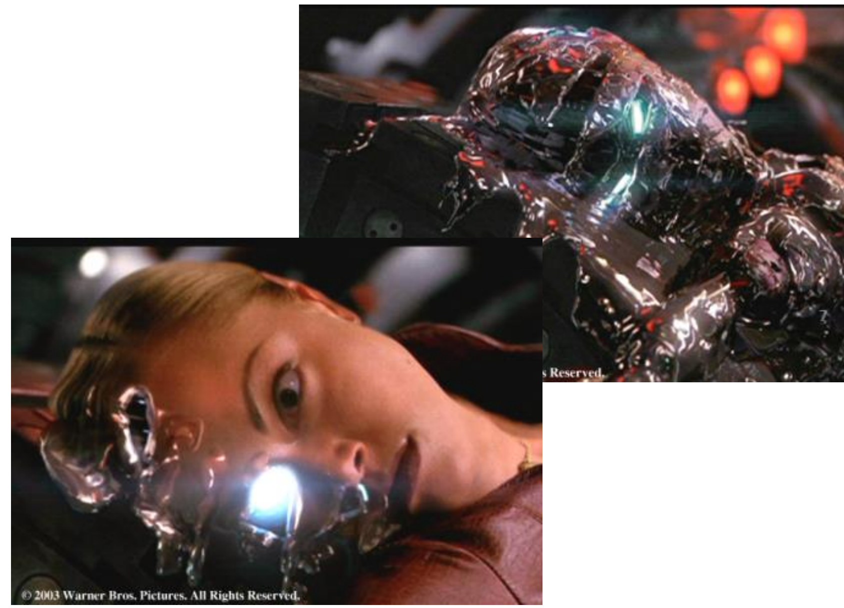
1. Introduction

- 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]

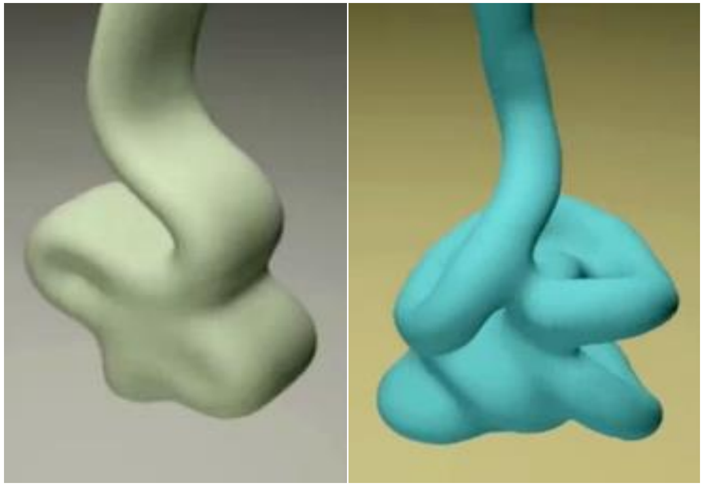
First enabled stable simulation of high viscous fluid



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

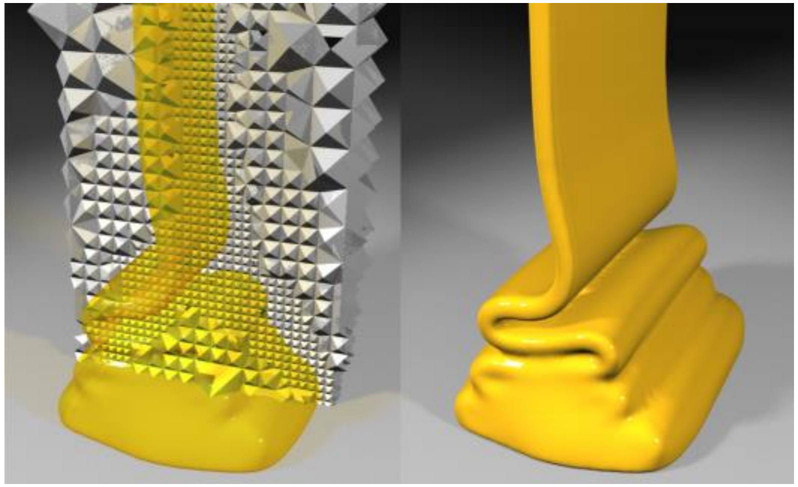
Implicit-explicit scheme for the full form of viscosity to correctly handle variable viscosity

2. Related Work (2/4)



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

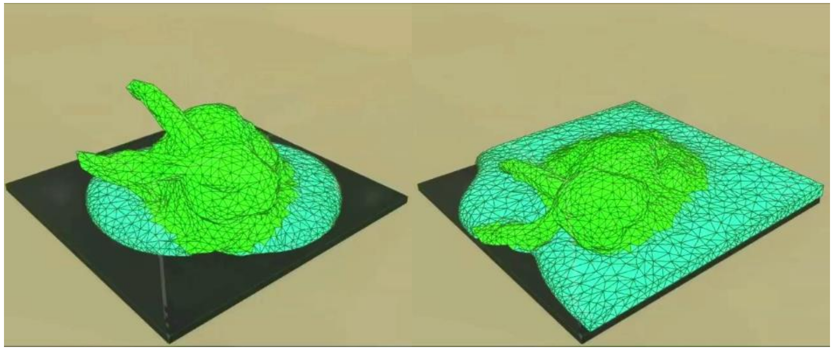
It possible to take larger time steps,
handle variable viscosity,
and generate coiling and buckling



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

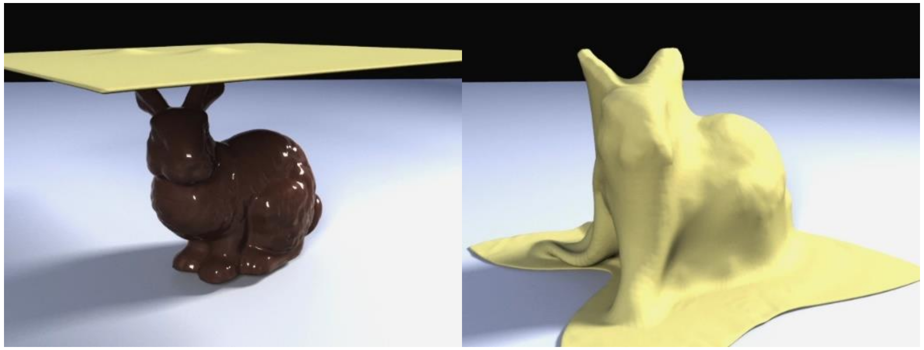
It is for an adaptive tetrahedral fluid simulator

2. Related Work (3/4)



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

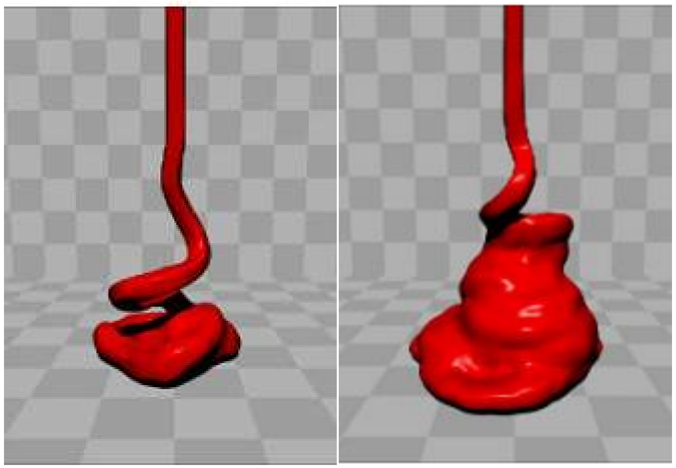
A Lagrangian FEM that can handle elastic, plastic, and fluid materials in a unified manner



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

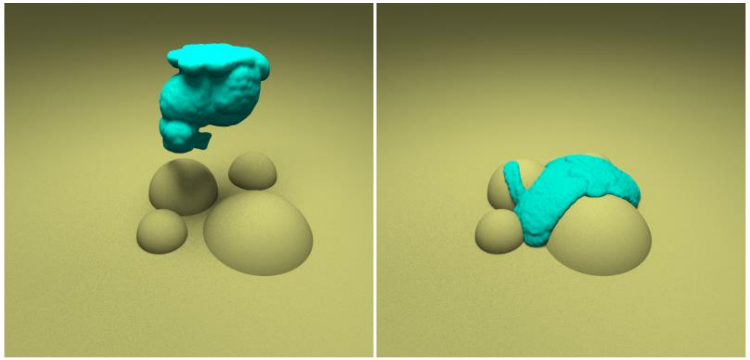
Dimensionally reduced discrete methods and generated coiling and buckling

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

$$\rho_i \frac{d\mathbf{u}_i}{dt} = -\nabla p_i + \nabla \cdot \mathbf{s}_i + \frac{\rho_i}{m} \mathbf{F}_i^{\text{ext}}, \quad (1)$$

$$\mathbf{s}_i = \mu_i \left(\nabla \mathbf{u}_i + (\nabla \mathbf{u}_i)^T \right), \quad (2)$$

ρ_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
$\mathbf{F}_i^{\text{ext}}$: external force
μ_i : dynamic viscosity

3. Algorithm (1/2)

Algorithm 1 Procedure of our method

```
// j: neighbor particle of i
// W_ij: kernel with a kernel radius h
for all particle i do
    find neighbor particles
for all particle i do
    apply external force u_i^* = u_i^t + ΔtF_i^ext/m
for all particle i do
    solve viscosity using Eqs. (3) and (4) // § 4
for all particle i do
    compute p_i using a particle-based fluid solver
for all particle i do
    compute F_i^p = −m^2 Σ_j (p_i/ρ_i^2 + p_j/ρ_j^2) ∇W_ij
for all particle i do
    integrate particle velocity u_i^{t+1} = u_i^{**} + ΔtF_i^p/m
    integrate particle position x_i^{t+1} = x_i^t + Δt u_i^{t+1}
```

Where:

- Line 2: W_{ij}: kernel with a kernel radius h
- Line 6: $\mathbf{u}_i^* = \mathbf{u}_i^t + \Delta t \mathbf{F}_i^{\text{ext}}/m$
- Line 10: compute p_i
- Line 12: $\mathbf{F}_i^p = -m^2 \sum_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W_{ij}$
- Line 14: $\mathbf{u}_i^{t+1} = \mathbf{u}_i^{**} + \Delta t \mathbf{F}_i^p/m$
- Line 15: $\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \Delta t \mathbf{u}_i^{t+1}$

3. Algorithm (2/2)

- More details of Eq.(2)

$$\mathbf{s}_i = \mu_i \left(\nabla \mathbf{u}_i + (\nabla \mathbf{u}_i)^T \right), \quad (2)$$

$$\mathbf{u}_i^{**} = \mathbf{u}_i^{*} + \frac{\Delta t}{\rho_i} \nabla \cdot \mathbf{s}_i^{**}, \quad (3)$$

$$\mathbf{s}_i^{**} = \mu_i \left(\nabla \mathbf{u}_i^{**} + (\nabla \mathbf{u}_i^{**})^T \right). \quad (4)$$

$\mathbf{u}_i^{*}$: first intermediate velocity
$\mathbf{u}_i^{**}$: second intermediate velocity
$\mathbf{S}_i^{**}$: intermediate viscous stress tensor
μ_i : dynamic viscosity

4.1 Implicit Integration for Full Form of Viscosity (1/3)

- Discretization of Eq.(3) and (4) using implicit integration in SPH framework

$$\mathbf{u}_i = \mathbf{u}_i^* + m\Delta t \sum_j \left(\frac{\mathbf{s}_i}{\rho_i^2} + \frac{\mathbf{s}_j}{\rho_j^2} \right) \nabla W_{ij}, \qquad (5)$$

$$\mathbf{s}_i = \mu_i \sum_j V_j \left((\mathbf{u}_j - \mathbf{u}_i) \nabla W_{ij}^T + \nabla W_{ij} (\mathbf{u}_j - \mathbf{u}_i)^T \right). \qquad (6)$$

$\mathbf{u}_i$: $\mathbf{u}_i^{**}$
$\mathbf{S}_i$: $\mathbf{S}_i^{**}$
V_j : stress tensor volume

4.1 Implicit Integration for Full Form of Viscosity (2/3)

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

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

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

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

$$q_{ij,yz} = \sum_j \left(a_{ij,z} v_{ij} + a_{ij,y} w_{ij} \right),$$

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

$\hat{\mu}_i$: μ_i / ρ_i^2

k : 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)

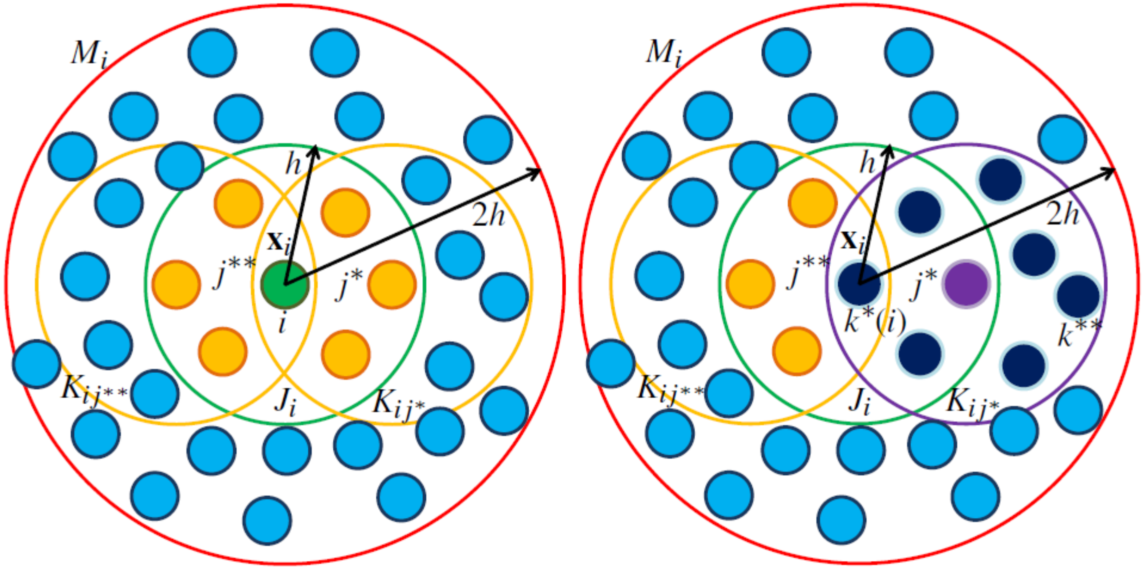
- This implicit formulation Eq. (7) is a linear system and can be rewritten in a matrix form as

$$\mathbf{CU} = \mathbf{U}^*$$

$\mathbf{C}$: coefficient matrix ($3N \times 3N$, N is number of particles)
$\mathbf{U}$: $[\dots, u_i, v_i, w_i, \dots]^T$ ($3N \times 1$, N 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 $\mathbf{Q}_{ij}$ in Eq. (8), we can rewrite Eq. (7) for x component of $\mathbf{u}_i, u_i$ as

$$
\begin{aligned}
u_i + \hat{m} \sum_j \Bigg(\hat{\mu}_i \Big(2\nabla W_{ij,x} \sum_j a_{ij,x} u_{ij} + \\
\nabla W_{ij,y} \sum_j (a_{ij,y} u_{ij} + a_{ij,x} v_{ij}) + \nabla W_{ij,z} \sum_j (a_{ij,z} u_{ij} + a_{ij,x} w_{ij}) \Big) \\
+ \hat{\mu}_j \Big(2\nabla W_{ij,x} \sum_k a_{jk,x} u_{jk} + \nabla W_{ij,y} \sum_k (a_{jk,y} u_{jk} + a_{jk,x} v_{jk}) \\
+ \nabla W_{ij,z} \sum_k (a_{jk,z} u_{jk} + a_{jk,x} w_{jk}) \Big) \Bigg) = u_i^* . \; (9)
\end{aligned}
$$

4.3 Solver and Coefficient Extraction (2/4)

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

  $c_{u_i u_i}$, $c_{v_i u_i}$, $c_{w_i u_i}$, $c_{u_j u_i}$, $c_{v_j u_i}$, $c_{w_j u_i}$, $c_{u_k u_i}$, $c_{v_k u_i}$, $c_{w_k u_i}$:

$$\begin{bmatrix} c_{u_i u_i} \\ c_{v_i u_i} \\ c_{w_i u_i} \end{bmatrix}^T \begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix} + \sum_j \begin{bmatrix} c_{u_j u_i} \\ c_{v_j u_i} \\ c_{w_j u_i} \end{bmatrix}^T \begin{bmatrix} u_j \\ v_j \\ w_j \end{bmatrix} + \sum_k \begin{bmatrix} c_{u_k u_i} \\ c_{v_k u_i} \\ c_{w_k u_i} \end{bmatrix}^T \begin{bmatrix} u_k \\ v_k \\ w_k \end{bmatrix} = u_i^*,$$

4.3 Solver and Coefficient Extraction (3/4)

$$
\begin{aligned}
c_{u_iu_i} &= 1+\hat{m}\hat{\mu}_i\left(2\omega_{ij,x}\alpha_{ij,x}+\omega_{ij,y}\alpha_{ij,y}+\omega_{ij,z}\alpha_{ij,z}\right),\\
c_{v_iu_i} &= \hat{m}\hat{\mu}_i\omega_{ij,y}\alpha_{ij,x},\\
c_{w_iu_i} &= \hat{m}\hat{\mu}_i\omega_{ij,z}\alpha_{ij,x},\\
c_{u_ju_i} &= \hat{m}\Big(-\hat{\mu}_i(2a_{ij,x}\omega_{ij,x}+a_{ij,y}\omega_{ij,y}+a_{ij,z}\omega_{ij,z})+\\
&\qquad \hat{\mu}_j(2\nabla W_{ij,x}\alpha_{jk,x}+\nabla W_{ij,y}\alpha_{jk,y}+\nabla W_{ij,z}\alpha_{jk,z})\Big),\\
c_{v_ju_i} &= \hat{m}\left(-\hat{\mu}_i a_{ij,x}\omega_{ij,y}+\hat{\mu}_j\nabla W_{ij,y}\alpha_{jk,x}\right),\\
c_{w_ju_i} &= \hat{m}\left(-\hat{\mu}_i a_{ij,x}\omega_{ij,z}+\hat{\mu}_j\nabla W_{ij,z}\alpha_{jk,x}\right),\\
c_{u_ku_i} &= -\hat{m}\sum_j\hat{\mu}_j(2\nabla W_{ij,x}a_{jk,x}+\nabla W_{ij,y}a_{jk,y}+\nabla W_{ij,z}a_{jk,z}),
\end{aligned} \tag{10}
$$

$$c_{v_ku_i} = -\hat{m}\sum_j\hat{\mu}_j\nabla W_{ij,y}a_{jk,x}, \tag{11}$$

$$c_{w_ku_i} = -\hat{m}\sum_j\hat{\mu}_j\nabla W_{ij,z}a_{jk,x}, \tag{12}$$

$$\alpha_{ij} : \left[\alpha_{ij,x},\alpha_{ij,y},\alpha_{ij,z}\right]^T = \sum_j \mathbf{a}_{ij}$$

$$\omega_{ij} : \left[\omega_{ij,x},\omega_{ij,y},\omega_{ij,z}\right]^T = \sum_j \nabla \mathrm{w}_{ij}$$

4.3 Solver and Coefficient Extraction (4/4)

Algorithm 1 Procedure of our method

// j: neighbor particle of i

// W_{ij}: kernel with a kernel radius h

for all particle i **do**

  find neighbor particles

for all particle i **do**

  apply external force $\mathbf{u}_i^* = \mathbf{u}_i^t + \Delta t \mathbf{F}_i^{ext}/m$

for all particle i **do**

  solve viscosity using Eqs. (3) and (4) // § 4

for all particle i **do**

  compute p_i using a particle-based fluid solver

for all particle i **do**

  compute $\mathbf{F}_i^p = -m^2 \sum_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W_{ij}$

for all particle i **do**

  integrate particle velocity $\mathbf{u}_i^{t+1} = \mathbf{u}_i^{**} + \Delta t \mathbf{F}_i^p/m$

  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

initialize a matrix

for all fluid particle i **do**

  compute $\hat{\mu}_i, \omega_{ij}$ and α_{ij}

compute $\hat{m}$

for all fluid particle i **do**

  initialize storage for u_k, v_k, and w_k

  add $c_{u_iu_i}, c_{v_iu_i}, c_{w_iu_i}, c_{u_iv_i}, c_{v_iv_i}, c_{w_iv_i}, c_{u_iw_i}, c_{v_iw_i}$, and $c_{w_iw_i}$ to the matrix

  for all fluid particle j **do**

    compute ∇W_{ij} and $\mathbf{a}_{ij}$

    add $c_{u_ju_i}, c_{v_ju_i}, c_{w_ju_i}, c_{u_jv_i}, c_{v_jv_i}, c_{w_jv_i}, c_{u_jw_i}, c_{v_jw_i}$, and $c_{w_jw_i}$ to the matrix

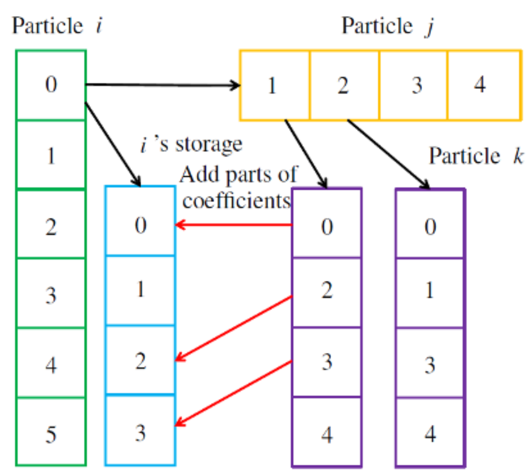
    for all fluid particle k **do**

      compute $\mathbf{a}_{jk}$

      add $c_{u_ku_i}, c_{v_ku_i}, c_{w_ku_i}, c_{u_kv_i}, c_{v_kv_i}, c_{w_kv_i}, c_{u_kw_i}, c_{v_kw_i}$, and $c_{w_kw_i}$ to the i's storage with k's id

  for all i's storage **do**

    add $c_{u_ku_i}, c_{v_ku_i}, c_{w_ku_i}, c_{u_kv_i}, c_{v_kv_i}, c_{w_kv_i}, c_{u_kw_i}, c_{v_kw_i}$, and $c_{w_kw_i}$ to the matrix using the storage



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 $\mathbf{u}_{solid}$ to fluid particles if viscosity of the fluid particles is higher than a criterion $\mu_{Dirichlet}$

Algorithm 2 Algorithm for solving viscosity

---

assemble the matrix // see Appendix A
solve the linear system with CG
for all fluid particle i **do**
  if $\mu^{\text{Dirichlet}} < \mu_i \wedge$ neighbor solid particle exists **then**
    enforce solid boundary condition $\mathbf{u}_i = \mathbf{u}^{\text{solid}}$

---

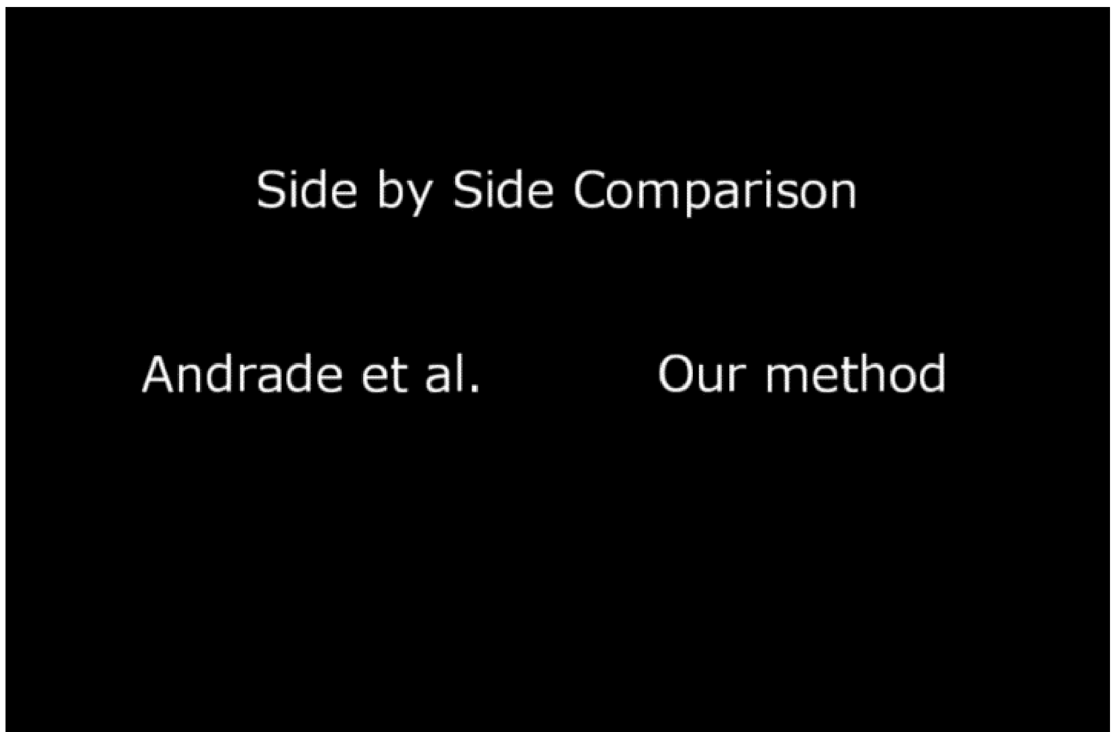
5. Result

- 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.

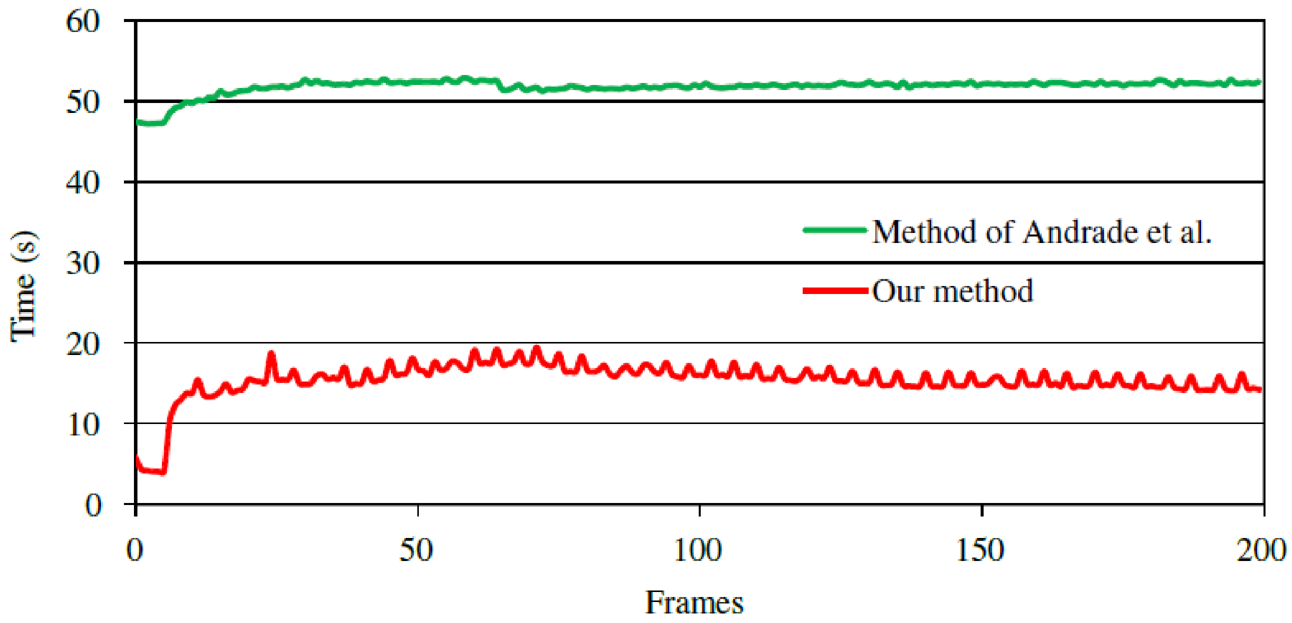
5.1 Numerical Stability

- 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]



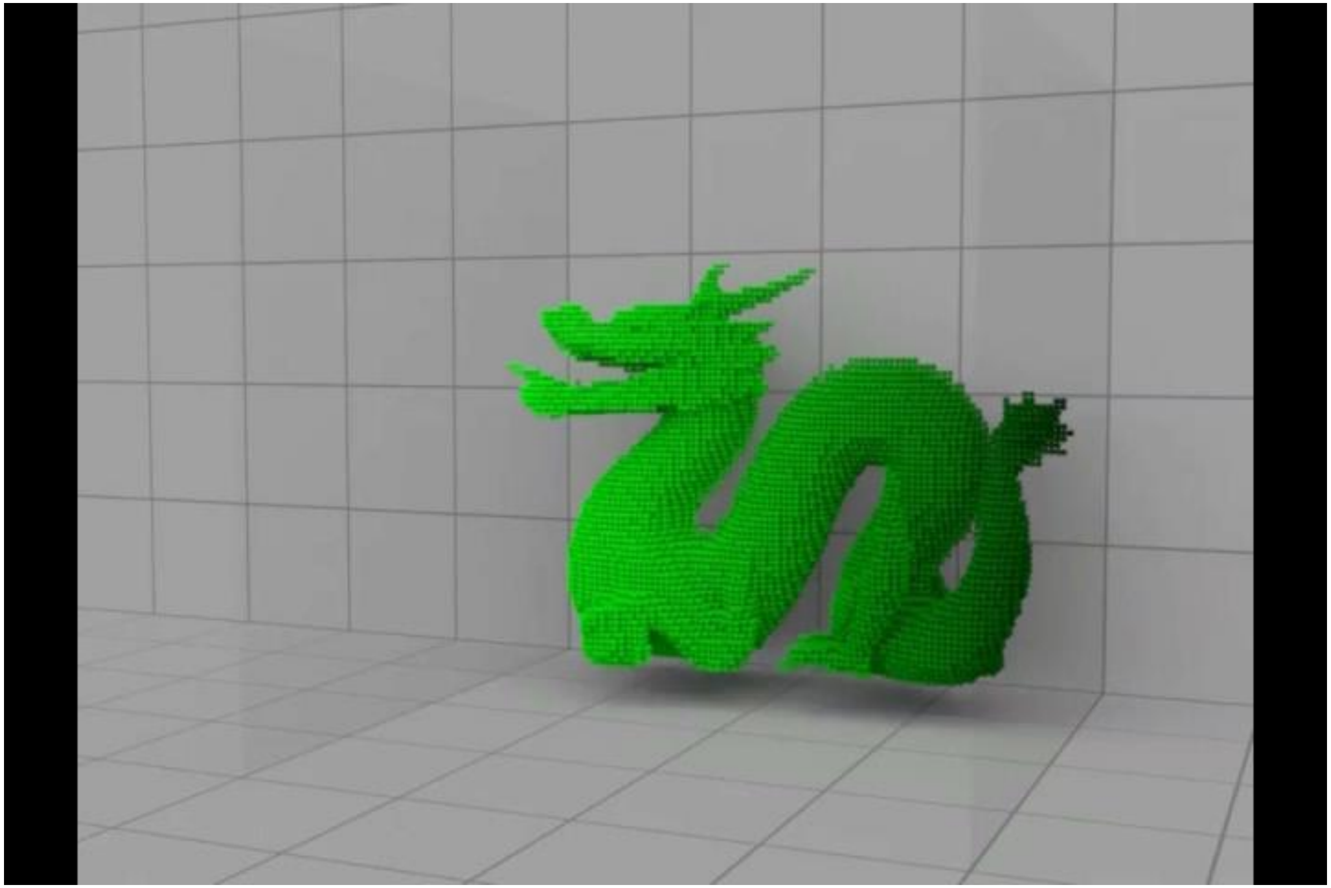
5.2 Performance

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



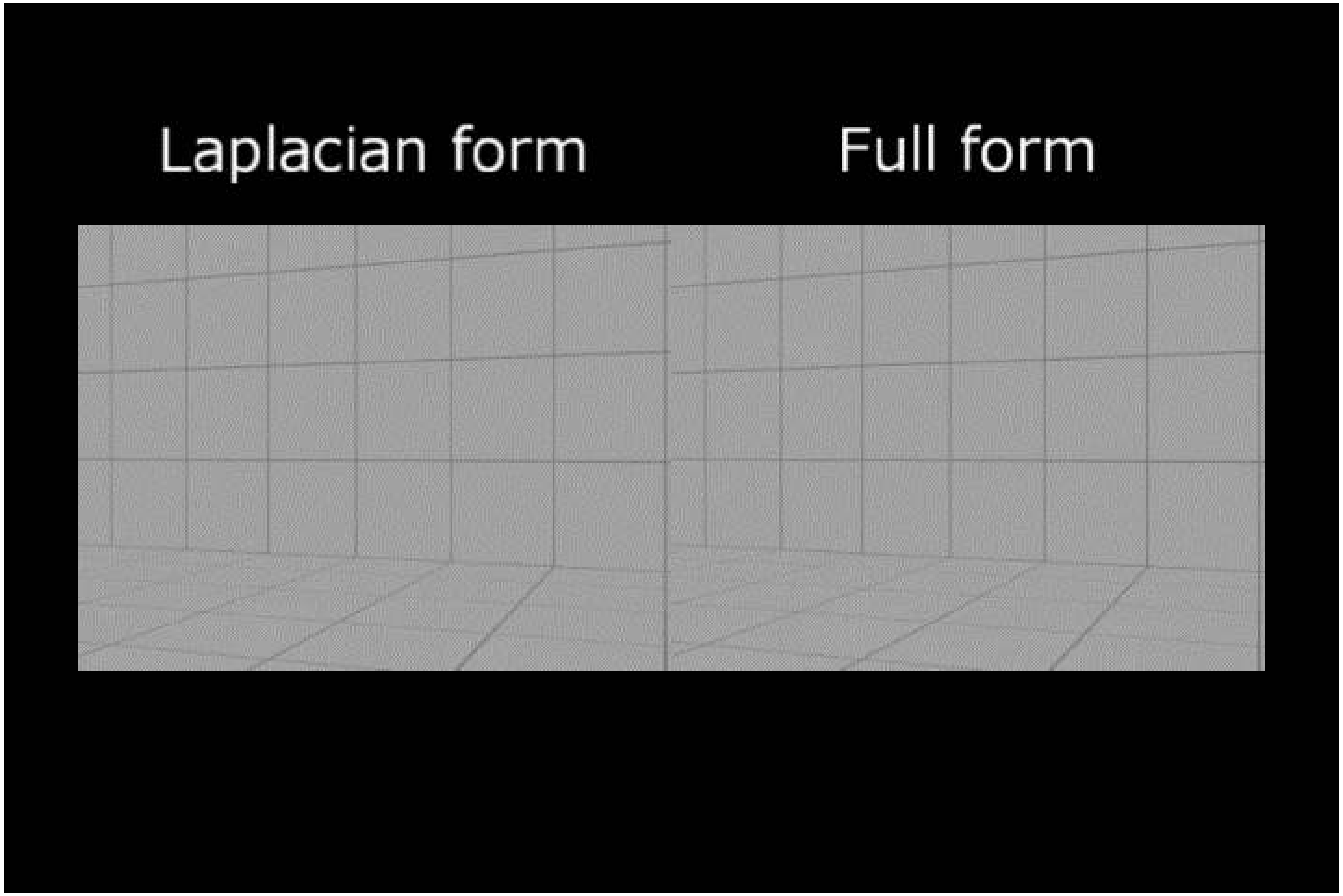
5.3 Variable Viscosity

- An example of a dragon consisting of particles with different viscosities from 0.0 (light green) to 800.0 kg/(ms) (dark green)



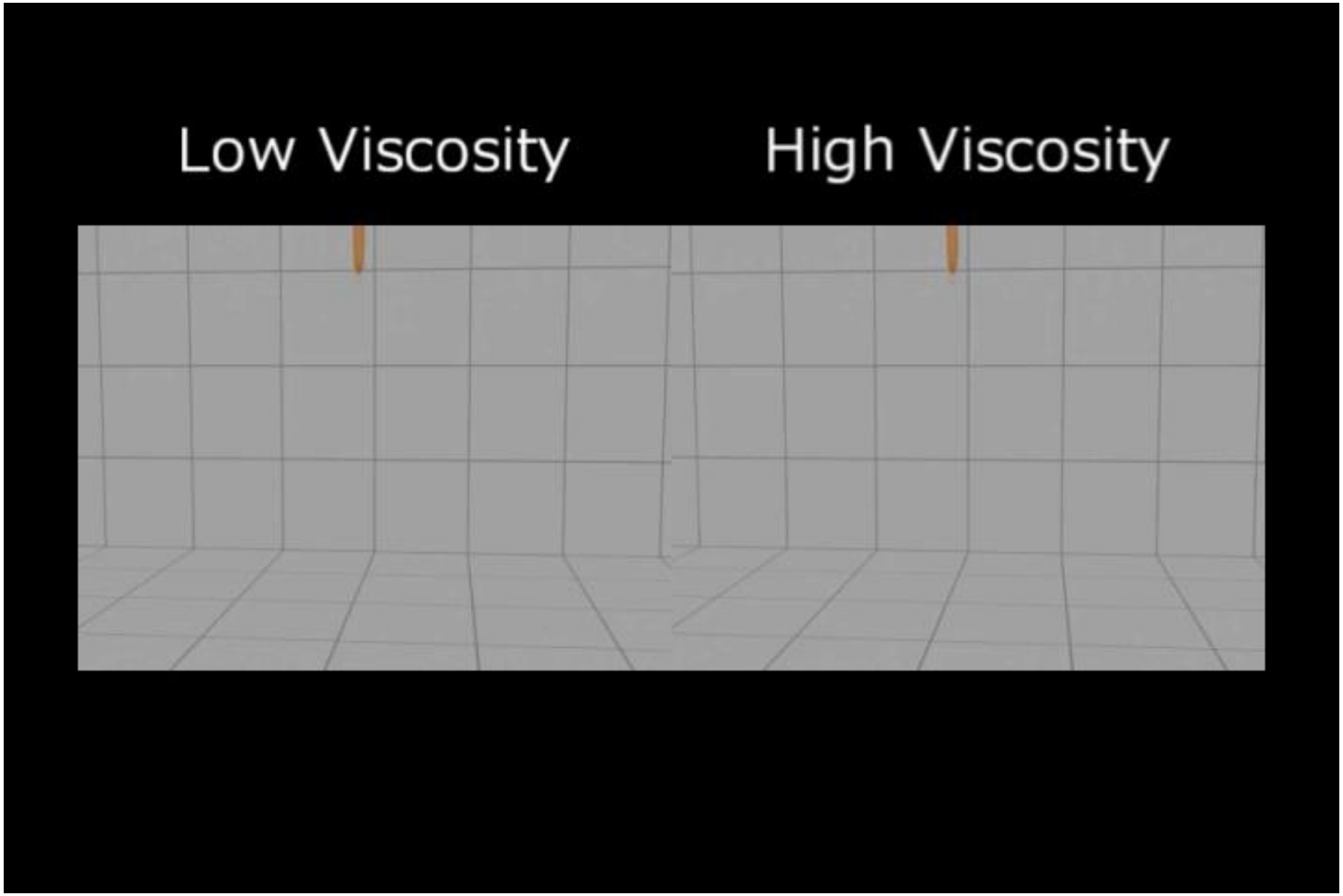
5.4 Buckling and Coiling (1/2)

- Buckling



5.4 Buckling and Coiling (2/2)

- Coiling



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