Particle-based fluids for viscous jet buckling

Computers & Graphics 52 (2015) 106–115

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Special Section

Particle-based fluids for viscous jet buckling

Luiz Fernando de Souza Andrade [a], Marcos Sandim [a], Fabiano Petronetto [b], Paulo Pagliosa [c], Afonso Paiva [a]

[a] Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, Brazil
[b] Universidade Federal do Espírito Santo, Vitória, Brazil
[c] Universidade Federal do Mato Grosso do Sul, Campo Grande, Brazil

ARTICLE INFO

Article history:
Received 29 November 2014
Received in revised form
24 June 2015
Accepted 17 July 2015
Available online 8 August 2015

Keywords:
SPH fluids
Jet buckling
Viscous liquids
CUDA
Computer animation

ABSTRACT

In this paper, we introduce a novel meshfree framework for animating free surface viscous liquids with jet buckling effects, such as coiling and folding. Our method is based on Smoothed Particle Hydrodynamics (SPH) fluids and allows more realistic and complex viscous behaviors than the previous SPH frameworks in computer animation literature. The viscous liquid is modeled by a non-Newtonian fluid flow and the variable viscosity under shear stress is achieved using a viscosity model known as Cross model. We demonstrate the efficiency and stability of our framework in a wide variety of animations, including scenarios with arbitrary geometries and high resolution of SPH particles. The interaction of the viscous liquid with complex solid obstacles is performed using boundary particles. Our framework is able to deal with different inlet velocity profiles and geometries of the injector, as well as moving inlet jet along trajectories given by cubic Hermite splines. Moreover, the simulation speed is significantly accelerated by using Computer Unified Device Architecture (CUDA) computing platform.

1. Introduction

A daily life example of viscous jet buckling is the coiling and folding of a thin thread of syrup or honey falling onto a spoon. The characteristic motion of a jet buckling is controlled by the balance among inertia, gravity and viscous forces that arise from the compressive stress caused by the impact of the fluid on a rigid surface. In the last years, Smoothed Particle Hydrodynamics (SPH) [1] has become a popular numerical meshfree tool for visually realistic animation of liquids [2]. However, simulating the complex free surface of a viscous jet buckling in an efficient and realistic way remains a big challenge for the previous SPH frameworks in computer animation. The difficulties are related to proposing a variable viscosity model which has a non-linear dependence of the fluid's shear rate, proposing an accurate and stable SPH approximation for viscous acceleration which involves second order derivatives of each component of the velocity field, and enforcing boundary conditions suited to SPH.

In this paper, we present a novel meshfree technique based on SPH fluids for simulating viscous jet buckling behaviors. Our technique allows a wide range of realistic viscous effects of the free surface of liquids, such as coiling and folding, as shown in Fig. 1. In order to capture the viscous behavior that is characteristic of jet buckling, the time interval between two consecutive frames needs to be very short, as discussed in Section 3.3, thus increasing the number of time-steps of the simulation total time. Since the computation in each time-step is highly intense, but can be performed in parallel and independently for each SPH particle, the problem can be suitably mapped to graphics processing units (GPUs). We use the Computer Unified Device Architecture (CUDA) by NVIDIA due to its efficiency, object-oriented programming capability, easy integration with the development environment we have used, and availability of a lot of libraries and demos which accompany the CUDA toolkit. The adequacy of using CUDA for standard SPH fluids can be demonstrated by other implementations reported in the literature [3,4]. In summary, the main contributions of this paper are:

Variable viscosity model: The viscous liquid is modeled as a non-Newtonian fluid flow and the variable viscosity is governed by a rheological model known as *Cross model* [5].

SPH viscous acceleration: We introduce a stable and robust SPH approximation of fluid's viscous acceleration using derivative operators of first order.

SPH boundary particles: Our framework allows us to impose boundary conditions on complex geometries to simulate rigid obstacles using boundary particles.

SPH jet buckling on CUDA: Using CUDA enables the animation of jet buckling scenes involving hundreds of thousands of SPH particles in affordable computational times, notably when compared to sequential processing, as showed by the experiments presented in Section 4, freeing the CPU for other tasks.

http://dx.doi.org/10.1016/j.cag.2015.07.021
0097-8493/

Fig. 1. The liquid rope coiling effect: a real image of honey coiling (left), our technique with free surface (middle) and with SPH fluid particles (right).

1.1. Related work

In order to better contextualize our approach and highlight its properties, we organize the existing frameworks for animating viscous jet buckling into two main groups, Eulerian mesh-based and Lagrangian meshfree-based methods.

Eulerian mesh-based: A seminal work in computer animation was introduced by Goktekin et al. [6]. They simulate solids and viscoelastic fluids with a small effect of buckling using an explicit grid-based method with viscosity transition between solid and non-Newtonian fluid controlled by a quasi-linear plasticity model. Batty and Bridson [7] developed an implicit and unconditionally stable method using marker-and-cell (MAC) grid. Although this method provides high-accuracy free surface boundary conditions, it is limited to viscous Newtonian fluids. Bergou et al. [8] proposed a discrete model for viscous threads using elastic rods to represent thin Newtonian liquid jets. Despite this method's realistic results, spurious results may occur when the jet becomes thick. Recently, this model was extend to discrete viscous thin sheets [9]. Batty and Houston [10] presented an adaptive tetrahedral mesh solver to animate Newtonian liquids with high viscosity. However, the level set surface generated by this method does not preserve temporal coherence due to the slow motion of the liquid. In computational physics literature, there are several papers using variations of generalized simplified MAC (GENSMAC) method to simulate viscous jet buckling in arbitrary 2D/3D domains with explicit [11–13] and implicit [14] free surface boundary conditions.

Lagrangian meshfree-based: SPH fluids have been applied with success in simulations of highly viscous liquids with variable viscosity [15–19]. However, none of these methods in computer animation have captured viscous buckling behavior. In computational physics, Rafiee et al. [20] used an incompressible version of SPH to simulate 2D jet buckling of non-Newtonian fluids, while Xu et al. [21,22] extended the traditional weakly compressible SPH method to deal with 3D simulations. This paper is inspired in [21] and it improves that work in several ways: our stable SPH approximation of momentum equation does not require additional terms, artificial stress and artificial viscosity, to prevent particle clustering and unphysical behavior of free surface.

A previous version of this paper [23] focused on buckling effects in planar boundaries only. Here, we focus on buckling effects on arbitrary surfaces and injectors with complex geometry. In addition to the material on jet buckling presented in Section 4, we include a wide variety of examples using injectors with complex geometry and with different velocity profiles at the inlet. Moreover, we demonstrate the ability of our technique in applications with complex rigid surfaces using boundary particles.

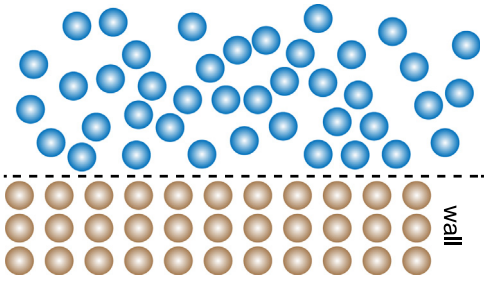

Fig. 2. SPH particles: fluid (blue) and boundary (brown) particles. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

2. Governing equations

The governing equations for simulating fluid flow are derived from mass and momentum conservation laws. Lagrangian framework describes these laws from the viewpoint of an infinitesimally small fluid element, i.e., a particle. In this framework the mass conservation is naturally satisfied, since the particle mass is constant, then the total mass of the system is preserved. For weakly compressible fluids, the momentum equation can be written as follows:

$$\frac{d\mathbf{v}}{dt} = -\frac{1}{\rho}\nabla p + \frac{1}{\rho}\nabla \cdot \tau + \mathbf{g} \tag{1}$$

where t denotes the time, $\mathbf{v}$ the velocity field, ρ the density, p the pressure, $\mathbf{g}$ the gravity acceleration vector and τ the shear stress tensor.

Lagrangian formulation of Eq. (1) represents the acceleration of a particle moving with the fluid flow. The term $-\frac{1}{\rho}\nabla p$ is related to particle acceleration due to pressure changes in the fluid. While, the term $\frac{1}{\rho}\nabla \cdot \tau$ describes the viscous acceleration due to friction forces caused by particles with different velocities. This last term plays a key role in viscous jet buckling animation.

Cross model

In order to animate a wide variety of buckling effects, Newtonian and non-Newtonian fluid flows are used in this paper. In particular, non-Newtonian fluids have non-linear dependence of the shear stress τ with respect to the rate-of-deformation tensor $\mathbf{D} = \nabla\mathbf{v} + (\nabla\mathbf{v})^{\top}$ as follows:

$$\tau = \rho\nu(D)\mathbf{D} \quad \text{with} \quad D = \sqrt{\tfrac{1}{2} \cdot \mathrm{tr}(\mathbf{D})^2}. \tag{2}$$

The Cross model [5] is one of the simplest and most used models for shear-thinning behavior, i.e., the fluid's viscosity decreases with increasing of the local shear rate D, thus the kinematic viscosity ν is defined as a function of D

$$\nu(D) = \nu_\infty + \frac{\nu_0 - \nu_\infty}{1+(KD)^n}, \tag{3}$$

where K and n are positive parameters, and ν_0, ν_∞ are the limiting values of the viscosity at low and high shear rates, respectively. The units of viscosity are m^2/s.

Assuming $K=0$ in Eq. (3), the non-Newtonian fluid model is simplified to a Newtonian fluid with constant kinematic viscosity ν_0.

3. Our technique

There are several SPH frameworks to animate fluid flow. In our animation framework, we use a SPH version for weakly compressible fluids [24], extended by our method. A wide description of SPH fluids for graphics can be found in [2].

3.1. SPH fluids

The main idea of SPH in fluid flow simulation is to discretize the fluid by a set of particles where each particle i has attributes such as position $\mathbf{x}_i$, velocity $\mathbf{v}_i$, pressure p_i, mass m_i (constant across all particles) and density ρ_i. A scalar or vector attribute $f_i = f(\mathbf{x}_i)$ is updated using a SPH approximation

$$f_i = \sum_{j \in N_i} f_j W_h(x_{ij}) \frac{m_j}{\rho_j}, \tag{4}$$

where j indexes the particles lying in the neighborhood N_i of the particle i and $x_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$. The kernel function W_h is a radially symmetric, positive, smooth, compactly supported function and the value h defines the region of influence of W_h. Fluid attributes and the differential operators of the momentum equation (1) can be approximated in a similar manner as follows.

Density computation: We compute the density using the traditional direct summation derived from Eq. (4) with *poly6* kernel presented in [24]

$$\rho_i = \sum_{j \in N_i} m_j W_h(x_{ij}). \tag{5}$$

This form preserves mass exactly without involving kernel derivatives.

Equation of state: In standard SPH frameworks, an incompressible fluid is approximated by a weakly compressible fluid. In other words, the pressure is computed directly from density through an equation of state. For simplicity, we choose an equation of state proposed by Morris et al. [25]

$$p_i = c^2(\rho_i - \rho_0), \tag{6}$$

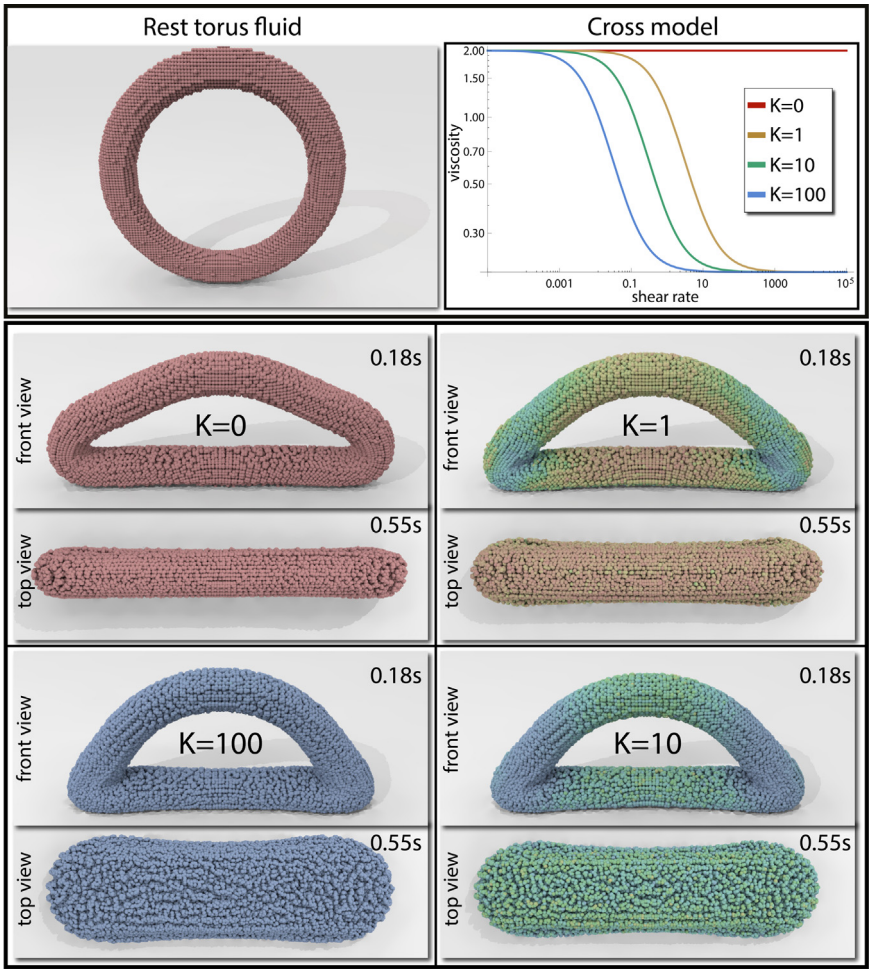

Fig. 3. Numerical simulation of the torus fluid spreading. The initial configuration of the torus fluid is illustrated in top-left. In top-right, we plot (loglog scales) the Cross model with parameters: $[\nu_\infty, \nu_0] = [0.2, 2]$, $n=1$ and $K=0$, 1, 10 and 100. Fluid flow simulations in the same times are showed in clockwise order from top-left for the Newtonian fluid ($K=0$) and non-Newtonian fluids with increases K parameter. The colors code the viscosity $\nu_i \in [\nu_\infty, \nu_0]$ in the SPH particles. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

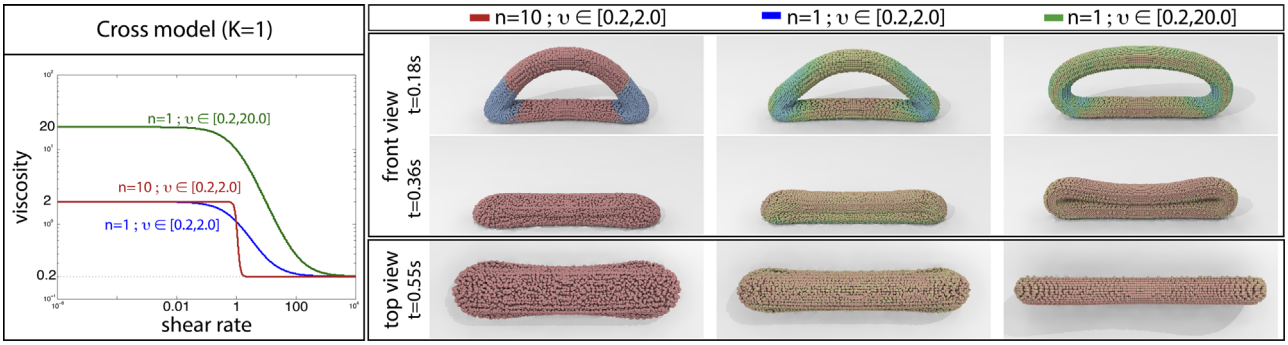

Fig. 4. Numerical simulation of the torus fluid spreading (Fig. 3 shows initial configuration of the torus fluid). Fixing $K=1$, the graphs (log–log scales) illustrate the Cross model with parameters: $n=10$ and $[\nu_\infty,\nu_0]=[0.2,2]$ (red), $n=1$ and $[\nu_\infty,\nu_0]=[0.2,20]$ (green), and $n=10$ and $[\nu_\infty,\nu_0]=[0.2,2]$ (blue). Varying only n (first two columns), a greater mobility in fluid is seen with $n=10$ (left column) because the transition occurs sharply in cross model and therefore the lower viscosity is achieved rapidly in simulation. Varying only ν_∞ and ν_0 (last two columns), when the right limit ν_0 is greater (right column) the behavior more viscous of the fluid causes the fluid remains for a longer time near its initial configuration. The colors code the viscosity $\nu_i \in [\nu_\infty,\nu_0]$ in the SPH particles. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

where c is the speed of sound in the fluid and ρ_0 is a reference density. The values $c=\sqrt{1.5}$ m/sS and $\rho_0=10^3$ kg/m^3 are tuned out to be suitable for all experiments.

Pressure acceleration: We use Müller's SPH pressure gradient with *spiky* kernel [24]. The pressure acceleration for each particle is given by

$$-\frac{1}{\rho_i}\nabla p_i = -\sum_{j\in N_i} m_j \frac{p_i+p_j}{2\rho_j}\nabla_i W_h(x_{ij}). \tag{7}$$

This choice avoids numerical instabilities associated with particle clustering, because the SPH gradient does not vanish when x_{ij} becomes closer to zero.

Viscous acceleration: In order to compute the shear stress τ_i at particle i in Eq. (2), we adopt the same SPH approximation as Paiva et al. [19] for the deformation tensor $\mathbf{D}_i=\nabla\mathbf{v}_i+\nabla\mathbf{v}_i^\top$ with

$$\nabla\mathbf{v}_i = \sum_{j\in N_i}\frac{m_j}{\rho_j}(\mathbf{v}_j-\mathbf{v}_i)\otimes\nabla_i W_h(x_{ij}). \tag{8}$$

After updating shear stress at all particles, the viscous acceleration is approximated by

$$\frac{1}{\rho_i}\nabla\cdot\tau_i = \sum_{j\in N_i}\left(\frac{\tau_i}{\rho_i^2}+\frac{\tau_j}{\rho_j^2}\right)\cdot\nabla_i W_h(x_{ij}). \tag{9}$$

3.2. Boundary conditions

We represent solid wall boundaries by using fixed virtual particles [26]. These boundary particles perform similar to the fluid particles and contribute to the SPH approximation of the fluid attributes. The density of the fluid particles is extrapolated for boundary particles using Eq. (5). Then, the pressure is computed in the boundary particles directly from (6). The increasing pressure at boundary particles prevents the fluid particles from penetrating the solid walls. In order to enforce the no-slip condition, the velocities of the boundary particles are set to zero throughout the entire simulation. Those particles are placed outside the computational domain with particle spacing equal to the initial particle spacing of the fluid particles. The wall boundaries are modeled by a few layers of the boundary particles as illustrated in Fig. 2. The particle deficiency at wall boundaries is alleviated by taking the total width of boundary particle layers at least equal to the radius of influence of the SPH kernel.

Another boundary condition is the stress-free condition for momentum Eq. (1), which states that the total normal stress must be zero at free surface. Mathematically, it can be expressed as $(-p\mathbf{I}+\tau)\cdot\mathbf{n}=\mathbf{0}$, where $\mathbf{I}$ is the identity matrix and $\mathbf{n}$ is the normal to the free surface. This condition is trivially satisfied by the SPH gradients because the boundary integrals are ignored in the SPH derivatives approximation [27].

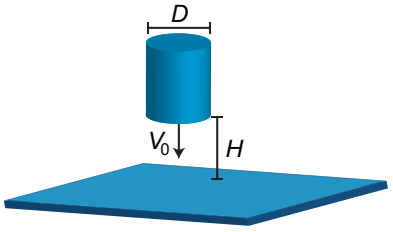

Fig. 5. Jet buckling setup.

3.3. Implementation

We implement our technique in C++ and CUDA C++ using the Microsoft Visual Studio 2013 for Windows and CUDA 6.5. The input data of the program are given in an initialization file or interactively by the user and include: the material properties of the fluid model, configuration of the jet (shape, position, trajectory along the simulation, injection rate, maximum number of particles), and simulation parameters (time-step δt, maximum number of steps, total time), among others. For each time-step δt, the program updates the particle attributes following the sequence in Algorithm 1.

Algorithm 1. Single time step of the SPH framework.

1: Inject new fluid particles into the scene
2: **for** each particle i **do**
3: Find neighbors N_i
4: **end for**
5: **for** each particle i **do**
6: Update ρ_i using Eq. (5)
7: Update p_i using Eq. (6)
8: **end for**
9: **for** each particle i **do**
10: Update $\nabla\mathbf{v}_i$ using Eq. (8)
11: Update τ_i using Eq. (2)
12: **end for**
13: **for** each particle i **do**
14: Compute pressure acceleration using Eq. (7)
15: Compute viscous acceleration using Eq. (9)
16: **end for**
17: **for** each particle i **do**
18: Update $\mathbf{v}_i$ and $\mathbf{x}_i$ with leap-frog scheme
19: **end for**

Fig. 6. The formation of coils in a falling stream of viscous liquid.

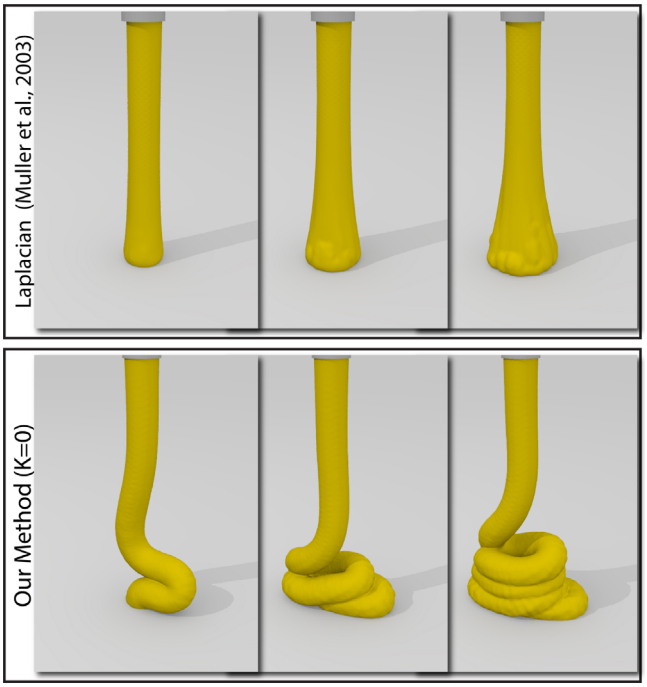

Fig. 7. Jet buckling of a Newtonian fluid. In top row, the result given by an implementation of [24] which uses the Laplacian velocity operator in the viscous acceleration. In bottom row, we show the result achieved by our technique with the same parameters (Cross model with $K=0$).

The first task is to add new fluid particles into the scene, which are generated by the jet inlet in CPU and then sent to a preallocated area (enough to store the maximum number of particles) in the global memory of the GPU. New particles are created iff:

1. The set of particles previously generated by the jet inlet has moved a given distance from the jet position.
2. The maximum number of particles into the scene has not yet been reached. This limit depends on the available global memory of the GPU.

The data structure for the particle system representing the fluid is implemented using the Thrust library [28] and organized as a structure of arrays (of position, velocity, density, pressure, etc.) in order to allow coalesced access to the GPU global memory.

Following tasks are performed entirely in parallel on GPU. For the neighbor search, we use the algorithm described in the *Particles* demo in the CUDA's toolkit [29], which is based on a subdivision of the simulation space into a regular grid of linearly indexed cells. The algorithm relies on several CUDA kernels. The first one calculates a hash value for each particle based on the index of the cell containing the particle. We then use the Thrust sorting function to reorder the particles according to their hash values. Finally, a kernel which uses one thread per particle builds its neighbor list by comparing the cell index of the current particle with the cell index of the previous particle in the sorted list (see [29] for details).

The remaining tasks in Algorithm 1 are implemented by four CUDA kernels which also use one thread per particle. Each kernel computes, respectively, density and pressure, deformation tensor and shear stress, pressure and viscous accelerations, and velocity and position of each particle. We use the leap-frog scheme to integrate the system of differential equations provided by the SPH approximation of (1) along the particle trajectories. The numerical stability in this explicit time integration scheme is ensured under time-step constraints given by Courant–Friedrichs–Lewy (CFL) and viscous force conditions [19]

$$\delta t \leq 0.1 \min\left(\frac{h}{c}, \frac{h^2}{8\nu_0}\right). \tag{10}$$

Respecting Eq. (10), we choose $\delta t = 10^{-6}$ s in our experiments.

The free surface rendering is performed by blobs (metaballs) from POV-Ray. The blob radius is equal to the radius of influence of the SPH kernel used in the simulation.

4. Results and comparisons

In this section, we present the results provided by our technique, including animations of viscous liquids with coiling and folding effects. Initially, we discuss the variable viscosity effects of non-Newtonian fluids in simulations of gravity spreading. Then, we apply our technique to a variety of situations involving jet buckling. All examples have been generated in a computer equipped with a CPU Intel i5 3570 3.4 Hz with 8GB of RAM, and a NVIDIA GeForce 650 with 384 CUDA cores and 1 GB of RAM.

Fig. 8. Jet buckling with different viscosity intervals. Our method provides different coiling effects varying the parameters of Cross model.

4.1. Torus fluid

In this first result, to demonstrate that our method can model shear-thinning behavior using the Cross model, we performed the numerical simulation of a torus fluid spreading on a plane surface due to gravity (9.8 m/s^2).

Fig. 3 shows the results of our method using the Cross model with $n=1$, $\nu_0=2.0$, $\nu_\infty=0.2$ and four values of K, namely $K=0$ for a Newtonian fluid (which corresponds to assigning a constant viscosity at ν_0) and $K=1.0$, 10.0 and 100.0 for a non-Newtonian fluid. The shear-thinning is increased at higher values of K allowing the fluid to spread more over the surface, mainly in direction of largest shear rate (see top view).

Another feature of the Cross model is the parameter n which determines the rate of decay of the viscosity as a function of the shear rate. The viscosity transition between the viscosity limits ν_∞ and ν_0 becomes faster at high values of n. Moreover, different rates between these limits also produce different behaviors in the fluid flow.

Fig. 4 illustrates the effect of the parameters n, ν_∞ and ν_0 in our method. Fixing $K=1$, the left graph shows the viscosity variation under the influence of these parameters, and on the right, fluid flow simulations combining these parameters. All simulations clearly depict the variable viscosity effect, typical characteristic property of shear-thinning fluids. The first two columns show the different shear-thinning behavior given by the variation of the index parameter, we use $n=10$ (left) and $n=1$ (middle). Testing $n=1$ and $[\nu_\infty,\nu_0]=[0.2,2]$, the fluid has a more homogeneous behavior due to the smoothness of the viscosity function provided by the Cross model (blue line in left graph). When the transition occurs sharply, $n=10$ (red line in left graph), we observe that the lower viscosity value ν_∞ is reached quickly in the simulation, and therefore a greater mobility in these particles is seen. The last two columns show the different shear-thinning behavior given by the variation of ν_∞ and ν_0. Increasing the upper limit to $\nu_0=20$ (green line in left graph), the fluid tends to be more viscous, remaining close to its initial shape even after many time steps.

Figs. 3 and 4 help us to understand the shear-tinning behavior provided by Cross model with different sets of parameters. The results show that the Cross model determines the spatial variation of the viscous force that reflects the behavior of the fluid flow. This model allows a wide range of viscous effects of the free surface of Newtonian and non-Newtonian fluids in a small set of user parameters.

4.2. Jet buckling

The jet buckling problem has been studied in several experimental and analytical investigations [30,31]. This phenomenon is characterized by the formation of a physical instability when a viscous fluid jet is injected with velocity V_0 hits a rigid plate.

The configuration of a jet buckling problem is given by the height H between the injector and the rigid plate and by the diameter D of the bounding circle of the injector's cross-section in meters (Fig. 5). In our simulations of this problem, we use values for H, D and physical parameters consistent with the conditions described by Tomé et al. [13].

When a stream of viscous fluid falls onto a surface, the deceleration of the stream near impact causes the buckling effect and the fluid starts to coil on itself. Fig. 6 illustrates the coiling formation in our results. Note that, as the coiling develops, the fluid tends to have a more viscous behavior at the surface, which prevents the fluid from further spreading. Hence, when the falling stream contacts the coils already formed, the viscosity decreases and new coils are created.

Fig. 7 shows a comparison of our technique with the popular SPH framework proposed by Müller et al. [24] in which the viscous term of Eq. (1) is simplified using the Laplacian velocity operator $\nu_0\Delta\mathbf{v}$. Choosing the same parameters in both simulations, we can verify that using the Laplacian velocity prevents the buckling development, leading to a spurious result.

In order to illustrate the relevance of the viscosity variation in our method, Fig. 8 shows two jet buckling with distinct behaviors by applying Cross model in different viscosity intervals.

Despite the fact that the fluids are fairly viscous, we can note, on the bottom of the fluid, the shear-thinning action, where the fluid becomes less viscous, spreading and mixing.

Fig. 9 illustrates the flexibility of our framework dealing with different inlet velocity profiles (constant or parabolic) and geometries of the injector. The particles are created inside of a polygonal

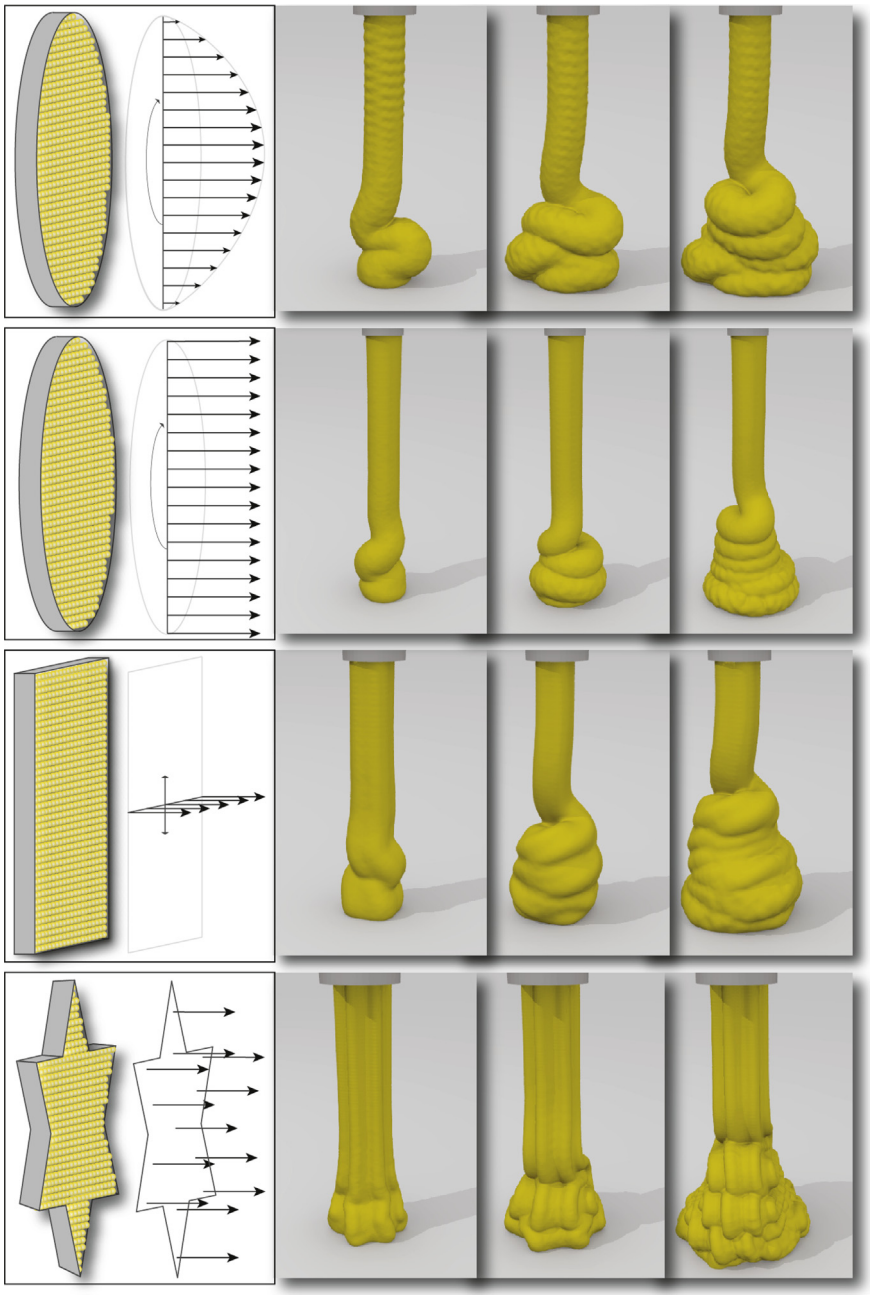

Fig. 9. Jet buckling simulations with different inlet velocity profiles and geometries of the injector (left column).

closed curve which represents a cross section of the injector. The particles can be generated in many ways, for rectangular injectors the particles are sampled using a simple regular grid, while in circular injectors a uniform particle sampling can be achieved using their polar coordinates. For injectors with arbitrary geometry, the ray casting algorithm is performed to determine which particles lies inside the injector (polygon). The viscous jet with circular and square injector and constant inlet velocity profile (second and third rows) have the same behavior where the coiling occurs in the axial direction. The circular injector with parabolic velocity (first row) also defines a coiling pattern with a different instability in axial component due to the non-constant velocity. The star shape injector (last row) generates a folding pattern in a preferential direction.

The shear-thinning behavior provided by inlet flow using rectangular jets are illustrated in Fig. 10. The flow behavior of the Newtonian fluid (using $K=0$ in Cross model) is more viscous than non-Newtonian fluid (using $K=1$) due to the viscosity variation given by our technique. It is noted that the number of layers is reasonably pronounced in the Newtonian case and less pronounced in the non-Newtonian case.

4.3. Moving inlet jet

Fig. 11 illustrates some 'stitching' patterns obtained with our method when a viscous liquid is injected by a moving jet. The buckling of the fluid and the motion of jet combine to give a wide range of regular and periodic patterns, like a "sewing machine".

Since the maximum number of particles in the scene is fixed (as mentioned before, limited by the amount of the GPU's global memory), we should remove from the scene a number of particles, which can be considered as ineffective, in order to keep the

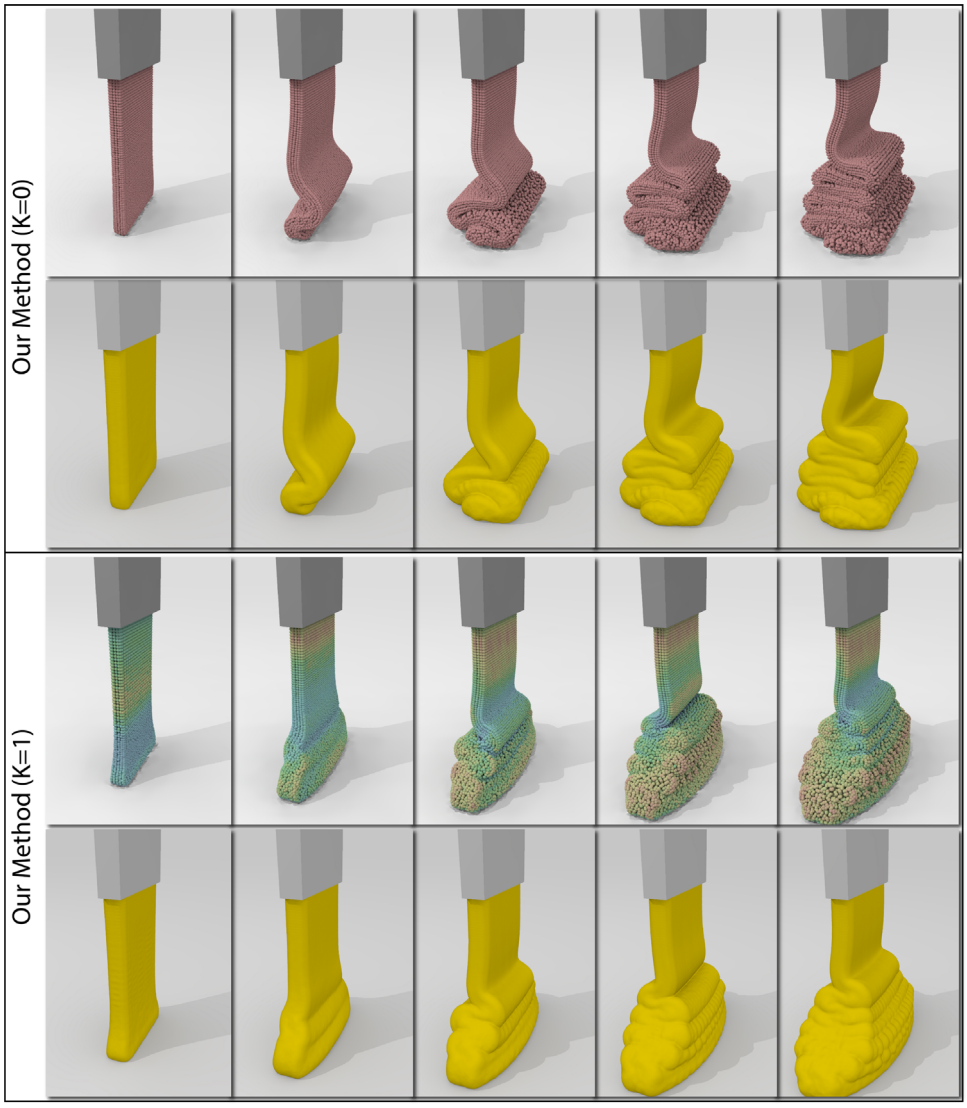

Fig. 10. Effect of the variable viscosity in our technique. At top, the simulation of a Newtonian fluid ($K=0$) defines well-defined fluid layers, while a non-Newtonian fluid ($K=1$) the fluid layers are mixed due to the shear-thinning.

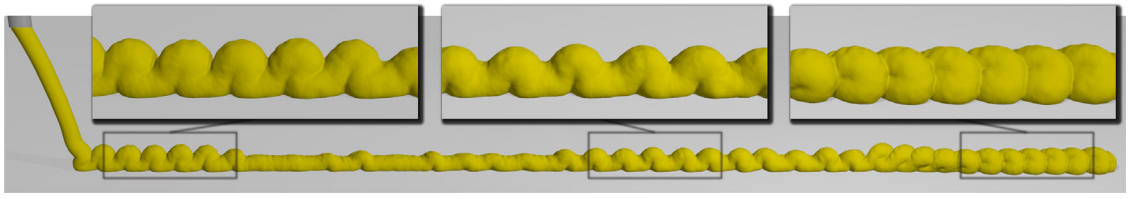

Fig. 11. Stitching patterns of a viscous thread poured by a moving jet.

moving jet able to generate new ones. We have adopted a view-dependent approach to remove particles: for each particle, we use the matrix $\mathbf{M}_{\text{vp}}$, the combination of the view and projection matrices derived from the virtual camera, in that order, to compute the OpenGL normalized device coordinates (NDCs) [32] of the particle position and then check its visibility. In OpenGL, the NDC space is an axis-aligned cube whose vertices have all coordinates in the range $(-1, 1)$. Given the homogeneous global position $\mathbf{p} = [x, y, z, 1]^\top$ of the particle, its NDCs are $(x_r/w_r, y_r/w_r, z_r/w_r)$, where $[x_r, y_r, z_r, w_r]^\top = \mathbf{M}_{\text{vp}} \cdot \mathbf{p}$ are its clip coordinates. The particle is classified as ineffective—and so marked to be removed from the scene—when it is not visible, i.e., lies outside of the NDC cube along the trajectory followed by the injector. The trajectory of the injector is determined by a smooth curve using a cubic Hermite spline and then displacing the injector's centroid along the curve.

4.4. On complex surfaces

Boundary particles can be used to represent not only the limits of the domain (ground or fluid container walls) but also the surface of solid objects which exert repulsive forces on fluid particles. Fig. 12 shows viscous liquid jet in a complex object modeled with boundary particles. On the left figure the injector is static while on the right figure the inlet jet is moving over a trajectory given by a smooth curve. In this simulation we can see

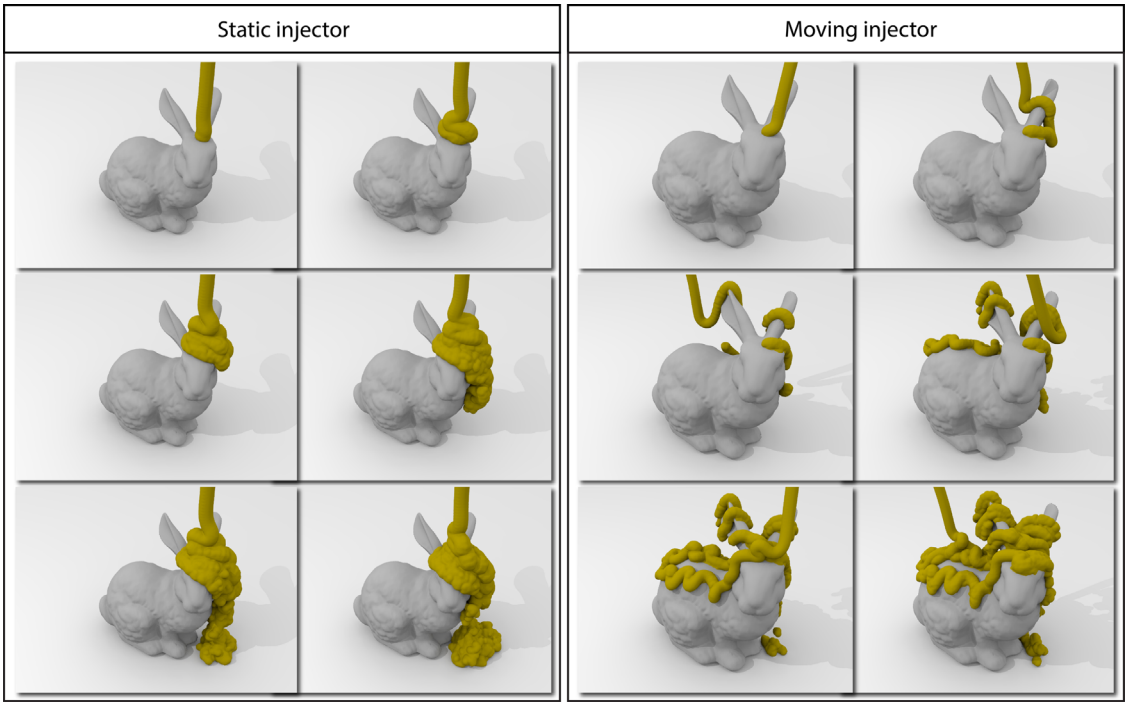

Fig. 12. Animation of a viscous liquid pouring on the Stanford Bunny modeled with 51k boundary particles. The jet buckling effect becomes more prominent when the injector is placed in a fixed position (top row) rather than in movement (bottom row).

Table 1
Performance statistics.

# **Part.**	**CPU** (s)	**GPU** (s)	**Speedup**	**Efficiency**
1 K	7.20	1.03	7.0	1.8%
4 K	35.66	2.40	14.9	3.9%
16 K	156.42	6.81	23.0	6.0%
64 K	682.00	25.80	26.4	6.9%
128 K	1374.86	51.35	26.8	7.0%
256 K	2887.20	104.84	27.6	7.2%
1 M	–	444.93	–	–

that the viscous fluid breaks apart in many pieces due to complex surface and moving injector. Moreover, the new layer of the fluid injected into the scene interacts with the fluid already deposited on the surface. The result shows an excellent agreement in terms of viscous fluid properties and ease of treating complex simulations with SPH method.

In our implementation a surface is entered as a triangle mesh from which we should generate, in a preprocessing step, layers of fixed boundary particles. There are several methods to sample particles on a mesh. The platform of open source SPH codes, e.g., *SPHysics* [33], provides a preprocessor [34] that allows users to use *Blender* (http://www.blender.org) to create the meshes representing the surfaces of the entire scene. Next, each mesh face is subdivided until the newly created vertices on the face are distributed such that the length of any mesh edge does not exceed a given threshold d_0 (usually the initial fluid particle spacing). The scene domain is then discretized in a uniform cubical grid with cells of size d_0, and the positions of the vertices resulting from faces subdivision are extrapolated onto—and therefore aligned to—the vertices of the grid. As a consequence, the higher the curvature of a significant region of a surface, the finer the grid resolution necessary for the whole scene. An alternative method, proposed in [35], consists in constructing a signed distance field in a narrow band near the triangle mesh. The distance field is computed on a voxel grid of resolution equivalent to d_0. On each triangle, particles are randomly positioned in a number that depends on the triangle's area, d_0, and a user defined density. Only initialized, the particles can float along the vertices of the voxel grid according to a version of the mechanism for sampling implicit surfaces with particles presented in [36].

Since we need to generate more than one layer of boundary particles (see Fig. 2), we have adopted a different approach which does not require a 3D grid. Firstly, we use *MeshLab* (http://www.meshlab.sourceforge.net) to refine the input triangle mesh so that the length of the any resulting edge approximates d_0, the initial fluid particle spacing. Next, we use the open source mesh generator *Gmsh* [37] (http://www.geuz.org/gmsh) to create the additional particle layers. The method implemented in *Gmsh* is based on the advancing front scheme proposed in NETGEN [38]. Taking the boundary triangles of the refined surface mesh as the initial front, the method relies on constraint rules to generate new internal vertices (distant $\sim d_0$ from the front) which are combined with the boundary triangles to form tetrahedral elements. The boundary particles are positioned at the newly created vertices, and the external faces of the newly created tetrahedrons, if any, become the next front. The process is repeated until we get a number of layers whose thickness is at least the radius of influence of the SPH kernel.

4.5. Comparison between CPU and GPU

Table 1 compares the performance of our implementation running a dam break sequentially on CPU and in parallel on GPU. The parameters of the Cross model are $K=1$, $n=1$, $\nu_0=0.05$, and $\nu_\infty=0.005$. The first column is the number of particles; the columns labeled CPU and GPU show the simulation times, in seconds, for 1000 steps on CPU and GPU, respectively; the column Speedup is the relation CPU/GPU; and the column Efficiency is the speedup divided by the number of CUDA cores used in the experiment (384). Though the low efficiency, the speedup of about 27 for the number of particles varying between 64 K and 256 K justifies the use of the GPU: keeping that speedup, the CPU

time for the simulation of 1 M particles (not measured in our test) would be about ~3.3 h, against ~7.4 min on GPU.

5. Concluding remarks

In this paper, we introduced a novel SPH-based technique for animating free surface viscous liquids with jet buckling. Unlike the previous SPH frameworks, our technique allows visually realistic viscous behaviors, such as coiling and folding. The technique relies on the SPH approximation of a non-Newtonian fluid, where the variable viscosity is ruled by the Cross model. We extend the previous version of this paper [23] to animate viscous jet bucking on complex surfaces using boundary particles. The effectiveness of the technique is demonstrated on a wide range of examples which match with the physics intuition, leading to an efficient and attractive scheme for animation. Moreover, simulation time is considerably improved by using CUDA computing platform.

One limitation of our technique is that its results depend of the time-step size, the condition (10) may not permit large time-steps in the simulation, thus resulting a time consuming animation.

A natural direction for future work is to extend our technique to deal with truly incompressible fluid flows.

Acknowledgments

A previous version of this paper was presented at Sibgrapi 2014 [23]. The authors are partially supported by FAPESP (Grants #13/19760–5 and #14/09546–9) and CNPq. We would like to thank the anonymous reviewers for their valuable comments and suggestions. The real image of honey (Fig. 1) was provided by Domiriel[1] under Creative Commons license.

Appendix A. Supplementary material

Supplementary data associated with this paper can be found in the online version at http://dx.doi.org/10.1016/j.cag.2015.07.021.

References

[1] Liu GR, Liu MB. Smoothed particle hydrodynamics. World Science; 2005.
[2] Ihmsen M, Orthmann J, Solenthaler B, Kolb A, Teschner M. SPH fluids in computer graphics. In: Eurographics 2014—state of the art reports; 2014. p. 21–42.
[3] Herault A, Bilotta G, Dalrymple RA. SPH on GPU with CUDA. J Hydraul Res 2010;48:74–9.
[4] Krog ØE, Elster AC. Fast gpu-based fluid simulations using sph. In: Applied parallel and scientific computing. Springer; 2012. p. 98–109.
[5] Cross MM. Rheology of non-newtonian fluids: a new flow equation for pseudoplastic systems. J Colloid Sci 1965;20(5):417–37.
[6] Goktekin TG, Bargteil AW, O'Brien JF. A method for animating viscoelastic fluids. ACM Trans Graph 2004;23(3):463–8.
[7] Batty C, Bridson R. Accurate viscous free surfaces for buckling, coiling, and rotating liquids. In: Proceedings of the 2008 ACM SIGGRAPH/eurographics symposium on computer animation; 2008. p. 219–28.
[8] Bergou M, Audoly B, Vouga E, Wardetzky M, Grinspun E. Discrete viscous threads. ACM Trans Graph 2010;29(4) 116:1–116:10.
[9] Batty C, Uribe A, Audoly B, Grinspun E. Discrete viscous sheets. ACM Trans Graph 2012;31(4) 113:1–113:07.
[10] Batty C, Houston B. A simple finite volume method for adaptive viscous liquids. In: Proceedings of the 2011 ACM SIGGRAPH/eurographics symposium on computer animation. SCA'11; 2011. p. 111–8.
[11] Tomé MF, Mckee S. Numerical simulation of viscous flow: buckling of planar jets. Int J Numer Methods Fluids 1999;29(6):705–18.
[12] Tomé M, Filho A, Cuminato J, Mangiavacchi N, Mckee S. GENSMAC3D: a numerical method for solving unsteady three-dimensional free surface flows. Int J Numer Methods Fluids 2001;37(7):747–96.
[13] Tomé M, Grossi L, Castelo A, Cuminato J, Mangiavacchi N, Ferreira V, et al. A numerical method for solving three-dimensional generalized Newtonian free surface flows. J Non-Newton Fluid Mech 2004;123(2–3):85–103.
[14] Oishi CM, Tomé MF, Cuminato JA, McKee S. An implicit technique for solving 3d low Reynolds number moving free surface flows. J Comput Phys 2008;227(16):7446–68.
[15] Clavet S, Beaudoin P, Poulin P. Particle-based viscoelastic fluid simulation. In: Proceedings of the 2005 ACM SIGGRAPH/eurographics symposium on computer animation; 2005. p. 219–28.
[16] Keiser R, Adams B, Gasser D, Bazzi P, Dutré P, Gross M. A unified lagrangian approach to solid-fluid animation. In: Symposium on point-based graphics 2005; 2005. p. 125–34.
[17] Paiva A, Petronetto F, Lewiner T, Tavares G. Particle-based non-newtonian fluid animation for melting objects. In: Sibgrapi 2006 (XIX Brazilian symposium on computer graphics and image processing). IEEE; 2006. p. 78–85.
[18] Solenthaler B, Schläfli J, Pajarola R. A unified particle model for fluid–solid interactions. Comput Anim Virtual Worlds 2007;18(1):69–82.
[19] Paiva A, Petronetto F, Lewiner T, Tavares G. Particle-based viscoplastic fluid/solid simulation. Comput-Aided Des 2009;41(4):306–14.
[20] Rafiee A, Manzari M, Hosseini M. An incompressible SPH method for simulation of unsteady viscoelastic free-surface flows. Int J Non-Linear Mech 2007;42(10):1210–23.
[21] Xu X, Ouyang J, Yang B, Liu Z. SPH simulations of three-dimensional non-Newtonian free surface flows. Comput Methods Appl Mech Eng 2013;256:101–16.
[22] Xu X, Ouyang J. A SPH-based particle method for simulating 3D transient free surface flows of branched polymer melts. J Non-Newton Fluid Mech 2013;202:54–71.
[23] Andrade LFS, Sandim M, Petronetto F, Pagliosa P, Paiva A. SPH fluids for viscous jet buckling. In: Sibgrapi 2014 (27th conference on graphics, patterns and images); 2014. p. 65–72.
[24] Müller M., Charypar D., Gross M.. Particle-based fluid simulation for interactive applications. In: Proceedings of the 2003 ACM SIGGRAPH/eurographics symposium on computer animation; 2003. p. 154–9.
[25] Morris JP, Fox PJ, Zhu Y. Modeling low Reynolds number for incompressible flows using SPH. J Comput Phys 1997;136:214–26.
[26] Koshizuka S, Nobe A, Oka Y. Numerical analysis of breaking waves using the moving particle semi-implicit method. Int J Numer Methods Fluids 1998;26(7):751–69.
[27] Fang J, Owens RG, Tacher L, Parriaux A. A numerical study of the SPH method for simulating transient viscoelastic free surface flows. J Non-Newton Fluid Mech 2006;139(1–2):68–84.
[28] Hoberock J, Bell N. Thrust: A parallel template library; 2010. URL: ⟨http://www.thrust.github.io⟩; version 1.7.0.
[29] Green S. Particle simulation using CUDA; 2012. URL: ⟨http://www.docs.nvidia.com/cuda/samples/5_Simulations/particles/doc/particles.pdf⟩.
[30] Cruickshank JO. Low-Reynolds-number instabilities in stagnating jet flows. J Fluid Mech 1988;193:111–27.
[31] Blount MJ, Lister JR. The asymptotic structure of a slender dragged viscous thread. J Fluid Mech 2011;674:489–521.
[32] Shreiner D, Sellers G, Kessenich J, Licea-Kane B. OpenGL programming guide. 8 ed.Addison-Wesley; 2013.
[33] Gomez-Gesteira M, Rogers B, Crespo A, Dalrymple R, Narayanaswamy M, Dominguez J. SPHysics—development of a free-surface fluid solver. Part 1: theory and formulations. Comput Geosci 2012;48:289–99.
[34] Mayrhofer A, Gómez-Gesteira M, Crespo AJC, Rogers BD. Advanced preprocessing for SPHysics. In: Proceedings of the 5th international SPHERIC workshop; 2010.
[35] Bell N, Yu Y, Mucha PJ. Particle-based simulation of granular materials. In: Proceedings of the 2005 ACM SIGGRAPH/eurographics symposium on computer animation. SCA '05; 2005. p. 77–86.
[36] Witkin AP, Heckbert PS. Using particles to sample and control implicit surfaces. In: Proceedings of the 21st annual conference on computer graphics and interactive techniques. SIGGRAPH'94; 1994. p. 269–77.
[37] Geuzaine C, Remacle JFA. Gmsh: a 3-D finite element mesh generator with built-in pre- and post-processing facilities. Int J Numer Methods Eng 2009;79(11):1309–31.
[38] Schberl J. NETGEN an advancing front 2d/3d-mesh generator based on abstract rules. Comput Vis Sci 1997;1(1):41–52.

[1] www.flickr.com/photos/domiriel/8037182858