Two-Dimensional Stochastic Rotation Dynamics for Fluid Simulation

Benoît Crespin, Cong Tam Tran, Manuella Cerbelaud, Arnaud Videcoq, Emmanuelle Darles

Scientific Research Publishing

Journal of Computer and Communications, 2020, 8, 27-38
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Two-Dimensional Stochastic Rotation Dynamics for Fluid Simulation

Benoît Crespin[1,2*], Công Tâm Tran[1,2], Manuella Cerbelaud[3], Arnaud Videcoq[3], Emmanuelle Darles[4]

[1]University Limoges, XLIM/ASALI, UMR CNRS, Limoges, France
[2]Center for Mathematical Modeling, Santiago, Chile
[3]University Limoges, IRCER, UMR CNRS, Limoges, France
[4]University Poitiers, XLIM/ASALI, UMR CNRS, Poitiers, France
Email: *benoit.crespin@unilim.fr

How to cite this paper: Crespin, B., Tran, C.T., Cerbelaud, M., Videcoq, A. and Darles, E. (2020) Two-Dimensional Stochastic Rotation Dynamics for Fluid Simulation. *Journal of Computer and Communications*, **8**, 27-38.
https://doi.org/10.4236/jcc.2020.82003

Received: January 23, 2020
Accepted: February 17, 2020
Published: February 20, 2020

Abstract

This paper presents a new method for fluid simulation based on Stochastic Rotation Dynamics. The SRD model relies on a particle-based representation, but does not consider incompressibility. We generalize this model by introducing additional computation steps in order to handle this type of behavior, and also two-way coupling between the fluid and immersed objects. As a proof of concept, our method is implemented on the CPU to produce different types of simulations such as dam-break flood, falling droplets and mixing of two fluids.

Keywords

Computer Graphics, Fluid Simulation, Particle Systems

1. Introduction

Fluid simulations in Computer Graphics seek to provide an approximate solution to the Navier-Stokes equations (NSE) which represent the motion of a fluid. These equations can have different formulations, in the general form, they express the total time derivative of the velocity v_i of a single particle of fluid i as:

$$\frac{Dv_i}{Dt} = -\frac{1}{\rho_i}\nabla p_i + \nu\nabla^2 v_i + \frac{F}{m_i} \quad (1)$$

where ρ_i represents the density of the particle, p_i is pressure, m_i is mass, and ν is the kinematic viscosity. The term $\frac{1}{\rho_i}\nabla p_i$ accounts for the accelera-

tion of the particle due to pressure differences inside the fluid, which generally dominate all other forces. The term $\nu\nabla^2 v_i$ represents the acceleration due to friction forces between particles with different velocities, where ν is usually determined empirically to provide a realistic behavior. Finally, F represents external forces pushing the fluid, e.g. gravity. Existing particle-based methods have to cope with several problems when solving NSE. The first is the *nearest neighbor's search*, since at each iteration, we must determine the interactions of each particle with its closest neighbors. Collision detection and handling are also important to prevent particles to cross boundaries or to simulate two-way coupling with rigid objects immersed in the fluid. Finally, for incompressible fluids such as water, extra care must be taken to guarantee that the overall volume stays approximately constant during the simulation. Stochastic Rotation Dynamics [1] [2] [3] [4] is known to provide a generally good approximation to Equation (1) under certain conditions. It relies on a particle-based representation and a grid of regular square cells containing the particles. At each time step, particles are stored into a single cell, and particles in the same cell contribute to the center of mass velocity of this cell. The velocity of a particle at the next time step is updated by combining the center of mass velocity of its associated cell and a random rotation. However, this model is generally not able to reproduce the behavior of incompressible fluids, as the number of particles inside a cell does not stay constant.

After presenting existing particle-based methods in the next section, we present in Section 3 a generalization of the SRD model introducing additional computation steps, called *local repulsion* and *cell pressure* steps, in order to handle incompressibility. We also present a two-way coupling method that process collisions between the fluid and immersed objects. The implementation of our method on the CPU and the results we obtained are discussed in Section 4, including different types of simulations such as dam-break flood, falling droplets and mixing of two fluids.

2. Particle-Based Fluid Simulations

The *Smooth Particle Hydrodynamics* model (SPH) was proposed for water simulation by Muller *et al.* in 2003 [5], and is now one of the most popular methods in Computer Graphics. A complete survey of this approach can be found in [6]. The main idea is that each quantity A_i associated to particle i (*i.e.* viscosity, density, etc.) can be approximated by interpolating quantities at neighboring particles j with a kernel function W which depends on the distance $\|\vec{ij}\|$ between i and j:

$$A_i \approx \sum_j \frac{m_j}{\rho_j} A_j W\left(\|\vec{ij}\|\right) \tag{2}$$

The nearest neighbors' search usually relies on a grid with fixed-size cells and represents approximately 80% of the total computation time. The algorithm then

runs as follows at each time step and for each particle: compute density, then compute pressure, viscosity and external forces, and finally apply these forces to update velocity and position. To handle collisions with fixed or moving objects, most existing works make use of additional particles located at boundaries, which contribute to density and pressure computations and prevent penetration of fluid particles inside solid objects. Different methods were also proposed to guarantee incompressibility, where the main idea is to constrain density computation to reach a desired target value before pressure forces are applied.

Particle-In-Cell (PIC) [7] and Fluid-Implicit-Particle (FLIP) [8] are hybrid models that rely both on a particle system and on a grid with fixed-size cells (see [9] for a complete survey). As with SPH, particles are used to represent the fluid with their own position and velocity, but density, pressure and viscosity forces are computed at fixed positions in cells. These positions are usually defined at the midpoint of each edge (staggered Marker and Cell, or MAC). At each time step, velocities must be interpolated from particles to cells, and forces from cells back to particles. PIC and FLIP approaches essentially differ in the interpolation scheme, which can introduce dissipation, artificial viscosity and/or visible noise at the interface. Both approaches are combined in [10] to alleviate these problems and enforce incompressibility, which can be more easily achieved than with SPH. However, pressure computations now require to solve a large linear equations system on the grid, representing usually more than 1/3 of the total simulation time. Collision handling with boundaries can be achieved directly using the grid by marking cells as empty (or air), fluid or solid; for moving objects boundary particles can be used as with SPH.

A specific type of particle-based models can be found in the field of human crowds simulations [11] [12] [13]. Here two-dimensional particles represent individuals whose goal is to reach a specific destination, but at the same time must also avoid collisions with obstacles or other individuals. These approaches rely on concepts very similar to those found in PIC/FLIP methods mentioned above to handle density, friction, incompressibility, collision detection, etc. Such systems can be used for example to simulate the behavior of large, dense crowds evacuating a building, which indeed closely resembles to particle dynamics in a fluid.

The Stochastic Rotation Dynamics model (SRD), also known as Multi Particle Collision Dynamics (MPCD), was first introduced in [1] for fluid simulation at mesoscopic scale (for material larger than the nanoscale size in condensed matter physics). As in SPH or PIC/FLIP methods, fluid particles are defined by their position r_i and their velocity v_i, and are distributed in a regular cubic grid. We note a_0 the linear size of each grid cell (or SRD cell), and γ the desired average number of fluid particles inside a cell. At each iteration, a *collision* step and a *streaming* step are successively applied.

During the collision step and for each SRD cell, a rotation is applied to the velocities of each particle i inside this cell:

$$v_i\left(t+\Delta t\right)=v\left(t\right)+\mathrm{rot}_\alpha\left(v_i\left(t\right)-v\left(t\right)\right) \tag{3}$$

where Δt is the time step between two iterations, v is the center of mass velocity for this cell, and rot_α represents a two-dimensional rotation of angle α. This angle can be chosen randomly within a given interval, hence the name "stochastic". Figure 1 shows the decomposition of this computation for a single cell. Before the collision step, a translation can be applied to all particles with a random vector within the interval $\left[-a_0/2, a_0/2\right]$. This extra computation forbids particles to always interact with the same neighbors in order to restore the Galilean invariance assumption [2], but was not found to have a significant impact in our work where the neighborhood of a particle changes rapidly.

During the streaming step, particles are simply advected by:

$$r_i\left(t+\Delta t\right) = r_i\left(t\right) + v_i\left(t\right)\Delta t \tag{4}$$

Finally, some extra steps are needed to compute collisions and interactions between particles and solid objects immersed in the fluid. For example, repulsive interaction forces are explicitly computed between particles and solid grains to avoid inter-penetration in [3].

The SRD approach is designed for applications in condensed matter physics, and it provides a good approximation to Equation (1) if sufficiently small values are chosen for the grid resolution a_0 and the time step Δt. Its major advantage over the methods presented above is its very simple formulation, and also the fact that it does not need an expensive nearest neighbor search. However, it also suffers several weaknesses for applications in Computer Graphics. The main issue is that the number of particles inside an SRD cell can be very different from the desired value γ, leading to compressibility effects when adding gravity. As a result, it is hardly possible with SRD to achieve regular fluid simulations such as water jets, falling droplets, dam-break flood, etc.

To overcome these problems, our approach combines the simplicity of the SRD model with specific modifications designed to take other phenomena into account. These modifications, inspired by PIC/FLIP methods, are presented in the next section.

3. 2D SRD for Fluid Simulation: Our Approach

Figure 2 summarizes the computation of a complete-time step with our approach,

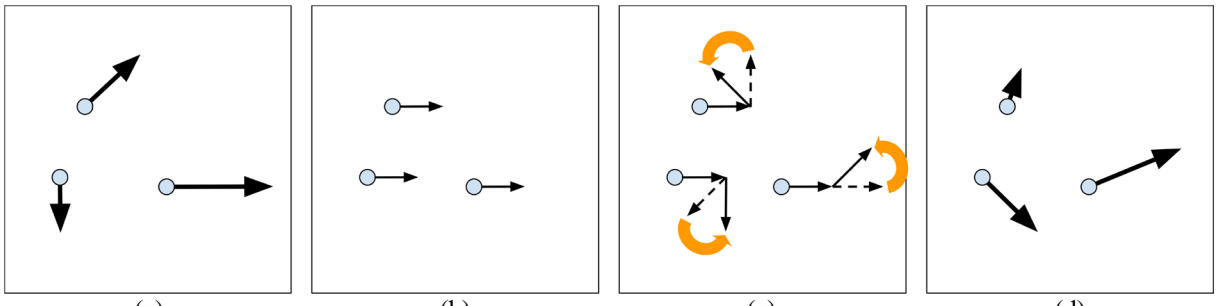

Figure 1. Collision step inside an SRD cell described by Equation (3): (a) Initial velocities v_i at time t; (b) Average velocity v assigned to each particle; (c) Rotation of $v_i - v$ by a random angle α; (d) Final velocities at time $\left(t+\Delta t\right)$.

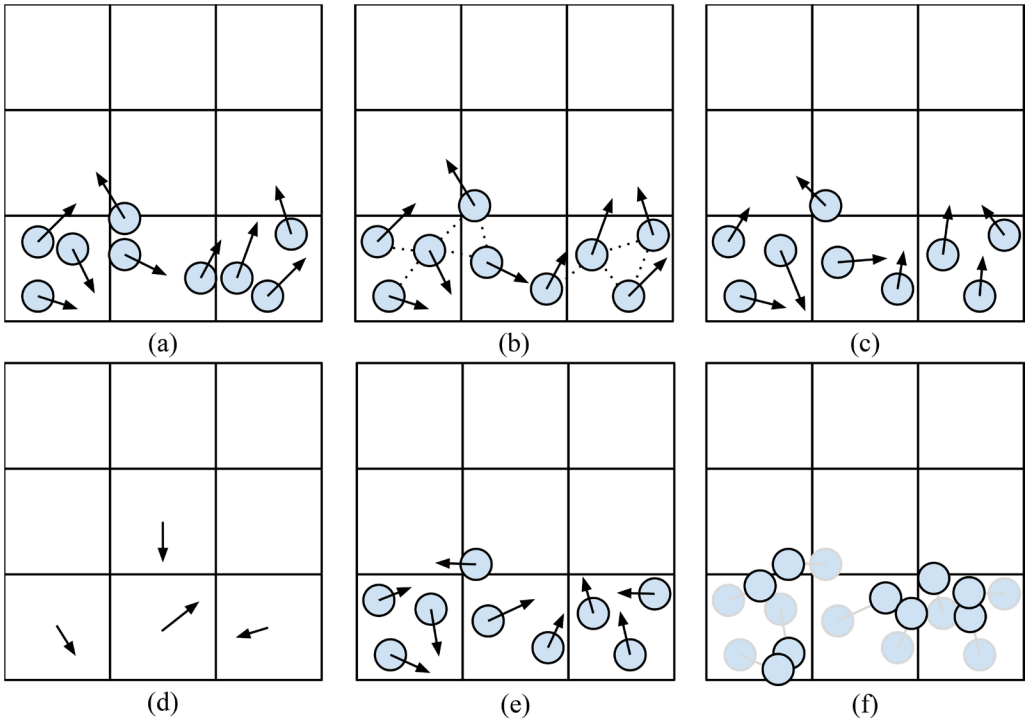

Figure 2. Summary of our algorithm: (a) Initial positions and velocities; (b) Local repulsion; (c) SRD collision step; (d) Computation of pressure gradient at the center of each cell; (e) Modification of velocities based on pressure; (f) Final positions.

starting with positions and velocities for the particles at time t (2a). First a *local repulsion* step is used to avoid particle clustering (2b), then we apply the SRD collision step described earlier (2c). To handle pressure effects, a pressure gradient is computed for each cell (2d) and is used to modify the local velocity of each particle (2e). Finally, we use this velocity to advect particles to their next position at time $t+\Delta t$ (2f). The following paragraphs focus on the computation of local repulsion and pressure forces, which are the key modifications we add to the original SRD method. We also describe how to handle collisions with fixed or moving objects.

3.1. Local Repulsion

The original SRD model does not prevent particles to lie too close to each other, which can generate particle clustering. The purpose of our *local repulsion* step is to displace particles such that they keep a minimal distance r_L between them. This parameter can be computed from the size of a cell a_0 and the desired average number of particles inside a cell γ:

$$r_L = \sqrt{\frac{2a_0^2}{\gamma\sqrt{3}}} \tag{5}$$

For each particle i, we first need to find which are its closest neighbors, thanks to a classical nearest neighbor search [6]. As this search is limited within radius r_L, which is lower than a_0, we only consider particles located inside the same

cell and its 8 neighboring cells. Let $\vec{ij}$ denote the vector from i to another particle j. If the distance $\left\|\vec{ij}\right\|$ between i and j is lower than r_L, then we add a displacement vector d to particle j, whereas i is displaced by $-d$:

$$d = \frac{r_L}{2}\left(1 - \frac{\left\|\vec{ij}\right\|}{r_L}\right)\frac{\vec{ij}}{\left\|\vec{ij}\right\|} \tag{6}$$

The velocities of i and j are also modified, using $-d\Delta v$ and $d\Delta v$ respectively, where Δv is a user-defined parameter (set to 0.1 in our implementation). The whole process can be repeated multiple times to ensure that all particles will eventually reach their correct position, as illustrated in Figure 2(b). In practice, our experiments showed that this simple approach converges very quickly, generally with only 3 runs.

3.2. Cell Pressure

Once the local repulsion completes, the velocity of each particle is updated using the SRD collision step (Equation (3)). However advecting particles at this stage is not sufficient to handle pressure differences inside the fluid, for example, if we apply a gravity force. As shown in Figure 3 (top) many particles can still enter a single cell, hence volume conservation is not guaranteed. To tackle this problem our *cell pressure* step defines a velocity field on the cell grid from the center of mass velocity v, which should stay *divergence-free*, *i.e.* $\nabla \cdot v = 0$. First, we compute the divergence $d_{x,y}$ at each grid cell:

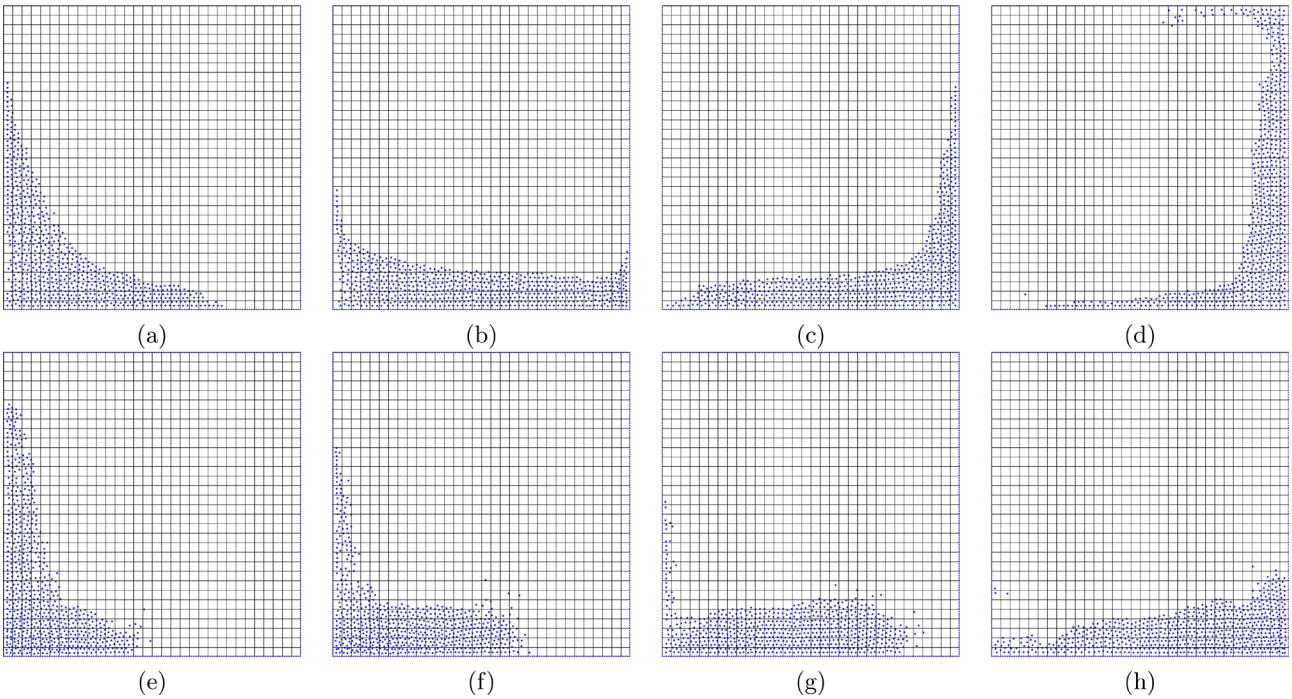

Figure 3. Top row: dam-break simulation using cell pressure computation, captured at frames 250 (a), 350 (b), 450 (c) and 650 (d). Bottom row (e-h): dam-break simulation using only local repulsion, captured for the same frames, where the loss of volume becomes visible.

$$d_{x,y} = \frac{-2a_0 r_{x,y}}{\Delta t}\left(v_{x+1,y} - v_{x-1,y} + v_{x,y+1} - v_{x,y-1}\right) \tag{7}$$

where $r_{x,y}$ represents the *density ratio* between the number of particles located inside the cell and the desired number γ. The pressure $p_{x,y}$ in each cell can be found by solving a linear equations system on the grid, as in PIC/FLIP methods. Our implementation uses the Jacobi method [14] and starts by setting $p^0_{x,y} = 0$. The following recurrence formula is applied during k iterations:

$$p^k_{x,y} = \frac{1}{4}\left(d_{x,y} + p^{k-1}_{x+2,y} + p^{k-1}_{x-2,y} + p^{k-1}_{x,y+2} + p^{k-1}_{x,y-2}\right) \tag{8}$$

In practice, if a cell does not contain any particle, we set its pressure $p^k_{x,y} = 0$ at each iteration. From the scalar pressure values, we can then compute the pressure gradient $\vec{p}_{x,y}$ in each cell by:

$$\vec{p}_{x,y} = \frac{\Delta t}{2a_0 r_{x,y}}\begin{pmatrix} p_{x+1,y} - p_{x-1,y} \\ p_{x,y+1} - p_{x,y-1} \end{pmatrix} \tag{9}$$

Finally, the velocity v_i of each particle is linearly interpolated with the pressure gradient $\vec{p}_{x,y}$ using the density ratio $r_{x,y}$ of its associated cell:

$$v_i = \left(1 - r_{x,y}\right)v_i + r_{x,y}\left(v_i - \vec{p}_{x,y}\right) \tag{10}$$

The number of iterations k for the Jacobi method can be chosen between 5 and 40, depending on the desired compromise between computation time and precision. In all our experiments, the value $k = 10$ was sufficient to ensure volume conservation, as shown in **Figure 3** (bottom).

3.3. Collision Handling

As noted previously, most existing works in fluid simulation use additional particles to handle collisions with objects. We apply the same technique for fixed boundaries, for example on the walls of the simulation box (see **Figure 4(a)**). These *solid* particles do not move but are considered during the local repulsion step to prevent fluid particles to cross a wall. However, it may still happen that a fluid particle flows out of the simulation box: in this case, we simply displace it back inside. We can also reduce its velocity in order to generate an adhesion

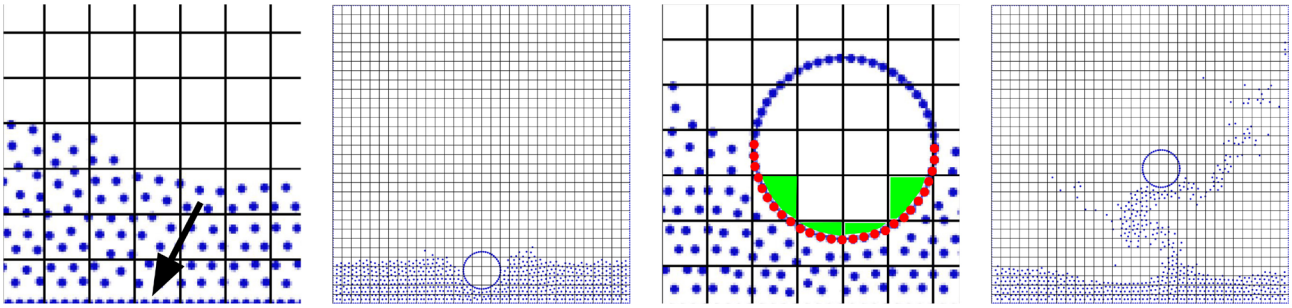

Figure 4. (a) Solid particles covering the bottom wall of the simulation box. (b and c) Ball falling inside the fluid at rest. (d) Ball flying up with the flow.

effect on the surface, or mirror its velocity in the opposite direction to generate a bouncing effect.

In the case of a moving object, we also have to create a set of solid particles on its boundaries at the beginning of the simulation. In the following, we take the example of a simple ball represented by a circle, coated with *solid* particles on its perimeter. We also store its position and its velocity v_b, used to advect the ball at each iteration. If a fluid particle enters the object during the local repulsion step, it will bounce or adhere to the surface as in the fixed case. Solid particles are marked with a boolean value if they interact with at least one fluid particle, *i.e.* if they are within distance r_L (represented in red in **Figure 4(c)**). At the end of this step, a fraction of the velocity of each marked particle is added to the velocity v_b, again using a user-defined parameter Δv set to 0.1 in our implementation. This generates a *two-way coupling* between the fluid and the object. During the cell pressure stage, solid particles are considered when computing divergence and pressure. This is especially important for the cells where a void is induced by the presence of the ball (represented in green in **Figure 4(c)**), which would otherwise lead to incorrect results. Finally, we can also simulate a buoyancy effect by reducing the gravity force applied to the ball depending on the number of marked particles on its boundary, roughly approximating the area of the fluid displaced by the ball.

Figure 4(b) and **Figure 4(d)** show two examples of our results. More complicated objects can also be handled with this approach, as long as their boundary can be discretized with solid particles. However, if the shape is not circular its orientation must also be taken into account, and the two-way coupling in this case should include a rotation matrix.

4. Results and Discussion

The fully animated version of the examples presented in this paper is available on video: https://bit.ly/336hb8R. Our CPU implementation running with Processing [15] contains approximately 500 lines of code, and can be downloaded here: https://bit.ly/31v8dk5. The code is structured into three basic classes:

- In the main class we define all the parameters, initialize the objects corresponding to particles and cells, then run the simulation iterations in an infinite loop. This loop also includes the interactions with the user (keyboard or mouse) and the rendering.
- The Particle class first declares the attributes of a particle such as its position, velocity, the index of its associated cell, etc. The computation of the local repulsion and SRD collision steps is defined here, as well as the application of the gravity and the pressure force, and finally the advection of a particle before the next iteration.
- The Cell class declares the attribute of a cell, including its position and a list of the particles it contains. All the code needed for cell pressure computation is defined here, such as average velocity, divergence, pressure, pressure gra-

dient and density ratio. An integer value is also used to characterize if the cell is empty, has an empty cell neighbor, or is surrounded by non-empty cells.

- An additional Ball class defines the behavior of the moving ball discussed in Section 1.

The two main parameters in our model are the size of a cell a_0 and the desired average number of particles inside a cell γ, even if other parameters can be tweaked. As an example in Figure 5 shows different configurations for the dam break simulation with varying values for a_0 and γ, that can be compared with Figure 3 where $a_0 = 10$ and $\gamma = 5$. If γ is very low, as in the first and third images, the simulation loses precision but the overall behavior is conserved. In all our experiments the other parameters are set as follows: the unit size is 1 pixel, the simulation box has size 640 by 640, the time step $\Delta t = 0.1$, the gravity is set to 9.81, the number of iterations for the local repulsion step to 3, and the number of Jacobi iterations for the cell pressure step $k = 10$.

Table 1 gives the computation time per iteration measured for the dam break simulation with different a_0 and γ values, as well as the percentage of time spent to compute local repulsion and cell pressure. As can be observed the repulsion step represents the main bottleneck for our CPU implementation, with more than 75% of the computation time. This can be explained by the fact that this step depends on the number of particles and makes use of an expensive nearest neighbor search. It is worth noticing that this percentage looks similar in SPH-based simulations, where the same bottleneck influences the simulation.

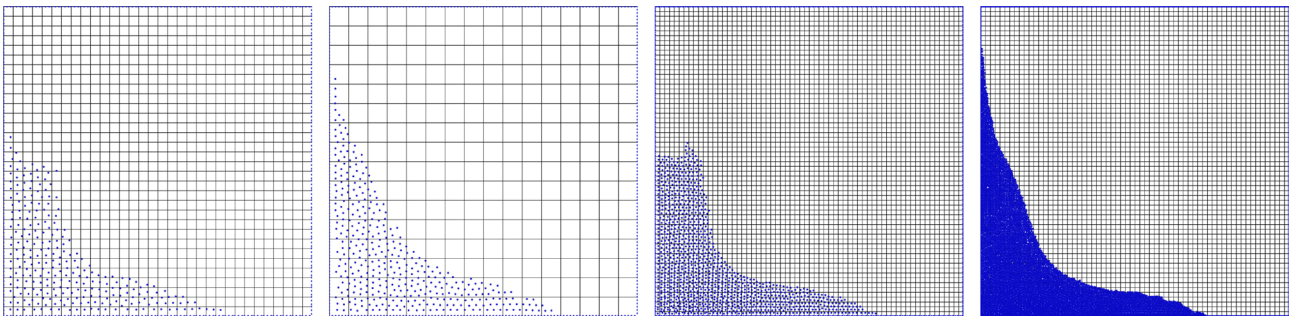

Figure 5. Dam-break simulation with different input parameters. From left to right: $(a_0 = 20, \gamma = 2)$ $(a_0 = 40, \gamma = 10)$ $(a_0 = 10, \gamma = 2)$ $(a_0 = 10, \gamma = 10)$.

Table 1. Computation times with different a_0 and γ values.

a_0	γ	#particles	#cells	Time per iteration (ms)	% repulsion	% cell pressure
	2	260	1024	0.8042989	76%	13%
40	10	376	256	1.8230367	96%	1%
20	5	660	1024	2.6646383	90%	4%
10	2	1134	4096	3.9304664	78%	12%
10	10	6114	4096	44.856846	96%	1%

On the contrary, the cell pressure step mostly depends on the number of cells and only has a limited impact on the total computation.

Different problems with our method are noticeable in some of the pictures presented in this paper. For example, the resolution of the grid can become visible at the interface between the fluid and the empty space, creating an aliasing effect as in the right image of Figure 5. Small bumps can also be observed on this interface, as in the left image of Figure 6 where the surface at rest is not perfectly flat. Another issue is that, due to the stochastic nature of the SRD method, our simulations exhibit non-symmetric behaviors, e.g. in Figure 6 where the droplet falling exactly at the center of the box generates different patterns on each side. In this case, the non-uniform distribution of particles inside the droplet also plays a part. One final issue is our local repulsion step which creates a rather rigid behavior in high pressure regions deep inside the fluid, where particles tend to wriggle to resist gravity.

On the other hand, despite its inherent approximations, our method succeeds at reproducing some interesting effects observed in fluid dynamics with dam-break flood or a falling droplet simulations. For example, realistic liquid sheets, *i.e.* particles in the air appearing linked by a cohesive force, are visible in Figure 4 and Figure 6. This behavior is rather surprising since our method does not specifically address the surface tension phenomenon that drives this type of effect. Figure 7 shows how our method is also able to generate vortices. Here the gravity is set to 0 and the fluid is separated into two populations of green and blue particles, depending on their initial position in the box. After the green particles

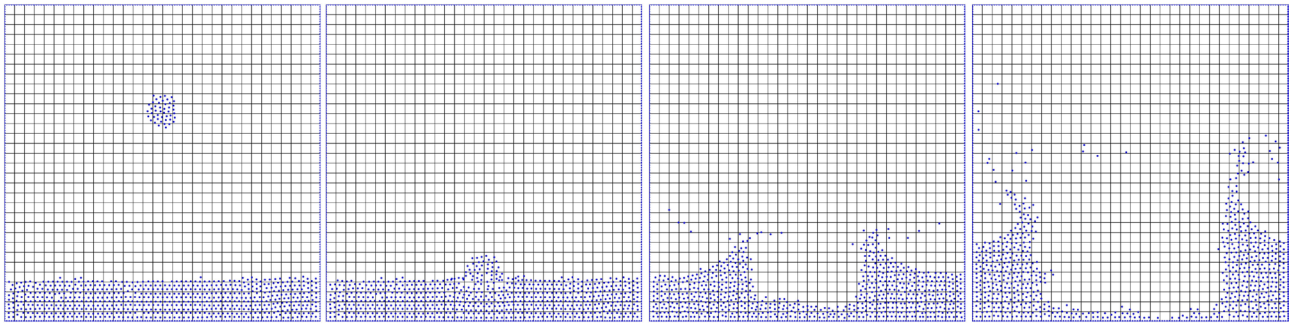

Figure 6. Droplet falling inside a volume of fluid, repelling particles on both sides and generating liquid sheets in the air.

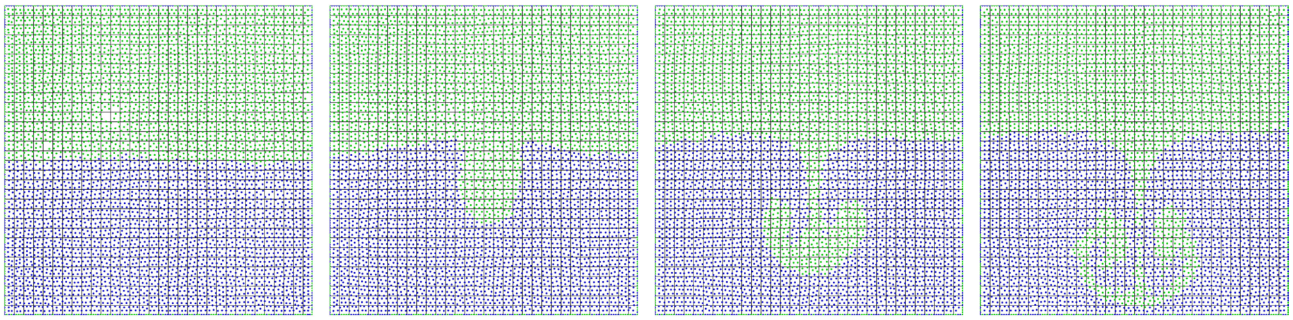

Figure 7. Vortex created at the interface between colored particles. Here gravity is set to 0 and the fluid is separated into two populations of green and blue particles. After the green particles are slowly pushed downwards, two vortices appear.

are slowly pushed downwards, two typical vortices start to appear.

5. Conclusions

The approach presented in this paper is based on the Stochastic Rotation Dynamics model, which by nature leads to instabilities because of the random rotation used in Equation (3). However, by extending this simple approach with additional steps related to pressure and collision handling, it is possible to obtain convincing results that faithfully reproduce the behavior of incompressible fluids. As a proof of concept, our method was implemented on the CPU to create different types of simulations such as dam-break flood, falling droplets and mixing of two fluids.

In future work, we would like to investigate further how our approach could be used to simulate viscous fluids, by modifying the angle parameter α in Equation (3) to damp the velocity of particles colliding inside a cell. The time step parameter could also play a part here since it directly impacts the overall number of SRD collisions.

We also wish to address the problems described in Section 4. The main improvement concerns the local repulsion step, which currently represents more than 75% of the total computation time and leads to instabilities in high pressure regions. This goal could be achieved through a parallelized GPU implementation, which is already available for compressible fluids [16]. This would reduce the cost of the nearest neighbor search and allow an increased number of particles in our simulations.

Acknowledgements

This work is supported by institutional grants from the LabEX SigmaLim (ANR-10-LABX-0074-01) and by the MIRES Federation.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Malevanets, A. and Kapral, R. (1999) Mesoscopic Model for Solvent Dynamics. *The Journal of Chemical Physics*, **110**, Article No. 8605. https://doi.org/10.1063/1.478857

[2] Ihle, T. and Kroll, D.M. (2001) Stochastic Rotation Dynamics: A Galilean-Invariant Mesoscopic Model for Fluid Flow. *Physical Review E*, **63**, Article ID: 020201. https://doi.org/10.1103/PhysRevE.63.020201

[3] Padding, J.T. and Louis, A.A. (2006) Hydrodynamic Interactions and Brownian Forces in Colloidal Suspensions: Coarse-Graining over Time and Length Scales. *Physical Review E*, **74**, Article ID: 031402. https://doi.org/10.1103/PhysRevE.74.031402

[4] Shakeri, A., Lee, K.-W. and Pschel, T. (2018) Limitation of Stochastic Rotation Dy-

namics to Represent Hydrodynamic Interaction between Colloidal Particles. *Physics of Fluids*, **30**, Article ID: 013603. https://doi.org/10.1063/1.5008812

[5] Muller, M., Charypar, D. and Gross, M. (2003) Particle-Based Fluid Simulation for Interactive Applications. In: *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 154-159.

[6] Ihmsen, M., Orthmann, J., Solenthaler, B., Kolb, A. and Teschner, M. (2014) SPH Fluids in Computer Graphics. Eurographics State-of-The-Art Reports, 21-42.

[7] Evans, M.W. and Harlow, F.H. (1957) The Particle-in-Cell Method for Hydrodynamic Calculations. Technical Report LA-2139, Los Alamos Scientic Lab, N. Mex.

[8] Brackbill, J.U., Kothe, D.B. and Ruppel, H.M. (1988) Flip: A Low-Dissipation, Particle-in-Cell Method for Fluid Flow. *Computer Physics Communications*, **48**, 25-38. https://doi.org/10.1016/0010-4655(88)90020-3

[9] Bridson, R. (2015) Fluid Simulation for Computer Graphics. 2nd Edition, Taylor & Francis, New York.

[10] Zhu, Y. and Bridson, R. (2005) Animating Sand as a Fluid. *ACM Transactions on Graphics*, **24**, 965-972. https://doi.org/10.1145/1073204.1073298

[11] Treuille, A., Cooper, S. and Popović, Z. (2006) Continuum Crowds. *ACM Transactions on Graphics*, **25**, 1160-1168. https://doi.org/10.1145/1141911.1142008

[12] Narain, R., Golas, A., Curtis, S. and Lin, M.C. (2009) Aggregate Dynamics for Dense Crowd Simulation. *ACM Transactions on Graphics*, **28**, 122:1-122:8. https://doi.org/10.1145/1618452.1618468

[13] Berglund, J. and Ristic, R. (2015) High-Density Real-Time Virtual Crowds via Unilaterally Incompressible Fluid Simulation. Technical Report, KTH, School of Engineering Sciences (SCI).

[14] Fernando, R. (2004) GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics. Pearson Higher Education, London.

[15] Reas, C. and Fry, B. (2014) Processing: A Programming Handbook for Visual Designers and Artists. The MIT Press, Cambridge, MA.

[16] Tran, C.T., Crespin, B., Cerbelaud, M. and Videcoq, A. (2018) Colloidal Suspension by SRD-MD Simulation on GPU. *Computer Physics Communications*, **232**, 35-45. https://doi.org/10.1016/j.cpc.2018.06.004