MethodsX 8 (2021) 101338

Contents lists available at ScienceDirect

MethodsX

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

Method Article

Implementation of lattice Boltzmann free-surface and shallow water models and their two-way coupling

Yann Thorimbert*, Bastien Chopard, Jonas Lätt

Department of Computer Science, University of Geneva, Carouge 1227, Switzerland

A B S T R A C T

• A detailed, practical description of a 2D lattice Boltzmann (LB) free-surface model and its coupling with a 1D LB shallow water model is provided.
• A Python code is provided, that implements the Gaussian droplet benchmark of the research article (Thorimbert et al., 2019) corresponding to this method article.
• Particular attention is given to the details of the free-surface implementation which, in the literature, vary among authors. These ambiguities must be addressed in order to build a reproducible scheme, as well as the exact implementation and parameters of the coupling model proposed in the associated research article.

A R T I C L E   I N F O
Method name: LBFS-SWFS coupling
Article history: Received 1 February 2021; Accepted 2 April 2021; Available online 8 April 2021

Method details

The model presented in the associated research article [13] involves three distinct ingredients: a D2Q9 free-surface solver, a D1Q3 shallow water solver and the coupling model itself. Whereas the implementation of a single-phase LB solver (for Navier–Stokes equations as well as for shallow water equations) is well-known by the community, the free-surface model is not free of ambiguities, as will be shown below. For this reason, the implementation of a LB free-surface solver is first discussed. In a second time, the implementation of the coupling scheme is presented. In this paper, code

DOI of original article: 10.1016/j.jocs.2019.01.006
* Corresponding author.
E-mail address: yann.thorimbert@unige.ch (Y. Thorimbert).

https://doi.org/10.1016/j.mex.2021.101338

variables and file names are referred to by using a typewriter font (e.g. `some_variable`). Python implementations of the D2Q9 free-surface solver (`lbfs.py`), D1Q3 shallow water solver (`lbsw.py`) and the coupling scheme (`drop.py`) are provided along with this article. In order to illustrate the link between the theoretical formulations of the coupling scheme and the practical implementation, this paper will refer to the code. The implementation of the Gaussian drop benchmark is also provided in `drop.py`.

Implementation of the lattice Boltzmann free-surface model

The LB free-surface model is used to simulate the interaction between a heavy fluid (e.g. water) and a light fluid (e.g. air), neglecting the dynamics of the light fluid, whose impact on the heavy fluid is only modeled through a pressure at the interface. Hence, the particle distribution functions in the cells belonging to the light fluid are ignored. The LB free-surface model constitutes an interesting alternative to pseudo-potential [1], color-gradient [2] or free-energy [3] approaches when the considered system is such that the difference of density and viscosity between the phases are large. The latter assumption becomes necessary to express a closing formula for the lattice populations streamed from the light fluid to the heavy fluid, as seen below. A free-surface flow in general is characterized by the condition of zero perpendicular normal stress and zero parallel shear stress to the interface, that is to say the forces at the interfaces are in equilibrium. This assumption is well verified in many common systems; again, a typical example is the water–air interface.

A D2Q9 free-surface model suitable for the lattice Boltzmann method (LBM) is presented in this section, using a volume-of-fluid (VOF) method to track the interface. Indeed, different approaches are available [4] to compute the position of the interface between light and heavy fluids: as opposed to interface-tracking techniques, which require the mesh to follow the interface along time, interface-capturing methods mark the location of the interface, whose shape is therefore deduced from these markers, allowing us to use a fixed grid. In this regard, the VOF interface-capturing method appears well suited for a LB free-surface solver, although the first VOF approach was proposed for a Navier-Stokes solver [5].

The algorithm presented here follows the description of [6,7], which itself finds its roots in the model developed by Korner and Singer [8] to study metal foams. Some parts of the algorithm (handling of interface artifacts, mass excess redistribution, completion of missing populations) used in this work, however, differ from the aforesaid sources. The aim of the present paper is to present a reproducible and simple implementation rather than an optimized formulation of the model. The file `lbfs.py` provides a Python implementation of the free-surface model.

Concepts of the LB free-surface volume-of-fluid model

The free-surface model considers two-phase fluid compounds with a large density ratio, akin to water–air systems in normal atmospheric conditions. It simulates the physics of the heavy fluid only, and replaces the effect of the light fluid onto the heavy one through a condition of zero parallel shear stress along two-phase interface. In the LB VOF approach, additionally to the usual fluid variables, every cell keeps track of the local mass m and volume fraction ϵ, which is related to the fluid density on interface cells as

$$\epsilon = \frac{m}{\rho}, \tag{1}$$

where m is the mass of heavy fluid in the considered cell. In bulk cells of the heavy fluid, the volume fraction takes the value $\epsilon = 1$.

To summarize, the cells of the fluid mesh are split into three categories:

- Fluid cells are completely occupied by the heavy fluid: $\epsilon = 1$.
- Empty (or gas) cells are completely occupied by the light fluid: $\epsilon = 0$.
- Interface cells are partially occupied by the heavy fluid: $\epsilon \in [0, 1]$.

A condition for the algorithm to work is that the interface layer between fluid and empty cells must be closed, so that an empty cell is never in the direct neighboring of a fluid cell (here,

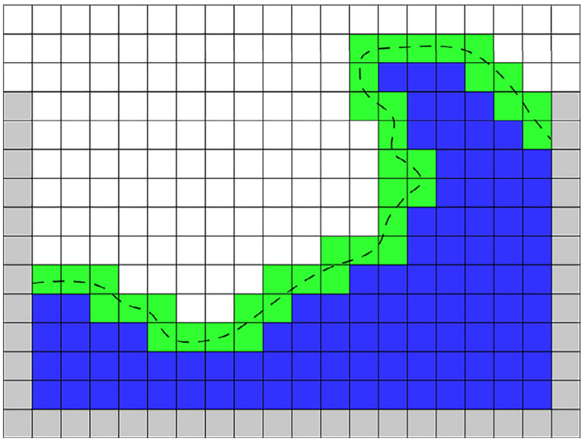

Fig. 1. Flags in a free-surface VOF simulation. Blue cells denote heavy fluid, green cells denote interface, white cells denote light fluid and gray cells are walls. The dashed line is a schematic view of the reconstructed surface.

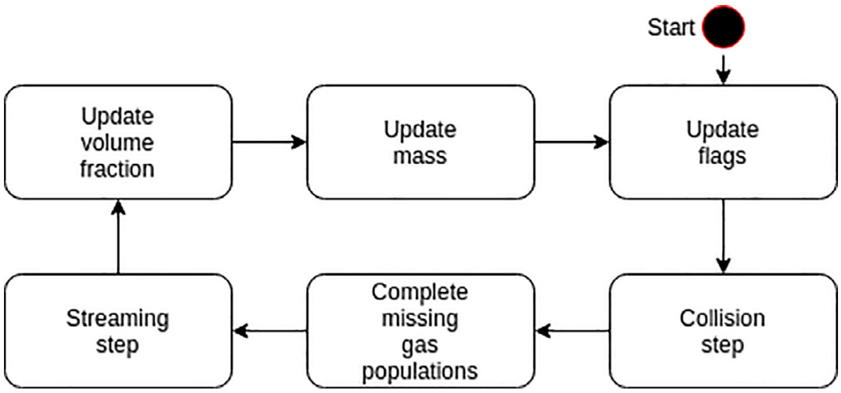

Fig. 2. Schematic representation of the key steps of the LB FS-VOF method.

'neighboring' has to be understood in the sense of the vicinity defined by the LB velocity set used). Fig. 1 depicts a typical free-surface flags field.

The different steps of the LB free-surface algorithm are discussed below. Fig. 2 summarizes these steps, that are clearly indicated in the method `iterate` of the `LatticeFS` class.

Completion of the missing populations from the empty cells

The populations from the empty cells cannot be streamed to the interface cells, since their value is arbitrary and meaningless. Assuming that the light fluid follows the motion of the heavy one, one requires that the kinematic viscosity of the heavy fluid is lower, and its density higher than that of the light fluid (again, think of a water–air interface). A possible completion scheme then (see [9] for further details), motivated by the conservation of the off-equilibrium populations at the interface, reads

$$f_I^{out}(\vec{x}+\vec{c}_i\Delta t) = f_i^{eq}(\rho_E, \vec{u}(\vec{x})) + f_I^{eq}(\rho_E, \vec{u}(\vec{x})) - f_i^{out}(\vec{x}), \tag{2}$$

where $f_I^{out}(\vec{x}+\vec{c}_i\Delta t)$ is the missing population coming from an empty cell, the superscript *out* denote the post-collision populations (before streaming), I denotes the population opposed to i and ρ_E is the fictitious density of the light fluid, accounting for the pressure of the gas on the surface and set to 1 in all the simulations of this work. In the free-surface code, this step is done in the method `gas_streaming` of the `LatticeFS` class. After this completion step has been applied to all the neighboring empty cells of the interface, a standard streaming step can be applied to the whole domain, without any flag-dependant treatment of the cells.

Note that the accuracy of Eq. (2) depends on the accuracy of the assumption of low viscosity, high density heavy fluid (compared to light fluid), since the fictitious equilibrium populations of the empty cell are computed using the velocity of the heavy fluid (i.e. it is assumed that the light fluid follows the heavy fluid at the interface).

It is also worth noting that in the reference algorithm from [10], the populations streamed from a direction that is normal to the interface are also treated as missing populations, even though the source cell is not an empty cell. In this work, this step of the scheme has been ignored, since no significant difference has been observed for the types of simulations targeted.

Mass update

The change of mass Δm_i due to the populations exchange in a given direction i reads

$$\Delta m_i(\vec{x}, t+\Delta t) = \frac{\epsilon(\vec{x}) + \epsilon(\vec{x}+\vec{c}_i\Delta t)}{2}\left[\underbrace{f_I(\vec{x}, t)}_{\text{entering mass}} - \underbrace{f_i(\vec{x}+\vec{c}_i\Delta t, t)}_{\text{leaving mass}}\right], \tag{3}$$

where I denotes the population in the opposite direction of i. It the above expression, an average of the volume fraction of the cells implied in the mass exchange is done in order to account for the amount of fluid in the cells. Finally, the mass of a cell is updated as

$$m(\vec{x}, t+\Delta t) = m(\vec{x}, t) + \sum_{i=0}^{Q} \Delta m_i(\vec{x}, t+\Delta t). \tag{4}$$

This step is performed in the method `mass_update` of the `LatticeFS` class.

Flag update

After the mass and volume fraction of the interface cells are updated, two lists of coordinates can be built:

- List `i2f` (interface to fluid): interface cells for which $\epsilon > 1+\kappa$.
- List `i2e` (interface to empty): interface cells for which $\epsilon < -\kappa$.

The small parameter κ allows for a slight inertia in the process of flag update, preventing fast switching of phase in locations where the volume fraction is near from the transition values. Indeed, the mass excess redistribution (discussed below) does not take into account the value of κ, hence this method introduces a slight delay, preventing the cells to change of type immediately.

The cells belonging to `i2f` will be converted to fluid cells, whereas cells belonging to `i2e` will be converted to empty cells.

An important point during this step is to properly handle mass excess; indeed, when a cell becomes empty, its mass is actually lower than 0, whereas it is larger than 1 when it becomes totally filled. The mass excess ($m_e = m - \rho$ in the case of a cell becoming fluid, and $m_e = m$ in the case of a cell becoming empty) must be redistributed somewhere else in the domain, in order to preserve the total mass of the fluid.

Different heuristics can be chosen for the mass redistribution. In this work, one simply counts the number of adjacent interface cells and equally redistribute the mass excess among these candidates. As already noted by Thürey [10], a more elaborated scheme would stream the mass excess according to the local flow direction.

Conversion of cell types

Depending on the type of conversion, the process to change the type of cell can be of different complexity, as the information beared by the cell must either be compressed or created. Note that this process cannot be asynchronous because of the nature of the mass redistribution scheme. Hence, in a first pass, the mass excesses are collected and the flags updated, before the mass is redistributed to the non-empty cells during a second pass. The following changes can occur:

- Empty to interface: as no information is contained in an empty cell, an average over all its non-empty adjacent cells is taken to determine the local macroscopic variables. The populations are then initialized at equilibrium, whereas the mass and volume fraction are initialized to zero, as this value is not used until the next update of the cell's properties, taking place after the collision and streaming (see Fig. 2). Note that, to be consistent with the completion scheme of the streaming through the interface, one could consider to initialize the populations with the average deviation from equilibrium of the adjacent non-empty cells. The flag of the cell is updated to "interface".
- Interface to empty: the mass excess is collected as $m_e = m$. The volume fraction and mass are then set to zero, and the density is set to ρ_E. The flag of the cell is updated to "empty".
- Interface to fluid: the mass excess is collected as $m_e = m - \rho$. The volume fraction and mass are then set to the unity, and the flag of the cell is updated to "fluid."
- Fluid to interface: The volume fraction and mass are set to unity (remind that the collision step occurs right after the flag update step). The flag of the cell is updated to "interface".

Although very rare, some artifacts can happen and cause instabilities, among which:

- Interface cells surrounded by interface cells only. In this case, if $\epsilon > 1/2$ the cell is converted to a fluid cell, otherwise it is converted to an empty cell.
- Interface cells surrounded by no empty cells. In this case the cell is converted to a fluid cell.
- Interface cells surrounded by no fluid cells. In this case the cell is converted to an empty cell.

In the free-surface code, the identification of the cells to update is performed in the method `get_updates` of the `LatticeFS` class, whereas the cell type conversion and the mass redistribution process are done in the method `update_flags`.

Implemention of the coupling scheme

In order to couple the shallow water and free-surface models presented above, the coupling strategy described in the associated research article has been implemented in the Python module `drop.py`, which makes use of the previously discussed free-surface implementation.

General scheme

Following the notation of the article, four points (A, B, C and D on Fig. 3) determine the different subdomains of the simulation. For the sake of simplicity, the shallow water domain actually spreads over the whole domain, although its values are ignored where the free-surface fluid is simulated. It has been shown in the article that the impact of this simplification on the performance is negligible for two reasons. First, the shallow water model is intrinsically more efficient than the free-surface one, for obvious reasons of dimensionality as well as complexity of the model. Second, the typical aim of such a coupling is to use the free-surface simulation over a restricted area, whereas the shallow water model simulates the vast majority of the system.

The global scheme (function `iterate_system` of the class `Coupling`) includes the following steps:

1. Iterate the shallow water lattice.
2. Iterate the free-surface lattice. For sake of simplicity, the extrapolated velocity profile at the interface column with the shallow water lattice is recomputed, though it is the same as the one used in the last coupling step. This velocity is then imposed to the free-surface sites as a Zhou-He boundary condition [11].
3. Coupling from shallow water to free-surface lattice.
4. Coupling from free-surface to shallow water lattice.

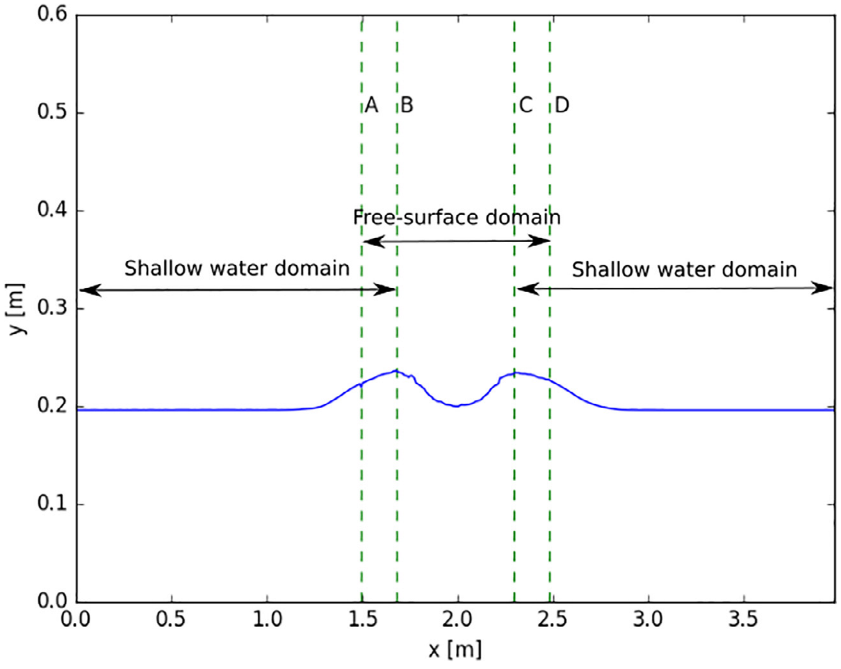

Fig. 3. Configuration of the lattices and their coupling. The points A, B, C and D define the limits of the different, overlapping domains. The configuration is symmetric.

Shallow water to free-surface coupling

In the function `sw2fs_generic` of the class `Coupling`, a loop runs over the cells of the interface column of the free-surface lattice. The type (empty, interface or fluid) of each cell is checked to ensure the consistency with the water height prescribed by the overlapping shallow water cell. Denoting h the water height to be imposed and y the height of the current cell examined ($y = 0$ denotes the bottom of the domain), the theoretical density of each cell is computed as

$$\rho(y) = 1 + g(h_0 - y), \tag{5}$$

where h_0 is the still water level. Similarly, the velocity $u(y)$ depends on the depth. In the implementation proposed, this velocity has a quadratic form:

$$u(y) = \frac{3\bar{u}y}{h}\left(1 - \frac{y}{2h}\right), \tag{6}$$

where $\bar{u}$ in this case corresponds to the velocity in the corresponding shallow water site.

The following cases are treated, where $floor(x)$ denotes the rounding down of x to the next integer:

- If $y = floor(h)$ then the cell must be an interface cell. If not, its tag is updated and its mass and volume-of-fluid values are set as $h - floor(h)$. If the cell was an empty cell prior to the conversion, then its populations are initialized at equilibrium using $\rho(h)$ and $u(h)$; otherwise they are left unchanged.
- If $y < floor(h)$ then the cell must be a fluid cell. If not, its tag is updated and its mass and volume-of-fluid values are set to unity. Again, the populations are initialized at equilibrium using $\rho(h)$ and $u(h)$ in case the cell was tagged as empty.
- If $y > floor(h)$ then the cell is emptied.

The water level and the flow velocity are also stored at this point in order to be used in the next velocity profile extrapolation.

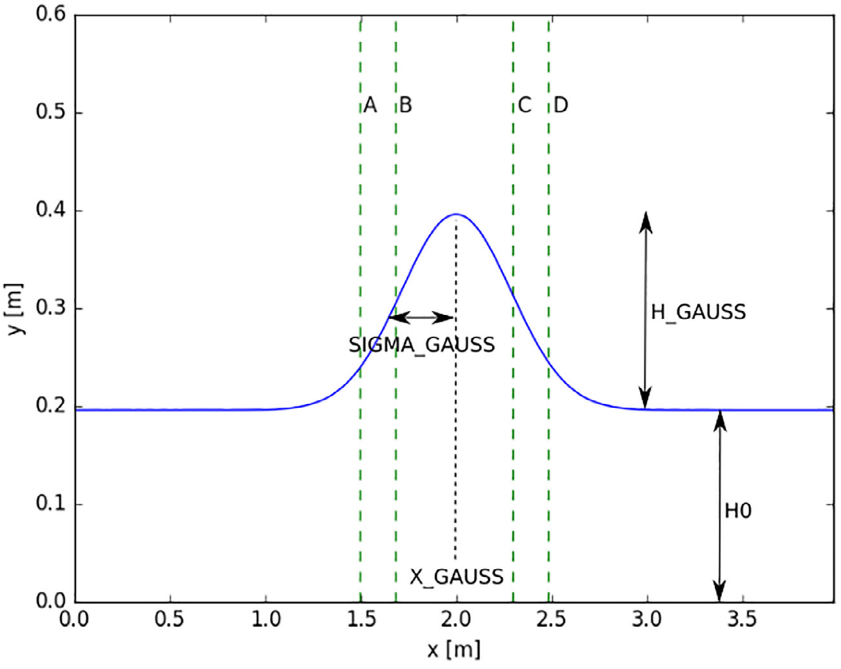

Fig. 4. Configuration of the initial perturbation in the gaussian drop benchmark along with the different variable names.

Free-surface to shallow water coupling

This step is essentially implemented inside the function `fs2sw` of the class `Coupling`. The height of the water column at the interface is computed along with its average velocity. The populations of the shallow water lattice are then set accordingly. Another important point here is to rescale the populations of the shallow water cell next to the interface as prescribed in [12]:

$$f_i = \frac{h_{FS}}{h_{SW}} f_i, \tag{7}$$

for each population i, where h_{SW} is the water height at this cell computed from the pre-scaling populations, and h_{FS} is the height at the free-surface site that overlaps the considered cell:

$$h_{FS} = \sum_{y=0}^{H} \epsilon(y), \tag{8}$$

with H the number of wet cells (i.e. cells that are either "fluid" or "interface") in the considered column of fluid.

Set up of a simulation

The initial configuration of the system in the Gaussian drop benchmark is illustrated in Fig. 4 along with the variable names corresponding to the different parameters of the initial perturbation in `drop.py`.

The function `save_plot` provided in the same file allows for the visualization of the water elevation at different time steps. The function `write_height` writes the water elevation in a file on the form of a list of values separated by white spaces, from $x = 0$ to $x = n_x$ (n_x being the total number of cells, not taking twice into account the overlapping cells). As indicated in the article, a linearly interpolated value is used where the lattices are overlapping.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.mex.2021.101338.

References

[1] X. Shan, H. Chen, Lattice Boltzmann model for simulating flows with multiple phases and components, Phys. Rev. E 47 (1993) 1815–1819, doi:10.1103/PhysRevE.47.1815.

[2] A.K. Gunstensen, D.H. Rothman, Microscopic modeling of immiscible fluids in three dimensions by a lattice Boltzmann method, EPL (Europhys. Lett.) 18 (2) (1992) 157.

[3] M.R. Swift, E. Orlandini, W.R. Osborn, J.M. Yeomans, Lattice Boltzmann simulations of liquid-gas and binary fluid systems, Phys. Rev. E 54 (1996) 5041–5052, doi:10.1103/PhysRevE.54.5041.

[4] T.E. Tezduyar, Interface-tracking and interface-capturing techniques for finite element computation of moving boundaries and interfaces, 2006, doi:10.1016/j.cma.2004.09.018.

[5] C.W. Hirt, B.D. Nichols, Volume of fluid /VOF/ method for the dynamics of free boundaries, J. Comput. Phys. 39 (1981) 201–225, doi:10.1016/0021-9991(81)90145-5.

[6] C. Korner, M. Thies, T. Hofmann, N. Thürey, U. Rüde, Lattice Boltzmann model for free surface flow for modeling foaming, J. Stat. Phys. 121 (1–2) (2005) 179–196, doi:10.1007/s10955-005-8879-8.

[7] N. Thürey, U. Rüde, Stable free surface flows with the lattice Boltzmann method on adaptively coarsened grids, Comput. Vis. Sci. 12 (5) (2009) 247–263, doi:10.1007/s00791-008-0090-4.

[8] C. Korner, R.F. Singer, Numerical simulation of foam formation and evolution with modified cellular automata, Metal foams and porous metal structures (1999) 91–96.

[9] M. Thies, Lattice Boltzmann Modeling With Free Surfaces Applied to Formation of Metal Foams, 2005 Ph.D. thesis, https://opus4.kobv.de/opus4-fau/frontdoor/index/index/docId/201.

[10] N. Thürey, Physically Based Animation of Free Surface Flows With the Lattice Boltzmann Method, University of Erlangen, 2007 Ph.D. thesis.

[11] Q. Zou, X. He, On pressure and velocity boundary conditions for the lattice Boltzmann BGK model, Phys. Fluids 9 (1997) 1591–1598, doi:10.1063/1.869307.

[12] N. Thürey, U. Rüde, M. Stamminger, Animation of open water phenomena with coupled shallow water and free surface simulations, in: Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2006, pp. 157–164.

[13] Y. Thorimbert, J. Lätt, B. Chopard, Coupling of lattice Boltzmann shallow water model with lattice Boltzmann free-surface model, Journal of Computational Science 33 (2019) 1–10, doi:10.1016/j.jocs.2019.01.006.