UNIVERSITY OF PORTSMOUTH

Lattice Boltzmann: Boundary Conditions and all that

HAMIDREZA KHALEGHZADEH

On the Edge

Last week we introduced the basic ideas of Lattice Boltzmann Models, and in particular gave the update rule for “typical” regions of the simulation where fluid is flowing freely.

What we didn’t discuss then is how deal with “edges” of the bulk fluid flow – inlet and outlets, edges, obstacles, and more generally phase boundaries.

This is a complex topic, and we can only treat some select common cases – enough to be able to set up some interesting simulations.

We will also discuss regimes in which LBMs are most useful for fluid dynamics.

Initial Conditions

In general for a fluid simulation we will be given some initial values of the fluid density and flow velocity as a function of position (ρ_0, $\boldsymbol{u}_0$)

For many purposes, it may be sufficient to start with:

$$f_i(x, y, t = 0) = f_i^{eq}\big(\rho_0(x, y), \boldsymbol{u}_0(x, y)\big)$$

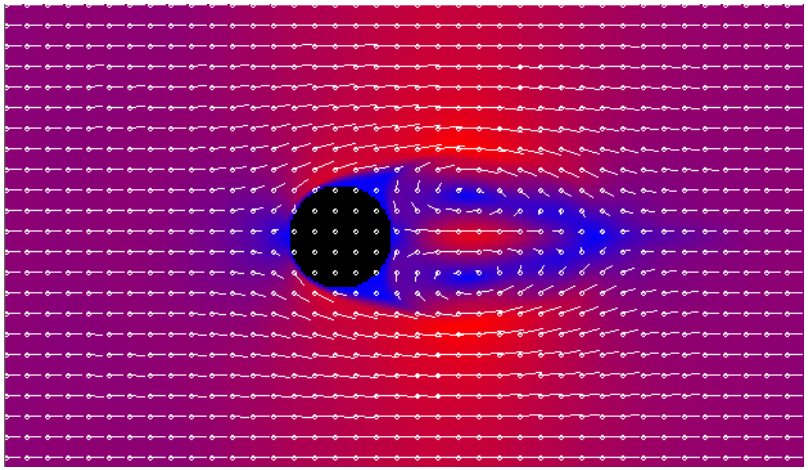
where the equilibrium distribution f_i^{eq} was defined last week.

LBM works best for *"weakly compressible"* flows where density doesn't change much as a function of time or position. In a simulation we often choose units so density is close to 1.0.

Update rule

After initialising the simulation grid, we have three steps for computing the next generation of cell states:

1. *Collision step*
2. *Streaming step*
3. *Streaming step in boundaries*



Stationary Obstacles and Edges

Some of the simplest boundary conditions that can be applied in LBMs are useful at *stationary obstacles* in the flow, or *stationary edges* bounding the flow.

In traditional fluid dynamics a common approach here is the *no-slip* condition – essentially we assume that fluid in contact with the obstacle is itself stationary.

Pretty much all kinds of boundary condition in LBMs have many and various approaches to their implementation, with different advantages and disadvantage. We will only consider the simplest approach for the no-slip condition, where the boundary condition is applied in the *collision step*, as a *bounce-back operation*.

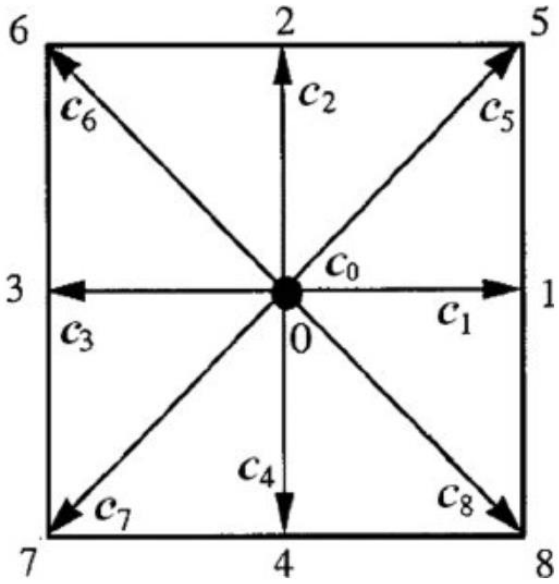
Bounce-Back Method

Considering the D2Q9 velocity states for illustration, define:

$\bar{0} = 0, \bar{1} = 3, \bar{2} = 4$, etc.

In general $\bar{\imath}$ is the index of the state with opposite velocity to state i, or:

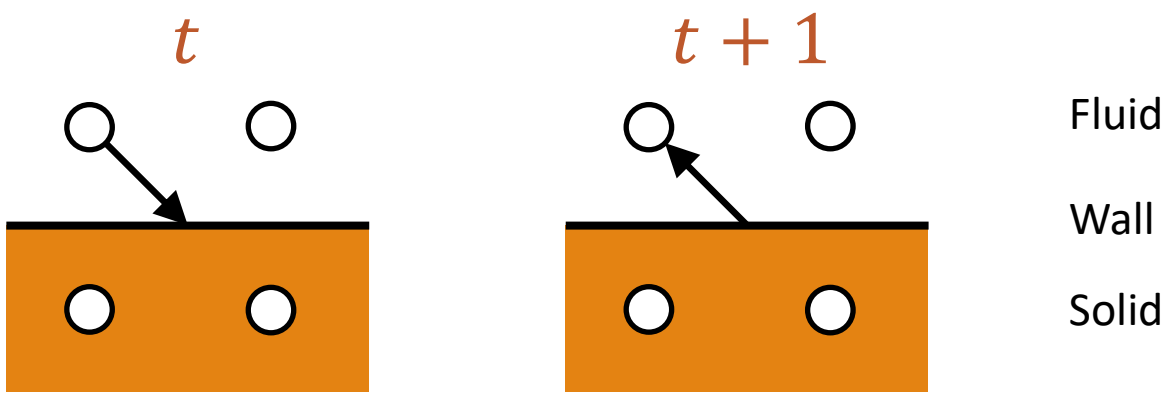
$$c_{\bar{\imath}} = -c_i$$



In the *bounce-back method* we treat cells that form the boundary of the obstacle as if they are within the simulation region, and the normal collision step at these cells is replaced with simply:

$$f_i^{out}(x, y) = f_{\bar{\imath}}^{in}(x, y)$$

Visualization of Bounce Back

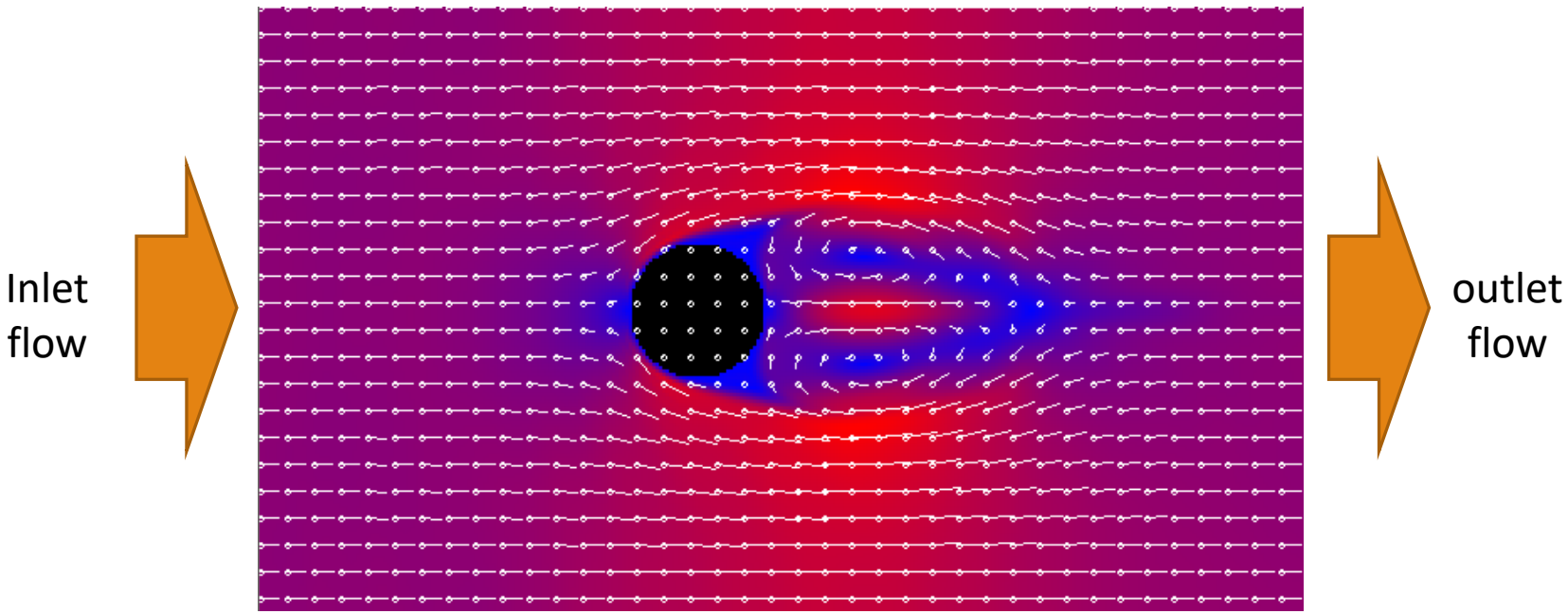


t state is before streaming, and *t+1* state after streaming, when particle has been bounced back from right hand cell in obstacle boundary.

May seem particle should be reflected forward. But that would imply net, time-average, transverse motion to right – in *no-slip* fluid is supposed to be static with respect to boundary.

Inlets and Moving Boundaries

Suppose there is a known flow into (or indeed out of, or transverse to) the simulation region, relative to a flat plane, or straight line, boundary, with a known velocity profile, e.g the situation on the *left side* of the simulation we ran last week:



Moving Planar Boundary – Step 1

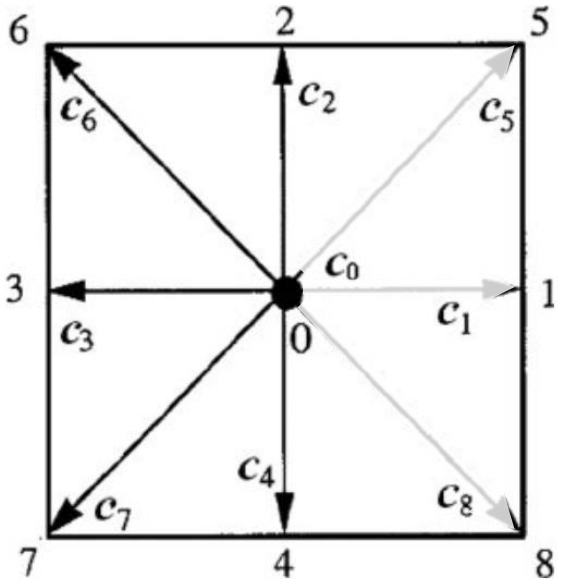
Consider the left hand boundary (for illustration), i.e. $x = 0$.

The *first stage* of the update on this boundary is to calculate the *macroscopic quantities*, ρ and $\boldsymbol{u}$.

By assumption, we know the flow velocity $\boldsymbol{u}$ at the boundary, but we don't know ρ.

From the immediately preceding streaming step, we will also know the distributions f_i for the solid velocity states, because these states were streamed from *within* the simulation region (to right).

Don't know greyed out states, because notionally would have come from outside simulation region.



Moving Planar Boundary – Step 1

But we have:

$$\rho = \sum_i f_i = (f_0 + f_2 + f_3 + f_4 + f_6 + f_7) + (f_1 + f_5 + f_8)$$

and:

$$\rho u_x = -(f_3 + f_6 + f_7) + (f_1 + f_5 + f_8)$$

Here $(f_1 + f_5 + f_8)$ is unknown, but it can be eliminated between the two equations to get:

$$\rho = \frac{(f_0 + f_2 + f_4) + 2(f_3 + f_6 + f_7)}{1 - u_x}$$

where the fs in the numerator and $\boldsymbol{u}$ are already known. So the second macroscopic quantity, ρ, is now known (though f_1, f_5 and f_8 individually are not).

Moving Planar Boundary – Step 2

Now we have the macroscopic quantities, since we can now compute the equilibrium distribution, we can move onto the collision step.

The collision step update is:

$$f_i^{out}(x,y) = f_i^{in}(x,y) - \frac{f_i^{in}(x,y) - f_i^{eq}(x,y)}{\tau}$$

The problem here is that we still don't know the starting distributions that should have been streamed in from outside the simulation - f_1, f_5 and f_8 in our example.

A popular and potentially more accurate scheme to estimate f^{out} components is called the *Zou-He method*.

Zou-He for Moving Boundary

This is also known as the *Non-Equilibrium Bounce-Back* method (see §5.5.2.1 in TLBM:PP).

We define the *non-equilibrium distribution components* as:

$$f_i^{neq} = f_i - f_i^{eq}$$

where all equilibrium components f^{eq} are already known.

To find the unknown components (1, 5, 8 in our example), we apply a modified bounce-back scheme to the non-equilibrium parts:

$$\boldsymbol{f_1^{neq}} = f_{\overline{1}}^{neq} = f_3^{neq} = \boldsymbol{f_3 - f_3^{eq}}$$

$$\boldsymbol{f_5^{neq}} = f_{\overline{5}}^{neq} - N_y = f_7^{neq} - N_y = \boldsymbol{(f_7 - f_7^{eq}) - N_y}$$

$$\boldsymbol{f_8^{neq}} = f_{\overline{8}}^{neq} + N_y = f_6^{neq} + N_y = \boldsymbol{(f_6 - f_6^{eq}) + N_y}$$

The "Unknown" Components

In terms of the total distribution components these equations are:

$$f_1 = f_1^{eq} + (f_3 - f_3^{eq})$$

$$f_5 = f_5^{eq} + (f_7 - f_7^{eq}) - N_y$$

$$f_8 = f_8^{eq} + (f_6 - f_6^{eq}) + N_y$$

Here N_y is one free parameter called the *transverse momentum correction*. It's value can be fixed by making sure the resulting distribution gives the known value for the boundary transverse velocity u_y – somewhat as we did for ρu_x on slide 9. There is a worked example on p198 of TLBM:PP.

In the example of last week's lab, the inlet flow had zero transverse velocity u_y, and N_y was identically zero.

Viscosity and Reynolds Number

Viscosity is a measure of a fluid's resistance to deformation, or how "thick" it is (treacle is more viscous that water, water is more viscous than air, etc)

In simple cases it can be quantified as a number ν, measured in metres squared per second (*kinematic viscosity*).

The *Reynolds Number* of a flow, denoted Re, is defined by:

$$Re = |\boldsymbol{u}|L/\nu$$

where $|\boldsymbol{u}|$ is some characteristic velocity of the flow and L some characteristic length scale (e.g. size of obstacle).

High Re tends to be associated *turbulent flow*, low Re with *laminar flow*.

Two flows geometrically similar but on different scales tend to behave similarly if they have the same Re (consider wind tunnel modelling)

Viscosity in LBMs

In lattice units, kinematic viscosity is related to the relaxation parameter τ:

$$\nu = {c_s}^2(\tau - 1/2)$$

where ${c_s}^2 = 1/3$ (see for example §7.2.1.1 in TLBM:PP for a discussion of this).

High Reynolds number simulations using LBMs are discussed in the literature, but may require more sophisticated treatment, e.g. of boundary conditions, than we have discussed here (the sample codes from the labs seem to become unstable for $Re \gtrsim 1000$).

References

Timm Kruger et al, *The Lattice Boltzmann Method: Principles and Practice*, 2017 (TLBM:PP)

Nils Thürey and Ulrich Rüde, *Free Surface Lattice-Boltzmann fluid simulations with and without level sets*, 2004

Palabos, University of Geneva, *Lattice Boltzmann sample codes in various other programming languages*†

†An early version of our lab codes was transcribed to Java from a Numpy script you may find there.