Create Your Own Lattice Boltzmann Simulation (With Python)

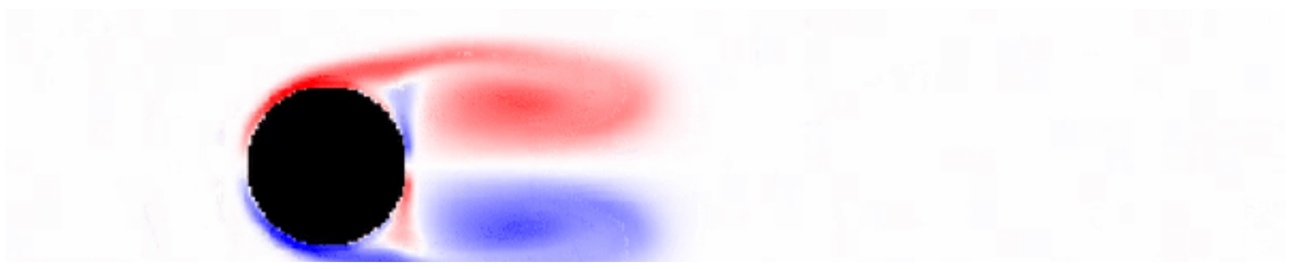
Philip Mocz · Follow
Published in The Startup · 5 min read · Dec 21, 2020

467 6

For today's recreational coding exercise, we simulate fluid flow past a cylinder using the Lattice Boltzmann method. This is a really cool and simple technique to simulate fluid flow: instead of evolving the fluid (Navier-Stokes) equations directly, microscopic particles on a lattice are simulated with streaming and collision processes. The power of the method comes from reducing the high-dimensionality of the microscopic physics onto the lattice, which has limited degrees of freedom.

You may find the accompanying Python code on github.

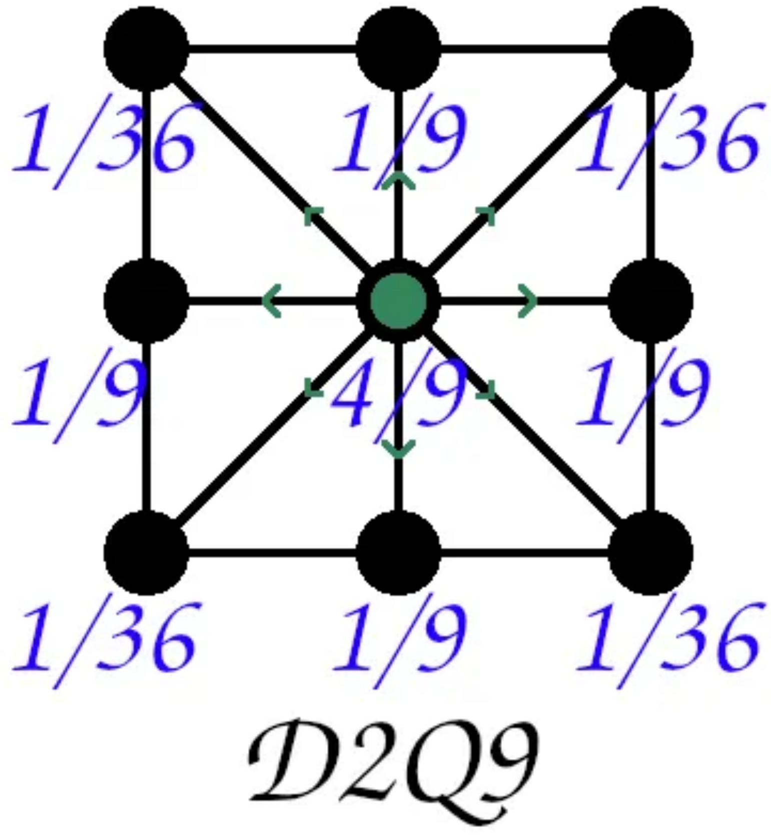
But first, below is a gif of what running our simulation looks like:



Search Medium Write

Fluid dynamics on a Lattice

We will begin with a microscopic description of a fluid that lives on a lattice. For this exercise, we will consider a 2 dimensional lattice with 9 possible velocities at each lattice site (D2Q9). There are 4 connections running North, South, East, and, West, 4 diagonal connections, and 1 connection from a node to itself representing zero velocity. Each lattice site also has a weight w_i associated with it:



The microscopic particles that make up a fluid can be described with the distribution function *f(x,v)*, which describes the the phase-space density of fluids at location ***x*** traveling with velocity ***v***.

The particles will do two things. **Stream** and **collide**. This behavior can be captured by the BGK approximation:

$$\left(\partial_t + \mathbf{v} \cdot \nabla\right) f = -\frac{f - f^{\mathrm{eq}}}{\tau}$$

where the left-hand side represents streaming, and the right-hand side approximates collisions. In this approximation, *τ* is the timescale of which collisions happen, and the distribution function *f* tends towards some equilibrium state *f*^eq as a result.

The equation may be discretized onto the lattice as follows

$$F_i(\mathbf{x}_i + \mathbf{v}_i \Delta t, t + \Delta t) - F_i(\mathbf{x}_i, t) = -\frac{F_i(\mathbf{x}_i, t) - F_i^{\mathrm{eq}}(\mathbf{x}_i, t)}{\tau}$$

where *i* denotes 1 out of the 9 lattice directions (with velocity v_i).

Moments of the discrete distribution function can be taken to recover fluid variables at each lattice site. For example, the **density**:

$$\rho = \sum F_i$$

and **momentum**:

$$\rho\mathbf{u} = \sum F_i \mathbf{v}_i$$

where the sum is over all lattice directions.

It can be shown that this description approximates the Navier-Stokes fluid equations.

Streaming

The first step in the Lattice Boltzmann method is to stream the particles. This step is incredibly simple. Conceptually, here is what happens. At each lattice site, for each direction *i*, the value F_i is shifted over to the neighboring lattice site along the connection.

Typically in the Lattice Boltzmann method uses units of Δt=Δx=1 and we will use this convention throughout. The streaming velocities are hence: (0,0), (0,1), (0,-1), (1,0), (-1,0), (1,1), (1,-1), (-1,1), (-1,-1).

Collisions

Next we need to define the *equilibrium state* as a result of collisions. This depends on the fluid model's equation of state. For this example, we will assume an **isothermal** (constant temperature) fluid, which has a constant sound speed. We define units using common conventions such that the lattice speed is *c*=1 (which corresponds soundspeed^2=1/3). The equilibrium state is given by:

$$F_i^{\mathrm{eq}} = w_i \rho \left(1 + 3(\mathbf{v}_i \cdot \mathbf{u}) + \frac{9}{2}(\mathbf{v}_i \cdot \mathbf{u})^2 + \frac{3}{2}(\mathbf{u} \cdot \mathbf{u}) \right)$$

which corresponds to the isothermal Navier-Stokes equations with a dynamic viscosity:

$$\mu = \rho \left(\tau - \frac{1}{2} \right) \Delta t$$

Boundary

Boundary conditions in Lattice Boltzmann are implemented on the microscopic level. In our simulation, we wish to add a solid cylinder. Lattice sites part of this cylinder may be flagged. Here particles will behave differently. In our example, we will consider **reflective** boundary conditions. Instead of collisions that lead to equilibrium, particles will simply bounce back. This is easily accomplished by swapping lattice directions:

$$F_i \leftrightarrow F_j$$

where *i* and *j* correspond to lattice directions that point in *opposite* directions.

Lattice Boltzmann Method

That's it conceptually. Let's put it all together! The following code sets up the lattice and initial condition for F_i, and alternates streaming and collision(+boundary) operators to evolve the system. It is remarkable that this restricted microscopic representation is able to capture macroscopic fluid behavior.

```python
# Simulation parameters
Nx          = 400    # resolution x-dir
Ny          = 100    # resolution y-dir
rho0        = 100    # average density
tau         = 0.6    # collision timescale
Nt          = 4000   # number of timesteps

# Lattice speeds / weights
NL = 9
idxs = np.arange(NL)
cxs = np.array([0, 0, 1, 1, 1, 0,-1,-1,-1])
cys = np.array([0, 1, 1, 0,-1,-1,-1, 0, 1])
weights = np.array([4/9,1/9,1/36,1/9,1/36,1/9,1/36,1/9,1/36]) # sums to 1
X, Y = np.meshgrid(range(Nx), range(Ny))
```

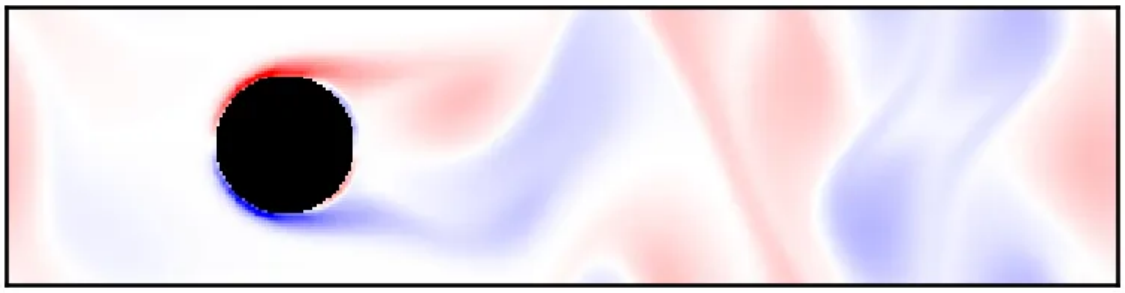
```python

# Initial Conditions - flow to the right with some perturbations
F = np.ones((Ny,Nx,NL)) + 0.01*np.random.randn(Ny,Nx,NL)
F[:,:,3] += 2 * (1+0.2*np.cos(2*np.pi*X/Nx*4))
rho = np.sum(F,2)
for i in idxs:
  F[:,:,i] *= rho0 / rho

# Cylinder boundary
cylinder = (X - Nx/4)**2 + (Y - Ny/2)**2 < (Ny/4)**2


# Simulation Main Loop
for it in range(Nt):

  # Drift
  for i, cx, cy in zip(idxs, cxs, cys):
    F[:,:,i] = np.roll(F[:,:,i], cx, axis=1)
    F[:,:,i] = np.roll(F[:,:,i], cy, axis=0)

  # Set reflective boundaries
  bndryF = F[cylinder,:]
  bndryF = bndryF[:,[0,5,6,7,8,1,2,3,4]]

  # Calculate fluid variables
  rho = np.sum(F,2)
  ux  = np.sum(F*cxs,2) / rho
  uy  = np.sum(F*cys,2) / rho

  # Apply Collision
  Feq = np.zeros(F.shape)
  for i, cx, cy, w in zip(idxs, cxs, cys, weights):
    Feq[:,:,i] = rho*w* (1 + 3*(cx*ux+cy*uy) + 9*(cx*ux+cy*uy)**2/2 - 3*(ux**2+uy**2

  F += -(1.0/tau) * (F - Feq)

  # Apply boundary
  F[cylinder,:] = bndryF
```

latticeboltzmann.py hosted with ❤ by GitHub view raw latticeboltzmann.py hosted with ❤ by GitHub view raw

Flow Past Cylinder

The initial conditions above place a static cylinder into a periodic box with rightward moving fluid. As the flow progresses, turbulence develops in the wake behind the cylinder. This is known as the Kármán vortex street.

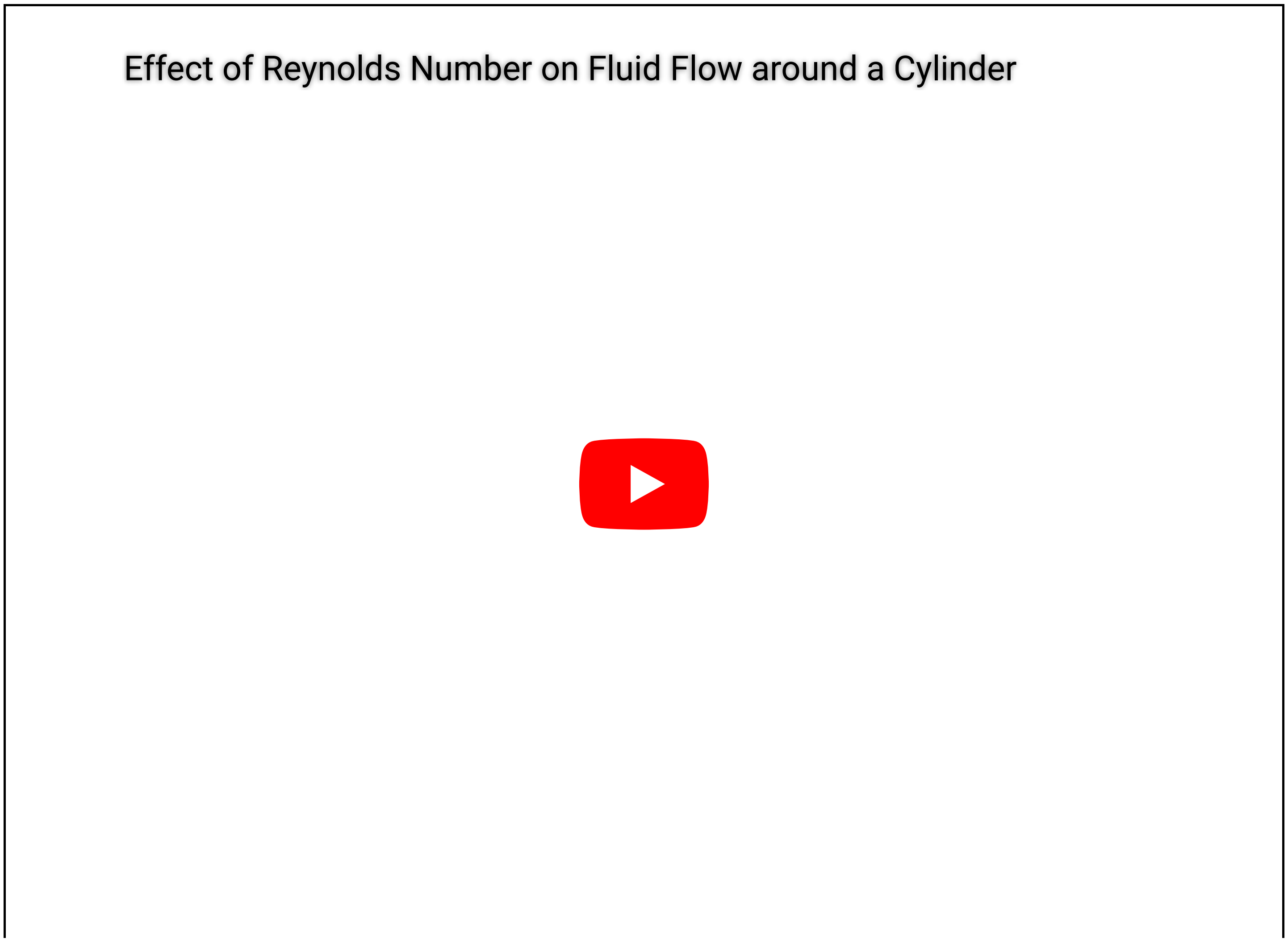
Running the code allows you to visualize the vorticity **ω=∇×v** of the flow past the cylinder in real time and will yield the figure:



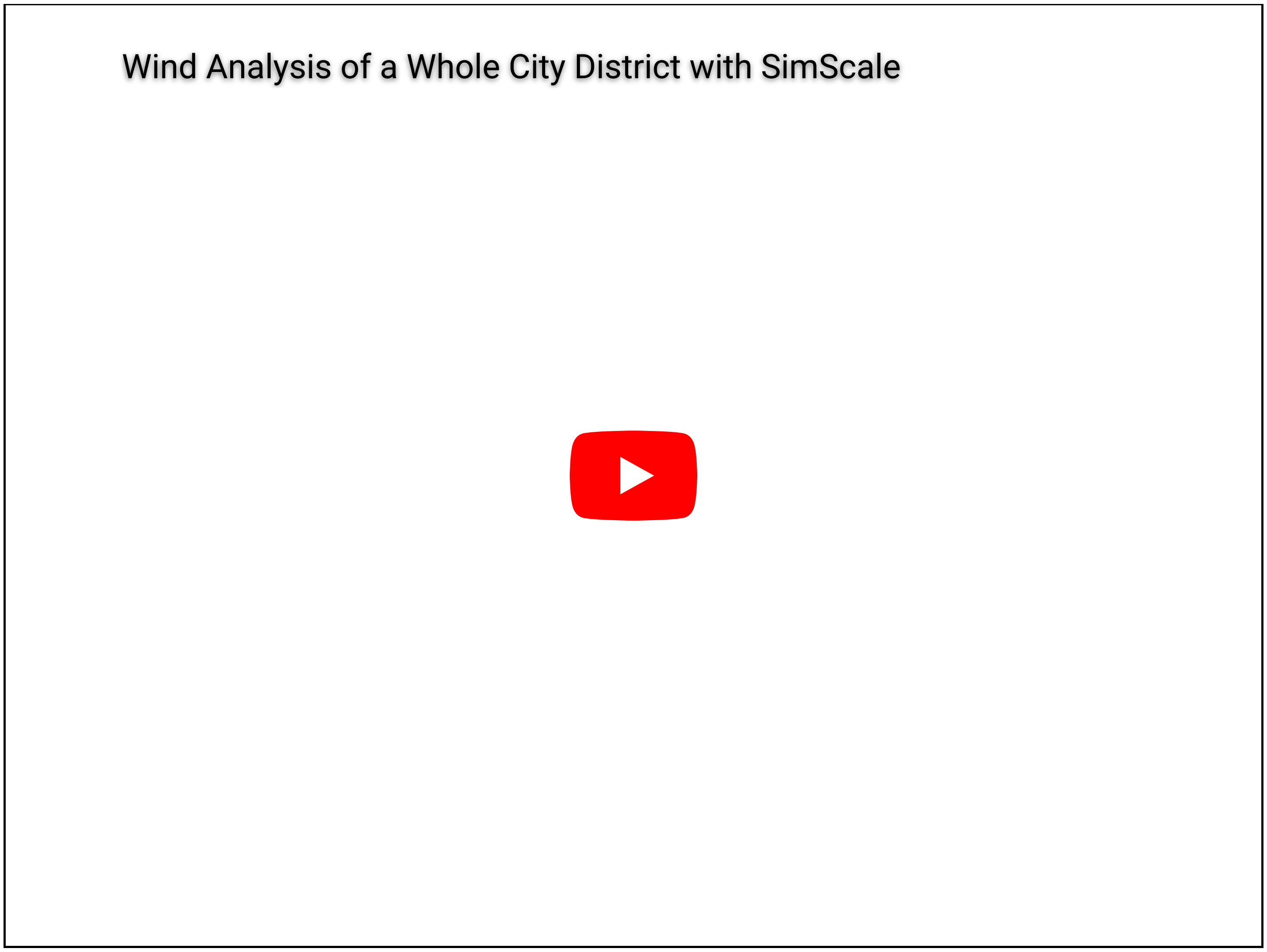
The Lattice Boltzmann method has many applications and extensions for complex multi-fluid flows with complicated boundaries.

The main limitation of the original Lattice Boltzmann method is that this representation is not great for capturing highly-compressible/supersonic flows. Such is often the case in astrophysical systems, hence the usage of the method in computational astrophysics remains rare. But there are many other applications.

Below is a Lattice Boltzman nnumerical experiment of the flow past cylinder problem where the viscosity is varied. It is seen that the wake behind the cylinder becomes turbulent once the viscosity is low enough (i.e., once the Reynolds number Re is large enough).



To give another more complicated example, below is a study of wind flow past buildings in a city. Lattice Boltzmann makes it quite easy to add buildings/modify boundary conditions because all one needs to do is flag a lattice site as part of the building. The simulation identifies locations where buildings cause high wind velocities.



I hope this tutorial demonstrated the simplicity and power of the Lattice

Boltzmann approach to fluid dynamics, and gave a reminder of the microscopic underpinnings of fluid behavior.

Download the Python code on github for our Lattice Boltzmann tutorial to visualize the flow past a cylinder in real time and play around with the setup. Enjoy!

Simulation Fluid Mechanics Lattice Python Physics