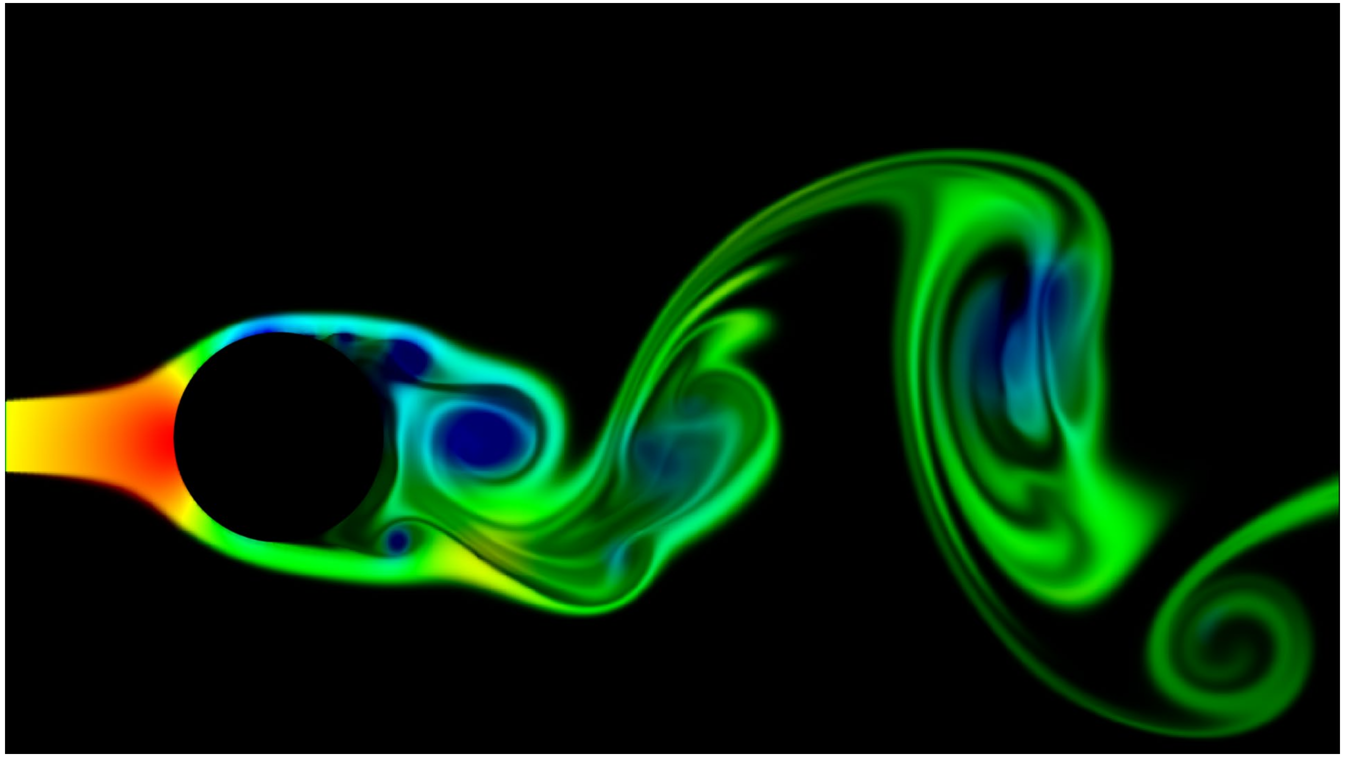


How to write an Eulerian Fluid Simulator

with 200 lines of JavaScript code

Matthias Müller, Ten Minute Physics

For the code and the demo see:

www.matthiasmueller.info/tenMinutePhysics

a few remarks…

Fluid =

or

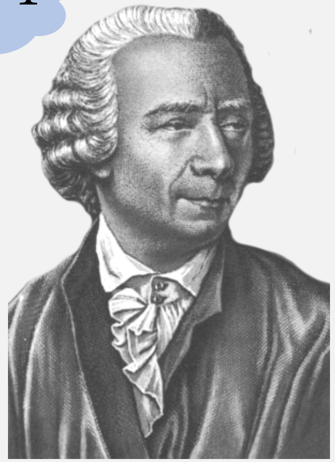
liquid

gas

Similar physical structures → same simulation method

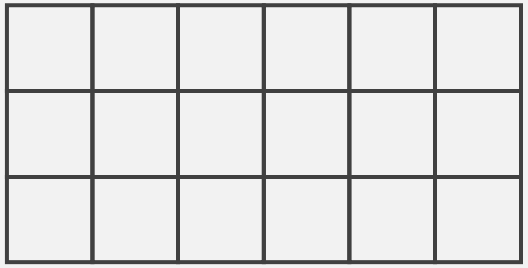
Eulerian

$e^{i\pi} = -1$

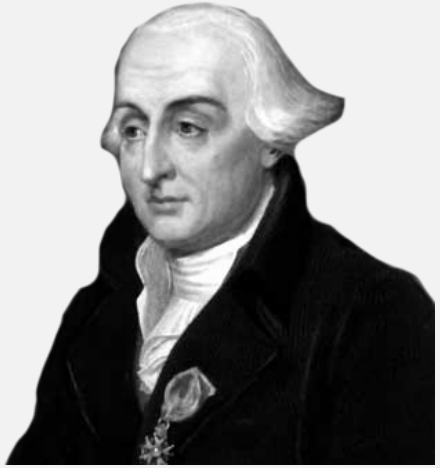


Swiss

Leonhard Euler (1707-1783)

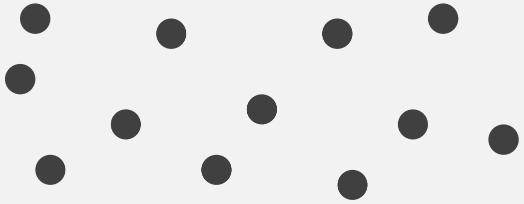


grid-based



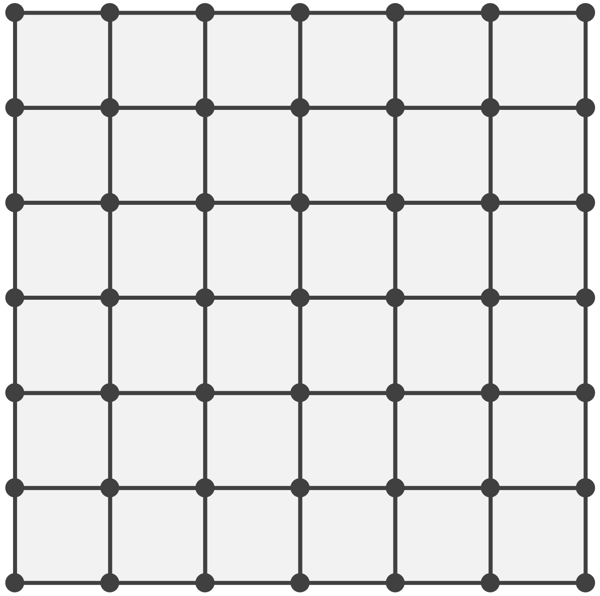
Italian

Joseph-Louis Lagrange (1736-1813)

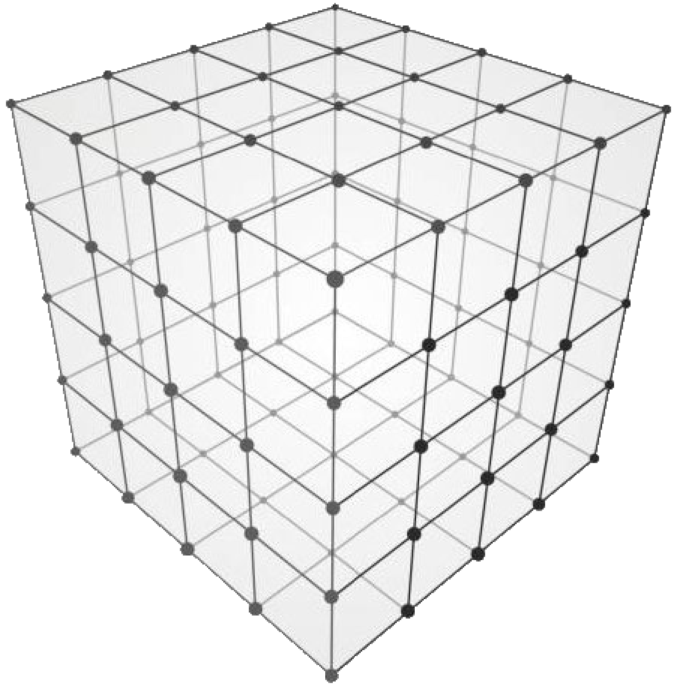


grid-free (e. g. particles)

From 2d to 3d



straight forward - left as an exercise

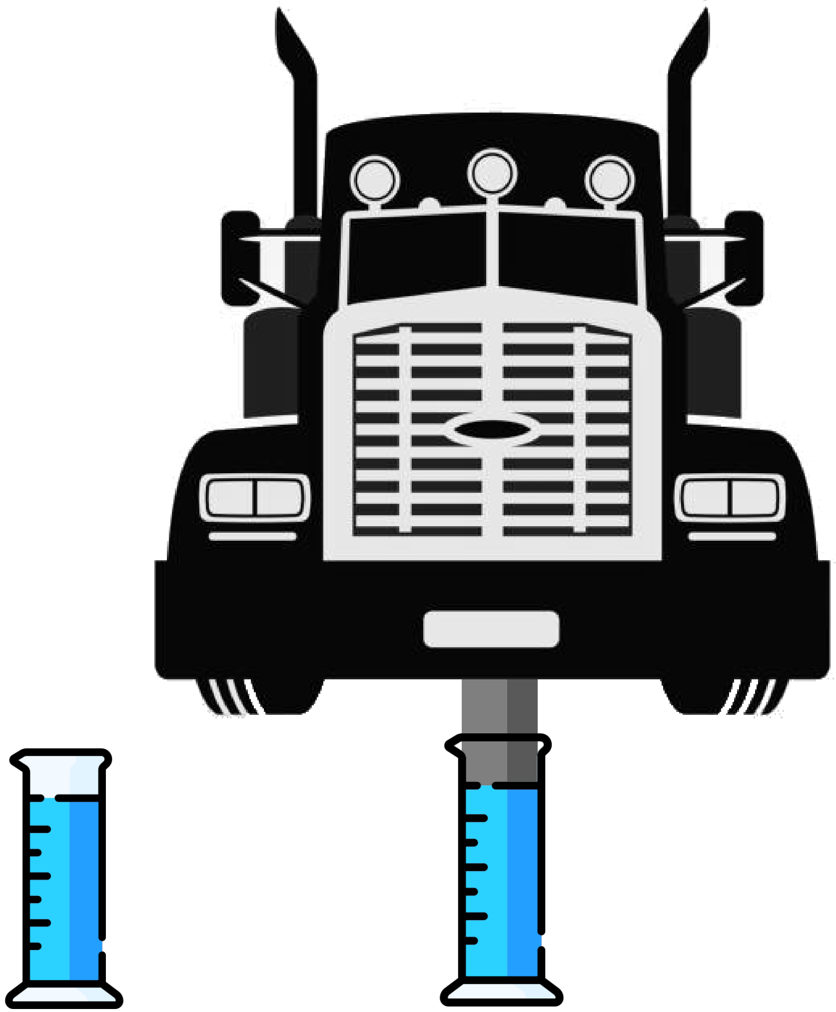


Incompressible Fluid

- We assume incompressibility
- Water is very close to being incompressible

  10,000 kg / cm^2 → 3% compression!

- Reasonable assumption for free gas as well



→ compressible simulation in upcoming tutorial

Inviscid Fluid

- We assume zero viscosity
- Good approximation for gas and water

→ viscous simulation in upcoming tutorial

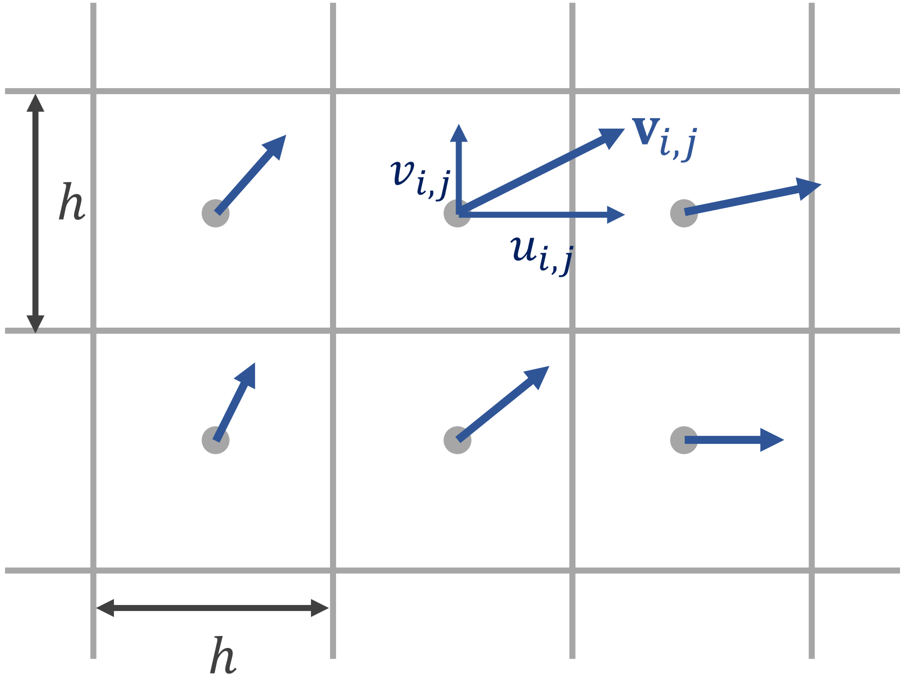
inviscid

viscous

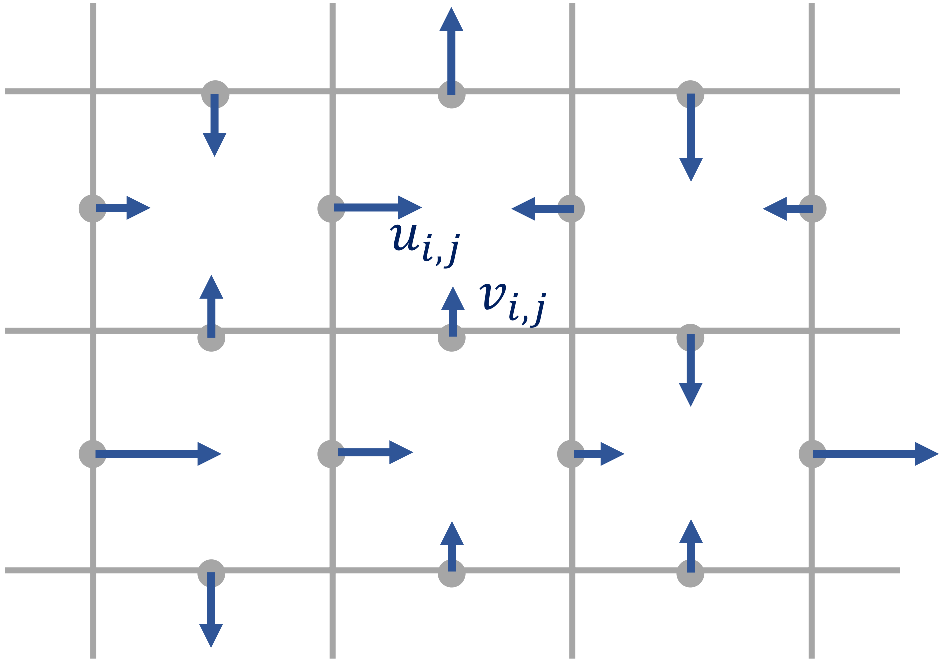
the method…

Fluid as a Velocity Field on a Grid

- Velocity is a 2d vector $\mathbf{v} = \begin{bmatrix} u \\ v \end{bmatrix}$ (vectors are written in boldface)

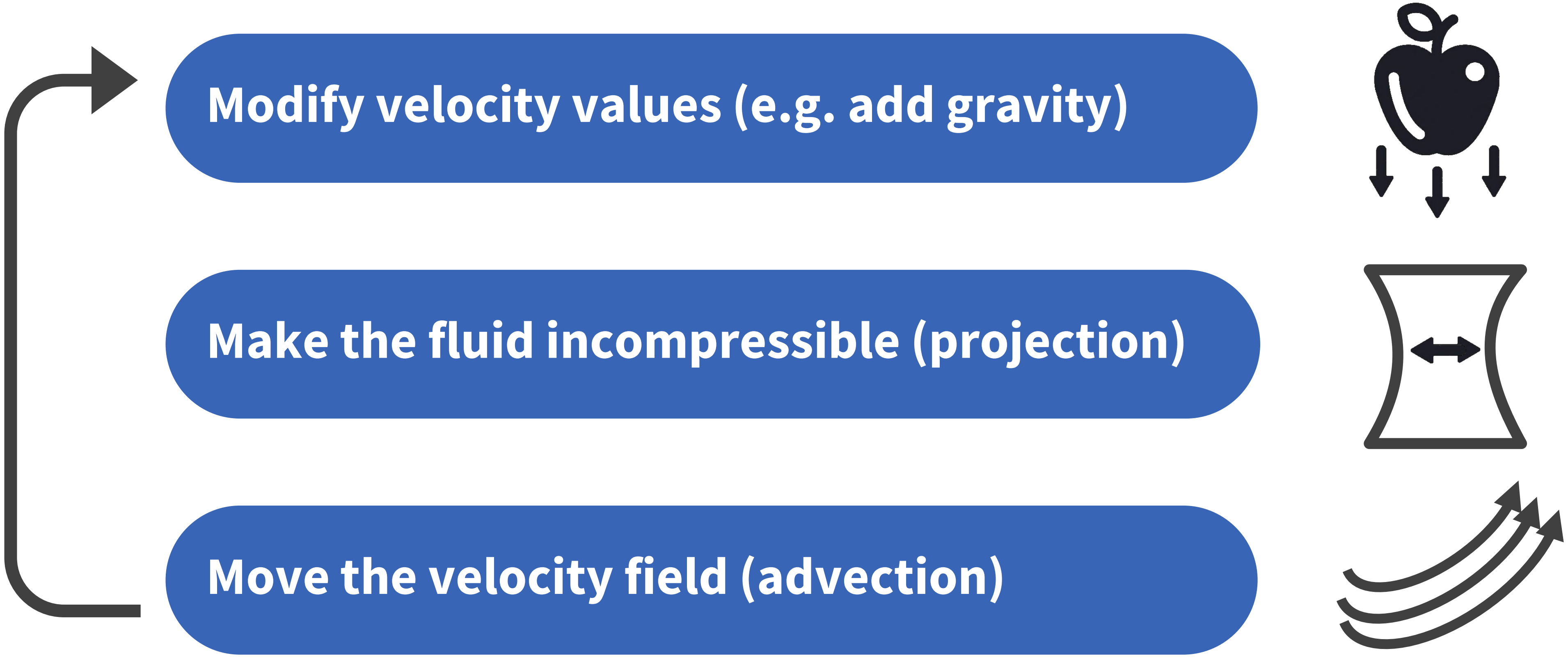


collocated grid



staggered grid

Fluid Simulation



Update Velocity

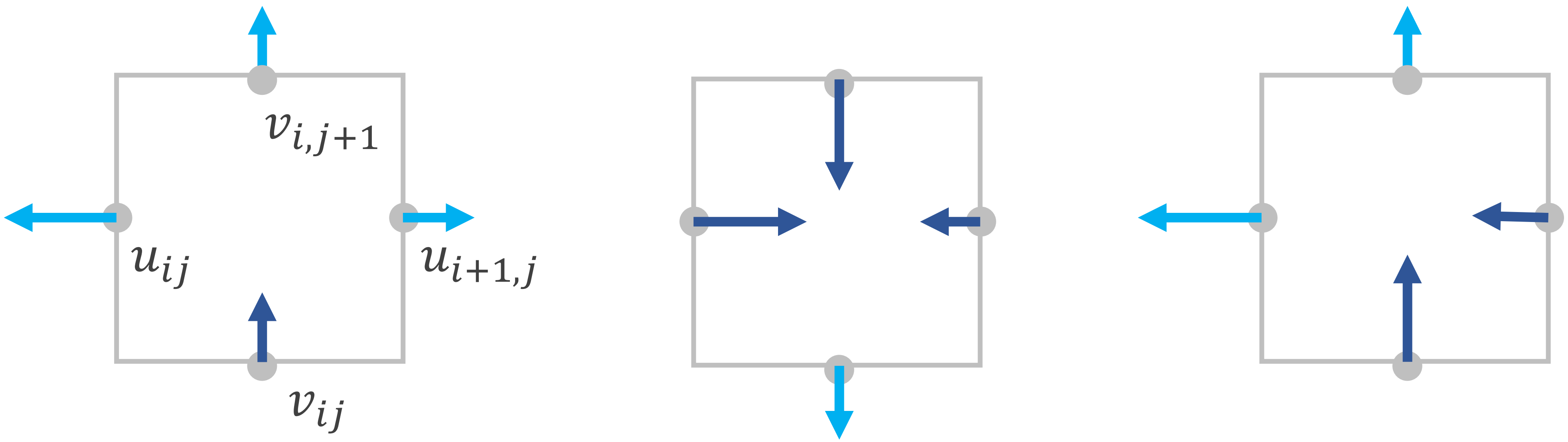
for all i, j

$$v_{i,j} \leftarrow v_{i,j} + \Delta t \cdot g$$

- Gravity g: $-9.81\ m/s^2$
- Timestep Δt: (e. g. $\frac{1}{30} s$)

Divergence（Total Outflow)

$$d \leftarrow u_{i+1,j} - u_{i,j} + v_{i,j+1} - v_{i,j}$$

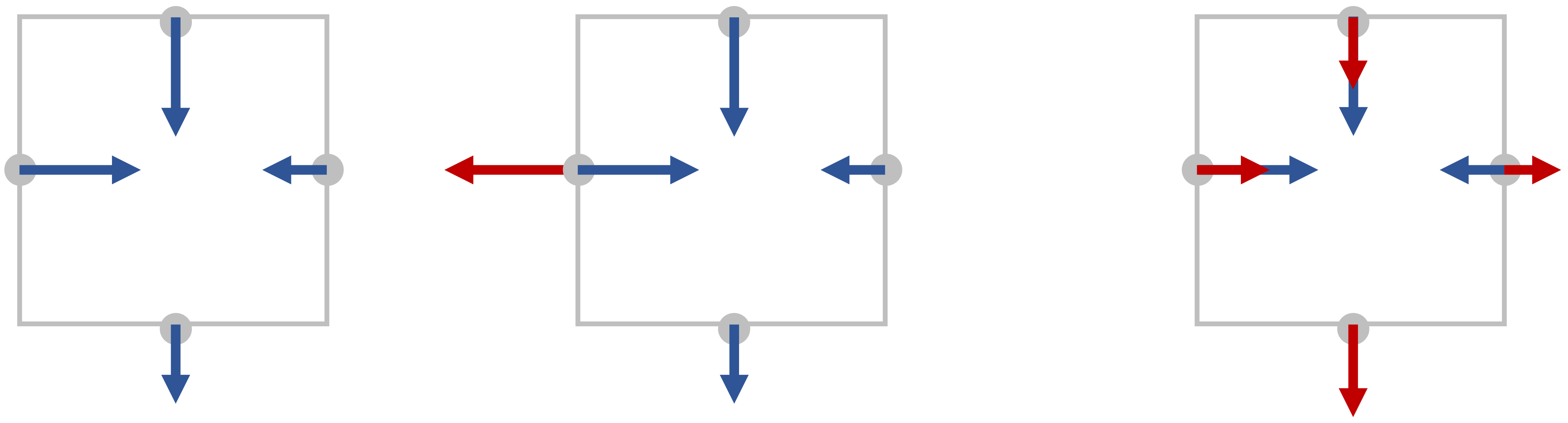


- Positive
- Too much outflow

- Negative
- Too much inflow

- Zero
- Incompressible

Forcing Incompressibility

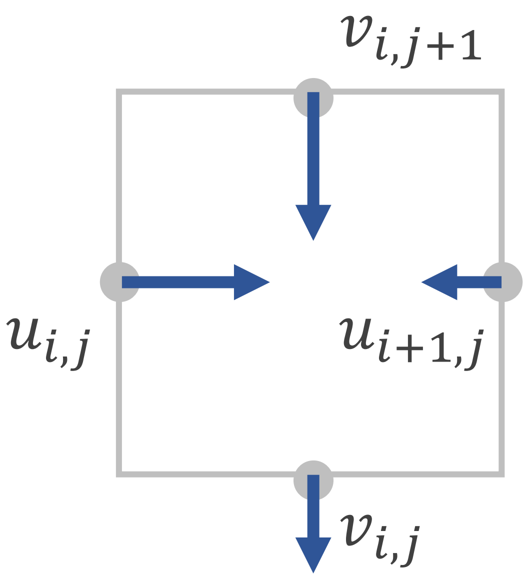


Too much inflow

Changing one velocity
A fluid cannot do that! ❌

Push all velocities outward
by the same amount ✅

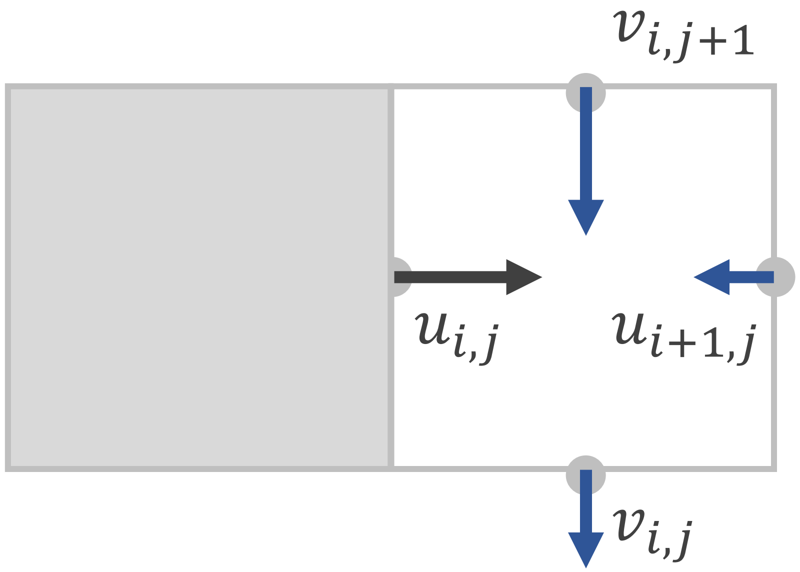
Forcing Incompressibility



$$d \leftarrow u_{i+1,j} - u_{i,j} + v_{i,j+1} - v_{i,j}$$

$$\begin{aligned}
u_{i,j} &\leftarrow u_{i,j} + d/4 \\
u_{i+1,j} &\leftarrow u_{i+1,j} - d/4 \\
v_{i,j} &\leftarrow v_{i,j} + d/4 \\
v_{i,j+1} &\leftarrow v_{i,j+1} - d/4
\end{aligned}$$

Obstacles / Walls



$$d \leftarrow u_{i+1,j} - u_{i,j} + v_{i,j+1} - v_{i,j}$$

$$\cancel{u_{i,j} \leftarrow u_{i,j} + d/4}$$

$$u_{i+1,j} \leftarrow u_{i+1,j} - d/3$$

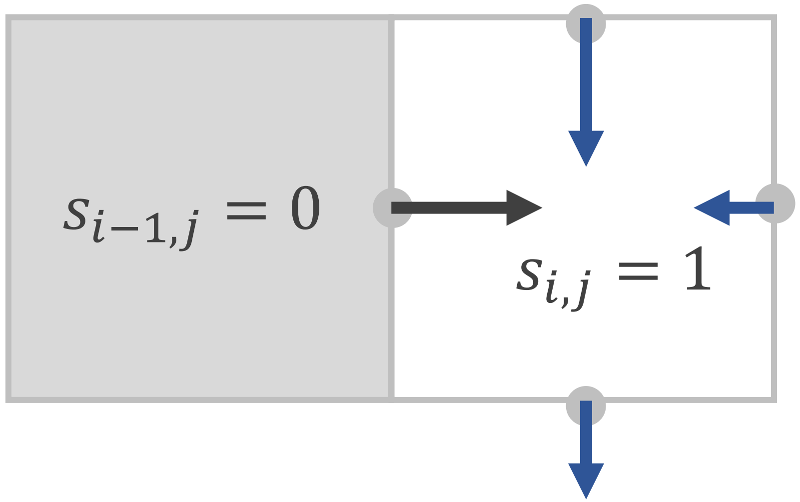
$$v_{i,j} \leftarrow v_{i,j} + d/3$$

$$v_{i,j+1} \leftarrow v_{i,j+1} - d/3$$

$u_{i,j} \neq 0$

- Moving object
- Turbine pushing fluid (e. g. wind tunnel)

General Case



$$d \leftarrow u_{i+1,j} - u_{i,j} + v_{i,j+1} - v_{i,j}$$

$$s \leftarrow s_{i+1,j} + s_{i-1,j} + s_{i,j+1} + s_{i,j-1}$$

$$\begin{aligned}
u_{i,j} &\leftarrow u_{i,j} + d\, s_{i-1,j}/s \\
u_{i+1,j} &\leftarrow u_{i+1,j} - d\, s_{i+1,j}/s \\
v_{i,j} &\leftarrow v_{i,j} + d\, s_{i,j-1}/s \\
v_{i,j+1} &\leftarrow v_{i,j+1} - d\, s_{i,j+1}/s
\end{aligned}$$

Solving the Grid

for n iterations

for all i, j

$$d \leftarrow u_{i+1,j} - u_{i,j} + v_{i,j+1} - v_{i,j}$$

$$s \leftarrow s_{i+1,j} + s_{i-1,j} + s_{i,j+1} + s_{i,j-1}$$

$$\begin{aligned}
u_{i,j} &\leftarrow u_{i,j} + d\, s_{i-1,j}/s \\
u_{i+1,j} &\leftarrow u_{i+1,j} - d\, s_{i+1,j}/s \\
v_{i,j} &\leftarrow v_{i,j} + d\, s_{i,j-1}/s \\
v_{i,j+1} &\leftarrow v_{i,j+1} - d\, s_{i,j+1}/s
\end{aligned}$$

- Gauss-Seidel method
- On the boundary we access cells outside of the grid!
- Add border cells
- Set $s_{i,j} = 0$ for walls
- or copy neighbor values

Measuring Pressure

for all i, j

$p_{i,j} \leftarrow 0$

for n iterations

for all i, j

$d \leftarrow u_{i+1,j} - u_{i,j} + v_{i,j+1} - v_{i,j}$

$s \leftarrow s_{i+1,j} + s_{i-1,j} + s_{i,j+1} + s_{i,j-1}$

...

$p_{i,j} \leftarrow p_{i,j} + \frac{d}{s} \cdot \frac{\rho\, h}{\Delta t}$

$p_{i,j}$ is physical pressure

ρ density

h grid spacing

Δt time step

Not necessary for simulation

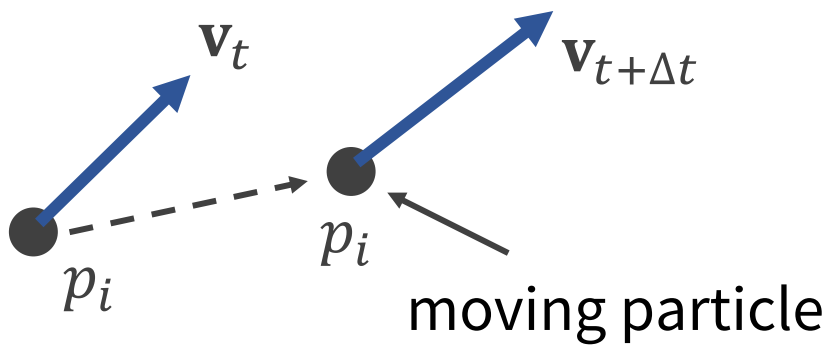
Overrelaxation

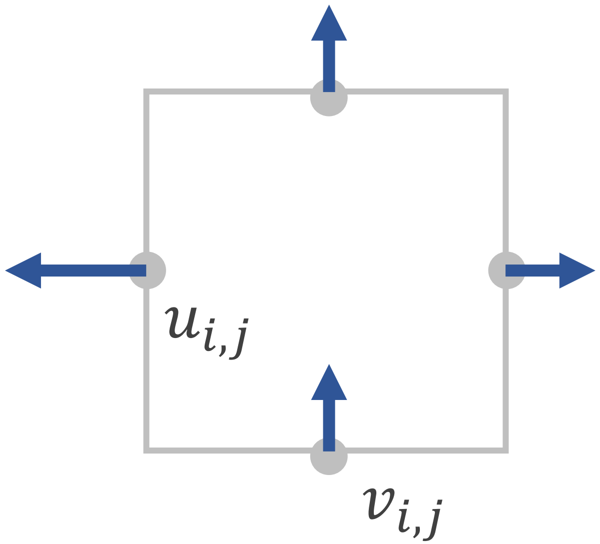
- Gauss-Seidel is very simple to implement
- Needs more iterations than global methods
- Acceleration: Overrelaxation – Magic!!

$$d \leftarrow \boldsymbol{o}\left(u_{i+1,j} - u_{i,j} + v_{i,j+1} - v_{i,j}\right)$$

- Multiply the divergence by a scalar $1 < o < 2$
- I use $o = 1.9$ in the code
- Increases convergence dramatically
- The computed pressure value is still correct!

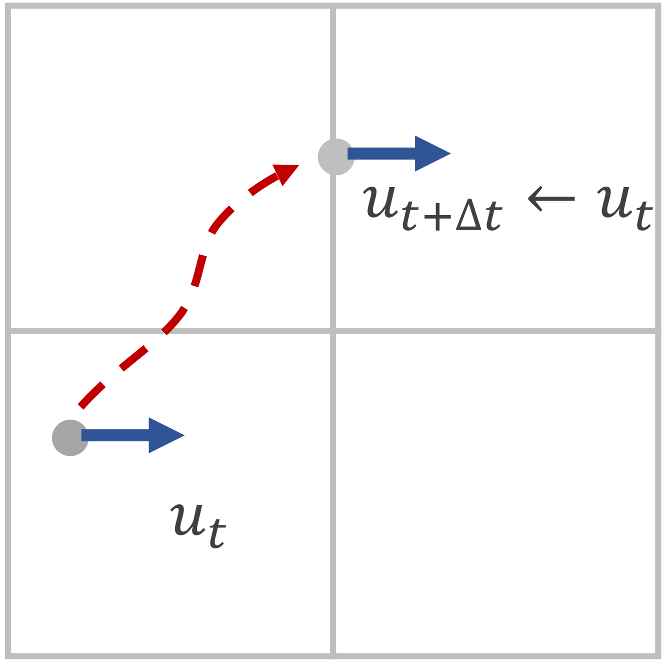
Last step: Advection





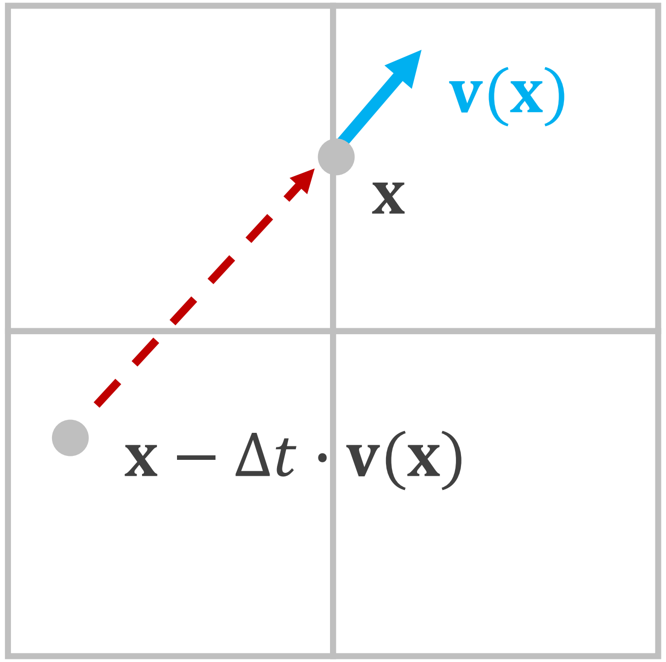
- In a fluid the velocity state is carried by the particles
- Particles move, grid cells are static
- We need to move the velocity values in the grid!

Semi-Lagrangian Advection



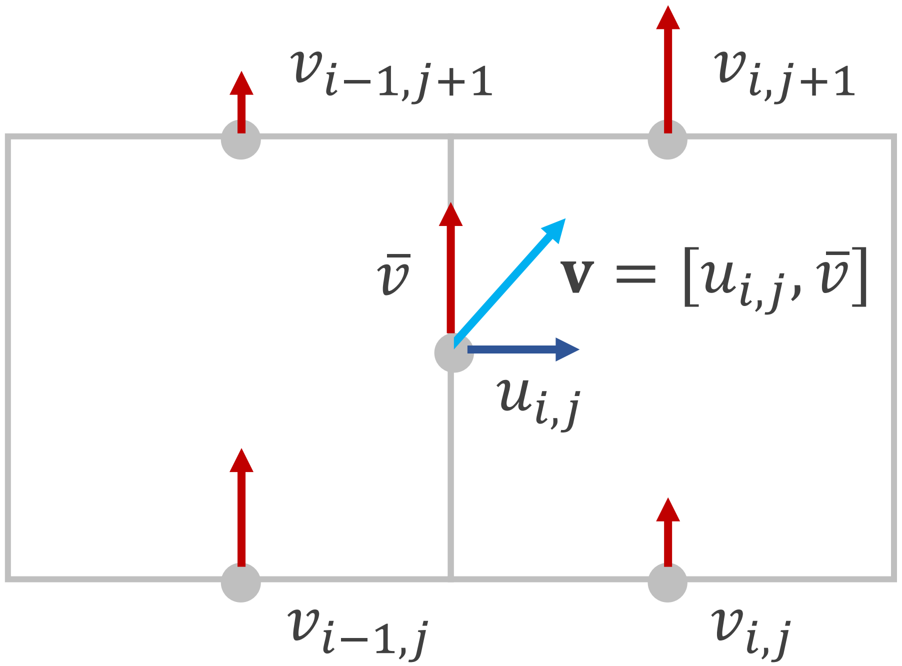
- Which fluid "particle" moved to the location where u is stored?
- Set the new velocity $u_{t+\Delta t}$ to the velocity u_t at the previous position

Semi-Lagrangian Advection



- Compute $\mathbf{v}$ at position $\mathbf{x}$ where u is stored
- The previous location can be approximated as $\mathbf{x} - \Delta t \cdot \mathbf{v}(\mathbf{x})$
- Assuming a straight path introduces viscosity!
  Can be reduced with vorticity confinement

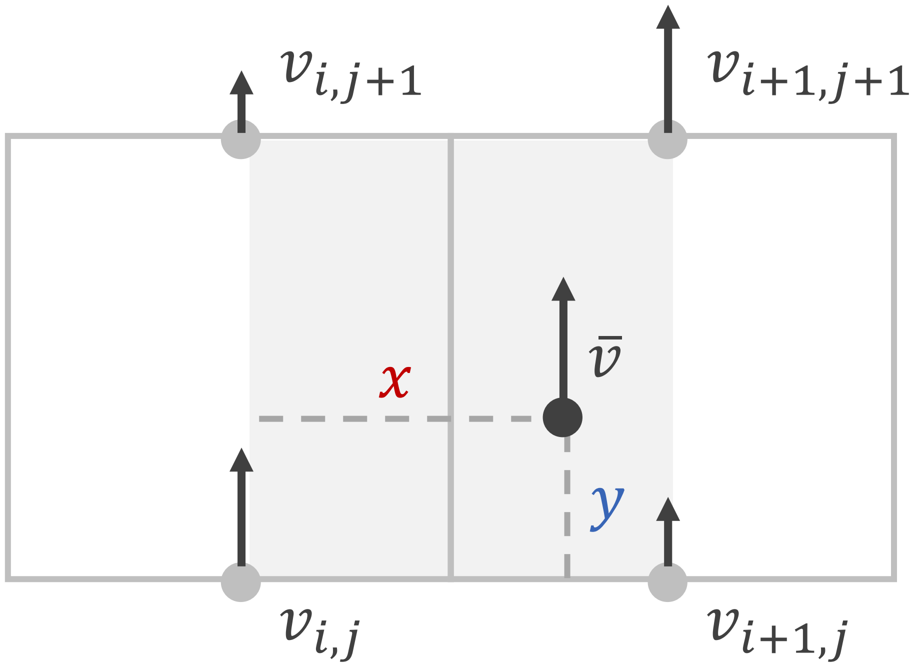
Get the 2d Velocity



- Average surrounding horizontal velocities

$$\bar{v} = (v_{i,j} + v_{i,j+1} + v_{i-1,j} + v_{i-1,j+1})/4$$

General Grid Interpolation

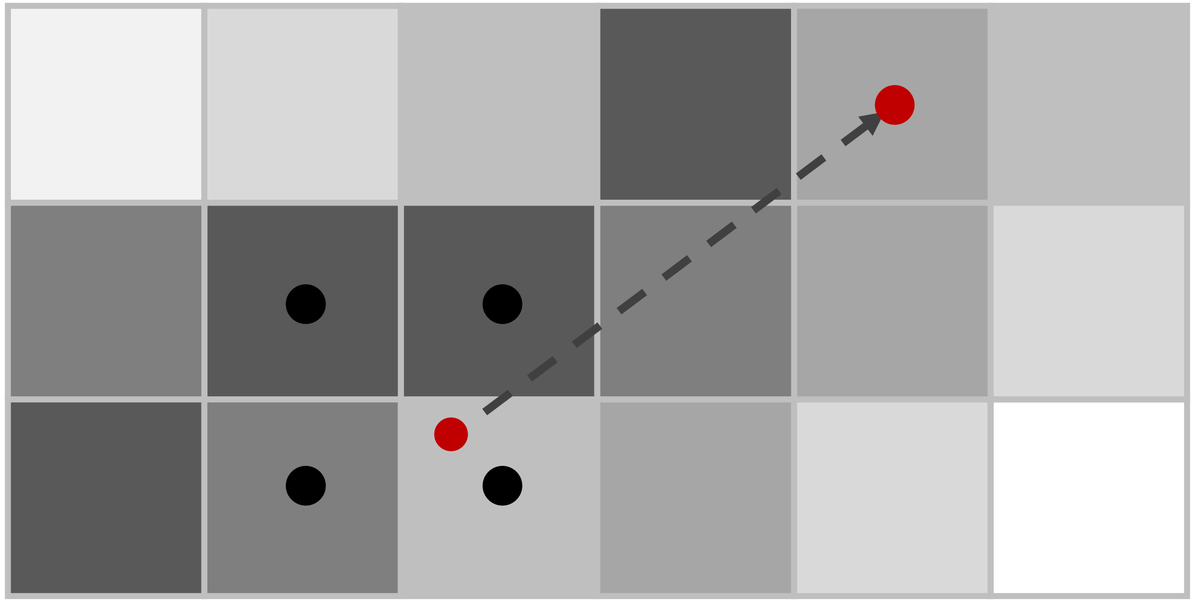


$$w_{00} = 1 - x/h \qquad w_{01} = x/h$$

$$w_{10} = 1 - y/h \qquad w_{11} = y/h$$

$$\bar{v} = w_{00} w_{10} v_{i,j} + w_{01} w_{10} v_{i+1,j} + w_{010} w_{11} v_{i,j+1} + w_{011} w_{11} v_{i+1,j+1}$$

Smoke Advection



- Store density value at the center of each cell
- Advect it like the velocity components

Streamlines

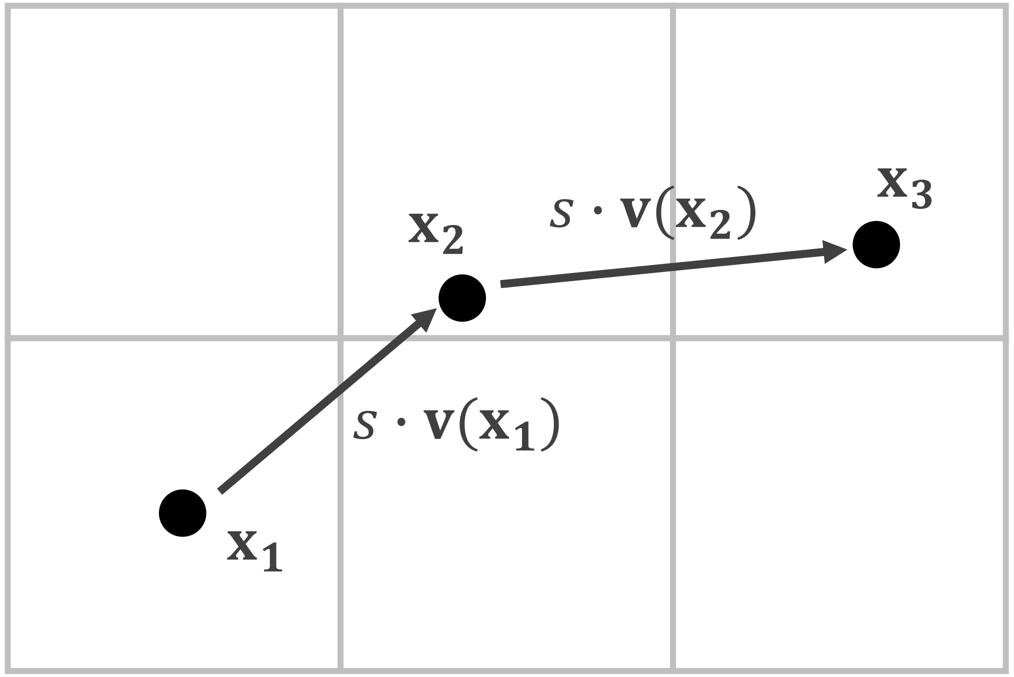
$\mathbf{x} \leftarrow$ start position

$s \leftarrow$ step size

for n steps

$\mathbf{v} \leftarrow$ sampleV$(\mathbf{x})$

$\mathbf{x} \leftarrow \mathbf{x} + s\mathbf{v}$



Let’s look into the code…