Introduction to Computer Graphics

– Animation (3) –

May 27, 2021

Kenshi Takayama

Fluid simulation

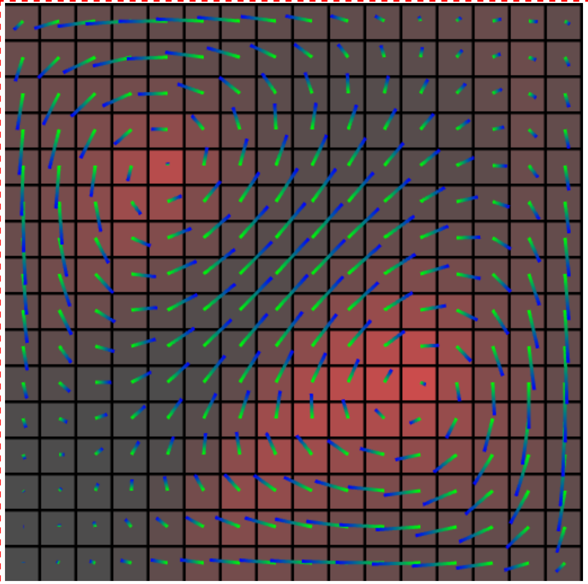
https://www.youtube.com/watch?v=KoEbwZq2ErU

https://www.youtube.com/watch?v=6WZZARzpckw

https://www.youtube.com/watch?v=WFwi0qLV8hQ

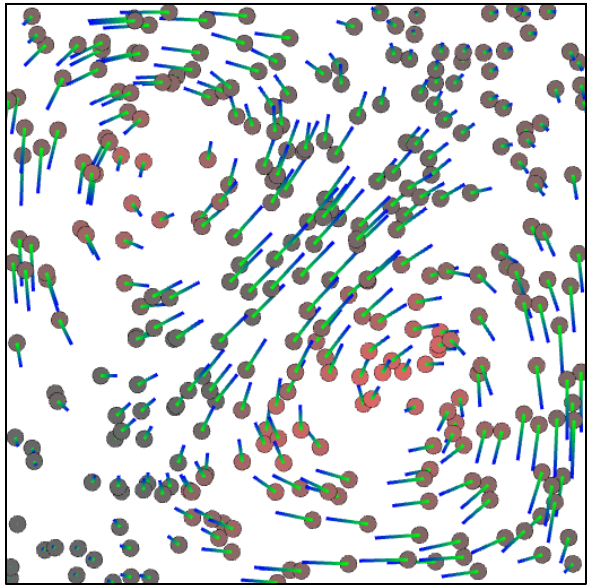
Position Based Fluids [Macklin SIGGRAPH13]
Detail-Preserving Paint Modeling for 3D Brushes [Chu NPAR10]

Two different approaches



Eulerian

- Store velocity etc. on lattice grid
  - e.g. density, temperature
- Straightforward to compute gradients
- Suitable for offline applications
  - Also good for real-time



Lagrangian

- Store data on particles which move according to velocity
- Needs some hacks for computing gradients etc
- Suitable for real-time applications

Stable Fluids [Stam, SIGGRAPH 99]

- Unconditionally stable regardless of timestep
  ➔ suitable for games

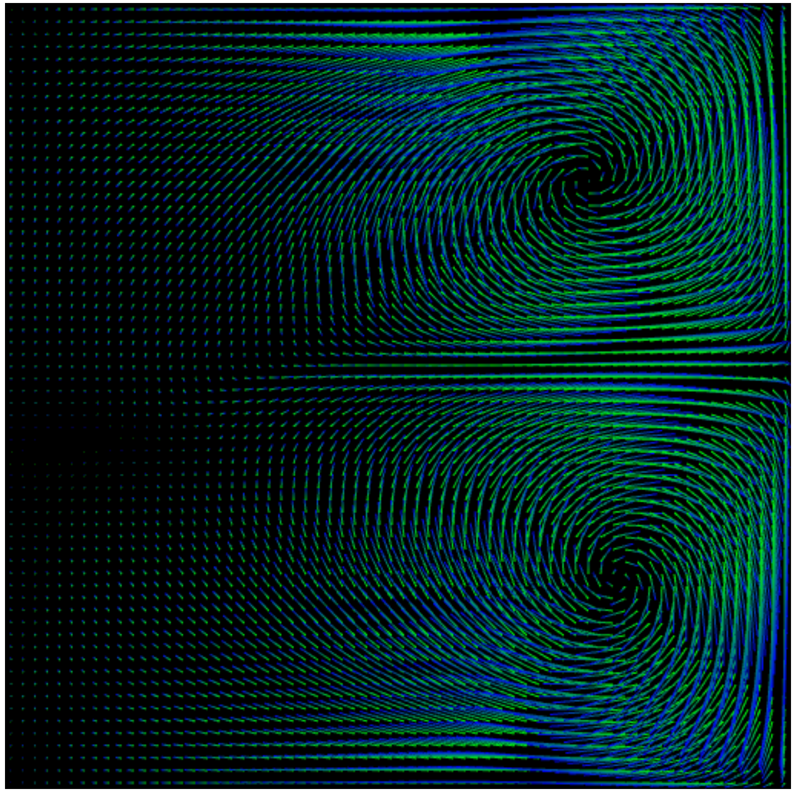
- Easy exposition for game developers
  - Real-Time Fluid Dynamics for Games (GDC 2003)
    - https://www.dgp.toronto.edu/public_user/stam/reality/Research/pdf/GDC03.pdf
  - Small sample code (< 500 lines)
    - http://www.dgp.toronto.edu/people/stam/reality/Research/zip/CDROM_GDC03.zip

- Goal: Understand this paper!

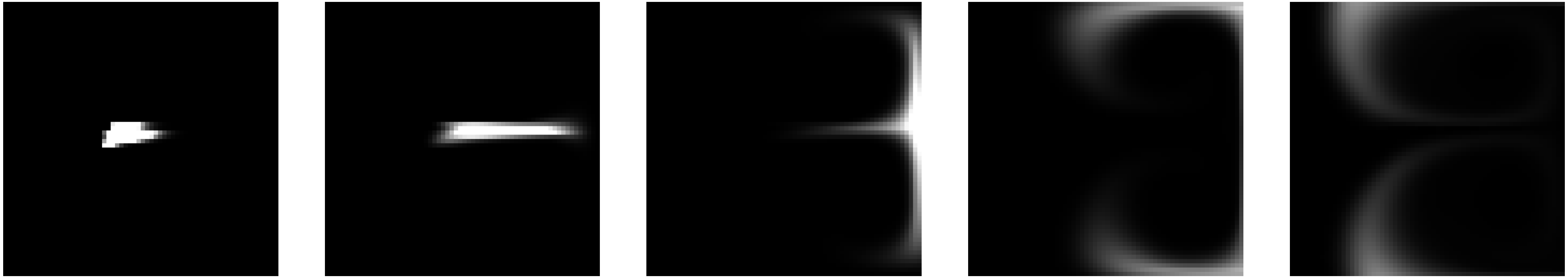
Advection of physical quantity by stationary velocity field

- Type of quantity:
  temperature, density, etc

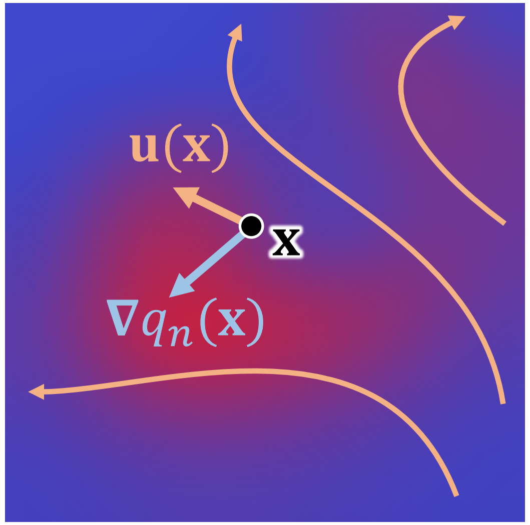
- Method:
  - Explicit method ➔ unstable
  - Semi-Lagrangian method ➔ stable



Velocity field



Explicit method [Foster 96]



- Given: stationary velocity field over 2D domain

$$\mathbf{u}: \mathbb{R}^2 \mapsto \mathbb{R}^2$$

- For some quanitity $q_n: \mathbb{R}^2 \mapsto \mathbb{R}$ at time t, compute its next state q_{n+1} at time $t + h$ using explicit method:

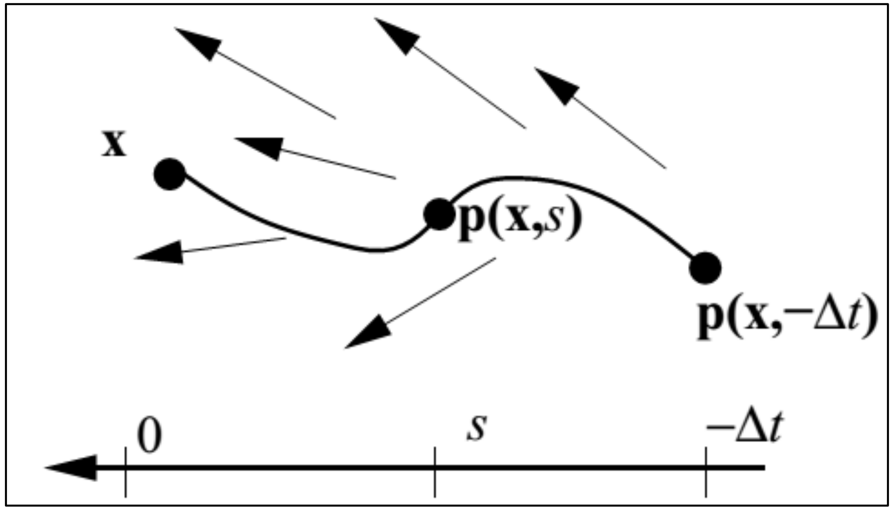
$$q_{n+1}(\mathbf{x}) = q_n(\mathbf{x}) - h\, \mathbf{u}(\mathbf{x}) \cdot \nabla q_n(\mathbf{x})$$

Careful with the sign!

- Amount of change is in proportion to timestep h
  ➔ Too large h will lead to explosion ☹

Realistic animation of liquids [Foster GMIP96]

Semi-Lagrangian method [Stam 99]



- *Imagine* a particle that would arrive at position $\mathbf{x}$ at time $t + h$
- Obtain the particle's position $\tilde{\mathbf{x}}$ at time t, and copy the current quantity sampled there:

$$\tilde{\mathbf{x}} = \mathrm{trace}(\mathbf{u}, \mathbf{x}, -h)$$
$$q_{n+1}(\mathbf{x}) = q_n(\tilde{\mathbf{x}})$$

  - No need to actually generate particles
  - Method for tracing: linear prediction, Runge-Kutta, etc

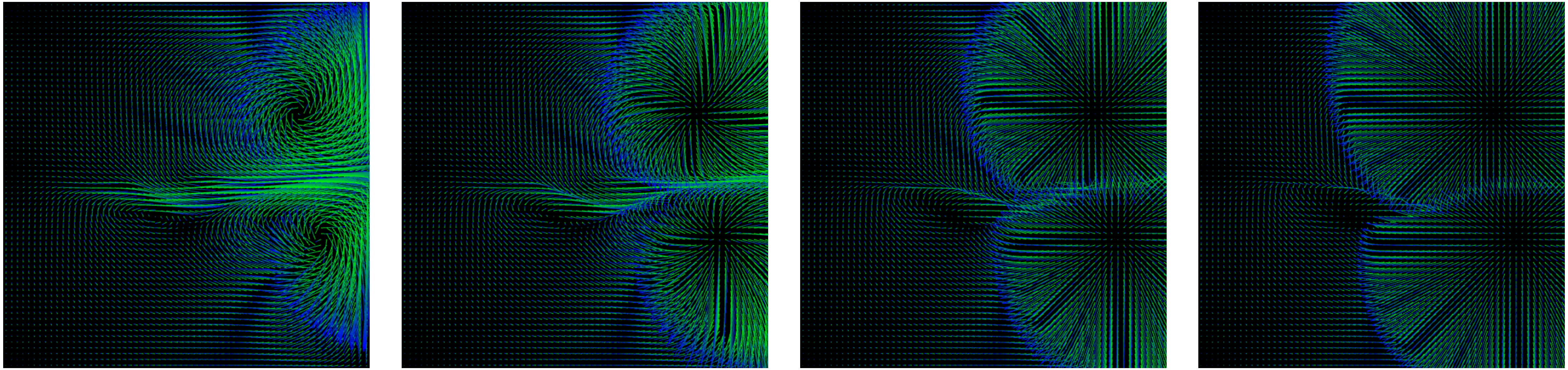
- Stable regardless of timestep!
  - Because we obtain q_{n+1} by *resampling* q_n

Dynamic change of velocity field

- Advect velocity itself by velocity field (using semi-Lagrangian), just like any other quantities

- BUT, the result is not realistic at all!

There should be more swirls!



Key to realism: incompressibility condition

$$\nabla \cdot \mathbf{u}(\mathbf{x}) = 0 \qquad \forall \mathbf{x}$$

- "Divergence is zero everywhere"
  - For each cell, the sum of incoming & outgoing amount is zero
  - Can be interpreted as mass conservation

- Vector field $\mathbf{w}$ obtained by advecting velocity doesn't satisfy the incompressibility condition!

- So, we compute a scalar field q such that

$$\nabla \cdot (\mathbf{w} - \nabla q) = 0$$

Helmholtz decomposition

and obtain new velocity field $\mathbf{u} = \mathbf{w} - \nabla q$ satisfying the incomp. condition

Poisson equation

$$\boldsymbol{\nabla} \cdot (\mathbf{w} - \boldsymbol{\nabla} q) = 0$$
$$\Leftrightarrow$$
$$\Delta\, q = \boldsymbol{\nabla} \cdot \mathbf{w}$$

$\Delta = \boldsymbol{\nabla} \cdot \boldsymbol{\nabla}$

- q minimizes the following energy (thus called projection)

$$E(q) = \int_{\Omega} \|\mathbf{w} - \boldsymbol{\nabla} q\|^2$$

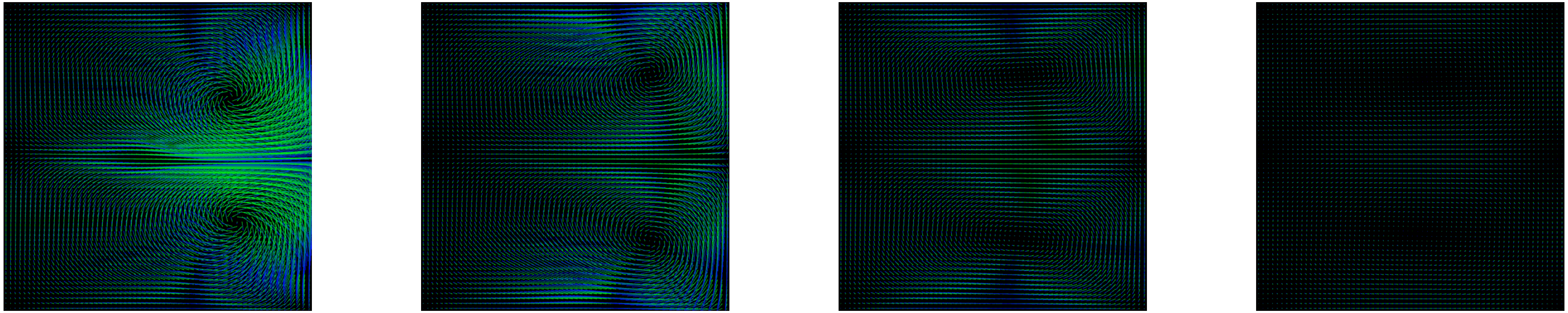
- Equation expressed using a large sparse matrix

- Solution methods:
  - Gauss-Seidel ➔ easy to implement, but slow (used by the sample code)
  - (Preconditioned) Conjugate Gradient ➔ fast
  - Multigrid ➔ very fast, difficult to implement (maybe?)

Diffusion

- Phenomenon of distribution tending toward smoother state

- When applied to velocity, we get viscous fluid



Diffusion equation

$$\frac{\partial q}{\partial t} = \nu\, \Delta q$$

ν : coefficient

- Explicit method

$$q_{n+1}(\mathbf{x}) = q_n(\mathbf{x}) + h\,\nu\,\Delta q_n(\mathbf{x})$$

  - Amount of change is in proportion to timestep h ➔ unstable

- Implicit method

$$q_n(\mathbf{x}) = q_{n+1}(\mathbf{x}) - h\,\nu\,\Delta q_{n+1}(\mathbf{x})$$

  - Stable regardless of timestep h
  - Equation expressed using a large sparse matrix (similar to Poisson eq.)

Incompressible Navier-Stokes equation

$$\frac{\partial \mathbf{u}}{\partial t} = \underbrace{-(\mathbf{u} \cdot \nabla)\mathbf{u}}_{\text{Advection}} \underbrace{- \frac{1}{\rho}\nabla p}_{\text{Pressure}} + \underbrace{\nu \Delta \mathbf{u}}_{\text{Viscosity}} + \underbrace{\mathbf{f}}_{\text{Ext. force}} \qquad \text{s.t.} \quad \nabla \cdot \mathbf{u} = 0$$

- Not to be confused:

  $(\mathbf{u} \cdot \nabla) = \left(u_{\mathrm{x}} \frac{\partial}{\partial x} + u_{\mathrm{y}} \frac{\partial}{\partial y}\right)$ is a *differential operator*, unlike divergence $\nabla \cdot \mathbf{u}$!

- x component:

$$\frac{\partial u_{\mathrm{x}}}{\partial t} = -\left(u_{\mathrm{x}} \frac{\partial u_{\mathrm{x}}}{\partial x} + u_{\mathrm{y}} \frac{\partial u_{\mathrm{x}}}{\partial y}\right) - \frac{1}{\rho} \frac{\partial p}{\partial x} + \nu \left(\frac{\partial^2 u_{\mathrm{x}}}{\partial x^2} + \frac{\partial^2 u_{\mathrm{x}}}{\partial y^2}\right) + f_{\mathrm{x}}$$

- Scalar field $q(\mathbf{x}) = p(\mathbf{x})/\rho$ obtained by projection corresponds to pressure
  - Acceleration emerging from high pressure region to low pressure region

Simulation pipeline

- Velocity update (vel_step)
  - Add external force
  - Diffuse
  - Project
  - Advect
  - Project

Equation for velocity $\mathbf{u}(\mathbf{x})$

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu \Delta \mathbf{u} + \mathbf{f}$$

Do projection *before* advection & difffusion!

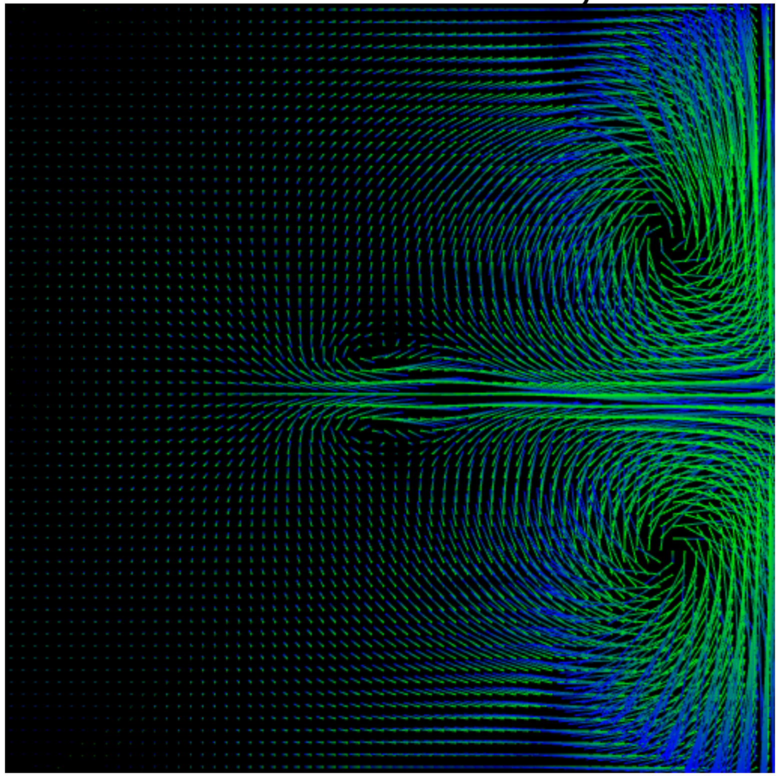
- Smoke density update (dens_step)
  - Add external source
  - Diffusion
  - Advection

Equation for density $d(\mathbf{x})$

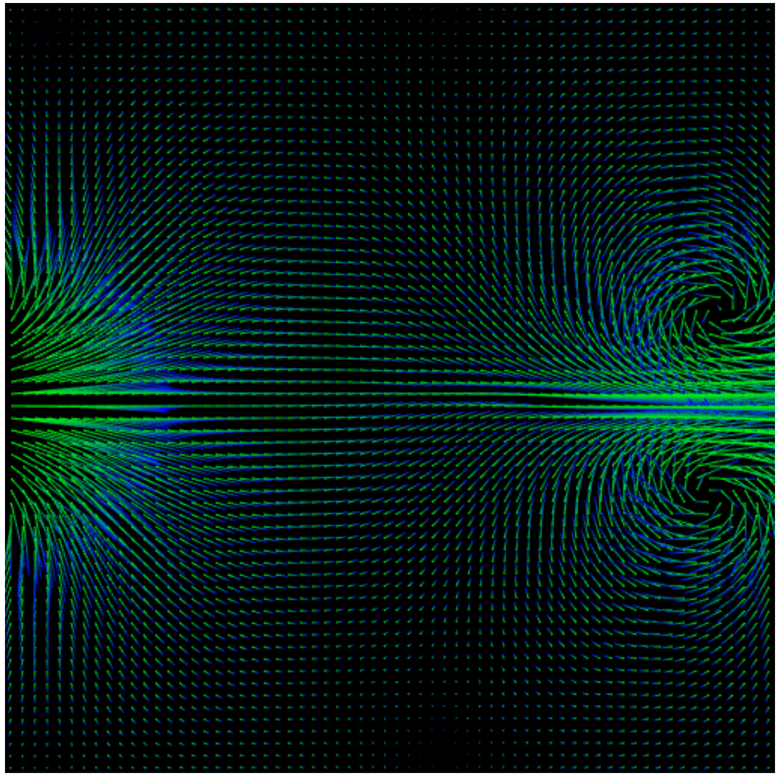
$$\frac{\partial d}{\partial t} = -(\mathbf{u} \cdot \nabla)d + \nu \Delta d + s$$

Boundary conditions

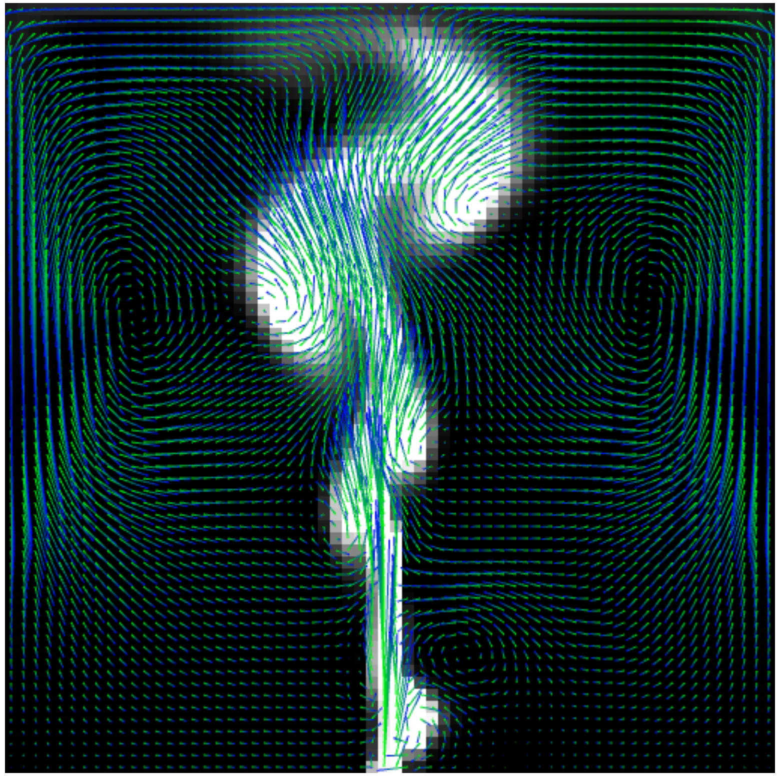
Zero velocity component normal to boundary



Make left/right (top/bottom) boundaries continuous



Keep adding constant amount



- For more complex cases, need more advanced techniques
  - E.g., curved boundaries, sheets thinner than grid spacing, etc

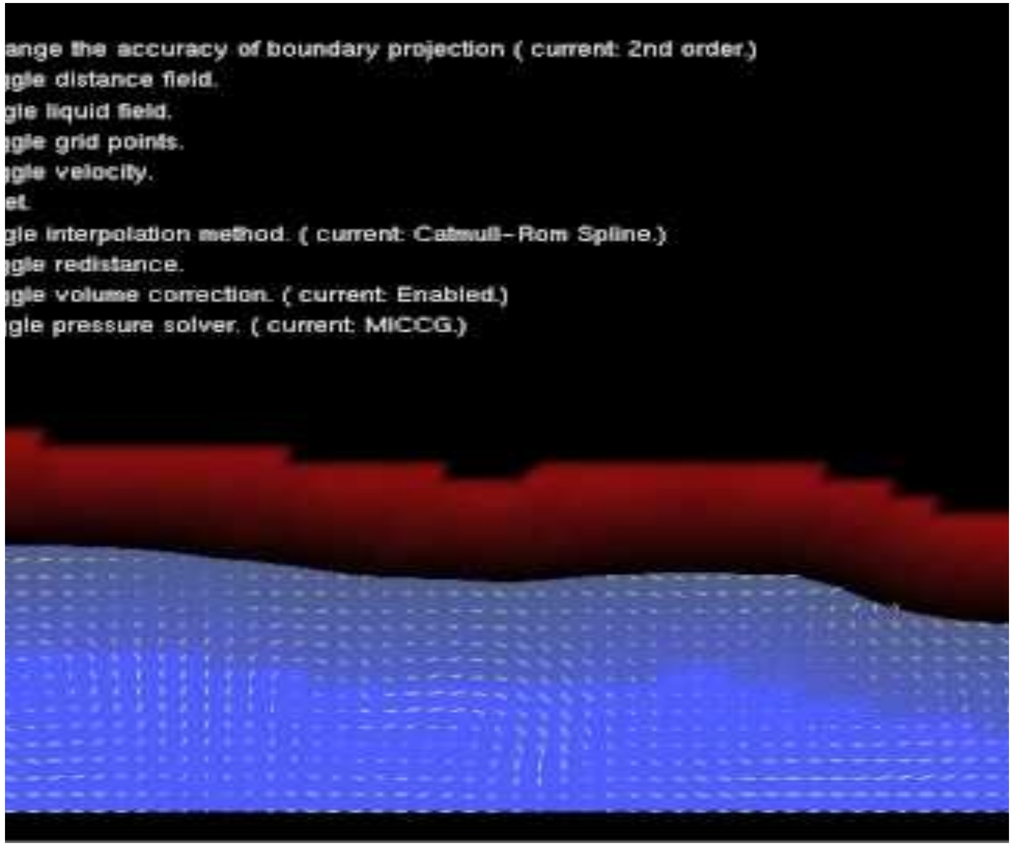
Advanced topics

Representing liquid surfaces using level set

- Introduce scalar field $\phi(\mathbf{x})$ representing distance to the liquid surface
  - $\phi(\mathbf{x}) < 0$ means liquid, $\phi(\mathbf{x}) > 0$ means air
  - Initialize properly

- Advect $\phi(\mathbf{x})$ by velocity

- During projection, set boundary condition $p(\mathbf{x}) = 0$ at surface $\phi(\mathbf{x}) = 0$



https://www.youtube.com/watch?v=Ss89OpQ_u54
http://code.google.com/p/levelset2d/

Practical animation of liquids [Foster SIGGRAPH01]

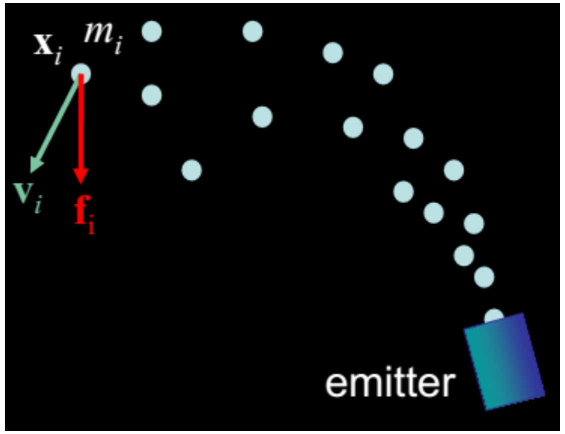
Various advection algorithms

- Semi-Lagrangian
- Upwind
- MacCormack
- WENO5
- QUICK

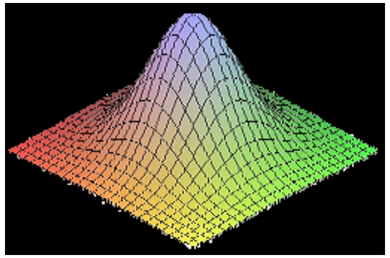
https://www.youtube.com/watch?v=UYu8JgbbHpE
http://code.google.com/p/2dsmoke/

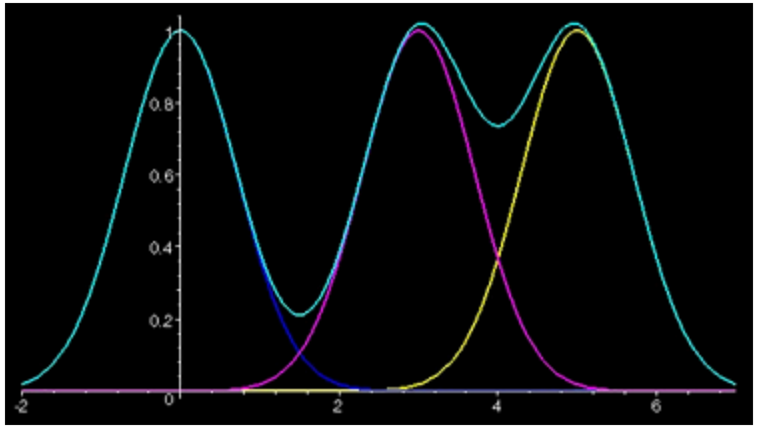
https://en.wikipedia.org/wiki/MacCormack_method
https://en.wikipedia.org/wiki/Upwind_scheme

Smoothed **P**article **H**ydrodynamics



- Representative of Lagrangian methods
- Move particles with mass according to velocity
- Derive continuous fields from discrete particles using smoothing kernel
    - $W(r) = \frac{315}{64\pi h^9}(h^2 - r^2)^3$
    - Density: $\rho(\mathbf{x}) = \sum_j m_j W(\|\mathbf{x} - \mathbf{x}_j\|)$
    - Velocity: $\mathbf{u}(\mathbf{x}) = \sum_j \frac{m_j}{\rho(\mathbf{x}_j)} \mathbf{u}_j \, W(\|\mathbf{x} - \mathbf{x}_j\|)$





- 1st & 2nd order derivatives can be obtained by ∇W & ΔW

Particle-Based Fluid Simulation for Interactive Applications [Muller SCA03]
https://slideplayer.com/slide/4790539/

Smoothed **P**article **H**ydrodynamics

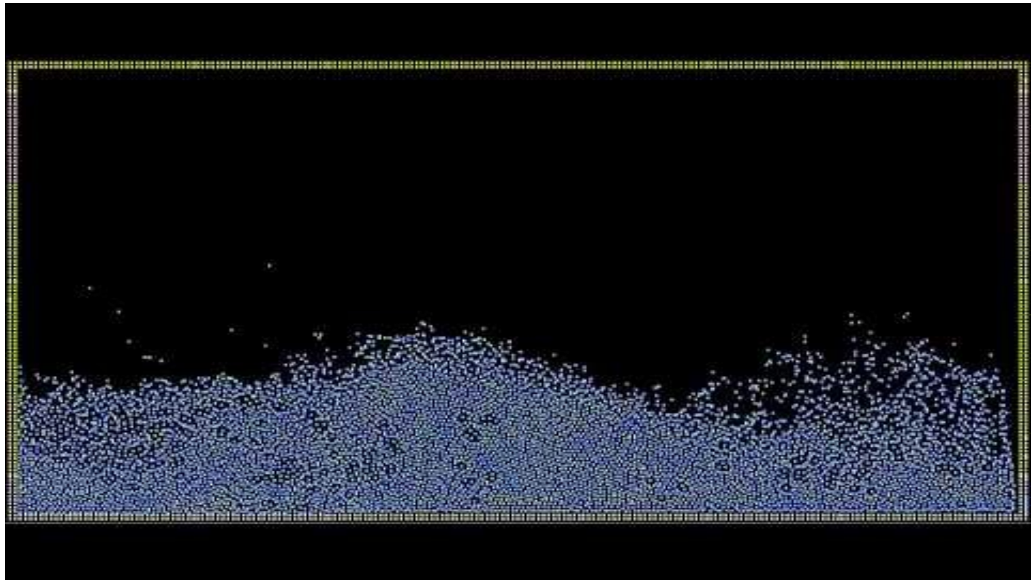
- Force acting on particle: $-\frac{1}{\rho(\mathbf{x}_i)}\nabla p(\mathbf{x}_i) + \nu\Delta\mathbf{u}(\mathbf{x}_i) + \mathbf{f}$

  Pressure — Viscosity — Ext. force

- By the ideal gas law $pV = NRT$, pressure & density are proportional:

$$p(\mathbf{x}) = k\,\rho(\mathbf{x})$$

➔ No need to solve Poission equation!



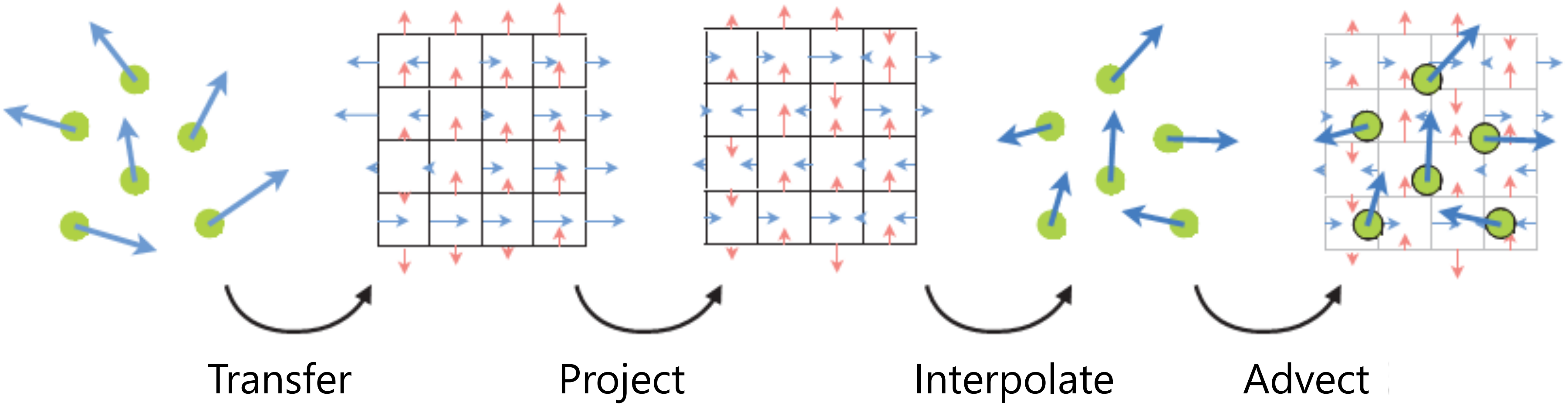
https://www.youtube.com/watch?v=M8WPINWAWPY

Particle-Based Fluid Simulation for Interactive Applications [Muller SCA03]
https://slideplayer.com/slide/4790539/

Hybrid of Eularian & Lagrangian

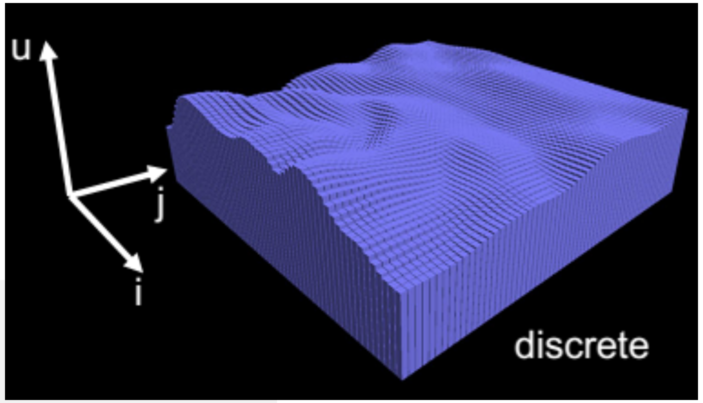
	Eulerian	Lagrangian
Advection	Numerical dissipation ☹	No numerical dissipation ☺
Projection	Accurate ☺	Inaccurate ☹

- PIC (**P**article **I**n **C**ell) and FLIP (**FL**uid **I**mplicit **P**article)



Animating sand as a fluid [Zhu,Bridson,SIGGRAPH05]

Approximation of water surface by height field




```
initialize u[i,j] as you like
set v[i,j] = 0
loop
    v[i,j] +=(u[i-1,j] + u[i+1,j] + u[i,j-1] + u[i,j+1])/4 – u[i,j]
    v[i,j] *= 0.99
    u[i,j] += v[i,j]
endloop
```

Real Time Fluids in Games (by Matthias Müller)
https://slideplayer.com/slide/4790539/

- WebGL code
  - http://madebyevan.com/webgl-water/
  - http://dblsai.github.io/WebGL-Fluid/

Pointers

- JavaScript code
  - http://www.ibiblio.org/e-notes/webgl/gpu/fluid.htm
  - https://nerget.com/fluidSim/
  - http://dev.miaumiau.cat/sph/
  - http://p.brm.sk/fluid/
- C++ code
  - http://code.google.com/p/flip3d/
  - http://code.google.com/p/levelset2d/
  - http://code.google.com/p/smoke3d/
  - http://code.google.com/p/2dsmoke/
  - http://www.cs.ubc.ca/~rbridson/download/simple_flip2d.tar.gz
- Shiokaze Framework: https://shiokaze.ryichando.graphics/
- Books
  - Fluid Simulation for Computer Graphics, by R. Bridson, 2008
  - The Art of Fluid Animation, by Jos Stam, 2015