Computer Graphics 3
Physics Based Graphics

Fluid Simulation

Inhalt

- Navier-Stoke's Equations
- Numeric Solution
- Applications

Literatur

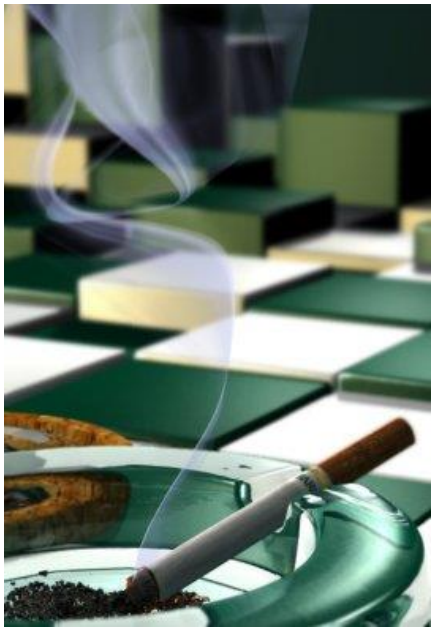
- Robert Bridson and Matthias Müller-Fischer, Fluid Simulation for Computer Animation, SIGGRAPH 2007 Course Notes (http://people.cs.ubc.ca/~rbridson/fluidsimulation)
- Jos Stam, Stable Fluids, Siggraph 1999
- Homepage Ron Fedkiw
- http://www.navier-stokes.net
- http://www.answers.com/topic/navier-stokes-equations

Demos

- http://jamie-wong.com/2016/08/05/webgl-fluid-simulation
- http://david.li/fluid

Navier-Stoke‘s-Equations

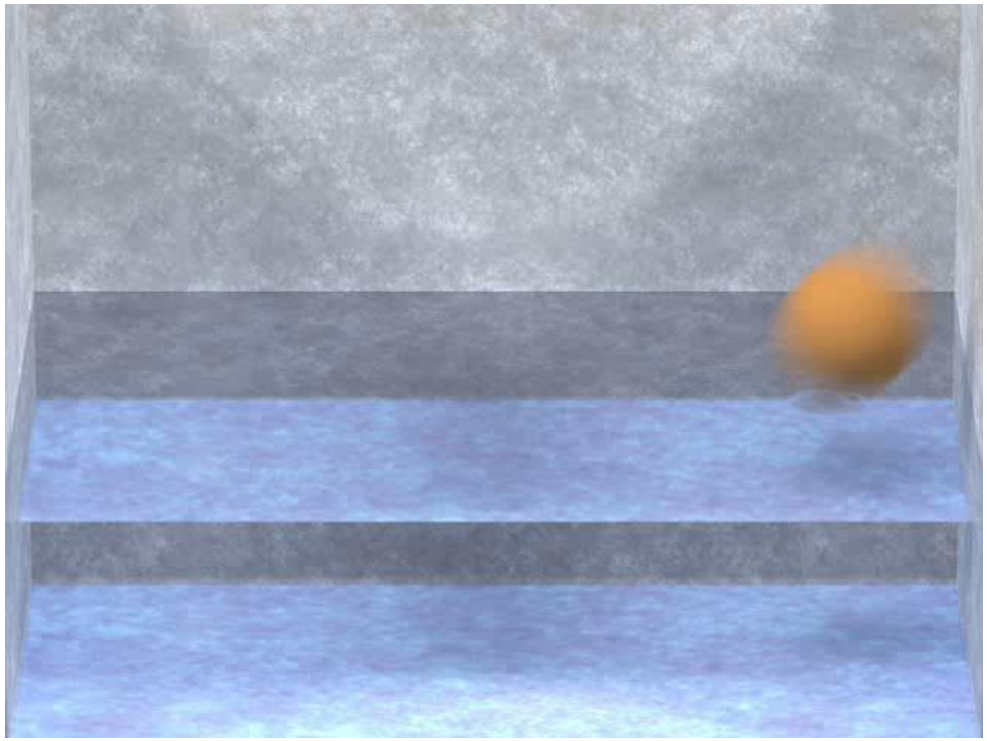
The Navier-Stokes equations describe the dynamics of fluids and can be applied with different parameters for gases and liquids.

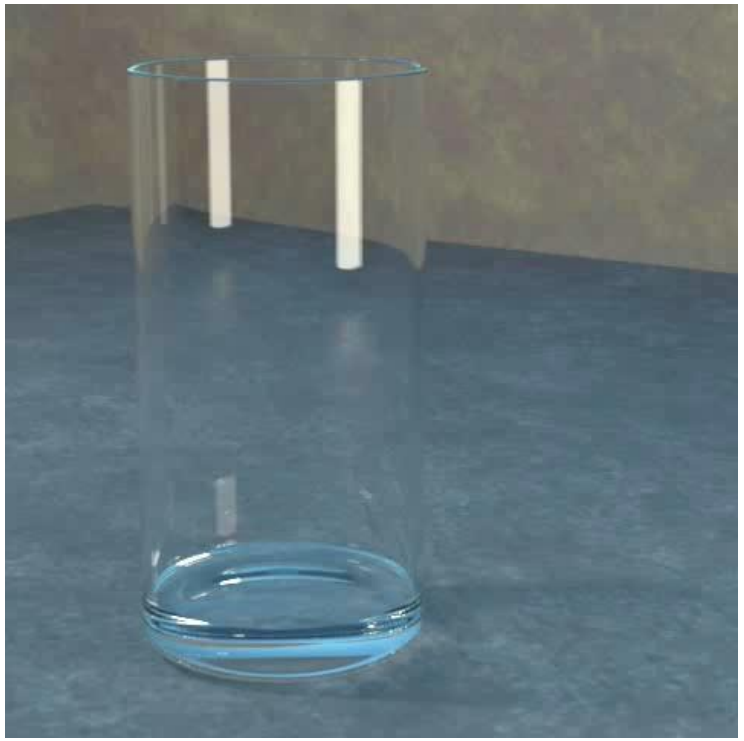


The Navier-Stokes equations are derived from the conservation of mass and momentum of the fluid.

Applications

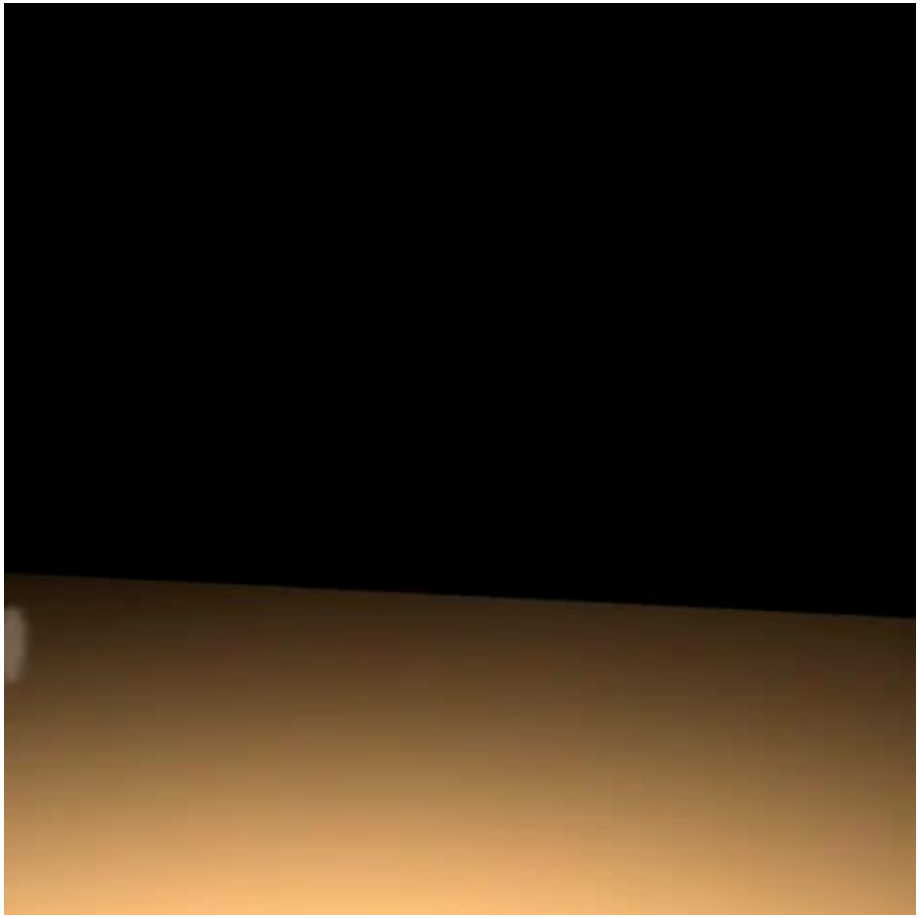
Water (Fedwik 2001)

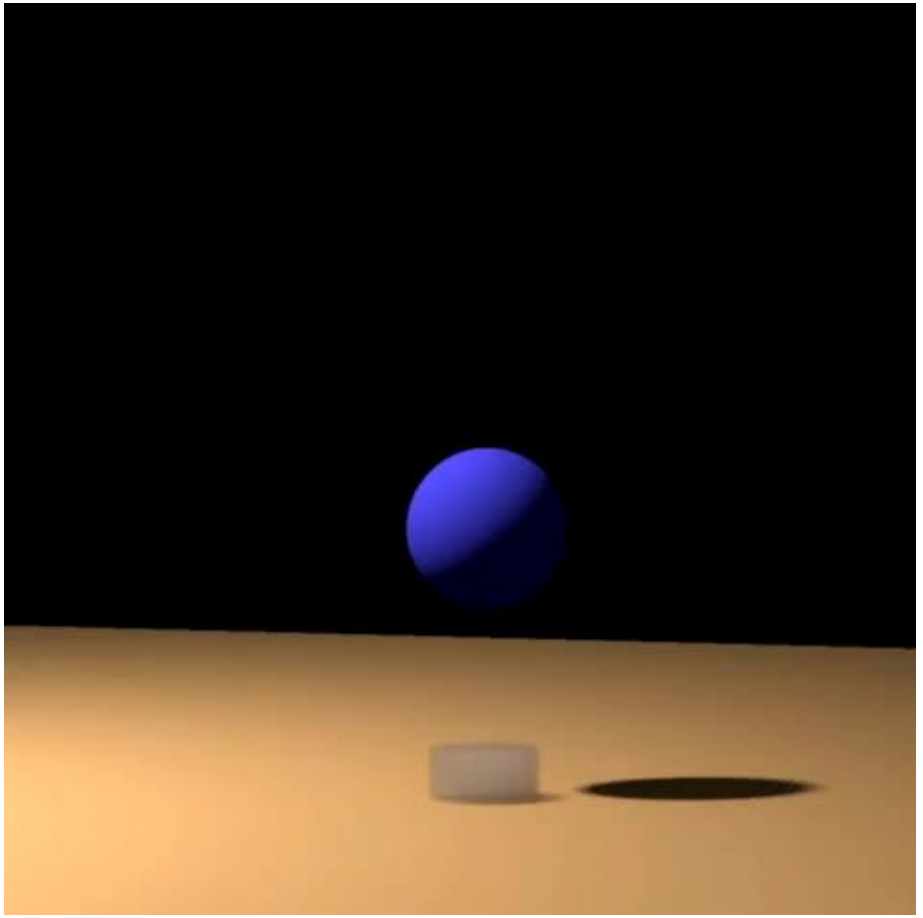




Applications

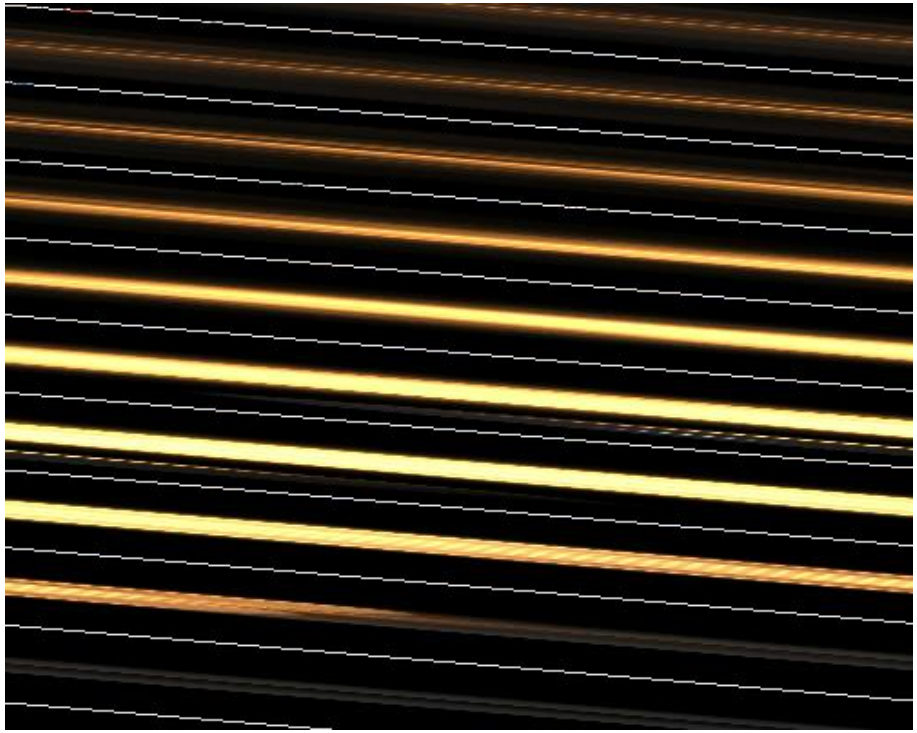
Smoke

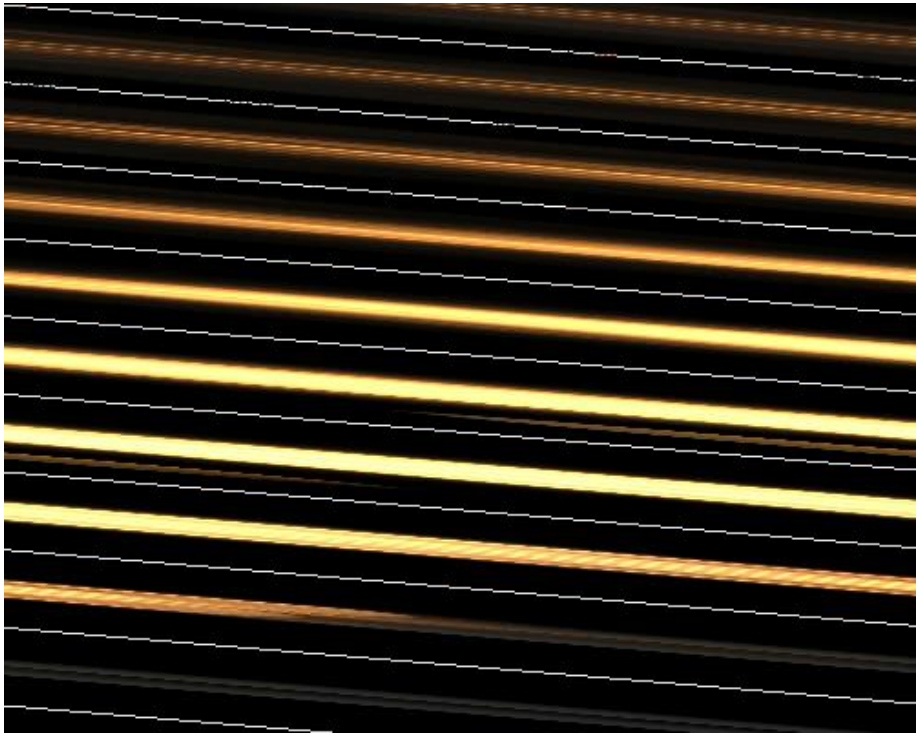




Applications

Fire (Fedwik 2002)





Navier-Stoke‘s-Equations

Incompressible Navier-Stoke’s:

- “Momentum Equation”

$$\frac{\partial \vec{\boldsymbol{u}}}{\partial t} + \vec{\boldsymbol{u}} \cdot \nabla \vec{\boldsymbol{u}} + \frac{1}{\rho} \nabla p = \vec{\boldsymbol{g}} + \nu \nabla \cdot \nabla \vec{\boldsymbol{u}}$$

- “Incompressibility condition”

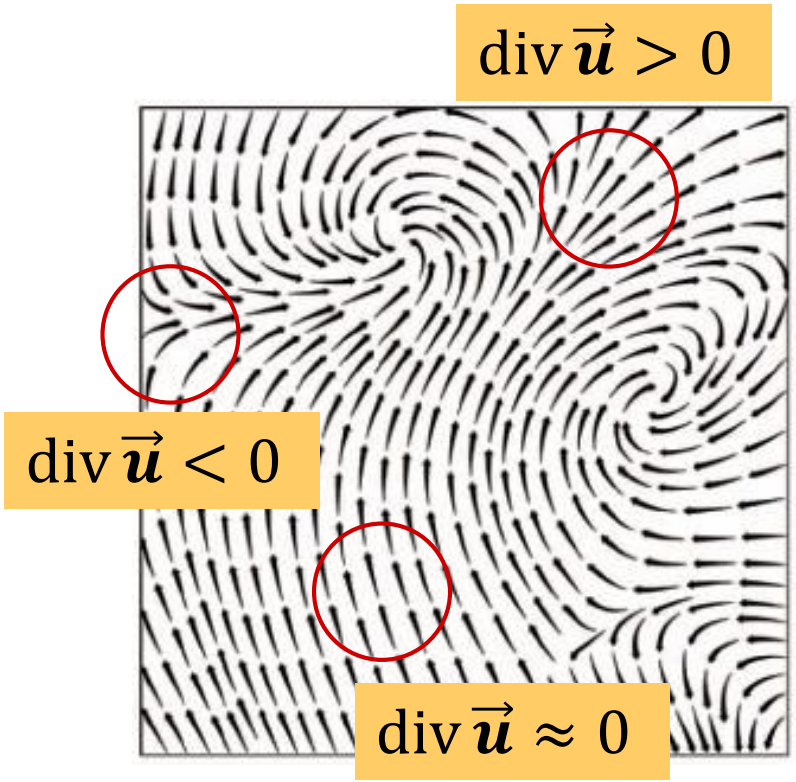
$$\nabla \cdot \vec{\boldsymbol{u}} = 0$$

- Symbols:
  $\vec{\boldsymbol{u}}$... velocity with components u, v, w
  ρ ... fluid density (water $1000 kg/m^3$, air $1.3 kg/m^3$)
  p ... pressure
  $\vec{\boldsymbol{g}}$ ... acceleration due to gravity or animator
  ν ... kinematic viscosity
  $\vec{\boldsymbol{f}}$ ... extension of $\vec{\boldsymbol{g}}$ to all accelerations due to body forces

Navier-Stoke's-Equations

- The divergence $\nabla \cdot \vec{u}$ or $\operatorname{div} \vec{u}$ of a vector field $\vec{u}$ describes how strongly the stream lines diverge ($\operatorname{div} \vec{u} > 0$) and converge ($\operatorname{div} \vec{u} < 0$), respectively

- Divergence does not depend on the time derivative but only on spatial derivatives:



$$\operatorname{div} \vec{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = \begin{pmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \end{pmatrix} \cdot \vec{u} = \nabla \cdot \vec{u}$$

Navier-Stoke's-Equations

Dimension-less Formulation:

- Firstly, define characteristic values for length (L), velocity (u_∞), density (ρ_∞) and pressure (p_∞)
- perform variable substitution:

$$\underline{\boldsymbol{x}}^* \equiv \frac{1}{L}\underline{\boldsymbol{x}}, t^* \equiv \frac{u_\infty}{L}t, \vec{\boldsymbol{u}}^* \equiv \frac{1}{u_\infty}\vec{\boldsymbol{u}}, p^* \equiv \frac{1}{\rho_\infty u_\infty^2}(p - p_\infty)$$

- collect characteristic values in dimension-less constants
  - Reynold's Number ... $\mathrm{Re} \equiv \frac{\rho_\infty u_\infty L}{\mu}$
  - Froude Number ... $\mathrm{Fr} \equiv \frac{u_\infty}{\sqrt{L\|\vec{\boldsymbol{g}}\|}}$
- define dimension-less force $\vec{\boldsymbol{f}}^* \equiv \frac{L}{u_\infty^2}\vec{\boldsymbol{g}} = \frac{1}{\mathrm{Fr}^2} \cdot \frac{\vec{\boldsymbol{g}}}{\|\vec{\boldsymbol{g}}\|}$
- Plugging all in yields dimension-less Navier-Stoke's-equations
- The asterix is left out in the following

Dimension-less Navier-Stoke‘s-Equations
incompressible viscous fluids:

$$\frac{\partial \vec{\boldsymbol{u}}}{\partial t} + \vec{\boldsymbol{u}} \cdot \nabla \vec{\boldsymbol{u}} = -\nabla p + \frac{1}{\mathrm{Re}} \nabla \cdot \nabla \vec{\boldsymbol{u}} + \vec{\boldsymbol{f}}, \qquad \nabla \cdot \vec{\boldsymbol{u}} = 0$$

- component form:

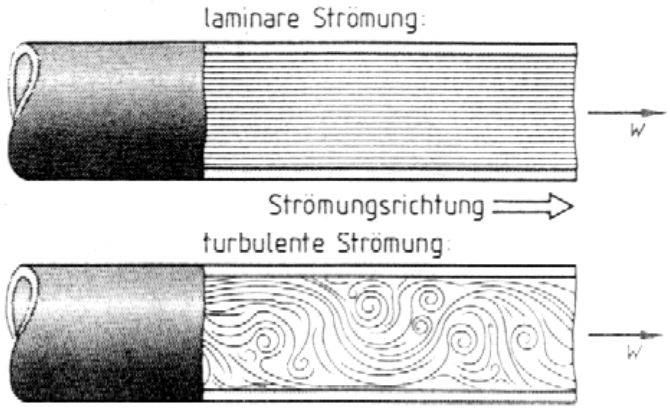
$$\frac{\partial u_i}{\partial t} + \sum_j u_j \frac{\partial u_i}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{1}{\mathrm{Re}} \sum_j \frac{\partial^2 u_i}{\partial x_j^2} + f_i \quad \forall i \in \{1,2,3\}$$

$$\sum_j \frac{\partial u_j}{\partial x_j} = 0$$

Navier-Stoke's-Equations

Reynold's Number is ratio between inertia force and frictional force. It determines the behavior of the fluid (for gases typically large and for fluids a broad range [water ... honey] possible)

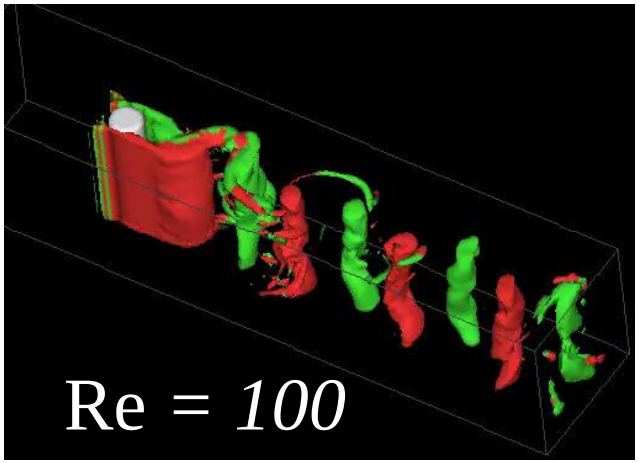
- $\mathrm{Re} < 1$: stationary flow, i.e. velocity field does not depend on time. Stream lines move along obstacles
- $\mathrm{Re} > 40$: periodic behavior
- Re even larger: quasi-periodic, individual vortices detach
- $\mathrm{Re} > \mathrm{Re}_{\mathrm{crit}} \approx 2000$: flow becomes turbulent, chaotic, vortical (due to non-linear equations)

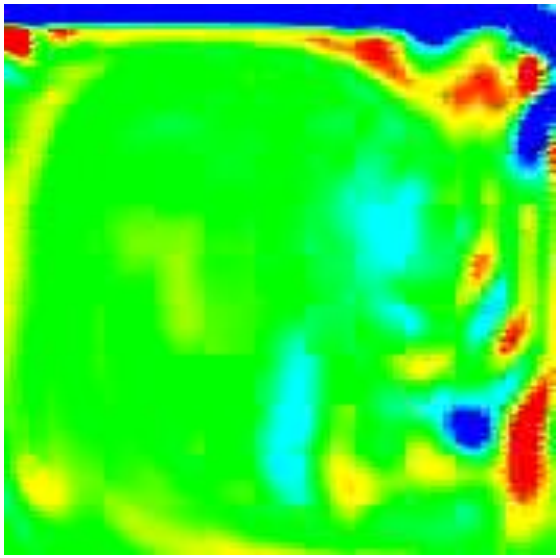


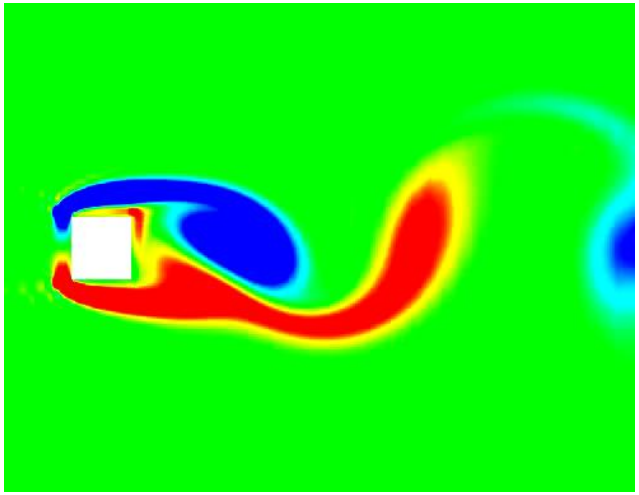
Quelle: MFink

Navier-Stoke‘s-Equations

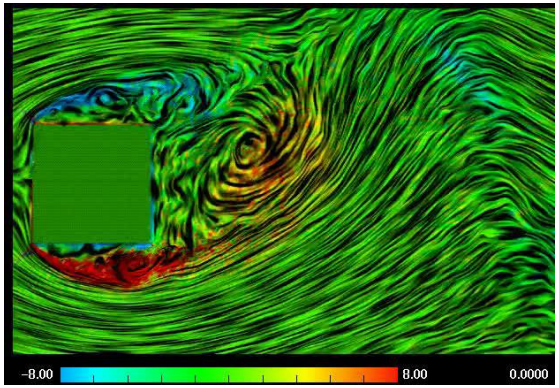
Reynold‘s Number Examples



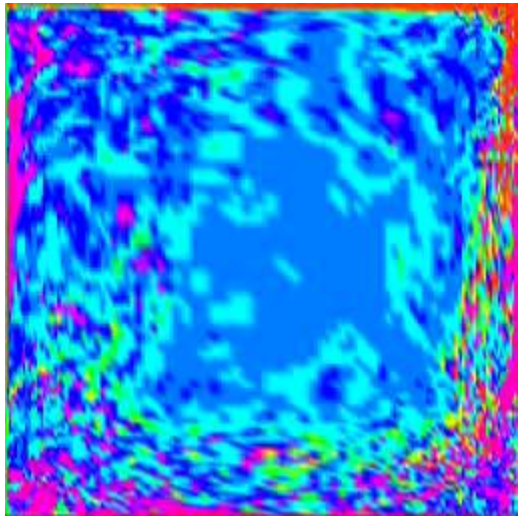




Re = *200*



Re = *22000*



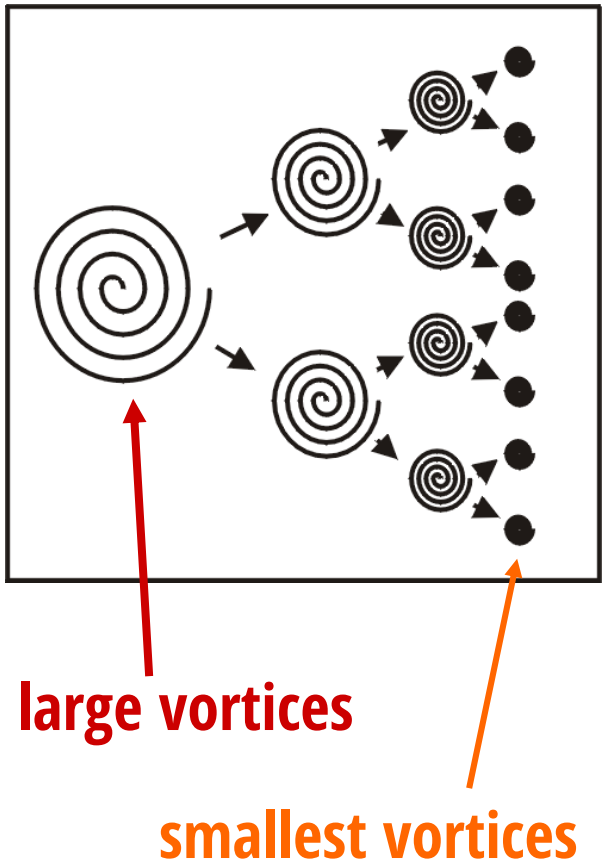
Re = *100000*

Kolmogorov-Cascade of turbulent Fluids

Too much kinetic energy is converted into friction through vortex cascades. Re determines the depth of the vortex cascade.

For high Reynold's numbers very small vortices are still relevant. Then a high grid resolution and small time steps are necessary.

The accurate simulation of turbulent fluids is computational expensive.



Navier-Stoke‘s-Equations

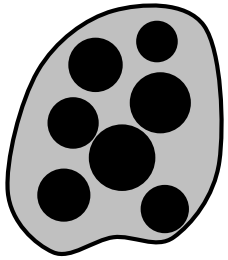
Discussion:

- One of the 7 „Millennium Problems“: Existence- and Uniqueness of „physically plausible “ (energy-restricted) solution of Navier-Stoke‘s-Equations in 3 dimensions! http://www.claymath.org/prizeproblems
- Precise simulation of air resistance of cars, buoyancy of airplanes, water resistance of ships, spray nozzles, weather forecasts, … computationally complex
- In CG we look for fast approximate solutions which provide a good visual impression without suppressing vortices.

Momentum Equation

$$\frac{\partial \vec{\boldsymbol{u}}}{\partial t} + \vec{\boldsymbol{u}} \cdot \nabla \vec{\boldsymbol{u}} + \frac{1}{\rho} \nabla p = \vec{\boldsymbol{g}} + \nu \nabla \cdot \nabla \vec{\boldsymbol{u}}$$

- Just a specialized version of $\vec{\boldsymbol{F}} = m\vec{\boldsymbol{a}}$
- Let's build it up intuitively
- Imagine modeling a fluid with a bunch of particles (e.g. blobs of water)
  - A blob has a mass m, a volume V, velocity $\vec{\boldsymbol{u}}$
  - We'll write the acceleration as $D\vec{\boldsymbol{u}}/Dt$ ("material derivative")



$$m \frac{D\vec{\boldsymbol{u}}}{Dt} = \vec{\boldsymbol{F}}$$

  - What forces act on the blob?

Forces on Fluids

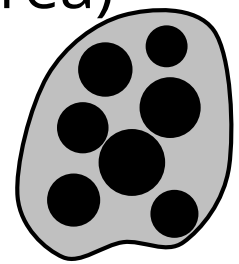
- Gravity: mg and other "body forces" $\vec{\boldsymbol{K}}$ designed by animator

$$m\frac{D\vec{\boldsymbol{u}}}{Dt} = m\vec{\boldsymbol{g}} + \vec{\boldsymbol{K}}$$

- And a blob of fluid also exerts contact forces on its neighbouring blobs…

Pressure

- The “normal” contact force is pressure (force/area)
  - How blobs push against each other, and how they stick together



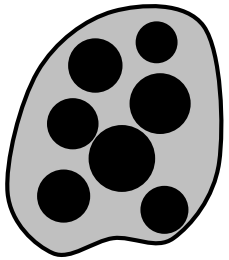
- If pressure is equal in every direction, net force is zero…
  Important quantity is pressure gradient:

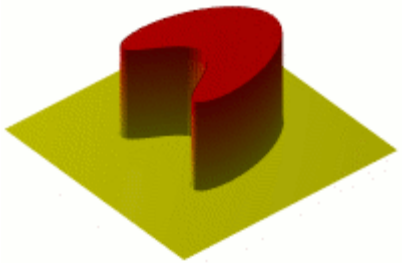
$$m\frac{D\vec{\boldsymbol{u}}}{Dt} = m\vec{\boldsymbol{g}} - V\nabla p + \vec{\boldsymbol{K}}$$

  We integrate over the blob’s volume
  (equivalent to integrating $-p_{\hat{\boldsymbol{n}}}$ over blob’s surface)
- What is the pressure? Coming soon…

Viscosity

- Think of it as frictional part of contact force: a sticky (viscous) fluid blob resists other blobs moving past it
- For now, simple model is that we want velocity to be blurred/diffused/…
- Blurring in PDE form comes from the Laplacia $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} = \nabla \cdot \nabla$:





heat equation:
$\frac{\partial \vec{\boldsymbol{u}}}{\partial t} = \nu \Delta \vec{\boldsymbol{u}}$

$$m\frac{D\vec{\boldsymbol{u}}}{Dt} = m\vec{\boldsymbol{g}} - V\nabla p + V\mu\Delta\vec{\boldsymbol{u}} + \vec{\boldsymbol{K}}$$

- new symbol:
  μ … dynamic viscosity (gives force instead of acceleration)

The Continuum Limit

$$m\frac{D\vec{\boldsymbol{u}}}{Dt} = m\vec{\boldsymbol{g}} - V\nabla p + V\mu\Delta\vec{\boldsymbol{u}} + \vec{\boldsymbol{K}}$$

- Model the world as a continuum:
  - # particles → ∞
    while mass and volume → 0
- Then $F = ma$ would become $0 = 0$:
  - Divide by mass before the limit:

$$\frac{D\vec{\boldsymbol{u}}}{Dt} = \vec{\boldsymbol{g}} - \frac{V}{m}\nabla p + \frac{V}{m}\mu\Delta\vec{\boldsymbol{u}} + \overbrace{\frac{1}{m}\vec{\boldsymbol{K}}}^{\vec{\boldsymbol{k}}}$$

user acceleration instead of force

- The fluid density is $\rho = m/V$:

$$\frac{D\vec{\boldsymbol{u}}}{Dt} = \vec{\boldsymbol{g}} - \frac{1}{\rho}\nabla p + \frac{\mu}{\rho}\Delta\vec{\boldsymbol{u}} + \vec{\boldsymbol{k}}$$

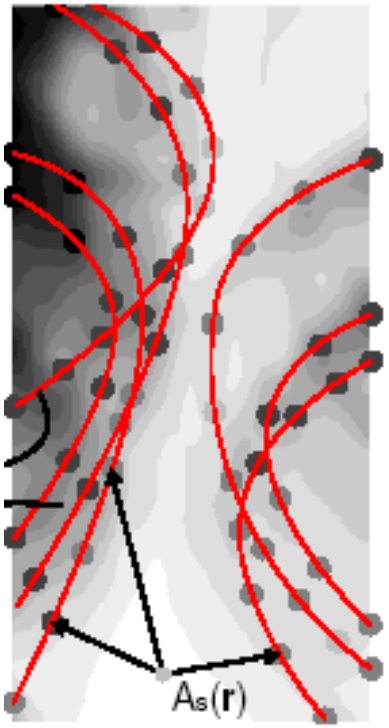
- With $\nu = \mu/\rho$ this is almost the same as the standard eq'n (in fact, the form we mostly use!)
- The only weird thing is Du/Dt…

Navier-Stokes-Gleichungen

Eulerian viewpoint

Lagrangian viewpoint

grid
based
simulation



Particle
based
simulation

Navier Stokes Equations

Lagrangian viewpoint:

- Treat the world like a particle system
- Label each quantum of matter, track where it goes (how fast, acceleration, etc.)

Eulerian viewpoint:

- Fix your point in space
- Measure stuff as it flows by

Think of measuring the temperature:

- Lagrangian: in a balloon, floating with the wind
- Eulerian: on the ground, wind blows by

The Material Derivative

- We have fluid moving in a velocity field $\vec{\boldsymbol{u}}$
- It possesses some quantity q
- At an instant in time t and a position in space $\underline{\boldsymbol{x}}$, the fluid at $\underline{\boldsymbol{x}}$ has $q(\underline{\boldsymbol{x}}, t)$ (therefore it is an **Eulerian** field)
- How fast is q of that blob of fluid changing in the **Lagrangian** viewpoint?
  - Answer: the material derivative Dq/Dt
  - It all boils down to the chain rule:

$$\frac{Dq(\underline{\boldsymbol{x}}, t)}{Dt} = \frac{\partial q}{\partial t} + \frac{\partial q}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial q}{\partial y}\frac{\partial y}{\partial t} + \frac{\partial q}{\partial z}\frac{\partial z}{\partial t} = \frac{\partial q}{\partial t} + \nabla q \cdot \vec{\boldsymbol{u}}$$

  - We usually rearrange it:

$$\frac{Dq(\underline{\boldsymbol{x}}, t)}{Dt} = \frac{\partial q}{\partial t} + \vec{\boldsymbol{u}} \cdot \nabla q$$

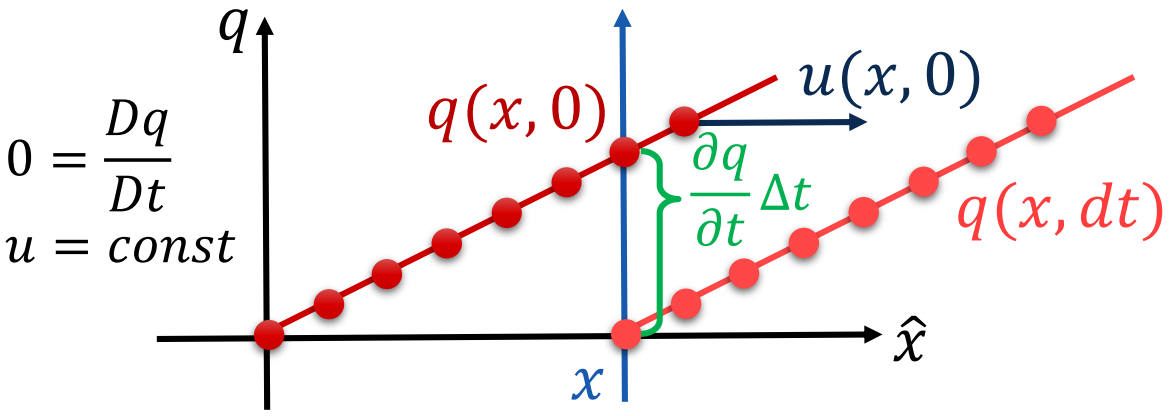
change at $\underline{x}$ — change due to movement

Advection Equation

- If quantity q moves with the fluid particles we call this process “advection” and can formulate this as

$$0 = \frac{Dq}{Dt} = \frac{\partial q}{\partial t} + \vec{\boldsymbol{u}} \cdot \nabla q$$

- That is, how fast q is changing at a fixed point in space ($\partial q/\partial t$) comes from two things:
  - How fast q is changing for the blob of fluid at x
  - How fast fluid with different values of q is flowing past



$q(x,0) = 10x$
$u(x,0) = 1$
$\Delta t = 5$

$$\frac{\partial q}{\partial t}\Delta t = -u\Delta t \frac{\partial q(x,0)}{\partial x}$$
$$= -1 \cdot 5 \cdot 10 = -50$$

Material Derivative of Vector Fields

- Say our fluid has a color variable (RGB vector) $\vec{\vec{\vec{C}}}$
- We still write

$$\frac{D\vec{\vec{\vec{C}}}(\underline{\boldsymbol{x}}, t)}{Dt} = \frac{\partial \vec{\vec{\vec{C}}}}{\partial t} + \vec{\boldsymbol{u}} \cdot \nabla \vec{\vec{\vec{C}}}$$

- The dot-product and gradient confusing?
- Just do it component-by-component:

$$\frac{D\vec{\vec{\vec{C}}}(\underline{\boldsymbol{x}}, t)}{Dt} = \begin{pmatrix} DR/Dt \\ DG/Dt \\ DB/Dt \end{pmatrix} = \begin{bmatrix} \partial R/\partial t + \vec{\boldsymbol{u}} \cdot \nabla R \\ \partial G/\partial t + \vec{\boldsymbol{u}} \cdot \nabla G \\ \partial B/\partial t + \vec{\boldsymbol{u}} \cdot \nabla B \end{bmatrix}$$

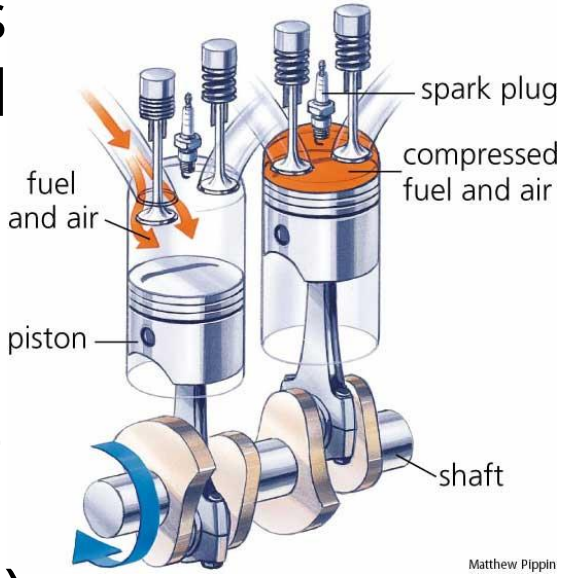
- This holds even if the vector field is velocity itself:

$$\frac{D\vec{\boldsymbol{u}}(\underline{\boldsymbol{x}}, t)}{Dt} = \frac{\partial \vec{\boldsymbol{u}}}{\partial t} + \vec{\boldsymbol{u}} \cdot \nabla \vec{\boldsymbol{u}} = \vec{\boldsymbol{g}} - \frac{1}{\rho} \nabla p + \frac{\mu}{\rho} \Delta \vec{\boldsymbol{u}} + \vec{\boldsymbol{k}}$$

The Incompressibility Condition

Compressibility

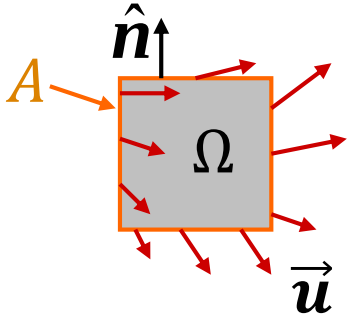
- Real fluids are compressible
- Shock waves, acoustic waves, pistons…
  - Note: liquids change their volume as well as gases, otherwise there would be no sound underwater
- But this is nearly irrelevant for animation
  - Shocks move too fast to normally be seen
    (easier/better to hack in their effects)
  - Acoustic waves usually have little effect on visible fluid motion
  - Pistons are boring



Incompressibility

- Rather than having to simulate acoustic and shock waves, eliminate them from our model: assume fluid is **incompressible**
- Turn stiff system into a constraint, just like rigid bodies!
- If you fix your eyes on any volume Ω of space, volume of fluid in = volume of fluid out:

$$\oiint_{\partial\Omega} \vec{\boldsymbol{u}} \cdot \hat{\boldsymbol{n}} dA = 0$$



- Let's use the divergence theorem:

$$\oiint_{\partial\Omega} \vec{\boldsymbol{u}} \cdot \hat{\boldsymbol{n}}\, dA = \iiint_{\Omega} \nabla \cdot \vec{\boldsymbol{u}} dV$$

- So for any region, the integral of $\nabla \cdot \vec{\boldsymbol{u}}$ is zero
- Thus it's zero everywhere: $\nabla \cdot \vec{\boldsymbol{u}} = 0$

Pressure

- Pressure p: whatever it takes to make the velocity field divergence free
- If you know constrained dynamics, $\nabla \cdot \vec{\boldsymbol{u}} = 0$ is a constraint, and pressure is the matching Lagrange multiplier
- Our simulator will follow this approach: solve for a pressure that makes our fluid incompressible at each time step.

Alternative

- To avoid having to solve linear systems, can turn this into a soft-constraint
  - Track density, make pressure proportional to density variation
  - Called “artificial compressibility”
- However, easier to accelerate a linear solver… if the linear system is well-posed

Inviscid Fluids

Dropping Viscosity

- In most scenarios, viscosity term is much smaller
- Convenient to drop it from the equations:
  - Zero viscosity = “inviscid”
  - Inviscid Navier-Stokes = “Euler equations”
- Numerical simulation typically makes errors that resemble physical viscosity, so we have the visual effect of it anyway
  - Called “numerical dissipation”
  - For animation: often numerical dissipation is larger than the true physical viscosity!

Aside: A Few Figures

- Dynamic viscosity of air: $\mu_{air} \approx 1.8 \times 10^{-5}\ Ns/m^2$
- Density of air: $\rho_{air} \approx 1.3\ kg/m^3$

- Dynamic viscosity of water: $\mu_{water} \approx 1.1 \times 10^{-3}\ Ns/m^2$
- Density of water: $\rho_{water} \approx 1000\ kg/m^3$

- The ratio, $\nu = \mu/\rho$ ("kinematic viscosity") is what's important for the motion of the fluid...
  ... air is 10 times more viscous than water!

- A.k.a. the incompressible Euler equations:

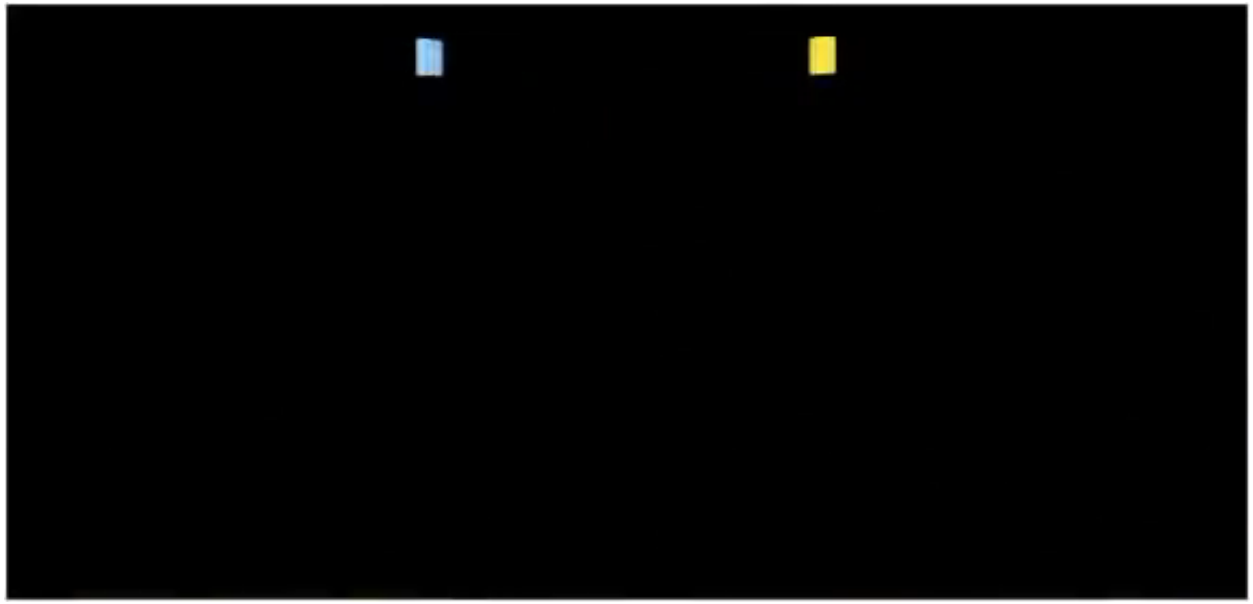
$$\frac{\partial \vec{\boldsymbol{u}}}{\partial t} + \vec{\boldsymbol{u}} \cdot \nabla \vec{\boldsymbol{u}} + \frac{1}{\rho} \nabla p = \vec{\boldsymbol{g}}, \qquad \nabla \cdot \vec{\boldsymbol{u}} = 0$$

Boundary Conditions

Boundary Conditions

- We know what's going on inside the fluid: what about at the surface?
- Three types of boundaries
  - **Solid wall:** fluid is adjacent to a solid body
  - **Free surface:** fluid is adjacent to nothing
    (e.g. water is so much denser than air, might as well forget about the air)
  - **Other fluid:** possibly discontinuous jump in quantities (density, …)

https://www.youtube.com/watch?v=kphzE0KXuNw



Solid Wall Boundaries

- No fluid can enter or come out of a solid wall:

$$\vec{\boldsymbol{u}} \cdot \hat{\boldsymbol{n}} = \vec{\boldsymbol{u}}_{solid} \cdot \hat{\boldsymbol{n}}$$

- For common case of $\vec{\boldsymbol{u}}_{solid} = 0$:

$$\vec{\boldsymbol{u}} \cdot \hat{\boldsymbol{n}} = 0$$

  Sometimes called the “no-stick” condition, since we let fluid pass by tangentially
- For viscous fluids, can additionally impose “no-slip” condition:

$$\vec{\boldsymbol{u}} = \vec{\boldsymbol{u}}_{solid}$$

Free Surface

- Neglecting the other fluid, we model it simply as

  pressure=constant

- Since only pressure gradient is important, we can choose the constant to be zero:

  $$p = 0$$

- If surface tension is important (not covered today), pressure is instead related to mean curvature of surface

Multiple Fluids

- At fluid-fluid boundaries, the trick is to determine “jump conditions”
- For a fluid quantity q, $[q] = q_1 - q_2$

- Density jump $[\rho]$ is known
- Normal velocity jump must be zero: $[\vec{\boldsymbol{u}} \cdot \hat{\boldsymbol{n}}] = 0$

- For inviscid flow, tangential velocities may be unrelated (jump is unknown)

- With no surface tension, pressure jump $[p] = 0$ vanishes

Numerical Simulation Overview

Splitting

- We have lots of terms in the momentum equation:

$$\frac{\partial \vec{\boldsymbol{u}}}{\partial t} = -\vec{\boldsymbol{u}} \cdot \nabla \vec{\boldsymbol{u}} + \vec{\boldsymbol{g}} + \nu \Delta \vec{\boldsymbol{u}} - \frac{1}{\rho} \nabla p$$

  a pain to handle them all simultaneously
- Instead we split up the equation into its terms, and integrate them one after the other
  - Makes for easier software design too:
    a separate solution module for each term
- First order accurate in time
  - Can be made more accurate, not covered today.

A Splitting Example

- Say we have a differential equation

$$\frac{dq}{dt} = f(q) + g(q)$$

- And we can solve the component parts:
  - $\mathrm{SolveF}(q, \Delta t)$ solves $dq/dt = f(q)$ for time Δt
  - $\mathrm{SolveG}(q, \Delta t)$ solves $dq/dt = g(q)$ for time Δt
- Put them together to solve the full thing:
  - $q^* = \mathrm{SolveF}(q^n, \Delta t)$
  - $q^{n+1} = \mathrm{SolveG}(q^*, \Delta t)$

- Does it work? – Up to $O(\Delta t)$:

$$\frac{dq}{dt} \approx \frac{q^{n+1} - q^n}{\Delta t} = \frac{q^{n+1} - q^*}{\Delta t} + \frac{q^* - q^n}{\Delta t} \approx g(q) + f(q)$$

Splitting Momentum

- We have four terms:

$$\frac{\partial \vec{\boldsymbol{u}}}{\partial t} = -\vec{\boldsymbol{u}} \cdot \nabla \vec{\boldsymbol{u}} + \vec{\boldsymbol{g}} + \nu \Delta \vec{\boldsymbol{u}} - \frac{1}{\rho} \nabla p$$

- First term: **advection –** move fluid in velocity field ($D\vec{\boldsymbol{u}}/Dt = 0$)

$$\frac{\partial \vec{\boldsymbol{u}}}{\partial t} = -\vec{\boldsymbol{u}} \cdot \nabla \vec{\boldsymbol{u}}$$

- Second term: **gravity**

$$\frac{\partial \vec{\boldsymbol{u}}}{\partial t} = \vec{\boldsymbol{g}}$$

- Third term: **diffusion**

$$\frac{\partial \vec{\boldsymbol{u}}}{\partial t} = \nu \Delta \vec{\boldsymbol{u}}$$

- Final term: **pressure update –** make the fluid incompressible:

$$\frac{\partial \vec{\boldsymbol{u}}}{\partial t} = -\frac{1}{\rho} \nabla p, \qquad \nabla \cdot \vec{\boldsymbol{u}} = 0$$

Space

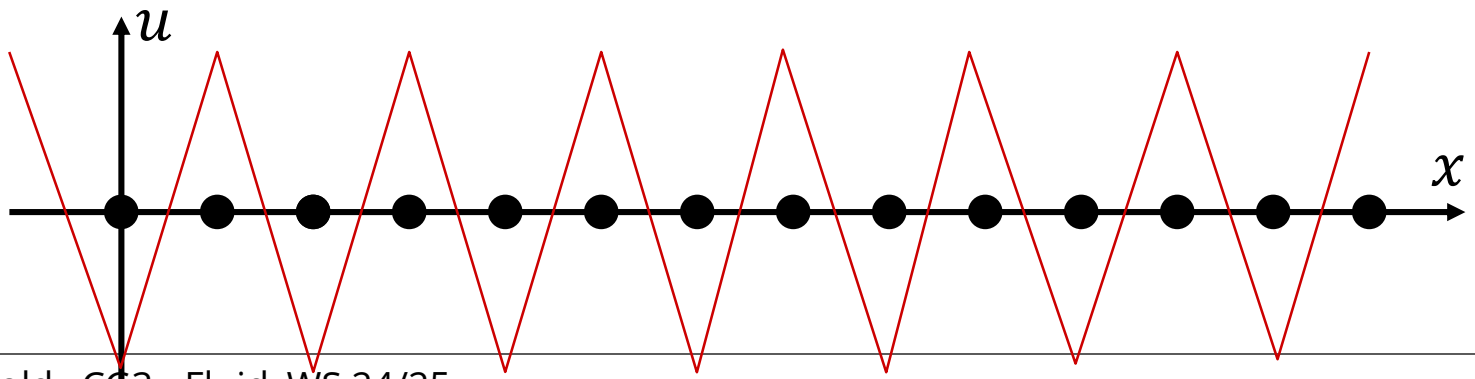
- That's our general strategy in time; what about space?
- We'll begin with a fixed Eulerian grid
  - Trivial to set up
  - Easy to approximate spatial derivatives
  - Particularly good for the effect of pressure
- Disadvantage: advection doesn't work so well
  - Later: particle methods that fix this

A Simple Grid

- We could put all our fluid variables at the nodes of a regular grid – But this causes some major problems
- In 1D: incompressibility means $\frac{\partial u}{\partial x} = 0$
- Approximate at grid point with: $\frac{u_{i+1} - u_{i-1}}{2\Delta x} = 0$
- Note the velocity at the grid point isn't involved!

A Simple Grid Disaster

- The only solutions to $\frac{\partial u}{\partial x} = 0$ are $u = constant$, but numerical solver has other solutions:



Staggered Grids

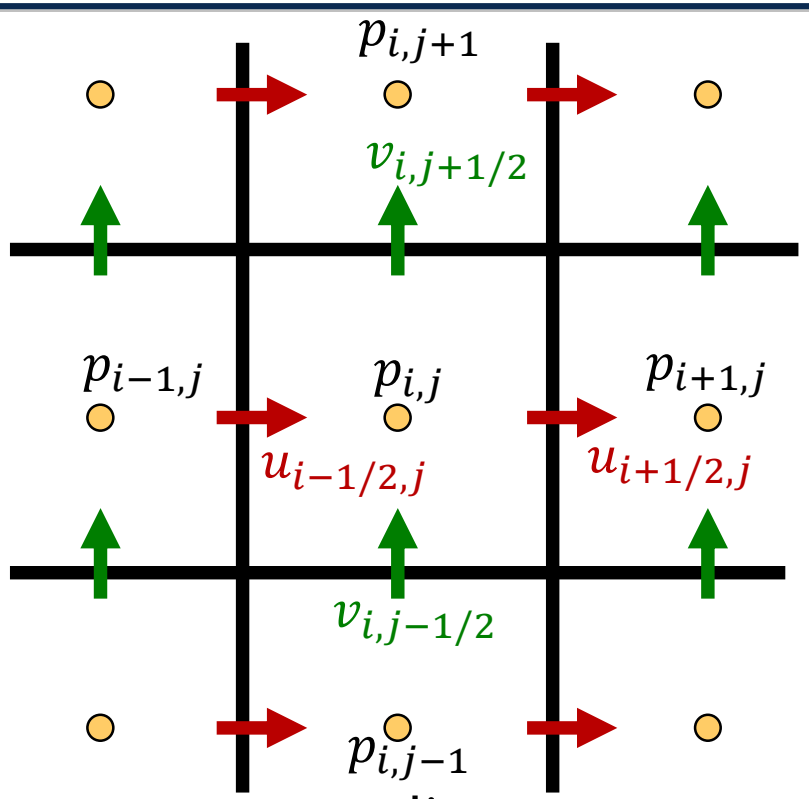
- Problem is solved if we don't skip over grid points
- To make it unbiased, we **stagger** the grid:
  put velocities halfway between grid points
- In 1D, we estimate divergence at a grid point as:

$$\frac{\partial u}{\partial x}(x_i) \approx \frac{u_{i+\frac{1}{2}} - u_{i-\frac{1}{2}}}{\Delta x}$$

- Problem solved!

The MAC Grid

- From the Marker-and-Cell (MAC) method [Harlow&Welch'65]
- A particular staggering of variables in 2D/3D that works well for incompressible fluids:
  - Grid cell (i, j, k) has pressure $p_{i,j,k}$ at its center
  - x-part of velocity $u_{i+1/2,j,k}$ in middle of x-face between grid cells (i, j, k) and $(i + 1, j, k)$
  - y-part of velocity $v_{i,j+1/2,k}$ in middle of y-face
  - z-part of velocity $w_{i,j,k+1/2}$ in middle of z-face

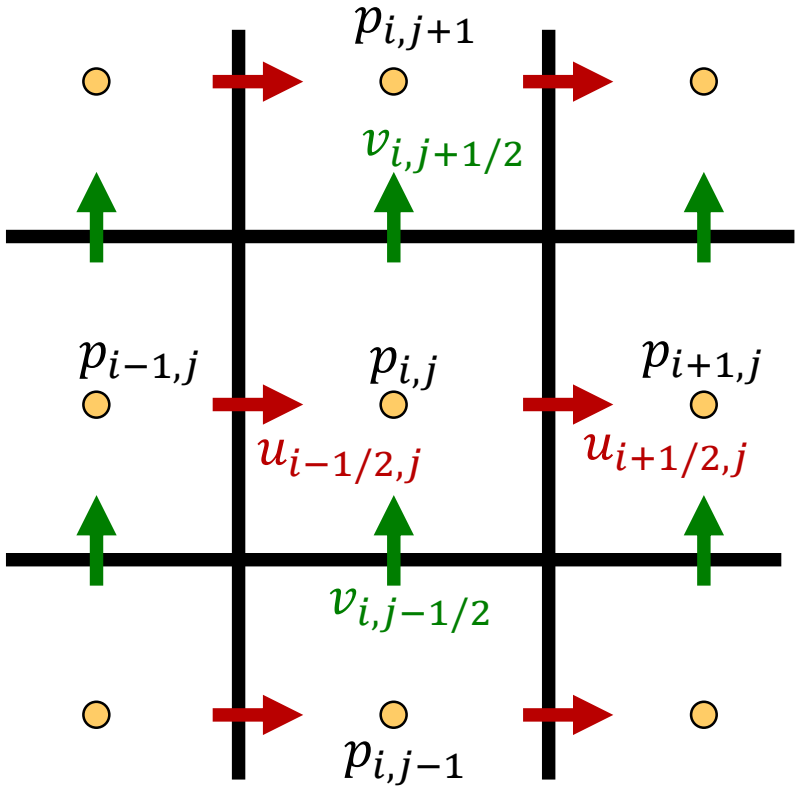


- pressure gradient needed at velocity sampling locations, e.g.

$$\left(\frac{\partial p}{\partial x}\right)_{i+\frac{1}{2},j} \approx \frac{p_{i+1,j} - p_{i,j}}{\Delta x}$$

Array storage

- Then for a $n_x \times n_y \times n_z$ grid, we store them as 3D arrays:
  - $p[n_x, n_y, n_z]$
  - $u[n_{x+1}, n_y, n_z]$
  - $v[n_x, n_{y+1}, n_z]$
  - $w[n_x, n_y, n_{z+1}]$
- And translate indices in code
  - $u_{i+\frac{1}{2},j,k} \equiv u[i+1, j, k]$ …
- Downside
  - Not having all quantities at the same spot makes some algorithms a pain
    - Even interpolation of velocities for pushing particles is annoying
  - One strategy: switch back and forth (colocated/staggered) by averaging
  - Robert's philosophy: avoid averaging as much as possible!



Advection Algorithms

Advecting Quantities

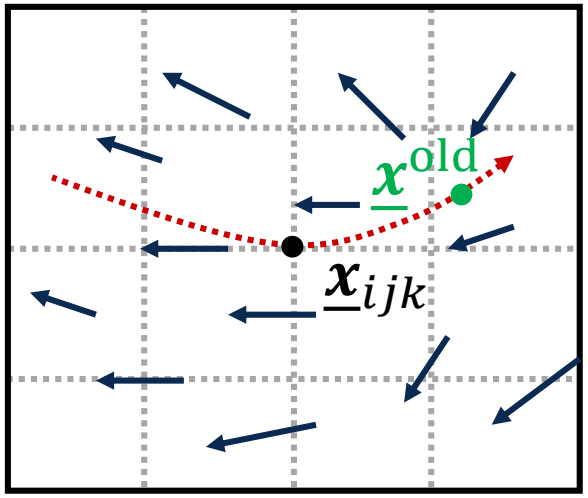
- The goal is to solve “the advection equation” for any grid quantity q (in particular, the components of velocity):

$$\frac{Dq}{Dt} = 0$$

- Rather than directly approximate spatial term, we’ll use the Lagrangian notion of advection directly
- We’re on an Eulerian grid, though, so the result will be called “semi-Lagrangian” (introduced to graphics by [Stam’99])

Semi-Lagrangian Advection

- $Dq/Dt = 0$ says q doesn't change as you follow the fluid.
- So $q(\underline{\boldsymbol{x}}^{\text{new}}, t^{\text{new}}) = q(\underline{\boldsymbol{x}}^{\text{old}}, t^{\text{old}})$
- We want to know q at each grid point at t^{new} (that is, we're interested in $\underline{\boldsymbol{x}}^{\text{new}} = \underline{\boldsymbol{x}}_{ijk}$)



- So we just need to figure out $\underline{\boldsymbol{x}}^{\text{old}}$ (where fluid at $\underline{\boldsymbol{x}}_{ijk}$ came from) and $q(\underline{\boldsymbol{x}}^{\text{old}})$ (what value of q was there before)
- We need to trace backwards through the velocity field.
- Simplest O(Δt) approach assumes constant $\vec{\boldsymbol{u}}$ and uses explicit Euler: $\underline{\boldsymbol{x}}^{\text{old}} = \underline{\boldsymbol{x}}^{\text{new}} - \Delta t \vec{\boldsymbol{u}}(\underline{\boldsymbol{x}}^{\text{new}})$
  - I.e. tracing through the time-reversed flow with one step of Forward Euler
  - Other ODE methods can (and **should**) be used

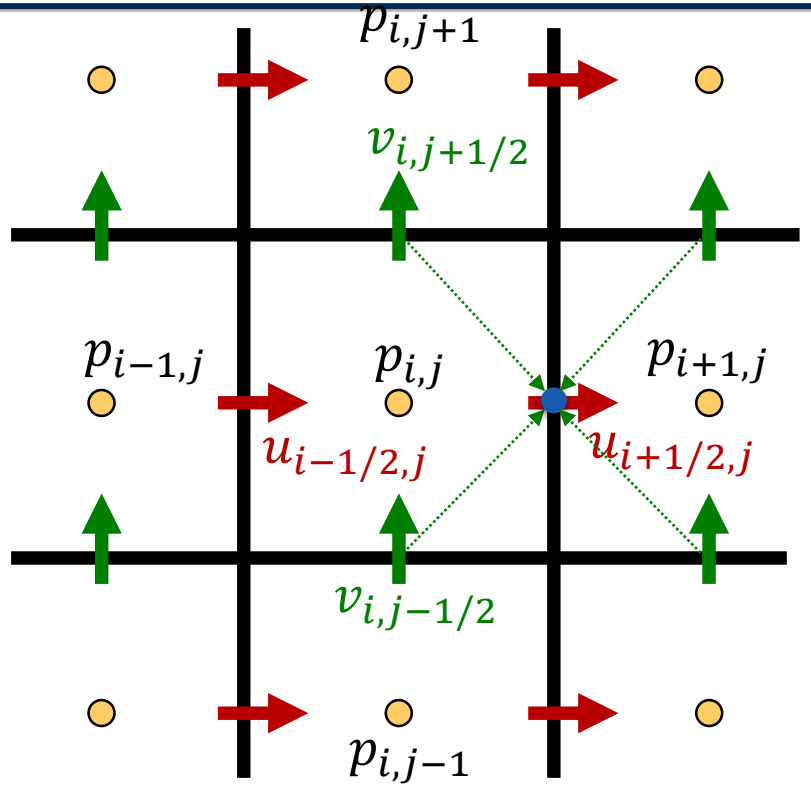
Aside: ODE methods

- Chief interesting aspect of fluid motion is vortices: rotation
- Any ODE integrator we use should handle rotation reasonably well
- Forward Euler **does not**
  - Instability, obvious spiral artifacts
- Runge-Kutta 2 is simplest semi-decent method

$$\underline{\boldsymbol{x}}^{n+1} = \underline{\boldsymbol{x}}^{n} + \Delta t \vec{\boldsymbol{u}}\left(\underline{\boldsymbol{x}}^{n} + \frac{1}{2}\Delta t \vec{\boldsymbol{u}}\left(\underline{\boldsymbol{x}}^{n}\right)\right)$$

Details of x^{old}

- We need $\vec{u}(\underline{x}_{ijk})$ for a quantity stored at (i, j, k)
  - For staggered quantities, say u, we need it at the staggered location: e.g. $\vec{u}\left(\underline{x}_{i+\frac{1}{2},j,k}\right)$
- We can get the velocity there by averaging the appropriate staggered components: e.g.



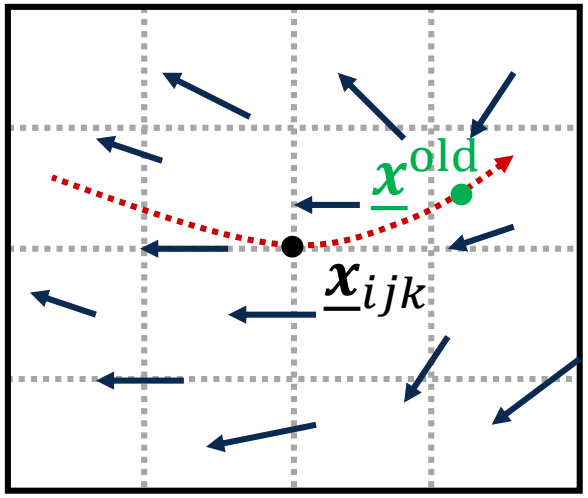
$$\vec{u}\left(\underline{x}_{i+\frac{1}{2},j,k}\right) = \begin{bmatrix} u_{i+\frac{1}{2},j,k} \\ \frac{1}{4}\left(v_{i,j-\frac{1}{2},k} + v_{i+1,j-\frac{1}{2},k} + v_{i,j+\frac{1}{2},k} + v_{i+1,j-\frac{1}{2},k}\right) \\ \frac{1}{4}\left(w_{i,j,k-\frac{1}{2}} + w_{i+1,j,k-\frac{1}{2}} + w_{i,j,k+\frac{1}{2}} + w_{i+1,j,k+\frac{1}{2}}\right) \end{bmatrix}$$

Which u to Use?

- Note that we only want to advect quantities in an incompressible velocity field
  - Otherwise the quantity gets compressed (often an obvious unphysical artifact)
- For example, when we advect u, v, and w themselves, we use the old incompressible values stored in the grid
  - Do not update as you go!
- In conclusion the **advection** step should be the **first** in the split integration of the four terms in the momentum equation

Finding $q(x^{old})$

- Odds are when we trace back from a grid point to x^{old} we won't land on a grid point
  - So we don't have an old value of q there
- Solution: interpolate from nearby grid points
  - Simplest method: bi/trilinear interpolation
  - Know your grid: be careful to get it right for staggered quantities!



Boundary Conditions

- What if x^{old} isn't in the fluid? (or a nearest grid point we're interpolating from is not in the fluid?)
- Solution: **extrapolate** from boundary of fluid
  - Extrapolate before advection, to all grid points in the domain that aren't fluid
- ALSO: if fluid can move to a grid point that isn't fluid now, make sure to do semi-Lagrangian advection there
  - Use the extrapolated velocity

Extrapolation

- One of the trickiest bits of code to get right
- For sanity check, be careful about what's **known** and **unknown** on the grid
  - Replace all unknowns with linear combinations of knowns
- More on this later (level sets)

- Solids: the inviscid "no-stick" boundary condition

$$\vec{\boldsymbol{u}} \cdot \hat{\boldsymbol{n}} = \vec{\boldsymbol{u}}_{solid} \cdot \hat{\boldsymbol{n}}$$

  only constraint the normal components of $\vec{\boldsymbol{u}}$
- In order to represent the tangential components we need to extrapolate into solids, and can't use the solid's own velocity (result would be bad numerical viscosity)

Body Forces

Integrating Body Forces

- Gravity vector or volumetric animator forces:

$$\frac{\partial \vec{\boldsymbol{u}}}{\partial t} = \vec{\boldsymbol{g}}$$

- Simplest scheme: at every grid point just add

$$\vec{\boldsymbol{u}}^{*} = \vec{\boldsymbol{u}}^{advect} + \Delta t \vec{\boldsymbol{g}}$$

Making Fluids Incompressible

Poission Equation

- We want to find a pressure p so that the updated velocity:

$$\frac{\partial \vec{\boldsymbol{u}}}{\partial t} = -\frac{1}{\rho}\nabla p \Rightarrow \vec{\boldsymbol{u}}^{n+1} = \vec{\boldsymbol{u}}^{*} - \frac{\Delta t}{\rho}\nabla p$$

  is divergence free: $\nabla \cdot \vec{\boldsymbol{u}}^{n+1} = 0$

- while respecting the boundary conditions:
  - $\vec{\boldsymbol{u}}^{n+1} \cdot \hat{\boldsymbol{n}} = \vec{\boldsymbol{u}}_{solid} \cdot \hat{\boldsymbol{n}}$ … at solid walls
  - $p = 0$ … at free surface
- plugging pressure update into divergence constraint:

$$\nabla \cdot \left(\vec{\boldsymbol{u}}^{*} - \frac{\Delta t}{\rho}\nabla p \right) = 0$$

- Turns into a “Poisson equation” for pressure:

$$\frac{\Delta t}{\rho}\Delta p = \nabla \cdot \vec{\boldsymbol{u}}^{*} \qquad \text{inside}$$

$$\frac{\Delta t}{\rho}\nabla p \cdot \hat{\boldsymbol{n}} = (\vec{\boldsymbol{u}}^{*} - \vec{\boldsymbol{u}}_{solid}) \cdot \hat{\boldsymbol{n}} \qquad \text{at solid walls}$$

$$p = 0 \qquad \text{at free surface}$$

Discrete Pressure Update

- pressure update:

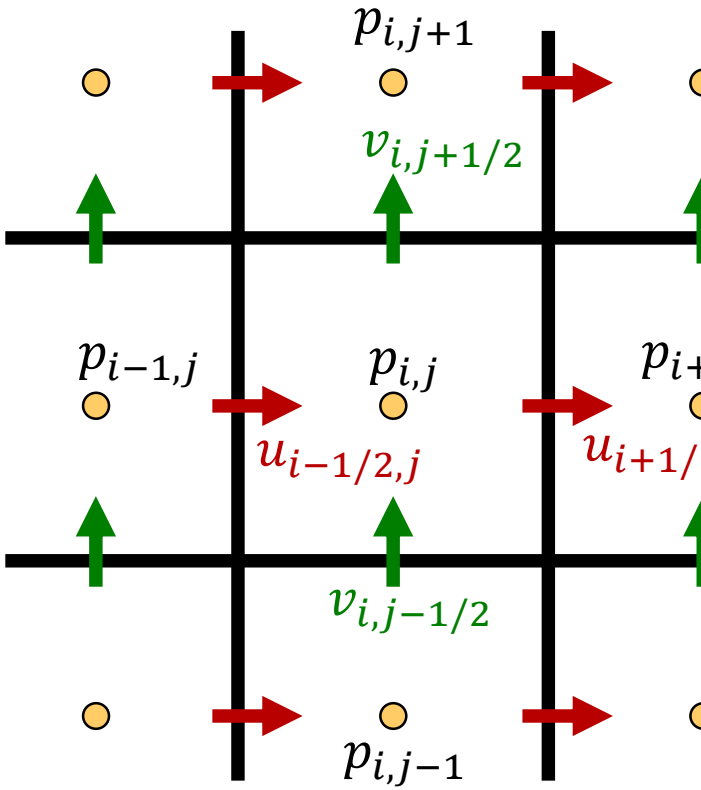
$$\frac{\partial \vec{\boldsymbol{u}}}{\partial t} = -\frac{1}{\rho}\nabla p \Rightarrow \vec{\boldsymbol{u}}^{n+1} = \vec{\boldsymbol{u}}^{*} - \frac{\Delta t}{\rho}\nabla p$$

- The discrete pressure update on the MAC grid:

$$u^{n+1}_{i+\frac{1}{2},j,k} = u^{*}_{i+\frac{1}{2},j,k} - \frac{\Delta t}{\rho}\cdot\frac{p_{i+1,j,k} - p_{i,j,k}}{\Delta x}$$

$$v^{n+1}_{i,j+\frac{1}{2},k} = v^{*}_{i,j+\frac{1}{2},k} - \frac{\Delta t}{\rho}\cdot\frac{p_{i,j+1,k} - p_{i,j,k}}{\Delta y}$$

$$w^{n+1}_{i,j,k+\frac{1}{2}} = w^{*}_{i,j,k+\frac{1}{2}} - \frac{\Delta t}{\rho}\cdot\frac{p_{i,j,k+1} - p_{i,j,k}}{\Delta z}$$

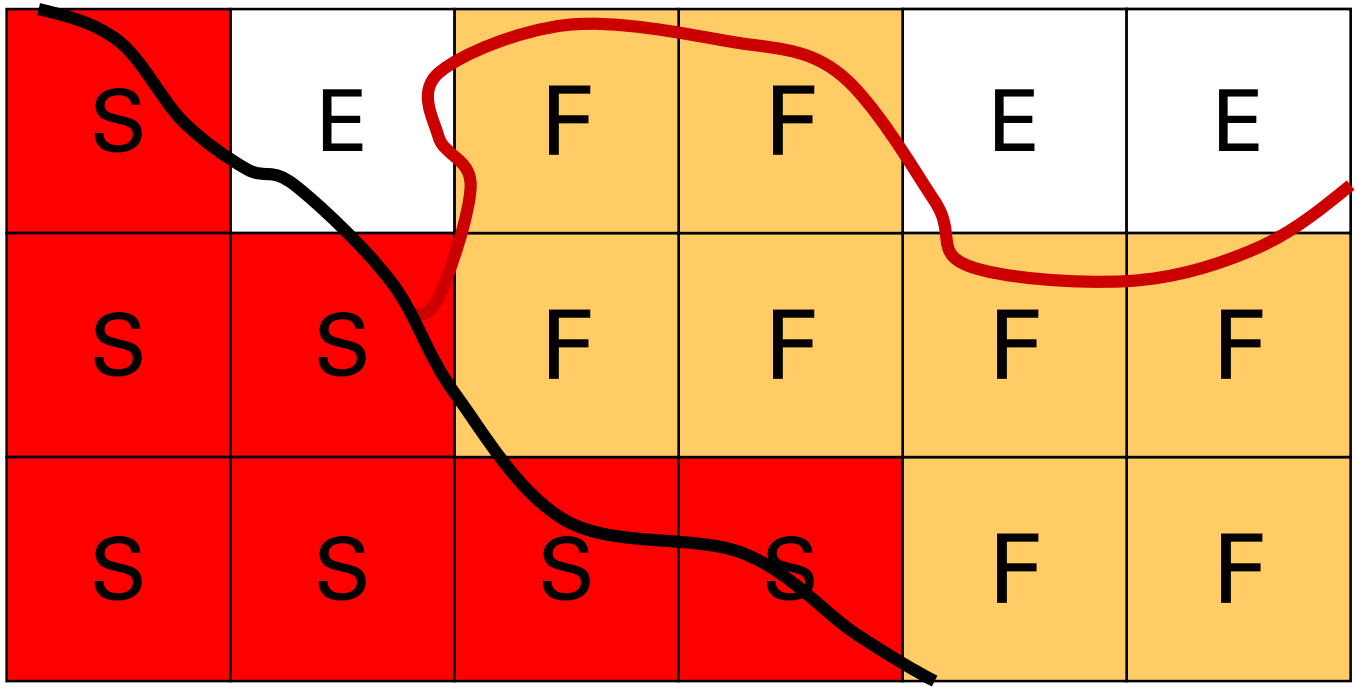


- discretized incompressibility condition on the new velocity (at (i,j,k)):

$$\frac{u^{n+1}_{i+\frac{1}{2},j,k} - u^{n+1}_{i-\frac{1}{2},j,k}}{\Delta x} + \frac{v^{n+1}_{i,j+\frac{1}{2},k} - v^{n+1}_{i,j-\frac{1}{2},k}}{\Delta y} + \frac{w^{n+1}_{i,j,k+\frac{1}{2}} - w^{n+1}_{i,j,k-\frac{1}{2}}}{\Delta z} = 0$$

Discrete Boundaries

- For now, let's voxelize the geometry:
  each grid cell is either fluid, solid, or empty



What's Active?

- Pressure: we'll only solve for p in Fluid cells
- Velocity: anything bordering a Fluid cell is good
- Boundary conditions:
  - empty cells: $p = 0$ (free surface)
  - solid cells: choose pressure such that pressure update of velocity ensures:

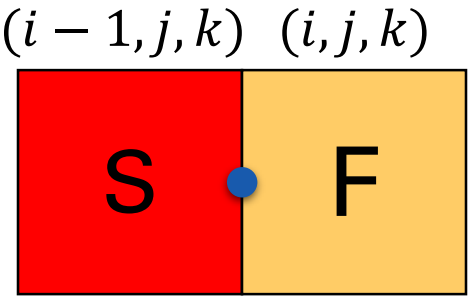
$$\vec{\boldsymbol{u}} \cdot \hat{\boldsymbol{n}} = \vec{\boldsymbol{u}}_{solid} \cdot \hat{\boldsymbol{n}}$$

(Note normal points to fluid cell!)
(Also note: although only one p in the center of a solid cell, it can have several active faces that constraint its p value…)

Example Solid Wall

- $(i-1,j,k)$ is solid, (i,j,k) is fluid
- Normal is $n = (1,0,0)^T$
- we want $u^{n+1}_{i-\frac{1}{2},j,k} = u_{\text{solid}}$
- The pressure update formula is:

$$u^{n+1}_{i-\frac{1}{2},j,k} = u^{*}_{i-\frac{1}{2},j,k} - \frac{\Delta t}{\rho} \cdot \frac{p_{i,j,k} - p_{i-1,j,k}}{\Delta x}$$



- So the "ghost" solid pressure is:

$$\tilde{p}_{i-1,j,k} = p_{i,j,k} + \frac{\rho \Delta x}{\Delta t}\left(u^{n+1}_{i-\frac{1}{2},j,k} - u^{*}_{i-\frac{1}{2},j,k}\right)$$

Discrete Pressure Equations

- Substitute pressure update formula into discrete divergence
- In each fluid cell (i, j, k) get an equation:

$$\frac{u^*_{i+\frac{1}{2},j,k} - \frac{\Delta t}{\rho} \cdot \frac{p_{i+1,j,k} - p_{i,j,k}}{\Delta x} - u^*_{i-\frac{1}{2},j,k} - \frac{\Delta t}{\rho} \cdot \frac{p_{i,j,k} - p_{i-1,j,k}}{\Delta x}}{\Delta x}$$
$$+ \frac{v^*_{i,j+\frac{1}{2},k} - \frac{\Delta t}{\rho} \cdot \frac{p_{i,j,k} - p_{i,j,k}}{\Delta y} - v^*_{i,j-\frac{1}{2},k} - \frac{\Delta t}{\rho} \cdot \frac{p_{i,j,k} - p_{i,j-1,k}}{\Delta y}}{\Delta y}$$
$$+ \frac{w^*_{i,j,k+\frac{1}{2}} - \frac{\Delta t}{\rho} \cdot \frac{p_{i,j,k+1} - p_{i,j,k}}{\Delta z} - w^*_{i,j,k-\frac{1}{2}} - \frac{\Delta t}{\rho} \cdot \frac{p_{i,j,k} - p_{i,j,k-1}}{\Delta z}}{\Delta z}$$
$$= 0$$

Simplified

- Now we assume $h = \Delta x = \Delta y = \Delta z$
- Collecting terms:

$$\frac{\Delta t}{\rho} \cdot \frac{6p_{i,j,k} - p_{i+1,j,k} - p_{i-1,j,k} - p_{i,j+1,k} - p_{i,j-1,k} - \cdots}{h^2}$$
$$= -\frac{u^*_{i+\frac{1}{2},j,k} - -u^*_{i-\frac{1}{2},j,k}}{h} - \frac{v^*_{i,j+\frac{1}{2},k} - v^*_{i,j-\frac{1}{2},k}}{h} - \cdots$$

- Empty neighbors: drop zero pressures
- Solid neighbors: e.g. $(i-1, j, k)$ is solid
  - Drop $p_{i-1,j,k}$ and reduce coefficient of $p_{i,j,k}$ by 1 (to 5…)
  - Replace $u_{i-\frac{1}{2},j,k}$ in right-hand side with u_{solid}

Linear Equations

- End up with a sparse set of linear equations to solve for pressure (Matrix is symmetric positive (semi-)definite)
- In 3D on large grids, direct methods unsatisfactory
- Instead use Preconditioned Conjugate Gradient, with Incomplete Cholesky preconditioner
- See course notes for full details (pseudo-code)
- Residual is how much divergence there is in $\vec{\boldsymbol{u}}^{n+1}$
  - Iterate until satisfied it's small enough

Voxelization is Suboptimal

- Free surface artifacts:
  - Waves less than a grid cell high aren't "seen" by the fluid solver – thus they don't behave right
  - Left with strangely textured surface
- Solid wall artifacts:
  - If boundary not grid-aligned, O(1) error – it doesn't even go away as you refine!
  - Slopes are turned into stairs, water will pool on artificial steps.
- More on this later…

Smoke

As per [Fedkiw et al.'01]

Soot

- Smoke is composed of soot particles suspended in air
- For simulation, need to track concentration of soot on the grid: $s(x,t)$
- Evolution equation:

$$\frac{Ds}{Dt} = k_s \nabla \cdot \nabla s$$

  - Extra term is diffusion (if $k_s \neq 0$)
  - Simplest implementation: Gaussian blur
- Boundary conditions:
  - At solid wall: $ds/dn = 0$ (extrapolate from fluid)
  - At smoke source: $s = s_{\text{source}}$
- Note: lots of particles may be better for rendering…

Heat

- Usually smoke is moving because the air is hot
  - Hot air is less dense; pressure vs. gravity ends up in buoyancy (hot air rising)
- Need to track temperature $T(x,t)$
- Evolution equation (neglecting conduction)

$$\frac{DT}{Dt} = -f_{\text{radiation}}(T) + f_{\text{source}}\left(T - T_{\text{source}}(x)\right)$$

- Boundary conditions: conduction of heat between solids and fluids
  - Extrapolate T from fluid, add source term…

Boussinesq and Buoyancy

- Density variation due to temperature or soot concentration is very small
- Use the “Boussinesq approximation”:
  fix density=constant, but add external buoyancy force in momentum equation:

$$\vec{f}_{\text{buoy}} = \begin{pmatrix} 0 \\ -\alpha \cdot s + \beta(T - T_{\text{amb}}) \\ 0 \end{pmatrix}$$

- Ambient temperature T_{amb}
- Constants α and β can be tuned
- If no soot and no temperature difference $\vec{f}_{\text{buoy}} = 0$

Open Boundaries

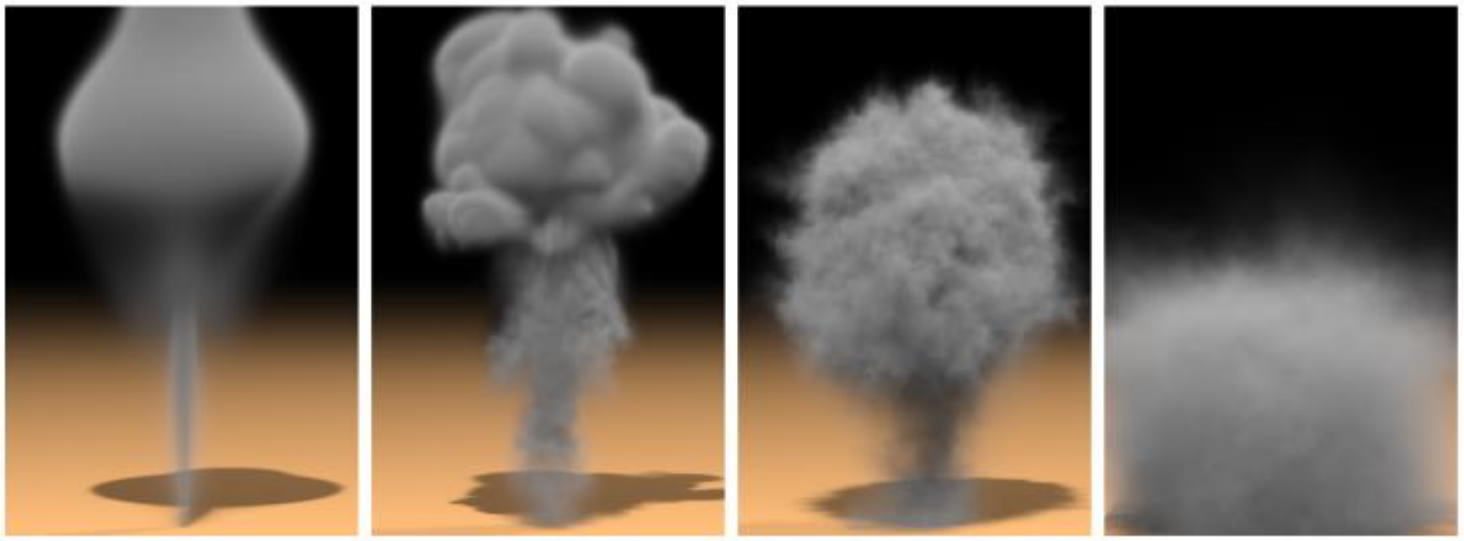
- If open air extends beyond the grid, what boundary condition?
- After making the Boussinesq approximation, simplest thing is to say the open air is a free surface ($p = 0$)
  - We let air blow in or out as it wants
- May want to explicitly zero velocity on far boundaries to avoid large currents developing

Vorticity Confinement

- vortices live on all scales and have high visual impact
- numerical dissipation disolves vortices too fast

ideas

- [Fedkiw 2001] add force that keeps vortices alive
- [Fedkiw 2005] trace vortex marker particles and couple them with forces to grid simulation



Vorticity Confinement

- vortex strength and axis of rotation can be measured with the rotation operator:

$$\vec{\omega} = \nabla \times \vec{u} = \begin{pmatrix} \partial_y w - \partial_z v \\ \partial_z u - \partial_x w \\ \partial_x v - \partial_y u \end{pmatrix}$$

$|\vec{\omega}|$ is called vorticity

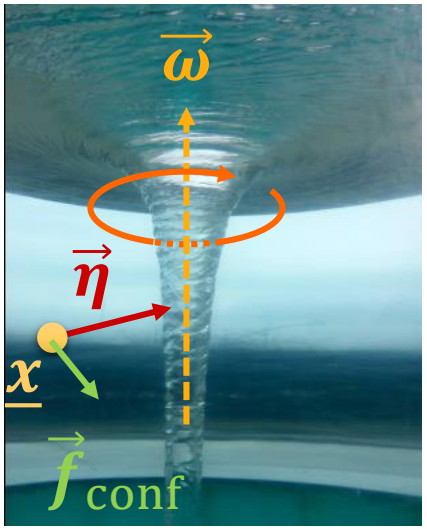
- the gradient vorticity $\vec{\eta} = \nabla|\vec{\omega}|$ points towards vortex core line, with direction

$$\hat{N} = \vec{\eta}/|\vec{\eta}|$$

- add vorticity confinement force to accelerate vortex:

ensures independence of grid resolution

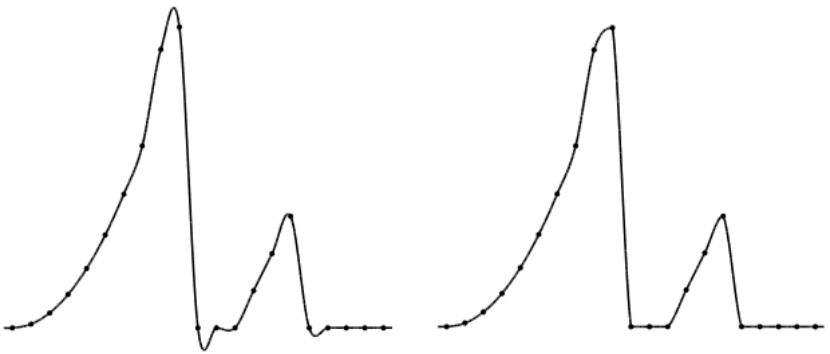
$$\vec{f}_{\text{conf}} = \epsilon \cdot \Delta x \cdot (\hat{N} \times \vec{\omega})$$



https://theawesomer.com/vortex-water-fountain/137734/

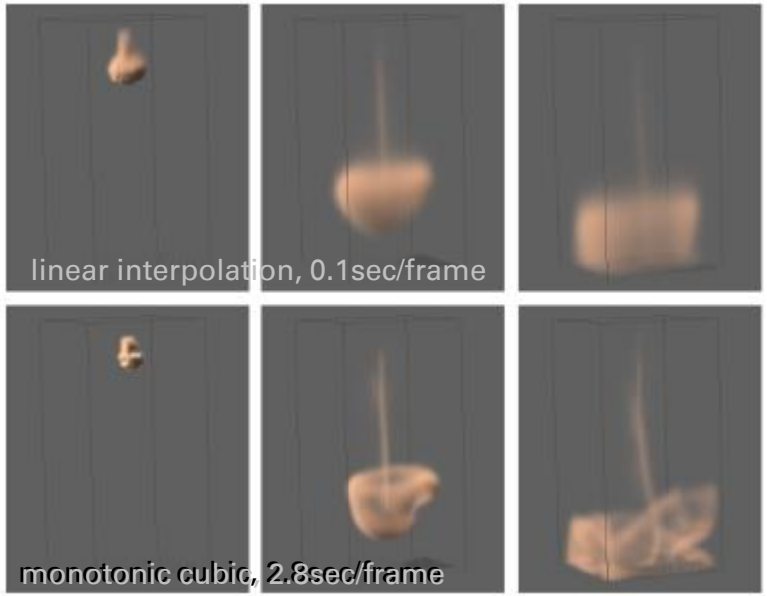
Monotonic Cubic Interpolation

- numerical dissipation of soot density and temperature counteracts the gains of vorticity confinement
- higher order interpolation over the grid fights back numerical dissipation but has the problem of overshooting
- Fedkiw [2001] propose to use a modified Hermite interpolation that is monotonic in the data.



Hermite Interpolation

monotonic Hermite Interpolation



Water

Sampling the Water-Air Interface

- The voxelization approach does not sufficiently reproduce water surfaces

Other approaches:

- Marker particles
  - sample fluid volume with particles $\{\underline{\boldsymbol{x}}_p\}$
  - define cell state to be *fluid* if it contains marker particles, *empty* or *solid* otherwise
  - move particles in incompressible grid velocity field: $\frac{d\underline{\boldsymbol{x}}_p}{dt} = \vec{\boldsymbol{u}}(\underline{\boldsymbol{x}}_p)$
  - render surface via meta balls around particles and resample resulting iso-surface on simulation grid to avoid bumpy artefacts
- Level Sets
  - represent interface by signed distance level set function $\phi(\underline{\boldsymbol{x}}) \in \boldsymbol{R}$ with interface surface where $\phi(\underline{\boldsymbol{x}}) = 0$
  - advect level set according to $\frac{D\phi}{Dt} = 0$
  - recompute distance values $|\phi(\underline{\boldsymbol{x}})| > 0$ by fast marching method

Fast Distance Function

- Felzenszwalb, P., & Huttenlocher, D. (2004). *Distance transforms of sampled functions*. Cornell University.
- Given a sampled function f on a n-D grid G, the distance Euclidean transform is given by

$$D_f(p) = \min_{q \in G} \left(\|q - p\|^2 + f(q) \right)$$

- in case of level set function set

$$f(p) = \begin{cases} 0 & \text{if } \phi(p) = 0 \\ \infty & \text{otherwise} \end{cases}$$

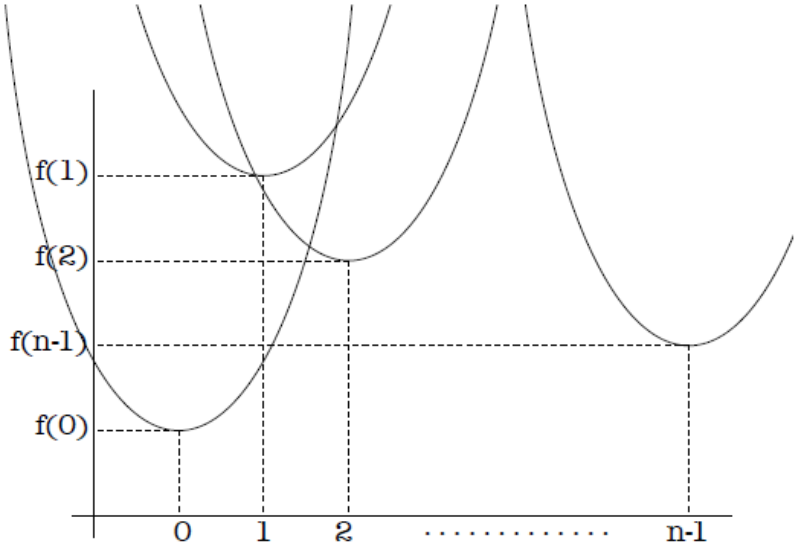
- nD-case can be reduced to (n-1)D-case, e.g. 2D case:

$$D_f(x, y) = \min_{x'} \left(\|x - x'\|^2 + D_{f|x'}(y) \right)$$

  i.e. compute 1D distance transform per column x‘ and then compute 1D distance transforms per row on result
- as 1D transform can be computed in linear time, 2D and 3D transform can also be computed in linear time

Fast Distance Function – 1D

- Felzenszwalb, P., & Huttenlocher, D. (2004). *Distance transforms of sampled functions*. Cornell University.
- Imagine for each location q a parabola $f(q) + q^2$
- Then 1D distance transform is height of lower envelop over all parabolae
- Algorithm:
  - compute lower envelop of parabolae
  - evaluate lowest envelops on grid points

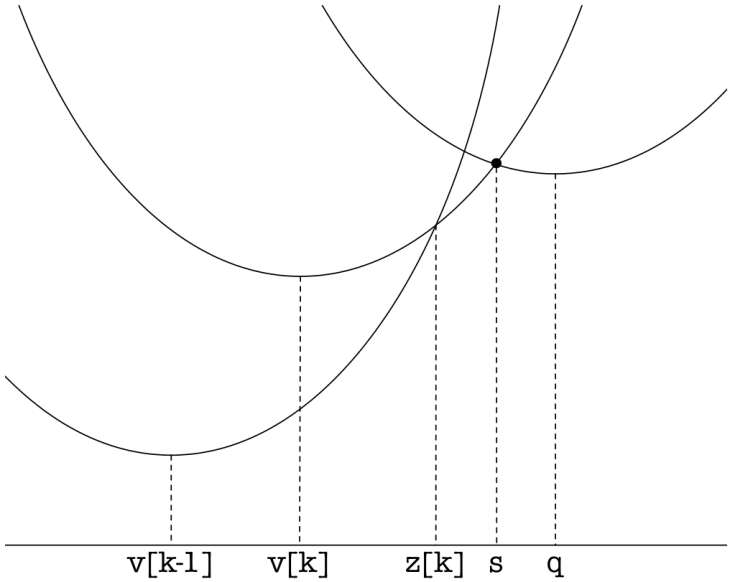


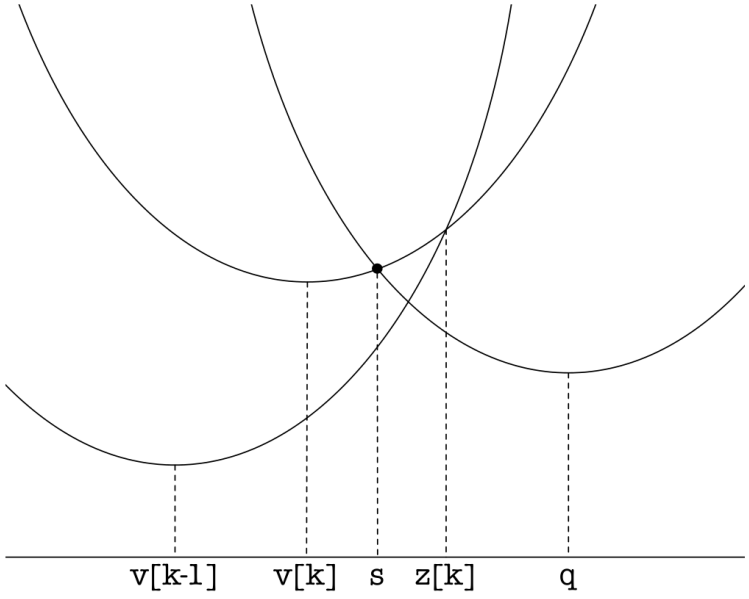
Fast Distance Function – 1D

- lower envelop is represented in list of k parabola parts
- These are stored in 2 arrays:
  - $v[i]$ … center of parabola part i
  - $z[i]$ .. $z[i+1]$ … horizontal range of parabola part i
- Two parabolas at q and p intersect in unique location:

$$s = \frac{f(p) + p^2 - (f(q) + q^2)}{2(p - q)}$$

- Envelop is built in sweep from left to right, adding one parabola at a time
- Based on relative location of intersection last parabola can be removed





Fast Distance Function – 1D

Algorithm $DT(f)$

1. $k \leftarrow 0$ (∗ Index of rightmost parabola in lower envelope ∗)
2. $v[0] \leftarrow 0$ (∗ Locations of parabolas in lower envelope ∗)
3. $z[0] \leftarrow -\infty$ (∗ Locations of boundaries between parabolas ∗)
4. $z[1] \leftarrow +\infty$
5. **for** $q = 1$ **to** $n - 1$ (∗ Compute lower envelope ∗)
6. $\quad s \leftarrow ((f(q) + q^2) - (f(v[k]) + v[k]^2))/(2q - 2v[k])$
7. $\quad$ **if** $s \leq z[k]$
8. $\quad\quad$ **then** $k \leftarrow k - 1$
9. $\quad\quad\quad$ **goto** 6
10. $\quad\quad$ **else** $k \leftarrow k + 1$
11. $\quad\quad\quad v[k] \leftarrow q$
12. $\quad\quad\quad z[k] \leftarrow s$
13. $\quad\quad\quad z[k + 1] \leftarrow +\infty$
14. $k \leftarrow 0$
15. **for** $q = 0$ **to** $n - 1$ (∗ Fill in values of distance transform ∗)
16. $\quad$ **while** $z[k + 1] < q$
17. $\quad\quad k \leftarrow k + 1$
18. $\quad \mathcal{D}_f(q) \leftarrow (q - v[k])^2 + f(v[k])$

Extrapolation

- extrapolation with signed distance

Initialization

- extract and store interface points (e.g. in redistancing)
- store for each grid point closest point on interface
- interpolate fluid quantities onto interface points

Extrapolation (1)

- copy values from closest point on interface

Extrapolation (2)

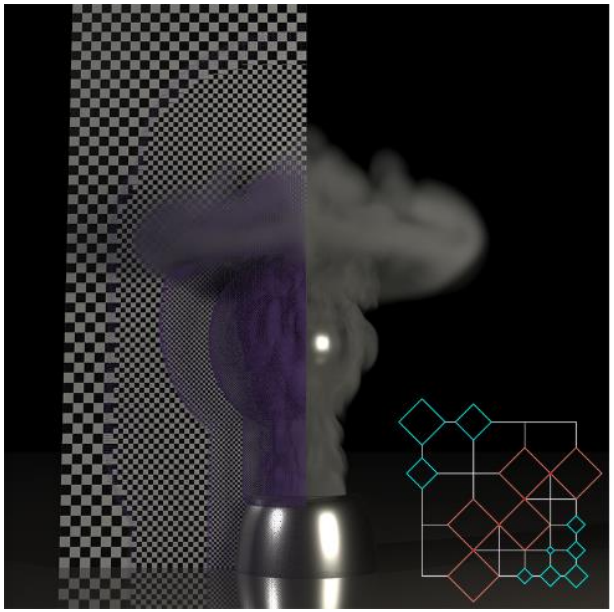
- align gradient of fluid quantities to point parallel gradient of level set function and phrase this as a PDE:

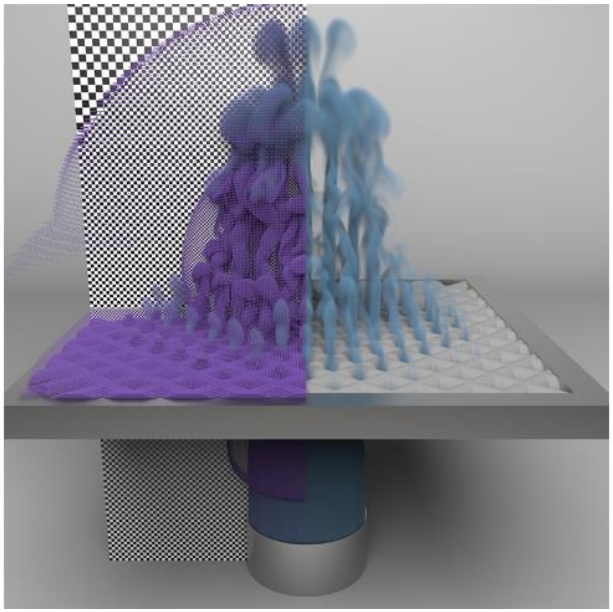
$$\nabla q \cdot \nabla \phi = 0$$

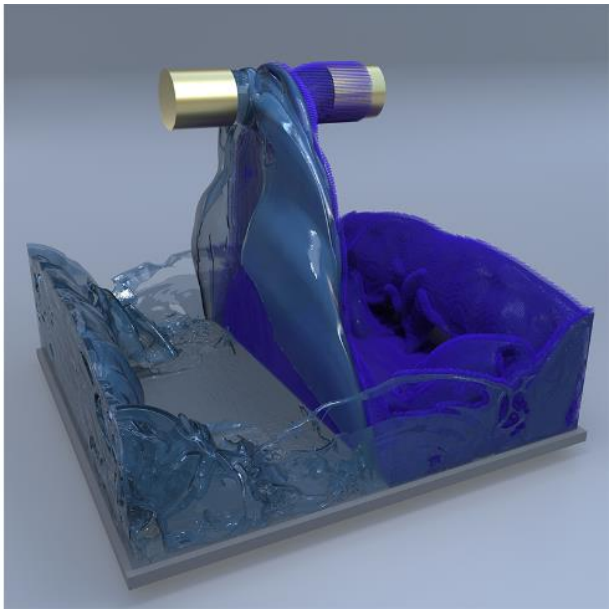
- solve the PDE with finite differences (Careful “upwind” scheme needed - want to get information from points with smaller distance, not bigger)

Adaptive Discretization

- Xiao, Yuwei, et al. "*An adaptive staggered-tilted grid for incompressible flow simulation*." *ACM Transactions on Graphics (TOG)* 39.6 (2020): 1-15.





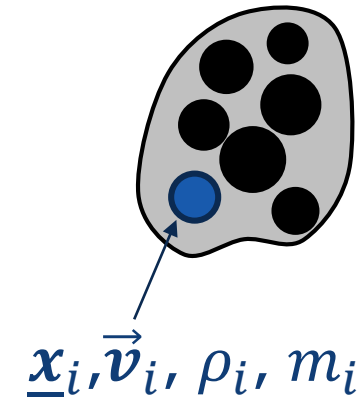


An Adaptive Staggered-Tilted Grid for Incompressible Flow Simulation

Yuwei Xiao[1] Szeyu Chan[2] Siqi Wang[3] Bo Zhu[4] Xubo Yang[1]

[1] Shanghai Jiao Tong University
[2] University of Pennsylvania
[3] New York University
[4] Dartmouth College

(With Audio)



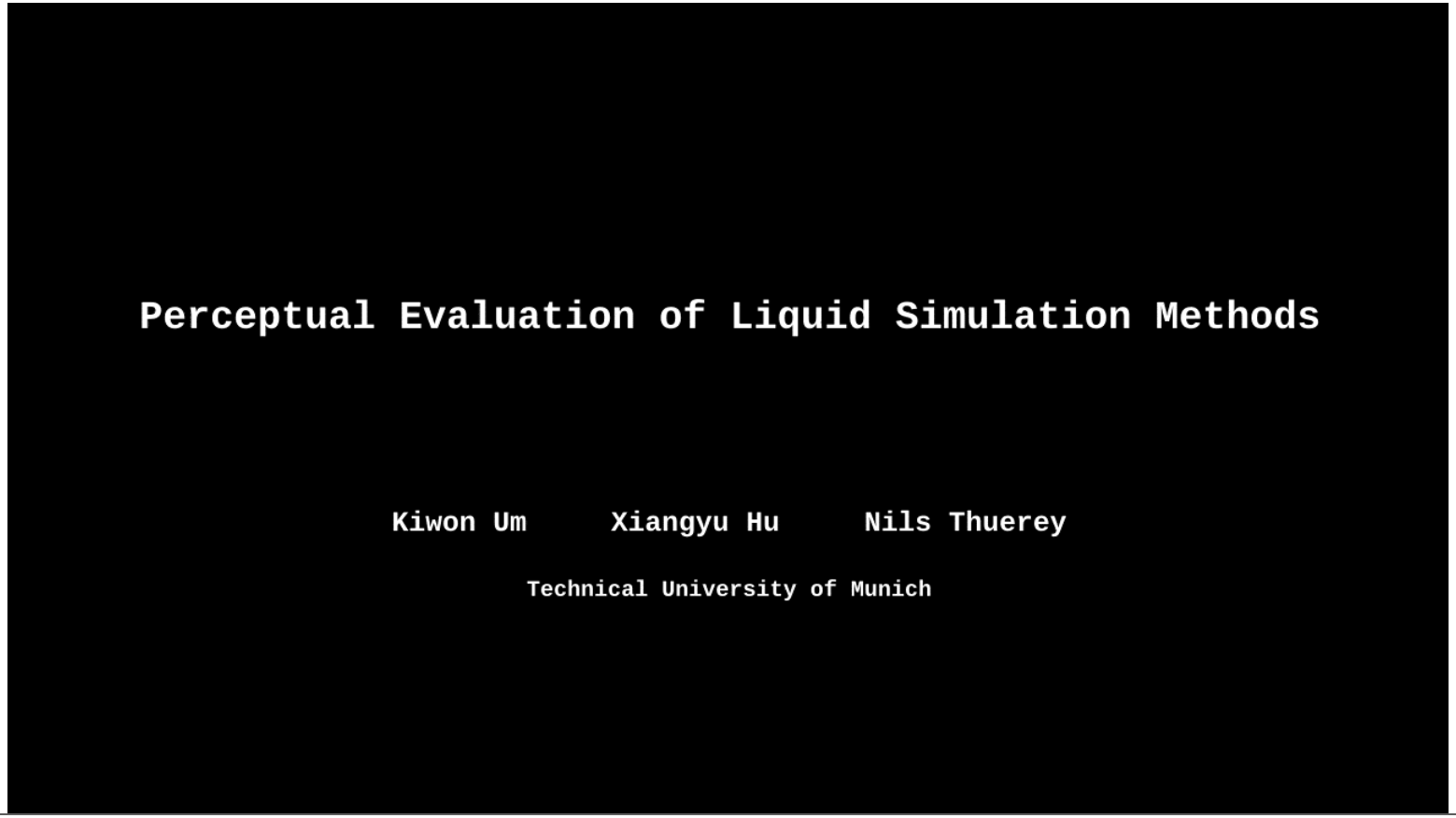
SPH-Simulation

Smoothed Particle Hydrodynamics

Ihmsen, M., Orthmann, J., Solenthaler, B., Kolb, A., & Teschner, M. SPH fluids in Computer Graphics. Eurographics STAR, 2014

Motivation

- Um, Hu and Thuerey performed perceptual comparison of fluid simulation approaches (Siggraph 2017) and found SPH approach to provide best realism at same computational speed.



SPH – Formulation

- Eulerian Momentum equation:

$$\frac{D\vec{\boldsymbol{u}}}{Dt} = \frac{\partial \vec{\boldsymbol{u}}}{\partial t} + \vec{\boldsymbol{u}} \cdot \nabla \vec{\boldsymbol{u}} = -\frac{1}{\rho}\nabla p + \nu \nabla \cdot \nabla \vec{\boldsymbol{u}} + \vec{\boldsymbol{f}}$$

- Lagragian momentum equation for particle i at location $\underline{\boldsymbol{x}}_i$ with velocity $\vec{\boldsymbol{v}}_i$, density ρ_i and where fluid has pressure p_i and external body force is $\vec{\boldsymbol{f}}_i$:

$$\frac{d\vec{\boldsymbol{v}}_i}{dt} = -\frac{1}{\rho_i}\nabla p_i + \nu \nabla \cdot \nabla \vec{\boldsymbol{v}}_i + \vec{\boldsymbol{f}}_i$$

  here we assume that particle moves with velocity $\vec{\boldsymbol{v}}_i$, what means that $\frac{d\underline{\boldsymbol{x}}_i}{dt} = \vec{\boldsymbol{v}}_i$

- mass of each particle is fixed to m_i ➔ mass preservation
- realistic kinematic viscosity for water $\nu = 10^{-6} m^2 \cdot s^{-1}$ demands for small time steps (to avoid numeric dissipation)

SPH – Interpolation

- quantity A_i at $\underline{\boldsymbol{x}}_i$ is interpolated from closest particles $\underline{\boldsymbol{x}}_j$:

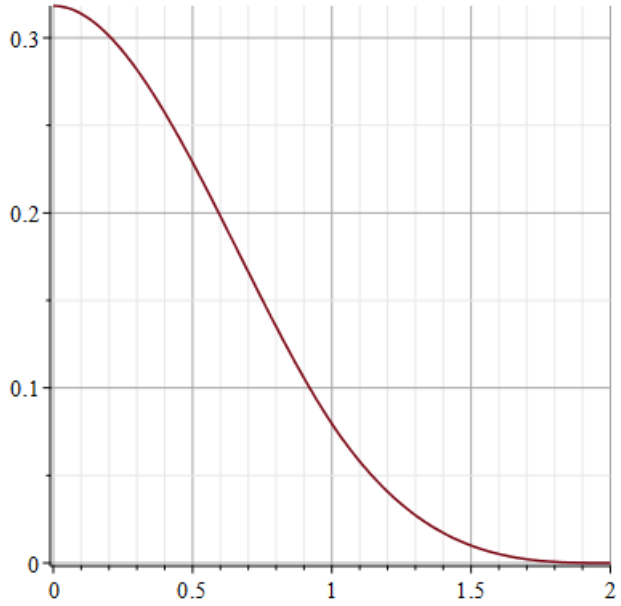
$$A_i = \sum_j \frac{m_j}{\rho_j} A_j W_{ij}\,, W_{ij} = W\left(\frac{\left\|\underline{\boldsymbol{x}}_i - \underline{\boldsymbol{x}}_j\right\|}{h}\right), W(q) = \frac{1}{h^d} f(q)$$

  with kernel W, smoothing length h and number of dimensions d.

- Typically used cubic kernel function:

$$f(q) = \frac{3}{2\pi}\begin{cases} \frac{2}{3} - q^2 + \frac{1}{2}q^3 & 0 \le q < 1 \\ \frac{1}{6}(2-q)^3 & 1 \le q < 2 \\ 0 & q \ge 2 \end{cases}$$

  with $h = 2$.



- also quintic spline used in practice
- for theory Gaussian is best but in practice problematic due to infinite support

SPH – partial derivatives

- standard approach $\nabla_i A_i = \sum_j \frac{m_j}{\rho_j} A_j \nabla_i W_{ij}$ and $\Delta_i A_i = \sum_j \frac{m_j}{\rho_j} A_j \Delta_i W_{ij}$ has numerical problems
- one rather discretizes $\rho \nabla A = \nabla(\rho A) - A \nabla \rho$ and uses the more robust approximations

$$\nabla_i A_i = \rho_i \sum_j m_j \left(\frac{A_i}{\rho_i^2} + \frac{A_j}{\rho_j^2} \right) \nabla_i W_{ij} \tag{6}$$

$$\nabla_i \cdot \vec{\boldsymbol{A}}_i = -\frac{1}{\rho_i} \sum_j m_j \vec{\boldsymbol{A}}_{ij} \cdot \nabla_i W_{ij} \tag{7}$$

$$\Delta_i A_i = 2 \sum_j \frac{m_j}{\rho_j} A_{ij} \frac{\underline{\boldsymbol{x}}_{ij} \cdot \nabla_i W_{ij}}{\underline{\boldsymbol{x}}_{ij} \cdot \underline{\boldsymbol{x}}_{ij} + 0.01 h^2} \tag{8}$$

with $\vec{\boldsymbol{A}}_{ij} = \vec{\boldsymbol{A}}_i - \vec{\boldsymbol{A}}_j, A_{ij} = A_i - A_j, \underline{\boldsymbol{x}}_{ij} = \underline{\boldsymbol{x}}_i - \underline{\boldsymbol{x}}_j$

- further variations at fluid-fluid and fluid-solid boundaries

SPH – Basic Algorithm

- In most applications fluid has near constant density around ρ_0
- define particle mass $m_i = h^3\rho_0$
- in simplest approach pressure is computed from deviation of density from ρ_0, e.g. by

$$p_i = k\left((\rho_i/\rho_0)^7 - 1\right) \quad (9)$$

with a stiffness constant k.

- time step width Δt is computed from Courant-Friedrich-Levy (CFL) condition to be

$$\Delta t \leq \frac{\lambda \cdot h}{\|\vec{v}_{\max}\|}$$

with safety factor $\lambda \approx 0.4$, which implies that a particle can move no more than $0.4h$ per time step

Algorithm 1 SPH with state equation.

for all *particle i* **do**
  find neighbors j
for all *particle i* **do**
  $\rho_i = \sum_j m_j W_{ij}$
  compute p_i using ρ_i (e.g. Eq. (9))
for all *particle i* **do**
  $\mathbf{F}_i^{pressure} = -\frac{m_i}{\rho_i}\nabla p_i$ (e.g. Eq. (6))
  $\mathbf{F}_i^{viscosity} = m_i \nu \nabla^2 \mathbf{v}_i$ (e.g. Eq. (8))
  $\mathbf{F}_i^{other} = m_i \mathbf{g}$
  $\mathbf{F}_i(t) = \mathbf{F}_i^{pressure} + \mathbf{F}_i^{viscosity} + \mathbf{F}_i^{other}$
for all *particle i* **do**
  $\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i(t) + \Delta t \mathbf{F}_i(t)/m_i$
  $\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t + \Delta t)$

Neighbor Search

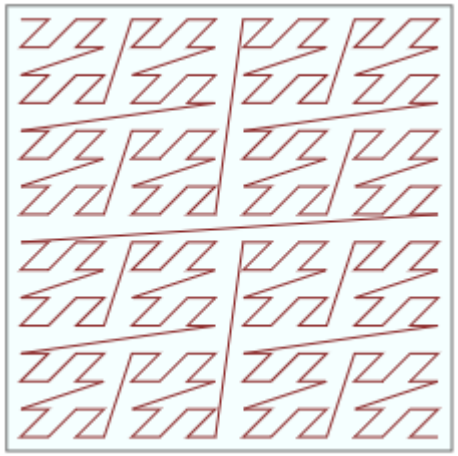
- for each particle we need to collect overlapping neighbors
- typically a regular and potentially hashed grid is used

Index Sort

- highly parallel construction of grid in 3 steps:
- for each particle compute grid cell index
- sort particles according to index
- store for each grid cell first particle in sorted list

- better locality by using Z-index Sorting of grid cells

- large number of GPU based approaches available



Z-index curve

SPH – Incompressibility Update

- similar to grid based methods, pressure can be used to make velocity field divergence free
- In SPH this is driven by density error: $\rho_{err} = \max_i |\rho_i - \rho_0|$
- simplest approach is to rely on the pressure from local density error according to (9).
- To ensure that ρ_{err} becomes lower than theshold one can repeat preasure update and either accumulate p_i, pressure force or particle displacement $\Delta \underline{\boldsymbol{x}}_i$
- Furthermore one can exploit the *splitting trick* by computing intermediate values $\underline{\boldsymbol{x}}_i^*, \vec{\boldsymbol{v}}_i^*$ and ρ_i^* without pressure force and in a second step do the pressure update.
- different choices for stiffness constant, e.g.:

$$k = \frac{\rho_i^* r_i^2}{2\rho_0 \Delta t^2}, \text{with radius } r_i \text{ from poisson equation}$$

SPH – Algorithm Variants

Computergraphik und Visualisierung

Algorithm 2 SPH with state equation and splitting.

for all *particle i* **do**
find neighbors j

for all *particle i* **do**
$\mathbf{F}_i^{viscosiy} = m_i \nu \nabla^2 \mathbf{v}_i$ (e.g. Eq. (8))
$\mathbf{F}_i^{other} = m_i \mathbf{g}$
$\mathbf{v}_i^* = \mathbf{v}_i(t) + \Delta t \frac{\mathbf{F}_i^{viscosity} + \mathbf{F}_i^{other}}{m_i}$

for all *particle i* **do**
$\rho_i^* = \sum_j m_j W_{ij} + \Delta t \sum_j (\mathbf{v}_i^* - \mathbf{v}_j^*) \cdot \nabla W_{ij}$
compute p_i using ρ_i^* (e.g. Eq. (9))

for all *particle i* **do**
$\mathbf{F}_i^{pressure} = -\frac{m_i}{\rho_i^*} \nabla p_i$ (e.g. Eq. (6))

for all *particle i* **do**
$\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i^* + \Delta t \mathbf{F}_i^{pressure} / m_i$
$\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t + \Delta t)$

Algorithm 3 Iterative EOS solver with splitting.

for all *particle i* **do**
find neighbors j

for all *particle i* **do**
$\mathbf{F}_i^{viscosity} = m_i \nu \nabla^2 \mathbf{v}_i$ (e.g. Eq. (8))
$\mathbf{F}_i^{other} = m_i \mathbf{g}$
$\mathbf{v}_i^* = \mathbf{v}_i(t) + \Delta t \frac{\mathbf{F}_i^{viscosity} + \mathbf{F}_i^{other}}{m_i}$
$\mathbf{x}_i^* = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i^*$

repeat
for all *particle i* **do**
compute ρ_i^* using $\mathbf{x}_i^*$
compute p_i using ρ_i^*, e.g. $p_i = k(\rho_i^* - \rho_0)$
compute ρ_{err}
for all *particle i* **do**
$\mathbf{F}_i^{pressure} = -\frac{m_i}{\rho_i^*} \nabla p_i$ (e.g. Eq. (6))
$\mathbf{v}_i^* = \mathbf{v}_i^* + \Delta t \frac{\mathbf{F}_i^{pressure}}{m_i}$
$\mathbf{x}_i^* = \mathbf{x}_i^* + \Delta t^2 \frac{\mathbf{F}_i^{pressure}}{m_i}$
until $\rho_{err} < \eta$
for all *particle i* **do**
$\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i^*$
$\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i^*$

SPH – Pressure Projection

- Alternatively, we can compute the pressure from a Poisson equation as in the Eulerian approach.
- There are two possible formulations:

$$\Delta p_i = \frac{\rho_0}{\Delta t} \nabla \cdot \vec{\boldsymbol{v}}_i^* \quad (14)$$

$$\Delta p_i = \frac{\rho_0 - \rho_i^*}{\Delta t^2} \quad (15)$$

- also combinations of both alternates can be used
- sparse linear solver needed

Algorithm 4 SPH with pressure projection (Inc SPH).

for all *particle i* **do**
  find neighbors j

for all *particle i* **do**
  $\mathbf{F}_i^{viscosiy} = m_i \nu \nabla^2 \mathbf{v}_i$ (e.g. Eq. (8))
  $\mathbf{F}_i^{other} = m_i \mathbf{g}$
  $\mathbf{v}_i^* = \mathbf{v}_i(t) + \Delta t \frac{\mathbf{F}_i^{viscosity} + \mathbf{F}_i^{other}}{m_i}$

for all *particle i* **do**
  $\rho_i^* = \sum_j m_j W_{ij} + \Delta t \sum_j (\mathbf{v}_i^* - \mathbf{v}_j^*) \cdot \nabla W_{ij}$

solve the PPE in Eq. (14) or Eq. (15)

for all *particle i* **do**
  $\mathbf{F}_i^{pressure} = -\frac{m_i}{\rho_i^*} \nabla p_i$ (e.g. Eq. (6))

for all *particle i* **do**
  $\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i^* + \Delta t \mathbf{F}_i^{pressure} / m_i$
  $\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t + \Delta t)$

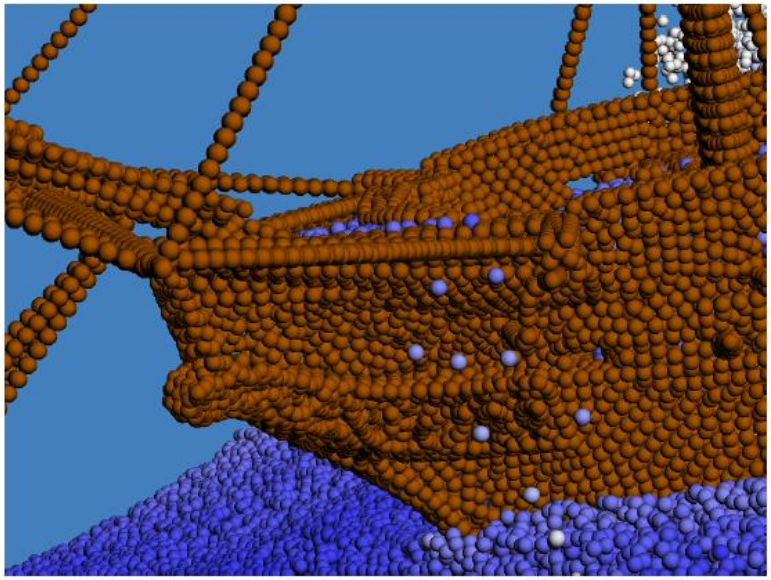
Multi-Phase Fluids

- difference in fluid densities leads to artefacts at the fluid-fluid interface (left)
- introduction of the *number density* $\delta_i = \sum_j W_{ij}$ and new interpolation formulae resolves the problem (right):

$$\tilde{\rho}_i = m_i \delta_i, \qquad \tilde{p}_i(\tilde{\rho}_i), \qquad \vec{\boldsymbol{F}}_i^{pressure} = -\sum_j \left(\frac{\tilde{p}_j}{\delta_j^2} + \frac{\tilde{p}_i}{\delta_i^2} \right) \nabla_i$$

SPH – Solid Body Coupling

- sample solids along boundary with particles



- add constraint forces that avoid interpenetration
- compute forces and torques of fluid on solids

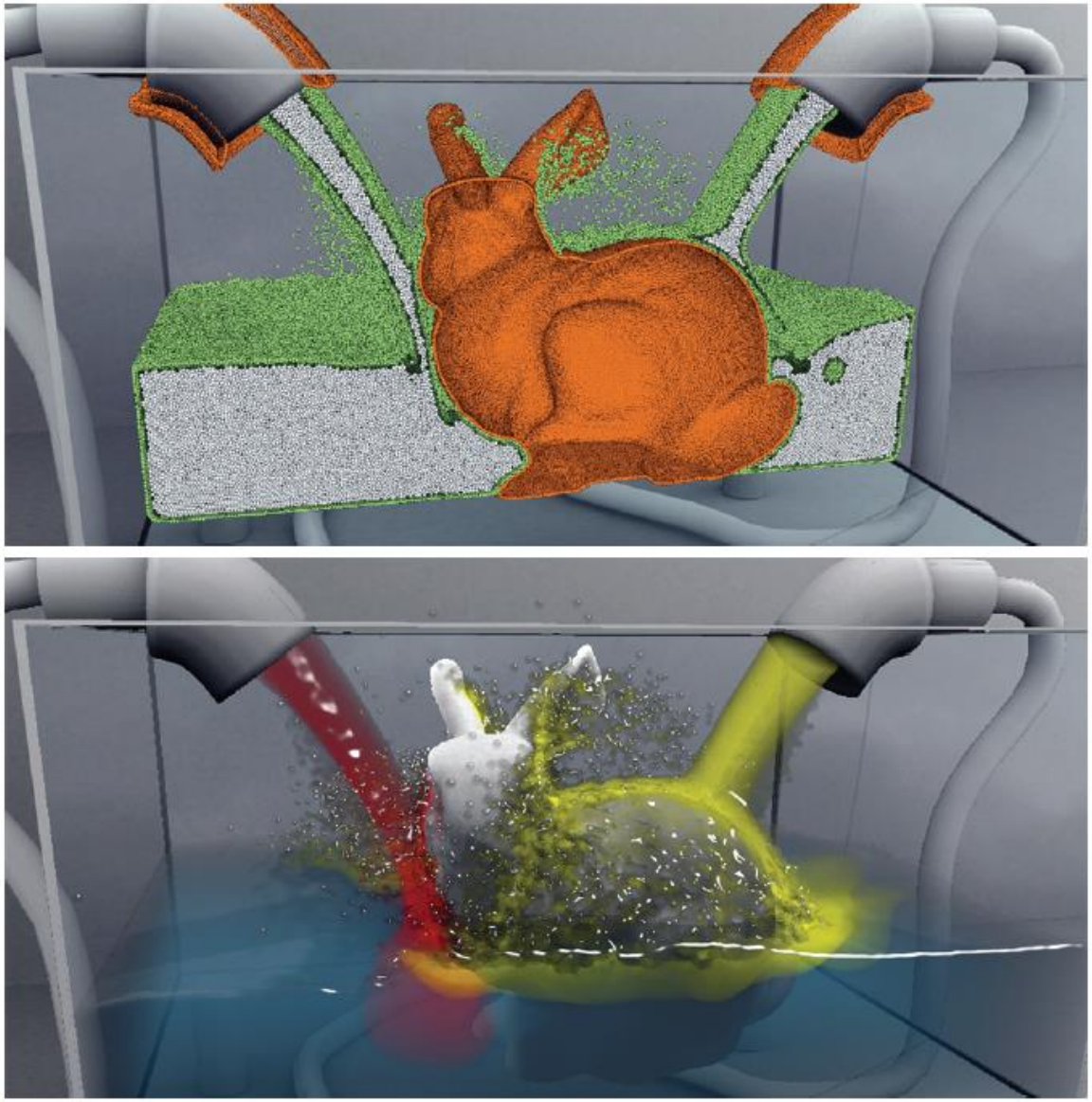
Algorithm 1 Simulation update with our boundary handling model

while *animating* **do**
  foreach *moving-rigid-body i* **do**
    synchronize boundary particles with rigid body state
  foreach *fluid-particle i* **do**
    find fluid and boundary neighbors
    activate neighboring boundary particles
  foreach *fluid-particle i* **do**
    compute density $\rho_i(t)$
    compute pressure $p_i(t)$ (e.g. WCSPH, PCISPH)
  foreach *fluid-particle i* **do**
    add fluid forces $\mathbf{F}_i^{p,\nu,c,ext}(t)$
    add forces exerted by boundary particles $\mathbf{F}_{f_i}^{total}$
  foreach *active-boundary-particle i* **do**
    add forces exerted by fluid particles $\mathbf{F}_{b_i}^{total}$
  foreach *rigid-body i* **do**
    compute the total force exerted by fluids $\mathbf{F}_{rigid_i}$
    compute the total torque exerted by fluids τ_{rigid_i}
  foreach *fluid-particle i* **do**
    update $\mathbf{x}_i, \mathbf{v}_i$
  update rigid bodies (e.g. Bullet)
end while

Akinci, N., Ihmsen, M., Akinci, G., Solenthaler, B., & Teschner, M. (2012). Versatile rigid-fluid coupling for incompressible SPH. *ACM Transactions on Graphics (TOG)*, *31*(4), 62.

SPH – Rendering

- define iso-surface from particles
- render with volume rendering techniques
- can be implemented on GPU (see scivis lecture on particle visualization)



Versatile rigid-fluid coupling for incompressible SPH

Infinite continuous adaptivity for incompressible SPH (SIGGRAPH 2017)