Fluid Simulation

Following: SiGGraph-DeformableModels_coursenotes08

SiGGraph-Fluid Simulation_coursenotes06

By Toni Susín (2015)

Navier-Stokes Equations

- The origins of the well established **Navier-Stokes** equations reach back to Isaac **Newton**, who, around 1700, formulated the basic equations for the theoretical description of fluids. These were used by L. **Euler** half a century later to develop the basic equations for momentum conservation and pressure.
- Amongst others, Louis M. H. **Navier** continued to work on the fluid mechanic equations at the end of the 18th century, as did Georg G. **Stokes** several years later. He was one of the first to analytically solve fluid problems for viscous media. The **NS equations** could not be practically used up to the middle of the 20th century, when the **numerical methods**, that are necessary to solve the resulting equations, were developed.

Navier-Stokes Equations

$$\underbrace{\rho\left(\frac{\partial \mathbf{u}}{\partial t}+\mathbf{u}\cdot\nabla\mathbf{u}\right)}_{\text{advection}}+\underbrace{\nabla P}_{\text{pressure}}=\underbrace{\nu\triangle\mathbf{u}}_{\text{viscosity}}+\underbrace{\rho\mathbf{g}}_{\text{forces}}$$

u fluid **velocity**

P fluid **pressure**

ν fluid **viscosity**

ρ fluid **density**

g **gravity**

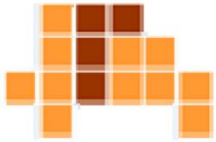
Navier-Stokes Equations

Two main solution approaches

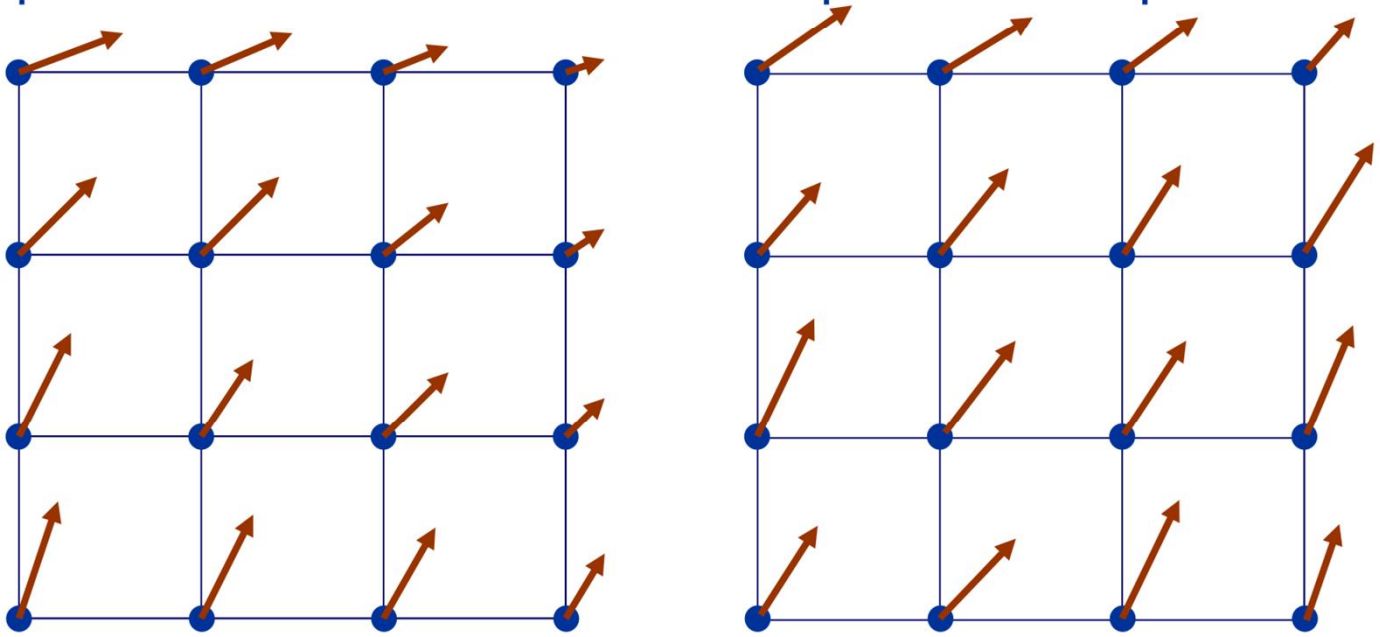
- **Grid based solution** (Eulerian Methods)
  - Mark and Cell
  - Finite Difference
  - Lattice-Boltzman
  - Finite Elements

Navier-Stokes Equations

Eulerian Approach



- quantities are considered at fixed positions in space



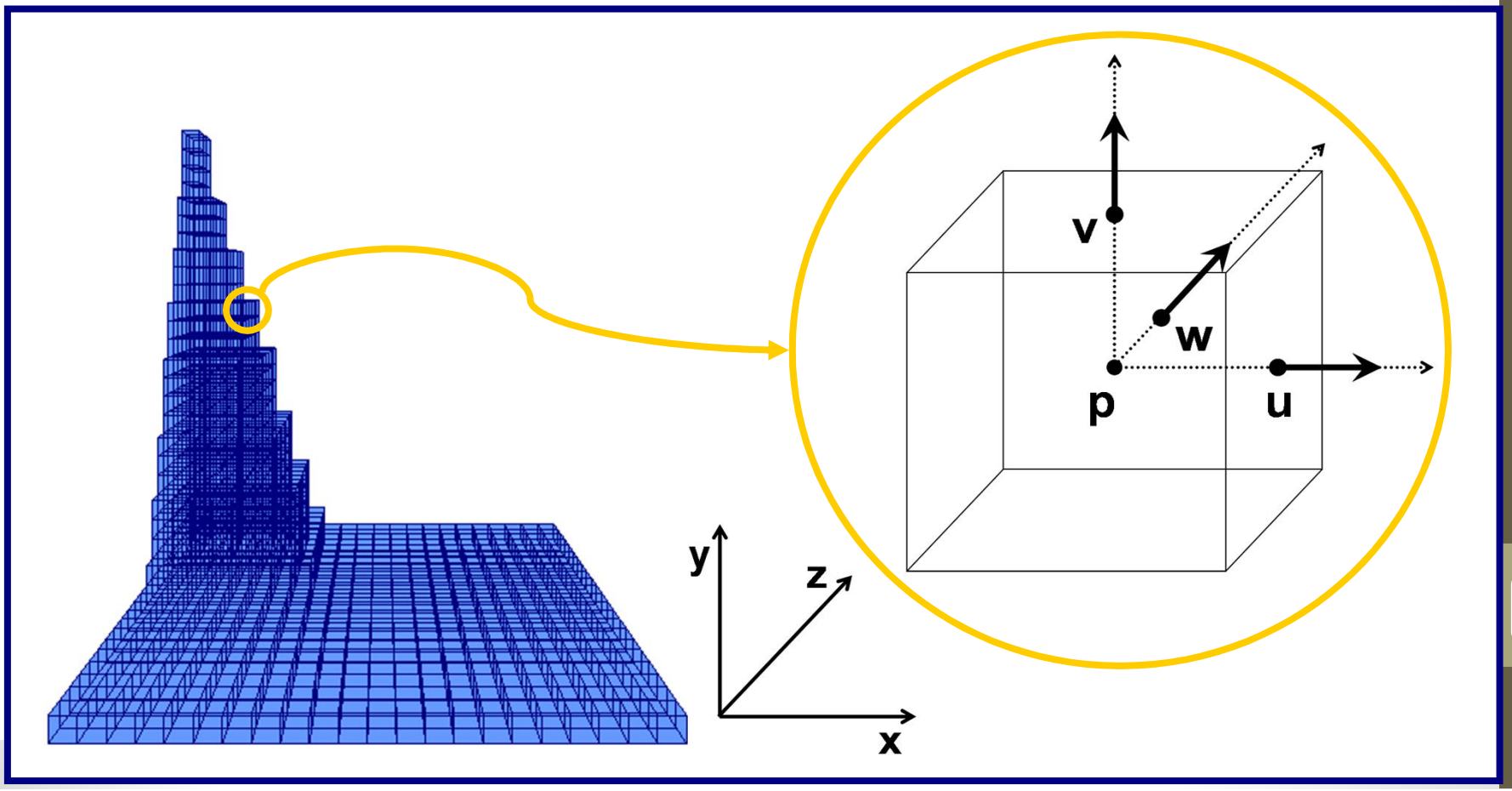
$\mathbf{v}\,(x,y,z,t)$ $\rho\,(x,y,z,t)$

$\mathbf{v}\,(x,y,z,t+\Delta t)$ $\rho\,(x,y,z,t+\Delta t)$

Navier-Stokes Equations

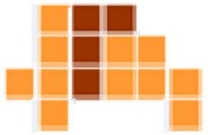
Two main solution approaches

- **Grid based solution** (Eulerian Methods)

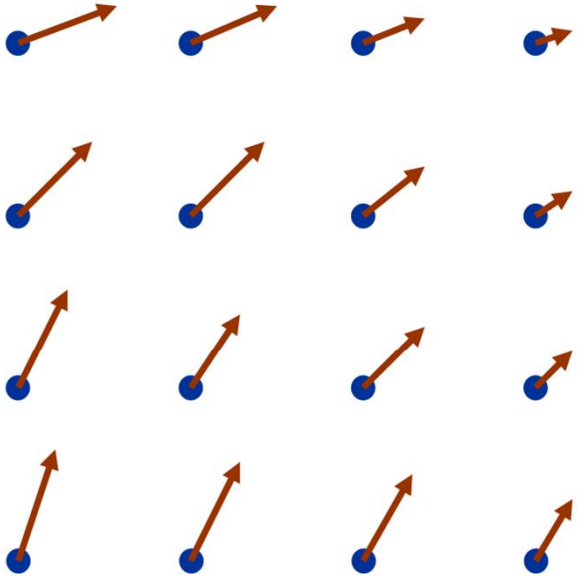


Navier-Stokes Equations

Lagrangian Approach

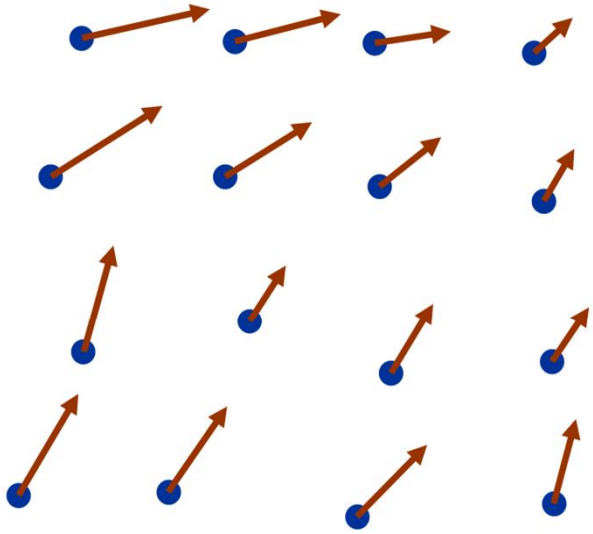


- quantities move with the flow (particle system)
- SPH is Lagrangian



$\mathbf{v}\,(x,y,z,t)$

$\rho\,(x,y,z,t)$



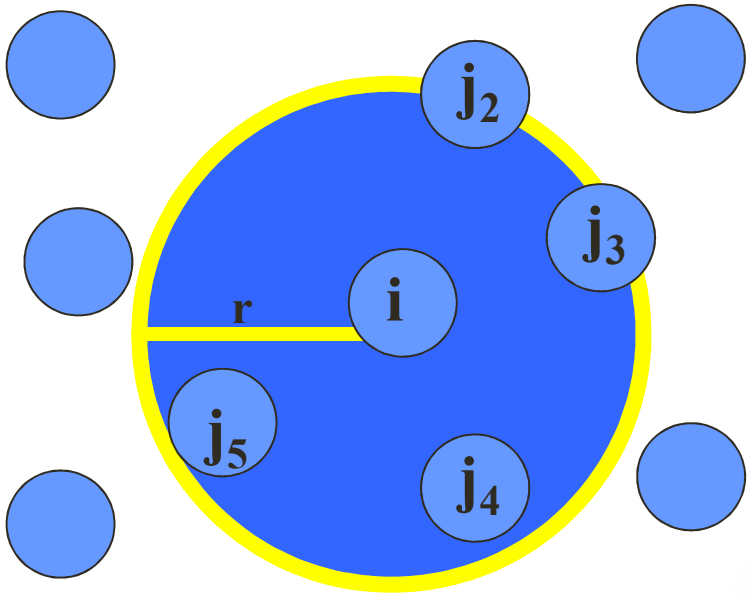
$\mathbf{v}\,(x+\Delta t\cdot u,\ y+\Delta t\cdot v,\ z+\Delta t\cdot w,\ t+\Delta t)$

$\rho\,(x+\Delta t\cdot u,\ y+\Delta t\cdot v,\ z+\Delta t\cdot w,\ t+\Delta t)$

Navier-Stokes Equations

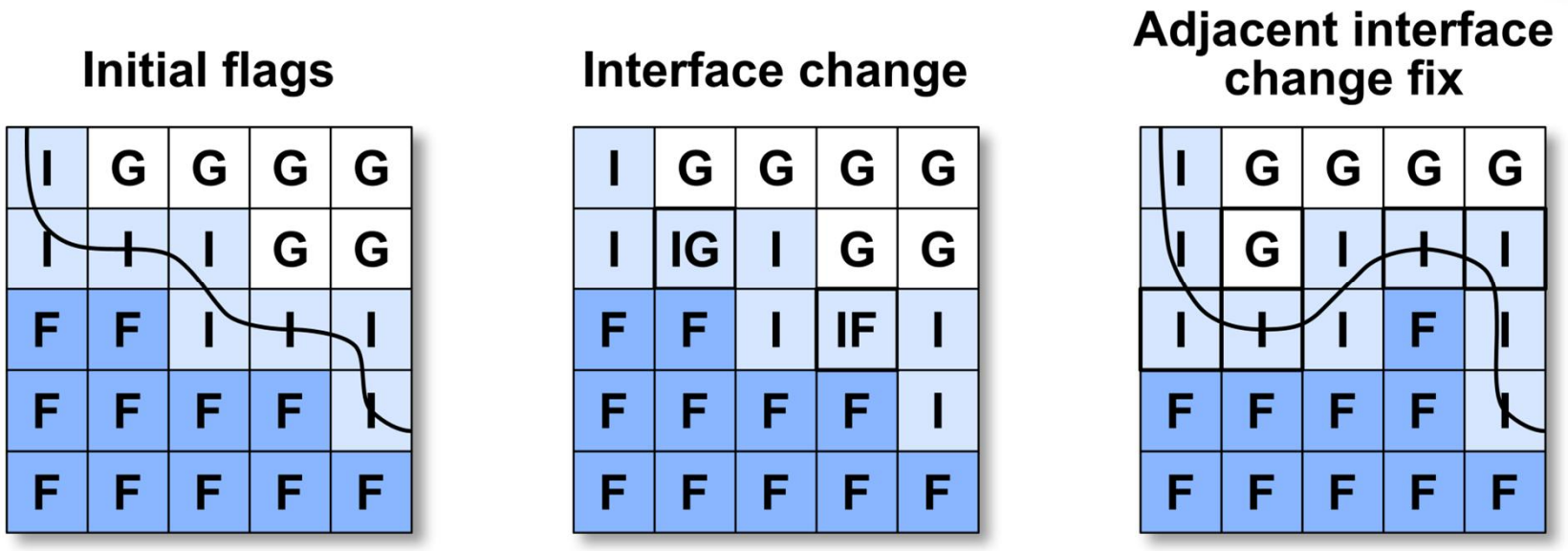
Two main solution approaches

- **Grid based solution** (Eulerian Methods)
- **Particle based solution** (Lagrangian Methods)
  - **SPH** methods



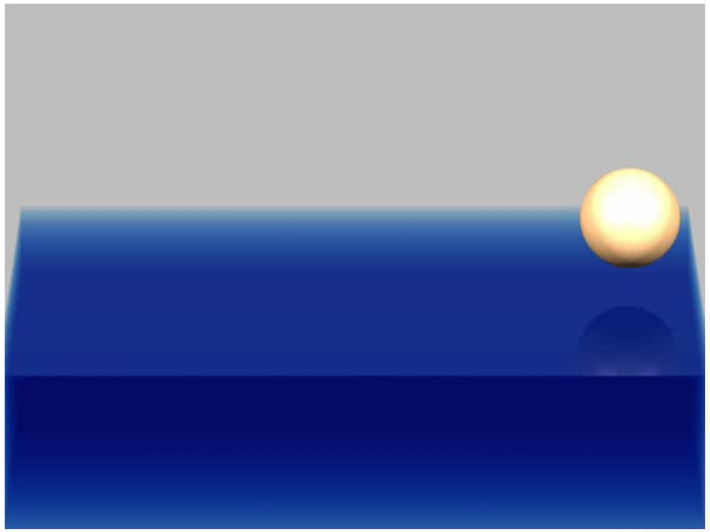
Eulerian Grid vs. Lagrangian Particles

- Volume of fluid.
- Free Surface tracking
- Interaction with other objects (buoyancy)



Exemples

- Jos Stam (2001) : Demo
- Foster (2004)

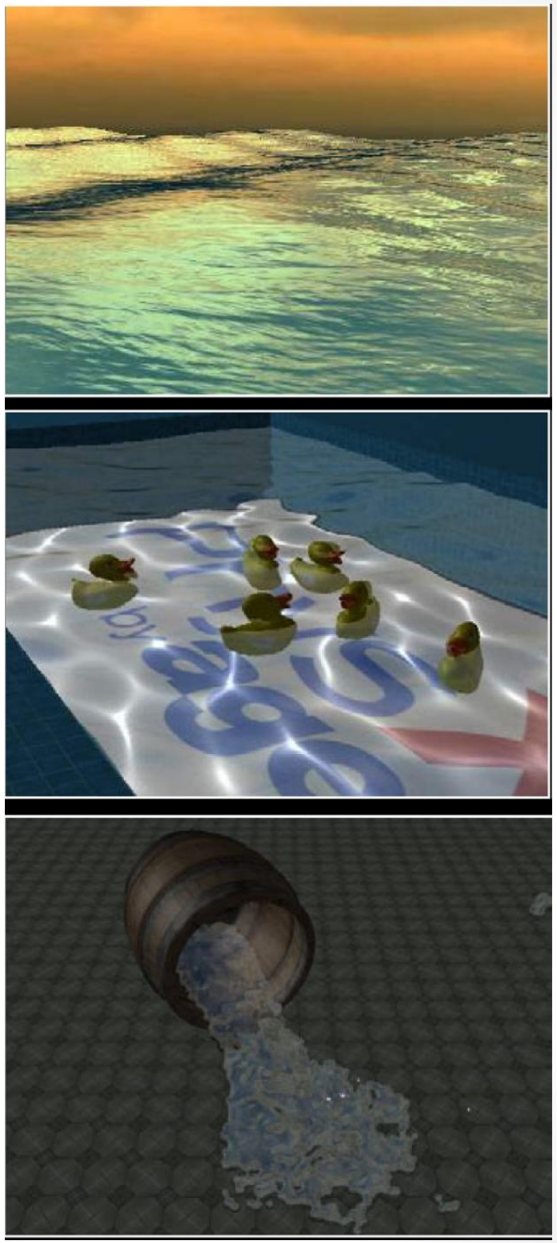


Fluids in Games

Following: SiGGraph-Fluid Simulation_coursenotes06

Fluids in Games

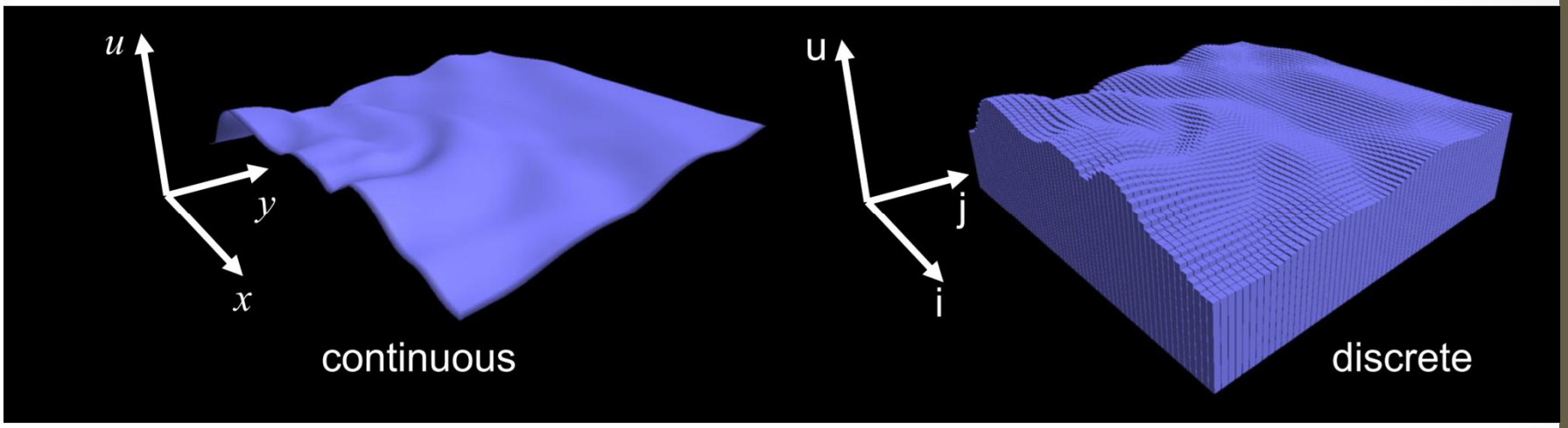
- **Procedural Water:**
  - Unbounded surfaces, Oceans
- **Height Fields Fluids:**
  - Ponds, Lakes
- **Particle systems:**
  - Splashing, spray, smoke



Procedural Fluids

- Simulate the **effect** no the cause
- No limits to the **creativity**
  - E.g. Superimpose sine waves
- **Difficult** but not impossible

Height Fields (2.5dim)



- Represent the fluid **surface** as a 2D function u(x,y)
- Pro: **reduce** from 3D to 2D
- Cons: One value per (x,y), therefore **NO breaking waves**

Height Fields

- A trivial algorithm with impressive results! Try this at home!
- Initialize `u[i,j]` with some interesting function
- Initialze `v[i,j]=0`

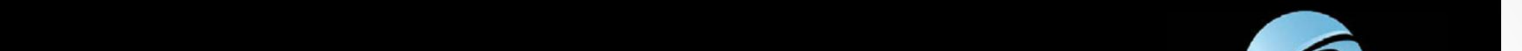
```
loop
   v[i,j] +=(u[i-1,j] + u[i+1,j] + u[i,j-1] + u[i,j+1])/4 - u[i,j]
   v[i,j] *= 0.99
   u[i,j] += v[i,j]
endloop
```

- Clamp on boundary e.g. def. `u[-1,j] = u[0,j]`

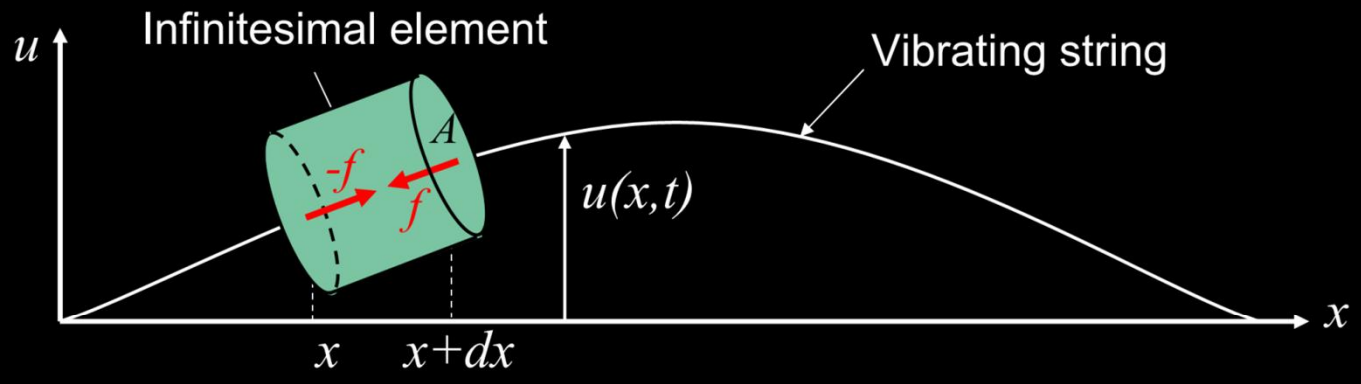
The Physics Behind it

- We model the water surface as an elastic membrane with low stiffness [Jeffrey02]
- Fairly good approximation
- Better: Derive a more complex surface model from the Navier Stokes Equations [Thuerey07]
- Most games use procedural water today
- Membrane model is an improvement and often sufficient for games.

The Physics Behind It

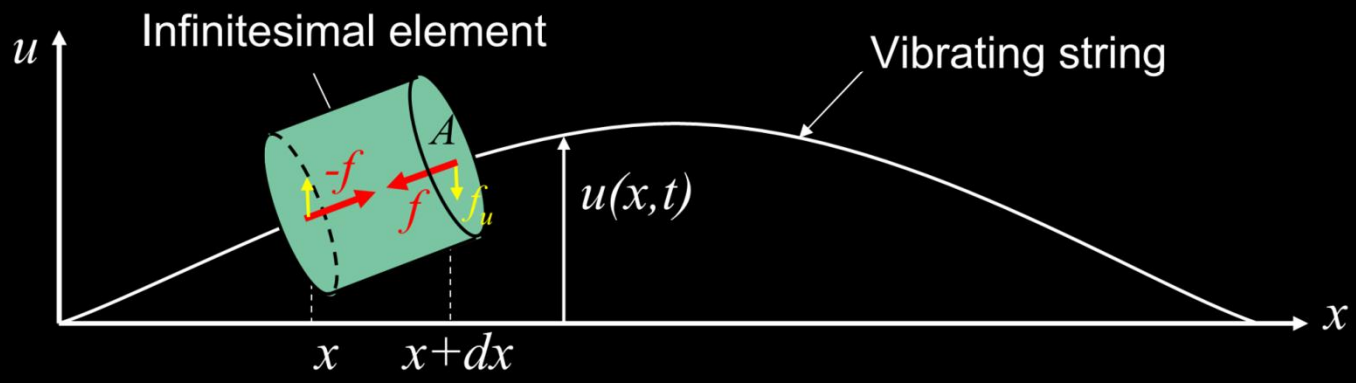


- A one dimensional membrane is a string



- $u(x,t)$ displacement normal to x-axis at time t
- Assuming small displacements and constant stress σ
- Force acting normal to cross section A is $f = \sigma A$

PDE for the 1D String



- Component of $f = \sigma A$ in u-direction:
  $f_u \approx u_x \sigma A$ where u_x is derivative of u w.r.t. x
- Newton's 2nd law for an infinitesimal segment

$$(\rho A dx) u_{tt} = \sigma A u_x|_{x+dx} - \sigma A u_x|_x \rightarrow \boxed{\rho u_{tt} = \sigma u_{xx}}$$

The 1D Wave Equation

- For the string: $\rho\, u_{tt} = \sigma\, u_{xx}$
- Standard form: $u_{tt} = c^2\, u_{xx}$, where $c^2 = \sigma/\rho$
- Solution: $u(x,t) = a \cdot f(x + ct) + b \cdot f(x - ct)$ for any function f.
- Thus, c is the speed at which waves travel

The 2D Wave Equation

- The wave equation generalizes to 2D as

$$u_{tt} = c^2 (u_{xx} + u_{yy})$$

$$u_{tt} = c^2 \nabla^2 u$$

$$u_{tt} = c^2 \Delta u$$

Discretization

- Replace the 2nd order PDE by two first order PDEs

$$u_t = v$$
$$v_t = c^2 (u_{xx} + u_{yy})$$

- Discretize in space and time
  (semi-implicit Euler, time step Δt, grid spacing h)

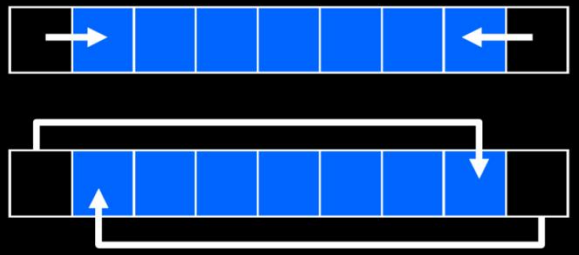
```
v^{t+1}[i,j] = v^t[i,j]
               +Δtc²(u[i+1,j]+u[i-1,j]+u[i,j+1]+u[i,j-1]-4u[i,j])/h²
u^{t+1}[i,j] = u^t[i,j] +Δt v^{t+1}[i,j]
```

- We are where we started! (correct scaling, no damping)

Remarks on Heightfields

SIGGRAPH2007

- The simulation is only conditionally stable
  - Stability condition: $\Delta t < h/c$
- Boundary conditions needed

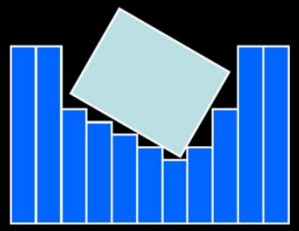


Clamp: Reflection

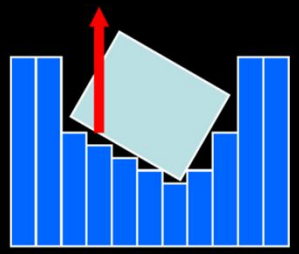
Periodic: Wrap around

Object Interaction

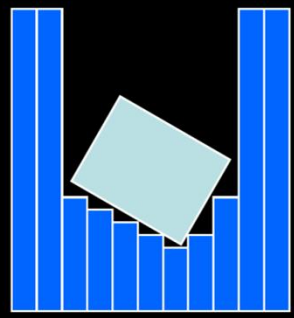
- Object → Water
  - Object pushes bars beneath it down
  - Add the removed water in the vicinity!



- Water → Object
  - Each bar below the object applies force $\mathbf{f} = -\Delta u\ \rho\ h^2\ \mathbf{g}$ to body at its location
  - Δu is the height replaced by the body, ρ water density, $\mathbf{g}$ gravity



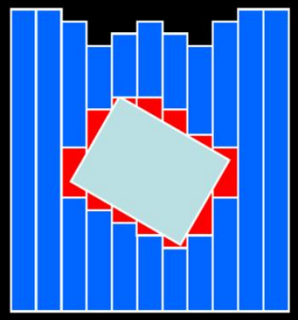
Fully Immersed Bodies

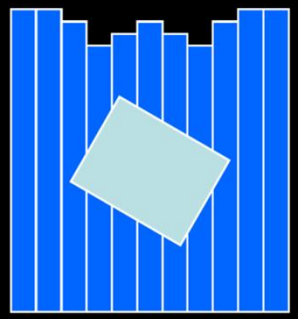


- Body below water surface
- Hole appears above the body
- Non-physical

- See story of divided sea

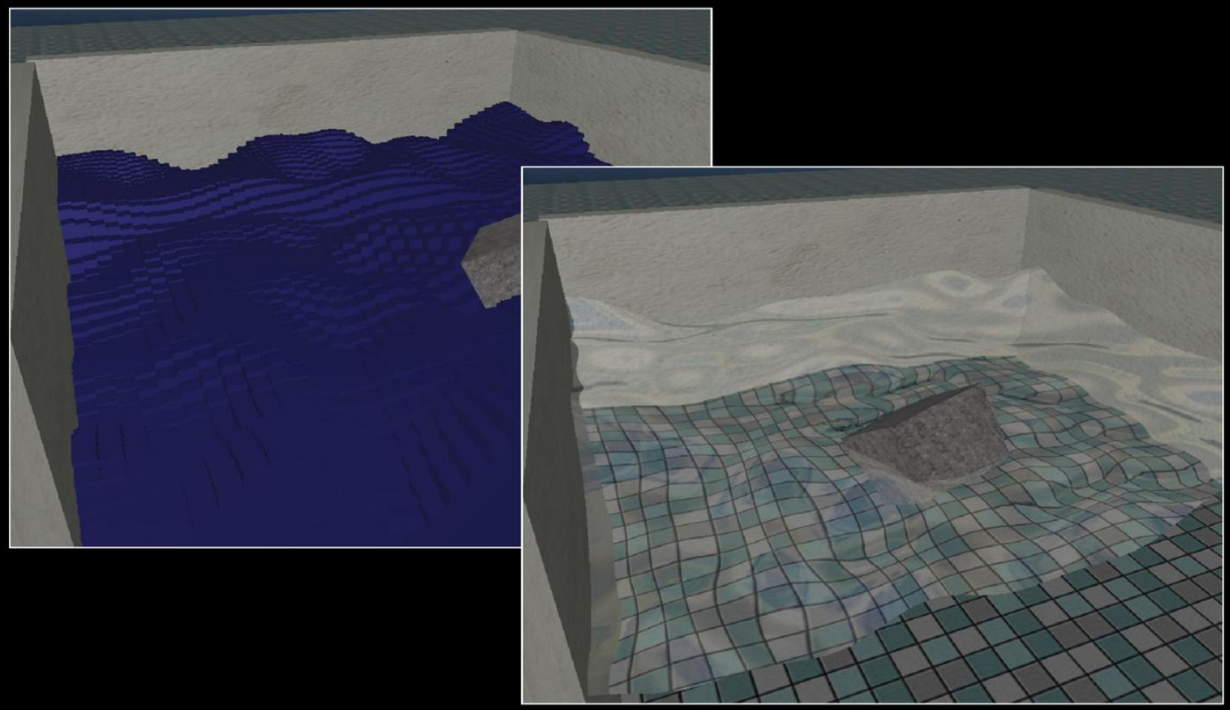
Solution



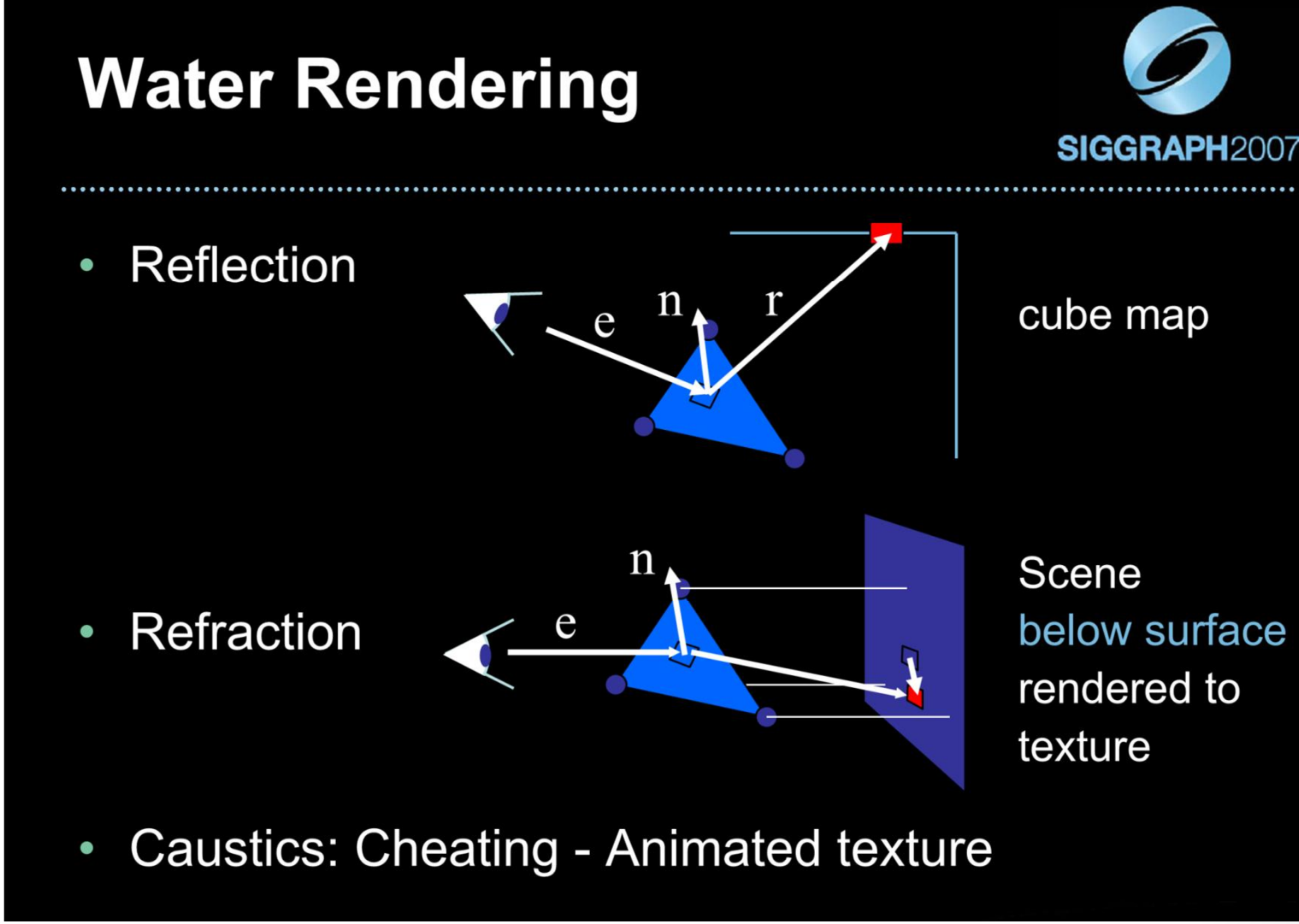


- New state variable `r[i,j]`:
  - Each column stores the part `r[i,j]` of `u[i,j]` currently replaced by solids
- At each time step:
  - `u[i,j]` is not modified directly
  - Δ `r[i,j]` = r^t`[i,j]`-r^{t-1}`[i,j]` is distributed as water `u` to the neighboring columns
  - In case of a negative difference water is removed

From Bars to Water



Water Rendering



- Reflection

cube map

- Refraction

Scene below surface rendered to texture

- Caustics: Cheating - Animated texture

Breaking Waves

- Heightfields cannot capture breaking waves
- Identify steep wave fronts
- Construct and track line along front
- Emit patch of connected particles
- Generate mesh with given thickness for rendering (plus particles for foam)
- See [Thuerey07]

Smoothed Particle Hydrodynamics (SPH)

Following:

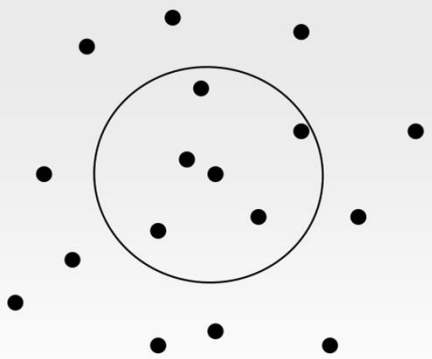
Simulation in Computer Graphics (University of Freiburg)

Smoothed Particle Hydrodynamics (SPH)

Some particle properties are determined by taking an average over neighboring particles

The fluid is represented by a particle system

Fluid dynamics



1. Only particles inside circle contribute to the average
2. Close particles should contribute more than distant particles

In the average: Use a weight function

Particle system

Before we consider the details…

How do we describe our particle system?

Each particle is specified by a state list:

mass, velocity, position, force,

density, pressure, color

Particle i ⟶ $(m_i, \mathbf{v}_i, r_i, \mathbf{F}_i, \rho_i, p_i, C_i)$

Particle system

- Simulation:

The acceleration of a particle is

$$\frac{dv_i}{dt} = a_i^{pressure} + a_i^{viscosity} + a_i^{interactive} + a_i^{gravity}$$

Remember that $a_i = \frac{F_i}{m_i}$

Let us now learn how to set up the particle list…

Particle system

- Simulation:

In our simulation we choose to have the same mass for all particles, $m_i = m$

The mass m is calculated by

$$m = \frac{(Density\ of\ \ fluid)\cdot(Total\ \ volume)}{Total\ \ number\ of\ \ particles}$$

Note! Do not change the mass during the simulation

Particle system

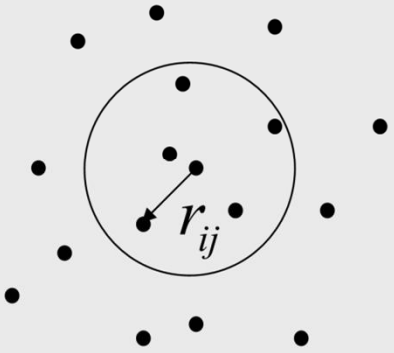
- Simulation:

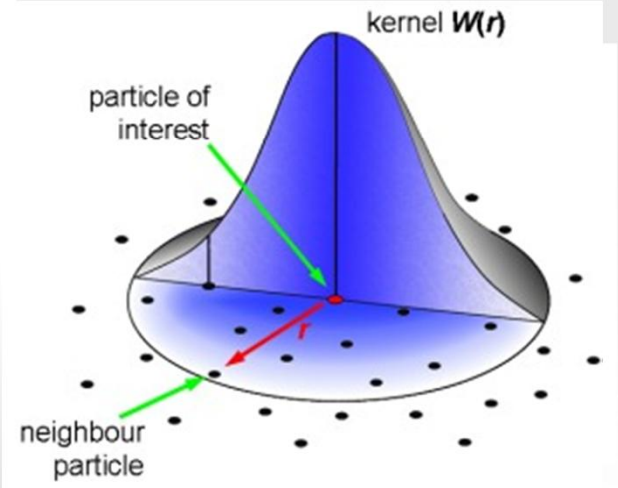
How do we determine the density of a particle?

$$\rho_i = \sum_j m_j W(r_{ij})$$

The sum include particle i

Weight function or Kernel function





What happens if particle *i* has no neighbours?

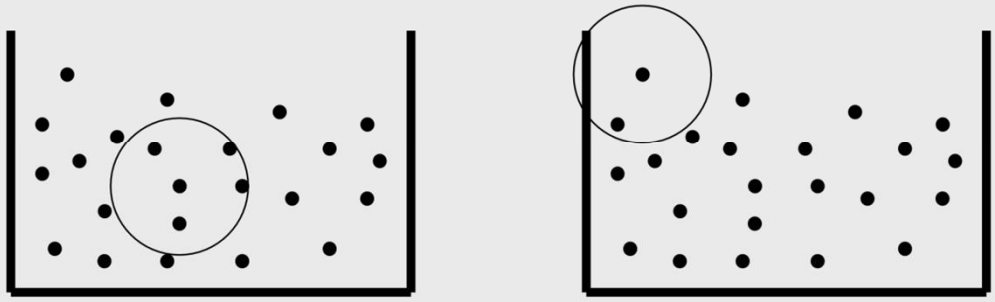
- The density calculation is done every time step
- The neighbor list must be updated every time step

Particle system

- Simulation:

Surface tracking

It is not trivial to know where the surface is…



- We can find the surface by monitoring the density
- If the density at a particle deviates too much compared to expected density we tag it as a surface particle

Particle system

- Simulation:

Pressure

We get the pressure from the relation:

$$p_i = c_s^{\,2}\left(\rho_i - \rho_0\right)$$

where c_s is the speed of sound and ρ_0 is the fluid reference density

Averaging:

In SPH we formally define averages in the following way:

$$\langle A(r)\rangle = \int_V A(r')W(r-r')dr'$$

In practice we use a discrete version of this:

$$\langle A\rangle_i \approx \sum_j \frac{m_j}{\rho_j} A_j W(r_{ij})$$

$$\langle \nabla A\rangle_i \approx \sum_j \frac{m_j}{\rho_j} A_j \nabla W(r_{ij})$$

$$\langle \nabla^2 A\rangle_i \approx \sum_j \frac{m_j}{\rho_j} A_j \nabla^2 W(r_{ij})$$

Example

$$\langle \rho\rangle_i \approx \sum_j \frac{m_j}{\rho_j} \rho_j W(r_{ij})$$

$$\approx \sum_j m_j W(r_{ij})$$

Meshless method!!

Navier-Stokes:

$$\frac{d\mathbf{v}}{dt} = -\frac{1}{\rho}\nabla p + \frac{\mu}{\rho}\nabla \cdot \nabla \mathbf{v} + \frac{f_{ext}}{m} + \mathbf{g}$$

Note: f_{ext} could for example be an interactive force

- Each term contributes as a force:

$$\frac{dv_i}{dt} = a_i^{pressure} + a_i^{viscosity} + a_i^{interactive} + a_i^{gravity}$$

Compute forces:

- First term (pressure) becomes:

$$\left\langle -\frac{1}{\rho}\nabla p \right\rangle_i \approx \sum_j P_{ij} \nabla W(r_{ij})$$

where $$P_{ij} = -m_j \left(\frac{p_i}{\rho_i^{\,2}} + \frac{p_j}{\rho_j^{\,2}} \right)$$

Compute forces:

- The second term (viscosity):

$$\left\langle \frac{\mu}{\rho} \nabla \cdot \nabla \mathbf{v} \right\rangle_i \approx \sum_j \mathbf{V}_{ij} \nabla^2 W(r_{ij})$$

where $$\mathbf{V}_{ij} = \mu m_j \frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_i \rho_j}$$

Note: $\nabla \cdot \nabla \mathbf{v} = \nabla^2 \mathbf{v}$

Summary

The acceleration of a particle can now be written:

$$\frac{dv_i}{dt} = a_i^{pressure} + a_i^{viscosity} + a_i^{interactive} + a_i^{gravity}$$

$$a_i^{pressure} \approx \sum_j P_{ij} \nabla W(r_{ij})$$

$$a_i^{viscosity} \approx \sum_j V_{ij} \nabla^2 W(r_{ij})$$

$$a_i^{interactive} \approx \frac{1}{m_i} f_i^{interactive}$$

$$a_i^{gravity} \approx (0,0,-g)$$

Remember that

$$F_i = m_i a_i$$

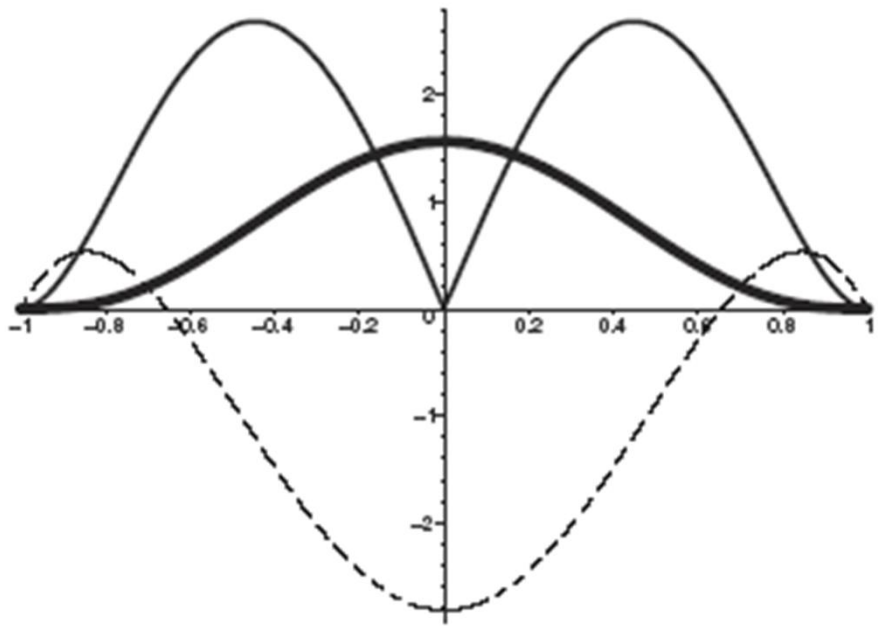
NOTE!

Sometimes one uses different kernels for each term in RHS

Smoothing Kernel -2

$$W_{\text{poly6}}(\mathbf{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3 & 0 \le r \le h \\ 0 & \text{otherwise} \end{cases}$$

- all points inside a radius of 'h' are considered for "smoothing".
- Thick line: the kernel
- Thin line: the gradient of kernel
- Dashed line: the laplacian of kernel



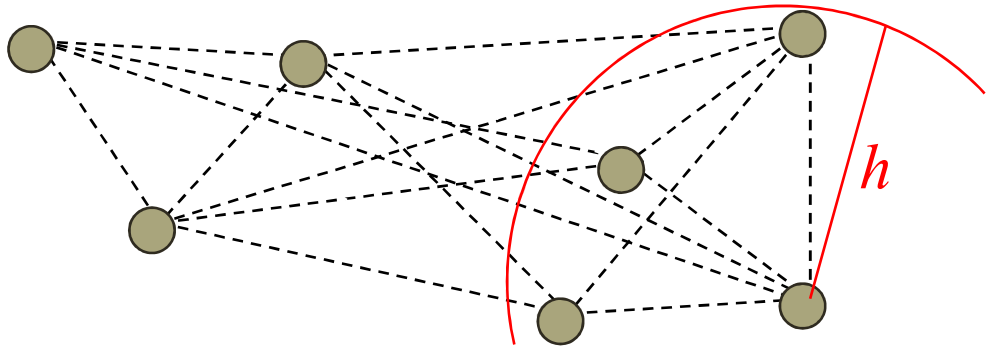
Smoothing Kernel -2

$$W_{\text{poly6}}(\mathbf{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3 & 0 \le r \le h \\ 0 & \text{otherwise} \end{cases}$$

$$\nabla W_{poly6}(\vec{r}, h) = \begin{cases} -\frac{945}{32\pi h^9}(h^2 - r^2)^2 r & 0 \le r \le h \\ 0 & otherwise \end{cases}$$

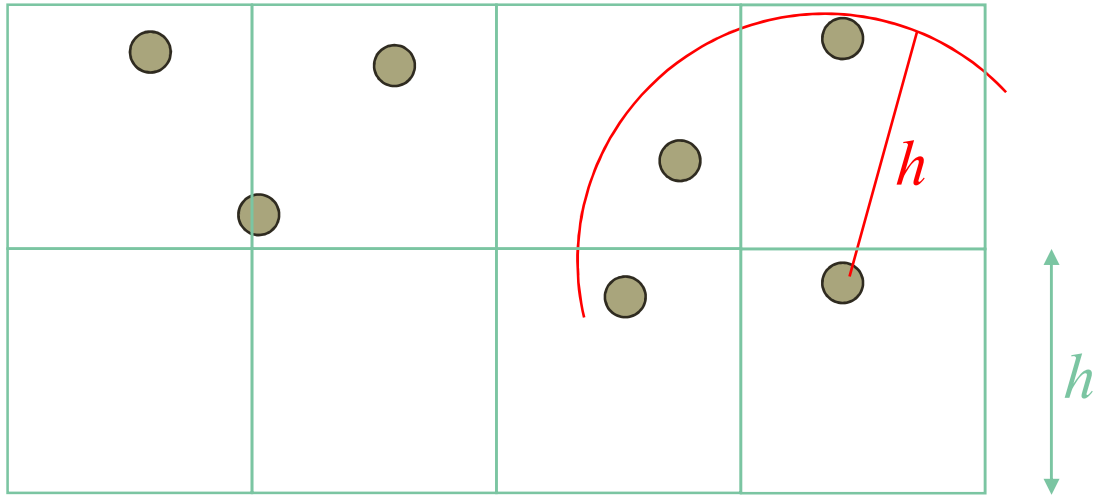
$$\nabla \cdot \nabla W_{poly6}(\vec{r}, h) = \begin{cases} \frac{945}{8\pi h^9}(h^2 - r^2)(r^2 - \frac{3}{4}(h^2 - r^2)) & 0 \le r \le h \\ 0 & otherwise \end{cases}$$

Smoothing Kernel $_{-3}$



- For n particles n^2 potential interactions!
- To reduce to linear complexity $O(n^2)$ define interaction cutoff distance h

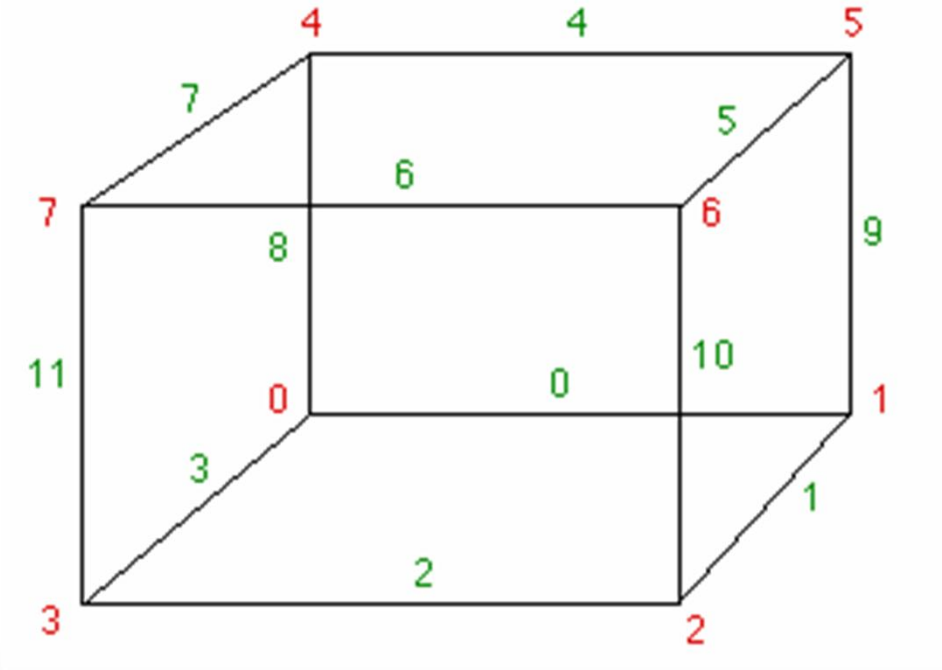
Smoothing Kernel $_{-4}$

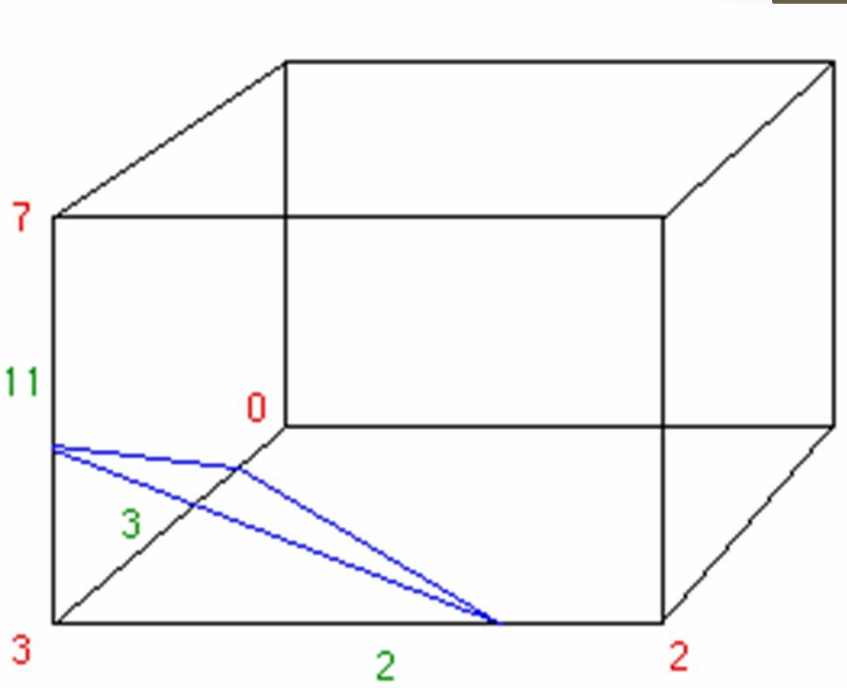


- Fill particles into grid with spacing h
- Only search potential neighbors in adjacent cells

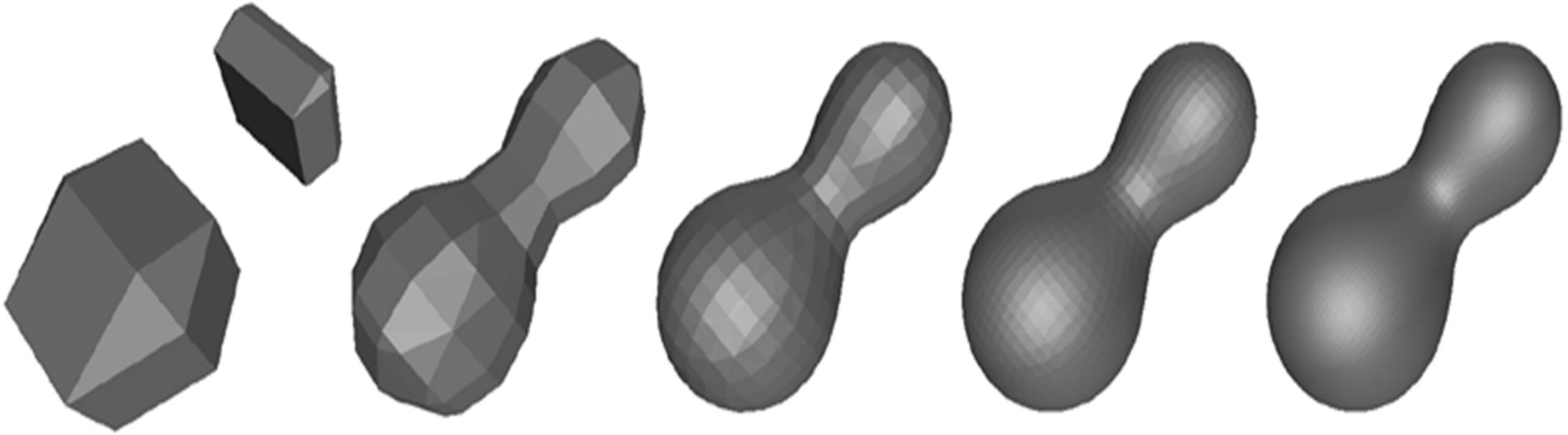
Marching Cubes -1

- To visualize the free surface





Marching Cubes -2



Grid size=10
70 Facets

Grid size=5
220 Facets

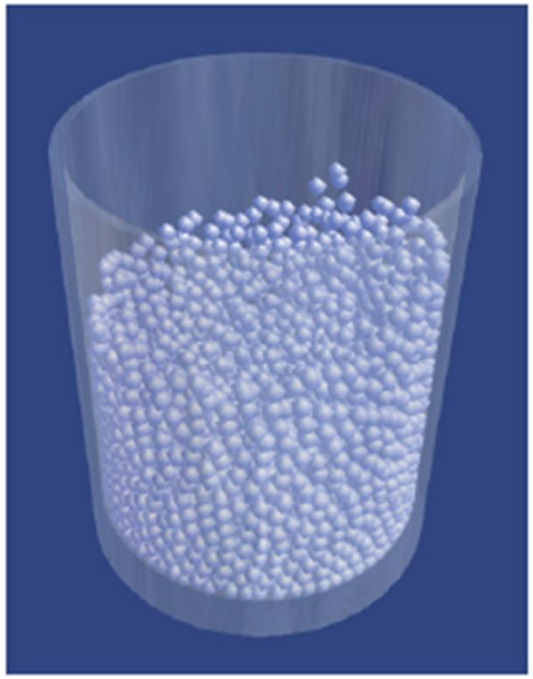
Grid size=2
1700 Facets

Grid size=1
6800 Facets

Grid size=0.5
27000 Facets

Result

- Interactive Simulation (5fps)



(a)

2200 particle

(b)

Point Splatting

(c)

Marching Cubs

General formulation:

$$W(\mathbf{r},h) = \frac{1}{Ch^d}\begin{cases} f(q), & 0 \le q \le 1 \\ 0, & \text{otherwise} \end{cases}$$

where $q = \big|\big|\mathbf{r}\big|\big|/h$ and $d = 3$ is the dimension. The kernels are based on the choices $f_{\text{poly6}}(q) = (1-q^2)^3$ for general interpolation, $f_{\text{spiky}}(q) = (1-q)^3$ for interpolating pressures, and $f_{\text{viscosity}}(q)$ for viscosity computations. The gradients are given by

$$\nabla W(\mathbf{r},h) = \frac{\mathbf{r}}{Ch^{d+2}}\begin{cases} q^{-1}f'(q), & 0 \le q \le 1 \\ 0, & \text{otherwise} \end{cases}$$

and the Laplacians are

$$\nabla^2 W(\mathbf{r},h) = \frac{1}{Ch^{d+2}}\begin{cases} f''(q) + (d-1)q^{-1}f'(q), & 0 \le q \le 1 \\ 0, & \text{otherwise} \end{cases}$$

$$f_{\text{viscosity}}(q) = \frac{q^2}{4} - \frac{q^3}{9} - \frac{1}{6}\log(q) - \frac{5}{36}.$$

The normalizing constants in two dimensions are determined by

$$C = \int_{0^1} 2\pi q f(q)\, dq.$$

For our three functions, these constants are

$$\begin{aligned} C_{\text{poly6}} &= \pi/4 \\ C_{\text{spiky}} &= \pi/10 \\ C_{\text{viscosity}} &= \pi/40. \end{aligned}$$

Kernel Function:

- cubic spline

$$W(\mathbf{x}_{ab}, h) = \frac{\sigma}{h^d}\begin{cases} 6q^3 - 6q^2 + 1 & 0 \le q \le \frac{1}{2} \\ 2(1-q)^3 & \frac{1}{2} \le q \le 1 \\ 0 & q > 1 \end{cases} \qquad q = |\mathbf{x}_{ab}| / h$$

with d denoting the dimensionality of the simulation and

$$\sigma = \begin{cases} \frac{4}{3} & d = 1 \\ \frac{40}{7\pi} & d = 2 \\ \frac{8}{\pi} & d = 3 \end{cases}$$

where $\mathbf{x}_{ab} = \mathbf{x}_a - \mathbf{x}_b$

Kernel Function:

- its derivative

$$\frac{\partial W(\mathbf{x}_{ab}, h)}{\partial \mathbf{x}_{ab}} = \frac{6\sigma}{h^{d+1}} \begin{cases} 3q^2 - 2q & 0 \le q \le \frac{1}{2} \\ -(1-q)^2 & \frac{1}{2} \le q \le 1 \\ 0 & q > 1 \end{cases} \qquad q = |\mathbf{x}_{ab}| / h$$

resulting in

$$\nabla W(\mathbf{x}_{ab}, h) = \frac{\partial W(\mathbf{x}_{ab}, h)}{\partial \mathbf{x}_{ab}} \frac{\mathbf{x}_{ab}}{|\mathbf{x}_{ab}|}$$

Simulation loop

For each time step:

1. Find neighbors to each particle and store in a list
2. Calculate density for each particle
3. Calculate pressure for each particle
4. Calculate all type of accelerations for each particle, and sum it up
5. Find new velocities and positions by using the same integration method as before…

Parameter values

Consider a container filled with 10 litre of water.
We will use SI-units for all parameters.

Number of SPH-particles: $N = 1000$

Mass: $m = 0.01 kg$

Density: $\rho_0 = 1000 \frac{kg}{m^3}$ (water)

Interaction radius: h is chosen such that 15-20 particles are interacting on average

Dynamical viscosity: $\mu = 0.001 \frac{\text{kg}}{\text{ms}}$

Speed of sound: $c_s = 1500 \frac{m}{s}$ (water) => **Time step:** $dt = 0.0001 s$

Speed of sound: $c_s = 1 - 10 \frac{m}{s}$ => **Time step:** $dt = 0.01 - 0.03 s$

The color property?

What is the use of this property?

- We can use it to detect the **position of the surface** of our fluid
- We can also use it to find the **normal vectors** at the surface (important for rendering!)
- The normal vectors allow us to implement surface tension
- By adding several color fields we can for example implement a simple model of **flame propagation**

The color field

- The color parameter is a quantity that is zero everywhere except at the particle where it has value one
- Similar to how we calculated density we now calculate the average color at particle i as

$$\langle C \rangle_i \approx \sum_j \frac{m_j}{\rho_j} C_j W(r_{ij})$$

- Deviations of the color field show us where the surface is, and in this case we choose to study the derivative of color field

- The gradient of a color field is

$$\langle \nabla C \rangle_i \approx \sum_j \frac{m_j}{\rho_j} C_j \nabla W(r_{ij})$$

- When the magnitude of the gradient is larger than a certain value, we tag the particle as a surface particle

Flame propagation

It is easy to introduce a simplified fire model by using the color field technique…

1. Add three new particle properties
    i. Burnable
    ii. Burn (on/off)
    iii. Burning time
2. For all surface particles that are burning
    i. Calculate an average color field gradient using the variable Burnable as particle color
    ii. Find all surface particles with average color field gradient higher than a certain value and flag them as burning (Burn on)
    iii. Decrease Burning time and check if Burn should be off
3. Visualize burning particles

Surface tension

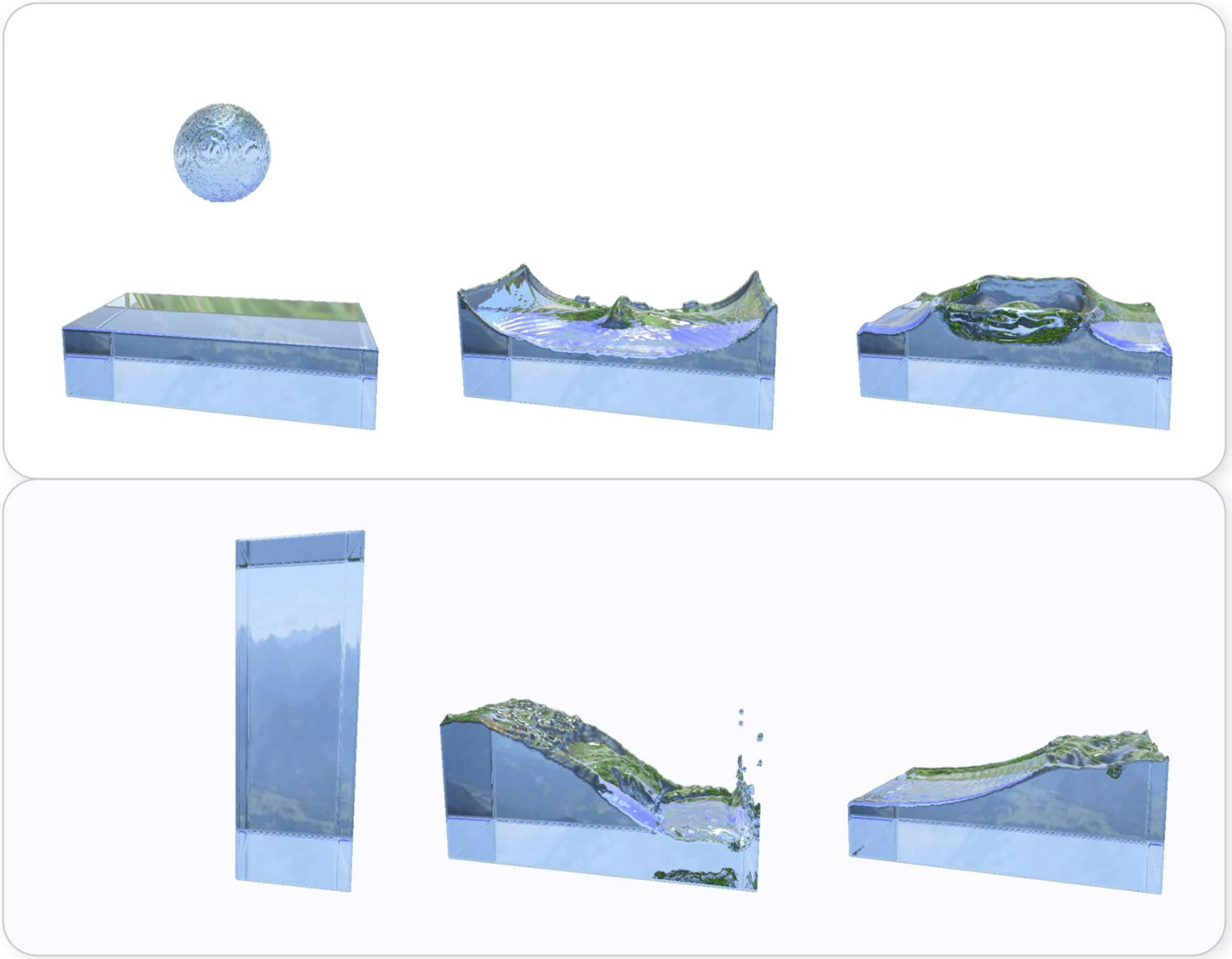
The force that tends to make surfaces smooth (like a drop of liquid) can be modeled in the following way:

$$a_i^{tension} = -\frac{\sigma_s}{\rho_i}\left\langle \nabla^2 C \right\rangle_i \frac{\mathbf{n}_i}{\left|\mathbf{n}_i\right|}$$

where $\mathbf{n}_i = \left\langle \nabla C \right\rangle_i$ and $\left\langle \nabla^2 C \right\rangle_i \approx \sum_j \frac{m_j}{\rho_j} C_j \nabla^2 W(r_{ij})$

Note: If the magnitude of n_i is small we can get numerical problem in the division above. To avoid this we only calculate $\mathbf{n}_i / \left|\mathbf{n}_i\right|$ if the magnitude of n_i exceeds a certain threshold.

SPH - Implementation



- Comentaris M.A.Vico: (SPH-GPU implementation)
- Per tal d'ordenar les particules per tal de trobar els veins eficientment primer vaig provar a implementar el Bitonic Sort: http://en.wikipedia.org/wiki/Bitonic_sorter
- Despres vaig trobar una forma millor de fer-ho en unes slides d'NVIDIA:

http://ondemand.gputechconf.com/gtc/2014/presentations/S4117-fast-fixed-radius-nearest-neighbor-gpu.pdf

- Una vegada les tenia ordenades vaig provar d'implementar el SPH classic tal i com el vas explicar a classe. Com no em funcionava, vaig mirar el paper que em vas passar, http://www.ligum.umontreal.ca/Clavet-2005-PVFS/pvfs.pdf
- I com no acabava d'anar be, vaig buscar gent que l'hagues intentat implementar i vaig trobar aquest blog: https://imdoingitwrong.wordpress.com/tag/sph/