Particle-based Fluids
Spatial Discretization

Fluid is discretized using particles
Particles = Molecules?

Particle approaches:

- **Molecular Dynamics**: relates each particle to one molecule
  - Can model molecular forces, integrate
  - 1 liter of water contains about $10^{25}$ molecules

- **SPH**: particle represents volume -> continuum assumptions apply
  - Properties such as density, pressure, etc are assumed to vary continuously in space
  - Field quantities defined at discrete particle locations, use interpolation to evaluate anywhere
Smoothed Particle Hydrodynamics (SPH)

- Fluid volume is discretized by particles
- Each particle represents a certain amount of fluid volume:
  \[ V_i = \frac{m_i}{\rho_i} \]
- Note: mass is constant, density is not
Smoothed Particle Hydrodynamics (SPH)

- Particles store attributes
- To evaluate an attribute, take weighted average of particle values within a neighborhood
- Smoothing kernel $W$ prescribes interpolation weights
Kernel Properties

- Typically radially symmetric
  \( r \): offset from kernel center

- Normalization condition:
  \[
  \int W(r) dr = 1
  \]

- If \( W \) is even (\( W(r,h)=W(-r,h) \)): second order accuracy

- Compact support
  \[ W(r,h) = 0 \text{ when } |r| > h \]
SPH Summation Equation

- Computing some quantity $A$ at an arbitrary position in space
- Sum up contribution of neighboring particles $j$

$$A(x) = \sum_j \frac{m_j}{\rho_j} A_j W(x - x_j, h)$$

- Quantity $A$ at arbitrary position $x$
- Particle volume
- Sum over all neighbor particles $j$ within $h$
- Quantity $A$ of particle $j$
- Smoothing kernel
A quantity $A$ of particle $i$ can be computed by summing up the contributions from neighbors $j$:

$$A_i = \sum_j \frac{m_j}{\rho_j} A_j W(x_i - x_j, h)$$

$$A_i = \sum_j \frac{m_j}{\rho_j} A_j W(x_{ij}, h)$$

$$A_i = \sum_j \frac{m_j}{\rho_j} A_j W_{ij}$$

Note: Neighborhood $j$ includes particle $i$
Differentiation

- Gradient and Laplacian can be easily calculated

\[ A_i = \sum_j \frac{m_j}{\rho_j} A_j W_{ij} \]

\[ \nabla A_i = \sum_j \frac{m_j}{\rho_j} A_j \nabla W_{ij} \]

\[ \nabla^2 A_i = \sum_j \frac{m_j}{\rho_j} A_j \nabla^2 W_{ij} \]
Numerical Solution to NS Equations

$$\rho \frac{\partial \mathbf{u}}{\partial t} = -\nabla p + \mu \nabla^2 \mathbf{u} + \rho \mathbf{g}$$

Neighbor Search

Density, Pressure

Body Force / V

Diffusion / V

Pressure Force / V
Estimating Density

\[ A_i = \sum_j \frac{m_j}{\rho_j} A_j W_{ij} \]

- Attribute A is now density \( \rho \)

\[ \rho_i = \sum_j \frac{m_j}{\rho_j} \rho_j W_{ij} \]

\[ \rho_i = \sum_j m_j W_{ij} \]

\[ \frac{\partial \rho}{\partial t} = -\nabla \rho + \mu \nabla^2 u + \rho \]
From Density to Pressure

- Pressure is computed through the ideal gas law: \( p = k \rho \)

- Modified version gives pressure of particle \( i \):
  \[
  p_i = k(\rho_i - \rho_0)
  \]

- Modified pressure is proportional to density deviation

- Gradient (forces) should not be affected by offset, but SPH approximation is numerically more stable

- Stiffness \( k \) defines speed of response
  
  \( \text{(note: stiff system -> smaller time step)} \)
Density and Pressure - Example

\[ p_i = k(\rho_i - \rho_0) \]

Example: \( \rho_0 = 1000 \)

Larger density
Larger pressure

\[ \begin{align*}
\rho_i &= 1200 \\
p_i &= k(1200 - 1000) \\
&= k \times 200
\end{align*} \]

Smaller density
Smaller pressure

\[ \begin{align*}
\rho_i &= 1010 \\
p_i &= k(1010 - 1000) \\
&= k \times 10
\end{align*} \]

Density below rest density
Negative pressure

\[ \rho_i = 800 \\
p_i = k(800 - 1000) \\
&= -k \times 200 \]
Pressure Force Density

\[ f_{\text{pressure}} = -\nabla p \]

\[ \rho \frac{\partial u}{\partial t} = -\nabla p + \mu \nabla^2 u + \rho g \]

Larger density
Larger pressure
Larger repulsion forces
\[ \rho_i = 1200 \]
\[ p_i = k(1200 - 1000) \]
\[ = k \times 200 \]

Smaller density
Smaller pressure
Smaller repulsion forces
\[ \rho_i = 1010 \]
\[ p_i = k(1010 - 1000) \]
\[ = k \times 10 \]

Density below rest density
Negative pressure
Attraction forces
\[ \rho_i = 800 \]
\[ p_i = k(800 - 1000) \]
\[ = -k \times 200 \]
Pressure Force Density

\[ f_{\text{pressure}} = -\nabla p \]

\[ \rho \frac{\partial u}{\partial t} = -\nabla p + \mu \nabla^2 u + \rho g \]

- Larger density
- Larger pressure
- Larger repulsion forces

- Smaller density
- Smaller pressure
- Smaller repulsion forces

- Density below rest density
- Negative pressure
- Attraction forces

- Pressure force aims to restore the rest state of the fluid
  (where \( \forall i \rho_i = \rho_0 \))
Attraction Forces

- Attraction forces can lead to numerical instability
- Typically, only repulsion forces are used:

\[ p_i = max(k(\rho_i - \rho_0), 0) \]
Pressure Force Density

\[ \nabla A_i = \sum_j \frac{m_j}{\rho_j} A_j \nabla W_{ij} \]

For particle \( i \) \( -\nabla p \) evaluates to:

\[ f_i^{\text{pressure}} = - \sum_j \frac{m_j}{\rho_j} p_j \nabla W_{ij} \]

Pressure force acts along vector between particles!

\[ f_1^{\text{pressure}} = - \frac{m_2}{\rho_2} p_2 \nabla W_{12} \]
\[ f_2^{\text{pressure}} = - \frac{m_1}{\rho_1} p_1 \nabla W_{21} \]

\[ \mathbf{F} = \mathbf{f} \mathbf{V} \]

if \( p_1 \neq p_2 \): forces not symmetric!

Note: Kernel is multiplied by distance vector \( x_{ij} \)
Pressure Force Density

- Non-symmetric forces violate Newton’s 3rd law: action = reaction
- Use arithmetic mean of pressure values

\[ f_{i\text{pressure}} = - \sum_j \frac{m_j}{\rho_j} \left\{ \frac{p_i + p_j}{2} \right\} \nabla W_{ij} \]

\[ \rho \frac{\partial u}{\partial t} = -\nabla p + \mu \nabla^2 u + \rho g \]
Incompressibility

◆ The stiffer the fluid (the larger k), the less compression

• Incompressibility by using very (!) large k (density variation < 1%)
Viscosity Force Density

- For particle $i$, $\mu \nabla^2 u$ evaluates to:

$$f_{i, \text{viscosity}} = \mu \sum_j \frac{m_j}{\rho_j} u_j \nabla^2 W_{ij}$$

- Once again need to make forces symmetric. Insight: viscosity forces are only dependent on velocity differences, not on absolute velocities

$$f_{i, \text{viscosity}} = \mu \sum_j \frac{m_j}{\rho_j} u_j - u_i \nabla^2 W_{ij}$$

Note: Viscosity is always needed to stabilize the particle system
Zero Viscosity vs Normal Viscosity vs High Viscosity
Gravitational Force Density

For particle $i$ we get:

$$f_{i}^{\text{gravity}} = \rho_{i}g$$

- Other forces can easily be included (collision forces, boundary forces, user interaction, etc.)
Different kernels can be used, see for example [Müller03]
Choice of kernel affects stability, accuracy and speed of SPH methods
Poly6 kernel: use for everything but pressure forces & viscosity

\[
W_{\text{poly6}}(\mathbf{r}, h) = \begin{cases} 
\frac{315}{64\pi h^9} (h^2 - r^2)^3 & 0 \leq r \leq h \\
0 & \text{otherwise}
\end{cases}
\]
Different kernels can be used, see for example [Müller03]
Choice of kernel affects stability, accuracy and speed of SPH methods
Spiky kernel: use for pressure forces

\[
W_{\text{spiky}}(r, h) = \frac{15}{\pi h^6} \begin{cases} 
(h - r)^3 & 0 \leq r \leq h \\
0 & \text{otherwise,}
\end{cases}
\]
Kernel Function

- Different kernels can be used, see for example [Müller03]
- Choice of kernel affects stability, accuracy and speed of SPH methods
- Viscosity kernel: Laplacian is always positive!

$$W_{\text{viscosity}}(r, h) = \frac{15}{2\pi h^3} \begin{cases} 
-\frac{r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1 & 0 \leq r \leq h \\
0 & \text{otherwise.} 
\end{cases}$$
Time Integration

• Total force density: \( f_i = f_i^{\text{pressure}} + f_i^{\text{viscosity}} + f_i^{\text{gravity}} \)

• Acceleration: \( a_i = \frac{f_i}{\rho_i} \)

◆ Symplectic Euler scheme:
  - New velocity: \( u_i(t + 1) = u_i(t) + \Delta t \frac{f_i}{\rho_i} \)
  - New position: \( x_i(t + 1) = x_i(t) + \Delta t u_i(t + 1) \)
Neighbor Search

- 3D: 30-40 neighbors per particle → Neighbor computation is most expensive part → we need fast data structures

- Domain partitioning into cells of size $h$
- Potential neighbors in 27 cells
- Create grid, insert particles, compute neighbors
How to get started with SPH...

- Start with simple implementation, worry about performance/optimizations once it works

- Define walls/boundary conditions geometrically → simple collision tests
  (if $x[y] < 0$ then $x[y] = 0$)

Debugging 1: Visualize particles / quantities (densities, velocities)

Debugging 2: Viscosity const large enough? Time step small enough?
Algorithm

SPH Algorithm

1 while animating do
2   for all $i$ do
3       find neighborhoods $N_i(t)$
4   for all $i$ do
5       compute density $\rho_i(t)$
6       compute pressure $p_i(t)$
7   for all $i$ do
8       compute forces $F_{p,v,g,ext}(t)$
9   for all $i$ do
10      compute new velocity
11      compute new position

\[
\rho_i = \sum_j m_j W_{ij}
\]
\[
p_i = k(\rho_i - \rho_0)
\]
\[
f_i^{\text{pressure}} = -\sum_j \frac{m_j p_i + p_j}{\rho_j} \nabla W_{ij}
\]
\[
f_i^{\text{viscosity}} = \mu \sum_j \frac{m_j}{\rho_j} (\mathbf{u}_j - \mathbf{u}_i) \nabla^2 W_{ij}
\]
\[
f_i^{\text{gravity}} = \rho_i g
\]
\[
\mathbf{u}_i(t+1) = \mathbf{u}_i(t) + \Delta t \frac{f_i}{\rho_i}
\]
\[
\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \Delta t \mathbf{u}_i(t+1)
\]
Multiple Fluids

- Particles carry different attributes
  - Mass
  - Rest density
  - Viscosity coefficient
  - Gas constant (stiffness)
  - Color attributes
  - Temperature

- Buoyancy emerges from individual rest densities (*note:* $V_1 = V_2$)
An Implicit Viscosity Formulation for SPH Fluids

Andreas Peer, Markus Ihmsen, Jens Cornelis, Matthias Teschner
University of Freiburg
Solids with SPH

- Unified model for fluids, solids, elastic objects -> phase changes
- Displacement from undeformed shape
  -> strain, stress, elastic forces

\[
a_i = \frac{1}{\rho_i} (f_{i\text{ pressure}} + f_{i\text{ viscosity}} + f_{i\text{ external}}) + f_{i\text{ rigid}} + f_{i\text{ elastic}}
\]
Position-Based Dynamics
Position-Based Dynamics

- Physics-inspired*: everything is a set of particles connected by *constraints*
  - Replace forces & numerical integration by constraint projection

- Very fast, very stable, simple, plausible results
  - used in video games
    - Nvidia PhysX
  - used in Autodesk’s Maya:
    https://autodeskresearch.com/publications/nucleus
Position-Based Dynamics

Unified Particle Physics for Real-Time Applications

Miles Macklin  Matthias Müller  Nuttapong Chentanez  Tae-Yong Kim

NVIDIA
Many references, but start here:
• “Position-Based Simulation Methods in Computer Graphics”, Bender, Muller and Macklin, Eurographics 2015
Position-Based Dynamics

- Setup similar to mass-spring systems
  - Discretize using mass points: mass $m_i$, position $x_i$, velocity $v_i$
- Rather than forces, use constraints:
  - distance constraint, area preservation, etc
A constraint $j \in [1, \ldots, M]$ consists of

- a cardinality $n_j$,
- a function $C_j : \mathbb{R}^{3n_j} \rightarrow \mathbb{R}$,
- a set of indices $\{i_1, \ldots, i_{n_j}\}$, $i_k \in [1, \ldots N]$,
- a stiffness parameter $k_j \in [0 \ldots 1]$ and
- a type of either equality or inequality.
PBD: main loop

(1) forall vertices $i$
(2) initialize $x_i^0, v_i^0, w_i = 1/m_i$
(3) endfor
(4) loop
(5) forall vertices $i$ do $v_i \leftarrow v_i + \Delta t w_i f_{ext}(x_i)$
(6) dampVelocities($v_1, \ldots, v_N$)
(7) forall vertices $i$ do $p_i \leftarrow x_i + \Delta t v_i$
(8) forall vertices $i$ do generateCollisionConstraints($x_i \rightarrow p_i$)
(9) loop solverIterations times
(10) projectConstraints($C_1, \ldots, C_{M+M_{coll}}, p_1, \ldots, p_N$)
(11) endloop
(12) forall vertices $i$
(13) $v_i \leftarrow (p_i - x_i)/\Delta t$
(14) $x_i \leftarrow p_i$
(15) endfor
(16) velocityUpdate($v_1, \ldots, v_N$)
(17) endloop
PBD: constraint projection

\[ c(p_i, p_j) = |p_i - p_j| - d \]
PBD: constraint projection
PBD: constraint projection

Taylor expansion (per constraint):

\[ C(p + \Delta p) \approx C(p) + \nabla_p C(p) \cdot \Delta p = 0 \]

Step must be in direction of constraint Jacobian:

\[ \Delta p = \lambda \nabla_p C(p) \]

Putting it all together:

\[ \Delta p = -\frac{C(p)}{\lvert \nabla_p C(p) \rvert^2} \nabla_p C(p) \]
PBD: constraint projection

$$\Delta p_i = -s \nabla_{p_i} C(p_1, \ldots, p_n)$$

$$s = \frac{C(p_1, \ldots, p_n)}{\sum_j |\nabla_{p_j} C(p_1, \ldots, p_n)|^2}$$

Is momentum conserved?
PBD: constraint projection

Conservation of linear momentum:

\[ \sum_i m_i \Delta p_i = 0 \]

Scale updates by inverse mass \( w_i = 1/m_i \)

\[ \Delta p_i = -s w_i \nabla_{p_i} C(p_1, \ldots, p_n) \]

\[ s = \frac{C(p_1, \ldots, p_n)}{\sum_j w_j |\nabla_{p_j} C(p_1, \ldots, p_n)|^2} \]
PBD: constraint projection
PBD: constraint projection

\[ C(p_1, p_2) = |p_1 - p_2| - d \]

\[ \nabla_{p_1} C(p_1, p_2) = n \]

\[ \nabla_{p_2} C(p_1, p_2) = -n \]

\[ n = \frac{p_1 - p_2}{|p_1 - p_2|} \]

\[ s = \frac{|p_1 - p_2| - d}{w_1 + w_2} \]

\[ \Delta p_1 = -\frac{w_1}{w_1 + w_2} (|p_1 - p_2| - d) \frac{p_1 - p_2}{|p_1 - p_2|} \]

\[ \Delta p_2 = +\frac{w_2}{w_1 + w_2} (|p_1 - p_2| - d) \frac{p_1 - p_2}{|p_1 - p_2|} \]
PBD: main loop

(1) forall vertices $i$
(2) initialize $x_i = x_i^0$, $v_i = v_i^0$, $w_i = 1/m_i$
(3) endfor
(4) loop
(5) forall vertices $i$ do $v_i \leftarrow v_i + \Delta t w_i f_{ext}(x_i)$
(6) dampVelocities($v_1, \ldots, v_N$)
(7) forall vertices $i$ do $p_i \leftarrow x_i + \Delta t v_i$
(8) forall vertices $i$ do generateCollisionConstraints($x_i \rightarrow p_i$)
(9) loop solverIterations times
(10) projectConstraints($C_1, \ldots, C_{M+M_{coll}}, p_1, \ldots, p_N$)
(11) endloop
(12) forall vertices $i$
(13) $v_i \leftarrow (p_i - x_i)/\Delta t$
(14) $x_i \leftarrow p_i$
(15) endfor
(16) velocityUpdate($v_1, \ldots, v_N$)
(17) endloop
Collision constraints

Assume: collisions already detected
- Simple objects/particles: easy

Collision response:
- Unilateral constraints (inequality)
- Potentially changing at every time-step
Collision constraints

\( x \)

\( q_c \)

\( p \)
Collision constraints
Collision constraints
Collision constraint

\[ C(p) = (p - q_c) \cdot n_c \]

Unilateral:  \( C(p) \geq 0 \)

- Project only if violated
PBD: main loop

(1) forall vertices $i$
(2) initialize $x_i = x_i^0, v_i = v_i^0, w_i = 1/m_i$
(3) endfor
(4) loop
(5) forall vertices $i$ do $v_i \leftarrow v_i + \Delta t w_i f_{ext}(x_i)$
(6) dampVelocities($v_1, \ldots, v_N$)
(7) forall vertices $i$ do $p_i \leftarrow x_i + \Delta t v_i$
(8) forall vertices $i$ do generateCollisionConstraints($x_i \rightarrow p_i$)
(9) loop $\text{solverIterations}$ times
(10) projectConstraints($C_1, \ldots, C_{M+M_{coll}}, p_1, \ldots, p_N$)
(11) endloop
(12) forall vertices $i$
(13) $v_i \leftarrow (p_i - x_i)/\Delta t$
(14) $x_i \leftarrow p_i$
(15) endfor
(16) velocityUpdate($v_1, \ldots, v_N$)
(17) endloop
PBD: main loop

- Not a traditional time-stepping scheme
  - Predictor-Corrector approach
- Stability due to constraint projection which acts directly on positions
  - note: no internal forces!
PBD: can you see any downsides to this approach?

(1) forall vertices $i$
(2) initialize $x_i = x_i^0$, $v_i = v_i^0$, $w_i = 1/m_i$
(3) endfor
(4) loop
(5) forall vertices $i$ do $v_i \leftarrow v_i + \Delta t w_i f_{ext}(x_i)$
(6) dampVelocities($v_1, \ldots, v_N$)
(7) forall vertices $i$ do $p_i \leftarrow x_i + \Delta t v_i$
(8) forall vertices $i$ do generateCollisionConstraints($x_i \rightarrow p_i$)
(9) loop solverIterations times
(10) projectConstraints($C_1, \ldots, C_{M+M_{coll}}, p_1, \ldots, p_N$)
(11) endloop
(12) forall vertices $i$
(13) $v_i \leftarrow (p_i - x_i)/\Delta t$
(14) $x_i \leftarrow p_i$
(15) endfor
(16) velocityUpdate($v_1, \ldots, v_N$)
(17) endloop
Other types of constraints?

- Position-based Fluids
  “Position Based Fluids”, Macklin & Muller, 2014
Assignment 2

- Out today
- Due on March 2\textsuperscript{nd} @ midnight
Reminder

◆ Project Ideas
  • Brief descriptions due today (or plan to talk to us this week)
    ▪ Team info, topic, etc.