# Smoothed Particle Hydrodynamics

David Suh

University of Chicago

2023

# What is SPH?

- *Smoothed-particle hydrodynamics*
- Simulate a fluid using $n$ particles.
  - What about points in space that don't have a particle?
  - We can estimate it using nearby particles.

# SPH Equations

- *Müller, et al. 2003*
- (Navier-Stokes):

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v}\cdot\nabla\mathbf{v}\right) = -\nabla p + \rho\mathbf{g} + \mu\nabla^2\mathbf{v}$$

- (Conservation of mass)

$$\frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\mathbf{v}) = 0$$

- $\mathbf{v}$ = velocity field, $p$ = pressure, $\rho$ = density, $\mu$ = viscosity constant, $\mathbf{g}$ = external force.

# SPH Equations 2

- We can simplify the equations using assumptions of SPH.
- Since number of particles, and particle mass, are constant – we can ignore conservation of mass.
- In a particle system: ignore $\mathbf{v} \cdot \nabla \mathbf{v}$ term. Some derivative changes. We can get

$$\mathbf{a}_i = \mathbf{f}_i / \rho_i$$

for the $i$-th particle, where

$$\begin{aligned} \mathbf{f}_i &= -\nabla p + \rho \mathbf{g} + \mu \nabla^2 \mathbf{v} \\ &= \mathbf{f}_i^{\text{pressure}} + \mathbf{f}_i^{\text{ext}} + \mathbf{f}_i^{\text{viscosity}} \end{aligned}$$

# SPH Equations 3

- Now, we can estimate the terms $\mathbf{f}_i^{\text{pressure}}, \mathbf{f}_i^{\text{ext}}, \mathbf{f}_i^{\text{viscosity}}$ in the context of SPH.
- Note that $\mathbf{r}_i$ is the position of particle $i$.

$$
\begin{aligned}
\rho_i &= \sum_j m_j W(\mathbf{r}_i - \mathbf{r}_j, h) \\
p_i &= k(\rho - \rho_0) \\
\mathbf{f}_i^{\text{pressure}} &= -\sum_j (m_j/\rho_j) \frac{p_j + p_i}{2} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h) \\
\mathbf{f}_i^{\text{ext}} &= \text{Const.} \\
\mathbf{f}_i^{\text{viscosity}} &= \mu \sum_j (m_j/\rho_j)(\mathbf{v}_j - \mathbf{v}_i) \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h)
\end{aligned}
$$

# Kernels

- The function $W$ is called the *smoothing kernel*.
- Basically, we use a weighted sum of the nearby particles to estimate properies of the current particle/position (e.g. density, forces).
- Many different kernels, specifics in paper. One example is

$$W_{\text{poly6}}(\mathbf{r}, h) = \frac{315}{64\pi h^9}(h^2 - r^2)^3 \chi_{\{0 \leq r \leq h\}}$$

  Note that $r = \|\mathbf{r}\|$. Why? Don't know, blame authors.

Calculate everything, then we can use basic leap-frog:

$$\mathbf{v} \leftarrow \mathbf{v} + \mathbf{a}\Delta t$$

$$\mathbf{r} \leftarrow \mathbf{r} + \mathbf{v}\Delta t$$

# Actual Programming (Computation)

- How do we code this? We use numpy.
- Run the density, and force calculations, for each particle.
- Visualization: marching squares/cubes

# Optimizations

- The system is generally slow, even if we use numpy operations. Accelerate how?
- JAX: Google's version of pytorch (but not really).
  - We have function $f : \mathbb{R}^m \to \mathbb{R}$. Suppose we have a matrix $[\mathbf{v}_1, \ldots, \mathbf{v}_n]^T$. How to get $[f(\mathbf{v}_1), \ldots, f(\mathbf{v}_n)] \in \mathbb{R}^n$ fast?
  - We can be very smart. Or, we use `jax.vmap`.
  - Also some compiling/unrolling/etc... for better performance.
  - approx. 3x speedup (benchmarking to be done).
- QuadTree/$R^*$-tree/KDTree – nearest - neighbors based on $h$. But is it really efficient? Not sure....

# Sample Video

# TODO:

- Benchmarking normal numpy vs. JAX CPU vs. JAX GPU
- More particles!!
- Tweaking variables $(h, \mu, \Delta t, \rho_0)$. No info online!?!? Just gotta grokk it.
- Better performance profiling.
- Transition to 3D. Should be just one number tweak, but we will see.