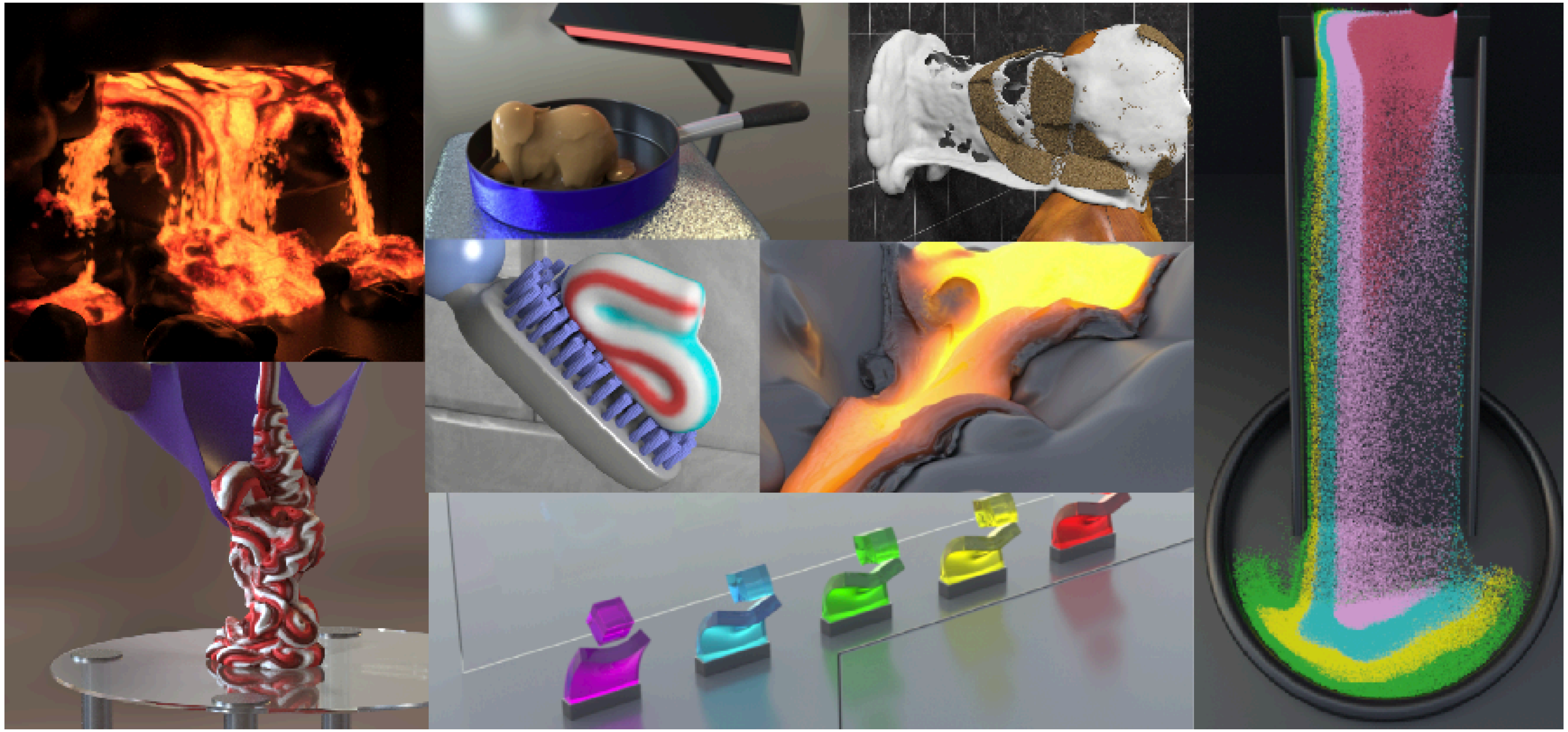
Instructor: Minchen Li

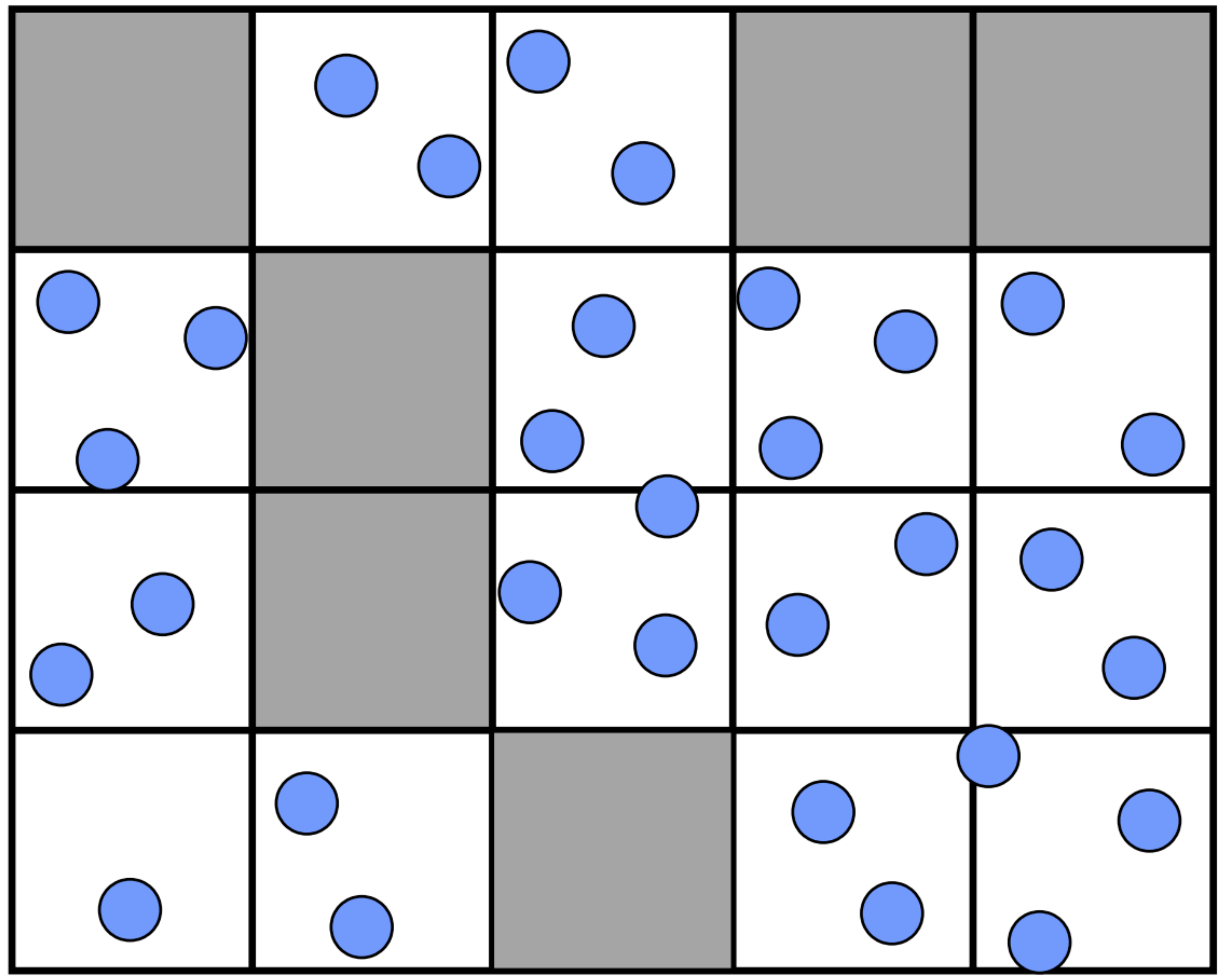


Lec 16: Material Point Methods

15-763: Physics-based Animation of Solids and Fluids (S25)

Recap: Hybrid Lagrangian/Eulerian Methods

Basic Idea



Introduce a background Eulerian Grid,
and measure quantities on the grid nodes

Transfer information between the particles and grid

Time Splitting

For each time step n:

$u^a \leftarrow$ Solve $\frac{\partial u}{\partial t} + u \cdot \nabla u = 0$ (advection) — Using particles

$u^b \leftarrow$ Solve $\frac{\partial u}{\partial t} = g$ (apply external force)

$u^c \leftarrow$ Solve $\frac{\partial u}{\partial t} = \nu \nabla \cdot \nabla u$ (diffusion)

$u^{n+1} \leftarrow$ Solve $\nabla \cdot u = 0$ (pressure projection)

Using the grid

— take advantage of
both representations

Recap: Extending to Solid Simulation

The Material-Point Method

Time Splitting

For each time step n:

$u^a \leftarrow$ Solve $\frac{\partial u}{\partial t} + u \cdot \nabla u = 0$ (advection) — Using particles

Transfer velocity from particles to grid

$u^b \leftarrow$ Solve $\frac{\partial u}{\partial t} = g$ (apply external force)

$u^c \leftarrow$ Solve $\frac{\partial u}{\partial t} = \nu \nabla \cdot \nabla u$ (diffusion)

$u^{n+1} \leftarrow$ Solve $\nabla \cdot u = 0$ (pressure projection)

Using the grid

Transfer velocity from grid to particles

The Particle-In-Cell Method

Time Splitting

For each time step n:

$u^a \leftarrow$ Solve $\frac{\partial u}{\partial t} + u \cdot \nabla u = 0$ (advection) — Using particles

Transfer <u>information</u> from particles to grid

$u^{n+1} \leftarrow$ Solve $\frac{\partial u}{\partial t} = \nabla \cdot \sigma + g$ — Using the grid

Transfer <u>information</u> from grid to particles

- Needs to track deformation gradient per particle using updated Lagrangian: $\mathbf{F}^{n+1} \approx \mathbf{F}^n + h\frac{\partial \mathbf{F}}{\partial t}$

$$\frac{\partial}{\partial t}\mathsf{F}(\mathbf{X}, t^{n+1}) = \frac{\partial \mathbf{V}}{\partial \mathbf{X}}(\mathbf{X}, t^{n+1}) = \frac{\partial \mathbf{v}^{n+1}}{\partial \mathbf{x}}(\phi(\mathbf{X}, t^n))\mathsf{F}(\mathbf{X}, t^n)$$

Today: The MPM Pipeline

- Particle-Grid Transfer
- Force Calculation
- Grid Updates
- Particle Advection

Today: The MPM Pipeline

- Particle-Grid Transfer
- Force Calculation
- Grid Updates
- Particle Advection

Particle-Grid Transfer

Fluids

Grid to particle is easy, can just use e.g. bilinear interpolation:

$x_p = Px_i$, $P \in \mathbb{R}^{dn_p \times dn_i}$ **stores the interpolation weights**

Particle to grid:

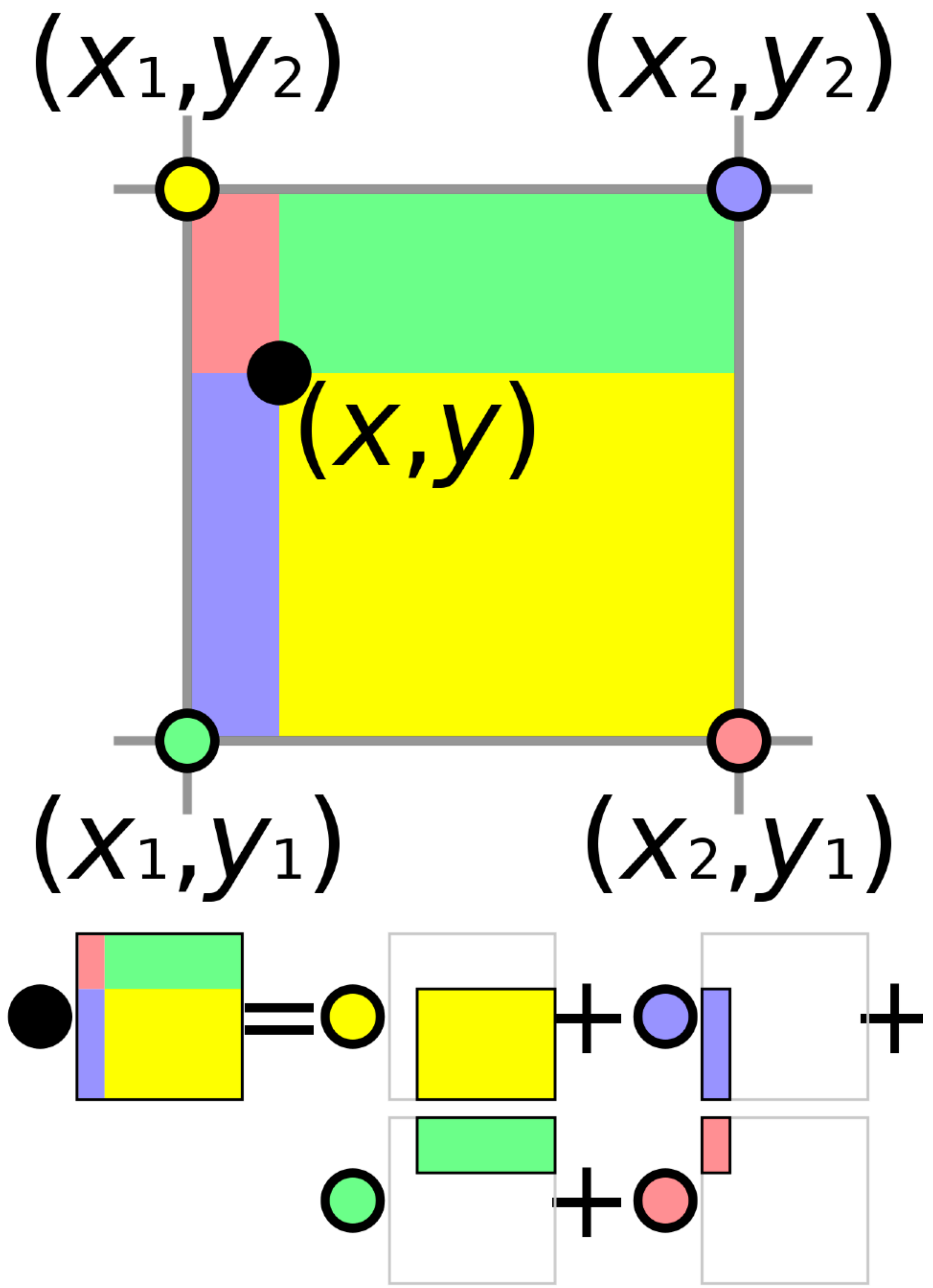
$x_i = D^{-1}P^T x_p$, **where** $D_{ij} = \delta_{ij} \sum_k P_{ki}$ **is for normalization**

Observation: 2D weights are areas, or product of 1D weights

Define a general transfer kernel:

$$N_i(x_p) = N(\frac{1}{h}(x_p - x_i))N(\frac{1}{h}(y_p - y_i))N(\frac{1}{h}(z_p - z_i))$$

where N is the 1D weight function.



Particle-Grid Transfer

Kernel and Weight Functions

Transfer kernel: $N_i(x_p) = N(\frac{1}{h}(x_p - x_i))N(\frac{1}{h}(y_p - y_i))N(\frac{1}{h}(z_p - z_i))$ **where N is the 1D weight function.**

eg.

$$N(x) = \begin{cases} 1 - |x| & 0 \leqslant |x| < 1 \\ 0 & 1 \leqslant |x| \end{cases}$$

Linear

$$N(x) = \begin{cases} \frac{3}{4} - |x|^2 & 0 \leqslant |x| < \frac{1}{2} \\ \frac{1}{2}\left(\frac{3}{2} - |x|\right)^2 & \frac{1}{2} \leqslant |x| < \frac{3}{2} \\ 0 & \frac{3}{2} \leqslant |x| \end{cases}$$

Quadratic

$$N(x) = \begin{cases} \frac{1}{2}|x|^3 - |x|^2 + \frac{2}{3} & 0 \leqslant |x| < 1 \\ \frac{1}{6}(2 - |x|)^3 & 1 \leqslant |x| < 2 \\ 0 & 2 \leqslant |x| \end{cases}$$

Cubic

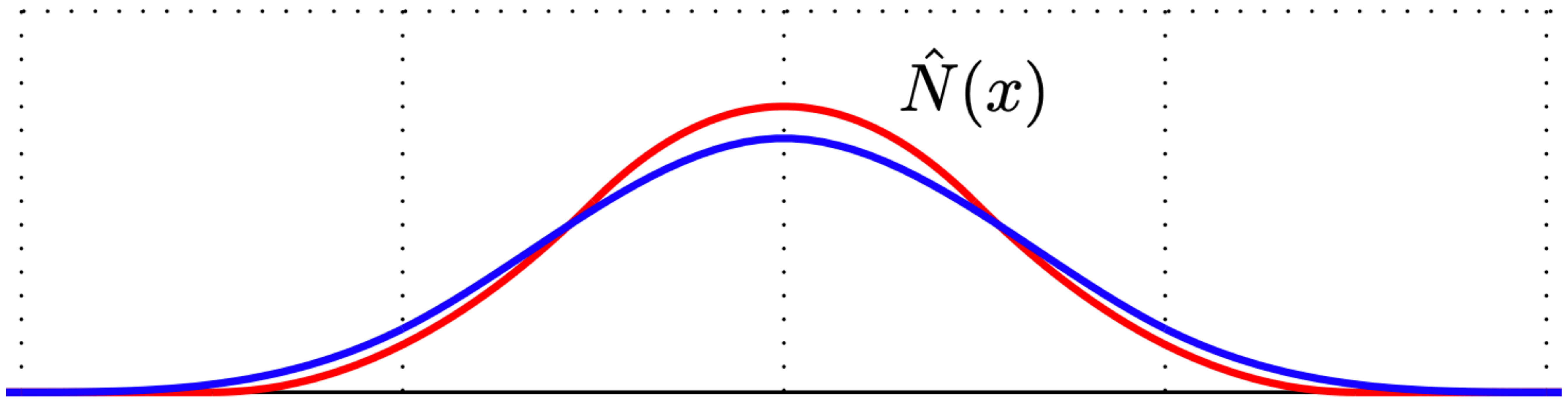


Figure 3: Cubic (blue) and quadratic (red) splines used for computing interpolation weights.

Particle-Grid Transfer

Particle-to-Grid Mass and Momentum Transfer

Transfer kernel: $N_i(\boldsymbol{x}_p) = N(\frac{1}{h}(x_p - x_i))N(\frac{1}{h}(y_p - y_i))N(\frac{1}{h}(z_p - z_i))$ **where N is the 1D weight function.**

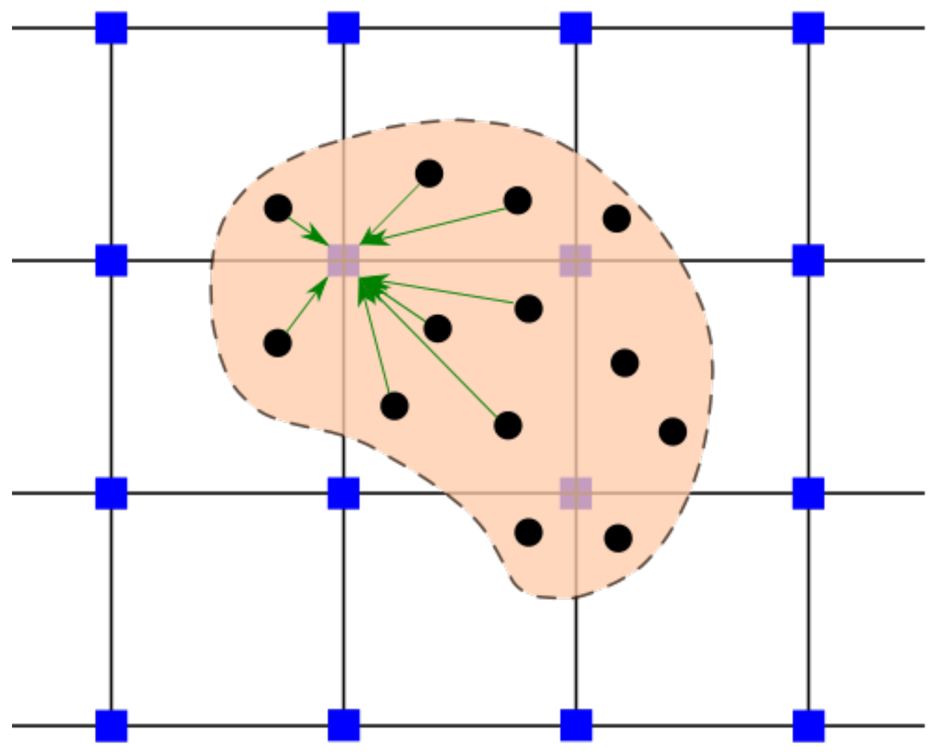
Mass: $m_i = \sum_p m_p N_i(\boldsymbol{x}_p)$ **Momentum:** $(m\boldsymbol{v})_i = \sum_p m_p \boldsymbol{v}_p N_i(\boldsymbol{x}_p)$

Mass conservation: $\sum_i m_i = \sum_i \sum_p m_p N_i(\boldsymbol{x}_p) = \sum_p m_p \sum_i N_i(\boldsymbol{x}_p) = \sum_p m_p$

— Partition-of-Unity of N

Momentum conservation: $\sum_i (m\boldsymbol{v})_i = \sum_p m_p \boldsymbol{v}_p$

Velocity: $\boldsymbol{v}_i = \frac{(m\boldsymbol{v})_i}{m_i}$



Velocity is averaged, but mass and momentum are accumulated!

Particle-Grid Transfer

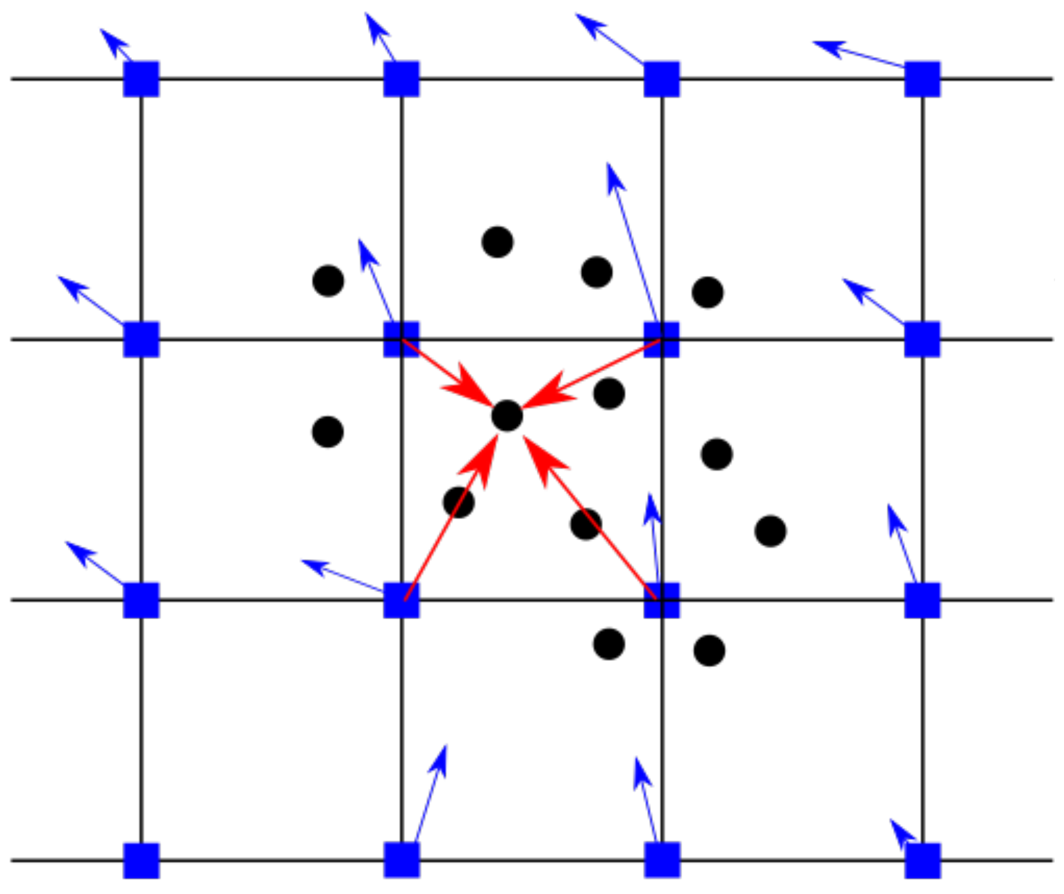
Grid-to-Particle Velocity Transfer

Mass: no change, thus conserved

Velocity: $v_p = \sum_i v_i N_i(x_p)$

Momentum conservation:

$$\sum_p m_p v_p = \sum_p m_p \sum_i v_i N_i(x_p) = \sum_i v_i \sum_p m_p N_i(x_p) = \sum_i m_i v_i$$



- Here, we are focusing on Particle-In-Cell transfer, other transfer schemes, e.g. FLIP, APIC, etc., are available for higher accuracy.

Today: The MPM Pipeline

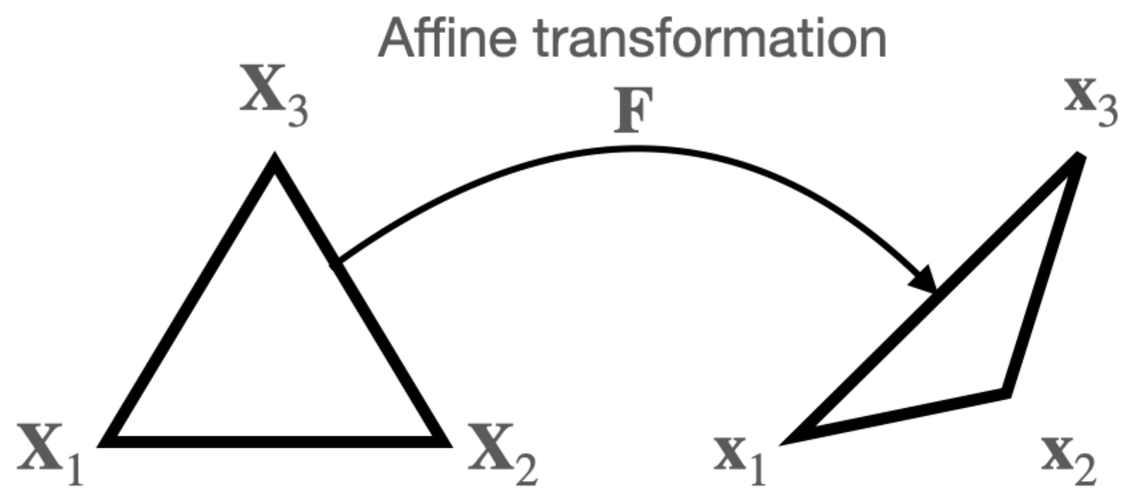
- Particle-Grid Transfer
- Force Calculation
- Grid Updates
- Particle Advection

Force Calculation

Total Lagrangian and Updated Lagrangian

FEM: $f = \frac{\partial \Psi}{\partial \mathbf{F}} \frac{\partial \mathbf{F}}{\partial x}$

$$\mathbf{F} = \frac{\partial \mathbf{x}}{\partial(\beta,\gamma)}\left(\frac{\partial \mathbf{X}}{\partial(\beta,\gamma)}\right)^{-1} \approx \frac{\partial \hat{\mathbf{x}}}{\partial(\beta,\gamma)}\left(\frac{\partial \mathbf{X}}{\partial(\beta,\gamma)}\right)^{-1}$$
$$= [\mathbf{x}_2 - \mathbf{x}_1, \mathbf{x}_3 - \mathbf{x}_1][\mathbf{X}_2 - \mathbf{X}_1, \mathbf{X}_3 - \mathbf{X}_1]^{-1},$$



— Total Lagrangian

For MPM:

- No triangle elements
- To handle large deformation, don't want to use material space $\mathbf{X}$ as reference

- **Track deformation gradient per particle using updated Lagrangian:** $\mathbf{F}^{n+1} \approx \mathbf{F}^n + \Delta t \frac{\partial \mathbf{F}}{\partial t}$

$$\frac{\partial}{\partial t}\mathbf{F}(\mathbf{X}, t^{n+1}) = \frac{\partial \mathbf{V}}{\partial \mathbf{X}}(\mathbf{X}, t^{n+1}) = \frac{\partial \mathbf{v}^{n+1}}{\partial \mathbf{x}}(\phi(\mathbf{X}, t^n))\mathbf{F}(\mathbf{X}, t^n)$$

$$\mathbf{F}_p^{n+1} = \mathbf{F}_p^n + \Delta t \frac{\partial \mathbf{v}^{n+1}}{\partial \mathbf{x}}(\mathbf{x}_p^n)\mathbf{F}_p^n = \left(\mathbf{I} + \Delta t \frac{\partial \mathbf{v}^{n+1}}{\partial \mathbf{x}}(\mathbf{x}_p^n)\right)\mathbf{F}_p^n$$

Force Calculation

Deformation Gradient Calculation

$$F_p^{n+1} = F_p^n + \Delta t \frac{\partial v^{n+1}}{\partial x}(x_p^n) F_p^n = \left(I + \Delta t \frac{\partial v^{n+1}}{\partial x}(x_p^n) \right) F_p^n$$

if we use the grid based interpolation formula for v^{n+1}, i.e.,

$$v^{n+1}(x) = \sum_i v_i^{n+1} N_i(x),$$

$$\frac{\partial v^{n+1}}{\partial x}(x) = \sum_i v_i^{n+1} \left(\frac{\partial N_i}{\partial x}(x) \right)^T,$$

$$\boxed{F_p^{n+1} = \left(I + \Delta t \sum_i v_i^{n+1} \left(\frac{\partial N_i}{\partial x}(x_p^n) \right)^T \right) F_p^n}$$

Temporal discretization with backward difference:

$$v_i^{n+1} = \frac{\hat{x}_i - x_i}{\Delta t} \quad \text{or} \quad \hat{x}_i = x_i + \Delta t v_i^{n+1}$$

$$F_{p\,\beta\gamma}(\hat{x}) = F^n_{p\,\beta\gamma} + \Delta t \sum_j \left(\frac{\hat{x}_{j\beta} - x_{j\beta}}{\Delta t} \right) \sum_\tau N_{j,\tau}(x_p^n) F^n_{p\,\tau\gamma}$$

— a grid-to-particle process

Force Calculation

Using the Derivative of Deformation Gradient

$$\boxed{\mathbf{F}_p^{n+1} = \left(\mathbf{I} + \Delta t \sum_i \boldsymbol{v}_i^{n+1} \left(\frac{\partial N_i}{\partial \boldsymbol{x}}(\boldsymbol{x}_p^n)\right)^T\right) \mathbf{F}_p^n}$$

Temporal discretization with backward difference:

$$F_{p\,\beta\gamma}(\hat{\boldsymbol{x}}) = F^n_{p\,\beta\gamma} + \Delta t \sum_j \left(\frac{\hat{x}_{j\beta} - x_{j\beta}}{\Delta t}\right) \sum_\tau N_{j,\tau}(\boldsymbol{x}_p^n) F^n_{p\,\tau\gamma}$$

1st Piola-Kirchoff Stress — **Particle volume**

$$\frac{\partial e}{\partial \hat{x}_{i\alpha}}(\hat{\boldsymbol{x}}) = \sum_p \sum_{\beta,\gamma} \boxed{P_{\beta\gamma}(\mathbf{F}_p(\hat{\boldsymbol{x}}))} \frac{\partial F_{p\,\beta\gamma}}{\partial \hat{x}_{i\alpha}}(\hat{\boldsymbol{x}}) \boxed{V_p^0}$$

$$\frac{\partial F_{p\,\beta\gamma}}{\partial \hat{x}_{i\alpha}}(\hat{\boldsymbol{x}}) = \delta_{\alpha\beta} \sum_\tau N_{i,\tau}(\boldsymbol{x}_p^n) F^n_{p\,\tau\gamma}$$

⬇

$$\frac{\partial e}{\partial \hat{x}_{i\alpha}}(\hat{\boldsymbol{x}}) = \sum_p P_{\alpha\gamma}(\mathbf{F}_p(\hat{\boldsymbol{x}})) F^n_{p\,\tau\gamma} N_{i,\tau}(\boldsymbol{x}_p^n) V_p^0.$$

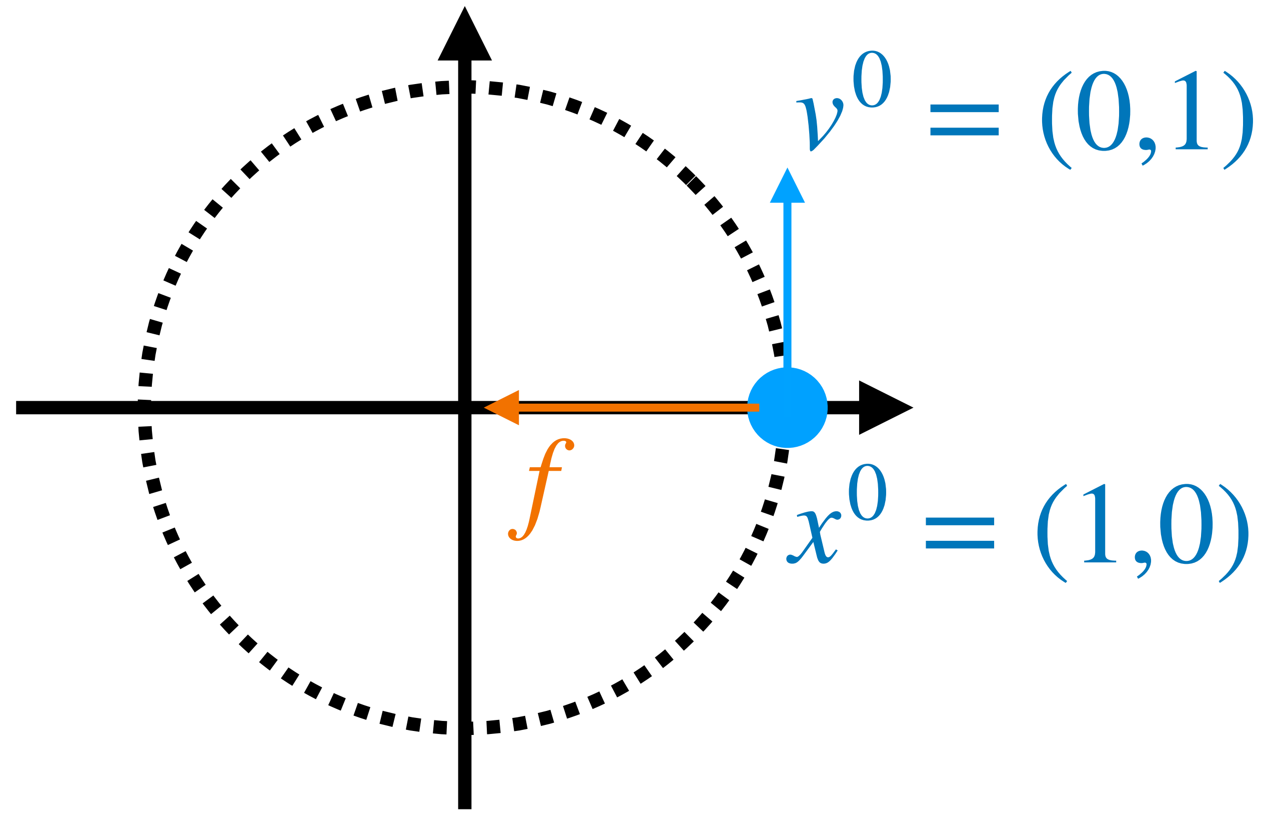
— a particle-to-grid process

Today: The MPM Pipeline

- Particle-Grid Transfer
- Force Calculation
- Grid Updates
- Particle Advection

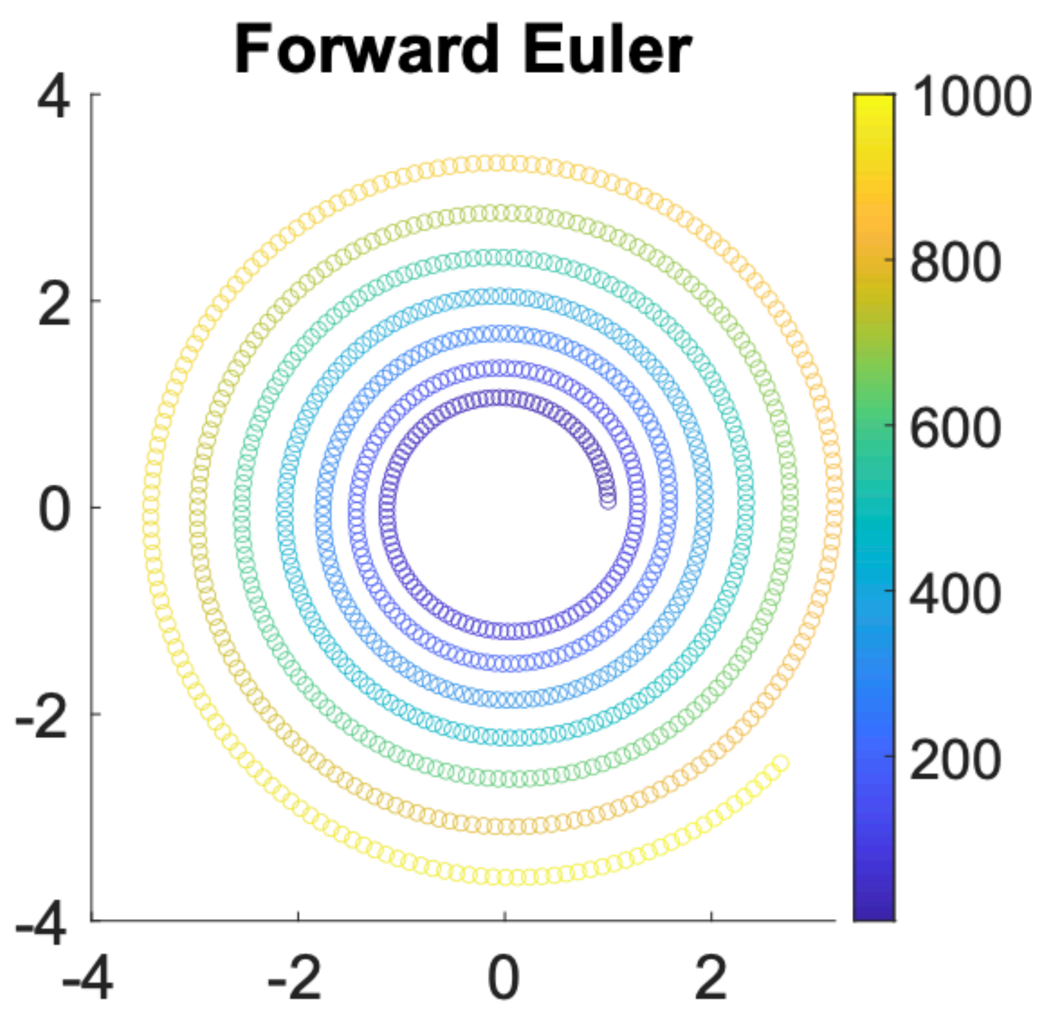
Grid Update (Time Integration)

Recap: Euler Methods



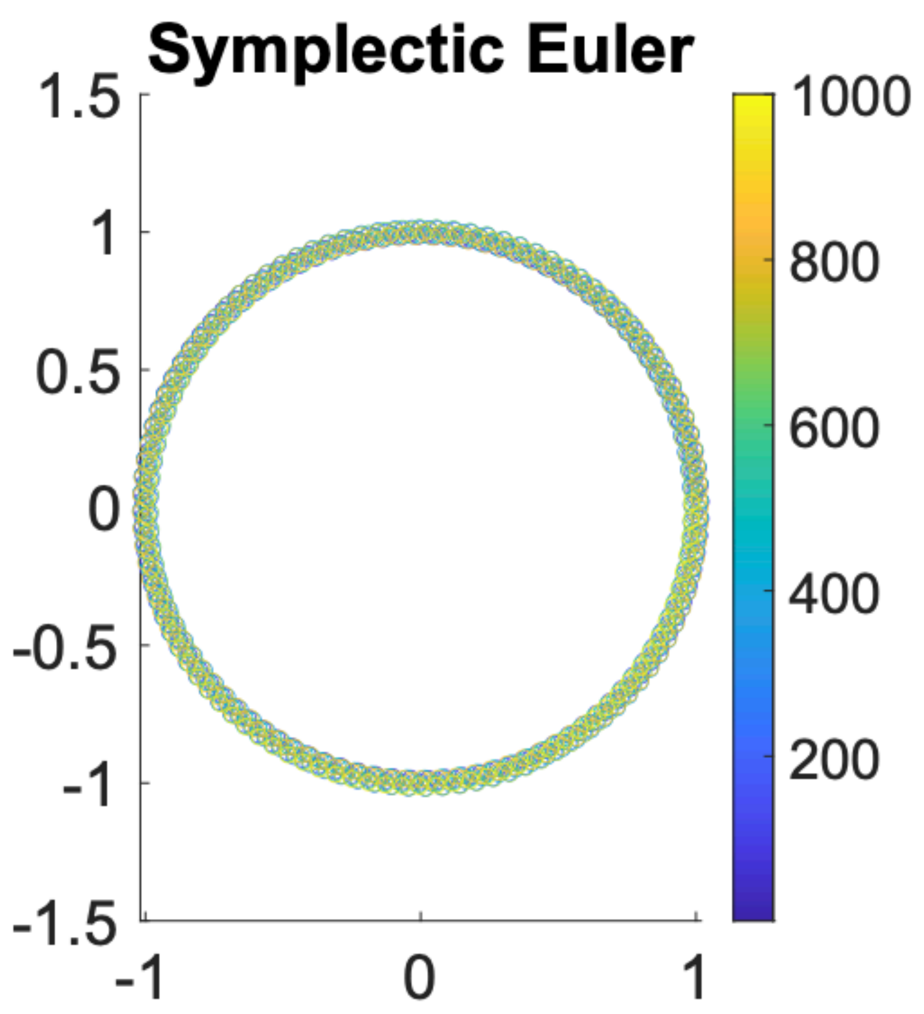
Problem Setup

$$x^{n+1} = x^n + \Delta t v^n, \quad v^{n+1} = v^n + \Delta t M^{-1} f^n$$



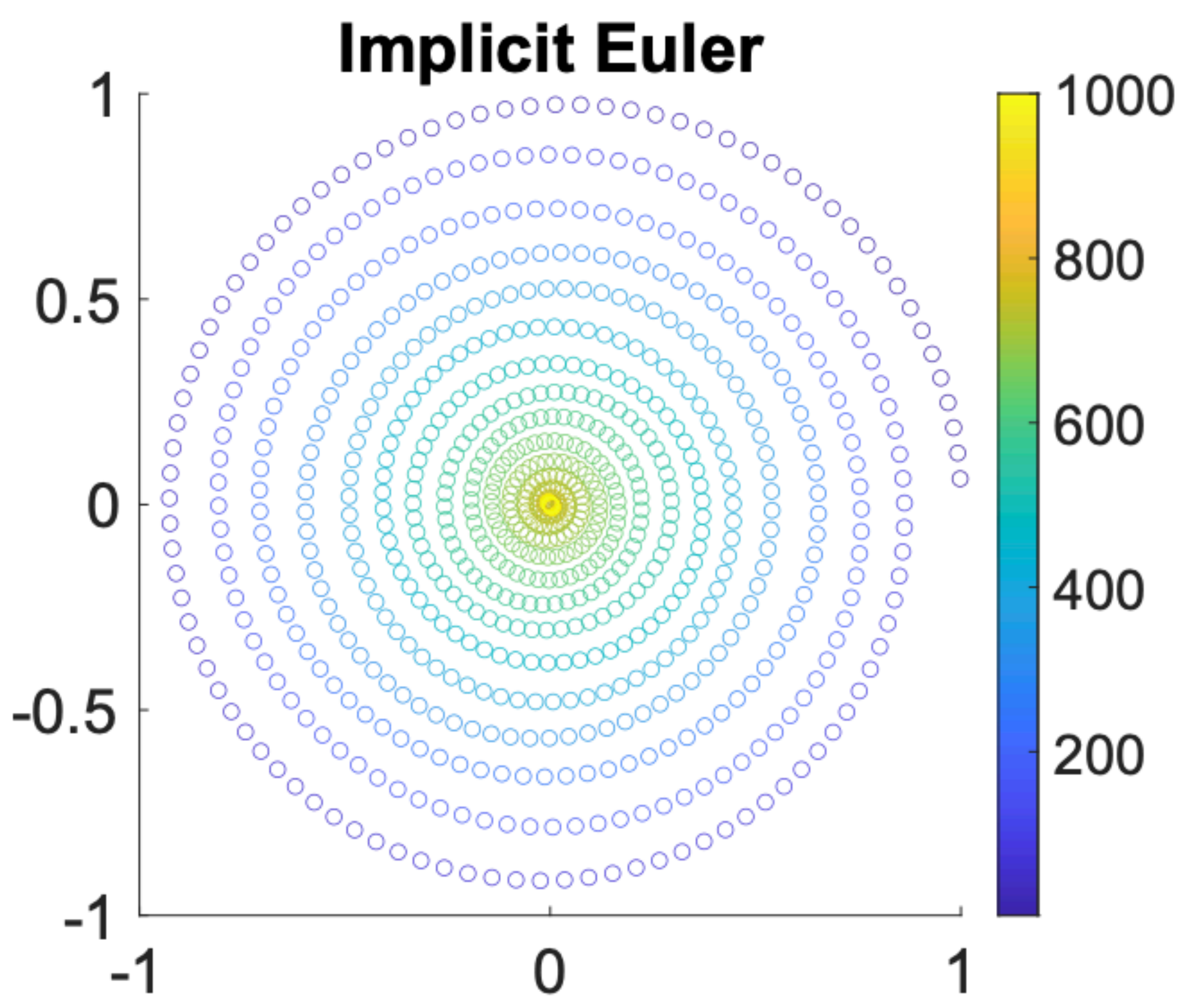
Unconditionally unstable

$$x^{n+1} = x^n + \Delta t v^{n+1} \quad v^{n+1} = v^n + \Delta t M^{-1} f^n$$



Conditionally stable

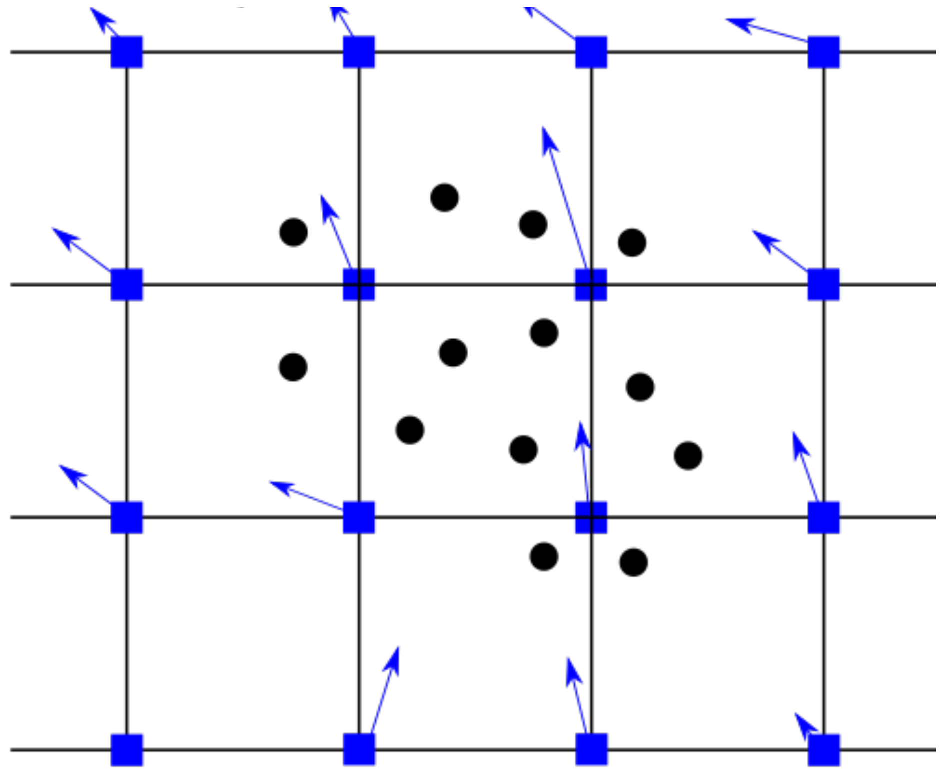
$$x^{n+1} = x^n + \Delta t v^{n+1}, \quad v^{n+1} = v^n + \Delta t M^{-1} f^{n+1}$$



Unconditionally stable

Grid Update (Time Integration)

Symplectic vs Implicit Euler



- Symplectic Euler

Different Same

$$\mathbf{f}_i^n = \mathbf{f}_i(\mathbf{x}_i^n) = -\sum_p V_p^0 \left(\frac{\partial \Psi_p}{\partial \mathbf{F}}(\mathbf{F}_p^n)\right)(\mathbf{F}_p^n)^T \nabla w_{ip}^n$$

  - [+]: efficient, easy to implement, plasticity is straightforward
  - [-]: stable time step size is often small and hard to predict

- Implicit Euler

$$-\mathbf{f}_i(\hat{\mathbf{x}}) = \frac{\partial e}{\partial \hat{\mathbf{x}}_i}(\hat{\mathbf{x}}) = \sum_p V_p^0 \frac{\partial \Psi}{\partial \mathbf{F}}(\hat{\mathbf{F}}_p(\hat{\mathbf{x}}))(\mathbf{F}_p^n)^T \nabla w_{ip}^n$$

  - [+]: time step size only restricted by grid-CFL
  - [-]: needs to implement Hessian (not as sparse as FEM), plasticity is non-trivial, numerical dissipation

Particle Advection

$u^a \leftarrow$ **Solve** $\dfrac{\partial u}{\partial t} + u \cdot \nabla u = 0$

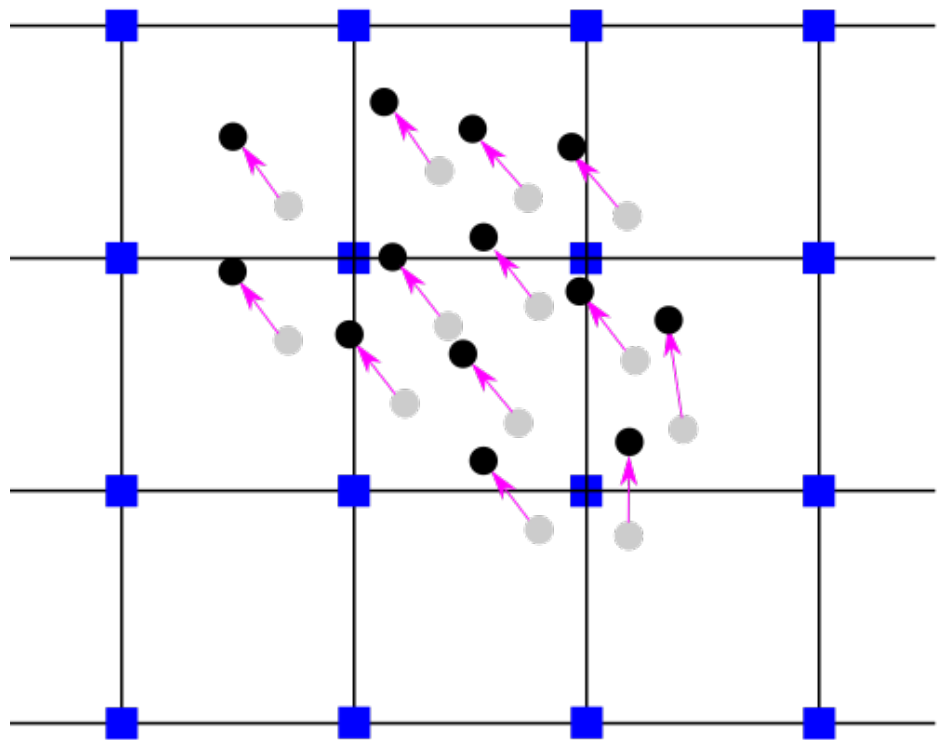
— objects are moving,
resulting in Eulerian velocity changes.

— derived from $\dfrac{du(\phi(\mathbf{X}, t), t)}{dt} = 0$

Our particles are Lagrangian particles!
— each particle marks a fixed region in material space
(Forces are evaluated in an Eulerian view)

Recall that $\mathbf{V}(\mathbf{X}, t) = \mathbf{v}(\phi(\mathbf{X}, t), t)$, so the advection equation becomes $\dfrac{\partial \mathbf{V}(\mathbf{X}, t)}{\partial t} = 0$

Solving advection using particles,
we just need to move the particles based on the current velocity!



Forward Euler: $\boxed{\mathbf{x}_p \leftarrow \mathbf{x}_p + \Delta t \mathbf{u}(\mathbf{x}_p, t)}$

Can use explicit Runge-Kutta, e.g. RK4, for higher accuracy.
These are popular in fluids, but not often used in MPM.

The MPM Pipeline (w. PIC transfer)

For each time step n with particle states $\mathbf{x}_p^n$, $\mathbf{v}_p^n$, $\mathbf{F}_p^n$:

// Particle-to-Grid Transfer:

$$m_i^n = \sum_p m_p N_i(\mathbf{x}_p^n) \qquad (m_i^n \mathbf{v}_i^n) = \sum_p m_p \mathbf{v}_p^n N_i(\mathbf{x}_p^n)$$

// Grid Update:

$$\mathbf{f}_i^n = \mathbf{f}_i(\mathbf{x}_i^n) = -\sum_p V_p^0 \left(\frac{\partial \Psi_p}{\partial \mathbf{F}}(\mathbf{F}_p^n) \right) (\mathbf{F}_p^n)^T \nabla w_{ip}^n$$

or

$$-\mathbf{f}_i(\hat{\mathbf{x}}) = \frac{\partial e}{\partial \hat{\mathbf{x}}_i}(\hat{\mathbf{x}}) = \sum_p V_p^0 \frac{\partial \Psi}{\partial \mathbf{F}}(\hat{\mathbf{F}}_p(\hat{\mathbf{x}}))(\mathbf{F}_p^n)^T \nabla w_{ip}^n$$

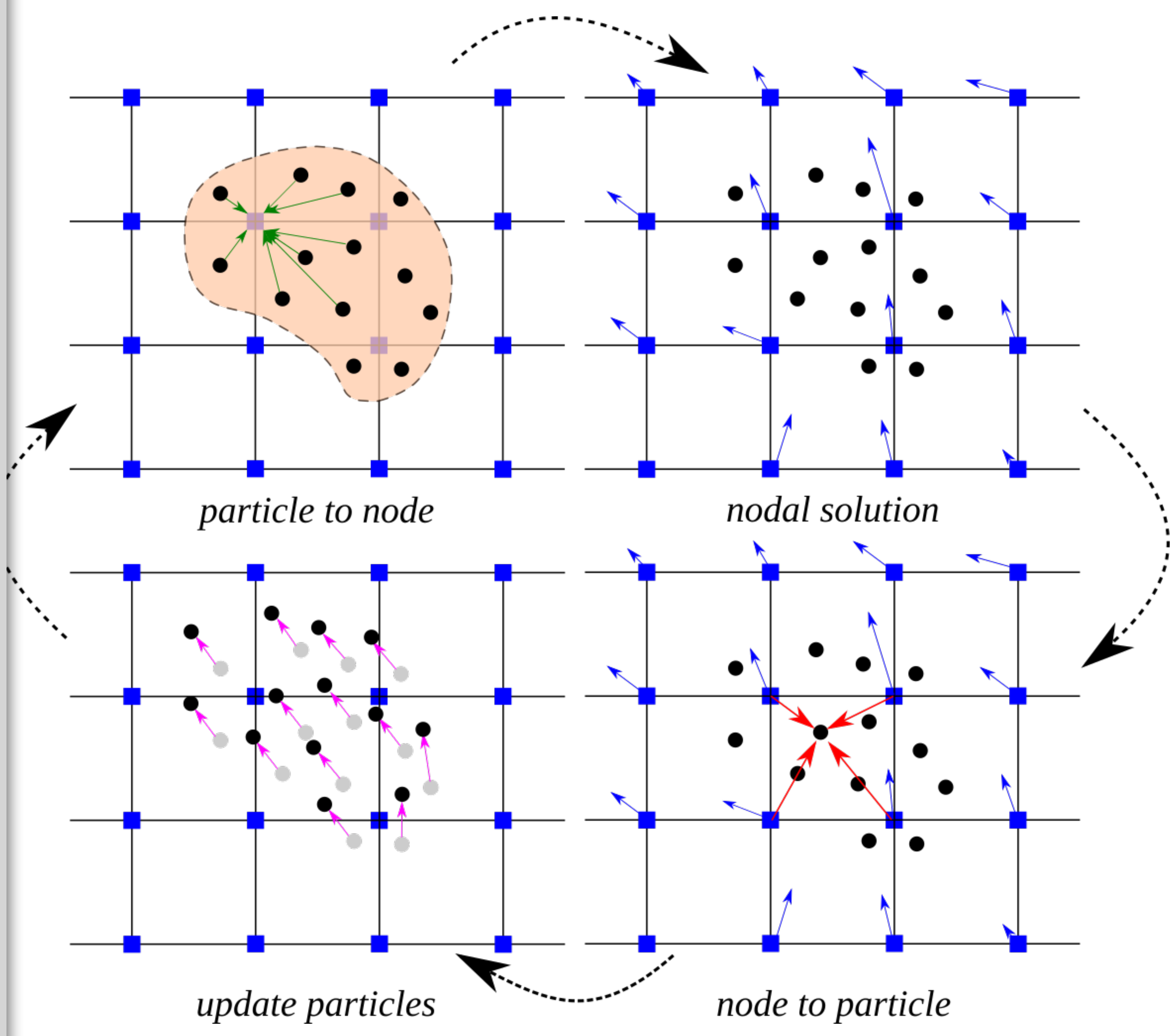
Solve $$\begin{cases} \hat{\mathbf{v}}_i = \mathbf{v}_i^n + \Delta t \mathbf{f}_i \\ \hat{\mathbf{x}}_i = \mathbf{x}_i^n + \Delta t \hat{\mathbf{v}}_i \end{cases}$$

// Grid-to-Particle Transfer:

$$\mathbf{v}_p^{n+1} = \sum_i \hat{\mathbf{v}}_i N_i(\mathbf{x}_p^n) \qquad \mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \sum_i \hat{\mathbf{v}}_i (\frac{\partial N_i}{\partial \mathbf{x}}(\mathbf{x}_p^n))^T)$$

// Particle Advection:

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1}$$



MPM Open-Source Projects

- C++:
  - Ziran: ductile fracture, viscoelastic solids, fluids, etc.
    - Ziran2019 [Wolper et al. 2019] [Fang et al. 2019]
    - Ziran2020 [Wolper et al. 2020] [Fang et al. 2020]
  - HOT: Hierarchical Optimization Time Integration for Implicit MPM [Wang et al. 2020]
- CUDA-based GPU explicit MPM:
  - Single-GPU: https://github.com/kuiwuchn/GPUMPM [Gao et al. 2018]
  - Multi-GPU: https://github.com/penn-graphics-research/claymore [Wang et al. 2020]
- Python-based differentiable explicit MPM: Taichi MPM, Warp MPM

This is the last lecture

Image Sources

- https://www.math.ucla.edu/~cffjiang/research/mpmcourse/mpmcourse.pdf
- https://sph-tutorial.physics-simulation.org/
- https://nheri-simcenter.github.io/Hydro-Documentation/common/technical_manual/desktop/hydro/mpm/mpm.html
- https://en.wikipedia.org/wiki/Bilinear_interpolation