Material Point Methods: A Hands-on Tutorial

GAMES 201 Lecture 8

Yuanming Hu

MIT CSAIL

July 27, 2020

Table of Contents

A little bit of MPM theory (in graphics)

Just like FEM, MPM belongs to the family of Galerkin methods. There are **no elements** in MPM, so MPM $\in$ Element-free Galerkin (EFG).

- MPM particles correspond to FEM quadrature points, instead of elements. MPM typically uses one-point quadrature rule.
- MPM equations are derived using **weak formulation**.

Table of Contents

Moving Least Squares MPM (MLS-MPM)

TL; DR: use MLS shape function in MPM.

- Originally proposed in SIGGRAPH 2018[1].
- Further improved in the SIGGRAPH Asia 2019 Taichi paper[2] to save memory bandwidth.
- Faster and easier to implement than classical B-spline MPM.
- Reason for simplicity and performance: MPM almost always uses the APIC transfer scheme, and MLS-MPM reuses APIC as much as possible.

[1] Y. Hu et al. (2018). "A moving least squares material point method with displacement discontinuity and two-way rigid body coupling". In: *ACM Transactions on Graphics (TOG)* 37.4, pp. 1–14.

[2] Y. Hu et al. (2019). "Taichi: a language for high-performance computation on spatially sparse data structures". In: *ACM Transactions on Graphics (TOG)* 38.6, pp. 1–16.

Notations

In this lecture,

- Scalars are non-bold. E.g., m_i and V_p^0.
- Vectors/matrices are bold lower-/upper-case letters respectively. E.g., $\mathbf{v}_p$ and $\mathbf{C}_p$.
- Subscript i for grid nodes; p for particles. E.g., $\mathbf{v}_i$ and $\mathbf{v}_p$.
- Superscripts are for time steps, e.g. $\mathbf{x}_p^n$ and $\mathbf{x}_p^{n+1}$.

Recap: Affine Particle-in-Cell[3] for incompressible fluids

1. Particle to grid (P2G)
   - $(m\mathbf{v})_i^{n+1} = \sum_p w_{ip}[m_p\mathbf{v}_p^n + m_p\mathbf{C}_p^n(\mathbf{x}_i - \mathbf{x}_p^n)]$ (Grid momentum)
   - $m_i^{n+1} = \sum_p m_p w_{ip}$ (Grid mass)
2. Grid operations
   - $\hat{\mathbf{v}}_i^{n+1} = (m\mathbf{v})_i^{n+1}/m_i^{n+1}$ (Grid velocity)
   - Apply Chorin-style pressure projection: $\mathbf{v}^{n+1} = \textbf{Project}(\hat{\mathbf{v}}^{n+1})$
3. Grid to particle (G2P)
   - $\mathbf{v}_p^{n+1} = \sum_i w_{ip}\mathbf{v}_i^{n+1}$ (Particle velocity)
   - $\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2}\sum_i w_{ip}\mathbf{v}_i^{n+1}(\mathbf{x}_i - \mathbf{x}_p^n)^T$ (Particle velocity gradient)
   - $\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t\mathbf{v}_p^{n+1}$ (Particle position)

[3] C. Jiang, C. Schroeder, and J. Teran (2017). "An angular momentum conserving affine-particle-in-cell method". In: *Journal of Computational Physics* 338, pp. 137–164.

(Explicit) Moving Least Squares MPM (MLS-MPM)

1. Particle to grid (P2G)
   - $\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \mathbf{C}_p^n)\mathbf{F}_p^n, \dots$ **(Deformation update)**
   - $(m\mathbf{v})_i^{n+1} = \sum_p w_{ip}\{m_p\mathbf{v}_p^n + [m_p\mathbf{C}_p^n - \frac{4\Delta t}{\Delta x^2}\sum_p V_p^0 \mathbf{P}(\mathbf{F}_p^{n+1})(\mathbf{F}_p^{n+1})^T](\mathbf{x}_i - \mathbf{x}_p^n)\}$ **(Grid momentum)**
   - $m_i^{n+1} = \sum_p m_p w_{ip}$ (Grid mass)
2. Grid operations
   - $\hat{\mathbf{v}}_i^{n+1} = (m\mathbf{v})_i^{n+1}/m_i^{n+1}$ (Grid velocity)
   - $\mathbf{v}_i^{n+1} = \text{BC}(\hat{\mathbf{v}}_i^{n+1})$ (Grid boundary condition. BC is the boundary condition operator.)
3. Grid to particle (G2P)
   - $\mathbf{v}_p^{n+1} = \sum_i w_{ip}\mathbf{v}_i^{n+1}$ (Particle velocity)
   - $\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2}\sum_i w_{ip}\mathbf{v}_i^{n+1}(\mathbf{x}_i - \mathbf{x}_p^n)^T$ (Particle velocity gradient)
   - $\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1}$ (Particle position)

Note that in classical B-spline MPM, deformation update usually happens after G2P.

Deformation update

Deformation gradients evolve because $\nabla \mathbf{v} = \left.\frac{\partial \mathbf{v}^n}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_p^n} \neq \mathbf{0}$.
(Local velocity field is not constant, so the material keeps deforming.)
Evaluating new deformation gradients:

$$\mathbf{F}_p^{n+1} = \left(\mathbf{I} + \Delta t \nabla \mathbf{v}\right) \mathbf{F}_p^n. \tag{1}$$

In MLS-MPM, APIC $\mathbf{C}_p^n$ is used as an approximation of $\nabla \mathbf{v}$.
Therefore in MLS-MPM we have

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \mathbf{C}_p^n) \mathbf{F}_p^n. \tag{2}$$

P2G: Computing internal forces

Recall that $(m\mathbf{v})_i^n = \sum_p w_{ip}\{m_p\mathbf{v}_p^n + [m_p\mathbf{C}_p^n - \frac{4\Delta t}{\Delta x^2}\sum_p V_p^0\mathbf{P}(\mathbf{F}_p^{n+1})(\mathbf{F}_p^{n+1})^T](\mathbf{x}_i - \mathbf{x}_p^n)\}$ (Grid momentum).

Two momentum terms:

- APIC: $w_{ip}[m_p\mathbf{v}_p^n + m_p\mathbf{C}_p^n(\mathbf{x}_i - \mathbf{x}_p^n)]$
- Particle elastic force (impulse): $\Delta t\mathbf{f}_{ip} = -w_{ip}\frac{4\Delta t}{\Delta x^2}\sum_p V_p^0\mathbf{P}(\mathbf{F}_p^{n+1})(\mathbf{F}_p^{n+1})^T](\mathbf{x}_i - \mathbf{x}_p^n)$

Assuming hyperelastic materials. Deriving $\mathbf{f}_i$ using potential energy gradients:

$$U = \sum_p V_p^0 \psi_p(\mathbf{F}_p) \tag{3}$$

$$\mathbf{f}_i = -\frac{\partial U}{\partial \mathbf{x}_i} \tag{4}$$

ψ_p: elastic energy density of particle p; U: total elastic potential energy.
V_p^0: particle initial volume.

P2G: Computing nodal force $\mathbf{f}_i$

Assume we move forward $\tau \to 0$, and then compute deformed grid node location $\hat{\mathbf{x}}_i = \mathbf{x}_i + \tau \mathbf{v}_i$, $\mathbf{C}_p = \frac{4}{\Delta x^2}\sum_i w_{ip}\mathbf{v}_i(\mathbf{x}_i - \mathbf{x}_p)^T$, updated $\mathbf{F}'_p = (\mathbf{I} + \tau\mathbf{C}_p)\mathbf{F}_p$:

$$
\begin{aligned}
\mathbf{f}_i &= -\frac{\partial U}{\partial \mathbf{x}_i} = -\sum_p V_p^0 \frac{\partial \psi(\mathbf{F}'_p)}{\partial \hat{\mathbf{x}}_i} && (5)\\
&= -\sum_p \frac{V_p^0}{\tau} \frac{\partial \psi_p(\mathbf{F}'_p)}{\partial \mathbf{v}_i} && (6)\\
&= -\sum_p \frac{V_p^0}{\tau} \frac{\partial \psi(\mathbf{F}'_p)}{\partial \mathbf{F}'_p} \frac{\partial \mathbf{F}'_p}{\partial \mathbf{C}_p} \frac{\partial \mathbf{C}_p}{\partial \mathbf{v}_i^n} && (7)\\
&= -\sum_p \frac{V_p^0}{\tau} \mathbf{P}_p(\mathbf{F}'_p) \cdot \tau \mathbf{F}_p^T \cdot \frac{4 w_{ip}}{\Delta x^2}(\mathbf{x}_i - \mathbf{x}_p) && (8)\\
&= -\frac{4}{\Delta x^2} \sum_p V_p^0 \mathbf{P}(\mathbf{F}'_p) \cdot \mathbf{F}_p^T w_{ip}(\mathbf{x}_i - \mathbf{x}_p) && (9)
\end{aligned}
$$

Grid operations: enforcing boundary conditions (BC)

BC in MPM should be applied on the grid. For all grid nodes i within the boundary:

$$\mathbf{v}_i^{n+1} = \quad \text{BC}_{\text{sticky}}(\hat{\mathbf{v}}_i^{n+1}) \quad = \mathbf{0} \tag{10}$$

$$\mathbf{v}_i^{n+1} = \quad \text{BC}_{\text{slip}}(\hat{\mathbf{v}}_i^{n+1}) \quad = \hat{\mathbf{v}}_i^{n+1} - \mathbf{n}(\mathbf{n}^T\hat{\mathbf{v}}_i^{n+1}) \tag{11}$$

$$\mathbf{v}_i^{n+1} = \quad \text{BC}_{\text{separate}}(\hat{\mathbf{v}}_i^{n+1}) \quad = \hat{\mathbf{v}}_i^{n+1} - \mathbf{n}\cdot\min(\mathbf{n}^T\hat{\mathbf{v}}_i^{n+1}, 0) \tag{12}$$

($\mathbf{n}$: surface normal)

Extras:

1. Adding gravity $\hat{\mathbf{v}}_i^{n+1} += \Delta t\mathbf{g}$
2. Moving collision object
3. Coulomb Friction

Summary: benefits of MLS-MPM

Why is MLS-MPM (SIGGRAPH 2018) easier and faster than classical B-spline MPM (SIGGRAPH 2013)?

1. Directly reuse APIC $\mathbf{C}_p$ as an approximation of $\nabla \mathbf{v}$ for deformation gradient update. No need to evaluate ∇w_{ip} (Fewer FLOPs)
2. Easy to move deformation update from G2P to P2G, because we only need $\mathbf{C}_p$ for deformation update. (Fewer bytes to fetch from main memory)
3. In P2G, APIC and MLS-MPM momentum contribution can be fused, since they are both "MLS". (Fewer FLOPs)

MLS-MPM is consistent with the weak formulation of the Cauchy momemtum equation. See the original MLS-MPM paper[4] for a correctness proof.

[4] Y. Hu et al. (2018). "A moving least squares material point method with displacement discontinuity and two-way rigid body coupling". In: *ACM Transactions on Graphics (TOG)* 37.4, pp. 1–14.

Table of Contents

Constitutive Models

Common constitutive models in MPM:

1. Elastic objects: NeoHookean & Corotated
2. Fluid: Equation-of-States (EOS)
3. Elastoplastic objects (snow, sand etc.): Yield criteria: ad-hoc boxing[5], Cam-clay[6], Drucker-prager[7], NACC, ...

Two critical aspects of a constitutive model in MPM:

1. (Elastic/plastic) deformation update
2. (PK1) stress evaluation

[5] A. Stomakhin et al. (2013). "A material point method for snow simulation". In: *ACM Transactions on Graphics (TOG)* 32.4, pp. 1–10.

[6] J. Gaume et al. (2018). "Dynamic anticrack propagation in snow". In: *Nature communications* 9.1, pp. 1–10.

[7] G. Klár et al. (2016). "Drucker-prager elastoplasticity for sand animation". In: *ACM Transactions on Graphics (TOG)* 35.4, pp. 1–12.

Constitutive models for elastic solids

Deformation update: simply $\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \mathbf{C}_p^n)\mathbf{F}_p^n$.

PK1 stresses of hyperelastic material models:

- Neo-Hookean:
  - $\psi(\mathbf{F}) = \frac{\mu}{2}\sum_i[(\mathbf{F}^T\mathbf{F})_{ii} - 1] - \mu\log(J) + \frac{\lambda}{2}\log^2(J)$.
  - $\mathbf{P}(\mathbf{F}) = \frac{\partial\psi}{\partial\mathbf{F}} = \mu(\mathbf{F} - \mathbf{F}^{-T}) + \lambda\log(J)\mathbf{F}^{-T}$
- (Fixed) Corotated:
  - $\psi(\mathbf{F}) = \mu\sum_i(\sigma_i - 1)^2 + \frac{\lambda}{2}(J-1)^2$. σ_i are singular values of $\mathbf{F}$.
  - $\mathbf{P}(\mathbf{F}) = \frac{\partial\psi}{\partial\mathbf{F}} = 2\mu(\mathbf{F} - \mathbf{R}) + \lambda(J-1)J\mathbf{F}^{-T}$

Cauchy stress $\boldsymbol{\sigma} = \frac{1}{J}\mathbf{P}\mathbf{F}^T$ is usually unused in MPM.

More details: check out the SIGGRAPH 2016 MPM course[8].

[8] C. Jiang et al. (2016). "The material point method for simulating continuum materials". In: *ACM SIGGRAPH 2016 Courses*, pp. 1–52.

Constitutive models weakly compressible fluids[9]

Volume ratio $J_p = V_p^n / V_p^0 = \det(\mathbf{F}_p^n)$.
The simplest equation of state: $p = K(1 - J)$, Cauchy stress $\boldsymbol{\sigma} = -p\mathbf{I}$. K : bulk modulus.

Computing $\det(\mathbf{F}_p^n)$ can be numerically unstable.

Recall that for $\mathbf{F}_{2\times 2}$, $\det(\mathbf{F}) = \mathbf{F}_{00}\mathbf{F}_{11} - \mathbf{F}_{01}\mathbf{F}_{10}$. The "$-$" opeartion leads to **catastrophic cancellation**. Same for $\mathbf{F}_{3\times 3}$ (Question: why doesn't this happen to NeoHookean/corotated materials?)

Deformation update: instead of maintaining $\mathbf{F}_p$,directly maintain $J_p^n = \det(\mathbf{F}_p^n)$:

$$
\begin{aligned}
\mathbf{F}_p^{n+1} &= (\mathbf{I} + \Delta t \mathbf{C}_p^n)\mathbf{F}_p^n && (13)\\
\Rightarrow \det(\mathbf{F}_p^{n+1}) &= \det(\mathbf{I} + \Delta t \mathbf{C}_p)\det(\mathbf{F}_p^n) && (14)\\
\Rightarrow J_p^{n+1} &= (1 + \Delta t \mathbf{tr}(\mathbf{C}_p^n))J_p^n && (15)
\end{aligned}
$$

[9] A. P. Tampubolon et al. (2017). "Multi-species simulation of porous sand and water mixtures". In: *ACM Transactions on Graphics (TOG)* 36.4, pp. 1–11.

Simulating weakly compressible fluids (lazy solution)

Setting μ to zero in (Fixed) corotated model.
(Recap) In corotated materials:

- $\psi(\mathbf{F}) = \mu \sum_i (\sigma_i - 1)^2 + \frac{\lambda}{2}(J-1)^2$. σ_i are singular values of $\mathbf{F}$.
- $\mathbf{P}(\mathbf{F}) = \frac{\partial \psi}{\partial \mathbf{F}} = 2\mu(\mathbf{F} - \mathbf{R}) + \lambda(J-1)J\mathbf{F}^{-T}$

Recap: Singular value decomposition (SVD)

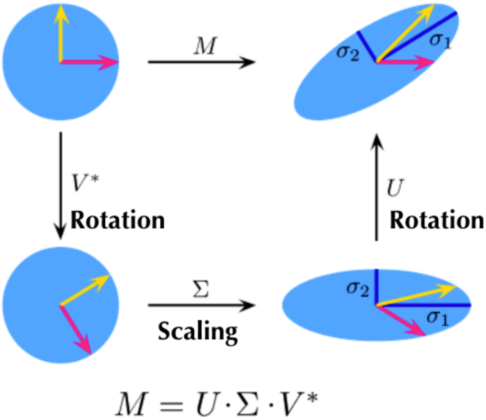
Theorem

(Existence of singular value decompositions) Every real matrix $\mathbf{M}_{n\times m}$ *can be decomposed into* $\mathbf{M}_{n\times m} = \mathbf{U}_{n\times n}\mathbf{\Sigma}_{n\times m}\mathbf{V}^T_{m\times m}$*, where* $\mathbf{U}$ *and* $\mathbf{V}$ *are orthonormal matrices, and* $\mathbf{\Sigma}$ *is a diagonal matrix.*

Diagonal entries $\sigma_i = \Sigma_{ii}$ are called **singular values**.

To learn more about linear algebra: check out Gilbert Strang's MIT OCW.

SVD: Intuition



(Source: Wikipedia)

2×2 and 3×3 SVD[10] in Taichi

Note that SVD is not unique. We additionally require

- $\det(\mathbf{U}) = \det(\mathbf{V}) = 1$.
- $|\mathbf{\Sigma}_{ii}|$ are sorted in decreasing order.
- Only the singular value with smallest magnitude can be negative.

Example

```
U, sig, V = ti.svd(M) # sig is an NxN diagonal matrix.
```

[10] A. McAdams et al. (2011). *Computing the singular value decomposition of 3x3 matrices with minimal branching and elementary floating point operations*. Tech. rep. University of Wisconsin-Madison Department of Computer Sciences.

Simulating elastoplastic solids

In hyperelastic settings:

$$\mathbf{F}_p = \mathbf{F}_{p,\textbf{elastic}}\mathbf{F}_{p,\textbf{plastic}}, \ \psi_p^n = \psi(\mathbf{F}_{p,\textbf{elastic}}),$$

i.e., the potential energy penalizes elastic deformation only.

Example

"Box" yield criterion[11]: deformation udpate:

1. Evolve $\hat{\mathbf{F}}_p^{n+1} = (\mathbf{I} + \Delta t\mathbf{C}_p^n)\mathbf{F}_{p,\textbf{elastic}}^n$
2. SVD: $\hat{\mathbf{F}}_p^{n+1} = \mathbf{U}\hat{\mathbf{\Sigma}}\mathbf{V}^T$
3. Clamping: $\mathbf{\Sigma}_{ii} = \max(\min(\hat{\mathbf{\Sigma}}_{ii}, 1+\theta_s), 1-\theta_c)$ (forget about too large deformations)
4. Reconstruct: $\mathbf{F}_{p,\textbf{elastic}}^{n+1} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$; move clamped parts to $\mathbf{F}_{p,\textbf{plastic}}^{n+1}$

[11] A. Stomakhin et al. (2013). "A material point method for snow simulation". In: *ACM Transactions on Graphics (TOG)* 32.4, pp. 1–10.

Table of Contents

Lagrangian forces in MPM[12]

TL; DR: Treat MPM particles as FEM vertices, and use FEM potential energy model. A triangular mesh is needed.

Benefits:

- (Compared to FEM): Self-collision is handled on the grid;
- (Compared to MPM): Numerical fracture is avoided due to the mesh connectivity.
- Can easily couple MPM and FEM.

Easy to implement in Taichi using AutoDiff: `ti example mpm_lagrangian_forces`

[12] C. Jiang et al. (2015). "The affine particle-in-cell method". In: *ACM Transactions on Graphics (TOG)* 34.4, pp. 1–10.

Introducing Taichi "field"

Upgrading Taichi: `pip install --upgrade taichi==0.6.22`

Use "**field**" instead of "tensor" since Taichi `v0.6.22`

- The name "tensor" is deprecated. Always use "field" instead.
- `ti.var` is deprecated. Use `ti.field` instead.
- Argument `dt` is deprecated. Use `dtype` instead.

Declaring fields in Taichi

```
# particle_x = ti.Vector(3, dt=ti.f32, shape=1024)
particle_x = ti.Vector.field(3, dtype=ti.f32, shape=1024)
particle_F = ti.Matrix.field(3, 3, dtype=ti.f32, shape=1024)
# density = ti.var(dtype=ti.f32, shape=(256, 256))
density = ti.field(dtype=ti.f32, shape=(256, 256))
num_springs = ti.field(dtype=ti.i32, shape=())
```

Fields := global variables in Taichi

Distinguishing global fields from local variables

Global variables are always declared with "field". Local variables are always declared without "field":

```
x = ti.Vector.field(3, dtype=ti.f32, shape=(128, 512)) # global

@ti.kernel
def foo():
    a = ti.Vector([0.2, 0.4]) # local
```

The word "field" refers to ...

1. a component of a (database) record. For example, `mass` and `volume` properties of a particle array.
2. a (physical) quantity that is assigned to every point in space. E.g., "velocity fields", and "magnetic fields". High-dimensional arrays of scalars/vectors/matrices are exactly "fields" sampled at discrete grid points.