# A Moving Least Squares Material Point Method with Displacement Discontinuity and Two-Way Rigid Body Coupling (Supplementary Document)

Yuanming Hu Yu Fang Ziheng Ge Ziyin Qu Yixin Zhu
Andre Pradhana Chenfanfu Jiang

## 1 Equivalence of APIC/PolyPIC and MLS on velocities

### 1.1 MLS with a linear polynomial basis

In the simple case of a complete linear polynomial basis ($m = l = 1$) for meshless solids, MLS is a $1^{\text{st}}$-order-consistent interpolation scheme for scattered data and derivatives. With $\mathbf{P}(\mathbf{x}_i - \mathbf{x}) = [1, (\mathbf{x}_i - \mathbf{x})^T]^T$, the reconstructed function value and its gradient estimation at $\mathbf{x}$ are

$$\begin{bmatrix} \hat{u} \\ \nabla \hat{u} \end{bmatrix} = \mathbf{M}^{-1}(\mathbf{x})\mathbf{Q}^T \Xi(\mathbf{x}) \begin{bmatrix} u_1 \\ \vdots \\ u_N \end{bmatrix}, \tag{1}$$

where $N$ is the total number of sample data points, $\Xi(\mathbf{x})$ is the diagonal weighting matrix with $\Xi_{ii} = \xi_i(\mathbf{x})$, and $\mathbf{Q}(\mathbf{x}) = [\mathbf{P}(\mathbf{x}_1 - \mathbf{x}), \ldots, \mathbf{P}(\mathbf{x}_N - \mathbf{x})]^T$, $\mathbf{M}(\mathbf{x}) = \mathbf{Q}^T \Xi \mathbf{Q}$.

### 1.2 APIC and MLS

APIC [Jiang et al., 2015] represents the local velocity field near each particle $p$ with $\mathbf{v}_p$ and matrix $\mathbf{C}_p$. At time $n$, the momentum transfer

$$m_i^n \mathbf{v}_i^n = \sum_p m_p N_i(\mathbf{x}_p^n) \left( \mathbf{v}_p^n + \mathbf{C}_p^n (\mathbf{x}_i - \mathbf{x}_p^n) \right)$$

is equivalent to accumulating momentum contributions from particle-wise local polynomial (affine) velocity fields to the grid. When we choose the weighting function $\xi_i(\mathbf{x})$ to be $N_i(\mathbf{x})$ where $N_i(\mathbf{x})$ is the B-spline kernel, grid-to-particle transfer is exactly evaluating Eq. 1 after grid velocity is integrated from $\mathbf{v}_i^n$ to $\hat{\mathbf{v}}_i^{n+1}$. To see that, we first write down APIC transfers

$$\mathbf{v}_p^{n+1} = \sum_i N_i(\mathbf{x}_p^n) \hat{\mathbf{v}}_i^{n+1}, \tag{2}$$

$$\mathbf{C}_p^{n+1} = \left( \sum_i N_i(\mathbf{x}_p^n) \hat{\mathbf{v}}_i^{n+1} (\mathbf{x}_i - \mathbf{x}_p^n)^T \right) \mathbf{D}_p^{-1}, \tag{3}$$

where $\mathbf{D}_p = \sum_i N_i(\mathbf{x}_p^n)(\mathbf{x}_i - \mathbf{x}_p^n)(\mathbf{x}_i - \mathbf{x}_p^n)^T$. The equivalence to Eq. 1 relies on realizing that, in the MLS perspective, $\mathbf{v}_p^{n+1}$ is the reconstructed function value $\hat{u}$ and $\mathbf{C}_p^{n+1}$ is the reconstructed function gradient $\nabla \hat{u}$.

In this case, the sample data $\{u_1, \ldots, u_N\}$ in Eq. 1 corresponds to the velocities of all grid nodes influenced by particle $p$. Correspondingly we have

$$\mathbf{Q}^T(\mathbf{x}_p^n) = \begin{bmatrix} 1 & (\mathbf{x}_1 - \mathbf{x}_p^n)^T & 1 & (\mathbf{x}_2 - \mathbf{x}_p^n)^T & \ldots \end{bmatrix}$$

and

$$\mathbf{M}(\mathbf{x}_p^n) = \sum_i \xi_i(\mathbf{x}_p) \begin{bmatrix} 1 & (\mathbf{x}_i - \mathbf{x}_p^n)^T \\ (\mathbf{x}_i - \mathbf{x}_p^n) & (\mathbf{x}_i - \mathbf{x}_p^n)(\mathbf{x}_i - \mathbf{x}_p^n)^T \end{bmatrix}. \tag{4}$$

Substituting them into Eq. 1 and replacing $\xi_i$ with $N_i$ results in a block-wise result containing Eq. 2 and Eq. 3.

It is not immediately clear what the relationship is between $\mathbf{D}_p$ in APIC and $\mathbf{M}(\mathbf{x}_p)$ in MLS. Note that in other meshless methods like EFG, $\mathbf{M}$ can be singular if neighboring data samples are co-planar, co-linear, or insufficient, in which case the pseudo-inverse is required. Note that $\mathbf{M}(\mathbf{x}_p)$ in 3D is a general $4 \times 4$ matrix (in linear polynomial space) if no restriction is put on $\xi_i(\mathbf{x})$. However when $\sum_i \xi_i(\mathbf{x}) = 1$ and $\sum_i \xi_i(\mathbf{x})(\mathbf{x}_i - \mathbf{x}) = \mathbf{0}$, we can see that $\mathbf{M}(\mathbf{x}_p)$ in Eq. 4 is block diagonal with its top left entry being just 1.

The choice of quadratic/cubic B-splines for $N_i(\mathbf{x})$ makes $\mathbf{M}(\mathbf{x}_p)$ even simpler. In fact, it is diagonal for APIC. The top left element is simply 1, while the bottom right block $\mathbf{D}_p$ is $\frac{1}{4}\Delta x^2 \mathbf{I}$ for quadratic $N_i(\mathbf{x})$ and $\frac{1}{3}\Delta x^2 \mathbf{I}$ for cubic $N_i(\mathbf{x})$, where $\Delta x$ is the grid spacing. The simple form for $\mathbf{D}_p$ comes from the properties of B-splines, and is not generally true for other interpolation functions.

### 1.3 PolyPIC and MLS

A related discussion without explicitly mentioning the notion of MLS is given by Fu et al. [2017] for PolyPIC. Here we briefly point out the main idea using the MLS view. PolyPIC uses a higher order basis for $\mathbf{P}(\mathbf{x})$. While particle-to-grid is also accumulating momentum contributions from particle-wise local polynomial velocity fields, the grid-to-particle transfer for each particle is essentially evaluating basis coefficients $\mathbf{c}(\mathbf{x})$ for minimizing the functional $J_{\mathbf{x}}(\mathbf{c})$. As discussed in the paper, $J_{\mathbf{x}}(\mathbf{c})$ represents a weighted average error encoding the difference between actual data samples $\hat{\mathbf{v}}_i^{n+1}$ and their approximation through the polynomial basis.

In practice PolyPIC stores coefficients $\mathbf{c}(\mathbf{x}_p)$ on each particle. The size of this coefficient vector depends on the dimension of the polynomial space. When only linear functions are used, PolyPIC reduces to APIC, and the coefficients simply contain $\mathbf{v}_p^{n+1}$ and all components in $\mathbf{C}_p^{n+1}$. When higher order basis functions are added to $\mathbf{P}(\mathbf{x})$ (such as multilinear functions and quadratic functions), PolyPIC provides a more accurate polynomial fit to the local velocity field. Notice that Fu et al. [2017] additionally use Gram-Schmidt to enforce a mass orthogonal quadratic basis to reach a diagonal moment matrix $\mathbf{M}(\mathbf{x})$, which is highly relevant to the classical idea of using an orthogonal polynomial basis in EFG methods [Lu et al., 1994].

## 2 MLS-MPM force differential

Computing the MLS-MPM force differential requires treating the force as a function of fictitiously deformed grid node positions ($\mathbf{x}_i = \mathbf{x}_i^n + \Delta t \hat{\mathbf{v}}_i$ where $\hat{\mathbf{v}}_i$ are the implicit unknown grid velocities and $\mathbf{x}_i^n$ are regular Cartesian lattice locations). Specifically,

$$\mathbf{f}_i(\mathbf{x}_i) = -\sum_p N_i(\mathbf{x}_p^n) V_p^0 D_p^{-1} \frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}_p(\mathbf{x}_i)) \mathbf{F}_p^{nT}(\mathbf{x}_i^n - \mathbf{x}_p^n), \tag{5}$$

where the implicit deformation gradient update is related to $\mathbf{x}_i$ as

$$\mathbf{F}_p(\mathbf{x}_i) = (\mathbf{I} + \Delta t \mathbf{C}_p(\mathbf{x}_i))\, \mathbf{F}_p^n, \tag{6}$$

and $\mathbf{C}$ is given by

$$\mathbf{C}_p(\mathbf{x}_i) = \left( \sum_i N_i(\mathbf{x}_p^n) \frac{\mathbf{x}_i - \mathbf{x}_i^n}{\Delta t} (\mathbf{x}_i^n - \mathbf{x}_p^n)^T \right) \mathbf{M}_p^{-1}, \tag{7}$$

where $\mathbf{M}_p$ is a constant in the case of quadratic/cubic B-spline $N_i(\mathbf{x})$.

Based on these relationships, the differentiation can be carried out directly using the chain rule. Implicit time stepping in MLS-MPM requires computing the action of the energy Hessian on an arbitrary grid increment $\delta\mathbf{u}$ (just like traditional MPM).

$$\begin{aligned}
-f_{i\alpha} &= \sum_p V_p^0 \frac{\partial \Psi}{\partial F_{\alpha\gamma}} F^{nT}_{\gamma\sigma} D_p^{-1} N_i(\mathbf{x}_p^n)(x_{i\sigma}^n - x_{p\sigma}^n) \\
-\delta f_{i\alpha} &= \sum_j \frac{\partial^2 E}{\partial x_{i\alpha} \partial x_{j\beta}} \delta u_{j\beta} \\
-\delta f_{i\alpha} &= \sum_p V_p^0 \sum_j \frac{\partial^2 \Psi}{\partial F_{\alpha\gamma} \partial F_{\omega\eta}} \frac{\partial F_{\omega\eta}}{\partial x_{j\beta}} \delta u_{j\beta} F^{nT}_{\gamma\sigma} D_p^{-1} N_i(\mathbf{x}_p^n)(x_{i\sigma}^n - x_{p\sigma}^n) \\
-\delta f_{i\alpha} &= \sum_p V_p^0 \sum_j \frac{\partial^2 \Psi}{\partial F_{\alpha\gamma} \partial F_{\omega\eta}} \left( D_p^{-1} N_j(\mathbf{x}_p^n)(x_{j\xi}^n - x_{p\xi}^n) F_{\xi\eta}^n \delta_{\omega\beta} \right) \delta u_{j\beta} F^{nT}_{\gamma\sigma} D_p^{-1} N_i(\mathbf{x}_p^n)(x_{i\sigma}^n - x_{p\sigma}^n) \\
-\delta f_{i\alpha} &= \sum_p V_p^0 \sum_j \frac{\partial^2 \Psi}{\partial F_{\alpha\gamma} \partial F_{\omega\eta}} \left( D_p^{-1} N_j(\mathbf{x}_p^n)(x_{j\xi}^n - x_{p\xi}^n) F_{\xi\eta}^n \right) \delta u_{j\omega} F^{nT}_{\gamma\sigma} D_p^{-1} N_i(\mathbf{x}_p^n)(x_{i\sigma}^n - x_{p\sigma}^n),
\end{aligned} \tag{8}$$

therefore,

$$-\delta \mathbf{f}_i = \sum_p V_p^0 \mathbf{A}_p \mathbf{F}_p^{nT} D_p^{-1} N_i(\mathbf{x}_p^n)(\mathbf{x}_i^n - \mathbf{x}_p^n), \tag{9}$$

where

$$\mathbf{A}_p = \frac{\partial^2 \Psi}{\partial \mathbf{F} \partial \mathbf{F}} : \sum_j D_p^{-1} N_j(\mathbf{x}_p^n) \delta \mathbf{u}_j (\mathbf{x}_j^n - \mathbf{x}_p^n)^T \mathbf{F}_p^n. \tag{10}$$

# 3 High-performance implementation details

**Benchmark settings**

We uniformly sample 8 particles per cell, with $\Delta x = 8 \times 10^{-3}$, in region $[0.1, 0.9]^3$. This results in a $100^3$ grid and 8M particles. APIC transfer and a linear elasticity constitutive model is used for simplicity. To get more accurate results and cover the total run time better, we make P2G and G2P cover more operations:

"P2G":

- Calculate force based on deformation gradient
- Scatter mass, particle elasticity force and APIC affine momentum to grid nodes
- Normalize grid velocity and apply gravity

"G2P":

- Gather velocity from grid
- Gather velocity gradient (if necessary) and the APIC affine velocity field
- Update deformation gradient
- Advect particles

**Implementation details**

Our discussion is focused on 3D cases, which are significantly more computationally expensive than 2D ones. To make discussions concrete, we further assume 32-bit floating-point precision and an Intel Core CPU with an architecture later than Haswell (released 2013), though most ideas can be used in other settings or architectures. We further focus the implementation on quadratic B-splines, the smallest kernel that makes all constitutive models stable.

## 3.1 Algorithmic improvement

As discussed in the main body of our paper, MLS-MPM halves the number of FLOPs needed for each particle. The unification of affine velocity field and deformation gradient eliminates the need for $\nabla N_i(\mathbf{x}_p^n)$, which speeds up both P2G and G2P. During P2G, MLS-MPM fuses the scattering of the affine momentum and particle force contribution into

$$N_i(\mathbf{x}_p^n)\mathbf{Q}_p(\mathbf{x}_i - \mathbf{x}_p^n),$$

where

$$\mathbf{Q}_p = \Delta t V_p^0 M_p^{-1} \frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}_p^n)\mathbf{F}_p^{nT} + m_p \mathbf{C}_p,$$

so that only one matrix-vector multiplication is needed for the inner loop (27 iterations for 3D and quadratic B-splines). During deformation gradient update in G2P, MLS-MPM avoids evaluating $\frac{\partial \mathbf{v}}{\partial \mathbf{x}}$ with $\nabla N_i(\mathbf{x})$ and can directly re-use $\mathbf{C}_p$ from APIC.

## 3.2 Performance engineering

When the algorithm keeps the same, the FLOPs required for each particle is at the same level for comparable implementations. In this case, the performance of such a compute-intensive program is roughly proportional to floating point unit (FPU) utilization, which means how many floating-point operations are actually done per cycle in the back-end of CPUs.

Our optimization goal is to (1) reduce the memory bandwidth consumption and to (2) improve floating point unit (FPU) utilization, while (3) keeping the implementation simple. Our high-performance MPM solver is designed based on these three principles. Note that due to issues such as memory alignment, compilers may not automatically generate vectorized instructions. In our implementation, we do that manually via CPU intrinsics.

As a result, our low-level performance engineering significantly improves FPU utilization by reducing memory bandwidth bound and unnecessary run-time computations such as memory addresses generation. As noted in the main paper, our P2G and G2P implementations lead to 1.93× and 7.38× FPU utilization compared with [Tampubolon et al., 2017].

**Sparse grid storage and blocked transfer**

SPGrid [Setaluri et al., 2014] is used for background grid storage. SPGrid cleverly uses Morton coding and the translation lookaside buffer (TLB) to maximize locality and minimize access latency. Grid nodes are arranged in the unit of blocks, and our block size is $4 \times 4 \times 8$. Notably, the blocked nature of SPGrid effectively reduces memory bandwidth usage during transfer. Padded local grids of size $6 \times 6 \times 10$ nodes are used as arenas for transfers, where particles within the block region will only read from and write to their arenas.

Before P2G or G2P, we gather local arena information from the global SPGrid and write back the change only after P2G. This strategy fits grid data into the L1 cache for maximum throughput and minimum latency. We call different optimized kernels for kernels with and without rigid boundary particles. Doing so raises the branching from particle level to block level, which maximizes the power of speculative execution. Practically,

only a narrow band of rigid boundaries needs CPIC. Typically, around 90% blocks use regular MLS-MPM transfers.

**Avoiding data race in P2G**

Even though P2G will be executed on local grid arenas only, data race will still be an issue when two neighboring local grid arenas are writing back to the same global grid node. Locking is one straightforward solution. We, however, want to avoid locking for performance and ease of implementation. To this end, one solution is to divide the blocks into separate groups where no global nodes are shared by two or more blocks in one group. In this way, processing blocks in a single group in parallel has no data race. One way to achieve such grouping is via the block coordinates, which are defined to be a Cartesian coordinate system in the unit of SPGrid blocks. We group blocks according to the parity of block coordinates (all axes considered jointly), leading to 4 groups in 2D and 8 groups in 3D.

**Low-level performance engineering**

We explicitly write CPU intrinsics to force the compiler to generate desired instructions. Fused multiply-add (FMA) instructions that compute $\mathbf{a} \times \mathbf{b} + \mathbf{c}$ are particularly useful in our case. Note that $\mathbf{a}, \mathbf{b}$ and $\mathbf{c}$ are vectors of four 32-bit floating point numbers, which can be used as 3D vectors. The remaining entry can be mass, which is fully utilized during P2G.

Compilers are conservative regarding unrolling, especially for a relatively large loop (27 elements for 3D quadratic B-spline transfers). In our case, manually unrolling the loops not only saves the cost of maintaining loop variables but also allows the compiler to generate array access offsets at compile time.

# 4 3D printed robot

Overall, the robot performs one degree-of-freedom rotation in an alternating tripod gait. A stable 4.5V is applied to the motor to ensure constant rotation speed. Specifically, following the descriptions in [Li et al., 2013], we use the same robot (RoboXplorer, Smart Lab) as the main body. Six legs are 3D printed using ABS material and attached to the main body. These legs have the same length of 2R = 4.1 cm, width = 1.0 cm, and thickness = 0.3 cm, but only two different curvatures $1/r = [-1, 1]/R$ are used in our experiments. The belly area of the main body is $13 \times 2cm^2$, and the total weight is 85g. The track used for the experiment is 180 cm long and 40 cm wide, filled with fine-grained sand 8 cm deep. Two cameras are used to record the motion of the robot from the front and the top at 30 FPS.

# References


C. Fu, Q. Guo, T. Gast, C. Jiang, and J. Teran. A polynomial particle-in-cell method. *ACM Trans Graph*, 36(6):222:1–222:12, 2017.

C. Jiang, C. Schroeder, A. Selle, J. Teran, and A. Stomakhin. The affine particle-in-cell method. *ACM Trans Graph*, 34(4):51:1–51:10, 2015.

C. Li, T. Zhang, and D. Goldman. A terradynamics of legged locomotion on granular media. *Science*, 339 (6126):1408–1412, 2013.

Y. Lu, T. Belytschko, and L. Gu. A new implementation of the element-free galerkin method. *Computer methods in applied mechanics and engineering*, 113(3-4):397–414, 1994.

R. Setaluri, M. Aanjaneya, S. Bauer, and E. Sifakis. Spgrid: A sparse paged grid structure applied to adaptive smoke simulation. *ACM Trans Graph*, 33(6):205, 2014.

A. P. Tampubolon, T. Gast, G. Klár, C. Fu, J. Teran, C. Jiang, and K. Museth. Multi-species simulation of porous sand and water mixtures. *ACM Trans Graph*, 36(4), 2017.