ACM SIGGRAPH / Eurographics Symposium on Computer Animation 2018
T. Beeler and N. Thuerey
(Guest Editors)

Volume 37 (2018), Number 8

A Temporally Adaptive Material Point Method with Regional Time Stepping

Yu Fang[†1], Yuanming Hu[†2], Shi-Min Hu[1], Chenfanfu Jiang[3]
(† equal contribution)
[1]Tsinghua University, Department of Computer Science, China
[2]Massachusetts Institute of Technology, CSAIL, United States
[3]University of Pennsylvania, SIG Center for Computer Graphics, United States

Abstract

Spatially and temporally adaptive algorithms can substantially improve the computational efficiency of many numerical schemes in computational mechanics and physics-based animation. Recently, a crucial need for temporal adaptivity in the Material Point Method (MPM) is emerging due to the potentially substantial variation of material stiffness and velocities in multi-material scenes. In this work, we propose a novel temporally adaptive symplectic Euler scheme for MPM with regional time stepping (RTS), where different time steps are used in different regions. We design a time stepping scheduler operating at the granularity of small blocks to maintain a natural consistency with the hybrid particle/grid nature of MPM. Our method utilizes the Sparse Paged Grid (SPGrid) data structure and simultaneously offers high efficiency and notable ease of implementation with a practical multi-threaded particle-grid transfer strategy. We demonstrate the efficacy of our asynchronous MPM method on various examples including elastic objects, granular media, and fluids.

CCS Concepts

•***Computing methodologies*** → ***Physical simulation;***

1. Introduction

Since its usage in Disney's *Frozen* [SSC*13], the Material Point Method (MPM) [SZS95] has been adopted in animating various physical materials in graphics ranging from granular soil [KGP*16, DBD16] to garment [JGT17], as a generalization of FLIP fluids [ZY10]. MPM inherently benefits from a hybrid particle/grid representation which enables the *automatical coupling* of history-dependent continuum materials in topologically complex scenarios. Consequently, MPM is becoming a promising multi-material multi-phase scheme in computer graphics [JST*16]. By virtue of the high regularity of the grid and the unique memory coherence of exceedingly localized memory footprint in each MPM time step, there exists a strong potential to push the performance of explicitly integrated MPM to a high level on emerging architectures.

Adaptive methods have a long history in graphics [MWN*17]. Spatial and temporal adaptivities for MPM follow a clear principle: computational resources can and should be concentrated in regions demanding stability, high accuracy, and vibrant details. Recently, Gao et al. [GTJS17] explored the option of using the SPGrid data structure [SABS14] for spatially adaptive background grid with dynamic particle resampling. They demonstrated the performance gain on various examples involving free surface and thin features.

In this work, we fill in the other crucial component: temporal adaptivity. Similarly to the idea for SPH by Goswami and Batty [GB14], we use regional time stepping (RTS) at different locations of the simulated domain. The time step size in the commonly used symplectic Euler integration is restricted by CFL [CFL28] conditions considering both the sound speed and advection. We derive practical time step bounds for the hyperelastic and fluid materials used in our work (§4). We design a time stepping scheduler operating at the granularity of small blocks to maintain a natural consistency with the hybrid particle/grid nature of MPM. Our algorithm robustly supports practical large deformation scenarios when particles move across grid cell blocks. Additionally, we present a parallelization strategy based on SPGrid that further speeds up simulation, which is not only efficient but also easy to implement.

Our contributions can be summarized as follows:

- Theoretical derivations of practical time step bounds that ensures simulation stability for explict time integration;
- A regional time stepping scheme for MPM enabling temporal adaptivity;
- A highly efficient parallelization strategy for block-wise transfers in MPM based on the SPGrid.

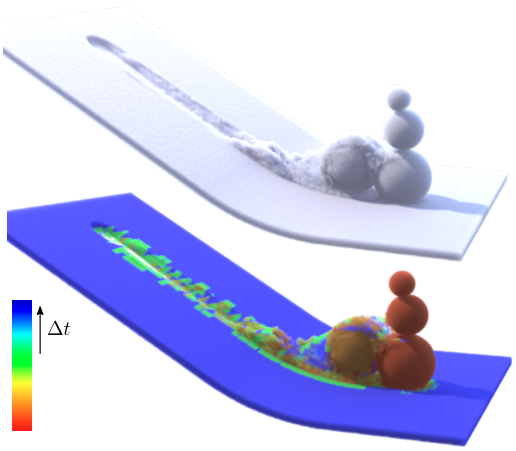


Figure 1: *Snow slope. Small time step for elastoplastic hardening.*

2. Related Work

It has been well demonstrated that both spatial adaptivity (such as local remeshing and hp-adaptivity) and adaptive time-stepping contribute substantially to simulation efficiency. Manteaux et al. [MWN*17] comprehensively reviewed existing work on them. Here we focus on those that are most related to this work.

Limiting time step size based on maximum material velocity is used by Foster and Fedkiw [FF01] as a way of enforcing the CFL condition for Eulerian liquids, and by Monaghan [Mon92] and Desbrun et al. [DC99] for SPH fluids. Adaptive selection of the global time step is used for cloth (and cloth contact) by Baraff and Witkin [BW98] and Bridson et al. [BFA02]. For simulating viscoplastic flow with remeshing, Bargteil et al. [BWHT07] used a Newmark integrator with adaptive time step as in [BMF03].

Our method focuses on adaptive regional time stepping. This core idea has been investigated for SPH by Desbrun et al. [DC99]. Our method is mostly related to the work on SPH by Goswami and Batty [GB14] where local time steps are chosen in a block-wise fashion. More recently, Reinhardt et al. [RHEW17] presented a fully asynchronous integration method for SPH where per-particle time step is independent throughout the whole simulation. For mesh-based Lagrangian simulation, Debunne et al. [DDCB00] used the stable timestep restriction to choose adaptive local time steps on a finite element mesh for elasticity. Thomaszewski et al. [TPS08] used the asynchronous variational integrator (AVI) for cloth, allowing per-element time step chosen according to the stability criterion of linear elasticity. Asynchronous time integration was also applied to coupled implicit/semi-implicit [SKZF11] and fully implicit backward Euler [ZLB16] simulations for nonlinear elasticity with a focus on controlling numerical damping.

Explicitly integrated MPM has been mostly applied to problems that easily need scaling up to tens of millions of grid degrees of freedom and hundreds of millions of Lagrangian particles. Due to its ease of performance optimization, explicit MPM in fact often requires less CPU time than implicit MPM (with an exception on simulating extremely stiff materials such as concrete) and provides higher accuracy since it causes less numerical damping. Advances in computational platform hardware have also revealed opportunities for transformative improvements in scale and performance of MPM. Zhang et al. [ZZL10] used an alternated grid updating algorithm to avoid data racing in particle-to-grid. Their domain is partitioned along a fixed direction and only scales well when the simulated material occupies a large bulk of the volume. Ruggirello et al. [RS14] compared two parallelization approaches for MPM with and without using ghost particles. Chiang et al. [CDH*09] proposed a GPU acceleration scheme for the Generalized Interpolation Material Point Method (GIMP) [BK04] with careful management of memory bandwidth. In computer graphics, Gao et al. [GTJS17] achieved low-level performance gain using a vectorized interpolation weight computation scheme for GIMP. Explicit MPM can also be combined with semi-implicit fluid solvers to couple particles with imcompressible fluids [GTH*18]. In this work, we describe an efficient scheme to store asynchronous MPM states and a new multi-threading parallelization approach for optimized particle-grid transfers that works for both traditional synchronous MPM (SyncMPM) and proposed asynchronous MPM (AsyncMPM).

3. MPM Background

Here we briefly review synchronous MPM. The material domain at t^n is discretized with particles at $\mathbf{x}_p^n$. Each particle has volume V_p^0, mass m_p, velocity $\mathbf{v}_p^n$, and an affine velocity matrix $\mathbf{C}_p^n$ (initialized to zero, see [JSS*15]). It also holds other physical quantities such as deformation gradient $\mathbf{F}_p^n$, pressure p_p^n, Lamé parameters μ_p, λ_p, yield stress etc. In each time step, a new grid for computation is created. Quantities on the grid nodes are denoted with subscript i, such as position $\mathbf{x}_i$, velocity $\mathbf{v}_i$ and force $\mathbf{f}_i$. Particles and grid nodes are related through B-spline interpolation functions $N_i(\mathbf{x})$, with $w_{ip}^n = N_i(\mathbf{x}_p^n)$. We use MLS-MPM [HFG*18] in this work, which is both simpler and faster than traditional MPM. (Synchronous) MLS-MPM advances to t^{n+1} as

1. **Particle-to-grid transfer.** Particles transfer mass m_p and momentum $(m\mathbf{v})_p^n$ to the grid using APIC [JSS*15]. Grid momentum $(m\mathbf{v})_i^n$ is divided by grid mass m_i^n to get grid velocity $\mathbf{v}_i^n$.
2. **Grid momentum update.** Internal forces are applied to the grid nodes with $\hat{\mathbf{v}}_i^{n+1} = \mathbf{v}_i^n + \Delta t \mathbf{f}_i / m_i^n$. For symplectic Euler, $\mathbf{f}_i = -\sum_p V_p^n \sigma_p w_{ip}^n (\mathbf{x}_i^n - \mathbf{x}_p^n)$, where σ_p is the Cauchy stress evaluated using $\mathbf{F}_p^n$. See [GSS*15] and [HFG*18] for implicit integration.
3. **Grid-to-particle transfer.** The updated $\mathbf{v}_p^{n+1}$ and $\mathbf{C}_p^{n+1}$ are updated on particles using APIC [JSS*15].
4. **Particle constitutive model update.** Particle $\mathbf{F}$ is evolved using $\mathbf{F}_p^{n+1} = \left(\mathbf{I} + \Delta t \mathbf{C}_p^{n+1}\right) \mathbf{F}_p^n$, and gets modified for plasticity if necessary [KGP*16].
5. **Particle advection.** Each particle moves as $\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1}$.

4. Time Step Restriction

4.1. Stability from Elastic Wave Propagation

In the explicit FEM formulation of homogeneous 3D solids, the stable time step size is restricted by a critical time step that is determined by the speed of sound and the element characteristic length.

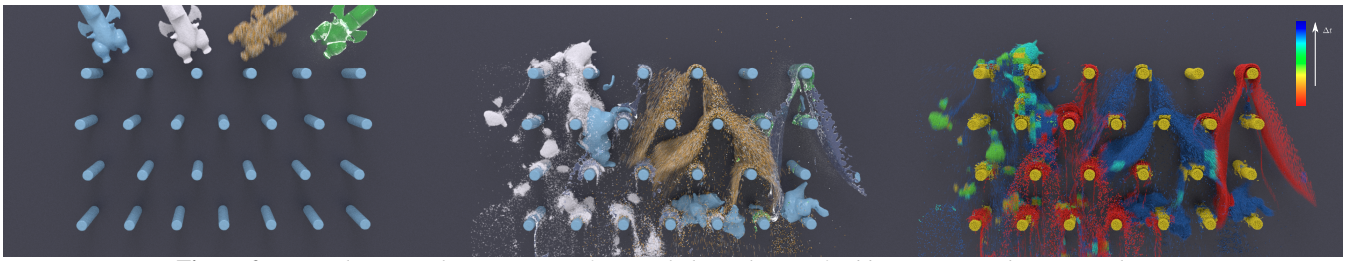


Figure 2: *Versatile regional time stepping for coupled simulation of rubber, snow, sand, water, and goo.*

The underlying principle is that the time step in an explicit simulation must be small enough to accurately capture the transmission of a dilational pressure wave across a character distance of the elements. In the case of MPM, the characteristic distance is conveniently defined to be grid cell spacing Δx. This is, in fact, more convenient than the traditional unstructured FEM meshes where the characteristic length varies across elements and critically depends on element quality. The dilational pressure wave speed is also known as the adiabatic sound speed [KFCS99]. In isotropic homogeneous solids, the pressure wave (also often called P-wave) is always longitudinal. Assuming an elastic media, the P-wave velocity is given by [BH92] $c = \sqrt{\frac{4\mu}{3\rho} + \frac{K_s}{\rho}}$, where μ is the shear modulus which relates Young's modulus E and Poisson's ratio ν through $\mu = \frac{E}{2(1+\nu)}$, ρ is current density, K_s is the adiabatic bulk modulus (or adiabatic incompressibility). Moreover, given an equation of state for a non-linear elastic material, K_s can be estimated with

$$K_s = \rho \frac{\partial p}{\partial \rho}, \tag{1}$$

where p is pressure. Thus we can use

$$c = \sqrt{\frac{4\mu}{3\rho} + \frac{\partial p}{\partial \rho}} \tag{2}$$

to evaluate the sound speed, leading to the critical time step

$$\Delta t_{cr} = \frac{\Delta x}{c}. \tag{3}$$

Then the time step limit can be chosen as $\Delta t \leq \alpha \Delta t_{cr}$ [ZCL16], where $\alpha \in [0.5, 0.9]$ is an additional scale for stabilization.

St. Venant-Kirchhoff Hyperelasticity. The St. Venant-Kirchhoff hyperelasticity model combined with the logarithm Hencky strain measure has been mostly used for granular media [KGP*16, TGK*17] and elastoplastic flow [GTJS17] in graphics. Note that plasticity does not contribute to the pressure wave. The principal Cauchy stress is given by [KGP*16]

$$\hat{\sigma} = \frac{2\mu}{J} \log \Sigma + \frac{\lambda}{J} \mathrm{tr}(\log \Sigma) \mathbf{I}, \tag{4}$$

where μ and λ are Lamé parameters, $J = \det(\mathbf{F})$, Σ comes from the polar SVD [MST*11] $\mathbf{F} = \mathbf{U}\Sigma\mathbf{V}^T$. The pressure is the negative of hydrostatic stress $p = -\frac{1}{3}\hat{\sigma}_{kk} = -K\frac{\log J}{J}$, where $K = \frac{2\mu}{3} + \lambda$ is the bulk Modulus. Correspondingly, by utilizing $J = \frac{\rho_0}{\rho}$ (thus $\frac{\partial J}{\partial \rho} = -\frac{J}{\rho}$) for some initial density ρ_0, we have

$$\frac{\partial p}{\partial \rho} = \frac{\partial p}{\partial J}\frac{\partial J}{\partial \rho} = \frac{K}{\rho_0}(1 - \log J). \tag{5}$$

Note that in the context of MPM particles, ρ_0 for each particle can be computed using its mass m_p and initial volume V_p^0 as $\rho_0 = m_p / V_p^0$. Substituting Eq. 5 back into Eq. 2 reveals

$$c = \sqrt{\frac{4\mu}{3\rho} + \frac{K}{\rho_0}(1 - \log J)}. \tag{6}$$

For linear elasticity, the $\frac{1}{J}$ factor does not appear in the stress expression (Eq. 4). The pressure derivative simplifies to $\frac{\partial p}{\partial \rho} = \frac{K}{\rho}$. Correspondingly we get the sound speed expression

$$c = \sqrt{\frac{E(1-\nu)}{(1+\nu)(1-2\nu)\rho}} \tag{7}$$

that is more commonly seen in linear elasticity analysis. We refer to [ZCL16] for more details on MPM with linear elasticity.

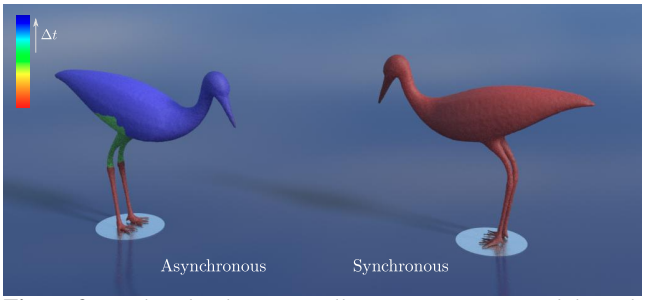
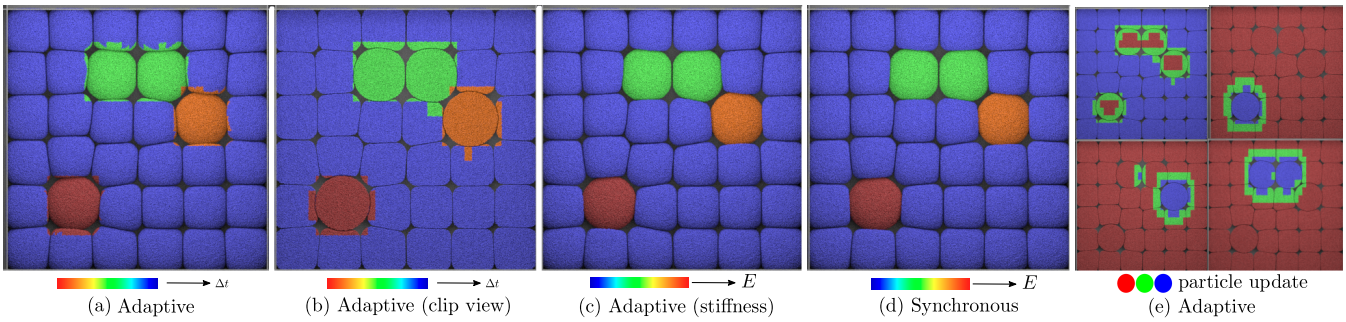


Figure 3: *Storks. They have spatially varying Young's modulus values at different body parts.*

Nearly Incompressible Fluid. Nearly incompressible and weakly compressible equation-of-state (EOS) fluids have been widely used in SPH [BT07]. In MPM, it is also used for simulating liquid in porous media [TGK*17] . The pressure is

$$p = k\left(\left(\frac{\rho}{\rho_0}\right)^{\gamma} - 1\right), \tag{8}$$

where γ is usually chosen to be 7 for nearly incompressible fluids, and 1 for weakly compressible fluids. k is the bulk modulus. Differentiating p with respect to ρ can be done quite easily in this case: $\frac{\partial p}{\partial \rho} = k\gamma\left(\frac{\rho}{\rho_0}\right)^{\gamma-1}$. Also note that the shear modulus $\mu = 0$ for



(a) Adaptive (b) Adaptive (clip view) (c) Adaptive (stiffness) (d) Synchronous (e) Adaptive

Figure 4: *Asynchronous integration of compressing elastic balls. (a) The regional time step sizes and its (b) clip view. (c) Stiffness variation of the balls. (d) The same scene with synchronous integration. (e) Particle update status in the asynchronous scheduler at different (but nearby) time steps. Blue: updated particles; Red: un-updated particles; Green: buffer region.*

inviscid water, thus

$$c = \sqrt{k\gamma\left(\frac{\rho}{\rho_0}\right)^{\gamma-1}} = \sqrt{\frac{k\gamma}{J^{\gamma-1}}}. \tag{9}$$

For weakly compressible fluids ($\gamma = 1$), the sound speed becomes a constant and does not depend on how compressed the fluid is.

4.2. Advection CFL

To avoid material inversion or particles penetrating into each other, MPM additionally requires a CFL restriction on particle advection which is not required in mesh-based FEM simulations. Considering the worst case scenario where two particles are moving toward each other without any resisting force, the restriction can be written as

$$\Delta t \leq \beta \frac{\Delta x}{|\mathbf{v}_p|}, \tag{10}$$

where β is usually chosen to be around 0.5. In first order advection schemes we have $\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \mathbf{v}_p^{n+1}\Delta t$. In practice we must decide Δt before computing $\mathbf{v}_p^{n+1}$, therefore $\mathbf{v}_p^n$ is used instead. This error and additional randomness on particle relative locations may result in some additional slight tweaking of β for each scene.

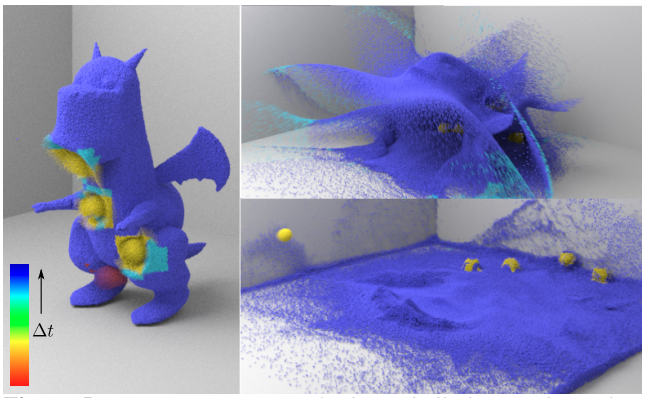


Figure 5: *Dragon massage. Soft elastic balls hit a cohesionless granular dragon. Time step size is jointly determined by material stiffness and particle velocity.*

5. Temporally Adaptive Asynchronous Time Integration

With two factors of time step restriction analyzed, we can now advance the simulation with the largest possible time step. However, in traditional synchronous approaches, the actual time step is limited by the element with the most restrictive time step, i.e., the whole simulation has to slow down due to small regions with extreme time step limits. The versatile nature of MPM makes this issue even more pronounced, since various materials may have considerably different stability behavior. Such limitation highlights the importance of temporal adaptivity: can we use different Δt in different regions? Scheduling, i.e., which parts are updated in what frequency and order, turns out to be the central problem of such adaptive asynchronous time integration, as discussed in this section.

5.1. Spatial and Temporal Granularity

Purely Lagrangian/Eulerian methods such as SPH and FEM typically use individual elements as scheduling units [RHEW17, ZLB16, TPS08]. For example, in [TPS08] where an explicit mesh is used, each element is treated as an independent scheduling unit. MPM, in contrast to the approaches mentioned above, is a hybrid method whose states are stored on particles while force computation happens on a background grid. Such a hybrid representation facilitates additional efficiency and correctness challenges for asynchronous time integration. Updating a single particle can be highly inefficient since all its neighboring particles have to be rasterized to a background grid before it can resample the updated local velocity field. In comparison to synchronous MPM where particles rasterize and resample only once per time step, updating a single particle at a time will lead to $O(N)$ operations per particle per global time step, where N is the average number of neighboring particles. Moreover, if neighboring particles are not at the same time as the updated particle, a synchronization scheme is typically necessary. In SPH or FEM, such synchronization via interpolation is usually easy because element states constitute of only position and velocity, while in MPM interpolating the deformation gradients is neither easy nor efficient.

Therefore, a reasonable choice is to bundle particles in small blocks as a whole. Blocks are small cubes of cells, e.g. $4\Delta x \times$

$8\Delta x \times 8\Delta x$. Performance-wise, because MPM is bounded by transfers, such a block-based transfer implementation is the most efficient approach to the best of our knowledge. Besides, scheduling at this granularity leads to a nice balance between scheduling quality and efficiency. A similar design decision was made in [GB14] where the time integration strategy was an asynchronous predictor-corrector scheme [SDTS03]. Since their approach is focused on liquids, the time step is only determined by particle velocity. In MPM, however, directly adopting such scheme may lead to a complicated correction step because correcting the rich state information stored on particles, especially deformation gradients, can be very challenging. Such difference in MPM and SPH results in our unique scheduling algorithm tailored for MPM that avoids correction.

Since particles carry all the state information of MPM, the state of a block is entirely represented by the states of particles inside. We denote the blocks as $\mathcal{B} = \{B_1, B_2, ...\}$, and each particle $p \in P$ belongs to exactly one block containing it. Particles contained in block B_i are denoted as P_i.

We use a hierarchy of time steps at different blocks, $\Delta t_i = T_i \times t_\varepsilon$, where T_i is powers-of-two integer multipliers and t_ε is the time step granularity. Δt_i takes the largest possible T_i constrained by the two factors as mentioned earlier. Sticking the time step multiplier to power-of-two's may cause a slightly higher update frequency, but it avoids potentially troublesome synchronization between blocks.

5.2. Block Updating Order and Dependency

Central to an asynchronous simulator is the scheduler, which determines in what order blocks are updated. We denote the whole state of one MPM simulation at time t as $\mathcal{M}^t$. Since MPM states are stored on particles, which are further grouped in blocks, $\mathcal{M}^t$ can be entirely described by blocks at time t, denoted as $\mathcal{B}^t = \{B_1^t, B_2^t, ...\}$. For simplicity, in this subsection, we assume the time step of blocks are fixed throughout the simulation, and we can store all history states ($\mathcal{B}^t$ for all possible t's) of a simulation without running out of memory.

In synchronous MPM, the time integration happens globally in the whole simulation state. This can be formulated as

$$\mathcal{M}^{t+\Delta t} = \mathcal{F}^{\Delta t}(\mathcal{M}^t), \tag{11}$$

where $\mathcal{F}^{\Delta t}$ is the function that takes one simulation state and returns the state advanced by Δt. In AsyncMPM, however, we advance the simulation at block granularity. For a block B_i, we have

$$B_i^{t+\Delta t} = \mathcal{F}^{\Delta t}_{\text{block}}(i, \mathcal{B}^t), \tag{12}$$

where $\mathcal{F}^{\Delta t}_{\text{block}}$ is a function that advances the state of a single block. $\mathcal{F}^{\Delta t}$ and $\mathcal{F}^{\Delta t}_{\text{block}}$ are global or regional versions of the algorithm described in section 3. The "locality" of function $\mathcal{F}^{\Delta t}_{\text{block}}$ plays an important role here: the state of block $B_i^{t+\Delta t}$ can be computed using only B_i^t and its neighbors at time t. In 1D, this can be formulated as

$$B_i^{t+\Delta t} = \mathcal{F}^{\Delta t}_{\text{block}}(i, \{B_{i-1}^t, B_i^t, B_{i+1}^t\}). \tag{13}$$

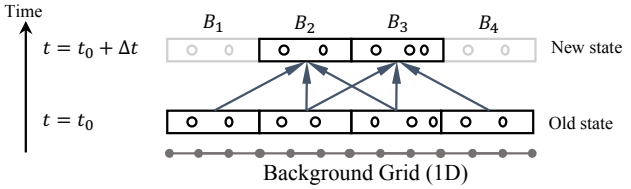


Figure 6: *An 1D MPM simulation example. Simulation states are completely described by blocks. A block in the next time step depends on only those of its neighboring blocks (itself included) in the current time step.*

and similarly in 2D,

$$B_{i,j}^{t+\Delta t} = \mathcal{F}^{\Delta t}_{\text{block}}(i, j, \{B_{i-1,j-1}^t, \; B_{i-1,j}^t, \; B_{i-1,j+1}^t, \; B_{i,j-1}^t, \; B_{i,j}^t, \; B_{i,j+1}^t, \; B_{i+1,j-1}^t, \; B_{i+1,j}^t, \; B_{i+1,j+1}^t \; \}). \tag{14}$$

In 3D there will be $3 \times 3 \times 3 = 27$ blocks needed to advance the center block. For illustration purpose, we show the "dependency graph" of a 1D case in Figure 6.

5.3. Scheduling Algorithm

Conceptually, we still have a global time t monotonously increasing, while whenever it advances, only a small fraction of blocks are updated on average.

We decompose the simulation domain into regions of blocks with the same Δt. Within each region, the algorithm behaves the same as SyncMPM. The only extra treatment is at the boundary of regions where time step changes. In order to correctly couple one region with a larger Δt and another neighboring region with a smaller Δt, at the boundary of the region with larger Δt we introduce some fictitious "buffer" blocks with the smaller Δt (Figure 7, left). These buffer blocks never participate in computation of the region with the larger Δt, while they are necessary for the regions with the smaller Δt, as illustrated in Figure 7 (right). A 1D scheduling example is illustrated in Figure 8.

5.4. Moving particles and changing time step limits

The previous discussion is based on the assumption that particles are fixed in space and block Δt's stay the same. However, in an actual simulation, particles are constantly moving and deforming, leading to varying particle-block relationship and changing block Δt.

When advancing a block with time step Δt, we maintain a particle pool (P_{total} in Alg. 1). Particles in nearby blocks with smaller, equal or greater Δt blocks are added to this pool in order, so that particles advanced with smaller Δt are given higher priority when duplication happens. Such duplication is caused by slightly different trajectories of a single particle when advanced with different time steps, i.e. a particle may end up with different blocks. In this case, we keep only the one with the smallest Δt because it is the most accurate.

Before advancing the blocks, we always recompute the current

Algorithm 1 AsyncMPM. i and d are block indices. Buf_i is the set of particles in the buffer block of B_i, T_i is the power-of-two block time step multiplier, P_i is the set of particles in block B_i. $\mathcal{F}^{\Delta t}_{\text{REGION}}$ takes a set of particles that reside in a region, and returns these particles temporally stepped by Δt using MPM.

function ADVANCE(Δt)
  $T \leftarrow \Delta t / t_\varepsilon$
  $P_{\text{equal}} = \bigcup_{T_i = T} P_i$
  $P_{\text{larger}} = \bigcup_{T_i = T, T_d > T, \text{d nears i}} \text{Buf}_d$
  $P_{\text{smaller}} = \bigcup_{T_i = T, T_d < T, \text{d nears i}} P_d$
  $P_{\text{total}} = P_{\text{smaller}} \cup P_{\text{equal}} \cup P_{\text{larger}}$
  for each block i **do**
    if $T_i = T$ **then**
      $\text{Buf}_i \leftarrow P_i$
  $P'_{total} = \mathcal{F}^{\Delta t}_{\text{REGION}}(P_{total}, \Delta t)$
  for each block i **do**
    if $T_i = T$ **then**
      $P_i = \{p \mid p \in P'_{total}$ **and** p resides in $B_i\}$
    else
      if $T_i > T$ **and** exist d, d nears i and $T_d = T$ **then**
        $B_i = \{p \mid p \in P'_{total}$ **and** p resides in $B_i\}$
function UPDATETIMESTEP()
  for each non-empty block i **do**
    Compute s with stiffness conditions
    Compute c with CFL conditions
    $l \leftarrow \min(s, c)/t_\varepsilon$
    while $l < T_i$ **do**
      $T_i \leftarrow T_i / 2$
    while $l > T_i$ **and** $t \bmod (T_i \times 2t_\varepsilon) = 0$ **do**
      $T_i \leftarrow T_i \times 2$
  $l \leftarrow \min_i T_i$ for all non-empty i
  for each empty block i **do**
    while $l < T_i$ **do**
      $T_i \leftarrow T_i / 2$
    while $l > T_i$ **and** $t \bmod (T_i \times 2t_\varepsilon) = 0$ **do**
      $T_i \leftarrow T_i \times 2$
function RUN()
  for each block non-empty i **do**
    $T_i \leftarrow 1$
    $P_i \leftarrow \{p | p$ resides in block $B_i\}$
  while $t < t_{\text{final}}$ **do**
    UPDATETIMESTEP()
    for $\Delta t = \max_i T_i \rightarrow \min_i T_i$ **do**
      if $t \bmod \Delta t = 0$ **then**
        ADVANCE(Δt)
      $t \leftarrow t + t_\varepsilon \times \min T_i$

particle-block relationship and update the time step of each block. If a block needs a smaller time step, we repeatedly halve the time step until the limit is satisfied. If a block needs a larger time step, we double its time step only when the current time is a multiple of the doubled time step. The constraint here is to make sure every block with a time step Δt will only be updated when the global t is a multiple of Δt, saving the troubles for block state synchronization.

After updating a region of blocks, its particles are reorganized

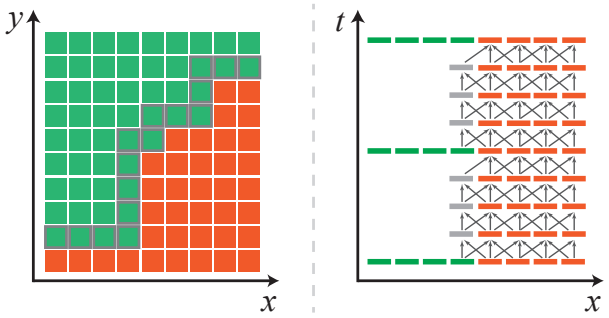


Figure 7: *"Buffer blocks" in gray. The green blocks have 4× larger Δt than orange blocks.* ***Left:*** *In this 2D example, buffer blocks (gray contour) are generated near the boundary of Δt change, on the side with larger Δt.* ***Right:*** *In this 1D example, buffer blocks are updated together with orange blocks Note that they depend on only other buffer blocks and orange blocks, and they are never used to update green blocks.*

to their residing blocks. At the interface of changing Δt, a particle may move into a block with a different time step. On one hand, our scheduler naturally handles the case when a particle moves into a block with a smaller Δt. On the other hand, for each particle, it may be advanced together with a nearby block with a relatively larger Δt. When it happens to move across the block boundary into the larger Δt block, this single time integration may break its time step bound. It may seem dangerous to update a stiff particle with a relatively large Δt, but the updated stiff particle will be recorded only if it happens to move across the block boundary. Emperically, we have not encountered any stability problem. The whole AsyncMPM cycle is detailed in Algorithm 1.

6. Implementation

In this section, we describe our efficient implementation of SyncMPM and AsyncMPM in detail. We developed our system based on *Taichi* [Hu18]. The source code will be made publicly available.

6.1. Particle and Grid Storage

The majority of MPM data are particles. In contrast to SyncMPM where particles are simply stored in a monolithic array, in AsyncMPM we store them in small arrays for all blocks to maintain the particle-block relationship. In the previous discussions about asynchronous MPM, we assumed we could store the full history of simulation, which is practically impossible. It turns out that we only need to store two copies of each block: the one with the most recent computed state, and the other one the most recent buffer. Intuitively in Figure 8, this means we only need to store the newest blue and gray blocks for each column. Since MPM is hardly bounded by memory space, such 2× increase in space consumption is acceptable.

The background grid is stored using SPGrid [SABS14]. SPGrid is a sparsely paged grid data structure using modern virtual memory system and Morton coding to speed up addressing and improve

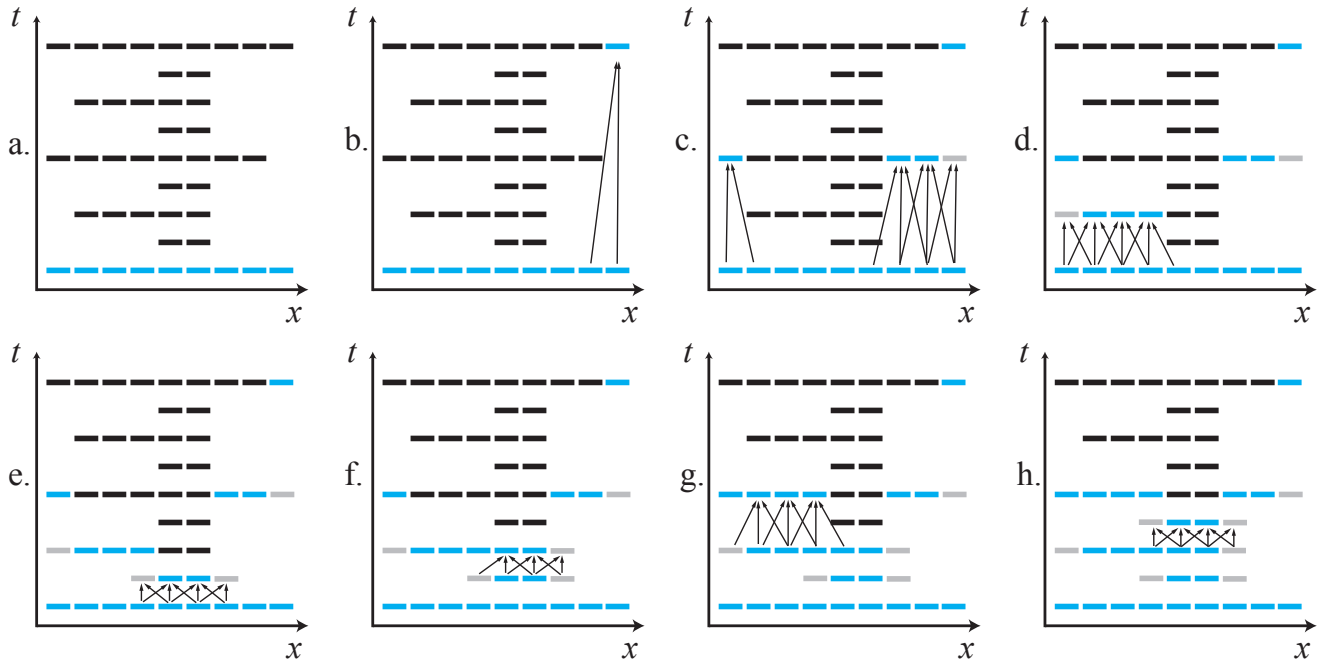


Figure 8: *An 1D scheduling example (a-h), with the smallest block $\Delta t = 1/8$. Blocks as represented as thick segments. Black blocks are uncomputed, blue blocks computed and gray blocks are buffer blocks. Step a-e demonstrate the time integration when $t = 0$. When there are multiple future blocks ready to compute, we compute them in a decreasing-time-step order. Step f has $t = 1/8$ and step g and h have $t = 1/4$.*

locality. In our implementation, we use blocks of size $4 \times 4 \times 8$, each occupying exactly a 4KB virtual memory page when eight 32-bit float-point numbers are stored at each node. During 3D MPM particle-grid transfer with a quadratic B-Spline kernel, each particle will touch 27 neighboring nodes. Avoiding cache misses can significantly increase the efficiency of such operation. Fortunately, all particles inside a block will touch at most $6 \times 6 \times 10$ nodes, which we use as a local arena grid to temporally store. Such storage scheme ensures the local grid fits into L1 data cache. The local arena is shown in Figure 9 (left).

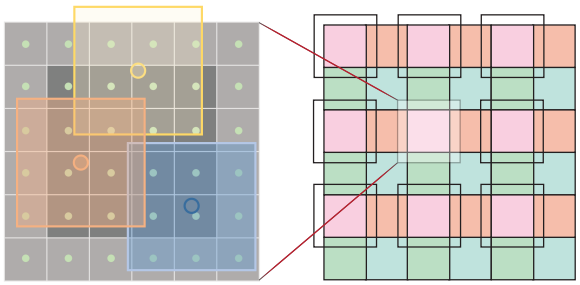


Figure 9: ***Left:*** *Local arena. In this 2D example, we use $4\Delta x \times 4\Delta x$ as the block size. Each particle will interact with grid nodes within a $3\Delta x \times 3\Delta x$ region. This means for all the particles inside the deep gray region, only the green nodes will be touched. This is result in a $6\Delta x \times 6\Delta x$ local arena for transfer operations.* ***Right:*** *parallel particle-to-grid transfer. When using parallelized particle rasterization on a $24\Delta x \times 24\Delta x$ grid (6×6 blocks as depicted), we color blocks into 4 colors so that the each expanded arena on blocks with the same color will not overlap. Such strategy avoids data race when using multithreading.*

6.2. Lock-free Multithreading

The grid-to-particle transfer is embarrassingly parallelizable since there is no data race. After particle-to-grid transfer, however, we need to accumulate mass and momentum in the local arena to the global SPGrid. There is a data race here since arenas of neighboring blocks overlap. Expensive per-cell locking would be necessary to ensure correct results when multiple threads are processing neighboring blocks.

To make this process lock-free, we partite the blocks into 2^D sets where D is the simulation dimensionality, so that in each set any two blocks do not share overlapping global node. Specifically in 3D, for a SPGrid block with the smallest coordinate (i, j, k), its group index is defined to be $g = 4I + 2J + K$, where $I = \lfloor i/4 \rfloor \bmod 2$, $J = \lfloor j/4 \rfloor \bmod 2$, $K = \lfloor k/8 \rfloor \bmod 2$ are the parity bits of block coordinates. For each pass, we only transfer blocks in a single set with multiple threads, and no two block arenas write to the same node. 2^D passes cover all the blocks. Figure 9 (right) illustrate a 2D example.

7. Results

In our test cases, AsyncMPM delivers both higher efficiency and less numerical damping.

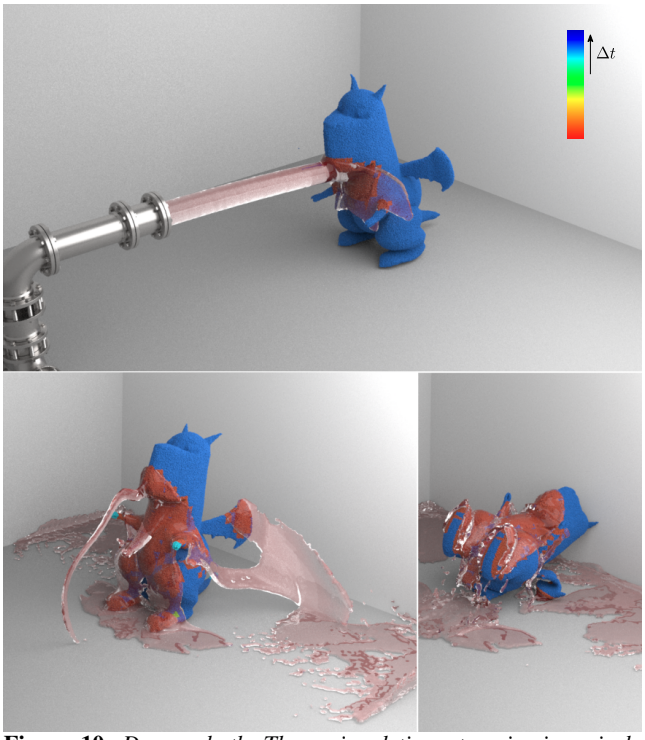


Figure 10: *Dragon bath. The regional time step size is majorly dominated by material sound speed.*

7.1. Efficiency

$1.4-9.8\times$ speed-up over SyncMPM is observed, as listed in Table 1. For stability, the time step of SyncMPM is set to the largest allowed value we found during the entire AsyncMPM simulation process based on our time step restriction criteria.

Stiffness can sometimes dominate the time steps. Figure 4 demonstrates such case with elastic balls of different stiffness. Harder balls are updated more frequently for stability. The snow slope example (Figure 1) showcases material stiffness changes due to hardening. The large stiffness after compression makes AsyncMPM especially advantageous. It is worth noting that an implicit solver is more suitable for this simulation, though our high-performance explicit solver is easier to implement and runs as fast as the implicit one in [SSC*13].

The actual time step limit in MPM is more frequently determined by the advection CFL condition. For instance, the fast moving balls (Figure 5) and water jet (Figure 10) trigger very restrictive time steps due to their high velocity. In traditional SyncMPM such regional high velocity unnecessarily slows down the whole simulation.

Lastly, MPM provides automatic coupling between various materials. AsyncMPM is especially useful in this case because different time steps can be used for different materials, as shown in Figure 2 and Figure 3.

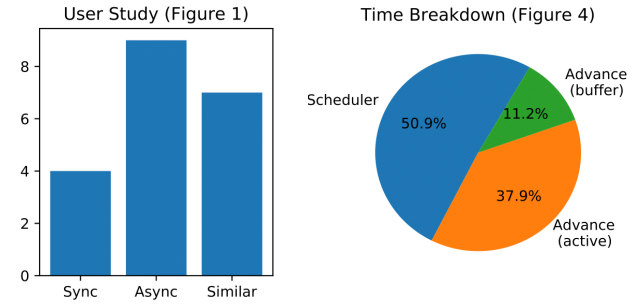


Figure 11: ***Left:*** *User study results (SyncMPM v.s. AsyncMPM) on the snow slope example (Figure 1).* ***Right:*** *Time breakdown of one time step on the elastic ball example (Figure 4).*

7.2. Quality

The efficiency of AsyncMPM comes with no sacrifice of quality. In fact, it conveys visually identical or sometimes even better results compared with SyncMPM.

For example, we found AsyncMPM leads to better preservation of motion. Since larger time steps are allowed in soft and slow-moving regions, there will be fewer particle-grid transfers, which are more or fewer dissipative. Figure 12 display the energy evolution of AsyncMPM and SyncMPM in the stork couple example, with $\Delta x = 0.0125$ and $\Delta x = 0.004$ respectively. It can be seen that at low resolution, the kinetic and potential energy fluctuations last longer in AsyncMPM due to less damping. At high resolution, though the damping difference is less significant, the total energy is preserved much better.

Qualitatively, we have conducted a user study on the "snow slope" example (Figure 1). 20 volunteers were shown video A (SyncMPM) and B (AsyncMPM) and asked to answer which one looks better (or "they are similar"). The results are shown in Figure 11 (left). Most of these volunteers think AsyncMPM is better, because "the slightly more powdery look in video B makes it more vivid than video A" or "the shape of snow in video B is more irregular and realistic". We hypothesis such difference is due to less dissipative transfers in AsyncMPM, as discussed in section 7.2. Though such user study is not sufficient to prove that AysncMPM is visually superior to SyncMPM, it however does give us more confidence on the conclusion that AsyncMPM improves performance without scarificing quality.

8. Discussion and Future Work

Even though our method provides a significant speedup on many practical examples, it is not always the preferred choice especially for cases where stiff materials occupy the main portion of a scene. In those cases, the overhead from the scheduler may exceed the performance saving from having larger time steps at some regions. A breakdown of run time is shown in Figure 11 (right). In this measurement, our asynchronous scheduler reduces the total number of block updates by $7.7\times$ compared with SyncMPM. Compared with our highly-optimized "Advance" part written in CPU-intrinsics, the scheduler is less optimized. Though the scheduler takes half of run

Example	Δx	$\min \Delta t$	$\max \Delta t$	Particle #	Density	Young's Modulus	Bulk Modulus	Yield Stress	Friction Angle	sync sec/frame	async sec/frame
(Fig. 3) Storks	4.0×10^{-3}	8.0×10^{-6}	6.4×10^{-5}	2.4M	10	4e3 / 2e4 / 4e5	-	-	-	80.0	32.9(2.4×)
(Fig. 4) Elastic Balls	4.0×10^{-3}	6.4×10^{-5}	1.024×10^{-3}	3.2M	400	1e3 / 4e4 / 8e3 / 2e5	-	-	-	120.3	24.5(4.9×)
(Fig. 1) Snow Slop	2.5×10^{-3}	8.0×10^{-6}	5.12×10^{-4}	2.5M	100 / 400	2e3 / 1.4e5 / 6e5	-	-	-	859.6	87.8(9.8×)
(Fig. 10) Dragon Bath	2.5×10^{-3}	1.6×10^{-5}	2.56×10^{-4}	1.4M	400	5e3	1e4	-	-	240.4	138.9(1.7×)
(Fig. 5) Dragon Massage	2.5×10^{-3}	3.2×10^{-5}	2.56×10^{-4}	1.3M	400	2e5	-	-	10	101.5	39.7(2.6×)
(Fig. 2) Dragon Combo	3.3×10^{-3}	1.6×10^{-5}	2.56×10^{-4}	1.5M	400	5e3 / 5e5	1e4	1	10	310.2	216.3(1.4×)

Table 1: *Settings and time per frame of our test scenes. All experiments were done on a six-core Intel i7-8700K (3.7GHz) CPU and 64 GB main memory.*

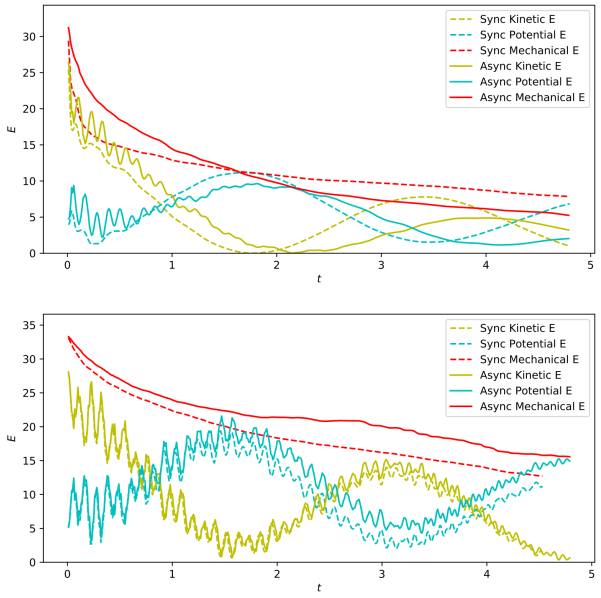


Figure 12: *Energy evolution of the stork example (Figure 3). Note the better kinetic and potential energy vibration preservation at low resolution (top, $\Delta x = 0.0125$) and the better mechanical energy preservation at high resolution (bottom, $\Delta x = 0.004$).*

time, it still leads to 4.9× speed-up. We believe more low-level optimization can make the performance benefit from our scheduler even larger.

We look forward to exploring mixed implicit-explicit integration schemes (IMEX) [FSH11] with regional time stepping to handle these cases better. We would also like to combine our temporal adaptivity with spatial adaptivity [GTJS17] for further improved efficiency. Additional future work includes extending our method to the GPU based on inspirations from [WTYH18].

9. Acknowledgement

We are grateful to the anonymous reviewers and editors for their valuable suggestions and comments. We thank Hannah Bollar from University of Pennsylvania for narrating the video. The work is partially supported by Jiang's StartUp Grant from the University of Pennsylvania, NSF IIS-1755544, the National Key Technology R&D Program of China (2017YFB1002701), the Natural Science Foundation of China (Project Number 61521002), a gift from Awowd Inc., a gift from NVIDIA Corporation, and a gift from SideFX.

References

[BFA02] BRIDSON R., FEDKIW R., ANDERSON J.: Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans Graph 21*, 3 (2002), 594–603. 2

[BH92] BINA C. R., HELFFRICH G. R.: Calculation of elastic properties from thermodynamic equation of state principles. *Rev Earth Planet Sci 20*, 1 (1992), 527–552. 3

[BK04] BARDENHAGEN S., KOBER E.: The generalized interpolation material point method. *Comp Model Eng Sci 5*, 6 (2004), 477–496. 2

[BMF03] BRIDSON R., MARINO S., FEDKIW R.: Simulation of clothing with folds and wrinkles. In *Symp Comp Anim* (2003), pp. 28–36. 2

[BT07] BECKER M., TESCHNER M.: Weakly compressible sph for free surface flows. In *Symp Comp Anim* (2007), pp. 209–217. 3

[BW98] BARAFF D., WITKIN A.: Large steps in cloth simulation. In *Proc ACM SIGGRAPH* (1998), SIGGRAPH '98, pp. 43–54. 2

[BWHT07] BARGTEIL A., WOJTAN C., HODGINS J., TURK G.: A finite element method for animating large viscoplastic flow. *ACM Trans Graph 26*, 3 (2007). 2

[CDH*09] CHIANG W.-F., DELISI M., HUMMEL T., PRETE T., TEW K., HALL M., WALLSTEDT P., GUILKEY J.: Gpu acceleration of the generalized interpolation material point method. In *Symp App Accel High Perform Comp* (2009). 2

[CFL28] COURANT R., FRIEDRICHS K., LEWY H.: Über die partiellen differenzengleichungen der mathematischen physik. *Mathematische annalen 100*, 1 (1928), 32–74. 1

[DBD16] DAVIET G., BERTAILS-DESCOUBES F.: A semi-implicit material point method for the continuum simulation of granular materials. *ACM Trans Graph 35*, 4 (2016), 102:1–102:13. 1

[DC99] DESBRUN M., CANI M.-P.: *Space-time adaptive simulation of highly deformable substances*. PhD thesis, INRIA, 1999. 2

[DDCB00] DEBUNNE G., DESBRUN M., CANI M.-P., BARR A.: Adaptive simulation of soft bodies in real-time. In *Comp Anim* (2000), pp. 15–20. 2

[FF01] FOSTER N., FEDKIW R.: Practical animation of liquids. In *Symp Comp Anim* (2001), pp. 23–30. 2

[FSH11] FIERZ B., SPILLMANN J., HARDERS M.: Element-wise mixed implicit-explicit integration for stable dynamic simulation of deformable objects. In *Symp Comp Anim* (2011), pp. 257–266. 9

[GB14] GOSWAMI P., BATTY C.: Regional time stepping for sph. In *Eurographics 2014* (2014), pp. 45–48. 1, 2, 5

[GSS*15] GAST T., SCHROEDER C., STOMAKHIN A., JIANG C., TERAN J.: Optimization integrator for large time steps. *IEEE Trans Vis Comp Graph 21*, 10 (2015), 1103–1115. 2

[GTH*18] GAO M., TAMPUBOLON A. P., HAN X., GUO Q., KOT G., SIFAKIS E., JIANG C.: Animating fluid sediment mixture in particle-laden flows. *ACM Transactions on Graphics (TOG), SIGGRAPH 2018 37*, 4 (2018). 2

[GTJS17] GAO M., TAMPUBOLON A. P., JIANG C., SIFAKIS E.: An adaptive generalized interpolation material point method for simulating elastoplastic materials. *ACM Trans Graph 36*, 6 (2017). 1, 2, 3, 9

[HFG*18] HU Y., FANG Y., GE Z., QU Z., ZHU Y., TAMPUBOLON A. P., JIANG C.: A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Transactions on Graphics (TOG), SIGGRAPH 2018 37*, 4 (2018). 2

[Hu18] HU Y.: Taichi: An open-source computer graphics library. *arXiv preprint arXiv:1804.09293* (2018). 6

[JGT17] JIANG C., GAST T., TERAN J.: Anisotropic elastoplasticity for cloth, knit and hair frictional contact. *ACM Trans Graph 36*, 4 (2017). 1

[JSS*15] JIANG C., SCHROEDER C., SELLE A., TERAN J., STOMAKHIN A.: The affine particle-in-cell method. *ACM Trans Graph 34*, 4 (2015), 51:1–51:10. 2

[JST*16] JIANG C., SCHROEDER C., TERAN J., STOMAKHIN A., SELLE A.: The material point method for simulating continuum materials. In *SIGGRAPH 2016 Course* (2016), pp. 24:1–24:52. 1

[KFCS99] KINSLER L. E., FREY A. R., COPPENS A. B., SANDERS J. V.: *Fundamentals of acoustics*. 1999. 3

[KGP*16] KLÁR G., GAST T., PRADHANA A., FU C., SCHROEDER C., JIANG C., TERAN J.: Drucker-prager elastoplasticity for sand animation. *ACM Trans Graph 35*, 4 (2016), 103:1–103:12. 1, 2, 3

[Mon92] MONAGHAN J. J.: Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics 30*, 1 (1992), 543–574. 2

[MST*11] MCADAMS A., SELLE A., TAMSTORF R., TERAN J., SIFAKIS E.: *Computing the Singular Value Decomposition of 3× 3 matrices with minimal branching and elementary floating point operations*. Tech. rep., University of Wisconsin-Madison, 2011. 3

[MWN*17] MANTEAUX P.-L., WOJTAN C., NARAIN R., REDON S., FAURE F., CANI M.-P.: Adaptive physically based models in computer graphics. In *Comp Graph Forum* (2017), vol. 36, pp. 312–337. 1, 2

[RHEW17] REINHARDT S., HUBER M., EBERHARDT B., WEISKOPF D.: Fully asynchronous sph simulation. In *Symp Comp Anim* (2017), p. 2. 2, 4

[RS14] RUGGIRELLO K. P., SCHUMACHER S. C.: A comparison of parallelization strategies for the material point method. In *11th WCCM* (2014), pp. 20–25. 2

[SABS14] SETALURI R., AANJANEYA M., BAUER S., SIFAKIS E.: Spgrid: A sparse paged grid structure applied to adaptive smoke simulation. *ACM Trans Graph 33*, 6 (2014), 205. 1, 6

[SDTS03] SERNA A., DOMÍNGUEZ-TENREIRO R., SÁIZ A.: Conservation laws in smooth particle hydrodynamics: The deva code. *The Astrophysical Journal 597*, 2 (2003), 878. 5

[SKZF11] SCHROEDER C., KWATRA N., ZHENG W., FEDKIW R.: Asynchronous evolution for fully-implicit and semi-implicit time integration. In *Comp Graph Forum* (2011), vol. 30, pp. 1983–1992. 2

[SSC*13] STOMAKHIN A., SCHROEDER C., CHAI L., TERAN J., SELLE A.: A material point method for snow simulation. *ACM Trans Graph 32*, 4 (2013), 102:1–102:10. 1, 8

[SZS95] SULSKY D., ZHOU S., SCHREYER H.: Application of a particle-in-cell method to solid mechanics. *Comp Phys Comm 87*, 1 (1995), 236–252. 1

[TGK*17] TAMPUBOLON A. P., GAST T., KLÁR G., FU C., TERAN J., JIANG C., MUSETH K.: Multi-species simulation of porous sand and water mixtures. *ACM Trans Graph 36*, 4 (2017). 3

[TPS08] THOMASZEWSKI B., PABST S., STRASSER W.: Asynchronous cloth simulation. In *Comp Graph Int* (2008), vol. 2. 2, 4

[WTYH18] WU K., TRUONG N., YUKSEL C., HOETZLEIN R.: Fast fluid simulations with sparse volumes on the gpu. In *Eurographics* (2018). 9

[ZCL16] ZHANG X., CHEN Z., LIU Y.: *The material point method: a continuum-based particle method for extreme loading cases*. 2016. 3

[ZLB16] ZHAO D., LI Y., BARBIC J.: Asynchronous implicit backward euler integration. In *Symp Comp Anim* (2016), pp. 1–9. 2, 4

[ZY10] ZHU B., YANG X.: Animating sand as a surface flow. In *Eurographics (Short Papers)* (2010), pp. 9–12. 1

[ZZL10] ZHANG Y., ZHANG X., LIU Y.: An alternated grid updating parallel algorithm for material point method using openmp. *Computer Modeling in Engineering & Sciences(CMES) 69*, 2 (2010), 143–165. 2