

GaussianFluent: Gaussian Simulation for Dynamic Scenes with Mixed Materials

Supplementary Material

A. Material Point Method

Overview We use an explicit MPM. Particles (also the 3D Gaussian splats) carry

$$m_p, V_p^0, \mathbf{X}_p, \mathbf{x}_p, \mathbf{v}_p, \mathbf{F}_p, \mathbf{A}_p, \mathbf{a}_p, \sigma_p, \mathbf{C}_p. \quad (\text{A1})$$

In summary, given the 3D GS of a static scene $\{X_p, A_p, \sigma_p, C_p\}$, we use simulation to dynamize the scene by evolving these Gaussians to produce dynamic Gaussians $\{x_p(t), a_p(t), \sigma_p, C_p\}$. Here, \mathbf{X}_p is the initial position, while \mathbf{x}_p is the current position that evolves over time with velocity \mathbf{v}_p . Furthermore, \mathbf{A}_p is the static covariance of the initial Gaussian; the dynamic covariance \mathbf{a}_p is derived at each step; \mathbf{F}_p is the deformation gradient used to calculate \mathbf{a}_p ; and the opacity σ_p and SH coefficient magnitudes \mathbf{C}_p are considered time-invariant.

A.1. The Material Point Method (MPM) Algorithm Steps

The Material Point Method (MPM) algorithm iteratively transfers data between particles and a background grid. A single time step can be broken down into the following three main stages.

A.1.1. Particle-to-Grid Transfer (P2G)

In the first stage, information is transferred from the Lagrangian particles to the nodes of the Eulerian grid. This process, known as rasterization, effectively creates a grid-based snapshot of the continuum's state. For each particle p , its mass m_p and momentum $\mathbf{p}_p = m_p \mathbf{v}_p$ are interpolated and added to the surrounding grid nodes i . This is done using interpolation functions N_{ip} (also known as shape functions), which depend on the particle's position relative to the grid.

The nodal mass m_i and nodal momentum \mathbf{p}_i are computed as follows:

$$m_i = \sum_p m_p N_{ip} \quad (\text{A2})$$

$$\mathbf{p}_i = \sum_p m_p \mathbf{v}_p N_{ip}. \quad (\text{A3})$$

From the nodal momentum and mass, the initial nodal velocity is found: $\mathbf{v}_i = \mathbf{p}_i / m_i$, provided $m_i > 0$.

A.1.2. Grid Update

This stage contains the core physics computations, which are performed entirely on the grid. First, forces acting on each grid node are calculated. These forces are typically composed of two parts:

- **Internal forces** $\mathbf{f}_i^{\text{internal}}$, which arise from the material's stress. These are computed by transferring particle stress information (derived from the deformation gradient \mathbf{F}_p) back to the grid.
- **External forces** $\mathbf{f}_i^{\text{external}}$, such as gravity or user-defined interactions.

The total force on a node is $\mathbf{f}_i = \mathbf{f}_i^{\text{internal}} + \mathbf{f}_i^{\text{external}}$.

With the total force, the grid node velocities are updated over the time step Δt using an explicit time integration scheme (e.g., Forward Euler):

$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \Delta t \frac{\mathbf{f}_i}{m_i}. \quad (\text{A4})$$

Boundary conditions, such as collisions with obstacles, are also enforced on the grid during this stage by modifying the nodal velocities.

A.1.3. Grid-to-Particle Transfer (G2P)

Finally, the updated kinematic information is transferred from the grid back to the particles. This stage, often called the "gather" step, updates the Lagrangian particles' state using the newly computed fields on the Eulerian grid, preparing them for the next time step. This process involves updating each particle's velocity, its deformation gradient, and finally its position.

First, the particle's velocity \mathbf{v}_p is updated by interpolating the new velocities \mathbf{v}_i^{n+1} from the surrounding grid nodes. This is essentially a weighted average, using the same interpolation functions N_{ip} as the P2G step:

$$\mathbf{v}_p^{n+1} = \sum_i \mathbf{v}_i^{n+1} N_{ip}. \quad (\text{A5})$$

This update can be a pure Particle-In-Cell (PIC) update, or it can be combined with the particle's previous velocity in a FLIP (Fluid-Implicit-Particle) scheme to reduce numerical dissipation.

Simultaneously, the particle's deformation gradient \mathbf{F}_p , which tracks the local rotation and strain of the material, must also be updated. This is done by first computing the velocity gradient $\nabla \mathbf{v}$ at the particle's position, which is also interpolated from the grid node velocities:

$$\nabla \mathbf{v}_p = \sum_i \mathbf{v}_i^{n+1} \nabla N_{ip}^T. \quad (\text{A6})$$

This gradient is then used to advance the deformation gradient forward in time:

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \nabla \mathbf{v}_p) \mathbf{F}_p^n, \quad (\text{A7})$$

where \mathbf{I} is the identity matrix. This update is crucial for correctly computing material stress in the next time step.

Lastly, with the new velocity \mathbf{v}_p^{n+1} computed, the particle's position \mathbf{x}_p is updated as:

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1}. \quad (\text{A8})$$

Once all particles have been updated, the information on the background grid is no longer needed and is typically reset or discarded. The simulation is now ready to begin the next time step with the P2G phase.

A.2. Evolution of 3D Gaussian Properties via Continuum Mechanics

This approach outlines a method for animating 3D GS by treating them as discrete particles within a physics-based system governed by continuum mechanics. The primary goal is to evolve a static scene, defined by initial properties, into a dynamic state for rendering.

The evolution of the key Gaussian properties for each time step is as follows:

- **Position Evolution (Mean):** The Gaussian's center, or mean, is its world-space position \mathbf{x}_p . This is updated using the particle's velocity \mathbf{v}_p , which is determined by the physical simulation, via explicit time integration:

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p. \quad (\text{A9})$$

- **Shape Evolution (Covariance):** The dynamic world-space covariance \mathbf{a}_p , which defines the Gaussian's shape and size, is computed directly from the deformation gradient \mathbf{F}_p . The deformation gradient describes the local deformation of the material around the particle. It maps the initial, undeformed shape (defined by the material-space covariance \mathbf{A}_p) to its current, deformed configuration:

$$\mathbf{a}_p(t) = \mathbf{F}_p(t) \mathbf{A}_p \mathbf{F}_p(t)^T. \quad (\text{A10})$$

- **Orientation Evolution (for Rendering):** To correctly render anisotropic appearances (e.g., using Spherical Harmonics), the particle's orientation must be tracked. The rotation component \mathbf{R}_p is extracted from the deformation gradient, typically via polar decomposition ($\mathbf{F}_p = \mathbf{R}_p \mathbf{S}_p$). This rotation is then applied to the appearance model during rendering.
- **Time-Invariant Properties:** Visual attributes such as opacity σ_p and material-space appearance coefficients (e.g., Spherical Harmonics, \mathbf{C}_p) are considered intrinsic material properties. They are typically held constant throughout the simulation.

B. Fracture mechanism with Continuum Damage Material Point Method

B.1. Introduction of CD-MPM

The yield surface serves as a dividing boundary in stress space: inside it, the material response is elastic; at the boundary plastic yielding begins; any trial state predicted beyond this boundary is reconciled by returning it to a suitable point on the boundary in accordance with ideal plasticity. As mentioned above, the yield surface of CD-MPM is defined as:

$$y(p, q) = (1 + 2\beta) q^2 + M^2(p + \beta p_0)(p - p_0) = 0. \quad (\text{A11})$$

If (p, q) lies in the elastic domain where $y \leq 0$, no plastic correction is applied.

$$(p_c, q_c) = \left(\frac{1-\beta}{2} p_0, 0 \right) \quad (\text{A12})$$

$$y_{tr} = y(p_{tr}, q_{tr}) \quad (\text{A13})$$

$$J_E(p) = \sqrt{-\frac{2p}{\kappa} + 1} \quad (\text{A14})$$

Here p_c, q_c identify the center of the yield ellipsoid ($y = 0$); p_{tr}, q_{tr} is the uncorrected trial stress state produced at simulation step n ; J_E is the determinant of the elastic deformation gradient (elastic volume ratio); κ is the Bulk Modules; and p_{n+1}, q_{n+1} is the state after applying the return mapping R :

$$R(p_{n+1}, q_{n+1}) = \begin{cases} (p_{tr}, q_{tr}), & y_{tr} \leq 0 \\ & \text{(Elastic)} \\ (p_0, 0), & y_{tr} > 0 \wedge p_{tr} > p_0 \\ & \text{(Case 1: upper tip projection)} \\ (-\beta p_0, 0), & y_{tr} > 0 \wedge p_{tr} < -\beta p_0 \\ & \text{(Case 2: lower tip projection)} \\ (p_x, q_x), & y_{tr} > 0 \wedge -\beta p_0 \leq p_{tr} \leq p_0 \\ & \text{(Case 3: center-trial line intersection)} \end{cases} \quad (\text{A15})$$

Here $y_{tr} = y(p_{tr}, q_{tr})$. If $y_{tr} \leq 0$, the trial point lies in the elastic domain and is accepted unchanged: $(p_{n+1}, q_{n+1}) = (p_{tr}, q_{tr})$. If $y_{tr} > 0$ and $p_{tr} > p_0$, the trial point lies beyond the positive p -axis tip and is projected to the upper tip $(p_0, 0)$. If $y_{tr} > 0$ and $p_{tr} < -\beta p_0$, it lies beyond the negative tip and is projected to $(-\beta p_0, 0)$. Otherwise ($y_{tr} > 0$ with $-\beta p_0 \leq p_{tr} \leq p_0$), we join the center (p_c, q_c) and the trial point (p_{tr}, q_{tr}) ; the intersection of this line segment with the yield ellipsoid $y(p, q) = 0$ defines (p_x, q_x) , and we set $(p_{n+1}, q_{n+1}) = (p_x, q_x)$. Besides p, q , we also update α and J_E as below:

$$\alpha_{n+1} = \alpha_n + \begin{cases} 0, & y_{tr} \leq 0 \\ \log(J_{E,tr}/J_{E,n+1}), & y_{tr} > 0 \end{cases}, \quad (\text{A16})$$

with

$$J_{E,n+1} = \begin{cases} J_E(p_0), & \text{Case 1} \\ J_E(-\beta p_0), & \text{Case 2} \\ J_E(p_x), & \text{Case 3} \end{cases} \quad (\text{A17})$$

B.2. Adapted Continuous Return Mapping

However, this piecewise return mapping is discontinuous at the right tip $p = p_0$. Consider trial states with $y_{tr} > 0$ and very large shear measure $q_{tr} \rightarrow \infty$. Take two sequences with $p_{tr} = p_0 - \varepsilon$ and $p_{tr} = p_0 + \varepsilon$ ($\varepsilon > 0$). For $p_{tr} = p_0 - \varepsilon$, the algorithm falls into the “center-trial line intersection” branch; as $q_{tr} \rightarrow \infty$ the direction from the center $(p_c, 0)$, with $p_c = \frac{1-\beta}{2} p_0$, to the trial point becomes vertical, so the mapped point tends to the upper apex of the yield ellipsoid $y(p, q) = 0$, namely,

$$\lim_{\varepsilon \rightarrow 0^+} \lim_{q_{tr} \rightarrow \infty} R(p_0 - \varepsilon, q_{tr}) = \left(\frac{1-\beta}{2} p_0, \frac{M(\beta+1)}{2\sqrt{1+2\beta}} p_0 \right), \quad (\text{A18})$$

$$\lim_{\varepsilon \rightarrow 0^+} \lim_{q_{tr} \rightarrow \infty} R(p_0 + \varepsilon, q_{tr}) = (p_0, 0),$$

showing a directional jump: one limit preserves (essentially) shear while the other preserves only the volumetric extension. And even some small q such as $q = p_0$ will also occur jumps like this.

methods are intended for static scenes and rely on multiple images captured under known lighting conditions to learn GS normals and other attributes. Rather than adopting the Physically Based Rendering (PBR) lighting model in Relightable 3DGS, which requires learning additional material attributes, *e.g.*, Fresnel parameters, for each GS, we employ the empirical Blinn-Phong reflection model, which only requires the normals for GS.

However, it is nontrivial to obtain GS normals using non-learning methods. As noted in Relightable 3DGS, numerical normal-estimation methods such as PCA are ill-suited to GS for two primary reasons: (i) GS particles are spatially sparse, and (ii) Gaussian centers, especially those with large kernels, are not tightly aligned with the visual surface. To overcome these issues, the regularization loss 1 we introduce in Sec. 3.1 promotes kernel densification and surface alignment, thereby enabling effective normal computation for each Gaussian splat using PCA.

Blinn-Phong Reflection Model Once the normal \mathbf{n} for each GS is computed, we apply the Blinn-Phong reflection model to determine its final color. For each Gaussian i with center \mathbf{p}_i and normal \mathbf{n}_i , we apply the Blinn-Phong reflection model using view direction \mathbf{v} (from \mathbf{p}_i to the camera) and, for each light m , light direction \mathbf{l}_m , distance r_m , and half vector $\mathbf{h}_m = (\mathbf{l}_m + \mathbf{v}) / \|\mathbf{l}_m + \mathbf{v}\|_2$. The diffuse and specular terms are $D_m = \max(\mathbf{n}_i \cdot \mathbf{l}_m, 0)$ and $S_m = [\max(\mathbf{n}_i \cdot \mathbf{h}_m, 0)]^p$, with shininess exponent p . Let \mathbf{c}_0 be the base color, \mathbf{I}_a the ambient light color, $\mathbf{I}_{L,m}$ the color of light m , and $T_{i,m}$ a per-light visibility term. Then

$$\mathbf{I}_i = \mathbf{c}_0 \odot \mathbf{I}_a + \sum_m T_{i,m} (\mathbf{c}_0 \odot \mathbf{I}_{L,m}) \frac{1}{r_m^2} (D_m + S_m), \quad (\text{A20})$$

where \odot denotes element-wise multiplication.

This lighting framework allows us to effectively simulate complex scenes with multiple objects and dynamic light sources, as shown in Figure 1 and Figure A7. For example, in the latter figure, we present a scene of multiple fruits with dynamic lighting on a table. Such dynamic illumination and shadowing are crucial for achieving visually consistent and plausible renderings during simulation, where the evolution of shadows is not considered in PhysGaussian [49].

D. Use of Large Language Models (LLMs)

We used a large language model solely as a writing aid to improve the clarity, grammar, and overall readability of the manuscript. Its role was limited to polishing the language and refining sentence structure, without contributing to research ideation, experimental design, or data analysis. All technical ideas, methods, results, and conclusions are entirely the work of the authors, and we take full responsibility for the final content.

E. Extra Experiment

Additional objects We show additional results for the cloth and pillow, as shown in Figure A2. Here, we conduct interior infilling for the pillow, while for the cloth, we directly use the GS reconstruction results.

Comparison with FruitNinja FruitNinja is *not designed to handle omnidirectional exposure during simulation*, making it unsuitable for complex volumetric infilling. This limitation stems



Figure A2. Additional qualitative results for the cloth and pillow.

from their fine-tuning strategy, which uses only horizontal and vertical cross-sections, failing to capture globally consistent textures. Additionally, FruitNinja requires per-object real images for fine-tuning. Since such data is not officially available for arbitrary objects, we use its official watermelon GS release for the experiments below. Specifically, we render three slices along the x-axis (front, middle, and back) and conduct fracture simulations. FruitNinja produces unrealistic interior textures, especially in the peripheral slices, and its watermelon seeds are virtually absent during the simulation. In contrast, our optimization scheme enforces consistency across the entire volume, yielding *simulation-ready* interior texture infilling.

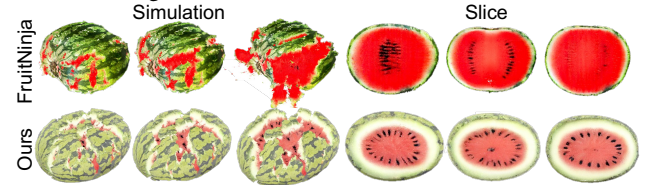


Figure A3. Comparison with FruitNinja regarding simulation and slice rendering.

Comparison with CD-MPM-TOG CD-MPM-TOG lacks physical stability during simulation, as its discontinuous return mapping leads to visual artifacts, as shown in Figure A4.

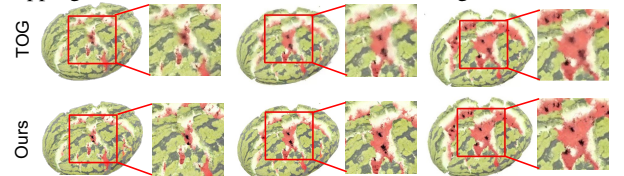


Figure A4. Watermelon simulation using different methods.

Ablation and Robustness Experiments As shown below, the results on a quarter watermelon demonstrate that removing texture generation stage 2 affects axis consistency, while omitting stage 1 causes the watermelon seeds to disappear. Furthermore, the robustness of our method is verified across multiple samples.

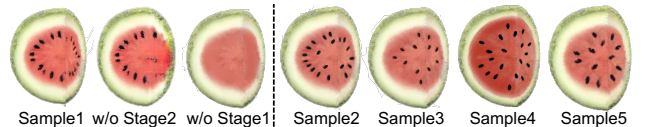


Figure A5. Ablation study and robustness evaluation on the watermelon model.



Figure A6. More examples of object simulation.



Figure A7. More examples of object simulation and illumination.