

# Scene-Level Heterogeneous Physics Simulation with 3D Gaussian Splats

Xiaoyang Liu<sup>1</sup>   Shangzhe Wu<sup>2,†</sup>   Kai Han<sup>1,†</sup>  
<sup>1</sup>The University of Hong Kong   <sup>2</sup>University of Cambridge  
xiaoyangliu@connect.hku.hk   sw2181@cam.ac.uk   kaihanx@hku.hk  
<https://visual-ai.github.io/raf/>

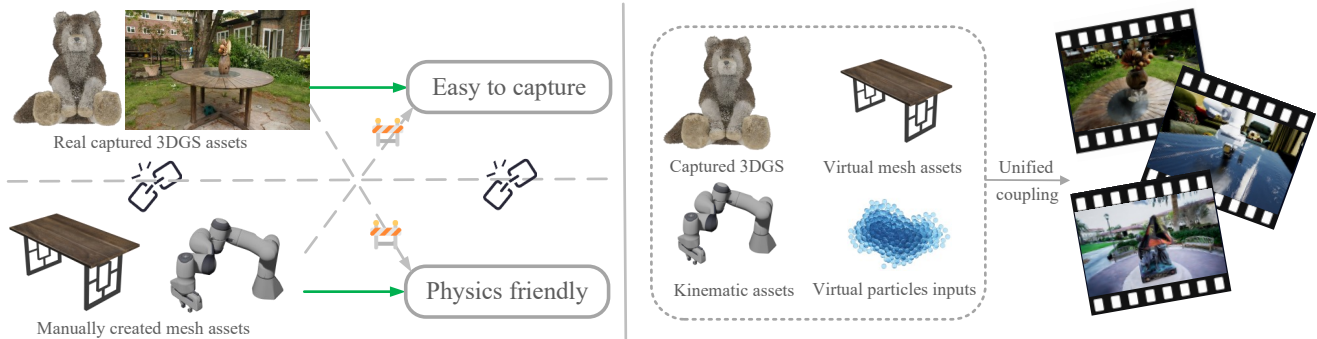


Figure 1. **Bridging the representation gap.** *Left:* Photorealistic 3DGS assets are easy to capture but incompatible with physics engines, while traditional meshes are physics-ready but lack capture fidelity. *Right:* Our representation abstraction framework unifies heterogeneous assets — captured 3DGS, virtual meshes, and particle systems — within a single scene-level physical simulation.

## Abstract

3D Gaussian Splatting (3DGS) has achieved state-of-the-art photorealistic rendering, but the representation gap prevents these assets from being physically interactive. Production-grade physics engines do not understand the 3DGS representation, while prior physics-for-3DGS methods are monolithic silos. These prior works are fundamentally limited, demonstrating only object-centric physics in isolated environments, such as on an ideal plane — they are incapable of interacting with complex static collision geometry or heterogeneous assets. We propose a novel framework that, for the first time, bridges this gap by enabling 3DGS assets to participate in scene-level, heterogeneous, multi-solver physical simulations. Our core contribution is a Representation Abstraction Framework that “translates” all diverse assets—including 3DGS, virtual meshes, and fluids—into a unified physical particle set. This abstraction is key to enabling complex behaviors, such as the non-rigid deformation of 3DGS assets, within a unified physics pipeline. This particle set, along with the static scene collision boundaries derived from scene capture, is processed within a solver-agnostic physics kernel. The physical results are then mapped back to drive each asset’s specific vi-

sual reconstruction. This architecture unlocks capabilities impossible with prior art. We demonstrate complex, two-way interactions between deformable 3DGS assets, standard CG assets (like fluids and meshes), and large-scale captured static environments, showcasing realistic coupled phenomena that were previously unattainable.

## 1. Introduction

3D Gaussian splatting (3DGS) [20] has recently become the state-of-the-art (SOTA) for real-time, photorealistic novel-view synthesis. It represents scenes with millions of explicit Gaussian splats, achieving unprecedented rendering quality and speed. However, while 3DGS has seen immense success in rendering static scenes or replaying pre-captured motion, these high-fidelity worlds remain largely “dead” [1, 17]—they lack physical interactivity. Transforming these assets from “passive renderables” to “active participants” is the next major frontier to unlock their full potential [2].

The core challenge in achieving this stems from a fundamental representation gap [35]. On one hand, 3DGS is a powerful visual representation defined by covariance ( $\Sigma$ ), spherical harmonics ( $\mathcal{C}$ ), and opacity ( $\sigma$ ). On the other hand, powerful, mature, production-grade physics engines (e.g., Unreal Engine [8], Houdini [29]) are representation-

<sup>†</sup>Corresponding authors.

Table 1. **Comparison of physics-based 3D Gaussian splatting methods.** Our framework unifies high-fidelity captured objects (GS particles) with external standard assets (meshes/fluids) in a single scene-level simulation with advanced rendering.

	PhysGaussian [35]	DecoupledGaussian [33]	Gaussian Splashing [10]	GausSim [28]	Ours
Core paradigm	Unified (MPM)	Decoupled & inpaint	Unified (PBD)	Unified (neural)	Hybrid representation
Dynamic object rep.	GS particles only	GS particles only	GS particles only	GS particles only	Hybrid (GS + mesh/fluid)
Sim. captured objects	✓	✓	✓	✓	✓
Sim. external assets	✗	✗	✗	✗	✓
Heterogeneous interaction	✗	✗	✗	✗	✓ (e.g., GS ↔ mesh)
Physics solver generality	Single (MPM)	Single (MPM)	Single (PBD)	Single (neural)	Multiple (MPM, SPH, rigid...)
Scene-level interaction	✗	✓	✗	✗	✓
Complex terrain support	✗ (Ideal plane)	✗ (Ideal plane)	✗ (Ideal plane)	✗ (Ideal plane/void)	✓ (Full 3D)
Rendering pipeline	Custom rasterizer	Custom rasterizer	Custom (PBR)	Custom rasterizer	Industrial (Unreal Engine 5)
Advanced lighting/shadows	✗	✗	Limited	✗	✓ (Lumen, RT)

agnostic, operating on a completely different set of physical representations (like particles, meshes, and rigid bodies) and supporting heterogeneous, multi-solver (MPM [30], SPH [13], PBD [25]) architectures. These engines cannot natively understand the novel 3DGS representation.

To bridge this chasm, prior work was forced down a “siloeed” path [5, 23, 35–37]: they built proprietary, monolithic physics solvers specifically for 3DGS. This approach, however, suffers from two fundamental flaws. First, it is homogeneous: it creates a non-extensible closed system in which only GS particles exist, making it fundamentally incapable of interacting with external, standard CG assets (like meshes, fluids, or rigid bodies). Second, it is isolated: it is not scene-level. It typically operates in a void or on an ideal plane and is fundamentally unable to support complex, pre-existing 3D static collision geometry. This “toy” approach is a dead end for scalable and general physics in complex worlds [9, 38].

In this paper, we argue against re-inventing the wheel with another siloeed solver. Instead, we propose a novel *Representation Abstraction Framework*. Our framework acts as a bridge that, for the first time, unifies 3DGS captured assets, traditional CG assets, and complex static scenes within powerful, “off-the-shelf” heterogeneous physics engines.

This framework aims to solve both flaws of the siloeed path. First, to solve the homogeneous problem, we translate and abstract all diverse dynamic assets—whether 3DGS (as soft bodies), virtual meshes (as PBD deformable bodies), or external fluids (as SPH particles)—into a unified physical particle set. Second, to solve the isolated problem, we also consume the static scene boundaries ( $M_{\text{static}}$ ), which are extracted from the scene capture [33].

These two elements—the unified dynamic particles and the static scene boundaries—are then fed into a unified simulation context. Here, a modular heterogeneous solver handles all physical interactions between assets [26] and the static scene (e.g., particle-mesh collision, fluid-on-soft-body coupling). After simulation, our framework “reverse-

translates” the physical results (e.g., deformation gradient  $F'$ , new position  $x'$ ) and intelligently routes them to each asset’s own specific visual reconstruction pipeline: 3DGS assets use  $F'$  to update their covariance  $\Sigma'$ , while mesh assets use particle positions to drive barycentric skinning.

We demonstrate this end-to-end system by validating it on a series of previously impossible, complex dynamic scenarios. For example, we show a captured 3DGS soft-body bear deforming under a virtual SPH fluid splash while realistically sliding across a captured static garden scene with complex geometry. Our work ultimately transforms 3DGS from static renderables into fully functional, interactable “physical sandboxes”, achieving dual physical and visual high-fidelity within a unified production renderer like Unreal Engine 5 [8].

In this paper, we make the following contributions: (i) We propose the first framework to enable scene-level, heterogeneous, multi-solver physics for 3D Gaussian Splatting. (ii) We introduce a representation abstraction layer that translates diverse dynamic assets (3DGS, meshes, fluids) and complex static scenes into a single physical simulation context. (iii) We demonstrate complex, two-way physical interactions between captured 3DGS assets, standard CG assets, and real-world static scene geometry. (iv) We deliver a complete end-to-end pipeline with unified rendering in Unreal Engine 5, leveraging advanced features such as Lumen global illumination.

## 2. Related work

### 2.1. Physics simulation of 3D Gaussian splatting

Despite 3DGS’s rapid adoption for high-fidelity rendering, prior efforts [5, 10, 16, 35] to endow it with physics have followed a *monolithic and siloeed* paradigm. These works fall into two main categories: analytical solvers (e.g., PhysGaussian [35] and others treating kernels as material points in MPM or adapting them for PBD) and neural solvers (e.g., GausSim [28], which uses a center of mass systems model

[31] governed by continuum mechanics to learn dynamics).

While groundbreaking, this entire monolithic approach suffers from fundamental limitations that restrict its use to “toy” scenarios [2, 33]. First, it locks the asset into a single-solver model, preventing complex multi-material simulation. Second, and more critically, these GS-specific solvers are siloed and non-extensible. They are incapable of interacting with external, standard virtual assets (such as meshes, fluids, or rigid bodies). Consequently, these methods are demonstrated in isolated environments, such as on an ideal plane, and cannot support complex, pre-existing 3D static collision geometry. In summary, the field has been missing a general and scalable solution to integrate high-fidelity GS assets into complex, scene-level simulations. Our work directly confronts this limitation.

## 2.2. Heterogeneous multi-solver physics

In the broader computer graphics community, high-fidelity physical simulation for production has long embraced a heterogeneous, multi-solver architecture [8, 12, 26, 29]. The power of modern physics engines lies not in a single, “one-size-fits-all” solver, but in the ability to couple multiple, specialized solvers, managing complex interactions between assets represented in entirely different ways (e.g., coupling SPH for fluids, MPM for granulars, and PBD for cloth). The core capability of these industrial-strength systems is a unified collision and coupling mechanism that manages interactions between these disparate solvers as a generalized process.

These powerful engines are, however, representation-agnostic. They operate on a common currency of particles and explicit triangle meshes [5]. They have no native understanding of rendering-oriented representations such as NeRF [24] or 3DGS [15, 20]. Therefore, a significant gap exists. Instead of “re-inventing the wheel” by building another limited, GS-specific solver, the more intelligent and scalable path is to bridge this gap. Our core contribution, a *Representation Abstraction Framework*, is precisely this bridge.

## 2.3. Neural scene representations

While implicit representations like NeRF [24] first achieved photorealism, their monolithic structure and slow rendering make them extremely difficult to physically simulate. This limitation led to the shift towards explicit or hybrid representations, culminating in 3D Gaussian splatting, whose explicit, point-based nature is far more suitable for physical interaction than an implicit field. However, recent works on dynamic 3DGS (or 4DGS [27, 34]) focus predominantly on kinematic motion. These methods (e.g., [11]) only replay a pre-captured or generated motion sequence. They are not physics-based; they cannot simulate new dynamics or respond to novel physical interactions. Simulating

new, physically-plausible dynamics for these assets remains a critical open challenge, which our work is the first to tackle within a general-purpose, heterogeneous, and scene-level simulation framework.

## 3. Method

We introduce a novel end-to-end framework that enables captured 3D Gaussian splatting (GS) assets to participate in complex, scene-level physical simulations with heterogeneous virtual assets. The core challenge is that modern physics engines are not designed to process GS representations, while existing GS-specific physics methods are siloed and cannot interact with complex, multi-solver scenes.

Our key contribution is a *Representation Abstraction Framework*, which acts as a bridge to resolve this. As shown in Figure 2, our framework is organized into three stages: (1) heterogeneous asset preparation and abstraction, (2) a unified simulation kernel, and (3) heterogeneous visual recoupling and rendering.

### 3.1. Asset preparation and abstraction

The goal of this stage is to consume a diverse set of optional input assets (shown in Fig. 2, left) and translate them into the two key components required by our simulation kernel: a single static collision mesh ( $\mathcal{M}_{\text{static}}$ ) and a single unified particles list.

**Populating the static world ( $\mathcal{M}_{\text{static}}$ ).** The  $\mathcal{M}_{\text{static}}$  defines all non-moving geometry our dynamic objects can interact with. It is a composite mesh assembled from all static sources, including both captured real-world data and optional imported assets. For in-the-wild scenes, we process multi-view images [3, 4, 22] through our captured asset pipeline. This pipeline uses techniques like Gaussian segmentation [6] and object-scene restoration (leveraging planar-based Gaussian [7] and surface reconstruction [19]) to extract the base static geometry from the real scene capture. Users can also augment the static world with their own custom geometry, such as imported 3D Gaussian splatting (3DGS) assets or virtual mesh assets. All selected static sources are processed and merged into the final watertight  $\mathcal{M}_{\text{static}}$ .

**Populating the dynamic world (unified particle list).** All dynamic participants are processed by our representation abstraction layer and converted into a single unified particles list. This layer handles a wide variety of asset types:

*Captured 3DGS assets:* Dynamic GS objects ( $\mathcal{G}_{\text{dyn}}$ ) are converted into a set of physical particles  $\mathcal{X}_{\text{gs}}$ . If the asset has been pre-processed by the object-scene restoration pipeline of DecoupledGaussian [33], we use its dense points directly. Otherwise, for standard 3DGS captures, we define the object’s interior by computing a continuous opacity field  $d(\mathbf{x})$

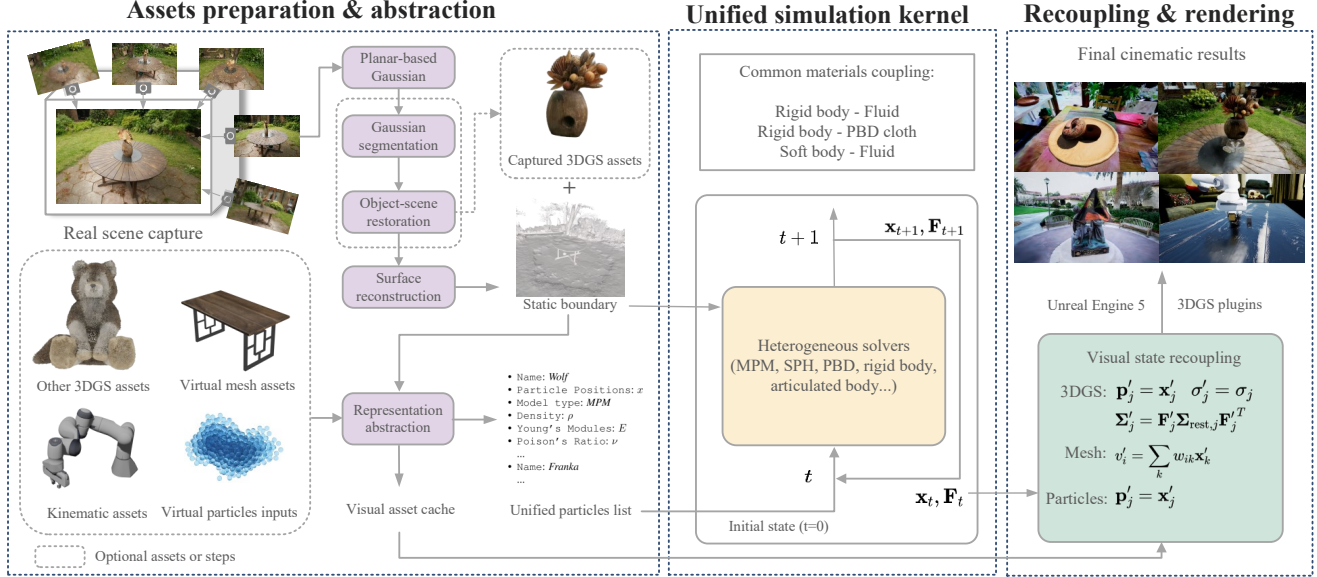


Figure 2. **Architecture of our Representation Abstraction Framework.** (1) *Asset preparation & abstraction.* Heterogeneous inputs (left), including real scene capture and various imported assets, are processed. A static collision mesh ( $M_{\text{static}}$ ) is reconstructed from multi-view images using planar-based Gaussian. The representation abstraction layer translates all assets into a unified particle list. Visual-only data (e.g.,  $\{\Sigma_{\text{rest}}, w_{ik}\}$ ) is cached. (2) *Unified simulation kernel.* The kernel consumes the particles and static mesh, using a suite of heterogeneous solvers (MPM, SPH, PBD-cloth, rigid-body, articulated-body) to output the updated physical state ( $\mathbf{x}', \mathbf{F}'$ ). (3) *Heterogeneous recoupling & rendering.* The visual state recoupling step “reverse-translates” the physical state using the cache. It performs asset-specific reconstruction and serialization. These are loaded into Unreal Engine 5 to produce the final cinematic results.

[35] and sampling it:

$$d(\mathbf{x}) = \sum_{g \in \mathcal{G}_{\text{asset}}} \sigma_g \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{k}_g)^T \Sigma_g^{-1} (\mathbf{x} - \mathbf{k}_g)\right) \quad (1)$$

For each resulting particle position  $\mathbf{x}_j$ , we instantiate a physical entity  $p_j$ . This entity, which is consumed by the simulation kernel, encapsulates all necessary information: its geometric and kinematic state (e.g., position  $\mathbf{x}_j$ , mass  $m_j$ , velocity  $\mathbf{v}_j$ ), its assigned physical model ( $\text{ModelType}_j$ ), and its constitutive parameters (e.g., Young’s modulus  $E$ , Poisson’s ratio  $\nu$  for MPM particles). The full structure is defined as:

$$p_j = \{\mathbf{x}_j, m_j, \mathbf{v}_j, \text{ModelType}_j, E_j, \nu_j, \dots\} \quad (2)$$

Concurrently, the visual entity  $v_j = \{\Sigma_{\text{rest},j}, \mathcal{C}_j, \sigma_j\}$  is stored in the visual asset cache for later recoupling.

*Virtual meshes:* Dynamic meshes are “particle-ized” using Poisson disk sampling.

$$\forall \mathbf{x}_i, \mathbf{x}_k \in \mathcal{X}_{\text{mesh}}, \quad \|\mathbf{x}_i - \mathbf{x}_k\| \geq r \quad \text{for } i \neq k \quad (3)$$

We establish a binding [18] between the template mesh  $\mathcal{M}_{\text{template}}$  and its particles by pre-computing barycentric weights  $\{w_{ik}\}$ , such that its rest position is defined by:

$$v_i^{\text{rest}} = \sum_k w_{ik} \mathbf{x}_k \quad (4)$$

This template mesh and its weights are also stored in the visual asset cache.

*External particle systems (e.g., fluid):* These inputs (Virtual Particles) are loaded directly as a set of initial particle positions  $\mathcal{X}_{\text{particle}}$ . Their `ModelType` is set (e.g., to SPH), and no separate visual entity is required.

*Kinematic assets (e.g., robot arm [14]):* These assets are not converted to particles. They are treated as standard virtual mesh assets (rigid bodies) whose motion is not solved by our physics kernel. Instead, their transforms are kinematically prescribed by the user-defined animation. They act as one-way colliders, affecting other dynamic objects (like the rigid-body cubes) while remaining unaffected by them.

### 3.2. Unified simulation kernel

This stage acts as a modular, solver-agnostic physics kernel. As shown in Figure 2, the kernel takes two unified inputs from Stage 1: the complete static collision mesh ( $\mathcal{M}_{\text{static}}$ ) and the unified particles list at the current time  $t$ .

The particles are initialized with their current physical state ( $\mathbf{x}_t, \mathbf{F}_t$ ), derived from the initial state ( $t = 0$ ) or the previous loop’s output. These inputs are processed by the heterogeneous solvers module, which manages and seamlessly interfaces with various specialized solvers (e.g., MPM, SPH, PBD-cloth, rigid body, and articulated body).

The kernel’s core capability is its ability to robustly manage heterogeneous interactions. This includes handling complex, continuous force exchanges (e.g., pressure from SPH particles onto MPM particles) as well as discrete collision events.

For discrete collision and frictional contact between all entities (e.g., particle-mesh or rigid-body-on-rigid-body), the kernel employs a unified impulse-based mechanism. As an example of this discrete coupling mechanism, when two entities  $i$  and  $j$  collide, a response impulse is computed to determine the post-collision velocities. Let  $r_n$  and  $\mathbf{r}_t$  denote the normal and tangential components of the relative velocity between the two colliding entities. The post-collision velocity components are determined by:

$$r_n^* = -er_n \quad (5)$$

$$\mathbf{r}_t^* = \mathbf{r}_t \max(0, 1 - \mu|r_n|/|\mathbf{r}_t|) \quad (6)$$

where  $e$  is the coefficient of restitution and  $\mu$  is the Coulomb friction coefficient.

This unified impulse-based approach ensures non-penetrating and frictional response across all dynamic assets when they collide with each other or with  $\mathcal{M}_{\text{static}}$ . Within each time step, the heterogeneous solvers are executed iteratively in a specific sequence — rigid-body and articulated-body solvers first, followed by MPM, SPH, and finally PBD — with updated particle states propagated between solvers and refined through inter-solver iterations, ensuring that complex coupling forces and constraints are consistently resolved within the same step. More detailed solver and continuous coupling implementations are provided in the Supp. S6. At the end of the step  $\Delta t$ , the simulation kernel returns the updated physical state ( $\mathbf{x}_{t+1}, \mathbf{F}_{t+1}$ ), which is passed to the final stage for visual reconstruction.

### 3.3. Heterogeneous recoupling and rendering

This final stage reverse-translates the sparse, abstract physical data from the simulation kernel back into high-fidelity, renderable visual assets. This stage requires two inputs: the updated physical state ( $\mathbf{x}', \mathbf{F}'$ ) from Stage 2, and the visual asset cache (containing  $\{\Sigma_{\text{rest}}, w_{ik}, \dots\}$ ) from Stage 1.

**Visual state recoupling.** The visual state recoupling step performs asset-specific reconstruction in parallel for all dynamic entities. As shown in Figure 2, different asset types use different reconstruction logic:

- *3DGS assets:* We propagate the physical deformation to the visual attributes. The new Gaussian center  $\mathbf{p}'_j$  is set to the updated particle position.

$$\mathbf{p}'_j = \mathbf{x}'_j \quad (7)$$

The new covariance  $\Sigma'_j$  is computed by applying the physical deformation  $\mathbf{F}'_j$  to the cached rest-state covari-

ance  $\Sigma_{\text{rest},j}$ :

$$\Sigma'_j = \mathbf{F}'_j \Sigma_{\text{rest},j} \mathbf{F}'_j{}^T \quad (8)$$

The opacity  $\sigma'_j$  is assumed to be an invariant material property and is carried over from its rest state:

$$\sigma'_j = \sigma_j \quad (9)$$

Finally, the rotation (extracted from  $\mathbf{F}'_j$ ) is used to rotate the SH coefficients  $\mathcal{C}_j$ . The mathematical derivation proving this  $F$  propagation to  $\Sigma$  is physically consistent is provided in the Supp. S3.

- *Mesh assets:* We retrieve the asset’s template mesh  $\mathcal{M}_{\text{template}}$  and its precomputed barycentric weights  $\{w_{ik}\}$  from the cache. The new deformed vertex position  $v'_i$  is computed by applying these same weights to the new particle positions  $\mathbf{x}'_k$ :

$$v'_i = \sum_k w_{ik} \mathbf{x}'_k \quad (10)$$

- *Particle assets:* For fluid assets (SPH), the physical particle position is its own visual representation:  $\mathbf{p}'_j = \mathbf{x}'_j$ .

**Serialization and final rendering.** The recoupled visual data is serialized into asset-aware formats (e.g., `.ply` files for 3DGS, `.abc` files for meshes, `.pbd` caches for particles). These serialized files are then loaded into our final cinematic results pipeline. We utilize Unreal Engine 5 as our final renderer. The engine natively ingests the `.abc` and `.pbd` caches, while our 3DGS plugins [32] (a compatible Gaussian Splatting rendering component) load the per-frame `.splat` files. This final step allows all our co-simulated assets to be rendered together, interacting correctly with advanced features like Lumen global illumination and ray-traced shadows.

## 4. Experiments

We validate our framework across five complex scenarios that are, by design, unattainable by prior siloed methods. Each scenario exercises a distinct capability: scene-level composition, heterogeneous multi-solver coupling, articulated dynamics, complex geometry interaction, and integration with imported 3DGS assets — all while maintaining photorealistic rendering via Unreal Engine 5.

### 4.1. Implementation details

**Simulation kernel.** We implement our core abstraction layer and simulation loop (described in Sec. 3). Our unified kernel, which is built upon the Genesis-world physics solver [12], acts as a modular, solver-agnostic kernel that manages and dispatches tasks to specialized solvers, including MPM (for soft bodies), SPH (for fluids), PBD (for cloth), standard rigid-body dynamics (for objects), and articulated-body dynamics (for kinematic chains).

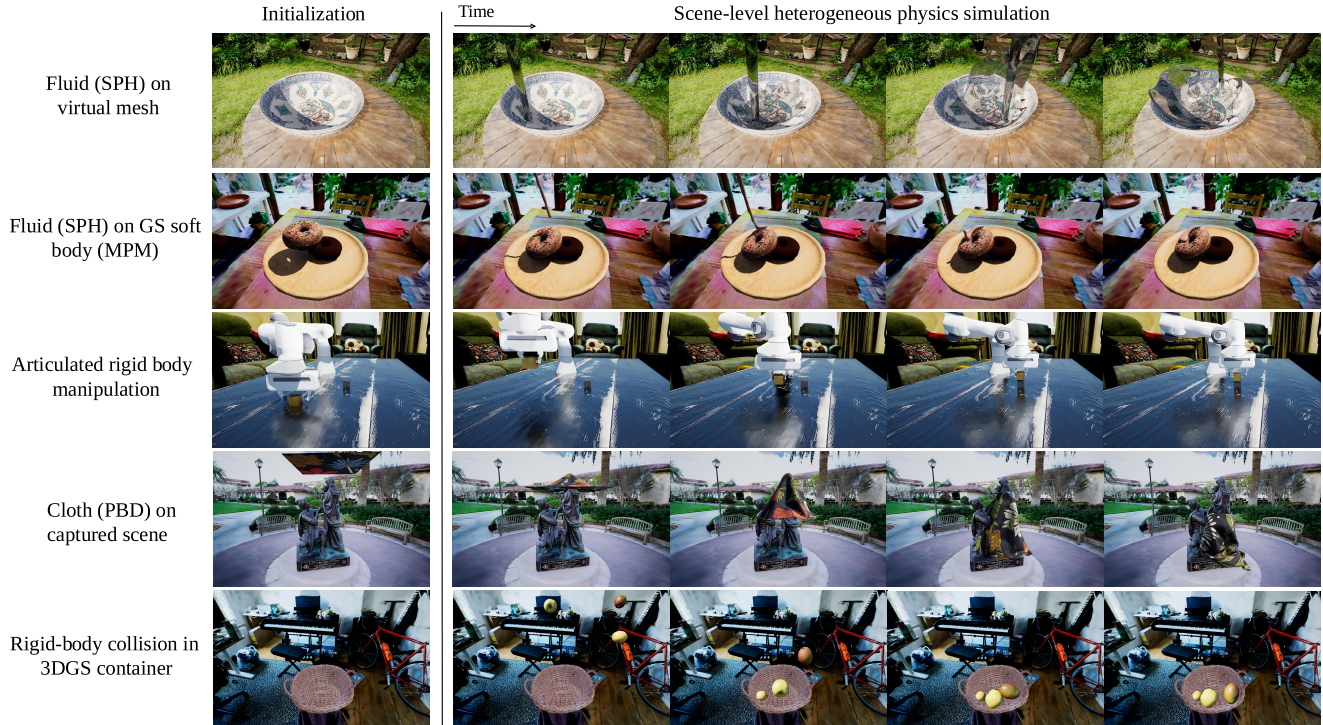


Figure 3. **Our qualitative showcase of new, scene-level heterogeneous simulations produced using our framework.** Each row demonstrates a unique coupling of different asset types and physics solvers, executed within a captured 3DGS environment and rendered in Unreal Engine 5. *Row 1: Fluid (SPH) on virtual mesh.* A virtual fluid (SPH) is poured into an imported virtual mesh (bowl), interacting with both the bowl and the static 3DGS scene (garden table). *Row 2: Fluid (SPH) on GS soft body (MPM).* A high-viscosity fluid (SPH) is coupled with a captured 3DGS soft body (MPM donut), demonstrating complex multi-solver, two-way interaction on a plate. *Row 3: Articulated rigid body manipulation.* A kinematically-driven robotic arm (articulated-body solver) manipulates a virtual rigid body (cube) on a table. *Row 4: Cloth (PBD) on captured statue.* A virtual cloth (PBD solver) realistically drapes and slides over the complex, non-convex geometry of a captured 3DGS statue, refuting the “ideal plane” limitation. *Row 5: Rigid-body collision in 3DGS container.* Multiple virtual rigid bodies (fruits) interact with each other and with the complex geometry of an imported 3DGS asset (the basket). These scenarios are, by design, unattainable by prior “siloe” methods.

**Asset preparation.** The preparation of static and dynamic assets follows the structure detailed in Section 3.1. For scene-level static geometry, two strategies are employed to construct the static collision mesh ( $\mathcal{M}_{\text{static}}$ ): direct capture and composition. For scenes captured in situ (e.g., *Kitchen, Statue*), we utilize our full asset pipeline to process multi-view images and extract the base static geometry ( $\mathcal{M}_{\text{static}}$ ) from the real scene capture. For compositional scenarios (e.g., *Garden, Rooms*), we load a primary 3DGS scene and then import additional assets—such as virtual mesh bowls or pre-captured 3DGS fruit baskets—which are also processed and merged into the final static collision geometry. Dynamic assets are uniformly converted into the unified particle list, regardless of their original source format.

**Rendering.** All scenarios are serialized (e.g., `.splat`, `.abc` caches) and rendered in Unreal Engine 5 using a compatible 3DGS rendering component and leveraging Lu-

men for global illumination and shadows.

**Baselines for comparison.** Our primary baselines are the “siloe” SOTA methods (e.g., PhysGaussian [35], GausSim [28]). Crucially, these baselines cannot be applied to our experimental settings. They are not designed for heterogeneous, scene-level composition. Our qualitative showcase (Sec. 4.2) therefore demonstrates capabilities that are, by design, impossible for them.

#### 4.2. Qualitative showcase of new capabilities

We present five complex scenarios, visualized in Figure 3 and our supplementary video, to demonstrate the breadth of new capabilities unlocked by our framework. These scenarios are chosen to validate our core claims by showcasing: (1) scene composition (SPH on virtual mesh), (2) multi-solver coupling (SPH on MPM 3DGS), (3) articulated dynamics (robot arm), (4) complex geometry inter-

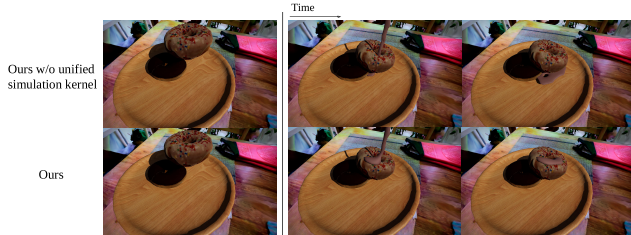


Figure 4. **Ablation study on our unified kernel.** *Top row* (Ours w/o unified simulation kernel): Our ablated baseline. Without the heterogeneous coupling mechanism, the SPH fluid particles (e.g., *Sauce*) and the MPM soft-body particles (e.g., *Donut*) do not interact, passing through each other in a physically implausible manner. *Bottom row* (Ours): Our full framework. The unified kernel correctly handles the SPH-on-MPM collision and coupling, allowing the sauce to realistically collide with, deform, and settle on top of the 3DGS *Donut*.

action (PBD-cloth on statue), and (5) interaction with imported 3DGS assets (rigid-bodies in 3DGS container). As the figure’s caption details, these scenarios—which unify SOTA physics and rendering—are unattainable by prior siloed methods.

**Summary of qualitative findings.** These five scenarios, taken together, validate our core claims. They demonstrate the unification of a wide range of solvers and an expansive set of heterogeneous assets (captured GS scenes, captured GS objects, imported GS assets, and virtual meshes/particles). Critically, Row 1 and Row 5 in Figure 3 demonstrate that our framework is a powerful composition engine for 3DGS, a capability far beyond the ideal plane limitations of prior “siloed” methods like PhysGaussian or GausSim.

### 4.3. Ablation study and comparative analysis

To scientifically validate the necessity and effectiveness of our core contributions, we conduct a series of rigorous ablation studies and visual comparisons. As no prior method supports heterogeneous, scene-level simulation, a direct numerical comparison with prior “siloed” methods (like PhysGaussian [35] or GausSim [28]) is not applicable. Instead, we ablate our own components to demonstrate their necessity.

#### 4.3.1. Ablation study

We isolate three core components of our framework: the unified coupling kernel, the scene-level geometry, and the representation abstraction.

**Validating the unified kernel.** To demonstrate the necessity of our heterogeneous coupling mechanism, we compare our full model against an ablated version, Ours w/o

unified simulation kernel. In this baseline, the SPH and MPM solvers operate in the same space but do not interact. As visualized in Figure 4, the ablated version (top row) fails, as the sauce and donut pass through each other, resulting in a physically implausible simulation. Our full model (bottom row) correctly handles the SPH-on-MPM coupling, where the (SPH) sauce collides with and deforms the (MPM) donut, demonstrating the critical importance of our unified kernel.

**Validating scene-level interaction.** To demonstrate that our framework has overcome the “ideal plane” limitation of prior “siloed” work, we compare our full model against an ablated baseline, Ours w/o static collision mesh. As visualized in Figure 5, the ablated baseline (top row) fails completely; without the complex  $\mathcal{M}_{\text{static}}$  from our asset preparation workflow, the PBD-cloth solver defaults to an “ideal plane” and the cloth passes nonsensically through the captured statue. Our full method (bottom row) correctly uses the statue’s high-fidelity collision mesh, resulting in realistic and complex draping behavior. This directly demonstrates that our scene-level pipeline is essential for integrating assets into captured environments.

**Validating representation abstraction.** Finally, we validate the importance of our visual recoupling step. We compare our full result against Ours w/o representation abstraction. As shown in Figure 6, the top row visualizes only the sparse, underlying simulation particles, failing to convey the photorealistic appearance of the asset. Our full method demonstrates how our recoupling “bridge” is essential for translating this sparse physical simulation back into a high-fidelity visual result.

#### 4.3.2. Comparison to photogrammetry pipeline

Having validated our pipeline’s internal components, we now validate our core motivation for using 3DGS. A critic might ask: why not use standard photogrammetry, which also produces physics-ready meshes from images? We demonstrate this fundamental trade-off our framework addresses.

We compare our method against a baseline generated from the same multi-view images using KiriEngine [21]. As shown in Figure 7, the photogrammetry baseline (left) exhibits severe geometric artifacts, appearing “blobby” or distorted on complex structures like bulldozer treads and table slats. In contrast, our pipeline (right), leveraging PGRS [7], preserves sharp and geometrically complete details. This demonstrates that while traditional meshes are natively physics-ready, they lack the fidelity required for complex scenes. Our framework bridges this gap, enabling



Figure 5. **Ablation study on scene-level geometry.** *Top row* (Ours w/o static collision mesh): Our ablated baseline. Without the static collision mesh ( $\mathcal{M}_{\text{static}}$ ), the cloth simulation is limited to an “ideal plane” and incorrectly passes through the statue. *Bottom row* (Ours): Our full framework. The PBD-cloth solver correctly interacts with the complex, non-convex geometry of the captured 3DGS statue (represented by  $\mathcal{M}_{\text{static}}$ ), resulting in realistic draping, folding, and sliding.

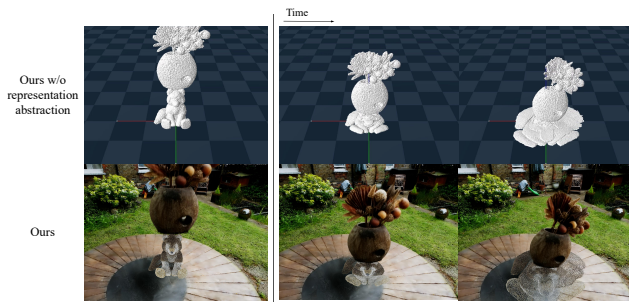


Figure 6. **Ablation study on our representation abstraction.** *Top row* (Ours w/o representation abstraction): Without the representation abstraction, only the sparse simulation particles are visible, lacking continuous surfaces and photorealistic appearance. *Bottom row* (Ours): Our full recoupling layer recovers high-fidelity visual attributes from the sparse physical state, demonstrating its necessity for photorealistic rendering.

high-fidelity 3DGS assets to participate in advanced physics simulations for the first time.

## 5. Conclusion

In this paper, we introduced the *Representation Abstraction Framework* (RAF), the first end-to-end pipeline to bridge the fundamental gap between 3DGS assets and industrial-strength heterogeneous physics engines. Our core innovation is an abstraction layer that unifies diverse visual representations into a single physical particle set consumable by a multi-solver kernel (MPM, SPH, PBD, etc.). This architecture overcomes the limitations of prior “siloed” physics-for-GS methods, enabling scene-level composition, complex two-way heterogeneous coupling, and realistic interaction with complex static geometry — all rendered with photorealistic fidelity in Unreal Engine 5. Our ablation studies

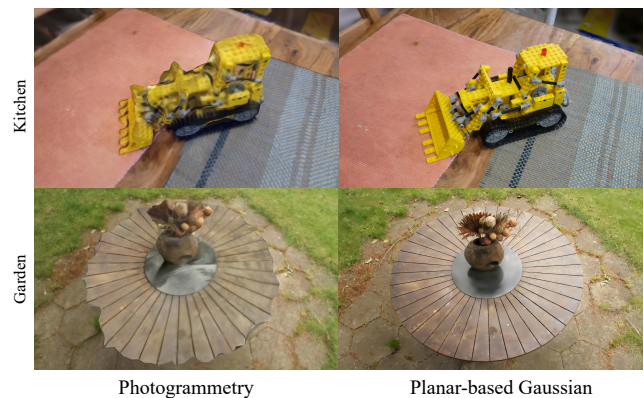


Figure 7. **Comparison with photogrammetry.** *Left (Photogrammetry)*: Standard mesh extraction from images suffers from severe geometric artifacts and missing details on fine structures. *Right (Ours)*: By leveraging 3DGS, our pipeline preserves sharp, complex geometries, which our RAF further transforms from passive renderables into interactive physical assets.

confirmed the necessity of the unified kernel, scene-level geometry handling, and visual recoupling bridge. Currently, our pipeline targets offline, cinematic-quality production; extending it to real-time interactive scenarios and automating physical material estimation from captured appearances are promising directions for future work.

**Acknowledgement** This work is supported by Hong Kong Research Grant Council - General Research Fund (Grant No. 17213825) and HKU Seed Fund for PI Research.

## References

- [1] Jad Abou-Chakra, Krishan Rana, Feras Dayoub, and Niko Sünderhauf. Physically embodied gaussian splatting: Embedding physical priors into a visual 3d world model for robotics. In *CoRL*, 2023. 1
- [2] Yanqi Bao, Tianyu Ding, Jing Huo, Yaoli Liu, Yuxin Li, Wenbin Li, Yang Gao, and Jiebo Luo. 3d gaussian splatting: Survey, technologies, challenges, and opportunities. *IEEE TCSVT*, 2025. 1, 3
- [3] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *ICCV*, 2021. 3
- [4] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *CVPR*, 2022. 3
- [5] Piotr Borycki, Weronika Smolak, Joanna Waczyńska, Marcin Mazur, Sławomir Tadeja, and Przemysław Spurek. Gasp: Gaussian splatting for physic-based simulations. *arXiv preprint arXiv:2409.05819*, 2024. 2, 3
- [6] Jiazhong Cen, Jiemin Fang, Chen Yang, Lingxi Xie, Xiaopeng Zhang, Wei Shen, and Qi Tian. Segment any 3d gaussians. In *AAAI*, 2025. 3
- [7] Danpeng Chen, Hai Li, Weicai Ye, Yifan Wang, Weijian Xie, Shangjin Zhai, Nan Wang, Haomin Liu, Hujun Bao, and Guofeng Zhang. Pgsr: Planar-based gaussian splatting for efficient and high-fidelity surface reconstruction. *IEEE TVCG*, 2024. 3, 7
- [8] Epic Games. Unreal engine. 1, 2, 3
- [9] Ben Fei, Jingyi Xu, Rui Zhang, Qingyuan Zhou, Weidong Yang, and Ying He. 3d gaussian splatting as new era: A survey. *IEEE TVCG*, 2024. 2
- [10] Yutao Feng, Xiang Feng, Yintong Shang, Ying Jiang, Chang Yu, Zeshun Zong, Tianjia Shao, Hongzhi Wu, Kun Zhou, Chenfanfu Jiang, et al. Gaussian splashing: Unified particles for versatile motion synthesis and rendering. In *CVPR*, 2025. 2
- [11] Quankai Gao, Qiangeng Xu, Zhe Cao, Ben Mildenhall, Wenchao Ma, Le Chen, Danhang Tang, and Ulrich Neumann. Gaussianflow: Splatting gaussian dynamics for 4d content creation. *Transactions on Machine Learning Research*, 2025. 3
- [12] Genesis Authors. Genesis: A Generative and Universal Physics Engine for Robotics and Beyond, 2024. 3, 5
- [13] Robert A Gingold and Joseph J Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly notices of the royal astronomical society*, 1977. 2
- [14] Sami Haddadin, Sven Parusel, Lars Johannsmeier, Saskia Golz, Simon Gabl, Florian Walch, Mohamadreza Sabaghian, Christoph Jähne, Lukas Hausperger, and Simon Haddadin. The franka emika robot: A reference platform for robotics research and education. *IEEE Robotics & Automation Magazine*, 2022. 4
- [15] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. In *ACM SIGGRAPH*, 2024. 3
- [16] Tianyu Huang, Yihan Zeng, Hui Li, Wangmeng Zuo, and Rynson WH Lau. Dreamphysics: Learning physical properties of dynamic 3d gaussians with video diffusion priors. In *AAAI*, 2025. 2
- [17] Ying Jiang, Chang Yu, Tianyi Xie, Xuan Li, Yutao Feng, Huamin Wang, Minchen Li, Henry Lau, Feng Gao, Yin Yang, et al. Vr-gs: A physical dynamics-aware interactive gaussian splatting system in virtual reality. In *ACM SIGGRAPH*, 2024. 1
- [18] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O’Sullivan. Geometric skinning with approximate dual quaternion blending. *ACM TOG*, 2008. 4
- [19] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, 2006. 3
- [20] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM TOG*, 2023. 1, 3
- [21] Kiri Engine Co., Ltd. KiriEngine Photogrammetry Software. Official Website, 2024. 7
- [22] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM TOG*, 2017. 3
- [23] Fangfu Liu, Hanyang Wang, Shunyu Yao, Shengjun Zhang, Jie Zhou, and Yueqi Duan. Physics3d: Learning physical properties of 3d gaussians via video diffusion. *arXiv preprint arXiv:2406.04338*, 2024. 2
- [24] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *ECCV*, 2020. 3
- [25] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 2007. 2
- [26] NVIDIA, Google DeepMind, and Disney Research. Announcing Newton, an Open-Source Physics Engine for Robotics Simulation. NVIDIA Developer Blog, 2025. Accessed on 2025-11-14. 2, 3
- [27] Jiawei Ren, Liang Pan, Jiayang Tang, Chi Zhang, Ang Cao, Gang Zeng, and Ziwei Liu. Dreamgaussian4d: Generative 4d gaussian splatting. *arXiv preprint arXiv:2312.17142*, 2023. 3
- [28] Yidi Shao, Mu Huang, Chen Change Loy, and Bo Dai. Gaus-sim: Foreseeing reality by gaussian simulator for elastic objects. In *ICCV*, 2025. 2, 6, 7
- [29] SideFX. Houdini 20.0. <https://www.sidefx.com/>, 2024. Accessed on 2025-11-14. 1, 3
- [30] Deborah Sulsky, Zhen Chen, and Howard L Schreyer. A particle method for history-dependent materials. *Computer methods in applied mechanics and engineering*, 1994. 2
- [31] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 1987. 3
- [32] Volinga AI. Volinga: Interactive 3D Gaussian Splatting Viewer. Official Website, 2024. 5

- [33] Miaowei Wang, Yibo Zhang, Weiwei Xu, Rui Ma, Changqing Zou, and Daniel Morris. Decoupledgaussian: Object-scene decoupling for physics-based interaction. In *CVPR*, 2025. 2, 3
- [34] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 4d gaussian splatting for real-time dynamic scene rendering. In *CVPR*, 2024. 3
- [35] Tianyi Xie, Zeshun Zong, Yuxing Qiu, Xuan Li, Yutao Feng, Yin Yang, and Chenfanfu Jiang. Physgaussian: Physics-integrated 3d gaussians for generative dynamics. In *CVPR*, 2024. 1, 2, 4, 6, 7
- [36] Tianyuan Zhang, Hong-Xing Yu, Rundi Wu, Brandon Y Feng, Changxi Zheng, Noah Snavely, Jiajun Wu, and William T Freeman. Physdreamer: Physics-based interaction with 3d objects via video generation. In *ECCV*, 2024.
- [37] Haoyu Zhao, Hao Wang, Xingyue Zhao, Hao Fei, Hongqiu Wang, Chengjiang Long, and Hua Zou. Physplat: Efficient physics simulation for 3d scenes via mllm-guided gaussian splatting. In *ICCV*, 2025. 2
- [38] Siting Zhu, Guangming Wang, Xin Kong, Dezhi Kong, and Hesheng Wang. 3d gaussian splatting in robotics: A survey. *arXiv preprint arXiv:2410.12262*, 2024. 2

# Scene-Level Heterogeneous Physics Simulation with 3D Gaussian Splats

## Supplementary Material

### Overview

In this supplementary material, we provide additional technical details and results. Section [S1](#) presents more simulation cases; Section [S2](#) details the particle initialization process; Section [S3](#) derives the physical consistency of Gaussian covariance recoupling; Section [S4](#) provides the performance analysis; Section [S5](#) describes our physics solvers, and Section [S6](#) explains the coupling mechanisms.

### Contents

<a href="#">S1.</a>	More Simulation Cases .....
<a href="#">S2.</a>	Particles Initialization .....
<a href="#">S3.</a>	Physical Consistency of Gaussian Covariance Recoupling .....
<a href="#">S4.</a>	Time Complexity and Performance Analysis .....
<a href="#">S5.</a>	Physics Simulation Solvers .....
<a href="#">S6.</a>	The Coupling Mechanism of Unified Simulation Kernel .....

## S1. More Simulation Cases

Due to the dynamic nature of physics-based simulation, static images cannot fully convey the temporal coherence and stability of our results. We provide a self-contained offline webpage in the supplementary material for better visualization. Please refer to `index.html` in the attached zip file to view comprehensive video demonstrations, including heterogeneous material assignments, granular-soft body interactions, and comparisons with baseline methods.

## S2. Particles Initialization

### S2.1. Particles Filling for Gaussian Splatting

To enable the physical simulation of 3D Gaussian Splatting (3DGS) assets, which are inherently surface-focused representations, we employ a robust internal particle filling mechanism to generate the necessary volumetric data. Following prior methodologies, we first define the continuous opacity field  $d(x)$  derived from the static GS representation  $\mathcal{G}_{static}$ :

$$d(x) = \sum_{g \in \mathcal{G}_{static}} \sigma_g \exp\left(-\frac{1}{2}(x - \mathbf{k}_g)^T \Sigma_g^{-1}(x - \mathbf{k}_g)\right)$$

This continuous field is discretized onto a dense Eulerian grid with a cell size of  $\Delta x$ . We identify the interior region of the object using a user-defined opacity threshold  $\sigma_{th}$ . Specifically, a grid cell  $i$  is considered internal if its center opacity  $d(\mathbf{x}_i) > \sigma_{th}$ .

Once the internal volume is identified, we generate the physical particle set  $\mathcal{X}_{GS}$  by randomly sampling positions  $\mathbf{x}_j$  within the designated internal grid cells. We employ a Poisson Disk Sampling strategy to ensure a quasi-uniform distribution of particles, preventing clustering and enhancing simulation stability. The minimum sampling distance is set to  $r_{min} = 0.5\Delta x$ .

Crucially, to address potential density disparities during multi-physics coupling, we implement a density homogenization step. If the native particle resolution of the 3DGS asset is significantly lower than that of interacting external solvers (e.g., high-resolution fluids), numerical instability or boundary leakage may occur. In such cases, we inject auxiliary ghost particles into the internal volume to match the target simulation density. These ghost particles participate fully in the physical time integration to enforce robust contact constraints but are explicitly excluded from the rendering pipeline.

### S2.2. Physics Property Allocation

Each sampled physical particle  $p_j \in \mathcal{X}_{GS}$ , including the ghost particles generated for stability, must be assigned kinematic and constitutive properties for the Material Point Method (MPM) simulation.

- **Initial Position and Velocity:** The initial position  $\mathbf{x}_j^0$  is the sampled position. The initial velocity  $\mathbf{v}_j^0$  is set to zero or inherited from a kinematic input.
- **Mass and Volume:** We assume the object has a constant, user-specified material density  $\rho_{mat}$ . The initial volume  $V_j^0$  of each particle is determined by the total volume of the object's occupied space  $V_{total}$  divided by the total number of particles  $N_{total}$ :  $V_j^0 = V_{total}/N_{total}$ . The mass is then calculated as  $m_j = \rho_{mat}V_j^0$ .  $V_{total}$  is approximated by summing the volume of all internal grid cells identified in Section S2.
- **Initial Deformation Gradient:** The initial elastic deformation gradient is set to the identity matrix:  $\mathbf{F}_j^{E,0} = \mathbf{I}$ .
- **Visual Entity Recoupling:** For subsequent visual reconstruction, we store the particle's corresponding rest-state Gaussian covariance  $\Sigma_{rest,j}$  in the Visual Asset Cache.  $\Sigma_{rest,j}$  is either initialized from the nearest existing GS kernel or set as an isotropic sphere  $\text{diag}(r_p^2, r_p^2, r_p^2)$  where  $r_p$  is the particle radius inferred from  $V_j^0$ . For ghost particles, this step is skipped to ensure they remain invisible.

## S3. Physical Consistency of Gaussian Covariance Recoupling

The visual recoupling of the 3D Gaussian covariance matrix  $\Sigma$  is derived from the principles of continuum mechanics, specifically how a local affine transformation affects a Gaussian function.

A 3D Gaussian kernel  $G(\mathbf{X})$  in the material space  $\Omega^0$  is defined by its center  $\mathbf{X}_p$  and covariance matrix  $A_p$  (denoted as  $\Sigma_{rest}$  in our notation):

$$G(\mathbf{X}) = e^{-\frac{1}{2}(\mathbf{X} - \mathbf{X}_p)^T A_p^{-1}(\mathbf{X} - \mathbf{X}_p)}$$

Under a continuous deformation map  $\mathbf{x} = \phi(\mathbf{X}, t)$ , the deformed kernel  $G(\mathbf{x}, t)$  in the world space  $\Omega^t$  is:

$$G(\mathbf{x}, t) = e^{-\frac{1}{2}(\phi^{-1}(\mathbf{x}, t) - \mathbf{X}_p)^T A_p^{-1}(\phi^{-1}(\mathbf{x}, t) - \mathbf{X}_p)}$$

Following PhysGaussian, we assume the material point undergoes a local affine transformation characterized by the first-order Taylor expansion around  $\mathbf{X}_p$ :

$$\tilde{\phi}(\mathbf{X}, t) \approx \mathbf{x}_p + \mathbf{F}_p(\mathbf{X} - \mathbf{X}_p)$$

where  $\mathbf{F}_p = \nabla_{\mathbf{X}}\phi(\mathbf{X}, t)$  is the deformation gradient at point  $\mathbf{X}_p$ . By inverting this affine map and substituting it back into the exponent, the deformed kernel  $\tilde{G}(\mathbf{x}, t)$  remains a Gaussian distribution in the world space  $\Omega^t$  with a new center  $\mathbf{x}_p$  and a new covariance matrix  $\mathbf{a}_p$  (denoted as  $\Sigma'$  in our notation):

$$\tilde{G}(\mathbf{x}, t) = e^{-\frac{1}{2}(\mathbf{x} - \mathbf{x}_p)^T (\mathbf{F}_p A_p \mathbf{F}_p^T)^{-1}(\mathbf{x} - \mathbf{x}_p)}$$

This derivation provides the physically grounded rule for updating the covariance matrix:

$$\Sigma' = \mathbf{F}\Sigma_{rest}\mathbf{F}^T$$

This ensures that the shape and orientation of the 3D Gaussian ellipsoids evolve consistently with the underlying continuum mechanics, preserving the non-rigid deformation (stretch and shear) induced by the simulation.

## S4. Time Complexity and Performance Analysis

**Rendering Workflow and Comparison.** A core contribution of our framework is the seamless integration of physically augmented 3D Gaussian Splatting (3DGS) assets into industrial-grade rendering ecosystems, specifically Unreal Engine 5 (UE5). Unlike prior methods that rely on standalone, Python-based rasterizers for visualization, our pipeline involves exporting the deformed states to UE5 to leverage advanced features such as global illumination and complex shadowing. Since this workflow includes manual operations (e.g., asset importation, scene composition, and sequence rendering in UE5), a direct runtime comparison with end-to-end, code-only rendering methods is neither feasible nor meaningful. Our focus is on achieving cinematic-quality offline rendering rather than real-time frame rates.

**Simulation Costs.** The computational cost of the physical simulation stage is inherently variable, governed largely by the user-defined simulation precision (grid resolution) and the complexity of the multi-physics coupling (e.g., particle density required for stability). However, the simulation remains efficient for offline content creation. For all five main demonstration scenarios presented in the paper, the physical simulation process was completed in under 10 minutes per sequence on a standard workstation.

**Summary.** We explicitly clarify that our method does not target real-time interactivity. Instead, the primary design goal is to unlock high-fidelity, heterogeneous interactions between captured 3DGS scenes and diverse virtual assets (meshes, particles)—scenarios that require complex solver coupling and are best served by an offline, high-quality rendering workflow.

## S5. Physics Simulation Solvers

### S5.1. Rigid Body Dynamics

The Rigid Body Solver is designed to simulate articulated mechanisms and multi-body systems using the Reduced Coordinate formulation (also known as Generalized Coordinates). Unlike maximal coordinate methods that treat every link as a free body constrained by algebraic equations, our solver parameterizes the system using joint an-

gles  $\mathbf{q} \in \mathbb{R}^{n_{dof}}$  and joint velocities  $\dot{\mathbf{q}} \in \mathbb{R}^{n_{dof}}$ . This approach naturally satisfies joint constraints and significantly improves computational efficiency for kinematic chains.

#### S5.1.1. Equations of Motion

The dynamics of the articulated system are governed by the standard manipulator equation:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau}_{ctrl} + \boldsymbol{\tau}_{ext} + \boldsymbol{\tau}_c \quad (\text{S1})$$

where:

- $\mathbf{M}(\mathbf{q})$  is the symmetric, positive-definite joint-space inertia matrix (Mass Matrix).
- $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$  represents the Coriolis and centrifugal forces.
- $\mathbf{g}(\mathbf{q})$  is the gravity vector generalized to joint space.
- $\boldsymbol{\tau}_{ctrl}$ ,  $\boldsymbol{\tau}_{ext}$ , and  $\boldsymbol{\tau}_c$  are the torques/forces from actuation, external disturbances, and contact constraints, respectively.

In our implementation, the bias terms  $\mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{C}\dot{\mathbf{q}} + \mathbf{g}$  are computed efficiently using recursive kinematics.

#### S5.1.2. Composite Rigid Body Algorithm (CRBA)

To compute the mass matrix  $\mathbf{M}(\mathbf{q})$ , we employ the Composite Rigid Body Algorithm (CRBA). The algorithm operates in two passes. First, it computes the composite inertia  $I_i^c$  for each link  $i$ , which represents the inertia of the subtree rooted at  $i$  as if all joints in the subtree were locked:

$$I_i^c = I_i + \sum_{j \in \text{children}(i)} {}^i\mathbf{X}_j^* I_j^c {}^j\mathbf{X}_i \quad (\text{S2})$$

where  ${}^j\mathbf{X}_i$  is the spatial transform from link  $i$  to  $j$ . Second, the entries of  $\mathbf{M}$  are populated by projecting the composite inertias onto the joint motion subspaces. This typically achieves  $O(N^2)$  complexity but is highly optimized for parallel execution on GPUs.

#### S5.1.3. Implicit Integration and Linear Solver

To ensure stability under high stiffness (PD control) and damping, we implement a semi-implicit integration scheme (often referred to as "Implicit Fast"). We explicitly modify the mass matrix to include the contributions of damping  $D$  and stiffness  $K$  matrices:

$$\tilde{\mathbf{M}} = \mathbf{M} + \mathbf{D}\Delta t + \mathbf{K}\Delta t^2 \quad (\text{S3})$$

The discrete-time linear system  $\tilde{\mathbf{M}}\Delta\dot{\mathbf{q}} = \tilde{\boldsymbol{\tau}}\Delta t$  is then solved to find the change in velocity. We utilize a dense  $LDL^T$  Cholesky decomposition to factorize  $\tilde{\mathbf{M}}$ :

$$\tilde{\mathbf{M}} = \mathbf{L}\mathbf{D}_{diag}\mathbf{L}^T \quad (\text{S4})$$

Forward and backward substitution steps are then performed to solve for accelerations  $\ddot{\mathbf{q}}$ . This method provides robust behavior for stiff robotic mechanisms without requiring excessively small time steps.

#### S5.1.4. Kinematics and Constraints

Forward kinematics are computed recursively from the root to leaves, updating the Cartesian position  $\mathbf{x}$  and orientation  $\mathbf{R}$  of each link. Constraint forces  $\boldsymbol{\tau}_c$  arising from collisions and joint limits are solved using an impulse-based approach or Projected Gauss-Seidel (PGS) solver, integrated tightly with the forward dynamics loop.

### S5.2. Material Point Method (MPM)

The Material Point Method (MPM) solver in our unified kernel serves as the primary engine for simulating continuum materials, including elastoplastic solids, granular media, and fluids. We implement the Moving Least Squares MPM (MLS-MPM) formulation combined with the Affine Particle-in-Cell (APIC) transfer scheme to ensure energy conservation and reduce numerical dissipation.

#### S5.2.1. Discretization and State Variables

The continuum body is discretized into a set of Lagrangian particles  $p$ , each carrying mass  $m_p$ , position  $\mathbf{x}_p$ , velocity  $\mathbf{v}_p$ , deformation gradient  $\mathbf{F}_p$ , and an affine velocity field matrix  $\mathbf{C}_p$ . A background Eulerian Cartesian grid with cell size  $\Delta x$  is employed as a scratchpad for computing spatial gradients and solving the equations of motion.

#### S5.2.2. Particle-to-Grid Transfer (P2G)

At the beginning of each time substep  $\Delta t$ , mass and momentum are transferred from particles to grid nodes  $i$  using quadratic B-spline shape functions  $N_i(\mathbf{x}_p)$ . Under the APIC formulation, the momentum transfer includes the affine velocity contribution to preserve angular momentum:

$$m_i = \sum_p N_i(\mathbf{x}_p) m_p \quad (\text{S5})$$

$$(m\mathbf{v})_i = \sum_p N_i(\mathbf{x}_p) m_p [\mathbf{v}_p + \mathbf{C}_p(\mathbf{x}_i - \mathbf{x}_p)] \quad (\text{S6})$$

In our MLS-MPM implementation, the internal forces derived from the material constitutive model are applied directly during this transfer via the affine term. Let  $\Psi(\mathbf{F})$  be the elastic energy density. The first Piola-Kirchhoff stress  $\mathbf{P}(\mathbf{F}) = \partial\Psi/\partial\mathbf{F}$  is computed and fused into the momentum update to avoid explicit grid force accumulation:

$$(\mathbf{v}_i)^{new} = \frac{(m\mathbf{v})_i}{m_i} + \Delta t \mathbf{g} - \frac{\Delta t}{m_i} \sum_p V_p \mathbf{P}(\mathbf{F}_p)^T \nabla N_i(\mathbf{x}_p) \quad (\text{S7})$$

where  $V_p$  is the particle volume and  $\mathbf{g}$  is the gravity vector.

#### S5.2.3. Constitutive Model and Plasticity

To handle finite deformations and plasticity, we adopt a Singular Value Decomposition (SVD) based return-mapping

algorithm. The trial deformation gradient is updated via the velocity gradient:

$$\mathbf{F}_p^{trial} = (\mathbf{I} + \Delta t \mathbf{C}_p) \mathbf{F}_p^n \quad (\text{S8})$$

We decompose  $\mathbf{F}_p^{trial} = \mathbf{U}_p \boldsymbol{\Sigma}_p \mathbf{V}_p^T$ . Plasticity is applied by clamping the singular values  $\boldsymbol{\Sigma}_p$  based on the specific material model (e.g., Drucker-Prager for sand, Von Mises for metal, or Fixed Corotated for elastic tissues), yielding the projected deformation gradient  $\mathbf{F}_p^{n+1}$ .

#### S5.2.4. Grid-to-Particle Transfer (G2P)

After solving the grid velocities, the updated states are transferred back to the particles. The particle velocity and the affine velocity field are updated as follows:

$$\mathbf{v}_p^{n+1} = \sum_i N_i(\mathbf{x}_p) \mathbf{v}_i^{new} \quad (\text{S9})$$

$$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2} \sum_i N_i(\mathbf{x}_p) \mathbf{v}_i^{new} (\mathbf{x}_i - \mathbf{x}_p)^T \quad (\text{S10})$$

Finally, particle positions are advected:  $\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1}$ .

### S5.3. Position-Based Dynamics (PBD)

The Position-Based Dynamics (PBD) solver is employed to simulate deformable objects such as cloth, soft bodies, and Lagrangian fluids. Our implementation adheres to the Extended Position-Based Dynamics (XPBD) formulation, which introduces a compliance parameter to decouple material stiffness from time step size and iteration counts.

#### S5.3.1. Simulation Loop

The solver follows a predictor-corrector scheme. For each particle  $i$ , a tentative position  $\mathbf{x}_i^*$  is first predicted using explicit Euler integration based on external forces  $\mathbf{f}_{ext}$  (e.g., gravity):

$$\mathbf{v}_i^* = \mathbf{v}_i^n + \Delta t \mathbf{M}^{-1} \mathbf{f}_{ext}, \quad \mathbf{x}_i^* = \mathbf{x}_i^n + \Delta t \mathbf{v}_i^* \quad (\text{S11})$$

Subsequently, the solver iteratively resolves a set of geometric constraints  $C(\mathbf{x}) = 0$ . The position correction  $\Delta \mathbf{x}$  for a constraint involving a set of particles is derived by minimizing the constraint potential energy:

$$\Delta \lambda = \frac{-C(\mathbf{x}^*) - \tilde{\alpha} \lambda}{\sum_j w_j |\nabla_j C(\mathbf{x}^*)|^2 + \tilde{\alpha}} \quad (\text{S12})$$

$$\Delta \mathbf{x}_i = w_i \Delta \lambda \nabla_i C(\mathbf{x}^*) \quad (\text{S13})$$

where  $w_i = 1/m_i$  is the inverse mass, and  $\tilde{\alpha} = \alpha/\Delta t^2$  is the discrete compliance derived from the physical compliance  $\alpha$ . Finally, velocities are updated:  $\mathbf{v}_i^{n+1} = (\mathbf{x}_i^{n+1} - \mathbf{x}_i^n)/\Delta t$ .

### S5.3.2. Structural Constraints

We implement specific constraints for different material types:

- **Distance Constraint (Cloth/Links):** To model stretching resistance, we enforce the distance between two particles  $i$  and  $j$ :

$$C_{dist}(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\| - l_0 = 0 \quad (\text{S14})$$

- **Bending Constraint (Cloth):** To simulate bending resistance in triangular meshes, we utilize the isometric bending constraint based on the dihedral angle  $\phi$  between adjacent triangle faces:

$$C_{bend}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) = \text{acos}(\mathbf{n}_1 \cdot \mathbf{n}_2) - \pi = 0 \quad (\text{S15})$$

- **Volume Constraint (Soft Bodies):** For tetrahedral meshes, volume conservation is enforced on each element to model material compressibility:

$$C_{vol}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) = \frac{1}{6}(\mathbf{x}_{21} \times \mathbf{x}_{31}) \cdot \mathbf{x}_{41} - V_0 = 0 \quad (\text{S16})$$

### S5.3.3. Position-Based Fluids (PBF)

For liquid simulation, we adopt the PBF framework which enforces constant density. The density  $\rho_i$  at particle  $i$  is estimated using the Poly6 kernel  $W_{poly6}$ :

$$\rho_i = \sum_j m_j W_{poly6}(\|\mathbf{x}_i - \mathbf{x}_j\|, h) \quad (\text{S17})$$

The density constraint requires  $C_{density}(\mathbf{x}_1, \dots, \mathbf{x}_n) = \frac{\rho_i}{\rho_0} - 1 = 0$ . Gradients are computed using the Spiky kernel  $\nabla W_{spiky}$  to avoid clustering instability.

Viscosity is handled via XSPH (Artificial Viscosity), which smoothes the velocity field before the position update:

$$\mathbf{v}_i^{new} = \mathbf{v}_i + c \sum_j \frac{m_j}{\rho_j} (\mathbf{v}_j - \mathbf{v}_i) W_{poly6}(\|\mathbf{x}_i - \mathbf{x}_j\|, h) \quad (\text{S18})$$

where  $c$  is a tunable viscosity coefficient.

### S5.4. Smoothed Particle Hydrodynamics (SPH)

The SPH solver provides a fully Lagrangian approach for simulating fluid dynamics, capable of handling free-surface flows and complex topological changes. Our implementation supports both Weakly Compressible SPH (WCSPH) for efficiency in dynamic scenarios and Divergence-Free SPH (DFSPH) for enforcing strict incompressibility with larger time steps.

#### S5.4.1. Discretization

Fluid quantities at a particle  $i$  are interpolated from its neighbors  $j$  using a smoothing kernel  $W_{ij} = W(\|\mathbf{x}_i -$

$\mathbf{x}_j\|, h)$  with support radius  $h$ . The density  $\rho_i$  is evaluated via standard summation:

$$\rho_i = \sum_j m_j W_{ij} \quad (\text{S19})$$

The momentum equation governing the fluid motion is given by:

$$\frac{d\mathbf{v}_i}{dt} = \frac{\mathbf{F}_i^{pressure}}{m_i} + \frac{\mathbf{F}_i^{viscosity}}{m_i} + \frac{\mathbf{F}_i^{surface}}{m_i} + \mathbf{g} \quad (\text{S20})$$

#### S5.4.2. Pressure Solvers

We provide two distinct formulations to resolve the pressure gradient term  $\mathbf{F}^{pressure}$ :

**Weakly Compressible SPH (WCSPH)** In this formulation, pressure is explicitly computed from density deviations using the Tait equation of state, allowing for slight compressibility (typically  $< 1\%$ ):

$$p_i = \frac{k\rho_0}{\gamma} \left( \left( \frac{\rho_i}{\rho_0} \right)^\gamma - 1 \right) \quad (\text{S21})$$

where  $k$  is the stiffness constant and  $\gamma = 7$ . The resulting pressure force is symmetric to ensure momentum conservation:

$$\mathbf{F}_i^{pressure} = -m_i \sum_j m_j \left( \frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W_{ij} \quad (\text{S22})$$

**Divergence-Free SPH (DFSPH)** For scenarios requiring stiff incompressibility, we implement DFSPH, which enforces two constraints using iterative pressure Poisson solvers. First, a *divergence-free solver* modifies the velocities to ensure  $\nabla \cdot \mathbf{v} = 0$  before position integration:

$$\frac{D\rho}{Dt} = -\rho(\nabla \cdot \mathbf{v}) = 0 \implies \sum_j m_j (\mathbf{v}_i - \mathbf{v}_j) \cdot \nabla W_{ij} = 0 \quad (\text{S23})$$

Second, a *constant-density solver* corrects position updates to maintain  $\rho = \rho_0$  (or equivalently, corrects predicted density error). This predictor-corrector scheme significantly improves stability under large time steps compared to WCSPH.

#### S5.4.3. Viscosity and Surface Tension

To simulate viscous fluids, we apply a Laplacian-based viscosity force that dampens relative velocities between neighboring particles:

$$\mathbf{F}_i^{viscosity} = \sum_j \frac{m_j}{\rho_j} \mu \frac{(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{x}_{ij}}{\|\mathbf{x}_{ij}\|^2 + \epsilon h^2} \nabla W_{ij} \quad (\text{S24})$$

where  $\mu$  is the dynamic viscosity coefficient. Additionally, surface tension is modeled as a pairwise cohesive force proportional to a tension coefficient  $\gamma_{tension}$ :

$$\mathbf{F}_i^{surface} = -\gamma_{tension} \sum_j m_j (\mathbf{x}_i - \mathbf{x}_j) W(\|\mathbf{x}_i - \mathbf{x}_j\|) \quad (\text{S25})$$

This term mimics molecular attraction, promoting the formation of droplets and minimizing surface area.

## S6. The Coupling Mechanism of Unified Simulation Kernel

The Unified Simulation Kernel manages heterogeneous interactions by abstracting all assets into particles or meshes and utilizing specialized, coupled solvers. To ensure physical plausibility and conservation laws across different integration schemes, the coupling logic is categorized into three distinct mechanisms based on the solver types involved: (1) Signed Distance Field (SDF) based impulse response for Rigid-MPM/SPH interactions, (2) Position-based penetration correction for Rigid-PBD interactions, and (3) Grid-mediated momentum exchange for MPM-SPH/PBD interactions.

### S6.1. SDF-based Impulse Response (Rigid-MPM and Rigid-SPH)

For interactions between rigid bodies and Eulerian-Lagrangian hybrid solvers (MPM) or purely Lagrangian solvers (SPH), a velocity-level impulse method is employed. This approach utilizes the high-resolution SDF of rigid geometries to modulate collision softness and friction.

For a particle  $p$  (either an MPM grid node or an SPH particle) located at  $\mathbf{x}_p$  with velocity  $\mathbf{v}_p$ , we first compute the signed distance  $d(\mathbf{x}_p)$  and the surface normal  $\mathbf{n}$  from the rigid body's geometry. A blending weight  $w$ , determined by a softness parameter  $\epsilon$ , allows for smooth contact handling:

$$w(d) = \min \left( \exp \left( -\frac{d}{\max(\eta, \epsilon)} \right), 1 \right) \quad (\text{S26})$$

where  $\eta$  is a small numerical stabilizer.

The relative velocity with respect to the rigid body velocity  $\mathbf{v}_r$  is defined as  $\mathbf{v}_{rel} = \mathbf{v}_p - \mathbf{v}_r$ . We decompose  $\mathbf{v}_{rel}$  into normal component  $v_n = \mathbf{v}_{rel} \cdot \mathbf{n}$  and tangential vector  $\mathbf{v}_t = \mathbf{v}_{rel} - v_n \mathbf{n}$ .

If  $v_n < 0$  (penetration velocity), the post-collision relative velocity is computed by applying restitution  $e$  and Coulomb friction  $\mu$ . The normal impulse response is  $\tilde{v}_n = -ev_n$ . The tangential response uses a friction cone constraint:

$$\tilde{\mathbf{v}}_t = \frac{\mathbf{v}_t}{\|\mathbf{v}_t\|} \max(0, \|\mathbf{v}_t\| - \mu|v_n|) \quad (\text{S27})$$

The final updated velocity of the particle is a blend of the rigid body velocity and the new relative velocity, weighted

by  $w$ :

$$\mathbf{v}_p^{new} = \mathbf{v}_r + w(\tilde{\mathbf{v}}_t + \tilde{v}_n \mathbf{n}) + (1 - w)\mathbf{v}_{rel} \quad (\text{S28})$$

Momentum conservation is strictly enforced by applying the reverse impulse  $-\Delta\mathbf{P}/\Delta t$  as an external force to the rigid body solver.

### S6.2. Position-based Penetration Correction (Rigid-PBD)

Interactions between rigid bodies and Position-Based Dynamics (PBD) particles rely on geometric projection rather than velocity impulses. This method prioritizes resolving interpenetration immediately at the position level.

For a PBD particle with mass  $m_p$  and radius  $r$ , collision is detected when the signed distance  $d(\mathbf{x}_p) < r$ . The particle position is projected outward along the contact normal  $\mathbf{n}$  to resolve the overlap:

$$\mathbf{x}_p^{new} = \mathbf{x}_p^{old} + k(r - d(\mathbf{x}_p))\mathbf{n} \quad (\text{S29})$$

where  $k \in [0, 1]$  represents the stiffness coefficient. Unlike the impulse-based method, friction is explicitly neglected in this formulation to maintain stability in the position projection step.

The velocity is implicitly updated via the positional change:  $\mathbf{v}_p^{new} = (\mathbf{x}_p^{new} - \mathbf{x}_p^{old})/\Delta t$ . The resulting momentum change  $\Delta\mathbf{P} = m_p(\mathbf{v}_p^{new} - \mathbf{v}_p^{old})$  is accumulated and applied as a reaction force on the coupled rigid body to satisfy Newton's third law.

### S6.3. Grid-mediated Momentum Exchange (MPM-SPH and MPM-PBD)

Coupling between the Material Point Method (MPM) and Lagrangian particles (SPH or PBD) is achieved via the background MPM Eulerian grid, which acts as a momentum exchange medium. This mechanism models a perfectly inelastic collision process within each time substep.

For a given MPM grid node  $g$ , we identify the set of neighboring external particles  $\mathcal{N}_g$  (SPH or PBD) within the local stencil. The algorithm forces a velocity synchronization where the grid velocity  $\mathbf{v}_g$  is overridden by the average velocity of the interacting particles:

$$\mathbf{v}_g^{sync} = \frac{1}{|\mathcal{N}_g|} \sum_{i \in \mathcal{N}_g} \mathbf{v}_{p_i} \quad (\text{S30})$$

This operation implies an instantaneous "sticking" condition. To conserve total momentum, the change in the grid's momentum  $\Delta\mathbf{P}_g = m_g(\mathbf{v}_g^{sync} - \mathbf{v}_g^{old})$  is calculated. This momentum difference is then distributed back to the external particles as a corrective feedback:

$$\mathbf{v}_{p_i}^{new} = \mathbf{v}_{p_i} - \frac{\Delta\mathbf{P}_g}{m_{p_i}} \quad (\text{S31})$$

This two-way transfer ensures that the MPM fluid effectively drags or is dragged by the immersed SPH/PBD particles, behaving similarly to a mixture model with high drag coefficients.