PhysGaia: A Physics-aware Benchmark with Multi-Body Interactions for Dynamic Novel View Synthesis
–Supplementary document–

A. Preliminary: 4D Gaussian Splatting

A.1. Gaussian primitive

3D Gaussian Splatting has recently achieved real-time rendering with state-of-the-art quality on static scenes. It adopts an explicit representation of the 3D scene through a collection of Gaussian ellipsoids $\Gamma = \{\gamma_1, ..., \gamma_K\}$. Each primitive γ_k is defined by an unnormalized 3D Gaussian kernel $\mathcal{G}_k(\boldsymbol{x})$, parameterized by its mean $\boldsymbol{\mu}_k$ and covariance $\boldsymbol{\Sigma}_k$:

$$\mathcal{G}_k(\boldsymbol{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \coloneqq \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k)\right), \tag{A}$$

where $\boldsymbol{\mu}_k \in \mathbb{R}^3$ denotes the primitive's center position, $\boldsymbol{\Sigma}_k \in \mathbb{R}^{3\times3}$ is an anisotropic covariance matrix, and $\boldsymbol{x} \in \mathbb{R}^3$ represents an arbitrary spatial coordinate. Since $\boldsymbol{\Sigma}_k$ must remain positive semi-definite during optimization—a constraint that is difficult to enforce directly—it is instead factorized into a rotation matrix $\boldsymbol{R}_k$ and a scaling matrix $\boldsymbol{S}_k$:

$$\boldsymbol{\Sigma}_k \coloneqq \boldsymbol{R}_k \boldsymbol{S}_k \boldsymbol{S}_k^\top \boldsymbol{R}_k^\top. \tag{B}$$

Beyond these Gaussian parameters $(\boldsymbol{\mu}_k, \boldsymbol{R}_k, \boldsymbol{S}_k)$, each primitive also requires an opacity value $\alpha_k \in [0, 1]$ and a feature vector $\boldsymbol{f}_k \in \mathbb{R}^d$, which typically encodes RGB colors or spherical harmonic (SH) coefficients. Thus, a Gaussian primitive is fully specified as $\gamma_k \coloneqq (\boldsymbol{\mu}_k, \boldsymbol{R}_k, \boldsymbol{S}_k, \alpha_k, \boldsymbol{f}_k)$.

A.2. Differentiable rasterization

Before rendering the Gaussian primitives Γ into the image space, each 3D Gaussian kernel, $\mathcal{G}_k(\boldsymbol{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, is first projected onto the 2D image space, resulting in a 2D Gaussian kernel, $\mathcal{G}_k^\pi(\boldsymbol{r}; \boldsymbol{\mu}_k^\pi, \boldsymbol{\Sigma}_k^\pi)$. Here, $\pi : \mathbb{R}^3 \to \mathbb{R}^2$ denotes the projection from world coordinates to image space. In the projected Gaussian representation, $\boldsymbol{r} \in \mathbb{R}^2$ indicates a pixel location in an image, and the 2D mean $\boldsymbol{\mu}_k^\pi \in \mathbb{R}^2$ and covariance $\boldsymbol{\Sigma}_k^\pi \in \mathbb{R}^{2\times2}$ are given by

$$\boldsymbol{\mu}_k^\pi \coloneqq \pi(\boldsymbol{\mu}_k) \qquad \text{and} \qquad \boldsymbol{\Sigma}_k^\pi \coloneqq \boldsymbol{J}\boldsymbol{W}\boldsymbol{\Sigma}_k\boldsymbol{W}^\top\boldsymbol{J}^\top, \tag{C}$$

where $\boldsymbol{J}$ is the Jacobian of the affine approximation of the projective transformation and $\boldsymbol{W}$ is the world-to-camera transformation matrix. When rendering the primitives in Γ from a target camera view, they are sorted according to their depth relative to the camera center. The color of a pixel $\boldsymbol{r}$ is then determined by α-blending:

$$\hat{\mathbf{I}}(\boldsymbol{r}) \coloneqq \sum_{k=1}^{K} \omega_k^\pi(\boldsymbol{r}) c(\boldsymbol{f}_k, \boldsymbol{r}), \tag{D}$$

where $\omega_k^\pi(\boldsymbol{r})$ denotes the contribution of primitive γ_k to pixel $\boldsymbol{r}$, and $c(\boldsymbol{f}_k, \boldsymbol{r})$ is its corresponding color. If $\boldsymbol{f}_k$ encodes spherical harmonics coefficients, the color is evaluated using the view direction associated with $\boldsymbol{r}$; otherwise, $\boldsymbol{f}_k$ can directly represent the primitive's RGB values. Following the α-blending formulation in 3DGS [5], the visibility weight is defined as

$$\omega_k^\pi(\boldsymbol{r}) \coloneqq \alpha_k \mathcal{G}_k^\pi(\boldsymbol{r}; \boldsymbol{\mu}_k^\pi, \boldsymbol{\Sigma}_k^\pi) \prod_{j=1}^{k-1} \left(1 - \alpha_j \mathcal{G}_j^\pi(\boldsymbol{r}; \boldsymbol{\mu}_j^\pi, \boldsymbol{\Sigma}_j^\pi)\right), \tag{E}$$

where $\alpha_k \mathcal{G}_k^\pi(\boldsymbol{r}; \boldsymbol{\mu}_k^\pi, \boldsymbol{\Sigma}_k^\pi)$ represents the opacity of the k^{th} projected primitive at pixel $\boldsymbol{r}$, and the product term encodes the transmittance, i.e., how much light passes through the preceding primitives along the ray. Further details can be found in the original Gaussian Splatting paper [5].

A.3. Deformation for Dynamic Modelings

To represent 4D scenes with Gaussian Splatting, recent methods [4, 6, 7, 14–16] learn time-dependent deformations that map canonical 3D Gaussian primitives to their posed states over time. At time t, the position, rotation, and scale of the k-th primitive are updated as

$$(\boldsymbol{\mu}_k^t, \boldsymbol{R}_k^t, \boldsymbol{S}_k^t) = \big(\boldsymbol{\mu}_k + \phi_\mu(\cdot, t),\ \phi_r(\cdot, t)\, \boldsymbol{R}_k,\ \boldsymbol{S}_k + \phi_s(\cdot, t)\big), \tag{F}$$

where the deformation functions $\phi_\mu(\cdot, t)$, $\phi_r(\cdot, t)$, and $\phi_s(\cdot, t)$ denote learnable fields that may take as input various combinations of canonical primitive attributes, spatial coordinates, or time embeddings, depending on the parameterization. The rotation update $\phi_r(\cdot, t)$ is implemented as a residual rotation, typically parameterized via an exponential map to ensure $\phi_r(\cdot, t) \in SO(3)$. These functions are commonly instantiated with MLPs, learnable control points [4, 6, 14], HexPlane [15], or low-order polynomials [7]. The resulting time-varying primitive is then given by $\gamma_k(t) \coloneqq (\boldsymbol{\mu}_k^t, \boldsymbol{R}_k^t, \boldsymbol{S}_k^t, \alpha_k, \boldsymbol{f}_k)$.

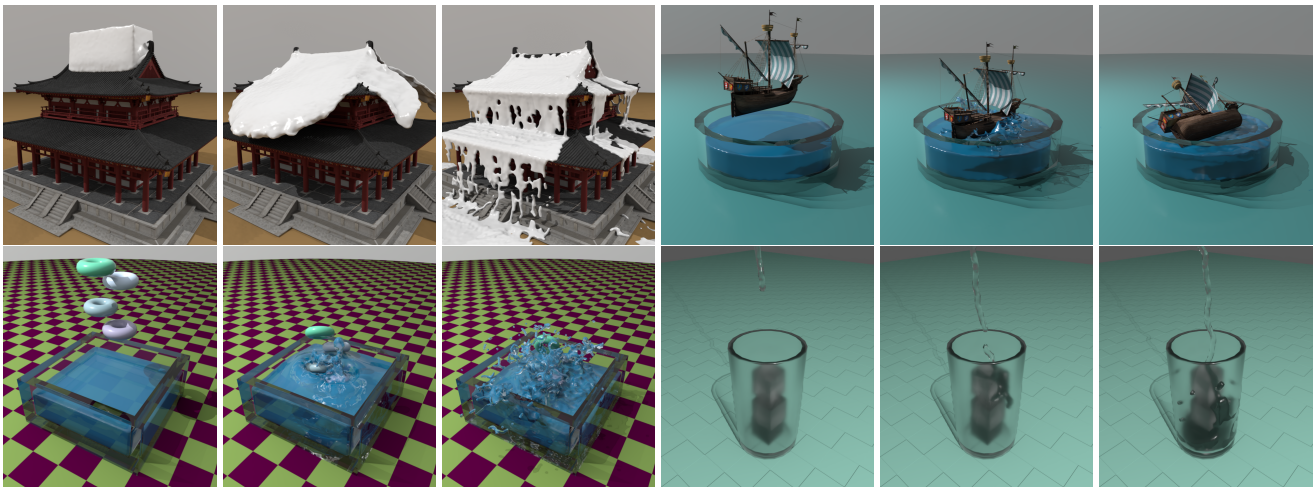
B. Scene Composition

Our benchmark consists of 17 scenes, categorized into four material types: liquids, gases, rheological materials, and textiles. Each category contains 4 to 5 scenes, as listed in Table A. This section details the solver properties for each material and provides key simulation details regarding multi-body interactions. Please refer to our supplemental document for other simulation configurations.

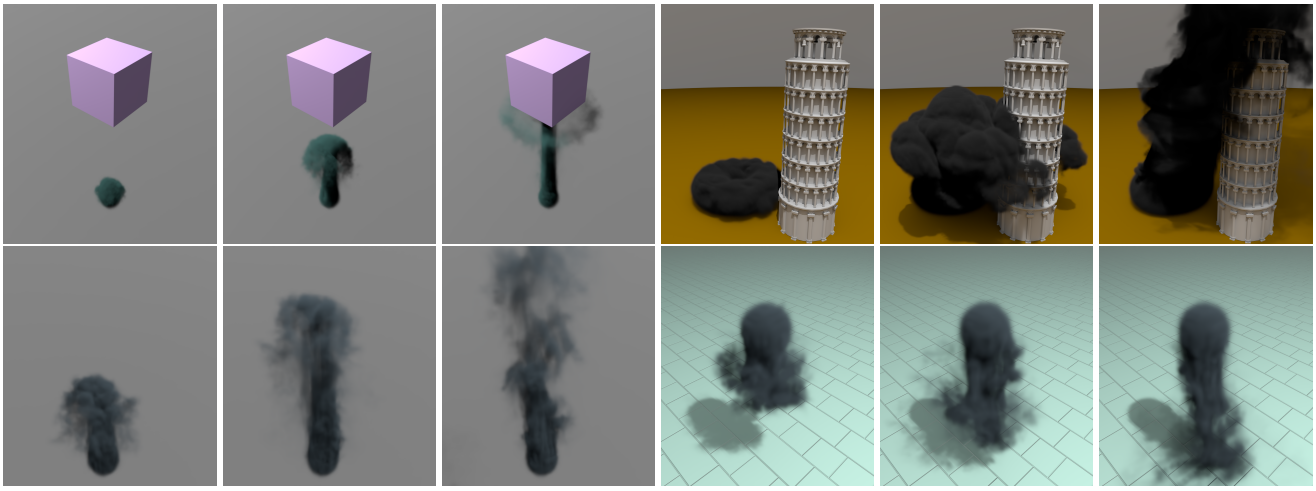
Table A. List of scenes included in our PhysGaia benchmark. Our benchmark consists of 17 scenes, categorized into four material types: liquids, gases, rheological materials, and textiles.

	Liquid	**Gas**	**Rheological substance**	**Textile**
Scene name	Cereal	Pisa	Jelly party	Lucy
	Ship	Box-smoke	Pancake	Basin
	Hanok	Single smoke	Bouncing balls	Flags
	Ice	Falling	Cow	Single flag
	–	–	–	Tube

Figure A and Figure B visualize more scenes in PhysGaia to supplement the Figure 3 in our main paper. As shown in Table 1 and Table 2 of the main paper, textile materials remain underexplored in physics-based datasets. To facilitate progressive research development in this domain, we include few simpler scenes such as *single flag* and *tube* that employ with only basic wind interactions. We believe this approach effectively supports advancements by providing more accessible starting points. Similarly, our benchmark also includes few simpler gas scenes, *simple smoke* and *falling*, for the same reason as with textile scenarios: to facilitate progressive development. For the *cow* scene, internal forces intrinsic to the cow itself introduce unique and more complex dynamics, rather than using only gravity.

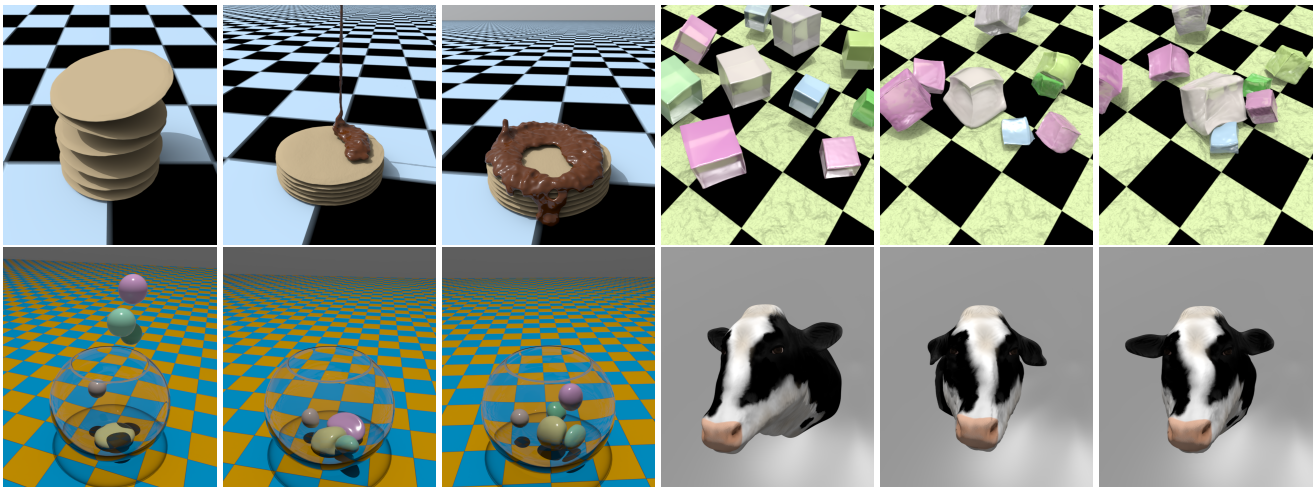


(a) Liquid: *hanok* (top-left), *ship* (top-right), *cereal* (bottom-left), *ice* (bottom-right)

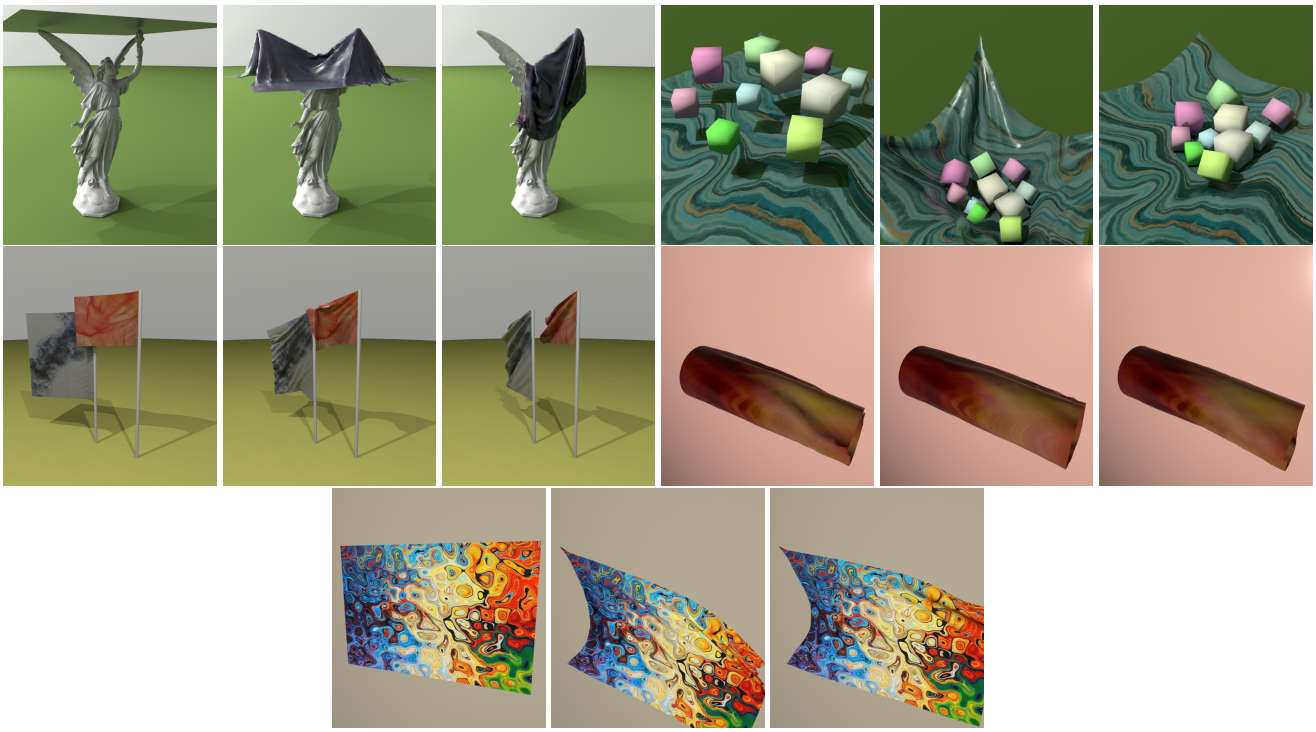


(b) Gas: *box-smoke* (top-left), *pisa* (top-right)), *simple smoke* (bottom-left), and *falling* (bottom-right)

Figure A. Examples from the proposed physics-aware dataset, PhysGaia. They exhibit complex physical interactions between multiple bodies composed of diverse materials such as liquid, gas, viscoelastic substance, and textile. This dataset will foster physics reasoning in dynamic scenes.



(a) Rheological substances: *pancake* (top-left), *jelly party* (top-right), *bouncing balls* (bottom-left), and *cow* (bottom-right)



(b) Textile: *lucy* (top-left), *basin* (top-right), *flags* (mid-left), *tube* (mid-right), and *single flag* (bottom)

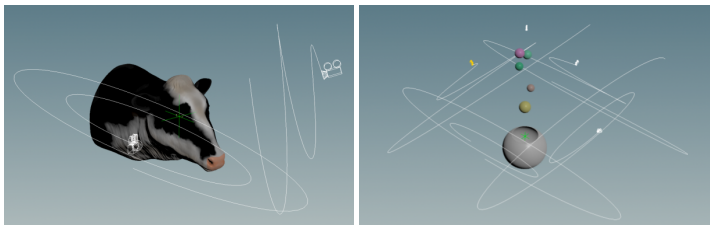
Figure B. Examples from the proposed physics-aware benchmark, PhysGaia. They exhibit complex physical interactions between multiple bodies composed of diverse materials such as liquid, gas, rheological substance, and textile. This benchmark will foster physics reasoning in dynamic scenes.

C. Common Setup

Simulator selection We select SideFX Houdini 20.5 as the foundation of our physics-informed data-generation pipeline because it integrates multiple physics solvers within a unified procedural environment. By sharing a common computational graph, it ensures consistent multi-body interactions under uniform boundary conditions. We access simulation data, such as particle positions and flow fields, on a per-frame basis via its Python API.

Rendering All frames are rendered at a resolution of 640×720. We use path tracing with 256 samples per pixel and apply NVIDIA OptiX for denoising. Scenes are illuminated with 1–3 point lights (intensity: 600–4000); shadows are disabled in textile-focused scenes like *tube-flag* to emphasize geometry. All cameras follow spiral trajectories to capture diverse views over time. All simulations were conducted on 1 NVIDIA Titan Xp GPUs and Intel Xeon E5-2630 v4 CPUs with 40 cores. While most scenes rendered in 2–3 hours, the *lucy* and *hanok* scenes required 12–24 hours due to their complex geometry and increased simulation costs.

Camera Trajectories Figure C visualizes our training camera trajectories, where we adopt either 2 or 4 cameras depending on the properties of each scene. For the *cow*, *single flag*, and *tube* scenes–which resemble 180° settings with background walls placed behind the objects–we use only 2 cameras, as shown in Figure Ca. In contrast, for 360° scenes, we place 4 cameras at evenly spaced viewpoints around the scene, enabling full coverage from four directions. All cameras follow spiral trajectories to capture diverse views over time. For the monocular setup, we select one of the training cameras to generate the training data.



(a) 180° scene: 2 cams (b) 360° scene: 4 cams

Figure C. Visualization of training camera's traj. All cameras follow spiral trajectories to capture diverse views over time. For the monocular setup, we select one of the training cameras to generate the training data.

D. Material-specific Physics Solver

D.1. Liquid

For liquid scenes, we adopt the Fluid-Implicit Particle (FLIP) solver [1] a hybrid particle-grid method. FLIP maintains particle velocities throughout the simulation, using the grid solely to compute and apply forces such as pressure and viscosity. This approach preserves fine-scale, high-frequency particle velocities, which is crucial for modeling realistic and rapid liquid behavior while adhering to the Navier–Stokes equations. While the Material Point Method (MPM) [13] can also model fluids, its direct velocity aggregation onto the grid limits its ability to capture highly dynamic fluid phenomena like splashing. Particle-only solvers [3] are another option, but FLIP is generally better suited for incompressible fluids due thanks to its hybrid grid-based representation.

Dynamic interactions For scenes where fluid spills onto fixed objects, such as *ice* and *hanok*, we employ a surface operator to simulate multi-body interaction. This approach is computationally efficient as interaction primarily occurs near the surface. In contrast, for scenes like *ship* and *cereal* where objects fall into liquid, causing both fluid and objects to move and influence each other through force exchange, we use a dynamic operator to accurately model these more complex interactions.

Physics law We assume incompressible fluids in our simulations. This assumption simplifies the governing equations while maintaining high fidelity for typical liquid behaviors. Fluid motion is governed by the conservation of momentum and mass. We consider gravity as the sole external force. The momentum equation for our fluid, describing the conservation of momentum, is given by

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right) = -\nabla p + \mu\nabla^2\mathbf{u} + \rho\mathbf{g}, \tag{G}$$

where ρ is density, $\mathbf{u}$ is velocity, t is time, p is pressure, μ is dynamic viscosity, and $\mathbf{g}$ is gravitational acceleration. The incompressibility condition ensures the conservation of mass, stating that the fluid's volume remains constant as follows:

$$\nabla \cdot \mathbf{u} = 0, \tag{H}$$

where $\mathbf{u}$ is velocity.

Physics parameters Table B summarizes the density (ρ) and viscosity (μ) values, which are the most important factors governing fluid motion. For the *ship* scene, we reduced the particle separation from 0.1 to 0.05 to decrease initial particle displacement and improve simulation accuracy. Also, the particle separation during dynamic simulation is set to 0.1 for the *cereal* scene and 0.05 for the *ship* scene.

Table B. Physics parameters used for liquid materials.

Scene name	Object	Density (kg/m^3)	Viscosity
Ship	Water	800	0
	Ship	300	-
Cereal	Water	1000	0
	Cereal	1000	-
Hanok	Snow	1000	10
Ice	Water	1000	0

D.2. Gas

For gas (smoke) simulation, we utilize the Pyro solver [11]. Pyro accurately models the temperature field, which is essential for capturing buoyancy effects in gaseous materials. It employs grid-based representations of density, velocity, and temperature, ensuring compliance with the Navier–Stokes equations governing fluid mechanics. Since ground-truth motion is represented as a velocity field and storing full velocity fields can require up to 2GB per frame, we provide a subsampled set of particle trajectories per scene to facilitate efficient data storage and processing.

Dynamic interactions In the *pisa* scene, we reduced the voxel size from the default 0.1 to 0.05 to better capture the tower's intricate details and added a lateral wind of speed 2 to wrap the plume around it. Other scenes involving gaseous materials utilized the default simulation settings.

Physics law Our gas simulations, similar to liquids, are governed by the Navier-Stokes equations. However, we specifically account for the effects of buoyancy as an additional external force due to temperature differences.

$$\rho \left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} \right) = -\nabla p + \mu \nabla^2 \mathbf{u} + \rho(\mathbf{g} + \mathbf{f}_b), \tag{I}$$

where $\mathbf{f}_b$ is the buoyancy force. The buoyancy force is derived from temperature differences relative to the ambient environment as follows:

$$\mathbf{f}_b = k_B \cdot (T - T_{\text{ambient}}) \cdot \mathbf{d}_B, \tag{J}$$

where k_B is the buoyancy constant, T is the local gas temperature, T_{ambient} is the ambient temperature, and $\mathbf{d}_B$ is the buoyancy direction.

Physics parameters In gas simulations, temperature is the most critical physical parameter, as it directly governs buoyancy, expansion, and overall flow dynamics. We initialized the temperature of gas materials to 3000K. As temperature evolves locally due to gas movement and interactions, its spatiotemporal variation is represented as a temperature field defined on a voxel grid. These temperature fields are accessible via the provided simulation source file with a Python API, similar to how flow fields are accessed for generating ground-truth trajectory data.

D.3. Rheological Substances

We use MPM [13] for rheological substances like snow and jelly. As an extension of FLIP [1], MPM is ideal for simulating chunk-based materials. It aggregates particle information onto a grid, performs computations, and then reprojects results back to particles, effectively capturing deformation and internal force propagation.

Dynamic interactions For the *pancake* scene, we reduced the grid size from the default 0.025 to 0.002 to more faithfully capture its thin-sheet dynamics; other scenes use the default grid size. To suppress spurious artifacts that can arise from aggregating particle properties onto the grid, we increased the number of samples participating in node calculations by oversampling. The oversampling scales are set to 6, 2, 4, and 2 for the *bouncing balls*, *cow*, *jelly party*, and *pancake* scenes, respectively. Additionally, for the *bouncing balls* scene, we added a static bowl-shaped collider so that the falling balls rebound off both one another and the bowl's surface.

Physics law Simulating viscoelastic materials requires modeling both their elastic (solid-like) and viscous (fluid-like) responses. Our approach follows fundamental conservation laws while adopting simplified constitutive models commonly used in MPM-based visual simulations, namely linear elasticity and Kelvin–Voigt viscosity. The conservation of mass is described by the continuity equation as follows:

$$\frac{D\rho}{Dt} + \rho(\nabla \cdot \mathbf{v}) = 0, \tag{K}$$

where ρ is the material density and $\mathbf{v}$ is the velocity field. Conservation of linear momentum is given by

$$\rho \frac{D\mathbf{v}}{Dt} = \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{g}, \tag{L}$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{g}$ is gravitational acceleration. We adopt a Kelvin–Voigt-type viscoelastic model, where the total stress is decomposed into elastic and viscous components as follows:

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}_{\text{elastic}} + \boldsymbol{\sigma}_{\text{viscous}}. \tag{M}$$

The elastic stress is modeled using linear isotropic elasticity as follows:

$$\boldsymbol{\sigma}_{\text{elastic}} = \frac{E\nu}{(1+\nu)(1-2\nu)}(\text{tr}(\boldsymbol{\epsilon}))\mathbf{I} + \frac{E}{2(1+\nu)}\boldsymbol{\epsilon}, \tag{N}$$

where E is Young's modulus, ν is Poisson's ratio, and $\boldsymbol{\epsilon}$ is the strain tensor defined as $\boldsymbol{\epsilon} = \frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^T)$, where $\mathbf{u}$ is the displacement field. The viscous stress accounts for rate-dependent deformation as follows:

$$\boldsymbol{\sigma}_{\text{viscous}} = 2\mu \left(\mathbf{E} - \frac{1}{3}(\text{tr}(\mathbf{E}))\mathbf{I} \right) + \zeta(\text{tr}(\mathbf{E}))\mathbf{I}, \tag{O}$$

where μ is the dynamic viscosity and ζ is the bulk viscosity. The strain-rate tensor $\mathbf{E}$ is defined as $\mathbf{E} = \frac{1}{2}(\nabla \mathbf{v} + (\nabla \mathbf{v})^T)$.

Physics parameters Gravity is applied to all scenes; however, for the *cow* scene, we introduce additional internal forces to induce cow motion and increase the complexity of the physical reasoning scenarios. Boundaries are set to be open in all directions, allowing objects to move freely without collisions against invisible walls. Table C lists detailed physics parameters, including Young's modulus (E) and Poisson's ratio (ν), used in the simulations.

Table C. Physics parameters used for rheological materials.

Scene name	Object	Type	E	ν	Viscosity
Pancake	Honey	Viscous	2.5×10^5	0.10	0.125
	Pancakes	Chunky	8.0×10^4	0.23	-
Bouncing balls	Balls	Elastic	1.0×10^5	0.50	-
	Fishbowl	Static	-	-	-
Jelly party	Jelly	Elastic	8.0×10^4	0.45	-
Cow	Cow	Elastic	5.0×10^4	0.45	-

D.4. Textile

For textile materials, we adopt the Vellum solver [12], which is based on the Extended Position Based Dynamics (XPBD) framework [8]. XPBD improves upon classical Position Based Dynamics (PBD) [9] by integrating a Lagrange multiplier and its update. This effectively decouples material stiffness from the solver's time-step size and iteration count, making Vellum a widely used method for simulating deformable objects, especially cloth.

Algorithm A Extended Position-Based Dynamics (XPBD) Algorithm (per substep)

Input:
  $\mathbf{p}_i$, $\mathbf{v}_i$: Current position and velocity of particle i
  $\mathbf{w}_i = 1/m_i$: Inverse mass of particle i
  $\mathbf{a}_{\text{ext},i}$: External acceleration (e.g., gravity)
  Δt: Simulation time step
  Iterations: Number of constraint projection iterations
// 1. Predict positions
for each particle i **do**
  $\mathbf{p}_i^* \leftarrow \mathbf{p}_i + \Delta t \mathbf{v}_i + \Delta t^2 \mathbf{a}_{\text{ext},i}$
end for
// 2. Constraint projection
for $k = 1$ to Iterations **do**
  for each constraint C_j **do**
    compute $\Delta \mathbf{p}_i$ using Eq. (P)
    $\mathbf{p}_i^* \leftarrow \mathbf{p}_i^* + \Delta \mathbf{p}_i$
  end for
end for
// 3. Velocity update
for each particle i **do**
  $\mathbf{v}_i \leftarrow (\mathbf{p}_i^* - \mathbf{p}_i)/\Delta t$
  $\mathbf{p}_i \leftarrow \mathbf{p}_i^*$
end for

Dynamic interactions To simulate interactions between objects and textiles–where both move and exchange forces, as seen particularly in the *basin* scene–we employ the shape match constraint on objects. This constraint helps maintain the overall shape of rigid objects by driving points toward their rest configuration, allowing the material to preserve its structural integrity while still interacting dynamically with textiles and other objects. For the *lucy* scene, we increased the simulation sub-step count fivefold over the default to robustly handle collisions with the statue's complex geometry. To simulate wind-induced fluttering in the *flags*, *single-flag*, and *tube* scenes, we applied external forces using a POP Wind node that blows parallel to the ground plane.

Physics law We simulate textile materials using the Vellum solver [12], which is based on Extended Position-Based Dynamics (XPBD) [8]. XPBD extends classical Position-Based Dynamics (PBD) by introducing a compliance parameter, allowing constraint stiffness to be decoupled from the time step and iteration count. Unlike force-based methods, XPBD does not explicitly solve equations of motion. Instead, particle positions are updated by iteratively projecting them to satisfy geometric constraints. The core algorithm for each simulation substep is shown in Algorithm A.

We primarily employ stretch and bend constraints to model textile behavior. For a constraint $C(\mathbf{p})$, where $C(\mathbf{p})$ is a scalar constraint function that measures the violation of a geometric constraint, such as maintaining distances (stretch) or bending angles between particles, with $C(\mathbf{p}) = 0$ indicating that the constraint is satisfied, the position correction is as follows:

$$\Delta \mathbf{p}_i = -\Delta\lambda \, \mathbf{w}_i \nabla_{\mathbf{p}_i} C, \tag{P}$$

where

$$\Delta\lambda = \frac{C(\mathbf{p})}{\sum_j \mathbf{w}_j |\nabla_{\mathbf{p}_j} C|^2 + \tilde{\alpha}}, \quad \tilde{\alpha} = \frac{\alpha}{\Delta t^2}. \tag{Q}$$

Here, α denotes the compliance parameter (inverse stiffness).

Physics parameters For textiles, we primarily adopt the default configuration from the Vellum solver, with detailed modifications summarized in Table D. In addition to stiffness, we specify a damping ratio for each constraint to reduce oscillations and stabilize the simulated cloth.

Table D. Physics parameters used for textile materials.

Scene name	Object	Stretch		Bend	
		Stiffness	Damping Ratio	Stiffness	Damping Ratio
Flags	Flag1 (small)	1.0×10^{10}	1.0×10^{-3}	1.0×10^{-4}	1.0×10^{-2}
	Flag2 (big)	1.0×10^{11}	1.0×10^{-3}	1.0×10^{-4}	1.0×10^{-2}
Basin	Boxes	1.0×10^{10}	1.0×10^{-3}	N/A	N/A
	Cloth	1.0×10^{13}	1.0×10^{-3}	1.0×10^{-4}	1.0×10^{-2}
Single flag	Flag	1.0×10^{10}	1.0×10^{-3}	1.0×10^{-4}	1.0×10^{-2}
Lucy	Cloth	1.0×10^{10}	1.0×10^{-3}	1.0×10^{-4}	1.0×10^{-2}
Tube	Tube flag	1.0×10^{10}	1.0×10^{-3}	1.0×10^{-4}	1.0×10^{-2}

E. Details of Experimental Results

E.1. Implementation details

We implement recent DyNVS methods, including D-3DGS [16], 4DGS [15], STG [7], MoSca [6], and SOM [14]. We follow the default training settings provided for each method. For 4DGS, we adopt the training configuration used for the HyperNeRF [10] dataset and reduce the grid learning rate to improve training stability. For SOM, we apply the recommended hyperparameters for the DyCheck [2]dataset. For point cloud initialization, we run COLMAP with dense matching and fusion, followed by uniform downsampling to approximately 40,000 points. All experiments are conducted on NVIDIA RTX A5000 and A6000 GPUs, with training times ranging from 30 minutes to 2 hours depending on the method. Note that all experiments are implemented based on their public codes [1] [2] [3] [4] [5].

E.2. Average performance: all scenes

Table E presents the average performance of all methods under both monocular and multiview settings, aggregated across all 17 scenes. For SOM [14], we observed better performance when scaling the estimated depth map using the COLMAP point cloud obtained from dense matching, compared to sparse matching.

Table E. Average quantitative results for both monocular and multiview settings, averaged across all 17 scenes. While multiview setups generally offer better reconstruction performance than monocular ones, even multiview results achieve PSNR scores below 30. This highlights the substantial difficulty in reconstructing the complex multi-body interactions in our benchmark.

Method	Monocular			Multiview		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
D-3DGS [16]	21.7	0.86	0.18	24.2	0.89	0.14
4DGS [15]	22.7	0.85	0.19	24.4	0.87	0.17
STG [7]	19.3	0.76	0.30	21.0	0.79	0.30
MoSca [6]	19.5	0.76	0.34	N/A	N/A	N/A
SoM [14]	19.3	0.80	0.26	N/A	N/A	N/A

[1] D-3DGS: https://github.com/ingra14m/Deformable-3D-Gaussians
[2] STG: https://github.com/oppo-us-research/SpacetimeGaussians
[3] 4DGS: https://github.com/hustvl/4DGaussians
[4] SOM: https://github.com/vye16/shape-of-motion
[5] MoSca: https://github.com/JiahuiLei/MoSca

E.3. Performance breakdown: monocular setting

We provide per-scene breakdown performance of monocular video reconstruction setting in Table F. We report the performance of D-3DGS [16], 4DGS [15], STG [7], MoSca [6] and SOM [14], which serve as the most common and recent baselines for the DyNVS task.

Table F. Per-scene breakdown results for all 17 scenes under the monocular setting.

Method	Cereal			Ship			Hanok		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
D-3DGS [16]	23.3	0.88	0.15	25.9	0.90	0.16	16.3	0.78	0.28
4DGS [15]	26.3	0.90	0.14	25.9	0.91	0.15	15.5	0.74	0.32
STG [7]	15.8	0.54	0.54	22.8	0.87	0.23	14.8	0.60	0.41
MoSca [6]	23.7	0.83	0.14	22.5	0.84	0.39	14.8	0.59	0.42
SOM [14]	21.2	0.81	0.22	23.8	0.87	0.23	14.7	0.67	0.40

Method	Ice			Pisa			Box-smoke		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
D-3DGS [16]	25.3	0.92	0.30	20.7	0.72	0.26	20.7	0.96	0.13
4DGS [15]	29.2	0.92	0.29	19.3	0.67	0.26	21.3	0.96	0.12
STG [7]	23.1	0.87	0.39	20.1	0.68	0.33	22.7	0.95	0.16
MoSca [6]	20.9	0.86	0.61	16.7	0.54	0.57	22.1	0.94	0.21
SOM [14]	18.7	0.84	0.41	17.1	0.65	0.41	23.8	0.95	0.18

Method	Single smoke			Falling			Jelly party		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
D-3DGS [16]	26.5	0.97	0.08	19.6	0.90	0.18	16.3	0.81	0.21
4DGS [15]	26.1	0.97	0.08	20.1	0.90	0.19	14.9	0.70	0.28
STG [7]	24.5	0.97	0.10	20.2	0.77	0.37	11.1	0.51	0.44
MoSca [6]	24.8	0.95	0.14	21.2	0.72	0.49	12.0	0.51	0.38
SOM [14]	23.9	0.95	0.18	15.3	0.81	0.33	14.9	0.69	0.28

Method	Pancake			Bouncing balls			Cow		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
D-3DGS [16]	22.8	0.88	0.13	19.8	0.77	0.15	21.6	0.92	0.10
4DGS [15]	17.0	0.78	0.22	22.7	0.86	0.10	23.6	0.92	0.11
STG [7]	10.6	0.60	0.39	11.9	0.51	0.60	20.6	0.87	0.16
MoSca [6]	22.2	0.87	0.12	19.9	0.81	0.12	17.1	0.78	0.32
SOM [14]	13.9	0.68	0.34	17.4	0.79	0.14	20.7	0.85	0.17

Method	Lucy			Basin			Flags		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
D-3DGS [16]	22.8	0.92	0.10	18.2	0.67	0.36	23.6	0.94	0.12
4DGS [15]	27.5	0.94	0.08	18.0	0.68	0.38	31.9	0.96	0.08
STG [7]	19.6	0.86	0.19	16.8	0.66	0.43	24.7	0.90	0.17
MoSca [6]	14.8	0.80	0.49	18.4	0.62	0.41	26.0	0.88	0.17
SOM [14]	21.0	0.90	0.17	16.3	0.61	0.43	27.4	0.93	0.12

Method	Single flag			Tube			Average		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
D-3DGS [16]	18.3	0.68	0.24	27.7	0.96	0.05	21.7	0.86	0.18
4DGS [15]	18.0	0.65	0.29	28.8	0.96	0.08	22.7	0.85	0.19
STG [7]	26.0	0.87	0.14	22.1	0.92	0.12	19.3	0.76	0.30
MoSca [6]	15.3	0.56	0.50	18.3	0.86	0.25	19.5	0.76	0.34
SOM [14]	16.3	0.60	0.24	22.5	0.93	0.15	19.3	0.80	0.26

E.4. Performance breakdown: multiview setting

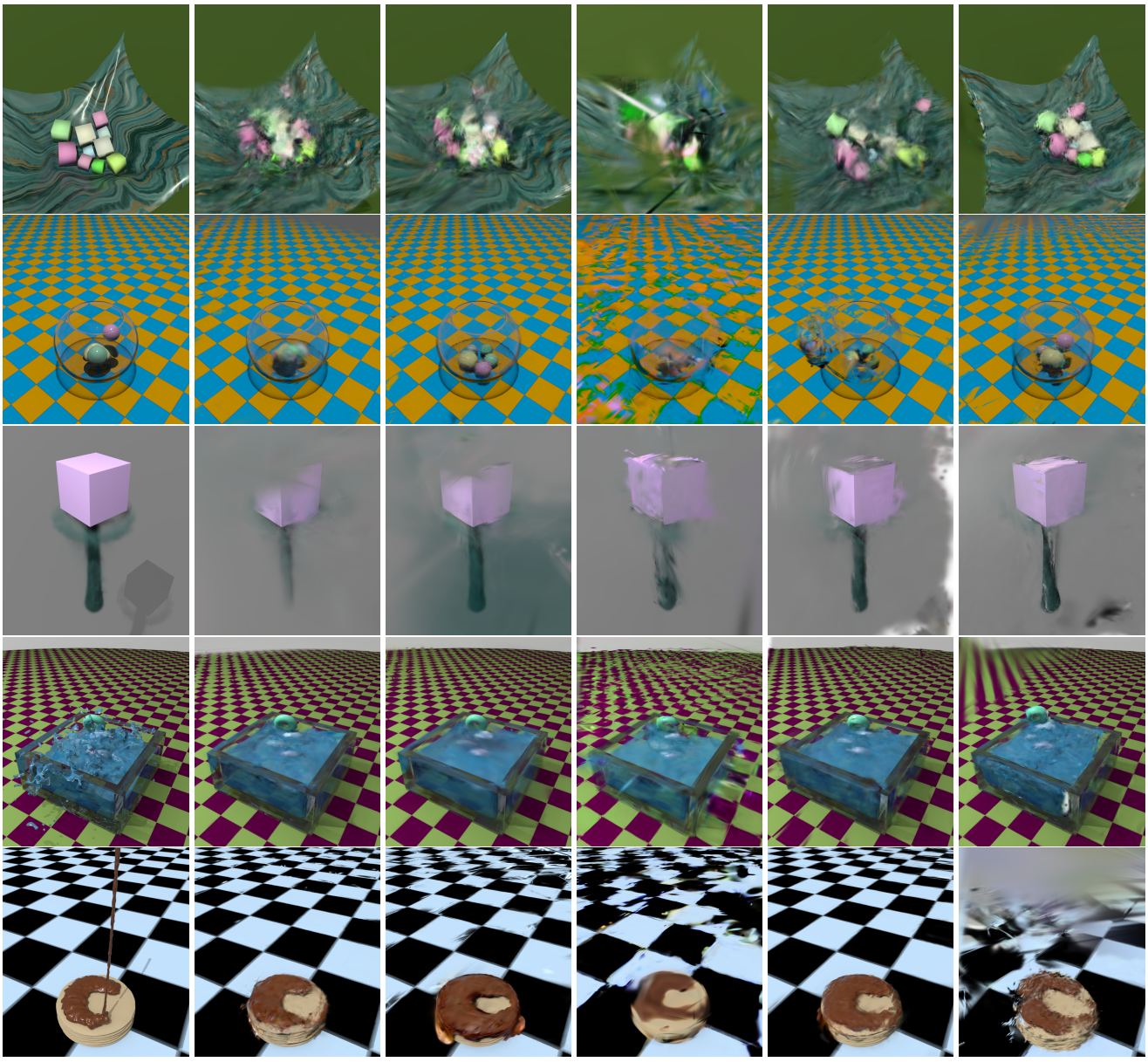
We provide per-scene breakdown performance for the multiview video reconstruction setting in Table G. We report results for D-3DGS [16], 4DGS [15], and STG [7], which serve as the most common and recent baselines for the DyNVS task. Note that since MoSca [6] and SOM [14] are specialized for monocular video setups, we omit its performance in the multiview evaluation.

Table G. Per-scene breakdown results for all 17 scenes under the multiview setting.

Method	Cereal			Ship			Hanok		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
D-3DGS [16]	28.1	0.93	0.11	28.8	0.93	0.11	15.6	0.76	0.31
4DGS [15]	27.6	0.92	0.13	25.6	0.91	0.15	15.8	0.75	0.33
STG [7]	15.8	0.58	0.59	25.9	0.90	0.20	15.7	0.65	0.43
Method	**Ice**			**Pisa**			**Box-smoke**		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
D-3DGS [16]	16.4	0.86	0.43	22.3	0.75	0.23	20.5	0.96	0.11
4DGS [15]	31.4	0.93	0.28	22.8	0.74	0.24	22.6	0.97	0.07
STG [7]	25.9	0.89	0.38	28.7	0.90	0.18	19.0	0.95	0.17
Method	**Single-smoke**			**Falling**			**Jelly party**		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
D-3DGS [16]	26.4	0.98	0.07	25.7	0.95	0.11	18.2	0.87	0.16
4DGS [15]	28.2	0.98	0.07	23.4	0.87	0.30	16.1	0.80	0.24
STG [7]	27.0	0.98	0.07	25.1	0.83	0.34	12.5	0.60	0.39
Method	**Pancake**			**Bouncing balls**			**Cow**		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
D-3DGS [16]	25.3	0.90	0.08	25.7	0.91	0.07	19.8	0.87	0.10
4DGS [15]	18.7	0.84	0.18	25.0	0.84	0.09	24.1	0.92	0.10
STG [7]	11.7	0.65	0.36	12.9	0.59	0.54	31.8	0.95	0.15
Method	**Lucy**			**Basin**			**Flags**		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
D-3DGS [16]	30.2	0.96	0.06	23.6	0.82	0.24	30.5	0.96	0.09
4DGS [15]	28.6	0.95	0.07	20.5	0.75	0.33	32.8	0.96	0.08
STG [7]	21.2	0.88	0.19	17.5	0.70	0.45	26.0	0.91	0.17
Method	**Single flag**			**Tube**			**Average**		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
D-3DGS [16]	21.6	0.81	0.14	32.6	0.98	0.05	24.2	0.89	0.14
4DGS [15]	19.7	0.72	0.21	31.3	0.97	0.07	24.4	0.87	0.17
STG [7]	16.9	0.62	0.36	23.8	0.93	0.11	21.0	0.79	0.30

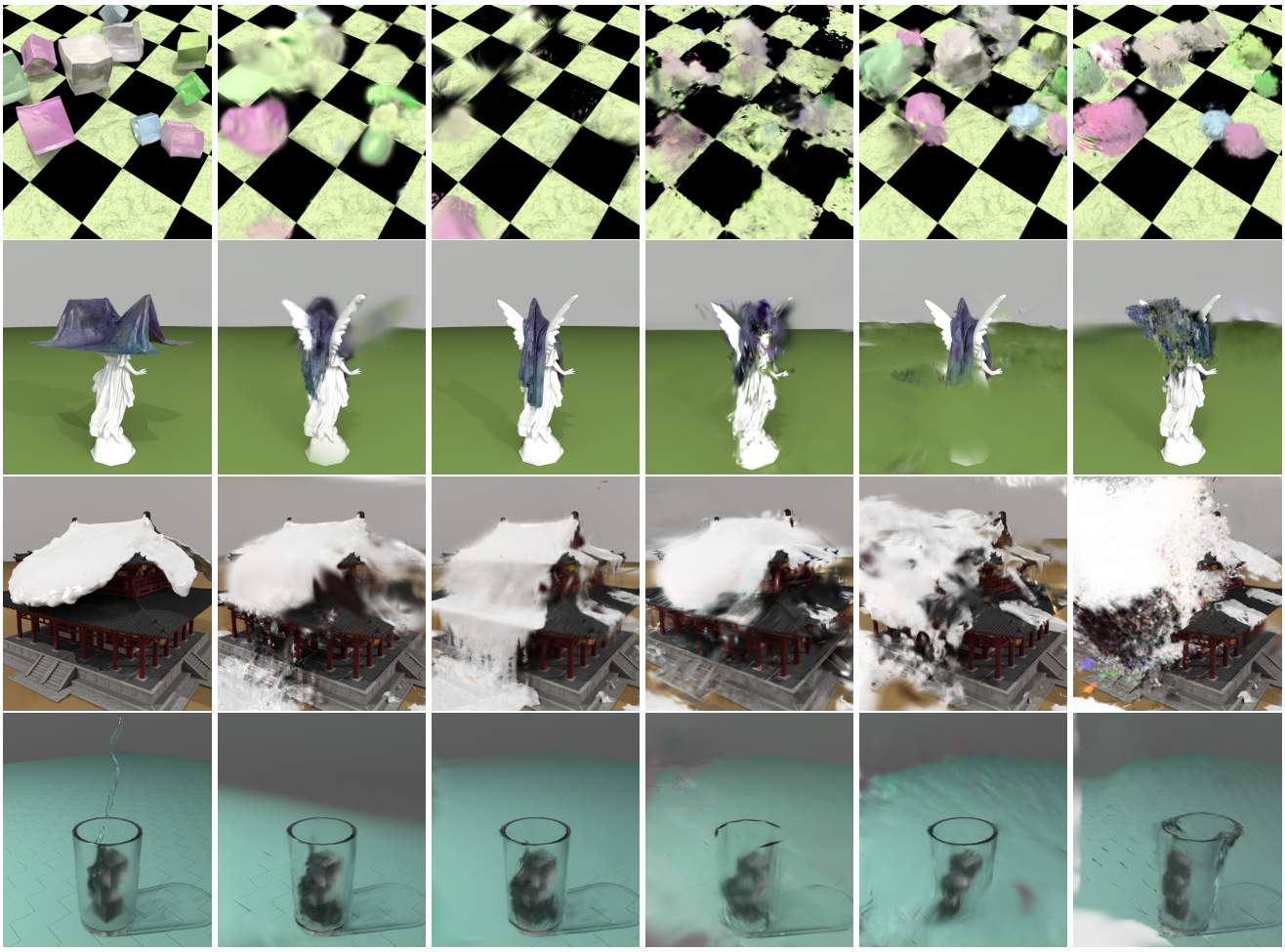
E.5. Additional Qualitative Results

Figure D and Figure E present qualitative results from monocular settings, revealing that all methods struggle to accurately capture multi-body interactions. This leads to common issues like needle-like artifacts and under-reconstruction of dynamic regions. For 4DGS [15], its default grid learning rate often causes NaN loss values; while reducing it stabilizes training, the output resembles static scenes due to poor dynamic capture.



(a) GT image (b) D-3DGS [16] (c) 4DGS [15] (d) STG [7] (e) MoSca [6] (f) SOM [14]

Figure D. Qualitative results of recent DyNVS methods on the *basin*, *bouncing balls*, *box-smoke*, *pancake*, and *ice* scenes with monocular training setup. These results show that all methods frequently exhibit needle-like artifacts and fail to reconstruct dynamic elements accurately.



(a) GT image (b) D-3DGS [16] (c) 4DGS [15] (d) STG [7] (e) MoSca (2024) (f) SOM [14]

Figure E. Qualitative results of recent DyNVS methods on the *lucy*, and *hanok* scenes with monocular setup. All methods struggle to accurately capture multi-body interactions by frequently exhibiting needle-like artifacts and failing to reconstruct dynamic elements accurately.

F. Limitations & Broad Impact

Limitations While our *PhysGaia* is built upon a physics simulator to provide physically consistent supervision, it inherits the inherent approximations of the underlying physical models. In practice, physics simulators rely on simplified constitutive assumptions and numerical discretizations to make the problem tractable and stable. As a result, certain aspects of real-world dynamics, such as complex material responses or fine-scale interactions, may not be fully captured. We view this not as a limitation of our dataset alone, but as an inherent characteristic of simulation-based approaches, and an important direction for future work.

Broad impact From a positive perspective, our work advances research on physically plausible 4D reconstruction. This, in turn, significantly improves monocular video reconstruction, an essential technology for future AR/VR applications. However, such progress in 4D reconstruction, especially given its potential for scene editing of existing videos, may raise intellectual property concerns regarding the original video content.

G. License

Our benchmark is released under the Creative Commons Attribution-NonCommercial (CC BY-NC) license. The mesh used for the *lucy* scene is sourced from the Stanford 3D Repository, which permits usage for research purposes. Other mesh objects are licensed under CC-BY 4.0, as detailed in Table H. For the texture maps in the textile categories, we utilized images from Pixabay, which are freely available with contributor consent and comparable to a CC-BY 4.0 license, also summarized in Table H. Therefore, all data included in our benchmark comply with usage rights and do not pose intellectual property issues.

Table H. License of the sources used for generating data.

Scene	Resource name	License	Access
Ice	Glass cup	CC BY 4.0	Sketchfab
Hanok	Korean building	CC BY 4.0	Sketchfab
Ship	Ship	CC BY 4.0	Sketchfab
Pisa	Torre Pisa	CC BY 4.0	Sketchfab
Bouncing balls	Fish bowl	CC BY 4.0	Sketchfab
Cow	Cow	CC BY 4.0	Sketchfab
Lucy	Lucy	research only	Stanford 3D Scan Repo
Scene	**Contributor**	**License**	**Access**
Flags	PaftDrunk	Content License	Pixabay
	Zoeysmom	Content License	Pixabay
Single flag	–	Content License	Pixabay
Lucy	WalterClark	Content License	Pixabay
Tube	–	Content License	Pixabay
Basin	Yourialka	Content License	Pixabay

References

[1] J. U. Brackbill. Flip: A low-dissipation, particle-in-cell method for fluid flow. *Journal of Computational Physics*, 1986. 5, 6

[2] Hang Gao, Ruilong Li, Shubham Tulsiani, Bryan Russell, and Angjoo Kanazawa. Monocular dynamic view synthesis: A reality check. In *NeurIPS*, 2022. 9

[3] 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. 5

[4] Yi-Hua Huang, Yang-Tian Sun, Ziyi Yang, Xiaoyang Lyu, Yan-Pei Cao, and Xiaojuan Qi. Sc-gs: Sparse-controlled gaussian splatting for editable dynamic scenes. In *CVPR*, 2024. 2

[5] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. In *ACM ToG*, 2023. 1

[6] Jiahui Lei, Yijia Weng, Adam Harley, Leonidas Guibas, and Kostas Daniilidis. Mosca: Dynamic gaussian fusion from casual videos via 4d motion scaffolds, 2024. arXiv preprint arXiv:2405.17421. 2, 9, 10, 11, 12, 13

[7] Zhan Li, Zhang Chen, Zhong Li, and Yi Xu. Spacetime gaussian feature splatting for real-time dynamic view synthesis. In *CVPR*, 2024. 2, 9, 10, 11, 12, 13

[8] Miles Macklin, Matthias Müller, and Nuttapong Chentanez. Xpbd: position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games*, 2016. 7, 8

[9] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 2007. 7

[10] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M. Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *ACM Trans. Graph.*, 2021. 9

[11] SideFX Software. Pyro solver. https://www.sidefx.com/docs/houdini/pyro/intro.html, 2012. 6

[12] SideFX Software. Vellum solver. https://www.sidefx.com/docs/houdini/vellum/overview.html, 2017. 7, 8

[13] Deborah Sulsky, Zhen Chen, and Howard L. Schreyer. A particle method for history-dependent materials. *Computer Methods in Applied Mechanics and Engineering*, 1994. 5, 6

[14] Qianqian Wang, Vickie Ye, Hang Gao, Jake Austin, Zhengqi Li, and Angjoo Kanazawa. Shape of motion: 4d reconstruction from a single video. In *ICCV*, 2025. 2, 9, 10, 11, 12, 13

[15] 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. 2, 9, 10, 11, 12, 13

[16] Ziyi Yang, Xinyu Gao, Wen Zhou, Shaohui Jiao, Yuqing Zhang, and Xiaogang Jin. Deformable 3d gaussians for high-fidelity monocular dynamic scene reconstruction. In *CVPR*, 2024. 2, 9, 10, 11, 12, 13