PhysInOne: Visual Physics Learning and Reasoning in One Suite

Supplementary Material

5.1. More Details of Physical Phenomena and Laws

We identify 71 basic physical phenomena commonly observed in everyday life. Each phenomenon involves one or more underlying physical laws, encompassing areas such as mechanics, optics, fluid dynamics, and magnetism. Table 8 lists these 71 phenomena along with the corresponding physical principles involved.

Table 8. The list of Physical Phenomena and Related Physical Laws

Index	Physical Phenomena	Physical Laws
1	Object collide with static, stationary objects	Laws of Momentum
2	Moving objects collide with non-static stationary objects	Laws of Momentum
3	Two moving objects collide	Laws of Momentum
4	Objects in equilibrium of wind and gravity	Equilibrium, Aerodynamics, Gravity
5	Wind applied to a stationary object	Aerodynamics
6	Wind applied to objects moving in same direction	Aerodynamics
7	Wind applied to objects moving in the opposite direction	Aerodynamics
8	Wind applied to moving objects changes its velocity (applied at an angle)	Aerodynamics
9	Object thrown up with angle	Gravity
10	Objects falling straight down	Gravity
11	Objects rolling down a straight slope	Gravity, Friction
12	Objects rolling up a slope	Gravity, Friction
13	Magnetic Attraction	Magnetism
14	Magnetic Repulsion	Magnetism
15	Objects near uniformly rotating pillar	Laws of Rotation
16	Objects near acceleratingly rotating pillar	Laws of Rotation
17	Objects inside uniformly rotating bowl	Equilibrium, Laws of Rotation, Gravity, Friction
18	Objects inside acceleratingly rotating bowl	Equilibrium, Laws of Rotation, Gravity, Friction
19	Objects on uniformly rotating plane	Laws of Rotation, Friction
20	Objects on acceleratingly rotating plane	Laws of Rotation, Friction
21	Objects on coarse surface with friction	Friction
22	Spring is compressed	Laws of Elasticity
23	Spring is stretched	Laws of Elasticity
24	Breakable object shatters	Laws of Plasticity
25	Mirror shatters	Laws of Plasticity, Law of Reflection
26	Elastic rope connection	Laws of Elasticity
27	Object bounces off spring board	Laws of Elasticity, Laws of Momentum
28	Objects interact with a balanced seesaw	Laws of Torque
29	Objects interact with an imbalanced seesaw	Laws of Torque
30	Free balloon floats to ceiling	Laws of Buoyancy
31	Tethered balloon pulls string taut	Laws of Buoyancy, Rope Restraint
32	Multiple balloons lifting	Laws of Buoyancy, Rope Restraint, Gravity
33	Laser hits flat mirror and reflects	Law of Reflection
34	Laser reflects off multiple mirrors	Law of Reflection
35	Laser hits and reflects off concave mirror	Law of Reflection
36	Laser hits and reflects off convex mirror	Law of Reflection
37	Mirror sweeps beam	Law of Reflection
38	Laser blocked by object	Light Obstruction

Index	Physical Phenomena	Physical Laws
39	Mirror reflection	Law of Reflection
40	Cart moving forward with rolling wheels	Complex Mechanical Structure Constraints
41	Objects on rotating turntable flies off	Laws of Rotation, Friction, Laws of Inertia
42	Rotating block pushes another objects	Laws of Inertia, Laws of Rotation, Laws of Momentum
43	One object carrying another	Laws of Inertia, Friction
44	Catapult launches objects	Special Mechanical Structure, Gravity
45	Chain suspends objects	Complex Mechanical Structure Constraints, Gravity
46	Objects Swing	Laws of Pendulum Motion, Gravity
47	Double Pendulum Moves	Laws of Multiple Pendulum Motion, Gravity
48	Crank push objects	Special Mechanical Structure
49	Wall composed of square blocks collapses	Gravity, Structural Stability
50	Wooden board supported by sticks collapses	Gravity, Structural Stability
51	Objects float on the fluid surface	Laws of Buoyancy, Fluid Dynamics
52	Objects drop into the fluid	Laws of Buoyancy, Fluid Dynamics
53	Objects' movement causes fluid motion	Fluid Dynamics
54	Flowing fluid carries objects along	Laws of Buoyancy, Fluid Dynamics
55	Fluid flows against stationary objects	Fluid Dynamics
56	Fluid transfers from one container to another	Fluid Dynamics, Conservation of Mass, Surface Tension
57	Fluid passes through several connected containers	Fluid Dynamics, Conservation of Mass, Surface Tension
58	Fluid flows through grid-like structures	Fluid Dynamics, Conservation of Mass
59	Fluid moves across mountainous or uneven landscapes	Fluid Dynamics
60	Increasing fluid volume elevates the surface level	Fluid Dynamics
61	Fluid flows along the contours of an object's surface	Fluid Dynamics
62	Jet-like fluid projection upward or outward	Fluid Dynamics
63	Fluid exhibits surface tension	Fluid Dynamics, Laws of Surface Tension
64	Fluid refracts light when crossing media	Fluid Dynamics, Optics, Law of Refraction (Snell's Law)
65	Sticky fluid drips and accumulates on objects	Laws of Cohesion, Viscous Flow
66	Sticky fluid falls from an object's surface	Laws of Cohesion, Laws of Viscous Flow (Navier-Stokes)
67	An elastic object falls and bounces on another surface	Laws of Elasticity
68	A plasticine object falls and deforms on a surface	Laws of Plasticity
69	A Newtonian fluid falls and spreads across a surface	Laws of Viscous Flow (Navier-Stokes)
70	A Non-Newtonian fluid falls and flows with variable resistance	Laws of Viscoplastics Flow
71	A granular substance falls and disperses across a surface	Laws of Friction

5.2. More Details of Collecting 3D Assets

To construct our dataset, we source 3D assets, including meshes and textures, from publicly available platforms such as SketchFab [77], Fab [30], ShareTextures and Blender Kit. All downloaded assets have been verified to carry licenses compatible with non-commercial use, including CC BY-NC, CC BY-SA, CC BY-NC-SA, CC0, CC BY and RF. We ensure that every file complies with the licensing terms and can be legally used for building a non-commercial dataset.

For assets obtained from SketchFab, we apply a filtering process to select meshes that meet the required licensing conditions and have verified that the authors did not explicitly prohibit AI-related usage.

For assets downloaded from Fab, we ensure that they are licensed under either the CC BY License or the Unreal Engine Standard License, and that the authors explicitly stated these assets can be used for AI-related purposes. Figure 7 and Figure 8 respectively show examples of assets under the CC BY License and the Unreal Engine Standard License, with clear indications of *Allows usage with AI*.

Assets from Blender Kit are distributed under a Royalty-Free (RF) license, which permits commercial use without requiring attribution. Our work complies with these terms as we are building a non-commercial, open-source dataset.

Additionally, ShareTextures offers a wide range of free textures under the CC0 license. We incorporate textures selected from this platform and convert them into materials compatible with Unreal Engine. Furthermore, we supplement these with materials sourced from Fab, resulting in five major material categories comprising a total of 623 materials: metal, stone, wood, fabric, and plastic. Figure 9 presents representative materials from each category.

Interactable Objects: The downloaded meshes are typically solid and lack joint-like structures, making them non-interactive even if they appear to have multiple movable parts. Common examples include swivel chairs, doors, windows, seesaws, and pendulums-objects frequently encountered in daily life. To transform a downloaded mesh from a completely solid object into an interactive one with movable joints, we follow several steps, leveraging Unreal Engine's Physical Constraint tool to define relationships between parts. Taking a swivel chair as an example: 1) We first use Unreal Engine's modeling tools to split the chair model into two parts: the base and the seat. 2) We then create a Chair Blueprint, in which we define two static mesh components, base and seat, and then add a Physical Constraint to establish the relationship between them. Unreal Engine's Physical Constraint supports four types of constraints: linear limit, linear motor, angular limit, and angular motor.

- **Linear Limit**: It restricts translational movement along one or more axes within a specified range.
- **Linear Motor**: It applies a constant or target-driven force to drive linear motion along an axis.
- **Angular Limit**: It constrains rotational movement around one or more axes within defined angular bounds.
- **Angular Motor**: It generates torque to achieve or maintain a target angular velocity or position.

Destructible Objects: In Unreal Engine, these objects are assets that can break apart or deform dynamically during physical activities, simulating realistic physical destruction. This feature enhances immersion by allowing objects such as walls, barrels, or glass panes to respond to forces like explosions, collisions, or player interactions. Unreal Engine implements destructible objects through NVIDIA's Apex Destruction module, which uses fracture meshes and physics simulation to create breakable assets. Creating a destructible object in Unreal Engine typically follows this pipeline:

1. **Prepare the Base Mesh**: To import a closed, manifold static mesh into Unreal Engine.
2. **Create a Destructible Mesh**: In the Content Browser, right-click the static mesh and select *Create Destructible Mesh*.
3. **Apply Fracture Settings**: In the Destructible Mesh Editor, set chunk count, random seed, and depth layers for hierarchical destruction.
4. **Configure Damage Properties**: Define damage threshold, impact damage, and enable debris timeout for performance.

Figure 10 showcases examples of objects.

5.3. More Details of Creating 3D Scenes

In this section, we will describe how we create 3D dynamic scenes utilizing multiple physical engines.

5.3.1. Creating 3D Scenes in Unreal Engine

As an example, we illustrate the construction of a *triple-physics activities* scenario. The pipeline is illustrated in Figure 11.

Step 1: Emulating Multiphysics

We randomly select three physical phenomena from Table 8. For example:

- Index 6: Wind applied to an object moving in the same direction;
- Index 12: Object rolling up a slope;
- Index 21: Object on a coarse surface with friction.

These descriptions are intentionally abstract and serve as guidelines for Step 2 (background setting) and Step 3 (object placement), ensuring that the final setup incorporates all three phenomena.

Step 2: Setting Up Backgrounds

Based on the selected phenomena, we choose an indoor background to facilitate the construction of a slope and the placement of a fan, naturally aligning with an indoor setting.

Step 3: Placing Multiobjects

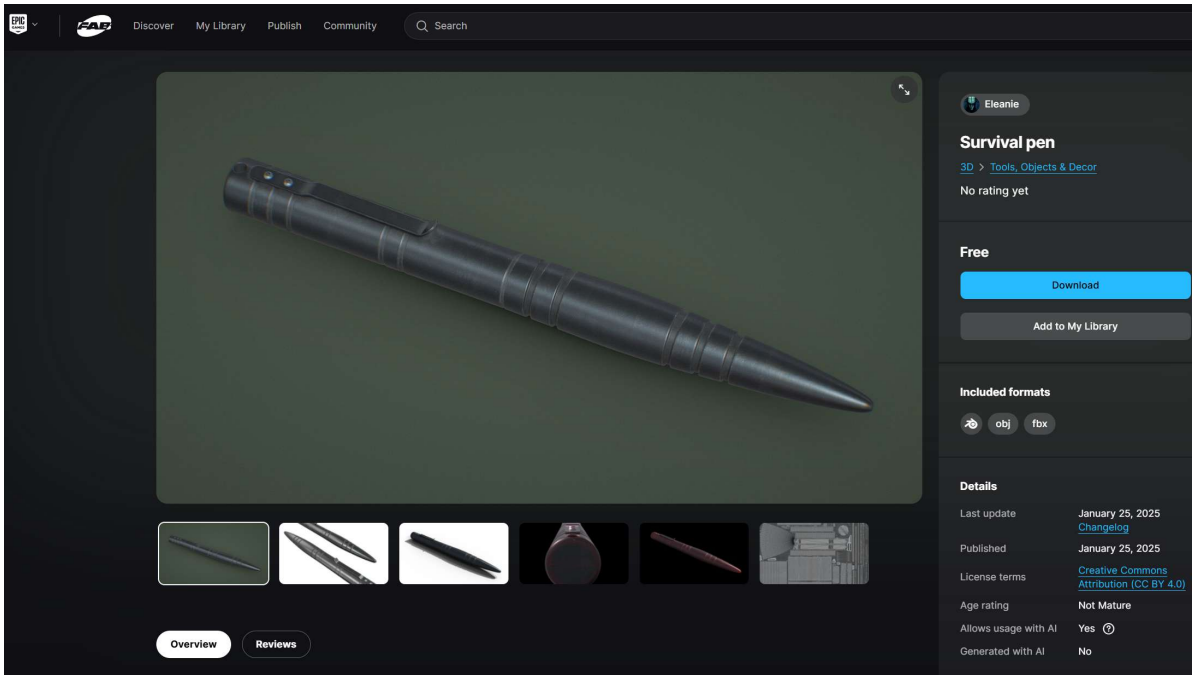

Figure 7. Exemplary 3D asset under CC BY License

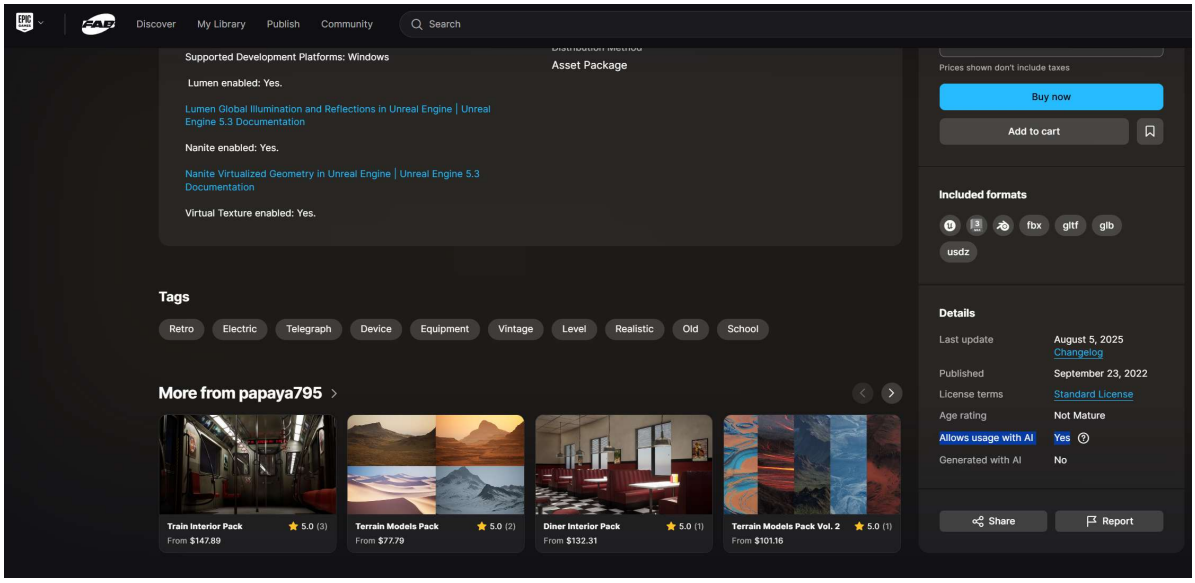

Figure 8. Exemplary 3D asset under UE Standard License

We position a wooden plank on the edge of a table to form a slope. Two small cars with initial velocities are placed at the bottom so they can roll upward. A fan is placed behind one car to apply wind force during motion.

Step 4: Varying Materials

To create more 3D scenes, we modify the slope to have a steel appearance and adjust its physical parameters. Similarly, the two cars are assigned different visual appearances and distinct physical properties.

5.3.2. Creating 3D Scenes Concerning Liquid

To construct a 3D scene with liquid, we follow the same four-step procedural pipeline. The pipeline is illustrated in Figure 12.

Step 1: Emulating Multiphysics

For liquid simulation, we include single-physics, double-physics, and triple-physics activities. We randomly select one to three physical phenomena from Table 8.

Step 2: Setting Up Backgrounds

One of 207 candidate backgrounds is selected. These

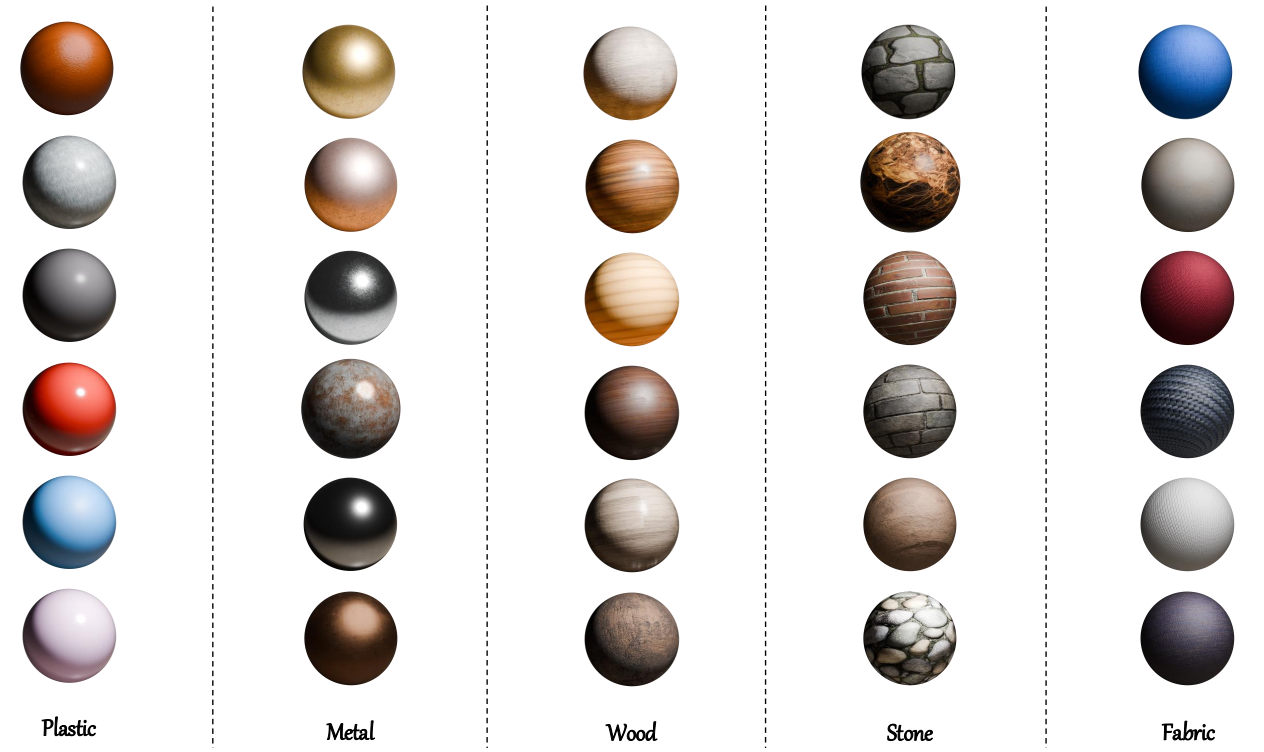

Figure 9. Examples of materials.

Figure 10. Examples of 3D assets.

provide contextual geometry, including diverse categories, such as bathroom, kitchen, *etc*..

Step 3: Placing Multiobjects

Multiple objects are placed against the background. We set objects that will not be driven by the liquid as solid (*e.g*., the container of the fluid), while objects that can be

Figure 11. The pipeline to create 3D scenes in Unreal Engine.

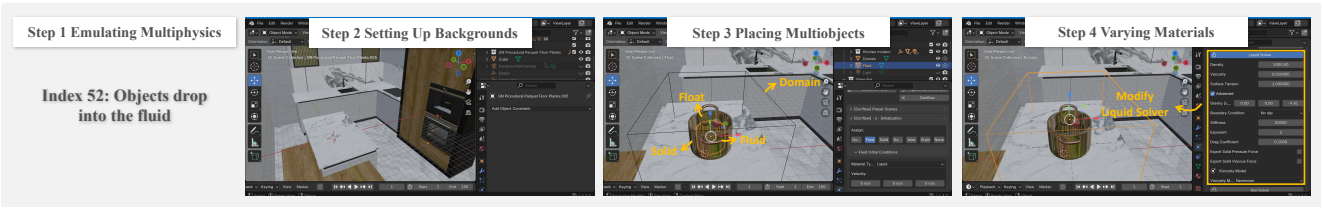

Figure 12. The pipeline to create 3D scenes concerning liquid.

moved by the liquid are set as float. For a solid object, a domain will be set up to define the area for liquid simulation. To realize cases like stirring or pouring, we also design animations through key frames. Liquid can be initialized in arbitrary shapes, mainly depending on the shape of the liquid container. To enhance diversity, Inlets are also incorporated into the initialization (*e.g.*, fountains).

Step 4: Varying Materials

We select different fluid types, including Newtonian, non-Newtonian fluids, *etc*.. We adjust various physical parameters, such as viscosity, surface tension, *etc*..

5.3.3. Creating 3D Scenes Concerning Special Materials

Our dataset includes a subset of 1,440 dynamic 3D scenes designed to evaluate current methods for physical property estimation. This subset spans five representative material categories, elastic solids, plasticine, Newtonian fluids, non-Newtonian fluids, and granular substances, where each category is paired with distinct initial geometries to capture structural and behavioral variability. To construct each dynamic 3D scene, we follow the same four-step procedural pipeline. The pipeline is illustrated in Figure 13.

Step 1: Emulating Multiphysics

We primarily capture the physical interactions of objects of different materials as they are released and collide under gravity, or as they fall onto boundary objects. We first randomly select one to three physical phenomena from the table 8. For example:

- Index 69: A Newtonian fluid falls and spreads across a surface.

Step 2: Setting Up Backgrounds

Then we will choose a background. These provide contextual geometry (*e.g.*, walls, floors, or distant scenery) but they are purely passive, that they neither participate in nor perturb the underlying physics.

Step 3: Placing Multiobjects

Objects are arbitrarily placed in the scene and defined as either boundary objects(static colliders) or the main object(deformable body) in the MPM simulation. These boundary objects have simple or complex geometry and directly contact with the main object, thus influencing local physical interactions based on boundary conditions. The main object is initialized as one of several candidate shapes, *e.g.*, cube, sphere, torus, complex mesh, *etc*..

Step 4: Varying Materials

The materials of the main objects are selected from the five categories, which determine their physical behavior, including deformation, flow, and response to contact.

5.4. More Details of Simulating Physical Dynamics

5.4.1. Chaos Physics

Unreal Engine's Chaos Physics system is employed to simulate physically accurate dynamics within the created 3D scenes. It provides deterministic and high-performance simulation for complex interactions. Specifically, in our dataset, Chaos is responsible for:

- **Rigid Body Dynamics**: Simulating the motion and interaction of solid objects under forces such as gravity, collisions, and impulses. This includes accurately handling of linear and angular momentum for static and dynamic objects.
- **Fracture and Destruction**: Managing breakable geometry through procedural fracturing and hierarchical destruction. Objects are divided into chunks that respond to impact forces, enabling realistic breakage and debris behavior during collisions or explosions.

Beyond these core functionalities, several interactive physical effects are implemented using Unreal Engine components combined with Chaos for accurate responses:

- **Wind Force**: The wind effect is implemented as a rectangular force field region within the scene. Inside this region,

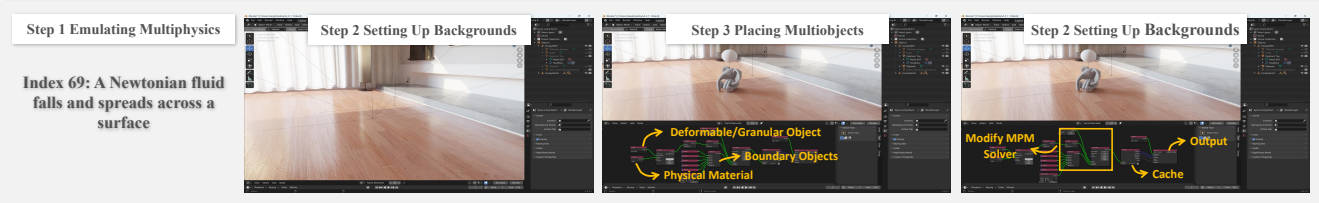

Figure 13. The pipeline to create 3D scenes concerning special materials.

a constant force direction is maintained, simulating a uniform wind flow. However, the force magnitude decays quadratically with the distance from the wind source, ensuring that objects closer to the origin experience stronger wind influence. At the far end of the force field, the force magnitude approaches zero, creating a realistic attenuation effect across the region. Chaos Physics computes the rigid body response to this spatially varying force distribution.

- **Laser Interaction**: Laser beams are represented as meshes for visualization. Collision detection is handled by Chaos Physics to determine intersections with scene objects. Upon impact, the surface normal is retrieved, and the reflection direction is computed based on the law of reflection. Reflection occurs only on materials designated as mirrors, enabling controlled optical behavior.
- **Magnetic Force**: We model magnetic interactions using a physics-based dipole field formulation. Specifically, the normalized magnetic vector field $\mathbf{B}(\mathbf{p})$ generated by a source magnet's poles $(\mathbf{p}_N, \mathbf{p}_S)$ at any spatial point $\mathbf{p}$ is established as $\mathbf{B}(\mathbf{p}) = \frac{\mathbf{r}_N}{\|\mathbf{r}_N\|^3} - \frac{\mathbf{r}_S}{\|\mathbf{r}_S\|^3}$, where $\mathbf{r}_{N,S} = \mathbf{p} - \mathbf{p}_{N,S}$.
  When a target magnet enters this continuous vector field, the system samples $\mathbf{B}$ at its respective poles to calculate the localized forces acting on each point. By aggregating these pole-specific forces, we analytically derive the translational force and rotational torque applied to the object. This formulation yields physically accurate kinematics, including homopolar repulsion, heteropolar attraction, and automatic streamline alignment.

5.4.2. Doriflow

We employ Doriflow[27], a Blender add-on for fluid simulation, to produce physically consistent fluid sequences for our dataset. We use Blender 4.3.0 and Doriflow 1.3. We introduce a subset of 300 scenes for liquid simulation, mainly focusing on Newtonian and non-Newtonian fluid.

Following the official Doriflow workflow, our simulations rely solely on explicitly defined geometry, emitters, and physical parameters. Assets of a 3D scene are imported into Blender and designated as either fluid emitters or solid colliders. Doriflow then performs the required preprocessing like generating internal voxel representations, preparing the simulation domain according to the user-specified resolution and bounds, and generating SPH particles, *etc*..

During simulation, Doriflow advances fluid states using its built-in GPU-accelerated solver. Each frame is computed through standard fluid-dynamics substeps, including advection, neighbor search, viscosity integration, external force application (*e.g*., gravity), pressure solving, and collision handling against the prepared collider geometry. The pressure and viscous forces involved in these steps are calculated as follows:

- **Pressure Force Computation**:

$$\mathbf{F}_{\text{press},i} = -\sum_j m_i m_j \frac{p_i + p_j}{2\rho_i \rho_j} \nabla W(\mathbf{r}_{ij}, h)$$

  This equation calculates the pressure force on particle i due to particle j using the gradient of the smoothing kernel, where $\mathbf{F}_{\text{press},i}$ is the pressure force on particle i, m_i and m_j are masses, p_i and p_j are pressures, ρ_i and ρ_j are densities, ∇W is the gradient of the smoothing kernel, $\mathbf{r}_{ij}$ is the distance vector, and h is the smoothing length.
- **Viscous Force Computation**:

$$\mathbf{F}_{\text{visc},i} = \sum_j \frac{\mu}{2} \left(\frac{\mathbf{v}_i - \mathbf{v}_j}{\rho_i + \rho_j} \right) \cdot \nabla^2 W(\mathbf{r}_{ij}, h)$$

  This expression quantifies the viscous dissipation between particles, with $\mathbf{F}_{\text{visc},i}$ representing the viscous force acting on particle i, μ denoting the dynamic viscosity coefficient, $\mathbf{v}_i$ and $\mathbf{v}_j$ representing the velocity vectors, and $\nabla^2 W$ indicating the Laplacian of the smoothing kernel.

Fluid injection follows the emission rules defined at setup, enabling controlled generation of diverse flow behaviors.

After simulation completes, Doriflow converts the internal fluid representation into surface meshes for every frame through its built-in meshing stage. These meshes are stored directly as Blender geometry tracks and can be rendered or exported without further processing.

Doriflow allows configuration of parameters such as simulation resolution and time step size. For objects with key frames, Doriflow simulates the fluid based on the motion of these meshes. To enhance the reality, for large scenes such as mountains and valleys, we also add white water effects in Doriflow, including bubbles and foams. We adjust the transparency and size to improve the visual quality.

This integrated workflow provides a reproducible, Blender-centric method for generating high-fidelity fluid motion suitable for large-scale datasets.

5.4.3. Taichi Lang

We implement a full Material Point Method (MPM) simulation pipeline using Taichi and all constitutive and projection models in our solver, following the definitions provided in PAC-NeRF [55].

For completeness, we summarize the constitutive laws and plasticity projections adopted for the five material categories. Let $\mathbf{F}$ be the deformation gradient, $J = \det(\mathbf{F})$ its Jacobian, and $\mathbf{T}$ the Cauchy stress. We also denote by $Z(\mathbf{F})$ a return-mapping operator that projects $\mathbf{F}$ back to the admissible elastic region whenever a yield condition is violated.

Elastic Solids: Elastic objects are modeled with the neo-Hookean law. The stress is written as:

$$J\,\mathbf{T}(\mathbf{F}) = \mu\big(\mathbf{F}\mathbf{F}^\top\big) + \big(\lambda \log J - \mu\big)\,\mathbf{I}, \tag{2}$$

where μ and λ are Lamé parameters related to Young's modulus E and Poisson ratio ν via

$$\mu = \frac{E}{2(1+\nu)}, \qquad \lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}. \tag{3}$$

Plasticine: Plasticine is treated as an St. Venant–Kirchhoff (StVK) solid in Hencky (logarithmic) strain, combined with von-Mises plasticity. We write the singular value decomposition (SVD) of $\mathbf{F}$ as $\mathbf{F} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ and define the Hencky strain $\boldsymbol{\epsilon} = \log \mathbf{\Sigma}$. The elastic stress is

$$J\,\mathbf{T}(\mathbf{F}) = \mathbf{U}\big(2\mu\,\boldsymbol{\epsilon} + \lambda\,\mathrm{tr}(\boldsymbol{\epsilon})\,\mathbf{I}\big)\mathbf{U}^\top. \tag{4}$$

Let $\hat{\boldsymbol{\epsilon}}$ be the normalized Hencky strain and τ_Y the yield stress. The von Mises yield measure is

$$\delta\gamma = \|\hat{\boldsymbol{\epsilon}}\| - \frac{\tau_Y}{2\mu}. \tag{5}$$

If $\delta\gamma \le 0$, the state is elastic and $Z(\mathbf{F}) = \mathbf{F}$. Otherwise,

$$Z(\mathbf{F}) = \mathbf{U}\exp\Big(\boldsymbol{\epsilon} - \delta\gamma\frac{\hat{\boldsymbol{\epsilon}}}{\|\hat{\boldsymbol{\epsilon}}\|}\Big)\mathbf{V}^\top. \tag{6}$$

Newtonian Fluids: For Newtonian fluids we use a J-based formulation with an additional viscous term. Let $\mathbf{v}$ denote the velocity field and $\nabla\mathbf{v}$ its spatial gradient. The stress is defined as:

$$J\,\mathbf{T}(\mathbf{F}) = \tfrac{1}{2}\,\mu\big(\nabla\mathbf{v} + \nabla\mathbf{v}^\top\big) + \kappa\big(J - J^{-6}\big). \tag{7}$$

Non-Newtonian Fluids: Non-Newtonian fluids are modeled with a viscoplastic extension of the above von Mises plasticity. Using the same SVD $\mathbf{F} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ and strain $\boldsymbol{\epsilon}$, let d be the spatial dimension and define

$$\begin{aligned}
\hat{\mu} &= \mu\,\frac{\mathrm{tr}(\mathbf{\Sigma}^2)}{d} \\
\boldsymbol{s} &= 2\mu\,\hat{\epsilon} \\
\hat{s} &= \|\boldsymbol{s}\| - \frac{\delta\gamma}{1 + \frac{\eta}{2\hat{\mu}\Delta t}}
\end{aligned} \tag{8}$$

where η is the plastic viscosity. The return map becomes

$$Z(\mathbf{F}) = \begin{cases} \mathbf{F}, & \delta\gamma \le 0, \\ \mathbf{U}\exp\Big(\frac{\hat{s}}{2\mu}\,\hat{\boldsymbol{\epsilon}} + \frac{1}{d}\mathrm{tr}(\boldsymbol{\epsilon})\,\mathbf{1}\Big)\mathbf{V}^\top, & \text{otherwise}, \end{cases} \tag{9}$$

with $\eta = 0$ recovering the plasticine model.

Granular Substances: Granular materials use a Drucker–Prager yield condition with the same StVK log-strain elasticity. With $\boldsymbol{\epsilon}$ and $\hat{\boldsymbol{\epsilon}}$ defined above, the yield criterion is

$$\mathrm{tr}(\boldsymbol{\epsilon}) > 0 \quad \text{or} \quad \delta\gamma = \|\hat{\boldsymbol{\epsilon}}\|_F + \alpha\frac{(d\lambda + 2\mu)\,\mathrm{tr}(\boldsymbol{\epsilon})}{2\mu} > 0, \tag{10}$$

where α relates to the friction angle θ_{fric} via

$$\alpha = \sqrt{\tfrac{2}{3}}\,\frac{2\sin\theta_{\mathrm{fric}}}{3 - \sin\theta_{\mathrm{fric}}}. \tag{11}$$

The return mapping is

$$Z(\mathbf{F}) = \begin{cases} \mathbf{U}\mathbf{V}^\top, & \mathrm{tr}(\boldsymbol{\epsilon}) > 0, \\ \mathbf{F}, & \delta\gamma \le 0,\ \mathrm{tr}(\boldsymbol{\epsilon}) \le 0, \\ \mathbf{U}\exp\Big(\boldsymbol{\epsilon} - \delta\gamma\frac{\hat{\boldsymbol{\epsilon}}}{\|\hat{\boldsymbol{\epsilon}}\|}\Big)\mathbf{V}^\top, & \text{otherwise}. \end{cases} \tag{12}$$

Also, we integrate it into Blender 3.6.23 through an extended version of the Taichi Elements addon. The system supports the entire workflow from geometry import and discretization to GPU-parallel simulation, complex boundary handling, caching, and reconstruction for visualization.

Geometric assets are loaded directly inside Blender, including native meshes and BlenderKit models. For each object, we specify whether it behaves as a deformable body or a static collider and assign its material model, chosen from elastic solids, plasticine, Newtonian fluids, non-Newtonian fluids, or granular substances. The selected geometry is then discretized into an MPM representation using Taichi Elements: mesh surfaces are voxelized and sampled into particles, and all particle attributes (mass, velocity and material parameters, *etc*..) are initialized and stored as Taichi GPU fields.

During simulation, each frame is advanced through several MPM substeps executed entirely on GPUs. In each substep, particle states are first transferred to the grid (P2G). For every particle, we compute a trial deformation update from its affine velocity field, apply the appropriate material-dependent deformation projection, evaluate the corresponding constitutive model, and transfer mass, momentum, and APIC affine terms to grid nodes through quadratic interpolation. This stage fully determines the behavior of different materials and follows the exact constitutive formulations of PAC-NeRF. After P2G, grid velocities are normalized, external forces such as gravity are applied in a dedicated kernel.

Boundary interactions are handled in a subsequent grid post-processing phase, where several collider kernels are executed sequentially. These include a global axis-aligned bounding box, analytic primitive colliders such as planes and spheres, and a complex mesh collider capable of handling arbitrary Blender geometry. Colliders operate directly on grid velocities by projecting or clamping them according to the selected contact mode (sticky, slip, or separate). The updated grid velocities are then transferred back to particles (G2P), after which particle positions, velocities, and affine fields are advanced.

To robustly support collisions with highly detailed Blender meshes, we implement a voxelized Signed Distance Field (SDF) collider. Input meshes are voxelized into a regular grid, inside/outside regions are identified, and each voxel is assigned a signed distance value and an estimated normal. During simulation, grid nodes near the mesh surface query the SDF and adjust their velocity according to the relevant contact rules. This approach provides a stable, high-resolution, and geometry-agnostic collision handling mechanism that integrates seamlessly with the Taichi Elements pipeline. To enhance dataset diversity, we stochastically vary the simulation parameters (*e.g.*, initial pose, velocity, or material coefficients) during refinement. In the Taichi Elements addon, parameters such as the simulation domain, simulation resolution, and time step are set for the scene. We use a uniform parameter configuration: $dt = 1/150, dx = dy = dz = 0.0083$.

In the visualization pipeline, continuous surfaces are reconstructed from particles using Blender Geometry Nodes, with standard Blender materials and rendering engines applied to enhance realism. However, for granular substances, we opt for direct particle rendering to accurately capture their discrete nature. Regarding material representation, commonly referenced physical materials are used as benchmarks: for instance, honey serves as a uniform rendering material for viscous Newtonian fluids, while sand is adopted for granular substances. The whole procedure yields 240 scenes per material category, resulting in a total of 1,200 scenes.

5.4.4. Physical Parameter Range

We report the ranges of example physical parameters in Table 9. The wide ranges of these parameters highlights the diversity of our dataset.

Table 9. Ranges of Example Physical Parameters.

	Friction Coefficient	Restitution	Density	SPH Viscosity	SPH Surface Tension	Fluid Viscosity	Plasticity Viscosity	Yield Stress	Friction Angle	..
min ∼ max	0. ∼ 0.9	0.1 ∼ 0.8	0.1 ∼ 21.3	0.0 ∼ 5.0	0.5 ∼ 3.0	25 ∼ 200	3 ∼ 10	0.1 ∼ 10	15 ∼ 60	..

5.5. More Details of Cameras and Rendering

5.5.1. Static Camera Position Sampling

The cameras are positioned on a spherical surface whose center coincides with the physical event's origin. The sphere's radius is chosen such that it fully encompasses all objects involved in the event as well as their expected motion range, ensuring complete coverage of the phenomenon. Initially, we uniformly sample 12 static cameras on the upper hemisphere, with camera viewpoints evenly distributed within a latitude range of 30° to 60°. Cameras that suffer from occlusions are resampled, ensuring that all 12 cameras are well positioned and provide unobstructed coverage. Figure 14 shows an example of static camera distribution.

For special material and liquid simulations, we leverage Blender to generate cameras and render corresponding RGB images, depth and segmentation information. The static camera viewpoints are uniformly sampled from the upper hemisphere, 15 camera viewpoints for special material simulations and 12 camera viewpoints for liquid simulations. Dynamic camera trajectories are generated based on predefined camera sampling strategies.

5.5.2. Monocular Camera Trajectory Sampling

To render monocular videos, we design three trajectory sampling strategies as detailed in below, and one of them is randomly selected for each dynamic 3D scene. Each strategy returns N sampled positions, which are used to interpolate more positions, forming a smooth trajectory whose temporal resolution matches the number of frames in the video.

Linear Drift Sampling: This method generates a trajectory by evenly spacing sample positions along longitude within a fixed range of 180°, starting from a randomly chosen initial longitude. The direction of progression (clockwise or counterclockwise) is selected randomly. For latitude, the first position is sampled within the range of $[-45^\circ, 45^\circ]$, respecting hemisphere constraints (upper, lower, or both). The number of sample points is randomly chosen between 10 and 20. Subsequent latitude values are obtained by adding random perturbations to the previous point, ensuring each new latitude remains within the allowed range and satisfies hemisphere restrictions. If a valid latitude cannot be found after repeated attempts, a fallback value is used. This approach creates a trajectory that is globally structured along longitude but locally varied in latitude, balancing deterministic progression with stochastic variation. Figure 15 shows an example of camera trajectory.

Sinusoidal Interpolation Sampling: This method generates a smooth trajectory between a randomly sampled start point and end point on the sphere. Both points are sampled within a longitude span of 180° and a latitude range of $[-45^\circ, 45^\circ]$, respecting hemisphere constraints (upper, lower, or both). The trajectory is interpolated using sinusoidal functions, ensuring a gradual transition in both longitude and latitude rather than abrupt changes. A dynamic radius factor is applied along the path to simulate natural camera movement. Compared to Linear Drift Sampling, this approach produces a more structured and aesthetically pleasing curve, generating smooth viewpoint transitions. Figure 16 shows an example of camera trajectory.

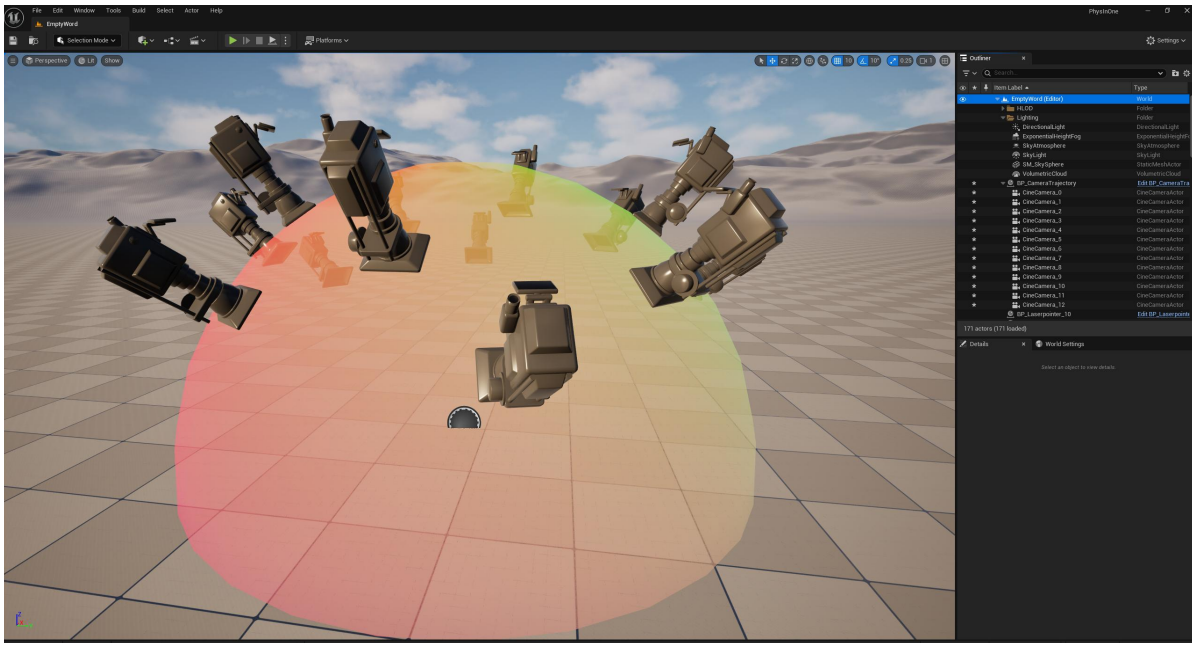

Figure 14. Static Camera Sampling

Circular Loop Trajectory Sampling on Sphere: This method samples points along a circular trajectory on a sphere. A loop center is randomly chosen within hemisphere constraints, and a circle of adjustable size is defined by a loop intensity parameter. The trajectory is parameterized by n evenly spaced angles, generating latitude and longitude offsets that form a closed loop around the center. Hemisphere clipping ensures valid positions, and an optional radial variation introduces diversity. This approach provides structured yet flexible sampling for spherical domains. Figure 17 shows an example of a camera trajectory.

5.5.3. Distribution of Video Lengths

Our dataset consists of 2 million videos generated from 153,810 unique dynamic 3D scenes. Figure 18 presents the distribution of video lengths within the dataset, highlighting that a substantial proportion, approximately 75%, of the videos are long-horizon, lasting longer than 5 seconds ($\geq$ 150 frames). This characteristic makes our dataset particularly well-suited for a variety of challenging tasks in long-horizon, physically-grounded video generation, simulation, understanding, *etc*..

5.6. More Details of Annotating and Splitting Data

5.6.1. Annotating

Our dataset has annotations of geometry, semantics, motion, physical properties, and textual descriptions.

- **Geometry**: Depth maps measured in meters with a resolution of 1120×1120 are rendered with RGB images. Meshes of all 3D assets are also available.
- **Semantics**: Segmentation maps with a resolution of 1120×1120, where each foreground object is assigned a unique ID starting from 1 and the background is assigned 0, are also rendered with RGB images.
- **Motion**: Each object is associated with a local coordinate origin. By recording the position and rotation of this origin in the world coordinate system at every time step, we capture the object's 3D trajectory. The 3D trajectory of any point on the object can be reconstructed using its local coordinates. For rigid objects, based on the camera information, depth and segmentation maps, we can reconstruct the 3D points. Through the position and rotation at each time step, we can calculate the transformation of the object between different frames, and thus we can obtain the dense flow for rigid objects.
- **Physical Properties**: Each object is linked to a physical material characterized by four key parameters: dynamic friction coefficient, static friction coefficient, density, and restitution coefficient. These parameters are stored in a JSON format, where the key corresponds to the object's segmentation ID and the value contains its physical attributes.
- **Text Annotating**: The text annotations are manually curated to inject human-level understanding of dynamics and physics into our dataset, addressing a key limitation of current state-of-the-art video understanding pipelines, which often fail to explain scenes involving highly coupled objects and complex, varying dynamics, let alone perform underlying physical reasoning. Specifically, we recruit human annotators to watch each scene at least three times from at least three different camera viewpoints. For each

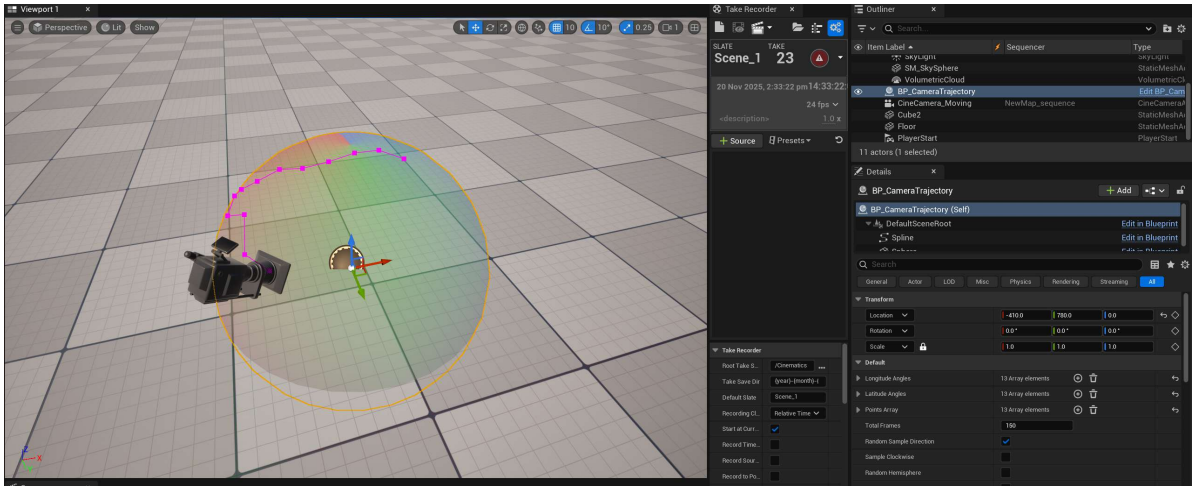

Figure 15. Linear Drift Sampling

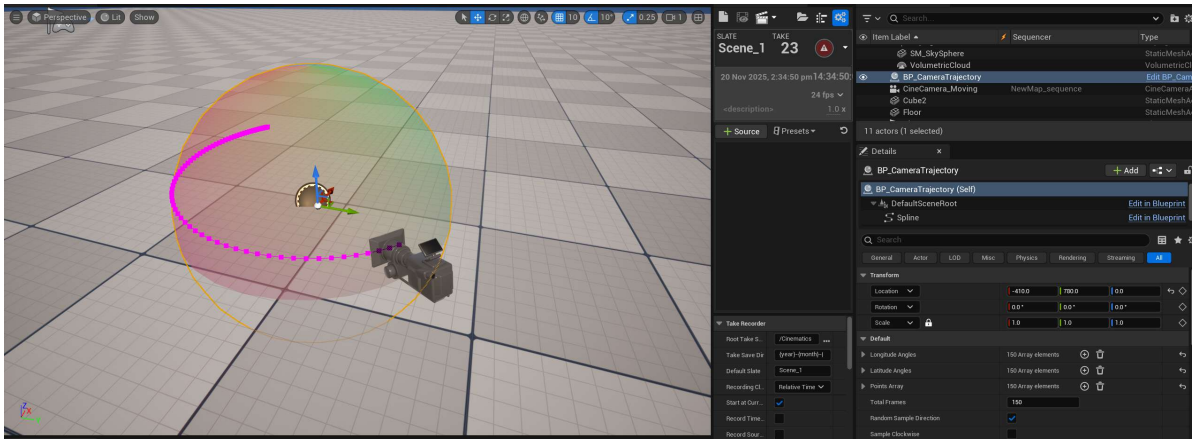

Figure 16. Sinusoidal Interpolation Sampling

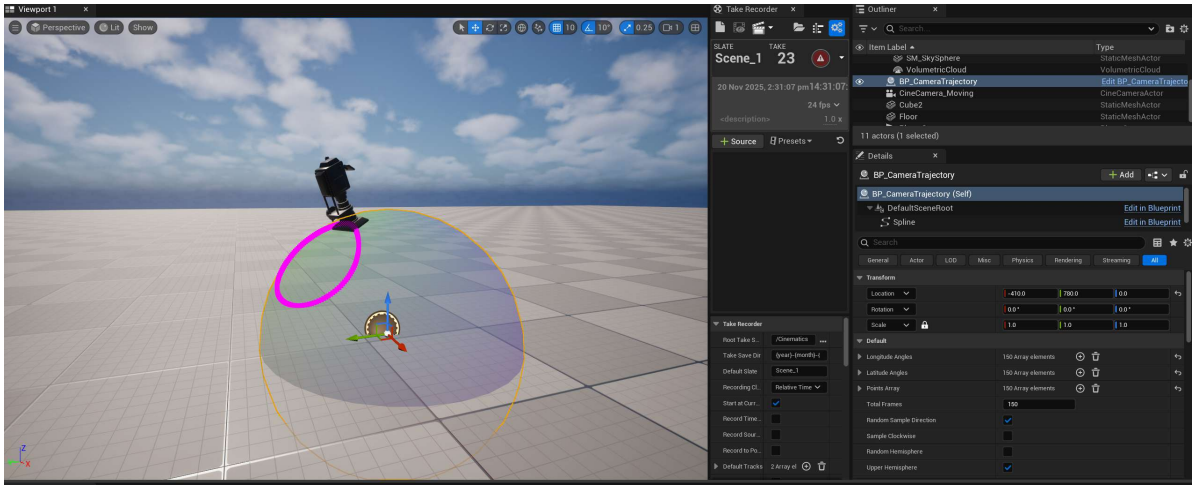

Figure 17. Circular Loop Trajectory Sampling on Sphere

scene, they are instructed to describe:

1. The initial state of every object (*e.g.*, position, orientation, initial velocity);
2. The dynamic behavior of each object together with the governing physical principles (*e.g.*, *"the white football falls due to gravity and rebounds upon collision with the ground"*);
3. The dominant physical laws illustrated by the scene

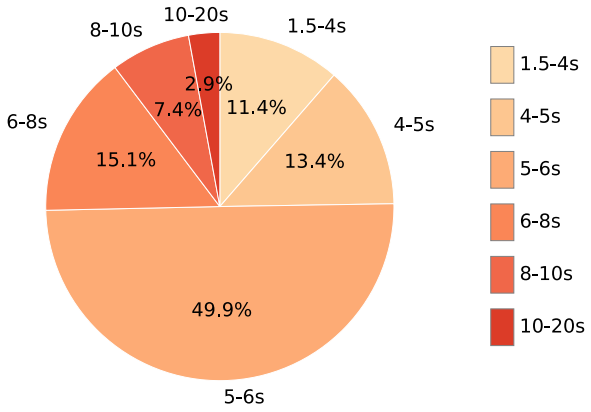

Figure 18. The distribution of video lengths.

as a whole (*e.g.*, conservation of momentum, elastic collision).

With accurate human annotations, we refine the descriptions using Qwen3-VL-235B-A22B-Thinking, a large language model, to correct grammatical errors, improve clarity, and enhance completeness. During this polishing step, we provide an additional prompt that emphasizes object appearance details and explicitly highlights the relevant physics laws, ensuring the final annotations are both fluent and physically informative.

5.6.2. Dataset Splitting

Each scene contains $1 \sim 3$ physical phenomena from Table 8. We partition all scenes into train, validation, and test splits at the ratio of 8:1:1, while ensuring that the proportion of each physical phenomenon is approximately preserved across splits. In the meantime, 3D assets in different partitions appear exclusively.

5.6.3. Dataset Quality

Our dataset quality is ensured through a structured workflow, including standardized assets, seasoned UE developers, multiple rounds of data validation by independent reviewers using unified criteria.

5.7. More Details of Video Generation

5.7.1. Experiment Setup

We evaluate three representative video generation models on our dataset: Stable Video Diffusion XL (**SVD**)[10], an image-to-video model, and two image-text-to-video models: CogVideoX-1.5-5B (**CogVideoX**)[91] and Wan 2.2-5B (**Wan2.2**)[83].

SVD is based on a U-Net architecture, where the diffusion denoiser follows a U-shaped design. It conditions the generation process on the first-frame image by concatenating input noise with the image and computing cross-attention between the denoising hidden states and image features extracted from a pre-trained CLIP encoder at multiple layers. In contrast, both Wan2.2 and CogVideoX adopt a Diffusion Transformer (DiT) architecture. Their key difference lies in how they incorporate conditional information: Wan2.2 injects the text prompt via cross-attention layers alongside image conditioning, whereas CogVideoX concatenates all condition signals, including the reference image and text embeddings, into a single long token sequence that is jointly processed by the transformer.

In this paper, we fine-tune each model until convergence on a subset of our dataset, with about 1K training steps for SVD, 2K steps for Wan2.2, and 1K steps for CogVideoX. To provide practical guidance for future users interested in adapting these models to our data, we explore three distinct fine-tuning strategies: Low-Rank Adaptation (LoRA), Supervised Fine-Tuning (SFT), and Final-Layer Tuning (FLT), resulting in a suite of adapted variants for each architecture.

All experiments use a learning rate of 0.0001 and a batch size of 64. For LoRA, we set the rank to 32 and the scaling factor α to be 1.0 uniformly across all applicable layers. For FLT, we freeze all parameters except those in the final two transformer blocks (or equivalent decoder layers), yielding approximately 400 MB of trainable parameters per model.

5.7.2. Details of PMF

The key motivation of introducing Fourier based metrics for evaluating generated videos is that we aim to focus on evaluating **motion dynamics** rather than the absolute correctness of pixel values. Specifically, given an AI-generated video $\mathcal{V}_{\text{gen}}$ and a reference video $\mathcal{V}_{\text{ref}}$, we apply the 3D Discrete Fourier Transform (DFT) independently to each video across its spatial and temporal dimensions ($H \times W \times T$):

$$\tilde{\mathcal{V}}(c;u,v,s) = \sum_{h,w,t} \mathcal{V}(c;h,w,t) \cdot e^{-2\pi i\left(\frac{uh}{H}+\frac{vw}{W}+\frac{st}{T}\right)} \quad (13)$$

where $\tilde{\mathcal{V}}(c;u,v,s)$ denotes the Fourier coefficients at spatial-temporal frequency indices (u,v,s) for color channel $c \in \{0,1,2\}$ (corresponding to RGB), capturing spatial frequencies (u,v) and temporal frequency s. For brevity, we use $\sum_{h,w,t}$ as shorthand for the full spatiotemporal summation $\sum_{h=0}^{H-1}\sum_{w=0}^{W-1}\sum_{t=0}^{T-1}$, which aggregates over all spatial and temporal positions or frequencies in the video.

The following well-known properties of the Fourier transform are instrumental to our analysis:

We further define the amplitude $A_{c;u,v,s}$, phase $\psi_{c;u,v,s}$, and normalized energy $E_{u,v,s}$ for each frequency component per color channel as follows:

$$A_{c;u,v,s} = \left|\tilde{\mathcal{V}}(c;u,v,s)\right| \quad (14)$$

$$\begin{aligned}\psi_{c;u,v,s} &= \arg\left(\tilde{\mathcal{V}}(c;\vec{\omega})\right) \\ &= \arctan\left(\frac{\mathrm{Im}\left(\tilde{\mathcal{V}}(c;u,v,s)\right)}{\mathrm{Re}\left(\tilde{\mathcal{V}}(c;u,v,s)\right)}\right)\end{aligned} \quad (15)$$

$$E_{u,v,s} = \frac{\sum_{c=0}^{2} \left|\tilde{\mathcal{V}}(c;u,v,s)\right|^2}{\sum_{u',v',s'} \sum_{c=0}^{2} \left|\tilde{\mathcal{V}}(c;u',v',s')\right|^2} \tag{16}$$

Crucially, under the standard assumption that the video signal is extended beyond its spatial and temporal boundaries via duplicate padding, we consider (1) a spatiotemporal shift, denoted by $\delta\mathbf{x} = (\delta h, \delta w, \delta t)$, and (2) a brightness rescaling in the original video signal, denoted by $\lambda \in (0, 1]$. This yields a shifted video $\mathcal{V}'$ defined as:

$$\mathcal{V}'(c;h,w,t) = \lambda \mathcal{V}(c;h+\delta h, w+\delta w, t+\delta t) \tag{17}$$

Then we apply Fourier transform on $\mathcal{V}'$:

$$\begin{aligned}
&\tilde{\mathcal{V}}'(c;u,v,s) \\
&= \sum_{h,w,t} \lambda \mathcal{V}(c;h+\delta h, w+\delta w, t+\delta t)\, e^{-2\pi i\left(\frac{uh}{H}+\frac{vw}{W}+\frac{st}{T}\right)} \\
&= \sum_{h',w',t'} \lambda \mathcal{V}(c;h',w',t')\, e^{-2\pi i\left(\frac{u(h'-\delta h)}{H}+\frac{v(w'-\delta w)}{W}+\frac{s(t'-\delta t)}{T}\right)} \\
&= \lambda e^{i\delta\psi} \sum_{h',w',t'} \mathcal{V}(c;h',w',t')\, e^{-2\pi i\left(\frac{uh'}{H}+\frac{vw'}{W}+\frac{st'}{T}\right)} \\
&= \lambda e^{i\delta\psi} \tilde{\mathcal{V}}(c;u,v,s)
\end{aligned} \tag{18}$$

where $\delta\psi = 2\pi\left(\frac{u\delta h}{H} + \frac{v\delta w}{W} + \frac{s\delta t}{T}\right)$. Note that we assume duplicate padding to prevent boundary artifacts induced by the shift, thereby allowing the summation limits to be extended validly.

Consequently, the magnitude, normalized energy, and phase transform as:

$$\begin{aligned}
A'_{c;u,v,s} &= \left|\tilde{\mathcal{V}}'(c;u,v,s)\right| \\
&= \left|\lambda e^{i\delta\psi}\tilde{\mathcal{V}}(c;u,v,s)\right| \\
&= \lambda\left|\tilde{\mathcal{V}}(c;u,v,s)\right| \\
&= \lambda A_{c;u,v,s}
\end{aligned} \tag{19}$$

$$\begin{aligned}
E'_{u,v,s} &= \frac{\sum_{c=0}^{2} \left|\tilde{\mathcal{V}}'(c;u,v,s)\right|^2}{\sum_{u',v',s'} \sum_{c=0}^{2} \left|\tilde{\mathcal{V}}'(c;u',v',s')\right|^2} \\
&= \frac{\sum_{c=0}^{2} \left|\lambda\,\tilde{\mathcal{V}}(c;u,v,s)\right|^2}{\sum_{u',v',s'} \sum_{c=0}^{2} \left|\lambda\,\tilde{\mathcal{V}}(c;u',v',s')\right|^2} \\
&= \frac{\lambda^2 \sum_{c=0}^{2} \left|\tilde{\mathcal{V}}(c;u,v,s)\right|^2}{\lambda^2 \sum_{u',v',s'} \sum_{c=0}^{2} \left|\tilde{\mathcal{V}}(c;u',v',s')\right|^2} \\
&= E_{u,v,s}.
\end{aligned} \tag{20}$$

$$\begin{aligned}
\psi'_{c;u,v,s} &= \psi_{c;u,v,s} + \delta\psi \\
&= \psi_{c;u,v,s} + 2\pi\left(\frac{u\delta h}{H} + \frac{v\delta w}{W} + \frac{s\delta t}{T}\right)
\end{aligned} \tag{21}$$

From the above derivation, we can see that, a spatiotemporal shift affects only the phase of Fourier coefficients, leaving the amplitude (and hence the energy $E_{u,v,s}$) invariant, while brightness rescaling does not affect the normalized energy or the phase. In a nutshell, the normalized energy is invariant under any spatiotemporal shift and brightness rescaling, and the phase is invariant under brightness rescaling but encodes the information of the spatiotemporal translation.

Hence, we observe that the motion dynamics in $\mathcal{V}'$ are identical to those in $\mathcal{V}$, that only the initial spatiotemporal state (*i.e.*, *when* or *where* the motion begins) and the absolute brightness differ. This insight inspires us to evaluate motion dynamics by comparing the differences between energy spectra.

We propose **PMF (Physical Motion Fidelity)**, a metric that compares the similarity between the normalized spectral energy distributions of $\mathcal{V}_{\text{gen}}$ and $\mathcal{V}_{\text{ref}}$ as follows:

$$\text{PMF}(\mathcal{V}_{\text{gen}}, \mathcal{V}_{\text{ref}}) = -\ln d_{TV}(E^{\text{gen}}, E^{\text{ref}}) \tag{22}$$

where d_{TV} refers to total variation that quantifies the difference between this two distributions:

$$d_{TV}(E^{\text{gen}}, E^{\text{ref}}) = \frac{1}{2}\sum_{u,v,s}\left|E^{\text{gen}}_{u,v,s} - E^{\text{ref}}_{u,v,s}\right| \tag{23}$$

where $E^{\text{gen}}_{u,v,s}$ and $E^{\text{ref}}_{u,v,s}$ denote the normalized energy spectra (as defined in Equation 16) of the generated and reference videos, respectively.

Since the energy spectrum $E_{u,v,s}$ is invariant to spatiotemporal shifts and brightness rescaling, PMF effectively discards misalignment in the initial state and focuses solely on the similarity of motion dynamics across spatial and temporal dimensions. A higher PMF score indicates greater similarity in motion patterns between $\mathcal{V}_{\text{gen}}$ and $\mathcal{V}_{\text{ref}}$. By comparing the energy spectra of $\widetilde{\mathcal{V}}_{\text{gen}}$ and $\widetilde{\mathcal{V}}_{\text{ref}}$, we quantify how well the generated video preserves natural motion patterns in the reference video, thereby decoupling differences in the initial spatiotemporal state.

To demonstrate the distinct focuses of PMF, we present several toy examples in Figure 19. We can see that PMF is sensitive to differences in motion patterns but invariant to simple spatiotemporal shifts or color changing, yielding high scores when dynamics match regardless of starting position. Meanwhile, it degrades when motion is fundamentally altered.

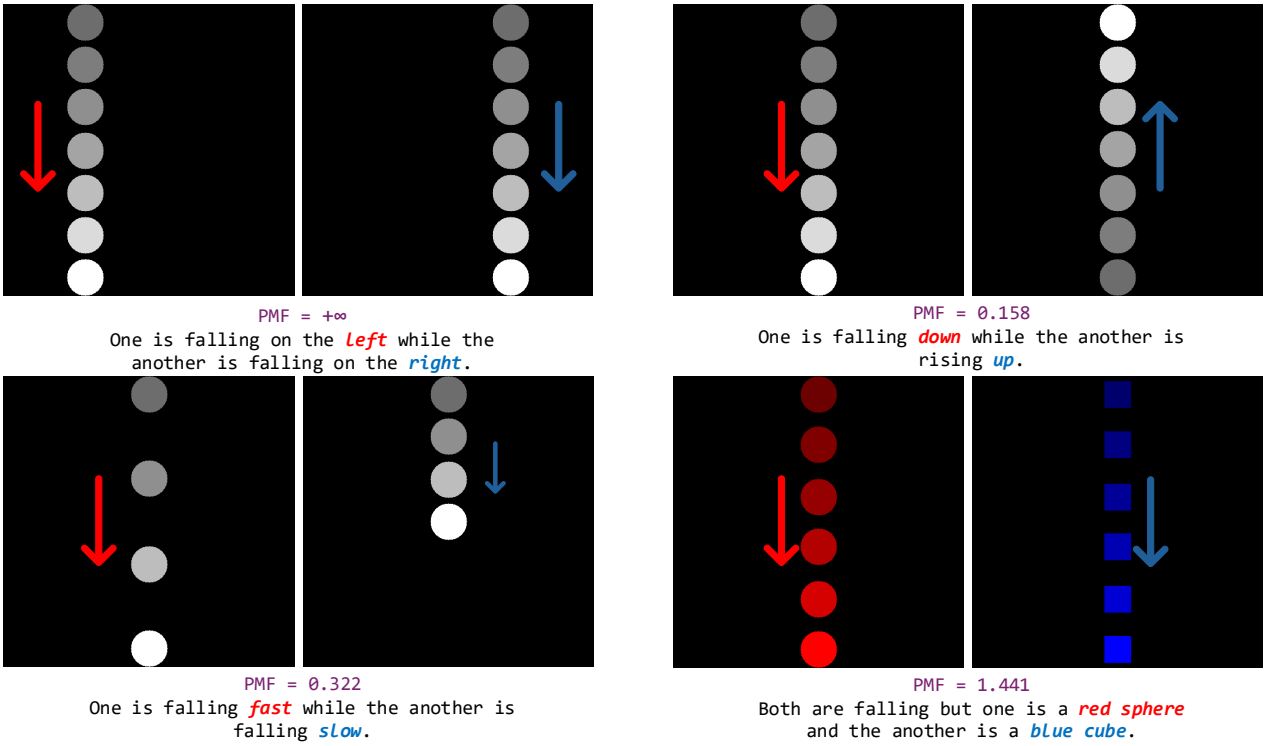

Figure 19. Demonstration for PMF. In the top-left pair, the only variance is the initial spatial location where the object begins to fall. As expected, PMF yields a perfect score, reflecting identical motion patterns regardless of absolute positioning. Conversely, in the top-right pair, the motion is completely reversed (*e.g.*, upward vs. downward fall). The metric degrades severely due to the starkly mismatched spectral energy distributions, correctly indicating poor motion similarity. In the bottom-left case, the initial position is fixed but the velocity is halved. Here, the PMF value increases substantially, effectively distinguishing the altered temporal dynamics through the energy spectrum. Finally, in the bottom-right example, where the object shape differs but the underlying motion dynamics are preserved, the PMF remains relatively high. This confirms that PMF is largely invariant to visual appearance variations and focuses primarily on the consistency of motion dynamics.

5.7.3. Details of Human Rating

To assess the physical plausibility of generated videos from the perspective of human commonsense, we conduct a human evaluation study on 801 AI-generated videos.

We recruit 34 human evaluators to rank the videos within each group according to heuristics that reflect commonsense physical reasoning. Before evaluating any generated videos, each participant is first shown the corresponding reference video to establish a ground-truth baseline for expected motion and object behavior. The heuristics are as follows: (1) **Trajectory Fidelity**: dynamic objects should follow motion paths that closely match the reference video in timing, direction, acceleration, and interaction; (2) **Prompt-Aligned Physics**: the dynamic objects should respect the physical constraints implied by the text prompt and remain generally plausible; (3) **Trajectory Smoothness**: the dynamic object motion should be temporally coherent and free of abrupt, jittery, or kinematically impossible transitions; (4) **Shape Consistency**: object geometry should remain stable over time, without unnatural deformations or flickering; (5) **Semantic Consistency**: object identity and category should be preserved (*e.g.*, a ball should not transform into a bird); and (6) **Object Persistence**: no objects should spontaneously appear, disappear, or duplicate, namely basic conservation of existence must hold.

Videos exhibiting severe physics violations beyond these categories (*e.g.*, unexplained gravity reversal, teleportation, or momentum non-conservation) are ranked lowest unless explicitly justified by the prompt. Human evaluators are asked to provide a strict total ordering within each group (no ties), with the origins of AI models concealed and the presentation order of the generated videos fully randomized to minimize bias. Rankings are converted to scores, where the best rank corresponds to the highest score. The final human evaluation scores for each model are averaged across all groups and participants. Higher scores indicate better physical plausibility as judged by human assessors.

5.8. More Details of Future Frame Prediction

5.8.1. Details of Long-term Prediction

We randomly select 103 dynamic 3D scenes from the test set. Each scene contains 10 seen viewing angles and 2 novel

viewing angles. For each scene, we evaluate the model's extrapolation performance on both seen and novel views using four metrics: PMF, PSNR, SSIM, LPIPS.

For frame-based metrics (PSNR, SSIM, LPIPS), scores are averaged across views and time steps. For PMF, scores are computed based on the predicted future video clip.

For FreeGave[54], we use videos from 10 viewing angles in training, utilizing the first half of each video's length. Initialization is performed by obtaining depth maps and camera poses for the first frame of all 10 views, back-projecting to generate approximately $1120 \times 1120 \times 10$ Gaussians, and randomly sampling 100,000 Gaussians as initialization. The model is optimized for 25,000 iterations. During testing, future frame prediction of the second half of video length is evaluated both on the 10 seen viewpoints and 2 novel viewpoints. DefGS[90], TRACE[53] and TiNeuVox[31] use the same training and testing settings as FreeGave.

ExtDM[98] uses 83,650 videos for training. Stage 1 trains an autoencoder with settings: `max_frame_distance = 20`, `batch size = 256`, `image size = 64×64`, `lr = 2e-4`, for 500,000 iterations. Each iteration samples two frames from a random video with frame distance $\leq$ `max_frame_distance`. Stage 2 trains a diffusion model with settings: `batch size = 64`, `image size = 64×64`, `lr = 2e-4`, `condition frames = 30`, `pred frames = 20`, for 20,000 iterations. Each iteration samples 50 frames: the first 30 as condition frames and the next 20 frames as network predictions. During testing, the last 30 frames of the first half of each video are used as condition frames, and the model autoregressively predicts the second half. Metrics for each scene are averaged over sequences from the 10 seen viewpoints. Note that, ExtDM is a video generation model and cannot evaluate novel-view future frame prediction.

For MAGI-1[1], we use the official pretrained MAGI-1-4.5B-distill model with 16 sampling steps and 32 condition frames. For each test video, the last 32 frames of the first half serve as condition frames, and the model predicts the second half directly. For all 103 test scenes, metrics are averaged across the 10 seen viewpoints.

5.8.2. Details of Continuous Short-term Prediction

We randomly select 103 scenes from validation set. Each scene includes 10 seen viewpoints and 2 novel viewpoints. For every scene, we evaluate the model's future frame prediction performance on both seen and novel viewpoints using four metrics: PMF, PSNR, SSIM, LPIPS. Starting from $t = 0$, at each time step, we incorporate new observations to update the model (or treat them as additional condition frames), then predict the next 10 frames for both seen and novel viewpoints. Metrics are computed for each prediction and averaged across seen views and novel views. If a video contains N frames, we aggregate results over $t = 0, \ldots, N - 10$ and report the mean score for seen and novel viewpoints.

To enable continuous prediction, we modify the training process for FreeGave and DefGS. At each incremental step, the model observes new frames from seen viewpoints, updates its parameters, predicts the next 10 frames, and computes averaged metrics for seen and novel viewpoints. Final scores are obtained by averaging metrics across all incremental steps. For ExtDM and MAGI-1, new observations are treated as additional conditioning frames.

FreeGave: For each scene, we use 10 viewpoints as training views. Initialization follows the Long-term Prediction setup: the first frame is optimized for 3000 iterations using vanilla Gaussian Splatting to reconstruct a static scene. For each new time step, FreeGave updates its velocity field prediction using newly observed frames.

DefGS: Initialization follows the settings in Long-term Prediction: the first frame undergoes 3000 iterations of optimizing vanilla Gaussians. For each new observation, we optimize the deformation field for 50 iterations and Gaussian kernels for 150 iterations. At each incremental step, we predict 10 future frames and compute metrics as the average over 10 seen viewpoints and 2 novel viewpoints.

ExtDM: We directly evaluate the model trained for Long-term Prediction. The number of condition frames is set as 30; if fewer frames are available, the first frame is repeated to reach 30. Only seen viewpoints are evaluated.

MAGI-1: We adopt the same settings used in Long-term Prediction. The number of condition frames is set as 32; if insufficient, the first frame is repeated to reach 32. Only seen viewpoints are evaluated.

5.9. More Details of Physical Properties Estimation

In this experiment, we evaluate two representative baselines, PAC-NeRF [55] and GIC [14], on our dataset. Both methods follow a reconstruction-then-regression strategy: first reconstructing the dynamic 3D scene from multi-view video observations, and then regressing the relevant physical parameters from the temporal evolution of the reconstructed representation, according to those predefined evolution equations in Section 5.4.3.

Specifically, PAC-NeRF learns a neural radiance field comprising a view-independent density field and color field. It then estimates the initial velocity and physical parameters (*e.g.*, viscosity, Young's modulus) via a regression module that analyzes the temporal dynamics of the learned radiance fields. Similarly, GIC reconstructs the full dynamic 3D scene using dynamic 3D Gaussian network and subsequently regresses the physical parameters from the optimized dynamic Gaussian representation.

To evaluate these methods, we select 20 scenes from our test set, ensuring uniform representation across all five material categories (*i.e.*, 4 samples per material). For reconstruction, we use 13 out of the 15 available camera views

as training views, reserving the remaining 2 views as novel views for evaluation. For the two baselines, we provide 90 frames for five materials. We evaluate the two methods on the whole frames in novel-view rendering and resimulation. Since previous datasets for physical properties estimation leverage approximately 20 frames to simulate the whole moving progress, we alternatively downsample the frames to seek the best results for PAC-NeRF and GIC during the training process. In addition, our dataset provides diverse cases with complex boundaries. We also include the corresponding SDF for the boundaries during training and testing. As shown in Table 10, we provide the initial guess for five types of materials, which is identical to the original settings from the two methods.

Table 10. The initial guess of physical parameters estimation for five types of materials on PhysInOne.

	Elastic Solids	Plasticine	Newtonian Fluids	Non-Newtonian Fluids	Granular Substances
PAC-NeRF [55]	$E = 316228$ $\nu = 0.25$	$E = 10000$ $\nu = 0.25$ $\tau_y = 1000$	$\mu = 10$ $\kappa = 1000$	$\mu = 100$ $\kappa = 100000$ $\tau_y = 10$ $\eta = 1$	$\theta_{fric} = 10$
GIC [14]	$E = 316228$ $\nu = 0.25$	$E = 10000$ $\nu = 0.25$ $\tau_y = 1000$	$\mu = 10$ $\kappa = 1000$	$\mu = 100$ $\kappa = 100000$ $\tau_y = 10$ $\eta = 1$	$\theta_{fric} = 10$

To visualize the results, we render the reconstructed scenes from novel views and resimulate the dynamics using the estimated physical parameters. In resimulation, we slightly perturb the initial position or orientation to demonstrate the model's ability to generalize beyond the original configuration.

We report additional quantitative results for novel-view rendering and resimulation in Table 11 and Table 12, respectively, with performance broken down by material types.

Table 11. Additional quantitative results of novel view synthesis based on estimated physical properties.

		PMF ↑	PSNR ↑	SSIM ↑	LPIPS↓
Elastic Solids	PAC-NeRF [55]	5.158	27.909	0.966	0.062
	GIC [14]	**5.570**	**30.143**	**0.968**	**0.057**
Plasticine	PAC-NeRF [55]	5.071	27.429	0.967	0.051
	GIC [14]	**5.535**	**30.191**	**0.971**	**0.044**
Newtonian Fluids	PAC-NeRF [55]	4.201	24.797	0.957	0.061
	GIC [14]	**5.047**	**29.381**	**0.967**	**0.050**
Non-Newtonian Fluids	PAC-NeRF [55]	4.197	24.696	0.959	0.064
	GIC [14]	**4.694**	**27.617**	**0.969**	**0.056**
Granular Substances	PAC-NeRF [55]	**4.289**	24.098	0.859	0.191
	GIC [14]	4.185	**25.033**	**0.863**	**0.180**

Table 12. Additional quantitative results of resimulation based on estimated physical properties.

		PMF ↑	PSNR ↑	SSIM ↑	LPIPS↓
Elastic Solids	PAC-NeRF [55]	4.701	25.463	0.964	0.062
	GIC [14]	**5.136**	**27.886**	**0.967**	**0.055**
Plasticine	PAC-NeRF [55]	4.614	24.850	0.963	0.058
	GIC [14]	**5.206**	**28.332**	**0.969**	**0.047**
Newtonian Fluids	PAC-NeRF [55]	4.166	24.253	0.956	0.061
	GIC [14]	**4.900**	**28.144**	**0.964**	**0.051**
Non-Newtonian Fluids	PAC-NeRF [55]	3.999	22.415	0.946	0.079
	GIC [14]	**4.543**	**25.675**	**0.966**	**0.058**
Granular Substances	PAC-NeRF [55]	4.174	23.615	0.882	0.167
	GIC [14]	**4.202**	**24.477**	**0.884**	**0.157**

5.10. More Details of Motion Transfer

Motion transfer refers to the task of transferring motion dynamics from a reference video to a static image, generating a video that inherits the motion pattern of the reference while preserving the appearance and semantic identity of the object(s) in the given image.

Current approaches typically represent motion through cross-frame pixel correspondences, *i.e.*, optical flow, which encode per-pixel displacement fields across time. For instance, MotionPro[99] employs an optical flow analyzer to extract motion trajectories from the reference video and injects these motion features into an image-to-video generation pipeline. Similarly, GoWithTheFlow[13] introduces a novel noise-warping mechanism that incorporates optical flow directly into the noise initialization stage of the diffusion process, thereby steering the denoising trajectory to replicate the reference motion in the generated output.

In our experiments, we select 273 motion-transfer pairs from our validation set, where the only difference between the reference video and the target video lies in the primary moving object (*e.g.*, different instances of a bouncing ball or falling toy). The condition image is the first frame of the target video. The text prompt, if used, is the one originally paired with the target video in our dataset.