

Boba: Batched Simulation for Physics-Based Gaussian Digital Twins

Yihan Pang¹, Hanxiao Jiang², Sushant Kondguli³, Sarita Adve¹, and Shenlong Wang¹

¹ University of Illinois Urbana-Champaign, Urbana, IL, USA
{yihanp2,sadve,shenlong}@illinois.edu

² Columbia University, New York, NY, USA
hanxiao.jiang@columbia.edu

³ Meta Reality Labs Research, USA
sushant.kondguli@gmail.com

Abstract. Digital twins replicate the appearance and physical behavior of real-world objects for interactive simulation in XR, robotics, and gaming. While neural representations such as 3D Gaussians enable high-fidelity twins reconstructed from images, existing systems simulate only one instance at a time and barely reach real-time performance, making large-scale rollout evaluation for robot learning, model predictive control, and multi-user XR impractical. We present Boba, the first batched simulator for physics-based Gaussian digital twins. Our key idea is to separate the static twin template from the dynamic simulation state and co-design the physics, deformation, and rendering/visualization pipelines for batched execution. This structure enables shared template computation across instances while minimizing memory traffic and synchronization overhead. Guided by this principle, we introduce compact surrogate spring-mass models and a set of system-level optimizations—including mixed-precision Gaussian skinning, memory-efficient execution, and shared-memory-aware batching—that significantly reduce compute and memory costs. On an XR-configured Jetson Orin, Boba’s local configuration achieves over $10\times$ single-instance speedup over the prior state-of-the-art PHYSTWIN pipeline. Its distributed configuration, split between the Orin and a server, reaches 25.1 ms end-to-end latency, $13.2\times$ lower than the same PHYSTWIN baseline on Orin, while lowering incremental power by 22.2% relative to Boba’s local configuration. For batched execution, Boba scales to 3,310 FPS on RTX 4090, a $16.2\times$ system-level aggregate-throughput improvement over Boba’s optimized single-instance configuration on the same GPU, and accelerates predictive control for deformable object manipulation, achieving speedups from $26\times$ to an estimated $> 2,000\times$ relative to PHYSTWIN-based planning. These results show that Boba can support power-constrained XR applications and scalable robot planning.

1 Introduction

Digital twins aim to replicate both the appearance and physical behavior of real-world objects, enabling interactive simulation for extended reality (XR), robotics,

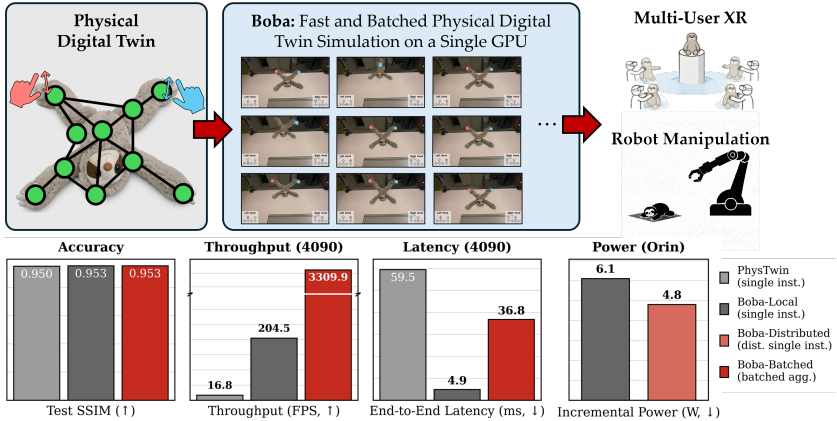


Fig. 1: Boba: Efficient and high-throughput physics-based Gaussian digital twin simulation. By co-designing physics, deformation, and rendering/visualization, Boba preserves fidelity, provides over $10\times$ single-instance speedup relative to the prior state-of-the-art PhysTwin pipeline, and reaches 3,310 FPS average aggregate throughput on 4090. For XR deployment, BOBA-DISTRIBUTED lowers the Jetson Orin’s incremental power by 22.2% relative to BOBA-LOCAL. Together, Boba’s configurations support applications such as multi-user XR and robot manipulation.

and gaming. Recent neural representations such as 3D Gaussians allow high-fidelity digital twins to be reconstructed directly from images, and systems such as PhysGaussian, VR-GS, and PhysTwin [8, 9, 28] couple these representations with deformable physics simulation. However, these pipelines simulate only one instance at a time and barely reach real-time rates, making interactive XR use difficult and limiting applications that require many parallel rollouts, such as robot learning, model predictive control, and multi-user XR systems.

A promising direction to address this limitation is **batched simulation**, where many simulation instances share model parameters and assets while executing in parallel on a GPU. Prior works [5, 14, 15, 20, 23, 24, 27] show that co-designing simulation algorithms and system infrastructure can dramatically increase throughput by executing many environments simultaneously with shared memory and vectorized computation. This paradigm has proven highly effective for reinforcement learning workloads and other rollout-heavy workloads.

However, existing batched simulators primarily target rigid-body physics with lightweight rendering. By contrast, physics-based neural digital twins require substantially more complex pipelines, including deformable dynamics, dense interactions, and neural rendering tightly coupled with simulation. These characteristics introduce large memory footprints, high memory traffic, and synchronization overhead, making straightforward batching inefficient.

In this work, we introduce Boba (Fig. 1), the first batched simulation framework for physics-based Gaussian digital twins, built on top of the PhysTwin representation. Boba instantiates its core optimized pipeline in three configurations: BOBA-LOCAL for optimized local execution, BOBA-DISTRIBUTED for power- and latency-constrained XR deployment with an edge-server split, and

BOBA-BATCHED for high-throughput rollouts. Our key idea is to enable scalable neural digital twin simulation through **template–state separation** and system co-design. The static twin template, such as spring topology, rest geometry, and skinning metadata, is shared across time and across simulation instances, while only dynamic state updates change during execution. This structure allows simulation, deformation, and rendering to be **co-designed for batched execution**, reducing redundant computation, memory traffic, and synchronization overhead. Fig. 2 overviews Boba’s design. Guided by this principle, we address both algorithmic and systems-level bottlenecks. We first introduce compact surrogate spring-mass models that reproduce the behavior of much denser systems while reducing memory and compute by an order of magnitude. We further accelerate Gaussian deformation and rendering through mixed-precision skinning, memory-traffic-aware execution, and fused GPU operations. Finally, we redesign the batched simulation with shared memory layouts and contention-free parallel scheduling for efficient large-scale deformable-twin simulation.

Together, these optimizations enable fast local execution, latency- and power-efficient distributed XR deployment, and high-throughput batched execution. On a ~ 15 W embedded GPU, BOBA-LOCAL brings PhysTwin-style simulation to real-time performance within an XR-class edge-device power budget. For power- and latency-constrained XR deployment, BOBA-DISTRIBUTED reduces latency and incremental power relative to BOBA-LOCAL, leaving more edge-device headroom for richer XR applications and longer device use. Separately, on RTX 4090, BOBA-LOCAL provides the same-GPU reference for BOBA-BATCHED, whose higher throughput accelerates deformable-object predictive control.

Our main contributions are threefold: **(1)** We introduce the **first batched simulator for physics-based Gaussian digital twins**, enabling many episodes of the same reconstructed object to run efficiently on a single GPU. **(2)** We propose a **model–system co-design** for scalable twin simulation, including compact surrogate spring-mass models, efficient Gaussian deformation and rendering/visualization, and shared-memory-aware batching that reduces synchronization and memory bottlenecks. **(3)** We demonstrate system impact across XR deployment, batching, and robot planning: BOBA-LOCAL achieves over $10\times$ **single-instance speedup over PHYSTWIN** across embedded and desktop GPUs, BOBA-DISTRIBUTED achieves a $13.2\times$ **latency improvement** over PHYSTWIN on Orin with 22.2% **lower incremental power** than BOBA-LOCAL, BOBA-BATCHED achieves **3,310 FPS** aggregate throughput in batched execution on RTX 4090 (Fig. 3), and accelerates PHYSTWIN-based predictive control, including a $26\times$ speedup on rope and an estimated $> 2,000\times$ speedup on cloth with matched self-collision.

2 Related Work

Interactive Physical Digital Twins. The integration of neural scene representations with physics simulation has enabled digital twins that jointly model photorealistic appearance and physically consistent dynamics. Prior work commonly

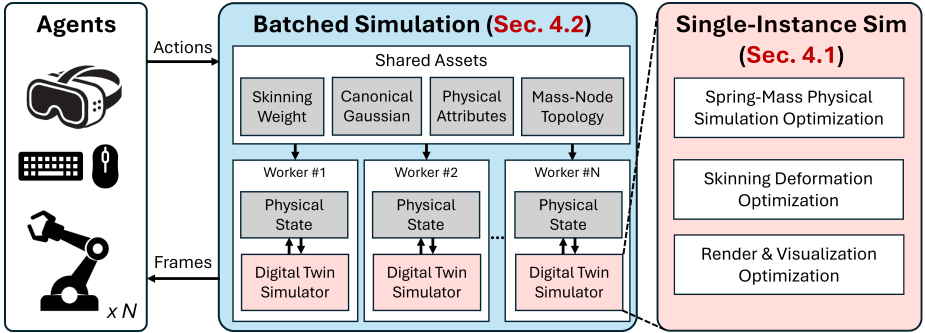


Fig. 2: Overview of Boba. Boba separates shared twin templates from per-instance dynamic state to enable high-throughput physics-based Gaussian digital twin simulation. Static assets are shared across workers, while each worker maintains its own physical state. The optimized single-instance pipeline serves as the base, and the batched framework adapts it across N independent episodes on a single GPU. For clarity, the figure shows Boba’s high-level organization and abstracts per-stage optimizations and the distributed communication path, which are detailed in the text and supplement.

adopts neural 3D parameterizations, including NeRF-based models [1, 3, 10] and Gaussian Splatting representations [9, 28, 31, 32], to build visually faithful and simulation-ready assets. Within this line of research, some methods [3, 9, 28] rely on hand-crafted or manually tuned physical parameters, which can introduce a gap between simulated responses and real-world motion. Other approaches [1, 10, 19, 31, 32] attempt to recover physical properties from video observations, yet often depend on synthetic data, limited excitation patterns, or dense multi-view geometry, limiting robustness and generality.

Recent progress has improved practical applicability. PhysTwin [8] represents a pivotal advance by learning physical digital twins directly from real human–object interaction videos. It integrates a spring-mass dynamics model with Gaussian Splatting for appearance reconstruction, enabling interactive simulation and subsequent robotics applications [30]. Follow-up works [2, 29] extend this paradigm to broader categories of physical effects, further validating the effectiveness of physics-grounded digital twins.

Nevertheless, prior works focus primarily on physical digital twin construction rather than computational scalability. They do not target low-power edge deployment or parallel simulation for MPC and RL. In contrast, we build upon PhysTwin with Boba, a multi-configuration system that targets different points in the throughput-latency-power trade-off, shifting the focus from fidelity alone to practical efficiency for XR and robotics.

Parallel Simulation Systems. High-throughput simulation has long been a focus in physics engine design, where runtime efficiency directly affects scalability and usability. Early GPU-centric engines, including Brax [4], Isaac Gym [13], and MuJoCo XLA (MJX) [26], introduced automatic batching, JIT compilation, and GPU-native execution for rigid-body physics. Later systems, such as Isaac Lab [14], ManiSkill3 [5, 15, 23, 27], Genesis [24], Madrona [20], and NVIDIA Newton [25], extended these ideas to modular, parallel, and differentiable workloads.

Although these systems achieve impressive throughput for rigid-body and modular primitives, they are not designed for batched deformable neural digital twins. Such simulations involve tightly coupled physics-based dynamics and the propagation of motion to large numbers of neural rendering primitives (e.g., Gaussians), introducing memory, synchronization, and numerical challenges beyond traditional engines. Building on PhysTwin [8], our work extends parallel simulation to this regime, supporting both energy-efficient single-instance execution and high-throughput batch rollouts.

3 Preliminaries: Physical Digital Twins

Physical digital twins aim to model real-world objects that both *look realistic* and *behave physically correctly* under interaction, enabling interactive simulation for XR, robotics, and gaming. Unlike static 3D reconstruction, a physical twin must support *dynamics*, *deformation*, and *rendering/visualization* in a unified pipeline. Modern neural twin systems therefore combine a physics simulator, a deformation model, and a differentiable renderer. The same differentiable pipeline is also used for inverse simulation, where observed interactions are used to estimate the twin state and parameters. We use recent *Gaussian-based* twins [8, 9, 28] as a concrete example to introduce notation.

Overview. A physical digital twin consists of two coupled representations. The *physical state* is $\mathbf{x}_t = \{\mathbf{x}_{t,i}\}_{i=1}^N$ with $\mathbf{x}_{t,i} \in \mathbb{R}^3$, describing the positions of physical elements at time t . Each element also carries a set of *physical attributes* $\boldsymbol{\theta}_i$ (e.g., mass, elasticity, damping, or other material parameters) that determine its dynamic behavior. The *visual representation* is a set of G 3D Gaussians with geometry and appearance parameters $\mathbf{g} = \{(\boldsymbol{\mu}_g, \Sigma_g, c_g, \alpha_g)\}_{g=1}^G$, where $\boldsymbol{\mu}_g$ is the Gaussian center, Σ_g encodes its anisotropic shape, c_g denotes color or appearance coefficients, and α_g denotes opacity. A skinning relation $\mathbf{w} = \{w_{ig}\}$ couples the two representations, allowing simulated motion to deform the Gaussians. Interaction is represented by an input $\mathbf{u}_t$, which may encode user-specified forces or constraints in XR, or action commands from an agent in robotics. The twin is executed as an unrolled process over time. At each step, the system advances the physical state using $\mathbf{u}_t$ over a simulation substep Δt , deforms the Gaussians, and renders an image. We use semicolons to separate dynamic state and input variables from fixed twin, rendering, and integration parameters:

$$\hat{I}_{t+1}(\mathbf{x}_t, \mathbf{u}_t; \boldsymbol{\theta}, \mathbf{w}, \mathbf{g}, \pi, \Delta t) = \text{Render}(\text{Deform}(\text{Sim}(\mathbf{x}_t, \mathbf{u}_t; \boldsymbol{\theta}, \Delta t), \mathbf{w}, \mathbf{g}), \pi),$$

where π denotes camera parameters, Sim advances the physical state from $\mathbf{x}_t$ given interaction $\mathbf{u}_t$, physical parameters $\boldsymbol{\theta} = \boldsymbol{\theta}_i$, and substep Δt , Deform maps the state to Gaussian geometry using $\mathbf{w}$ and $\mathbf{g}$, and Render produces an image.

Dynamical Simulation. The simulator maintains discrete physical states $\mathbf{x}_t$, while the underlying object dynamics are governed by a deformable model. A generic form is $m_i \ddot{\mathbf{x}}_i(\tau) = \mathbf{f}_i^{\text{int}}(\mathbf{x}(\tau), \dot{\mathbf{x}}(\tau); \boldsymbol{\theta}) + \mathbf{f}_i^{\text{ext}}(\mathbf{u}(\tau))$, where τ denotes continuous time, $\mathbf{f}^{\text{int}}$ models deformation forces determined by the physical parameters $\boldsymbol{\theta}$,

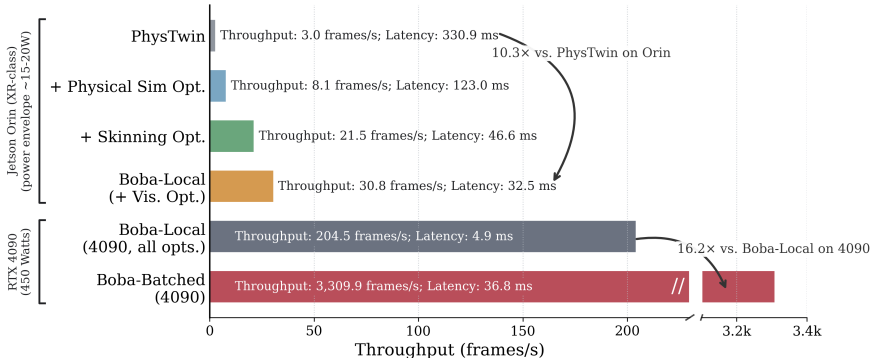


Fig. 3: System performance breakdown. Boba progressively accelerates single-instance execution on Orin, with BOBA-LOCAL achieving a 10.3× speedup over PHYS-TWIN. On RTX 4090, BOBA-LOCAL serves as the same-GPU single-instance baseline for BOBA-BATCHED; BOBA-BATCHED reaches 3,310 FPS average aggregate throughput, a 16.2× improvement over this baseline.

and $\mathbf{f}^{\text{ext}}$ represents external interaction effects induced by the input. The internal model may follow Spring-Mass, FEM, PBD, or MPM [7, 11, 12, 16, 21, 22]; discretizing and integrating with a substep Δt yields $\mathbf{x}_{t+1} = \text{Sim}(\mathbf{x}_t, \mathbf{u}_t; \boldsymbol{\theta}; \Delta t)$.

Deformation. The deformation stage maps the simulated state to the visual representation. For Gaussian twins, simulated motion is transferred to Gaussian parameters through linear blend skinning, which we write abstractly as $\mathbf{g}_{t+1} = \text{Deform}(\mathbf{x}_{t+1}, \mathbf{w}, \mathbf{g})$, where $\mathbf{x}_{t+1} = \text{Sim}(\mathbf{x}_t, \mathbf{u}_t; \boldsymbol{\theta}; \Delta t)$ and $\mathbf{g}_{t+1}$ denotes the deformed Gaussian parameters at time $t + 1$.

Rendering. Images are rendered via splatting by compositing contributions from deformed Gaussians along viewing rays, which we denote as $\text{Render}(\cdot)$.

Inverse Simulation. Given paired observations and interactions $\{(I_t, \mathbf{u}_t)\}_{t=1}^T$, inverse simulation estimates the state trajectory and twin parameters $(\mathbf{x}_{1:T}, \boldsymbol{\theta}, \mathbf{w}, \mathbf{g}, \pi)$ by minimizing $\sum_{t=1}^T \|I_t - \hat{I}_t\|$. Gradients propagate through Render , Deform , and Sim , enabling joint estimation of geometry, appearance, and physical parameters. While this pipeline enables realistic digital twins, the tightly coupled simulation, deformation, and rendering stages introduce substantial compute and memory overhead, motivating scalable simulation methods.

4 Method

Boba is a scalable framework for physics-based Gaussian digital twins of deformable objects, supporting latency- and power-constrained single-instance execution, either locally or with an edge-server split, and high-throughput batched simulation. Our key idea is to restructure and optimize the simulation pipeline around *template-state separation*, where the static twin template, such as spring topology, physical attributes, rest geometry, and skinning metadata, is shared across time and across simulation instances, while only dynamic state variables

change during execution. This design reduces redundant computation, memory traffic, and synchronization overhead that arise when directly scaling the PhysTwin pipeline, where spring-mass simulation and Gaussian deformation introduce substantial per-instance compute and data-movement costs. Guided by this idea, Boba introduces coordinated optimizations across the pipeline, including efficient spring-mass simulation, accelerated Gaussian deformation, optimized rendering/visualization and communication, and batching-aware execution kernels.

We first describe single-instance optimizations for spring-mass simulation and Gaussian deformation, together with the system-level optimizations that support local and distributed execution (Sec. 4.1). We then introduce batching-aware data layouts and kernels for high-throughput execution across many digital twin instances (Sec. 4.2). Detailed rendering/visualization and communication optimizations are provided in the supplementary material. Fig. 2 overviews Boba.

4.1 Single-Instance Efficient Twin Simulation

We first optimize the PhysTwin pipeline for efficient single-instance execution while preserving physical fidelity. Our goal is to reduce per-step compute and memory traffic so that a single digital twin can run interactively under tight latency and power budgets. Observing that deformable simulation exhibits both redundancy and modular computation, we organize Boba’s single-instance optimizations into three groups: (1) spring-mass simulation acceleration through spatiotemporal coarsening, rest-state collision pruning, and locality-aware execution, (2) efficient Gaussian deformation updates through faster rotation extraction, sparse skinning, and mixed-precision computation, and (3) system-level rendering/visualization and communication optimizations that reduce post-deformation overhead and enable BOBA-DISTRIBUTED’s edge-server path. We describe the first two groups in detail below because they form the shared simulation/deformation path used across Boba’s local, distributed, and batched configurations. We summarize the third group in this section, with additional details provided in the supplement.

Spring-Mass Optimization Boba reduces spring-mass computation with spatiotemporal coarsening, rest-state collision pruning, and locality-aware execution. Fig. 4 shows the spatiotemporal coarsening and rest-state collision-pruning steps.

Spatiotemporal Coarsening. We start by reducing the cost of deformable simulation through spatial and temporal coarsening of the spring-mass system. This opportunity arises from redundancy in deformable dynamics: nearby nodes often exhibit correlated motion and similar physical attributes, and deformation observed at the Gaussian level is often insensitive to small variations in node density or integration step. Let $(\mathbf{x}_t, \boldsymbol{\theta}, \mathbf{w}, \mathbf{g}, \Delta t)$ denote the original PhysTwin system, where $\mathbf{x}_t$ stores the spring-mass node positions. Our goal is to construct a compact spring-mass system $(\mathbf{x}'_t, \boldsymbol{\theta}', \mathbf{w}', \mathbf{g}, \Delta t')$ with significantly fewer nodes ($M \ll N$) and a coarser integration step $\Delta t' > \Delta t$ such that the resulting deformation remains visually consistent with the original model. Reducing the node count shrinks the dynamic state from $O(N)$ to $O(M)$ and correspondingly

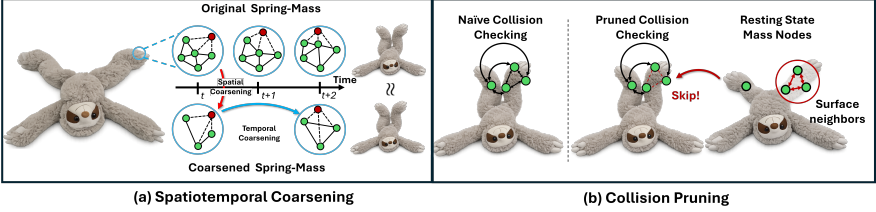


Fig. 4: Spring-mass optimization. (a) **Spatiotemporal coarsening** compresses the spring-mass system by reducing node resolution and simulation steps while preserving the resulting Gaussian deformation. (b) **Rest-state collision pruning** skips collision computations for node pairs that are nearby in rest space.

reduces spring-force evaluation and skinning computation. We therefore perform spatiotemporal coarsening by jointly reducing the spatial resolution M of the spring graph and increasing the simulation timestep to $\Delta t'$. The compact system parameters $(\theta', \mathbf{w}')$ are refit so that the resulting Gaussian trajectories match those of the original model. In practice, we optimize directly on rendered rollouts to ensure that visual behavior remains consistent with the original system.

Rest-state collision pruning. We then exploit locality to reduce unnecessary collision computation. Self-collision detection often dominates runtime in deformable simulation because many node pairs must be tested for potential interaction. Unlike rigid-body simulation, where convex proxies or bounding volumes can cheaply cull collisions, deformable surfaces lack stable proxies that remain valid under large deformation. We observe that nodes that are close in the rest configuration typically correspond to structural adjacency on the object and rarely produce meaningful collision impulses during simulation. We therefore precompute a symmetric rest-space pruning mask over node pairs. A pair (i, j) is flagged if its rest-space distance is below αd_{col} , where d_{col} is the collision distance and $\alpha > 1$ is a safety margin. During simulation, the collision kernel consults this mask and skips impulse computation for flagged pairs. The same pruning rule is applied during training and runtime to avoid train-runtime mismatch.

Locality-aware node ordering. We improve memory efficiency by exploiting spatial locality in the spring-mass system. Spring-force evaluation is often memory-bound because each spring references two endpoint nodes whose positions and velocities may be far apart in memory, leading to scattered global-memory reads on the GPU. To improve locality, we reorder nodes using a Morton (Z-order) key computed from their rest-space positions, which maps nearby 3D coordinates to nearby memory indices. This produces a permuted state vector $\tilde{\mathbf{x}} = P\mathbf{x}$ in which spatially nearby nodes occupy contiguous memory locations. We remap all spring endpoints to the reordered node indices and group springs into spatial blocks for evaluation. This permutation does not change the algorithmic complexity of spring-force computation, but it significantly improves memory-coalescing behavior and cache efficiency during GPU execution. Because the access pattern is determined by spring endpoint indices rather than the nodes' current positions, this preprocessing is performed only once and reused throughout simulation.

Skinning Optimization Boba accelerates skinning, where mass-node motion is propagated to Gaussian primitives via linear blend skinning (LBS). Each node needs a rigid transform, with translation from the spring-mass update and rotation from local deformation. At the scale of thousands of nodes and hundreds of thousands of Gaussians, LBS is bottlenecked by rotation extraction and memory traffic. Boba addresses these bottlenecks with three optimizations: rotation extraction with temporal reuse, sparse skinning with bounded influence sets, and mixed-precision LBS.

Rotation extraction with temporal reuse. The standard approach estimates node rotations using polar decomposition via an SVD of the 3×3 deformation matrix $\mathbf{F}$. While mathematically lightweight, general-purpose SVD implementations can incur substantial memory overhead because they materialize working copies and allocate temporary buffers, which becomes a bottleneck when executed for many nodes and batched instances. Boba replaces the SVD with a rotation-only solver based on the polar-decomposition identity. We compute $\mathbf{C} = \mathbf{F}^\top \mathbf{F}$, obtain $\mathbf{C}^{-1/2}$ through symmetric eigendecomposition with eigenvalue clamping, and recover the rotation as $\mathbf{R} = \mathbf{F}\mathbf{C}^{-1/2}$. The solver is implemented as a fused GPU kernel that reuses neighborhood data already loaded in memory and writes results directly in quaternion form. We further exploit temporal coherence by reusing rotations across simulation substeps. For each node, we cache the previous deformation matrix and rotation, and recompute the rotation only when the Frobenius-norm change $\|\mathbf{F} - \mathbf{F}_{\text{cache}}\|_F$ exceeds a threshold. This avoids redundant rotation extraction when local deformation changes minimally.

Sparse and mixed-precision skinning. We reduce the cost of Gaussian skinning by exploiting both sparsity and mixed precision in linear blend skinning (LBS). First, we limit the number of node influences used for each Gaussian. Since deformation is typically smooth at the Gaussian scale, blending transforms from a large number of mass nodes yields diminishing returns. Boba therefore selects the K_{lbs} nearest nodes in rest space and computes normalized skinning weights on this reduced set, significantly lowering both computation and memory traffic. Second, we adopt a mixed-precision implementation of LBS to reduce memory bandwidth. Cached node rotations and rest-pose skinning quantities are stored in FP16, while numerically sensitive operations—including deformation matrix construction, the rotation reuse test, and eigen-based rotation extraction—remain in FP32 for stability. Translation blending is also performed in FP32 to avoid accumulated rounding errors. The final Gaussian poses are cast back to FP32 before rendering to maintain compatibility with the Gaussian splatting renderer.

Rendering/visualization and distributed deployment. Boba also optimizes the post-deformation system path used to render and deliver updated twin states. In the local and distributed single-instance configurations, Boba keeps frame compositing and display transfer on the GPU where possible, reducing CPU-GPU synchronization and memory copies. In the distributed configuration, static template data are cached on both devices, and only compact dynamic Gaussian pose

updates are transmitted from the server to the edge device. These optimizations reduce rendering/visualization and communication overhead without changing the simulated state. Additional details are in the supplement.

4.2 High-Throughput Batched Digital Twin Simulation

Boba supports a throughput-oriented setting where many digital twin instances are simulated concurrently. Prior batched simulators primarily target rigid-body environments with low-dimensional per-episode state. In contrast, neural digital twins maintain high-dimensional deformation, geometry, and rendering state, making straightforward per-instance replication prohibitively expensive in memory and bandwidth. Moreover, scatter-heavy physics kernels suffer from severe atomic contention when executed across many instances. Boba addresses these challenges through a batched design that separates shared templates from dynamic state and co-designs force assembly for parallel execution.

Template-state separation. We adopt the template-state decomposition common in parallel simulation systems [20, 26] and extend it to deformable Gaussian digital twins. The shared template stores static data including spring topology, rest geometry, skinning metadata, Gaussian parameters, and rest-derived precomputations. Per-instance dynamic state—such as node positions, velocities, and cached rotations—is packed into contiguous batched arrays. GPU kernels index instance-specific segments through offsets while reusing the shared template. This avoids duplicating large static metadata across instances and enables efficient batched simulation with modest memory growth.

Conflict-free force accumulation. Spring-mass dynamics require accumulating forces from all incident springs at each node. In batched settings, naive implementations rely on atomic additions because many incident springs may concurrently add force contributions to the same node within each instance; repeated across many instances, these atomics become a major bottleneck. Boba eliminates atomics by decoupling force computation from force accumulation. First, a spring-parallel kernel computes each spring force once and stores it in a temporary buffer F_{temp} . Second, a node-parallel kernel gathers the forces of its incident springs using a precomputed adjacency map and accumulates them locally. Each adjacency entry stores a signed spring index indicating whether the node adds or subtracts that spring’s contribution. This gather-based assembly eliminates atomic contention and allows force accumulation to scale with node degree rather than the total number of springs. Combined with template sharing, the design enables high-throughput batched simulation of deformable digital twins.

5 Experiments

We evaluate Boba through three questions: **(1) Efficiency:** *Can it accelerate single-instance physics-based Gaussian digital twin simulation while preserving physical and visual fidelity, and can distributed execution further reduce latency*

and incremental power for XR-class deployment? **(2) Scalability:** Can it scale deformable digital twin simulation to large batch sizes with high GPU efficiency, avoiding bottlenecks such as atomic contention? **(3) System Impact:** Can it support high-throughput robot-control rollouts and real-time XR interaction under tight latency and power budgets? We build on PhysTwin as the base deformable twin model, but Boba’s optimizations, including spatiotemporal coarsening, efficient Gaussian deformation, and batching-aware simulation kernels, generalize to a broader class of physics-based Gaussian digital twins.

5.1 Experiment Settings

Hardware and network. We evaluate Boba on two hardware classes: an XR-class edge device and a desktop-class GPU system. For the edge device, we use an NVIDIA Jetson Orin [18], configured to emulate an XR-class compute budget by restricting CPU/GPU compute resources; we refer to this device as *Orin*. For the desktop-class GPU system, we use a workstation with an Intel i9 CPU [6] and an NVIDIA RTX 4090 GPU [17], which we refer to as *4090*. In the distributed setting, Orin serves as the edge device and the 4090 system serves as the server; Orin connects over Wi-Fi and the server connects via Ethernet. Full hardware and network details are in the supplement.

Evaluated Capabilities. We evaluate BOBA under two capabilities: single-instance execution and batched execution.

Single-instance execution. We evaluate three single-instance deployment configurations: (i) PHYSTWIN, which runs the original pipeline locally and is evaluated on both Orin and 4090; (ii) BOBA-LOCAL, which runs Boba’s optimized single-instance pipeline locally on both Orin and 4090 without batching-specific optimizations; and (iii) BOBA-DISTRIBUTED, Boba’s edge-server variant, with Orin as the edge device and 4090 as the server. Both BOBA-LOCAL and BOBA-DISTRIBUTED use an optimized rendering/visualization pipeline, and BOBA-DISTRIBUTED additionally uses communication-specific optimizations. Details of the communication and rendering/visualization are provided in the supplementary material.

Batched-instance execution. This scenario targets throughput-oriented execution where the server executes many instances concurrently. We compare: (i) PHYSTWIN-MULTIPROC, which scales by launching one independent single-instance process per simulated instance, with each process maintaining its own pipeline state and memory allocations; and (ii) BOBA-BATCHED, which executes multiple instances within a single process using Boba’s single-instance optimization stack together with batched-only extensions for shared state, conflict-free force accumulation, and batch-aware rendering/visualization. The batch-aware rendering/visualization path is detailed and evaluated in the supplement.

Datasets & Metrics. We use the same objects as PhysTwin [8]. For **single-instance evaluation**, we use the action sequences provided by PhysTwin. We report *end-to-end latency*; for the XR local/distributed setting, we additionally

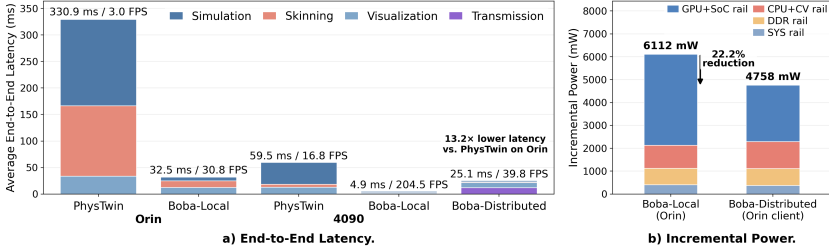


Fig. 5: Latency and incremental power across single-instance deployment settings.

report *incremental power*, measured on the Orin as workload-induced power above the idle baseline; server-side power is not included. For local configurations, latency includes simulation, skinning, and rendering/visualization on the target device. For BOBA-DISTRIBUTED, latency includes server-side simulation/skinning, transmission, and edge-side rendering/visualization. Power measurement details are provided in the supplementary material. For **single-instance quality evaluation**, we use the two quality tasks from PhysTwin: reconstruction & resimulation and future prediction. We report 3D metrics using single-direction Chamfer Distance and tracking error against manually annotated ground-truth points from PHYSTWIN, and 2D metrics using PSNR, SSIM, LPIPS, and silhouette IoU, evaluated at the center viewpoint and averaged across frames and cases.

For **batched-instance evaluation**, actions are generated using a random interaction generator that mimics human manipulation. Each batch element is an independent digital-twin instance with its own state and action. We report *aggregate throughput*, *per-step latency*, and *maximum batch capacity*. A batch step denotes applying one action input to each of the N instances, advancing simulation and deformation/skinning, and producing one rendered/visualized frame per instance; per-step latency is its wall-clock time. Aggregate throughput is total FPS across all N instances. For each case, we sweep batch size, measure aggregate FPS over the full run at each size, report the maximum, and average these maxima across cases. Maximum batch capacity is the largest feasible batch size for each case and batched implementation.

5.2 Single-Instance Results

Latency. Fig. 5a shows end-to-end latency across configurations. On Orin, BOBA-LOCAL reduces latency from 330.9 ms for PHYSTWIN to 32.5 ms (10.2 $\times$). On 4090, the same optimization stack reduces latency from 59.5 ms to 4.9 ms (12.1 $\times$), indicating that our local optimizations transfer across platforms. BOBA-DISTRIBUTED offloads simulation and skinning to the server while the edge applies the streamed updates and performs rendering/visualization, reducing latency to 25.1 ms. This is a further 1.3 $\times$ improvement over BOBA-LOCAL on Orin and a 13.2 $\times$ end-to-end improvement over PHYSTWIN on Orin, despite communication.

Power. Fig. 5b shows incremental power for BOBA-LOCAL and BOBA-DISTRIBUTED. BOBA-DISTRIBUTED reduces this power compared with BOBA-LOCAL on Orin, from 6112 mW to 4758 mW (22.2%). The largest measured reduction occurs on

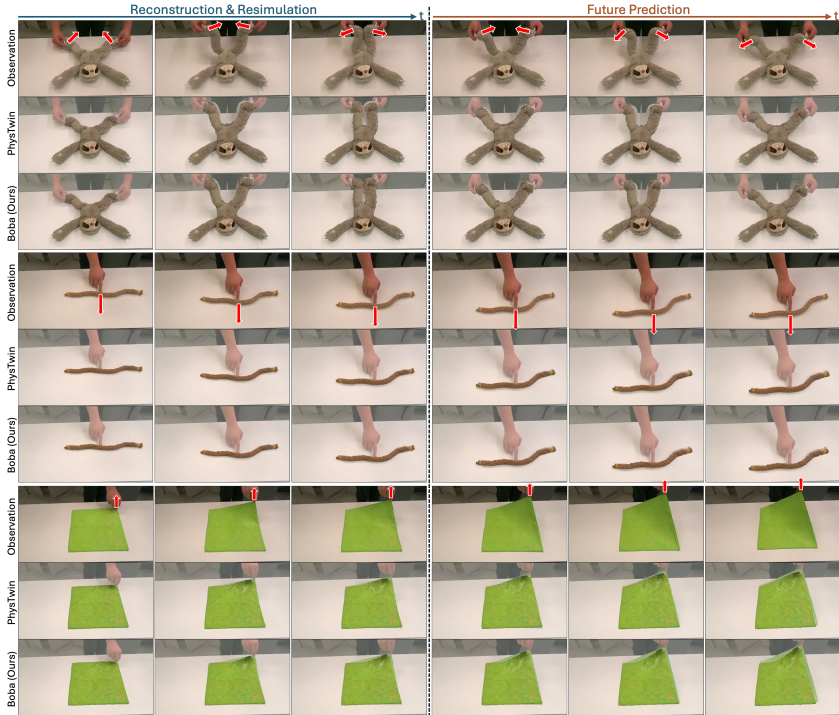


Fig. 6: Visual fidelity. Boba closely matches the PHYSTWIN baseline across reconstruction/resimulation and future prediction, with real observations shown as reference. Results use BOBA-LOCAL; local hardware affects performance but not the visual output.

Orin’s combined GPU+SoC rail, which reports the GPU and on-chip system logic together. Since Boba’s simulation and skinning stages are GPU-heavy, this reduction is consistent with moving these stages off the edge device, although the shared rail does not let us attribute the reduction to GPU and SoC components separately. Other reported rails, including CPU, memory, and system power, change more modestly, indicating that the remaining communication and state-reconstruction work does not offset the compute-side power savings. Overall, BOBA-DISTRIBUTED reduces latency while lowering device-side power, leaving more on-device compute headroom for concurrent XR tasks.

Fidelity. Fig. 6 and Table 1 summarize visual comparisons and objective fidelity metrics. Visually, Boba closely matches PHYSTWIN across reconstruction & resimulation and future prediction while preserving consistent dynamics. Across objective metrics, BOBA-LOCAL closely matches PHYSTWIN in both 3D and image-space metrics, indicating that our compute optimizations do not materially degrade physical or perceptual fidelity. BOBA-DISTRIBUTED, our distributed deployment design, remains close to BOBA-LOCAL across both reconstruction & resimulation and future prediction. This indicates that its communication and rendering/visualization pipeline introduces no measurable fidelity loss while enabling lower-latency and lower-power deployment.

Table 1: Quantitative results on reconstruction & resimulation and future prediction.

Task Method	Reconstruction & Resimulation						Future Prediction					
	CD↓	Track↓	IoU↑	PSNR↑	SSIM↑	LPIPS↓	CD↓	Track↓	IoU↑	PSNR↑	SSIM↑	LPIPS↓
PhysTwin	0.005	0.009	87.4	28.7	0.966	0.025	0.012	0.022	71.4	23.3	0.950	0.049
Boba-Local/Boba-Batched	0.006	0.009	89.6	28.4	0.963	0.031	0.014	0.022	75.5	23.6	0.953	0.050
Boba-Distributed	0.006	0.009	90.2	28.6	0.964	0.030	0.014	0.022	75.6	23.7	0.953	0.050

Table 2: Batch capacity and throughput by parallelization strategy.

Method	Avg. Batch Capacity	Min Batch Capacity	Max Batch Capacity	Throughput (FPS)
PhysTwin-MultiProc	11.0	4	27	44.7
Boba-Batched (Template-share only)	369.2	213	688	2392.8
Boba-Batched (+ Atomic-free)	368.3	210	688	3309.9

5.3 Batch Scalability and Ablations

Batch scalability. As shown in Table 2, batching-specific optimizations substantially improve scalability and throughput. Relative to PHYSTWIN-MULTIPROC, template sharing alone increases average batch capacity from 11.0 to 369.2 (33.6 $\times$) and improves throughput from 44.7 to 2392.8 frames/s (53.5 $\times$). Adding atomic-free scheduling on top yields the full BOBA-BATCHED design: capacity remains nearly unchanged, with average capacity changing only from 369.2 to 368.3 and maximum capacity remaining 688, while increasing throughput to 3309.9 frames/s, a further 38.3% improvement over template sharing alone. The slight capacity decrease comes from the extra index mapping required by the atomic-free path, but this small memory cost is outweighed by the throughput gain. Overall, template sharing drives capacity gains, while atomic-free scheduling further improves throughput by avoiding atomic contention.

Compute and Batching Breakdown. Fig. 3 summarizes local optimization results on Orin and 4090 and batched execution on 4090 in throughput and latency. We exclude BOBA-DISTRIBUTED to isolate local and batched execution from edge-server communication; detailed per-optimization results are in the supplement. On Orin, PHYSTWIN runs at 3.0 frames/s; physical simulation, skinning, and rendering/visualization optimizations raise throughput to 8.1, 21.5, and 30.8 frames/s, yielding a 10.3 $\times$ end-to-end speedup for BOBA-LOCAL. On 4090, the same local stack improves PHYSTWIN from 16.8 to 204.5 frames/s, providing the optimized same-GPU single-instance reference. BOBA-BATCHED adds batching-specific execution and the throughput-oriented batched rendering/visualization path detailed in the supplement, reaching 3309.9 aggregate frames/s, a 16.2 $\times$ system-level throughput improvement over this reference.

5.4 Application

As shown in Fig. 7, Boba supports downstream applications with different system requirements; here we focus on robot planning. Using BOBA-BATCHED as the rollout simulator in the same MPC setting as PHYSTWIN, where MPC samples candidate action sequences, simulates rollouts, and executes the action that best moves the deformable object toward the target, we observe substantial gains from

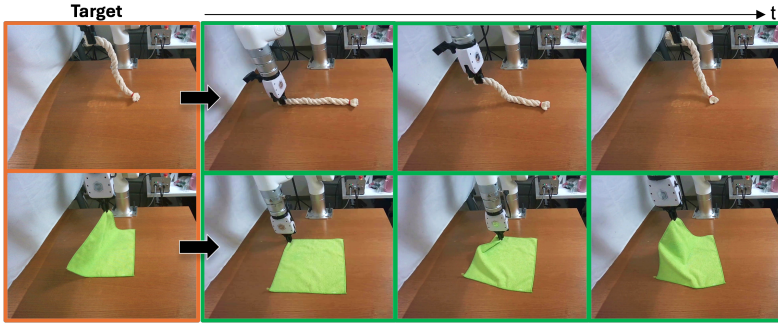


Fig. 7: Robot-planning application. BOBA-BATCHED serves as the MPC rollout simulator for rope and cloth manipulation, enabling high-throughput parallel rollout.

Boba’s batching-friendly design. BOBA-BATCHED achieves a $26\times$ speedup on rope. For cloth, under matched collision settings, PHYSTWIN-based planning with self-collision takes over 24 hours; based on the first 100 samples, BOBA-BATCHED provides an estimated speedup of over $2000\times$. Even without self-collision, it still achieves more than $16\times$ speedup. This gap primarily comes from avoiding repeated collision-graph construction and serialization overhead in PHYSTWIN, highlighting Boba’s advantage for realistic, larger-scale robot-planning workloads. The supplement also shows a Quest 3 XR prototype for controller-based manipulation of a simulated Gaussian physical twin in a reconstructed indoor scene, and a PPO training example using Boba as a high-throughput RL simulator.

6 Conclusion

We introduced **Boba**, a scalable framework for physics-based Gaussian digital twins of deformable objects. Built on template–state separation, Boba co-designs simulation, deformation, rendering/visualization, and batching to reduce overhead while preserving PHYSTWIN-level fidelity. Its three configurations cover complementary regimes: BOBA-LOCAL achieves over $10\times$ speedup over PHYSTWIN on XR-class Orin; BOBA-DISTRIBUTED, using an edge-server split, reaches 25.1 ms latency, a $13.2\times$ improvement over PHYSTWIN on Orin, with 22.2% lower incremental power than BOBA-LOCAL; and BOBA-BATCHED reaches 3,310 FPS aggregate throughput on RTX 4090 and accelerates robot planning, with a measured $26\times$ speedup on rope and an estimated $> 2,000\times$ speedup on cloth with matched self-collision. These results position Boba as a practical substrate for power-constrained XR interaction and scalable deformable-object robot planning.

7 Acknowledgments

The authors thank the anonymous reviewers for their valuable feedback and Benjamin Goldfisher for early project assistance. This work was supported in part by Meta Platforms Technologies, LLC and the National Science Foundation (NSF) grants 2217144, 2312102, 2331878, 2340254, 2414227, and 2404385.

Bibliography

- [1] Chen, H.y., Tretschk, E., Stuyck, T., Kadlecsek, P., Kavan, L., Vouga, E., Lassner, C.: Virtual elastic objects. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 15827–15837 (2022) 4
- [2] Chen, Y., Hu, Y., Sun, L., Kusnur, T., Herlant, L., Jiang, C.: Empm: Embodied mpm for modeling and simulation of deformable objects. arXiv preprint arXiv:2601.17251 (2026) 4
- [3] Feng, Y., Shang, Y., Li, X., Shao, T., Jiang, C., Yang, Y.: Pie-nerf: Physics-based interactive elastodynamics with nerf. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4450–4461 (2024) 4
- [4] Freeman, C.D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., Bachem, O.: Brax—a differentiable physics engine for large scale rigid body simulation. arXiv preprint arXiv:2106.13281 (2021) 4
- [5] Gu, J., Xiang, F., Li, X., Ling, Z., Liu, X., Mu, T., Tang, Y., Tao, S., Wei, X., Yao, Y., Yuan, X., Xie, P., Huang, Z., Chen, R., Su, H.: Maniskill2: A unified benchmark for generalizable manipulation skills. In: International Conference on Learning Representations (2023) 2, 4
- [6] Intel: Intel core i9-13900kf processor. Intel Products (2022), <https://www.intel.com/content/www/us/en/products/sku/230497/intel-core-i913900kf-processor-36m-cache-up-to-5-80-ghz/specifications.html>, accessed: 30 June 2026 11
- [7] Jiang, C., Schroeder, C., Teran, J., Stomakhin, A., Selle, A.: The material point method for simulating continuum materials. In: Acm siggraph 2016 courses, pp. 1–52 (2016) 6
- [8] Jiang, H., Hsu, H.Y., Zhang, K., Yu, H.N., Wang, S., Li, Y.: Phystwin: Physics-informed reconstruction and simulation of deformable objects from videos. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 7219–7230 (2025) 2, 4, 5, 11
- [9] Jiang, Y., Yu, C., Xie, T., Li, X., Feng, Y., Wang, H., Li, M., Lau, H., Gao, F., Yang, Y., et al.: Vr-gs: A physical dynamics-aware interactive gaussian splatting system in virtual reality. In: ACM SIGGRAPH 2024 Conference Papers. pp. 1–1 (2024) 2, 4, 5
- [10] Li, X., Qiao, Y.L., Chen, P.Y., Jatavallabhula, K.M., Lin, M., Jiang, C., Gan, C.: Pac-nerf: Physics augmented continuum neural radiance fields for geometry-agnostic system identification. arXiv preprint arXiv:2303.05512 (2023) 4
- [11] Liu, T., Bargteil, A.W., O’Brien, J.F., Kavan, L.: Fast simulation of mass-spring systems. ACM Transactions on Graphics (TOG) 32(6), 1–7 (2013) 6

- [12] Macklin, M., Müller, M., Chentanez, N.: Xpbd: position-based simulation of compliant constrained dynamics. In: Proceedings of the 9th International Conference on Motion in Games. pp. 49–54 (2016) [6](#)
- [13] Makoviychuk, V., Wawrzyniak, L., Guo, Y., Lu, M., Storey, K., Macklin, M., Hoeller, D., Rudin, N., Allshire, A., Handa, A., State, G.: Isaac gym: High performance gpu-based physics simulation for robot learning (2021) [4](#)
- [14] Mittal, M., Roth, P., Tigue, J., Richard, A., Zhang, O., Du, P., Serrano-Muñoz, A., Yao, X., Zurbrügg, R., Rudin, N., Wawrzyniak, L., Rakhsha, M., Denzler, A., Heiden, E., Borovicka, A., Ahmed, O., Akinola, I., Anwar, A., Carlson, M.T., Feng, J.Y., Garg, A., Gasoto, R., Gulich, L., Guo, Y., Gussert, M., Hansen, A., Kulkarni, M., Li, C., Liu, W., Makoviychuk, V., Malczyk, G., Mazhar, H., Moghani, M., Murali, A., Noseworthy, M., Poddubny, A., Ratliff, N., Rehberg, W., Schwarke, C., Singh, R., Smith, J.L., Tang, B., Thaker, R., Trepte, M., Wyk, K.V., Yu, F., Millane, A., Ramasamy, V., Steiner, R., Subramanian, S., Volk, C., Chen, C., Jawale, N., Kuruttukulam, A.V., Lin, M.A., Mandlekar, A., Patzwaldt, K., Welsh, J., Zhao, H., Anes, F., Laffleche, J.F., Moënné-Loccoz, N., Park, S., Stepinski, R., Gelder, D.V., Amevor, C., Carius, J., Chang, J., Chen, A.H., de Heras Ciechomski, P., Daviet, G., Mohajerani, M., von Mural, J., Reutsky, V., Sauter, M., Schirm, S., Shi, E.L., Terdiman, P., Vilella, K., Widmer, T., Yeoman, G., Chen, T., Grizan, S., Li, C., Li, L., Smith, C., Wiltz, R., Alexis, K., Chang, Y., Chu, D., Fan, L.J., Farshidian, F., Handa, A., Huang, S., Hutter, M., Narang, Y., Pouya, S., Sheng, S., Zhu, Y., Macklin, M., Moravanszky, A., Reist, P., Guo, Y., Hoeller, D., State, G.: Isaac lab: A gpu-accelerated simulation framework for multi-modal robot learning. arXiv preprint arXiv:2511.04831 (2025), <https://arxiv.org/abs/2511.04831>, accessed: 30 June 2026 [2](#), [4](#)
- [15] Mu, T., Ling, Z., Xiang, F., Yang, D., Li, X., Tao, S., Huang, Z., Jia, Z., Su, H.: Maniskill: Generalizable manipulation skill benchmark with large-scale demonstrations. arXiv preprint arXiv:2107.14483 (2021) [2](#), [4](#)
- [16] Müller, M., Heidelberger, B., Hennix, M., Ratcliff, J.: Position based dynamics. Journal of Visual Communication and Image Representation **18**(2), 109–118 (2007) [6](#)
- [17] NVIDIA: GeForce RTX 4090 Family. NVIDIA Products (2022), <https://www.nvidia.com/en-us/geforce/graphics-cards/40-series/rtx-4090/>, accessed: 30 June 2026 [11](#)
- [18] NVIDIA: NVIDIA AGX Orin. NVIDIA Robotics and Edge AI (2024), <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/>, accessed: 30 June 2026 [11](#)
- [19] Qiao, Y.L., Gao, A., Lin, M.: Neuphysics: Editable neural geometry and physics from monocular videos. Advances in Neural Information Processing Systems **35**, 12841–12854 (2022) [4](#)
- [20] Shacklett, B., Rosenzweig, L.G., Xie, Z., Sarkar, B., Szot, A., Wijmans, E., Koltun, V., Batra, D., Fatahalian, K.: An extensible, data-oriented architecture for high-performance, many-world simulation. ACM Trans. Graph. **42**(4) (2023) [2](#), [4](#), [10](#)

- [21] Sifakis, E., Barbic, J.: Fem simulation of 3d deformable solids: a practitioner’s guide to theory, discretization and model reduction. In: *Acm siggraph 2012 courses*, pp. 1–50 (2012) [6](#)
- [22] Sulsky, D., Chen, Z., Schreyer, H.L.: A particle method for history-dependent materials. *Computer methods in applied mechanics and engineering* **118**(1-2), 179–196 (1994) [6](#)
- [23] Tao, S., Xiang, F., Shukla, A., Qin, Y., Hinrichsen, X., Yuan, X., Bao, C., Lin, X., Liu, Y., kai Chan, T., Gao, Y., Li, X., Mu, T., Xiao, N., Gurha, A., Rajesh, V.N., Choi, Y.W., Chen, Y.R., Huang, Z., Calandra, R., Chen, R., Luo, S., Su, H.: Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *Robotics: Science and Systems* (2025) [2](#), [4](#)
- [24] Team, G.A.: The role of simulation in scalable robotics, genesis world 1.0, and the path forward. *Genesis AI Blog* (May 2026), <https://www.genesis.ai/blog/the-role-of-simulation-in-scalable-robotics-genesis-world-10-and-the-path-forward>, accessed: 30 June 2026 [2](#), [4](#)
- [25] The Newton Contributors: Newton: Gpu-accelerated physics simulation for robotics and simulation research. <https://github.com/newton-physics/newton> (Apr 2025), accessed: 30 June 2026 [4](#)
- [26] Todorov, E., Erez, T., Tassa, Y.: Mujoco: A physics engine for model-based control. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. pp. 5026–5033. IEEE (2012). <https://doi.org/10.1109/IROS.2012.6386109>, accessed: 30 June 2026 [4](#), [10](#)
- [27] Xiang, F., Qin, Y., Mo, K., Xia, Y., Zhu, H., Liu, F., Liu, M., Jiang, H., Yuan, Y., Wang, H., et al.: Sapien: A simulated part-based interactive environment. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. pp. 11097–11107 (2020) [2](#), [4](#)
- [28] Xie, T., Zong, Z., Qiu, Y., Li, X., Feng, Y., Yang, Y., Jiang, C.: Physgaussian: Physics-integrated 3d gaussians for generative dynamics. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 4389–4398 (2024) [2](#), [4](#), [5](#)
- [29] Yang, Y., Zhang, Z., Zhang, X., Zeng, Y., Li, H., Zuo, W.: Physworld: From real videos to world models of deformable objects via physics-aware demonstration synthesis. *arXiv preprint arXiv:2510.21447* (2025) [4](#)
- [30] Zhang, K., Sha, S., Jiang, H., Loper, M., Song, H., Cai, G., Xu, Z., Hu, X., Zheng, C., Li, Y.: Real-to-sim robot policy evaluation with gaussian splatting simulation of soft-body interactions. *ICRA* (2026) [4](#)
- [31] Zhang, T., Yu, H.X., Wu, R., Feng, B.Y., Zheng, C., Snively, N., Wu, J., Freeman, W.T.: Physdreamer: Physics-based interaction with 3d objects via video generation. In: *European Conference on Computer Vision*. pp. 388–406. Springer (2024) [4](#)
- [32] Zhong, L., Yu, H.X., Wu, J., Li, Y.: Reconstruction and simulation of elastic objects with spring-mass 3d gaussians. In: *European Conference on Computer Vision*. pp. 407–423. Springer (2024) [4](#)