GaussTwin: Unified Simulation and Correction with Gaussian Splatting for Robotic Digital Twins

Yichen Cai[1], Paul Jansonnie[15], Cristiana de Farias[1], Oleg Arenz[1], Jan Peters[1234]

***Abstract*— Digital twins promise to enhance robotic manipulation by maintaining a consistent link between real-world perception and simulation. However, most existing systems struggle with the lack of a unified model, complex dynamic interactions, and the real-to-sim gap, which limits downstream applications such as model predictive control. Thus, we propose GaussTwin, a real-time digital twin that combines position-based dynamics with discrete Cosserat rod formulations for physically grounded simulation, and Gaussian splatting for efficient rendering and visual correction. By anchoring Gaussians to physical primitives and enforcing coherent SE(3) updates driven by photometric error and segmentation masks, GaussTwin achieves stable prediction-correction while preserving physical fidelity. Through experiments in both simulation and on a Franka Research 3 platform, we show that GaussTwin consistently improves tracking accuracy and robustness compared to shape-matching and rigid-only baselines, while also enabling downstream tasks such as push-based planning. These results highlight GaussTwin as a step toward unified, physically meaningful digital twins that can support closed-loop robotic interaction and learning. Code and videos are available at `https://6cyc6.github.io/gstwin/`.**

I. Introduction

Building a real-time, dynamic digital twin offers significant benefits for robotic manipulation. Unlike traditional simulation, which creates a virtual environment separate from the real world, a dynamic digital twin predicts future states, continuously corrects and synchronizes the robot with reality, and generates visual representations of changing environment conditions. This capability helps bridge the real-to-simulation gap, enabling more effective planning, tracking, and the generation of robot–object interaction videos for policy learning [1], [2]. However, developing digital twins that rely on a unified physics model capable of handling a wide range of properties and behaviors, such as rigid and deformable bodies, contact interactions, and different material types, remains a significant challenge.

In practice, digital twins have typically been constructed using point clouds [3]–[6], meshes [7], or NeRF [8] to represent the 3D environment, but these representations face trade-offs among efficiency, differentiability, and fidelity. Recently, 3D Gaussan Splatting (3DGS) [9] has emerged as an alternative for 3D scene representation. By modeling a scene as a dense set of 3D Gaussians, 3DGS enables efficient and differentiable novel-view image synthesis with low memory cost and high visual fidelity. Building on this idea, works such as, [10]–[12] combine 3DGS-based reconstruction with neural dynamic models to construct more visually accurate digital twins that capture dynamics directly from videos. Here, by integrating Graph Neural Networks with a particle-grid representation, [12] demonstrated strong capabilities in modeling deformable object dynamics, achieving accurate long-horizon predictions while leveraging 3DGS for rendering. However, these learning-based approaches often struggle to generalize beyond their training distribution and demand extensive, labor-intensive data collection and processing.

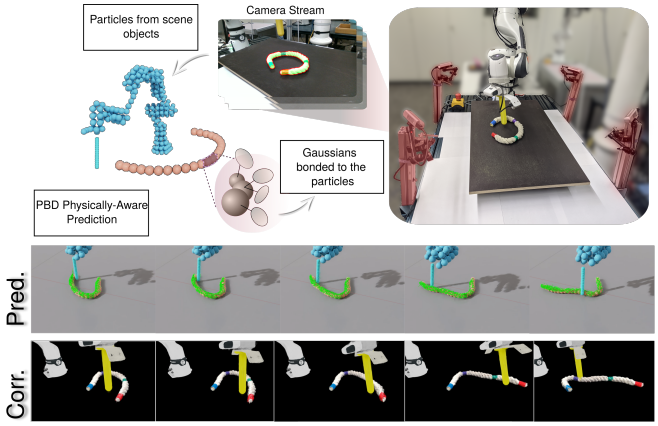

Fig. 1: Tracking performance of GaussTwin on rigid objects and DLO. The system takes multi-view camera observations as input. First, the objects in the scene are masked, represented by particles, and bonded to 3D Gaussians. The motion of these particles is then predicted at each time step using PBD, and subsequently refined through Gaussian splatting optimization. The bottom two rows illustrate the prediction–correction process carried out by GaussTwin.

Concurrently with learning-based methods, another line of work has explored hybrid approaches that integrate physics-based simulation with visual or data-driven corrections. As was often the case in earlier digital twin implementations, point clouds or signed distance fields were commonly used for the visual correction step [3]–[6], [13]. However, these representations are typically tailored to specific object types such as ropes or soft tissues, which restricts their general applicability. In addition, both point clouds and SDFs are prone

This work was funded by the German Research Foundation (DFG) - Project number PE 2315/18-1, the EU's Horizon Europe project ARISE - Grant number 101135959, and partially supported by the German Federal Ministry of Research, Technology and Space (BMFTR) under the Robotics Institute Germany (RIG). This work was also supported by a hardware donation from NVIDIA through the Academic Grant Program, and by the Lichtenberg high-performance computer of TU Darmstadt.

We acknowledge that ChatGPT is used for grammar enhancement, and Copilot is used to generate some auxiliary functions in the code.

[1]Intelligent Autonomous Systems Lab, Technical University of Darmstadt, Germany. [2]Hessian.AI, Germany. [3]German Research Center for AI (DFKI), SAIROL, Germany. [4]Robotics Institute Germany (RIG). [5]NAVER LABS Europe.

Corresponding author: `yichen.cai@tu-darmstadt.de`

to occlusion and sensor noise. In parallel with the advances described above, 3DGS has also emerged as a promising alternative for the visual correction component of prediction-correction models. For instance, [2], [14] introduced a 3DGS-based visual corrective world model, which uses a simulator for prediction and then refines the results through visual feedback. Both methods are built upon Position-Based Dynamics (PBD) [15], a simulation framework that enforces constraints on positions of particles to achieve fast and stable real-time dynamics. Specifically, [14] proposed a hybrid Particle-Gaussians representation, where particles evolve in a PBD simulator while bonded 3D Gaussians are used for image rendering and visual feedback. Their approach further employed a shape-matching algorithm with oriented particles [16], [17] to simulate both rigid and deformable objects within the PBD framework. However, the shape-matching algorithm lacks physically meaningful properties, leading to inaccurate simulation predictions, especially for Deformable Linear Objects (DLOs), e.g., a rope. Moreover, it requires a high visual correction gain because the Gaussians are optimized independently, which often leads to strong oscillations. In follow-up work, [2] adopted rigid-body dynamics to achieve more accurate predictions and applied their improved corrective model to a manipulation task, where they used it to push a T-shaped object. While this enhanced rigid-body tracking, it came at the cost of losing the ability to simulate and track deformable objects.

In this paper, we present Gaussian Splatting for Robotic Digital Twins (GaussTwin), a *unified hybrid framework* that overcomes the limitations of prior 3DGS-based corrective models (See Fig. 1). Instead of relying on shape matching alone or restricting to rigid-body dynamics, we extend PBD with the discrete Cosserat rod model [18], which provides a physically meaningful formulation capable of accurately modeling both rigid objects and DLOs within the same framework. To further improve stability, we introduce a joint optimization scheme for 3D Gaussians that constrains them to move coherently with their associated rigid bodies or rod segments, preventing independent drift that previously caused oscillations and required high correction gains. Together, these contributions enable GaussTwin to unify the prediction and correction of rigid and deformable objects, while ensuring more stable, efficient, and physically accurate digital twin simulations. The key contributions of this paper are summarized as follows:

- We introduce GaussTwin, a hybrid framework that combines PBD with 3D Gaussians to jointly predict and correct the state of both rigid bodies and DLOs, thereby bridging the real-to-sim gap.
- By leveraging segmentation masks, enforcing coherent motion of 3D Gaussians with their corresponding rigid bodies or rod segments, and incorporating physically plausible constraints in PBD, our method achieves stable and precise prediction–correction without sacrificing real-time performance.
- Through simulation and real-world experiments, we demonstrate that GaussTwin achieves more accurate and robust tracking compared to prior 3DGS-based corrective models [14].
- We show the effectiveness of our method in a downstream robotic planning task, highlighting its potential to support closed-loop interaction and control in real-world environments.

II. Related Works

Building a model that can simulate and track both rigid and linear deformable objects is crucial for complex robotic manipulation tasks. Many physics simulators are available to simulate rigid bodies [19], [20] and DLOs [20]–[23]. However, they fail to predict and track states of the objects over a long horizon due to mismatches with the real world, especially for objects with unknown physical parameters. To address this, some works estimate physical parameters through reconstruction methods, thereby helping bridge the sim-to-real gap. Most early works estimate parameters for objects with known geometry using synthetic data [24]–[26] or point clouds [27], [28]. With the development of 3D reconstruction techniques [9], [29] in computer vision, recent works leverage NeRF or Gaussian Splattings to reconstruct the geometry and texture of objects and to estimate parameters from videos. However, these methods remain challenging for estimating the parameters of objects with complex physical properties, such as DLOs. In addition, they are limited to specific motions and scenarios. Another branch of work [10]–[12], [30]–[33] employs data-driven methods, which train a deep neural network to learn dynamics directly from videos. By incorporating physical priors, [11], [12] propose a physics-informed graph neural network that learns deformable object dynamics from sparse real-world robot interaction videos. When combined with 3DGS, the model achieves improved reconstruction and prediction performance across a range of objects and motions. However, these methods still require building a complex system to collect and preprocess training data. Additionally, they struggle to generalize to out-of-training scenarios involving external disturbances, collisions, and interactions among multiple objects.

Recently, [2], [14] proposed corrective world models. By integrating real-world visual feedback into the simulator, these methods construct a real-time digital twin that effectively bridges the real-to-sim gap. Furthermore, [5], [6] build a PBD simulation to simulate soft tissues and leverage the point cloud observation to correct the predicted state of the simulation. Although they achieved improved tracking performance, perception noise and computational cost limited their usage. Most related to our method, [14] builds a hybrid particle-Gaussians model from RGB-D images and initializes it in a PBD simulation. In this case, future states of the particle-Gaussians model are initially predicted within the PBD simulator. Visual forces derived from rendered images of the bonded 3D Gaussians are applied to the particles to correct their states. However, they use shape matching to ensure that unconnected particles retain the shape of the

Algorithm 1: Unified PBD Simulation

```
InitializeRigidBodyStates()
InitializeRopeStates()
while simulation not stopped do
    for i = 1 to numSubsteps do
        CollectCollisionPairs()
        StatePrediction()
        for j = 1 to solverIterations do
            ClearDeltas()
            (Δx^C, Δq^C) ← SolveRigidParticleContact()  (5)
            (Δx^S, Δq^S) ← SolveShearStretch()  (7)
            (Δx^B, Δq^B) ← SolveBendTwist()  (9)
            (Δx^R, Δq^R) ← SolveRigidBodyContact()  (4)
            ApplyDeltas()
        end for
        UpdateVelocities()
    end for
end while
```

object they represent, which has no physical properties of its own. Thus, the simulation's prediction might introduce large errors, especially for DLOs. In [2], they switch to rigid body dynamics, but lose the ability to track DLOs. [34] uses a similar method to track DLOs, but the simplified rope model restricted their tracking performance. We extend their method and build a unified PBD simulation capable of modeling and tracking both rigid objects and DLOs.

III. Preliminary

A. Position-Based Dynamics Simulation

Position-Based Dynamics (PBD) [15] has been extensively used for building interactive physics systems due to its simplicity, stability, visual plausibility, and computational speed [35]. These properties make it especially suitable for building a real-time visual corrective digital twin. In general, PBD is a particle-based system paired with various constraints based on the dynamics of the objects. By integrating rigid body dynamics [36] and the discrete Cosserat rod model [18], PBD can be extended to a unified framework that jointly simulates different types of objects, including rigid bodies and DLOs. The overall PBD simulation process is outlined in Algorithm 1. It begins with system initialization, where the state of each rigid body is defined. Specifically, the *i-th* rigid body is represented by its mass m_i, position $\boldsymbol{x}_i$, velocity $\boldsymbol{v}_i$, inertial matrix $\boldsymbol{I}_i$, orientation $\boldsymbol{q}_i \in \boldsymbol{SO}(3)$, and angular velocity $\boldsymbol{\omega}_i$. Following the Cosserat rod model, a DLO (such as a rope) can be approximated by a sequence of linear segments along its centerline. The state of a DLO is therefore defined by two sets of variables: positional variables, including the mass m_i, position $\boldsymbol{x_i}$, and velocity $\boldsymbol{v_i}$ of each particle along the centerline; and rotational variables, including the inertia matrix $\boldsymbol{I}_i$, orientation $\boldsymbol{q}_i$, and angular velocity $\boldsymbol{\omega}_i$ of each segment.

After initialization, the simulation enters its main loop, which is executed over several substeps. Each substep begins with a collision check to build the contact constraints, after which PBD predicts the positions and orientations using semi-implicit Euler integration. Next, a Jacobian solver is used to solve the constraints $\boldsymbol{C}(\boldsymbol{p}_0, ..., \boldsymbol{p}_n)$, which is the core step of the algorithm. Here, each $\boldsymbol{p}_i$ denotes either a position or an orientation. The constraints may be either an equality or an inequality, scalar or vector. To compute the update displacements, each constraint is locally linearized as $\boldsymbol{C}(\boldsymbol{p} + \Delta\boldsymbol{p}) = \boldsymbol{C}(\boldsymbol{p}) + \nabla_{\boldsymbol{p}}\boldsymbol{C}\,\Delta\boldsymbol{p} = 0$. The displacement direction is restricted in the projected space and can be solved using a Lagrangian multiplier

$$\begin{aligned} \boldsymbol{\lambda} &= -\left(\sum_k \left(\nabla_{\boldsymbol{p}_k}\boldsymbol{C}\right)\boldsymbol{W}_k\left(\nabla_{\boldsymbol{p}_k}\boldsymbol{C}\right)^T\right)^{-1}\boldsymbol{C}(\boldsymbol{p}), \\ \Delta\boldsymbol{p}_i &= \boldsymbol{W}_i\left(\nabla_{\boldsymbol{p}_i}\boldsymbol{C}\right)^T\boldsymbol{\lambda}, \end{aligned} \tag{1}$$

where k is the number of states involved in a constraint, and $\boldsymbol{W} = \boldsymbol{diag}[m_1^{-1}, ...m_{n_x}^{-1}, \boldsymbol{I}_1^{-1}, ...\boldsymbol{I}_{n_q}^{-1}]$ encodes the inverse masses and inertias. We note that, each variable update is weighted by $\boldsymbol{W}_i$ to ensure conservation of momentum. Since all constraints are independent, they can be efficiently solved in parallel on a GPU. The accumulated updates are then applied to the predicted state and cleared before the next solver iteration. Finally, the corrected predictions are used to update the velocities. Further details on the constraint formulations can be found in Section IV-A, and a comprehensive discussion of PBD is provided in [15].

B. 3D Gaussian Splattings

3D Gaussian Splatting [9] explicitly represents the scene by a set of 3D Gaussians. Each Gaussian $\boldsymbol{g}_i$ is defined by its mean $\boldsymbol{\mu}_i \in \mathbb{R}^3$, covariance matrix $\boldsymbol{\Sigma}_i \in \mathbb{R}^{3\times3}$, opacity $\alpha_i \in \mathbb{R}$, and color $\boldsymbol{c}_i \in \mathbb{R}^3$. The covariance matrix is decomposed into a rotation matrix $\boldsymbol{R}_i$ and a diagonal scale matrix $\boldsymbol{S}_i$ by $\boldsymbol{\Sigma}_i = \boldsymbol{R}_i\boldsymbol{S}_i\boldsymbol{S}_i^T\boldsymbol{R}_i^T$. For the training, the rotation is reparameterized as a unit quaternion $\boldsymbol{q}_i$, and the diagonal elements of the scale matrix are formed as a scale vector $\boldsymbol{s}_i$. Given a camera configuration, the Gaussians are projected onto the image plane and sorted by depth. The image is then rendered by α-blending.

IV. Methodology

In this section, we present the main aspects of GaussTwin, which allows us to align simulation with visual correction to create a physically accurate digital twin for manipulation tasks. An overview of the pipeline is shown Fig. 1. An object-Gaussian bond is initially constructed from RGB-D images captured by multiple cameras. After initialization, the tracking, which consists of a prediction step and a correction step, runs at 25 Hz. We first describe the constraints used in the PBD simulation and then discuss the details of the initialization and the tracking procedures.

A. Constraint definition

To simulate rigid bodies in PBD, we employ a number of different constraints. For contact collision constraints between the rigid bodies, we have

$$C^R(\boldsymbol{x}_i, \boldsymbol{q}_i, \boldsymbol{x}_j, \boldsymbol{q}_j; \boldsymbol{n}, \boldsymbol{b}_i, \boldsymbol{b}_j) = \boldsymbol{n}\cdot(\boldsymbol{b}_j - \boldsymbol{b}_i) \geq 0, \tag{2}$$

where $\boldsymbol{n}$ is the contact normal, $\boldsymbol{b}_i$ is the contact position of the rigid body i in the world frame, and the indices i and

j denote the two rigid bodies involved in the contact. Here, the generalized inverse masses are

$$\begin{aligned} w_i &= m_i^{-1} + (\boldsymbol{r}_i \times \boldsymbol{n})^\top \boldsymbol{I}_i^{-1} (\boldsymbol{r}_i \times \boldsymbol{n}), \\ w_j &= m_j^{-1} + (\boldsymbol{r}_j \times \boldsymbol{n})^\top \boldsymbol{I}_j^{-1} (\boldsymbol{r}_j \times \boldsymbol{n}), \end{aligned} \tag{3}$$

where $\boldsymbol{r}_i$ and $\boldsymbol{r}_j$ are vectors from the center of mass to the contact position. The resulting state correction of each rigid body is then given by

$$\begin{aligned} \Delta \boldsymbol{x}_i^R &= -\frac{m_i^{-1}}{w_i + w_j} C^R \boldsymbol{n},\ \Delta \boldsymbol{x}_j^R = \frac{m_j^{-1}}{w_i + w_j} C^R \boldsymbol{n}, \\ \Delta \boldsymbol{q}_i^R &= -\frac{1}{2} [\boldsymbol{I}_i^{-1} (\boldsymbol{r}_i \times \frac{1}{w_i + w_j} C^R \boldsymbol{n}), 0] \boldsymbol{q_i}, \\ \Delta \boldsymbol{q}_j^R &= \frac{1}{2} [\boldsymbol{I}_j^{-1} (\boldsymbol{r}_j \times \frac{1}{w_i + w_j} C^R \boldsymbol{n}), 0] \boldsymbol{q_j}. \end{aligned} \tag{4}$$

The next constraint is the collision between a rigid body i and a particle j, formulated as $C^C(\boldsymbol{x}_{rb_i}, \boldsymbol{q}_{rb_i}, \boldsymbol{x}_j; \boldsymbol{n}, \boldsymbol{b}, r_j) = \boldsymbol{n} \cdot (\boldsymbol{x}_{rb_i} - \boldsymbol{b}) - r_j \geq 0$. This constraint is solved in a manner similar to the contact collision constraints between rigid bodies. The only difference is that ω_j is simplified to m_j^{-1} and there is no orientation update for the particle. In this case, the update rule is as follows:

$$\begin{aligned} \Delta \boldsymbol{x}_i^C &= -\frac{m_i^{-1}}{w_i + w_j} C^C \boldsymbol{n},\ \Delta \boldsymbol{x}_j^C = \frac{m_j^{-1}}{w_i + w_j} C^C \boldsymbol{n}, \\ \Delta \boldsymbol{q}_i^C &= -\frac{1}{2} [\, \boldsymbol{I}_i^{-1} (\boldsymbol{r}_i \times \frac{1}{w_i + w_j} C^C \boldsymbol{n}), 0] \boldsymbol{q_i}. \end{aligned} \tag{5}$$

For more details, please refer to [37].

The following constraint we address is the shear-stretch and bend-twist constraints used for elastic rods (i.e. DLOs). To describe these constraints, an orthogonal frame with basis $\{\boldsymbol{d}_1(\boldsymbol{q}), \boldsymbol{d}_2(\boldsymbol{q}), \boldsymbol{d}_3(\boldsymbol{q})\}$ is attached to the center of each rod segment, where the vectors are denoted as directors. $\boldsymbol{q}$ represents the rotation from a fixed world coordinate system with basis $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3\}$ to the local frame. The third director $\boldsymbol{d}_3(\boldsymbol{q}) = \boldsymbol{R}(\boldsymbol{q})\boldsymbol{e}_3$ is parallel to the normal direction of each rod segment. The shear-stretch constraint for two adjacent particles at positions $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is then given by

$$\boldsymbol{C}^S(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{q}) = \frac{1}{l_{ij}} (\boldsymbol{x}_j - \boldsymbol{x}_i) - \boldsymbol{d}_3(\boldsymbol{q}) = \boldsymbol{0}, \tag{6}$$

where l_{ij} is the length of the segment at its rest position. This constraint maintains the distance of each segment at its rest length and ensures that the direction of the tangent aligns with the rod segment's $\boldsymbol{d}_3(\boldsymbol{q})$ direction. For faster computations, we simplify the inertia matrix for each rod segment into a single scalar m_q following [36]. The resulting state updates are as follows:

$$\begin{aligned} \Delta \boldsymbol{x}_i &= \frac{m_i^{-1} l_{ij}}{m_i^{-1} + m_j^{-1} + 4 m_q^{-1} l_{ij}^2} \boldsymbol{C}^S(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{q}), \\ \Delta \boldsymbol{x}_j &= -\frac{m_j^{-1} l_{ij}}{m_i^{-1} + m_j^{-1} + 4 m_q^{-1} l_{ij}^2} \boldsymbol{C}^S(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{q}), \\ \Delta \boldsymbol{q} &= \frac{2 m_q^{-1} l_{ij}^2}{m_i^{-1} + m_j^{-1} + 4 m_q^{-1} l_{ij}^2} \boldsymbol{C}^S(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{q}) \boldsymbol{q} \bar{\boldsymbol{e}}_3. \end{aligned} \tag{7}$$

Here $\bar{\boldsymbol{e}}_3$ is the quaternion representation of $\boldsymbol{e}_3$. Finally, we have the bend-twist constraint

$$\begin{aligned} \boldsymbol{C}^B(\boldsymbol{q}_i, \boldsymbol{q}_j) = \mathfrak{Im}(\bar{\boldsymbol{q}}_i \boldsymbol{q}_j - \bar{\boldsymbol{q}}_i^0 \boldsymbol{q}_j^0) = \boldsymbol{\Omega} - \alpha \boldsymbol{\Omega}^0 = \boldsymbol{0} \\ \alpha = \operatorname{sign}(\boldsymbol{\Omega} + \boldsymbol{\Omega}^0), \end{aligned} \tag{8}$$

where $\boldsymbol{\Omega}$ is the discrete Darboux vector $\boldsymbol{\Omega} = \frac{2}{l_{ij}} \mathfrak{Im}\big[\bar{\boldsymbol{q}}_i \boldsymbol{q}_j\big]$, with l_{ij} being the average length of two segments and $\mathfrak{Im}[\cdot]$ refers to the imaginary part of a quaternion. The bend-twist constraint moves the state of the rod such that its curvature and bending match its rest shape described by $\boldsymbol{\Omega}^0$. Here, the displacements of the particles are given by:

$$\begin{aligned} \Delta \boldsymbol{q}_i &= \frac{m_{q_i}^{-1}}{m_{q_i}^{-1} + m_{q_j}^{-1}} \boldsymbol{q}_j \boldsymbol{C}^B(\boldsymbol{q}_i, \boldsymbol{q}_j), \\ \Delta \boldsymbol{q}_j &= -\frac{m_{q_j}^{-1}}{m_{q_i}^{-1} + m_{q_j}^{-1}} \boldsymbol{q}_i \boldsymbol{C}^B(\boldsymbol{q}_i, \boldsymbol{q}_j). \end{aligned} \tag{9}$$

Since the Cosserat rod model explicitly represents bending and twisting through a continuum mechanics formulation [18], it provides a meaningful physical description of rope deformation. This yields more physically consistent deformation and contact reactions than shape matching, which enforces geometric constraints without modeling the underlying mechanics. Furthermore, by making the parameterization of the Cosserat rod model explicit, we can enable system identification, which we plan to explore in future work.

B. Scene Initialization

Given multiple RGB-D images from various viewpoints, we use SAM2 [38] to extract instance masks of the object. A bounding box is generated, and we fill it with evenly distributed spheres of 5 mm radius in 3D space. We also segment the workspace surface and use RANSAC [39] to fit a plane over it. This plane serves as the ground plane in the simulation. For each sphere center representing the object, we project it onto the image planes. We discard points that fall outside the mask, below the ground plane, or have a depth smaller than the corresponding pixel value. The remaining spheres are then used to approximate the object. Given the density of the rigid body, its mass, center of mass, pose, and inertial matrix are computed by integrating over the remaining spheres. The object's density can be obtained either by querying the VLM or by using an online system identification. In this work, we assume the density is known and leave automated estimation to future work. As in [2], we also use these spheres for collision detection in the simulation. For a DLO, we first train its 3D Gaussian representation. Then, we use it to render an image from a top view. The 2D skeleton of the rope is extracted from the rendered image using Lee's algorithm [40]. Finally, we order the skeleton points by finding the longest path, yielding a sequence of evenly spaced spheres along the centerline. The radius of each sphere is computed from the point cloud. The physical properties used to set up the rope model in PBD are estimated based on the material, radius, and length.

After obtaining the necessary information to set up the PBD simulation, we initialize the 3D Gaussians from the

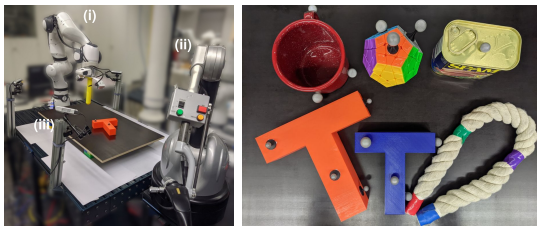

Fig. 2: On the left, we show the experimental setup, with the (i) Franka Research 3 robot with the custom end effector tool, (ii) the haptic interface for teleoperation, and (iii) the set of Intel RealSense D415 scene mounted cameras. On the right, we show the set of all objects used for real-robot experiments.

segmented object point cloud and optimize them using the photometric loss as described in [9]. We train the Gaussians using the Adam optimizer for 1000 steps. The learning rates are set $1e^{-5}$ for means, $1e^{-3}$ for quaternions, $2.5e^{-3}$ for colors, $2e^{-3}$ for scales, and $5e^{-2}$ for opacities. Finally, the optimized Gaussians are anchored to the nearest sphere of the object. For the robot, we manually populate each link with spheres for collision checking in PBD and store the corresponding optimized 3D Gaussians.

C. Online Tracking

After scene initialization, the objects with bonded Gaussians are loaded into the PBD simulation. Online tracking then proceeds in two stages: a prediction step and a correction step. The system operates at 25 Hz. After receiving the current RGB images and robot configurations, we initiate the prediction step by setting the robot links' poses in simulation to those calculated from the intermediate joint configuration using its forward kinematics. Next, a PBD step, as described in Section III-A, is executed, and the Gaussians are moved coherently with the predicted state. We implement the PBD using NVIDIA warp [23], and each simulation step takes 0.1 ms on a single NVIDIA 4090.

Subsequently, three images are rendered according to the calibrated camera configurations using the 3D Gaussians. Since only the Gaussians of the robot and the object are saved for rendering, we run EfficientTAM [41] to obtain the segmentation of the object and remove the background of the ground truth image. Instead of optimizing the Gaussians to move separately, we force them to move rigidly. We apply a transformation $T \in \boldsymbol{SE}(3)$ on the means and covariance matrices of the object Gaussians. The photometric loss is calculated by comparing the mean squared error between the rendered images and the received camera images. The transformation is optimized to minimize photometric loss using 6 Adam optimization steps [42]. The correction force on each particle is calculated by $\boldsymbol{f}_i = K_p(\sum_j T(\boldsymbol{\mu}_j) - \boldsymbol{\mu}_j)/N_i$, where K_p is a tunable gain, $T(\boldsymbol{\mu}_j)$ is the transformed Gaussians mean, and N_i is the number of Gaussians bonded to each particle. The correction force and moment for each object are computed by aggregating the particle forces as $\boldsymbol{f} = \sum_i \boldsymbol{f}_i$ and $\boldsymbol{\tau} = \sum_i \boldsymbol{f}_i \times \boldsymbol{r}_i$, where $\boldsymbol{r}_i$ is the vector from the center of mass to the position of the transformed

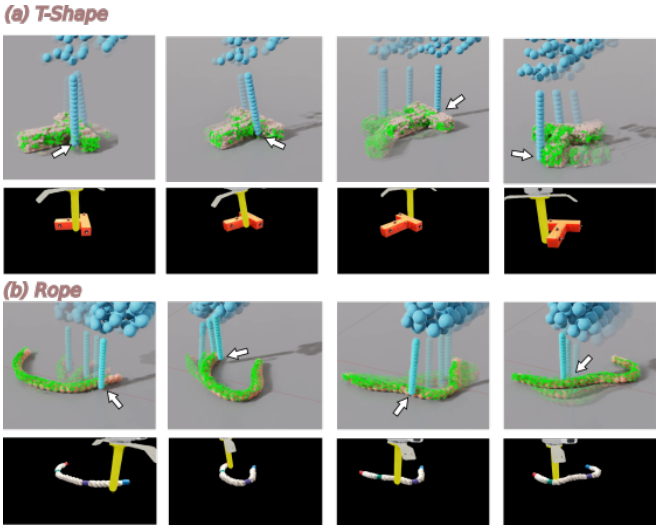

Fig. 3: We show qualitative results for tracking both the `rope` and `T-shaped block` objects. The ground truth, overlaid on the particle simulation, is shown in green, while the simulated spheres are depicted in blue. The arrow indicates the direction of the applied force. Below the particle representation, we show rendered 3D Gaussians throughout the experiment.

Gaussian. Finally, the robot links' poses are set according to the current joint configuration, the correction forces are added to the simulation, and one PBD simulation step is executed. Here, the object segmentation takes approximately 24 ms, pose optimization requires 10 ms, and the remaining 6 ms are spent on simulation and I/O, resulting in a total latency of about 40 ms.

V. Experimental Results

In this section, we validate our GaussTwin framework. To analyze its performance, we design a set of experiments to answer the following question: (i) *What matters the most for reliable object tracking, to have rigid body dynamics models or to perform shape matching between time steps?*; (ii) *Does adding shape segmentation improve our performance?*; (iii) *Can our system handle DLOs, such as a rope?*; and finally (iv) *Can we leverage GaussTwin prediction and correction steps to perform downstream tasks such as planning?*

A. Experimental Setup

1) Datasets: We evaluate our methods on both the simulated dataset from [14] and a real-world dataset. Here, the simulated dataset comprises three tasks: (i) single-object pushing, where a rigid object is placed on a white table and pushed horizontally by a pusher tool mounted at the robot's end-effector; (ii) multiple-object pushing, where one T-shaped and one I-shaped block are placed on the table and pushed by the pusher; and (iii) object pushover, where a T-shaped block standing upright on the table is first pushed over and then pushed horizontally. Each task includes five scenes, with trajectories lasting 8 seconds. Three cameras record the scenes at 1280×720 resolution and at 25 Hz frame rate. We note that [14] also includes a deformable object pushing and

the pickup task. However, as the physics prior deviated too much from the physics in the simulation, evaluation on this task was unreliable and thus not performed.

To collect the real-world dataset, we used a seven Degrees-of-Freedom (DoF) Franka Research 3 robot equipped with a 3D-printed cylindrical pusher. Four Intel RealSense D415 cameras were mounted at the corners of the robot's workspace to capture visual data, and a haptic interface Virtuose 6D was employed to teleoperate the robot. Here, we use all four cameras for initialization, and three for online tracking. Finally, we use OptiTrack markers for motion tracking to obtain the ground truth of the object position and orientation for evaluation. Our experimental setup is shown in Fig. 2-left and the set of objects used in the real-world experiments in Fig. 2-right. The objects include a `mug`, a `spam can` from the YCB dataset [43], two 3D-printed `T-shaped block` with different sizes and colors, a `Rubik's Cube`, and a `rope`. In the real-world dataset, four tasks similar to those in the simulation were performed. (i) single object pushing, where we teleoperated the robot to push rigid objects on the table; (ii) a pushover task, where we operated the robot to push down a standing `T-shaped block` to lie on the table, (iii) rope pushing, where we teleoperated the robot to push a rope on the table, and (iv) multiple-object pushing, where we placed two rigid objects on the table and teleoperated the robot to push them, including collisions between objects. The length of each trajectory ranges from 30 seconds to 40 seconds. Three cameras capture the scene at 848x480 resolution and 25 Hz framerate. Fig. 3 shows an example of both the `rope` and `T-shaped block` pushing tasks.

2) Baselines: We compare our method to two different baselines. Namely,

PEGS: Physically Embodied Gaussian Splatting (PEGS) [14] uses shape matching in PBD and optimizes the mean and quaternion of each individual Gaussian to derive the visual correction force.

RBD: Follows the model from [2]. Rigid Body Dynamics (RBD) leverages rigid body physics, similarly to us. However, their Gaussians optimization procedure is done in the same manner as in PEGS.

We note that, as our baselines do not fully work on deformable objects, we only compare on rigid-body tasks. Additionally, to verify the effectiveness of specific elements of our method, we also implemented two modified versions of our method for ablation:

GaussTwin (only mask): We provide the segmentation mask but optimize the Gaussians independently.

GaussTwin (only pose): We optimize the Gaussians coherently but do not provide the segmentation mask.

3) Metrics: On the simulated dataset, we evaluate the mean 3D trajectories error of known query points sampled on the object. At each time step, the query points are transformed according to the object pose in the PBD simulation after the correction step, as the prediction. On the real-world dataset, we compare the mean position and orientation errors of rigid bodies. For ropes, we evaluate the IoU between the rendered images using the 3D Gaussians transformed by the state of each rope segment in the PBD simulation after the correction step, and the segmented mask of the rope from the ground truth images using SAM2 [38]. We report the average value over three viewpoints.

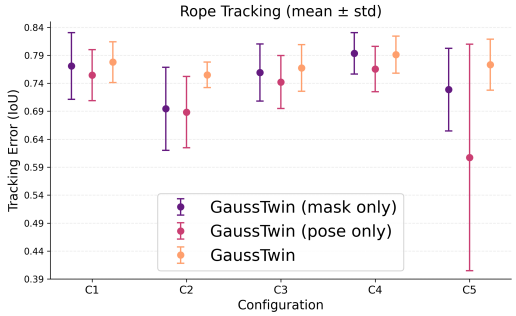

Fig. 4: Error bars showing rope tracking error for five different configurations. The error is measured as the IoU between ground-truth rope pixels and the projected spheres. We compare GaussTwin with two ablations: (i) GaussTwin using only the pose, and (ii) GaussTwin using only the mask.

TABLE I: Baseline comparison on the dataset from [14]

Method	Push†	Push-Down †	Multi-object
PEGS+M*	0.59 ± 0.61	3.72 ± 2.43	0.58 ± 0.81
PEGS+M+P*	0.54 ± 0.57	3.40 ± 2.12	0.44 ± 0.62
RBD	0.42 ± 0.45	1.21 ± 0.94	0.48 ± 0.74
GaussTwin	**0.34 ± 0.35**	**0.86 ± 0.60**	**0.38 ± 0.39**

Each entry is reported as mean in cm ± standard deviation.
† push and push-down tasks are reported for single objects on the scene.
* PEGS+M and PEGS+M+P follow our ablations with the additional masking and pose elements.

B. Results and Analysis

1) Simulated Dataset: As shown in Table I, our method outperforms baselines in all tasks with rigid bodies. For all experiments, our tracking error is consistently lower and exhibits reduced variance. Interestingly, we observe that RBD, despite not having segmentation masks, still performs well, likely because the simulated experiments have relatively short horizons and contain fewer occlusions. Moreover, methods that employ rigid-body dynamics generally outperform those based on shape matching, particularly in object push-down tasks, because the underlying physics simulation more faithfully captures object tumbling dynamics. It underscores the importance of using physically plausible constraints in PBD simulation.

2) Real-World Dataset: On the real-world dataset, we compare our method with baselines and ablations of our model. Table II shows that our model outperforms baselines in all tasks. Our model can stably track objects in various scenarios over a long horizon and maintain positional error within 1 cm. Comparing tracking error with and without masks, we observe that segmentation masks improve tracking performance, particularly for orientation tracking, at the cost of 24 ms of additional segmentation time. Nevertheless, the total latency remains 36 ms, which is suitable for real-time

TABLE II: Comparison of Methods on Different Real-World Tasks

	Single Object Push			Single Object Push Down			Multi-object		
Method	TE ($\pm$ std)	RE ($\pm$ std)	Lat.	TE ($\pm$ std)	RE ($\pm$ std)	Lat.	TE ($\pm$ std)	RE ($\pm$ std)	Lat.
SM [14]	3.39 $\pm$ 2.18	33.64 $\pm$ 23.61	11	1.12 $\pm$ 0.91	6.19 $\pm$ 5.84	11	5.65 $\pm$ 4.77	29.35 $\pm$ 26.43	13
RBD [2]	3.49 $\pm$ 2.84	17.56 $\pm$ 20.01	**10**	0.99 $\pm$ 0.43	5.41 $\pm$ 4.77	**10**	4.06 $\pm$ 3.46	18.86 $\pm$ 18.97	**12**
GaussTwin (only mask)	0.60 $\pm$ 0.42	4.85 $\pm$ 3.32	35	0.91 $\pm$ 0.45	4.71 $\pm$ 3.50	35	2.08 $\pm$ 1.91	15.32 $\pm$ 15.08	40
GaussTwin (only pose)	3.40 $\pm$ 4.34	18.72 $\pm$ 20.31	11	1.15 $\pm$ 0.49	4.95 $\pm$ 3.43	11	10.18 $\pm$ 9.69	51.63 $\pm$ 42.25	13
GaussTwin (full)	**0.43 $\pm$ 0.17**	**3.32 $\pm$ 1.46**	36	**0.70 $\pm$ 0.32**	**3.01 $\pm$ 1.80**	36	**0.85 $\pm$ 0.45**	**7.87 $\pm$ 6.11**	42

TE = Mean translation error in cm across multiple objects and the entire trajectory ($\pm$ standard deviation).
RE = Mean rotation error in degrees and the entire trajectory ($\pm$ standard deviation).
Lat. = Mean runtime latency in ms.

usage. In the single-object pushing task, models that do not use the segmentation mask all fail to follow the object over a long horizon. Comparing variants of method using rigid body dynamics, we find that coherently optimizing the Gaussians yields lower mean error and variance across all scenarios, thereby verifying the effectiveness of our optimization strategy. Furthermore, most methods fail to track the orientation of objects correctly in the multiple-object pushing task because the scene includes symmetric objects, such as the mug and Rubik's Cube. While best among all methods, the rotation error of GaussTwin is still relatively large, since it is difficult to correct errors using the photometric loss on textureless, symmetric objects once their orientation deviates too far from ground truth.

For the rope-pushing scenario, our model successfully tracks the dynamic deformation of the rope under various pushing trajectories, which is shown in Fig. 4. Comparing the IoU of our model and its variations, we observe that without the mask, the tracking performance drops significantly. Although the IoU is affected by the fact that the rendered images from 3D Gaussians occupy more pixels than the ground truth mask, we still achieve a value over 0.75 in all scenarios. It is noteworthy that optimizing the Gaussians coherently for each segment of the rope also improves the tracking performance for deformable objects. We also show the Gaussian rendering effect during tracking in Fig. 3.

3) Model-Based Planning: We also evaluated our particle-based dynamics model (without visual correction) for planning a sequence of pushes to align a T-shaped object with randomly sampled positions and orientations. We parameterized each push as a four-dimensional action specifying the initial and final end effector position relative to the object center, where we used polar coordinates for the initial position and planar Cartesian coordinates for the final position. We specified a reward function consisting of five terms: two for penalizing the squared position and absolute yaw error, a penalty for starting the push too close to the object, to prevent initializing the push inside of the object, a Gaussian prior on the polar-coordinate radius at $r = 0.15$, and a Gaussian zero mean prior on the final position. To optimize a push, we used GMMVI [44] to learn a 2-component Gaussian mixture model to approximate a target distribution with log-densities given by the reward function and executed a sample with high reward. We stopped the pushing sequence when no push that led to a significant improvement was found, which typically happened after two pushes. As shown in Table III, the planned motion sequence could align the object with a position error of around 1 cm.

TABLE III: Mean and standard deviation of reward, position and yaw errors after the first and final push.

	Reward	Pos. error (cm)	Yaw error (rad.)
1st Push	-41.5 ± 41.2	1.4 ± 0.7	0.37 ± 0.42
Last Push	-3.48 ± 2.92	1.2 ± 0.7	0.01 ± 0.01

Each entry is reported as mean $\pm$ standard deviation.

VI. Conclusion and Future Work

In this work, we introduced GaussTwin, a hybrid framework that unifies position-based dynamics with 3D Gaussian splatting to enable real-time digital twins for both rigid and deformable objects. Our evaluation was structured around four guiding questions. First, we asked whether reliable tracking depends more on rigid-body dynamics or shape matching. Our results show that physically grounded rigid-body dynamics significantly outperform shape-matching approaches, especially for tasks involving complex object motions. Second, we asked if adding segmentation masks improves performance. We found that segmentation consistently enhances both positional and rotational accuracy, confirming the value of visual feedback in maintaining stability over long horizons. Third, we examined whether GaussTwin can handle deformable linear objects such as ropes. Our experiments demonstrate that it can robustly track and correct DLOs, validating the effectiveness of extending PBD with Cosserat rod models and coherent Gaussian optimization. Finally, we asked whether GaussTwin's prediction–correction loop can support downstream tasks such as planning. We showed that it enables model-based push planning with centimeter-level accuracy, underscoring its utility for real-world manipulation. By explicitly addressing these four questions, we establish GaussTwin as a unified and physically consistent digital twin framework that bridges the real-to-sim gap. For future work, we plan to extend GaussTwin with automatic parameter estimation and integrate it into vision-based policy learning pipelines.

REFERENCES

[1] J. Yu, L. Fu, H. Huang, K. El-Refai, R. A. Ambrus, R. Cheng, M. Z. Irshad, and K. Goldberg, "Real2render2real: Scaling robot data without dynamics simulation or robot hardware," *arXiv preprint arXiv:2505.09601*, 2025.

[2] J. Abou-Chakra, L. Sun, K. Rana, B. May, K. Schmeckpeper, M. V. Minniti, and L. Herlant, "Real-is-sim: Bridging the sim-to-real gap with a dynamic digital twin for real-world robot policy evaluation," *arXiv preprint arXiv:2504.03597*, 2025.

[3] J. Schulman, A. Lee, J. Ho, and P. Abbeel, "Tracking deformable objects with point clouds," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2013, pp. 1130–1137.

[4] T. Tang, Y. Fan, H.-C. Lin, and M. Tomizuka, "State estimation for deformable objects by point registration and dynamic simulation," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2017, pp. 2427–2433.

[5] F. Liu, Z. Li, Y. Han, J. Lu, F. Richter, and M. C. Yip, "Real-to-sim registration of deformable soft tissue with position-based dynamics for surgical robot autonomy," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2021, pp. 12 328–12 334.

[6] X. Liang, F. Liu, Y. Zhang, Y. Li, S. Lin, and M. Yip, "Real-to-sim deformable object manipulation: Optimizing physics models with residual mappings for robotic surgery," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2024, pp. 15 471–15 477.

[7] A. Petit, V. Lippiello, and B. Siciliano, "Real-time tracking of 3d elastic objects with an rgb-d sensor," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2015, pp. 3914–3921.

[8] J. Abou-Chakra, F. Dayoub, and N. Sünderhauf, "Particlenerf: A particle-based encoding for online neural radiance fields," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2024, pp. 5975–5984.

[9] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering." *ACM Trans. Graph.*, vol. 42, no. 4, pp. 139–1, 2023.

[10] M. Zhang, K. Zhang, and Y. Li, "Dynamic 3d gaussian tracking for graph-based neural dynamics modeling," in *Conference on Robot Learning (CoRL)*, 2024.

[11] H. Jiang, H.-Y. Hsu, K. Zhang, H.-N. Yu, S. Wang, and Y. Li, "Phystwin: Physics-informed reconstruction and simulation of deformable objects from videos," in *Intl. Conf. on Computer Vision (ICCV)*, 2025.

[12] K. Zhang, B. Li, K. Hauser, and Y. Li, "Particle-grid neural dynamics for learning deformable object models from rgb-d videos," *Robotics: Science and Systems (R:SS)*, 2025.

[13] J. Xiang, H. Dinkel, H. Zhao, N. Gao, B. Coltin, T. Smith, and T. Bretl, "Trackdlo: Tracking deformable linear objects under occlusion with motion coherence," *IEEE Robotics and Automation Letters*, vol. 8, no. 10, pp. 6179–6186, 2023.

[14] J. Abou-Chakra, K. Rana, F. Dayoub, and N. Suenderhauf, "Physically embodied gaussian splatting: A realtime correctable world model for robotics," in *Conference on Robot Learning (CoRL)*, 2024.

[15] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff, "Position based dynamics," *Journal of Visual Communication and Image Representation*, vol. 18, no. 2, pp. 109–118, 2007.

[16] M. Müller, B. Heidelberger, M. Teschner, and M. Gross, "Meshless deformations based on shape matching," *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3, pp. 471–478, 2005.

[17] M. Müller and N. Chentanez, "Solid simulation with oriented particles," in *ACM SIGGRAPH 2011 papers*, 2011, pp. 1–10.

[18] T. Kugelstadt and E. Schömer, "Position and orientation based cosserat rods." in *Symposium on Comp. Animation*, vol. 11, 2016, pp. 169–178.

[19] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2012, pp. 5026–5033.

[20] NVIDIA, "Isaac Sim," https://github.com/isaac-sim/IsaacSim.

[21] R. Laezza, R. Gieselmann, F. T. Pokorny, and Y. Karayiannidis, "Reform: A robot learning sandbox for deformable linear object manipulation," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2021, pp. 4717–4723.

[22] Y. Yang, J. A. Stork, and T. Stoyanov, "Learning differentiable dynamics models for shape control of deformable linear objects," *Robotics and Autonomous Systems*, vol. 158, p. 104258, 2022.

[23] M. Macklin, "Warp: A high-performance python framework for gpu simulation and graphics," https://github.com/nvidia/warp, March 2022, NVIDIA GPU Technology Conference (GTC).

[24] M. Geilinger, D. Hahn, J. Zehnder, M. Bächer, B. Thomaszewski, and S. Coros, "Add: Analytically differentiable dynamics for multi-body systems with frictional contact," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 6, pp. 1–15, 2020.

[25] T. Du, K. Wu, P. Ma, S. Wah, A. Spielberg, D. Rus, and W. Matusik, "Diffpd: Differentiable projective dynamics," *ACM Transactions on Graphics (ToG)*, vol. 41, no. 2, pp. 1–21, 2021.

[26] Y. Qiao, J. Liang, V. Koltun, and M. Lin, "Differentiable simulation of soft multi-body systems," *Advances in Neural Information Processing Systems*, vol. 34, pp. 17 123–17 135, 2021.

[27] P. Sundaresan, R. Antonova, and J. Bohgl, "Diffcloud: Real-to-sim from point clouds with differentiable simulation and rendering of deformable objects," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2022, pp. 10 828–10 835.

[28] F. Liu, E. Su, J. Lu, M. Li, and M. C. Yip, "Robotic manipulation of deformable rope-like objects using differentiable compliant position-based dynamics," *IEEE Robotics and Automation Letters*, vol. 8, no. 7, pp. 3964–3971, 2023.

[29] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[30] B. Evans, A. Thankaraj, and L. Pinto, "Context is everything: Implicit identification for dynamics adaptation," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2022, pp. 2642–2648.

[31] M. Yan, Y. Zhu, N. Jin, and J. Bohg, "Self-supervised learning of state estimation for manipulating deformable linear objects," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2372–2379, 2020.

[32] B. Ai, S. Tian, H. Shi, Y. Wang, C. Tan, Y. Li, and J. Wu, "Robopack: Learning tactile-informed dynamics models for dense packing," in *ICRA Workshop on 3D Visual Representations for Robot Manipulation*, 2024.

[33] H. Shi, H. Xu, Z. Huang, Y. Li, and J. Wu, "Robocraft: Learning to see, simulate, and shape elasto-plastic objects in 3d with graph networks," *Intl. Journal of Robotics Research*, vol. 43, no. 4, pp. 533–549, 2024.

[34] H. Dinkel, M. Büsching, A. Longhini, B. Coltin, T. Smith, D. Kragic, M. Björkman, and T. Bretl, "Dlo-splatting: Tracking deformable linear objects using 3d gaussian splatting," *arXiv preprint arXiv:2505.08644*, 2025.

[35] J. Bender, M. Müller, and M. Macklin, "A survey on position based dynamics, 2017," *Proceedings of the European Association for Computer Graphics: Tutorials*, pp. 1–31, 2017.

[36] C. Deul, P. Charrier, and J. Bender, "Position-based rigid-body dynamics," *Computer Animation and Virtual Worlds*, vol. 27, no. 2, pp. 103–112, 2016.

[37] M. Müller, M. Macklin, N. Chentanez, S. Jeschke, and T.-Y. Kim, "Detailed rigid body simulation with extended position based dynamics," in *Computer Graphics Forum*, vol. 39, no. 8. Wiley Online Library, 2020, pp. 101–112.

[38] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson *et al.*, "Sam 2: Segment anything in images and videos," *arXiv preprint arXiv:2408.00714*, 2024.

[39] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[40] T.-C. Lee, R. L. Kashyap, and C.-N. Chu, "Building skeleton models via 3-d medial surface axis thinning algorithms," *CVGIP: graphical models and image processing*, vol. 56, no. 6, pp. 462–478, 1994.

[41] Y. Xiong, C. Zhou, X. Xiang, L. Wu, C. Zhu, Z. Liu, S. Suri, B. Varadarajan, R. Akula, F. Iandola *et al.*, "Efficient track anything," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025, pp. 11 513–11 524.

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[43] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Benchmarking in manipulation research: The ycb object and model set and benchmarking protocols," *arXiv preprint arXiv:1502.03143*, 2015.

[44] O. Arenz, P. Dahlinger, Z. Ye, M. Volpp, and G. Neumann, "A unified perspective on natural gradient variational inference with gaussian mixture models," *Transactions on Machine Learning Research*, 2023.