RigidFormer: Learning Rigid Dynamics using Transformers

Zhiyang Dou[1] **Minghao Guo**[1] **Haixu Wu**[1] **Doug Roble**[2]
Tuur Stuyck[2] **Wojciech Matusik**[1]

[1] MIT [2] Meta

Abstract

Learning-based simulation of multi-object rigid-body dynamics remains difficult because contact is discontinuous and errors compound over long horizons. Most existing methods remain tied to mesh connectivity and vertex-level message passing, which limits their applicability to mesh-free inputs such as point clouds and leads to high computational cost. Efficiently modeling high-fidelity rigid-body dynamics from mesh-free representations therefore remains challenging. We introduce **RigidFormer**, an object-centric Transformer-based model that learns mesh-free rigid-body dynamics with controllable integration step sizes. RigidFormer reasons at the *object level* and advances each object through compact anchors; *Anchor-Vertex Pooling* enriches these anchors with local vertex features, retaining contact-relevant geometry without dense vertex-level interaction. We propose *Anchor-based RoPE* to inject anchor geometry into attention while respecting the unordered nature of objects and anchors: object-token processing is permutation-equivariant, and the mean-pooled anchor descriptor is invariant to anchor reindexing while preserving shape extent. RigidFormer further enforces *rigidity* by projecting updates onto the rigid-body manifold using differentiable Kabsch alignment. On standard benchmarks, RigidFormer outperforms or matches mesh-based baselines using point inputs, runs faster, generalizes to unseen point resolutions and across datasets, and scales to 200+ objects; we also show a preliminary extension to command-conditioned articulated bodies by treating body parts as interacting object-level components. Code will be released upon publication.

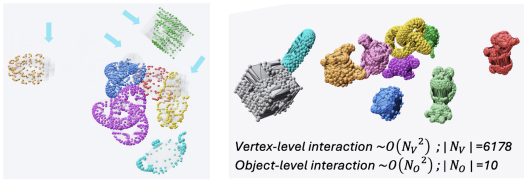
1 Introduction

Rigid-body dynamics arises throughout robotics, graphics, and embodied AI. With accurate meshes, reliable physical parameters, and well-tuned contact models, classical physics engines [9, 11, 23, 24, 35] can produce faithful trajectories. In practice, however, these prerequisites are often missing: objects may only be available as imperfect or incomplete geometry (e.g., polygon soups or point clouds), with contact properties that are difficult to calibrate. This motivates *mesh-free* modeling. A common choice is a point-based representation, which is easy to acquire, topology-free, and resolution-flexible, making it a natural interface between perception and dynamic scene modeling [7, 21, 30, 31, 39, 47, 48]. It also integrates naturally with modern generative pipelines [17, 36, 43, 49]. However, despite the appeal of point-based inputs, many state-of-the-art learned simulators remain mesh-dependent [1, 29, 33, 40, 44], requiring explicit edge and face connectivity that is not available for point inputs. Moreover, since they typically operate at the vertex-, edge-, or facet-level, their computational costs grow rapidly with resolution, significantly limiting inference efficiency.

We present RigidFormer, an object-centric Transformer-based model that learns mesh-free multi-object rigid-body dynamics. Our design is guided by three observations. First, a rigid body responds to an impulse as a coherent whole: interaction effects do not need to "diffuse" across surface

Preprint.

vertices edge by edge, as in vertex-centric simulators based on local message passing [1, 29, 33], which can introduce substantial computational overhead and slow down inference. RigidFormer therefore adopts an object-centric representation that reasons over objects rather than vertices: it takes object-level point clouds as input, even partial ones (see Fig. 1 (a)), and encodes each object into a compact token without requiring connectivity. Interactions are then modeled primarily among *object tokens*, matching the rigid-body assumption that each object moves as a coherent whole; see Fig. 1 (b). This shift greatly improves efficiency, e.g., 23.9 FPS versus 3.0 FPS, while maintaining simulation quality. We implement this design with Transformers [37], whose attention mechanism flexibly captures multi-object interactions without relying on hand-designed graphs.



(a) (b)

Figure 1: (a) Dynamics modeling from partial point clouds. (b) Object-level interactions reduce the complexity from vertex-level $O(N_V^2)$ to object-level $O(N_O^2)$.

Second, we exploit the low-dimensional structure of rigid-body motion during state advancement. Although an object may contain thousands of points, rigid motion lives in a low-dimensional space (6-DoF per object). Therefore, we solve for each object's state update using a small number of *Anchors*, which enables efficient and geometry-aware dynamics updates. Anchor/keypoint and $SE(3)$ representations have precedent in pose tracking and manipulation models [5, 38]; here, anchors serve as learned simulation states for long-horizon contact dynamics rather than tracked category keypoints or directly regressed part transforms. Given the central role of positional embeddings in Transformer generalization [15, 34, 46], we propose an *Anchor-based Rotary Positional Embedding (ARoPE)* (Sec. 3.3) to encode object geometry for attention. Its symmetry properties are deliberately scoped: the object Transformer remains equivariant to object-token permutations because no sequence-index positional embeddings are used, while the mean-pooled ARoPE descriptor is invariant to anchor reindexing within each object. A distance-kernel *Anchor-Vertex Pooling* module further supplies local contact geometry with vertex-order-invariant aggregation. Rather than directly regressing rotations and translations, which can be error-prone [51], we predict anchor motions and obtain the rigid transform via *differentiable rigid projection*, which projects updates onto the rigid-body manifold while preserving gradient flow, improving long-horizon stability. Finally, inspired by time-conditioned neural simulation [8, 45, 50], RigidFormer conditions on the temporal discretization, enabling a single model to operate across step sizes (Sec. 4.1). Larger Δt improves long-horizon accuracy by reducing autoregressive error accumulation, while smaller Δt captures finer temporal detail when needed.

As a result, RigidFormer offers an efficient, stable, and scalable framework for rigid-body dynamics modeling; a comparison with previous methods is in Tab. 1. On MOVi [14], it matches or surpasses prior mesh-based methods using only point positions, without requiring mesh connectivity. It leverages known material parameters when available, generalizes to unseen point resolutions and across datasets, supports step-size control, and scales beyond 200 objects while maintaining both accuracy and high inference speed. RigidFormer also handles partial point cloud inputs (see Fig. A6). We further show a preliminary application of the same object-anchor design to controllable articulated bodies by treating body parts as interacting object-level components. We summarize our contributions below:

Table 1: **Method comparison.** Mesh-free: no triangle connectivity required. Variable step: handles multiple Δt. Preproc.-free: no offline geometry computation. Warmup frames: number of input frames required for rollout.

Method	Mesh-Free	Var. Δt	Preproc.-Free	#Warmup Frames ↓	Runtime (FPS) ↑
MGN	✗	✗	✓	**2**	5.7
FIGNet	✗	✗	✓	3	3.0
SDF-Sim	✗	✗	✗[†]	3	–
HopNet	✗	✗	✗[‡]	3	0.2
Ours	✓	✓	✓	**2**	**23.9**

[†] SDF pre-learning: ∼5 hours for MOVi-B.
[‡] Simplicial-complex construction: ∼15 days on MOVi-B.

- We introduce an efficient and scalable mesh-free Transformer-based neural simulator named RigidFormer for multi-object rigid-body dynamics from point representations, supporting simulation across different time-step sizes.
- We propose an object-level formulation with Anchor-based RoPE for geometry-aware attention with explicit object-token permutation equivariance and anchor-order invariance, along with vertex-

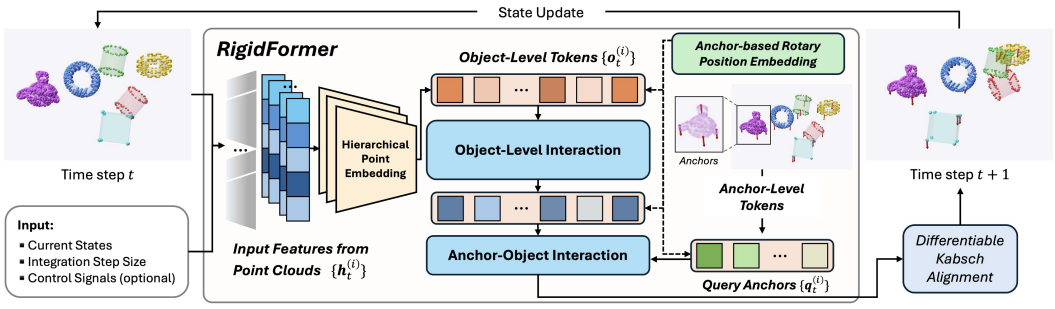


Figure 2: **RigidFormer Pipeline.** Inputs include two recent point-level states, the time-step size, and optional control signals. At time t, we encode each object point cloud into an object token. *Object-Level Interaction* models direct object–object effects, while *Anchor-Object Interaction* lets anchors attend to multi-scale object features and cross-object context. Each object's state is advanced efficiently using a compact set of anchors. We integrate anchor dynamics using Verlet integration followed by differentiable Kabsch alignment to enforce rigidity and obtain the state at $t+1$. Anchor-based RoPE is used to improve generalization across object counts and geometries.

order-invariant Anchor-Vertex Pooling and a low-dimensional anchor state advance that reduces complexity; rigidity and long-horizon stability are enforced via projection onto the rigid-body manifold during the simulation.

- We validate RigidFormer across diverse experiments, demonstrating fast inference, generalization, scalability, and a preliminary application to command-conditioned articulated bodies.

2 Related Work

Classical numerical rigid-body simulators [2, 9, 24, 35] resolve contact by solving constrained optimization or complementarity problems. Differentiable simulators (e.g., DiffTaichi [16], Warp [23], and Brax [11]) enable gradient-based learning and inverse problems, but they rely on explicit physics engines and typically assume mesh-based geometry rather than mesh-free point inputs.

Early learning-based dynamics models often targeted relatively simple systems with explicit, low-dimensional state representations, typically in 2D. Interaction Networks [3] and Neural Physics Engine [6] established object- and relation-centric inductive biases and motivated graph-based simulators for dynamics modeling. In rigid-body systems, state-of-the-art neural simulators typically rely on mesh-based inputs in order to faithfully capture the dynamics [1, 29, 33, 40]. MeshGraphNets [29] extend message passing to mesh discretizations and achieve strong performance for mesh-based simulation. FIGNet [1] improves collision modeling by constructing interactions over mesh faces rather than nodes. HopNet [40] incorporates higher-order topology and physics-informed message passing for rigid interactions; however, obtaining the required topological structures can be expensive. HCMT [44] uses hierarchical mesh structures and Transformer-style long-range modeling for collision-induced dependencies in flexible-body collision dynamics in the 2D domain. All aforementioned methods require mesh connectivity and incur substantial vertex-level interaction cost as resolution grows. SDF-Sim [33] represents shapes with learned signed distance functions, reducing collision-handling bottlenecks but requiring additional shape learning. Compared to these works, RigidFormer is mesh-free: it models rigid-body dynamics using point inputs, shifts interaction reasoning to the object level, and uses anchor-based advance to avoid dense vertex interactions at inference time; see Tab. 1. Rigid-motion and keypoint-based representations have also appeared in related robotics settings. SE3-Nets [5] predict rigid $SE(3)$ transforms for object parts from point clouds and action inputs, demonstrating the value of rigid-motion inductive bias for manipulation. 6-PACK [38] learns anchor-based 3D keypoints for category-level 6D pose tracking. These works are complementary to ours: their keypoints are used for pose estimation in manipulation or tracking, whereas ours are for more dynamic simulation states that are advanced by learned dynamics, coupled with geometry-aware ARoPE and differentiable rigid projection for long-horizon multi-object rollout.

Point-based representations for dynamics have been explored recently. Kim and Fuxin [19] propose a hierarchical point-cloud representation with continuous point convolutions to improve contact accuracy. Whitney et al. [41, 42] learn point-based dynamics by disentangling visual observations from physical states, but accuracy degrades in contact-rich regimes due to the coupling. In contrast,

RigidFormer adopts an object-level Transformer that effectively models inter-object interactions for points, leading to higher-quality dynamics prediction.

3 Methodology

We consider a system of M rigid objects, where M can vary across scenes during both training and inference. Object i is represented by a point set $\mathbf{x}_t^{(i)} \in \mathbb{R}^{N_v^{(i)} \times d}$ at time t, where $N_v^{(i)}$ is the number of points and d denotes the per-point feature dimension, and the full state is $\mathbf{x}_t = \{\mathbf{x}_t^{(i)}\}_{i=1}^{M}$. We learn a model f_θ that updates the next state from two consecutive observations: $\mathbf{x}_{t+1} = f_\theta(\mathbf{x}_{t-1}, \mathbf{x}_t, \Delta t)$. An overview of our pipeline is in Fig. 2.

3.1 Object-Centric Interaction Modeling

We concatenate, for each vertex of object i: (1) the nearest-neighbor displacement vector $\mathbf{d}_t^{(i)} \in \mathbb{R}^{N_v^{(i)} \times 3}$ from the vertex to the nearest point on another object or the ground plane, (2) the per-step position increment $\mathbf{v}_t^{(i)} = \mathbf{x}_t^{(i)} - \mathbf{x}_{t-1}^{(i)} \in \mathbb{R}^{N_v^{(i)} \times 3}$ (used as a discrete velocity surrogate), (3) the reference-offset $\mathbf{r}_t^{(i)} = \mathbf{x}_t^{(i)} - \mathbf{x}_{\text{ref}}^{(i)} \in \mathbb{R}^{N_v^{(i)} \times 3}$ where the reference is the first frame in the sequence, and (4) physics parameters $\boldsymbol{\phi}^{(i)} = [m, \mu, \epsilon] \in \mathbb{R}^3$ (mass, friction, restitution), broadcast to every vertex of object i. These yield the input feature $\mathbf{h}_t^{(i)} \in \mathbb{R}^{N_v^{(i)} \times 12}$. Inspired by PointNet [30], we build an encoder Enc_θ with hierarchical feature extraction to aggregate per-vertex features into a fixed-dimensional object embedding: $\mathbf{o}_t^{(i)} = \mathrm{Enc}_\theta(\mathbf{h}_t^{(i)}) \in \mathbb{R}^D$. After computing per-vertex features, we extract multi-scale geometry at the global level and three subsampled levels; these features are then concatenated and fused into one object-level embedding. This design captures both fine-grained local geometry and coarse global structure while remaining robust to variable vertex counts (see validation in Sec. 4.1). The encoder is *shared* across all objects, promoting generalization to various geometries. Using one token per object drastically shortens the Transformer sequence compared with vertex-level modeling, substantially improving efficiency; see Appendix F.1.

Given object embeddings $\mathbf{O}_t = [\mathbf{o}_t^{(1)}, \ldots, \mathbf{o}_t^{(M)}] \in \mathbb{R}^{M \times D}$, our decoder is a stack of L Transformer blocks that takes as input $\mathbf{O}_t$ concatenated with N_r=16 learned register tokens. For clarity, we describe the object-token update and omit the registers from notation. At layer ℓ, the decoder applies residual self-attention, step-size FiLM conditioning [28], and a residual feed-forward update: $\tilde{\mathbf{Z}}_t^{(\ell)} = \mathrm{SelfAttn}(\mathbf{Z}_t^{(\ell-1)}) + \mathbf{Z}_t^{(\ell-1)}$, $\hat{\mathbf{Z}}_t^{(\ell)} = \boldsymbol{\gamma}_\ell(\mathbf{c}) \odot \tilde{\mathbf{Z}}_t^{(\ell)} + \boldsymbol{\beta}_\ell(\mathbf{c})$, and $\mathbf{Z}_t^{(\ell)} = \mathrm{FFN}(\hat{\mathbf{Z}}_t^{(\ell)}) + \hat{\mathbf{Z}}_t^{(\ell)}$, for $\ell = 1, \ldots, L$. The FiLM code $\mathbf{c} = (s, s^2) \in \mathbb{R}^2$ encodes the integration step size, where $s \propto \Delta t$ captures first-order time scaling and s^2 mirrors the Δt^2 factor in Verlet integration. The layer-specific MLPs $\boldsymbol{\gamma}_\ell$ and $\boldsymbol{\beta}_\ell$ produce channel-wise scale and shift parameters, allowing the same decoder to adapt its features across different temporal discretizations. Motivated by gated attention in language modeling [32], we modulate each attention update with a query-conditioned sigmoid gate: $\mathbf{y} = \sigma(\mathbf{G}(\mathbf{Q})) \odot \mathrm{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, where $\mathbf{G}(\mathbf{Q})$ has the same per-head channel shape as the attention output. In our setting, the gate acts as a learned attenuator for noisy or weakly relevant interaction reads, which stabilizes autoregressive dynamics rollouts and improves long-horizon accuracy (Sec. 4.2). The output of this stage is a set of updated object-level tokens $\{\mathbf{Z}_t^{(L)}\}$ that summarize scene context and object interactions. We next use anchors as queries into these object tokens, combine the retrieved context with local vertex features, and advance the system through anchor dynamics.

3.2 Anchor-based State Advance

Rigid-body motion is low-dimensional (6-DoF) even when an object contains thousands of points. Therefore, directly updating all vertices is expensive and redundant: dense attention over MN_v points costs $O((MN_v)^2)$, and per-vertex regression destabilizes prediction. Meanwhile, regressing rotations and translations directly [51] can be error-prone and unstable due to discontinuities in common $SE(3)$ parameterizations. In RigidFormer, we instead select a small set of N_a=4 *anchors* per object using FPS [13, 31], reducing the interaction cost to $O((MN_a)^2)$ with $N_a \ll N_v$ across timesteps. We form anchor queries by extracting features at anchor locations and projecting them with an MLP to obtain $\mathbf{Q}_t \in \mathbb{R}^{(MN_a) \times D}$. Each anchor query attends to the decoder object tokens via

cross-attention, retrieves cross-object interaction context, and the network then predicts a per-anchor acceleration $\mathbf{a}_t^{(i,k)}$. Implementation details of the predictor are provided in Appendix G.

Anchor queries summarize a rigid object's dynamics, but accurate acceleration prediction during contact depends strongly on which vertices lie close to the contact site. To inject this fine-grained, contact-local geometry into each anchor without paying the cost of full per-vertex attention, we attach an Anchor-Vertex Pooling (AVP) module that aggregates per-vertex encoder features around each anchor with a learnable isotropic distance kernel:

$$\mathbf{u}_t^{(i,k)} = \frac{\sum_{v=1}^{N_v^{(i)}} w_t^{(i,k,v)}\, \mathbf{f}_t^{(i,v)}}{\sum_{v=1}^{N_v^{(i)}} w_t^{(i,k,v)}}, \quad w_t^{(i,k,v)} = \exp\left(-\frac{\|\mathbf{x}_t^{(i,v)} - \mathbf{q}_t^{(i,k)}\|}{\sigma}\right),$$

where $\mathbf{f}_t^{(i,v)}$ is the encoder feature at vertex v, $\mathbf{q}_t^{(i,k)}$ is the position of anchor k, and the kernel bandwidth σ is trained jointly with the rest of the network; padded vertices are masked in the batched implementation. Because the weights depend only on point-anchor distances and the aggregation is a normalized sum, AVP is invariant to vertex ordering, and its attention weights are unchanged by a common rigid transform of the point and anchor coordinates. The pooled feature is then passed through a lightweight MLP to obtain $\mathbf{u}_t^{(i,k)} \in \mathbb{R}^{256}$, which is concatenated to the anchor query before predicting acceleration, enriching it with collision-aware local context. We advance anchors with Verlet integration to obtain candidate anchor positions from predicted accelerations: $\hat{\mathbf{q}}_{t+1}^{(i,k)} = \mathbf{a}_t^{(i,k)}\Delta t^2 + 2\mathbf{q}_t^{(i,k)} - \mathbf{q}_{t-1}^{(i,k)}$.

Scatter to All Vertices via Differentiable Rigid Projection. We recover the rigid transform $(\mathbf{R}^{(i)}, \mathbf{t}^{(i)}) \in SE(3)$ by aligning reference anchors $\mathbf{q}_{\text{ref}}^{(i,k)}$ to the candidate anchors $\hat{\mathbf{q}}_{t+1}^{(i,k)}$ using Kabsch alignment [18]:

$$\mathbf{H} = \sum_k (\mathbf{q}_{\text{ref}}^{(i,k)} - \bar{\mathbf{q}}_{\text{ref}}^{(i)})(\hat{\mathbf{q}}_{t+1}^{(i,k)} - \bar{\hat{\mathbf{q}}}^{(i)})^\top, \quad \mathbf{U}, \mathbf{\Sigma}, \mathbf{V}^\top = \text{SVD}(\mathbf{H}),$$

$$\mathbf{R}^{(i)} = \mathbf{V}\,\text{diag}(1, 1, \det(\mathbf{V}\mathbf{U}^\top))\mathbf{U}^\top, \quad \mathbf{t}^{(i)} = \bar{\hat{\mathbf{q}}}^{(i)} - \mathbf{R}^{(i)}\bar{\mathbf{q}}_{\text{ref}}^{(i)},$$

where k indexes anchors, and $\bar{\hat{\mathbf{q}}}^{(i)}$ and $\bar{\mathbf{q}}_{\text{ref}}^{(i)}$ denote centroids of the predicted and reference anchor sets. Then, we update the full-resolution point set by broadcasting the transform to all vertices: $\mathbf{x}_{t+1}^{(i,v)} = \mathbf{R}^{(i)}\mathbf{x}_{\text{ref}}^{(i,v)} + \mathbf{t}^{(i)}, \quad \forall v \in \{1, \ldots, N_v\}$. This projection enforces rigidity by construction and improves long-horizon rollout stability. Since gradients through SVD can be unstable near degenerate singular values, we implement rigid registration with RoMa [4] for robust differentiability.

3.3 Anchor-based Rotary Positional Embedding

As a Transformer-based model, effective position embeddings are essential for RigidFormer to generalize rigid-body dynamics, where interactions vary with object count and 3D geometry. In RigidFormer, object tokens have no inherent ordering—permuting input objects should reorder the predicted dynamics by the same permutation, i.e., the model is permutation-equivariant over objects. Moreover, contact outcomes depend on relative 3D geometry rather than absolute indices. Naively encoding each object with a single point (e.g., its centroid) discards shape- and state-dependent cues needed for accurate collisions, while encoding all vertices is largely redundant, hurts generalization, and adds prohibitive computational overhead.

We propose *Anchor-based Rotary Positional Embedding (ARoPE)*, which encodes the spatial extent of each object using a sparse set of anchor positions, making the attention geometry-aware while remaining efficient and generalizable across different numbers of objects with various shapes. Concretely, for object i with anchors $\{\mathbf{x}_k^{(i)}\}_{k=1}^{N_a} \subset \mathbb{R}^3$ (N_a=4), we apply a shared 3D rotary anchor map $\psi_\omega(\cdot)$ to each anchor, mapping each coordinate through d_c=32 rotary phase channels to obtain a per-anchor 96-D phase descriptor. We then aggregate the per-anchor descriptors by mean-pooling:

$$\text{ARoPE}(\{\mathbf{x}_k^{(i)}\}) = \frac{1}{N_a}\sum_{k=1}^{N_a} \underbrace{\bigoplus_{j=1}^{3}\left[\{\omega_l x_{k,j}^{(i)}\}_{l=1}^{d_c/2};\, \{\omega_l x_{k,j}^{(i)}\}_{l=1}^{d_c/2}\right]}_{\psi_\omega(\mathbf{x}_k^{(i)})}.$$

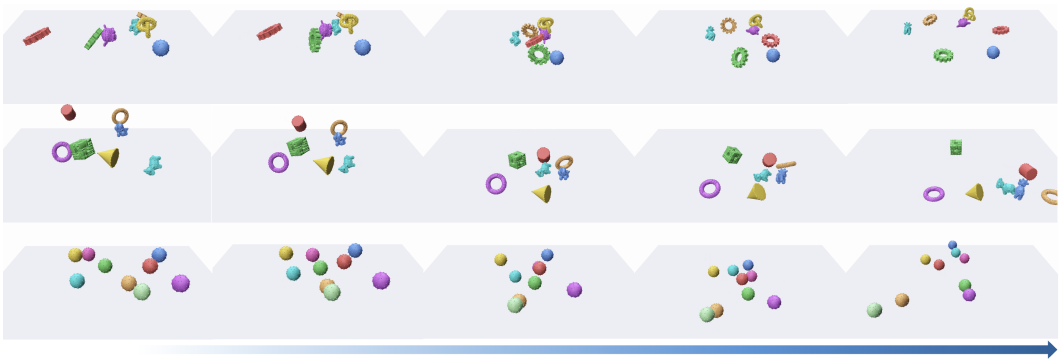


Figure 3: **Qualitative results from RigidFormer.** Meshes are shown only for visualization; our model operates on point inputs. Additional visualization results on the MOVi datasets, including partial point-cloud inputs, are provided in Appendix A.

where ω_l are log-spaced frequencies, $\oplus$ denotes concatenation, and the repeated terms form the even–odd channel pairs used by RoPE. The resulting descriptor provides the RoPE angles used in attention. For an attention head with query/key channels split into a rotary part and a pass-through part, $\mathbf{q} = [\mathbf{q}_r; \mathbf{q}_p]$ and $\mathbf{k} = [\mathbf{k}_r; \mathbf{k}_p]$, ARoPE applies

$$\tilde{\mathbf{q}} = [\mathbf{q}_r \odot \cos \mathbf{a}_q + \mathrm{rot}(\mathbf{q}_r) \odot \sin \mathbf{a}_q; \mathbf{q}_p], \quad \tilde{\mathbf{k}} = [\mathbf{k}_r \odot \cos \mathbf{a}_k + \mathrm{rot}(\mathbf{k}_r) \odot \sin \mathbf{a}_k; \mathbf{k}_p],$$

where $\mathbf{a}_q$ and $\mathbf{a}_k$ are the ARoPE descriptors for the query and key tokens, and $\mathrm{rot}(\cdot)$ swaps each even–odd channel pair with a sign flip as in standard RoPE [34]. Mean-pooling these per-anchor rotary features—rather than concatenating raw anchor coordinates as in a naive multi-point variant—matches the symmetry that anchor identities are arbitrary, while the encoding still depends on world-frame positions and therefore captures object centroid and shape extent. ARoPE is invariant to anchor reindexing: for any anchor permutation π, $\mathrm{ARoPE}(\{\mathbf{x}^{(i)}_{\pi(k)}\}_{k=1}^{N_a}) = \frac{1}{N_a}\sum_{k=1}^{N_a} \psi_\omega(\mathbf{x}^{(i)}_{\pi(k)}) = \mathrm{ARoPE}(\{\mathbf{x}^{(i)}_{k}\}_{k=1}^{N_a})$, because the sum is unchanged by reordering. Applying ARoPE to object tokens yields improved performance and generalization across varying object counts and geometries (Sec. 4.2). Detailed proofs and discussion are given in Appendix B.

3.4 Training Objectives

Our objective combines position and acceleration Smooth L1 losses [12]: $\mathcal{L} = \lambda_{\text{pos}}(\mathcal{L}^{\text{raw}}_{\text{pos}} + \mathcal{L}^{\text{rigid}}_{\text{pos}}) + \lambda_{\text{acc}}(\mathcal{L}^{\text{raw}}_{\text{acc}} + \mathcal{L}^{\text{rigid}}_{\text{acc}})$, where "raw" and "rigid" denote losses computed before and after Kabsch alignment, respectively, with $\lambda_{\text{pos}}{=}10$ and $\lambda_{\text{acc}}{=}1$. All four terms are supervised at the selected anchors; full-resolution vertices are deterministic after the rigid projection. We train RigidFormer on NVIDIA A100 GPUs for 300 epochs with AdamW [22] ($\beta_1{=}0.9$, $\beta_2{=}0.999$, weight decay 0.01) at a base learning rate of 10^{-4}. The schedule combines a 10-epoch linear warmup (from 10% of the base rate) with cosine decay to $\eta_{\min}{=}10^{-6}$, and we clip the gradient norm at 1.0 for stability. Additional implementation details are provided in Appendix C and G.

4 Experiments

We conduct experiments to evaluate RigidFormer: (i) *Accuracy* vs. state-of-the-art learning-based simulators; (ii) *Generalization* across datasets; (iii) *Resolution generalization* to unseen test-time point counts; (iv) *Anchor robustness* to varying anchor counts and anchor perturbations (random sampling); (v) *Ablations* of anchor-based 3D RoPE, gated attention, and differentiable rigid projection; (vi) *Scalability and controllability* for large scenes and articulated bodies; (vii) *Runtime performance*; and (viii) *Dynamics modeling for partial point clouds*. Please refer to the *supplementary video* for more qualitative results.

Datasets & Metrics. We use the MOVi (Multi-Object Video) datasets [14]: *MOVi-A* (basic geometric shapes), *MOVi-B* (complex geometric shapes), and *MOVi-Sphere* (spheres). Following [1, 33, 40], we report Translation RMSE (m) for per-object center-of-mass position and Orientation RMSE

Table 2: **Performance on MOVi-A, MOVi-B, and MOVi-Sphere.** Each cell reports position RMSE (m) / orientation RMSE (°) at prediction horizons of 50, 75, and 100 frames. Per-column top-two ranks are denoted by shading: 1st , 2nd .

Model	MOVi-A			MOVi-B			MOVi-Sphere		
	50	75	100	50	75	100	50	75	100
MGN+	0.705 / 31.21	N/A	N/A	0.538 / 26.91	N/A	N/A	N/A	N/A	N/A
MGN-LargeRadius+	0.119 / 15.07	N/A	N/A	0.460 / 26.34	N/A	N/A	N/A	N/A	N/A
FIGNet	0.115 / 14.84	N/A	N/A	0.127 / 13.99	N/A	N/A	N/A	N/A	N/A
FIGNet$_{reimpl}$	0.132 / 7.10	0.285 / 14.62	0.492 / 23.30	0.141 / 7.39	0.300 / 15.16	0.516 / 24.96	N/A	N/A	N/A
HCMT$_{reimpl}$ *	0.239 / 5.70	0.538 / 11.82	0.951 / 18.40	0.237 / 4.72	0.527 / 9.80	0.932 / 17.43	0.243 / 4.19	0.541 / 8.21	0.956 / 13.81
VPD$^{\dagger}_{reimpl}$	0.235 / 5.10	0.489 / 11.66	0.827 / 20.37	0.275 / 4.65	0.581 / 9.70	0.987 / 16.99	0.244 / 4.47	0.510 / 9.50	0.855 / 17.65
HopNet	0.054 / 5.64	0.115 / 11.84	0.196 / 18.83	0.047 / 4.91	0.101 / 10.35	0.176 / 17.91	0.034 / 4.05	0.073 / 8.21	0.124 / 13.68
RigidFormer †,*	0.049 / 5.06	0.103 / 10.90	0.177 / 18.32	0.050 / 3.97	0.095 / 8.51	0.161 / 15.33	0.026 / 3.00	0.057 / 6.48	0.099 / 11.19

$*$ Transformer-based model; † Point inputs.

(deg) via quaternion geodesic distance. We evaluate at physical frames during autoregressive rollout, following the HopNet protocol [40]. For our variable step sizes (1/5/10), predictions are mapped to the corresponding physical frames for fair comparison. All metrics are averaged over the test set. We use MOVi-B for ablations due to its diversity and vertex-level variation (see statistics in Appendix D).

4.1 Main Comparison

Tab. 2 reports the matched step-size-1 setting on the MOVi benchmarks. We compare RigidFormer (point input, †) against mesh-based simulators (e.g., HopNet, FIGNet, MGN), point-based methods (VPD), and transformer-based approaches (HCMT). Despite using no mesh connectivity, RigidFormer obtains the best orientation error in all reported columns and the best or second-best translation error in most columns. The most relevant comparison is HopNet, the strongest prior baseline: on MOVi-B at 100 frames, RigidFormer improves over HopNet from $0.176\,\mathrm{m}/17.91°$ to $0.161\,\mathrm{m}/15.33°$, while using only point inputs. The advantage is larger over VPD and HCMT, which obtain $0.987\,\mathrm{m}/16.99°$ and $0.932\,\mathrm{m}/17.43°$ at the same horizon, indicating that RigidFormer substantially reduces long-horizon translation drift. For fairness, we match the input point count in our model to the vertex count used by mesh-based methods, even for sparse simple shapes (e.g., 51 points for a cube and 64 for a cone). Compared with SDF-Sim [33], which reports $0.160\,\mathrm{m}/18.03°$, RigidFormer obtains $0.050\,\mathrm{m}/3.97°$ at step size 1 and $0.029\,\mathrm{m}/1.51°$ at step size 10, without SDF fitting. Qualitative results are shown in Fig. 3, and step-size-conditioned rollouts are analyzed in Tab. 4.

Cross-dataset Generalization. Tab. 3 reports cross-dataset transfer by training on one MOVi variant and testing on another. In the matched step-size-1 block, RigidFormer consistently improves over FIGNet and remains competitive with HopNet, with particularly strong results when training on MOVi-Sphere or MOVi-A. The larger-step block shows that step-size conditioning further reduces our long-horizon errors in every transfer split. For MOVi-B-trained transfer, the gains are more pronounced in orientation than in translation, suggesting that rotation reasoning transfers robustly while translation remains more sensitive to geometric shift; we analyze this trend in Appendix D.1.

Table 3: **Generalization performance across MOVi-Sphere (S), MOVi-A (A), and MOVi-B (B).** The left block compares FIGNet, HopNet, and RigidFormer under the matched step-size-1 setting, with top-two rank shading: 1st , 2nd . The right block reports RigidFormer with larger step sizes.

Train	Test	FIGNet		HopNet		RigidFormer step=1	
		75	100	75	100	75	100
S	A	0.370 / 14.08	0.626 / 22.46	0.112 / 11.29	0.197 / 17.74	0.107 / 11.27	0.183 / 17.92
	B	0.662 / 18.08	1.149 / 29.66	0.106 / 9.75	0.188 / 17.13	0.096 / 9.42	0.161 / 16.81
A	S	0.239 / 11.55	0.417 / 19.66	0.100 / 9.16	0.172 / 15.17	0.087 / 7.94	0.153 / 14.27
	B	0.512 / 13.49	0.864 / 22.41	0.117 / 10.77	0.202 / 18.66	0.123 / 9.29	0.208 / 17.08
B	S	0.536 / 12.74	0.905 / 20.42	0.095 / 9.09	0.160 / 15.04	0.123 / 10.52	0.207 / 17.88
	A	0.520 / 14.53	0.871 / 22.89	0.120 / 12.08	0.202 / 19.15	0.140 / 12.25	0.234 / 20.28

Train	Test	RigidFormer step=5		RigidFormer step=10	
		75	100	75	100
S	A	0.083 / 8.31	0.153 / 14.82	0.070 / 6.84	0.120 / 11.64
	B	0.079 / 7.73	0.134 / 14.50	0.064 / 5.58	0.103 / 10.57
A	S	0.067 / 6.22	0.122 / 11.75	0.062 / 4.70	0.104 / 8.55
	B	0.099 / 7.75	0.173 / 14.85	0.080 / 5.70	0.131 / 10.93
B	S	0.097 / 7.69	0.166 / 13.99	0.084 / 5.83	0.137 / 10.26
	A	0.109 / 9.55	0.190 / 16.85	0.092 / 7.82	0.153 / 13.25

Point Resolution Generalization. Tab. 5 reports point-cloud results on MOVi-B. RigidFormer is trained with randomly sampled point counts $\{128, 256, 512, 1024\}$ and evaluated at 768 points. Despite the unseen test-time resolution, the model remains stable across 25/50/75/100-step rollouts; at 100 steps, step sizes 10, 5, and 1 obtain $0.137/\ 11.13°$, $0.161/\ 14.83°$, and $0.189/\ 16.22°$, respectively.

Step Sizes. We investigate the effect of step-size conditioning in Tab. 4. Larger step sizes consistently yield better long-horizon performance because the model performs fewer autoregressive updates over

Table 4: **Step-size-conditioned rollout performance.** Each cell reports position RMSE (m) / orientation RMSE (°).

Step Size	MOVi-A			MOVi-B			MOVi-Sphere		
	50	75	100	50	75	100	50	75	100
10	0.018 / 1.47	0.068 / 6.98	0.118 / 11.93	0.029 / 1.51	0.069 / 5.89	0.115 / 10.85	0.014 / 1.09	0.045 / 4.30	0.076 / 7.47
5	0.035 / 3.33	0.083 / 8.55	0.148 / 15.08	0.040 / 3.06	0.078 / 7.25	0.136 / 13.55	0.021 / 2.11	0.047 / 5.23	0.086 / 9.58
1	0.049 / 5.06	0.103 / 10.90	0.177 / 18.32	0.050 / 3.97	0.095 / 8.51	0.161 / 15.33	0.026 / 3.00	0.057 / 6.48	0.099 / 11.19

Table 5: **Generalization to different point resolutions.** Trained with point counts 128, 256, 512, 1024 and evaluated at 768 points. Each cell reports position RMSE (m) / orientation RMSE (°).

Step size	Rollout horizon (steps)		
	50	75	100
10	0.071 / 1.64	0.101 / 6.11	0.137 / 11.13
5	0.082 / 3.55	0.113 / 8.06	0.161 / 14.83
1	0.094 / 4.38	0.132 / 9.21	0.189 / 16.22

Table 6: **Effect of Gated Attention and Differentiable Rigid Alignment.** Top-two values are shaded by rank among variants.

Gated Attention	Differentiable Alignment	Metric	step_size=1			step_size=5			step_size=10		
			50	75	100	50	75	100	50	75	100
✗	✓	Pos	0.077	0.153	0.259	0.050	0.107	0.191	0.033	0.090	0.152
		Ori	4.70	9.98	17.12	3.51	7.77	13.93	1.57	6.02	10.78
✓	✗	Pos	0.052	0.098	0.169	0.042	0.082	0.146	0.030	0.072	0.121
		Ori	4.08	8.79	15.79	3.14	7.39	13.75	1.55	5.98	11.04
✓	✓	Pos	0.050	0.095	0.161	0.040	0.078	0.136	0.029	0.069	0.115
		Ori	3.97	8.51	15.33	3.06	7.25	13.55	1.51	5.89	10.85

the same physical horizon. Step size $= 10$ gives the lowest 100-frame errors on MOVi-A, MOVi-B, and MOVi-Sphere. The step size $= 1$ rows remain the matched setting for comparison with prior one-step rollout protocols, while step size $= 5$ provides an intermediate trade-off between rollout frequency and local prediction difficulty. More discussion can be found in Appendix J.

Dynamics Modeling for Partial Point Clouds. We further evaluate RigidFormer in a partial-observation setting where only a fraction of each object's points is visible at test time. Concretely, we randomly mask 25% of the points within each object's bounding box and roll out from the resulting partial inputs; the same model trained on full point clouds is used without re-training. As shown in Fig. 4, RigidFormer produces stable rollouts under occluded inputs, retaining accurate inter-object contacts and low long-horizon drift. This indicates that the object-level Transformer with anchor-based dynamics generalizes to partial inputs without specialized completion or recovery modules. Additional qualitative results are provided in Fig. A6.

Figure 4: **Partial point-cloud rollouts.** Three example sequences (rows) where 25% of each object's points are masked at test time. RigidFormer produces stable rollouts with accurate inter-object contacts. Meshes are shown only for visualization; our model operates on point inputs.

4.2 Ablation Studies

Positional Embedding. We compare our Anchor-based Rotary Positional Embedding (ARoPE) with standard sinusoidal PE [37], learned absolute PE [10], and geometry-aware OBB, PCA, and SE(3) baselines. ARoPE uses the set-aggregated anchor descriptor from Sec. 3.3 to rotate query and key vectors as a function of sparse 3D anchor coordinates, while mean aggregation makes the descriptor invariant to anchor reindexing. As reported in Tab. 7, ARoPE achieves the best or tied-best position

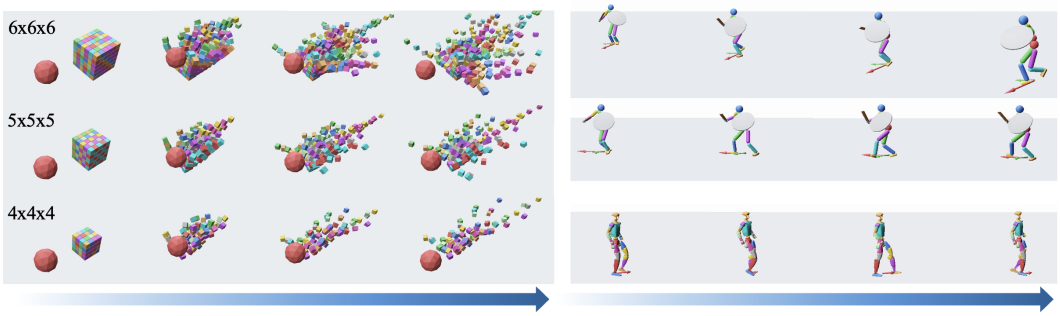


Figure 5: **Scalability and controllability. Left:** Simulation with $216, 125, 64$ cubes; RigidFormer stays stable as object count grows. **Right:** Preliminary direction-controlled articulated dynamics. Humanoid: commanded facing and moving directions are in green and red arrows, respectively; Unitree G1: the arrow shows the moving direction. RigidFormer follows heading commands while preserving coherent part-level motion.

error in eight of nine cells and the best orientation error in most cells; SE(3) and sinusoidal PE are close in selected metrics. Overall, the results suggest that direct anchor-coordinate RoPE provides a stable geometry signal for heterogeneous MOVi-B shapes.

Gated Attention. We ablate the sigmoid gating in RigidFormer. Tab. 6 shows that gating mainly improves translation accuracy and long-horizon stability. At 100 steps, it reduces position error from $0.259 \rightarrow 0.161$ (step_size$=1$), $0.191 \rightarrow 0.136$ (step_size$=5$), and $0.152 \rightarrow 0.115$ (step_size$=10$). Orientation error also improves in most cells; at step size 10 and 100 steps, it remains essentially tied with the ungated variant.

Table 7: **Comparison of Different Positional Embeddings.** Top-two values are shaded by rank among PE variants.

PE	Metric	step_size=1			step_size=5			step_size=10		
		50	75	100	50	75	100	50	75	100
Sinusoidal	Pos	0.051	0.100	0.172	0.042	0.083	0.145	0.030	0.074	0.122
	Ori	4.10	8.68	15.30	3.16	7.39	13.52	1.55	6.04	10.91
Learned	Pos	0.054	0.103	0.176	0.042	0.082	0.146	0.030	0.073	0.122
	Ori	4.24	9.01	15.94	3.18	7.43	13.77	1.56	5.98	11.04
OBB	Pos	0.060	0.114	0.189	0.045	0.087	0.155	0.033	0.079	0.133
	Ori	4.38	9.44	16.83	3.31	7.69	14.35	1.57	5.96	11.08
PCA	Pos	0.380	0.867	1.548	0.240	0.647	1.249	0.107	0.499	0.918
	Ori	7.02	15.11	25.79	5.23	12.52	22.62	2.27	9.32	16.74
SE(3)	Pos	0.052	0.099	0.167	0.041	0.080	0.139	0.029	0.069	0.114
	Ori	4.25	9.21	16.36	3.15	7.45	13.89	1.52	5.97	11.04
ARoPE (Ours)	Pos	0.050	0.095	0.161	0.040	0.078	0.136	0.029	0.069	0.115
	Ori	3.97	8.51	15.33	3.06	7.25	13.55	1.51	5.89	10.85

Differentiable Kabsch Alignment. Tab. 6 also compares RigidFormer **w/** vs. **w/o** differentiable rigid projection. With gating enabled, the differentiable variant consistently reduces 100-step position error ($0.169 \rightarrow 0.161$, $0.146 \rightarrow 0.136$, and $0.121 \rightarrow 0.115$ for step sizes 1, 5, and 10). Orientation error also decreases at the same horizons, indicating that gradient flow through the rigid projection reduces rollout drift.

Different Numbers of Anchors. Tab. 8 studies the effect of the anchor count k on MOVi-B. Increasing from $k = 3$ to $k = 4$ improves translation accuracy across step sizes and rollout horizons. Using $k = 8$ remains competitive and yields lower orientation error in several long-horizon settings, but it uses twice as many anchor queries and offers a less favorable translation–cost trade-off. We therefore use 4 anchors as the default efficiency–quality trade-off.

Randomized FPS Anchors. We train and evaluate RigidFormer to assess whether the model learns *intrinsic geometric features* of the object, rather than relying on specific anchor identities. Tab. 9 reports a perturbation study where FPS anchors are randomly re-sampled during both training and evaluation; it is separate from the default fixed-N_a=4 evaluation used elsewhere. Randomized FPS with 4 anchors gives the strongest translation accuracy, while 8 anchors can reduce rotation error in several settings. These results indicate that RigidFormer is robust to anchor selection and that the default N_a=4 provides a good quality–efficiency trade-off.

4.3 Scalability, Controllability and Efficiency

Large-Scale Simulation. We demonstrate the scalability of RigidFormer on the WreckingBall dataset, which contains scenes with 64, 125, and 216 cubes (visualized in Fig. 5, left). At 50 physical steps, RigidFormer achieves position RMSEs of 1.210, 0.690, and 0.130, with orientation errors of

Table 8: **Different Numbers of Anchors.** Results for different numbers of anchors in RigidFormer. Each cell reports position RMSE (m) / orientation RMSE (°).

Anchors	Step size	Rollout horizon (steps)		
		50	75	100
3	10	0.033 / 1.63	0.081 / 6.74	0.135 / 12.71
	5	0.047 / 3.63	0.094 / 8.67	0.166 / 16.80
	1	0.060 / 4.82	0.118 / 10.64	0.198 / 19.01
4	10	0.029 / 1.51	0.069 / 5.89	0.115 / 10.85
	5	0.040 / 3.06	0.078 / 7.25	0.136 / 13.55
	1	0.050 / 3.97	0.095 / 8.51	0.161 / 15.33
8	10	0.031 / 1.51	0.079 / 5.53	0.130 / 10.07
	5	0.046 / 3.14	0.097 / 7.20	0.170 / 13.38
	1	0.061 / 4.08	0.123 / 8.67	0.208 / 15.26

Table 9: **Randomized FPS anchors.** We train and evaluate with FPS anchors randomly re-sampled as a perturbation study. Each cell reports position RMSE (m) / orientation RMSE (°).

Anchors	Step size	Rollout horizon (steps)		
		50	75	100
3	10	0.031 / 1.72	0.075 / 6.36	0.126 / 11.74
	5	0.042 / 3.72	0.085 / 8.40	0.149 / 15.12
	1	0.055 / 4.80	0.110 / 10.46	0.187 / 18.19
4	10	0.029 / 1.59	0.070 / 5.76	0.115 / 10.39
	5	0.041 / 3.21	0.081 / 7.23	0.139 / 13.38
	1	0.052 / 4.41	0.099 / 9.31	0.163 / 16.16
8	10	0.030 / 1.52	0.083 / 5.58	0.141 / 10.06
	5	0.044 / 3.23	0.095 / 7.28	0.171 / 13.29
	1	0.056 / 4.04	0.114 / 8.64	0.193 / 15.17

$20.50°$, $16.50°$, and $4.60°$ for the three scenes, respectively. The model remains stable across scales, validating the effectiveness of our object-level interaction and anchor-based state advance. In this setting, RigidFormer runs at 20 FPS.

Controllable articulated-body simulation. We further evaluate RigidFormer as a preliminary extension to articulated characters under directional control. For the ASE humanoid with 15 body parts and the Unitree G1 robot with 31 body parts, we treat each body part as an object-level component and condition the model on desired heading commands through FiLM. As shown in Fig. 5, RigidFormer produces coherent whole-body motion that follows different heading commands, suggesting that the object-centric formulation can extend beyond independent rigid objects. At 100 physical steps, RigidFormer obtains position/orientation errors of 0.062 / $14.47°$ on ASE Humanoid and 0.072 / $16.26°$ on G1. We note that this result is presented as an extension study rather than the primary evidence for the method.

Runtime Performance. Tab. 10 reports runtime on MOVi-B using a 50-step autoregressive rollout averaged over 10 iterations on an NVIDIA GeForce RTX 5080. RigidFormer achieves $8\times$ and $101\times$ speedups over FIGNet and HopNet, respectively. This efficiency mainly comes from the object-centric Transformer design, which reasons over compact object-level states rather than dense vertex-level structures. Further runtime analysis is provided in Appendix F.1.

Table 10: Runtime comparison.

Method	ms/step	FPS
HopNet	4228.7	0.2
FIGNet	336.0	3.0
RigidFormer (Ours)	**41.9**	**23.9**

5 Conclusion

We presented RigidFormer, an object-centric, mesh-free Transformer-based simulator for multi-object rigid-body contact dynamics from point clouds. RigidFormer shifts interaction reasoning to the *object level* and updates dynamics using a compact set of anchors, reducing vertex-level computation while maintaining accuracy. To inject 3D geometry into attention, we introduce Anchor-based RoPE, whose mean-pooled anchor descriptor is invariant to anchor reindexing and whose use for object tokens preserves permutation equivariance over unordered objects. RigidFormer further enforces rigidity by projecting updates onto the rigid-body manifold via differentiable Kabsch alignment, and it supports controllable integration through step-size conditioning within a single model. Experiments validate the efficiency, effectiveness, and versatility of RigidFormer. *Limitations and future work.* The current formulation primarily targets object-level rigid-body contact dynamics and relies on object labels to identify which points belong to each object. Although we report partial point-cloud results, prediction becomes challenging when partial observations capture too little of an object's shape. Future work could move toward noisier perception settings, mixed rigid–deformable scenes, and more fine-grained adaptive time stepping. We provide additional discussion in Appendix K.

Acknowledgments and Disclosure of Funding

This work was conducted within the framework of the collaboration between MIT and Meta.

References

[1] Kelsey R Allen, Yulia Rubanova, Tatiana Lopez-Guevara, William Whitney, Alvaro Sanchez-Gonzalez, Peter Battaglia, and Tobias Pfaff. Learning rigid dynamics with face interaction graph networks. *arXiv preprint arXiv:2212.03574*, 2022.

[2] Genesis Authors. Genesis: A universal and generative physics engine for robotics and beyond. *URL https://github. com/Genesis-Embodied-AI/Genesis*, 2024.

[3] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29, 2016.

[4] Romain Brégier. Deep regression on manifolds: a 3d rotation case study. In *2021 International Conference on 3D Vision (3DV)*, pages 166–174. IEEE, 2021.

[5] Arunkumar Byravan and Dieter Fox. SE3-Nets: Learning rigid body motion using deep neural networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 173–180. IEEE, 2017. doi: 10.1109/ICRA.2017.7989023.

[6] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016.

[7] Hsiao-yu Chen, Edith Tretschk, Tuur Stuyck, Petr Kadlecek, Ladislav Kavan, Etienne Vouga, and Christoph Lassner. Virtual elastic objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15827–15837, 2022.

[8] Ricky TQ Chen, Brandon Amos, and Maximilian Nickel. Learning neural event functions for ordinary differential equations. *arXiv preprint arXiv:2011.03902*, 2020.

[9] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[11] C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

[12] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[13] Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical computer science*, 38:293–306, 1985.

[14] Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J. Fleet, Dan Gnanapragasam, Florian Golemo, Charles Herrmann, Thomas Kipf, Abhijit Kundu, Dmitry Lagun, Issam Laradji, Hsueh-Ti (Derek) Liu, Henning Meyer, Yishu Miao, Derek Nowrouzezahrai, Cengiz Oztireli, Etienne Pot, Noha Radwan, Daniel Rebain, Sara Sabour, Mehdi S. M. Sajjadi, Matan Sela, Vincent Sitzmann, Austin Stone, Deqing Sun, Suhani Vora, Ziyu Wang, Tianhao Wu, Kwang Moo Yi, Fangcheng Zhong, and Andrea Tagliasacchi. Kubric: A scalable dataset generator. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3749–3761, June 2022.

[15] Byeongho Heo, Song Park, Dongyoon Han, and Sangdoo Yun. Rotary position embedding for vision transformer. In *European Conference on Computer Vision*, pages 289–305. Springer, 2024.

[16] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019.

[17] Wenlong Huang, Yu-Wei Chao, Arsalan Mousavian, Ming-Yu Liu, Dieter Fox, Kaichun Mo, and Li Fei-Fei. Pointworld: Scaling 3d world models for in-the-wild robotic manipulation. *arXiv preprint arXiv:2601.03782*, 2026.

[18] Wolfgang Kabsch. A solution for the best rotation to relate two sets of vectors. *Foundations of Crystallography*, 32(5):922–923, 1976.

[19] Chanho Kim and Li Fuxin. Object dynamics modeling with hierarchical point cloud-based representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20977–20986, 2024.

[20] Kimi Team, Guangyu Chen, Yu Zhang, Jianlin Su, Weixin Xu, Siyuan Pan, Yaoyu Wang, Yucheng Wang, Guanduo Chen, Bohong Yin, Yutian Chen, Junjie Yan, Ming Wei, Y. Zhang, Fanqing Meng, Chao Hong, Xiaotong Xie, Shaowei Liu, Enzhe Lu, Yunpeng Tai, Yanru Chen, Xin Men, Haiqing Guo, Y. Charles, Haoyu Lu, Lin Sui, Jinguo Zhu, Zaida Zhou, Weiran He, Weixiao Huang, Xinran Xu, Yuzhi Wang, Guokun Lai, Yulun Du, Yuxin Wu, Zhilin Yang, and Xinyu Zhou. Attention residuals, 2026. URL `https://arxiv.org/abs/2603.15031`. arXiv:2603.15031.

[21] Jiahui Lei, Yijia Weng, Adam W Harley, Leonidas Guibas, and Kostas Daniilidis. Mosca: Dynamic gaussian fusion from casual videos via 4d motion scaffolds. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 6165–6177, 2025.

[22] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[23] Miles Macklin. Warp: A high-performance python framework for gpu simulation and graphics. In *NVIDIA GPU Technology Conference (GTC)*, volume 3, 2022.

[24] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[25] Xue Bin Peng. Mimickit: A reinforcement learning framework for motion imitation and control. *arXiv preprint arXiv:2510.13794*, 2025.

[26] Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. Amp: Adversarial motion priors for stylized physics-based character control. *ACM Transactions on Graphics (ToG)*, 40(4):1–20, 2021.

[27] Xue Bin Peng, Yunrong Guo, Lina Halper, Sergey Levine, and Sanja Fidler. Ase: Large-scale reusable adversarial skill embeddings for physically simulated characters. *ACM Transactions On Graphics (TOG)*, 41(4):1–17, 2022.

[28] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[29] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Learning mesh-based simulation with graph networks. In *International conference on learning representations*, 2020.

[30] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[31] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

[32] Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, et al. Gated attention for large language models: Non-linearity, sparsity, and attention-sink-free. *arXiv preprint arXiv:2505.06708*, 2025.

[33] Yulia Rubanova, Tatiana Lopez-Guevara, Kelsey R Allen, William F Whitney, Kimberly Stachenfeld, and Tobias Pfaff. Learning rigid-body simulators over implicit shapes for large-scale scenes and vision. *Advances in Neural Information Processing Systems*, 37:125809–125838, 2024.

[34] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[35] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[36] Arash Vahdat, Francis Williams, Zan Gojcic, Or Litany, Sanja Fidler, Karsten Kreis, et al. Lion: Latent point diffusion models for 3d shape generation. *Advances in Neural Information Processing Systems*, 35:10021–10039, 2022.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[38] Chen Wang, Roberto Martín-Martín, Danfei Xu, Jun Lv, Cewu Lu, Li Fei-Fei, Silvio Savarese, and Yuke Zhu. 6-PACK: Category-level 6D pose tracker with anchor-based keypoints. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10059–10066. IEEE, 2020. doi: 10.1109/ICRA40945.2020.9196679.

[39] Qianqian Wang, Yen-Yu Chang, Ruojin Cai, Zhengqi Li, Bharath Hariharan, Aleksander Holynski, and Noah Snavely. Tracking everything everywhere all at once. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 19795–19806, 2023.

[40] Amaury Wei and Olga Fink. Integrating physics and topology in neural networks for learning rigid body dynamics. *Nature Communications*, 16(1):6867, 2025.

[41] William F Whitney, Tatiana Lopez-Guevara, Tobias Pfaff, Yulia Rubanova, Thomas Kipf, Kimberly Stachenfeld, and Kelsey R Allen. Learning 3d particle-based simulators from rgb-d videos. *arXiv preprint arXiv:2312.05359*, 2023.

[42] William F Whitney, Jacob Varley, Deepali Jain, Krzysztof Choromanski, Sumeet Singh, and Vikas Sindhwani. Modeling the real world with high-density visual particle dynamics. *arXiv preprint arXiv:2406.19800*, 2024.

[43] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4541–4550, 2019.

[44] Youn-Yeol Yu, Jeongwhan Choi, Woojin Cho, Kookjin Lee, Nayong Kim, Kiseok Chang, Chang-Seung Woo, Ilho Kim, Seok-Woo Lee, Joon-Young Yang, et al. Learning flexible body collision dynamics with hierarchical contact mesh transformer. *arXiv preprint arXiv:2312.12467*, 2023.

[45] Jingyang Yuan, Gongbo Sun, Zhiping Xiao, Hang Zhou, Xiao Luo, Junyu Luo, Yusheng Zhao, Wei Ju, and Ming Zhang. Egode: An event-attended graph ode framework for modeling rigid dynamics. *Advances in Neural Information Processing Systems*, 37:59093–59118, 2024.

[46] Chong Zeng, Yue Dong, Pieter Peers, Hongzhi Wu, and Xin Tong. Renderformer: Transformer-based neural rendering of triangle meshes with global illumination. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers*, pages 1–11, 2025.

[47] Junyi Zhang, Charles Herrmann, Junhwa Hur, Varun Jampani, Trevor Darrell, Forrester Cole, Deqing Sun, and Ming-Hsuan Yang. Monst3r: A simple approach for estimating geometry in the presence of motion. *arXiv preprint arXiv:2410.03825*, 2024.

[48] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 16259–16268, 2021.

[49] Haoyu Zhen, Qiao Sun, Hongxin Zhang, Junyan Li, Siyuan Zhou, Yilun Du, and Chuang Gan. Tesseract: learning 4d embodied world models. *arXiv preprint arXiv:2504.20995*, 2025.

[50] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Extending lagrangian and hamiltonian neural networks with differentiable contact models. *Advances in Neural Information Processing Systems*, 34:21910–21922, 2021.

[51] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5745–5753, 2019.

Appendix Contents

A More Qualitative Results

Performance on partial point cloud inputs is shown in Fig. A6. We randomly mask 25% of the points inside each object's bounding box and evaluate the model from the resulting partial observations. Additional videos are included in the supplementary website.

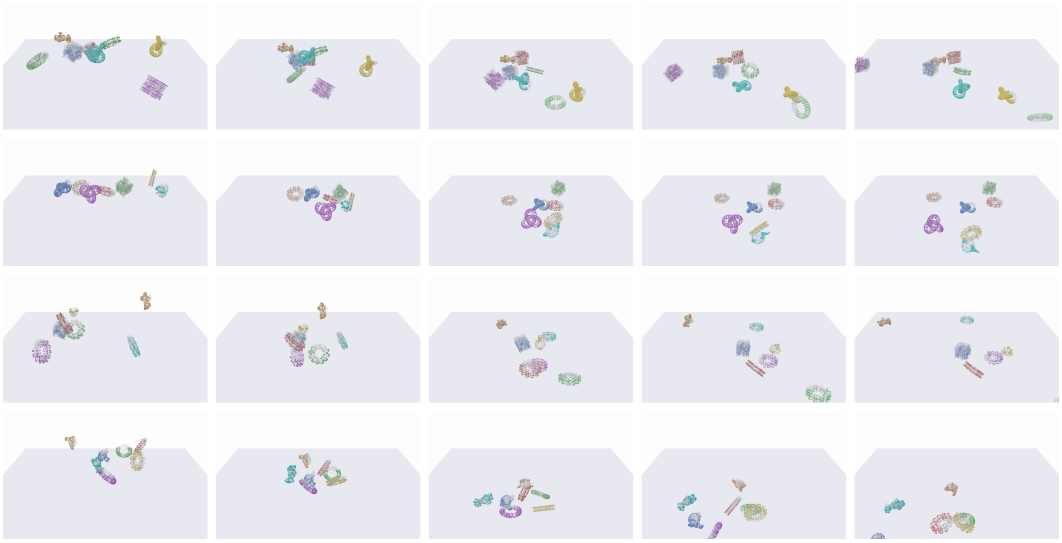


Figure A6: **Partial point-cloud qualitative results.** RigidFormer predicts accurate rigid-body interactions from partial observations with low drift and stable contacts. Meshes are shown only for visualization; our model operates on point inputs. Please refer to our video for more results.

We provide additional qualitative visualizations in Figs. A7–A9, covering held-out MOVi-A, MOVi-B, and MOVi-Sphere test scenes. Each row corresponds to one selected sample and shows four uniformly sampled frames from the rollout. These examples complement Fig. 3 and show that RigidFormer produces stable multi-object rollouts across different object geometries and contact configurations.

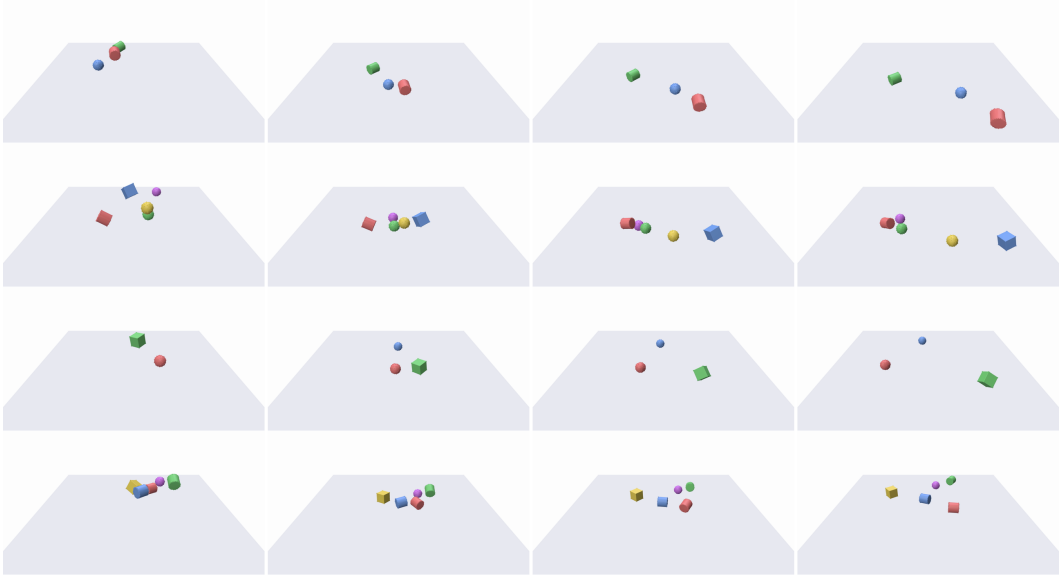


Figure A7: **More qualitative results on MOVi-A.** Each row is one held-out test sample; columns show four frames from the rollout.

Figure A8: **More qualitative results on MOVi-B.** Each row is one held-out test sample; columns show four frames from the rollout.

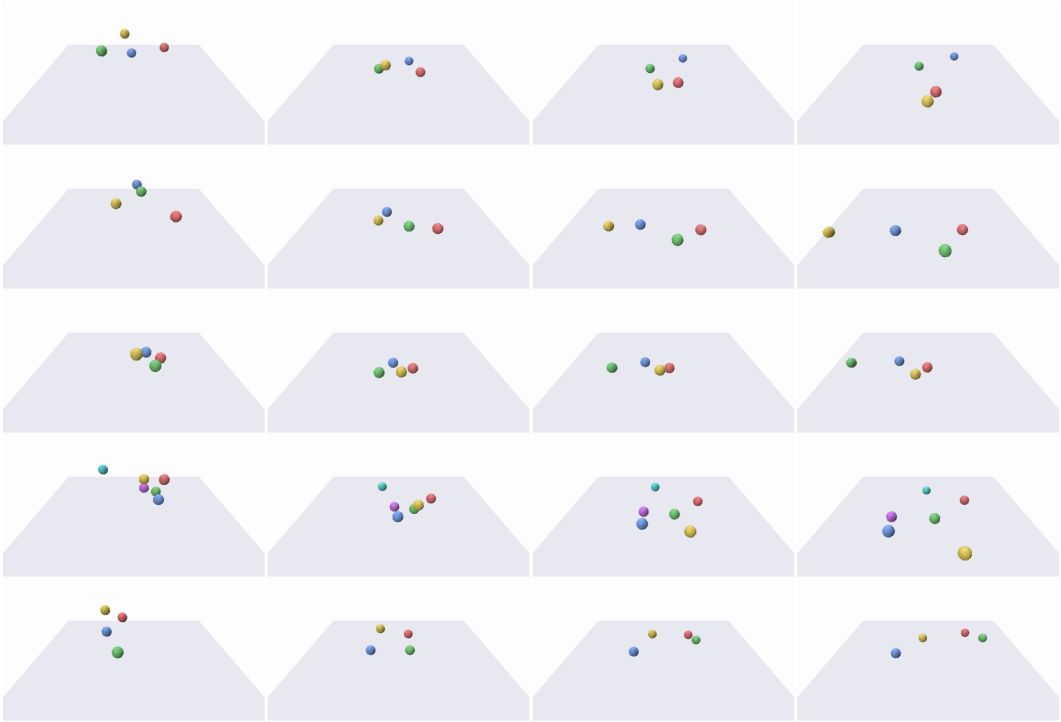


Figure A9: **More qualitative results on MOVi-Sphere.** Each row is one held-out test sample; columns show four frames from the rollout.

B Theoretical Properties of the Object-Anchor Representation

We state the exact symmetry and structural guarantees used in Sec. 3.3. These properties concern indexing choices and rigid projection in the representation; they do not claim full physical equivariance of the simulator under arbitrary rigid transformations of the world.

Anchor reindexing invariance. Let $\psi_\omega(\mathbf{x})$ denote the shared 3D rotary anchor map used by ARoPE, and define the object descriptor

$$\rho(A_i) = \frac{1}{N_a}\sum_{k=1}^{N_a} \psi_\omega(\mathbf{x}_k^{(i)}), \qquad A_i = \{\mathbf{x}_k^{(i)}\}_{k=1}^{N_a}. \tag{1}$$

For any permutation π of the anchors,

$$\rho(\{\mathbf{x}_{\pi(k)}^{(i)}\}_{k=1}^{N_a}) = \frac{1}{N_a}\sum_{k=1}^{N_a} \psi_\omega(\mathbf{x}_{\pi(k)}^{(i)}) = \frac{1}{N_a}\sum_{k=1}^{N_a} \psi_\omega(\mathbf{x}_k^{(i)}) = \rho(A_i). \tag{2}$$

Thus the ARoPE descriptor is invariant to anchor ordering. The same statement holds for any shared per-anchor map followed by a symmetric pooling operator; we use mean pooling for simplicity and scale stability.

Vertex-order invariance of AVP. For a fixed anchor $\mathbf{q}^{(i,k)}$, AVP computes weights

$$w^{(i,k,v)} = \exp\left(-\frac{\|\mathbf{x}^{(i,v)} - \mathbf{q}^{(i,k)}\|}{\sigma}\right), \qquad \mathbf{u}^{(i,k)} = \frac{\sum_v w^{(i,k,v)}\mathbf{f}^{(i,v)}}{\sum_v w^{(i,k,v)}}. \tag{3}$$

If the vertices and their encoder features are reindexed by any permutation τ, both the numerator and denominator are unchanged after relabeling the summation index. Therefore AVP is invariant to vertex ordering. The weights also depend only on Euclidean distances: for any common rigid transform $\mathbf{x}' = \mathbf{R}\mathbf{x} + \mathbf{t}$ and $\mathbf{q}' = \mathbf{R}\mathbf{q} + \mathbf{t}$ with $\mathbf{R} \in SO(3)$, $\|\mathbf{x}' - \mathbf{q}'\| = \|\mathbf{x} - \mathbf{q}\|$. Thus the AVP attention weights are unchanged by common rigid transforms of an object's current point and anchor coordinates. In the batched implementation, weights on padded vertices are set to zero before normalization, which is equivalent to summing only over valid vertices.

Object permutation equivariance. Consider a permutation matrix P acting on the M object tokens, and let $\tilde{P} = \mathrm{diag}(P, I_{N_r})$ act on the concatenation of object tokens and the N_r=16 register tokens. Self-attention without sequence-index positional embeddings satisfies

$$\mathrm{Attn}(\tilde{P}\mathbf{Q}, \tilde{P}\mathbf{K}, \tilde{P}\mathbf{V}) = \tilde{P}\,\mathrm{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}), \tag{4}$$

because the row-wise softmax and value aggregation commute with simultaneous token permutation. Token-wise RMSNorm, FiLM, elementwise gated attention, and FFNs also commute with $\tilde{P}$ since their parameters are shared across object positions. Therefore each decoder block maps a permutation of object tokens to the same permutation of outputs, while the register tokens remain a shared permutation-invariant workspace. Applying the same argument layer by layer proves equivariance of the object-level decoder.

Anchor prediction and rigid projection. The anchor predictor uses shared weights for each object and applies cross-object attention to the permuted object-token set; hence permuting object order only permutes the predicted anchor updates by object. Within one object, Kabsch alignment depends on sums of corresponding centered source and target anchors. If source and target anchors are reindexed by the same permutation, the cross-covariance matrix is unchanged, so the recovered rigid transform $(\mathbf{R}^{(i)}, \mathbf{t}^{(i)})$ is unchanged. The full simulator is therefore equivariant to object reordering and invariant to anchor reindexing under consistent anchor correspondences.

Rigidity by construction. After Kabsch alignment, every output vertex is generated as

$$\mathbf{x}_{t+1}^{(i,v)} = \mathbf{R}^{(i)}\mathbf{x}_{\mathrm{ref}}^{(i,v)} + \mathbf{t}^{(i)}, \qquad \mathbf{R}^{(i)} \in SO(3). \tag{5}$$

For any two vertices u, v of the same object,

$$\|\mathbf{x}_{t+1}^{(i,u)} - \mathbf{x}_{t+1}^{(i,v)}\|_2 = \|\mathbf{R}^{(i)}(\mathbf{x}_{\text{ref}}^{(i,u)} - \mathbf{x}_{\text{ref}}^{(i,v)})\|_2 = \|\mathbf{x}_{\text{ref}}^{(i,u)} - \mathbf{x}_{\text{ref}}^{(i,v)}\|_2. \tag{6}$$

Thus intra-object distances are preserved exactly by the final projection, independent of the raw anchor accelerations predicted by the network. The simulator is intentionally not SE(3)-equivariant overall: gravity, ground contact, and evaluation coordinates are expressed in the world frame.

C Training Details

C.1 Training Schedule

We do not use curriculum learning or scheduled sampling in the reported MOVi experiments. The final model is trained directly with sequence length $T{=}8$ and random integration step sizes $\Delta t \in \{1, 5, 10\}$ sampled with near-uniform probabilities. The same FiLM-conditioned model is therefore exposed to short and long temporal discretizations throughout training.

We optimize with AdamW using a base learning rate 10^{-4}, weight decay 0.01, $\beta_1{=}0.9$, and $\beta_2{=}0.999$. The learning rate follows a 10-epoch linear warmup from 10% of the base rate, followed by cosine decay to 10^{-6}. Gradients are clipped to norm 1.0. Training runs for 300 epochs with batch size 18 per process in the main MOVi-B configuration. We use HopNet-style $960/120/120$ train/validation/test split files for each MOVi variant. In the main runs, validation loss is disabled. Evaluation uses 2 warmup frames followed by autoregressive rollout.

C.2 Loss Implementation Details

All loss terms in Sec. 3.4 are implemented using the Smooth L1 Loss [12] (PyTorch `nn.SmoothL1Loss()` function[1]), applied element-wise at anchor locations; the implementation sums coordinate losses over x, y, z and averages over valid anchors, objects, and time steps. Concretely, for two tensors $\mathbf{u}$ and $\mathbf{v}$ of the same shape, the Smooth L1 loss is defined as

$$\ell_{\text{SL1}}(\mathbf{u}, \mathbf{v}) = \frac{1}{|\mathbf{u}|} \sum_j \begin{cases} \frac{1}{2}(u_j - v_j)^2, & \text{if } |u_j - v_j| < 1, \\ |u_j - v_j| - \frac{1}{2}, & \text{otherwise,} \end{cases} \tag{7}$$

which corresponds to the default setting of `nn.SmoothL1Loss()` in PyTorch. Compared with ℓ_2 loss, Smooth L1 is less sensitive to occasional large errors during autoregressive rollout, while remaining smooth near zero and thus stable for optimization.

In our training objective, both the position and acceleration losses are computed using Smooth L1 before and after rigid projection. All four terms are evaluated at anchor locations. Specifically, the raw position loss is

$$\mathcal{L}_{\text{pos}}^{\text{raw}} = \ell_{\text{SL1}}\left(\hat{\mathbf{q}}_{t+1}^{\text{raw}}, \mathbf{q}_{t+1}^{\text{gt}}\right), \tag{8}$$

and the rigid position loss is

$$\mathcal{L}_{\text{pos}}^{\text{rigid}} = \ell_{\text{SL1}}\left(\hat{\mathbf{q}}_{t+1}^{\text{rigid}}, \mathbf{q}_{t+1}^{\text{gt}}\right), \tag{9}$$

where $\hat{\mathbf{q}}_{t+1}^{\text{raw}}$ denotes the predicted anchor positions before Kabsch alignment, $\hat{\mathbf{q}}_{t+1}^{\text{rigid}}$ denotes the corresponding anchors after rigid projection, and $\mathbf{q}_{t+1}^{\text{gt}}$ is the ground-truth anchor state. Full-resolution vertices are supervised indirectly through the rigid transform induced by the anchor update.

Similarly, the raw and rigid acceleration losses are defined as

$$\mathcal{L}_{\text{acc}}^{\text{raw}} = \ell_{\text{SL1}}\left(\hat{\mathbf{a}}_t^{\text{raw}}, \mathbf{a}_t^{\text{gt}}\right), \qquad \mathcal{L}_{\text{acc}}^{\text{rigid}} = \ell_{\text{SL1}}\left(\hat{\mathbf{a}}_t^{\text{rigid}}, \mathbf{a}_t^{\text{gt}}\right), \tag{10}$$

where the ground-truth acceleration is computed from the trajectory using the same Verlet discretization,

$$\mathbf{a}_t^{\text{gt}} = \frac{\mathbf{q}_{t+1}^{\text{gt}} - 2\mathbf{q}_t^{\text{gt}} + \mathbf{q}_{t-1}^{\text{gt}}}{\Delta t^2}. \tag{11}$$

[1] `https://docs.pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html`

The final loss is therefore

$$\mathcal{L} = \lambda_{\text{pos}} \left(\mathcal{L}_{\text{pos}}^{\text{raw}} + \mathcal{L}_{\text{pos}}^{\text{rigid}} \right) + \lambda_{\text{acc}} \left(\mathcal{L}_{\text{acc}}^{\text{raw}} + \mathcal{L}_{\text{acc}}^{\text{rigid}} \right), \tag{12}$$

with $\lambda_{\text{pos}} = 10$ and $\lambda_{\text{acc}} = 1$ in the reported experiments. For multi-step training, residuals are normalized by Δt^2 before the Smooth L1 reduction, matching the acceleration scale used by Verlet integration.

D Data Statistics

As detailed in Tab. D11, the datasets vary significantly in object complexity. **MOVi-A** features simple geometric primitives—cubes (51 vertices), cylinders (64 vertices), and spheres (64 vertices)—uniformly distributed across scenes. **MOVi-B** introduces 11 object categories with diverse mesh complexities ranging from 51 vertices (cube) to 1,142 vertices (torus_knot), including complex shapes such as gear (569 vertices), sponge (908 vertices), spot (682 vertices), and suzanne (523 vertices); all categories appear with approximately equal frequency ($\sim$9% each). **MOVi-Sphere** serves as a controlled baseline containing only spheres (64 vertices each). This diversity enables comprehensive evaluation across varying geometric complexities, from simple 51-vertex primitives to intricate 1,142-vertex meshes. Each dataset contains 1,200 scenes simulated for 480 frames.

Table D11: **Object Types and Vertex Counts in MOVi Benchmarks.**

Dataset	Object Name	Vertices	Ratio
MOVi-A	cube	51	33.6%
	cylinder	64	33.8%
	sphere	64	32.6%
MOVi-B	cone	64	8.8%
	cube	51	9.4%
	cylinder	64	8.8%
	gear	569	8.8%
	sphere	64	9.2%
	sponge	908	8.8%
	spot	682	9.7%
	suzanne	523	9.1%
	teapot	274	9.4%
	torus	647	8.9%
	torus_knot	1142	9.2%
MOVi-Sphere	sphere	64	100.0%

Table D12: **Dataset summary statistics** (verified across all raw scenes).

Dataset	Scenes	Total instances	Object types	Frames/scene
MOVi-A	1,200	7,810	3 primitives	480
MOVi-B	1,200	7,712	11 primitives	480
MOVi-Sphere	1,200	7,809	1 primitive	480

D.1 Cross-Dataset Transfer Analysis

Tab. 3 shows that the MOVi-B-trained model transfers slightly less well to the other MOVi variants than models trained on MOVi-A or MOVi-Sphere. Averaging the 100-step position errors over the two held-out target datasets, the MOVi-B-trained model obtains 0.221, 0.178, and 0.145 for step sizes 1, 5, and 10, respectively. The corresponding averages are 0.181, 0.148, and 0.118 for MOVi-A training, and 0.172, 0.144, and 0.112 for MOVi-Sphere training. The gap is more visible in translation than in orientation, consistent with the rigid projection stabilizing rotations while contact-induced translation remains more sensitive to dataset shift. One interpretation is related to geometry complexity. MOVi-A contains three simple primitives, MOVi-Sphere contains only spheres, whereas MOVi-B contains 11 categories with much broader vertex-count and shape variation (Tab. D11). Since the transfer

experiments use the same architecture, evaluation protocol, and physics-parameter inputs, geometry is a natural factor to consider. Simpler source geometries may limit shape-specific cues and encourage the model to rely more on object-level motion and interaction patterns. In contrast, training on MOVi-B may adapt the encoder and Anchor-Vertex Pooling module more strongly to B-specific surface and vertex-density statistics, which do not always transfer cleanly to simpler target geometries. This interpretation is meant only to explain the average trend and a few marginal single-cell cases; it is not evidence that cross-dataset transfer is generally easier than in-domain evaluation.

E Data Augmentation

To improve generalization and prevent overfitting, we employ two complementary data augmentation strategies during training:

Rotation Augmentation. We apply random Z-axis rotations to the entire scene during training. Given the original vertex positions $\mathbf{X} \in \mathbb{R}^{N\times 3}$, we apply:

$$\mathbf{X}' = \mathbf{X}\mathbf{R}_z(\theta)^\top \tag{13}$$

where $\mathbf{R}_z(\theta)$ is a rotation matrix around the Z-axis (gravity direction). The rotation angle θ is sampled from discrete values $\{5°, 10°, 15°, \ldots, 355°\}$ with angle step $5°$. Rotation is sampled once per training batch: with probability 0.5, the whole batch is rotated by the same sampled angle. This improves robustness to yaw rotations around the gravity axis rather than imposing full 3D rotation invariance, which would be inappropriate for scenes with gravity and ground contact.

Importantly, the rotation is applied consistently to all objects in a scene and to all frames in a sequence, preserving the relative object relationships and temporal dynamics. Since the full trajectory is rotated before acceleration targets are computed, accelerations transform consistently:

$$\mathbf{a}' = \mathbf{R}_z(\theta)\mathbf{a} \tag{14}$$

Object Permutation Augmentation. Multi-object dynamics should be permutation-equivariant: the physics outcome should not depend on the arbitrary ordering of objects in the input. The architecture already satisfies this property for object tokens (Appendix B); during training, we additionally randomly permute object ordering with 50% probability as a robustness check. Given N_{obj} objects, we sample a random permutation $\pi \in S_{N_{\text{obj}}}$ and reorder all per-object tensors accordingly:

$$\{\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(N_{\text{obj}})}\} \to \{\mathbf{X}^{(\pi(1))}, \ldots, \mathbf{X}^{(\pi(N_{\text{obj}}))}\} \tag{15}$$

This augmentation is applied in the data loader before collation and feature computation, ensuring consistent permutation across all object attributes (positions, velocities, physics parameters). It helps detect implementation mistakes that would otherwise introduce order dependence and improves robustness to arbitrary input arrangements at inference time.

Rotation is applied after moving each training batch to GPU, while object permutation is applied on the data-loading side before collation. Both add negligible overhead to the training pipeline.

F Runtime Performance and Computational Costs

F.1 Runtime Performance

We profile inference on MOVi-B scene 72, containing 10 objects with a total of $4{,}016$ vertices, using a 50-step rollout setting. Per-object vertex counts are $\{64, 51, 1142, 64, 64, 682, 682, 647, 51, 569\}$, covering diverse MOVi-B meshes including torus_knot (1,142), spot (682), suzanne (647), gear (569), cube (51), and several primitives (64 vertices each). As summarized in Tab. F13, the model has 174.8M parameters and a peak memory footprint of 1.80 GB.

The model core (point encoder + object-state interaction + anchor-object interaction) takes 18.61 ms per step, corresponding to ~ 54 FPS. Within the model core, the point encoder takes 2.67 ms (6.4%), object-state interaction takes 6.01 ms (14.4%), and anchor-object interaction takes 9.94 ms (23.7%). Geometric processing takes 23.25 ms per step, corresponding to ~ 43 FPS, and is dominated by CUDA KNN search at 22.80 ms (54.5%), while the differentiable Kabsch transform is negligible at 0.45 ms (1.1%). These results suggest that further speedups should primarily target the KNN search.

Table F13: RigidFormer Inference performance on the MOVi-B evaluation set

Overall Performance		
Parameters	174.8M	
Peak Memory	1.80 GB	
Total Objects	10	
Total Vertices	4016	
Model Core Components		
Component	Time (ms)	Percent
Point Encoder	2.67	6.4%
Object-Level Interaction	6.01	14.4%
Anchor-Object Interaction	9.94	23.7%
Subtotal	**18.61**	**54 FPS**
Geometric Processing		
CUDA KNN	22.80	54.5%
Diff Kabsch Transform	0.45	1.1%
Subtotal	**23.25**	**43 FPS**
Total	**41.86**	**23.9 FPS**

Vertex-Level Processing Analysis. To illustrate the efficiency of our object-level tokenization, we analyze the hypothetical computational cost if attention were computed at the vertex level. With $N_v = 4{,}016$ vertices, a single self-attention layer would require $4{\times}N_v^2{\times}D = 4{\times}4016^2{\times}768 \approx 49.5$ GFLOPs. In contrast, our object-level decoder self-attends over 10 object tokens and 16 unpositioned register tokens, requiring $4 \times 26^2 \times 768 \approx 2.1$ MFLOPs per layer for the attention matrix-value product. This is a $\sim 2.4{\times}10^4$ reduction in the quadratic attention term before accounting for linear projections and FFNs. The measured runtime in Tab. F13 includes the full model and geometric processing; the main bottleneck is KNN rather than object-level attention.

G Detailed Network Architecture

Tab. G14 summarizes the architecture used in the main experiments. The model has 174.8M trainable parameters and consists of a hierarchical PointNet encoder, a 4-layer object-level Transformer decoder with gated attention and register tokens, and a query-based anchor predictor. We do not use spatial state tokens in the reported model.

Anchor-Vertex Pooling projection. The AVP module described in Sec. 3 performs a normalized weighted sum of per-vertex encoder features, producing a vector with the same width as the encoder backbone (1024 channels in the reported model). To control the predictor input size and let the model learn a contact-aware compressed representation, we apply a lightweight two-layer MLP $1024 \to 256 \to 256$ with SiLU activation; the last linear layer is zero-initialized so that AVP starts as a near-identity contribution to the anchor query and the model can ramp it up smoothly during training. The projected $\mathbf{u}_t^{(i,k)} \in \mathbb{R}^{256}$ is the feature concatenated to the anchor query in the predictor.

Multi-scale cross-attention to decoder layer outputs. Although Sec. 3 describes the predictor as cross-attending to "the decoder object tokens", in the implementation each anchor query attends, in parallel, to a small set of decoder layer outputs $\{\mathbf{Z}_t^{(\ell)}\}_{\ell \in \mathcal{S}}$ rather than only the final layer. With four decoder layers we use $\mathcal{S}{=}\{0, 1, 2, 4\}$, i.e., the encoder-side input plus the three internal layer outputs, giving four scales of object representation. For each scale ℓ, the anchors run a separate cross-attention block with shared dimensions but independent parameters. The per-scale outputs are concatenated along the feature dimension and fused through a single linear layer back to $D{=}768$, after which the predictor head regresses the per-anchor acceleration. This multi-scale formulation lets the predictor mix shallow geometry-aware features with deeper interaction-aware features in one pass, similar in spirit to feature-pyramid heads that read intermediate representations.

Table G14: **Architecture Overview.** Summary of model components and their configurations.

Component	Configuration	Notes
Input Configuration		
Max objects per scene	$M = 16$	–
Max vertices per object	$N_v = 1200$	–
FPS anchors per object	$N_a = 4$	–
Encoder input dim	12D	no normals, no time concat
– Displacement vector	$\mathbf{d} \in \mathbb{R}^3$	nearest contact geometry
– Velocity	$\mathbf{v} \in \mathbb{R}^3$	$\mathbf{x}_t - \mathbf{x}_{t-1}$
– Relative position	$\mathbf{r} \in \mathbb{R}^3$	$\mathbf{x}_t - \mathbf{x}_{\text{ref}}$
– Physics params	$[m, \mu, \epsilon] \in \mathbb{R}^3$	mass, friction, restitution
Time step	$[s, s^2] \in \mathbb{R}^2$	per-layer FiLM conditioning
Encoder (Hierarchical PointNet)		
Backbone convolutions	Conv1d MLP	shared over objects
Hierarchical pooling	4 levels (100%, 50%, 25%, 12.5%)	multi-scale geometry
Object embedding	$D = 768$	fixed-size object token
Decoder (4-Layer Transformer)		
Hidden dimension	$D = 768$	–
Attention heads	$H = 6$ (head dim = 128)	–
Object self-attention	Gated SDPA	elementwise sigmoid gate
Register tokens	16 learnable tokens	unpositioned global workspace
Time conditioning	per-layer FiLM	code $[s, s^2]$
FFN	SwiGLU, $2.5\times$ expansion	RMSNorm + QK norm
Block attention residuals	block size S=4	inter-block residual [20]
Dropout	0.1	attention and FFN
Anchor Predictor		
Anchor queries	N_a per object	FPS anchors
Anchor-Vertex Pooling	256D feature	learned isotropic distance kernel
Predictor input	271D	anchor state + AVP/context features
Cross-attention	object and cross-object context	anchor-level prediction
Output	3D anchor acceleration	followed by Verlet + Kabsch
Position Encoding and Projection		
ARoPE dimension	96	set-aggregated anchor RoPE
Rigid projection	Kabsch via RoMa	differentiable alignment
Total Parameters	174.8M	main model

Cross-object key/value in the predictor. For each scale ℓ, the keys and values of the anchor cross-attention span *all* valid objects in the scene rather than only the anchor's own object. Concretely, the keys and values are the full $\mathbf{Z}_t^{(\ell)} \in \mathbb{R}^{M \times D}$ tensor, while the queries are the per-object anchor features grouped as (M, N_a, D). As a result, an anchor on object i can directly attend to the contextualized state of object $j \neq i$, providing an explicit cross-object pathway at the anchor level in addition to the cross-object interaction already present in the decoder's object-level self-attention. This is useful for contact reasoning: anchors near an inter-object collision can read state information from the partner object without going through another decoder layer. We use the standard scaled dot-product attention with the same gated-attention configuration described for the decoder, padded objects are masked, and ARoPE is applied to query/key positions so that the cross-object reads remain geometry-aware.

H Large-Scale Simulation

To evaluate the scalability of RigidFormer to scenes with significantly more objects than the MOVi benchmarks, we construct the WreckingBall dataset featuring dense multi-object collision scenarios.

H.1 Dataset Generation

We simulate a wrecking ball scenario where a spherical projectile collides with a wall of stacked cubes. Four scene configurations are generated with varying grid sizes:

- **WreckingBall-28**: $3 \times 3 \times 3$ cube arrangement (M=27 cubes + 1 ball = 28 objects)
- **WreckingBall-65**: $4 \times 4 \times 4$ cube arrangement (M=64 cubes + 1 ball = 65 objects)
- **WreckingBall-126**: $5 \times 5 \times 5$ cube arrangement (M=125 cubes + 1 ball = 126 objects)
- **WreckingBall-217**: $6 \times 6 \times 6$ cube arrangement (M=216 cubes + 1 ball = 217 objects)

Data Generation Parameters. For each grid configuration, we generate 20 independent simulation samples, yielding a total of 80 samples across all configurations. The dataset is split into 72 training samples (18 per grid size) and 8 test samples (2 per grid size). Each trajectory spans T=600 frames at native simulation frequency. Cubes are initialized in a regular grid with small random perturbations to induce varied collapse patterns. The wrecking ball is launched with randomized initial velocity toward the cube wall.

Output Data Format. Each sample contains the following per-frame data:

- **Vertex positions**: Per-object vertex positions $\mathbf{X}_t^{(i)} \in \mathbb{R}^{N_v^{(i)} \times 3}$ for each frame.
- **Mesh connectivity**: Static face connectivity stored per object for visualization only.
- **FPS anchors**: Deterministic Farthest Point Sampling anchors; the reported model uses N_a=4 anchors per object.

We do not use surface normals or spatial/global state tokens in this experiment.

Physics parameters are consistent across all configurations: cube mass m=1.0 kg, friction coefficient μ=0.5, and restitution ϵ=0.3. Each cube mesh contains 8 vertices (corner points), yielding total vertex counts of 224 (28×8), 520 (65×8), 1008 (126×8), and 1736 (217×8) for the four configurations, respectively.

H.2 Training Configuration

We train RigidFormer on all four WreckingBall configurations jointly, using the mixed dataset of 72 training samples. The model uses the same object-anchor architecture as the main MOVi experiments, including FiLM conditioning on $[s, s^2]$, but without surface normals or spatial/global state tokens. It uses N_a=4 FPS anchors per object, and the maximum number of objects per frame is set to 220 to accommodate the largest configuration (217 objects) with a buffer. Training uses a step size s=10 for temporal prediction, with a sequence length of 8 frames per training sample. Batch size is set to 3 due to the large object count. We do not use curriculum learning in this experiment.

The object-level tokenization enables efficient scaling: attention complexity grows as $O(M^2)$ in object count rather than $O((MN_v)^2)$ in vertex count. For WreckingBall-217 with 1736 total vertices, vertex-level attention would require ∼3M attention pairs, while our object-level design requires only ∼54K pairs (217 objects + 16 registers), a 55× reduction.

I Controllable Articulated Body Simulation

To provide a preliminary test of RigidFormer beyond independent rigid objects, we construct two articulated-body datasets with high-level control inputs. We treat each body part as an object-level component and condition the model on command signals.

I.1 Dataset Generation

ASE HumanoidHeading. We generate training data from the Adversarial Skill Embeddings (ASE) [27] framework using the HumanoidHeading task. The humanoid character is equipped with a sword and shield, comprising M=15 articulated body parts. Data generation employs a hierarchical reinforcement learning architecture with pre-trained checkpoints: a Low-Level Controller (LLC) that generates joint torques $\boldsymbol{\tau} \in \mathbb{R}^{N_{\text{dof}}}$ from latent skill codes, and a High-Level Controller (HLC) that outputs latent skill codes conditioned on heading objectives.

Each trajectory spans T=300 frames at 30 Hz (10 seconds of simulation). Per-frame control signals $\mathbf{c}_t \in \mathbb{R}^5$ include:

- Target speed $v_{\text{tar}} \in \mathbb{R}$ (locomotion velocity magnitude)
- Target direction $\mathbf{d}_{\text{tar}} \in \mathbb{R}^2$ (movement direction unit vector)
- Target facing direction $\mathbf{f}_{\text{tar}} \in \mathbb{R}^2$ (heading orientation unit vector)

Meshes are simplified to a maximum of 300 vertices per body part using quadric edge collapse decimation. Mesh connectivity is used only for visualization; the model consumes point positions and per-point features, without surface normals.

G1 Steering. We generate data from the MimicKit [25] framework using the Unitree G1 Steering task. The G1 robot comprises $M=31$ articulated body parts with significantly more complex geometry than ASE humanoids. Data generation uses the Adversarial Motion Priors (AMP) [26] steering policy trained for directional locomotion.

Each trajectory spans $T=300$ frames at 30 Hz. Per-frame control signals $\mathbf{c}_t \in \mathbb{R}^3$ include:

- Target speed $v_{\text{tar}} \in \mathbb{R}$ (locomotion velocity magnitude)
- Target direction $\mathbf{d}_{\text{tar}} \in \mathbb{R}^2$ (movement direction unit vector)

For G1, each body-part object is represented by 200 points sampled from the original robot mesh, without surface normals. Joint positions $\mathbf{q} \in \mathbb{R}^{N_{\text{dof}}}$ are stored for kinematic analysis.

Table I15: **Articulated Body Dataset Statistics.**

Property	**ASE Heading**	**G1 Steering**
Source Framework	ASE [27]	MimicKit [25] (AMP)
Character	Humanoid + Sword/Shield	Unitree G1
Body Parts (M)	15	31
Max Vertices/Part	300	200
Control Dim	5	3
Frames (T)	300	300
Frequency	30 Hz	30 Hz

I.2 Implementation Details

Control Signal Integration. To condition RigidFormer on high-level control signals, we use the same FiLM mechanism as temporal step-size conditioning. The no-normal per-vertex encoder input remains 12D; the FiLM conditioning vector is augmented from $[s, s^2]$ to $[s, s^2, \mathbf{c}_t]$, where $\mathbf{c}_t \in \mathbb{R}^5$ for ASE and $\mathbf{c}_t \in \mathbb{R}^3$ for G1. Thus, control commands modulate the encoder/decoder features through FiLM rather than being concatenated to every point feature.

Mesh Processing Pipeline. Raw data export proceeds in two stages:

1. **Physics Export:** We run the trained policy in Isaac Gym [24] and record per-frame rigid body transforms (position and quaternion) for each body part.
2. **Mesh Conversion:** Body part transforms are applied to reference meshes from MuJoCo XML files. Meshes exceeding vertex limits undergo quadric decimation.

Temporal Subsampling. We train with step size $s=10$ for both ASE and G1 (3 Hz effective rate) to capture meaningful locomotion dynamics rather than high-frequency contact oscillations.

Training Configuration We use $N_a=4$ FPS anchors per body part and the same object-anchor architecture as in the main rigid-body experiments, including FiLM conditioning. Surface normals and spatial/global state tokens are disabled for both ASE and G1; the only additional conditions are the high-level control signals injected through FiLM. The batch size is 4 for both datasets. We present this result as an extension study rather than the primary evidence for the method.

J Discussion on Temporal Step Sizes

Step-size conditioning controls the effective temporal resolution of the learned simulator. A small step size is closest to standard one-step rollout protocols and returns dense future states, but a fixed physical horizon then requires many autoregressive calls, increasing both computation and

accumulated rollout error. A larger step advances the learned world model over a longer interval per call, allowing it to expose sparse long-horizon future states more efficiently.

This trade-off is useful for planning. In many robotics settings, such as model-predictive planning or trajectory optimization, the planner needs to compare possible future object configurations and interaction outcomes, but does not always require every high-frequency contact frame. Coarse large-step rollouts can therefore provide a cheap long-horizon search signal, while smaller steps remain useful for local control, contact refinement, and short-horizon correction. A natural extension is adaptive time stepping: use large steps to explore candidate futures, then refine selected plans with smaller steps or uncertainty-aware rollouts.

K Limitations, Future Work, and Impact

K.1 Limitations and Future Work

RigidFormer is designed for object-level rigid-body dynamics from point clouds and assumes object labels that indicate which points belong to each object. Although we include partial point-cloud results by masking points inside each object's bounding box, severe partial observations can become challenging when the visible points capture too little of the object's shape. Future work could study stronger occlusions, real sensor noise, online object segmentation from raw observations, mixed rigid–deformable scenes, and more fine-grained adaptive time stepping.

The rigid projection step exactly preserves intra-object distances, but contact handling is still learned from data rather than solved with an explicit complementarity-based physics engine. This design keeps the model mesh-free and efficient, while leaving room for contact-aware losses, lightweight correction layers, or hybrid learned-analytic constraints in scenes with unusually sharp contact regimes. Our current study also focuses primarily on rigid objects, with articulated bodies treated as collections of object-level parts; extending the same representation to mixed rigid–deformable scenes is a natural next step.

K.2 Impact Statement

This work aims to improve machine learning methods for physical dynamics modeling, especially mesh-free simulation of rigid-body contact from point clouds. Efficient learned simulation can benefit robotics, graphics, and embodied AI by reducing the cost of rollout prediction, supporting planning over object interactions, and enabling controllable physical reasoning in virtual environments.

The main deployment consideration is reliability under distribution shift. Models derived from this work should be validated under the intended geometry, material, and contact conditions before being used in safety-relevant control loops. In high-stakes settings, predictions should be paired with uncertainty checks, safety constraints, and human or system-level oversight. The method operates on geometric state data rather than personal data, so we do not anticipate direct privacy risks from the core technique.