NeuPhysics: Editable Neural Geometry and Physics from Monocular Videos

Yi-Ling Qiao[†] **Alexander Gao**[†] **Ming C. Lin**
University of Maryland, College Park
[†]Equal Contribution

Abstract

We present a method for learning 3D geometry and physics parameters of a dynamic scene from only a monocular RGB video input. To decouple the learning of underlying scene geometry from dynamic motion, we represent the scene as a time-invariant signed distance function (SDF) which serves as a reference frame, along with a time-conditioned deformation field. We further bridge this neural geometry representation with a differentiable physics simulator by designing a two-way conversion between the neural field and its corresponding hexahedral mesh, enabling us to estimate physics parameters from the source video by minimizing a cycle consistency loss. Our method also allows a user to interactively edit 3D objects from the source video by modifying the recovered hexahedral mesh, and propagating the operation back to the neural field representation. Experiments show that our method achieves superior mesh and video reconstruction of dynamic scenes compared to competing Neural Field approaches, and we provide extensive examples which demonstrate its ability to extract useful 3D representations from videos captured with consumer-grade cameras.

1 Introduction

Monocular RGB videos are ubiquitous today, and computers' ability to understand spatial and physical properties embedded in their contents is essential for many applications. Can you imagine one day being able to build a fully-interactive digital twin of your surrounding world using just a smartphone? You could capture a moving, fully 3D portrait of yourself (Sec. 4.1), estimate the physical material parameters of objects and interact with them virtually (Sec. 4.3), playback existing recordings from new angles (Sec. 4.2), or even edit your favorite cartoon videos (Sec. 4.4). In this paper, we design a system that makes progress toward realizing all of these things. Taking a single RGB video as input, our method estimates the geometry and dynamics parameters of the scene, parameterized by neural fields [37], and enables subsequent editing and physically-based simulation.

While humans can easily infer 3D shapes and physical properties of their environment even from a single eye, these tasks remain challenging for computer algorithms. Geometry reconstruction from a single view is an intrinsically under-constrained problem: there is usually more than one 3D structure that could result in a given 2D projection of a scene. Another key challenge lies in choosing a suitable representation to model the complex geometry of the real world. Explicit representations like surface mesh [4, 38] are widely used in computer graphics, but they suffer from limited resolution and fixed topology. Implicit representations such as signed distance functions (SDFs) describe complex geometric structure more elegantly, but are not directly suitable for reasoning about dynamics properties. Estimating dynamics parameters is yet more challenging, since it is contingent upon correctly estimating geometry and its variation over time.

In this paper, we design an end-to-end differentiable rendering and simulation pipeline that can directly learn the geometry and dynamics from a monocular video. By assuming the existence of a

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

one-to-one dense mapping between temporal frames in 3D space, our method decouples the geometry estimation task into two complementary parts: a static reference frame, and a mapping function that encodes motion. The static reference frame consists of three neural fields: Appearance, Rigidity, and an SDF for geometry. Given a 3D spatial coordinate as input, these fields respectively output its RGB color, rigidity mask, and signed distance to the nearest surface, in the reference frame. Time-variant motion is encoded in a separate Motion field, which takes as input a 3D spatial coordinate (sampled along a ray), plus a learned latent code denoting a particular discrete time step, and outputs the 3D displacement from the sampled coordinate back to its corresponding reference frame coordinate. We utilize the density function from NeuS [61] for rendering, and the Rigidity mask from Non-Rigid NeRF [57] to regularize the motion and obtain foreground/background separation.

A convenient property that emerges from factorizing the scene into static geometry + dynamic motion fields is that by querying the motion field, we can establish a point-wise correspondence across frames, which we leverage to infer dynamics properties of the scene. Specifically, physics parameters are optimized by minimizing the difference between results from the differentiable simulator [12] and motion field. Upon learning reasonable parameters, the differentiable simulator can be used for physically-based animation and editing of the scene. Experiments show that our method outperforms competing approaches in surface reconstruction, video reconstruction, and novel view synthesis tasks, both qualitatively and quantitatively. Multiple examples also demonstrate our full pipeline's physically-based scene understanding and interactive scene-editing capabilities. Our code and data are available here: `https://sites.google.com/view/neuphysics`

In summary, the key contributions of this work are as follows:

- We present a framework for estimating and editing the geometry, appearance, and physics parameters directly from a single monocular video (Figure 3).
- By decoupling the learning of static SDF geometry and dynamic motion, our method effectively regularizes this under-constrained problem and enables *reconstruction of high-quality dynamic surface mesh sequences from monocular videos* (Figure 4).
- By connecting the neural-fields-based geometry reconstruction to a differentiable physics engine, this approach eliminates the need for manual construction of intermediate 3D models, and avoids poor initialization for the differentiable simulator, empowering users to *estimate the modeling and physics parameters from scratch* (Figure 6).
- We design a two-way conversion (Figure 2& 3) between the explicit hexahedral mesh and implicit neural fields in our end-to-end modeling, simulation and rendering pipeline, enabling users to (1) *extract high-quality explicit meshes* from the implicit fields, and (2) *interact with the implicit 3D field (e.g. add, delete, move, deform, and simulate)* by directly manipulating its explicit mesh counterpart.

2 Related Work

Differentiable volume rendering with implicit neural representations has recently been used to synthesize novel views, estimate geometry and appearance from images and videos. Neural Radiance Fields (NeRF) [37], and following works [9, 53, 2, 68] have achieved photorealistic rendering results even on complex scenes. The coordinate-based multi-layer perceptron (MLP) representation equips the scene with high resolution [74], can be effectively regularized [73], and is memory-efficient. Compared with differentiable surface rendering [41, 42], volume rendering can backpropagate gradients through the entire space, which leads to better optimization of complex geometry that contains topological changes, and mitigates getting stuck in local minima. The original density field from NeRF cannot directly reconstruct high-quality surface meshes, so neural SDF fields [61, 70, 44, 43, 67] have been proposed for multi-view geometry reconstruction. Compared to other discrete representations like meshes [60, 25, 34], point clouds [14, 33] and voxels [6, 65], SDF fields have higher resolution, and do not require pixel-level segmentation [40, 69, 26] for reconstruction tasks. Current approaches that employ neural SDF fields do not converge to accurate geometry when faced with dynamic scenes, while our method does so from a single video, which we show in figure 4.

Recent work has extended NeRF to videos of dynamic scenes [46, 1, 72], but they primarily focus on 2D image synthesis, not reconstructing 3D geometry. These works either directly condition the neural fields on time [13, 16, 63] or they learn a canonical field separately from time-dependent motion [45, 47, 57, 5]. We choose the latter strategy because it better constrains the learning problem

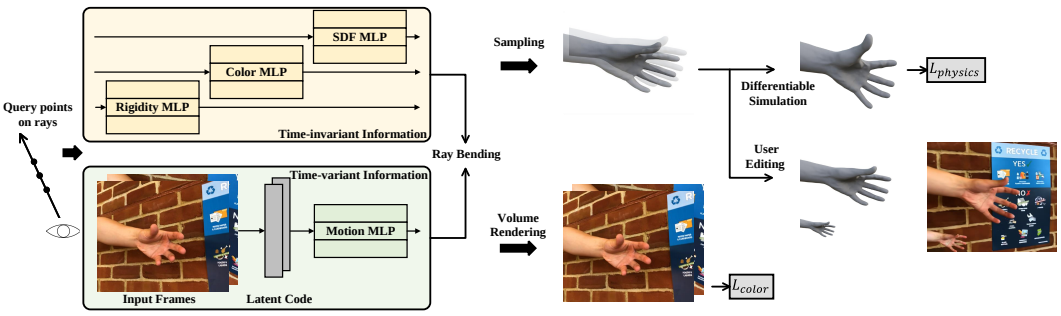


Figure 1: **Overall workflow**. *NeuPhysics* takes a monocular video as supervision. The time-invariant fields are defined on a reference frame, while time-dependent motion is applied to each point sampled in space. We then compute color loss from the volume rendered images. A differentiable simulator is embedded after optimizing the neural fields, to learn dynamics parameters and allow scene editing. A detailed computational graph can be found in Figure 3.

based on a reasonable inductive bias, and importantly, it provides us with point-wise correspondence between time steps that is useful for dynamics estimation. By representing geometry with a neural SDF field, and applying strong geometric priors, we can reconstruct dynamic 3D geometry better than existing dynamic extensions of NeRF, while even improving image reconstruction as shown in Section 4.1.

Differentiable simulators enable physics priors to better fit into data-driven approaches, and have therefore increasingly been applied to robotics and computer graphics tasks [54, 11, 21, 30, 55, 66]. These physically-based simulation methods have been developed for dynamical systems with different properties, including rigid bodies [7, 48, 8, 15, 36], soft bodies [24, 23, 49, 17, 12, 29, 28, 20], articulated bodies [19, 62, 50], cloth [31, 32], and fluids [58, 59, 22, 56].

3 Methods

Given a monocular video consisting of N RGB image frames $\{\mathbf{I}_i\}_{i=1}^N$, our approach estimates the 3D geometry within the camera's field of view, point-wise correspondence of the dynamic 3D scene across time, and static/dynamic segmentation. By integrating the differentiable physics module, we can learn physics parameters from the dynamic motion sequence, which we leverage for video editing and forecasting. Figure 1 shows an overview of our paper.

3.1 Rendering Pipeline

SDF and color networks. Similar to recent Neural Fields methods [37], we parameterize the geometry and appearance of the scene using coordinated-based Multi Layer Perceptrons (MLPs), which can render 2D images using the volume rendering scheme proposed in NeuS [61]. Geometry is represented by an SDF network $s_{\theta_{SDF}}(\mathbf{p}) : \mathbb{R}^3 \to \mathbb{R}$ that maps a spatial coordinate $\mathbf{p} \in \mathbb{R}^3$ to its signed distance to the nearest surface. Appearance is represented by another MLP $\mathbf{c}_{\theta_{color}}(\mathbf{p}) : \mathbb{R}^3 \to \mathbb{R}^3$ that describes the RGB value of each spatial point. During rendering, the points along a ray cast from a given pixel can be written as $\{\mathbf{p}(t) = \mathbf{o} + t\mathbf{v} | t \geq 0\}$, where $\mathbf{o}$ is the camera center, $\mathbf{v}$ is the unit direction of the ray, and t is distance from the camera center. The pixel's color is then calculated by integrating along the ray:

$$\mathbf{C}(\mathbf{o}, \mathbf{v}) = \int_0^{+\infty} w(s(\mathbf{p}(t))) \cdot \mathbf{c}(\mathbf{p}(t)) \, \mathrm{d}t, \tag{1}$$

where $w(\cdot)$ [61] is a weight function describing the contribution of each point's color based on its SDF value. More details can be found in the Appendix.

Motion network. However, the above framework is only designed to handle multi-view images of a static scene, which motivates another component to accommodate temporal information. Instead of directly conditioning the existing networks upon time, we choose to decouple the dynamic and static

scene components. Therefore, the SDF and color networks are defined on a static reference frame, while a motion network $\mathbf{b}_{\theta_{motion}}(\mathbf{l}_i, \mathbf{p}) : \mathbb{R}^{d_l} \times \mathbb{R}^3 \rightarrow \mathbb{R}^3$ estimates the offset field [57] for each time step, where $\mathbf{l}_i \in \mathbb{R}^{d_l}$ is a learned latent feature encoding temporal information about the i-th frame $\mathbf{I}_i$. The motion network bends each query ray $\{\mathbf{p}(t)\}$ to $\{\mathbf{p}'(t) = \mathbf{p}(t) + \mathbf{b}(\mathbf{l}_i, \mathbf{p}(t))\}$, which is then queried on the reference frame for SDF and color values.

Rigidity network. Since we want to segment and simulate dynamic objects within the scene downstream, we apply a rigidity network [57] $r_{\theta_{rigidity}}(\mathbf{p}) : \mathbb{R}^3 \rightarrow \mathbb{R}$ to distinguish dynamic foreground from static background. The mask $r(\mathbf{p}) \in [0, 1]$ defined on the reference frame indicates the likelihood that a point belongs to the swept volume of moving elements in the scene. The offset value becomes $\mathbf{b}^{\theta_{rigidity}}_{\theta_{motion}}(\mathbf{l}_i, \mathbf{p}(t)) = r_{\theta_{rigidity}}(\mathbf{p}) \cdot \mathbf{b}_{\theta_{motion}}(\mathbf{l}_i, \mathbf{p}(t))$. This formulation pushes dynamic regions toward larger $r(\cdot)$ values during training, while driving static areas toward zero.

3.2 Simulation Pipeline

Geometry reconstruction. Given the signed distance field $s(\cdot)$ and rigidity mask $r(\cdot)$, we can reconstruct the geometry of the dynamic and static parts of the scene. For example, the foreground in the i-th frame is contained within the swept volume of moving objects described by:

$$\mathcal{A}_i = \{\mathbf{p} | s(\mathbf{p} + \mathbf{b}(\mathbf{l}_i, \mathbf{p})) \leq k_{sdf}, r(\mathbf{p} + \mathbf{b}(\mathbf{l}_i, \mathbf{p})) \geq k_{rigidity}\}, \tag{2}$$

where k_{sdf} and $k_{rigidity}$ are constant thresholds for the SDF and rigidity values, respectively. This formulation is amenable to conversion into an explicit mesh representation. For rendering purposes, a *surface mesh* could be obtained using the marching cubes algorithm [35]. Alternatively, we represent the dynamic elements as a *volume mesh*. We sample over a discrete 3D grid to find vertices within volume $\mathcal{A}_i$, and if a grid point $\mathbf{p}$ satisfies Equation 2, we add that voxel to a hexahedral volume mesh $\mathbf{m}_i$. We choose hexahedral rather than tetrahedral volume mesh due to its simplicity and regular structure, and because hexahedral mesh can be simulated by DiffPD [12], a differentiable FEM simulator. Figure 2 visualizes the conversion between the implicit and explicit representation. Triangle surface mesh and tetrahedral volume meshes could be explored in future work.

Physics Engine. From the input video of a dynamic scene, we can extract a mesh sequence $\{\mathbf{m}_i\}_{i=1}^N$ with consecutive motion. An individual mesh model $\mathbf{m}_i$ is described by its vertices and hexahedral elements $\{\mathbf{Q}_i \in \mathbb{R}^{n_v \times 3}, \mathbf{E}_i \in \mathbb{N}^{n_e \times 8}\}$, where the rows of $\mathbf{Q}_i$ are the 3D coordinates of n_v vertices, and each row in $\mathbf{E}_i$ records the indices of eight vertices that form a hexahedron. We focus on modeling dynamic objects as soft bodies in this work. Compared to rigid bodies, soft bodies offer more expressiveness to model a wide range of deformable objects that that are important in applications like physically-based mixed reality and digital twins. As our physics engine, we use Projective Dynamics [3] implemented by DiffPD [12]. The governing equation can be written as

$$\mathbf{M}(\mathbf{Q}_{i+1} - \mathbf{Q}_i - h\dot{\mathbf{Q}}_i) = h^2(\nabla E(\mathbf{Q}_{i+1}, \theta_{physics}) + \mathbf{f}_{ext}), \tag{3}$$

where $\mathbf{M}$ is the mass matrix, $\mathbf{Q}_i$ is the vertex locations at frame i, h is the time step, $\dot{\mathbf{Q}}_i$ is the velocity, E is the potential energy due to deformation, $\theta_{physics}$ are the physics parameters, and $\mathbf{f}_{ext}$ are the external forces. Note that the gradient of the potential energy ∇E can be interpreted as the internal forces within the soft materials. This system uses implicit time integration for increased numerical stability, thus ∇E is defined on the frame $i + 1$. Objects are modeled by homogeneous co-rotated materials, where the material parameters $\theta_{physics}$ include Young's modulus and Poisson's ratio. A detailed formulation of the elastic energy E can be found in the supplementary materials of [12].

System identification can be valuable for constructing faithful digital twins, producing plausible animations, and control of dynamical systems, but estimating states $\mathbf{Q}$ and $\dot{\mathbf{Q}}$, and parameters $\theta_{physics}$ and $\mathbf{f}_{ext}$ from only a monocular video is challenging due to the high-dimensionality and nonlinearity of the system. Recent progress in differentiable physics tackles this challenge by making the entire simulation differentiable. Gradients $\frac{\partial \mathcal{L}}{\partial \mathbf{Q}_i}$, $\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{Q}}_i}$, $\frac{\partial \mathcal{L}}{\partial \theta_{physics}}$, and $\frac{\partial \mathcal{L}}{\partial \mathbf{f}_{ext}}$, where $\mathcal{L}$ is a user-defined scalar loss function, are all available, and allow us to estimate the parameters using a gradient-based optimizer like Adam [27].

Parameter Estimation. Running one simulation step forward in time from the i-th frame gives us an updated model at frame $i+1$, dependent on parameters $\theta_{physics}$: $\mathbf{m}'_{i+1}(\theta_{physics}) = sim(\theta_{physics}, \mathbf{m}_i, i \rightarrow i+1)$. By simulating the reconstructed meshes in the differentiable physics engine $sim(\theta_{physics}, \cdot)$ and comparing with the ground truth reconstruction results as a supervision signal, we optimize the physics parameters $\theta_{physics}$ using gradient-based methods.

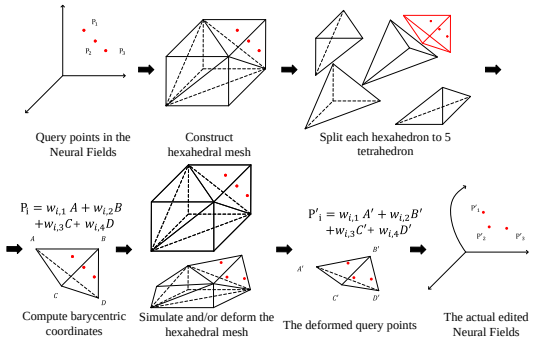


Figure 2: **The editing pipeline**. Users can edit the neural fields by operating on a hexahedral mesh.

One idea for a suitable loss function is to use a metric like Chamfer Distance $distance(\mathbf{m}_{i+1}, \mathbf{m}'_{i+1}(\theta_{physics}))$ between the extracted and simulated mesh $\mathbf{m}_{i+1}$ and $\mathbf{m}'_{i+1}(\theta_{physics})$, searching for the $\theta_{physics}$ that minimizes the distance. However, this strategy requires sampling and mesh reconstruction for every time step, which is computationally expensive. An ablation study and profiling numbers can be found in Figure 10. Additionally, the reconstructed mesh might vary significantly across time steps, thus cannot guarantee point-wise correspondence, making the dynamics discontinuous and non-differentiable w.r.t. the networks.

To save sampling time and make the gradient well-defined, we design a cycle-consistency loss to compute the distance between the simulation results and neural fields. Starting from an arbitrary mesh $\mathbf{m}_i$ sampled at the i-th frame, we run the differentiable simulation for j steps and get $\mathbf{m}'_{i+j}(\theta_{physics}) = sim(\theta_{physics}, \mathbf{m}_i, i \rightarrow i+j)$. When $\mathbf{m}'_{i+j}(\theta_{physics})$ is deformed back to the reference frame using the learned motion field, the result should correspond to the original shape $\mathbf{m}_i$. Since $\mathbf{m}'_{i+j}(\theta_{physics})$ and $\mathbf{m}_i$ share a common topology, an exact point-wise correspondence exists, and we can directly subtract their point clouds to obtain their distance:

$$\mathcal{L}_{physics} = \left\| (\mathbf{m}_i + \mathbf{b}^{\theta_{rigidity}}_{\theta_{motion}}(\mathbf{l}_i, \mathbf{m}_i)) - (\mathbf{m}'_{i+j}(\theta_{physics}) + \mathbf{b}^{\theta_{rigidity}}_{\theta_{motion}}(\mathbf{l}_{i+j}, \mathbf{m}'_{i+j}(\theta_{physics}))) \right\|_2 \quad (4)$$

where $\mathbf{b}(\mathbf{l}_i, \mathbf{m}_i)$ is the motion field at frame i that maps mesh $\mathbf{m}_i$ back to the reference frame. $\mathcal{L}_{physics}$ is differentiable w.r.t. the motion network, rigidity network, and physics parameters. Importantly, independent of the simulation duration, we only need to sample $\mathbf{m}_i$ once, and it can be used as a reference point to compare all other frames.

Scene Editing. The simulator in our pipeline imposes a strong physics prior for forecasting motion and modifying dynamics. Figure 2 shows how we edit the signed distance and color fields through the hexahedral mesh. Based on hexahedral mesh $\mathbf{m}_i$ extracted from the i-th frame, we can simulate it to get a modified state $\mathbf{m}'_{i+1}$. In order to render the modified scene, we add an additional 'bending' layer $\mathbf{b}_{edit}(\mathbf{m}'_i, \cdot)$ on top of the primary motion field $\mathbf{b}(\mathbf{l}_i, \cdot)$.

When querying a ray point $\mathbf{p}'$ to render the modified scene, we first check if the point is displaced by our editing operation, i.e. if it is located within the hexahedral mesh $\mathbf{m}'_i$. The inside-outside-test would be trivial if all hexahedra in $\mathbf{m}'_i$ remained in a regular grid, however, they may have deformed during simulation. Thus, we split each hexahedron into five tetrahedra – a query point $\mathbf{p}'$ is inside the hexahedron if and only if it is inside one of the tetrahedra. Following [71], we decide whether $\mathbf{p}'$ is inside a tetrahedron by its barycentric coordinates. For a tetrahedron consisting of four vertices $\{\mathbf{p}'_i = (x'_i, y'_i, z'_i)^\top\}_{i=1}^4$, the barycentric coordinates $\{c_i\}_{i=1}^4$ of $\mathbf{p}' = (x', y', z')^\top$ can be computed as $c_i = Det_i / Det_0$, where:

$$Det_0 = \begin{vmatrix} x'_1 & y'_1 & z'_1 & 1 \\ x'_2 & y'_2 & z'_2 & 1 \\ x'_3 & y'_3 & z'_3 & 1 \\ x'_4 & y'_4 & z'_4 & 1 \end{vmatrix}, Det_1 = \begin{vmatrix} x' & y' & z' & 1 \\ x'_2 & y'_2 & z'_2 & 1 \\ x'_3 & y'_3 & z'_3 & 1 \\ x'_4 & y'_4 & z'_4 & 1 \end{vmatrix}, Det_2 = \begin{vmatrix} x'_1 & y'_1 & z'_1 & 1 \\ x' & y' & z' & 1 \\ x'_3 & y'_3 & z'_3 & 1 \\ x'_4 & y'_4 & z'_4 & 1 \end{vmatrix}, ..., \quad (5)$$

and Det_i is obtained by substituting the i-th row of Det_0 with $(x', y', z', 1)$. $\mathbf{p}'$ is inside or on the surface of this tetrahedron if and only if $0 \leq c_i \leq 1$ holds for $i = 1, 2, 3, 4$. To determine tetrahedra to check for a given point, we use k-nearest neighbors $\mathbf{Q}'_{\mathbf{p}'} = KNN(\mathbf{m}'_i, \mathbf{p}')$ to locate mesh vertices $\mathbf{Q}'_{\mathbf{p}'}$ nearest to query point $\mathbf{p}$, and select tetrahedra that contain those vertices.

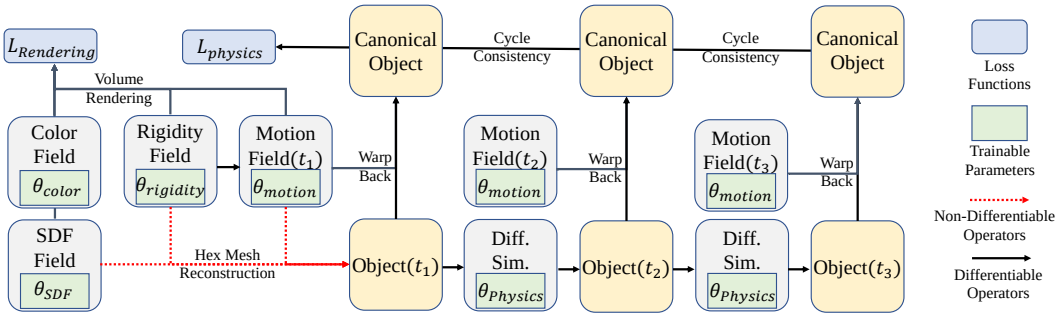


Figure 3: **Computational Graph During Training**. We show how data transfers among modules in a basic 3-step dynamical systems. All operators are differentiable, except the hex mesh reconstruction for the first frame. In our designed cycle consistency mechanism, the rigidity and motion networks serve as the bridge between the rendering and physics module.

If $\mathbf{p}'$ does belong to a tetrahedron, then it is displaced along with the mesh. Assuming the modification can be linearly interpolated inside a tetrahedron, the original coordinate $\mathbf{p}$ in the i-th frame becomes the weighted sum $\mathbf{p} = \sum_{i=1}^{4} c_i \mathbf{p}_i$, where $\mathbf{p}$ is the reference coordinate corresponding to $\mathbf{p}'_i$ (before editing). To account for the area that was once occupied by $\mathbf{m}_i$ but is now empty after editing: we simply assign those points a large positive SDF value such that they are effectively transparent during volume rendering and do not influence 3D reconstruction.

3.3 Training and Losses

With video as the main supervision signal used to train our system, we wish to minimize the distance between rendered and the ground truth images $\mathcal{L}_{color} = \left\| \mathbf{I}_i - \hat{\mathbf{I}}_i \right\|_1$. We include additional terms to regularize training. First, an important property of an SDF is its unit norm. Eikonal term [18], $\mathcal{L}_{Eikonal} = \sum_{i,k} (\|\mathbf{p}_{i,k}\|_2 - 1)^2$ is therefore added to the loss. We also want to constrain the motion of our scene (i.e. to be locally rigid, incompressible, and contain mostly static ragions), so we include an offset loss $\mathcal{L}_{offset} = \sum_{i,k} \|\mathbf{b}(\mathbf{l}_i, \mathbf{p}_{i,k})\|_2 + r(\mathbf{p}_{i,k})$ and divergence loss [57] $L_{divergence} = \sum_{i,k} \|\text{div}(\mathbf{b}(\mathbf{l}_i, \mathbf{p}_{i,k}))\|_2$. Finally, we use the physics loss $\mathcal{L}_{physics}$ when we train for physics parameters. The total loss function is:

$$\mathcal{L}_{total} = \mathcal{L}_{color} + w_{Eik}\mathcal{L}_{Eikonal} + w_{off}\mathcal{L}_{offset} + w_{div}\mathcal{L}_{divergence} + w_{phy}\mathcal{L}_{physics} \quad (6)$$

By default, we set $w_{Eik} = 0.1$, $w_{off} = 50.0$, and $w_{div} = 2.0$, though these weights may be further tuned depending on the scene. Figure 3 demonstrates the data and flow of gradients during training. In our proposed cycle-consistency physics loss, the rendering and simulation modules are connected by the rigidity and motion networks. Instead of jointly training all the parameters together, we adopt a sequential training strategy where the rendering networks are first trained to optimize $\mathcal{L}_{total}$ for 300,000 epochs, and the physics parameters are then trained to optimize $\mathcal{L}_{physics}$ for 100 epochs. The physics module is trained independently after the geometry networks converge because the differentiable simulation is much more expensive than querying of MLPs. An ablation study for the efficiency of our training strategy is conducted in Section 4.5. In future work it would be interesting to try a faster simulation engine which could be incorporated as a physics prior during training of the rendering networks. A full training cycle typically takes around 18 hours on an NVIDIA A5000 GPU. For the training data (used with consent), we note that the background must contain sufficient detail, or the Structure from Motion implementation by COLMAP [51, 52] could fail.

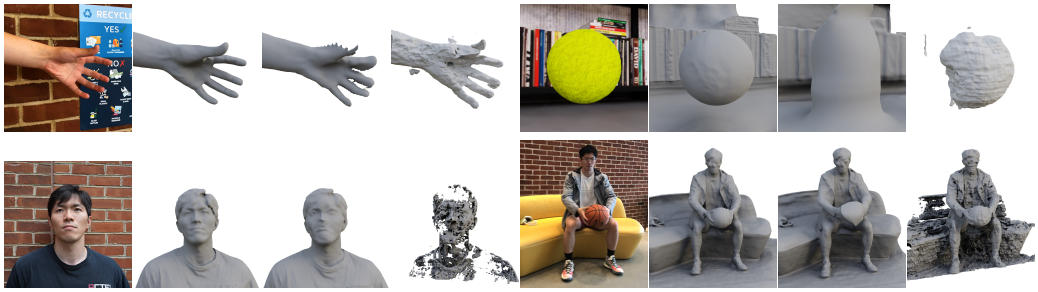
4 Experiments

4.1 Dynamic Geometry Reconstruction

We visualize mesh reconstruction quality of dynamic scenes in Figure 4. Table 1 shows quantitative results. Our method frequently outperforms competitive approaches. Additional geometry reconstruction comparisons can be found in Appendix E.

Method	Sitting			Standing			Basketball			Hand		
	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑
D-NeRF	**0.163**	0.914	34.217	0.251	0.868	**33.036**	0.332	0.831	32.803	**0.165**	0.885	34.146
NeuS	0.239	0.911	**34.563**	0.301	0.829	32.888	0.330	0.862	34.440	0.305	0.908	34.485
NRNeRF	0.223	0.892	32.019	0.269	0.863	31.549	0.340	0.830	32.411	0.205	0.913	33.999
Ours	0.166	**0.937**	**34.563**	**0.250**	**0.869**	32.881	**0.235**	**0.912**	**35.974**	0.225	**0.915**	**34.710**

Table 1: **Quantitative evaluation of video reconstruction**. We evaluate our method on both real-world and synthetic videos, outperforming alternative methods in most cases.



(a) Input (b) Ours (c) NeuS (d) D-NeRF (e) Input (f) Ours (g) NeuS (h) D-NeRF

Figure 4: **Surface reconstruction from dynamic videos**. Our method achieves higher-quality, more detailed reconstruction (e.g. on muscles, faces, and books) than other approaches.

4.2 Novel View Synthesis

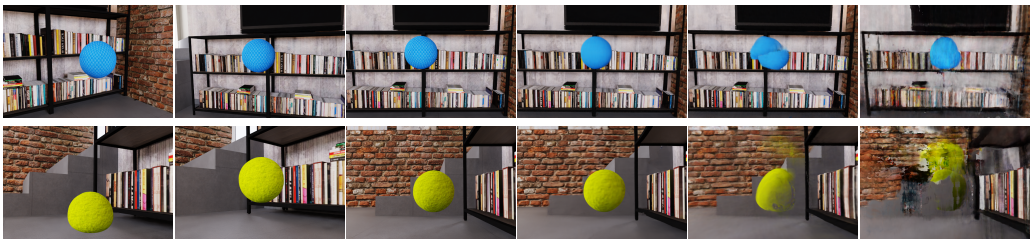
We simulate four cases of deformable objects moving in a room. We render each scene with two different camera trajectories, so that one path may be used for training, while the other is used as a ground truth to evaluate novel view synthesis. Quantitative results in Table 2 show that our method achieves the best synthesis quality in all metrics. Datasets and results are shown in Figure 5. (a) and (b) are frames from the training videos. (c) is taken at the same time step as (b), but it is a novel view from the test path. Notice that our method successfully synthesizes the dynamic scene from a novel view, while other methods struggle to reconstruct geometry and texture accurately.

4.3 Parameter Estimation

In these experiments, we first use our network to reconstruct the dynamic scene, then apply the embedded differentiable simulator to estimate physical parameters. Finally, we edit the physical parameters to generate novel video frames. More visual results can be found in the supplementary video. Figure 6 shows the initial frame of a bouncing ball sequence, where the ball is released from a height and collides with the ground. The ball is modeled by co-rotated material. Using DiffPD [12], we estimate both Young's modulus and vertical acceleration in two separate optimization tasks. The physics loss is minimized for 100 epochs (as described in Section 3.2) using Adam [27] optimizer with a learning rate of 0.01. We initialize Young's modulus to $2 \times 10^5 Pa$ and acceleration to $0 m/s^2$, and by the end of optimization, have estimated Young's modulus at $2.96 \times 10^6 Pa$ and acceleration at $3.78 m/s^2$. Having identified accurate physics parameters, we may choose to modify some of them from their true values, and generate videos that contain modified physical properties from the original on which they are based. For example, the original ball has no horizontal speed, so we add $0.3 m/s$ left and right initial velocities in Figure 6 (b) and (c), respectively. We also decrease the Young's modulus in Figure 6 (d), (e), and (f) from $2 \times 10^7 Pa$ to $2 \times 10^5 Pa$, where the deformation is getting more significant.

Method	Tennis Ball			Basketball			Blue Ball			Blue Cube		
	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑
NRNeRF	0.455	0.566	30.286	0.473	0.550	29.684	0.409	0.479	30.152	0.620	0.398	29.044
D-NeRF	0.596	0.392	29.170	0.565	0.425	29.073	0.596	0.317	28.937	0.775	0.229	27.855
Ours	**0.336**	**0.632**	**31.103**	**0.368**	**0.625**	**30.568**	**0.334**	**0.513**	**30.407**	**0.589**	**0.449**	**29.997**

Table 2: **Quantitative evaluation of novel view synthesis**. In this experiment, we evaluate a novel camera trajectory different from the training data. Even in this highly under-constrained task, our method benefits from the consistency of the geometry and significantly outperforms S.O.T.A. using all metrics.



(a) Input 1 (b) Input 2 (c) Novel view on 2 (d) Ours (e) NRNeRF (f) D-NeRF

Figure 5: **Qualitative evaluation of geometry reconstruction and novel view synthesis**. (a, b) Frames from the training videos. (c) Ground truth evaluation view. Consistent with numerical results in Table 2, our method generates images from novel views with higher quality than previous methods.

(a) Original Image (b) Left velocity (c) Right velocity (d) Harder material (e) Moderate material (f) Softer material

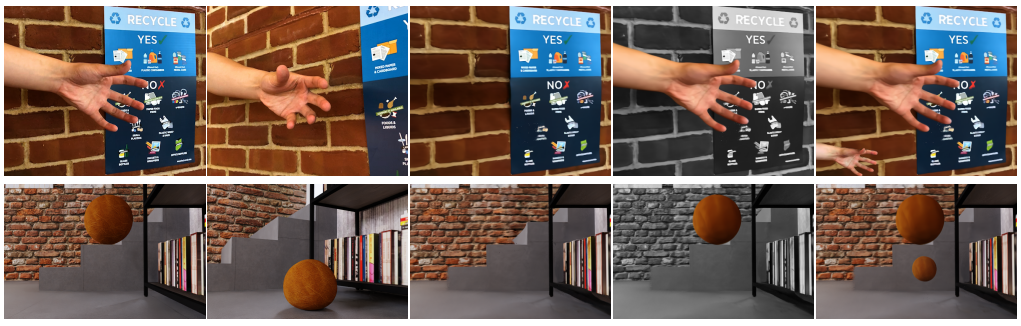
Figure 6: **Parameter estimation and editing by differentiable physics**. NeuPhysics can estimate the acceleration and Young's Modulus, generate new videos with different initial velocities and material parameters.

4.4 Editing

In this section, we demonstrate how our hybrid representation provides a flexible interface for users to directly modify the geometry on the 3D hexahedral mesh and volume render the results.

Region of Interest. First, we select a region of interest that we wish to edit. Our system naturally provides the scene rigidity mask and SDF values, so a foreground area $\mathcal{A}_i$ at the i-th frame can be computed as Equation 2. If users require a more precisely defined region of interest, a custom bounding box or even more complex hexahedra bounding volume $\mathcal{A}'$ can be provided. A hexahedra mesh $\mathbf{m}$ can be reconstructed from the final region of interest, $\mathcal{A}' \cap \mathcal{A}_i$,.

Delete. Figure 7 (a) and (b) are two distinct time steps from a given input video. In one, a hand flexes and deforms, and in the second, an elastic ball bounces. First, we delete the foreground from View 1 by simply setting the SDF value to a positive number (e.g. 1.0) for the query points inside the hexahedral mesh $\mathbf{m}$. Since we assume a bell-shaped density distribution centered at 0 [61], an area with large SDF value is less likely to be sampled, and therefore appears transparent. After removing the foreground from View 1, even though some parts of the background were occluded in the original, the other views of the scene provide sufficient textural/structural information to complete the missing background. Since editing takes place entirely in 3D space, the resulting 2D images feel sharp and physically grounded, as shown in (c).



(a) View 1 (b) View 2 (c) Delete (d) Appearance (e) Duplicate

Figure 7: **Geometry and appearance editing in videos**. Our method can delete, add, move, or change the appearance. Editing in the 3D space can provide users with realistic and clean results.

Appearance. We can also modify the output of the color network to change the appearance of a specific region. For example, we might like to emphasize the moving foreground of a video by desaturating the background. This is easily done by changing the original color $[r, g, b] = \mathbf{c}_{\theta_c}(\mathbf{p})$ to be $[\frac{r+g+b}{3}, \frac{r+g+b}{3}, \frac{r+g+b}{3}]$ for all the query points outside $\mathbf{m}$. Results are shown in Figure 7 (d).

Duplicate. Translation and scaling are basic operations in graphic design software. With our pipeline, users are able to copy, move, and scale the hexahedral mesh $\mathbf{m}$, creating a new version $\mathbf{m}'$. During volume rendering, the SDF and color values of the points $\mathbf{p}'$ inside $\mathbf{m}'$ are set equal to their counterparts $\mathbf{p}$ in $\mathbf{m}$. In Figure 7 (d), we duplicate an existing object and modify its size and position.

4.5 Ablation Study

In this section, we conduct several ablation studies using the bouncing ball example to justify our network design choices and training strategies.

	Rendering Module	Physics Module
Epoch	300,000	20
Time per epoch	~ 0.22 s	~ 70 s
Total time	~ 18 h	~ 0.5 h

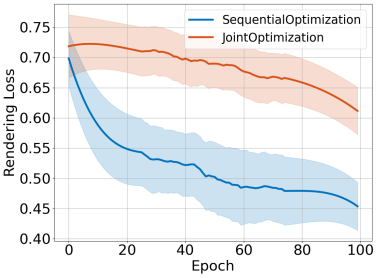


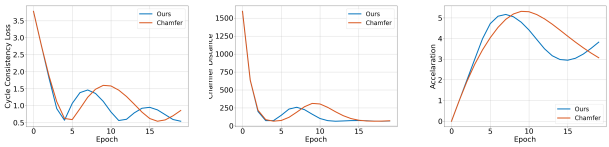
Figure 8: Joint training can be time-consuming and converges slowly.

Sequential Vs. Joint Optimization. Our method learns the scene sequentially by first optimizing the rendering networks, followed by the physics parameters. A seemingly more concise strategy might be to jointly train all parameters from scratch. Unfortunately, the joint training strategy is impractical for two reasons: (1) the table in Figure 8 indicates that the physics module converges rapidly thanks to the effectiveness of the differentiable simulator, but running the forward and backward simulation is costly (~ 70 s) compared to querying the rendering networks (~ 0.22 s) – training the rendering module and physics module together would require many more iterations (300,000 epochs), with each taking much longer ($\sim$ 70sec per epoch), making the training infeasible, and (2) before the rendering module converges and provides satisfactory geometry estimation, the geometry makes little sense for simulation, so gradients flowing from the $\mathcal{L}_{physics}$ do not help the optimization. The smoothed plot in Figure 8 indicates that joint training slows down the rendering optimization early on. Sequential optimization is therefore more practical.

We conducted two additional tests to study the relationship between the rendering and physics modules. First, during the sequential optimization, we also update the rendering networks. This harms the rendering quality (increases loss from 0.03 to 0.10) and does not help physics parameter estimation. Second, we attempt to optimize the rendered image from physics simulation to match the ground truth image. We found this process to be inefficient (2 minutes per image) and observed no significant benefits.

Physics Module. Within our sequential optimization strategy, there are still multiple options for physical parameter estimation. In our pipeline, we reconstruct the volume mesh $\mathbf{m}_1$ of the first frame, then predict the following mesh sequence $\mathbf{m}'_i = sim(\theta_{physics}, \mathbf{m}_1, 1 \to i)$ with the differentiable simulator. The physics loss for each frame can be computed in a cycle-consistency manner as in Equation 4. Our proposed cycle-consistency loss naturally guarantees a point-wise correspondence for each vertex, and does not require mesh reconstruction for every time step. For each epoch, we only reconstruct the volume mesh for the first frame, which reduces computation and makes the loss differentiable w.r.t. the rendering networks.

A more direct physics loss could rely on reconstructing the geometry of each frame and computing the Chamfer distance between the simulated and reconstructed meshes. We compare the two strategies in Figure 10. 'Ours' is optimized with our proposed cycle-consistency loss $\mathcal{L}_{physics}$ and 'Chamfer' is optimized with the Chamfer distance. (c) shows the value of *acceleration*, the optimization variable. (a) and (b) indicate that the losses have comparable performance in both metrics. However, 'Chamfer' takes significantly longer since every frame must be reconstructed. Our proposed cycle-consistency loss runs faster and maintains differentiability between the rendering and physics simulation module. Conceptually, decoupling the NeRF and differentiable physics has



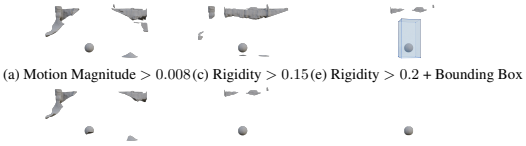
(a) Cycle Consistency Loss (d) Chamfer Distance Loss (f) Acceleration

Time	Ours	Reconstruct+Chamfer
Reconstruction	~ 2 s	~ 50 s
Simulation	~ 65 s	~ 65 s
Total	~ 67 s	~ 115 s

Figure 10: **Compare with optimizing Chamfer Distance.** An alternative to our proposed cycle-consistency loss is to use Chamfer Distance for the physics module. The two metrics have similar effects but our method can save much reconstruction time.

another drawback: a mesh reconstructed purely from SDF cannot capture the interior *stress and deformation*. Our proposed cycle-consistency loss addresses this concern, as it takes advantage of the learned motion field, so it is capable of finding the point-wise correspondence and describing the deformation.

Rigidity Map. Finding the correct dynamic foreground object is a prerequisite for physical simulation. The raw output of the SDF field can only reconstruct a global scene, so the rigidity map is used to segment the dynamic part of the scene (assumed to be the foreground in this case). The rigidity map collects the motion information from all frames and assigns a high rigidity mask value to dynamic regions. Figure 9 (c, d) visualizes the rough segmentation results from using different rigidity thresholds. We empirically find 0.2 to be a reasonable default threshold for separating the foreground from background. If the rigidity map does not provide clean separation, users may manually specify a bounding box covering the moving object swept volume, and get the final foreground as shown in (e, f). An alternative segmentation approach is to use the magnitude of the motion field as a criterion to separate dynamic foreground, i.e. large motion area corresponds to dynamic objects (and the converse). However, in some frames, dynamic parts might only contain small offsets from the canonical frame. For example, a cyclic bouncing ball could overlap with the canonical position and thus have offset values of zero, even though it should be classified as a dynamic area. Figure 9 (a) and (b) show results of filtering the scene using motion magnitude. It is not a good strategy, since a large portion of the background remains even as the ball is partially clipped off in (b).



(a) Motion Magnitude > 0.008 (c) Rigidity > 0.15 (e) Rigidity > 0.2 + Bounding Box

(b) Motion Magnitude > 0.01 (d) Rigidity > 0.2 (f) Final Foreground

Figure 9: **Region of Interest.** Motion magnitude (a, b) is an intuitive criterion for static/dynamic separation, but does not work well in practice. The rigidity map (c, d) provides a rough, but reasonable separation. (e, f) We can also specify a more precise ROI with a bounding box.

5 Conclusion

In this paper, we design an end-to-end pipeline that can successfully estimate and enable physically-based editing of geometry, appearance, and physics of a dynamic scene from a single video. Some possible future directions include incorporating semantic priors to further constrain the optimization, and leveraging pre-trained networks [10] as supervision to further regularize training. For example, optical flow / scene flow networks could be used as a motion detection signal, while a depth estimation network could help to initialize the signed distance field. Another useful future goal would be to estimate texture [64] and UV maps directly, allowing artists to construct 3D models in standard formats with less effort. Training and inference could also be accelerated dramatically by adopting a multi-resolution hash encoding [39]. Finally, we could couple this differentiable framework with other types of dynamics, to support a wider range of objects and scenes, e.g. cloth, tetrahedra mesh [49], or even robots [20].

Acknowledgements. This research is supported in part by Dr. Barry Mersky and Capital One E-Nnovate Endowed Professorships, and ARL Cooperative Agreement W911NF2120076.

References

[1] Benjamin Attal, Eliot Laidlaw, Aaron Gokaslan, Changil Kim, Christian Richardt, James Tompkin, and Matthew O'Toole. Törf: Time-of-flight radiance fields for dynamic scene view synthesis. *Advances in neural information processing systems*, 34, 2021.

[2] Mark Boss, Raphael Braun, Varun Jampani, Jonathan T Barron, Ce Liu, and Hendrik Lensch. Nerd: Neural reflectance decomposition from image collections. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12684–12694, 2021.

[3] Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: Fusing constraint projections for fast simulation. *ACM Transactions on Graphics (TOG)*, 33(4), 2014.

[4] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[5] Hsiao-yu Chen, Edgar Tretschk, Tuur Stuyck, Petr Kadlecek, Ladislav Kavan, Etienne Vouga, and Christoph Lassner. Virtual elastic objects. *arXiv preprint arXiv:2201.04623*, 2022.

[6] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European conference on computer vision*, pages 628–644. Springer, 2016.

[7] Filipe de Avila Belbute-Peres, Kevin A. Smith, Kelsey R. Allen, Josh Tenenbaum, and J. Zico Kolter. End-to-end differentiable physics for learning and control. In *Neural Information Processing Systems*, 2018.

[8] Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis wyffels. A differentiable physics engine for deep learning in robotics. *Frontiers in Neurorobotics*, 13, 2019.

[9] Frank Dellaert and Lin Yen-Chen. Neural volume rendering: Nerf and beyond. *arXiv preprint arXiv:2101.05204*, 2020.

[10] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. Depth-supervised nerf: Fewer views and faster training for free. *arXiv preprint arXiv:2107.02791*, 2021.

[11] Tao Du, Kui Wu, Andrew Spielberg, Wojciech Matusik, Bo Zhu, and Eftychios Sifakis. Functional optimization of fluidic devices with differentiable stokes flow. *ACM Transactions on Graphics (TOG)*, 2020.

[12] Tao Du, Kui Wu, Pingchuan Ma, Sebastien Wah, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. DiffPD: Differentiable projective dynamics with contact. *arXiv:2101.05917*, 2021.

[13] Yilun Du, Yinan Zhang, Hong-Xing Yu, Joshua B. Tenenbaum, and Jiajun Wu. Neural radiance flow for 4d view synthesis and video processing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.

[14] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613, 2017.

[15] 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, 2021. URL `http://github.com/google/brax`.

[16] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5712–5721, 2021.

[17] Moritz Geilinger, David Hahn, Jonas Zehnder, Moritz Bächer, Bernhard Thomaszewski, and Stelian Coros. ADD: Analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Transactions on Graphics (TOG)*, 39(6), 2020.

[18] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. *arXiv preprint arXiv:2002.10099*, 2020.

[19] Sehoon Ha, Stelian Coros, Alexander Alspach, Joohyung Kim, and Katsu Yamane. Joint optimization of robot design and motion parameters using the implicit function theorem. In *Robotics: Science and Systems (RSS)*, 2017.

[20] Eric Heiden, Miles Macklin, Yashraj S Narang, Dieter Fox, Animesh Garg, and Fabio Ramos. DiSECt: A Differentiable Simulation Engine for Autonomous Robotic Cutting. In *Proceedings of Robotics: Science and Systems*, Virtual, July 2021. doi: 10.15607/RSS.2021.XVII.067.

[21] Eric Heiden, David Millard, Erwin Coumans, Yizhou Sheng, and Gaurav S Sukhatme. Neuralsim: Augmenting differentiable simulators with neural networks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9474–9481. IEEE, 2021.

[22] Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to control PDEs with differentiable physics. In *ICLR*, 2020.

[23] Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B. Tenenbaum, William T. Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. ChainQueen: A real-time differentiable physical simulator for soft robotics. In *ICRA*, 2019.

[24] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. DiffTaichi: Differentiable programming for physical simulation. In *ICLR*, 2020.

[25] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3907–3916, 2018.

[26] Petr Kellnhofer, Lars C Jebe, Andrew Jones, Ryan Spicer, Kari Pulli, and Gordon Wetzstein. Neural lumigraph rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4287–4297, 2021.

[27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[28] Jatavallabhula Krishna Murthy, Miles Macklin, Florian Golemo, Vikram Voleti, Linda Petrini, Martin Weiss, Breandan Considine, Jérôme Parent-Lévesque, Kevin Xie, Kenny Erleben, et al. gradSim: Differentiable simulation for system identification and visuomotor control. In *ICLR*, 2021.

[29] Sizhe Li, Zhiao Huang, Tao Du, Hao Su, Joshua Tenenbaum, and Chuang Gan. Contact Points Discovery for Soft-Body Manipulations with Differentiable Physics. In *International Conference on Learning Representations (ICLR)*, 2022.

[30] Xuan Li, Jessica McWilliams, Minchen Li, Cynthia Sung, and Chenfanfu Jiang. Soft hybrid aerial vehicle via bistable mechanism. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7107–7113. IEEE, 2021.

[31] Yifei Li, Tao Du, Kui Wu, Jie Xu, and Wojciech Matusik. Diffcloth: Differentiable cloth simulation with dry frictional contact. *ACM Trans. Graph.*, 2022.

[32] Junbang Liang, Ming C. Lin, and Vladlen Koltun. Differentiable cloth simulation for inverse problems. In *Neural Information Processing Systems*, 2019.

[33] Chen-Hsuan Lin, Chen Kong, and Simon Lucey. Learning efficient point cloud generation for dense 3d object reconstruction. In *proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[34] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7708–7717, 2019.

[35] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.

[36] Pingchuan Ma, Tao Du, Joshua B. Tenenbaum, Wojciech Matusik, and Chuang Gan. RISP: Rendering-invariant state predictor with differentiable simulation and rendering for cross-domain parameter estimation. In *International Conference on Learning Representations*, 2022.

[37] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[38] Kaichun Mo, Shilin Zhu, Angel Chang, Li Yi, Subarna Tripathi, Leonidas Guibas, and Hao Su. PartNet: A Large-scale Benchmark for Fine-grained and Hierarchical Part-level 3D Object Understanding. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[39] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. doi: 10.1145/3528223.3530127. URL `https://doi.org/10.1145/3528223.3530127`.

[40] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3504–3515, 2020.

[41] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Transactions on Graphics (TOG)*, 38(6), 2019.

[42] Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, and Wenzel Jakob. Radiative backpropagation: an adjoint method for lightning-fast differentiable rendering. *ACM Transactions on Graphics (TOG)*, 39(4), 2020.

[43] Atsuhiro Noguchi, Umar Iqbal, Jonathan Tremblay, Tatsuya Harada, and Orazio Gallo. Watch it move: Unsupervised discovery of 3d joints for re-posing of articulated objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3677–3687, 2022.

[44] Michael Oechsle, Songyou Peng, and Andreas Geiger. Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *International Conference on Computer Vision (ICCV)*, 2021.

[45] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. *ICCV*, 2021.

[46] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M. Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *ACM Trans. Graph.*, 40(6), dec 2021.

[47] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. *arXiv preprint arXiv:2011.13961*, 2020.

[48] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Scalable differentiable physics for learning and control. In *ICML*, 2020.

[49] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Differentiable simulation of soft multi-body systems. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[50] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Efficient differentiable simulation of articulated bodies. In *ICML*, 2021.

[51] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[52] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016.

[53] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020.

[54] Changkyu Song and Abdeslam Boularias. Learning to slide unknown objects with differentiable physics simulations. In *Robotics: Science and Systems (RSS)*, 2020.

[55] Andrew Spielberg, Allan Zhao, Yuanming Hu, Tao Du, Wojciech Matusik, and Daniela Rus. Learning-in-the-loop optimization: End-to-end control and co-design of soft robots through learned deep latent representations. In *Neural Information Processing Systems*, 2019.

[56] Tetsuya Takahashi, Junbang Liang, Yi-Ling Qiao, and Ming C Lin. Differentiable fluids with solid coupling for learning and control. In *AAAI*, 2021.

[57] Edgar Tretschk, Ayush Tewari, Vladislav Golyanik, Michael Zollhöfer, Christoph Lassner, and Christian Theobalt. Non-rigid neural radiance fields: Reconstruction and novel view synthesis of a dynamic scene from monocular video. In *IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2021.

[58] Kiwon Um, Robert Brand, Yun (Raymond) Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative PDE-solvers. In *Neural Information Processing Systems*, 2020.

[59] Nils Wandel, Michael Weinmann, and Reinhard Klein. Learning incompressible fluid dynamics from scratch – towards fast, differentiable fluid models that generalize. In *ICLR*, 2021.

[60] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proceedings of the European conference on computer vision (ECCV)*, pages 52–67, 2018.

[61] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *NeurIPS*, 2021.

[62] Keenon Werling, Dalton Omens, Jeongseok Lee, Ionnis Exarchos, and C Karen Liu. Fast and feature-complete differentiable physics for articulated rigid bodies with contact. In *Robotics: Science and Systems (RSS)*, 2021.

[63] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. Space-time neural irradiance fields for free-viewpoint video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9421–9431, 2021.

[64] Fanbo Xiang, Zexiang Xu, Miloš Hašan, Yannick Hold-Geoffroy, Kalyan Sunkavalli, and Hao Su. NeuTex: Neural Texture Mapping for Volumetric Neural Rendering. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[65] Haozhe Xie, Hongxun Yao, Xiaoshuai Sun, Shangchen Zhou, and Shengping Zhang. Pix2vox: Context-aware 3d reconstruction from single and multi-view images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2690–2698, 2019.

[66] Jie Xu, Viktor Makoviychuk, Yashraj Narang, Fabio Ramos, Wojciech Matusik, Animesh Garg, and Miles Macklin. Accelerated policy learning with parallel differentiable simulation. In *International Conference on Learning Representations*, 2021.

[67] Gengshan Yang, Minh Vo, Natalia Neverova, Deva Ramanan, Andrea Vedaldi, and Hanbyul Joo. Banmo: Building animatable 3d neural models from many casual videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2863–2873, 2022.

[68] Guandao Yang, Serge Belongie, Bharath Hariharan, and Vladlen Koltun. Geometry processing with neural fields. *Advances in Neural Information Processing Systems*, 34, 2021.

[69] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. *Advances in Neural Information Processing Systems*, 33:2492–2502, 2020.

[70] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[71] Yu-Jie Yuan, Yang-Tian Sun, Yu-Kun Lai, Yuewen Ma, Rongfei Jia, and Lin Gao. Nerf-editing: Geometry editing of neural radiance fields. In *Computer Vision and Pattern Recognition (CVPR)*, 2022.

[72] Jiakai Zhang, Xinhang Liu, Xinyi Ye, Fuqiang Zhao, Yanshun Zhang, Minye Wu, Yingliang Zhang, Lan Xu, and Jingyi Yu. Editable free-viewpoint video using a layered neural representation. *ACM Transactions on Graphics (TOG)*, 40(4):1–18, 2021.

[73] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv:2010.07492*, 2020.

[74] Kai Zhang, Fujun Luan, Zhengqi Li, and Noah Snavely. Iron: Inverse rendering by optimizing neural sdfs and materials from photometric images. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022.