

Neural 4D Evolution under Large Topological Changes from 2D Images

AmirHossein Naghi Razlighi¹ Tiago Novello² Asen Nachkov¹ Thomas Probst Danda Paudel¹

¹INSAIT, Sofia University
Sofia, Bulgaria

²IMPA
Rio de Janeiro, Brazil

Abstract

In the literature, it has been shown that the evolution of the known explicit 3D surface to the target one can be learned from 2D images using the instantaneous flow field, where the known and target 3D surfaces may largely differ in topology. We are interested in capturing 4D shapes whose topology changes largely over time. We encounter that the straightforward extension of the existing 3D-based method to the desired 4D case performs poorly.

In this work, we address the challenges in extending 3D neural evolution to 4D under large topological changes by proposing two novel modifications. More precisely, we introduce (i) a new architecture to discretize and encode the deformation and learn the SDF and (ii) a technique to impose the temporal consistency. (iii) Also, we propose a rendering scheme for color prediction based on Gaussian splatting. Furthermore, to facilitate learning directly from 2D images, we propose a learning framework that can disentangle the geometry and appearance from RGB images. This method of disentanglement, while also useful for the 4D evolution problem that we are concentrating on, is also novel and valid for static scenes. Our extensive experiments on various data provide awesome results and, most importantly, open a new approach toward reconstructing challenging scenes with significant topological changes and deformations. Our source code and the dataset is publicly available at <https://github.com/insait-institute/N4DE>.

1. Introduction

Modeling and reconstructing the environment is an essential task for both biological and artificial systems in order to function on a higher level – serving a variety of purposes from navigation and mapping [3, 37], understanding and interaction [35, 36], visualization and collaboration [8], to artistic expression [4]. Scanning 3D objects and scenes in particular has recently gained widespread attention due to the advent of robust algorithms for photorealistic reconstruction using handheld cameras [15, 23, 33], and

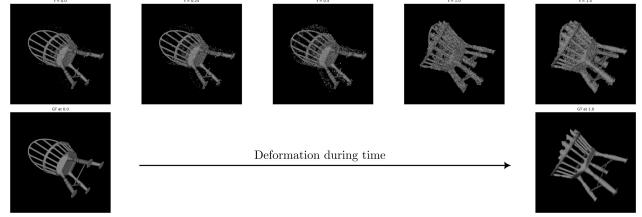


Figure 1. **Task scheme.** Our method learns the deformation animation of an object between two frames with large topological changes between them.

their commercial success ¹. Further democratization of the technology – similar to large language models – however is certainly limited by the static world assumption underlying those approaches [28]. This leads to undesired artifacts deteriorating both the geometric accuracy and the visual quality of the reconstruction. We are therefore interested in the problem of 4D reconstruction of *deformable scenes* from multiple posed RGB images.

Extending the reconstruction algorithms to handle deforming scenes introduces additional layers of ambiguity. Those could be resolved by increasing the number of cameras and frame rate, reducing the ill-posed problem towards a series of static many-view reconstructions [42]. This is however neither practical nor resource efficient. Instead, an appropriate deformation model is required to enable information flow across views that may be sparse across time and space [10]. Existing frameworks for dynamic scene reconstruction, however, come with several caveats, such as deformation priors that are object-specific [6, 31, 41], topology-restricting [28, 29, 32, 34], or too weak and therefore hard to optimize [9, 17]. Moreover, depending on the underlying representation of 3D scene and its deformation, extracting the 3D surface directly is not always possible (more details can be found in Section 2).

We propose **N4DE**, to efficiently reconstruct the surface of a dynamic, deformable scene by supervising the model based on RGB images only. To this end, we draw inspi-

¹See for example Reality Scan, Luma Ai, AR Code

ration from the complementary properties of implicit and explicit scene representations in three different ways:

First, we use signed distance functions (SDFs), which have proven their ability to implicitly represent complex surfaces [27], and have been successfully used for image-based reconstruction in conjunction with volumetric rendering [26, 39, 45, 47, 48]. Recently, Mehta et al. [22] showed promising results on how to perform evolution of implicit surfaces using explicit guidance. In fact, NIE demonstrates that we can incrementally update the parameters of an SDF – implemented as a multi-layer perceptron (MLP) – using the flow-field that is induced by an energy function that is being minimized during fitting. Performing deformations iteratively in implicit space also allows for spatially and temporally smooth transitions across topologies [22, 25]. This makes it a very suitable prior in our framework.

Second, we use the well-known HashGrid encoder [24] to discretize 3D space, thereby avoiding the slow convergence properties of fully implicit scene representations [23, 25]. Given a point (\mathbf{x}, t) in space-time, we extract its 3D latent representation of the location \mathbf{x} via trilinear interpolation, and feed it into an SDF module that is conditioned on the time t . We observe that this simple approach is not only able to model non-isometric deformations such as breaking a sphere, but also can evolve surfaces from one topology to another. In fact, we initialize all models in this work using the same unit sphere. Moreover, due to our choice of architecture, an explicit mesh representation can be directly obtained at any point of the continuous evolution. Importantly, this also includes interpolating and extrapolating the scene to unseen points in time.

Third, since our method already offers the surfaces explicitly, we are not bound to use expensive volumetric rendering to evaluate the photometric loss like [23, 39]. Instead, we propose an optional Rendering Module to place Gaussian splats directly on the mesh [12, 31], and to use a continuous implicit parametrization for the appearance-related properties of the splats. This way, we disentangle geometry and appearance, while allowing both to continuously change over time and space. Moreover, we can enjoy the fast rendering speed and convergence of Gaussian Splatting [15], while circumventing the issue of initialization and splitting or removing splats over the course of optimization.

To summarize, our contributions are as follows:

- **A new architecture for reconstructing deformable objects from images:** We use a HashGrid-based [24] approach with an SDF head trained using the Neural Implicit Evolution [22] method.
- **A continuous approach to Gaussian splatting.** Optionally, to predict color from target images, we estimate the splats properties on each surface point using a continuous implicit function, and optimize it jointly with the SDF head. The approach is **initialization-free** and does not

require merging or splitting splats.

- **Interpolation and extrapolation of deformations:** Our model is able to render the scene in unseen time steps. This fact shows that our model is actually learning the *deformation* and not just overfitting on the observed frames.
- **Experiments on diverse datasets:** We evaluate our model on a diverse set of object-centric scenes with different kinds of deformations. Alongside recognized benchmark scenes, we also evaluate on synthetic deforming animations created and rendered via Blender [5].

2. Related works

Our approach is situated at the intersection of implicit and explicit representations with regards to surface reconstruction, rendering, as well as deformation.

Scene Representations for Differentiable Rendering.

In the recent literature, Neural Radiance Fields (NeRF [23]) and Gaussian Splatting (GS [15]) have gained widespread attention. Both approach the problems of novel-view synthesis by reconstructing the 3D scene from a given set of posed RGB images. The core difference of the two methods lies in the scene representation and rendering operation: On the one hand, NeRF is built on an implicit scene representation akin to the plenoptic function (mapping 3D coordinates and 2D viewing angle to density and colors), and requires rather expensive raymarching and sampling to evaluate the volumetric rendering integral. On the other hand, representing the scene explicitly by fitting Gaussian primitives in GS yields much faster optimization and real-time rendering speed out of the box. Despite the competitive visual quality however, a crucial aspect is the ability to extract accurate scene geometry. While GS often results in a lack of geometric consistency and does not offer a direct way to obtain a mesh representation of the scene without additional postprocessing or constraints [12, 13, 38, 43], integrating implicit surface representations like Signed Distance Function (SDF) into NeRF’s volumetric rendering has proven to be a very effective strategy [26, 39, 45, 47, 48], directly allowing for surface extraction via MarchingCubes [20]. In this work, we combine implicit surface representations with GS-based rendering to reap the benefits of both worlds. Towards this direction, [44] learn continuous (implicit) splat properties on an (explicit) point cloud, yielding a compact representation that exploits similarities between neighboring splats, whereas [2, 21] link the SDF at the location of a Gaussian splat to its properties, leading to improved surface reconstruction. However, the problems of initializing and managing the splats and their location remain unsolved. We address those by deriving the splat location from the surface reconstructed via SDF. Moreover, our method is equipped with a powerful deformation prior to handle also non-static scenes.

Grid Encoding. Learning an implicit function typically

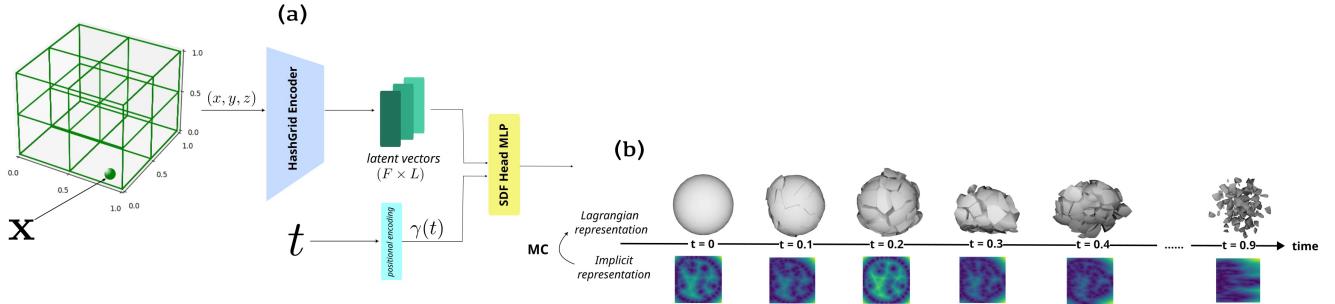


Figure 2. **Architecture of the SDF module.** Each point $\mathbf{x} \in [0, 1]^3$ is encoded using (a) HashGrid which is presented in Section 3.1. Then, the coordinate encoding (of dimension $F \times L$) are concatenated with the positional encoding of time ($\gamma(t)$) and fed into a MLP (SDF Head). (b) Signed distance value for each point is estimated, and the Lagrangian representation of mesh is extracted via Marching Cubes [20]. As our experiments show, the model can learn *continuous* representation of the deformation with respect to time.

relies on encoding the input position in a higher dimension using positional embeddings [23] or periodic activation functions [46]. Although accurate, it has been shown that the convergence is rather slow and can be drastically sped up via discretization or space partitioning [18, 24], typically done at multiple scales. Another aspect of the input space encoding is given by the dimensionality of the signal: in discrete space, the memory complexity grows exponentially with the number of dimensions. This becomes especially relevant for deformable scenes that require a time domain. To address this problem, [9] propose to only discretize K planes from each pair of dimensions, and [7] only discretize 3D space, keeping the time embedding continuous. We follow [7] for both our SDF and rendering heads due to its simplicity.

Deformation Priors. Many 4D approaches do not explicitly employ a deformation prior, and hope to compress the time-variant 3D scene without explicit guidance on the deformation [9, 17], leading to visually pleasing representations of even longer videos, but at the cost of geometric consistency, sample efficiency and convergence speed. On the other hand, [28, 29, 32, 34] learning MLP-based coordinate warping networks jointly with a static template. For known objects such as hands and body, task-specific articulated shape models can be used [6, 31, 41]. [14] use low rank approximations to factorize 4D space. Above approaches however are restricted to model only deformations from a single template, and cannot handle larger changes in topology.

Inspired by Mehta et al. [22], we are interested in modeling a deformation as driven by a continuous evolution of the level set equation. As demonstrated by Neural Implicit Surface Evolution [25] (NISE), implicit neural SDFs can be time conditioned, to represent the deforming surfaces that evolve naturally without topological constraints. Integrating NISE as a prior into our approach makes the ill-posed task of deformable reconstruction from images tractable.

3. Method

This section presents our model for reconstructing dynamic scenes under large topological changes from a sequence of posed RGB images. Our scene representation comprises two main components. First, we define an *implicit neural representation* (INR) to model the scene geometry evolution implicitly through a dynamic *signed distance function* (SDF) (Section 3.1). Second, we introduce a rendering module, represented by a new method to perform 3DGs [15] via a continuous MLP estimation of the splat properties (Section 3.4).

3.1. SDF Module

We use a HashGrid (based on Instant-NGP [24]) to implicitly encode the geometry of the dynamic scene. A general schema of our approach towards geometry prediction based on the SDF module is available in Figure 2. For this sake, we introduce a neural network $f_\theta : \mathbb{R}^3 \times \mathbb{R} \rightarrow \mathbb{R}$ which receives both spatial coordinates \mathbf{x} and time t . Then, for a given time step $t \in [0, 1]$, $f_\theta(\mathbf{x}, t)$ represent the SDF value of point \mathbf{x} at time t . We denote the corresponding zero-level sets at each time t by $S_t = \{\mathbf{x} | f_\theta(\mathbf{x}, t) = 0\}$.

While HashGrid-based MLPs offer local benefits for learning 3D shapes, directly extending their domain to $\mathbb{R}^3 \times \mathbb{R}$ to represent f would require 4D grids, implying in costly 4D interpolations. Wang et al. [40] avoids this problem by considering a 3D grid for each time step. However, this approach is hard to scale and suffers from high number of parameters and thus, high memory usage. Therefore, we take a different path based on a single 3D HashGrid encoder to overcome many of the aforementioned issues.

Precisely, we define a neural network $f_\theta(\mathbf{x}, t)$, with \mathbf{x} being a point in the space and t the time instance, that returns the signed distance to the surface at time t :

$$f_\theta(\mathbf{x}, t) = \text{MLP}(E(\mathbf{x}), \gamma(t)), \quad (1)$$

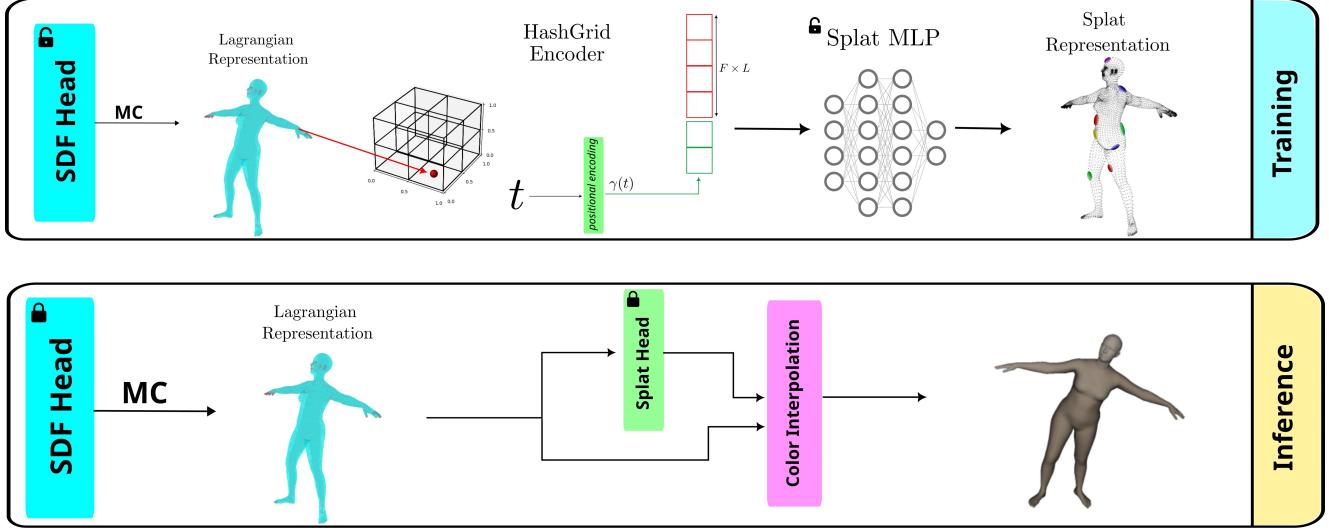


Figure 3. **Overall pipeline for training and inference with the rendering module.** In each iteration, the surface points estimated by the SDF Head are extracted via marching cubes [20]. They are then encoded via a HashGrid encoder [24] and the time embedding (via positional encoding) is concatenated to them. These features go through the rendering module to estimate the splat’s appearance properties (excluding for position and scale). At inference time, we use the final geometry and splat properties to do color interpolation and render the colored geometry.

where θ are the MLP parameters, E is the HashGrid coordinate encoder, and γ is a positional encoder for the time.

Since we are dealing with positional encoding of time, we scale time steps to $[0, 1]$. This choice will be explained further in the appendix. We extract the coordinate encoding from the HashGrid using trilinear interpolation. We control the speed-quality trade-off by changing the number of features F and the number of levels of detail L . For each point $\mathbf{x} \in \mathbb{R}^3$, we obtain a final encoding of size $F \times L$.

As it will become clear in the next Section 3.2, we require the difference of zero level-sets from one iteration to the next to be well behaved. Specifically, given a flow field $V^i(\mathbf{x})$ at iteration i ,

$$\forall \mathbf{x} \in \mathbb{R}^3, t \in [0, 1] \exists \delta \in \mathbb{R}_{\geq 0} : \text{CH}(S_t^{i+1}, S_t^i) < \delta \|V^i(\mathbf{x})\|_2, \quad (2)$$

where $\text{CH}(\cdot, \cdot)$ is the Chamfer distance, and S_t^i is the zero-level set of the SDF $f_{\theta_i}(\cdot, t)$. We show in the appendix that is indeed the case when using the coordinate encoding from the HashGrid, making the zero level-sets updates S_t^i and S_t^{i+1} move continuously between voxels.

3.2. Implicit Surface Evolution

We build our SDF prediction head on top of the approach of Mehta et al. [22], using the parametrized and time-conditioned SDF module, denoted as $f_{\theta}(\mathbf{x}, t)$ (see Eq. 1). To perform an iteration of the evolution, we first use marching cubes [20] on the implicit SDF to extract a Lagrangian

representation. With slight abuse of notation, we denote it as S_t . Then, we minimize an energy function ε defined in this explicit representation. This energy induces a flow field $V(\mathbf{x})$ to deform our Lagrangian as integration of the following partial differential equation (PDE):

$$\frac{d\mathbf{x}}{dt} = -\frac{\partial \varepsilon}{\partial \mathbf{x}} \rightarrow V(\mathbf{x}) \quad (3)$$

In particular, we define the ε to be sum of *Multi-scale photometric loss* and *Laplacian regularization*. Now, to update the parameters θ of f_{θ} , we need $\frac{\partial f_{\theta}}{\partial t}$. We can calculate it as:

$$\frac{\partial f_{\theta}}{\partial t} = -\nabla_{\mathbf{x}} f_{\theta} \cdot V \quad (4)$$

Then, we compute a non-parametric *next-best level-set estimate* using:

$$s^{i+1} = f_{\theta}^i(\mathbf{x}, t) - \Delta t \nabla_{\mathbf{x}} f_{\theta}^i(\mathbf{x}, t) \cdot V^i(\mathbf{x}), \quad (5)$$

which is defined on all finite vertex locations \mathbf{x} of the current Laplacian surface S_t . Δt is a hyper parameter dependent on dynamics of the flow field. f_{θ}^i shows the SDF Module parameterized by θ in iteration i . Now, the loss between current estimate of SDF with next-best zero-level set can be minimized to update the model parameters as,

$$L_{\text{evo}} = \frac{1}{|S_t|} \sum_{\mathbf{x}_j \in S_t} \|s^{i+1}(\mathbf{x}_j) - f_{\theta}^i(\mathbf{x}_j, t)\|^2. \quad (6)$$

We use this loss together with additional regularizations that will be explained in Section 3.3. An overview of the whole pipeline to estimate the geometry is shown on Fig. 2.

The MLP in the SDF module is composed of 4 hidden layers with hidden size of 128 neurons. We use hyperbolic tangent as the activation function due to its smoothness property suitable for SDF estimation. More details about model implementation and the choice of architecture are provided in the supplementary materials.

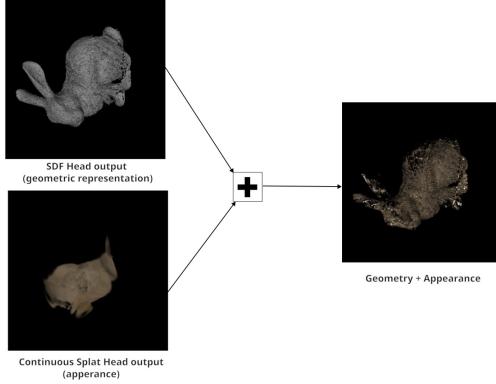


Figure 4. Overview of the proposed pipeline, applied on a Stanford bunny. A geometric/appearance representation is extracted using the method presented in Sec 3.1/Sec 3.4. The final colored mesh is given by the combination of these two representations.

3.3. Regularizations

One of the most important aspects of our model is the shared weights throughout the training process for each of the specific time steps. This means the evolution of e.g. $t = 0.2$ affects the evolution of $t = 0.1$ and vice versa. We will explain why this has a good effect in the training process for animations in which the same object is deforming. First, we use a *Laplacian regularization* among with our photometric loss in the first step of our pipeline as ε . This is the energy function which we try to minimize and by that, we calculate the flow field to compute *next-best non parametric level-set*. So, the ε in 3 is:

$$\varepsilon = L_{\text{photometric}} + \lambda_s L_{\text{ssim}} + \lambda_l L_{\text{laplacian}} \quad (7)$$

where $L_{\text{laplacian}}$ captures the smoothness of the surface and how evenly the vertices are distributed. We set multiplier $\lambda_l = 0.0002$ in the beginning and after specific number of epochs (nearly 500 epochs), we decrease it. L_{ssim} represents the SSIM loss between the estimated RGB image and the ground truth. This loss is essential for when we are doing Colored Prediction, since it introduces the structural difference between our estimate and the ground truth. In most of the experiments, we choose λ_s to be 0.01.

After obtaining the flow field, we can compute the *next-best estimate of level-set* as $s(x)$. So, we use this to update

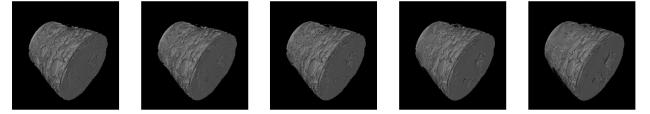


Figure 5. Estimated meshes at different timesteps for the Static Bracelet scene. The scene is only supervised at $t = 0$ and by the effect of the $\frac{\partial S_\theta}{\partial t}$ regularizer, it learns to be constant along time.

our SDF module (f_θ) parameters. The L2 loss between our current SDF estimate and $s(x)$ is called L_{evo} . We also add eikonal regularization [11] to find a better optimum for our SDF model. Another regularization which is used is *time consistency*. Since most deformation happen over larger timescales, we aim to regularize the change within a small window of frames. For this sake, we experimented with sampling of points in time and minimizing the difference between SDF predictions of these 2 consecutive timesteps. However, we found this method to be prohibitively expensive to compute during each epoch. Instead, we opt for penalizing the derivative of SDF w.r.t. to time. The complete loss can we written as,

$$L = L_{\text{evo}} + \lambda_{\text{eikonal}} (\|\nabla_x f_\theta\| - 1)^2 + \lambda_t \left| \frac{\partial f_\theta}{\partial t} \right|, \quad (8)$$

with a scheduled multiplier λ_t . It starts from an initial value (e.g. 0.05) and it is damped exponentially through each training iteration. This is to emphasize time consistency between consecutive frames in the beginning, whereas later we can reduce it to capture the differences between frames in a more detailed way.

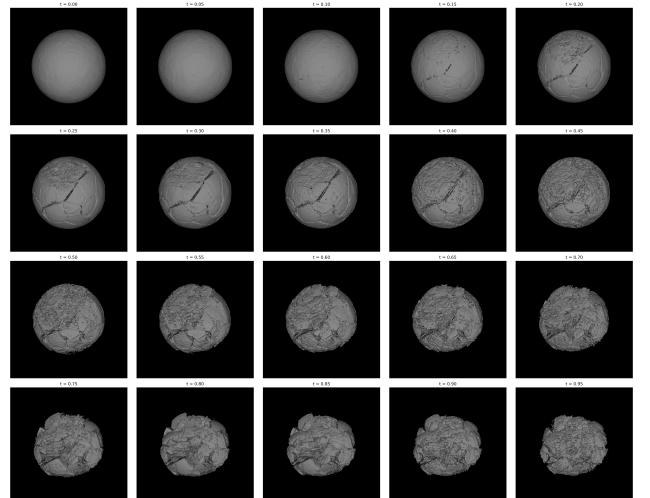


Figure 6. Estimated meshes at different timesteps for the Breaking Sphere (Longer) scene. The SDF Module is trained on 5 frames ($t = 0, 0.25, 0.5, 0.75, 1$) but the total deformation animation is learned and morphing is done in unseen time-steps.

3.4. Rendering Module

Our suggested pipeline is able to disentangle the geometry and appearance entirely and output two different representations for them: The SDF representation (for geometry) and the splat representation (for appearance). For the rendering module, we use a new approach toward Gaussian Splatting [15]. We represent the splats not explicitly, but implicitly via an MLP. In each epoch, we place the splats on the surface points of the estimated mesh (by the SDF module). Since we are taking a *continuous approach* towards Gaussian Splatting [15], we need to capture the evolution that is happening in each frame for the mesh (in geometry). It is important to note that we are extracting the mesh in each epoch all over again (as explained in 3.1). So, each vertex present in one epoch can be moved in any direction or even destroyed and replaced by another set of vertices. Our approach can cover this task and learn the splat properties implicitly. Another important aspect of our method is that the model does not need any careful initialization of the splat centroids. Actually, we initialize the splats to be on a sphere with unit radius, no matter what the target scene is, and they still converge via our method.

For this sake, we first extract the surface vertices from the SDF module. We suppose that we are placing one splat at each of these vertices. In the following, we omit the time parameter t for clarity. Given a SDF function f_θ^i trained at iteration i and \mathcal{M} represents the marching cubes process [20] yielding the centroids of splats as,

$$V_i = \mathcal{M}(f_\theta^i). \quad (9)$$

But since the vertices V_i are continuously evolving and changing, we need an approach for the MLP to be able to learn this evolution implicitly. The challenge is that we do not know the mapping between the new vertices (V_{i+1}) and the previous set of vertices (V_i). Here we use a second HashGrid encoder [24] to encode the surface points in an efficient, yet smooth, continuous way.

The encoded coordinates are fed to the MLP estimator of the rendering module as $r_\theta(x)$ to regress the appearance features as,

$$[\sigma(x), \text{SH}(x), R(x)] = r_\theta(x) \quad \forall x \in V_i. \quad (10)$$

The output contains opacity σ , spherical harmonic coefficients SH, and rotation R . The scale of the splats is not estimated, we instead set the scale of the splats to be equal to $\frac{1}{d_{vox}}$ where d_{vox} is the voxel distance in our 3D grid. The complete loss to train the rendering module is then given by,

$$\begin{aligned} L = & \|I_{\text{est}} - I_{\text{gt}}\|_1 + \lambda_s \text{SSIM}(I_{\text{est}}, I_{\text{gt}}) \\ & + \lambda_{\text{bg}} \|I_{\text{est}} - I_{\text{bg}}\|_2^2, \end{aligned} \quad (11)$$

with I_{est} and I_{gt} as the estimated RGB images and ground-truth RGB images, respectively. λ_s is set to 0.01 our experiments, and λ_{bg} can be used in case the object is small compared to the background, to prevent all splats to collapse to be predicted as background I_{bg} . The reason for fixing the scale of the splats is to avoid for them to get to large and cover multiple surface points (or interior and exterior surface points). Instead, we want to get a splat representation that is geometry-aware. So that each splat should try to cover the surface point perfectly and with correct orientation, color, and opacity. While this representation is useful in many cases (for example, representing geometry via splats on the surface), it may leave some empty space between the vertices (on each face of the mesh).

Finally in order to render the appearance to the geometry and get colored renderings, we first obtain the spherical harmonics of the surface points by via r_θ . Then, we evaluate the spherical harmonics \mathcal{S} to obtain the RGB color of a vertex v based on the viewing direction d as,

$$\text{RGB}(v) = \mathcal{S}(\text{SH}(v), d), \quad (12)$$

using a SH order of 3. Then, we interpolate the colors of vertices along each face (based on barycentric coordinates) of the mesh and rasterize the image:

$$c_{v_p} = \lambda_1 c_{v_1} + \lambda_2 c_{v_2} + \lambda_3 c_{v_3} \quad (13)$$

c is the color function, v_1 to v_3 are vertices of a specific triangle face of the mesh and v_p is a vertex inside this face. λ_i are barycentric coordinates on the face created by v_i . By this rasterization technique based on the learned R_θ , we have 3 representations for a dynamic deformable scene: 1. Geometry (extracted from the SDF module 3.1) 2. Surface-aligned splats 3. Enhanced RGB-colored mesh (via color interpolation)

4. Experiments

In this section, we perform experiments to evaluate our model (N4DE) and show its capabilities. The experiments are done on *Nvidia A6000 RTX* and *Nvidia A100 40GB*. Our appendix provides the experiment details and a comparison with NIE [22] showing that N4DE greatly improves dynamic scene reconstruction.

4.1. Dynamic Deformable Scenes

We test our method, N4DE, in several dynamic scenes, ranging from simple transformations (e.g., scaling) to complex changes (e.g., breaking). Table 1 presents quantitative evaluations demonstrating that N4DE can effectively reconstruct challenging cases, such as the Breaking Sphere in Figure 6 and Chair Deformation in Figure 1. To our knowledge, our method is the first to handle such topological deformations without assumptions. For a comprehensive evaluation of efficiency and quality, we also conduct

Table 1. Quantitative measurements of SDF Module 3.1 on different scenes (geometry-only). (All reported metrics are the **average values** calculated separately for each frame.)

Scene	\downarrow MSE	\uparrow PSNR	\uparrow SSIM	\downarrow LPIPS	\downarrow Chamfer distance	Num. Frames	Epochs
Static Stanford Bunny (single time-step)	0.0036	24.4196	0.8240	0.1458	0.0057	1	3000
Static Bracelet (single time-step)	0.0060	22.2141	0.7123	0.2450	0.0021	1	3000
Static Voronoi sphere (multi time-steps)	0.0044	23.5453	0.9087	0.0753	0.0447	10	2440
Static Multi-Object Bunny (single time-step)	0.0021	26.6825	0.8914	0.1203	0.0118	1	3500
Static Screaming Face (single time-step)	0.0076	21.1543	0.8345	0.1710	0.0167	1	3000
Dynamic Breaking sphere	0.0027	29.0797	0.8673	0.1395	0.0181	3	3000
Dynamic Breaking sphere (longer)	0.0033	26.5121	0.8491	0.1659	0.0241	5	3000
Dynamic Chair deformation	0.0066	22.7789	0.8326	0.1867	0.0089	2	5000
Dynamic Bunny deformation	0.0053	24.5015	0.8630	0.1416	0.0127	3	3000
Dynamic Eagle Statue deformation	0.0042	24.2799	0.8950	0.1027	0.0029	2	2000
Dynamic SMPL [19] scene #1	0.0030	25.4215	0.9274	0.1000	0.0050	3	3000
Dynamic SMPL [19] scene #2	0.0037	24.7243	0.9187	0.1037	0.0071	3	3000
Dynamic SMPL [19] scene #3	0.0052	23.0766	0.9071	0.1307	0.0275	3	3000

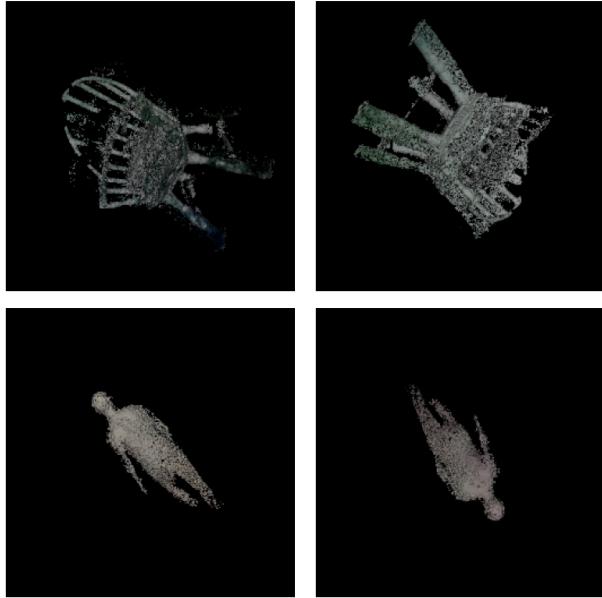


Figure 7. **Rendered colored meshes.** We showcase *Solid Colored Deforming Chair* and *SMPL Scene #2* [19]. The splat head is trained using only 1 random sampled image from target view.

the same experiments using the NIE approach [22] (detailed in the appendix), where it frequently fails in cases where our method succeeds.

4.2. Static Scenes

While our model is designed to capture deforming animations over time, it is still capable of reconstructing static objects in one single frame or during different time steps. For static scenes in one single time step, we configure our *Hash-Grid Encoder* with $F = 8$ features per-level and $L = 16$ levels, resulting in $F \times L = 128$ dimensional coordinate

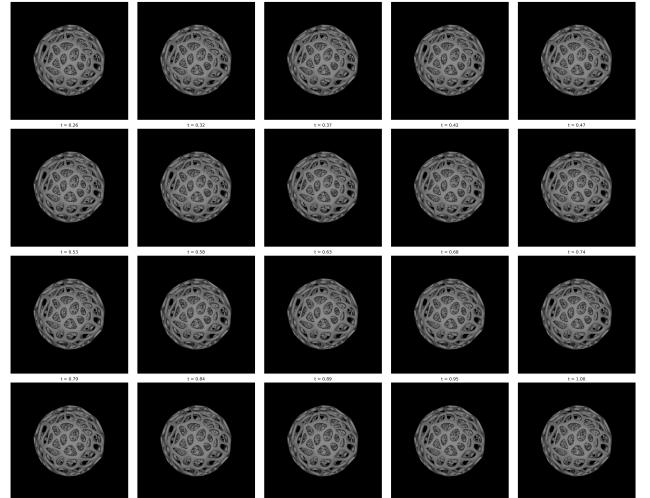


Figure 8. **Ten frame static Voronoi sphere reconstruction.** Although the model is supervised on 10 frames ($t = 0, 0.1, \dots, 0.9$), because of the time consistency regularization, the prediction is consistent among other times in this interval too.

embedding for each point. As an example, we reconstructed the *bracelet scene* with one single time step ($t = 0$) and 100 different views of resolution 256. Also, We've reconstructed the static *Voronoi sphere* scene with $n = 10$ different time steps and 10 views of each frame. The results of these experiments are available as quantitative (Tab. 1) and qualitative (Fig. 8).

4.3. Time Consistency regularization

For static scenes, we want the model to learn a *zero deformation* animation, or simply to be static among time. We therefore put a high weight on λ_t of the SDF loss in Eq. 8. Note that we can use the benefits of our well-behaved con-

tinuous implicit function and calculate the derivative of the output of our model with respect to time using PyTorch’s automatic differentiation [30].

For the dynamic cases (with the assumption of the same object being deformed along time, with little change from two consecutive frames), we want to start from the same mesh in the beginning and after some epochs, start to capture the fine differences between frames. So, we define a damping schedule $\lambda_t(e) = \lambda_t^0 \cdot 0.995^e$, which decreases exponentially over epochs e . λ_t^0 is the initial multiplier which in most experiments is set to 0.05. This time consistency term even works as expected in cases of *static scenes*, in which, the prediction is all consistent during every $t \in [0, 1]$, even if we did not supervise the model on that specific time step. An experiment showing this statement is plotted in Fig. 8.

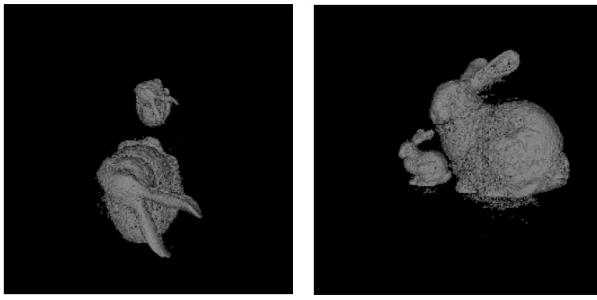


Figure 9. Multi object reconstructions. The *Multi Object* experiment showcases that our model is also capable of evolving a simple sphere into more than 1 object of target. The two images show the estimated geometry from 2 different views.

4.4. Interpolation and Extrapolation

One important outcome of our model is that it is capable of learning the *animation* and not just overfit on the supervised time-steps. It means that after training on sample time steps $t \in [i, j]$ if we infer the model on $\forall t; i < t < j$ we will get a meaningful mesh, with a morphing between t_i and t_j meshes. This fact can be seen in our dynamic experiments and static experiments (being consistent among all time-steps in a continuous manner) like Fig. 5 and Fig. 8.

4.5. Multi-object Reconstruction

One of the most important aspects of our model is that it is capable of reconstructing multiple objects. It is worth noting that in all of the cases (including the multi-object experience) we are starting from a simple sphere and evolving into the target objects based on the 2D image loss. Since we are using an implicit model, the initial sphere splits into two separate bunnies with different scales, as shown in Figure 9.

Table 2. Quantitative measurements of Rendering Module 3.4 on different scenes (geometry+appearance)

Scene	\downarrow MSE	\uparrow PSNR	Num. Frames	Epochs
Textured Stanford Bunny	0.0091	20.5140	1	3000
Textured SMPL [19] Scene #1	0.0175	20.1257	3	3000
Textured SMPL [19] Scene #2	0.0041	24.3593	3	3000
Textured SMPL [19] Scene #3	0.0055	22.9957	3	2700
Solid Colored Deforming Chair	0.0134	19.2670	2	3000

4.6. Implementation Details

Our pipeline consists of a *HashGrid Encoder*. The number of features per level is F and number of levels in the grid encoder is L , resulting in a coordinate embedding of $F \times L$ dimension for each point. For high-quality results we set $F = 8$ and $L = 16$ resulting in a 256 dimensional latent vector. The dimension of the look-up table T is 2^{19} and the minimum scale of the grid is 64 with scale of 1.5 and linear interpolation. We can lower each of these values to make the training and inference times faster with a cost of final quality. The **SDF Head MLP** contains 4 hidden layers with 128 neurons and *Tanh* activation function. The input of the model is $F \times L + 64$ which is concatenation of the encoded vectors with positional encoding of time. The **Rendering Module** also consists of a HashGrid [24] exactly like the SDF Module but with a minimum grid scale of 16 and scale of 1.3819. The *Rendering Module* has a shared backbone which processes the coordinate embeddings. This shared backbone consists of 3 hidden layers with *ReLU* activation and 128 number of neurons. The output of this shared backbone is a down-sampled 64 dimensional feature vector which is being fed to 3 different heads: *Spherical harmonics, Opacity, and Rotation*. Each of these heads is simply a linear layer with weight of dimensionality $64 \times d_{out}$. You can see the details of this Rendering module in 3. Some of the quantitative results of rendering module’s output are in the Tab. 2 and some rendered samples for qualitative comparisons are presented in Fig. 7 and Fig. 4.

5. Conclusions

We presented N4DE, a new approach towards a general way of reconstructing 4D scenes with large topological changes -like breaking and changes in the structure- via a neural evolution approach. This approach opens a new way towards modeling 4D deformations in the future. One limitation of our method is that it requires increasing the complexity of the network if we want to capture more complex deformations with high quality. This challenge is a step forward to enhance our method in the future.

References

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. [4](#)
- [2] Hanlin Chen, Chen Li, and Gim Hee Lee. Neusg: Neural implicit surface reconstruction with 3d gaussian splatting guidance. *ArXiv*, abs/2312.00846, 2023. [2](#)
- [3] Hao-Tien Lewis Chiang, Zhuo Xu, Zipeng Fu, Mithun George Jacob, Tingnan Zhang, Tsang-Wei Edward Lee, Wenhao Yu, Connor Schenck, David Rendleman, Dhruv Shah, Fei Xia, Jasmine Hsu, Jonathan Hoech, Pete Florence, Sean Kirmani, Sumeet Singh, Vikas Sindhwani, Carolina Parada, Chelsea Finn, Peng Xu, Sergey Levine, and Jie Tan. Mobility vla: Multimodal instruction navigation with long-context vlms and topological graphs. *ArXiv*, abs/2407.07775, 2024. [1](#)
- [4] Byungkuk Choi, Haekwang Eom, Benjamin Mouscadet, Stephen Cullingford, Kurt Ma, Stefanie Gassel, Suzi Kim, Andrew Moffat, Millicent Maier, Marco Revelant, Joe Letteri, and Karan Singh. Animatomy: an animator-centric, anatomically inspired system for 3d facial modeling, animation and transfer. *SIGGRAPH Asia 2022 Conference Papers*, 2022. [1](#)
- [5] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. [2](#)
- [6] Enric Corona, Tomás Hodan, Minh Vo, Francesc Moreno-Noguer, Chris Sweeney, Richard A. Newcombe, and Lingni Ma. Lisa: Learning implicit shape and appearance of hands. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20501–20511, 2022. [1, 3](#)
- [7] Jiemin Fang, Taoran Yi, Xinggang Wang, Lingxi Xie, Xiaopeng Zhang, Wenyu Liu, Matthias Nießner, and Qi Tian. Fast dynamic radiance fields with time-aware neural voxel. *SIGGRAPH Asia 2022 Conference Papers*, 2022. [3](#)
- [8] Shuo Feng, Weiping He, Xiaotian Zhang, Mark Billinghurst, and Shuxia Wang. A comprehensive survey on ar-enabled local collaboration. *Virtual Reality*, 27:2941 – 2966, 2023. [1](#)
- [9] Sara Fridovich-Keil, Giacomo Meanti, Frederik Warburg, Benjamin Recht, and Angjoo Kanazawa. K-planes: Explicit radiance fields in space, time, and appearance. *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12479–12488, 2023. [1, 3](#)
- [10] Han Gao, Rui long Li, Shubham Tulsiani, Bryan C. Russell, and Angjoo Kanazawa. Monocular dynamic view synthesis: A reality check. *ArXiv*, abs/2210.13445, 2022. [1](#)
- [11] Amos Groppe, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. *arXiv preprint arXiv:2002.10099*, 2020. [5](#)
- [12] Antoine Guédon and Vincent Lepetit. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5354–5363, 2024. [2](#)
- [13] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. *ArXiv*, abs/2403.17888, 2024. [2](#)
- [14] Hankyu Jang and Daeyoung Kim. D-tensorf: Tensorial radiance fields for dynamic scenes. *ArXiv*, abs/2212.02375, 2022. [3](#)
- [15] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), 2023. [1, 2, 3, 6](#)
- [16] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics (ToG)*, 39(6):1–14, 2020. [4](#)
- [17] Tianye Li, Mira Slavcheva, Michael Zollhoefer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, Richard Newcombe, and Zhaoyang Lv. Neural 3D Video Synthesis from Multi-view Video . In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5511–5521, Los Alamitos, CA, USA, 2022. IEEE Computer Society. [1, 3](#)
- [18] Ke Liu, Feng Liu, Haishuai Wang, Ning Ma, Jiajun Bu, and Bo Han. Partition speeds up learning implicit neural representations based on exponential-increase hypothesis. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5451–5460, 2023. [3](#)
- [19] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. SMPL: A skinned multi-person linear model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, 34(6):248:1–248:16, 2015. [7, 8, 3, 5](#)
- [20] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Seminal graphics: pioneering efforts that shaped the field*, pages 347–353. 1998. [2, 3, 4, 6, 1, 5](#)
- [21] Xiaoyang Lyu, Yang tian Sun, Yi-Hua Huang, Xiuzhe Wu, Ziyi Yang, Yilun Chen, Jiangmiao Pang, and Xiaojuan Qi. 3dgsr: Implicit surface reconstruction with 3d gaussian splatting. *ArXiv*, abs/2404.00409, 2024. [2](#)
- [22] Ishit Mehta, Manmohan Chandraker, and Ravi Ramamoorthi. A level set theory for neural implicit evolution under explicit flows. In *European Conference on Computer Vision*, pages 711–729. Springer, 2022. [2, 3, 4, 6, 7, 5](#)
- [23] 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. [1, 2, 3](#)
- [24] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a mul-

- tiresolution hash encoding. *ACM transactions on graphics (TOG)*, 41(4):1–15, 2022. 2, 3, 4, 6, 8, 1
- [25] Tiago Novello, Vinícius da Silva, Guilherme Gonçalves Schardong, Luiz Schirmer, Hélio Lopes, and Luiz Velho. Neural implicit surface evolution. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 14233–14243, 2022. 2, 3
- [26] Michael Oechsle, Songyou Peng, and Andreas Geiger. Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5589–5599, 2021. 2
- [27] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019. 2
- [28] Keunhong Park, Utkarsh Sinha, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Steven M Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5865–5874, 2021. 1, 3
- [29] 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. *arXiv preprint arXiv:2106.13228*, 2021. 1, 3
- [30] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017. 8
- [31] Pramish Paudel, Anubhav Khanal, Ajad Chhatkuli, Danda Pani Paudel, and Jyoti Tandukar. ihuman: Instant animatable digital humans from monocular videos. *ArXiv*, abs/2407.11174, 2024. 1, 2, 3
- [32] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10313–10322, 2020. 1, 3
- [33] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1
- [34] Liangchen Song, Anpei Chen, Zhong Li, Z. Chen, Lele Chen, Junsong Yuan, Yi Xu, and Andreas Geiger. Nerf-player: A streamable dynamic scene representation with decomposed neural radiance fields. *IEEE Transactions on Visualization and Computer Graphics*, 29:2732–2742, 2022. 1, 3
- [35] Byung soo Kim, Pushmeet Kohli, and Silvio Savarese. 3d scene understanding by voxel-crf. *2013 IEEE International Conference on Computer Vision*, pages 1425–1432, 2013. 1
- [36] Ayca Takmaz, Elisabetta Fedele, Robert W. Sumner, Marc Pollefeys, Federico Tombari, and Francis Engelmann. Open-mask3d: Open-vocabulary 3d instance segmentation. *ArXiv*, abs/2306.13631, 2023. 1
- [37] Holly A. Taylor and Barbara Tversky. Descriptions and depictions of environments. *Memory & Cognition*, 20:483–496, 1992. 1
- [38] Matias Turkulainen, Xuqian Ren, Iaroslav Melekhov, Otto Seiskari, Esa Rahtu, and Juho Kannala. Dn-splat: Depth and normal priors for gaussian splatting and meshing. *ArXiv*, abs/2403.17822, 2024. 2
- [39] 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. *arXiv preprint arXiv:2106.10689*, 2021. 2
- [40] Yiming Wang, Qin Han, Marc Habermann, Kostas Daniilidis, Christian Theobalt, and Lingjie Liu. Neus2: Fast learning of neural implicit surfaces for multi-view reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3295–3306, 2023. 3
- [41] Chung-Yi Weng, Brian Curless, Pratul P. Srinivasan, Jonathan T. Barron, and Ira Kemelmacher-Shlizerman. Humanerf: Free-viewpoint rendering of moving people from monocular video. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16189–16199, 2022. 1, 3
- [42] Bennett Wilburn, Neel Joshi, Vaibhav Vaish, Eino-Ville Talvala, Emilio R. Antúnez, Adam Barth, Andrew Adams, Mark Horowitz, and Marc Levoy. High performance imaging using large camera arrays. *ACM SIGGRAPH 2005 Papers*, 2005. 1
- [43] Yaniv Wolf, Amit Bracha, and Ron Kimmel. Surface reconstruction from gaussian splatting via novel stereo views. *ArXiv*, abs/2404.01810, 2024. 2
- [44] Minye Wu and Tinne Tuytelaars. Implicit gaussian splatting with efficient multi-level tri-plane representation. *ArXiv*, abs/2408.10041, 2024. 2
- [45] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. *ArXiv*, abs/2106.12052, 2021. 2
- [46] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. *Advances in Neural Information Processing Systems*, 34:4805–4815, 2021. 3, 2
- [47] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. *Advances in Neural Information Processing Systems*, 34:4805–4815, 2021. 2
- [48] Zehao Yu, Songyou Peng, Michael Niemeyer, Torsten Sattler, and Andreas Geiger. Monosdf: Exploring monocular geometric cues for neural implicit surface reconstruction. *ArXiv*, abs/2206.00665, 2022. 2

Neural 4D Evolution under Large Topological Changes from 2D Images

Supplementary Material

6. Initialization scheme

For initialization, we choose a simple yet efficient initialization approach for both of our **SDF Module** (Sec. 3.1) and **Rendering Module** (Sec. 3.4).

6.1. Initializing SDF Module

We first initialize the SDF Module to be a sphere in all sampled time-steps between 0 and 1. For this sake, we sample N points (in our experiments, $N = 2^{18}$) in each iteration and feed them into the SDF Module. We expect the outcome SDF values to represent a *unit sphere*. For this sake, we can use the SDF equation of a unit sphere:

$$s(\mathbf{x}) = \sqrt{x^2 + y^2 + z^2} - 1 \quad (14)$$

Also, we concatenate the time t as the fourth dimension with the above N points. Note that $t \sim U(0, 1)$. We use a combination of a simple MSE loss function and a MAPE loss function to initialize model parameters to predict a sphere:

$$\text{Loss} = \|f_\theta(\mathbf{x}, t) - s(\mathbf{x})\|_2^2 + \lambda_{mape} \cdot \frac{|f_\theta(\mathbf{x}, t) - s(\mathbf{x})|}{|s(\mathbf{x})|} \quad (15)$$

Here, λ_{mape} is the multiplier for the MAPE loss. We set $\lambda_{mape} = 0.2$ in our experiments. You can see a sample of the model predictions ($f_\theta(\mathbf{x}, t)$) in Fig. 10.

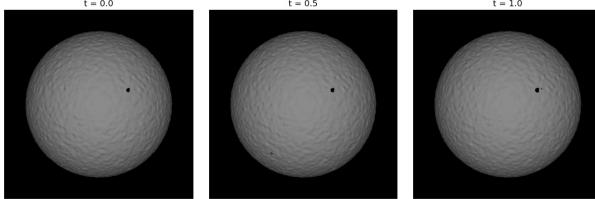


Figure 10. SDF Module (Sec. 3.1) initialized as sphere via our initialization schema.

6.2. Initializing Rendering Module

After initializing the SDF Module to predict the sphere initially, we also fit the Rendering Module to fit the splats on the surface of these spheres in all time steps $t \in [0, 1]$. Suppose V is the set of vertices after marching cubes process on the zero level-set of f_θ :

$$V_t = MC(f_\theta(\cdot, t)) \quad (16)$$

Here, MC denotes the marching cubes [20] process. We position each splat on the vertices of the current Lagrangian

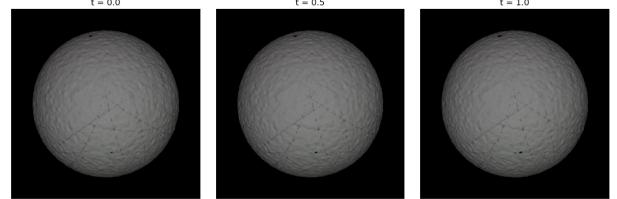


Figure 11. Rendering Module (Sec. 3.4) outcomes after our inference approach, rendered in different time steps. The Rendering Module is initialized by placing splats on the surface of a sphere to cover the sphere completely.

representation (V_t). For scaling, we experimented with regressing the scales via the Rendering Module and also fixing them. The results have shown that pre-defining them to be $\frac{1}{d_{voxel}}$ where d_{voxel} is the voxel size in our 3D sampling grid. In our experiments, based on the mesh resolution, d_{voxel} can be $\frac{1}{150}$ or $\frac{1}{200}$ or $\frac{1}{256}$. Then, to fit the splats on the surface of the mesh, we define the following loss function:

$$\text{Loss} = \|I_{est} - I_{GT}\|_1 \quad (17)$$

Here, I_{est} represents the rasterized estimated image, and I_{GT} represents the rendered (without texture) image of the sphere from the SDF Module. We know that for Gaussian Splatting [15] to converge, we do not need to have the correct estimate of colors necessarily. Whenever we want to infer the Rendering Module, we input the surface points (extracted from the SDF Module) and the time step (t) to the Rendering Module and get the predicted spherical harmonics. Then, using these spherical harmonics and the viewing direction, we render our colored mesh based on the interpolation method explained in 3.4. The outcome of this process after initializing the Rendering Module is plotted in Fig. 11.

7. Model Architecture

In this section, we will discuss the different factors that influence the outcome of our model and why we chose these architectural choices.

7.1. The choice of HashGrid Encoder

We chose the HashGrid encoder [24] because of two properties: 1. It fits the concept of *evolution* in our SDF Module 2. It helps to learn the movements of surface points fed into the Rendering Module.

To explain further these two benefits, please note that based on the HashGrid's resolution (in each level), two

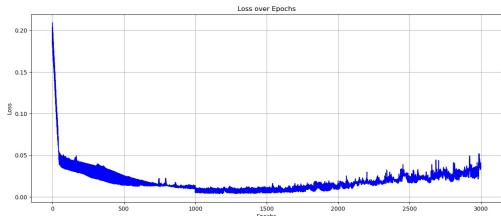


Figure 12. Loss plot (photometric loss) of NIE trained on 2 frames of the “*dynamic chair deformation*” scene. This loss plot shows the loss status of the last frame in each epoch ($t = 1$). It is evident in the loss plot that the model is struggling to fit both time frames well enough, and in final epochs, as the model fits to a mesh more similar to $t = 0$, the photometric loss for $t = 1$ is increasing.

points $\mathbf{x}_1, \mathbf{x}_2$ are encoded similarly to each other if they are close enough in the world coordinate. In mathematical form, it can be written as:

$$\|E(\mathbf{x}_1) - E(\mathbf{x}_2)\|_2 < \epsilon_1 \iff \|\mathbf{x}_1 - \mathbf{x}_2\|_2 < \epsilon_2 \quad (18)$$

7.2. SDF Module

The SDF Module consists of a HashGrid Encoder [24] with minimum resolution $N_{min} = 64$, number of Hash Table entries equal to $\log_2 T = 19$ and $F = 8$ features per Hash-Grid vertices and $L = 16$ different levels. Please note that these properties (most notably, F and L) can be decreased for speed. The per-level scale is set to $s = 1.5$, and *linear* interpolation is chosen as the interpolation method. The output of this coordinate encoder (with $F \times L$ dimension) is concatenated with the positional encoding of time $\gamma(t)$; $t \in [0, 1]$ with 64 as the number of output frequencies. The concatenated vector is fed into the SDF Head MLP. The SDF Head MLP consists of 4 hidden layers. The number of neurons per layer is set to 128. Increasing this number to higher values (such as 256) has increased the model’s outcome in the sense of quality measures in our experiments but effectively decreases the speed of training and inference. The hyperbolic tangent (*Tanh*) is chosen here as the activation function due to its nice property of *many times differentiability*, which is helpful for SDF prediction. Please refer to Fig. 2 for detailed explanations of the SDF module.

An explanation about why we scale t to be in $[0, 1]$ is because in scenes where we use $\frac{\partial f_\theta}{\partial t}$, we need the time steps to be close to each other. Here, essentially, $t = 0$ means the start of the animation, $t = 1$ means the end of the animation, and $t \in (0, 1)$ will be some interpolation between these start and end points (which we are directly supervising in some time steps, e.g., $t = 0.5$).

7.3. Rendering Module

For the Rendering Module, we also use a second separate HashGrid encoder to encode the coordinates of input points.

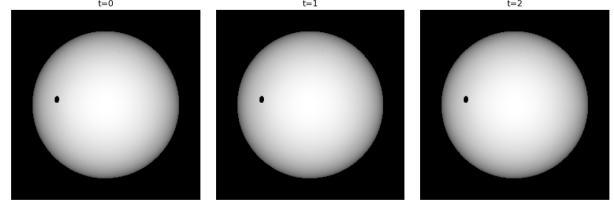


Figure 13. NIE [22] model initialized as all spheres and rendered in 3 sample time steps.

Note that the input of this module is the estimated surface points from the SDF Module. The details of the HashGrid encoder are the same as the encoder explained in Sec. 7.2. Number of output frequencies for encoding time ($\gamma(t)$) is 64. The Rendering Head MLP comprises a **shared backbone** and **separate heads**. The shared backbone is an MLP consisting of 3 hidden layer and 128 neurons per layer. The output of this shared backbone is a 64 dimensional feature vector. This $f \in \mathcal{R}^{64}$ feature vector is fed to 3 separate heads to predict *Spherical Harmonics, Rotation, and Opacity*. We experimented regressing *Scaling* for each splat but understood that setting the scale of each splat is equal to $\frac{1}{d_{voxel}}$ where d_{voxel} is the voxel size in the sampling 3D grid is more efficient.

8. Comparison with the Baseline

We compare our model (N4DE) with the baseline (NIE [22]). Since NIE is aimed to reconstruct 3D scenes via the evolution method proposed in the paper and is focused not on dynamic scenes but on static scenes, we make some changes to the code to support multi-frame scenes. We show that in many of our dynamic scenes, the outcome of our model is comparably higher quality and more aligned with the correct mesh in that time step. Also, in many cases, the NIE method fails and crashes in the training iterations. We inspected the model and understood that in some iterations, the NIE predicts all SDF values to be negative, and thus, the marching cubes step [20] fails, and the whole training pipeline crashes.

8.1. Extending NIE for multi-frame scenes

We extend the NIE [22] approach to fit multi-frame scenes by inputting a 4-dimensional input instead of the previous 3-dimensional input. For this sake, we simply concatenate the time t to the coordinate vector \mathbf{x} and then pass it to the SIREN [46] network proposed in the NIE paper. Please note that we also used the same initialization as in the main NIE paper. The initialized network’s predictions in different time steps are shown in Fig. 13.

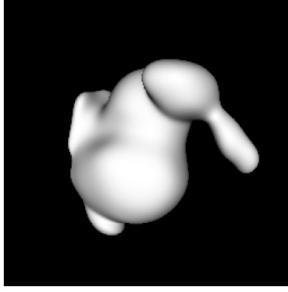


Figure 14. NIE [22] model trained on single frame static Stanford Bunny, by giving 4D inputs with $t = 0$ concatenated. (Left) Predicted (Right) Ground Truth.

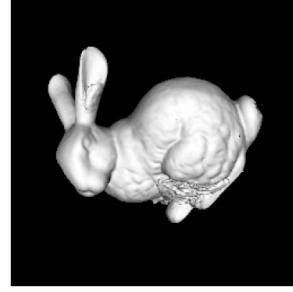
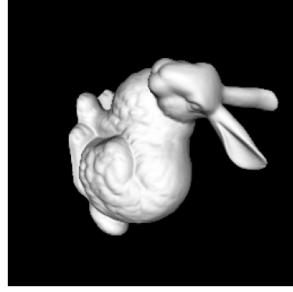


Figure 15. NIE [22] model trained on single frame static Stanford Bunny, by giving 3D inputs only. (Left) Predicted (Right) Ground Truth.

8.2. Dynamic Scenes

We compare outcomes of our model (N4DE), which are also present in Tab. 1 with training the NIE model on the same scenes. The results are available in 3. As you can see, many of the experiments failed; thus, the table’s related rows are filled with F . We inspected the training process and understood that NIE predicts *all-negative values* for SDF, and thus, the Marching Cubes [20] process present in the training pipeline of NIE fails. This leads to **crashing the whole training pipeline**. Please note that we experimented with the initial learning rate proposed in the main paper ($lr = 0.000002$) and also with a lower learning rate. In both cases, the failing scenes still failed and crashed in the training pipeline due to the said reason.

8.3. Static Scenes

Adding the 4th dimension (time) as input to the SIREN [46] model proposed in the NIE’s approach affects the output quality significantly. Aside from the dynamic scenes, even in static scenes like the Stanford bunny, we noticed that training and inferring the NIE model with concatenating $t = 0$ simply causes to lose a lot of fine details on the output mesh (see Fig. 14).

However, doing the same scene without concatenating time $t = 0$ and inputting only the 3D input coordinates, we get the reconstruction with good enough details and an acceptable result (See Fig. 15).

This essentially shows that the NIE [22] is **not capable of handling 4D inputs even in static scenes, without a significant drop in the final reconstruction’s quality.**

8.4. Changing NIE to accept time as 4th input

We develop our experiments more by concatenating $\gamma(t)$ instead of t (where γ stands for positional encoding). We try this approach to see if it fixes the problem of “*adding a 4th dimension to the inputs to NIE decreases the quality in static scenes significantly and crashes the training process in dynamic scenes*”. It fixes this problem, at least in the case

that static scenes concatenate $t = 0$ as the 4th dimension to the input, and the model can reconstruct the mesh with an acceptable quality similar to the original architecture. The table 3 is filled with this kind of customization on NIE’s [22] architecture. Even with this kind of customization on NIE’s [22] architecture, most of the dynamic scenes, specifically the ones that have significant topological changes between their two consecutive frames (like SMPL [19] scenes and the breaking sphere scene) fail due to “*all negative SDF value predictions and marching cubes failure*” (refer to Tab. 3). The scenes that did not fail (like the deforming bunny, can be seen in Fig. 21), the NIE [22] model seems to overfit on one frame only or learn canonical representation between all frames instead of learning the animation itself.

On the other hand, our method (N4DE) represents each time step distinctly from the other one while also being able to fill in the time gaps (to unseen frames) and interpolate between time steps. Refer to Fig. 21 for a sample qualitative comparison.

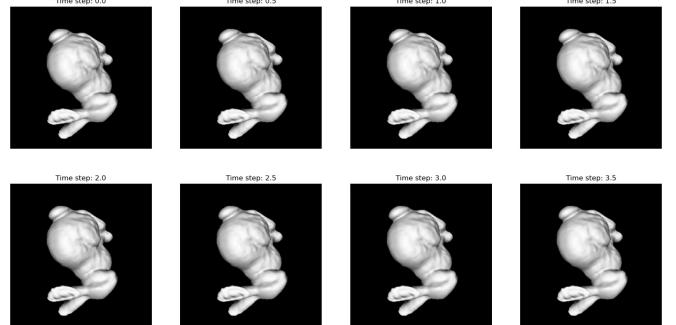


Figure 16. NIE [22] model trained on multi-frame deforming bunny scene. As you can see, the model does not learn the correct animation conditioned on time and instead overfits to be like the first frame ($t = 0$).

9. N4DE vs NIE

In this section, we mention the main benefits of using N4DE instead of just customizing NIE to accept time as 4th dimension and overfit on each frame:

1. It is really hard in NIE to find optimal hyper-parameters (because of using SIREN) for each scene. On the other hand, N4DE does not require such different configurations for each scene.
2. NIE in the best case (concatenating $\gamma(t)$ to the \mathbf{x} and inputting the resulting $3 + 64$ dimensional vector to the MLP) still is incapable of representing the deformation **animation** and it just learns a mesh representation that is very similar to the ground truth in $t = 0$ and looks like an average along time.
3. NIE - even in static scenes - cannot learn a good, detailed, meaningful representation based on RGB images. It only works with high quality if we supervise it with silhouette-like gray-scale images (Images rendered with Phong Shader and without texture).
4. N4DE has an obvious superiority compared to NIE regarding training time and inference time (refer to Tab. 3 for some comparisons). This is because HashGrid and a much smaller MLP are used as the SDF head.

To further investigate the outcomes of NIE, we've plotted the "*Deformable Breaking Sphere*" with 5 frames. After 2303 epochs, this scene failed during the "*all negative SDF prediction*" issue. You can see the model's predictions before this failure happens in Fig. 17.

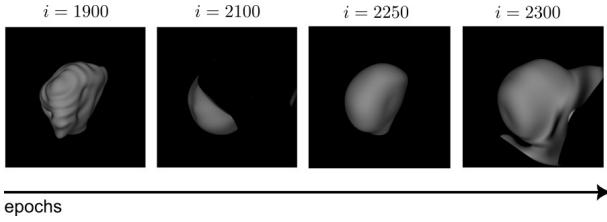


Figure 17. NIE [22] model trained on five frames of the "*Deformable Breaking Sphere*" scene. It failed after 2303 epochs. The last frame ($t = 5$) evolution is plotted in this figure during epochs. Compare it with our model's outcomes on the same scene in Fig. 6

Another critical factor in our method is **training speed**. Learning animation is time-consuming, and if we want high-quality reconstructions, we may have to sacrifice speed sometimes. However, because of using the HashGrid [24] encoder, we can have a much smaller MLP as the SDF head and, thus, decrease the training time significantly. In Tab. 3, you can see the speed comparisons between our method and NIE. Please note that these time estimates are calculated based on averaging the number of seconds that took up to a specific epoch (i) and dividing by the number of epochs. The total sum of the seconds taken up to a specific epoch is extracted from our tensorboard [1] logs for

better estimates.

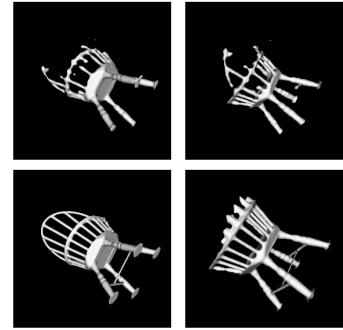


Figure 18. NIE model's reconstruction of the "*Deformable chair*" scene. (Top) NIE reconstruction in $t = 0$ and $t = 1$. (Bottom) Ground truth in $t = 0$ and $t = 1$.

The most important factor our model aims at is the ability to **reconstruct deformation animations**. NIE [22] model in its pure form is incapable of doing this and fails in many animated scenes if we change the 3D input ($\mathbf{x} = (x, y, z)$) to the 4D input ((x, y, z, t)). Even with some customizations to increase the input dimensionality and help the learning process of NIE, with inputting $(x, y, z, \gamma(t))$ where γ stands for positional encoding, the model still cannot reconstruct and distinguish different time steps of the animation. Aside from the reconstruction output (which is so similar to the ground truth in $t = 0$), the problem of "*all negative SDF predictions*" and crashing the training pipeline still happens in some of the deformable scenes (refer to Tab. 3 and Fig. 17 and Fig. 20).

Another difference between N4DE and NIE [22] is the ability to keep the reconstruction quality by changing the ground truth images. The base ground truth images used in NIE [22] are the rendered different views from the mesh by Phong Shader method and using nvdiffrast [16] library. While this kind of supervision is useful, it is also a strong supervision method (in comparison with RGB supervision). So, we expect our model to keep its reconstruction quality on RGB (e.g., textured) ground truth images (rendered from ground truth meshes). This fact is actual for N4DE, and can be seen in Fig. 7 and Fig. 19. On the other hand, evaluating NIE [22] on a static scene (only in $t = 0$) but with RGB supervision makes the reconstruction's quality degrade (Please refer to Fig. 19).

One of the most important outcomes of our model (N4DE) is the fact that we are not just overfitting to each individual frame, but we are, in fact, learning the deformation animation by the evolution method we use from [22]. This outcome can be seen when we infer our model on time steps that the model is not supervised on (refer to Fig. 1 and Fig. 6 and Fig. 8). This phenomenon and the effect of our time regularizer ($\frac{\partial f_\theta(\cdot, t)}{\partial t}$) is explained more in Sec. 4.4. The

Table 3. Comparing the Baseline [22] with our approach in deformable scenes.

	Dynamic Bunny Deformation						Dynamic Breaking Sphere						Dynamic SMPL [19] Scene #1						Dynamic Deformable Chair						Dynamic Deforming Status					
	MSE	PSNR	SSIM	LPIPS	Chamfer dist.	Avg. time per epoch	MSE	PSNR	SSIM	LPIPS	Chamfer dist.	Avg. time per epoch	MSE	PSNR	SSIM	LPIPS	Chamfer dist.	Avg. time per epoch	MSE	PSNR	SSIM	LPIPS	Chamfer dist.	Avg. time per epoch	MSE	PSNR	SSIM	LPIPS	Chamfer dist.	Avg. time per epoch
N4DE(ours)	0.0053	24.5015	0.8630	0.1416	0.0127	3.8206s	0.0033	26.5121	0.8491	0.1659	0.0241	7.24s	0.0030	25.4215	0.9274	0.1000	0.0050	5.9141s	0.0066	22.7789	0.8326	0.1867	0.0089	2.8807s	0.0042	24.2799	0.8950	0.1027	0.0029	3.0294s
NIE* [23]	0.0131	22.2669	0.8273	0.1149	0.0411	8.4096s	F	F	F	F	F	12.03s	F	F	F	F	F	8.9504s	0.0127	20.1629	0.8196	0.1599	0.0142	5.47s	0.0055	25.4908	0.9105	0.0776	0.0032	5.4291s

*F indicates that the training failed because of *all negative* values of SDF prediction (and thus, marching cubes [20] failure). It essentially breaks the training process.

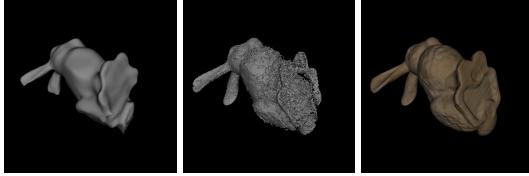


Figure 19. (Left) NIE’s reconstruction when supervised via RGB textured image. (Middle) N4DE’s reconstruction when supervised via RGB textured image. (Right) a sample view of the ground truth textured mesh.

interesting fact is that in our experiments (like “*Deformable Breaking Sphere*”), we found out that even if we do not use time regularization, we still get meaningful and good mesh estimates in the unseen time steps. If we want smoother, noiseless, and better quality estimates in unseen time steps, we can tune the multiplier for time regularization in the experiments. Notice that calculating and back-propagating for this regularizer adds some overhead and costs as extra time in the “Avg. training time”. It’s a trade-off of speed vs quality again. Even if with some more customizations and changes in the NIE’s architecture (like changing the inputs more and moving them to higher dimensions, changing the order of training and randomizing the time step selection (t) in each epoch, etc.) in the best case it will be capable of overfitting on each frame independently. It is because nothing creates a correct relationship between each frame and the next one (t_i and t_{i+1}) in their loss function or the architectural choice. So, eventually, it will still be incapable of learning the *animation* itself (and estimating a correct mesh in the unseen time steps).

This fact can also be seen in our current experiments. If you refer to Fig. 20 and Fig. 12, you can see that whenever model tries to fit perfectly on one time-step ($t = i$), the other time steps’ photometric loss goes up and has an increasing trend. In scenes with little changes between frames (like 2 frame experiment on “*Chair deformation*”), it will result in a reconstruction that is so similar to the first frame ($t = 0$) and has some details of the $t = 1$ (See Fig. 18). But in longer scenes like “*Deformable Breaking Sphere*,” the reconstruction is not exactly similar to the ground truth in $t = 0$, but it looks like an average along time (See $i = 1900$ in Fig. 17).

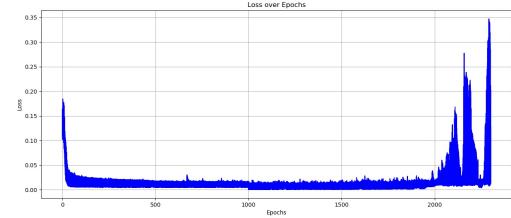


Figure 20. Loss plot (photometric loss) of NIE trained on 5 frames of the “deformable breaking sphere” scene. This loss plot shows the loss status of the last frame in each epoch ($t = 4$). This experiment failed after 2303 epochs. This is also evident in the loss plot. As you can see, we have sudden peaks in certain epochs, the most significant ones in the last epochs (near 2300).

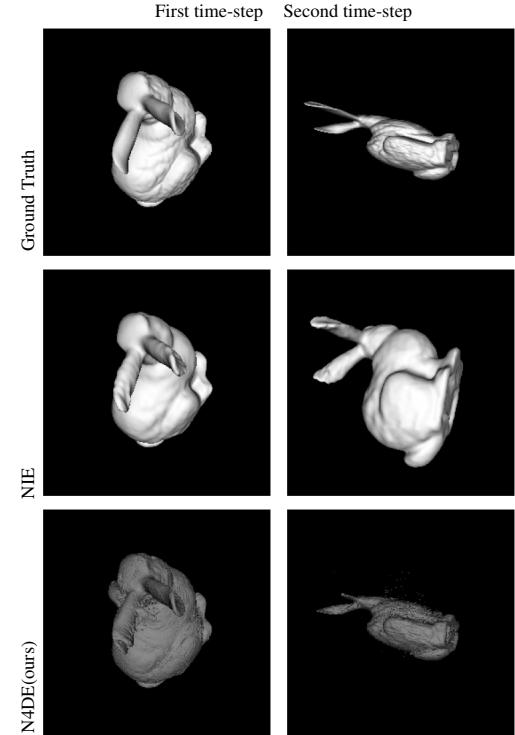


Figure 21. Visualized first two time-steps of *Deformable Bunny* scene and its estimates via NIE and N4DE. As can be seen, NIE did not learn the animation properly and estimates the same mesh in different time steps, but our estimates are near the GT while distinguishing between each animation frame. Notice the Ground Truth images to observe how sudden and large the scaling deformations are over time.