SAARLAND UNIVERSITY

Faculty of Mathematics and Computer Science
Department of Computer Science
Master's Thesis

Monocular Non-Rigid 4D Reconstruction using a Deformable Implicit Scene Model

submitted by
Erik C.M. Johnson, Saarland University
Saarbrücken, Germany

on
December 15, 2021

supervised by
Dr. Vladislav Golyanik, Max Planck Institute for Informatics
Saarbrücken, Germany

reviewed by
Prof. Dr. Christian Theobalt, Max Planck Institute for Informatics
Saarbrücken, Germany
&
Dr. Vladislav Golyanik, Max Planck Institute for Informatics
Saarbrücken, Germany

Declarations

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any media or materials other than the ones referred to in this thesis.

Erik C.M. Johnson	Location & Date

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Erik C.M. Johnson	Location & Date

Acknowledgements

I would like to thank Dr. Vladislav Golyanik for his invaluable supervision throughout the development of this thesis. Also, to Prof. Dr. Christian Theobalt for reviewing this thesis and for his dedicated curation of the department in which it was produced. I am proud to have been included in such an excellent group of researchers as D6 of the Max-Planck-Institut für Informatik.

Among this group, I must especially thank Dr. Marc Habermann, Dr. Lingjie Liu, and Soshi Shimada who all provided expert guidance throughout this project and proofread many versions of the same content for this thesis. Also, to Edgar Tretschk and Dr. Mohamed Elgharib for their contributions in shaping this work.

On my academic journey I have received encouragement from some truly great professors and I would be remiss not to acknowledge their impact on me: Prof. Dr. Alan Steele, Prof. Dr. Joachim Weickert, and Prof. Dr. Calvin Plett.

I owe my parents too much to state for their unwavering support. Also, my friends outside of D6 have been there for me throughout my degree: Robin, Pia, Maytham, BDN, Jeffrey, Irusha, Smokey, and Samir. Whether cooking in a proper kitchen, giving me an excuse to do some traveling in Germany, going for breakfast, playing video games, or just chatting, the past two years would have looked entirely different without them. Particular note goes to Samir, who took time from his ludicrously busy schedule to read an early draft of this thesis.

This thesis uses the UMTL LaTeX template from the DFKI.

Abstract

The reconstruction of geometry for non-rigidly deforming objects from a single view is a very challenging problem. Many previous methods have attempted to solve this problem; however, none have achieved sufficient robustness for industrial application. A novel approach to this problem is required to advance the state-of-the-art.

Recent advances in neural Implicit Scene Representations (ISRs) have demonstrated tremendous promise in many applications requiring learning scenes from images. While some ISR methods have already shown promise for reconstructing the geometry of static scenes from multi-view data, none have yet demonstrated the capability of non-rigid 4D reconstruction from a single monocular view.

A new method is proposed, called Unbiased 4D or Ub4D, that uses a neural deformation model to bend rays into a canonical space for reconstructing the geometry of deforming objects from a monocular sequence of images. Given the ability of other methods to acquire coarse geometries, we also formulate an optional loss that uses 3D correspondences of the object as an additional prior.

Ub4D demonstrates state-of-the-art results when compared to other methods solving the problem of monocular 4D reconstruction for non-rigidly deforming objects. We examine Ub4D with ablation studies, present extensive results, and investigate semantic meaning in the learned latent codes. Applying ISRs to such a challenging problem shows significant potential for future improvement and motivates further investigation.

Contents

List of Figures

List of Tables

List of Abbreviations

ARAP As-Rigid-As-Possible.

CD Chamfer Distance.

CDF Cumulative Distribution Function.

CNN Convolutional Neural Network.

CT Computed Tomography.

FMM Fast Multipole Method.

FNO Fourier Neural Operator.

FST Fourier Slice Theorem.

GT Ground Truth.

HD Hausdorff Distance.

ICP Iterative Closest Point.

ISR Implicit Scene Representation.

LSTM Long Short-Term Memory.

MFOF Multi-Frame Optical Flow.

MLP Multi-Layer Perceptron.

NN Neural Network.

NRSfM Non-Rigid Structure-from-Motion.

PC Principal Component.

PCA Principal Component Analysis.

PDE Partial Differential Equation.

PDF Probability Density Function.

PE Positional Encoding.

SDF Signed Distance Function.

SfM Structure from Motion.

List of Symbols

$\alpha_i^{(z)}$ Discrete opacity for a sample z along a bent ray $\tilde{\boldsymbol{r}}_i$.

$\boldsymbol{b}_i$ Bending network function $\boldsymbol{b}_i : \mathbb{R}^3 \to \mathbb{R}^3$ projecting points from frame i space to canonical space by addition: $\boldsymbol{x}_c = \boldsymbol{x}_i + \boldsymbol{b}_i(\boldsymbol{x}_i)$. Short-hand for conditional function taking a frame-specific latent code $\boldsymbol{l}_i$ (i.e. $\boldsymbol{b}_i(\cdot) = \boldsymbol{b}(\cdot; \boldsymbol{l}_i)$).

$\boldsymbol{c}$ Color network function $\boldsymbol{c} : (\mathbb{R}^3, \mathbb{R}^3) \to \mathbb{R}^3$ returning a color for a point in space and a direction.

$\boldsymbol{d}$ Direction $\boldsymbol{d} \in \mathbb{R}^3$ of a straight ray $\boldsymbol{r}$.

f Signed Distance Function (SDF) $f : \mathbb{R}^3 \to \mathbb{R}$ mapping points in canonical space to their SDF values.

γ Weight for a loss function. One of γ_{SEG}, γ_{EIK}, γ_{NBR}, γ_{DIV}, or γ_{FLO}. Defined relative to the value of L_{COL}.

$\hat{I}$ Color rendering functional returning the color for a provided bent ray $\tilde{\boldsymbol{r}}_i$.

$I^{(p)}$ Ground truth color for a specific pixel p.

L Loss function. One of L_{COL}, L_{SEG}, L_{EIK}, L_{NBR}, L_{DIV}, or L_{FLO}.

$\boldsymbol{l}_i$ Latent code for frame i. This is provided to the bending network to compute the frame's deformation (i,e. $\boldsymbol{b}_i(\cdot) = \boldsymbol{b}(\cdot; \boldsymbol{l}_i)$). It is optimized during training.

λ_1 Scale parameter used in a kernel of the form w_λ affecting the spatial weighting of the scene flow.

λ_2 Scale parameter used in a kernel of the form w_λ affecting the falloff of the scene flow extrapolation.

$\boldsymbol{m}_{i \to j}$ Scene flow function $\boldsymbol{m}_{i \to j} : \mathbb{R}^3 \to \mathbb{R}^3$ returning an estimate of the scene flow for a given position in frame i space. Projects a point in frame i space into frame j space by addition: $\boldsymbol{x}_j = \boldsymbol{x}_i + \boldsymbol{m}_{i \to j}(\boldsymbol{x}_i)$.

$\mathcal{N}$ Neighbour function. Returns all neighbours of its input as a set.

n_s Number of samples.

$\boldsymbol{o}$ Origin $\boldsymbol{o} \in \mathbb{R}^3$ of a straight ray $\boldsymbol{r}$.

ω Weight functional returning the occlusion-aware weight for a distance t along a bent ray $\tilde{\boldsymbol{r}}_i$.

$\omega_i^{(z)}$ Weight for a sample z along a bent ray $\tilde{\boldsymbol{r}}_i$.

$\mathcal{P}$ Batch set of pixels in the input image.

p Pixel in the input image.

Φ_s Logistic Cumulative Distribution Function (CDF) parameterized by s, i.e. $\Phi_s(x) = (1 + e^{-x/s})^{-1}$.

ϕ_s Logistic Probability Density Function (PDF) parameterized by s, i.e. $\phi_s(x) = \frac{e^{-x/s}}{s(1+e^{-x/s})^2}$. Note that $\phi_s(x) = \frac{\mathrm{d}}{\mathrm{d}x}\Phi_s(x)$.

$\boldsymbol{r}$ Straight ray $\boldsymbol{r} : \mathbb{R} \to \mathbb{R}^3$ parameterized per-pixel by an origin $\boldsymbol{o} \in \mathbb{R}^3$ and a direction $\boldsymbol{d} \in \mathbb{R}^3$. Takes as argument the distance along the ray t: $\boldsymbol{r}(t) = \boldsymbol{o} + t\,\boldsymbol{d}$.

$\boldsymbol{r}^{(p)}$ Straight ray $\boldsymbol{r}$ for a specifically identified pixel p.

$\tilde{\boldsymbol{r}}_i$ Bent ray $\tilde{\boldsymbol{r}}_i : \mathbb{R} \to \mathbb{R}^3$ of a straight ray $\boldsymbol{r}$ for a given frame i by applying the bending network function $\boldsymbol{b}_i$ as $\tilde{\boldsymbol{r}}_i(t) = \boldsymbol{r}(t) + \boldsymbol{b}_i(\boldsymbol{r}(t))$.

$\tilde{\boldsymbol{r}}_i^{(p)}$ Bent ray $\boldsymbol{r}$ for a specifically identified pixel p: $\tilde{\boldsymbol{r}}_i^{(p)}(t) = \boldsymbol{r}^{(p)}(t) + \boldsymbol{b}_i\big(\boldsymbol{r}^{(p)}(t)\big)$.

ρ Density function $\rho : \mathbb{R}^3 \to \mathbb{R}$ mapping a canonical space point into a volumetric density value.

s Scale parameter for logistic CDF Φ_s and PDF ϕ_s.

$\hat{S}$ Accumulated density (segmentation) functional returning the total density encountered along a bent ray $\tilde{\boldsymbol{r}}_i$.

$S^{(p)}$ Ground truth segmentation (accumulated density) for a specific pixel p.

T Transmittance functional returning the probability that a bent ray $\tilde{\boldsymbol{r}}_i$ reaches a given distance t.

$T_i^{(z)}$ Transmittance for a sample z along a bent ray $\tilde{\boldsymbol{r}}_i$.

t Distance $t \in \mathbb{R}$ along a ray.

$\boldsymbol{v}_i^{(k)}$ Mesh vertex k in frame i for a corresponding mesh sequence. For example, vertex $\boldsymbol{v}_i^{(k)}$ from frame i is the same vertex as $\boldsymbol{v}_j^{(k)}$ in frame j.

w_λ Kernel weighting function $w_\lambda : \mathbb{R} \to \mathbb{R}$ of the form: $w_\lambda(x) = e^{-\lambda x^2}$ with parameter λ defining the scale of the kernel.

$\mathcal{X}_i$ Set of points sampled in frame i.

z Discrete sample $z \in \mathbb{Z}$ along a ray corresponding to a distance value of $t^{(z)}$.

Chapter 1
Introduction

The ability for anyone to capture the world with a consumer device has created dramatically new opportunities for content creation. Consumer video capture has altered how many industries operate and created entirely new industries. This ease of capturing high quality video allows for new methods of content creation. One such evolution of media capture will be towards extracting animated 3D models from video.

Already the ability to extract static 3D models from sets of images using photogrammetry has revolutionized the creation of assets for digital productions. Figure 1.1 shows an example of such an asset. In contrast, creating animated 3D assets requires time-intensive work, expensive multi-camera setups, or custom object-specific algorithms. Extending the ability to scan objects to single cameras for generic deforming objects would enable creators to build completely new experiences. This problem of monocular 4D reconstruction for non-rigidly deforming objects is the focus of this work.

Figure 1.1: Photogrammetry asset in a digital scene from an Epic press release [87]. Quixel is a company that uses photogrammetry to scan real-world objects for use in digital productions.

The solution to this problem is critical to the future of XR technologies. A positive future for these technologies relies on lowering the barriers to creation. The ability to produce digital replicas of real-world objects from the ubiquitous smartphone camera would allow individuals more freedom in their creations. Solving this problem would also open up the possibility of extracting animated 3D models from existing footage. For example, giving anyone access to all filmed animals and, moreover, all movements of those animals captured on film. Video is the great library of everything and it can be made available as a catalogue of digital assets for creating currently unimaginable immersive experiences.

1.1 Challenges

The problem of monocular 4D reconstruction is fundamentally challenging. The reason for this is the ill-posed nature of the problem: a single monocular view does not provide sufficient information to uniquely solve for the 3D object surface. Even knowing the camera parameters and considering a perfectly rigid object, there is an infinite space of object scales and depths creating the same image observations. The problem becomes even more ill-posed when allowing for non-rigid deformations of the object. Such a case is visualized in Figure 1.2 for multiple depths and surface deformations, all of which project to the same 2D image plane pixels.

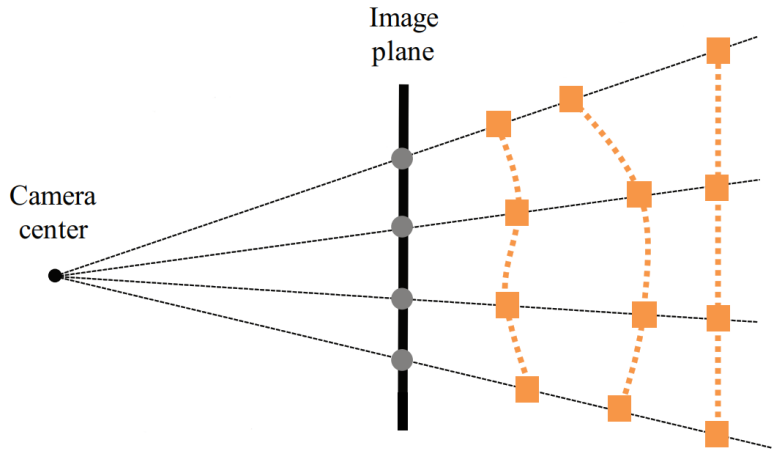


Figure 1.2: Illustration of monocular depth ambiguity modified from Jain et al. [28]. Note that the multiple different depths and surface deformations all project from 3D to the same 2D pixels on the image plane.

Another challenge encountered in many applications leveraging Neural Networks (NNs) is hyperparameter tuning. A hyperparameter is an untrained parameter that defines the construction or operation of the NN architecture, referred to as such to distinguish them from those that are trained, e.g. the weights and biases. Despite demonstrating excellent results, NN approaches commonly employ arbitrarily-sized network layers and latent codes without being able to provide theoretical justification for those decisions. Frequently, hyperparameters are set according to empirical observations. Hyperparameter tuning represents its own challenging area of research, particularly for complex models requiring significant training effort.

1.2 Overview

We examine existing approaches to the problem of monocular 4D reconstruction in Chapter 2. While some works show good results for humanoid characters or planar objects, they do so by imposing strict priors on the class of image sequences to which they can be applied. A recent explosion of works in the domain of neural volumetric rendering from Implicit Scene Representations (ISRs) is reviewed in Chapter 3. This new class of method has already been demonstrated to be capable of producing high quality static 3D models in competition with traditional photogrammetry methods like Structure from Motion (SfM). In this work, we explore the possibility of extending this capability to non-rigidly deforming objects using only a single monocular view.

Chapter 4 proposes our novel approach to the problem of monocular 4D reconstruction through the use of a deforming ISR. Our method, which we call Unbiased 4D or Ub4D, uses a neural deformation model to bend rays into a canonical space. In this canonical space, the ray is rendered from the ISR which is supervised by the ground truth pixel color in order to learn the canonical surface and the per-frame deformation. We provide extensive investigation of this approach in Chapter 5 for a wide range of object types represented in a newly created dataset. Ub4D achieves state-of-the-art results and we examine the correctness of our decisions with an ablation study. Further, we investigate semantic meaning in the latent codes of our neural deformation model.

Despite achieving state-of-the-art results with our novel approach, there still remains extensive possibilities for future improvement. Chapter 6 details current limitations and future work. The application of ISRs to the problem of monocular 4D reconstruction for non-rigidly deforming objects shows promise that will continue beyond this work.

Chapter 2
Related Work

Monocular 4D reconstruction of non-rigidly deforming objects is an active research area with multiple approaches to solving the problem. We focus our attention in this section entirely on monocular methods. While our method, Ub4D, builds on ideas from multi-view approaches and Implicit Scene Representations (ISRs), we discuss these in Chapter 3 as they do not tackle the same problem that we do.

The other methods solving the problem of monocular 4D reconstruction for non-rigid objects are Non-Rigid Structure-from-Motion (NRSfM), template-based tracking, and some neural surface extraction methods. Despite solving the same problem, they differ significantly from Ub4D and these differences will be highlighted. We first discuss NRSfM in Section 2.1, then template-based tracking in Section 2.2, and finally neural surface extraction in Section 2.3.

2.1 Non-Rigid Structure from Motion

Non-Rigid Structure-from-Motion (NRSfM) is a classical factorization approach to jointly recovering deforming object shape and camera motion from monocular image sequences [9, 82]. It takes as input 2D image-space keypoints tracked over the sequence that are typically centered in each frame by removing the average [2, 9, 82]. This is referred to as the measurement matrix $\boldsymbol{W} \in \mathbb{R}^{2F \times P}$ for F frames tracking P keypoints. This measurement matrix is then factored into two components: $\boldsymbol{S} \in \mathbb{R}^{3F \times P}$ containing the 3D position for each keypoint in each frame and $\boldsymbol{R} \in \mathbb{R}^{2F \times 3F}$ containing the camera rotation and projecting the 3D points to 2D [2, 9, 75, 82]. This approach is summarized in Figure 2.1.

This approach was described for rigid objects under orthographic projection in Tomasi and Kanade [82] and then subsequently extended to handle non-rigidly deforming objects in Bregler et al. [9]. When allowing for non-rigid deformations, the problem formulation becomes highly ill-posed and many works introduce constraints on the allowed space of deformations [9, 35, 68, 84]. Another trend in NRSfM works is moving towards the use of dense Multi-Frame Optical Flow (MFOF) [20, 61] instead of sparse keypoint tracks [2, 19, 75]. Recently, methods leveraging advances in deep neural networks have

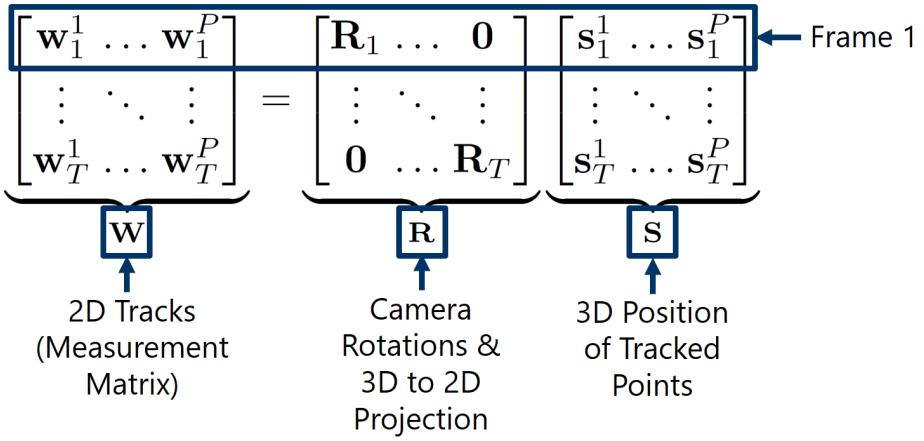


Figure 2.1: NRSfM factorization approach to monocular 4D reconstruction. This approach also seeks to recover the camera motion. Each matrix is depicted as a block matrix showing the relevent individual elements: $\mathbf{w}_i^j$, $\mathbf{R}_i$, and $\mathbf{s}_i^j$, for frame i and tracked point j. Base image from [75], annotations ours.

become more common [32, 75, 90].

From these neural approaches to NRSfM, Sidhu et al. [75] specifically employs an auto-decoder to deform a mean shape which is trained against a dense MFOF measurement matrix, while also optimizing the camera rotations. They call their method Neural NRSfM or N-NRSfM. This work demonstrates state-of-the-art results for NRSfM approaches and we show some of their qualitative results in Figure 2.2. We compare against the results of their method on our introduced datasets in Section 5.4.

Figure 2.2: Qualitative results of Sidhu et al. [75]. Note how little the camera moves and that the reconstructions are planar surfaces rather than water-tight meshes.

While NRSfM approaches have a significant heritage and solid theoretical framework, they suffer from drawbacks that make this approach challenging to apply for real-world captures. The first being that the formulation takes 2D tracks over the entire sequence

as input which assumes that the same points are visible in all frames [9, 82]. While some works discuss the ability to handle occlusions [19, 75, 84] and some simply set occluded tracks to zero [32], none provide real-world results with significant camera motion showing multiple sides of an object. Additionally, computing 2D tracks is a challenging problem on its own [69, 79]. Another issue with NRSfM is that its reconstructions are planar surfaces in practice whereas water-tight meshes are a more desirable output for many object classes. Finally, the ability of NRSfM to handle large or complex deformations can be limited by the constraints placed on the allowed deformations [32].

In contrast to NRSfM approaches, our method, Ub4D, takes as input monocular RGB images and assumes known camera parameters. This allows Ub4D to handle scenes with large camera motion that creates significant occlusions in the 2D tracks. Additionally using an analysis-by-synthesis approach gives additional robustness to noise when compared to a matrix factorization. One benefit of NRSfM relative to Ub4D, is that it provides 3D correspondences which our method as presented in Chapter 4 does not. We discuss the potential extraction of 3D correspondences from Ub4D in Section 6.2.

2.2 Template-Based Tracking

Another approach to monocular 4D reconstruction is using a known template and then tracking it over the image sequence [67, 99]. This approach is frequently used with the additional prior that the object of interest is a human character [23, 24, 94]. These methods take as input a monocular RGB sequence and an object template. Then they formulate an energy E that is minimized for each frame by optimizing the object's shape $\boldsymbol{S}$, rotation $\boldsymbol{R}$, and translation $\boldsymbol{t}$ [23, 24, 67, 94, 99]. This energy formulation captures the error in the projection of the deformed template onto the image, as well as various additional priors, e.g. spatial smoothness, temporal smoothness, and As-Rigid-As-Possible (ARAP) [78] deformations. This approach is summarized in Figure 2.3.

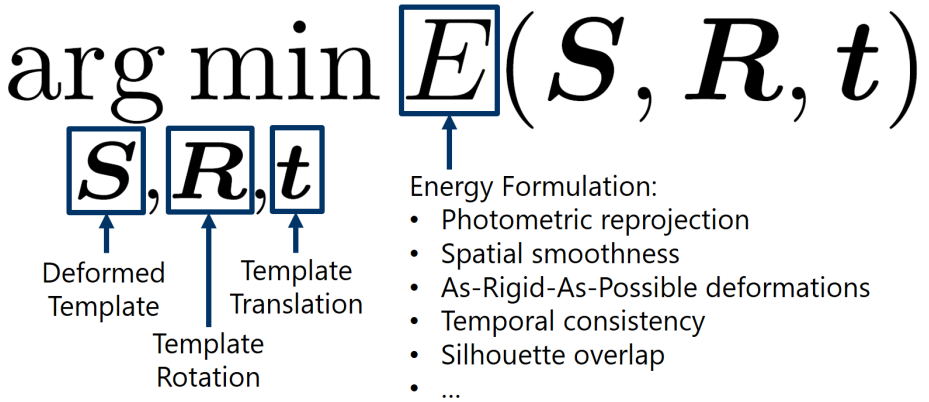


Figure 2.3: Template-based tracking approach to monocular 4D reconstruction. This approach deforms a template over the RGB sequence to explain the image observations and minimize constraining regularizers.

Methods assuming humanoid objects typically use this additional prior to pose a rigged template using optimized skeleton parameters, in addition to applying non-rigid deformations [23, 24, 94]. While Habermann et al. (2019) [23] and Xu et al. [94] are purely

optimization methods without a training phase, Habermann et al. (2020) [24] requires per-character multi-view training data in addition to a template. While these humanoid-specific approaches demonstrate excellent results for humans, the path to fully generalizing these methods is infeasible due to how much they rely on the humanoid prior. On the other hand, Salzmann et al. [67] and Yu et al. [99] are both object class independent methods.

Specifically, Yu et al. [99] use a combination of a reprojection energy and constrain the solution with spatial, temporal, and ARAP [78] deformation regularization. They also initialize the state for each frame using the previous frame. Their method is called Direct, Dense, and Deformable; or DDD. This work demonstrates state-of-the-art template-based tracking results for general objects and we show some of their qualitative results in Figure 2.4. We compare against the results of their method on our introduced datasets in Section 5.4.

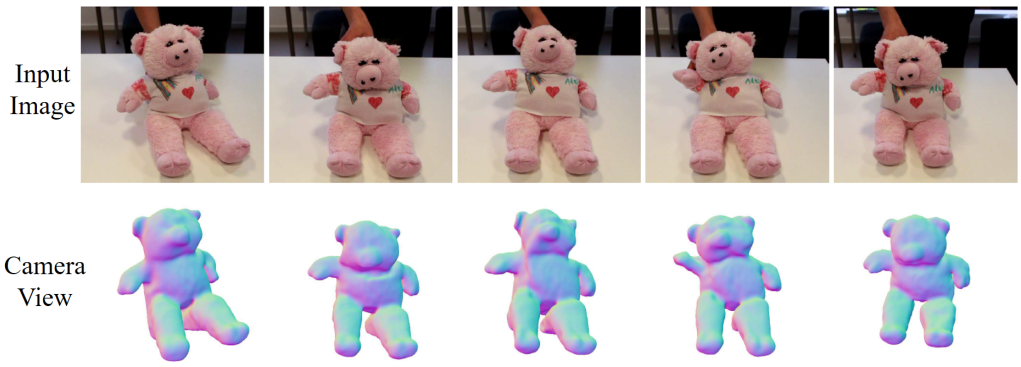


Figure 2.4: Qualitative results of Yu et al. [99].

While template-based tracking methods have demonstrated good results when restricting the object class, generic methods can struggle with large deformations and can result in self-intersections (see Figure 5.4). The most significant limitation of template tracking methods is the requirement to acquire a template of the object. This requires a rigid sequence around the object or character [94, 99] which can then be reconstructed using e.g. COLMAP [71, 72]. Needing access to a template makes this method impractical for in-the-wild application, limiting its future application mostly to controlled environments. Unlike NRSfM, template tracking methods will output water-tight meshes, provided that the input template is a water-tight mesh.

In contrast to template-based tracking methods, our method, Ub4D, does not require access to an object template nor does it make any additional assumptions about the class of the object. This provides the path towards in-the-wild application of Ub4D and similar approaches. Two of our introduced datasets include rigid sub-sequences; however, we demonstrate that these are not necessary in general (see Section 5.2). While template-based tracking provides 3D correspondences over the sequence through vertex correspondences of the template, Ub4D produces per-frame geometry that does not correspond over the sequence. As shown in Section 5.6, Ub4D can be seen as constructing a template in the canonical space from the monocular RGB sequence itself, without specifically requiring a rigid sequence.

2.3 Neural Surface Reconstruction

While some of the methods discussed previously in Section 2.1 and Section 2.2 have leveraged neural networks (e.g. Sidhu et al. [75] and Habermann et al. (2020) [24]), this section discusses approaches that are primarily neural in their implementation. We refer to these approaches as neural surface reconstruction. Most of these approaches focus on reconstruction from a single view [15, 21, 31, 64, 65, 73, 91] which is an even more ill-posed problem than approached in this work. They tackle this restricted setting by requiring explicit 3D ground truth supervision [15, 21, 64, 65, 73, 91] or focusing on specific object categories [31], both of which limit the applicability of these methods. Li et al. [39] is a neural surface reconstruction approach that operates on input monocular RGB sequences; however, it constructs a category-specific template from additional input single views of the same object category. Compared to these works, Yang et al. [95] takes monocular video as input and is not specific to an object category making it the most comparable work.

Yang et al. [95] computes optical flow and segmentations from the input monocular video, then outputs an articulated mesh with per-frame joint angles and camera parameters. They call their method LASR. LASR optimizes a base shape with an articulated skeleton with skinning weights for the rest shape and then uses a Convolutional Neural Network (CNN) to determine the joint angles and camera parameters of each frame in the sequence. The base shape is determined by deforming a sphere in a coarse-to-fine manner. One downside of this particular approach is its inability to model objects that are not topologically equivalent to a sphere (i.e. genus-0). This work demonstrates state-of-the-art results for comparable neural surface extraction approaches and we show some of their qualitative results in Figure 2.5. We compare against the results of their method on our introduced datasets in Section 5.4.

The neural surface extraction approach is recent relative to NRSfM and template-based tracking; however, has demonstrated promising results for single view reconstruction. However, reliance on 3D ground truth supervision or object category limits applicability and scalability of this class of methods. Few neural surface extraction methods have yet been developed that reconstruct non-rigidly deforming general objects. Yang et al. [95] show promising results for optimized articulated meshes, but are restricted by deforming from an initial sphere which restricts the topology of the results.

In contrast, our method, Ub4D, has no restriction on the topology of its output (see Figure 5.3 that shows Ub4D reconstructing a very high genus object) which increases the range of object categories that it can handle. Additionally, unlike many neural surface reconstruction approaches, we do not use 3D ground truth for supervision. Our optional scene flow loss uses only estimated geometric proxies. The method of Yang et al. [95] produces 3D correspondences, which Ub4D does not. Section 6.2 discusses ideas for extracting 3D correspondences from Ub4D.

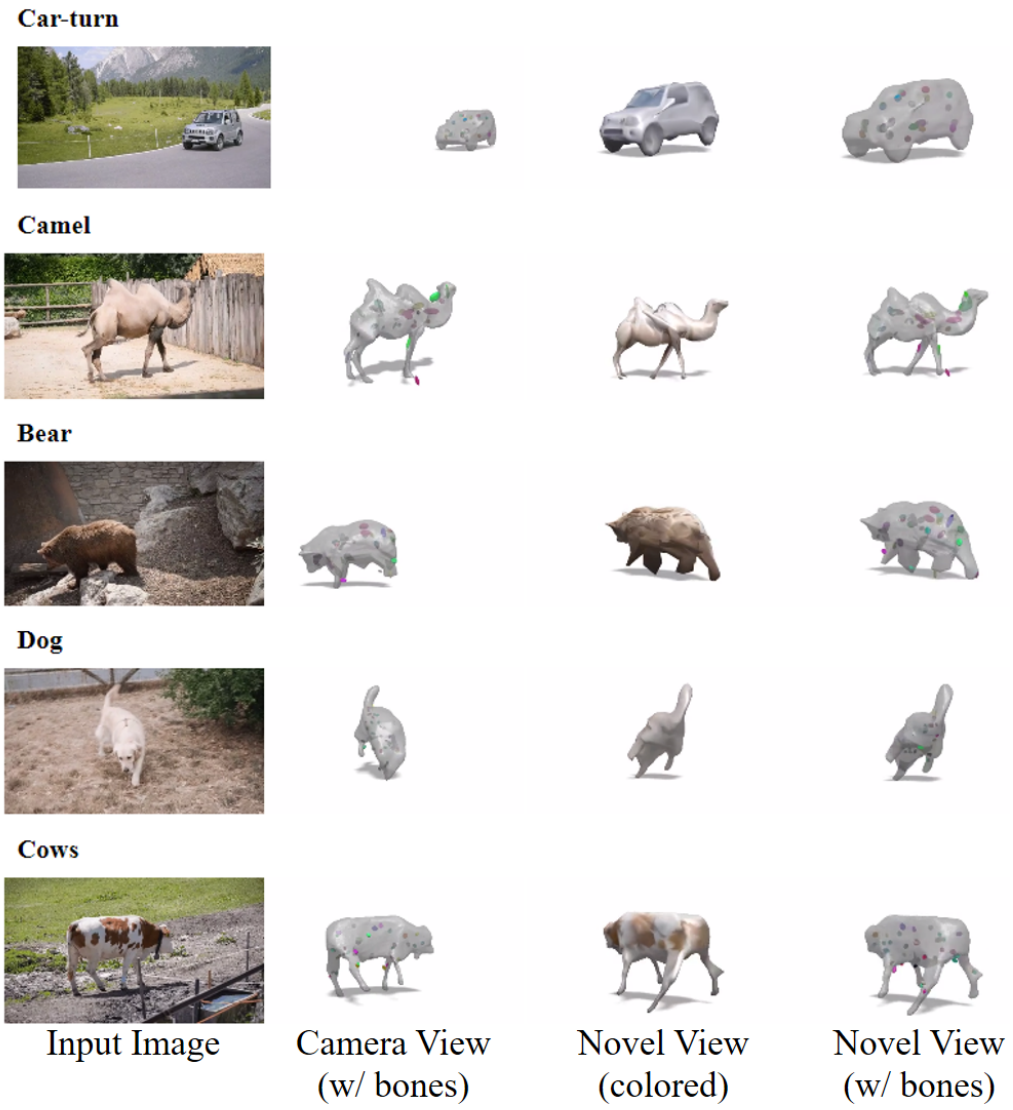


Figure 2.5: Qualitative results of Yang et al. [95]. Colored dots show the bone locations. Note how the novel views suffer from monocular depth ambiguities and are relatively coarse in their representation.

Chapter 3
Background

This work builds upon ideas that are not currently part of the literature on monocular 4D reconstruction. These are volumetric rendering and Implicit Scene Representations (ISRs) which are discussed in Section 3.1 and Section 3.2, respectively. ISRs are particularly important for our approach to monocular 4D reconstruction and in addition to the original ISR concept, we also discuss how they are learned from images (Section 3.2.1), their extension to non-rigidly deforming scenes (Section 3.2.2), and their use for static geometry reconstruction (Section 3.2.3). We also provide details on the metrics used for quantifying explicit geometry reconstruction quality in Section 3.3. An important algorithm for our method is marching cubes, which is discussed in Section 3.4.

3.1 Volumetric Rendering

In this section, we introduce the mathematical formulation of volumetric rendering and a key simplification from which Implicit Scene Representations (ISRs) take inspiration. The rendering of volumetric phenomena is a common task in computer graphics. This involves the simulation of four distinct processes that occur for light in volumes: absorption, out-scattering, in-scattering, and emission [13, 30, 51, 57]. Absorption and out-scattering both reduce the radiance along a path through a medium, whereas in-scattering and emission increase the radiance. The differential radiative transfer equation for a point in space $\boldsymbol{x} \in \mathbb{R}^3$ and a direction $\boldsymbol{d} \in \mathbb{R}^3$ is:

$$\frac{\mathrm{d}\,L(\boldsymbol{x},\boldsymbol{d})}{\mathrm{d}t} = \underbrace{-\mu_a(\boldsymbol{x})\,L(\boldsymbol{x},\boldsymbol{d})}_{\text{absorption}} \underbrace{-\,\mu_s(\boldsymbol{x})\,L(\boldsymbol{x},\boldsymbol{d})}_{\text{out-scattering}} + \underbrace{\mu_s(\boldsymbol{x})\,L_s(\boldsymbol{x},\boldsymbol{d})}_{\text{in-scattering}} + \underbrace{\mu_a(\boldsymbol{x})\,L_e(\boldsymbol{x},\boldsymbol{d})}_{\text{emission}}, \tag{3.1}$$

where μ_a is the absorption coefficient, μ_s is the scattering coefficient, L is the radiance, L_s is the in-scattered radiance, and L_e is emitted radiance [13, 30, 51, 57]. This can be seen as a light-field formulation defining radiances at a given point $\boldsymbol{x}$ traveling in a certain direction $\boldsymbol{d}$. Typically, absorption and out-scattering are combined into a generalized attenuation term with $\mu_t = \mu_a + \mu_s$ as the attenuation coefficient [51, 57].

The differential form of Equation (3.1) is typically integrated for computer solution [51,

57]. The integral equation for a straight line from a viewing point $\boldsymbol{x}$ to an example surface point $\boldsymbol{x}_z$ is:

$$\begin{aligned} L(\boldsymbol{x}, \boldsymbol{d}) = & \underbrace{T_r(\boldsymbol{x}_z \to \boldsymbol{x})\, L(\boldsymbol{x}_z, \boldsymbol{d})}_{\text{from surface point } \boldsymbol{x}_z} \\ & + \underbrace{\int_0^z T_r(\boldsymbol{x}_\zeta \to \boldsymbol{x})\, \mu_a(\boldsymbol{x}_\zeta)\, L_e(\boldsymbol{x}_\zeta, \boldsymbol{d})\ \mathrm{d}\zeta}_{\text{volume emission}} \\ & + \underbrace{\int_0^z T_r(\boldsymbol{x}_\zeta \to \boldsymbol{x})\, \mu_s(\boldsymbol{x}_\zeta) \int_{S^2} f_p(\boldsymbol{d}_i, \boldsymbol{x}_\zeta, \boldsymbol{d})\, L_i(\boldsymbol{x}_\zeta, \boldsymbol{d}_i)\ \mathrm{d}\boldsymbol{d}_i\ \mathrm{d}\zeta}_{\text{in-scattered radiance}}, \end{aligned} \tag{3.2}$$

where:

$$T_r(\boldsymbol{y} \to \boldsymbol{x}) = \exp\left(- \int_{\boldsymbol{y}}^{\boldsymbol{x}} \mu_t(\boldsymbol{\gamma})\ \mathrm{d}\boldsymbol{\gamma} \right)$$

is the beam transmittance modeling the attenuation from $\boldsymbol{y}$ to $\boldsymbol{x}$. Additionally, f_p is the in-scattering phase function (e.g. Henyey-Greenstein [26]) and L_i is incoming radiance, whose product is integrated over the sphere S^2. Equation (3.2) allows practical implementation in the ray-tracing computer graphics model [30] with efficient solution possible through the use of Monte Carlo integration methods [51, 57].

One relevant acceleration is the use of texture alpha compositing [58] developed for scientific visualization of volumetric data [12, 17, 34]. This method represents the volume using textures that are combined based on their sampled alpha values. If we have a set of n ordered color $\boldsymbol{c}_i$ and alpha α_i samples, the combined color can be computed as:

$$\boldsymbol{c} = \sum_{i=1}^{n} \alpha_i\, \boldsymbol{c}_i \prod_{j=1}^{i-1} (1 - \alpha_j)\,. \tag{3.3}$$

To compare this with Equation (3.2), consider the discretization for a volume in a scene without any surface and ignoring *all* scattering:

$$L = \sum_{i=1}^{z} \left(1 - \exp\left(-\delta_i\, \mu_a(\boldsymbol{x}_i)\right)\right) L_e(\boldsymbol{x}_i, \boldsymbol{d}) \exp\left(- \sum_{j=1}^{i-1} \delta_j\, \mu_a(\boldsymbol{x}_j) \right) \tag{3.4}$$

$$= \sum_{i=1}^{z} \left(1 - \exp\left(-\delta_i\, \mu_a(\boldsymbol{x}_i)\right)\right) L_e(\boldsymbol{x}_i, \boldsymbol{d}) \prod_{j=1}^{i-1} \exp\left(-\delta_j\, \mu_a(\boldsymbol{x}_j)\right)\,, \tag{3.5}$$

where δ_i is the sample distance. Note that this integral approximation is the improved version from Max and Chen [46], accounting for the nearer portion of each finite region occluding the further portion, compared to a naïve Riemann discretization [62]. Comparing Equation (3.5) and Equation (3.3), we can see that they can be directly related by setting:

$$\alpha_i = 1 - \exp\left(-\delta_i\, \mu_a(\boldsymbol{x}_i, \boldsymbol{d})\right)\,. \tag{3.6}$$

3.2 Implicit Scene Representations

An Implicit Scene Representation (ISR) is a method for representing a scene without using explicit geometry (e.g. triangles). This can provide substantial benefits for learning tasks due to the challenges of gradient propagation and topological changes encountered when using explicit geometry. The most common representation, and the one used in this work, is an ISR using one or multiple Neural Networks (NNs) taking 3D coordinates as input. The output of the NN depends on the desired properties and application of the ISR. Examples of outputs include occupancy [14, 47, 50], Signed Distance Function (SDF) value [53, 92, 97], volume density [48, 86, 92], color [44, 48, 86, 92, 97], or alpha value [44].

Early examples of ISR approaches used input 3D ground truth data to train their NNs [14, 47, 53]. They demonstrated applications like shape completion [47, 53], novel shape generation [14, 47], and single view 3D reconstruction [14, 47]. However, access to 3D ground truth training data is highly restrictive and limits the applicability of these early ISR approaches. Being able to learn an ISR from 2D images provides a much wider range of application.

3.2.1 Image Supervision

The key element enabling image supervision is the differentiability of the rendering process. This allows for the supervision of a rendered ray by the pixel from which it originated. Different rendering approaches have been proposed, including ray marching with a Long Short-Term Memory (LSTM) network [77], texturing estimated depth maps [50], and predicting radiance at a surface point with a Multi-Layer Perceptron (MLP) network [97]. However, we focus on the approach most similar to volumetric rendering: treating the scene as an emissive volume without scattering [44, 48]. This case was shown to reduce to alpha compositing in Section 3.1.

Specifically, the discretized rendering equation of Mildenhall et al. [48] is:

$$\hat{C}(\boldsymbol{r}) = \sum_{i=1}^{n} \left(1 - \exp(-\delta_i\, \sigma_i)\right) \boldsymbol{c}_i \prod_{j=1}^{i-1} \exp\left(-\delta_j\, \sigma_j\right) \,, \tag{3.7}$$

where σ_i is a sample from the density network, $\boldsymbol{c}_i$ is a sample from the color network, and δ_i is the interval width. The density and color networks take 3D coordinates and viewing directions as input. These networks form the ISR used by Mildenhall et al. [48]. Note that Equation (3.7) exactly resembles Equation (3.5) for $\sigma_i = \mu_a(\boldsymbol{x}_i)$ and $\boldsymbol{c}_i = L_e(\boldsymbol{x}_i, \boldsymbol{d})$. Figure 3.1 provides a visualization of this neural volumetric rendering process.

This form of ISR trained from images provides state-of-the-art results for novel view synthesis. The model of an emissive volume without scattering works very well in this case as it can capture the light emitted from surfaces and allows for occlusions. However, these early volumetric models bake the object lighting into the ISR which is not the case for explicit mesh representations. Another downside of such coordinate-based density representations is that there is no true surface and extracting surfaces with a density threshold can result in noisy surfaces as will be demonstrated in Section 5.3. For our problem of monocular 4D reconstruction, there are two extensions that are of interest: non-rigidly deforming scenes (Section 3.2.2) and improved explicit geometry extraction (Section 3.2.3).

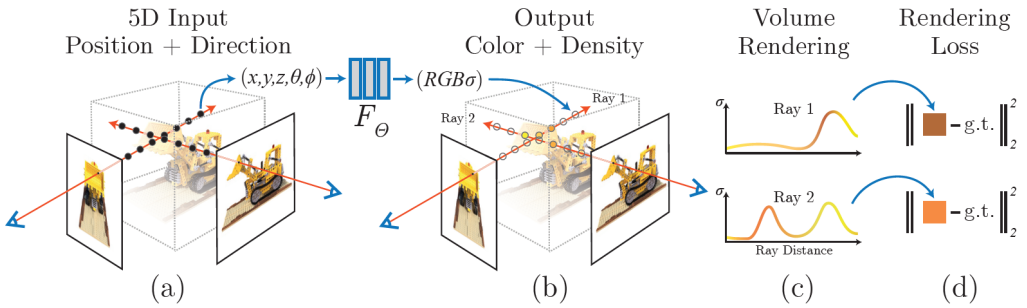


Figure 3.1: Demonstrative figure of the neural volumetric rendering process from Mildenhall et al. [48]. The process proceeds as follows: (a) Discrete samples are taken along rays fired into the scene. (b) Each sample point's position and direction are fed to a density and color network. (c) These samples are rendered into a single color estimate by Equation (3.7). (d) A loss is applied against the true pixel color which supervises the ISR.

3.2.2 Non-Rigidly Deforming Scenes

ISRs have been extended to handle image sequences of non-rigidly deforming scenes. A naïve idea for such an extension is to provide the networks with an additional time input. However, this alone does not produce high quality results [86, 93] or requires RGB-D input [93]. Another idea is to use a learned 3D scene flow to transform between neighbouring timesteps [41]. Finally, a per-frame deformation to a canonical space can be used [54, 59, 86].

This deformation can be modeled using an MLP conditioned on a learned per-frame latent code [54, 86]. This approach is depicted in Figure 3.2. A convenient way to view this deformation is as rendering along a bent ray in the canonical space. Thus the deformation network is also referred to as a bending network. Another key idea introduced in Tretschk et al. [86] is the zero initialization of the per-frame latent codes provided to the bending network. We show in Section 5.8 that this makes a substantial difference when compared with random Gaussian initialization.

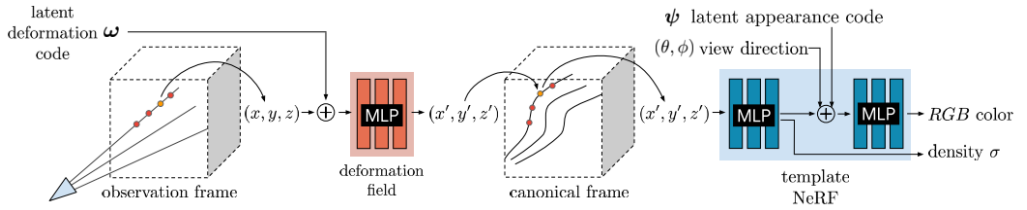


Figure 3.2: Demonstrative figure of neural volumetric rendering with a per-frame deformation from Park et al. [54]. Discrete samples along the rays in the observation frame are deformed through a per-frame learned deformation field into a canonical frame. Then the same rendering process as shown in Figure 3.1 is performed along these bent rays.

3.2.3 Explicit Geometry Reconstruction

While Mildenhall et al. [48] demonstrated that explicit geometry can be extracted from the density network through the use of the marching cubes algorithm [45] discussed in Section 3.4, the results contain significant noise and require determining a threshold for the density. Various ideas have been introduced to improve the explicit geometry extracted from ISRs. Most works focusing on surface reconstruction use Signed Distance Functions (SDFs) to model geometry [92, 96, 97]. An SDF is beneficial since it provides an obvious threshold (i.e. 0) whereas density-based representations must determine a threshold without any obvious optimal value. An alternative approach is given by Oechsle et al. [52] where they formulate the rendering method so that the volume rendering converges to an exact surface defined by a threshold during training. When considering importance sampling, the methods of Wang et al. [92] and Yariv et al. (2021) [96] both resemble Oechsle et al. [52] since these works construct the density from a probability distribution that narrows on the surface over time. They define the probability distribution as a function of the SDF value.

This notion of constructing the density from the SDF is especially powerful since it lends itself to mathematical proofs, in addition to working well when implemented. Our method is most similar to that of Wang et al. [92] since they use the logistic probability distribution to construct the density, whereas Yariv et al. (2021) [96] use the Laplace distribution. Using the notation of Section 3.1, Wang et al. [92] define the density as:

$$\mu_a(\boldsymbol{x}_i) = \max\left\{\frac{-\frac{\mathrm{d}\Phi_s}{\mathrm{d}t}\big(f(\boldsymbol{x}_i)\big)}{\Phi_s\big(f(\boldsymbol{x}_i)\big)}, 0\right\}, \tag{3.8}$$

where Φ_s is the logistic Cumulative Distribution Function (CDF) and f is the scene SDF. Then we can consider Equation (3.5) to be a weighted sum as:

$$\boldsymbol{c} = \sum_{i=1}^{z} \mathrm{w}(\boldsymbol{x}_i)\, L_e(\boldsymbol{x}_i, \boldsymbol{d})\,, \tag{3.9}$$

where:

$$\mathrm{w}(\boldsymbol{x}_i) = \Big(1 - \exp\big(-\delta_i\, \mu_a(\boldsymbol{x}_i)\big)\Big) \prod_{j=1}^{i-1} \exp\big(-\delta_j\, \mu_a(\boldsymbol{x}_j)\big)\ . \tag{3.10}$$

This weight function w is visualized for a toy example in Figure 3.3. Wang et al. [92] show that this weight function is unbiased with respect to the surface (i.e. 0 level set of SDF f) and, in Section 4.2.5, we provide an alternative proof for any smooth parametric path.

3.3 Geometry Reconstruction Metrics

In determining the accuracy of our geometry reconstruction, we employ two common metrics: the Chamfer Distance (CD) and the Hausdorff Distance (HD). Note though that we use a different definition of the HD than as it was originally defined [27]. We define the HD as the asymmetric average distance from one set to the nearest points in another set:

$$d_{\mathrm{HD}}(S_1, S_2) = \frac{1}{|S_1|} \sum_{x \in S_1} \min_{y \in S_2} ||x - y||_2^2\,. \tag{3.11}$$

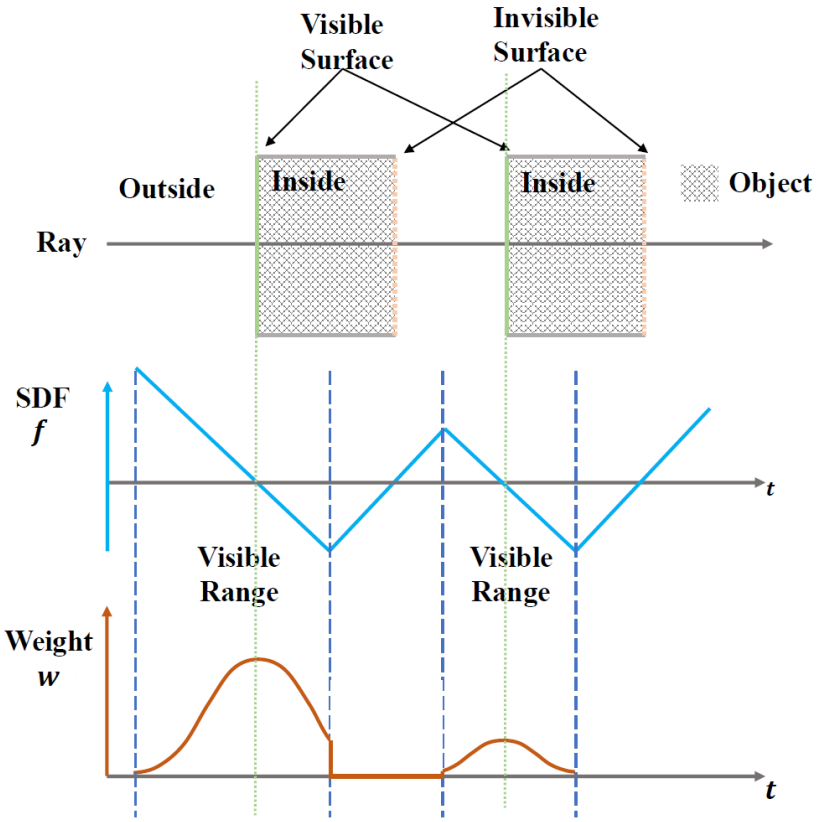


Figure 3.3: Demonstrative figure from Wang et al. [92] for the specially designed weight function in Equation (3.10). Note how the weight function computed from the SDF attains a local maximum at each potentially visible surface along the ray. The width of the weight distribution is parameterized by s in the logistic CDF of Equation (3.8).

When using the HD we always consider the estimated vertices as S_1 and measure the average distance to the ground truth mesh S_2. Since this is a mesh rather than a point cloud, the HD considers the set S_2 to include all surface points of the mesh and not just its vertices. Then we define the CD as the symmetric average distance between the sets in terms of this HD:

$$d_{\mathrm{CD}}(S_1, S_2) = d_{\mathrm{HD}}(S_1, S_2) + d_{\mathrm{HD}}(S_2, S_1)\,. \tag{3.12}$$

This definition of the CD is consistent with its original definition [4].

3.4 Marching Cubes

Marching cubes is an algorithm for converting an implicit representation into an explicit surface. It was originally described by Lorensen and Cline [45]. The algorithm samples a value for each point in a uniform 3D grid. It then identifies the triangles necessary to represent each cell from a set of 15 possible configurations (excluding symmetries) based on thresholding of the 8 corner samples. All 15 possible cases are shown in Figure 3.4.

The specific vertex positions for the triangulation are determined by linearly interpolating based on the values for the 2 grid points of the edge requiring a vertex.

This algorithm is very simple and easy to implement; however, it does suffer from drawbacks. The most notable drawback is a lack of adaptability to local complexity. This is similar to the "teapot in a stadium" problem in computer graphics, typically encountered for spatial partitioning [57]. That is, it either wastes samples in empty space or results in low resolution output. While adaptive modifications exist [70, 74], we find the original algorithm suffices in our case, given that we are interested in localized objects and do not require real-time performance.

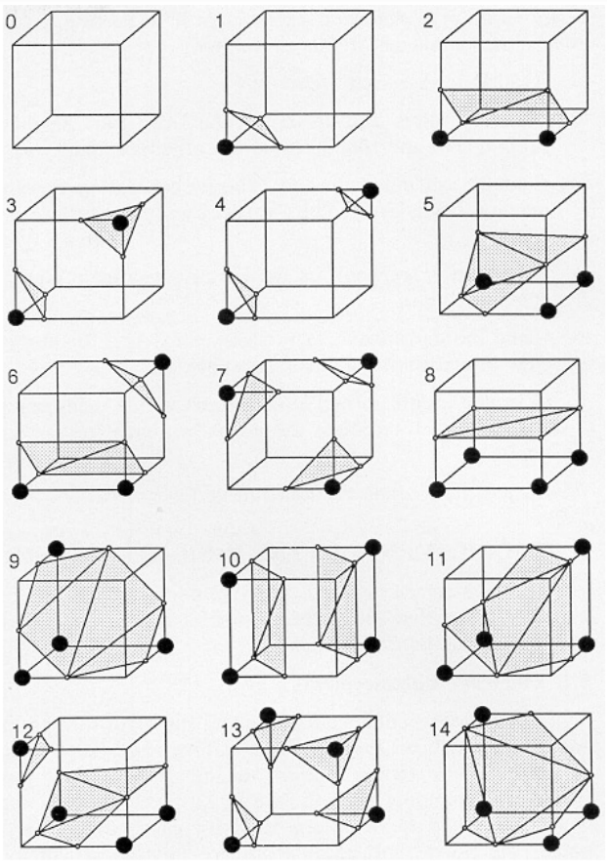


Figure 3.4: Triangle configurations for marching cubes from Lorensen and Cline [45]. Note that this does not include the symmetries for each case.

Chapter 4
Method

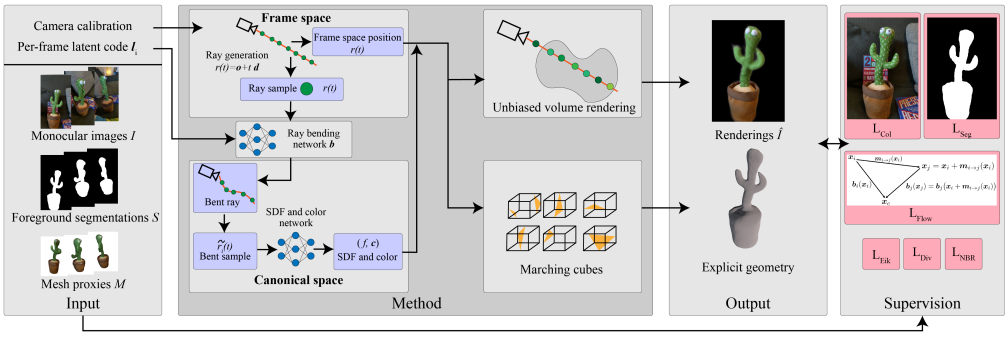


Figure 4.1: Our Ub4D approach takes as input a sequence of images and respective foreground segmentations recorded with a single calibrated RGB camera. In addition, each frame is also equipped with a learnable latent code. Optionally, we accept a corresponding proxy geometry of the object. Given this, our method learns a canonical and colored SDF representing the static scene. Our bending network maps the frame space to canonical space and volume rendering and marching cubes can produce per-frame renderings and geometries, respectively. We weakly supervise our scene representation with image-based losses as well as spatio-temporal priors including our novel scene flow loss.

We present our method, called Unbiased 4D or Ub4D, for reconstructing non-rigidly deforming objects from a monocular RGB sequence. Figure 4.1 presents an overview showing the components of the method. Ub4D takes as input a monocular RGB sequence with segmentations, as well as the camera parameters. In addition, we optionally accept a corresponding geometry proxy for use in our novel scene flow loss. In an analysis-by-synthesis approach, Ub4D jointly learns a per-frame deformation to a canonical space and the surface in that canonical space as a Signed Distance Function (SDF). Explicit geometry is then extracted for each frame.

Section 4.1 explains the model we use for temporal, non-rigid deformations. Then, in Section 4.2 we provide the rendering method that converts the scene representation into

RGB images and segmentations. These renderings are supervised with losses given in Section 4.2.3. We also prove that Ub4D's reconstructions are unbiased with respect to the surface in Section 4.2.5. Next, we present our novel approach to constraining the reconstruction with the scene flow from the geometric proxy in Section 4.3. We summarize the losses and regularizers from the previous sections in Section 4.4 and give details on converting the coordinate-based implicit representation into explicit geometry in Section 4.5. Finally, Section 4.6 provides further details on the practical implementation of Ub4D.

4.1 Non-Rigidity Model

Ub4D models temporal non-rigid deformations as a vector field projecting points from the frame space into a shared canonical space. This can be seen as a frame-dependent bending of the straight rays originating from the camera. Given a straight ray with an origin $\boldsymbol{o} \in \mathbb{R}^3$ and a viewing direction $\boldsymbol{d} \in \mathbb{R}^3$ as $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$, we bend the ray with a frame-dependent bending network $\boldsymbol{b}_i : \mathbb{R}^3 \to \mathbb{R}^3$ as:

$$\tilde{\boldsymbol{r}}_i(t) = \boldsymbol{r}(t) + \boldsymbol{b}_i(\boldsymbol{r}(t)) \,, \tag{4.1}$$

where i denotes the frame. We desire that this bending network transforms points from frame space into a single canonical representation of the object shared by all frames.

This bent ray is a directed parametric path in $\mathbb{R}^3$ like the straight ray, but, where the derivative of the straight ray is constant (i.e. $\frac{\mathrm{d}\boldsymbol{r}(t)}{\mathrm{d}t} = \boldsymbol{d}$), the bent ray has an instantaneous direction at each point along it. We can compute this instantaneous direction analytically as:

$$\frac{\mathrm{d}\tilde{\boldsymbol{r}}_i(t)}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t}\big[\boldsymbol{r}(t)\big] + \frac{\mathrm{d}}{\mathrm{d}t}\big[\boldsymbol{b}_i(\boldsymbol{r}(t))\big] = \boldsymbol{d} + \frac{\partial \boldsymbol{b}_i}{\partial \boldsymbol{r}(t)}\frac{\mathrm{d}}{\mathrm{d}t}\big[\boldsymbol{r}(t)\big] = \boldsymbol{d} + \frac{\partial \boldsymbol{b}_i}{\partial \boldsymbol{r}(t)}\boldsymbol{d} \,, \tag{4.2}$$

where $\frac{\partial \boldsymbol{b}_i}{\partial \boldsymbol{r}(t)}$ is the Jacobian of the bending network with respect to its input $\boldsymbol{r}(t)$, the point along the straight ray. Note that, in our case, this Jacobian exists everywhere since the bending network is implemented with a Multi-Layer Perceptron (MLP), which takes as input a continuous point in 3D space; thus our bent ray is a *smooth* parametric path.

While this discussion presents the bending network as a per-frame vector field throughout $\mathbb{R}^3$, it is actually implemented using a per-frame learned latent code $\boldsymbol{l}_i \in \mathbb{R}^{64}$. This latent code is given as input along with a point in space to an MLP $\boldsymbol{b} : (\mathbb{R}^3, \mathbb{R}^{64}) \to \mathbb{R}^3$ and the latent codes are optimized during training. Note that this latent code is the *only* frame-specific element in our method and no network other than the bending network receives it. One can see this as a factorization between the temporal and spatial domains where our method forces time to be entirely modeled in the latent code and bending network. We further inspect the latent codes after training in Section 5.8.

Our non-rigidity model is similar to that of Tretschk et al. [86]. However, we propose a different regularization to enable the modeling of larger deformations, which also removes the need to learn a rigidity score throughout the scene. Whereas Tretschk et al. [86] penalizes the bending network output for its absolute length, we instead enforce that the deformation of the current frame is similar to that of the neighboring frames. This assumes that neighboring frames represent similar object states, which is a more reasonable assumption for dynamic scenes compared to the absolute amount of deformation.

Specifically, for n_s samples along a straight ray $\boldsymbol{r}$, we penalize the bending network as:

$$L_{\text{NBR}} = \frac{1}{n_s} \sum_{z=1}^{n_s} \sum_{j \in \mathcal{N}(i)} \omega_i^{(z)} \, ||\boldsymbol{b}_i(\boldsymbol{r}(t^{(z)})) - \boldsymbol{b}_j(\boldsymbol{r}(t^{(z)}))||_2^2 \,, \tag{4.3}$$

where $\omega_i^{(z)}$ is the visibility weight at sample z along the bent ray (see Section 4.2.2) and $\mathcal{N}(i)$ are the neighbours of frame i. We also penalize the divergence of the bending network as:

$$L_{\text{DIV}} = \frac{1}{n_s} \sum_{z=1}^{n_s} \omega_i^{(z)} \, |\nabla \cdot \boldsymbol{b}_i(\boldsymbol{r}(t^{(z)}))|^2 \,, \tag{4.4}$$

where we use the unbiased, approximated divergence from Tretschk et al. [86].

4.2 Rendering Bent Rays

In this section we show how we render bent rays given in the form of Equation (4.1) with the goal of learning the surface. Treating these bent rays as a functional input, we extend the rendering method of Wang et al. [92] to dynamic scenes and prove that it remains unbiased with respect to the surface. Section 4.2.1 explains the continuous form of the rendering method and Section 4.2.2 shows how we discretize it for implementation. Note that this discretization refers to space, while time is discrete throughout. Next, Section 4.2.3 presents the reconstruction losses and the Eikonal regularization of the SDF. The derivation of the discrete opacity equation (Equation (4.14)) is a significant detail in the rendering method and is given in Section 4.2.4. Finally, Section 4.2.5 provides a proof of the unbiased nature of our rendering method.

4.2.1 Continuous Form

For each bent ray, we are interested in computing two properties: the rendered color of the ray and whether a ray hits the object or not. First, the rendered color can be computed with an integral along the ray:

$$\hat{I}(\tilde{\boldsymbol{r}}_i) = \int_0^\infty \omega(\tilde{\boldsymbol{r}}_i, t) \, \boldsymbol{c}\left(\tilde{\boldsymbol{r}}_i(t), \tfrac{\mathrm{d}\tilde{\boldsymbol{r}}_i(t)}{\mathrm{d}t}\right) \mathrm{d}t \,, \tag{4.5}$$

where $\omega(\tilde{\boldsymbol{r}}_i, t)$ is an occlusion-aware weighting functional for a distance t along the bent ray $\tilde{\boldsymbol{r}}_i$ and $\boldsymbol{c} : (\mathbb{R}^3, \mathbb{R}^3) \to \mathbb{R}^3$ is a color network taking a point along the bent ray and its instantaneous direction at that point. We compute $\omega(\tilde{\boldsymbol{r}}_i, t)$ as:

$$\omega(\tilde{\boldsymbol{r}}_i, t) = T(\tilde{\boldsymbol{r}}_i, t) \, \rho(\tilde{\boldsymbol{r}}_i(t)) \,, \tag{4.6}$$

where:

$$T(\tilde{\boldsymbol{r}}_i, t) = \exp\left(-\int_0^t \rho(\tilde{\boldsymbol{r}}_i(\tau)) \, \mathrm{d}\tau\right) , \tag{4.7}$$

and:

$$\rho(\tilde{\boldsymbol{r}}_i(t)) = \max\left\{\frac{-\frac{\mathrm{d}\Phi_s}{\mathrm{d}t}\big(f(\tilde{\boldsymbol{r}}_i(t))\big)}{\Phi_s\big(f(\tilde{\boldsymbol{r}}_i(t))\big)}, 0\right\}, \tag{4.8}$$

where Φ_s is the Cumulative Distribution Function (CDF) of the logistic distribution. Here T is the accumulated transmittance, which allows the modeling of occlusion and ρ is the density computed from the SDF f that we primarily seek to learn. Note that the form of ρ in Equation (4.8) is the derivative of the natural logarithm.

We can integrate the weights in order to determine if a ray hits the object:

$$\hat{S}(\tilde{\boldsymbol{r}}_i) = \int_0^\infty \omega(\tilde{\boldsymbol{r}}_i, t)\ \mathrm{d}t\,, \tag{4.9}$$

which gives 1 for rays hitting the object and 0 for rays entirely in free space. In order to implement this rendering method, we must formulate these equations using discrete samples along the ray.

4.2.2 Discretization

We seek discrete equations for the color and whether a ray hits the object or not, given n_s samples along the bent ray: $\{\tilde{\boldsymbol{r}}_i(t^{(z)}) : z \in \mathbb{Z}, z \in [1, n_s]\}$ where $t^{(z)} < t^{(z+1)}, \forall z$. These samples are taken in a hierarchical manner similar to Mildenhall et al. [48]: an importance sampling based on weight values from an initial uniform sampling. However, unlike Mildenhall et al. [48], we maintain only a single copy of our networks like Wang et al. [92].

Given these samples we can formulate the discrete color equation as:

$$\hat{I}(\tilde{\boldsymbol{r}}_i) = \sum_{z=1}^{n_s-1} \omega_i^{(z)}\, \boldsymbol{c}\left(\tilde{\boldsymbol{r}}_i(t^{(z)}), \tilde{\boldsymbol{r}}_i(t^{(z+1)}) - \tilde{\boldsymbol{r}}_i(t^{(z)})\right)\,, \tag{4.10}$$

where we approximate the instantaneous ray direction from Equation (4.5) using a forward difference. This was shown in Tretschk et al. [86] to have minimal impact on the novel view synthesis quality. We can compute the discrete weight samples $\omega_i^{(z)}$ as:

$$\omega_i^{(z)} = T_i^{(z)}\, \alpha_i^{(z)}\,, \tag{4.11}$$

where:

$$T_i^{(z)} = \prod_{\zeta=1}^{z-1} \left(1 - \alpha_i^{(\zeta)}\right)\,, \tag{4.12}$$

and:

$$\alpha_i^{(z)} = 1 - \exp\left(-\int_{t^{(z)}}^{t^{(z+1)}} \rho(\tilde{\boldsymbol{r}}_i(t))\ \mathrm{d}t\right) \tag{4.13}$$

$$= \max\left\{\frac{\Phi_s\Big(f\big(\tilde{\boldsymbol{r}}_i(t^{(z)})\big)\Big) - \Phi_s\Big(f\big(\tilde{\boldsymbol{r}}_i(t^{(z+1)})\big)\Big)}{\Phi_s\Big(f\big(\tilde{\boldsymbol{r}}_i(t^{(z)})\big)\Big)}, 0\right\}\,. \tag{4.14}$$

Note that in Equation (4.11) we reformulate from the density accumulation of Equation (4.6) to an alpha compositing formulation as shown in Section 3.1. The discrete opacity $\alpha_i^{(z)}$ is defined as in Equation (4.13) and we show in Section 4.2.4 that, given our specific definition of ρ from Equation (4.8), this is can be easily computed as in Equation (4.14).

As a direct analog to Equation (4.9), we can compute the segmentation of a ray as:

$$\hat{S}(\tilde{\boldsymbol{r}}_i) = \sum_{z=1}^{n_s-1} \omega_i^{(z)} \,, \tag{4.15}$$

where $\hat{S}(\tilde{\boldsymbol{r}}_i)$ is 1 for rays hitting the object and 0 for rays entirely in free space. Now that we have established how we can compute the color and segmentation of a bent ray in practice, we can look at the reconstruction losses used to learn the surface.

4.2.3 Reconstruction Losses and Eikonal Regularizer

While the previous analysis looked at a bent ray from a single pixel, we now consider a batch of pixels $\mathcal{P}$. If we write the straight ray originating from a pixel p as $\boldsymbol{r}^{(p)}$, then the bent ray for that pixel is:

$$\tilde{\boldsymbol{r}}_i^{(p)}(t) = \boldsymbol{r}^{(p)}(t) + \boldsymbol{b}_i\big(\boldsymbol{r}^{(p)}(t)\big) \,. \tag{4.16}$$

With this, the color loss can be expressed for a specific frame i as:

$$L_{\text{COL}} = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} |\hat{I}\big(\tilde{\boldsymbol{r}}_i^{(p)}\big) - I^{(p)}| \,, \tag{4.17}$$

where $I^{(p)}$ is the ground truth color of pixel p. The segmentation loss uses binary cross entropy between the estimated segmentation $\hat{S}\big(\tilde{\boldsymbol{r}}_i^{(p)}\big)$ and pixel p's respective ground truth segmentation value $S^{(p)} \in \{0, 1\}$:

$$L_{\text{SEG}} = -\frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} S^{(p)} \ln\Big(\hat{S}\big(\tilde{\boldsymbol{r}}_i^{(p)}\big)\Big) + \big(1 - S^{(p)}\big) \ln\Big(1 - \hat{S}\big(\tilde{\boldsymbol{r}}_i^{(p)}\big)\Big) \,. \tag{4.18}$$

In addition to these reconstruction losses, we also impose an Eikonal regularization loss on the SDF network f:

$$L_{\text{EIK}} = \frac{1}{|\mathcal{P}|\, n_s} \sum_{p \in \mathcal{P}} \sum_{z=1}^{n_s} \left(\left| \nabla f\Big(\tilde{\boldsymbol{r}}_i^{(p)}\big(t^{(z)}\big)\Big) \right| - 1 \right)^2 \,. \tag{4.19}$$

Note that this Eikonal loss is performed on bent ray sample points, thus it is applied most at the points in canonical space to where rays are bent and where the density is highest (due to the two-stage hierarchical sampling discussed in Section 4.2.2). This is thought to be beneficial since we are most interested in the visible surface crossings being the best approximation to an SDF.

4.2.4 Derivation of the Discrete Opacity $\alpha_i^{(z)}$ Equation for Bent Rays

In this section we show that the derivation of discrete opacity $\alpha_i^{(z)}$ shown in Equation (4.14) follows from volume rendering principles and the definition of the density ρ in Equation (4.8). This analysis is similar to that in the Appendix A of Wang et al. [92]. We extend their derivation to our bent rays, which were shown in Section 4.1 to be smooth parametric paths.

In Equation (4.8), we defined the density as:

$$\rho(\tilde{\boldsymbol{r}}_i(t)) = \max\left\{\frac{-\frac{\mathrm{d}\Phi_s}{\mathrm{d}t}\big(f(\tilde{\boldsymbol{r}}_i(t))\big)}{\Phi_s\big(f(\tilde{\boldsymbol{r}}_i(t))\big)}, 0\right\}. \tag{4.20}$$

In order to proceed, we must expand the numerator through the chain rule:

$$\begin{aligned}\tfrac{\mathrm{d}\Phi_s}{\mathrm{d}t}\big(f(\tilde{\boldsymbol{r}}_i(t))\big) &= \phi_s\big(f(\tilde{\boldsymbol{r}}_i(t))\big)\tfrac{\mathrm{d}}{\mathrm{d}t}\big(f(\tilde{\boldsymbol{r}}_i(t))\big) \\ &= \phi_s\big(f(\tilde{\boldsymbol{r}}_i(t))\big)\big[\nabla f(\tilde{\boldsymbol{r}}_i(t)) \cdot \tfrac{\mathrm{d}\tilde{\boldsymbol{r}}_i(t)}{\mathrm{d}t}\big]\,,\end{aligned} \tag{4.21}$$

where ϕ_s is the Probability Density Function (PDF) of the logistic distribution. There is no need to expand the instantaneous viewing direction $\frac{\mathrm{d}\tilde{\boldsymbol{r}}_i(t)}{\mathrm{d}t}$ further, but, critically, we know this derivative exists since our bent rays were shown in Section 4.1 to be smooth.

Placing Equation (4.21) into Equation (4.20), gives:

$$\rho(\tilde{\boldsymbol{r}}_i(t)) = \max\left\{-\frac{\phi_s\big(f(\tilde{\boldsymbol{r}}_i(t))\big)\big[\nabla f(\tilde{\boldsymbol{r}}_i(t)) \cdot \frac{\mathrm{d}\tilde{\boldsymbol{r}}_i(t)}{\mathrm{d}t}\big]}{\Phi_s\big(f(\tilde{\boldsymbol{r}}_i(t))\big)}, 0\right\} \tag{4.22}$$

There are two regions of interest for Equation (4.22) identified in Appendix A of Wang et al. [92]: a ray entering geometry and a ray exiting geometry.

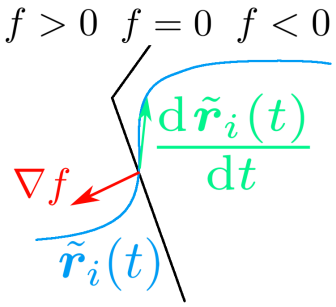


Figure 4.2: Graphical depiction of a bent ray (traveling left to right) entering an SDF surface. Note that the instantaneous viewing direction and gradient of the SDF must have a negative dot product in order for the bent ray to enter the geometry.

We first present the case where a ray enters the geometry as depicted in Figure 4.2. Since we know in this case that:

$$\nabla f(\tilde{\boldsymbol{r}}_i(t)) \cdot \tfrac{\mathrm{d}\tilde{\boldsymbol{r}}_i(t)}{\mathrm{d}t} < 0\,, \tag{4.23}$$

and that both ϕ_s and Φ_s are non-negative, we can drop the maximum in Equation (4.22) and then return the numerator to its more condensed form:

$$\begin{aligned}\rho(\tilde{\boldsymbol{r}}_i(t)) &= \frac{-\phi_s\big(f(\tilde{\boldsymbol{r}}_i(t))\big)\Big[\nabla f(\tilde{\boldsymbol{r}}_i(t)) \cdot \frac{\mathrm{d}\tilde{\boldsymbol{r}}_i(t)}{\mathrm{d}t}\Big]}{\Phi_s\big(f(\tilde{\boldsymbol{r}}_i(t))\big)} \\ &= \frac{-\frac{\mathrm{d}\Phi_s}{\mathrm{d}t}\big(f(\tilde{\boldsymbol{r}}_i(t))\big)}{\Phi_s\big(f(\tilde{\boldsymbol{r}}_i(t))\big)}\end{aligned} \tag{4.24}$$

The remaining derivation for the discrete opacity $\alpha_i^{(z)}$ follows exactly as in Appendix A of Wang et al. [92]:

$$\begin{aligned}
\alpha_i^{(z)} &= 1 - \exp\left(-\int_{t^{(z)}}^{t^{(z+1)}} \rho(\tilde{\boldsymbol{r}}_i(t))\mathrm{d}t\right) \\
&= 1 - \exp\left(-\int_{t^{(z)}}^{t^{(z+1)}} \frac{-\frac{\mathrm{d}\Phi_s}{\mathrm{d}t}\big(f(\tilde{\boldsymbol{r}}_i(t))\big)}{\Phi_s\big(f(\tilde{\boldsymbol{r}}_i(t))\big)}\mathrm{d}t\right) \\
&= 1 - \exp\Big(\ln\big[\Phi_s\big(f(\tilde{\boldsymbol{r}}_i(t^{(z+1)}))\big)\big] - \ln\big[\Phi_s\big(f(\tilde{\boldsymbol{r}}_i(t^{(z)}))\big)\big]\Big) \\
&= 1 - \frac{\Phi_s\big(f(\tilde{\boldsymbol{r}}_i(t^{(z+1)}))\big)}{\Phi_s\big(f(\tilde{\boldsymbol{r}}_i(t^{(z)}))\big)} \\
\therefore \alpha_i^{(z)} &= \frac{\Phi_s\big(f(\tilde{\boldsymbol{r}}_i(t^{(z)}))\big) - \Phi_s\big(f(\tilde{\boldsymbol{r}}_i(t^{(z+1)}))\big)}{\Phi_s\big(f(\tilde{\boldsymbol{r}}_i(t^{(z)}))\big)}\,.
\end{aligned} \tag{4.25}$$

Note that in the case of a bent ray entering the geometry, Equation (4.25) will be non-negative ($\because f(\tilde{\boldsymbol{r}}_i(t^{(z)}) > f(\tilde{\boldsymbol{r}}_i(t^{(z+1)})$ and Φ_s is non-negative and monotonically increasing) and as such is equivalent to a maximum with zero.

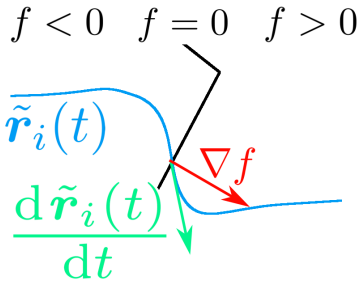


Figure 4.3: Graphical depiction of a bent ray (traveling left to right) exiting an SDF surface. Note that the instantaneous viewing direction and gradient of the SDF must have a positive dot product in order for the bent ray to exit the geometry.

The second case to consider is where a ray exits the geometry as depicted in Figure 4.3. Given that:

$$\nabla f(\tilde{\boldsymbol{r}}_i(t)) \cdot \tfrac{\mathrm{d}\tilde{\boldsymbol{r}}_i(t)}{\mathrm{d}t} > 0\,, \tag{4.26}$$

and that both ϕ_s and Φ_s are non-negative, Equation (4.20) gives that $\rho(\tilde{\boldsymbol{r}}_i(t)) = 0$. Thus the discrete opacity $\alpha_i^{(z)}$ is:

$$\begin{aligned}
\alpha_i^{(z)} &= 1 - \exp\left(-\int_{t^{(z)}}^{t^{(z+1)}} \rho(\tilde{\boldsymbol{r}}_i(t))\,\mathrm{d}t\right) \\
&= 1 - \exp\left(-\int_{t^{(z)}}^{t^{(z+1)}} 0\,\mathrm{d}t\right) \\
&= 0\,.
\end{aligned} \tag{4.27}$$

Since Equation (4.25) will be non-positive when exiting the geometry ($\because f(\tilde{\boldsymbol{r}}_i(t^{(z+1)}) > f(\tilde{\boldsymbol{r}}_i(t^{(z)})$ and Φ_s is non-negative and monotonically increasing), we can write the derived equation for $\alpha_i^{(z)}$ satisfying both cases as:

$$\alpha_i^{(z)} = \max\left\{\frac{\Phi_s\Big(f\big(\tilde{\boldsymbol{r}}_i(t^{(z)})\big)\Big) - \Phi_s\Big(f\big(\tilde{\boldsymbol{r}}_i(t^{(z+1)})\big)\Big)}{\Phi_s\Big(f\big(\tilde{\boldsymbol{r}}_i(t^{(z)})\big)\Big)}, 0\right\}. \tag{4.28}$$

□

4.2.5 Unbiased Nature of our Rendering Method

In this section, we show that our rendering method is unbiased with respect to the surface of the object, i.e. $f\big(\tilde{\boldsymbol{r}}_i(t)\big) = 0$. This proof follows a similar progression to that in the Appendix B of Wang et al. [92]. From Figure 4.2, it can be seen that for a smooth parametric path to intersect the surface, there *must* be a finite region $t \in (t_l, t_r)$ such that $\nabla f\big(\tilde{\boldsymbol{r}}_i(t)\big) \cdot \frac{\mathrm{d}\tilde{\boldsymbol{r}}_i(t)}{\mathrm{d}t} < 0$.

We can re-write the weight as:

$$\begin{aligned}
\omega(\tilde{\boldsymbol{r}}_i, t) &= T(\tilde{\boldsymbol{r}}_i, t)\,\rho(\tilde{\boldsymbol{r}}_i(t)) \\
&= \exp\left(-\int_0^t \rho(\tilde{\boldsymbol{r}}_i(\tau))\,\mathrm{d}\tau\right)\rho(\tilde{\boldsymbol{r}}_i(t)) \\
&= \exp\left(-\int_0^{t_l} \rho(\tilde{\boldsymbol{r}}_i(\tau))\,\mathrm{d}\tau\right)\exp\left(-\int_{t_l}^t \rho(\tilde{\boldsymbol{r}}_i(\tau))\,\mathrm{d}\tau\right)\rho(\tilde{\boldsymbol{r}}_i(t)) \\
&= T(\tilde{\boldsymbol{r}}_i, t_l)\exp\Big(\ln\big[\Phi_s\big(f(\tilde{\boldsymbol{r}}_i(t))\big)\big] - \ln\big[\Phi_s\big(f(\tilde{\boldsymbol{r}}_i(t_l))\big)\big]\Big)\,\rho(\tilde{\boldsymbol{r}}_i(t)) \\
&= T(\tilde{\boldsymbol{r}}_i, t_l)\frac{\cancel{\Phi_s\big(f(\tilde{\boldsymbol{r}}_i(t))\big)}}{\Phi_s\big(f(\tilde{\boldsymbol{r}}_i(t_l))\big)}\frac{\Big[-\nabla f(\tilde{\boldsymbol{r}}_i(t))\cdot\frac{\mathrm{d}\tilde{\boldsymbol{r}}_i(t)}{\mathrm{d}t}\Big]\,\phi_s\big(f(\tilde{\boldsymbol{r}}_i(t))\big)}{\cancel{\Phi_s\big(f(\tilde{\boldsymbol{r}}_i(t))\big)}} \\
\therefore \omega(\tilde{\boldsymbol{r}}_i, t) &= \underbrace{\frac{T(\tilde{\boldsymbol{r}}_i, t_l)}{\Phi_s\big(f(\tilde{\boldsymbol{r}}_i(t_l))\big)}}_{\text{constant}}\underbrace{\Big[-\nabla f(\tilde{\boldsymbol{r}}_i(t))\cdot\tfrac{\mathrm{d}\tilde{\boldsymbol{r}}_i(t)}{\mathrm{d}t}\Big]\,\phi_s\big(f(\tilde{\boldsymbol{r}}_i(t))\big)}_{F(t)}.
\end{aligned} \tag{4.29}$$

Then we can establish that for:

$$F(t) = \underbrace{\Big[-\nabla f(\tilde{\boldsymbol{r}}_i(t))\cdot\tfrac{\mathrm{d}\tilde{\boldsymbol{r}}_i(t)}{\mathrm{d}t}\Big]}_{G(t)}\,\phi_s\big(f(\tilde{\boldsymbol{r}}_i(t))\big), \tag{4.30}$$

$\exists s > 0$ such that $F(t)$ is maximized by $f(\tilde{\boldsymbol{r}}_i(t^*)) = 0$, $t^* \in (t_l, t_r)$. Consider another value $t^\dagger \in (t_l, t_r)$, $t^\dagger \neq t^*$ where $G(t^\dagger) = 1$ is maximum and $G(t^*) = \epsilon$ is minimum for some necessarily non-zero value ϵ. This corresponds to the worst case for the unbiasedness since $0 < G(t) \leq 1$, $\forall t \in (t_l, t_r)$. Then:

$$G(t^*)\,\phi_s\big(f(\tilde{\boldsymbol{r}}_i(t^*))\big) \overset{?}{>} G(t^\dagger)\,\phi_s\big(f(\tilde{\boldsymbol{r}}_i(t^\dagger))\big) \tag{4.31}$$

$$\frac{\phi_s(0)}{\phi_s\big(f(\tilde{\boldsymbol{r}}_i(t^\dagger))\big)} \overset{?}{>} \frac{G(t^\dagger)}{G(t^*)} = \frac{1}{\epsilon}. \tag{4.32}$$

Taking the limit of the left-hand side of Equation (4.32) as s approaches 0 and using the definition of the Logistic PDF ϕ_s:

$$\lim_{s\to 0} \frac{\phi_s(0)}{\phi_s\big(f(\tilde{\boldsymbol{r}}_i(t^\dagger))\big)} = \lim_{s\to 0} \frac{\exp\left(\frac{f(\tilde{\boldsymbol{r}}_i(t^\dagger))}{s}\right)}{4\left(1+\exp\left(-\frac{f(\tilde{\boldsymbol{r}}_i(t^\dagger))}{s}\right)\right)^2} = \infty\,. \tag{4.33}$$

Thus, for every possible ϵ, $\exists s > 0$ such that:

$$\frac{\phi_s(0)}{\phi_s\big(f(\tilde{\boldsymbol{r}}_i(t^\dagger))\big)} > \frac{1}{\epsilon}\,, \tag{4.34}$$

which implies $F(t^*) > F(t^\dagger)$, $\forall t^\dagger \in (t_l, t_r)$, $t^\dagger \neq t^*$. □

4.3 Scene Flow Loss

So far, very large scene deformations remain a challenge for Ub4D since it can create erroneous multiple geometries in the canonical space to best explain the monocular observations. This is a version of the monocular ambiguities discussed in Section 1.1. This is particularly noticeable for scenes containing large translations (see Figure 5.9). To resolve this, we accept an additional input in the form of a coarse and coherent per-frame geometric proxy. From these coarse 3D correspondences, we can compute an estimate of the scene flow, which can then be used to regularize the bending network. This greatly reduces the effect of duplicate geometries in the canonical space as will be shown in the ablation of Section 5.7.2.

Consider a function $\boldsymbol{m}_{i\to j} : \mathbb{R}^3 \to \mathbb{R}^3$ that returns the scene flow from frame i to j at a point in frame i. The scene flow allows us to transform points from a frame i into another frame j as $\boldsymbol{x}_j = \boldsymbol{x}_i + \boldsymbol{m}_{i\to j}(\boldsymbol{x}_i)$. Given that the bending network $\boldsymbol{b}_i$ projects a point $\boldsymbol{x}_i$ in frame space into canonical space resulting in the point $\boldsymbol{x}_c$, it follows:

$$\begin{aligned} \boldsymbol{x}_c &= \boldsymbol{x}_i + \boldsymbol{b}_i(\boldsymbol{x}_i) \\ &= \boldsymbol{x}_j + \boldsymbol{b}_j(\boldsymbol{x}_j) \\ &= \boldsymbol{x}_i + \boldsymbol{m}_{i\to j}(\boldsymbol{x}_i) + \boldsymbol{b}_j\big(\boldsymbol{x}_i + \boldsymbol{m}_{i\to j}(\boldsymbol{x}_i)\big)\,. \end{aligned} \tag{4.35}$$

This expresses the idea that a point in frame i and its corresponding point in frame j determined through the scene flow $\boldsymbol{m}_{i\to j}(\boldsymbol{x}_i)$ should be mapped to the same point $\boldsymbol{x}_c$ in canonical space by the bending network (see Figure 4.4). We can then formulate it as a loss for a set $\mathcal{X}_i$ of points sampled in frame i space:

$$L_{\text{FLO}} = \frac{1}{|\mathcal{X}_i|} \sum_{\boldsymbol{x}_i \in \mathcal{X}_i} ||\boldsymbol{m}_{i\to j}(\boldsymbol{x}_i) + \boldsymbol{b}_j\big(\boldsymbol{x}_i + \boldsymbol{m}_{i\to j}(\boldsymbol{x}_i)\big) - \boldsymbol{b}_i(\boldsymbol{x}_i)||_2^2\,. \tag{4.36}$$

In order to sample the set of points $\mathcal{X}_i$ at which to apply L_{FLO}, we use that frame's proxy geometry as a guide. For each point in $\mathcal{X}_i$, we first uniformly sample a vertex from the coarse geometry, then sample a Gaussian with width λ_1 centered about that vertex. This is done since the vertices are the points from which we extrapolate the scene flow and therefore nearby points have a higher accuracy, as well as having a higher magnitude due to the falloff that will be given in Equation (4.38). A visualization of the extrapolated scene flow is shown in Figure 4.5.

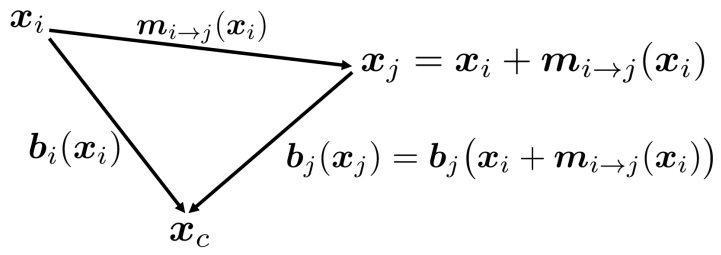


Figure 4.4: Graphical depiction of the relationship between the scene flow from frame i to frame j (i.e. $\boldsymbol{m}_{i\to j}(\boldsymbol{x}_i)$) with the bending network projecting both points to the same canonical position $\boldsymbol{x}_c$.

The scene flow from the geometric proxies can only be directly computed on the proxy surface. However, our implicit surface representation can potentially be evaluated at any point in $\mathbb{R}^3$. Thus, we extrapolate this scene flow to any point in $\mathbb{R}^3$ with a convex combination over the vertices using a kernel function depending on the distance to the vertices inspired by the spatial weighting approach in bilateral filtering [83]:

$$\boldsymbol{m}'_{i\to j}(\boldsymbol{x}_i) = \frac{\sum_{k=1}^{N_v} w_{\lambda_1}\big(||\boldsymbol{x}_i - \boldsymbol{v}_i^{(k)}||_2\big)\,\big(\boldsymbol{v}_j^{(k)} - \boldsymbol{v}_i^{(k)}\big)}{\sum_{k=1}^{N_v} w_{\lambda_1}\big(||\boldsymbol{x}_i - \boldsymbol{v}_i^{(k)}||_2\big)}\,, \tag{4.37}$$

where $w_{\lambda_1}(x) = e^{-\lambda_1 x^2}$ is a kernel function with λ_1 as a scale parameter affecting the weighting of vertex flow estimates. Additionally, we add an attenuation term, so that the scene flow falls off as the distance to the nearest vertex increases:

$$\boldsymbol{m}_{i\to j}(\boldsymbol{x}_i) = w_{\lambda_2}\Big(\min_{k=1}^{N_v} ||\boldsymbol{x}_i - \boldsymbol{v}_i^{(k)}||_2\Big)\,\boldsymbol{m}'_{i\to j}(\boldsymbol{x}_i)\,, \tag{4.38}$$

where $w_{\lambda_2}(x) = e^{-\lambda_2 x^2}$ is a kernel function with λ_2 as a scale parameter defining the rate of the falloff. This falloff assumes that we are interested in modeling only the object for which we have the coarse geometric proxy.

4.4 Summary of Loss Functions and Regularizers

Here we summarize the loss functions and regularizers of Ub4D. Our total combined loss function is:

$$L = L_{\text{COL}} + \gamma_{\text{SEG}}\, L_{\text{SEG}} + \gamma_{\text{EIK}}\, L_{\text{EIK}} + \gamma_{\text{NBR}}\, L_{\text{NBR}} + \gamma_{\text{DIV}}\, L_{\text{DIV}} + \gamma_{\text{FLO}}\, L_{\text{FLO}}\,, \tag{4.39}$$

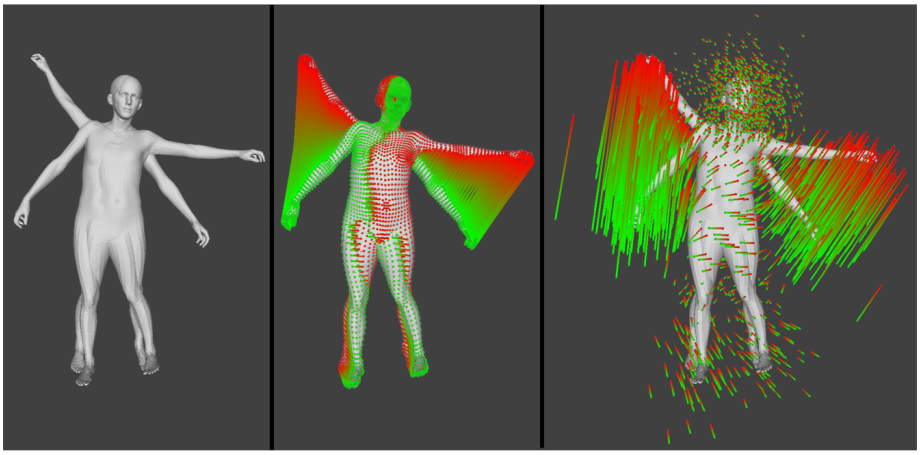
where γ are weights that are defined relative to the color loss. We perform an ablation study in Section 5.7.1 to validate this total loss.

Color Loss L_{COL} defined in Equation (4.17) is a reconstruction loss measuring the model's ability to render the correct color for each ray.

Segmentation Loss L_{SEG} defined in Equation (4.18) is a reconstruction loss measuring the model's ability to localize the object. Typical values for γ_{SEG} are in the range $[0.25, 1.5]$.

Eikonal Loss L_{EIK} defined in Equation (4.19) is a regularizer that enforces that the SDF network f remains an SDF. Typical values for γ_{EIK} are in the range $[0.5, 1.0]$.

Neighbour Offset Loss L_{NBR} defined in Equation (4.3) is a regularizer that enforces neighbouring frames to have similar deformations. This can also be seen as a temporal



Two Geometric Proxies | Vertex Correspondences | Estimated Scene Flow at Sampled Positions

Figure 4.5: Visualization of the extrapolated scene flow at sampled points for two manually selected frames on right. Left shows the geometric proxies for the two frames and the middle shows the vertex correspondences. The first frame points are colored in green and the second frame points are colored in red. The color gradient is only for visualization purposes. Note how the estimated scene flow is similar to the vertex correspondences and that the sampled positions (green points) focus around the first frame's proxy.

smoothness regularizer since it penalizes large temporal derivatives. Typical values for γ_{NBR} are in the range $[1 \times 10^3, 50 \times 10^3]$.

Divergence Loss L_{DIV} defined in Equation (4.4) is a regularizer that enforces the bending network to have low divergence. This penalizes the model for collapsing or expanding space and can be seen as an As-Rigid-As-Possible (ARAP) [78] prior. Typical values for γ_{DIV} are in the range $[1, 1 \times 10^3]$.

Scene Flow Loss L_{FLO} defined in Equation (4.36), and discussed at length in Section 4.3, is a novel loss expressing the concept that points corresponding by scene flow should project to the same canonical point. Typical values for γ_{FLO} are in the range $[0, 100]$.

4.5 Surface Extraction

After implicitly learning the surface with the losses described above, we extract explicit geometry for each frame. To do this, we apply the marching cubes algorithm [45] as described in Section 3.4 framewise. For points sampled in frame i, we transform them from the frame space into the canonical space, i.e. $\boldsymbol{x}_c = \boldsymbol{x}_i + \boldsymbol{b}_i(\boldsymbol{x}_i)$ where $\boldsymbol{x}_i$ is a point sampled for marching cubes and $\boldsymbol{x}_c$ is the canonical space point at which we then evaluate the SDF f. We restrict the selection of frame march points to the camera frustum of the given frame since any space not seen in that frame is unconstrained by our reconstruction losses and can contain aberrant geometry.

4.6 Implementation

We base our implementation on the codebase of Wang et al. [92], which is implemented in PyTorch [55]. Our method consists of 3 MLP networks: *Bending*, *SDF*, and *Rendering* corresponding to the $\boldsymbol{b}_i$, f, and $\boldsymbol{c}$ functions, respectively. We provide a diagram showing the networks in Figure 4.6 and give additional details in Table 4.1. We also configure the starting weights of the *SDF* network using the geometric initialization method of Atzmon and Lipman [3].

At the start of training we follow Tretschk et al. [86] and initialize the latent codes with zeros. Section 5.8 shows that this zero initialization is important for semantically meaningful latent codes and improving the method performance. Additionally, the last layer of the *Bending* network has its weights initialized with zeros. This starts Ub4D with straight rays and introduces bending over time as needed.

For each iteration during training, we sample 512 pixels uniformly over the image for which to fire rays. We sample 64 positions along each straight ray, jitter these samples, and then importance sample 64 additional positions based on the SDF values. Jittering is performed along the straight ray for the initial 64 samples.

Network	*Activation*	*Weight Normalization* [66]
Bending	ReLU	No
SDF	SoftPlus ($\beta = 100$)	Yes
Rendering	ReLU	Yes

Table 4.1: Network parameters of Ub4D.

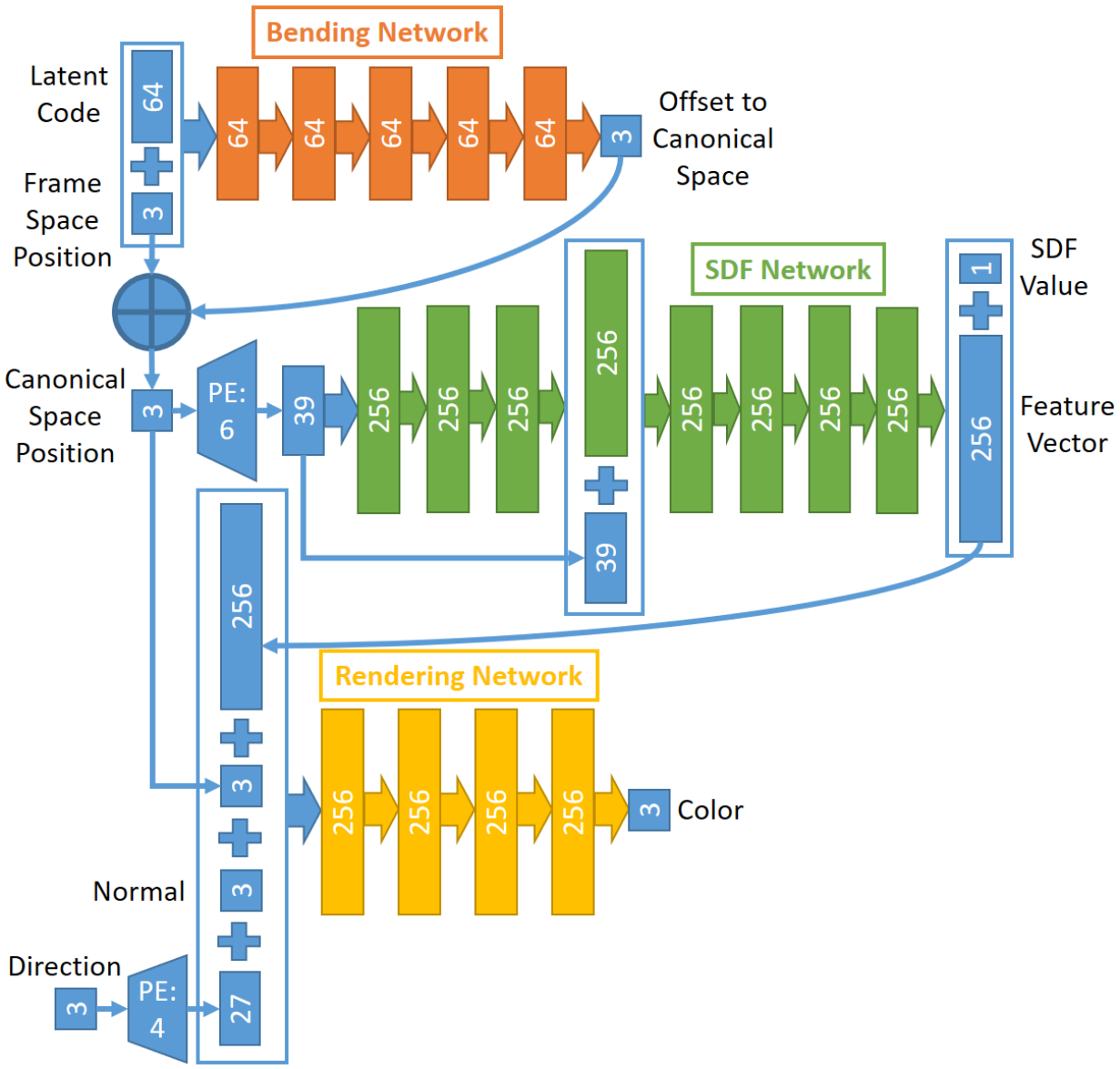


Figure 4.6: Network diagram of Ub4D. PE denotes Positional Encoding (PE) [48] with the given number of additional frequences. Otherwise, the numbers show the width of the vectors or fully connected layer. The addition node performs element-wise addition while the plus blocks represent vector concatenation.

Chapter 5
Experiments

The following experiments validate our design choices and demonstrate improvement over the state-of-the-art. We first regress Ub4D on a static scene in Section 5.1. Then in Section 5.2 we introduce our new dataset on which we conduct the subsequent experiments. Section 5.3 compares our Signed Distance Function (SDF) implicit scene representation to the common alternative: a density representation. Next, Section 5.4 compares Ub4D with competing methods for monocular 4D reconstruction. We then provide additional qualitative results in Section 5.5 and show the learned canonical geometry in Section 5.6. Section 5.7 investigates our design decisions by ablation. Finally, in Section 5.8 we analyze the per-frame latent codes of the *Bending* network.

5.1 Static Scene Regression

As a first experiment, we run Ub4D on a stationary and non-deforming object from the DTU dataset [29]. We compare against NeuS [92] since they demonstrate state-of-the-art results for multi-view reconstruction of static objects. The colored output geometry of Ub4D for two frames and NeuS [92] are shown in Figure 5.1. Ub4D produces geometry that is visually indistinguishable from the NeuS [92] reconstruction and that does not deform over the sequence. This is exactly as expected since the object is perfectly stationary and non-deforming.

This demonstrates the ability of Ub4D to handle scenes with no deformation. As discussed in Section 4.6, at the start of training we set the bending network's final layer weights to be zero, which means the initial output of $\boldsymbol{b}_i$ is the zero vector. This starts learning with straight rays that should only be bent if it is necessary to model object deformations.

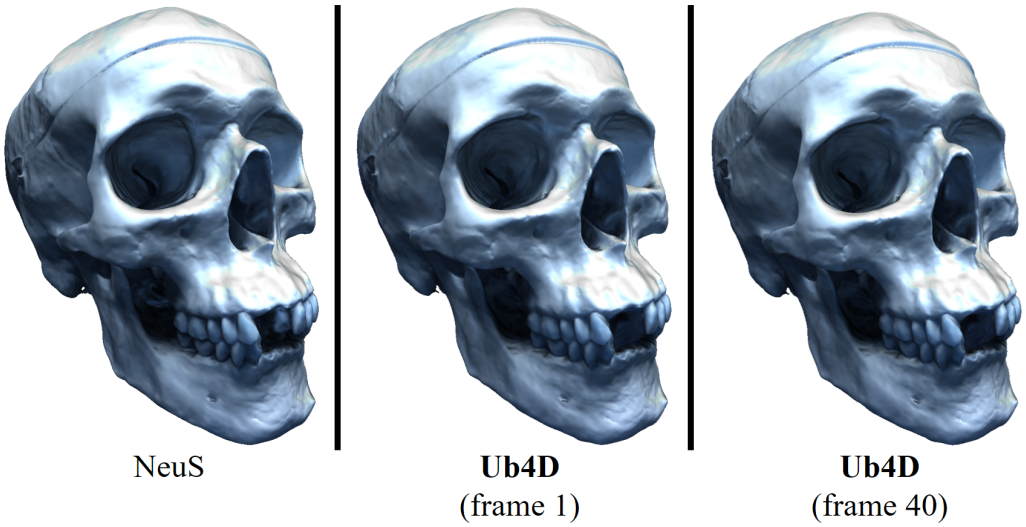


Figure 5.1: Static scene regression on scan 65 from DTU dataset [29]. Since our method takes segmentations as input, we also provide the segmentation to NeuS [92]. Note that Ub4D produces qualitatively equivalent results to NeuS and that the object does not deform over the sequence as expected for stationary and non-deforming objects.

5.2 Novel Dataset

We now introduce our novel dataset used in evaluating our method. Ub4D differs from other monocular 4D reconstruction techniques in that it is a scanning approach, similar to Structure from Motion (SfM), requiring sufficient camera movement to observe the target from multiple view points during its deformations. One can see this as needing to build up a canonical representation. While a full 360 degree path would be ideal, we find that 180 degree viewing range is sufficient to produce good reconstructions.

To meet this requirement on camera movement and provide different scenarios under which to evaluate Ub4D, we introduce five scenes as a new dataset for monocular 4D reconstruction. These five scenes are: *Cactus*, *RootTrans*, *Lego*, *Humanoid*, and *RealCactus*. They are summarized in Table 5.1 and sample frames are shown in Figure 5.2. This provides three synthetic scenes, of which two have proxies and Ground Truth (GT) geometry, and two real world scenes.

Synthetic Scenes The synthetic scenes are *Cactus*, *RootTrans*, and *Lego*. These are all created in Blender [16] which allows us to recover the GT camera parameters and segmentations. For *Cactus* and *RootTrans*, we also extract the GT geometry as meshes. While it is possible to extract GT geometry for *Lego*, the animation complexity and detail of the model make it prohibitively expensive. *Cactus* and *RootTrans* are created by animating rigid objects captured with COLMAP [71, 72], while *Lego* is created by animating a model provided by Mildenhall et al. [48] [1]. While the *Lego* scene includes the first 100 frames depicting a rigid object, neither *Cactus* nor *RootTrans* include any frames with a non-deforming object. This shows that Ub4D does not require access to a rigid sub-sequence.

[1] Released under CC-BY-3.0 and modifications are made to create the *Lego* dataset. Original model was created by Blend Swap user Heinzelnisse.

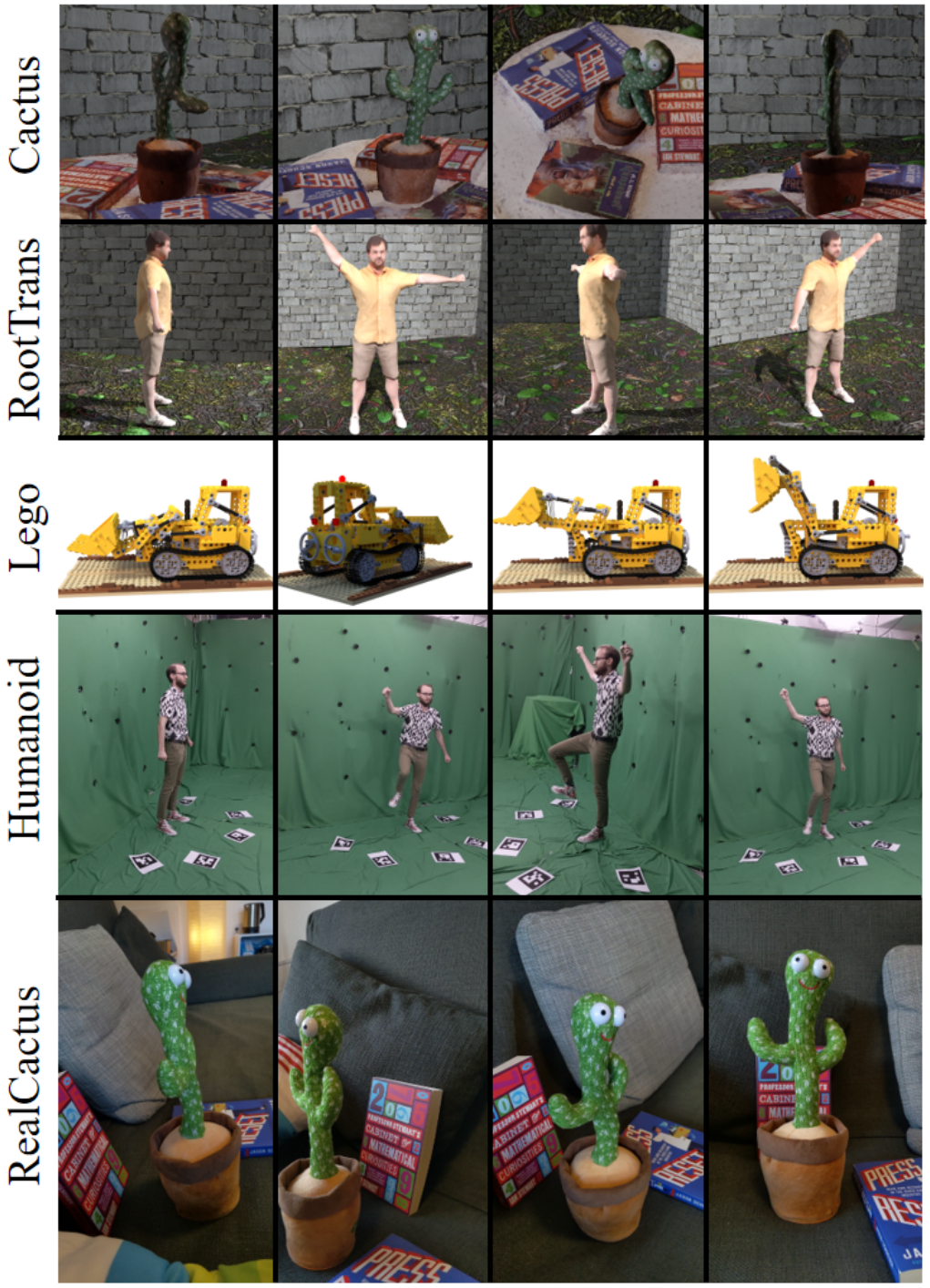


Figure 5.2: Sample frames for each scene in our novel dataset. Further details about each scene are contained in Table 5.1. Note that images are not to scale.

Geometric proxies are computed for *Cactus* and *RootTrans* in very different manners. The *Cactus* scene uses geometric proxies computed by heavily decimating the GT meshes. While the use of even decimated GT meshes is not optimal, it allows for a sanity check: if performance does not increase with access to decimated GT meshes, then estimated meshes would not provide a benefit. The *RootTrans* scene uses geometric proxies computed by fitting a SMPL model to each frame [6]. This provides a much fairer comparison to evaluate the impact of our novel scene flow loss. Sample images and proxies, as well as GT geometry, from both *Cactus* and *RootTrans* are shown in Figure 5.4.

We use the *Lego* scene in Section 5.3 to compare geometry representation options. The *Cactus* and *RootTrans* scene are used for comparing with competing methods in Section 5.4 and for our ablation studies in Section 5.7.

Real World Scenes The real world scenes are *Humanoid* and *RealCactus*. Both scenes are captured with a Google Pixel 2 smartphone camera, illustrating that our method can operate on input from consumer-grade capture equipment. In order to estimate camera parameters, we employ the freely available and open-source COLMAP [71, 72]. For determining segmentations, we manually annotate a handful of frames ($\sim$10) and then train a segmentation network, based on the U-Net architecture [63], to predict segmentations from RGB images. We use the *Humanoid* and *RealCactus* scenes in Section 5.5 to demonstrate generalization of Ub4D to real-world scenes. While the first approximately 100 frames of *RealCactus* depict a non-deforming object, *Humanoid* includes no frames with a rigid object. This shows that our method does not require a rigid sub-sequence to produce plausible results on real-world input.

Name	*Creation*	*Frames*	*Resolution*	*Geometric Proxies?*	*GT?*
Cactus	Blender [16]	150	1024×1024	Yes (decimated GT)	Yes
RootTrans	Blender [16]	150	1024×1024	Yes (SMPL [6])	Yes
Lego	Blender [16]	150	800×800	No	No
Humanoid	Real World	171	960×1280	No	No
RealCactus	Real World	150	1080×1920	No	No

Table 5.1: Summary of the datasets introduced in this work. GT indicates access to GT geometry in the form of per-frame meshes. Synthetic scenes are above the dashed line, real-world captures below.

5.3 Geometry Representation Comparison

In this section, we use the *Lego* scene to justify the choice of using an SDF to represent our scene rather than a density network. Previous works focusing on static surface extraction have used SDFs as their implicit representation [92, 97], whereas works focusing on novel view synthesis of both static and dynamic scenes have used density networks [48, 54, 86]. We compare to NR-NeRF [86] in Figure 5.3 to show the benefits of SDF scene representation for surface reconstruction, as well as motivate the choice to use a neighbour penalization of the bending network. The surface of the density network is computed by applying marching cubes [45] as described in Section 3.4 for an empirically determined suitable threshold. Note the noisy nature of the surface extracted from NR-NeRF [86] and the inability to model large deformations. Ub4D produces more accurate geometry, particularly under large deformations.

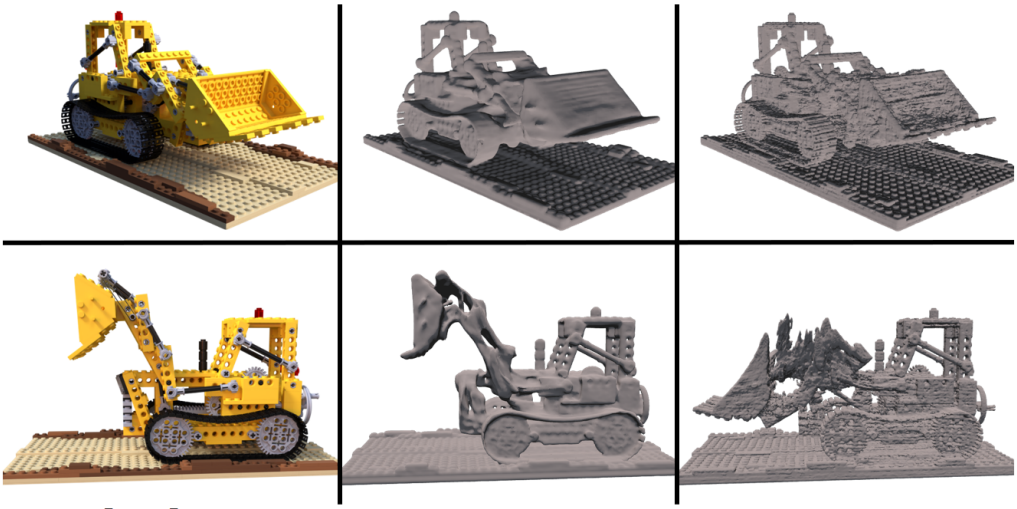


Figure 5.3: Comparison of Ub4D using an SDF scene representation (without scene flow loss) to a density-based scene representation, called NR-NeRF [86]. To generate surfaces for NR-NeRF, we apply marching cubes [45] with a threshold of 50. The density-based representation leads to an overall noisier surface compared to our approach. Further, NR-NeRF fails to model the large deformation in the bottom row due to its absolute length regularization of bending. Instead, we penalize neighbouring frame offsets, which allows to accurately reconstruct such larger deformations.

5.4 Method Comparison

We compare with state-of-the-art monocular 4D reconstruction methods from the literature. These methods are LASR [95], N-NRSfM [75], and DDD [99] representing very different approaches to solving the problem.

LASR [95] This method is a template-free approach that deforms a sphere in order to match the observations. It also regresses an articulated bone structure to explain the deformations. Despite being template-free, it does assume a genus-0 representation. This is a very strong assumption (see Figure 5.3 for an example of a high genus object), which Ub4D does not make. LASR operates on input monocular RGB images and segmentations. It does not require camera parameters.

N-NRSfM [75] This method is a Non-Rigid Structure-from-Motion (NRSfM) approach that uses a structured latent representation with an auto-decoder to deform a mean shape. This neural extension of NRSfM only produces a deformed plane, whereas Ub4D reconstructs the complete watertight object geometry. N-NRSfM operates on input Multi-Frame Optical Flow (MFOF) that is computed from our monocular RGB images and segmentations using Ansari et al. [2].

DDD [99] This method is a template-based approach that deforms the input template to match the observations. It optimizes the vertex reprojections to satisfy an energy formulation, which includes regularizers such as an As-Rigid-As-Possible (ARAP) term and a temporal smoothness term. DDD assumes a fixed camera and models the object motion as camera-relative.

Our primary metric is the Chamfer Distance (CD) between the reconstruction and the ground truth. For N-NRSfM and Ub4D, we also report the Hausdorff Distance (HD) since the reconstruction for this method is only a planar surface which can be unfairly penalized by the back-face of the ground truth. We present both CD and HD metrics in Section 3.3. A quantitative method comparison using these metrics is shown in Table 5.2. In order to fairly compare all methods, we apply Iterative Closest Point (ICP) [5] in order to align the output to the ground truth. Since Ub4D also takes camera parameters as input, this operation is not necessary for our reconstruction; however, we also report the metrics after apply ICP for completeness. Note that Ub4D quantitatively outperforms these other methods.

Scene	*Method*	*CD* (*HD*) (↓)
Cactus	LASR [95]	20.23
	N-NRSfM [75]	102.00 (5.74)
	DDD [99]	34.71
	Ub4D (ours)	**3.06** (2.42)
	Ub4D after ICP	2.71 (2.24)
RootTrans	LASR [95]	0.39
	N-NRSfM [75]	0.38 (0.09)
	DDD [99]	0.26
	Ub4D (ours)	**0.23** (0.14)
	Ub4D after ICP	0.03 (0.02)

Table 5.2: Quantitative comparison to previous work. We report the Chamfer Distance (CD) between the ground truth object geometry and the respective reconstructions averaged over the sequence. Since N-NRSfM [75] produces only a planar surface (not a watertight mesh), we additionally report the Hausdorff Distance (HD) averaged over the sequence for it and our method. Lower is better for both CD and HD. Note that we quantitatively outperform the previous work.

Figure 5.4 gives a qualitative comparison of the methods. Note that all other methods demonstrate a clear tendency to overfit to the particular input view. Ub4D is the only method that is able to capture medium scale details like the rim of the pot in *Cactus* and the end of the character's shorts in *RootTrans*.

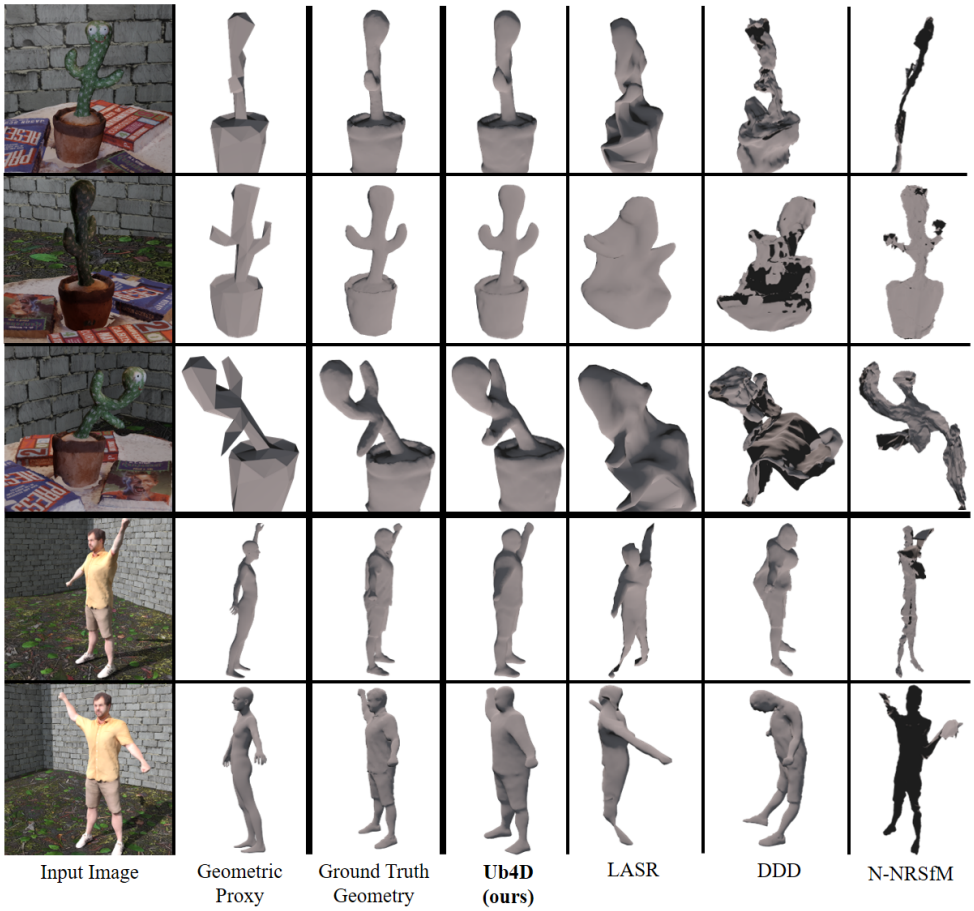


Figure 5.4: Qualitative comparison for select frames of our synthetic sequences rendered from novel views. Note that competing methods struggle with reconstructing the dense and deforming surface, while our method captures the large scale deformations as well as medium scale details.

5.5 Qualitative Results

Further qualitative results on our synthetic scenes are shown in Figure 5.5 and Figure 5.6 for the *Cactus* and *RootTrans* scenes. In addition to the colored reconstruction, we also provide a visualization of the error where dark blue is zero error and red is any error over half-way between the mean and max error, taken over the entire sequence. This provides a stable coloring that gives a sense of the highest errors throughout the entire sequence. Note that the majority of high error regions occur due to monocular ambiguity as discussed in Section 1.1.

In particular, rows 1 and 3 of Figure 5.5 show this effect. Row 1 demonstrates a very convincing input view reconstruction; however, the error coloring and novel view show that the method has created an erroneous large spur on the right arm. Row 3 shows that the method vertically misplaces the left arm due to the monocular ambiguity. We can also see the monocular ambiguity resulting in a high error region due to misplacing the character's right arm along the depth channel in row 4 of Figure 5.6.

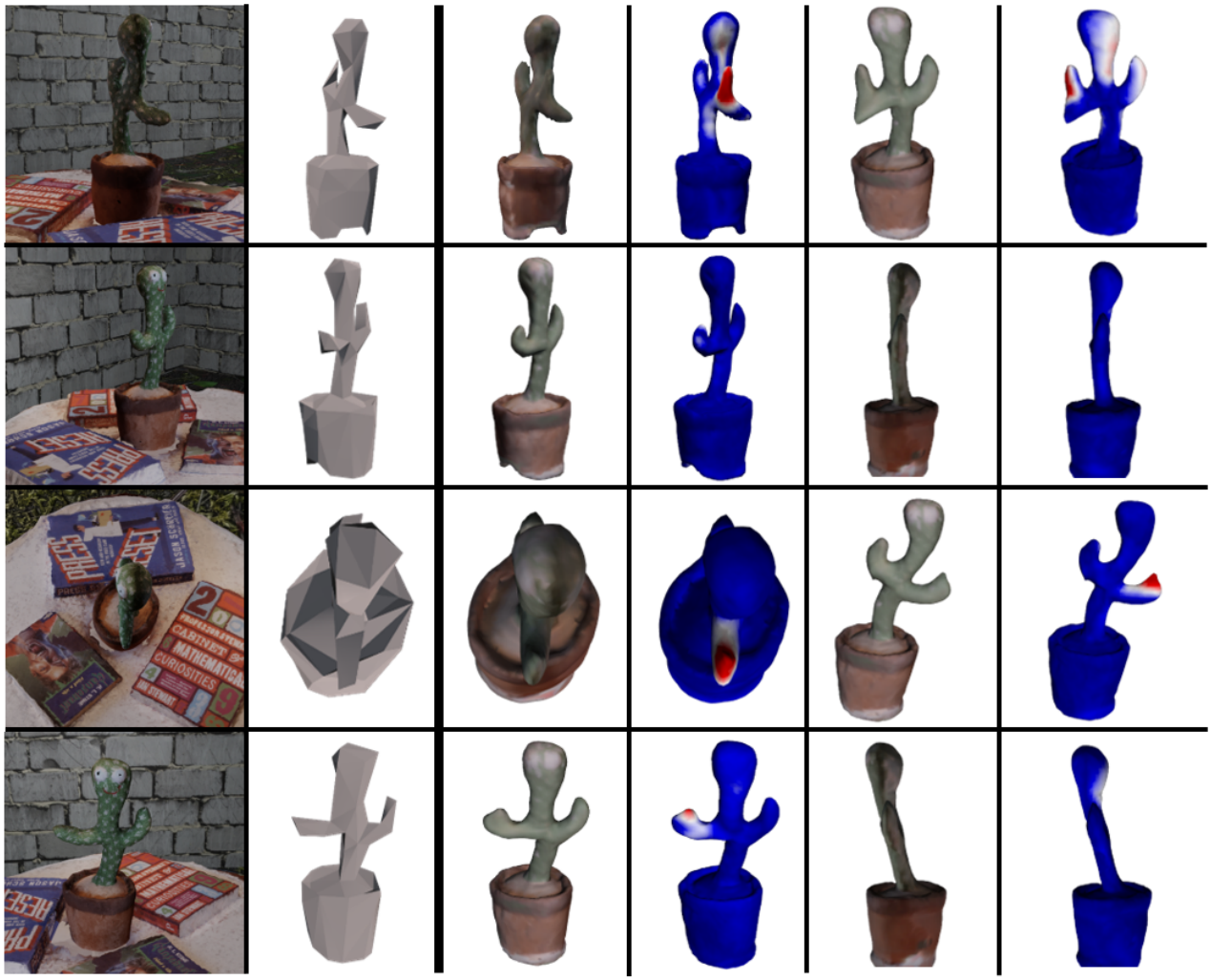


Figure 5.5: Results for our *Cactus* sequence. We include error coloring where blue is zero error and red is high error, relative to the entire sequence.

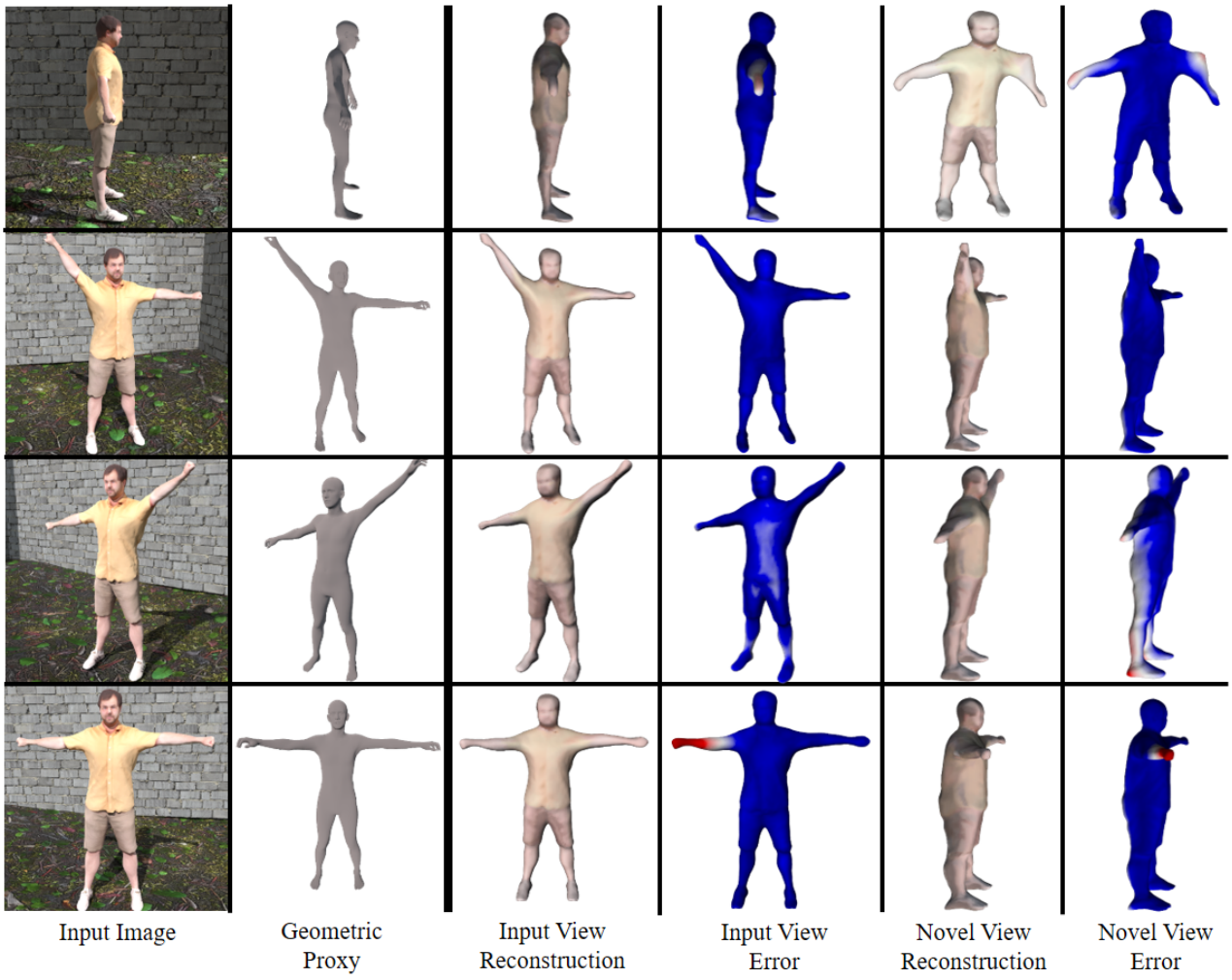


Figure 5.6: Results for our *RootTrans* sequence. We include error coloring where blue is zero error and red is high error, relative to the entire sequence.

Figure 5.7 shows results on the real-world scenes: *RealCactus* and *Humanoid*. Note that our reconstruction overlays accurately on the input image and also produces geometry that looks plausible from a novel view. Neither of these sequences provide geometric proxies and our reconstruction is still high quality for motions with large deformations.

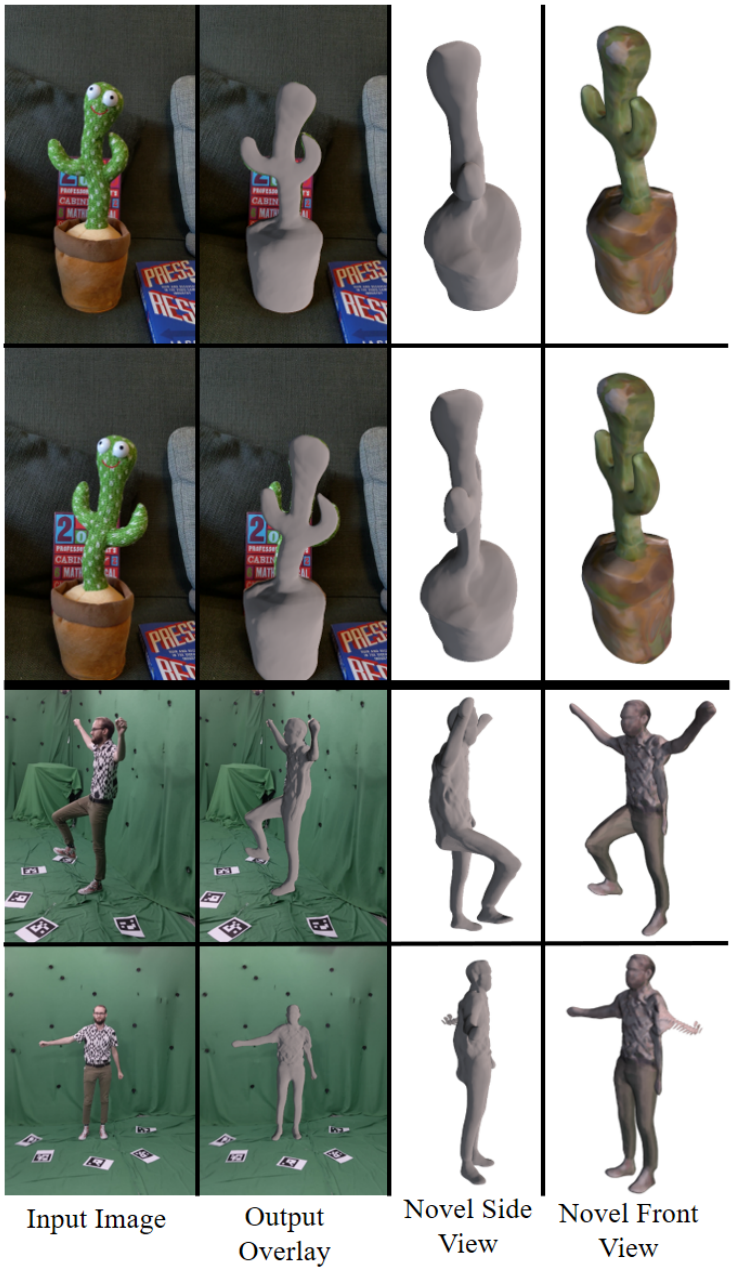


Figure 5.7: Qualitative results of Ub4D on *RealCactus* and *Humanoid* exhibiting large deformations. No 3D geometric proxies are used in these experiments. Note that the recovered geometry nicely overlays onto the input image but also looks plausible from a novel 3D viewpoint.

5.6 Canonical Space Visualization

In addition to extracting the per-frame geometry, we can also extract the canonical representation. To do this, we simply use the procedure described in Section 3.4 to the SDF network without applying the bending network. Figure 5.8 shows the canonical space for each of our introduced sequences. This suggests that Ub4D can also be seen as jointly constructing an object template in the canonical space and non-rigidly animating that template with the bending network. Note that sometimes the canonical space does not directly correspond to the object itself, as in the case of *Humanoid*, and this is discussed in Section 6.1.

Figure 5.8: Visualization of the canonical spaces for each sequence. Note how similar they appear to a template of the object despite there being no explicit requirement that the canonical space be meaningful.

5.7 Ablation Studies

We validate our major design decisions through ablation studies on the *Cactus* and *RootTrans* sequences. In Section 5.7.1, we examine the quantitative effect of ablating our loss formulation. Our novel scene flow loss is examined in Section 5.7.2 with a qualitative demonstration of its importance.

5.7.1 Loss Ablations

Our full supervision (Equation (4.39) in Section 4.4) consists of six loss terms: L_{COL}, L_{SEG}, L_{EIK}, L_{FLO}, L_{NBR}, and L_{DIV}. We compare the full method to removing the terms: 1) L_{FLO}, which is our novel flow loss, 2) L_{EIK}, which directly regularizes the *SDF* network and indirectly regularizes the bending network, and 3) L_{FLO}, L_{NBR}, and L_{DIV}, which are all direct bending network regularizers. We report the quantitative results for these loss ablations with both HD and CD metrics (see Section 3.3) in Table 5.3. Most importantly, the full combination of losses provides the best result validating the contribution of each term.

Scene	*Method*	*HD* (↓)	*CD* (↓)
Cactus	w/o L_{FLO}	4.67 †	8.32 †
	w/o L_{EIK}	3.58	5.47
	w/o L_{FLO}, L_{NBR}, L_{DIV}	3.27 †	5.34 †
	Ub4D (Ours)	**2.42**	**3.06**
RootTrans	w/o L_{FLO}	32.39	60.25
	w/o L_{EIK}	0.16	0.29
	w/o L_{FLO}, L_{NBR}, L_{DIV}	2.20 †	3.83 †
	Ub4D (Ours)	**0.14**	**0.23**

Table 5.3: Quantitative ablation study. We report the Chamfer distance (CD) between the ground-truth scene geometry and the respective reconstructions, as well as the Hausdorff distance (HD) from reconstruction to ground truth, both averaged over the sequence. † denotes that some frames for this ablation do not produce any geometry and we compute the average without these frames. Lower is better. Note that our full method provides the best result in both scenes and for both metrics, which validates the proposed combination of losses.

Concerning 1), our flow loss especially helps for the large root translation and arm motion of the *RootTrans* sequence. Without using this loss, multiple different geometries are synthesized, which fit the reconstruction losses. Then, the bending network can "switch" between the different copies throughout the sequence. This results in overfitting to the camera pose and allows exploiting monocular depth ambiguities to generate geometry that is not seen in other views. Section 5.7.2 demonstrates this in detail.

Regarding 2), we found that not using L_{EIK} leads to overall noisier surfaces and thus the quality is reduced. Finally concerning 3), without any explicit regularization of the bending network, the deformations can be almost arbitrary again leading to overfitting to individual frames by violating 3D consistency resulting in a reduced accuracy. We even observed that the network was not able to produce any geometry for some frames, which further validates the necessity of explicit regularization of the bending network.

5.7.2 Scene Flow Loss Ablation

A key problem unique to our formulation, i.e. specifically when deforming to a canonical space with given camera parameters, is the duplication of geometry in the canonical space. While such duplication would not be a problem if it aided the reconstruction, the reality is that this severely reduces the reconstruction quality (see e.g. Table 5.3). The reason for this reduced quality is that the duplicated geometry typically exploits monocular ambiguities resulting in the reconstruction being incorrectly placed along the depth channel. This causes inconsistent shapes and large single frame discontinuities in reconstruction position.

Figure 5.9 demonstrates this problem and shows that our novel scene flow loss resolves this ambiguity. Column 3 (without scene flow loss) shows three different canonical copies in rows 1, 2, and 3 being used to satisfy the reconstruction loss without the scene flow loss. Note that our frustum culling from Section 4.5 removes the other copies when they would be outside the camera view. While L_{NBR} penalizes this, it can easily enter a local minimum that uses large single frame deformations to "switch" which canonical copy lies in the camera frustum. Our geometric proxy guides the bending network to follow the scene motion resulting in a single consistent copy with a correct scale.

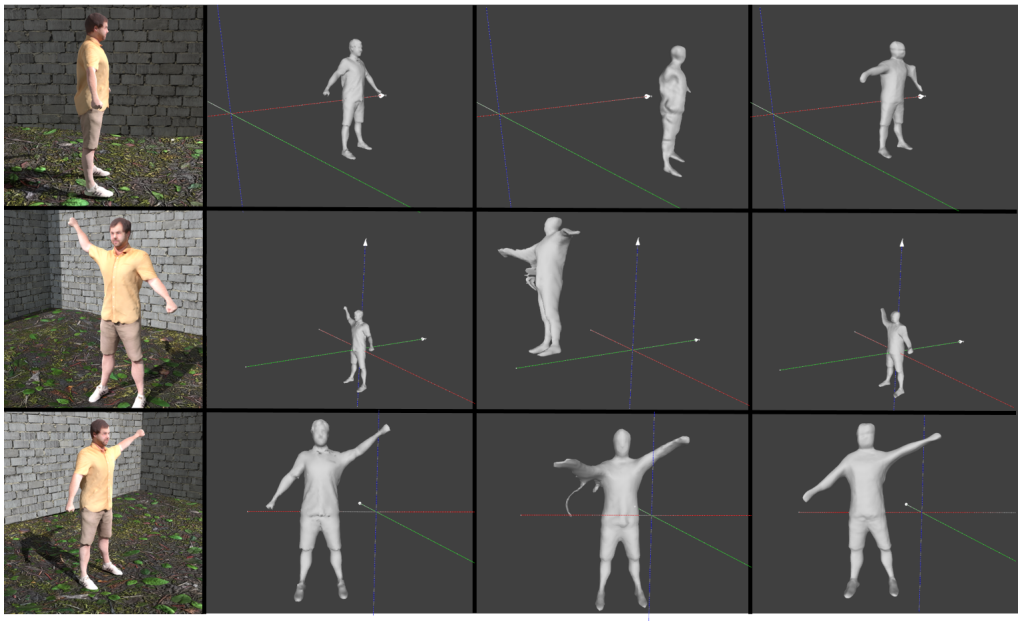


Figure 5.9: Qualitative ablation of the scene flow loss (L_{FLO}) on the *RootTrans* sequence. Note that without the proposed scene flow loss, the network can produce multiple distinct copies of the character at different scales by exploiting the monocular depth ambiguity. The scene flow loss prevents such behaviour, and therefore our method does not overfit to individual frames leading to a single, consistent moving 3D geometry.

5.8 Per-Frame Latent Code Analysis

In Ub4D, the entirety of the model's understanding of time is encoded into a per-frame latent code provided to the bending network (Section 4.1). Initializing these latent codes with zeros (Section 4.6) gives our latent space a valuable property: a smooth, semantically meaningful latent representation. Demonstrating such a latent representation allows us to interpolate latent codes for certain applications, e.g. temporal super-resolution. It also opens the door for employing such deformation models using latent codes to analyze motion (e.g. periodicity detection, metrically comparing deformation states).

To validate the semantic meaning of our latent representation we perform Principal Component Analysis (PCA) [56] on the 64 dimensional learned latent codes. The results are shown in Figure 5.10. Note that even though the latent space is never directly constrained in Ub4D, neighbouring frames (i.e. similar colors in Figure 5.10) tend to be nearby.

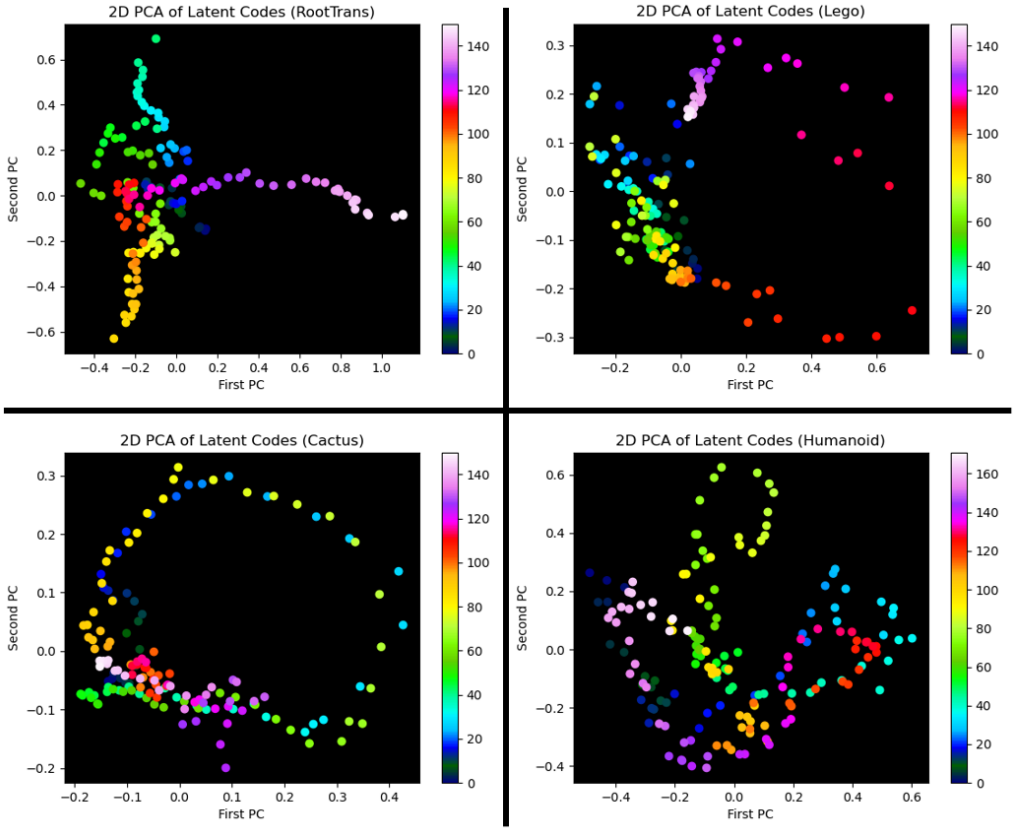


Figure 5.10: 2D PCA of learned 64D latent codes for the *RootTrans*, *Cactus*, *Lego*, and *Humanoid* scenes. The first two Principal Components (PCs) explain 38%, 32%, 25%, and 24% of the variance, respectively. The colors correspond to frames. Note how similar colors are nearby.

We wish to compare against the standard latent code initialization approach: random Gaussian initialization. However, the same concept of performing PCA [56] does not

suffice. This is because PCA uses directions of maximum variance and randomly initialized latent codes could structure themselves "inside" of the variance. While a more complex dimensional reduction technique (e.g. t-SNE [88]) could yield results, a failure to visualize a meaningful structure would not definitively show that such a structure does not exist. Therefore, we use a reduced latent code dimension allowing visualization without dimensional projection.

Taking the *Cactus* scene, we train using 2D latent codes: once initializing with zeroes as proposed in Tretschk et al. [86] and once initializing with random Gaussian samples. We show the resulting learned latent codes in Figure 5.11. Note how spatially coherent the zero-initialized latent codes become during training, whereas the random Gaussian initialized latent codes do not have this property. Observing the particular structure of the zero-initialized case and slight clustering of similar frames in the random Gaussian initialization, one could imagine these latent codes as charged molecules, with similar states attracting and differing states repelling, resulting in a particular fold.

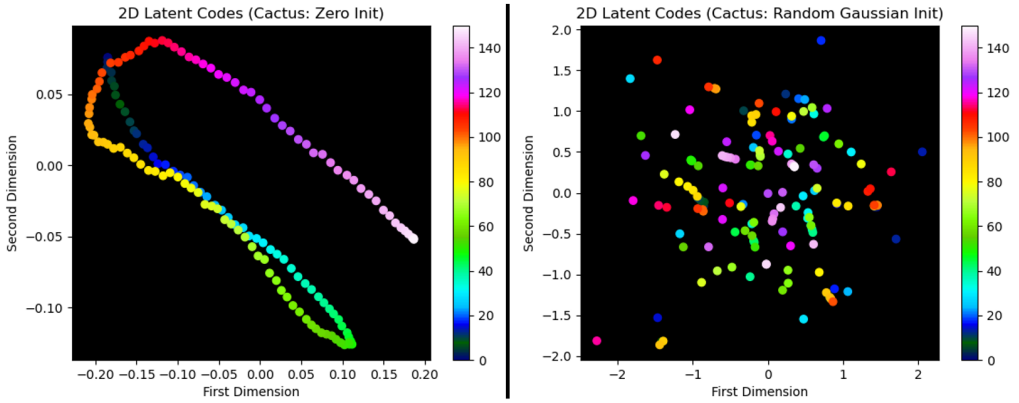


Figure 5.11: Learned 2D latent codes for the *Cactus* scene showing the latent code provided to the network without any dimensional projection. We initialize with zeroes on the left and random Gaussian samples on the right. The colors correspond to frames. Note how similar colors are nearby for zero-initialized latent codes, whereas the random Gaussian initialization does not give rise to such a property.

Comparing metrics for the 2D latent codes between zero initialization versus random initialization gives Chamfer Distances (CDs) of 2.37 and 4.56, respectively. This shows a clear quantitative benefit to zero initialization of the latent codes in the 2D case. The pattern also exists for 64D latent codes with CDs of 3.06 for zero-initialized and 11.43 for random-initialized. However, note that the CD is actually *better* when using 2D latent codes than 64D in both cases. The optimal latent code dimensionality is likely some function of scene complexity and this suggests that it is closer to 2 dimensions rather than 64. More interesting and general would be to determine some way of computing that optimal latent dimensionality from an analysis of structure in the higher dimensional learned latent codes.

Chapter 6
Discussion

Ub4D introduces a new approach to monocular 4D reconstruction through the use of Implicit Scene Representations (ISRs). Despite the significance and novel technical insights of this work, there still remains much to be done in the future. We first present some limitations of the current approach in Section 6.1. Then, Section 6.2 discusses the potential for extracting 3D correspondences and Section 6.3 shows the unexplored ability to generate entirely new geometry with novel latent codes. We investigate runtime improvements for Ub4D in Section 6.4. Finally, Section 6.5 explores future work into the relationship between volumetric ISRs, Partial Differential Equations (PDEs), and the Fourier transform.

6.1 Limitations

One limitation of Ub4D is that errors in the geometric proxy can adversely impact our reconstruction. Our scene flow loss is designed such that it does not require highly accurate correspondences to allow us to handle large deformations and prevent multiple canonical copies; however, we still inherit errors from the geometric proxy. Figure 6.1 illustrates this for a geometric proxy that is offset from the ground truth which results in the region with a high error on our reconstruction. Note that our method still results in a decreased Chamfer distance for this frame compared to the geometric proxy (0.76 vs 0.92) despite this failure. This indicates that Ub4D improves on the geometric proxy through its use of image observations.

Another limitation is that acquiring a geometric proxy may limit application if results without the scene flow loss are not satisfactory. Figure 6.2 shows one example of an issue encountered without the scene flow loss. In this case, an additional appendage is used to satisfy the reconstruction losses while not grossly violating the other regularizers of our method. It is possible that the use of image space keypoint matches could be used in order to penalize such a duplication without requiring a complete 3D geometric proxy.

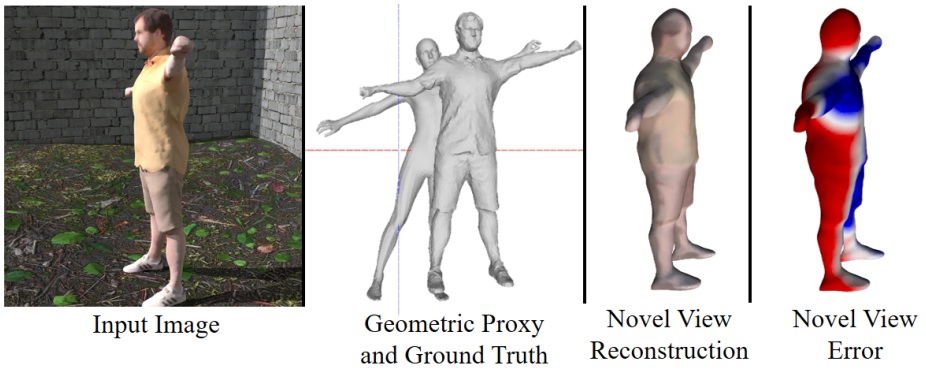


Figure 6.1: Impact of a significant geometric proxy error. Red regions have a higher Chamfer distance to the ground truth.

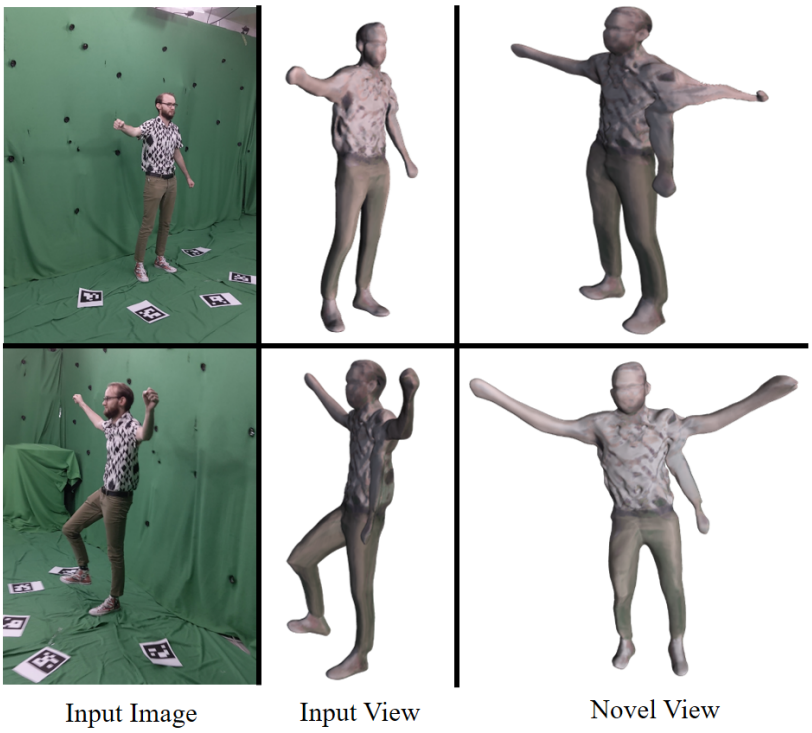


Figure 6.2: Example of our method using an additional appendage to satisfy the reconstruction losses.

6.2 Extraction of 3D Correspondences

One very important output that could be leveraged by many applications is the extraction of 3D correspondences. Unfortunately, Ub4D as presented in Chapter 4 produces non-corresponding per-frame geometry. This is because we execute marching cubes on every frame, mapping each point to the canonical space where we evaluate the SDF. An alternative approach would be to execute marching cubes once in the canonical space

(producing geometry as shown in Figure 5.8) and then map each vertex to its frame space point. We refer to this approach as animating canonical space and it would provide 3D correspondences throughout the scene.

Unfortunately, finding the corresponding frame space point for a given canonical space point is the opposite of our bending network which projects points from frame space *into* canonical space. We could attempt to solve this with the following minimization problem:

$$\boldsymbol{x}_i = \arg\min_{\boldsymbol{x}^*} ||\boldsymbol{x}^* + \boldsymbol{b}_i(\boldsymbol{x}^*) - \boldsymbol{x}_c||_2^2 \,. \tag{6.1}$$

Results for this approach are shown in Figure 6.3. Note that it fails for large deformations likely due to two reasons. First, the mapping between frame space and canonical space is not necessarily a bijection, meaning multiple frame space points can map to the same canonical space point. Second, we have no direct spatial smoothness regularizer on our bending network and this likely results in local minima.

Figure 6.3: Animating the canonical space to extract correspondences. Note that while such an approach works for small deformations, there are significant local failures for larger deformations.

Another approach to extracting corresponding geometry would be to simultaneously train an inverse bending network with a cycle-consistency loss [100]. This could allow for mapping points from canonical space into frame space directly by using this inverse bending network.

6.3 Novel Latent Codes

Some applications require semantically meaningful latent codes which we have shown in Section 5.8. This allows us to generate entirely new geometries by providing novel latent codes along some desired path. Figure 6.4 shows samples of a novel latent path for the 2D latent codes from the left-hand side of Figure 5.11. This ability is likely constrained by the convex hull of the observations; however, for large scale datasets this could provide the ability to generate non-rigid transitions between different animation cycles. A further investigation is required into the ability to generate novel geometries from novel latent codes, particularly when using higher dimensional latent codes.

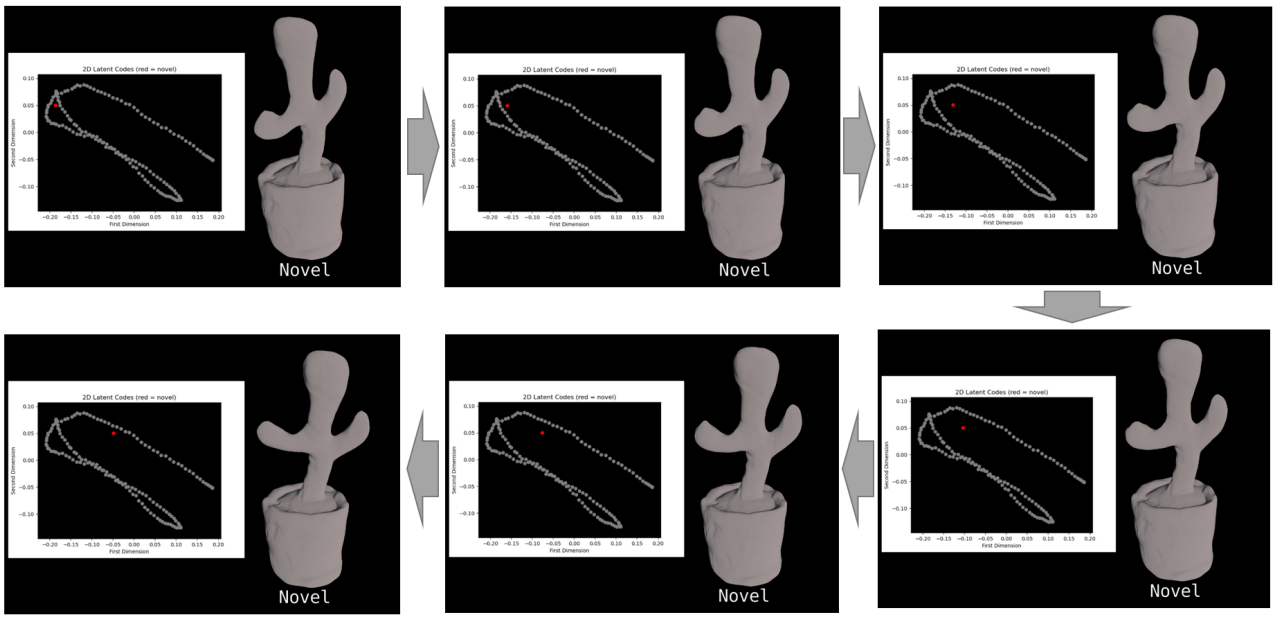


Figure 6.4: Synthesizing entirely new geometries with novel latent codes. Red dot in latent space plot shows the provided latent code while grey dots show the original sequence. Note the smoothness and plausibility of the deformation.

6.4 Runtime Improvements

A natural question for many computational applications is investigating performance improvements. This thesis did not consider the runtime of Ub4D and still leaves much to be done for improving it, both for training the model and extracting the geometry. Training times for each scene are given in Table 6.1 and the time for extracting the geometry as a function of the grid resolution is shown in Table 6.2.

Many proposed performance improvements for ISRs focus on improving the inference runtime by constructing approximate data structure representations of the learned neural radiance field [25, 43, 60, 98]. While modifications to these data structures could be made to account for bent rays, Ub4D only performs inference to extract geometry which takes at most the same time as training in a naïve implementation. An initial simple approach

could be to construct one of the described data structures for each timestep; however, a more interesting approach would be to consider using a separate data structure for time-dependent spatial warping. Replacing the bending network's output with learned coefficients of some basis representation (e.g. cosine functions) may allow for faster inference, additional interpretability, and is already a common idea in works for representing spatial motion [1, 89, 94]. Future work investigating novel geometry synthesis or novel view synthesis of deformable ISRs could motivate the development of faster inference for interactive simulation or rendering.

More relevant for the application of ISRs to monocular 4D reconstruction, is reducing the time spent to train a model. This direction has also seen recent progress [42, 80]. Since these works focus on general Multi-Layer Perceptron (MLP) integration [42] and coordinate-based neural representations [80], they can be utilized in Ub4D without significant modification. A more tailored approach exploiting temporal and motion priors may also provide a benefit for training time by initializing the bending network or even replacing it with learning coefficients for a specific basis, as mentioned above.

Another approach to accelerating training could be through leveraging the multigrid paradigm, which has seen success for efficient solution of Partial Differential Equations (PDEs) [8, 10]. Such an approach could work particularly well with a basis representation of spatial deformation, where relaxations and prolongations can be easily defined. A variant of Ub4D allowing the extraction of 3D correspondences could also leverage its own coarse correspondences through the scene flow loss during training. While this would not perform the same objective as using a coarse geometry proxy (i.e. guiding the network towards a single canonical copy), it could improve the training time through a coarse-to-fine learning approach.

Scene	*Iterations* (1000)	*Train Time* (hours)
Cactus	300	17.0
RootTrans	450	26.4
Lego	450	19.9
RealCactus	450	22.1
Humanoid	450	21.5

Table 6.1: Number of iterations and training time for Ub4D. Given for each of the scenes from Section 5.2.

Resolution	*March Time*	
	(seconds)	(hours)
64	14	0.00
128	69	0.02
256	536	0.15
512	4224	1.17
1024	34.99×10^6	18.32

Table 6.2: Time required to march geometry (without frustum culling). Frustum culling time is insignificant relative to the network evaluation time given here.

6.5 Volumetric ISRs, PDEs, and the Fourier Transform

The Fourier Transform is an integral transform first described by Joseph Fourier for solving the heat Partial Differential Equation (PDE) in 1822 [18]. It is impossible to state the extent to which the Fourier transform, and integral transforms more generally, have impacted all scientific and technical pursuits. Therefore, the proposal to investigate the dedicated application of the Fourier transform is hardly novel; however, there exists deeper connections surrounding ISRs using volumetric rendering equations that motivate interest in this direction.

Indeed, some works have already achieved success with related ideas. The Fourier Neural Operator (FNO) NN layer proposed in Li et al. [40] replaces a standard CNN layer with a multiplication in the Fourier domain that allows the modeling of global interactions more efficiently. They show that this FNO architecture performs remarkably well for learning the solutions to PDEs given the initial conditions [40]. Lange and Kutz [37] propose a method called FC^2T^2 using the Taylor series [81] and Fast Multipole Method (FMM) [22] to efficiently approximate convolutions. Despite these works, there still remains a specific gap at the intersection of volumetric ISRs, PDEs, and the Fourier transform.

The core of this remaining gap is in the fundamental relationship between ISR rendering and the mathematics behind the medical imaging technique of Computed Tomography (CT). Volumetric rendering as the basis for ISRs was presented in Section 3.1. It was shown in its differential form in Equation (3.1) before integrating to the more familiar (for the graphics community) integral form in Equation (3.2), which makes clear the fundamental connection to PDEs. CT solves the same problem as learning an ISR from images: given 2D images, solve for the density of the 3D scene.

The mathematical idea used in CT to solve for the 3D volume from 2D integral projection images is the Fourier Slice Theorem (FST). The FST was originally described in 2D by Bracewell [7] for solving a radio astronomy problem and generalized to arbitrary dimensions by Ng [49]. The generalized FST of Ng [49] can be formulated with operators as:

$$\mathcal{F}_m \circ \mathcal{P}_m^n \circ \mathcal{B} = \mathcal{S}_m^n \circ \mathcal{B}^{-1} \circ \mathcal{F}_n \,, \tag{6.2}$$

where $\mathcal{F}_m$ and $\mathcal{F}_n$ are m and n dimensional Fourier transforms, respectively; $\mathcal{P}_m^n$ is an integral projection from n to m dimensions; $\mathcal{S}_m^n$ is a slice operator from n to m dimensions; and $\mathcal{B}$ and $\mathcal{B}^{-1}$ are a basis transformation and its inverse, respectively. This is used in CT imaging for $m = 2$, $n = 3$ by reconstructing the right-hard side (i.e. undoing the slice operator) from the 2D Fourier transforms of captured projection images (i.e. 2D x-rays), then applying the basis transformation and taking the inverse 3D Fourier transform to recover the 3D density function [11]. Critically, by taking the inverse m^{th}-dimensional Fourier transform on both sides, we get:

$$\mathcal{P}_m^n \circ \mathcal{B} = \mathcal{F}_m^{-1} \circ \mathcal{S}_m^n \circ \mathcal{B}^{-1} \circ \mathcal{F}_n \,. \tag{6.3}$$

Unlike in CT imaging, this relationship gives an alternative method for rendering images. Such a method may provide benefits in terms of render speed or intermediate representation, particularly if the right-hand side of Equation (6.3) can be approximated, either entirely or in part, with a neural operator [33].

Unfortunately, direct application of the FST to volumetric rendering is known not to work due specifically to occlusion [36, 38, 85]. Revisiting the volume rendering equation

(Equation (3.5)) for a volume without any surface and ignoring all scattering:

$$L = \sum_{i=1}^{z} \underbrace{\left(1 - \exp\left(-\delta_i \, \mu_a(\boldsymbol{x}_i)\right)\right)}_{\text{attenuation}} L_e(\boldsymbol{x}_i, \boldsymbol{d}) \underbrace{\prod_{j=1}^{i-1} \exp\left(-\delta_j \, \mu_a(\boldsymbol{x}_j)\right)}_{\text{occlusion}}, \tag{6.4}$$

identifies the occlusion portion that violates the requirements on the projection operator $\mathcal{P}_m^n$ in the generalized FST of Equation (6.2). However with only the attenuation portion in Equation (6.4), as in x-ray images, we can trivially apply the FST and directly render images from the density function with Fourier transforms [36, 38, 85].

This suggests the existence of a non-integral method of rendering images from a spatial scene description. Additionally, the similarity of volumetric ISRs to the derivative of the scene light field provides a possible connection to PDEs and their solution methods. Such a connection warrants further exploration into the nature of the limitations preventing the application of the FST to volumetric ISR rendering.

A final note on this topic is to acknowledge the effect of Positional Encoding (PE) proposed in Mildenhall et al. [48] or the periodic activation functions of Sitzmann et al. [76] on the Fourier transform of the learned ISR. Both of these ideas use periodic functions either to modulate MLP inputs [48] or internally in the NN architecture [76]. This affects the Fourier transform of the learned scene representation in ways that may be exploitable in some meaningful way, either in conjunction with the above FST/PDE relationship or on their own. For example, PE without any DC component, results in an input that is infinite in support and periodic, thus having a finite-support Fourier transform.

Chapter 7
Conclusion

Many challenging problems that have long remained out of reach for industry application are seeing renewed interest with novel neural approaches. These range from formulating a portion of a classical approach using a Multi-Layer Perceptron (MLP) network to reformulating the objective as a learning problem. This work lies in the latter camp; while we take inspiration from the Non-Rigid Structure-from-Motion (NRSfM) and template-based tracking approaches, we apply recent advances from the disparate area of neural Implicit Scene Representations (ISRs) to the problem. We learn an ISR for the non-rigidly deforming object, from which we then extract explicit geometry.

In this work we demonstrated the use of ISRs for the problem of monocular 4D reconstruction for non-rigidly deforming objects. Our method, Unbiased 4D or Ub4D, showed state-of-the-art results in reconstructing a range of object types. Applying ISRs, and in particular deforming ISRs, to monocular reconstruction problems is a promising new approach capable of enabling new applications. Moreover, there exist unexplored frontiers in latent modeling and in the complex relationship between volumetric ISRs, Partial Differential Equations (PDEs), and the Fourier transform.

References

[1] Ijaz Akhter, Yaser Sheikh, Sohaib Khan, and Takeo Kanade. Nonrigid structure from motion in trajectory space. In *Neural Information Processing Systems (NeurIPS)*, 2008.

[2] Mohammad D. Ansari, Vladislav Golyanik, and Didier Stricker. Scalable dense monocular surface reconstruction. In *International Conference on 3D Vision (3DV)*, 2017.

[3] Matan Atzmon and Yaron Lipman. Sal: Sign agnostic learning of shapes from raw data. In *Computer Vision and Pattern Recognition (CVPR)*, 2020.

[4] Harry G. Barrow, Jay M. Tenenbaum, Robert C. Bolles, and Helen C. Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. *SRI Artificial Intelligence Center*, 1977.

[5] Paul J. Besl and Neil D. McKay. Method for registration of 3-d shapes. In *Sensor fusion IV: Control Paradigms and Data Structures*, 1992.

[6] Federica Bogo, Angjoo Kanazawa, Christoph Lassner, Peter Gehler, Javier Romero, and Michael J. Black. Keep it SMPL: Automatic estimation of 3D human pose and shape from a single image. In *European Conference on Computer Vision (ECCV)*, 2016.

[7] Ronald N. Bracewell. Strip integration in radio astronomy. In *Australian Journal of Physics*, 1956.

[8] Achi Brandt. Multi-level adaptive solutions to boundary-value problems. In *Mathematics of Computation*, 1977.

[9] Christoph Bregler, Aaron Hertzmann, and Henning Biermann. Recovering non-rigid 3d shape from image streams. In *Computer Vision and Pattern Recognition (CVPR)*, 2000.

[10] Andrés Bruhn, Joachim Weickert, Timo Kohlberger, and Christoph Schnörr. A multigrid platform for real-time motion computation with discontinuity-preserving variational methods. In *International Journal of Computer Vision (IJCV)*, 2006.

[11] Thorsten M. Buzug. Computed tomography. In *Springer handbook of medical technology*, 2011.

[12] Brian Cabral, Nancy Cam, and Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Symposium on Volume Visualization (VolVis)*, 1994.

[13] Subrahmanyan Chandrasekhar. Radiative transfer. *Courier Corporation*, 1960.

[14] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[15] 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 (ECCV)*, 2016.

[16] Blender Online Community. Blender - a 3d modelling and rendering package. *Blender Foundation*, 2018.

[17] Randima Fernando et al. Gpu gems: programming techniques, tips, and tricks for real-time graphics. *Addison-Wesley Reading*, 2004.

[18] Jean B.J. Fourier. Théorie analytique de la chaleur. *Chez Firmin Didot, père et fils*, 1822.

[19] Ravi Garg, Anastasios Roussos, and Lourdes Agapito. Dense variational reconstruction of non-rigid surfaces from monocular video. In *Computer Vision and Pattern Recognition (CVPR)*, 2013.

[20] Ravi Garg, Anastasios Roussos, and Lourdes Agapito. A variational approach to video registration with subspace constraints. In *International Journal of Computer Vision (IJCV)*, 2013.

[21] Vladislav Golyanik, Soshi Shimada, Kiran Varanasi, and Didier Stricker. Hdm-net: Monocular non-rigid 3d reconstruction with learned deformation model. In *EuroVR*, 2018.

[22] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. In *Journal of Computational Physics*, 1987.

[23] Marc Habermann, Weipeng Xu, Michael Zollhöfer, Gerard Pons-Moll, and Christian Theobalt. Livecap: Real-time human performance capture from monocular video. In *ACM Transactions On Graphics (TOG)*, 2019.

[24] Marc Habermann, Weipeng Xu, Michael Zollhöfer, Gerard Pons-Moll, and Christian Theobalt. Deepcap: Monocular human performance capture using weak supervision. In *Computer Vision and Pattern Recognition (CVPR)*, 2020.

[25] Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. *arXiv preprint arXiv:2103.14645*, 2021.

[26] Louis G. Henyey and Jesse L. Greenstein. Diffuse radiation in the galaxy. In *The Astrophysical Journal*, 1941.

[27] Daniel P. Huttenlocher, Gregory A. Klanderman, and William J. Rucklidge. Comparing images using the hausdorff distance. In *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 1993.

[28] Eakta Jain, Yaser Sheikh, Moshe Mahler, and Jessica Hodgins. Augmenting hand animation with three-dimensional secondary motion. In *ACM SIGGRAPH*, 2010.

[29] Rasmus Jensen, Anders Dahl, George Vogiatzis, Engin Tola, and Henrik Aanæs. Large scale multi-view stereopsis evaluation. In *Computer Vision and Pattern Recognition (CVPR)*, 2014.

[30] James T. Kajiya and Brian P. Von Herzen. Ray tracing volume densities. In *ACM SIGGRAPH*, 1984.

[31] Angjoo Kanazawa, Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. Learning category-specific mesh reconstruction from image collections. In *European Conference on Computer Vision (ECCV)*, 2018.

[32] Chen Kong and Simon Lucey. Deep non-rigid structure from motion. In *International Conference on Computer Vision (ICCV)*, 2019.

[33] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021.

[34] Jens Kruger and Rüdiger Westermann. Acceleration techniques for gpu-based volume rendering. In *IEEE Visualization (VIS)*, 2003.

[35] Suryansh Kumar, Anoop Cherian, Yuchao Dai, and Hongdong Li. Scalable dense non-rigid structure-from-motion: A grassmannian perspective. In *Computer Vision and Pattern Recognition (CVPR)*, 2018.

[36] Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *ACM SIGGRAPH*, 1994.

[37] Henning Lange and J. Nathan Kutz. Fc2t2: The fast continuous convolutional taylor transform with applications in vision and graphics. In *arXiv preprint arXiv:2111.00110*, 2021.

[38] Marc Levoy. Volume rendering using the fourier projection-slice theorem. In *Graphics Interface (GI)*, 1992.

[39] Xueting Li, Sifei Liu, Shalini De Mello, Kihwan Kim, Xiaolong Wang, Ming-Hsuan Yang, and Jan Kautz. Online adaptation for consistent mesh reconstruction in the wild. In *Neural Information Processing Systems (NeurIPS)*, 2020.

[40] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[41] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *Computer Vision and Pattern Recognition (CVPR)*, 2021.

[42] David B. Lindell, Julien N.P. Martel, and Gordon Wetzstein. Autoint: Automatic integration for fast neural volume rendering. In *Computer Vision and Pattern Recognition (CVPR)*, 2021.

[43] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. In *Neural Information Processing Systems (NeurIPS)*, 2020.

[44] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. In *ACM Transactions on Graphics (TOG)*, 2019.

[45] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM SIGGRAPH*, 1987.

[46] Nelson Max and Min Chen. Local and global illumination in the volume rendering integral. *Lawrence Livermore National Lab (LLNL)*, 2005.

[47] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[48] 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 *European Conference on Computer Vision (ECCV)*, 2020.

[49] Ren Ng. Fourier slice photography. In *ACM SIGGRAPH*, 2005.

[50] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Computer Vision and Pattern Recognition (CVPR)*, 2020.

[51] Jan Novák, Iliyan Georgiev, Johannes Hanika, Jaroslav Křivánek, and Wojciech Jarosz. Monte carlo methods for physically based volume rendering. In *ACM SIGGRAPH Courses*, 2018.

[52] 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.

[53] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[54] 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 *International Conference on Computer Vision (ICCV)*, 2021.

[55] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems (NeurIPS)*, 2019.

[56] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. In *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 1901.

[57] Matt Pharr, Wenzel Jakob, and Greg Humphreys. Physically based rendering: From theory to implementation. *Morgan Kaufmann*, 2021.

[58] Thomas Porter and Tom Duff. Compositing digital images. In *ACM SIGGRAPH*, 1984.

[59] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Computer Vision and Pattern Recognition (CVPR)*, 2021.

[60] Daniel Rebain, Wei Jiang, Soroosh Yazdani, Ke Li, Kwang Moo Yi, and Andrea Tagliasacchi. Derf: Decomposed radiance fields. In *Computer Vision and Pattern Recognition (CVPR)*, 2021.

[61] Susanna Ricco and Carlo Tomasi. Dense lagrangian motion estimation with occlusions. In *Computer Vision and Pattern Recognition (CVPR)*, 2012.

[62] Bernhard Riemann. Ueber die darstellbarkeit einer function durch eine trigonometrische reihe. *Dieterich*, 1867.

[63] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015.

[64] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *International Conference on Computer Vision (ICCV)*, 2019.

[65] Shunsuke Saito, Tomas Simon, Jason Saragih, and Hanbyul Joo. Pifuhd: Multi-level pixel-aligned implicit function for high-resolution 3d human digitization. In *Computer Vision and Pattern Recognition (CVPR)*, 2020.

[66] Tim Salimans and Durk P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Neural Information Processing Systems (NeurIPS)*, 2016.

[67] Mathieu Salzmann, Richard Hartley, and Pascal Fua. Convex optimization for deformable surface 3-d tracking. In *International Conference on Computer Vision (ICCV)*, 2007.

[68] Mathieu Salzmann, Julien Pilet, Slobodan Ilic, and Pascal Fua. Surface deformation models for nonrigid 3d shape recovery. In *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2007.

[69] Peter Sand and Seth Teller. Particle video: Long-range motion estimation using point trajectories. In *International Journal of Computer Vision (IJCV)*, 2008.

[70] Scott Schaefer and Joe Warren. Dual marching cubes: Primal contouring of dual grids. In *Pacific Graphics (PG)*, 2004.

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

[72] 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.

[73] Soshi Shimada, Vladislav Golyanik, Christian Theobalt, and Didier Stricker. Ismogan: Adversarial learning for monocular non-rigid 3d reconstruction. In *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[74] Renben Shu, Chen Zhou, and Mohan S. Kankanhalli. Adaptive marching cubes. In *The Visual Computer*, 1995.

[75] Vikramjit Sidhu, Edgar Tretschk, Vladislav Golyanik, Antonio Agudo, and Christian Theobalt. Neural dense non-rigid structure from motion with latent space constraints. In *European Conference on Computer Vision (ECCV)*, 2020.

[76] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Neural Information Processing Systems (NeurIPS)*, 2020.

[77] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: continuous 3d-structure-aware neural scene representations. In *Neural Information Processing Systems (NeurIPS)*, 2019.

[78] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *Symposium on Geometry Processing (SGP)*, 2007.

[79] Narayanan Sundaram, Thomas Brox, and Kurt Keutzer. Dense point trajectories by gpu-accelerated large displacement optical flow. In *European Conference on Computer Vision (ECCV)*, 2010.

[80] Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P. Srinivasan, Jonathan T. Barron, and Ren Ng. Learned initializations for optimizing coordinate-based neural representations. In *Computer Vision and Pattern Recognition (CVPR)*, 2021.

[81] Brook Taylor. Methodus incrementorum directa et inversa. *Innys*, 1717.

[82] Carlo Tomasi and Takeo Kanade. Shape and motion from image streams under orthography: a factorization method. In *International Journal of Computer Vision (IJCV)*, 1992.

[83] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *International Conference on Computer Vision (ICCV)*, 1998.

[84] Lorenzo Torresani, Aaron Hertzmann, and Chris Bregler. Nonrigid structure-from-motion: Estimating shape and motion with hierarchical priors. In *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2008.

[85] Takashi Totsuka and Marc Levoy. Frequency domain volume rendering. In *ACM SIGGRAPH*, 1993.

[86] 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 *International Conference on Computer Vision (ICCV)*, 2021.

[87] Unreal Engine. Unreal engine 4.24 to ship with free quixel megascans, unreal studio features, and more. *Epic*, 2019.

[88] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. In *Journal of Machine Learning Research (JMLR)*, 2008.

[89] Chaoyang Wang, Ben Eckart, Simon Lucey, and Orazio Gallo. Neural trajectory fields for dynamic novel view synthesis. *arXiv preprint arXiv:2105.05994*, 2021.

[90] Chaoyang Wang and Simon Lucey. Paul: Procrustean autoencoder for unsupervised lifting. In *Computer Vision and Pattern Recognition (CVPR)*, 2021.

[91] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *European Conference on Computer Vision (ECCV)*, 2018.

[92] 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. In *Neural Information Processing Systems (NeurIPS)*, 2021.

[93] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. Space-time neural irradiance fields for free-viewpoint video. In *Computer Vision and Pattern Recognition (CVPR)*, 2021.

[94] Weipeng Xu, Avishek Chatterjee, Michael Zollhöfer, Helge Rhodin, Dushyant Mehta, Hans-Peter Seidel, and Christian Theobalt. Monoperfcap: Human performance capture from monocular video. In *ACM Transactions on Graphics (ToG)*, 2018.

[95] Gengshan Yang, Deqing Sun, Varun Jampani, Daniel Vlasic, Forrester Cole, Huiwen Chang, Deva Ramanan, William T. Freeman, and Ce Liu. Lasr: Learning articulated shape reconstruction from a monocular video. In *Computer Vision and Pattern Recognition (CVPR)*, 2021.

[96] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. In *Neural Information Processing Systems (NeurIPS)*, 2021.

[97] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Ronen Basri, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. In *Neural Information Processing Systems (NeurIPS)*, 2020.

[98] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. *arXiv preprint arXiv:2103.14024*, 2021.

[99] Rui Yu, Chris Russell, Neill D.F. Campbell, and Lourdes Agapito. Direct, dense, and deformable: Template-based non-rigid 3d reconstruction from rgb video. In *International Conference on Computer Vision (ICCV)*, 2015.

[100] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *International Conference on Computer Vision (ICCV)*, 2017.

Appendix A
Experimental Details

We present hyperparameters and additional details for the experiments with respect to our method and previous works. Sections A.1, A.2, A.3, and A.4 give the experimental details for Ub4D, LASR, DDD, and N-NRSfM, respectively.

A.1 Ub4D

The hyperparameter settings for the experiments presented are contained in Table A.1. Refer to Equation 4.39 for the meaning of each γ hyperparameter and Section 4.3 for the meaning of the λ hyperparameters. Additionally, for the *RealCactus* experiment we use constant γ_{NBR} and γ_{DIV} weights, rather than a $\frac{1}{100}$ factor exponentially increasing schedule as suggested by Tretschk et al. [86].

Scene	γ_{SEG}	γ_{EIK}	γ_{NBR}	γ_{DIV}	γ_{FLO}	λ_1	λ_2
Cactus	1.0	0.5	20000	200	10	700	75
RootTrans	1.0	0.5	20000	200	10	700	75
Lego	0.75	0.25	10000	100	0	-	-
RealCactus	1.0	0.75	50000†	1000†	0	-	-
Humanoid	1.25	0.25	2500	2	0	-	-

Table A.1: A subset of hyperparameters used in acquiring results presented. † indicates a constant weight without the increasing schedule of Tretschk et al. [86].

A.2 LASR

We run LASR [95] in the manner shown in their code[2]. We progressively increase the number of bones and faces in a coarse-to-fine manner following the configurations provided. This progression is shown in Table A.2. For the *RootTrans* sequence, we use a

[2] `https://github.com/google/lasr`

slightly modified version of the code. This was to prevent a complete failure case where bone re-initialization without CNN re-initialization results in the mesh entering a local minimum that no longer reprojects on the image. This code modification was made with the assistance of the lead author of LASR [95][3].

Step	*Bones*	*Faces*	*Hypotheses*	*Epochs*
r1	21	1280	16	20
r2	26	1600	1	10
r3	31	1920	1	10
r4	31	2240	1	10
r5	36	2560	1	10
final	36	2880	1	10

Table A.2: A subset of parameters used when running LASR [95].

A.3 Direct, Dense, Deformable

We run the method of Yu et al. [99] in the manner shown in their code[4]. We empirically explored a set of values and found those of Table A.3 to perform best when comparing results after rigid alignment with ICP [5] to the ground truth.

Parameter	*Value*
Photometric weight	1
ARAP weight	20

Table A.3: A subset of parameters used when running the method of Yu et al. [99].

A.4 Neural NRSfM

We run Neural NRSfM (N-NRSfM) in the manner shown in their code[5]. In order to acquire the Multi-Frame Optical Flow (MFOF) W matrix used as input by this implementation, we use the code of Ansari et al. [2]. Additionally, this implementation requires input in a specific format which is computed using proprietary code provided by the authors. The loss function weights used are given in Table A.4.

Parameter	*Value*
β	1
γ	1×10^{-4}
η	1
λ	0

Table A.4: A subset of parameters used when running the method of Sidhu et al. [75].

[3] https://github.com/google/lasr/issues/7
[4] https://github.com/cvfish/PangaeaTracking
[5] http://vcai.mpi-inf.mpg.de/projects/Neural_NRSfM/