ZD-NeRF: Constraining Dynamic Neural Radiance Fields using Zero-Divergence Neural ODEs

Samuel Guard

October 21, 2023

ZD-NeRF: Constraining Dynamic Neural Radiance Fields using Zero-Divergence Neural ODEs

Submitted by: Samuel Guard

Copyright

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Master of Computer Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Abstract

Synthesising accurate novel views of a subject as it moves and deforms over time is an incredibly hard problem as it is under-constrained. In this work, we attempt to constrain it using Neural Ordinary Differential Equations with a Neural Radiance Field (NeRF). Specifically, the subject is constrained by ensuring it preserves volume over time. The proposed model in this work was unsuccessful in achieving its goal of improving on the state-of-the-art but has much potential that could not be demonstrated within the time constraints.

Code can be found here: https://github.com/SamGuard/ZD-NeRF.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisor Adam Hartshorne for the idea of this project and all the guidance they gave me along the way.

Thank you to my partner Tosia for all the support and food given to me over the course of this year.

Chapter 1

Introduction

Novel view synthesis is the process of creating unseen views of a subject given some data. This is a non-trivial problem: there are an inconceivable number of physical processes that contribute to how we see the world around us, and to achieve photo-realistic view synthesis a model of all this needs to be created. This model must achieve a level of complexity such that it can approximate our world well enough to make it believable, but not so complex it becomes intractable to create and use. Being able to achieve photo-realistic view synthesis would open many uses for this technology in film, such as a camera angle that can be adjusted without needing to re-film the entire scene.

Attempts to solve this problem started to be published around the 1990s [MT93, SD95, PE90], with a focus on applying their solutions to object recognition. [PE90]'s method is based on creating a "standard view" - this will later be referred to as a "canonical pose" in more recent research. The field has developed since then to where photo-realistic reconstruction is possible under certain conditions. At the forefront of this is Neural Radiance Fields (NeRF) [MST+21] and is what this work is focused on. Introduced in 2020, this new method is based on representing a scene by the radiance (the light reflecting on a surface) at each point in the scene using a Neural Field.

Dynamic view synthesis is the synthesis of views of a subject at a given time. This is obviously closely related to view synthesis but with many more challenges. With static scenes, you can safely assume that the structure of an object will remain constant under different views, however, this is not the case with dynamic scenes. Dynamic scenes add another dimension to the problem - time. This leads to the problem being under-constrained as it changes from interpolating between camera angles to interpolating between camera angles across time. An example of this increased difficulty could be a footballer kicking a ball through the air past other players: as the ball travels it is obscured from the camera temporarily. Does the ball phase in and out of existence during this? Obviously, we know that it does not, but how will a model of this environment know? We introduce constraints to the model to help guess information when none is available. In the context of NeRFs there have been many attempts to constrain this problem, such as reducing the scope of the dynamic NeRF to modelling humans [XAS21] and regularising movement in the scene so it must be smooth [SGP+22].

In this work, Dynamic NeRFs are constrained by enforcing a property on how the scene can deform over time. This property is the volume: using a divergence-free vector field to enforce consistency across time using a method introduced by [DZY+21]. This exploration has the

goal of improving the consistency of dynamic scene reconstruction for data consisting of sparse views.

Chapter 2

Background

2.1 View Synthesis

As stated before, view synthesis is the process of generating novel views from a set of images. The most recent "big leap" in view synthesis is Neural Radiance Fields, which is what this paper is based on, but it is important to avoid having "tunnel vision" on this recent development and instead take a look at other state-of-the-art methods and how we got to this point. As with many long-standing problems, solutions have gradually shifted to the use of neural networks due to their powerful ability to generate content by using neural rendering. Neural rendering is only loosely defined but [TFT+20] defines it as:

> Deep image or video generation approaches that enable explicit or implicit control of scene properties such as illumination, camera parameters, pose, geometry, appearance, and semantic structure.

Generative models came into prevalence in the 2010s using neural networks as their foundation; a large spike in interest came about after the introduction of Generative Adversarial Networks (GANs) [GPAM+14]. A GAN consists of two networks working against each other in a zero-sum game, each trying to best the other. Deep generative models naturally converged with view synthesis leading to the creation of photo-realistic methods [HZLH17, LPM+19]. The largest issue with using methods such as GANs is that they learn to generate images that look realistic at the expense of the accuracy of the generated views of the given subject. When it comes to novel view synthesis, it is important to recognise that we do not want *a view* of a subject, we want a specific view, and not just a plausible view but the view which is most likely given the data. Why? Having this control allows for many more potential applications for a model to be considered.

Methods for view synthesis each include their own way of rendering the final image. These can generally fall into the following categories: image-to-image rendering, classical rendering, and light transport.

To carry out view synthesis, data about the subject must be collected first. The minimum data required for view synthesis is one or more images of the subject. There exist models such as [TS20] which attempts to synthesise views from a single viewpoint using models that have been trained on large image datasets. These can only be so effective - they cannot be used for full-view synthesis, only small adjustments to the camera. In a real-world scenario, it is

likely that more than one image of a subject is available. Using two or three distinct views, it becomes possible to synthesise views of a subject from many angles, provided the subject has a simple shape, such as a matchbox. Methods for this include [SD95] and are some of the earliest methods for view synthesis due to their small data complexity. While using a small number of views can allow for effective view synthesis for simple objects, self-occlusion becomes an issue for more complex objects. Even for seemingly simple objects, parts of the object can be obstructed from the view of the camera, for example, a mug. This is why, for most view syntheses, many images are required of a subject to accurately generate novel views of a subject. For dynamic scenes, it is common for a single camera (virtual or real) to take images in an orbit around a subject. As well as this, it is common to use multiple static cameras aiming at a subject for view synthesis, for example, a televised game of football [CR03].

2.2 Shape Representations

While not all view synthesis algorithms create an internal representation of the scene that is being observed, it can be beneficial to this as it can allow for more complex models to be achieved. Below are some methods of representing scenes.

2.2.1 Voxels

Voxels are a method of representing an environment by dividing it into regular cubes. Voxels were first used for novel view synthesis in 1999 [SD99], which claimed to be able to carry out photo-realistic scene reconstruction. While the elegant algorithm could successfully reconstruct scenes there is an underlying issue with voxels: memory complexity. To achieve a high-quality model the grid of voxels must be very dense - increasing the density of voxels has an $O(n^3)$ memory complexity. So, doubling the number of voxels in each dimension will increase the total memory by a factor of 8. As well as this, creating a dynamic scene using this method would increase the memory complexity even further.

2.2.2 Polygons

Polygons have been the standard way of representing shapes for decades. There are different ways of rendering polygons, the most common being rasterisation and ray casting. The advantages of using polygons are that they offer a scalable way to represent high-fidelity scenes and hardware acceleration, which is now a standard feature of almost every computer. The downside of using polygons for view synthesis is there is no clear way of differentiating the polygons with respect to a target image. This is because the environment is too under-constrained. Surfels [PZvBG00] are a hybrid of voxels and polygons which constrains an environment by placing 2D surfaces across it in a grid. Surfels suffer from the same downside as voxels: to achieve a high-quality model the grid must become fine-grained, leading to high memory usage.

2.2.3 Neural Methods

In recent years, a large number of methods have been created to apply neural networks to shape representations. Most relevant to this work is neural implicit representations. A neural implicit representation is a method of continuously representing a signal using a neural network.

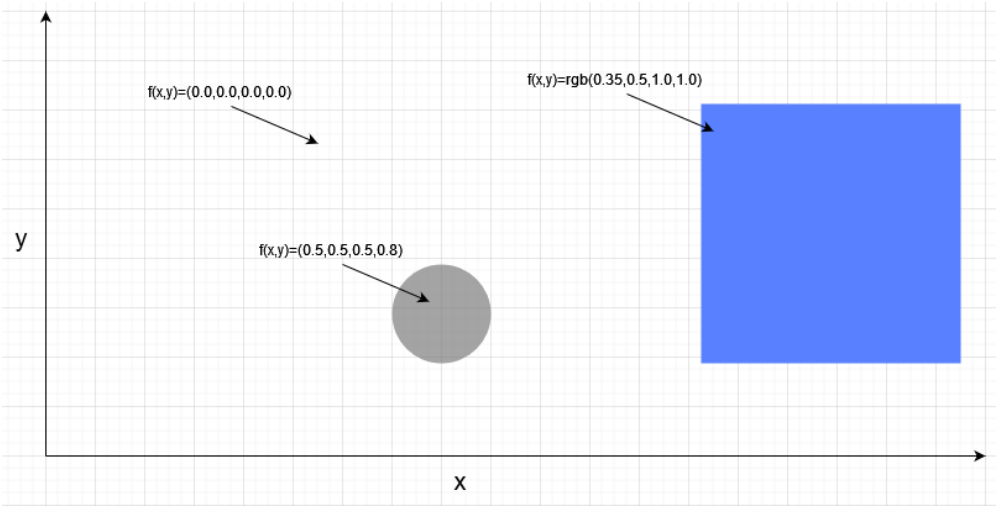


Figure 2.1: Figure shows a diagram of how a neural field can represent a scene. Example points are sampled showing their RGB and alpha values.

An example of this could be representing the shape of an object by learning a Signed Distance Function (SDF) for that object [MPJ+19].

2.3 Neural Fields and NeRF

A neural field is a field that is partially or fully parameterised by a neural network [XTS+22] and can be used to learn spatial representations. A Neural Radiance Field is a method of view synthesis introduced by [MST+21] that uses a neural field to represent the radiance of a space which is rendered using volumetric rendering techniques. This method has seen an explosion of interest since its publication in 2020 due to how effective it is at novel view synthesis. In essence, a NeRF uses images of a scene to attempt to construct a neural field which represents the light reflected from every point in the scene. This representation can then be rendered using volumetric rendering techniques. 2.1 shows a diagram of how a scene could be represented by a neural field - each point in space has an RGB and α value. The α is used to represent the opacity at the given point. This allows for transparent objects to be represented. An important feature that has not been mentioned yet is that the neural field is not only parameterised by a position but also by a camera pose. This can be thought of as learning the reflectance function for every point in the scene. This is a key part of what makes a NeRF able to render photo-realistic images as both specular and diffuse reflections can be rendered.

To transform the neural field into an image, a NeRF shoots rays from a virtual pin-hole camera out through a virtual screen. The virtual screen represents the grid of pixels that are being rendered. The colour $C(r)$ for each ray $r(t) = o + td$ is defined using the equation below.

$$C(r) = \int_{t_n}^{t_f} T(t)\sigma(r(t))c(r(t), d)dt$$
$$T(t) = exp(-\int_{t_n}^{t} \sigma(r(t)), d)dt \tag{2.1}$$

[MST+21]

t_n, and t_f are the near and far bounds (how much of the ray to integrate along). $c(r(t), d)$ is the colour of the neural field at point $r(t)$ with camera pose of d. $\sigma(r(t))$ is the density at point $r(t)$. $T(t)$ represents the transmittance. But what does this all mean? The purpose of the equation for $C(r)$ is to accumulate colour along a ray from the neural field, weighting the colour of the less transparent points more heavily. The transmittance equation is used to reduce the contribution of points that are either partially or fully occluded from the camera by integrating up to the points being sampled from the neural field. As both c and σ are defined by a neural network, this integral is approximated by sampling at evenly spaced points between t_n and t_f and summing. As this equation is the combination of samples from a neural network, the process is easily differentiable. This means that the loss can be calculated by comparing the image generated from a known view and its ground truth, which can then be used to optimise the network using a gradient-based optimisation method.

The largest issue with NeRF as of the release of the paper in 2020 was the time taken to train the neural field. It was noted in the original paper that this algorithm had a lot of potential for optimisation. To obtain photo-realistic results it could take days to train. As well as this, if the subject moved at all, this would result in a hazy "mist" around the moving parts as the NeRF.

Since the introduction of NeRF by [MST+21], many attempts have been made to expand the capabilities and improve the algorithm. These attempts can be seen in papers such as [TCY+22] where the neural field is also parameterised by the weather, allowing for views of the same setting to be rendered for different weather conditions. NeRF++ [ZRSK20] improved upon NeRF by simply changing the structure of the neural network that defines the neural field so that the camera's pose has no impact on the density. This development greatly improved the quality of the NeRF that could be obtained. [PSB+21] and [PCPMMN21] both presented methods for view synthesis using a NeRF to represent a canonical pose and were released around the same time. They both rely on creating a canonical pose using NeRF and a deformation function to warp points onto the canonical pose. [PSB+21] used this to account for movements in the subject being observed to create a single high-fidelity model. [PCPMMN21] used this to create a Dynamic NeRF.

2.3.1 Other Applications of NeRFs

NeRFs are exceptional at view synthesis but that does not mean they can only be used for this purpose. In training a NeRF, a neural representation of a scene is created which can be sampled at any point. A clear application for a NeRF is scene reconstruction: neural representations are not particularly useful to someone looking to manipulate a scene, although there are methods that can be used to edit NeRFs in a limited way [KMS22]. Scene reconstruction can be carried out using methods such as marching cubes. Another kind of reconstruction is presented in [XAS21], using a NeRF to reconstruct a virtual human and extract their pose. This brings us to our next potential area for a NeRF: using a NeRF as a loss function. This could be useful for constructing models that may be in a more useful form that could use NeRF's abilities to help it fit the data.

2.4 Dynamic NeRFs

Dynamic NeRFs are an expansion on NeRF - they attempt to take view synthesis of a 3D world to 4D using NeRFs. This is so that a NeRF can be used to synthesise views of scenes

that change over time. The simplest way of achieving this is to parameterise the neural field with the position, camera pose and time, which would lead to a 6D neural field. This method is often called a TNeRF.

Dynamic view synthesis is much less developed than static view synthesis. This is because of the steep increase in difficulty. In static scenes, you can safely assume that the structure of an object will remain constant throughout all images. So, even when part of a subject is occluded from an image, it is still possible to know what the entire subject should look like. For dynamic scenes, this is not possible as attempting to solve what part of a subject looks like when it is occluded from the camera has an infinite number of solutions.

Before dynamic NeRFs, different approaches to this problem have been taken. Dynamic novel view synthesis papers were less common in the early 2000s with many papers focusing on scene reconstruction instead. Many of these methods depend on RGB and depth data [WAO+09, DTF+15] or use a template to fit the input image onto [WAO+09]. The current state-of-the-art methods, excluding NeRFs, are centred around Simultaneous Location And Tracking (SLAM) [NFS15, LZNH20] or pretraining convolutional neural networks on large datasets [BZTN20, BPZ+21]. Again, these methods generally rely on RGB-D data with the purpose of scene reconstruction, not view synthesis.

2.4.1 DNeRF

[PCPMMN21] introduces a way of rendering dynamic scenes called DNeRF. Their method uses a NeRF to create a canonical pose of the scene at $t = 0.0$, and a deformation field is then used to map sample points at any time back to $t = 0.0$. The assumption underlying DNeRF is that the scene that novel views are being generated for does not change in appearance, only in structure. This mapping of points back to $t = 0.0$ has the effect of "bending rays", this effect is demonstrated nicely by figure 2.2.

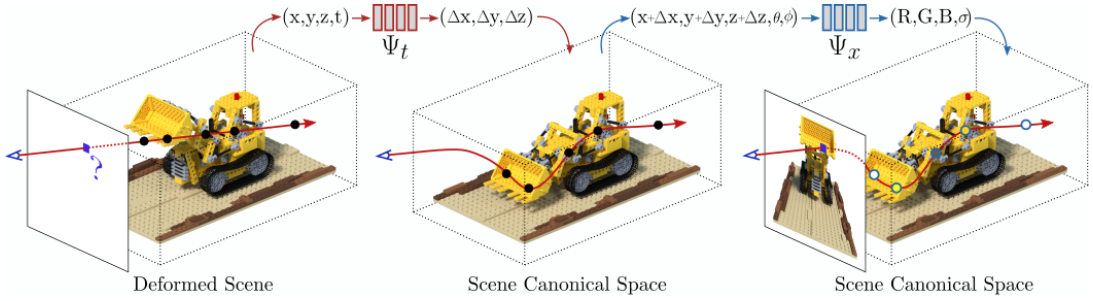


Figure 2.2: Figure from [PCPMMN21] showing their "ray bending" method.

The issue with using a canonical pose for a dynamic NeRF is that it becomes increasingly hard to learn a deformation function to represent how a canonical pose changes the longer the scene is. Theoretically, it should be possible to simply increase the size of the MLP in TNeRF to be able to represent longer scenes. As well as this, a dynamic NeRF constructed in this way can struggle to model specular surfaces, although [YLL23] has shown a complex method for reducing this issue.

The benefit of using a canonical pose is that it allows for sparse data to be used to reconstruct a dynamic scene more effectively than TNeRF as long as the scene being reconstructed is short and does not include many dynamic specular reflections. The quantitative results obtained in the DNeRF paper outperform TNeRF in most of their experiments, but the difference between

Figure 2.3: The figure above was created by [PCPMMN21] to display and compare their qualitative results. The quality of the TNeRF and DNeRF is comparable, with DNeRF besting TNeRF in these three examples.

the results is not drastic. 2.3 shows the results obtained from DNeRF - it can be clearly seen that DNeRF outperforms TNeRF but not drastically, which hints that TNeRF could achieve better results if expanded upon.

As well as this, the TNeRF outperforms DNeRF on the "Lego" scene, where a Lego Tractor moves its bucket. A potential explanation for this is that the scene contains more specular reflections than the other scenes, leading it to perform worse.

2.4.2 NeRFlow

NeRFlow is Dynamic NeRF which uses a TNeRF with a neural ordinary differential equation (see 2.5) to model the flow of the scene over time. This method does not use a canonical pose to apply the flow field too, instead, it uses the neural ODE to punish the TNeRF for not following the target consistencies. These consistencies are:

1. Appearance Consistency
2. Density Consistency
3. Motion Consistency

1 and 2 function similarly and use the assumption that a scene at a future time will be made up the same as the scene is currently, just moved/deformed. This relies on using a neural ODE to represent the motion of the scene. Consistency 3 uses two assumptions: empty space has no motion, and objects generally move smoothly. This penalises the neural ODE to enforce this consistency.

This paper also introduces a method for modelling dynamic specular reflections, by using two

independent TNeRFs, one for diffuse and one for specular reflections. The diffuse model is modified to only use a position and not a camera pose whereas the specular model is given both the position and camera pose. As you may be able to tell, parameterising the models in this way will lead to both the diffuse and specular models attempting to model diffuse reflections. As noted in the NeRFlow paper, specular reflections generally will only contribute small (but important) details to an image. This feature was used to ensure that the specular TNeRF kept to only modelling specular reflections by punishing the average magnitude of the neural field. This can be calculated by sampling random points within the NeRFs bounds and averaging their magnitude.

2.5 Neural Ordinary Differential Equation

Introduced by [CRBD18], Neural Ordinary Differential Equations (neural ODE) are a way of modelling flow continuously using a neural network. A vector field is a function that maps any point in a domain to a vector, for example:

$$V(x, y) = \begin{pmatrix} y \\ -x \end{pmatrix} \tag{2.2}$$

Visualised, this vector field will be:

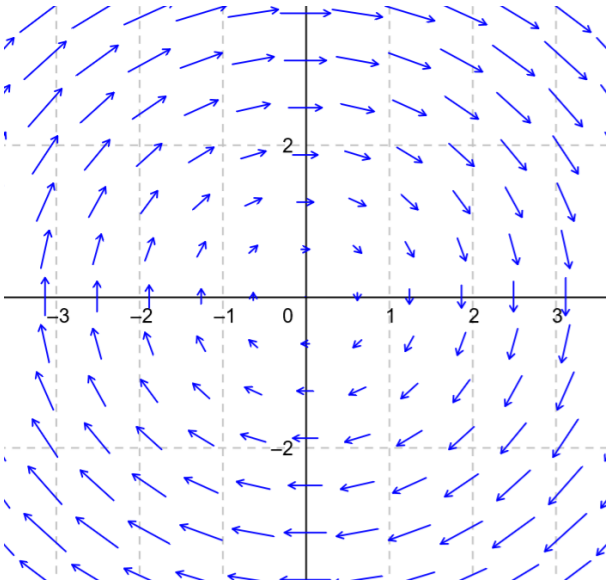


Figure 2.4: Figure plotting vector field V

Letting the vector field represent the instantaneous motion, a point can be moved through the vector field by integrating this motion with respect to time. The vector field is an ordinary differential equation as the instantaneous motion of a point p is

$$\frac{\delta p}{\delta t} = f(p, t) \tag{2.3}$$

where t is time and f is some function of t and p. [CRBD18] introduced the idea of using a neural network as f and using a differentiable ODE Solver to model the path of points through the vector field. A numerical ODE solver is used as there is no reasonable way to analytically

integrate a function defined by a neural network analytically. It is important that the ODE solver is differentiable so that the errors can be propagated to the neural network.

Overall, neural ODEs are a method of learning a vector field to model motion.

Chapter 3

Methodology

3.1 Problem Definition

The problem this work aimed to solve is: given a set of images and their camera poses, learn a function:

$$f(O, D) \rightarrow \begin{pmatrix} R \\ G \\ B \\ \alpha \end{pmatrix} \tag{3.1}$$

where O and D are the origin and direction of a ray, (R, G, B) is the colour and α is the opacity. The colour returned is the colour of the pixel to be rendered. This pixel can then be compared to the ground truth.

To achieve the above, a function was implemented so that the volumetric rendering could sample the colour and density at any point in space. This function was of the form:

$$f(x, y, z, \phi, \theta, t) \rightarrow \begin{pmatrix} R \\ G \\ B \\ A \end{pmatrix} \tag{3.2}$$

where (R, G, B) is the colour value, A is the density, at position (x, y, z) with camera pose (ϕ, θ) and at time t. The goal is to define a function f that accurately represents the light emitted at each point in space and time in the scene that has been imaged.

3.2 Proposed Solution

As discussed in the introduction, the problem of dynamic view synthesis is under-constrained. To constrain this problem, without specialising the model for specific objects such as a person, it is assumed the objects in the scene will preserve their volume throughout time. Objects are able to transform over time, but cannot grow or shrink in volume. An example of this could be humans: a human can walk, wave, crouch, etc. by deforming parts of the body. None of these actions changes the volume of the person.

To do the above, a representation of how the scene looks and deforms over time was needed. To achieve both of these goals, it was decided that a TNeRF (a NeRF with an extra dimension

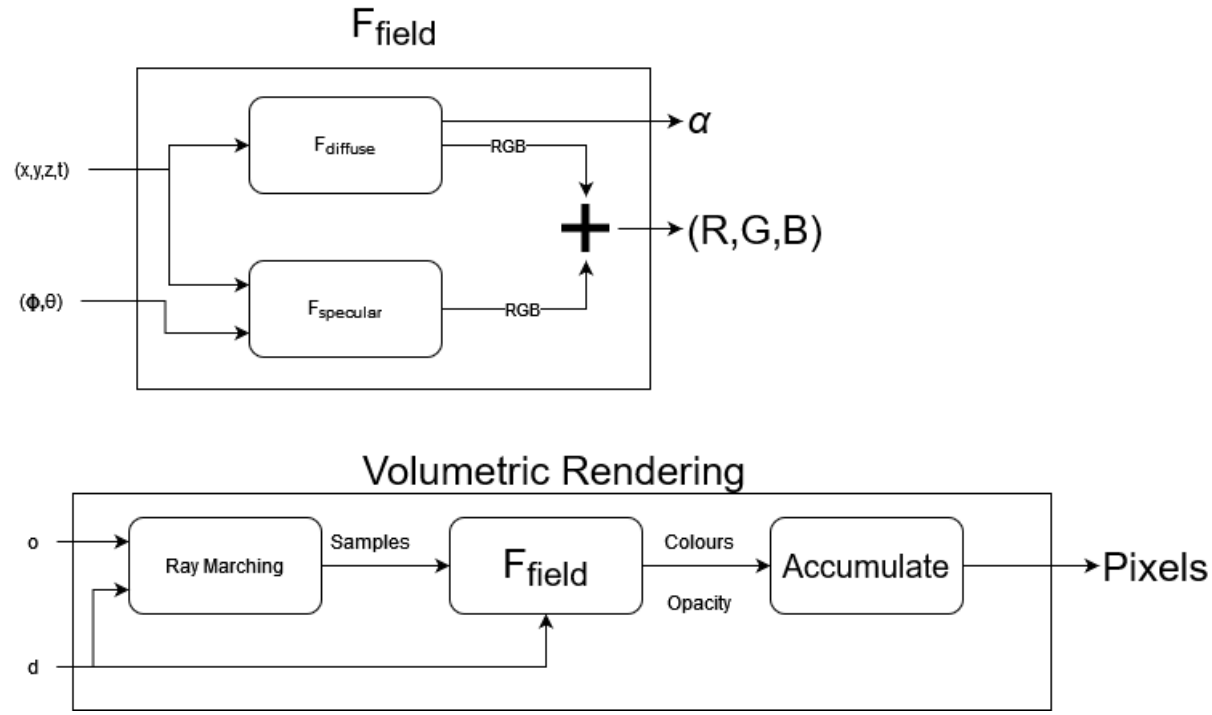


Figure 3.1: Diagram showing volumetric rendering using our NeRF implementation. o and d are the ray origin and direction respectively.

for time) alongside a neural ODE would be used. The TNeRF was used to represent the scene over time and the neural ODE was used to model how the scene changes over time so that it could enforce constraints. Using the TNeRF and neural ODE in this way is the same as the method put forward in [DZY+21]. This was expanded on by constructing the neural ODE such that it conserved the volume of the scene over time.

3.3 Neural Radiance Field

The NeRF described in [MST+21] was used but the neural field was parameterised by a position, a camera pose and time. This alone represents the appearance of the scene. This was selected to represent the scene as it provided a "blank slate" to apply the constraints to and does not use a canonical pose like other dynamic NeRFs ([PCPMMN21], [GTZN21]) which have the drawbacks mentioned in the DNeRF background section.

The method presented in [DZY+21], which allowed diffuse and dynamic specular reflections to be modelled by utilizing separate TNeRFs for diffuse and specular reflections, was used to represent the appearance of the scene. Below are the equations for calculating the colour of a sample and the specular loss

$$f(x, y, z, \phi, \theta, t) = f_{diffuse}(x, y, z, t)\sigma(x, y, z) + f_{specular}(x, y, z, \phi, \theta, t) \tag{3.3}$$

$$\mathcal{L}_{specular} = \frac{1}{N} \sum_{rand(x,y,z,\phi,\theta)}^{N} |f_{specular}(x, y, z, \phi, \theta, t)|^2 \tag{3.4}$$

where σ is the density.

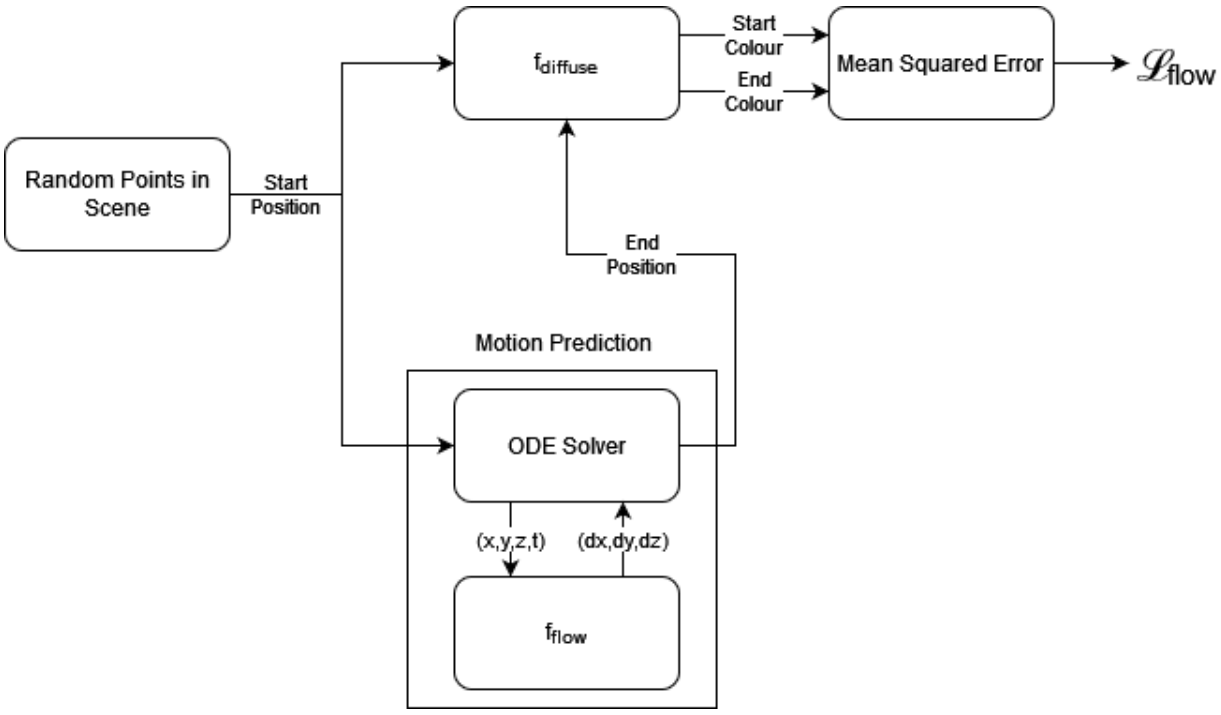


Figure 3.2: Diagram of how the loss is calculated for the constraint.

3.4 Neural ODE

A neural ODE was used to represent how the scene deforms over time by defining a neural field that will give the velocity for each point in space and time. Using a numerical ODE solver, the path of a point through a scene can be calculated. The purpose of learning this field was to be able to model the flow of scenes over time, which opened the door to constraining the NeRF. Importantly, the neural ODE has no effect on the rendering process and was only used during training. Again, the NeRF was used to represent the appearance and the neural ODE was used to represent the structure changing over time.

3.4.1 Time Consistency

Below describes how the constraint from [DZY+21]'s paper was included into the model. Figure 3.2 shows an overview of this process. This constraint relies on optimising the neural ODE against $\mathcal{L}_{flow}$.

The constraint that was enforced relied on the assumption that if a point has a diffuse colour c and density d at time t then this same point should have a colour c and density d at $t + \delta$ where δ is the change in time. "Point" in this case refers not to a coordinate in space but to a point on an object that may move over time. This was formed into a loss function by calculating the difference in colour between the two sampled points:

$$\mathcal{L}_{flow} = |c_t - c_{t+\delta}|^2 + |d_t - d_{t+\delta}|^2 \tag{3.5}$$

The neural ODE was used to model the movement of points in the scene to obtain c_t, $c_{t+\delta}$, d_t, $d_{t+\delta}$ as follows. Let $\mathcal{X} = (x, y, z)$ be the point of interest in the scene, then $c_t = f_{diffuse}(\mathcal{X}, t)$,

$d_t = \sigma(\mathcal{X})$. The other two variables are calculated by:

$$\begin{aligned} \mathcal{X}' &= f_{flow}(\mathcal{X}, [t, t+\delta]) \\ c_{t+\delta} &= f_{diffuse}(\mathcal{X}, t+\delta) \\ d_{t+\delta} &= \sigma(\mathcal{X}', t+\delta) \end{aligned} \tag{3.6}$$

To apply the constraint, random points were sampled within the scene's "aabb" bounding box and the $\mathcal{L}_{flow}$ was calculated. Both the neural ODE and the radiance field contribute to the loss which means the loss can propagate gradients into both of these models allowing them both to be trained simultaneously. Some random points will fall into empty space. It is important to ignore the diffuse colour of empty space as it has no meaning. To do this the occupancy grid (discussed more in 3.5.1) was used to see if a point was in empty space or not. If the point was in empty space it was removed from the set of samples.

Learning the vector field initially is hard - with random initial weights it is easy to see how approaching an accurate vector field could be difficult. That was why before the vector field constraint was applied it was first trained using key points. Key points are points that are tracked through the scene and can be obtained either through algorithms like SIFT [Low99] or (for synthetic datasets) can be extracted from vertex positions. The data of this took the form of a list of points being measured at different points in time. For the explanation of the training process below, T is a sequential list of times at which the points have been measured (e.g. $T = 0, 0.5, 1.0$), and K is a table of points which contains the coordinates of the point for a given time in T (e.g. $K_{t=0.5} = (0, 1, 0), (-2, 1, 4)$).

1. Select s consecutive elements t from list T
2. Select the points p that correspond to the time of t from K
3. With the first element of p being the initial value, use the neural ODE to predict the values at times t.
4. Calculate the L2 loss between the predicted points and the actual points

s had a large effect on the training. Letting $s = |T|$ would use the entire dataset at once to train the points but this had issues as the first steps in the algorithm would have a drastic effect on the later points leading to inconsistent training. Letting $s = 1$ would train from a single point to the next, but this struggled to approach a smooth solution.

3.4.2 Divergence-Free Vector Field

To take the constraints one step further, an assumption was made that the scene being modelled preserves volume over time. To enforce this structure in our NeRF, we constrain the neural ODE. Instead of allowing the function underlying the neural ODE to be just a neural network, the function is specially constructed such that the vector field produced is volume-preserving at all times. To make the vector field preserve volume, there must be zero divergence at every point in the vector field. This is expressed as the Jacobian of the function having a trace of zero (the values along the diagonal sum to zero). The method that was used to achieve this is presented below.

The divergence of a vector field $V(x, y, z)$ can be calculated using the expression [LZ20]:

$$\nabla \cdot V(x, y, z) = \frac{\delta V_x}{\delta x} + \frac{\delta V_y}{\delta y} + \frac{\delta V_z}{\delta z} \tag{3.7}$$

where V_x, V_y, V_z represent the equation for each dimension in the output.

For a vector field to be volume preserving $\nabla \cdot V(x, y, z) = 0$. As the underlying function of a neural ODE is a neural field, the neural field needs to be constructed to have zero divergence at all points. Using the divergence as the loss is a way to approximately achieve our goal. Doing this would have added unnecessary computation and would not achieve an exact solution. Instead, we used the first derivative of the neural network to calculate the curl at any given point. Curl can simply be described as the rotation around a point. Curl can be calculated simply as libraries such as PyTorch have built-in support for calculating derivatives. The curl of a given vector field V can be calculated by using the following equation.

$$\nabla \times V(x, y, z) = \begin{pmatrix} \frac{\delta V_z}{\delta y} - \frac{\delta V_y}{\delta z} \\ \frac{\delta V_x}{\delta z} - \frac{\delta V_z}{\delta x} \\ \frac{\delta V_y}{\delta x} - \frac{\delta V_x}{\delta y} \end{pmatrix} \tag{3.8}$$

Now, taking the divergence of this function will result in 0 as each dimension V_x, V_y, V_z has no component x, y, z in their respective dimensions.

Another method for constructing a divergence-free neural ODE was presented by [RPLC22]. This process involves learning an anti-symmetric matrix field and taking their divergence to obtain a divergence-free vector neural ODE. This method was not chosen over the curl method due to its complexity, even though it would have potentially increased the speed of training.

3.5 Implementation Details

The work was implemented in Python using PyTorch which is a library for tensor computation and automatic differentiation. This was chosen because it is prevalent in the NeRF community and in machine learning research as a whole.

3.5.1 NeRFAcc

NeRFAcc [LGTK23] was the library used as the basis for our NeRF. NeRFAcc stands for "NeRF Accelerated" and is a library for easily implementing NeRFs which also handles their acceleration. All that is required to create a NeRF using NeRFAcc are two functions to calculate the density and the RGB value at given points. NeRFAcc accelerates the rendering process by reducing the overall number of points that need to be sampled. It does this by pruning samples that fall into empty space or have a low transmittance (samples that are occluded from the camera).

Their example implementation of DNeRF [PCPMMN21] was used as a starting point for this work.

3.5.2 Torchdiffeq

Torchdiffeq (https://github.com/rtqichen/torchdiffeq) is a numerical ODE solver for PyTorch. This was a key part of the neural ODE implementation as it gave an easy-to-use function

to turn any PyTorch neural network into a neural ODE. It also implements O(1)-memory backpropagation.

3.5.3 Models

The models implemented for the NeRF and the neural ODE were both MLPs. As mentioned before, our NeRF consisted of two separate TNeRFs: diffuse and specular. The diffuse network is simply an MLP that takes a position and time and maps it to a colour and opacity. The specular network is also an MLP that maps an input position, time and camera pose to a colour.

The neural ODE was implemented using an MLP which then had its curl extracted using the above method. No information could be found discussing the attributes of different activation functions for neural ODEs apart from needing to be Lipschitz continuous, so informal experiments were carried out to investigate different activation functions. ReLU and Tanh were trialled as they are Lipschitz continuous and they were mentioned in the original neural ODE paper, but no noticeable difference was found between them. Researching this further felt off-topic, so I arbitrarily chose Tanh and moved my attention elsewhere.

Chapter 4

Experiments, Investigation and Conclusion

4.1 Outline

In this chapter, we explore the capabilities of this new model with a focus on comparing this model to others. To test the model, both quantitative and qualitative data have been obtained using a number of different synthetic test cases. It was decided to use all synthetic datasets as these can be easily created without the need for expensive camera rigs or data processing pipelines using software such as COLMAP [SF16] to extract the camera pose. As well as this, extracting key points from the synthetic data was a much simpler process as they could be directly exported from the model itself. Key point extraction is not a difficult task and could have been carried out on existing datasets, but creating this pipeline in this short time frame would have taken valuable time away from more important areas. This does mean that these results obtained do not give a full picture of this model's performance, but it can show its potential.

Alongside our model, a TNeRF and a NeRFlow-like model were run on the same datasets. As explained before, a TNeRF is a NeRF which is parameterised with time, as well as position and the camera's pose. This was implemented as our model with the consistencies disabled as it will give the best view of how the consistencies affect the model. The NeRFlow model was not used to compare against as it has been constructed in a different way, with different rendering techniques, so directly comparing the NeRFlow model to our own model would not give meaningful results. A NeRFlow-like model was implemented by removing our extra constraint on the flow field - this leaves a model that is not a replica of the NeRFlow model but it will demonstrate the effectiveness of our changes. In the below experiments, the NeRFlow model will be referred to as "NeRFlow-Like" to make it clear that this is not the exact model from the NeRFlow paper.

Figure 4.1 shows the testing plan. "No. images" is the number of images in the dataset, "Observ. Method" is the method used to observe the object which refers to how many cameras there were observing the scene and how they moved over time.

Name	No. Images	Observ. Method	Description
Soft Ball	50	Single camera orbiting	Shows a ball falling and deforming on 2 static cubes. Its purpose is to test the flow field's capability of modelling complex deformations and preserving volume as the ball deforms
Spinning Ball	50	Single camera orbiting	Shows a sphere being squashed while also spinning in the opposite direction of the camera. Its purpose is to show the model's performance on a simple scene where much of the subject is occluded at any given time.
Monkey	150	3 cameras at random angles	A dataset consisting of the Suzanne Monkey, with specular reflections. The light is the only object to move. This is to test the handling of shadows and reflections.
Lego	50	Single random camera views	A Lego tractor lifts its scoop into the air. This dataset has no key points. This dataset's purpose is to test how having no key points to pre-train the flow field affects performance.

Table 4.1: Datasets

4.2 Experiment Limitations

Not all of the experiments in 4.1 could be carried out. This was because of three factors: (1) overrun in the development, (2) time taken to produce results, (3) compute resources/model size. Development significantly overran due to errors, the time taken to test if a model is working as intended, and parameter configuration. These three factors contributed to development taking significantly longer than planned. (2) and (3) go hand-in-hand in contributing to not obtaining full results, as completing a test of the full model could easily take 8+ hours to train the NeRF on a compute cloud using a single NVIDIA GeForce RTX 3090. To train the full model would take up to 16GB of memory which was often not possible.

It was only possible to obtain results for the "soft ball" and "spinning ball" datasets after a total of ≈ 35 hours of compute time. Altogether, it was disappointing to not have been able to perform more so that this model's performance could be dissected more.

Another limitation is the amount of training done for each dataset. In NeRFlow's paper, their model was trained for 10 hours with just the render loss and then another 10 hours with all losses. This was not possible for the reasons mentioned above.

4.3 Results

The results will be displayed and discussed below. A follow-up investigation follows this section investigating the results obtained.

The qualitative results can be found in figures 5.1 5.2. These are images synthesised from a novel view. The model achieved good results for the spinning ball 5.1 dataset, with it being comparable to the NeRFlow-Like model. Our model and the NeRFlow-Like model both resemble the shape of the ground truth model more than the TNeRF. This speaks to the effectiveness of the flow field as the image generated by TNeRF has many fine-grained details

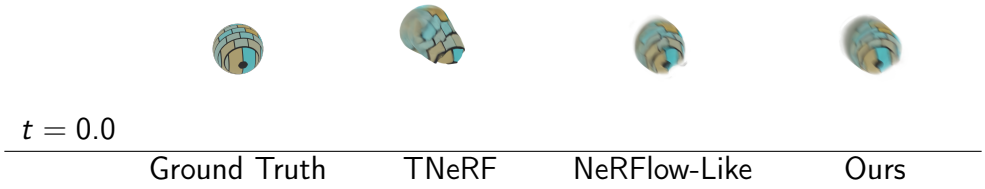


Figure 4.1: Subset of the Spinning Ball results. The full results can be found in the appendix.

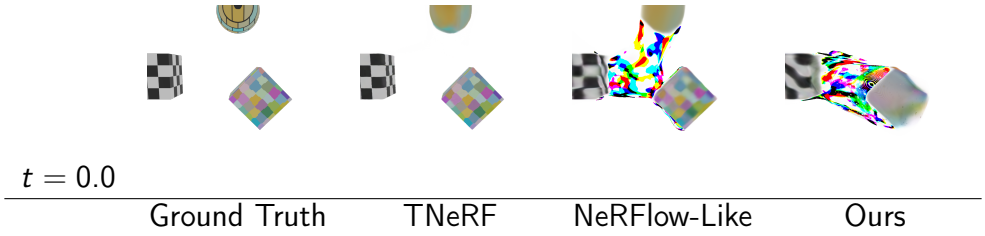


Figure 4.2: Subset of the Soft Ball results. The full results can be found in the appendix.

but the shape is entirely wrong. By also looking at the images from known viewpoints it is easy to see the TNeRF has overfitted to the training data. This is not true for NeRFlow-Like or our method - the shape of the object has been preserved across viewpoints but there are fewer fine-grained details. This can be seen most clearly in the last set of images for the spinning ball, where our model appears to be performing the best.

Figure 5.2 shows the results of the Soft Ball dataset. The results here show an entirely different story from the previous dataset. Starting with TNeRF, this model learnt a clear representation of the two static boxes and the ball at the start and end. However, when the ball was in motion, it lost all shape and became blurry. This is a reasonable reconstruction and, unlike in the previous dataset, it did not appear to overfit. It is likely that with more time and a larger model, the TNeRF could have produced better results. NeRFlow-Like and our model produced erroneous results - what likely led to this result is the neural ODE being unable to model the flow of the scene causing it to wrongly punish the diffuse NeRF. This then led to the diffuse NeRF becoming worse in a feedback loop which eventually led to both the diffusion NeRF and the neural ODE getting stuck in an unrecoverable state. The investigation below looks further into these results. The NeRFlow-Like model appeared to approach a more accurate solution than our model.

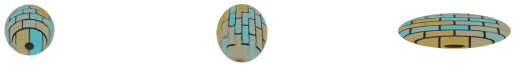
Quantitative results can be found in figure 4.2. These results reflect the observations made in the qualitative results. The only exception is the difference in score between our model and NeRFlow-Like for the Spinning Ball dataset but there is only a small difference between these values.

	Spinning Ball	Soft Ball
TNeRF	15.182	27.370
NeRFlow-Like	21.383	25.429
Ours	19.894	21.481

Table 4.2: Table showing the peak signal-to-noise ratio (PSNR) for each dataset and model.

4.4 Investigation of the Results

This section aims to explain the results obtained. To begin with, we will take a closer look at the TNeRFs performance for the Spinning Ball dataset. Previously, it was claimed that the TNeRF overfit the data. Figure 4.3 shows renders from the points of view of the training data.



$t = 0.0$ $t = 0.5$ $t = 1.0$

Figure 4.3: Renders from known points of view by TNeRF showing that it is overfitting.

Comparing these renders to the test view you can tell this very closely resembles the ground truth. Obviously, the model will always perform better on the training data but the novel views show fine detail on the surface of the subject on an incorrect shape. This shows that the TNeRF has not smoothly interpolated between data points. Why is this important? It is important because these results are not seen in either the NeRFlow-Like or our model, showing that constraining the problem using the neural ODE was effective for this test case, although this came at the cost of some finer details. To further investigate this, the flow field was visualised and the results are as follows:

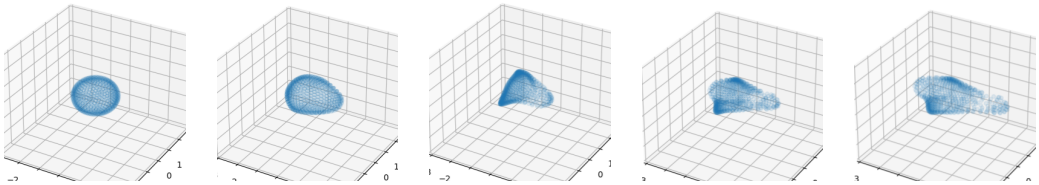


Figure 4.4: NeRFlow-Like

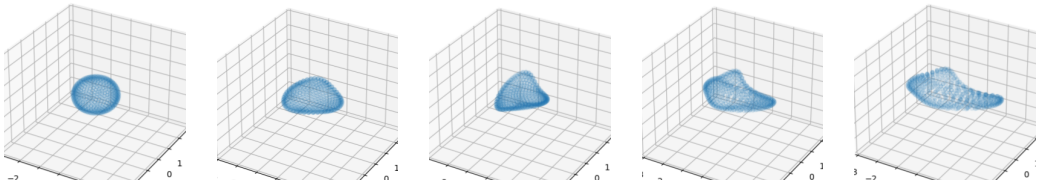


Figure 4.5: Ours

Both figures show incorrect flows, with odd shapes that appear to be rotating. A simple explanation for this could be the method used to visualise this is bad. The flow is modelled by selecting the points at $t = 0.0$ and integrating for each timestep. Integrating over such a large set of times could lead to errors building up over time. A more likely explanation is that the flow field in both models only learnt a rough model of the flow and that alone was enough to constrain the model. As the Spinning Ball dataset is simple compared to other examples, it is clear that if the neural ODE cannot learn the flow for this test case, it almost certainly cannot for more complex scenes.

The neural ODE is fundamental to our method so it is important to attempt to pinpoint the exact issue. During development, it was noted that the neural ODE would not perfectly model the flow of the scene after the initial training on key points. Below is a figure displaying the neural ODE post-training on the Spinning Ball key points using the zero-divergence method.

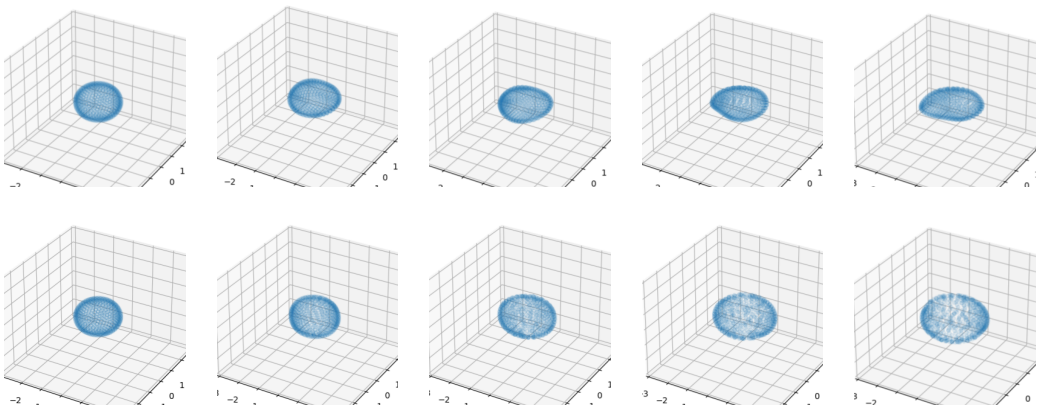


Figure 4.6: Top row shows without divergence, the bottom row shows with divergence

While imperfect, both neural ODEs approach a reasonable solution after training on the key points, reaching a better solution than seen in figure 4.5. This must mean that the flow learnt here has worsened throughout the rest of the training. The most likely reasons for this are (1) the flow field learnt from the key points does not represent the actual flow of the scene, (2) incorrect hyperparameters or (3) not trained for long enough. (1) is likely an issue as the key points do not represent the motion of the entire scene, only the motion of non-empty space. This could become an issue when the TNeRF incorrectly learns that an empty space is not empty for any reason, which then has consistency enforced upon it. (2) training parameters could cause the training process to fail in odd ways. An example of this is $\mathcal{L}_{flow}$ which if weighted too heavily could cause the model to enter into an unrecoverable state by giving everywhere a high density value. (3) could be a simple explanation for this issue - it is not unreasonable to think that the neural ODE will temporarily become worse once training on the TNeRF starts before approaching a good solution. The solution to this would not be to train for longer, but instead to investigate why so much training is required for the flow field to approach a good solution.

After some further investigation, it became clear what caused the issues seen in the results for the Soft Ball dataset. Modelling the flow learnt by both NeRFlow-Like and our method showed zero movement. This likely led to the blurry model being learnt. The reason for the odd red, green and blue colours appearing was found to be caused by a simple bug. In a regular NeRF, a sigmoid function is applied to the colour output so that the colour is always between 0 and 1 but in our method two MLPs are used to represent colour - the diffuse and specular MLPs. To get the colour from our NeRF the diffuse and specular colours are summed. The colours output from the two MLPs had the sigmoid curve applied separately and then summed. This caused the colour to have a domain of 0 to 2 causing invalid colours to be rendered. The fix for this was simply summing the two raw outputs and then applying the sigmoid curve. It is unfortunate that this bug made it into the final model but you may notice that the TNeRF did not suffer from this issue, even though it had the same bug. This implies that the bug was only able to have a significant effect on the end model because of the poor performance of

NeRFlow-Like and our model on the Soft Ball dataset. This claim is backed up by the bug not appearing in the Spinning Ball results.

The reasons why both the NeRFlow-Like and our model performed poorly are unclear - an obvious reason could be errors in the implementation. This is something that must be considered in all software development. Great care has been taken throughout the development process to avoid introducing errors, with test environments being created to test components of the model. An example of this is the script created to train and visualise the neural ODEs. Any software that does not achieve the set goals can simply be put down to bugs but this does not reap any valuable information and could very well be wrong. Another explanation for the poor performance of the models could be incorrect training processes and parameters. The training process from [DZY+21] was closely followed but the parameters were not. Parameters were found through trial and error throughout the development process. In hindsight, this was a significant mistake. The test should have, at least initially, been carried out with the exact parameters used in the NeRFlow paper.

4.5 Conclusion

The work here did not achieve its goal of improving upon the state-of-the-art in dynamic NeRFs. The few results obtained showed some promise but it was not possible to recreate the results obtained in [DZY+21]. The investigation above looks into the potential reasons for this outcome, with the most likely explanation being: errors in the implementation and an incorrect training process. This outcome occurred because not enough emphasis was initially placed on replicating the results obtained in the NeRFlow paper. With more time, this work has the potential to improve upon the NeRFlow paper and the state-of-the-art in dynamic NeRFs by applying what has been learnt from the analysis of this model. The idea behind this work seems to be reasonable and with more time it could have come to life. It would not be unreasonable to think that only small changes would be required for this to happen.

It is disappointing that this work has ended the way it did but I have learned a lot in the process. Before September, I had no experience with PyTorch or knowledge of Neural Radiance Fields and it has been a long journey to get to where I am now.

Bibliography

[BPZ+21] Aljaz Bozic, Pablo Palafox, Michael Zollhofer, Justus Thies, Angela Dai, and Matthias Niessner. Neural deformation graphs for globally-consistent non-rigid reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1450–1459, June 2021.

[BZTN20] Aljaz Bozic, Michael Zollhofer, Christian Theobalt, and Matthias Niessner. Deepdeform: Learning non-rigid rgb-d reconstruction with semi-supervised data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[CR03] K. R. Connor and I. Reid. A multiple view layered representation for dynamic novel view synthesis. In *Proceedings of the British Machine Vision Conference*, pages 72.1–72.10. BMVA Press, 2003. doi:10.5244/C.17.72.

[CRBD18] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[DTF+15] Mingsong Dou, Jonathan Taylor, Henry Fuchs, Andrew Fitzgibbon, and Shahram Izadi. 3d scanning deformable objects with a single rgbd sensor. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[DZY+21] Y. Du, Y. Zhang, H. Yu, J. B. Tenenbaum, and J. Wu. Neural radiance flow for 4d view synthesis and video processing. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 14304–14314, Los Alamitos, CA, USA, oct 2021. IEEE Computer Society.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[GTZN21] Guy Gafni, Justus Thies, Michael Zollhofer, and Matthias Niessner. Dynamic neural radiance fields for monocular 4d facial avatar reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8649–8658, June 2021.

[HZLH17] Rui Huang, Shu Zhang, Tianyu Li, and Ran He. Beyond face rotation: Global and local perception gan for photorealistic and identity preserving frontal view

synthesis. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[KMS22] Sosuke Kobayashi, Eiichi Matsumoto, and Vincent Sitzmann. Decomposing neRF for editing via feature field distillation. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[LGTK23] Ruilong Li, Hang Gao, Matthew Tancik, and Angjoo Kanazawa. Nerfacc: Efficient sampling accelerates nerfs. *To Be Updated*, 2023.

[Low99] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[LPM+19] Wen Liu, Zhixin Piao, Jie Min, Wenhan Luo, Lin Ma, and Shenghua Gao. Liquid warping gan: A unified framework for human motion imitation, appearance transfer and novel view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[LZ20] Dazhong Lao and Shanshan Zhao. *Fundamental Theories and Their Applications of the Calculus of Variations*. Springer Nature, 2020.

[LZNH20] Yang Li, Tianwei Zhang, Yoshihiko Nakamura, and Tatsuya Harada. Splitfusion: Simultaneous tracking and mapping for non-rigid scenes. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5128–5134, 2020.

[MPJ+19] Mateusz Michalkiewicz, Jhony K. Pontes, Dominic Jack, Mahsa Baktashmotlagh, and Anders Eriksson. Implicit surface representations as layers in neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[MST+21] 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. *Commun. ACM*, 65(1):99–106, dec 2021.

[MT93] M. Maruyama and T. Teraoka. Recognition of 3d nonrigid objects by learning view change transformations. In *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*, volume 1, pages 219–222 vol.1, 1993.

[NFS15] Richard A. Newcombe, Dieter Fox, and Steven M. Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[PCPMMN21] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[PE90] Tomaso Poggio and Shimon Edelman. Poggio, t. & edelman, s. a network that learns to recognize 3d objects. nature 343, 263-266. *Nature*, 343:263–6, 02 1990.

[PSB+21] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2021.

[PZvBG00] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, page 335–342, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[RPLC22] Jack Richter-Powell, Yaron Lipman, and Ricky T. Q. Chen. Neural conservation laws: A divergence-free perspective. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[SD95] S.M. Seitz and C.R. Dyer. Physically-valid view synthesis by image interpolation. In *Proceedings IEEE Workshop on Representation of Visual Scenes (In Conjunction with ICCV'95)*, pages 18–25, 1995.

[SD99] Steven M Seitz and Charles R Dyer. Photorealistic scene reconstruction by voxel coloring. *International journal of computer vision*, 35:151–173, 1999.

[SF16] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[SGP+22] Liangchen Song, Xuan Gong, Benjamin Planche, Meng Zheng, David Doermann, Junsong Yuan, Terrence Chen, and Ziyan Wu. Pref: Predictability regularized neural motion fields. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXII*, pages 664–681. Springer, 2022.

[TCY+22] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P. Srinivasan, Jonathan T. Barron, and Henrik Kretzschmar. Block-nerf: Scalable large scene neural view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8248–8258, June 2022.

[TFT+20] A. Tewari, O. Fried, J. Thies, V. Sitzmann, S. Lombardi, K. Sunkavalli, R. Martin-Brualla, T. Simon, J. Saragih, M. Nießner, R. Pandey, S. Fanello, G. Wetzstein, J.-Y. Zhu, C. Theobalt, M. Agrawala, E. Shechtman, D. B Goldman, and M. Zollhöfer. State of the art on neural rendering. *Computer Graphics Forum*, 39(2):701–727, 2020.

[TS20] Richard Tucker and Noah Snavely. Single-view view synthesis with multiplane images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[WAO+09] Michael Wand, Bart Adams, Maksim Ovsjanikov, Alexander Berner, Martin Bokeloh, Philipp Jenke, Leonidas Guibas, Hans-Peter Seidel, and Andreas Schilling. Efficient reconstruction of nonrigid shape and motion from real-time 3d scanner data. *ACM Trans. Graph.*, 28(2), may 2009.

[XAS21] Hongyi Xu, Thiemo Alldieck, and Cristian Sminchisescu. H-nerf: Neural radiance fields for rendering and temporal reconstruction of humans in motion. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 14955–14966. Curran Associates, Inc., 2021.

[XTS+22] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond. *Computer Graphics Forum*, 2022.

[YLL23] Zhiwen Yan, Chen Li, and Gim Hee Lee. Nerf-ds: Neural radiance fields for dynamic specular objects, 2023.

[ZRSK20] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020.

Chapter 5

Appendix

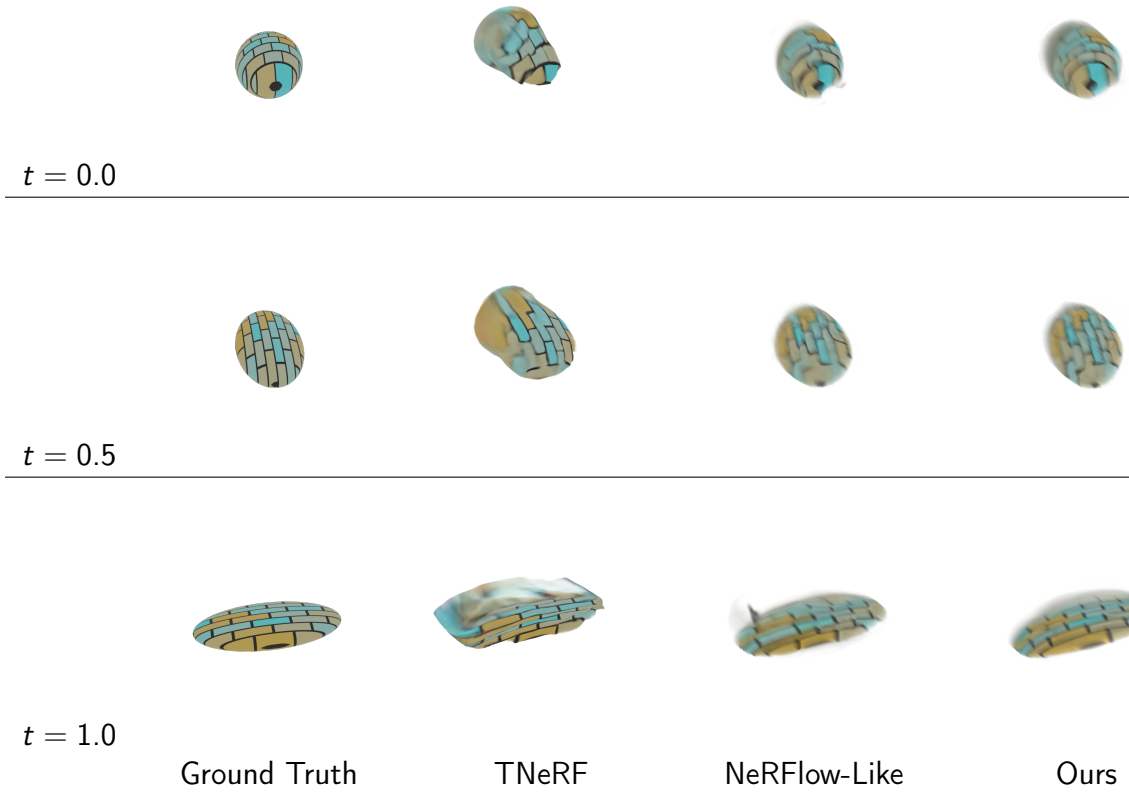


Figure 5.1: Spinning Ball Results

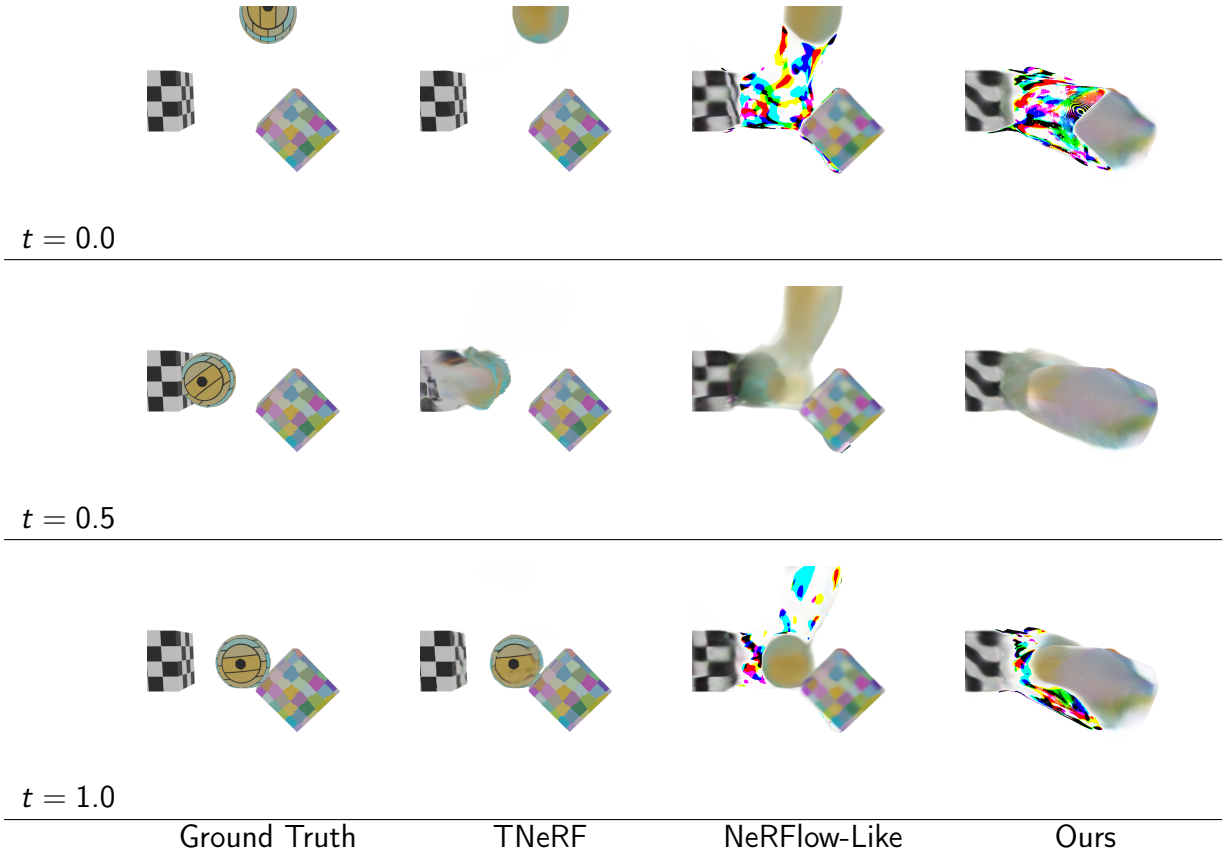


Figure 5.2: Soft Ball Results