EUROGRAPHICS 2022
D. Meneveaux and G. Patanè
(Guest Editors)

Volume 41 (2022), Number 2
STAR – State of The Art Report

Neural Fields in Visual Computing and Beyond

Yiheng Xie[1,2] Towaki Takikawa[3,4] Shunsuke Saito[5] Or Litany[4] Shiqin Yan[1] Numair Khan[1] Federico Tombari[6,7]
James Tompkin[1] Vincent Sitzmann[8†] Srinath Sridhar[1†]

[1]Brown University [2]Unity Technologies [3]University of Toronto [4]NVIDIA [5]Meta Reality Labs Research [6]Google [7]Technical University of Munich
[8]Massachusetts Institute of Technology [†]*Equal advising*

https://neuralfields.cs.brown.edu/

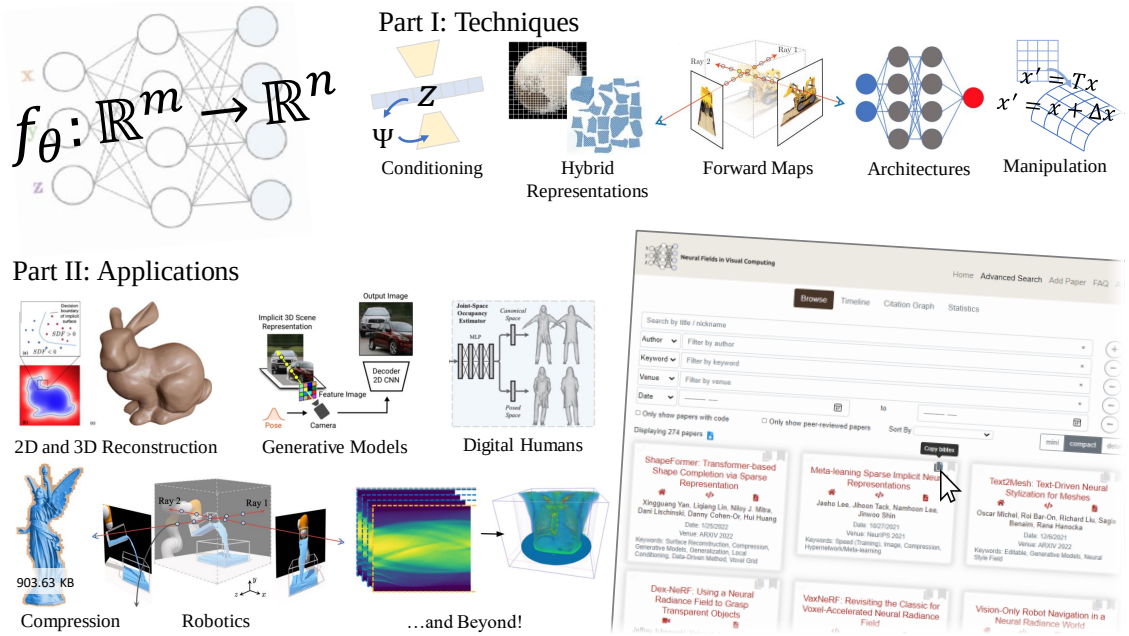

Figure 1: **Contribution of this report.** Following a survey of over 250 papers, we provide a review of (**Part I**) techniques in neural fields such as prior learning and conditioning, representations, forward maps, architectures, and manipulation, and of (**Part II**) applications in visual computing including 2D image processing, 3D scene reconstruction, generative modeling, digital humans, compression, robotics, and beyond. This report is complemented by a community-driven website with search, filtering, bibliographic, and visualization features.

Abstract

*Recent advances in machine learning have led to increased interest in solving visual computing problems using methods that employ coordinate-based neural networks. These methods, which we call **neural fields**, parameterize physical properties of scenes or objects across space and time. They have seen widespread success in problems such as 3D shape and image synthesis, animation of human bodies, 3D reconstruction, and pose estimation. Rapid progress has led to numerous papers, but a consolidation of the discovered knowledge has not yet emerged. We provide context, mathematical grounding, and a review of over 250 papers in the literature on neural fields. In **Part I**, we focus on* neural field techniques *by identifying common components of neural field methods, including different conditioning, representation, forward map, architecture, and manipulation methods. In **Part II**, we focus on* applications of neural fields *to different problems in visual computing, and beyond (*e.g., *robotics, audio). Our review shows the breadth of topics already covered in visual computing, both historically and in current incarnations, and highlights the improved quality, flexibility, and capability brought by neural field methods. Finally, we present a companion website that acts as a living database that can be continually updated by the community.*

CCS Concepts

• ***Computing methodologies*** → ***Machine Learning; Artificial Intelligence;***

1. Introduction

Visual computing involves the synthesis, estimation, manipulation, display, storage, and transmission of data about objects and scenes across space and time. In computer graphics, we synthesize 3D shapes and 2D images, render novel views of scenes, and animate articulating human bodies. In computer vision, we reconstruct 3D appearance, shape, object pose, and deformation. In human-computer interaction (HCI), we enable the interactive investigation of spacetime data. Beyond visual computing and into adjacent disciplines like robotics, we use the structure of 3D scenes to plan and execute actions. Within these disciplines, **fields** are widely used to continuously parameterize an underlying physical quantity of an object or scene over space and time. For instance, fields have been used to visualize physical phenomena [Sab88], compute image gradients [SK97], compute collisions [OF03], or represent shapes via constructive solid geometry (CSG) [Eva15].

In visual computing tasks, we solve many different optimization problems, including using machine learning and training data. One class of methods has gained significant attention since 2019: coordinate-based neural networks that represent a field. We refer to these methods as **neural fields**. Given sufficient parameters, fully connected neural networks can encode continuous signals over arbitrary dimensions at arbitrary resolution. Recent success with neural networks has caused a resurgence of interest in visual computing problems, leading to more accurate, higher fidelity, more expressive, and memory-efficient solutions. This has allowed neural fields to gain traction as a useful parameterization of 2D images [KAL*21], 3D shape [PFS*19, MON*19, CZ19], view-dependent appearance [SZW19, MST*20], and human bodies and faces [NMOG19, DLJ*20, SYMB21, YTB*21, RTE*21].

Such attention manifests as rapid progress and an explosion of papers (Figure 2), with a need to consolidate the discovered knowledge. However, a shared mathematical formulation to describe related techniques has not yet emerged, making it hard to communicate ideas and train students. Furthermore, there is "*selective amnesia*" [SC21] of older or even concurrent works, causing research repetition. Finally, rapid progress makes any survey quickly out-of-date, requiring new summarization approaches.

Following a survey of over 250 papers, we address the above issues by defining a neural field via fields of physical quantities, providing a shared mathematical formulation across common techniques, categorizing, describing, and relating many applications, and presenting a **living database** via a community website.

In **Part I**, we describe *neural field techniques* that are common across papers with a consistent notation and vocabulary. For instance, we identify and formalize recent hybrid discrete-continuous representations of neural fields, and techniques for learning priors such as local and global conditioning and meta-learning. Further, neural fields can be combined with a wide variety of differentiable forward maps, and we identify many such maps including surface and volume renderers and partial differential equations.

In **Part II**, we describe a broad cross-section of *applications of neural fields* to problems in visual computing. This lets us identify commonalities, connections, and trends across works representing shape and appearance of scenes and objects, including 3D reconstruction, digital humans, generative modeling, data compression, and 2D image processing. We also review works that solve tasks in adjacent communities including robotics (Section 11), medical imaging (Section 13.1), audio processing (Section 13.1), and physics-informed problems (Section 13.2).

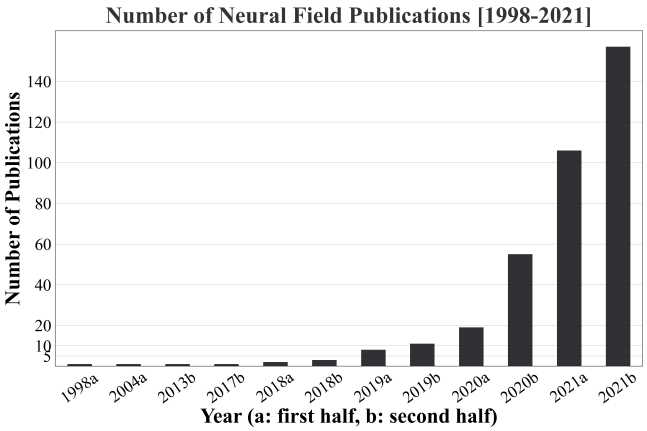

Figure 2: Neural fields were proposed two decades ago, yet their growth in visual computing has been concentrated in the last two years with over 250 papers.

Our living database is a companion website that provides search, filter, and visualization features to present the works described in this report (Figure 1), along with bibliography exporting features. The website allows our community to submit new works in neural fields and automatically updates the databases using keyword categorizations from Part I and II taxonomies. The website requires minimal maintenance, and its source code is open to allow future reports and surveys to provide the same functionality.

In summary, neural field techniques are powerful and widely applicable to visual computing problems and beyond. This state-of-the-art report provides the mathematical formulation of techniques and discussion of applications to make sense of this exciting area, along with a community-driven website to continue to help researchers keep track of new developments in the future.

1.1. Background

Fields Following the original definition in physics [FLS65, McM02] we define fields as the following:

> **Definition 1** A *field* is a quantity defined for all spatial and/or temporal coordinates.

We can represent a field as a function mapping a coordinate $\mathbf{x}$ to a quantity, which is typically a scalar or vector. Table 1 provides several examples. The physics community has also studied spinor and tensor fields. Finally, field coordinates are not limited to space and time, such as frequency coordinates in spectrograms. This review will focus on scalar and vector fields defined over spacetime, which are most relevant to visual computing.

In practice, the underlying field generation process may not have a known analytic form. Thus, functions may be described by parameters Θ that are hand crafted, optimized, or learned. We denote such a field as producing quantity $\mathbf{q} = \Phi(\mathbf{x}; \Theta)$. Furthermore,

Table 1: **Examples of fields in physics and visual computing.**

Examples	Field Quantity	Scalar/Vector	Coordinates
Gravitational Field	Force per unit mass (N/kg)	Vector	$\mathbb{R}^n$
3D Paraboloid: $z = x^2 + y^2$	Height z	Scalar	$\mathbb{R}^2$
2D Circle: $r^2 = x^2 + y^2$	Radius r	Scalar	$\mathbb{R}^2$
Signed Distance Field (SDF)	Signed distance	Scalar	$\mathbb{R}^n$
Occupancy Field	Occupancy	Scalar	$\mathbb{R}^n$
Image	RGB intensity	Vector	$\mathbb{Z}^2$ pixel locations x, y
Audio	Amplitude	Scalar	$\mathbb{Z}^1$ time t

we often index sampled functions using discrete values, such as at camera *pixels*, or choose discrete function parameterizations using *voxels* or discretized *level sets*. However, discrete parameterizations are limited by the Nyquist sampling rate, causing high memory requirements for 3D tasks. Adaptive grids like octrees and k–d trees can reduce memory, but their generation can lead to costly combinatoric optimizations.

Neural Networks A neural network connects many layers of artificial neurons to learn to non-linearly map a fixed-size input to a fixed-size output [LBH15]. A multi-layer perceptron (MLP) neural network can approximate any function through their learned parameters (universal approximation theorem [KA03]). Since 2010, there has been significant interest in using neural networks with corresponding investment in hardware and software to support neural networks. This has significantly lowered the barrier to apply neural networks to a wide range of problems.

Neural Fields Following the universal approximation theorem, any field can be parameterized by an MLP neural network. Thus:

> **Definition 2** A *neural field* is a field that is parameterized fully or in part by a neural network.

Neural fields are both continuous and adaptive by construction. Unlike the memory required for discrete parameterizations that scales poorly with spatio-temporal resolution, the memory required for neural fields instead scales with the number of parameters of the neural network—so-called network complexity. While other continuous parameterizations can represent large extents (for example, a Fourier series is a parameterization of a field), it is often difficult to know the required complexity ahead of time for efficient representation. Neural fields help to resolve this problem by using their parameters only where field detail is present. Neural fields are often parameterized as MLPs with activation functions whose gradients are well-defined. By their analytic differentiability, gradient descent, and over-parameterization [FC19], neural fields are effective at regressing complex signals through optimizations for problems that are otherwise ill-posed.

Our definition of a neural field does not include neural networks whose co-domain has a spatial extent, for instance, a network that outputs a grid of voxels. These methods often use convolutional layers and output a 2D or 3D grid of RGB color, occupancy, or latent features [STH*19, LSS*19]. While the regular grid output of these architectures can be seen as samplings of fields, the functions parameterized by the neural network are not neural fields as they do not ingest spatio-temporal coordinates.

Terminology Within visual computing, neural fields have been called *implicit neural representations*, *neural implicits*, or *coordinate-based neural networks*. In neuroscience, the term *neural field* may describe theories about the organization and function of the brain [CbGPW14]. The term also describes a regular grid of neurons in a neural network [Lem92]. We exclude works in these two areas as they do not follow from our definition.

1.2. Related Surveys and Articles

Within visual computing, there are survey papers on specific areas like neural rendering [TFT*20, TTM*21] and 3D reconstruction [ZSG*18], as well as lists of works in scene representations [NeRb, NeRa, Sit, NeRc]. Our survey aims to connect our visual computing communities together and share findings. Finally, neural fields are domain-agnostic and can model arbitrary quantities beyond shape and appearance. Neural fields have been used extensively in computational physics via physics-informed neural networks (PINNs) [RPK19a], with an existing survey [KKL*21].

Part I. Neural Field Techniques

A typical neural fields algorithm in visual computing proceeds as follows (Figure 3): Across space-time, we *sample coordinates* and feed them into a *neural network* to produce field quantities. The field quantities are samples from the desired *reconstruction domain* of our problem. Then, we apply a *forward map* to relate the reconstruction to the *sensor domain* (e.g. RGB image), where supervision is available. Finally, we calculate the reconstruction error or *loss* that guides the neural network optimization process by comparing the reconstructed signal to the sensor measurement.

During this process, many problems affect our ability to successfully reconstruct the signal. To better understand and apply neural fields, we identify five classes of techniques (Table 2). We can aid reconstruction from incomplete sensor signals via *prior learning and conditioning* (Section 2). We can improve memory, computation, and neural network efficiency via *hybrid representations* using discrete data structures (Section 3). We can supervise reconstruction via differentiable *forward maps* that transform or project our domain (*e.g.,* 3D reconstruction via 2D images; Section 4). With appropriate *network architecture* choices, we can overcome neural network spectral biases (blurriness) and efficiently compute derivatives and integrals (Section 5). Finally, we can *manipulate neural fields* to add constraints and regularizations, and to achieve editable representations (Section 6). Collectively, these classes constitute a 'toolbox' of techniques to help solve problems with neural fields.

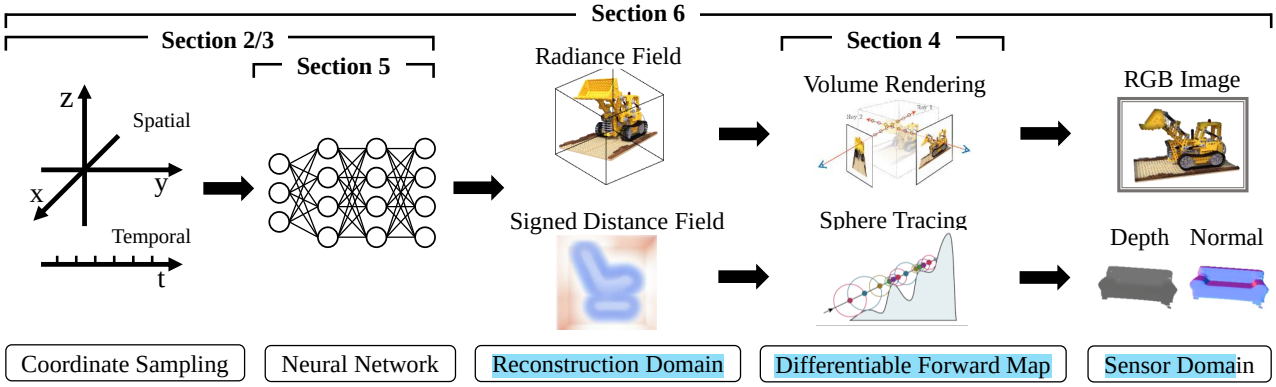

Figure 3: **A typical feed-forward neural field algorithm.** Spatiotemporal coordinates are fed into a neural network which predicts values in the reconstruct a domain. Then, this domain is mapped to the sensor domain where sensor measurements are available as supervision. Figures adapted from [MST*20, LZP*20].

Class and Section	Problems Addressed
Prior Learning and Conditioning (Section 2)	Inverse problems, ill-posed problems, edit ability, symmetries.
Hybrid Representations (Section 3)	Computation & memory efficiency, representation capacity, edit ability.
Forward Maps (Section 4)	Inverse problems.
Network Architecture (Section 5)	Spectral bias, integration and derivatives.
Manipulating Neural Fields (Section 6)	Edit ability, constraints, regularization.

Table 2: The five classes of techniques in the neural field toolbox each addresses problems that arise in learning, inference, and control.

2. Prior Learning and Conditioning

Suppose we wish to estimate a plausible 3D surface shape given a partial point cloud. This problem arises in reconstructing 3D street scenes from Lidar scans as we only observe points upon surfaces. To accomplish this task, we need a suitable prior over 3D surfaces. One option is to hand-craft a prior via heuristics such as smoothness or sparseness. However, this approach is limited in the complexity of heuristics that we can conceive. Alternatively, we can *learn* such a prior from data, and these can be encoded within the parameters and architecture (Section 5) of a neural network.

Beyond this, we might wish to adapt our prior based on specific conditions. In our street scene Lidar example, we might vary the expected surface shape of vehicles based on which kind of vehicle it is: a bicycle, a car, or a truck. For neural fields, this is accomplished by *conditioning* the neural field on a set of latent variables $\mathbf{z}$ that encode the properties of a specific field. By varying the latent variables, we can then vary the neural field. In this section, we discuss how to pose optimization problems that learn latent variables $\mathbf{z}$, how to infer $\mathbf{z}$ given a set of incomplete observations, and how to condition the neural field on $\mathbf{z}$ to decode them into different fields.

2.1. Conditional Neural Fields

A conditional neural field lets us vary the field by varying a set of latent variables $\mathbf{z}$. These latent variables could be samples from an arbitrary distribution, or semantic variables describing shape, type, size, color, etc., or come from an encoding of other data types such as audio data [GCL*21]. For example, by conditioning our neural fields on semantic variables which describe cars, we would like to decode these to a field that represents the shape and appearance of the corresponding car. Instance-specific information can then be encoded in the conditioning latent variable $\mathbf{z}$, while shared information can be encoded in the neural field parameters. If this latent variables are defined on a semantic or smooth space, they can be interpolated or edited.

In the following subsections, we discuss techniques to learn in an unsupervised manner what the latent variables $\mathbf{z}$ are, how to infer them given (partial) observations $\mathcal{O}$, and how to condition neural fields upon $\mathbf{z}$ to be decoded into a corresponding field. $\mathbf{z}$ is typically a low-dimensional vector, and is often referred to as a *latent code* or *feature code*. We discuss different encoding schemes (Section 2.1.1), both global and local conditioning (Section 2.1.2), and different mapping functions Ψ (Section 2.1.3).

2.1.1. Encoding the Conditioning Variable z

Feed-forward Encoders/Amortized Inference For feed-forward encoder methods, the conditioning latent code $\mathbf{z} = \mathcal{E}(\mathcal{O})$ is generated via an encoder $\mathcal{E}$, typically a neural network (Figure 4, left). The parameters in $\mathcal{E}$ can encode priors that can be pre-trained on data or auxiliary tasks. The decoder is the neural field that is conditioned by the latent code. This conditioning method is fast since inference requires only a single forward pass through the encoder and decoder. Examples of encoders for different data types include PointNet [QSMG17] for point clouds, ResNet [HZRS16] for 2D images, or VoxNet [MS15] for voxel grids.

Auto-decoders Only the decoder is explicitly defined, and encoding happens through stochastic optimization. That is, the latent code $\mathbf{z} = \arg\min_{\mathbf{z}} \mathcal{L}(\mathbf{z}, \Theta)$ is obtained by minimizing some loss function $\mathcal{L}$, which may be an expectation over a dataset (Figure 4,

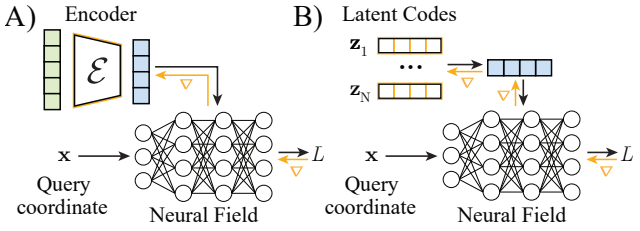

Figure 4: **Inference: Encoding vs. Auto-decoding.** (A) In amortized or encoder-based inference, encoder $\mathcal{E}$ maps observations $\mathcal{O}$ to latent variable $\mathbf{z}$. Parameters of $\mathcal{E}$ are jointly optimized with the conditional neural field. (B) Auto-decoding lacks an encoder, and each neural field is represented by a *separately-optimized* latent code $\mathbf{z}_i$, which are jointly optimized with the conditional neural field. Gradient flow displayed in orange.

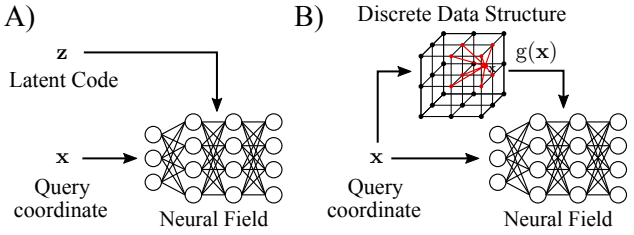

Figure 5: **Local vs. Global Conditioning.** (A) In global conditioning, a latent code $\mathbf{z}$ defines the neural field across *all* input coordinates $\mathbf{x}$. (B) In local conditioning, a discrete data structure provides a *coordinate-dependent* latent code $\mathbf{z} = g(\mathbf{x})$ such that the neural network is coordinate-dependent. Figure adapted from [PNM*20].

right). Every training observation is initialized with its own latent code $\mathbf{z}_i$. To optimize a particular latent code $\mathbf{z}_i$, we map it to neural field parameters Θ (Section 2.1.3), compute reconstruction loss, and back-propagate that loss to $\mathbf{z}_i$ to take a gradient descent step. At training time, the per-observation latent codes $\mathbf{z}_i$ and the neural field Φ_Θ are jointly optimized. At test time, given a new observation $\hat{\mathcal{O}}$, we freeze parameters of Φ and optimize the latent code $\hat{\mathbf{z}}$ that minimizes reconstruction error. Thus, auto-decoders solve $\arg\max_{\mathbf{z}} \mathbb{P}(\mathbf{z}|\mathcal{O})$ via first-order optimization; the optimization acts as encoder $\mathcal{E}$ to map observations $\mathcal{O}$ to latent variables $\mathbf{z}$.

Given the inference-time optimization, auto-decoding is significantly slower than a feed forward encoder approach. However, auto-decoding does not introduce additional parameters and does not make assumptions about observations $\mathcal{O}$. For instance, a 2D CNN encoder assumes $\mathcal{O}$ to be on a 2D pixel grid, whereas an auto-decoder can ingest tuples of pixel coordinates and colors independently of their spatial arrangement. This can lead to robustness in certain out-of-distribution scenarios. For instance, in 3D reconstruction from images when observing a camera pose not in the training set, a convolutional encoder is constrained by the 2D geometry of its kernels whereas an auto-decoder is not [SZW19, LLS*21]. Due to these benefits, several works have adopted the auto-decoder approach [PFS*19, GSL*20, RTE*21, YWC*21, LZZ*21, JA21, TTG*21, SRF*21].

Hybrid Approaches These approaches initialize $\mathbf{z}$ with a forward pass of a parametric encoder $\mathcal{E}$, then continue to optimize $\mathbf{z}$ iteratively via auto-decoding [MQK*21].

2.1.2. Global and Local Conditioning

In *global conditioning*, a single latent code $\mathbf{z} \in \mathbb{R}^n$ determines the entire neural field Φ across coordinate values (Figure 5, left). This is desirable in representation learning, when we want $\mathbf{z}$ to be *compact*—to encode as much information as possible. However, certain signals are not amenable to the learning of a single, continuous space of latent variables $\mathbf{z}$. Specifically, signals that are a *combination* of otherwise independent parts, such as the geometry of a room that is made up of separate objects. In such cases, representing the whole room via a global set of latent variables $\mathbf{z}$ means that these latent variables have to represent all the possible *configurations* of the room, which grows exponentially with the number of objects. As such, global latent codes succeed at modeling distributions with relatively few degrees of freedom, such as the shape and appearance of single objects or human bodies.

In *local conditioning*, we introduce multiple latent codes $\mathbf{z}$. Each $\mathbf{z}$ has spatial extent over a local neighborhood in the space of coordinates $\mathbf{x}$. Thus, latent codes are a function of $\mathbf{x}$: $\mathbf{z} = g(\mathbf{x})$ (Figure 5, right). Example g are discrete data structures like 2D raster grids [SHN*19, YYTK21, TY21], 3D voxel grids [PNM*20, LGL*20, JSM*20, CAPM20, CLI*20], surface patches [TTG*20], or orthographic 2D projections of 3D grids like floor maps [DBS*21, PNM*20]. As each latent code $\mathbf{z}$ only has to encode information about its *local* neighborhood, it consequently does not need to store information about the configuration anymore. For instance, imagine that we split a room into small 3D cubes, and in each cube store a latent code that describes the geometry in that cube. These latent codes do not need to encode all possible configurations of the room anymore.

Similarly, for local conditioning, an encoder $\mathcal{E}$ only has to encode local properties into latent variables $\mathbf{z}$. The encoder can preserve the spatial extent of the observation $\mathcal{O}$, such as 2D or 3D CNNs to extract features from images or discretized volumes. This can leverage encoder properties such as translation equivariance, granting better out-of-distribution generalization. On the other hand, latent codes $\mathbf{z}$ do not capture global scene information, which provides fewer constraints and less high-level control.

Hybrid Approaches Global and local conditioning can be combined. For instance, for human face images, to attempt to disentangle a property that is shared across instances (like hair color) from another that is region specific (like skin wrinkle).

2.1.3. Mapping z to Neural Field Parameters Θ

Given a set of latent variables $\mathbf{z}$, we now wish to use them to parameterize the neural network that represents the corresponding field. Different approaches have been proposed in prior work. Though on first sight, they seem incomparable, all of them follow the same principle of defining a function Ψ that maps latent variables $\mathbf{z}$ to a subset of neural network parameters $\Theta = \Psi(\mathbf{z})$ that then parameterize the neural field Φ_Θ. Different methods of conditioning differ in which parameters Θ are output by Ψ, as well as the form of Ψ itself. These design choices may impact generalization ability, parameter count, and computational cost.

Conditioning by Concatenation Neural fields can be conditioned on latent variables $\mathbf{z}$ by directly concatenating coordinate inputs $\mathbf{x}$ with $\mathbf{z}$. For instance, consider a neural field $\Phi : \mathbb{R}^2 \to \mathbb{R}^3$ that maps 2D pixel coordinates to RGB colors, and consider a vector of latent variables $\mathbf{z} \in \mathbb{R}^n$. Conditioning via concatenation would yield a conditional neural field $\Phi' : \mathbb{R}^{2+n} \to \mathbb{R}^3$, that takes as input concatenated coordinates $\mathbf{x}$ and latent codes $\mathbf{z}$.

It is not obvious how conditioning via concatenation falls into the framework defined above, where all conditional neural fields are expressed as predicting a subset of the parameters Θ of a neural field Φ_Θ via a function Ψ. However, conditioning via concatenation is equivalent to defining an affine function $\Psi(\mathbf{z}) = \mathbf{b}$ that maps latent codes $\mathbf{z}$ to the vector of biases $\mathbf{b}$ of the first layer of Φ [SCT*20, DPS*18, MGB*21]. In other words, in the case of conditioning via concatenation, the subset of parameters that is predicted by Ψ is only the biases of the first layer of the neural network Φ, and Ψ is parameterized as a simple affine mapping.

Hypernetworks Hypernetworks [HDL16] parameterize the function Ψ as a neural network that takes the latent code $\mathbf{z}$ as input and outputs neural field parameters Θ via a forward pass [SZW19, SMB*20, SRF*21, NWH21, CTT*21]. We can view this as a general form of conditioning because every other form of conditioning may be obtained from a hypernetwork by outputting only subsets of parameters Θ, by factorizing parameters of Φ via low-rank approximations or via additional scales and biases, or by varying Ψ architectures such as using only a single linear layer. For instance, conditioning via concatenation is a special case of a hypernetwork where Ψ is an affine transform and only the biases of the first layer of Φ are predicted. Full hypernetworks provide more complex embeddings of network weights than concatenation [GW20].

FiLM and Other Conditioning Between conditioning via concatenation and full hypernetworks, one can condition an MLP by predicting feature-wise transformations (FiLM) [DPS*18, CMK*21, MGB*21]. To FiLM-condition a neural field Φ, we use a network Ψ to predict a per-layer (and potentially per-neuron) scale γ and bias β vector from latent variables $\mathbf{z}$: $\Psi(\mathbf{z}) = \{\gamma, \beta\}$. The input $\mathbf{x}_i$ to the i-th layer Φ_i is transformed as $\Phi_i = \gamma_i(\mathbf{z}) \odot \mathbf{x}_i + \beta_i(\mathbf{z})$. Yet another trade-off in the size of the subset of neural field parameters predicted is struck by predicting the factors of a low-rank decomposition of the weight matrices of the neural field Φ [SIE21].

2.2. Gradient-based Meta-learning

An alternative to the conditional neural field approach is gradient-based meta-learning [FAL17]. Here, all neural fields in our target distribution are viewed as specializations of an underlying *meta-network* with parameters θ [SCT*20, TMW*21]. Individual instances are obtained from fitting this meta-network to a set of observations $\mathcal{O}$, minimizing a reconstruction loss $\mathcal{L}$ in a small number of gradient descent steps with step size λ:

$$\Theta_i^{j+1} = \Theta_i^j - \lambda \nabla \sum_{\mathcal{O}} \mathcal{L}(\Phi(\mathcal{O}; \Theta_i^j)), \quad \Theta_i^0 = \theta. \tag{1}$$

Similar to the auto-decoder framework in conditional neural fields, the inference function $\mathcal{E}$ is implemented via an iterative optimization algorithm. The meta-network can be seen as an initialization that is sufficiently close to all neural fields in the target distribution.

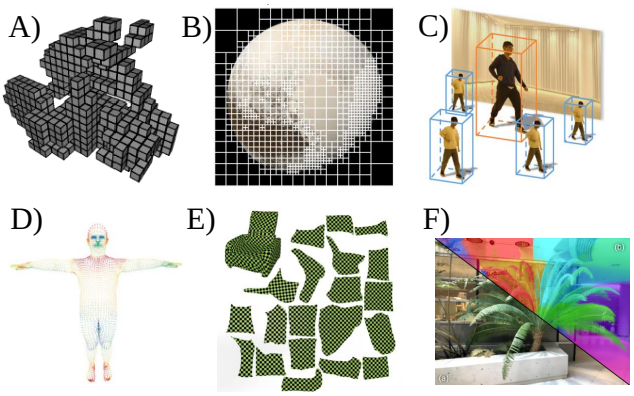

Figure 6: Examples of hybrid representations: (A) neural sparse voxel grid [LGL*20], (B) multi-scale voxel grid with neural 2D image compression [MLL*21], (C) object bounding boxes with neural radiance fields [ZLY*21], (D) mesh [PZX*21] (E) atlas [GFK*18b], and (F) Voronoi decomposition with neural radiance fields [RJY*21].

In a conditional neural field, the prior is expressed via the parameters of Ψ that enforce that the parameters of Φ lie in a low-dimensional space as defined by latent variable $\mathbf{z}$. However, in gradient-based meta-learning, the prior is expressed by constraining the optimization to not move the neural field parameters Θ too far away from the parameters of the meta-network θ. Gradient-based meta-learning enables fast inference, as only a few gradient descent steps are required to obtain Θ. As this does not assume a low-dimensional set of latent variables, in principle we retain the full expressivity of the neural field Φ.

3. Hybrid Representations

The second category of items in the neural field toolbox are *hybrid representations*. These combine neural fields with discrete data structures that decompose the space of input coordinates. This allows neural fields to scale up to large signals [RWG*13] (Figure 6). Discrete data structures are used extensively in visual computing, including regular grids, adaptive grids, curves, point clouds, and meshes. Discrete structures have several important benefits: 1) They typically reduce computation. For instance, bounding volume hierarchies (BVHs) [RW80, MOB*21] enable fast queries on hardware accelerators [PBD*10, WWB*14]. 2) They also allow for more efficient use of network capacity, since large MLP networks have diminishing returns in representation capacity [RJY*21]. 3) When representing geometry, discrete structures allow for empty space skipping, and so accelerate rendering. 4) Discrete structures are also suitable for simulation such as via finite element methods, and can help in manipulation and editing tasks (Section 6).

The general approach to spatial decomposition is to store some or all of the neural field parameters Θ_Φ in a data structure g. Given a coordinate $\mathbf{x}$, we query g to retrieve parameters Θ of the neural field. Two common approaches to how we map parameters Θ to data structure g are network tiling and embedding.

Network Tiling A collection of *separate* (usually small) neural fields Φ are tiled across the input coordinate space, covering disjoint regions. The network architecture is shared, but their parameters are distinct for each disjoint region. Given a coordinate $\mathbf{x}$, we simply look up the network parameters in the data structure g:

$$\mathbf{q} = \Phi(\mathbf{x}, \Theta) = \Phi(\mathbf{x}, g(\mathbf{x})). \quad (2)$$

Embedding We store latent variables $\mathbf{z}$ in the data structure, as in Section 2.1.2. Neural field parameters Θ become a function of the *local* embedding $\mathbf{z} = g(\mathbf{x})$ via a mapping function Φ:

$$\mathbf{q} = \Phi(\mathbf{x}, \Theta) = \Phi(\mathbf{x}, \Psi(\mathbf{z})) = \Phi(\mathbf{x}, \Psi(g(\mathbf{x}))) \quad (3)$$

Tiling is a special case of embedding where Ψ is the identity function and $\mathbf{z}$ is the parameters Θ. For design choice details of function Ψ, please refer to Section 2.

3.1. Defining g and its Common Forms

We define a discrete data structure g as a vector field that maps coordinates to quantities using a sum of Dirac delta functions δ:

$$g(\mathbf{x}) = \alpha_0 \delta(\mathbf{x}_0 - \mathbf{x}) + \alpha_1 \delta(\mathbf{x}_1 - \mathbf{x}) + \ldots + \alpha_n \delta(\mathbf{x}_n - \mathbf{x}), \quad (4)$$

where coefficients α_i (scalar or vector) are the quantities stored at coordinates $\mathbf{x}_i$. For uniform voxels, the coordinates $\mathbf{x}_i$ are distributed on a regular grid, whereas for a point cloud the coordinates $\mathbf{x}_i$ are distributed arbitrarily. For network tiling, α_i is an entire set of network parameters Θ_i; for embedding, α_i is latent variable $\mathbf{z}_i$.

Interpolation The discrete data structure g may use an *interpolation scheme* to define g outside the coordinates $\mathbf{x}_i$ of the Dirac deltas, such as nearest neighbor, linear, or cubic interpolation. If so, instead of Dirac deltas, function δ are basis functions of non-zero, compact, and local support. For instance, voxel grids are regularly combined with nearest-neighbor interpolation, where $g(\mathbf{x})$ is defined as α_i at the $\mathbf{x}_i$ closest to $\mathbf{x}$.

3.1.1. Regular Grids

Regular grids, such as 2D pixels and 3D voxels, discretize the coordinate domain with regular intervals. Their regularity makes them simple to index and apply standard signal processing techniques. Although simple, grids suffer from poor memory scaling in high dimensions, and the Nyquist-Shannon theorem requires dense sampling for high-frequency signals. To overcome this, grids can adaptive [MLL*21] or sparse [CLI*20] to focus the capacity around higher frequency regions, and can be implemented with data structures like hierarchical trees [LGL*20, TLY*21] and textures [SHN*19, PNM*20].

Grid tiling discretizes the coordinate domain with a grid and define each local region with smaller neural networks [RWG*13]. This can help learn larger scale signals [JK20] , make inference faster [RPLG21], and can be suitable for parallel computing [SJK21]. Tiling may increase overfitting given sparse training data [HJKK21], and tile boundary artifacts are possible, though network parameter interpolation can be used to reduce boundary artifacts [DVPL21, MMNM21a].

Grids of embeddings can similarly model larger-scale signals [CLI*20], enable the use of small neural networks [TLY*21], and benefit from interpolation [LGL*20]. Grids of embeddings can also be generated from other neural fields [MLL*21], generative models [ILK21], images [SSSJ20, TY21, CMK*21], or user input [HMBL21]. Please see Section 9 for a more detailed discussion.

3.1.2. Irregular Grids

Irregular grids discretize the coordinate domain with a grid that does not follow a regular sampling pattern (and hence avoding the Nyquist-Shannon sampling limit). These can be morphed to adaptively increase capacity in complex data regions. They may declare connectivity between coordinates explicitly such as in meshes, or implicitly such as in Voronoi cells. They too may also be organized into hierarchies, such as within a BVH or scene graph [GSRN21, OMT*21].

Point Clouds Point clouds are a collection of sparse discrete coordinates. Each location can hold an embedding [TTG*20] or a network [RJY*21]. Although sparse in its support, point clouds can volumetrically define regions through Voronoi cells via nearest neighbor interpolation. For continuous interpolation, we can use Voronoi cells with natural neighbor interpolation [Sib81] or soft-Voronoi interpolation [WPLN*20].

Object-centric Representations These are also a collection of points but where each has an orientation and a bounding box or volume [WSH*19]. Neural field parameters are stored at each point or at each vertex of the bounding volume, and can store embeddings [OMT*21] or networks [GFWF20, ZLY*21].

Mesh Meshes are a common data structure in computer graphics with well understood properties and processing operations. For triangle meshes, embeddings can be stored on vertices [PZX*21] and interpolated with barycentric interpolation. For complex polygons, mean-value coordinates [Flo03] and harmonic coordinates [JMD*07] are options.

4. Forward Maps

In many applications, the *reconstruction domain* (how we represent the world) is different from the *sensor domain* (how we observe the world). Solving an *inverse problem* recovers the reconstruction from observations obtained from sensors, i.e., finding the parameters Θ of a neural field Φ given observations from the sensor Ω.

We represent the (unknown) reconstruction as a neural field $\Phi : \mathcal{X} \to \mathcal{Y}$ that maps world coordinates $\mathbf{x}_{\text{recon}} \in \mathcal{X}$ to quantities $\mathbf{y}_{\text{recon}} \in \mathcal{Y}$. A sensor observation is often also a field $\Omega : \mathcal{S} \to \mathcal{T}$ that maps sensor coordinates $\mathbf{x}_{\text{sens}} \in \mathcal{S}$ to measurements $\mathbf{t}_{\text{sens}} \in \mathcal{T}$.

For instance, in 3D reconstruction from 2D images, the sensor domain contains camera images lying on a 2D raster grid, and the reconstruction domain could be some continuous 3D representation (e.g. SDF, radiance field). A 2D photograph from a camera is a field $\Omega : \mathbb{R}^2 \to \mathbb{R}^3$ representing irradiance. If we wish to model the reconstruction as a radiance field $\Phi : \mathbb{R}^5 \to \mathbb{R}^3$ of anisotropic photon flux, we need a mapping between Φ and Ω. We call such models *forward maps*; Figure 7 shows several examples.

Problem	Sensor	Sensor Domain	Forward Module	Reconstruction Domain	Literature
3D Reconstruction	Digital Camera	Image (2D discrete array)	Rendering	Geometry, Appearance	[SHN*19, LSS*19, MST*20, YKM*20]
Geodesy Estimation	Accelerometer	Gravitational acceleration	$F = m\vec{a} = Gm_1m_2/\vec{r}^2$	Density (mass)	[IG21]
CT Reconstruction	X-ray Detector	Projection domain	Radon Transform	Density/Intensity	[SLX*21, ZIL*21, SPX21]
MRI Reconstruction	RF Detector	Frequency domain	Fourier Transform	Density/Intensity	[SPX21]
Audio Reconstruction	Microphone	Waveform	Fourier Transform	Spectrogram	[GCM*21]
Synthetic Aperture Sonar	Microphone	Waveform	Convolution (w/ PSF)	Point Scattering Distribution	[RBBJ]

Table 3: Examples of forward maps. Detailed discussion on CT, MRI, Synthetic Aperture Sonar can be found in 13. RF: Radio Frequency, CT: Computed Tomography, MRI: Magnetic Resonance Imaging, PSF: Point Spread Function.

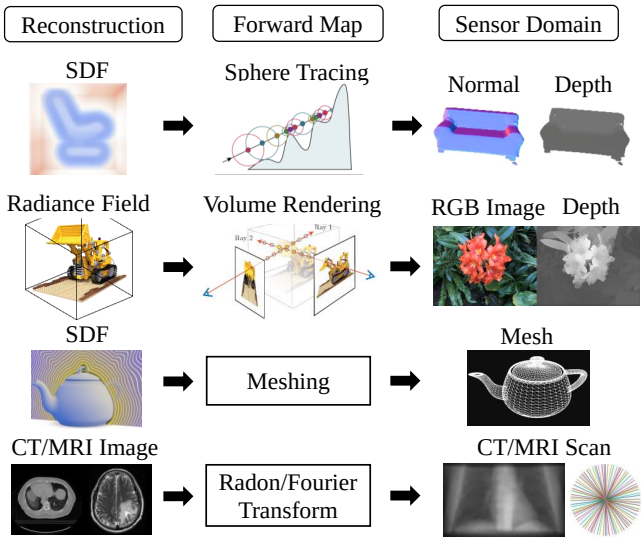

Figure 7: Forward maps relate reconstruction domains to sensor domains. Types of differentiable forward maps: a) sphere tracing [LZP*20], b) volume rendering [LSS*19, MST*20], c) meshing (e.g., marching cubes), d) Radon and Fourier Transformation [SPX21], e) partial derivatives [RPK19b]

A forward map is an operator $F : (\mathcal{X} \to \mathcal{Y}) \to (\mathcal{S} \to \mathcal{T})$, that is a mapping of functions. The forward map may depend on additional parameters, and may be composed with downstream operators such as sampling or optimization. We call a forward map *parameter differentiable* if for $\mathbf{y} = F(\Phi(\mathbf{x}))$ we can calculate the derivative $\frac{\partial \mathbf{y}}{\partial \theta}$, and *input differentiable* if we can calculate $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$.

Given a parameter differentiable forward map, we can solve the following optimization problem to recover the neural field Φ:

$$\arg\min_{\Theta} \int_{(\mathbf{x}_{\text{recon}}, \mathbf{x}_{\text{sens}}) \in (\mathcal{X}, \mathcal{S})} \| F(\Phi(\mathbf{x}_{\text{recon}})) - \Omega(\mathbf{x}_{\text{sens}}) \| \qquad (5)$$

using differentiable programming and an algorithm such as stochastic gradient descent.

In the 2D image example, the forward map can be modeled with equations of radiative transfer, which integrate over a volumetric anisotropic vector field. Neural Radiance Fields (NeRFs) [MST*20] are one example that recovers isotropic density and anisotropic radiance fields from 2D image measurements. Even though many inverse problems are ill-posed with no guarantee that a solution exists or is unique, empirically forward maps help us to find good solutions in a variety of applications (Part II).

4.1. Rendering

We define a *renderer* as a forward map which converts some neural field representation of 3D shape and appearance to an image. Renderers take as input camera intrinsic and extrinsic parameters, as well as the neural field Φ to generate an image. Extrinsic parameters define the translation and rotation of the camera, while intrinsic camera parameters define any other information for the image formation model such as field of view and lens distortion [HZ03]. Renderers often utilize a *raytracer* which takes as input a ray origin and a ray direction (*i.e.,* a single pixel from the camera model), and returns some information about the neural field. The information may be geometric like surface normals and intersection depth, or they may also aggregate or return some arbitrary features.

4.1.1. Ray-surface Intersection

If a neural field represents a shape's surface, we can use several methods to obtain geometric information such as the surface normal and intersection point. For common shape neural fields representations such as occupancy and signed distance fields, ray-surface intersection amounts to a root finding algorithm. One such method is ray marching, which takes discrete steps on a ray to find the surface. Given the first interval, a method such as the Secant method can be used to refine the root within the interval. This method will fail to converge at the right surface if the surface is thinner than the interval which discrete steps are taken. Taking jittered steps can alleviate this by stochastically varying the step length and using supersampling [DW85]. Interval arithmetic can be used to guarantee convergence at the cost of iteration count [FSSV07]. In some cases, a neural network may be employed within ray marching to predict the next step [SZW19, NSP*21]. If the surface is Lipschitz-bounded (as is the case for signed distance fields), then sphere tracing [Har96] can efficiently find intersections with guaranteed convergence. Segment tracing [GGPP20] can further speed up convergence at the cost of additional segment arithmetic computation at each step.

These methods are all differentiable, but naively back-propagating through the iterative algorithm is computationally expensive. Instead, the surface intersection point alone can be used to compute the gradient [NMOG20, YKM*20]. If a hybrid representation is used, we can exploit a bounding volume hierarchy to speed up raytracing. In some cases, the data structure can also be rasterized onto the image to reduce the number of rays.

4.1.2. Surface Shading and Lighting

Once the ray-surface intersection point has been retrieved, we can calculate the radiance contribution *from* the point towards the camera. This is done with a bidirectional scattering distribution function

(BSDF) [CT82, BS12] which can be differentiable or be parameterized as a neural field [GN98]. To aggregate lighting contribution *to* the point, the most phyiscally accurate method is to solve Kajiya's rendering equation [Kaj86] with a multi-bounce Monte Carlo algorithm [PJH16]. However, this can be especially computationally expensive combined with a neural field.

To overcome this issue, approximations of incident lighting (related to precomputed radiance transfer) can be used. This include cubemaps [Gre86] or the use of spherical basis functions such as spherical harmonics [Gre03]. Neural fields can also be employed to approximate incident lighting [WPYS21]. In the case where the exact material properties or the lighting environment does not need to be modeled, a neural field with position and view direction as input can directly model the radiance towards the camera [MST*20].

4.1.3. Volume Rendering

The surface rendering equation cannot model scenes with inhomogeneous media such as clouds and fog. Even for scenes with opaque surfaces, inverse surface rendering has difficulty recovering high frequency or thin geometry such as hair and other mesoscale details because the gradients are only defined at surfaces. Instead of Kajiya's rendering equation, volume rendering uses the *volume rendering integral* [KH84] based on the equations of radiative transfer [Cha13], for which the integral can be numerically approximated using quadrature (in practice, stochastic ray marching [PH89, MST*20]).

In a differentiable setting, under the assumptions of exponential transmittance, integration can be performed with a simple cumulative sum of samples across a ray, making backpropagation efficient and dense with respect to coordinates [MST*20]. This also propagates gradients throughout space, making the optimization easier. Non-exponential formulations exist [LSCL19, VJK21]; some are not physically accurate but work as an approximation.

One of the important factors in stochastic ray marching-based volume rendering is the number of samples. A higher sample count will mean more accurate models, but at the cost of computational cost and memory. Some approaches to mitigate this include using a coarse neural field model to importance sample [MST*20], using a classifier network [NSP*21], or using analytic anti-derivatives [LMW21b].

4.1.4. Hybrid Volume and Surface Rendering

Many differentiable renderers try to combine the strengths of a opaque surface-based renderer and a volume renderer. Volume rendering is typically under-constrained due to the stochastic samples taken on intervals, resulting in noise near surfaces. Surface rendering only provides gradients at the object surface, which do not smoothly propagate across the spatial domain.

We can combine approaches by re-parameterizing the implicit surface as a density field with soft boundaries using a Laplace distribution [YGKL21], a logistic density function [WLL*21], a Gaussian distribution, or a smoothed step function. We can also importance sample around the surface intersection point [OPG21].

4.2. Physics-informed Neural Networks

Partial differential equations (PDEs) are also powerful forward modules that map network outputs to gradient space supervision. Most of the works thus far have relied on a completely data-driven paradigm where additional constraints can be imposed by the choice of representation or implicit biases and invariance from the network architecture. Another class of methods, known as physics-informed neural networks (PINNs), use *learning bias* [KKL*21] which supervises boundary and initial values (from an incomplete simulation or observations) using a loss, and the rest of space by sampling or regularizing with equations of physics, typically partial differential equations (PDEs).

In visual computing, the PINN paradigm is often seen with signed distance functions [GYH*20]. One of the core properties of an SDF is that they satisfy the Eikonal equation:

$$\|\nabla u(x)\| = \frac{1}{f(x)},\ x \in \mathbb{R}^n \tag{6}$$

. The boundary values are the point cloud $\mathcal{X}$ which correspond to the 0-level set. In the special case when $f(x) = 1$, $u(x)$ becomes an SDF. The Eikonal loss is therefore: $\mathcal{L} = \sum_{\mathbf{x}\in\mathcal{X}} \|\|\nabla\Phi(x)\| - 1\|$.

PDEs are a natural description of the dynamics in the natural world. As such, a large collection of PDEs have been proposed in the discipline of physics, and naturally many of those PDEs have been used in conjunction with Neural Fields as PINNs. We refer readers to Karniadakis et. al for a much more comprehensive review of PINNs [KKL*21].

4.3. Identity Mapping Function

In some applications, the sensor domain may be the same as the reconstruction domain. In these cases, the forward model is the identity mapping function. The task is simply overfitting a neural field to data. Some examples are: supervising a neural signed distance field directly by ground-truth values [PFS*19], using neural fields to "memorize" images, or audio signals [SMB*20].

5. Network Architecture

The design choices that we make about the structure and components of a neural network have a significant impact on the quality of the field it parameterizes. The most obvious of these choices is the *network structure* itself, such as how many layers are in the MLP and how many neurons are in each layer. We assume that a reasonable structure with enough learning capacity has been chosen, and focus our discussion on other design decisions that provide inductive biases for effectively learning neural fields.

5.1. Overcoming Spectral Bias

Real-world signals are complex, making it challenging for neural networks to achieve high fidelity. Furthermore, neural networks are biased to fit functions with low spatial frequency [RBA*19, HMZ*21]. Several designs address this shortcoming.

Positional Encoding First proposed in the Natural Language Processing community, the coordinate input of the neural network may be transformed by a *positional encoding* $\gamma : \mathbb{R}^n \to \mathbb{R}^m$, which is a set of scalar functions $\gamma_i : \mathbb{R}^n \to \mathbb{R}$ which maps a coordinate vector $\mathbf{x} = [x_1, x_2, ..., x_n]^T \in \mathbb{R}^n$ to a vector of embedded coordinates:

$$\gamma(\mathbf{x}) = [\gamma_1(\mathbf{x}), \gamma_2(\mathbf{x}), ..., \gamma_m(\mathbf{x})]. \quad (7)$$

Sinusoidal functions are widely used to equip neural fields with the ability to fit high-frequency signals. Proposed by [ZBDB20a], they can be formally written as γ_i where:

$$\gamma_{(2i)}(x) = sin(2^{i-1}\pi x), \quad (8)$$

$$\gamma_{(2i+1)}(x) = cos(2^{i-1}\pi x). \quad (9)$$

These sinusoidal embeddings, also known as Fourier feature mapping, were subsequently popularized for the task of novel view synthesis [MST*20, TSM*20]. In the context of neural tangent kernels [JGH18], sinusoidal positional encoding can be shown to induce a kernel with a spatial selectivity that increases with the frequency of the functions ϕ_i. Positional encodings can thus also be seen as controlling the interpolation properties of a neural field.

Wang et al. [WLYT21] showed that the choice of the frequency of γ_i biases the network to learn certain, band-width limited frequency content, where lower encoding frequencies result in blurry reconstruction, and higher encoding frequencies introduce salt-and-pepper artifacts. For more stable optimization, one approach is to mask out high-frequency encoding terms at the beginning of the optimization, and progressively increase the high-frequency encoding weights in a coarse-to-fine manner [LMTL21, BHumRZ21]. SAPE proposed a masking scheme [HPG*21] that also allowed the encoding of spatially varying weights. Finally, alternative positional encoding functions ϕ_i have also been proposed [ZRL21, WLYT21, MRNK21]. Zheng et al. [ZRL21] conducted a comprehensive study of various positional encoding functions.

Activation Functions An alternative approach to enable the fitting of high-frequency functions is to replace standard, monotonic nonlinearities with periodic nonlinearities, such as the periodic sine as in SIREN [SMB*20] enabling fitting of high-frequency content. While a derivation of the properties of the neural tangent kernel is outstanding, some theoretical understanding can be gained by analyzing the relationships of the gradients of the output of the Neural Field with respect to neighboring coordinate inputs—for high frequencies of the sinusoidal activations, these gradients have been shown to be orthogonal, leading to an ability to perfectly fit values at any input coordinates without any interpolation whatsoever. It has been pointed out that positional encoding with Fourier features is equivalent to periodic nonlinearities with one hidden layer as the first neural network layer [BHumRZ21].

5.2. Integration and Derivatives

Derivatives A key benefit of neural fields is that they are highly flexible general function approximators whose derivatives $\nabla_{\mathbf{x}}\Phi(\mathbf{x})$ are easily obtained via automatic differentiation [G*89, BPRS18], making them attractive for tasks that require supervision of derivatives via PDEs. This, however, raises additional requirements in terms of the architecture of the MLP parameterizing the field.

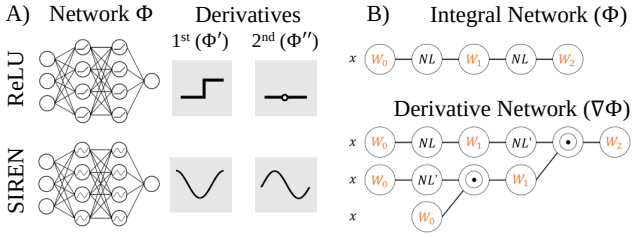

Figure 8: Derivatives and anti-derivatives of neural fields: A) Neural field derivatives depend on the activation function. ReLU yields (piece-wise) constant derivatives. Sinusoidal activations [SMB*20] yields (co-)sinusoidal derivatives. B) The computation graph of a neural network can be assembled via automatic differentiation, and shares parameters with its anti-derivative. This enables solving of optimization problems that require supervision of its integral [LMW21b]. NL and NL' denote a non-linear activation function and its derivative, and $\odot$ denotes element-wise multiplication. Figures adapted from [SMB*20, LMW21b].

Specifically, the derivatives of the network must be nontrivial (*i.e.,* nonzero) to the degree of the PDE (Figure 8, left). For instance, to parameterize solutions of the wave equation — a second-order PDE — the second-order derivatives of the neural field must be nontrivial. This places restrictions on the activation function used in the network. For instance, ReLU nonlinearity has piecewise constant first derivatives, and zero high-order derivatives, and so cannot generally parameterize solutions to the wave equation. Other common nonlinearities, such as softplus, tanh, or sigmoid, may address this issue, but often need to be combined with positional encodings to parameterize high-frequency content of neural fields. Alternatively, the periodic sine may be used as an activation function [SMB*20].

Integrals In some cases, we are further interested in solving an inverse problem where we measure the *derivative* quantity of a field, but are later interested in obtaining an expression for an integral over the neural field. In these cases, it is possible to sample the neural field and approximate the integral using numerical quadrature, as is often done for instance in volume rendering [MST*20]. However, the number of samples needed to approximate the integral may be arbitrarily large for a given accuracy.

While automatic differentiation is ubiquitous in deep learning, automatic integration is not well-explored. The seminal work AutoInt [LMW21b] proposed to directly parameterize the *antiderivative* of the field as a neural field Φ. We may then instantiate the computational graph of its *gradient network* $\nabla_{\mathbf{x}}\Phi(\mathbf{x})$, and fit the gradient network to the given derivative values. At test time, a single integral can be calculated in one forward pass, as a consequence of the universal theorem of calculus (Figure 8, right).

6. Manipulating Neural Fields

While many data structures in visual computing have post-processing tools (e.g. smoothing a polygonal mesh), neural fields have limited tools for editing and manipulation, which significantly limits their use cases. Fortunately, this is an active area of research.

6.1. Input Coordinate Remapping

The simplest approach to editing neural fields is to transform its spatial or temporal coordinate inputs. A simple rigid translation of a neural field can be expressed as $g(x) = f(x+b)$. We discuss other more sophisticated coordinate remapping methods below (Table 5).

6.1.1. Spatial Transformation via Explicit Geometry

An intuitive solution for controlling shape or appearance neural fields is to leverage explicit shape information. For object modeling problems (Section 3.1.2), structural priors are often available in the form of bounding boxes and coarse explicit geometry. Object bounding boxes offer a convenient, explicit handle to add and rigidly transform each object [ZLY*21, OMT*21, LGL*20].

For articulated objects, the kinematic chain offers explicit control over object geometry through its joint angles. At inference time, animating object geometry is achieved by changing joint angle inputs [MQK*21]. While this approach is effective for a small number of joints, it may introduce spurious correlations in case of long kinematic chains such as human body [LMR*15]. To avoid this issue, we can represent target shapes as the composition of local neural fields [GCS*20, DLJ*20]. Each local field is defined with respect to the joint transformations on a template mesh, and composed by max operation [DLJ*20] or weighted blending based on relative joint locations [SYZR21, NSLH21]. The primary drawback of articulated compositional neural fields is that each local field is independently modeled, often producing artifacts around joints with unobserved joint transformations. An alternative is to warp an observation space into a canonical space, where the reconstruction is defined, using the articulation of a template model [HXL*20, SYMB21]. The warping is commonly represented as linear blend skinning (LBS), where the deformations of surface is defined as the weighted sum of joint transformations. The blending weights are computed by a nearest-neighbor query on a template mesh [HXL*20] or learning as neural skinning fields [SYMB21, MZBT21, CZB*21, TSTPM21] (Table 5).

6.1.2. Spatial Transformation via Neural Fields

Unlike articulated objects with known transformations, modeling general dynamic scenes requires flexible representations that can handle arbitrary transformations. As target geometry and appearance are often modeled with neural fields, a continuous transformation of the field itself is a natural choice. Learning neural fields for spatial transformations is a highly under-constrained problem without 3D supervision [NMOG19]. This motivates the use of regularization loss terms based on physical intuitions:

- **Smoothness**: The first derivative of warp fields w.r.t. spatiotemporal coordinates should be smooth, assuming no sudden movements. This is necessary to constrain unobserved regions [GSKH21, PSB*21, TTG*21, DZY*21].
- **Sparsity**: 3D scenes generally contain large empty space. Thus, enforcing sparsity of the predicted motion fields avoids suboptimal local minima [GSKH21, XHKK21, TTG*21, DZY*21].
- **Cycle Consistency**: If a representation provides both forward and backward warping (e.g. [GSKH21, WELG21, LNSW21], forward/inverse LBS [SYMB21, MZBT21]), we can employ cycle consistency as a loss function. Unlike other regularization terms, cycle consistency does not dampen the prediction as the global optimum should also satisfy this constraint.
- **Auxiliary Image-space Loss**: Image-space information such as optical flows [LNSW21, XHKK21, DZY*21, GSKH21, WELG21] and depth maps [LNSW21, GSKH21, WELG21] can also be used in auxiliary loss functions.

6.1.3. Temporal Re-mapping

Conditioning the neural field on temporal coordinates allows time editing such as speed-up/slow-down ($\Phi(a \cdot t)$), offset ($\Phi(t+b)$), and reversal ($\Phi(-t)$) [ZLY*21, LNSW21].

6.2. Editing via Network Parameters

Neural fields can also be edited by directly manipulating the latent features or the learned network's weights. These methods are task-agnostic, and are applicable to editing geometry, texture, as well as other physical quantities beyond visual computing. A subset of parameters can be selectively modified (*e.g.,* geometry network, texture latent code). Network weight editing and manipulation methods include (see also Table 4):

- **Latent Code Interpolation/Swapping**: For neural fields conditioned on latent codes, interpolation or sampling in the latent space can change properties of the representation [CZ19].
- **Latent Code/Network Parameters Fine-Tuning**: After pre-training, we can fine-tune parameters to fit new, edited observations at test time. Editing may leverage explicit representations, such as sketches on 2D images [LZZ*21], or moving primitive shapes [HAESB20]. The neural field is coupled to the explicit supervision via differentiable forward maps.
- **Editing via Hypernetworks**: Hypernetworks can learn to map a new statistical distribution (*e.g.,* a new texture style [CTT*21]) to a pre-trained neural field, by replacing its parameters.

Part II. Applications of Neural Fields

In Part II of this report, we review neural field works based on their application domain. The recent explosion of interest has seen neural fields used for a wide range of problem in visual computing such as 3D shape and appearance reconstruction, novel view synthesis, human modeling, and medical imaging. Additionally, neural fields are increasingly being used in applications outside of visual computing including in physics and engineering. For each application domain, we limit our discussion to only neural field work while providing pointers to more traditional methods as context.

7. 3D Scene Reconstruction

The reconstruction of representations of 3D scenes from real-world measurements is critical for robotics and autonomous vehicles, and for graphics applications like games and visual effects. Unsurprisingly, the earliest work that we are aware of that uses neural fields was for 3D shape representation [LS04]. 3D scenes have properties including geometry, appearance, materials, and lighting for both static and dynamic parts. *Reconstruction* is the solution to an inverse problem that maps available observations to a representation.

Task	Updated Parameter	Update Method	References
Geometry editing	Latent code	Backprop	DualSDF [HAESB20]
Shape removal/color editing	Network, latent code	Backprop	EditNeRF [LZZ*21]
Texture style transfer	Appearance network	Hypernetwork	Chiang et al. [CTT*21]
Facial expression animation	Latent code	Interpolation/swap	Gafni et al. [GTZN21], FLAME-in-NeRF [ASS21]
3D Geometry and/or texture editing	Latent code	Interpolation/swap	SRN [SZW19], DVR [NMOG20], IDR [YKM*20]
Camera view, lighting, time interpolation	Latent code	Interpolation/swap	XFields [BMSR20]
Diffuse albedo	Latent code	Interpolation/swap	NeRFactor [ZSD*21]

Table 4: Neural fields can be edited by directly changing network parameters such as weights and latent features. We list related work that performs editing through parameter editing.

Warp Representation	Definition	Optimized Parameter	Related Works
Position	$\mathbf{x}' = \Phi(\mathbf{x})$	$\mathbf{x}'$	[CLCO*21, ASS21, NBB21]
Rigid Transformation	$\mathbf{x}' = \mathbf{T}\mathbf{x}$	$\mathbf{T}$	[NBB21, PSB*21, PSH*21]
Displacement	$\mathbf{x}' = \mathbf{x} + \Delta\mathbf{x}$	$\Delta\mathbf{x}$	References in caption
Velocity	$\mathbf{x}' = \mathbf{x} + \int_0^\tau \mathbf{v} dt$	v	[NMOG19, BMSR20]
Discrete Cosine Transform	$\mathbf{x}' = (\frac{2}{T})^{1/2} \sum_{k=1}^{K} \phi_{\mathbf{x},k} cos(\frac{\pi(2t+1)k}{2T})$	$\phi_{\mathbf{x},k}$	[WELG21]
Vector Field	$\mathbf{x}' = \mathbf{x} + t\mathbf{v}(\mathbf{x})$	$\mathbf{v}(\mathbf{x})$	[ANVL21]
Linear Blend Skinning (LBS)	$\mathbf{x}' = \sum_i \mathbf{T}_i w_i \mathbf{x}$	$\mathbf{w}$	[CZB*21, SYMB21, WMM*21]
Residual + LBS	$\mathbf{x}' = \sum_i \mathbf{T}_i w_i \mathbf{x} + \Delta\mathbf{x}$	$\Delta\mathbf{x}$	[LHR*21, TSTPM21]

Table 5: Spatial transformation of coordinates: x and x' are source and target coordinates; t is time; $\Phi : \mathbb{R}^m \rightarrow \mathbb{R}^3$ is neural deformation field; $\mathbf{T} \in \mathbb{R}^{4\times4}$ is rigid transformation matrix; $\Delta\mathbf{x} \in \mathbb{R}^3$ is displacement; $\mathbf{v} \in \mathbb{R}^3$ is velocity; $\phi_{\mathbf{x},k} \in \mathbb{R}$ is the k-th DCT coefficient at point $\mathbf{x}$; T is number of frames; $K \in \mathbb{Z}^+$ is number of DCT coefficients; $\mathbf{v}$ is flow vector; w_i are skinning weights given by the human body model. References: [LNSW21, PCPMMN21, YTB*21, TTG*21, PBT*21, WWZ*21, ZLY*21, GSKH21]

For 3D, available observations are discrete (due to sensors), often sparse (few images), incomplete (partial point clouds), residing in a lower dimension (2D images), or lack vital topological information (point clouds). In this section, we discuss reconstructing, displaying, and editing 3D scenes while identifying which techniques from Part I that they use.

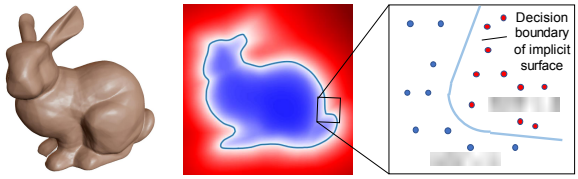

Figure 9: Neural fields for geometry commonly parameterize a surface as the level set of either an SDF [PFS*19, YBHK21], or occupancy function [MON*19, CZ19]. Figure adapted from DeepSDF [PFS*19].

7.1. Reconstruction of 3D Shape and Appearance

Reconstructing 3D Scenes with 3D Supervision A large amount of work is focused on reconstructing representations of geometry in the form of signed distance functions or occupancy functions, given 3D supervision. An example of 3D supervision are point clouds: A point cloud is a set $\mathcal{X} \subset \mathbb{R}^3$ of 3D points. Point clouds often originate from sampling an underlying surface (*e.g.,* with LiDAR). In addition to spatial location, they may contain information such as the surface normal, or appearance information.

Work on neural fields for geometry reconstruction often focuses on learned priors for reconstruction. AtlasNet [GFK*18b] proposed to represent 3D shapes via predicting a set of 2D neural fields that lift 2D local patch coordinates to 3D (an "atlas" of patches). Concurrently, FoldingNet [YFST18] proposed a similar idea but is not continuous. These patches were globally conditioned on a latent inferred from either a PointNet [QSMG17] or ResNet [HZRS16] encoder. IM-Net [CZ19] and Occupancy Networks [MON*19] proposed to represent 3D geometry via an occupancy function. Both proposed to generalize across ShapeNet [CFG*15] via global conditioning-via-concatenation, where the latent was regressed from either a CNN (from an image) or a PointNet encoder (from a point cloud). Concurrently, DeepSDF [PFS*19] proposed to represent 3D surfaces via their signed distance function, similarly generalizing across ShapeNet objects with global conditioning-via-concatenation, but performing inference in the auto-decoder framework instead. Figure 9 visualizes the SDF representation. Even though these methods were proposed in conjunction with learning a shape prior, it is possible to overfit a DeepSDF or neural occupancy function on a single 3D shape.

AtlasNet was further extended [DGF*19] by learning elementary structures in a data-driven manner, and inferring point correspondences across instances. Four concurrent papers proposed to leverage local features stored in voxel grids for local conditioning (Section 2.1.2) to improve on prior-based inference. IF-

Net [CAPM20], Chibane et al. [JSM*20], and Convolutional Occupancy Networks [PNM*20] use 3D CNNs or PointNet operating to process voxelized point clouds into embedding grids to locally parameterize an occupancy network or signed distance function, where conditioning proceeds via concatenation. In contrast, Chabra et al. [CLI*20] similarly leverages local conditioning from a 3D voxel grid, but infers the latent codes in the voxel grid via auto-decoding. CvxNet [DGY*20] proposes to represent a 3D shape as a composition of *implicitly defined* convex polytopes. LDIF [GCS*20] proposes to represent a 3D shape via a collection of local occupancy functions whose weighted sum represents the global geometry. Littwin et al. [LW19] infer an occupancy function via an image encoder and global conditioning via a hypernetwork.

Many methods aim to improve the quality of learned shapes. MetaSDF [SCT*20] was the first method to use gradient-based meta-learning (Section 2.1.3) for neural fields, using it to infer 3D signed distance functions from point clouds. Yang et al. [YWC*21] show that optimizing not only the embeddings but also the network weights regularized to be close to the original weights can better resolve ambiguities in unobserved regions. IGR [GYH*20] uses a neural field to parameterize an SDF, but instead of requiring ground-truth signed distance values in a identity forward map, they leverage a partial differential equation forward map via the Eikonal equation to learn SDFs from a point cloud. Atzmon et al. [AHY*19] derive analytical gradients of the 3D position of points on a level set with respect to the neural field parameters. SIREN [SMB*20] similarly learns an SDF with the Eikonal equation from oriented point clouds, but enables encoding of high-frequency detail by leveraging sinusoidal activations. Deep Medial Fields [RLS*21] proposes to parameterize geometry via its medial field, the local thickness of a 3D shape, enabling faster level-set finding and other applications. NGLoD [TLY*21] learns multiple level of details for SDFs with a compact hierarchical sparse octree of embeddings. SAL [AL20a] uses a neural field to parameterize SDF, and show that the SDF can be learned by optimizing against the unsigned distance function of the point cloud given suitable initialization. SALD [AL20b] extends this with supervision of normals. Neural Splines [WTBZ21] use as input oriented point clouds and use kernel regression of the neural field to optimize for normal alignment. PIFu [YYTK21] performs prior-based reconstruction of geometry and appearance reconstruction by extracting features from images with a fully convolutional CNN, and, when querying a 3D point, projecting it on the image plane to use image features for local conditioning-via-concatenation. Concurrently, DISN [XWC*19] enhances single-view reconstruction by using a camera pose estimation that allows to project 3D coordinates onto the image plane and gather local CNN features. Combined with the global feature, the result is a more accurate SDF.

Differentiable Rendering A major breakthrough in 3D reconstruction was the adoption of differentiable rendering (Section 4), which allowed reconstruction of 3D neural fields representing shape and/or appearance given only 2D images, instead of 3D supervision. This has significant implications since 3D data is often expensive to obtain, while 2D images are omni-present. A particularly important social implication is that non-experts can become 3D content creators, without the barrier of specialized hardware

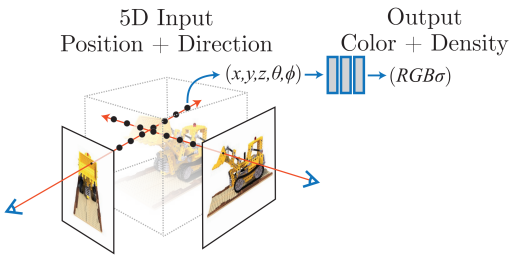

Figure 10: NeRF parameterizes a 3D scene as a 3D neural field mapping 3D coordinates to radiance and density, and can be rendered via volume rendering. Figure adapted from [MST*20].

or capture rigs. SRNs [SZW19] proposed using a differentiable sphere-tracing based renderer to reconstruct 3D geometry and appearance from only 2D observations. It leveraged global conditioning via a hypernetwork and inference via auto-decoding to enable reconstruction of a 3D neural field of geometry and appearance from only a single image for the first time. Concurrently, Liu et al. [LSCL19] use a CNN to predict an embedding to parameterize occupancy, and a form of differentiable volume rendering to produce a silhouette. Similar to DeepSDF, Occupancy Networks, and IM-Net, SRNs were designed to generalize, although they can be overfit given lots of 2D observations of a single 3D scene.

SDF-SRN [LWL20] enables learning an SRN from only a single observation per object at training time by enforcing a loss on the 2D projection of the 3D signed distance function. Kohli et al. [KSW20] leverages SRNs as a representation learning backbone for self-supervised semantic segmentation. Liu et al. [LZP*20] similarly reconstruct 3D geometry from 2D images with differentiable sphere-tracing. DVR [NMOG20] represents geometry via an occupancy function, and finds the zero-level set via ray-marching, subsequently querying a texture network for RGB color per ray. Importantly, they derive an analytical gradient of the ray-marcher, which significantly reduces memory consumption at training time, but also requires ground-truth foreground-background masks. They demonstrate the learning of appearance and shape priors across scenes via encoder-based conditioning to enable reconstruction from a single image, as well as overfitting on single scenes for high-quality, watertight 3D reconstruction.

NeuralVolumes [LSS*19] first proposed differentiable volume rendering, but leverages linearly interpolated voxel grids of color and density as a representation. Though the paper mentions parameterization of color and density functions as neural fields, this was only a part of the ablation studies, and reportedly underperformed a 3D CNN decoder in terms of resolution. cryoDRGN [ZBDB20b] implemented a differentiable volume renderer for the cryo-electron microscopy forward model to reconstruct protein structure from cryo-electron microscopy images, and proposed the positional encoding to allow fitting of high-frequency detail. Here, the neural field is parameterized in the Fourier domain, and the forward model is implemented via the Fourier slice theorem. NeRF [MST*20, TSM*20] combined volume rendering with a single ReLU MLP, parameterizing a monolithic neural field, and added positional encodings (Figure 10). By fitting a single neural field to a large number of images of a single 3D scene, this

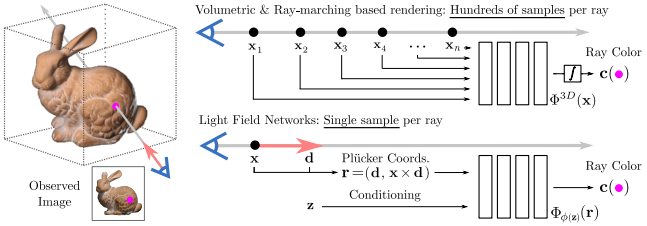

Figure 11: Instead of encoding a 3D scene with a 3D neural field that requires hundreds of samples along a ray to render, appearance & geometry may be encoded as a neural light field directly mapping an oriented ray to a color, enabling rendering with a single sample per ray, but requiring a multi-view consistency prior. Figure adapted from [SRF*21].

achieved photo-realistic novel view synthesis from only 2D images of arbitrary scenes for the first time. The visual quality and elegant formulation of NeRF has since inspired a large collection of follow-up work. Nerf++ [ZRSK20] improves representation of unbounded 3D scenes via an inverted-sphere background parameterization. Reizenstein et al. [RSH*21] and Arandjelović et al. [AZ21] propose attention-based accumulation of samples along a ray. DoNeRF [NSP*21] proposes to jointly train a NeRF and a ray depth estimator for fewer samples and faster rendering at test time. [GKJ*21, YLT*21a, RPLG21, HSM*21, LGL*20] propose various variants of local conditioning (without generalization, for overfitting a single scene) to speed up the rendering of NeRFs. Mip-NeRF [BMT*21] proposes to control the frequency of the positional encoding for multi-scale resolution control. NeRF−−, BARF and iNeRF [WWX*21, LMTL21, YCFB*20] propose to back-propagate into camera parameters to enable camera pose estimation given a reasonable initialization.

PixelNeRF [YYTK21] and GRF [TY21] perform prior-based reconstruction by extracting features from images with a fully convolutional CNN, and, when querying a 3D point, projecting it on the image plane to use image features for local conditioning-via-concatenation, similar to PIFu and DISN [SHN*19, XWC*19]. With more context views, a similar approach can be used for multi-view-stereo-like 3D reconstruction [CXZ*21, CBLPM21]. NeRF-VAE embeds a globally-conditioned NeRF with encoder-based inference in a VAE-like framework [KSZ*21]. While volume rendering has better convergence properties than surface rendering and enables photorealistic novel view synthesis, the quality of the reconstructed geometry is worse, due to the lack of an implicit, watertight surface representation. IDR [YKM*20] leverages an SDF parameterization of geometry, a sphere-tracing based surface-renderer, and positional encodings to enable high-quality geometry reconstruction. Kellnhofer et al. [KJJ*21] distill an IDR model into a Lumigraph after rendering to enable fast novel view synthesis at test time. Concurrently, UNISURF [OPG21], NeuS [WLL*21], VolSDF [YGKL21] propose to relate the occupancy function of a volume to its volume density, thereby combining volume rendering and surface rendering, leading to improved rendering results and better geometry reconstruction. Ray marching requires many samples along a ray to faithfully render complex 3D scenes. Even for relatively simple scenes, rendering requires hundreds or even thousands of evaluations of the neural scene representation per ray.

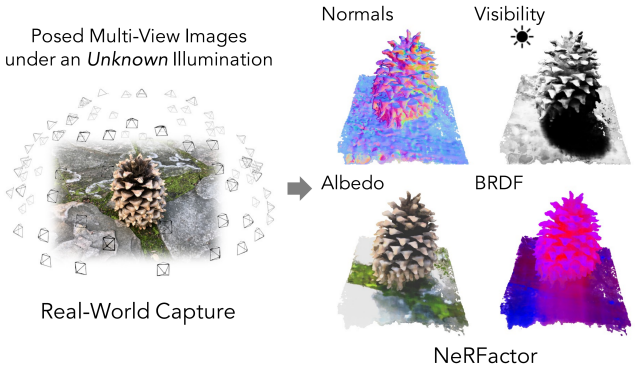

Figure 12: NeRFactor proposes neural fields for light visibility, BRDF, albedo, and surface normals, enabling re-rendering of 3D scenes under varying illumination and lighting following reconstruction. Figure adapted from [ZSD*21].

To overcome this limitation, we can parameterize the *light field* of a scene, which maps every ray to a radiance value. This enables real-time novel view synthesis with a single neural field sample per ray, and geometry extraction from the neural light field without ray-marching. Figure 11 displays the difference between ray-marching and rendering in light fields.

To prevent overfitting to the input views and allow view synthesis, we can learn multi-view consistency via global conditioning, hypernetworks, and inference via auto-decoding [SRF*21], or through ray embedding spaces [AHZ*22]. Alternatively, *densely* sampled rays of a single scene allow easier overfitting for view synthesis. NeuLF [LLYX21] parameterizes forward-facing scenes via light fields, overfitting on single scenes, and addresses multi-view consistency via enforcing similarity of randomly sampled views with the context views in the Fourier domain. Similar to light fields, NeX [WPYS21] parameterizes a set of multi-plane images as a 2D neural field, where the network inputs are pixel locations. Rendering is computationally efficient without ray marching.

7.2. Reconstruction of Scene Material and Lighting

The goal of *material reconstruction* is to estimate the material properties of a surface or participating media from sparse measurements such as images. For opaque surfaces, this may be the parameters of a bidirection scattering distribution function (BSDF). For participating media, this may be phase functions. This is a difficult problem, because materials are diverse and create complex light transport effects. Even just for BSDFs, there is a whole taxonomy [MDH*20] of characteristic properties. In addition, to accurately estimating materials, we must also perform *lighting reconstruction* or have prior knowledge of the lighting.

The forward modeling of light transport involves surface rendering [Kaj86] and volume rendering [KH84] for participating media, which both equations having recursive integrals with no closed form solution for forward modeling. Approaches to inversely solve these equations differ in the degree of approximation they make. Because neural networks are general function approximators, they can be useful for estimating arbitrary functions and integrals that

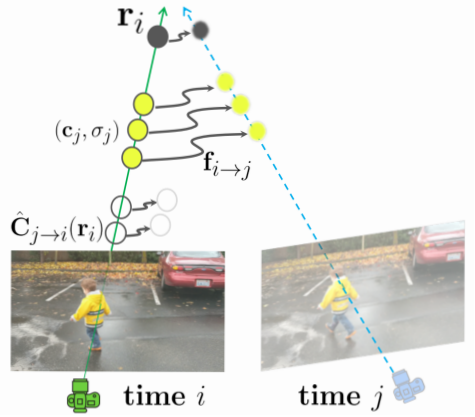

Figure 13: NSFF proposes novel time and view synthesis for dynamic scenes by reconstructing the 3D scene via a time dependent neural field. Figure adapted from [LNSW21].

are hard to solve. Some methods reconstruct *appearance* as approximate incident radiance, and other methods attempt to separate appearance into *materials* via explicit scattering distribution functions and *lighting*, enabling applications such as relighting.

Neural Fields of Material Parameters Many papers use a neural field to parameterize the parameter space of existing material models. Neural Reflectance Fields [BXS*20a] can reconstruct both the SVBRDF and geometry by assuming a known point light source, and use a neural field which parameterizes density, normals, and parameters of a microfacet BRDF [WMLT07] with a volume rendering forward map with one bounce direct illumination. NeRV [SDZ*21] extends this to handle more varied lighting setups with an environment map and a one bounce indirect illumination forward map, along with an additional neural field which parameterizes visibility, but assume the environment map is known a priori. NeRD [BBJ*21] also use a neural field which paramterizes density, normals, and the parameters of a Disney BRDF model [CT82, BS12], but remove any assumptions on the lighting by using spherical gaussians to represent lighting of which the parameters are directly optimized. They do not model visibility or indirect illumination. PhySG [ZLW*21] also directly optimize spherical gaussians, but uses a neural field to parameterize an SDF and a Ward BRDF [War92] along with a hard-surface-based forward map to improve surface reconstruction accuracy. NeRFactor [ZSD*21] learns an embedding-based neural field of BRDF parameters as a prior on the material parameter space (Figure 12).

7.3. Dynamic Reconstruction

In addition to novel-view synthesis, dynamic scene reconstruction also allows measurement, mixed reality, and visual effects. The challenges of modeling dynamic scenes are that the input data is even sparser in spacetime and often no 4D ground-truth data are available. With careful design choices, regularization loss terms, and the strong inductive bias of neural fields, several works have proposed solutions to this inverse problem. Figure 13 shows one of such approaches. We review the existing approaches based on warp representation and how to embed temporal information. For dynamic reconstruction of humans, please refer to Section 8.

Embedding Modeling temporally changing objects or scenes requires additional embedding that encodes frame information [RBZ*20]. Occupancy Flow [NMOG19] models a dynamic component as a neural field conditioned by normalized temporal coordinates. Similarly dynamic scene reconstruction methods based on differentiable rendering often condition neural fields with temporal coordinates [XHKK21, LNSW21, PCPMMN21, DZY*21]. Another approach is to jointly optimize per-frame latent code [PSB*21, LSZ*21, TTG*21, PSH*21, ALG*21] as embedding. While embedding based on temporal coordinates automatically incorporates temporal coherency as inductive bias, per-frame latent codes can enable the captures of more scene details.

Warp Representation To model dynamic scene from limited input data, we can split the problem into modeling a scene in the canonical space and warping it into each time frame. In Section 6.1.2, we provide common warp representations and techniques to regularize the warp fields. While most approaches use the learned warping function as the final scene representation, several works use warp fields only for regularizing the predicted radiance fields [LNSW21, GSKH21]. Unlike other dynamic reconstruction approaches, where non-rigid complex warping is modeled by neural fields, STAR [YLSL21] models motion as a global rigid transformation to primarily focus on tracking of a foreground object.

8. Digital Humans

Human shape and appearance has received special attention in computer vision and graphics in the last decade, and is one of the most popular application areas of neural fields. The adoption of neural fields has produced unprecedentedly high-quality synthesis and reconstruction of human faces, bodies, and hands. The state-of-the-art continues to evolve quickly.

Face Modeling The data-driven parametric morphable model was introduced by Blanz and Vetter [BV99]. However, these explicit representations lack realism and impose topological limitations making it difficult to model hair, teeth, etc. The expressiveness of fields, such as SDF and radiance fields, has made them an excellent candidate to address these limitations.

Given a collection of high-quality 3D scans, i3DMM [YTB*21] used coordinate-based neural networks that predict SDF and color to develop a 3D morphable head model with hair. Contrary to mesh-based representation, they learn an implicit reference shape as well as a deformation for each shape instance enabling editing abilities by disentangling color, identity, facial expressions, and hairstyle. Ramon et al. [RTE*21] use coordinate-based neural networks building upon IDR [YKM*20] to reconstruct a 3D head. To reduce the required amount of input images, a pre-trained DeepSDF-based latent space is used to regularize the test-time optimization. SIDER [CAMNS21] incorporates coarse geometric guidance via a fitted FLAME model [LBB*17] followed by implicit differentiation, enabling single-view optimization of facial geometry using SDF parameterized by a coordinate-based network.

While the aforementioned approaches focus on geometric accuracy assuming hard surfaces, NeRF have recently been adopted for photo-realistic view synthesis of human heads. Nerfies [PSB*21] utilizes a casual video footage captured with a moving hand-held camera to learn radiance fields and deformation fields of a human head. HyperNeRF [PSH*21] extends Nerfies by incorporating auxiliary hyper dimensions to handle large topological change.

Another line of works enable the semantic control of radiance fields by conditioning on head pose and facial expression parameters obtained from a 3D morphable face model [GTZN21, ASS21]. Supervised by multi-view video sequences, Wang et al. [WBL*21] use a variational formulation to encode dynamic properties in spatially varying animation codes stored in voxels. Notably, the above methods require per-subject training, and reconstruction from limited observations remains challenging. Several works show promise in few-shot, generalizable reconstruction by exploiting test-time fine-tuning [GSL*20] or pixel-aligned image features [RZS*21].

Body and Hand Modeling Neural fields have demonstrated efficacy in 3D reconstruction of clothed humans from image inputs [SHN*19, SSSJ20, HCJS20, HXL*20, ZYLD21] or point clouds [CAPM20, BSTPM20a]. Due to substantial variations in shape and appearance of clothed human bodies, a global latent embedding does not lead to plausible reconstruction. PIFu [SHN*19] addresses this by introducing a coordinate-base neural network conditioned on pixel-aligned local embeddings. Its followup works also employ the framework of PIFu for human digitization tasks from RGB inputs [LXS*20, HCJS20, SJL*21, YWM*21, HZJ*21] or RGB-D inputs [LYP*20, YZG*21]. To further improve the fidelity of reconstruction, multi-level feature representations are shown effective [SSSJ20,CAPM20]. Yang et al. [YWM*21] jointly predict skinning fields and skeletal joints for animating the reconstructed avatars. Also, several works explicitly leverage a parametric body model such as SMPL [LMR*15] to improve robustness under different poses [ZYLD21] and to enable an animation-ready avatar reconstruction from a single image [HXL*20, HXS*21].

Human bodies are dynamic as they are both articulated and deformable. Several works show that providing the structure of human bodies significantly improves the learning of radiance fields [PZX*21, PDW*21, CZK*21, LHR*21]. Given a fitted body model to images, Neural Body [PZX*21] diffuses the latent embeddings on the body via sparse convolution, which are inputs to the neural field. Neural Actor [LHR*21] projects queried 3D points onto the closest point on the fitted body mesh, and retrieves latent embeddings on the UV texture map. A-NeRF [SYZR21] explicitly incorporates joint articulations, and learns radiance fields in the normalized coordinate space. In addition, as shown for face modeling, pixel-aligned local embeddings are highly effective to support novel-view rendering of unseen subjects [SZZ*21, KKCF21].

Template-mesh registration is another important task in body modeling. IP-Net [BSTPM20a] proposes to jointly infer inner body and clothing occupancy fields given input scans to aid registration. 3D-CODED [GFK*18a] proposes Shape Deformation Networks to fit a template to a target shape and infer correspondences. Halimi et al. [HIL*20] show template-based shape completion by conditioning the deformation field on the part and whole encoding. LoopReg [BSTPM20b] presents a self-supervised learning of dense correspondence fields on the predicted implicit surface to a template human mesh, which improves the robustness of surface registration. A continuous local shape descriptor for dense correspondence is proposed by Yang et al. [YLB*20]. Wang et al. [WGT21] model occupancy fields of input scans in an un-posed canonical space by predicting piece-wise transformation fields (PFT). Since shapes are modeled in the canonical space, we can perform template registration without self-intersection of different body parts.

Lastly, several recent works model a parametric model of human bodies [MZBT21, AXS21], clothed human [SYMB21, TSTPM21, PBT*21, WMM*21], hands [KYZ*20], or clothing [CPA*21] as neural implicit surfaces. One unique property of human body is the articulation with non-rigid deformations. Occupancy flow [NMOG19] models the non-rigid deformation of human bodies by warp fields. NASA [DLJ*20] presents articulated, per-body-part occupancy fields to model a pose-driven human body. However, articulated occupancy fields may suffer from artifacts around joints due to discontinuities. Another approach to handle articulation is jointly learning shapes in the un-posed canonical space and transformations from the canonical space to the posed space [SYMB21, MZBT21, PBT*21, CZB*21], leading to continuous deformations around body joints. The transformation can be modeled as warp fields (Section 6.1.2) in the form of displacements [PBT*21] or Linear Blend Skinning weights [SYMB21, MZBT21, CZB*21, TSTPM21, WMM*21].

9. Generative Modeling

Assuming a dataset of samples drawn from a distribution $\mathbf{y} \sim \mathcal{D}$, generative modeling defines a latent distribution $\mathcal{Z}$, such that every sample $\mathbf{y}$ can be identified with a corresponding latent $\mathbf{z} \sim \mathcal{Z}$. The mapping from $\mathcal{Z}$ to samples from $\mathcal{D}$, is performed via a learned generator $\mathbf{g}$, parameterized as a deep neural network, $\mathbf{g}(\mathbf{z}) = \mathbf{y}$.

Neural fields provide greater flexibility for generative modeling because their outputs can be sampled at arbitrary resolutions, and because input samples can be shifted by simply applying transforms to the input coordinates. Each sample $\mathbf{y}$ is associated with a neural field Φ that can be densely queried across the coordinate domain to obtain the sample $\mathbf{y}$. Consistent with the notation in Section 2, latent variables $\mathbf{z}$ can either *globally* or *locally* condition the neural field Φ, yielding a conditional neural field $\Phi(\mathbf{x}, \mathbf{z})$.

Generative Modeling of Images In generative modeling of images, samples $\mathbf{y}$ are images, and we assume that there exists a distribution $\mathcal{D}$ of "natural" images. The neural field Φ maps 2D pixel coordinates to RGB colors, $\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$. Bond-Taylor and Willcocks [BTW21] adopt a single-step gradient-based meta-learning approach for variational inference of images with a SIREN [SMB*20] decoder. INR-GAN [SIE21] leverage global conditioning and a hypernetwork predicting a low-rank decomposition of the weight matrices of Φ. CIPS [ADK*21] and StyleGAN3 [KAL*21] leverage global conditioning with a FiLM-style conditioning. The latter three approaches leverage Fourier Features [TSM*20] to allow generation of high-frequency content. At the time of writing this report, neural field-based image generative models are the state-of-the-art.

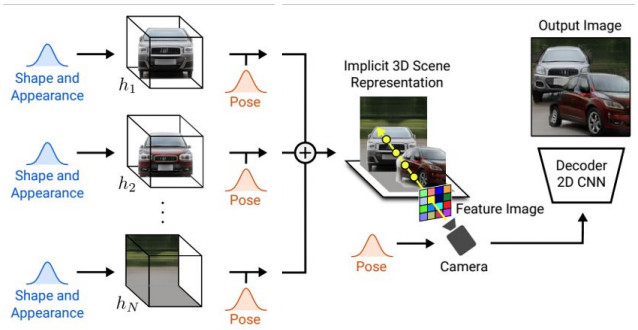

Figure 14: By leveraging object-centric local conditioning, Giraffe [NG21b] learns a generative model that learns to disentangle a scene into separate 3D object representations.

Generative Modeling of 3D Shape IM-Net [CZ19] generates 3D shapes by parameterizing a neural field of occupancy $\Phi : \mathbb{R}^3 \rightarrow \mathbb{R}^1$ and globally conditioning on latents $\mathbf{z}$ via concatenation. Similarly, Occupancy Networks [MON*19] employ a variational autoencoder (VAE) to learn a generative model of 3D occupancy fields. DeepSDF, on the other hand, learns the distribution of SDFs by directly optimizing shape latent codes in the spirit of Generative Latent Optimization [BJLPS17] without encoders.

Generative Modeling of 3D Shape and Appearance Instead of directly modeling the distribution of images, neural fields can parameterize distributions over 3D shape and appearance given only an image dataset. The neural field $\Phi : \mathbb{R}^3 \rightarrow \mathbb{R}^n$ then maps a *3D* coordinate to a quantity that encodes shape and appearance. This is combined with a neural forward rendering model that renders an image given camera parameters.

GRAF [SLNG20] first adapted this via a neural radiance field and volume rendering forward model, globally conditioned via concatenation. Chan et al [CMK*21] improved upon GRAF by using sinusoidal activation functions [SMB*20] in the MLP, and global FiLM conditioning. StyleNeRF [Ano22] combine StyleGAN-style FiLM conditioning with accelerated rendering techniques such as low-resolution rendering and 2D upsampling. By leveraging local conditioning via object-centric compositions (see Section 3), GIRAFFE [NG21b] generates an output image as the composition of multiple neural fields, which allows control over shape, appearance and scene layout (Figure 14). CAMPARI [NG21a] trains an additional camera generator along with the 3D volume generator to generalizes to complex pose distributions.

Multi-modal Generative Modeling An advantage of neural fields is that they are, in principle, agnostic to the signal they parameterize. Du et al. [DCTS21] combine global conditioning, the auto-decoder framework, and latent space regularization to learn multi-modal (such as audio-visual) manifolds.

10. 2D Image Processing

A compelling feature of 2D neural fields is the ability to represent continuous images. The first neural network to parameterize an image was demonstrated by Stanley et al. [Sta07]. These networks were not fit via gradient descent, instead relying on architecture search in a genetic algorithm framework, and could thus not represent images with fine detail. Fitting natural images with neural fields in a modern deep-learning framework was first demonstrated by SIREN [SMB*20] and FFN [TSM*20], and by [SCA19] who decomposed the image into continuous vector layers. Unlike grid-based convolutional architectures, continuous images can be sampled at any resolution. As a result, they have been used for a variety of image processing tasks described below.

Image-to-image Translation These techniques take an input image and map it to another image that preserves some representation of the content. Common tasks include image enhancement, super-resolution, denoising, inpainting, semantic mapping and generative modeling [IZZE17]. Since this task requires learning a prior from data, an encoder-decoder architecture is often used, where the encoder is a convolutional neural network, and the decoder is a locally conditioned 2D neural field (Section 2).

Chen et al. [CLW21a] propose such a locally conditioned neural field with a convolutional encoder for the task of image super-resolution, naturally leveraging the resolution independence of the neural field decoder. Shaham et al. [SGZ*21] leverage a convolutional encoder to produce a low-resolution, 2D feature map from an input image, which is then upsampled with nearest-neighbor interpolation to locally condition a decoder neural field. They demonstrate speedups for a variety of image-to-image translation tasks, such as segmentation and segmentation-to-RGB image as compared to the fully convolutional baseline. Neural Knitworks [CCA*21] discretize the 2D space into patches to introduce the appropriate receptive field, for inpainting, super-resolution, and denoising (Figure 15). CIPS [ADK*21] proposes an image synthesis architecture whose input pixels coordinates are conditionally-independent given latent vector $\mathbf{z}$. Global information is given by $\mathbf{z}$, which is used by a hypernetwork to modulate the neural field weights. INR-GAN [SIE21] similarly uses a hypernetwork and latent code $\mathbf{z}$ to modulate the linear layer weights and biases in the neural field, for image generation. Henzler et al. [HMR20] learns 2D texture exemplars and maps them into 3D by sampling random noise fields at desired positions as the neural texture field input. PiCA [MSS*21] applies a lightweight SIREN to predict human facial texture over a guide mesh, where the 2D coordinate input lie in the UV space. Li et al. [LTW*21] use a 2D neural field to parameterize deformation, for recovering images distorted by turbulent refractive media.

Image Reconstruction X-Fields [BMSR20] parameterizes the image as a 2D neural field conditioned on time and illumination to enable time and illumination interpolation. Alternatively, Nam et al. [NBB21] represents dynamic images (video) with 2D neural fields, with additional homography warp, optical flow, or occlusion operations to model dynamic changes. Kasten et al. [KOWD21] represent dynamic images as time-dependent 2D neural fields, where individual foreground components are segmented as atlas, and alpha composited to obtain the final rendering.

11. Robotics

Robotics requires complex perception systems that allow agents to efficiently infer, reason about, and manipulate representations of

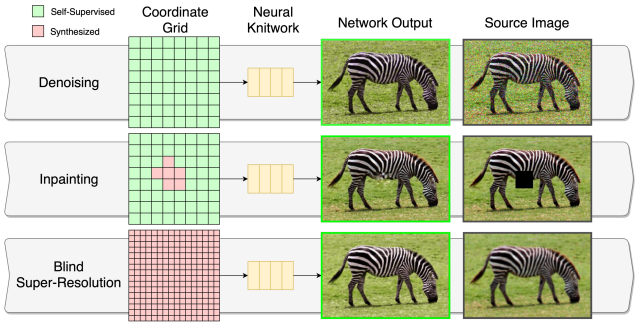

Figure 15: Neural Knitworks [CCA*21] utilizes patch based 2D neural fields for image processing tasks: denoising, blind super-resolution and inpainting.

real-world scenes. Many robotics problems share similarities with visual computing, such as using 3D reconstruction for robot navigation, and so neural fields have recently been used in robotics too. We discuss neural fields for robot perception, planning, and control.

11.1. Localization via Camera Parameter Estimation

Cameras observe the 3D world via 2D images. The projection of a world location onto the image plane is obtained through the extrinsic and intrinsic matrices, where the extrinsic matrix $[\mathbf{R}|\mathbf{t}]$ defines the 6DoF transformation between the world coordinate frame and the camera coordinate frame, while the intrinsics matrix $\mathbf{K}$ describes the projection of a 3D point onto the 2D image plane. Commonly, these camera parameters are obtained via SFM and keypoint matching with off-the-shelf tools like COLMAP [SF16, SZPF16]. Since neural rendering is end-to-end differentiable, camera parameters can be jointly estimated with the neural field making them useful for Simultaneous Localization and Mapping (SLAM) [SLOD21] and Absolute Pose Regression (APR) [CWP21].

There exist numerous ways to parameterize camera intrinsic and extrinsic matrices. Representing the extrinsic matrix, particularly rotation, however, is a long-standing challenge. Since the 3-by-3 rotation matrix lies on $SO(3)$, continuity is not guaranteed [ZBL*19, LEC*20]. Consequently, alternative parameterizations were used in neural field literature: exponential coordinates [YCFB*20], Rodriguez formula [WWX*21], continuous 6D representation [MCL*21, ZBL*19], Euler angle [AMBG*21], and homography warp (for planar scenes) [MCL*21, YCFB*20].

Furthermore, joint reconstruction and registration is a long-standing chicken-and-egg problem: camera parameters are needed to reconstruct the scene, and a reconstruction is needed to estimate camera parameters. One simplification is to assume known reconstruction and solve only the registration problem. iNeRF [YCFB*20] estimates camera poses given pre-trained neural fields, thus *inverting* the problem statement. Due to the strong assumption, iNeRF has a limited use case of registering new, un-posed images to an already-reconstructed scene. Furthermore, the optimization problem is known to be non-convex. A coarse initialization of camera parameters alleviates this challenge. As such, several works *refine* camera parameters during reconstruction [AMBG*21, LMTL21, JAC*21, WWX*21, CWP21]. The chicken-and-egg nature of the problem is reflected by NeRF– [WWX*21], in which the authors jointly optimizes the scene and cameras, but retrain scene reconstruction with the optimized camera parameters for better reconstruction quality.

The more challenging problem of estimating unknown camera parameters, while jointly reconstructing the scene has been looked into by several works [LMTL21, WWX*21]. However, the scene is assumed to be forward-facing and the camera pose initialization relies on hand-crafted rules (*e.g.,* cameras centered at origin, facing the -z axis, with focal lengths equal to the reference image size [WWX*21]). GNeRF [MCL*21] further relaxes these assumptions, and supports inward-facing 360° scenes. The optimization between reconstruction and registration proceeds iteratively, rather than jointly, and are softly coupled via a discriminator. Approaches to overcome local minima in optimization and preserve details include coarse-to-fine training [LMTL21, JAC*21, CWP21], high-frequency positional encoding weights [CWP21, LMTL21], and curriculum learning [JAC*21] for optimizing focal lengths followed by intrinsics and lens distortion.

In the examples above, supervision is provided via appearance. Additional signals such as depth map from RGB-D sensor could provide strong signals for camera parameter optimization [SLOD21]. Similarly, time-of-flight image may also reduce the problem complexity [ALG*21]. A closely related topic is object pose estimation. While object pose can be modelled explicitly, a different approach is to predict a *probability distribution* of pose over $SO(3)$ [MEJ*21]. The probability distribution is itself a neural field, mapping rotation to probability. The approach is especially useful for symmetric objects with multiple valid solutions.

11.2. Planning

Planning in robotics is the problem of identifying a sequence of valid robot configuration states to achieve an objective. This includes path planning for navigation, trajectory planning for grasping or manipulation, or planning for interactive perception [BHS*17]. Given the spatial nature of planning problems, neural fields have been used as a representation in many solutions.

Among the first attempts to use neural networks for robot path planning was by Lemmon [Lem91, Lem92]. While this work refers to a 2D grid of neurons as a "neural field," the neurons are mapped one-to-one to a 2D map and is thus a (non-continuous) coordinate-based network. Their method finds the variational solution of Bellman's dynamic programming equation [Bel54] used in path planning. Other coordinate-based representations have been used for planning, including estimating affordance maps [KS15, QMGR20]. Zhang et al. parameterize a Fisher Information Field [ZS20] via neural network. World constraints can be specified during planning, for example, by specifying constraints as a level set in a high-dimensional space [SFE*20] which is learned as a neural field.

Grasping and manipulation problems require knowing 2D or 3D position of a robot gripper relative to the surface of objects. Some approaches model the shape of the object as a collection of points [BKP11], and learn potentially good points for grasping or manipulation using a Markov Random Field (MRF). This approach can handle points in any continuous 3D position as long as an MRF can be built. Similarly, other data-driven grasping methods

Figure 16: Neural Level of Detail (NGLoD) combines level-of-detail with neural fields as a hybrid continuous-discrete shape representation, demonstrating compression of 3D geometry. Figure adapted from [TLY*21].

often use coordinate-based representations [BMAK14]. ContactNets [PHP20] models contact between objects via learned neural fields. Neural fields can also synthesize human grasps [KYZ*20]. GIGA [JZS*21] uses locally-conditioned neural fields encoding quality, orientation, width, and occupancy for grasp selection.

11.3. Control

Controllers are responsible for realizing plans, while ensuring that physical constraints and mechanical integrity are preserved. Control can be achieved either by relying on a planner or directly from observations. Neural fields have been used for this task by learning an obstacle barrier function approximated by an SDF [LQCA20]. Similarly, Bhardwaj et al. [BSM*21a] solve the robot arm self-collision avoidance task by using neural fields to predict the closest distance between robot links, given its joint configuration. In visuomotor control, control is driven directly by visual observations. Li et al. [LLS*21] use NeRF to facilitate view-invariant visuomotor control to achieve robot goal states specified via a 2D goal image. This is achieved by auto-decoding a dynamics model to support future prediction and novel view synthesis.

12. Lossy Compression

The goal of lossy data compression is to approximate a signal as best as possible with as few bits as possible. These opposing forces naturally form a tradeoff which can be characterized as a Pareto frontier: the rate-distortion curve. In practice, signals are often stored as discrete sequences of data which are transformed into an alternate basis such as the discrete cosine transform which help to decorrelate the signal (making downstream tasks like quantization and entropy coding more effective). Many standards exist, such as JPEG [Wal92] for images or HEVC [SOHW12] for videos. Recent work has explored the potential of neural fields as an alternate signal storage format which directly represents the continuous signal with a parameteric, continuous function. Compression may be achieved in one of two ways. First, by leveraging the inductive bias of the network architecture itself, and simply overfitting a neural field to a signal, i.e., finding parameters Θ of a neural field Φ. Only the parameters and architecture of the neural field need to be stored. Second, *prior-based* compression schemes achieve compression via learning a space of low-dimensional latent code vectors $\mathbf{z}$ that may be decoded into neural field parameters, where the storage cost of the decoder is amortized over many latent codes.

Encoding 3D geometry with neural fields presents an alternative to conventional mesh representations that may enable a significantly reduced memory footprint. Subsequently, SIREN [SMB*20] demonstrated fitting a wide array of signals with neural fields: audio, video, images, and 3D geometry (including large-scale scenes). Lu et al. [LJLB21] show that SIREN in conjunction with scalar weight quantization can compress dense volumetric data better than state-of-the-art approaches at the cost of higher encoding and decoding latency. Davies et al. [DNJ21] compare neural field 3D geometry with decimated meshes and found better reconstruction quality with the same memory footprint. Takikawa et al. [TLY*21] show that a hierarchical tree structure can be used to learn multiresolution signals that can perform level-of-detail more effectively than decimated meshes. Dupont et al. [DGA*21] compared the memory use of neural fields parameterizing images, and found neural fields can outperform JPEG [Wal92] but not state-of-the-art image compression techniques. ACORN [MLL*21] use an adaptive quadtree data structure to fit high resolution images with neural fields, but do not outperform traditional image compression methods. For dynamic 3D scenes, DyNeRF [LSZ*21] uses 28 MB of memory for a 10-second 30 FPS *3D* video sequence. Light Field Networks [SRF*21] allow a large reduction in memory used over a classical discrete light field. General network compression and quantization techniques can further reduce network size [Isi21, BBSC21]. Bird et al. apply entropy penalization to NeRF [BBSC21], and obtain higher compression rates compared to standard HEVC video encoding [SOHW12] and LLFF [MSOC*19] for forward-facing scenes.

The above methods study a variety of continuous signal modalities, and use different datasets, architectures, and metrics, making comparisons difficult. Few rigorous comparisons exist between neural field and conventional compression schemes. Many works lack ablation of design choices and the ability to make direct queries to the without decoding. While neural fields for compression remains in its infancy at the time of writing, it is nonetheless a valuable perspective to consider signals as functions and neural networks as a data format.

13. Beyond Visual Computing

Visual computing problems are a subset of all inverse problems which can be parameterized by neural fields. These problems often share the same challenges involving incomplete observations and the need for a flexible parameterization. In this section we will highlight some of the emerging research directions in neural fields beyond visual computing.

13.1. Alternative Signal Modalities

Most of the works surveyed so far have been concerned with modeling the imaging process of consumer cameras, which measure the visible electromagnetic radiation via optical lenses, using sensors that digitize irradiance into intensity over a 2D raster grid. Nonetheless, neural fields can also model alternative signal modalities such as non-line-of-sight imaging [SWL*21], non-visible x-

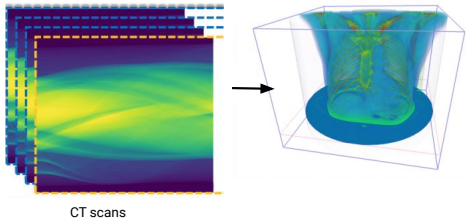

Figure 17: Reconstruction of 3D density field (right) from limited-angle CT measurements (left). Figure adapted from [ZIL*21].

rays for computed tomography (CT) [SPX21, ZIL*21, SLX*21], magnetic resonance imaging (MRI) [SPX21], pressure waves for audio [RBBJ], chemiluminescence [PXZ*21], time-of-flight imaging [ALG*21], as well as volumetric light displays [ZBW*20].

Medical Imaging In CT and MRI, raw sensor measurements are the Radon and Fourier transformation of spatially-varying density, respectively [SPX21]. The sensor domains are not human-readable, while the reconstructed density volume (whose 2D slices are called the image domain) is. Reconstructing the density volume is an ill-posed problem [SPX21], and classical techniques are sensitive to measurement noise [ZIL*21]. Reconstruction and NVS in medical imaging is also limited by capture constraints such as scene movement, finite sensor resolution, limited viewing angle, and sparse views (to reduce X-ray exposure, and speed up procedure).

Neural fields can either parameterize the sensor domain [SLX*21], or the density domain directly [SPX21, ZIL*21]. In the former, the neural field maps sensor coordinates to predicted sensor activations, and can augment real measurement data, before applying classical reconstruction technique (filtered back-projection [KS01]) [SLX*21]. In the latter the neural field directly predicts the density value at a 3D spatial coordinate, and is supervised by mapping its output to the sensor domain via Radon (CT) or Fourier (MRI) transform [SPX21, ZIL*21] (Figure 17).

Similarly, in cryo-electron microscopy (cryo-EM), the 2D sensor measures the convolution between a point spread function and electron density. In CryoGRGN [ZBDB20b], the neural field maps a 3D coordinate to the deconvolved electron density.

Audio Audio signals can be represented as raw waveforms or spectrograms, and each can be parameterized by neural fields [SMB*20, GCM*21]. Raw waveform is a continuous 1D field function that maps temporal coordinates to amplitude, often stored as a collection of discrete samples. A spectrogram is the Fourier transform of a waveform, which is a 2D field function mapping time and frequency to amplitude. Neural field methods fit all time-dependent waveforms, which means that the techniques are applicable to all waves mechanical (e.g., seismic waves, ocean waves, acoustic waves) or electromagnetic (e.g., wave optics, radio waves).

Synthetic aperture sonar (SAS) also exhibits the common problems in inverse problems: noisy sensor domain and ill-posed problem. Reed et al. [RBBJ] use a neural field mapping a 2D position to a distribution of point scatter (reconstruction domain), and obtain supervision by mapping to the sensor domain via convolution.

13.2. Physics-informed Problems

Physics-informed problems have solutions that are restricted to a set of partial differential equations (PDEs) based on laws of physics. The solutions are often continuous in spatio-temporal coordinates. Neural fields are therefore a natural parameterization of the solution space, given that neural networks are continuous, differentiable, and universal function approximators. These neural fields are also referred to as physics-informed neural networks (PINNs), whose use was first popularized by [RPK19a]. Problems constrained by nonlinear PDEs often require arbitrarily-small step sizes for traditional methods, which often require prohibitive computation resources. Parameterizing the solution via neural networks re-formulates these problems as *optimization*, rather than *simulation*, which is more data-efficient [RPK19a]. Since many physical processes in nature are governed by PDEs (boundary conditions), supervising (or regularizing) the neural network via its gradients is a common technique. In fact, the Eikonal regularization for SDF is an example in visual computing. Physics-informed problems include topology optimization [ZLCT21], geodesy estimation [IG21], collision dynamics estimation [PHP20], solving the Schrodinger Equation [RPK19a], Navier-Stokes equation [RPK19a], and Eikonal equation [SAR20].

Discussion & Conclusion

14. Discussion

As we have seen, neural fields have a rich but evolving technical 'toolbox', and a rapidly increasing space of applications both within and outside of visual computing. In our view, there are several factors that have resulted in this progress.

First, the idea of parameterizing a continuous *field* using an MLP without the need to use more complex neural network architectures has simplified the training of fields and reduced the entry barrier [GN98]. Neural fields provide a fundamentally different approach to signal processing that is no longer discrete and more faithful to the original *continuous* signal. Second, techniques such as positional encoding and sinusoidal activations have significantly improved the quality of neural fields leading to large leaps in applications focused on quality. Finally, applications in novel view synthesis and 3D reconstruction have been particularly important in popularizing neural fields because of the visually appealing nature of these applications [MST*20, SZW19, GFK*18b, PFS*19]. Third, researchers have realized that differentiable volume and voxel rendering commonly used in novel view synthesis methods [MSOC*19, YFKT*21] can be useful in solving others tasks like 3D reconstruction [YGKL21] and even semantic segmentation [ZLLD21, VRG*21].

Future Directions Despite the progress, we believe that neural fields have only started to scratch the surface and there remains great potential for continued progress in both techniques and applications. In terms of **techniques**, a common limitation of many neural fields is their inability to generalize well to unseen data. We believe that integration of stronger priors can enable these methods to generalize better and be data efficient. Other inductive biases in the form of task-specific heuristics, laws of physics, or network

architecture can further help generalization. Building a **common framework** for incorporating these priors is a fruitful direction for future work. Furthermore, the rapid progress has been at the expense of methodical evaluation and analysis of common techniques. We need shared **datasets** and **benchmarks** on which different techniques can be evaluated and compared.

In terms of **applications**, a majority of neural field methods have thus far been used to solve "low-level" (*e.g.,* image synthesis) and "mid-level" (*e.g.,* 3D reconstruction) tasks. The application of neural fields for "high-level" **semantic tasks** remains an open problem. Examples of these problems include understanding 3D scene layout [WYN21], 3D scene interaction [LMW*21a], and grouping of data into more meaningful entities [TLV21]. Furthermore, neural fields have focused on a single data modality, but exploring the fusion of **multiple modalities** could be a fruitful topic of research. For example, synthesizing fields based on language input [JMB*21], or joint modeling of image and text or audio input could foster closer connections with the NLP community. Finally, future work should consider moving beyond supervised learning of fields and consider **weakly- or self-supervised learning** as an alternative. Building upon advances in 3D deep learning such as transformation equivariance [SPJ*22, SSM*20, STD*21] could allow neural fields to be data efficient and generalize better.

Finally, the neural fields community must become more self-aware to build a **culture** that promotes scholarship, sustainable growth, inclusivity, and diversity. We must pay careful attention to past work and avoid repetition of work through "*selective amnesia*" [SC21]. We must avoid becoming a "*scientific bandwagon*" [Sha56] by acknowledging limitations of neural fields and by collaborating with domain experts to identify shortcomings. While the explosion of work in neural fields helps us make important advances, we must be aware of its impact on the mental health and well-being of researchers [SC21].

Societal Impact As the applications of neural fields increase, so will their societal impact, especially in domains where they make a significant difference. In generative modeling of audio, imagery, 3D scenes, etc., neural fields have enabled the generation of realistic content for the purpose of deception, for instance, impersonating the image and voice of actors without their consent. Recent work explores the detection of digitally-altered imagery [TVRF*20]. Similarly, improving the capabilities of inverse-problem solvers, such as denoising and blur-removal algorithms, has implications for privacy. For instance, this may enable de-anonymizing recordings that were previously assumed to not contain sufficient information to allow identification of participating actors. It may also decrease the cost of surveillance, making it more widely available as the hardware requirements may decrease. Finally, neural fields could also have negative environmental impact as significant computational resources are spent optimizing neural networks with GPUs.

Neural fields can also positively impact society. Democratizing the generation of photo-realistic imagery helps more artists and content creators to tell stories. The ability to aggregate information from low-dimensional supervision (*e.g.,* 2D images) relaxes hardware constraints for 3D content creation. Neural fields for computer vision may help build robotic automation that helps people.

15. Conclusion

This report provides an overview of the flourishing research direction of neural fields. From a review of over 250 papers, we have summarized five classes of techniques using a shared mathematical formulation, including prior learning and conditioning, hybrid representations, forward maps, network architectures, and manipulation methods. Then, we have surveyed applications across graphics, vision, robotics, medical imaging, and computational physics. Neural fields research will continue to grow. To aid in continued understanding, we have established a living report as a community-driven website, where authors can submit their own papers and classify them using the taxonomy developed in this report. Neural fields will be a key enabler for progress across many areas of computer science, and we look forward to the research yet to come.

Authors

Yiheng Xie (https://yxie20.github.io/) is a final-year undergraduate student at Brown University and a researcher at Unity Technologies. His research interests include 3D reconstruction, physically-based rendering, and robotics. He is a member of Brown Visual Computing (BVC), advised by Professor Srinath Sridhar, and Brown Humanity Centered Robotics Initiative (HCRI), advised by Professor Michael Littman.

Towaki Takikawa (https://tovacinni.github.io/) is a Ph.D. student at the University of Toronto with Prof. Sanja Fidler and Prof. Alec Jacobson. He is also a Research Scientist at NVIDIA in the Hyperscale Graphics Systems group. He received his bachelors in Computer Science at the University of Waterloo.

Shunsuke Saito (http://www-scf.usc.edu/~saitos/) is a Research Scientist at Meta Reality Labs in Pittsburgh. He finished his Ph.D. at University of Southern California, where he worked with Prof. Hao Li. Prior to USC, he spent one year at University of Pennsylvania as Visiting Researcher. He obtained B.Eng. and M.Eng. in Applied Physics at Waseda University in 2013 and 2014.

Or Litany (https://orlitany.github.io/) is a Research Scientist at NVIDIA. Prior, he was a postdoc at Stanford University working under Prof. Leonidas Guibas, and a postdoc at FAIR working under Prof. Jitendra Malik. Previously, he was a postdoc at the Technion and a research intern at Microsoft, Intel and Google. He received his PhD from Tel-Aviv University, advised by Prof. Alex Bronstein. He received my B.Sc. in Physics and Mathematics from the Hebrew University under the auspices of "Talpiot."

Shiqin Yan (https://player-eric.com/about) is a masters student in Computer Science at Brown University. He is passionate about solving real-world problems with large-scale web-based systems.

Numair Khan (http://cs.brown.edu/~nkhan6/) is a PhD student at Brown University where his research focuses on methods and representations for scene reconstruction, differentiable rendering, and novel-view synthesis.

Federico Tombari (https://federicotombari.github.io/) is a Research Scientist and Manager at Google Zurich (Switzerland), where he leads an applied research team in Computer Vision and