UNIVERSITÀ
DI PARMA

Università degli Studi di Parma

Dottorato di Ricerca in Tecnologie dell'Informazione

Ciclo XXXVI

Geometry and Learning for Efficient 3D Perception

Coordinatore

Chiar.mo Prof. Marco Locatelli

Tutore

Chiar.mo Prof. Massimo Bertozzi

Dott. Paolo Zani

Dott. Paolo Medici

Dottorando

Marco Orsingher

Anni Accademici

2020/2021 - 2022/2023

A Giulia

Abstract

Building a 3D representation of the world is a longstanding challenge in computer vision and machine learning, with applications in virtual and augmented reality, autonomous driving, industrial site scanning, cultural heritage preservation, and more. The main goal of this thesis is to develop efficient algorithms for processing 3D data, by combining classical geometry-based methods with modern deep learning approaches. Efficiency is a crucial aspect of 3D perception, since data are typically acquired by low-cost noisy sensors and must be processed on mobile platforms with limited computational budget. Furthermore, the exponential growth of 3D data sources calls for scalable and efficient processing pipelines.

Our first contribution is a novel framework for multi-view 3D reconstruction in urban scenarios. We significantly improve a state-of-the-art classical approach for dense reconstruction, by designing a local-to-global optimization strategy that leads to geometrically consistent surfaces. Moreover, we show how to scale it up to arbitrarily large scenes with a divide and conquer procedure that combines view clustering and view selection, thus allowing for a massive parallelization of the 3D reconstruction process.

Secondly, we present two algorithmic advances in efficient training of neural representation for novel view synthesis. We propose to speed up the learning process by focusing on informative rays, which are defined in the 2D image space by high-entropy pixels and in the 3D object space by a sparse set of cameras that ensures scene coverage, while keeping optimal relative baseline. Additionally, we leverage multi-view geometry as pseudo-ground truth to guide the neural implicit field towards high-fidelity 3D models.

We also tackle the point cloud upsampling task, with the aim of refining noisy and low-resolution data from cheap range sensors into dense and uniform point clouds. To this end, we formulate the first learning-based approach that allows 3D upsampling with arbitrary scaling factors, including non-integer values, with a single trained model. The main idea is to convert the input to a probabilistic representation and to train a Transformer network to map between samples from such domain and points on the underlying object surface. This flexibility is crucial in real-world applications with computational and bandwidth constraints.

Finally, we propose two novel methods for neural network compression. We first show that feature-based knowledge distillation can be improved by complementing the direct feature matching baseline with a teacher features-driven regularization loss, thus enabling the student model to learn more robust latent representations. Then, we introduce a neural compression approach that combines network pruning with self-distillation and significantly improves the sparsity-accuracy tradeoff for several perception tasks. This allows to deploy neural architectures on constrained hardware for fast inference with unprecedented performances.

Acknowledgments

This thesis has been developed in collaboration with Ambarella Inc. I would like to start by expressing my deepest gratitude to Alberto Broggi for answering the cold email of an unknown software engineer in July 2020 and for allowing me to join the company. Three years later, we had the chance to know each other better while drinking a (fake!) bubble tea in Taipei, and I am more than happy to keep working with him after this experience.

Moreover, I was extremely lucky to have Paolo Zani and Paolo Medici as technical supervisors. You taught me everything I know about 3D computer vision and you shaped my mind as a researcher, while letting me free to explore my own scientific interests. I cannot thank you enough for your constant and valuable feedback.

This PhD was made joyful by my two partners in crime, Francesco and Anthony. Thank you for your support and your friendship. I would also like to extend this gratitude to all the other colleagues at Ambarella, especially to Alessandro, Gabriele, Giulio, Pierpa, Lucky, Mu-Ti, George, Emily, Momo, Julian, Tihao, and to the amazing people I met in Parma and Hsinchu.

My heartfelt thanks also go to Massimo Bertozzi for supervising the university side of my doctoral studies and for helping me to navigate safely in academia. Thank you for reviewing my papers, suggesting conferences and choosing amazing restaurants in Lecce.

Beside the scientific achievements, I was fortunate to have the chance of filling my PhD with several life-changing experiences. To everyone I met in Lecce, Aachen, Sicily, Prague, Taiwan and Gran Canaria: you are part of this thesis as well.

Ci tengo a ringraziare particolarmente la mia famiglia, i miei genitori Daniele e Laura, mia sorella Elena e mio fratello Matteo. Grazie anche a Raffaele e Laura, per tutti i momenti che abbiamo passato e che passeremo insieme: siete parte della mia famiglia. Infine, il ringraziamento più importante va a Giulia, per starmi a fianco ogni giorno, per ascoltarmi, amarmi e completarmi. Il vostro supporto emotivo e la fiducia che avete riposto in me sono stati fondamentali per affrontare questo percorso.

Contents

1 Introduction

1.1 Motivation

Understanding the physical and semantic properties of the 3D world is essential for the large-scale deployment of embodied artificial intelligence. The increasing relevance of 3D perception stems from the exponential growth of 3D data sources, including smartphones, wearable devices, mobile robots, scanning sensors and self-driving cars. The goal of 3D perception algorithms is to analyze and process these data with both classical geometry-based methods and modern deep learning approaches, with applications in autonomous driving, industrial site scanning, healthcare, virtual and augmented reality, cultural heritage preservation, and more.

Crucially, 3D perception pipelines are typically deployed on mobile and wearable platforms with limited memory and computational budget. Furthermore, input data from low-cost sensors might be noisy or incomplete. For these reasons, we especially focus on the development of *efficient* 3D perception algorithms from cheap acquisition devices, such as cameras and sparse LiDARs or radars. We tackle efficiency in terms of scalability and massive parallelization for geometry-based methods, we design learning-based approaches that can be trained fast and deployed efficiently on target hardware, and we propose how to optimally combine both worlds.

In the following sections, we first give some technical background in computer vision and machine learning, which serves as the theoretical foundations for the rest of the manuscript. Then, we define the goal of our research and outline the original contributions.

Figure 1.1: Visualization of a depth map computed by a neural network [1]. Darker colors mean farther objects.

1.2 Background

In this section, we start by providing an overview of 3D perception in terms of data representation, processing paradigms and open challenges. Then, we describe multi-view geometry principles and formalize the relationship between the 3D world and its 2D projections. Finally, we introduce the deep learning architectures that will be used throughout the rest of this thesis.

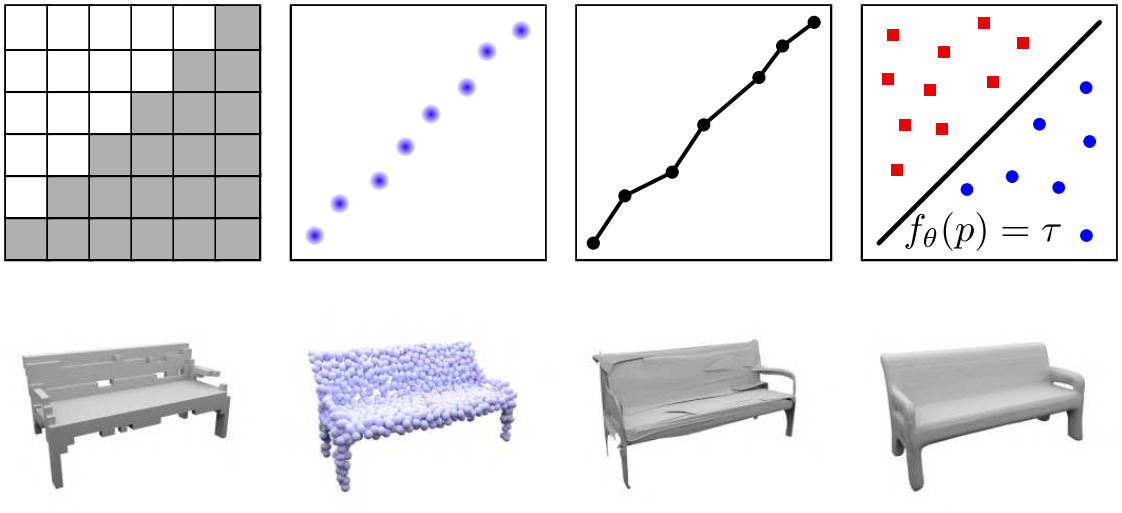
1.2.1 3D Perception

3D Data Representation

Images are conveniently represented as 2D grids of pixels. This representation is universally accepted as both efficient to process and expressive, since it explicitly stores the visual content. On the other hand, 3D data might be stored in different formats, according to their acquisition system and their role in the processing pipeline:

RGB-D data represents the 3D world with a view-dependent depth map, which associates a depth value to each pixel in the image plane as an additional color channel (D stands for *depth*). Depth maps can be obtained either from the sensor directly, from (possibly multi-view) stereo algorithms [2, 3] or as the output of a neural network [4, 1]. An example is visualized in Figure 1.1.

Voxels are the naive extension of pixels to 3D, as shown in Figure 1.2a. The world is discretized into a fixed grid of size $W \times H \times D$ and a voxel properties can be accessed by querying its indices (i, j, k). Despite recent algorithmic efforts [5], the main disadvantage of voxels is their cubic memory footprint, even in empty areas.



(a) Voxel grid. **(b)** Point cloud. **(c)** Mesh. **(d)** Implicit function.

Figure 1.2: Comparison of different ways for representing 3D data [9].

Point Clouds model the world as an *unordered* list of points, as shown in Figure 1.2b. Each point typically stores its position, but it can be extended to include the unit normal vector to the surface, the color and the semantic class. Point clouds are expressive and can either come from range sensors, such as LiDAR or radar, or computed by back-projecting multiple depth maps in 3D. However, they lack geometric connectivity between points, which might be needed for practical applications.

Meshes extend point clouds by connecting points with polygons. Figure 1.2c shows that a mesh approximates the actual surface of the object shape, whereas a point cloud can be seen as a set of discrete samples from such surface. For real-world data, this representation is typically obtained with surface reconstruction algorithms [6, 7] from an input point cloud, and it is the most common data format for computer graphics [8].

Implicit Functions represent a 3D shape as the set of points $\mathbf{X} \in \mathbb{R}^3$ that satisfy $f(\mathbf{X}) = 0$ (see Figure 1.2d). In simple cases such as common geometric shapes, the function f has a well-known closed formula, while a recent trend is to approximate it with a neural network for more complex objects and scenes [9, 10, 11]. Implicit representations have arbitrary resolution, since they can be queried at continuous spatial locations. Moreover, the marching cubes algorithm [12] allows seamless conversion to polygonal meshes.

3D Perception Tasks

A 3D perception task is broadly defined as any computational pipeline that operates on 3D data, in order to infer physical and semantic properties of the world. These tasks can be classified according to their inputs and desired outputs.

The wide availability of low-cost cameras in smartphones, autonomous vehicles and surveillance systems has motivated a longstanding effort in image-based 3D perception, with the goal of understanding the 3D world from its 2D observations. The link between 2D and 3D is typically performed by depth estimation algorithms, which assign a depth value to each pixel in the image, such that the corresponding 3D points can be computed via back-projection. In the classical *multi-view 3D reconstruction* pipeline presented in Chapter 2, depth maps are computed from calibrated cameras with multi-view stereo [3, 13, 14]. However, neural networks can also be trained to directly map colors to depth [4, 1] and binocular stereo geometry can be exploited to generate disparity values natively from the sensor, if available [2].

Furthermore, much research has been devoted to point cloud processing. Since we mainly focus on 3D data from cheap sensors, we assume that the input point cloud is sparse or noisy and we tackle *point cloud upsampling* in Chapter 4. Once the input has been converted to a dense and uniform representation, the resulting point cloud can be further analyzed by classification [15, 16], segmentation [15, 16] or detection [17] algorithms.

A more recent trend is to learn an implicit 3D representation of the scene directly from images [11, 18, 19], without depth maps or point clouds as an intermediate step. The goal is to obtain differentiable and compact models, that can be easily rendered for *novel view synthesis*. We will show in Chapter 3 that this approach has unique advantages over the classical graphics pipeline, which focuses on explicit triangular meshes.

Finally, in some cases, 3D data might be converted back to 2D, in order to exploit standard computer vision algorithms. For example, a common solution in LiDAR-based perception for autonomous driving is to convert the raw point cloud into a bird's-eye-view of the scene [20], or to a range image [21]. Similarly, explicit semantic reasoning in 3D can be avoided by rendering multiple views of the object and analyzing each view independently [22]. This approach is beyond the scope of this thesis, as we focus on 3D perception from low-cost acquisition devices, which do not generate sufficiently dense data to be rendered. For the same reason, we refer the reader to [8] for a detailed overview of classical graphics algorithms, that directly operates on triangular meshes and are not considered in this thesis.

Challenges and Open Problems

Perceiving the world in 3D from sensors' observations has several challenges, which may vary according to the processing paradigm. In this thesis, we will focus on the following challenges and propose our original contributions to solve them, as presented in Section 1.3:

- Classical approaches based on multi-view geometry have solid mathematical foundations, but they typically assume a static scene with constant illumination in all the views. Moreover, these methods struggle with complex materials and textureless areas.
- Deep learning in 3D is promising, as it can potentially exploit geometric priors on large-scale data and leverage the expressive power of neural networks. On the other hand, ground truth data in 3D are complex to obtain, costly to store and their representation highly influences the design of the network.
- Even when learning-based methods are successful at the given task, the final trained model might be too heavy and slow for inference on edge devices. A possible solution is to compress the neural network before deployment with quantization, pruning or distillation algorithms. However, the best sparsity-accuracy tradeoff might be hard to find.
- The ideal approach would be to combine the best of both worlds and to inject multi-view geometry constraints into learning-based frameworks. In this case, the main challenge is the design of geometry-aware loss functions and architectures, which is an open issue.
- Modern approaches that represent 3D scenes with implicit functions have several advantages over classical graphics, but they are slow to both render and train, and they are not forced to infer multi-view consistent representations.

1.2.2 Multi-View Geometry

Several 3D perception tasks, such as 3D reconstruction or monocular depth estimation in Chapter 2, assume to have a set of 2D images as input. Multi-view geometry [23] provides a set of formal tools to compute the relationship between 2D pixels on the image plane and their corresponding 3D point $\mathbf{X} \in \mathbb{R}^3$ in the real world. In this thesis, we assume an ideal pinhole model for all the cameras, where all the light rays pass through a single point $\mathbf{C} \in \mathbb{R}^3$, as shown in Figure 1.3. We now present some basic mathematical relationships based on such camera model, and we refer the interested reader to [23] for more details.

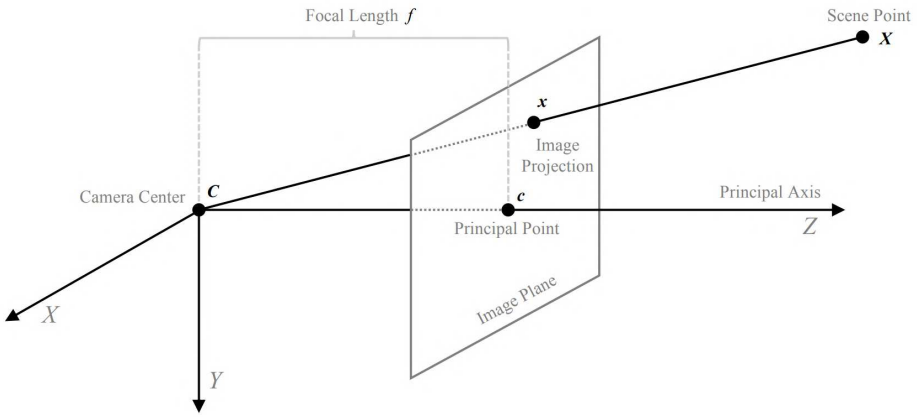


Figure 1.3: Single-view geometry of an ideal pinhole camera [24].

Single-View Geometry

Let $\mathbf{R} \in SO(3)$ and $\mathbf{t} \in \mathbb{R}^3$ be the rotation matrix and the translation vector, respectively, that describe the transformation between the world and the camera frames. Using homogeneous coordinates, the projection of a 3D point to a 2D pixel can be formulated as:

$$\mathbf{x} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \mathbf{PX} \tag{1.1}$$

Such projection can be understood as follows. The 3D point $\mathbf{X}$ is first transformed from the world frame to the camera frame with *extrinsic parameters* $(\mathbf{R}, \mathbf{t})$. Then, this point is observed by the camera with *intrinsic parameters* $\mathbf{K}$:

$$\mathbf{K} = \begin{bmatrix} f & s & c_u \\ 0 & af & c_v \\ 0 & 0 & 1 \end{bmatrix} \tag{1.2}$$

where $(c_u, c_v) \in \mathbb{R}^2$ is the location of the principal point, f is the focal length with anisotropy a and s is a shearing factor due to possibly non-rectangular pixels ($s \simeq 0$ in modern cameras). Both extrinsic and intrinsic parameters can be recovered with *camera calibration* algorithms [23, 25] and the 3×4 matrix $\mathbf{P}$ is known as *projection matrix*.

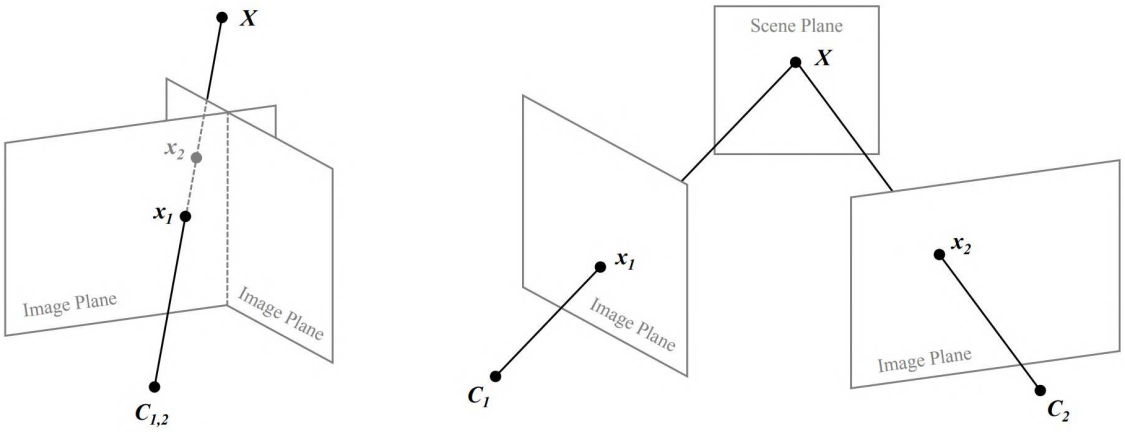


Figure 1.4: Two-view geometry with homography, that maps points between image planes of a purely rotating camera (left) or two cameras observing a planar scene (right) [24].

Two-View Geometry

In order to recover the 3D structure of the scene from its 2D projections, we need to invert the image formation process. However, depth information is lost in Equation 1.1, as any point along the same optical ray would project to the same pixel. For this reason, the only way to recover the position of a 3D point from images is by combining multiple observations of such point on at least two different cameras. Assuming generic cameras with unknown extrinsic and intrinsic parameters, there are two ways of describing the geometric relation between two images, depending on the camera motion: *homography* and *epipolar geometry* [23].

Homography The homography maps points between two different planes. Therefore, it applies when the camera is purely rotating or is observing a planar scene. In the first case, the projection center and the viewing rays are shared by the two views, and the homography simply describes the mapping between the two image planes. In the second case, we must concatenate an homography from the first image plane to the planar scene with another homography from the scene plane to the second image. Both cases are depicted in Figure 1.4. Mathematically, the observations $\mathbf{x}_1$ and $\mathbf{x}_2$ in homogeneous coordinates are linked by the homography matrix $\mathbf{H}$. Without loss of generality, if $\mathbf{R}_1 = \mathbf{I}$ and $\mathbf{t}_1 = 0$, such matrix can be decomposed as:

$$\mathbf{H} = \mathbf{K}_2 \left(\mathbf{R}_2 - \frac{\mathbf{t}_2 \mathbf{n}^\top}{d} \right) \mathbf{K}_1^{-1} \tag{1.3}$$

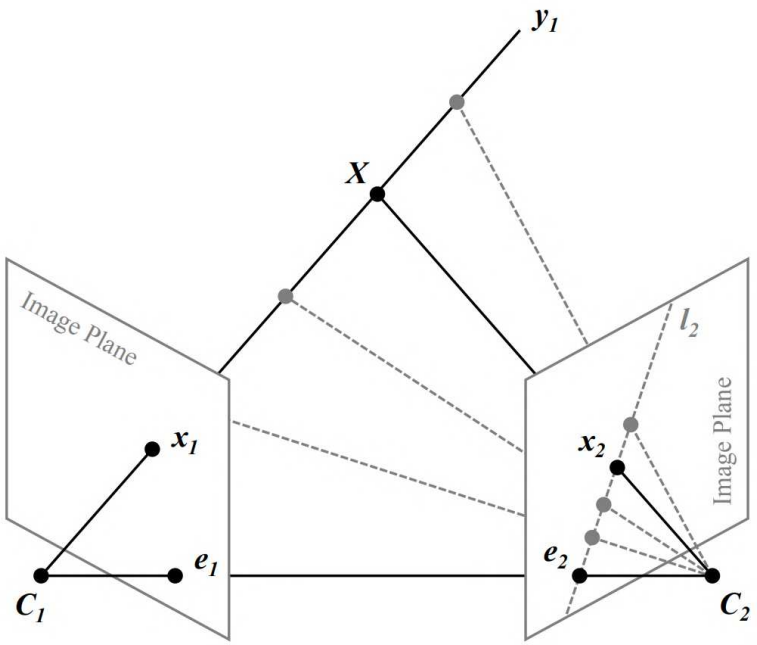


Figure 1.5: Epipolar geometry describing the relationship between two cameras with arbitrary relative motion [24].

Here, the parameters $(\mathbf{n}, d)$ are the unit normal vector and the orthogonal distance of the scene plane, respectively. In the common case of a single moving camera with known intrinsics, Equation 1.3 can be further simplified as $\mathbf{K}_1 = \mathbf{K}_2$. Since $\mathbf{H}$ has 8 degrees of freedom, due to the projective ambiguity, it can be estimated with *direct linear transform* (DLT) from a minimum of 4 image correspondences [23].

Epipolar Geometry When two images depict a generic scene or the camera undergoes a generic motion, the homography does not apply and we need to introduce epipolar geometry. Let $\mathbf{e}_1 = \mathbf{P}_1\mathbf{C}_2$ and $\mathbf{e}_2 = \mathbf{P}_2\mathbf{C}_1$ be the *epipoles*, which are the projections of the principal point of the other camera in the current image. For a given point $\mathbf{X}$ observed as $\mathbf{x}_1$ and $\mathbf{x}_2$ on the two image planes, we can define the *epipolar lines* $\mathbf{l}_1 = \mathbf{e}_1 \times \mathbf{x}_1$ and $\mathbf{l}_2 = \mathbf{e}_2 \times \mathbf{x}_2$. Moreover, the plane defined by $\mathbf{X}$ and the two projection centers $(\mathbf{C}_1, \mathbf{C}_2)$ is called the *epipolar plane* Π. This abstraction is visualized in Figure 1.5. Note that $\mathbf{l}_2 = \mathbf{P}_2\Pi$ is also the intersection of the epipolar plane with the second image plane and the observation $\mathbf{x}_2$ must lie on such line. This relationship is known as the *epipolar constraint*, which is formalized as:

$$\mathbf{x}_2^\top \mathbf{l}_2 = 0 \implies \mathbf{x}_2^\top \mathbf{P}_2 \Pi = \mathbf{x}_2^\top \mathbf{P}_2 \mathbf{P}_1^+ \mathbf{l}_1 = \mathbf{x}_2^\top \mathbf{P}_2 \mathbf{P}_1^+ [\mathbf{e}_1]_\times \mathbf{x}_1 = 0 \tag{1.4}$$

where $\mathbf{P}_1^+$ is the pseudo-inverse of $\mathbf{P}_1$ and $[\mathbf{e}_1]_\times$ is a skew-symmetric matrix representing the cross product. Such constraint can be expressed in terms of the relative pose between the two cameras. Let $\mathbf{R}_{12} = \mathbf{R}_2\mathbf{R}_1^\top$ and $\mathbf{t}_{12} = \mathbf{t}_2 - \mathbf{R}_{12}\mathbf{t}_1$:

$$\mathbf{x}_2^\top \mathbf{P}_2\mathbf{P}_1^+[\mathbf{e}_1]_\times\mathbf{x}_1 = \mathbf{x}_2^\top \mathbf{K}_2^{-\top}[\mathbf{t}_{12}]_\times\mathbf{R}_{12}\mathbf{K}_1^{-1}\mathbf{x}_1 = \mathbf{x}_2^\top \mathbf{F}\mathbf{x}_1 = 0 \tag{1.5}$$

The matrix $\mathbf{F}$ is called the *fundamental matrix* and it maps points from one image to lines in the other image. When the intrinsic parameters of the cameras are known, the epipolar constraint can be simplified and enforced by the *essential matrix* $\mathbf{E}$:

$$\mathbf{F} = \mathbf{K}_2^{-\top}[\mathbf{t}_{12}]_\times\mathbf{R}_{12}\mathbf{K}_1^{-1} = \mathbf{K}_2^{-\top}\mathbf{E}\mathbf{K}_1^{-1} \iff \mathbf{E} = \mathbf{K}_2^\top\mathbf{F}\mathbf{K}_1 = [\mathbf{t}_{12}]_\times\mathbf{R}_{12} \tag{1.6}$$

Both the fundamental and the essential matrix can be estimated from a non-minimal set of 8 correspondences by rearranging Equation 1.4 [23, 26, 27].

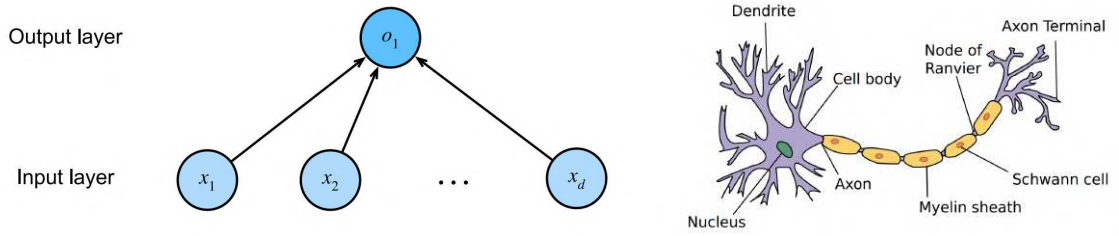
1.2.3 Deep Learning

Machine Learning (ML) is a subfield of artificial intelligence that focuses on the development of algorithms and models capable of learning and making predictions from raw data, without being explicitly programmed for the task of interest. Given a *dataset* of input-output pairs $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$ and a parametric *model* $\hat{\mathbf{y}}_i = f_\theta(\mathbf{x}_i)$, such model is trained to minimize a *loss function* $\mathcal{L}$ that measures the quality of current predictions:

$$\theta^* = \arg\min_\theta \sum_{(\mathbf{x}_i,\mathbf{y}_i)\in\mathcal{D}} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i) \tag{1.7}$$

The parametric model is typically a *neural network* and its optimal parameters θ^* are computed with stochastic gradient descent [28]. In recent years, the massive amount of available data and compute allowed to train increasingly bigger models on huge datasets. For this reason, modern ML is also called *deep learning*, due to the depth of current neural networks, which enables them to learn complex and hierarchical representations from data. This capability is essential for the success of deep learning across multiple domains, such as image understanding [29], natural language processing [30], speech recognition [31], and 3D reconstruction [4, 11].

In the remainder of this section, we provide an overview of network architectures used in thesis to achieve state-of-the-art results in 3D perception tasks. We refer the interested reader to [32, 33] for a broader coverage of deep learning systems.



(a) Connectivity pattern of linear regression. **(b)** A biological neuron.

Figure 1.6: Linear regression can be seen as a single-layer neural network [32].

Multi-Layer Perceptrons

The simplest way to model an input-output relationship is to assume a linear dependence, such that the parameters θ are just the weights $\mathbf{w}$ and bias $\mathbf{b}$ of a linear regression model:

$$\hat{\mathbf{y}} = \mathbf{w}^\top \mathbf{x} + \mathbf{b} \tag{1.8}$$

Linear regression can be seen as a single-layer neural network, in which every input is directly connected to every output, as shown in Figure 1.6a. The term *neural network* has its historical roots in the analogy of this representation with the biological neuron visualized in Figure 1.6b. The information from other neurons is received in the dendrites and weighted by synaptic weights. Then, such information is aggregated in the nucleus as a weighted sum and sent via the axon to the axon terminals, where it is fed into another neuron.

In order to model complex nonlinear relationships, neural networks need nonlinear components in their computational graphs. The affine transformation in Equation 1.8 is typically followed by an *activation function* σ that operates elementwise on its arguments:

$$\hat{\mathbf{y}} = \sigma(\mathbf{w}^\top \mathbf{x} + \mathbf{b}) \tag{1.9}$$

Finally, we can stack multiple layers of computations, each implementing Equation 1.9 on its inputs, thus producing even more expressive models. The resulting model is called *Multi-Layer Perceptron* (MLP) and visualized in Figure 1.7, while the intermediate layers between input and output are referred to as *hidden* layers. Despite their simple structure, it can be shown that MLPs with a single hidden layer can approximate any function, given enough nodes [34].

In this thesis, MLPs are used to model neural radiance fields in Chapter 3, as well as fundamental building blocks of convolutional networks in Chapter 5 and Transformers in Chapter 4.

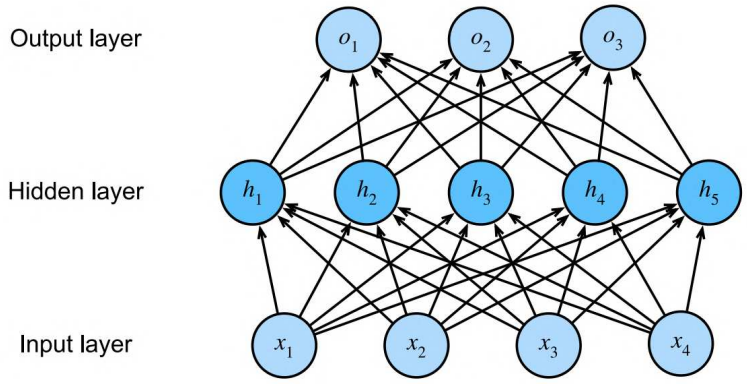


Figure 1.7: A multi-layer perceptron with a single hidden layer [32].

Convolutional Neural Networks

When the input data are defined on a spatial grid, such as 2D images or 3D voxels, the fully connected layers in MLPs have two key issues: (i) they are inefficient in terms of both time and memory, and (ii) they ignore the inherent local structure of regular grids. In order to process such data efficiently and effectively, we need a *translation-invariant* operator that focuses on *local* regions of the image (or the voxel grid). Inspired by classical image processing, such operator is a convolution. *Convolutional layers* compute the elementwise product of the input with a learnable kernel and add a scalar bias to produce the output.

Since these layers only capture the local structure of the input by design, information about the global context of the image is lost in their computation. In order to balance local and global information, *pooling layers* are interleaved with convolutional layers along the network. Note that these layers are used to aggregate local information progressively, but they do not involve any learnable parameter, as they compute local statistics of their inputs.

Given these fundamental building blocks, we can define a *Convolutional Neural Network* (CNN) as a network architecture which is composed by stacking multiple blocks of convolutional layers, elementwise nonlinearity and a pooling layer. These blocks are essentially hierarchical feature extractors [35] for a final MLP classifier, as shown in Figure 1.8 for a basic CNN implementation [36]. Modern CNNs also include residual connections in order to propagate further the inputs during the forward pass and the gradients during the backward pass [37].

In this thesis, we use CNNs as the main source of comparison for monocular depth estimation in Chapter 2, as well as the architecture of choice for developing efficient compression algorithms in Chapter 5 for image classification, object detection and semantic segmentation.

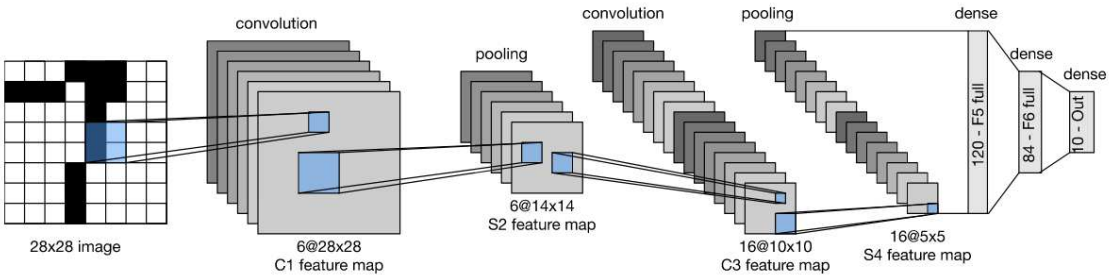


Figure 1.8: The pioneering LeNet-5 architecture for handwritten zip code recognition [36].

Transformers

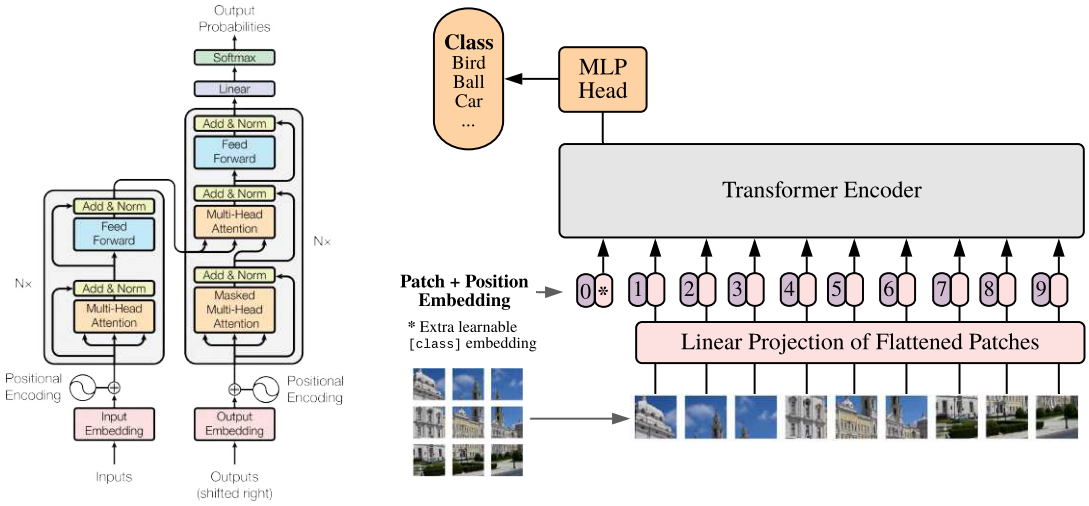
In recent years, deep learning has been revolutionized by the introduction of the Transformer architecture [30]. While this model was originally introduced for natural language processing applications, several other fields are progressively converging towards this network architecture, including computer vision [38] and 3D perception [39, 40]. The core idea behind the Transformer is the *attention* mechanism. Given a database $\mathcal{D} = \{\mathbf{k}_i, \mathbf{v}_i\}_{i=1}^m$ of *keys* and *values*, the attention value of a *query* $\mathbf{q}$ over $\mathcal{D}$ is defined as:

$$\text{Attention}(\mathbf{q}, \mathcal{D}) = \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i \tag{1.10}$$

where $\alpha(\mathbf{q}, \mathbf{k}_i) \in \mathbb{R}$ are scalar attention weights, which are typically computed as a dot product and normalized via softmax [30]. The name derives from the fact that this operation generates a linear combination of the values and pays more attention to the terms where weights are large.

In practice, the same set of queries, keys and values is transformed with h independently learned linear projections, in order to capture different patterns in the input data. This design is called *multi-head attention*, as the outputs of different heads are concatenated for further processing. The key intuition behind the Transformer model is to remove convolutional layers and to rely only on multi-head attention and MLPs. The full architecture with positional encoding, layer normalization [41] and residual connections [37] is shown in Figure 1.9a.

This architecture was designed to process text sequences, where the database $\mathcal{D}$ is composed by subwords, also known as *tokens*. In order to process visual data, the Vision Transformer in Figure 1.9b proposed to convert an image into a sequence of patches, which can be processed as tokens in text. Similarly, we will use an attention-based architecture in Chapter 4 for efficient point cloud upsampling, by enabling each point to attend to its nearest neighbors.



(a) Natural language processing [30].

(b) Vision Transformer for image inputs [38].

Figure 1.9: Deep learning research is converging towards the Transformer architecture for multiple modalities [30, 38].

1.3 Contributions

The main goal of this thesis is to develop *efficient* algorithms for processing 3D data from cheap sensors, by combining classical geometry-based methods with machine learning. To this end, we present the following contributions to existing literature:

- We propose a novel pipeline for *multi-view 3D reconstruction* in urban scenarios [14], by improving the core components of a state-of-the-art geometrical approach [13, 42]. Furthermore, we enable our method to scale up to arbitrarily large scenes with a divide and conquer procedure that combines view clustering and view selection [43]. This allows to massively parallelize the 3D reconstruction process, with practical applications in autonomous driving and urban simulation.

- We present two algorithmic advances in efficient training of neural representation for *novel view synthesis* [18]. Firstly, a sparse set of cameras is computed to ensure scene coverage and optimal relative baseline. Secondly, pixels in the camera plane are sampled based on their local entropy, in order to focus on relevant informative rays. Moreover, we design a framework that generates pseudo-ground truth in 3D with multi-view geometry [19], thus allowing to extract high-fidelity 3D models from the implicit representation.

- We formulate the first learning-based approach for *point cloud upsampling* with arbitrary scaling factors, including non-integer values, with a single trained model [44]. The key intuition is to map the low-resolution input to a probability distribution on a canonical domain. High-resolution samples from such distribution are then mapped back to the

surface by a Transformer decoder. This flexibility is crucial in real-world applications with computational and bandwidth constraints.

- We propose two novel methods for *neural network compression* [45]. Firstly, we show that feature-based Knowledge Distillation (KD) can be improved by complementing the direct feature matching baseline with a teacher features-driven regularization loss, thus enabling the student model to learn more robust latent representations. Secondly, we introduce a neural compression approach that combines network pruning with KD and significantly improves the sparsity-accuracy tradeoff for several perception tasks.

1.4 Thesis Structure

Chapter 1 In this chapter, we have introduced the motivation of our research, and we have provided the technical background in both computer vision and machine learning, which is needed to understand the remainder of the manuscript. The rest of the thesis is organized per task and presents our original contributions.

Chapter 2 After an introduction on a classical image-based 3D reconstruction pipeline from multiple views, we describe a novel approach for monocular reconstruction in urban scenarios [14] and an algorithm to scale it up to entire cities [43].

Chapter 3 We start by presenting the neural radiance fields framework for novel view synthesis, especially focusing on the training loop and the geometry extraction phase. Then, we show how to speed up training by selecting informative rays [18] and how to compute clean surface meshes with explicit multi-view constraints [19].

Chapter 4 This chapter provides a detailed literature review on point cloud upsampling, as well as neural point processing with Transformers. Then, we propose a learning-based approach for arbitrary upsampling [44] and validate it with extensive experiments against state-of-the-art, highlighting its robustness and generalization capabilities.

Chapter 5 Firstly, we introduce the topic of neural network compression, by covering the related work in architectural pruning and knowledge distillation. Secondly, we describe a novel regularization loss for feature-based KD and a neural compression method to combine pruning with KD [45], in order to deploy efficient and accurate models.

Chapter 6 This chapter concludes the thesis and suggests future work directions.

1.5 Publications

The original contributions and the content in this thesis are based on the material published in the following peer-reviewed papers:

- **Orsingher M.**, Zani P., Medici P., and Bertozzi M., *Revisiting PatchMatch Multi-View Stereo for Urban 3D Reconstruction*, IEEE Intelligent Vehicles Symposium (IV), 2022.
- **Orsingher M.**, Zani P., Medici P., and Bertozzi M., *Efficient View Clustering and Selection for City-Scale 3D Reconstruction*, International Conference on Image Analysis and Processing (ICIAP), 2022.
- **Orsingher M.**, Dell'Eva A., Zani P., Medici P., and Bertozzi M., *Informative Rays Selection for Few-Shot Neural Radiance Fields*, International Conference on Computer Vision Theory and Applications (VISAPP), 2024.
- **Orsingher M.**, Zani P., Medici P., and Bertozzi M., *Learning Neural Radiance Fields from Multi-View Geometry*, Learning to Generate 3D Shapes and Scenes Workshop at European Conference on Computer Vision (ECCV), 2022.
- Dell'Eva A.*, **Orsingher M.***, and Bertozzi M., *Arbitrary Point Cloud Upsampling with Spherical Mixture of Gaussians*, International Conference on 3D Vision (3DV), 2022. * Equal contribution.
- Dell'Eva A.*, **Orsingher M.***, Lee Y. M., and Bertozzi M., *Teacher Features-Driven Regularization for Knowledge Distillation*, IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR), 2024 (currently under review). * Equal contribution.

2

Multi-View 3D Reconstruction

2.1 Introduction

This chapter focuses on the multi-view 3D reconstruction problem, where the goal is to compute a dense 3D model of the scene from a set of images. The 2D observations can be captured by multiple cameras at different time instants, such as tourists' smartphones taking pictures of famous landmarks [46, 47, 48] or by a single camera moving in the environment, e.g. mounted on a self-driving car [49, 4, 50]. Specifically, we target scalable 3D reconstruction for autonomous driving, with the aim of building a detailed representation of the vehicle's surroundings at scale, from arbitrary camera configurations. This task is typically solved with a geometry-based pipeline [51, 3], which is briefly described in Section 2.2 as the underlying theoretical framework for our original contributions [52, 53]. While end-to-end 3D reconstruction frameworks [54, 55, 56] and learning-based components have been proposed at various stages of the pipeline [57, 58, 59], classical methods still dominate the leaderboards [60, 61] in complex outdoor scenarios. To this end, we propose a novel dense reconstruction algorithm in Section 2.3 and present a way to efficiently scale it up to entire cities in Section 2.4.

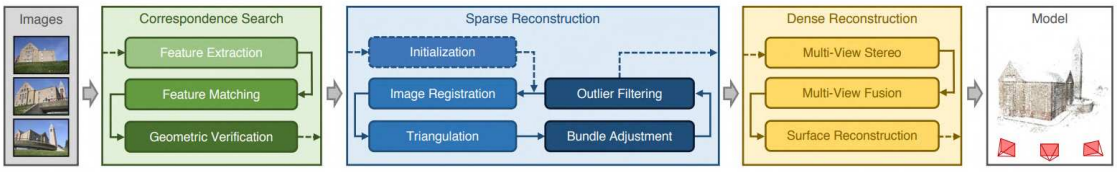


Figure 2.1: Overview of the classical image-based 3D reconstruction pipeline with correpondence search, sparse modeling with structure from motion and dense reconstruction with multi-view stereo [24].

2.2 3D Reconstruction Pipeline

This section provides a general overview on the classical image-based 3D reconstruction pipeline shown in Figure 2.1 and serves as foundation to understand the contributions presented in the remainder of the chapter. Specifically, we review the three main components of a system that models the 3D world from images: (i) searching for corresponding features between multiple views (Section 2.2.1), (ii) computing camera poses with sparse keypoints (Section 2.2.2) and (iii) reconstructing the dense 3D surface (Section 2.2.3).

2.2.1 Correspondence Search

In Chapter 1, we showed that the two-view geometry between any pair of cameras can be estimated from a set of corresponding pixels. We now present how to establish such correspondences among multiple views. This process can be typically decomposed in three steps. Firstly, distinctive features are *extracted* and *described* independently for each image. Then, a *matching* algorithm is required to associate similar images that observe the same points. Finally, geometric verification allows to build a *scene graph* with images as nodes and pairwise two-view geometry as edges. These steps are visualized in Figure 2.2.

Feature Extraction

An image *feature* is a locally distinct region of pixels that should be distinctive and invariant under both geometric and radiometric changes. The goal is to describe the whole image with a sparse set of features that can be recognized repeatably in different views of the same scene. Such features can be either computed by classical algorithms [62, 63, 64] or learned by neural networks [57, 65, 66]. We refer the reader to [67] for an in-depth review of existing methods and a comparison between the two approaches. In both cases, each feature is identified by its 2D pixel coordinates and described by a N-dimensional vector $\mathbf{d} \in \mathbb{R}^N$ that encodes information

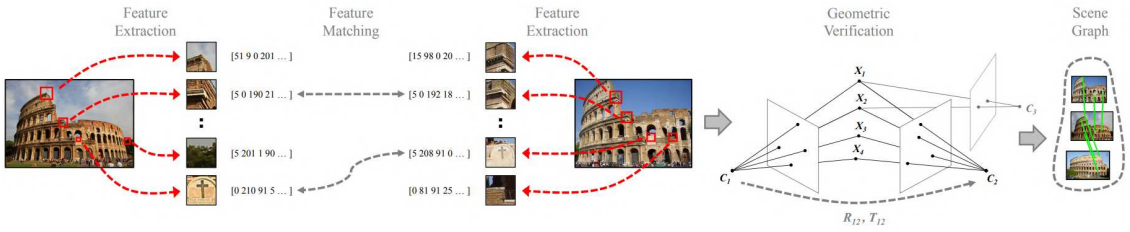


Figure 2.2: During the correspondence search phase, features are extracted for each image and matched against all the other views. Then, geometric verification produces the scene graph as intermediate representation [24].

about local geometry and appearance. Given a collection of images $\mathcal{I}$, the first step of the 3D reconstruction pipeline is to associate each image I_i with the corresponding set of features $\mathcal{F}_i$.

Feature Matching

Once each image is described by its features $\mathcal{F}_i$, the next step is to find which pairs of images have scene overlap. This process is called *feature matching*, and it aims at finding image pairs with a sufficient number of local features with similar descriptors. In principle, the naive approach would be to compare each local feature of each image against all the features of all the other images. The inherently quadratic complexity of this method is prohibitively expensive in real-world scenarios, and efficient methods exist to tackle this problem in a scalable way [68, 46, 47]. The output of this step is a set $\mathcal{C}$ of potentially overlapping image pairs (I_a, I_b) and their correspondences $\mathcal{M}_{ab} \subset \mathcal{F}_a \times \mathcal{F}_b$.

Geometric Verification

The feature matching phase might produce wrong associations, as it does not guarantee that corresponding features actually map to the same scene point. Therefore, an additional verification step is required to find a valid geometric transformation between image pairs. Such transformation can be either a homography or a fundamental/essential matrix, depending on the spatial configuration of the two cameras [23]. We consider a mapping to be geometrically valid if a sufficient number of features can be matched with a robust estimation technique, such as RANSAC [69]. The final output of the correspondence search module is a verified set $\bar{\mathcal{C}} = \{I_a, I_b, \bar{\mathcal{M}}_{ab}, \mathbf{G}_{ab}\}$ of image pairs with inlier correspondences and their geometric relation $\mathbf{G}_{ab} \in \{\mathbf{H}, \mathbf{F}, \mathbf{E}\}$, which can be chosen by minimizing the GRIC score [70]. Moreover, pairwise 3D points $\mathcal{X}_{ab}$ can be computed by triangulating the inlier correspondences. This geo-

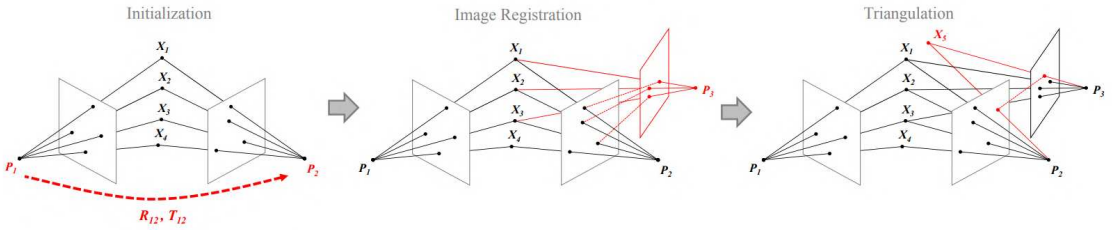


Figure 2.3: The different stages of incremental sparse reconstruction [24].

metrically verified set can be visualized as a graph with images as nodes and edges with two-view geometry to indicate scene overlap.

2.2.2 Sparse Reconstruction

The scene graph $\bar{\mathcal{C}}$ contains pairwise relative poses between image pairs with scene overlap, as well as 3D points corresponding to matching features. The goal of the sparse reconstruction phase is to compute *globally consistent* camera poses and sparse keypoints in a common reference frame. This problem is called *Structure-from-Motion* (SfM), as it deals with estimating 3D points (*structure*) from images at different viewpoints (*motion*). Formally, the output of the sparse reconstruction phase is a set of 3D points $\mathcal{X}$ and the calibration matrices $\mathcal{P} = \{(\mathbf{K}_c, \mathbf{R}_c, \mathbf{t}_c)\}$ for each image in $I_c \in \mathcal{I}$.

Incremental Reconstruction

The SfM problem can be solved with three fundamentally different approaches:

1. *Hierarchical* methods apply a divide and conquer scheme by reconstructing multiple subgraphs in parallel and merging them afterward [71].
2. *Global* methods formulate the problem as a joint optimization over all the relative two-view geometries [72, 73, 74, 75]. This approach is typically efficient, but not robust.
3. *Incremental* methods allow to repeatedly enhance and refine the solution, thus ensuring robustness against outliers and wrong estimates from previous steps [76, 77, 68, 46].

We now focus on the steps of incremental reconstruction, which is the default choice for large-scale scenarios. The process is visualized Figure 2.3: it starts from a sufficiently good pair of cameras and iteratively alternates between triangulating features and registering new views.

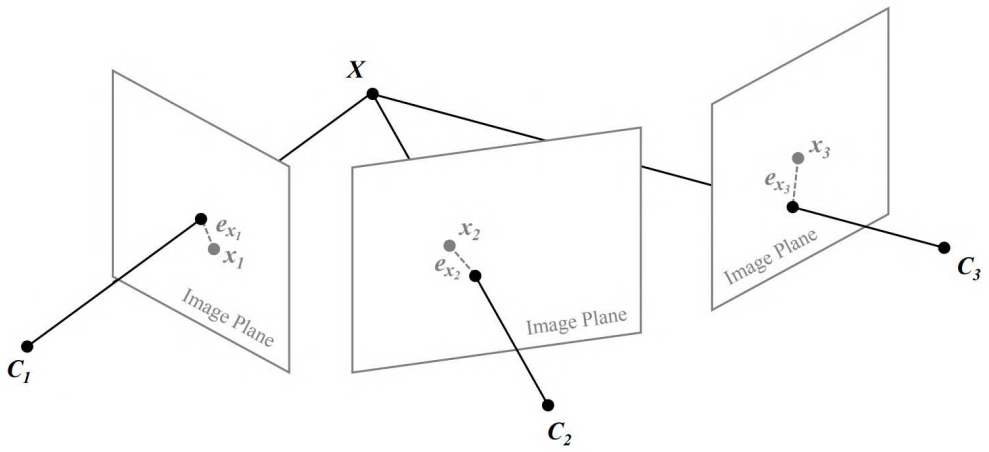


Figure 2.4: Bundle adjustment is the problem of jointly refining the position of 3D points and the camera poses to minimize the reprojection errors for the optical rays corresponding to image features [24].

Initialization Iterative optimization procedures strongly rely on a good initial solution. Incremental SfM algorithms are typically initialized with a carefully selected image pair having a sufficiently wide baseline and belonging to a densely connected region of the scene graph [78]. This results in a metric reconstruction of the two images, as well as a set of 3D points obtained by triangulating inlier correspondences.

Registration At each iteration, a new image is selected and registered against the current metric reconstruction. This requires to establish 2D-3D correpondences between features in the new image and the 3D points belonging to the currently estimated scene structure. Given such correspondences, the *Perspective-n-Point* (PnP) problem can be solved with RANSAC [79, 69] to find the corresponding camera calibration $(\mathbf{K}_c, \mathbf{R}_c, \mathbf{t}_c)$ of the selected view, in a *common* reference frame.

Triangulation The camera added to the reconstruction in the registration step might observe scene points that are not yet reconstructed. If such points are seen by at least another image, then they can be triangulated to extend the 3D model $\mathcal{X}$.

Bundle Adjustment

In the incremental reconstruction process, registration and triangulation are performed alternately, as they rely on each other's outputs. This means that uncertainty in both steps propa-

gate across iterations and the solution might quickly drift to a non-recoverable State. For this reason, a joint non-linear refinement of both cameras and points is required.

Such refinement is known as *bundle adjustment* (BA), since the goal is to *adjust* a *bundle* of optical rays by minimizing the reprojection errors of 3D points on image planes:

$$\mathcal{P}^*, \mathcal{X}^* = \arg\min_{\mathcal{P},\mathcal{X}} \sum_{\mathbf{P}_i \in \mathcal{P}} \sum_{\mathbf{X}_j \in \mathcal{X}} ||\mathbf{x}_{ij} - \mathbf{P}_i \mathbf{X}_\mathbf{j}||^2 = \arg\min_{\mathcal{P},\mathcal{X}} \sum_{\mathbf{P}_i \in \mathcal{P}} \sum_{\mathbf{X}_j \in \mathcal{X}} ||\mathbf{e}_{ij}||^2 \tag{2.1}$$

where $\mathbf{e}_{ij}$ is the error in pixel space between the observation on camera $\mathbf{P}_i$ of point $\mathbf{X}_j$ (known) and its projection based on the current estimate of the 3D scene (unknown). The shape of this cost function is highly non-linear and it cannot be optimized with standard global optimization methods. The typical approach is to iteratively approximate the cost function and to minimize it with gradient descent or the Levenberg-Marquardt algorithm [80, 81].

To sum up, incremental SfM starts from two carefully selected images and iteratively registers new cameras, while adding new points to the scene structure via triangulation. After each iteration, bundle adjustment is solved to refine the current solution. The procedure ends when all the cameras have been registered and the output is a sparse 3D reconstruction of the scene, along with calibration and poses, in a globally consistent reference frame.

Visual SLAM

The incremental reconstruction approach described in the previous section assumes that an unstructured collection of images is available offline. However, in practical applications, such as robotics and autonomous driving, images are acquired sequentially by a moving camera attached to the robot or the vehicle. This setting is usually referred to as visual *Simultaneous Localization and Mapping* (SLAM) [82], where *localization* is the process of recovering camera poses and *mapping* means to reconstruct sparse 3D points, which must be performed *simultaneously*. There are two main differences with respect to the classical SfM problem presented before. Firstly, the sequential nature of data greatly simplifies the correspondence search phase, since the search for matching features can be reduced to nearby frames. Secondly, points and poses must be estimated in *real-time* and continuously updated to allow a safe navigation of the agent. In this section, we briefly review the general architecture of a SLAM system as shown in Figure 2.5 and refer the interested reader to [83] for more details.

Visual Odometry The core of any SLAM algorithm is the *visual odometry* module, which estimates the robot poses sequentially by examining the changes that the motion induces on

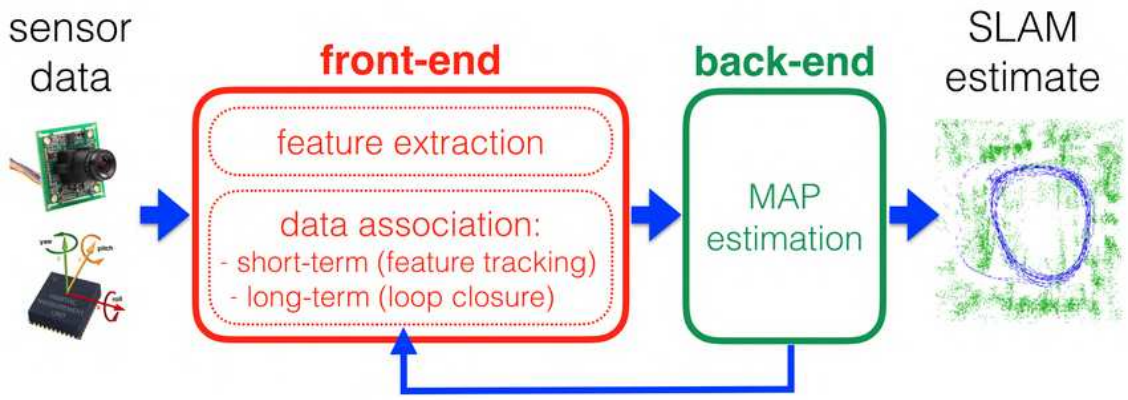


Figure 2.5: General architecture of a modern SLAM system: a front-end performs visual odometry with loop closure detection at high frequency, while the back-end solve non-linear map optimization at low frequency [83].

the images acquired by its camera [84]. This is performed by the front-end at high frequency. Without loss of generality, let $\mathbf{R}_0 = \mathbf{I}$ and $\mathbf{t}_0 = 0$ at iteration $k = 0$. Then, for $k > 0$, as soon as a new image I_k is available:

1. Features are extracted and matched between I_{k-1} and I_k.
2. The relative motion $(\mathbf{R}_{k,k-1}, \mathbf{t}_{k,k-1})$ between the two frames is estimated by computing the essential matrix based on feature correspondences.
3. The current pose $(\mathbf{R}_k, \mathbf{t}_k)$ is computed recursively by concatenating all the relative transformations.

At each iteration, 3D points can be triangulated from feature correspondences to simultaneously build a representation of the environment.

Loop Closure Detection Visual odometry is consistent only locally, since it accumulates drift over time due to its incremental nature. The general solution to restore global consistency is to detect when the robot visits the same location multiple times, such that an additional pose constraint can be added to the optimization problem. Loop closures are computed by the front-end, typically using the *bag-of-words* approach [85].

SLAM Backend The last key component of modern SLAM systems is a back-end module that runs at lower frequency and computes a non-linear optimization of both 3D points and

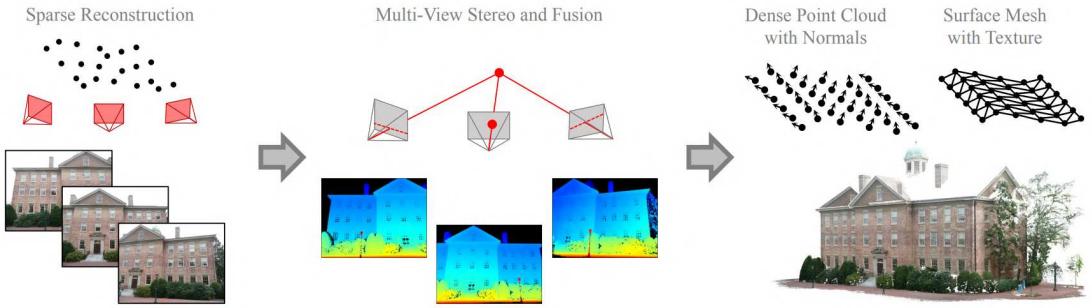


Figure 2.6: The dense reconstruction phase aims at computing a complete 3D model of the scene, starting from posed images and sparse keypoints [24].

camera poses, in order to produce a globally consistent solution. This refinement can be formulated either as a full bundle adjustment or as a motion-only BA, which is also known in literature as *pose graph optimization* [86], when mapping is not required.

2.2.3 Dense Reconstruction

At this stage of the 3D reconstruction pipeline, the scene structure is composed by sparse keypoints corresponding to triangulated image features, while each camera has been calibrated and located in a global reference frame. The goal of the dense reconstruction phase is to convert this approximate representation into a complete 3D model, such as a dense point cloud or a textured surface mesh. Figure 2.6 shows that the problem can be solved in three steps. Firstly, a *multi-view stereo* (MVS) algorithm computes a depth value and a normal vector for each pixel of each image. Then, multi-view consistent estimates are back-projected in 3D to generate a dense point cloud with normals, which can be finally meshed into a textured surface.

Multi-View Stereo

The goal of the multi-view stereo stage is to assign a valid depth value d and a normal vector $\mathbf{n}$ to each pixel of each image. While many paradigms for MVS exist, we focus here on the PatchMatch approach [87, 13, 42, 3] shown in Algorithm 2.1, which produces state-of-the-art results on most benchmarks [60, 61] and serves as baseline for the original contributions presented in Section 2.3. We refer the reader to [88] for a detailed review of MVS algorithms. Since the same steps are repeated independently for each pixel of each image, we now consider a generic pixel $\mathbf{p}$ of a reference image $I_{ref} \in \mathcal{I}$ and let $\mathcal{I}_{src} = \mathcal{I} \setminus \{I_{ref}\}$.

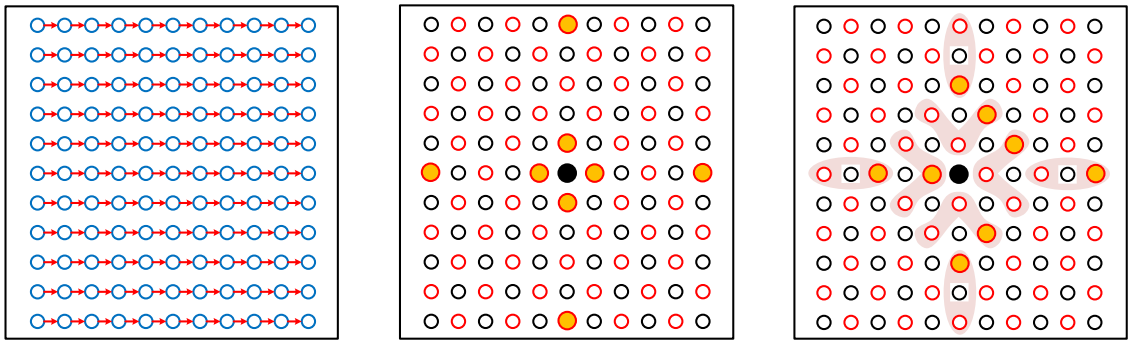
Algorithm 2.1 PatchMatch Multi-View Stereo

Input → Images $\mathcal{I} = \{I_i\}$ with calibration parameters $\mathcal{P} = \{(\mathbf{K}_i, \mathbf{R}_i, \mathbf{t}_i)\}$ from SfM
for all images $I_{ref} \in \mathcal{I}$ **do**
  $\mathcal{I}_{src} = \mathcal{I} \setminus \{I_{ref}\}$
  for all pixels $\mathbf{p} \in I_{ref}$ **do**
    $(d, \mathbf{n}) =$ `Initialization()`
    for N_{iter} iterations **do**
      $\mathcal{S} =$ `Propagation()`
      $(d, \mathbf{n}) =$ `Evaluation`$(\mathcal{I}, \mathcal{P}, \mathcal{S})$
      $\bar{\mathcal{S}} =$ `Perturbation()`
      $(d, \mathbf{n}) =$ `Evaluation`$(\mathcal{I}, \mathcal{P}, \bar{\mathcal{S}})$
    end for
  end for
end for
Output → Pixelwise depths $\mathcal{D} = \{\mathbf{D}_i\}$ and normals $\mathcal{N} = \{\mathbf{N}_i\}$

INITIALIZATION PatchMatch is an iterative optimization method, which needs an initial solution to progressively refine. The depth value is initialized randomly within a given range $d \in [d_{min}, d_{max}]$, while a random normal vector $\mathbf{n}$ is generated by following [89]. Then, the following three steps are performed for a given number of iterations.

PROPAGATION In this phase, a set of K new hypothesis $\mathcal{S} = \{(d_k, \mathbf{n}_k)\}_{k=1}^{K}$ is sampled from neighboring pixels. Several propagation schemes have been proposed and they are visualized in Figure 2.7. Sequential propagation [3] traverses pixels in parallel scanlines, while checkerboard approaches [89, 87] allow to simultaneously update half of the pixels by dividing them in a red-black diffusion scheme. In general, the former tends to be more robust in challenging cases, while the latter are much more efficient and parallelizable. In modern GPU implementations [89, 87], each red (or black) pixel can be assigned to a CUDA core and processed independently at each iteration.

EVALUATION Each hypothesis in $\mathcal{S}$ represents the local tangent plane to the scene surface at the 3D point corresponding to the current pixel $\mathbf{p}$. This allows to compute the corresponding pixels on all the images in $\mathcal{I}_{src}$ with the plane-induced homography in Equation 1.3, as shown in Figure 2.8. At this point, we need a way to compute a cost for each hypothesis and to update the current one. The cost c_{ik} of projecting the pixel $\mathbf{p}$ on the image I_i according to the hypothesis k is usually given by a robust statistics of a local patch around $\mathbf{p}$, such as *normalized cross-correlation*



(a) Sequential propagation along parallel scanlines [3].

(b) Symmetric checkerboard propagation [89].

(c) Adaptive checkerboard propagation [87].

Figure 2.7: Overview of different propagation schemes for PatchMatch MVS. In (b) and (c), the black point represents the current pixel, while orange points are the pixels where the new hypotheses are sampled.

(NCC) or *sum-of-squared-differences* (SSD) [3]. Then, such cost is aggregated across multiple views in $\mathcal{I}_{src}$ to produce a single cost per hypothesis as follows:

$$c_k = \frac{\sum_{I_i \in \mathcal{I}_{src}} w(I_i) c_{ik}}{\sum_{I_i \in \mathcal{I}_{src}} w(I_i)} = \mathcal{L}_{photo}(d_k, \mathbf{n}_k) \tag{2.2}$$

where $w(I_i)$ is a weighting function to account for occlusions and visibility. The current depth and normal solution for $\mathbf{p}$ is updated with the solution minimizing such cost:

$$(d, \mathbf{n}) = \underset{(d_k, \mathbf{n}_k) \in \mathcal{S}}{\arg\min}\, \mathcal{L}_{photo}(d_k, \mathbf{n}_k) \tag{2.3}$$

We will see in Section 2.3 that the purely photometric loss function in Equation 2.3 can be extended to include terms about multi-view geometric consistency [13] and planar priors [42].

PERTURBATION After propagation and evaluation, a refinement step is needed to escape bad local minima and enrich the diversity of solution space. Let $(d_c, \mathbf{n}_c)$ be the current hypothesis, $(d_p, \mathbf{n}_p)$ a randomly perturbed hypothesis and $(d_r, \mathbf{n}_r)$ a randomly generated depth and normal. Then, a small set of new candidate solutions is defined as:

$$\bar{\mathcal{S}} = \{(d_p, \mathbf{n}_c), (d_r, \mathbf{n}_c), (d_c, \mathbf{n}_p), (d_c, \mathbf{n}_r), (d_r, \mathbf{n}_r), (d_p, \mathbf{n}_p)\} \tag{2.4}$$

Each new hypothesis is tested against the current one by following Equation 2.3 and the final estimate is assigned to the pixel $\mathbf{p}$.

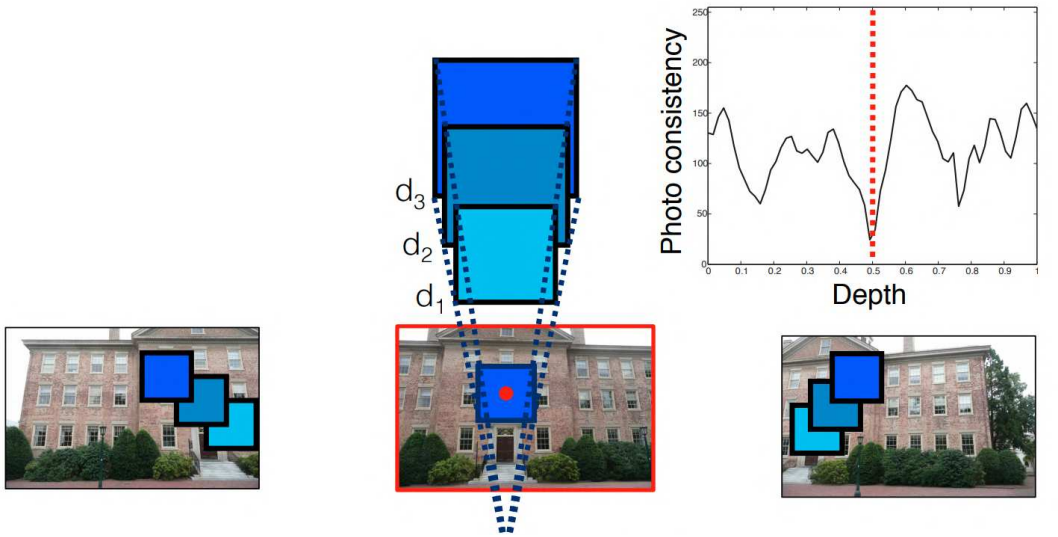


Figure 2.8: Depth hypotheses are evaluated based on their photometric consistency when the pixel is projected onto source views with plane-induced homography [24].

Multi-View Fusion

Given pixelwise depths and normals $(\mathcal{D}, \mathcal{N})$ from multi-view stereo, the goal of this second step is to lift these estimates from the image planes to a dense, consistent point cloud in scene space. Let $(d, \mathbf{n})$ be the MVS output for a pixel $\mathbf{p}$ in a camera with known intrinsics $\mathbf{K}$ and extrinsics $(\mathbf{R}, \mathbf{t})$. This pixel can be back-projected in 3D by inverting Equation 1.1 with known depth d:

$$\mathbf{X} = \mathbf{R}^\top \left(d\mathbf{K}^{-1} \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} - \mathbf{t} \right) \tag{2.5}$$

Then, such point is observed by all the other images, and it is marked as valid if the following three conditions hold for a minimum number of views: the reprojection error, the relative depth difference and the angle between normals must all be lower than a given threshold. This can be repeated for each pixel of each image, and the result is a list of multi-view consistent points with 3D position, color and surface normal [87, 3]. Note that the same procedure can be used to generate consistent 3D semantics if a segmentation of each image is available [90].

Surface Reconstruction

A dense point cloud is a rich and complete description of the 3D scene structure. However, it lacks connectivity between points, and some applications in computer graphics or video games

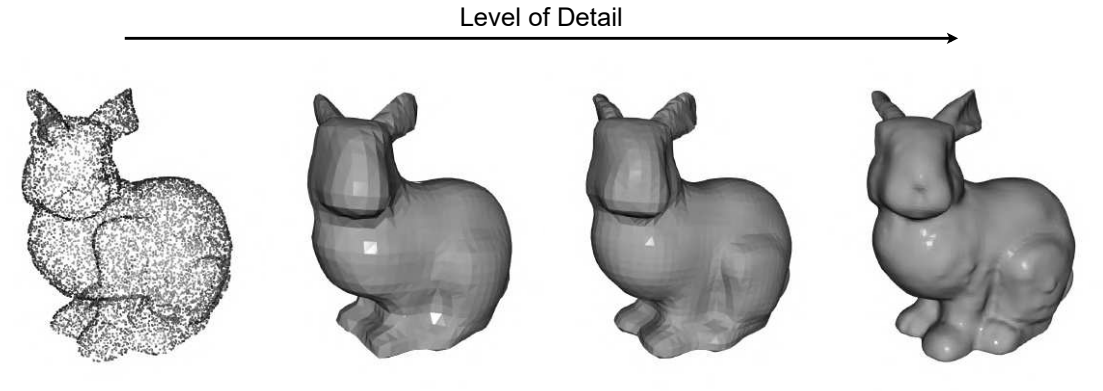


Figure 2.9: Triangular meshes extracted with marching cubes at different levels of detail from the input point cloud (left), using the implicit surface reconstruction method NKSR [91].

typically require a more compact and continuous representation, such as a textured mesh. This can be achieved with surface reconstruction algorithms, that approximate the geometry of a scene with locally planar surfaces, such as triangles.

Classical approaches, such as [6, 7], try to find an implicit function whose value is 0 at the given points and whose gradient equals the local normal vector. The key idea is to solve for the indicator function of the shape $\mathrm{I}_\Omega(\mathbf{X}) = 1$ if $\mathbf{X} \in \Omega$, and 0 otherwise. The solution is computed from Poisson equation that links the normal field at the shape boundary and the gradient of this indicator function. The zero-level set of the resulting implicit function defines the surface of the shape, and a mesh is obtained by extracting the corresponding isosurface.

On the other hand, learning-based methods [9, 10, 91, 92, 93] model an implicit occupancy field by learning to map a query point $\mathbf{X}$ either to its binary occupancy or its signed distance to the shape Ω. This mapping is built by fitting a simple MLP to the input point cloud. Given a neural implicit field $f_\theta : \mathbb{R}^3 \to \{0, 1\}$, the marching cubes algorithm [12] can be used to extract a triangular mesh. The algorithm works by iterating over a uniform grid of cubes superimposed with a spatial region and testing each vertex of each cube. For a given cube, if the vertices are all positive or all negative, then it is either completely inside or outside the surface, respectively. Otherwise, the cube intersects the surface and a triangle is generated. The smaller the cubes, the better the resulting mesh will approximate the actual shape and capture intricate details, as shown in Figure 2.9 for the ears of the bunny.

2.3 Revisiting PatchMatch MVS for Urban Scenes

This section is based on the original contributions published in [52]:
Orsingher M., Zani P., Medici P., and Bertozzi M., *Revisiting PatchMatch Multi-View Stereo for Urban 3D Reconstruction*, IEEE Intelligent Vehicles Symposium (IV), 2022.

2.3.1 Motivation

One of the main challenges in autonomous driving is to build a complete and reliable 3D representation of the environment around the vehicle. Since the classical sensor fusion approach requires a complex sensor suite that is both expensive and challenging to calibrate, recently there is a growing interest towards image-based 3D reconstruction in urban scenarios. This is motivated by practical applications, such as photorealistic offline simulation, ground truth generation for neural networks and semantic 3D understanding [94, 95, 96]. Moreover, self-driving cars are typically equipped with an array of cameras. Here we consider the simplest case of a single forward-facing camera, and we generalize to a multi-camera setup in Section 2.4.

The 3D reconstruction pipeline described in Section 2.2 dominates the leaderboards for complex outdoor scenes [60, 61], but tends to fail in textureless areas and non-Lambertian surfaces. In such regions, matching pixels across multiple views and computing their photometric consistency is prone to errors, thus leading to artifacts and noise in the final 3D model. Furthermore, the local optimization procedure of PatchMatch fails to propagate good solutions in degenerate cases. In this section, we build on recent algorithmic advances in PatchMatch MVS [13, 42, 97] and we propose three key improvements to tackle its failure modes in urban scenarios:

1. The initialization step in Algorithm 2.1 is augmented with approximate 3D geometry information from sparse keypoints.
2. A novel geometric loss function is introduced to ensure consistency between estimated depths and normals, leading to more accurate geometry.
3. A multi-scale interaction between the local optimization of PatchMatch and a global refinement algorithm [98] is proposed to regularize the solution.

The resulting MVS algorithm, coupled with a state-of-the-art visual SLAM system [99], leads to a reliable framework for outdoor monocular 3D reconstruction from images, which is shown in Figure 2.10.

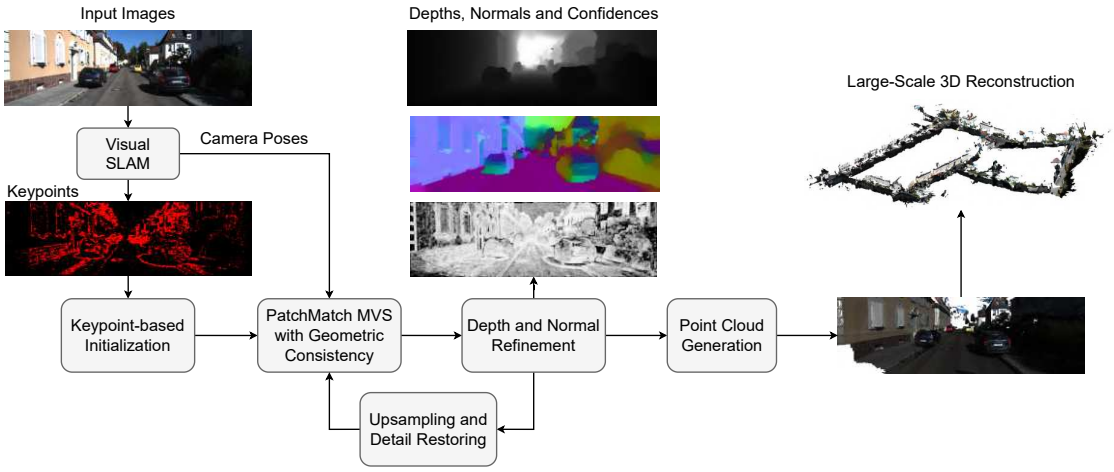


Figure 2.10: Overview of the proposed approach for large-scale 3D reconstruction of urban environments from images [52].

2.3.2 Related Work

Monocular Depth Networks

Monocular depth estimation (MDE) is inherently an ill-posed problem using classical geometry-based methods, since depth information is lost in the 3D-2D projection described by Equation 1.1. However, neural networks can be trained to predict depth from a single image, by exploiting the relationship between appearance and geometry in large-scale datasets. Early works in learning-based MDE proposed simple convolutional architectures, trained in a fully supervised way from additional LiDAR sensors [101, 102]. Despite showing promising results, acquiring ground truth data for depth is expensive and the resulting supervision is still sparse, since LiDAR rays do not have a one-to-one mapping with camera pixels.

In order to avoid this issue, Zhou *et al.* [100] pioneered a fully self-supervised approach to jointly learn depth and pose by using view synthesis as a proxy task. Recent improvements in network architectures and loss functions [4, 103, 1] have shown performances almost on par with supervised methods. The basic idea is that source views in $\mathcal{I}_{src}$ can be differentiably warped onto the reference image plane I_{ref} by using the depth and the relative pose predicted by two separate neural networks. If the predicted depth and pose are correct, then this warping should generate exactly I_{ref}, up to occlusions, that can be used a self-supervision (see Figure 2.11).

The main advantage of these methods is that they learn a direct image-to-depth mapping, which can be directly deployed for real-time navigation of autonomous vehicles. Moreover, they usually deal effectively with known problems of classical methods, such as moving objects

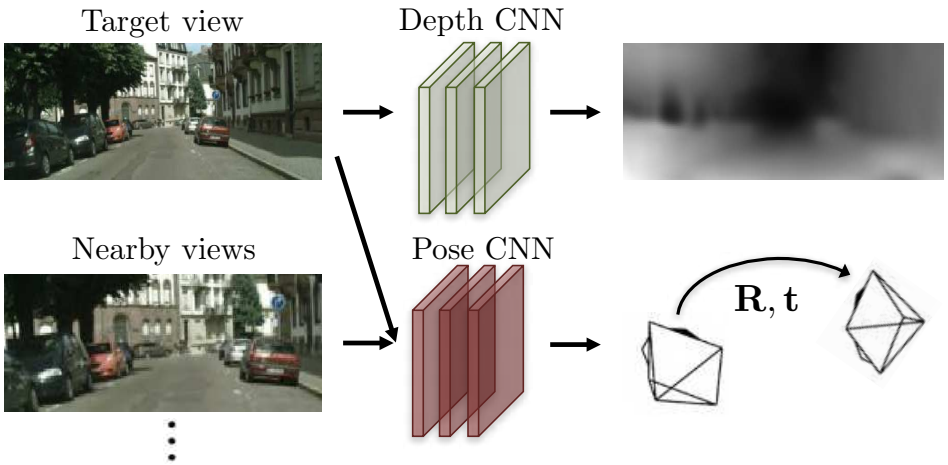


Figure 2.11: Self-supervised monocular depth estimation networks are trained to predict a depth and a relative pose that allow to warp source views onto the reference image plane [100].

that break multi-view photometric consistency. However, neural networks are usually limited in the input resolution and they require fine-tuning on unseen data, as they do not generalize zero-shot to different camera configurations. Furthermore, pixelwise normals are typically not handled, despite few notable exceptions [104, 105]. This is a critical issue when the downstream task is 3D reconstruction, since the joint estimation of depth and normals improves the geometric consistency of the resulting model.

Advances in PatchMatch MVS

The PatchMatch MVS algorithm presented in Algorithm 2.1 is based on seminal papers [3, 89, 87] that progressively improved its building blocks and allowed for an efficient GPU implementation with massive parallelization. Recently, several works proposed to enhance the framework by introducing multi-scale estimation [13], geometric consistency [3, 13], planar priors [42] and semantic information [90, 106].

In order to deal with textureless areas, Xu *et al.* [13] pioneered the idea of estimating geometry on multiple scales. When using image pyramids, patches at coarser scales have significantly more texture if the patch size is fixed, and photometric consistency can be measured with higher fidelity. Depths and normals at finer scales are then initialized with upsampled outputs of previous levels [107]. Three hierarchy levels are typically enough for improved results.

Another way to improve the geometry produced by PatchMatch MVS is to explicitly enforce geometric consistency between depth estimates. This is done by extending Equation 2.3 to

include a geometric cost, which is typically the forward-backward reprojection error [3, 13]. A pixel $\mathbf{p}_{ref}$ in the reference image is back-projected in 3D according to its current depth estimate by following Equation 2.5 and observed in each source image as the pixel $\mathbf{p}_i$. Then, it is warped again onto the reference view with the source depth hypothesis as $\hat{\mathbf{p}}_i$ and the following cost is computed, for each image I_i and each depth hypothesis d_k:

$$c_{ik} = \min\left(||\mathbf{p}_{ref} - \hat{\mathbf{p}}_i||^2, \tau\right) \tag{2.6}$$

where τ is a robust threshold against occlusions. This cost is aggregated across multiple views with the same weights of Equation 2.2 and the term $\mathcal{L}_{geo}(d_k)$ is added to Equation 2.3.

Furthermore, since textureless areas usually correspond to piecewise planar surfaces in the scene, planar priors can be added to PatchMatch optimization to regularize the solution in those regions [42]. Such priors are derived by building a Delaunay triangulation on pixels with sparse but reliable correspondences, i.e. having a matching cost lower than a given threshold. The vertices of each triangle are then projected in 3D and the parameters of the plane through these three points are computed. This gives a prior depth and normal for all the pixels inside a given triangle. Equation 2.3 is extended to include $\mathcal{L}_{plane}(d, \mathbf{n})$ and we refer the reader to [42] for its detailed formulation.

We combine all these elements in a unified multi-scale framework with planar priors and geometric consistency, which we refer to as our *baseline*. Note that this combination has also been proposed in [108], which was published *after* our work [52]. Moreover, we do not include semantic information [90, 106], as it would require an additional output from a segmentation network for each frame, and we only rely on raw images as input.

2.3.3 Method

Keypoint-based Initialization

The usual practice in existing literature [89, 51, 87, 13, 42] is to start from a random initial solution, as presented in the initialization phase of Algorithm 2.1. However, it is common to have a depth guess for at least a sparse subset of pixels. For example, a set of well-triangulated 2D features and corresponding 3D keypoints are a by-product of SfM algorithms and SLAM systems. This is essentially a free approximation of the scene geometry that is currently overlooked in 3D reconstruction pipelines. We propose to reproject this sparse point cloud onto each image and use the corresponding depth values to bootstrap the optimization.

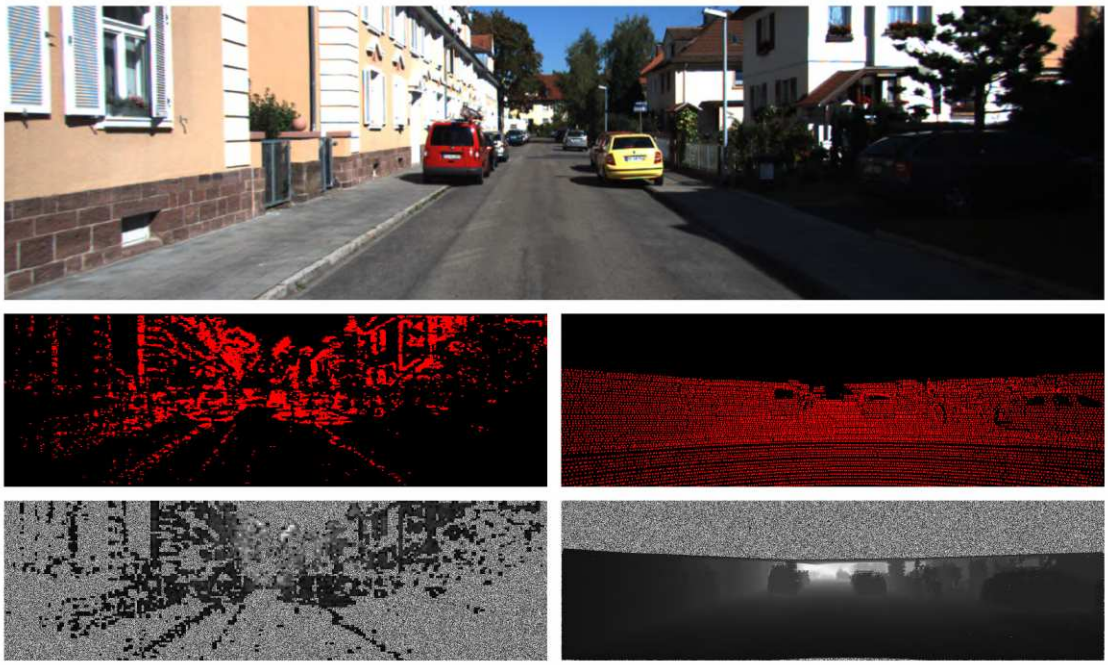


Figure 2.12: Examples of the proposed PatchMatch initialization with densified visual SLAM keypoints (left) and triangulated sparse LiDAR points (right) for the input image (top) [52].

A possible way to integrate sparse 3D points into the initialization phase is to compute a Delaunay triangulation on the valid pixels and fit plane parameters to each triangle, as done for estimating the planar priors in [42]. This approach assumes that features are regularly distributed in the image, which is the case for classical MVS datasets [61, 60] with wide baselines and well-textured scenes [109]. On the other hand, keypoints from visual SLAM in urban scenarios suffer from very low triangulation angles between frames and they are usually concentrated in specific areas, while being almost absent in large textureless regions, such as roads. Moreover, noisy and erroneous matches are much more frequent in real-world monocular outdoor data, thus making this approach unstable and unreliable.

Therefore, in our framework we simply densify the initial solution by propagating each depth value in a local support region of 5×5 pixels, as shown in Figure 2.12 (left). These hypotheses are then perturbed with random Gaussian noise to promote diversification, while pixels too far from any keypoint are still randomly initialized. For the sake of completeness, we also show in Figure 2.12 (right) that the Delaunay triangulation method is well-suited for urban data when sparse measurements from a LiDAR sensor are available.

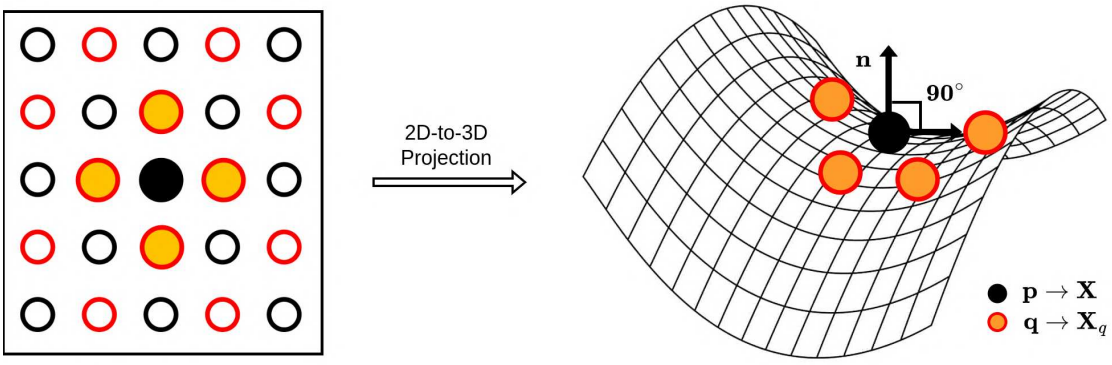


Figure 2.13: Visualization of the self-consistency between a depth value and a normal vector, which must be locally perpendicular to the 3D surface in scene space.

Improved Geometric Consistency

The geometric consistency term in Equation 2.6 does not fully exploit the local planar geometry of the scene, since such consistency is enforced only of depth estimates and not for normal vectors. To this end, we draw inspiration from recent monocular depth estimation literature [110, 111] and propose to add another geometric loss function that ensures self-consistency between the depth value and the normal vector of a given pixel $\mathbf{p}$.

Formally, a normal vector $\mathbf{n}$ is *consistent* with the corresponding depth value d if it is locally perpendicular to the surface at the corresponding 3D point $\mathbf{X}$, which is computed with Equation 2.5. This constraint, which is visualized in Figure 2.13, can be enforced by back-projecting in 3D a set Ω of pixels around $\mathbf{p}$ and formulating the following cost function:

$$\mathcal{L}_{cons}(d, \mathbf{n}) = \sum_{\mathbf{q} \in \Omega} w_q \cdot \mathbf{n}^\top (\mathbf{X}_q - \mathbf{X}) \tag{2.7}$$

where $\mathbf{X}_q$ is the 3D point corresponding to pixel $\mathbf{q}$ and $w_q = e^{-||I(\mathbf{q}) - I(\mathbf{p})||}$ downweights pixels with different color values, which are less likely to belong to the same surface. Note that this formulation optimizes self-consistency between depths and normals within the reference frame and does not require to warp pixels on source views.

Finally, the overall loss at each evaluation phase in our framework is the sum of all the previously introduced terms, including photometric, geometric and planar consistency:

$$(d, \mathbf{n}) = \underset{(d_k, \mathbf{n}_k) \in \mathcal{S}}{\arg\min} \left(\mathcal{L}_{photo}(d_k, \mathbf{n}_k) + \mathcal{L}_{plane}(d_k, \mathbf{n}_k) + \mathcal{L}_{geo}(d_k) + \mathcal{L}_{cons}(d_k, \mathbf{n}_k) \right) \tag{2.8}$$

Figure 2.14: Visualization of pixelwise confidence (right) for an example input frame (left) [52].

Global Refinement

Despite the efforts in optimizing consistency terms for both multi-view appearance and geometry in Equation 2.8, the output depth and normal maps might still contain errors that can harm the accuracy and the completeness of the resulting 3D model. Existing works [13, 42, 3] partially solve this issue by performing a simple median filtering after each iteration, in order to remove outliers. However, the root cause of most errors is the inherently local nature of the PatchMatch algorithm, since the global context is never considered during optimization. Therefore, we propose to leverage a confidence-based global refinement algorithm [98] at each scale to regularize the predicted geometry.

Let $(\mathbf{D}, \mathbf{N})$ be the pixelwise depths and normals of the reference image at the current scale and $\mathbf{C}$ a confidence map with values $c_p \in [0, 1]$ for each pixel $\mathbf{p}$, where 0 means that the estimate is unreliable and 1 that we can strongly rely on it. We define such confidence as a smooth function of the forward-backward reprojection error e_p:

$$c_p = \exp\left(-\left(\frac{e_p}{\bar{e}}\right)^2\right) \tag{2.9}$$

where $\bar{e}$ is the mean error over the entire set of observations. A visualization of the resulting confidence map is given in Figure 2.14 (right), where it can be seen that pixels with low confidence correspond to textureless regions or areas with non-Lambertian illumination, while high confidence is assigned to distinctive pixels.

The refinement procedure optimizes both unary terms that penalizes deviations from reliable input values and a binary term that promotes globally smooth solutions:

$$\mathbf{D}^*, \mathbf{N}^* = \underset{\mathbf{D},\mathbf{N}}{\arg\min} \left(\mathcal{L}_{unary}(\mathbf{D}, \mathbf{C}) + \mathcal{L}_{unary}(\mathbf{N}, \mathbf{C}) + \mathcal{L}_{binary}(\mathbf{D}, \mathbf{N}, \mathbf{C})\right) \tag{2.10}$$

The global smoothness is guaranteed by building a graph with reliable pixels as nodes. Each edge of the graph is then associated with a weight that takes into account the distance between the nodes in image space and the likelihood of belonging to the same surface in scene space.

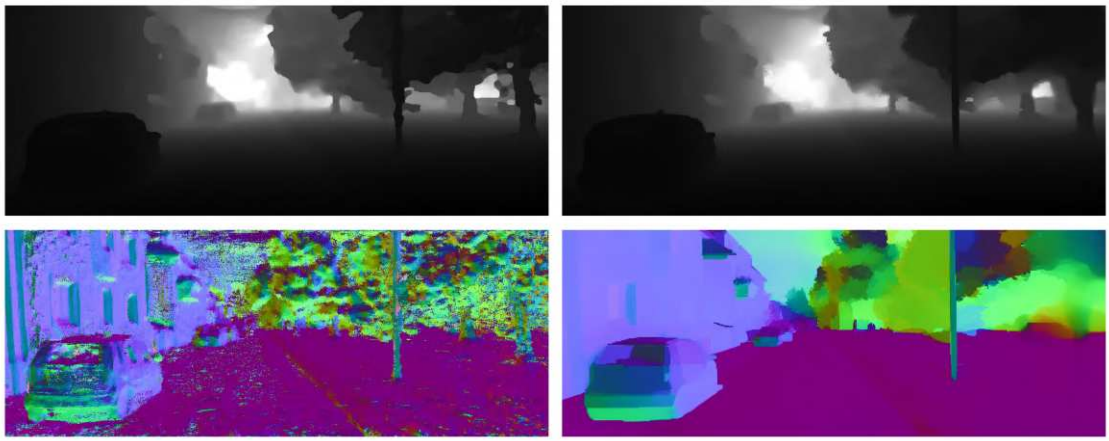


Figure 2.15: Comparison of raw (left) and refined (right) depths (top) and normals (bottom) for the input image in Figure 2.14 [52].

We refer the reader to [98] for a detailed description of such weights and the loss terms for regularization. Figure 2.15 shows a comparison between raw geometry from PatchMatch and the outputs refined by the proposed algorithm.

Note that while a previous work [97] adopted the same refinement approach to MVS output, the key improvements of the presented method are (i) a much simpler confidence metric and (ii) the use of the algorithm *at each scale* of the pyramid, rather than a single time at the end. This means that the multi-view consistency of refined solutions is further enforced at finer scales, thus producing more complete and accurate 3D models.

2.3.4 Experiments

Implementation Details

The whole framework has been implemented in C++/CUDA and the code runs on a single NVIDIA GTX 1080Ti GPU. Camera poses and sparse keypoints for initialization are computed with the visual SLAM system DVSO [99], in order to enable 3D reconstruction from images alone. PatchMatch optimization is performed for $N_{iter} = 8$ iterations and the reprojection error is clamped at $\tau = 2$ pixels, which is also the threshold for marking a 3D point as valid during the point cloud generation step. Moreover, a 3D point must also have a relative depth difference lower than 0.01 m and an angle between normals below 10 for at least two views.

Method	Abs Rel	Sq Rel	RMSE	RMSE_{log}	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
MonoDepth2 [4]	0.115	0.882	4.701	0.190	0.879	0.961	0.982
PackNet [103]	0.107	0.882	4.538	0.186	0.889	0.962	0.981
ManyDepth [112]	0.087	0.685	4.142	0.167	0.920	0.968	0.983
DORN [102]	0.077	0.290	2.723	0.113	0.949	0.988	0.996
BTS [113]	0.059	0.245	2.756	0.096	0.956	0.993	0.998
MiDAS [1]	0.062	-	2.573	0.092	0.959	0.995	**0.999**
Colmap [51]	0.099	3.451	5.632	0.184	0.952	0.979	0.986
ACMM [13]	0.042	0.498	2.871	0.166	0.982	0.989	0.992
ACMP [42]	0.034	0.381	1.930	0.152	0.987	0.992	0.994
DeepMVS [114]	0.088	0.644	3.191	0.146	0.914	0.955	0.982
DeepTAM [115]	0.053	0.351	2.480	0.089	0.971	0.990	0.995
MonoRec [50]	0.050	0.295	2.266	0.082	0.973	0.991	0.996
Ours	**0.025**	**0.201**	**1.459**	**0.069**	**0.992**	**0.996**	0.998

Table 2.1: Quantitative results on the KITTI dataset. Best and second results are marked **bold** and underlined, respectively. Yellow columns are error metrics where lower is better, orange columns are accuracy metrics where higher is better [101].

Quantitative Evaluation

We evaluate the proposed framework on the KITTI dataset [116], which is captured in the city of Karlsruhe, Germany, with an autonomous driving platform equipped with a LiDAR sensor to provide ground truth geometry and two front cameras with resolution 1382×512 pixels. Depth maps are computed at full resolution and clamped at 80 m for computing metrics. Consistently with recent literature, the improved ground truth from [117] and the metrics from [101] are used for evaluation. Let d_{gt} be the ground truth value for a predicted depth d in a depth map $\mathbf{D}$ with size $W \times H$. The following metrics are averaged across all the frames:

$$
\begin{aligned}
\text{Abs Rel} &= \frac{1}{WH} \sum_{d \in \mathbf{D}} \frac{|d - d_{gt}|}{d_{gt}} \\
\text{Sq Rel} &= \frac{1}{WH} \sum_{d \in \mathbf{D}} \frac{||d - d_{gt}||^2}{d_{gt}} \\
\text{RMSE} &= \sqrt{\frac{1}{WH} \sum_{d \in \mathbf{D}} ||d - d_{gt}||^2} \\
\text{RMSE}_{log} &= \sqrt{\frac{1}{WH} \sum_{d \in \mathbf{D}} ||\log d - \log d_{gt}||^2}
\end{aligned}
\tag{2.11}
$$

Method	MS	KP	GC	Ref	Abs Rel	Sq Rel	RMSE	RMSE$_{log}$
Baseline [108]	✗	✗	✗	✗	0.034	0.381	1.930	0.152
Baseline (ref) [97]	✓	✗	✗	✓	<u>0.031</u>	0.284	1.716	<u>0.074</u>
Ours	✓	✗	✗	✗	0.033	0.321	1.882	0.102
Ours	✓	✓	✗	✗	0.032	0.305	1.603	0.081
Ours	✓	✓	✓	✗	0.032	<u>0.273</u>	<u>1.586</u>	0.076
Ours (full)	✓	✓	✓	✓	**0.025**	**0.201**	**1.459**	**0.069**

Table 2.2: Ablation studies on the KITTI dataset. Best and second results are marked **bold** and <u>underlined</u>, respectively. In all the metrics, lower is better. Legend: MS - Multi-Scale, KP - KeyPoint-based initialization, GC - our Geometric Consistency, Ref - global Refinement.

Finally, the ∂ parameter in orange boxes of Table 2.1 is defined as:

$$\partial = \max\left(\frac{d}{d_{gt}}, \frac{d_{gt}}{d}\right) \tag{2.12}$$

Previous works typically perform quantitative evalutation on the Eigen test split [101], but MVS methods require temporally adjacent frames with estimated poses. Therefore, we generate the results on the intersection between the KITTI odometry benchmark and the Eigen test split, following [50]. This procedure leads to 8634 images for testing.

We compare our approach to a broad range of methods, including both classical [51, 13, 42] and learning-based [50, 114, 115] MVS algorithms. In addition, we selected both self-supervised [103, 4, 112] and fully supervised state-of-the-art monocular depth networks [102, 113], as well as a recent vision transformer-based approach [1]. Quantitative results are shown in Table 2.1, where it can be seen that our contributions significantly boost the performances of concurrent methods. Moreover, note that the gap with respect to monocular networks is shared by most MVS approaches, due to their ability of exploiting information about camera poses and adjacent frames, rather than directly trying to map images to depth.

Ablation Studies

In order to quantify separately the effect of each contribution on the predicted geometry, ablation studies are presented in Table 2.2, showing that all our novel components consistently increase the depth estimation metrics compared to the baseline [108]. The two elements with the greatest influence are the global refinement algorithm and the multi-scale estimation framework. This can also be noticed when comparing the baseline to another work [97] that makes

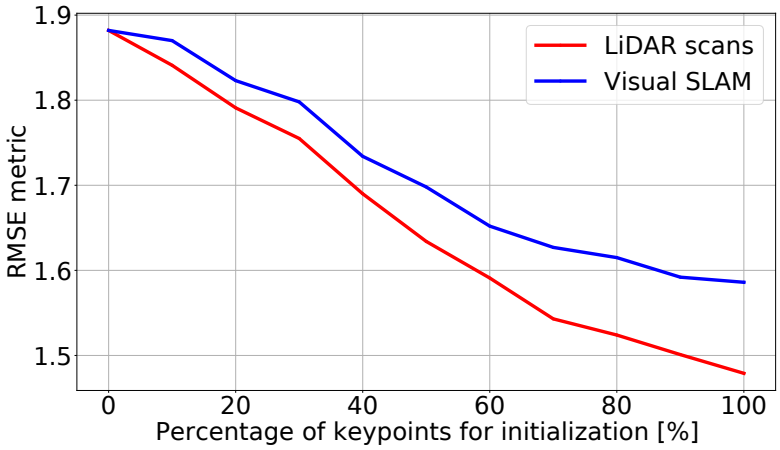


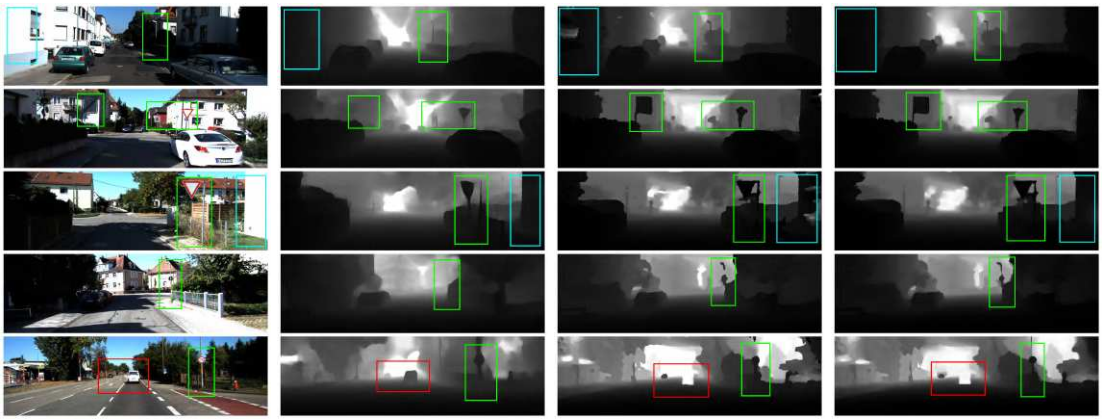
Figure 2.16: RMSE as a function of the number of input depth values for both LiDAR data and visual SLAM keypoints.

use of the same refining step. Moreover, keypoint-based initialization and the novel depth-normal consistency term allow to further boost the performances of the proposed method.

We also performed additional experiments to investigate deeper the influence of the quantity and the quality of initial depth values on the final results. While our framework is designed to work with raw images as input, the KITTI dataset provides sparse LiDAR scans aligned to each frame, and this is a fairly common setup in autonomous driving. For this reason, Figure 2.16 shows the RMSE metric as a function of the number of points for both LiDAR and visual SLAM input data. Intuitively, adding more initial guesses keeps improving the results, and in general LiDAR points give a better initialization than SLAM keypoints. This is motivated by the fact that laser scanners can provide depth in textureless regions where SLAM typically fails to extract keypoints, which are also the same areas in which MVS struggles the most.

Qualitative Results

We provide a qualitative comparison against state-of-the-art in Figure 2.17, which shows the depth maps computed by two self-supervised monocular networks [4, 103], the baseline multi-scale PatchMatch MVS with planar priors [108], and the proposed approach. While learning-based methods provide smooth and visually pleasant results, they also miss out on details in fine-grained structures, such as light poles and traffic signals, which are crucial for safe autonomous navigation. Moreover, note how the local-only optimization strategy of [108] leads to noise and artifacts in large textureless areas. Our method can effectively deal with both issues, by combining strong geometric consistency and global regularization.



(a) Input image **(b)** MonoDepth2 [4] **(c)** Baseline [108] **(d)** Ours

Figure 2.17: Qualitative results on the KITTI dataset. Bounding box legend: Green - traffic signals detail, Cyan - textureless areas, Red - moving objects. Differences are best viewed when zoomed in.

Figure 2.18: 3D reconstruction results from sequence 05 of the KITTI odometry dataset: individual snapshots (left) of the large-scale point cloud (right, top), which is consistent with the estimated camera trajectory (right, bottom).

Figure 2.17 also shows that both our approach and the baseline [108] fail to recover the correct depth of a moving object. This is a typical failure case of MVS algorithms, as the underlying assumption of static environment is not satisfied. On the other hand, the direct image-to-depth mapping learned by monocular networks can still predict a reliable result. While a recent work [50] proposed an effective way to deal with dynamic objects in MVS systems, this issue is mitigated when depth is used for 3D reconstruction as downstream task, for two main reasons. Firstly, moving objects violate the multi-view consistency needed for a pixel to be back-projected in 3D, therefore those pixels are implicitly masked away. Secondly, the perception stack of self-driving cars typically includes a segmentation network [118], that can be used to skip those areas explicitly.

Finally, the 3D reconstruction of a complete sequence from the KITTI odometry dataset is shown in Figure 2.18. The extremely dense and large-scale point cloud has been obtained by computing pixelwise depths and normals with the proposed improved MVS algorithm, which are then back-projected in 3D with Equation 2.5. The resulting 3D model (right, top) is consistent with the estimated trajectory (right, bottom) and individual snapshots (left) show the density and the texture richness of the point cloud. The incomplete parts are mainly due to the severely limited field of view of the frontal camera in the dataset. Additional reconstructions in urban scenarios with a full array of cameras are shown in Section 2.4.

2.4 Scaling to Entire Cities

This section is based on the original contributions published in [53]:
Orsingher M., Zani P., Medici P., and Bertozzi M., *Efficient View Clustering and Selection for City-Scale 3D Reconstruction*, International Conference on Image Analysis and Processing (ICIAP), 2022.

2.4.1 Motivation

In the previous section, we presented a complete pipeline for *monocular* 3D reconstruction of urban environments from images acquired by the frontal camera of a self-driving car. However, modern autonomous vehicles are usually equipped with a multi-camera array on top [119], in order to cover the full 360° visible range. A naive way to exploit this additional information would be to run 3D reconstruction independently for each camera and merge the resulting

point clouds. This is inefficient and suboptimal, since strong multi-view constraints exist across both different cameras at the same time instant and the same camera at different frames.

Another frequent situation overlooked by the framework presented in Section 2.3 is when the front-left (or front-right) camera at time t shares visibility with the back-left (or back-right) camera at time $t + k$ (for some $k > 0$). Optimal viewpoints for triangulating 3D points might be acquired by different sensors at very different time instants. Furthermore, considering only temporally adjacent frames as $\mathcal{I}_{src}$ in Algorithm 2.1 is flawed when the vehicle revisits the same place multiple times, since the same geometry would be re-computed unnecessarily. Finally, scaling 3D reconstruction up to entire cities when multiple cameras generate data at high frame rate poses several efficiency challenges.

In this section, we modify the proposed framework to support city-scale 3D reconstruction from arbitrary camera configurations. We still assume to have calibrated images as input, as well as a sparse set of keypoint computed by either SfM or SLAM. Then, we present two original contributions to enable such scaling:

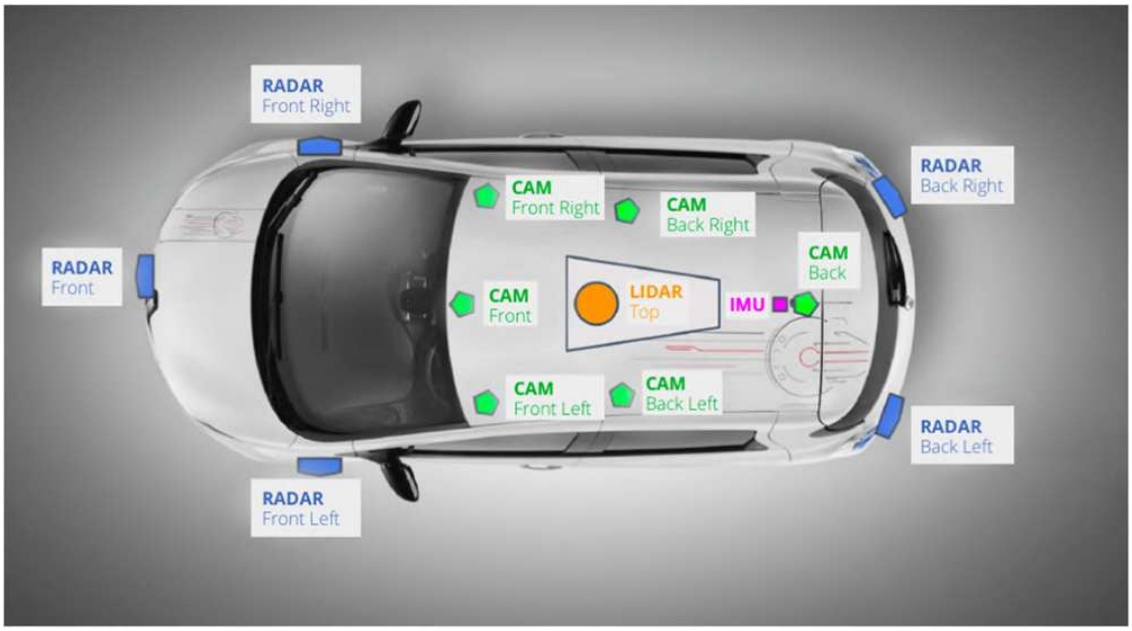
1. A *view clustering* algorithm is designed to identify local clusters of images with shared visibility and to allow for parallelization across such clusters.

2. A *view selection* procedure is proposed to select a representative subset of the input images, thus further boosting the efficiency.

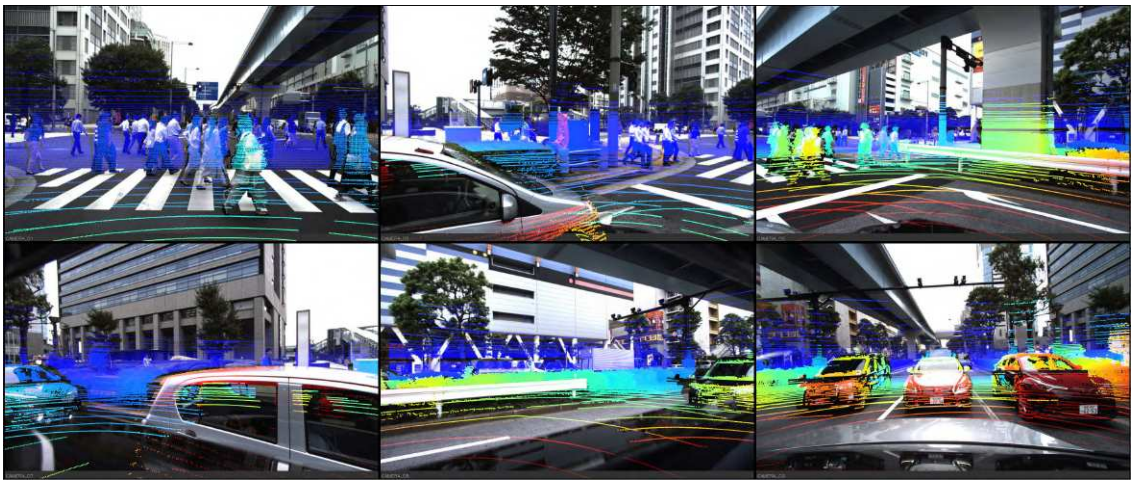
2.4.2 Related Work

Nowadays, arbitrarily large datasets of high-resolution images have become available thanks to almost unlimited sources of data, such as Internet pictures uploaded by millions of users on Social Media. While this large amount of data can be used to effectively reconstruct a precise 3D view of the world, the increasing size and high redundancy introduce both feasibility and efficiency challenges.

Most approaches in the literature have been developed to reconstruct large buildings using community photo collections [46, 48, 121]. These works rely on unique and recurrent architectonic features that belong to the building of interest, when computing shared visibility information among images. Furukawa *et al.* [48] propose to perform global view selection on the whole set of images to remove redundancy and to build a visibility graph with the remaining cameras. These similarities are collected in a *similarity matrix* that represents the adjacency matrix of the visibility graph. Then, an optimization procedure is applied to iteratively divide



(a) Sensor suite of the nuTonomy self-driving car, with six cameras covering 360° around the vehicle [120].



(b) Visibility of a typical multi-camera array and reprojected LiDAR scans [119].

Figure 2.19: Modern autonomous vehicles have multiple sensors with cross-view constraints and full coverage of their surroundings.

the graph into clusters using normalized-cuts, enforcing a size constraint, and add cameras back if a coverage constraint is violated. This process is repeated until convergence.

Several other methods are based on this visibility graph formulation. Ladikos *et al.* [122] apply spectral graph theory and use mean shift to select the number of clusters, while Mauro *et al.* [123] employ the game theoretic model of dominant sets to find regular overlapping clusters. In a subsequent work [124], they propose to place selection after clustering and formulate an *Integer Linear Programming* (ILP) problem with cameras as binary variables. The goal is to select the minimum number of views, such that coverage and matchability are guaranteed. We adopt the same approach, but provide a more efficient alternative by exploiting the fact that neighboring keypoints are likely to share the same camera subgraph, while their formulation requires to execute the expensive Bron-Kerbosch algorithm [125] for each keypoint in the cluster.

Furthermore, all the methods presented so far cluster images according to their relative visibility information and without considering the 3D structure of the scene. Zhang *et al.* [121] suggest performing joint camera clustering and surface segmentation, which are formulated as a constrained energy minimization problem. Similarly, we propose a clustering algorithm which operates directly in 3D by exploiting the approximately uniform distribution of poses and geometry as produced by a moving vehicle.

Existing works designed for architectural Internet datasets fail to tackle efficiently the unique set of challenges posed by large-scale urban 3D reconstruction. Firstly, they strongly rely on well-distributed keypoints from SLAM/SfM, which are difficult to extract in urban scenarios due to the presence of large portions of textureless surfaces, such as roads, vegetation or sky, as we already discussed. Secondly, moving vehicles provide an unprecedented amount of data and existing algorithms are limited to significantly smaller datasets, thus making them impractical in the considered setting.

2.4.3 Method

An overview of our approach for city-scale 3D reconstruction is shown in Figure 2.20. We propose to interleave SfM/SLAM and MVS with a view clustering algorithm that operates directly on the output of sparse reconstruction to find independent local clusters of images, followed by a view selection step that reduces visibility redundancies. In this way, dense reconstruction can be performed in parallel for each cluster, thus enabling unbounded scaling. Finally, local point clouds are merged to a large-scale 3D model.

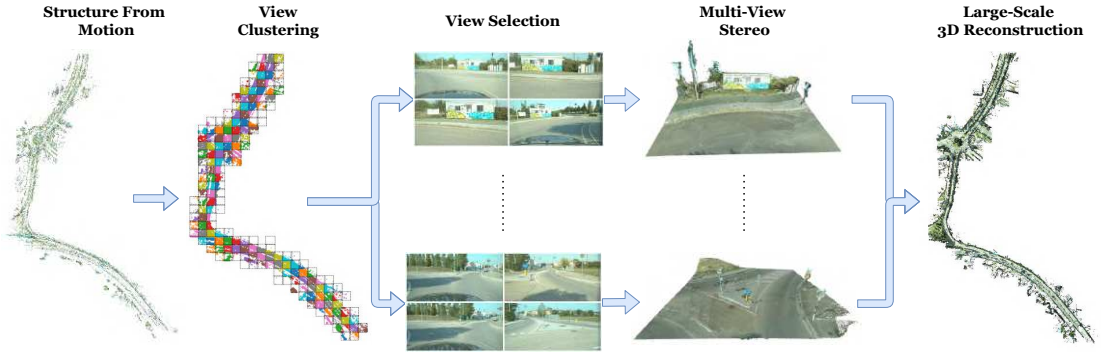


Figure 2.20: Overview of the proposed approach for city-scale 3D reconstruction from multi-camera systems [43].

View Clustering

Let $\mathcal{P}$ be a set of calibrated images acquired by a moving vehicle with N_{cams} cameras in arbitrary configuration, each with frame rate f. Calibration parameters have been obtained by either SfM [51] or SLAM [99], which also generates a sparse set of keypoints $\mathcal{X}$. The view clustering algorithm starts by building a street-level 2D grid with fixed block size (x_b, y_b) and a degree of overlap $d_{overlap}$, within the range of the sparse keypoints. Figure 2.21 (left) shows that this procedure generates lots of empty clusters, as well as several blocks with noisy keypoints far from the vehicle trajectory. Therefore, we adopt a filtering step to remove redundant clusters and assign each keypoint to the corresponding block by projecting it onto the (x, y) plane, as visualized in Figure 2.21 (center). In this way, the approximate 3D geometry of the scene is divided into partially overlapping regions that can be processed independently and in parallel.

At this point, we need to associate each camera to its visible clusters. Such visibility could be naively based on the number and the quality of keypoints seen by each camera for each cluster. However, due to their high sparsity and irregular distribution in urban scenarios, we propose to augment the 3D information of each non-empty cluster by sampling uniform points with resolution r within the boundaries of the corresponding block. As $r \to 0$, this is effectively an approximation of the intersection between the viewing frustum of the camera and the cluster itself, thus allowing to associate cameras to clusters within their field of view, even if keypoints were not extracted at that specific location.

Specifically, for each block, we project the augmented 3D points onto each camera within a given distance from its centroid and associate to the cluster all the cameras with at least one valid projection. Then, clusters with less than 10 views are iteratively merged with their neighbors, in order to provide enough information for the MVS phase. The output of the view clustering

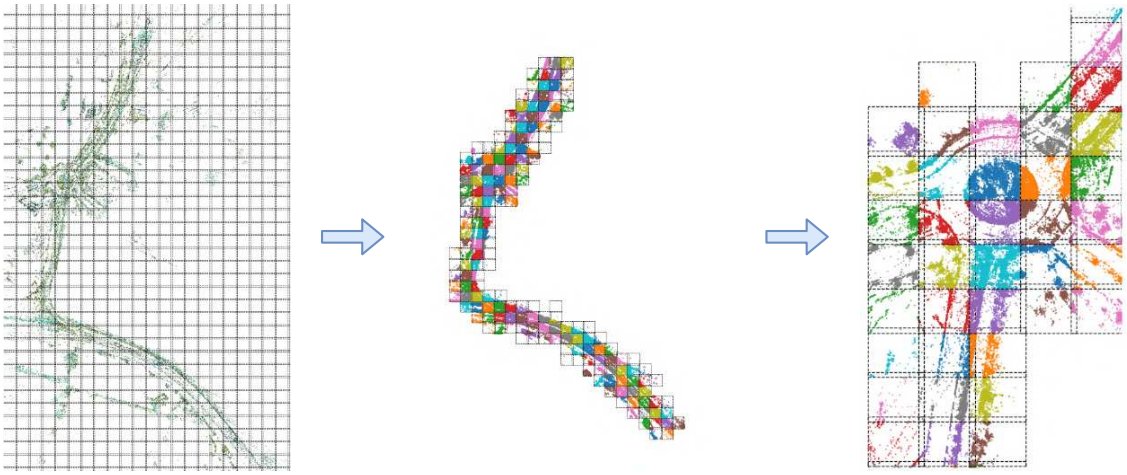


Figure 2.21: View clustering example: the raw 2D grid built from SfM (left), the full sequence clustered and filtered (center) and overlapping clusters in more detail (right). Each cluster is represented with its borders and in a different color [43].

algorithm is a set of C independent clusters of cameras with shared visibility, shown in detail in Figure 2.21 (right). This divide and conquer approach makes city-scale 3D reconstruction tractable and efficient, while guaranteeing the completeness of the final geometry with a sufficient degree of overlap $d_{overlap} > 0$.

From an algorithmic point of view, our main contribution is to avoid the computation of complex pairwise relationships between cameras for the whole dataset [48, 124], which scales quadratically as $O(N^2)$, where N is the total number of images. In urban scenarios, up to $N = 60 \times f \times N_{cams}$ frames are acquired every minute and the similarity matrix approach would become quickly impractical. On the other hand, our clustering algorithm can assign the same view to at most K neighboring blocks, *independently* from the size of the scene. Even in the worst case scenario where each cluster has $N_c = K\frac{N}{C}$ cameras associated, computing C separate and smaller visibility graphs is still significantly cheaper. More formally, this would require $O\left(\sum_{c=1}^{C} N_c^2\right)$ operations:

$$\sum_{c=1}^{C} N_c^2 = \frac{K^2 N^2}{C} < N^2 \iff K < \sqrt{C} \tag{2.13}$$

In practical cases, $K \leq 8$ by design, since a camera is associated at most to a cluster and its immediate neighbors, while C can easily grow to several hundreds or even thousands as the vehicle explores new areas of the world. Moreover, this improvement gap expands as the dataset size increases, and this is the key reason why the proposed approach can scale up to arbitrarily large

scenarios. While this analysis holds true even when each block is reconstructed *sequentially*, we underline that the proposed clustering procedure enables a high degree of *parallelization*, since each cluster can be reconstructed independently.

View Selection

In the view clustering phase, we associate a camera to each cluster where at least a single point is visible. On one hand, this ensures that depth maps during MVS will be computed with all the information available. On the other hand, it is also likely to produce a highly redundant set of cameras for each block, which is in contrast with our efficiency needs. To this end, we also design a view selection algorithm to choose the optimal subset of views for each cluster. In this context, *optimal* means the smallest subset of cameras that satisfy the following constraints:

- *Visibility* $\rightarrow$ Each point in the cluster must be seen by at least N_{vis} cameras.
- *Matchability* $\rightarrow$ Each camera must have at least N_{match} other cameras to be successfully matched with. Two cameras are considered to be *matchable* if they see a sufficient number of common points.

Differently from previous literature [48, 124], we avoid using the mean Gaussian-weighted triangulation angle between the two cameras and their shared features as a similarity measure. In the considered settings, keypoints are too sparse for this to be a reliable metric and Gaussian parameters would need to be tuned separately for each block. Therefore, an ILP problem is formulated for each cluster, with binary variables $x_i \in \{0, 1\}$ representing cameras:

$$\begin{aligned} \min \quad & \sum_{i=1}^{N_c} x_i \\ \text{s.t.} \quad & \sum_{i=1}^{N_c} x_i \geq N_{min} \\ & \mathbf{A}_i^\top \mathbf{x} \geq 0 \qquad \forall i = 1, \dots, N_c \\ & \mathbf{B}_j^\top \mathbf{x} \geq N_{vis} \qquad \forall j = 1, \dots, P_c \end{aligned} \tag{2.14}$$

A linear formulation for the first constraint can be derived from the visibility graph $\mathbf{S}_c$ of the considered cluster as follows. Let $\tilde{\mathbf{S}}_c = \mathbf{S}_c > 0$ be a binary matrix with $\tilde{s}_{c,ij} = 1$ if cameras i and j share common keypoints, 0 otherwise. The constraint vectors $\mathbf{A}_i$ are computed as the rows of the matrix $\mathbf{A} = \tilde{\mathbf{S}}_c - N_{match} \cdot \mathbf{I}$, being $\mathbf{I}$ the identity matrix of size $N_c \times N_c$. This effectively activates the constraint only for the selected cameras, ignoring the others.

Similarly, we can build the visibility constraint vectors $\mathbf{B}_j$ for each point in the cluster with binary coefficients $b_{ij} = 1$ if the point j is visible in camera i, 0 otherwise. This is a crucial improvement with respect to [124], where each point must be seen by at least one clique in the visibility subgraph associated to that point. Their formulation requires to execute the Bron-Kerbosch algorithm [125] for finding maximal cliques separately for each point, which is both inefficient and redundant, since the same cameras are likely to see several points.

Moreover, we further boost the efficiency of our algorithm by providing a good initial guess to the ILP solver. Since a minimum number of cameras N_{min} must be selected for each cluster, we sort them by the number of visible points and force the solver to select the N_{min} cameras with the highest visibility. This threshold is adaptively selected according to the cluster size and clamped between boundary values N_{low} and N_{high}.

2.4.4 Experiments

Implementation Details

To the best of our knowledge, the existing large-scale urban datasets with multi-camera arrays [120, 119] contain relatively short sequences (5-20 s) at a low frame rate ($\approx$ 10 Hz). This makes it difficult to evaluate our approach on those data. Therefore, we acquired custom sequences in Parma, Italy, with a vehicle equipped with $N_{cams} = 7$ cameras with resolution 3840 x 1920 and frame rate $f = 30$ Hz, in order to represent diverse real-world situations.

The software stack of the vehicle provides camera poses and sparse keypoints with a custom SfM algorithm, which constitute the input to our framework. The dense reconstruction phase is executed by the MVS algorithm presented in Section 2.3, while we rely on the open-source library OR-Tools [126] for solving the ILP problem in the view selection step. The algorithms presented in this section have been implemented in C++ and tested on a consumer Intel Core i5-5300U 2.30 GHz CPU. The following set of parameters has been used for experiments: block size $(x_b, y_b) = (20, 20)$ m, with $d_{overlap} = 2$ m; sampling resolution $r = 1$ m; $N_{vis} = N_{match} = 2$, $N_{low} = 10$ and $N_{high} = 30$ for view selection.

Quantitative Evaluation

Table 2.3 provides a quantitative description of the input sequences and the algorithm results for each stage of the pipeline. The first thing to note is that urban data contain relatively few keypoints ($|\mathcal{X}|$), when compared to architectural image sets with similar size [48]. This justifies the choices of sampling additional points for each cluster and avoiding similarity measures

Dataset	Seq. 1	Seq. 2	Seq. 3	Seq. 4	Seq. 5
# views (N)	5131	6782	8477	9912	11956
# keypoints ($\lvert\mathcal{X}\rvert$)	389621	321696	349616	372434	393246
# clusters (C)	156	173	200	261	324
N after clustering	27103	32757	39014	49758	58364
K after clustering	5.28	4.83	4.60	5.02	4.88
Avg. N_c after clustering	173.75	189.34	195.07	190.64	180.13
$t_{clustering}$ (s)	22.35	28.52	32.7	37.32	40.89
N after selection	4951	5842	6266	8411	10138
K after selection	0.96	0.86	0.74	0.84	0.85
Avg. N_c after selection	31.73	33.76	31.33	32.22	31.29
$t_{selection}$ (s)	50.54	76.13	92.4	124.85	144.02
t_{tot} (s)	72.89	104.65	125.1	162.17	184.91

Table 2.3: Quantitative results for each sequence and each stage of the pipeline.

based on the triangulation angle. Secondly, the clustering phase produces extremely redundant data (N after clustering), with each view assigned to approximately 5 different clusters (K after clustering), on average.

Most of the clustering time ($t_{clustering}$) is spent for the camera association procedure, since several hundreds of views must be tested for each cluster. When processing extremely large sequences with millions of images, this phase can be naturally parallelized since each cluster is independent from the others, after the sampling step. Moreover, the optimization algorithm during view selection exploits the clustering redundancy to automatically remove useless views. The number of output views (N after selection) is consistently lower than the input for each sequence, even considering that some cameras covering the overlap between two clusters are assigned to both. Finally, the efficiency condition imposed in equation (2.13) ($K < \sqrt{C}$) is satisfied for each sequence by a large margin, and this gap increases with the dataset size.

Direct quantitative comparison with state-of-the-art approaches is difficult, since they all target a substantially different scenario [48, 121], where the assumption of uniformly distributed poses is violated. In terms of the running time as a function of the input number of images, Figure 2.22 shows that the proposed method is much faster and scales linearly with the dataset size, while [48, 121] both require several hours to process a few thousands of images. This demonstrates that they are not suited for urban applications, where thousands of images are acquired every minute by the vehicle. The considered ILP baseline [124] shares the same issue and numerical performances are available only for datasets smaller by an order of magnitude.

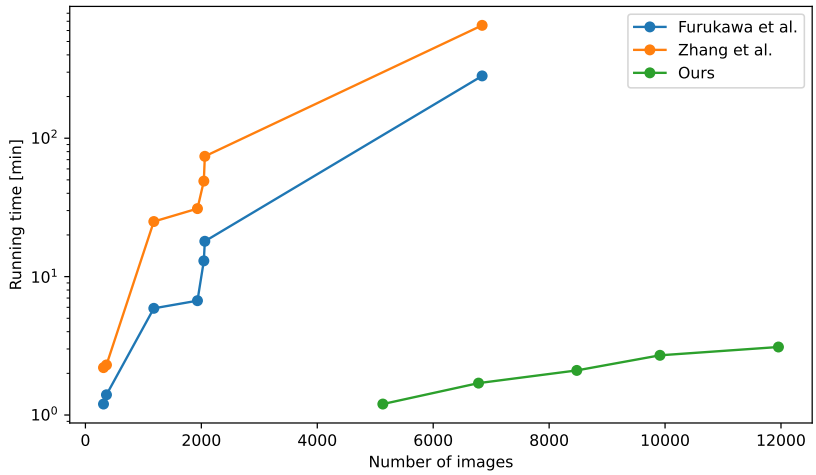


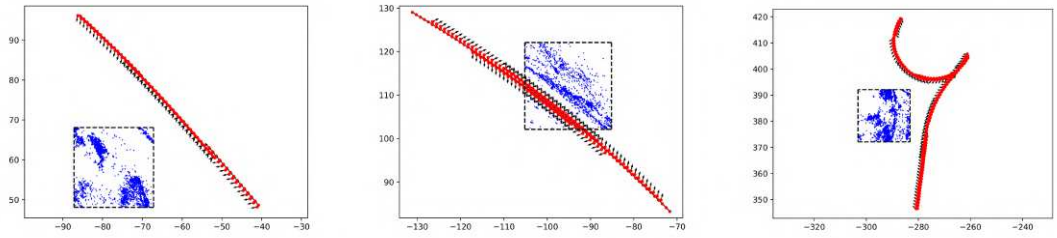
Figure 2.22: Running time comparison between proposed method and state-of-the-art solutions [48, 121]. The Y axis is in log-scale for better visualization [43].

The reported runtime for 706 images divided into 36 clusters is around 3 minutes. However, the Bron-Kerbosch algorithm scales exponentially as $O(3^{\frac{N}{3}})$ with the number of cameras N. In the collected sequences, an instance of such algorithm with $N \approx 10^2$ would be executed for each keypoint of each cluster, making the approach impractical even for very short scenes. On the other hand, the proposed framework can approximately process 4000 images per minute.

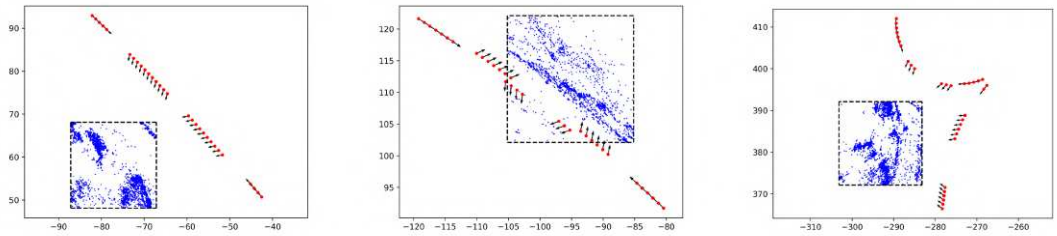
Qualitative Evaluation

Figure 2.23 provides a visualization of clustered cameras, before and after view selection, in order to show how redundancy is exploited and reduced by the optimization algorithm. Three main situations arise in urban data: (i) when all the views lie along a straight line outside the cluster boundaries (Figure 2.23a, left), the ILP solver selects a well-distributed subset of cameras (Figure 2.23a, left); (ii) when the vehicle trajectory intersects the cluster (Figure 2.23b, center), two disjoint sets of cameras are selected (Figure 2.23b, center); (iii) for complex trajectories such as roundabouts (Figure 2.23a, right), the framework generalizes well by selecting cameras from diverse viewpoints (Figure 2.23b, right).

Furthermore, Figure 2.24 shows that the proposed method effectively cluster images based on shared visual content, which allows to produce a detailed 3D reconstruction of the world. Each point cloud has been cropped at the corresponding cluster boundaries. Only qualitative evaluation of the resulting reconstruction is provided, since performances depend solely on the



(a) Clustering: basic (left), intersecting (center) and complex (right).



(b) Selection: basic (left), intersecting (center) and complex (right).

Figure 2.23: Each cluster has **black** dashed borders, blue keypoints and cameras with red viewing direction [43].

choice of MVS algorithm, which has been evaluated in Section 2.3. The goal of the presented approach is to show that 3D reconstruction can be achieved in very large-scale scenarios where processing all the images in a single batch is not practically feasible.

Finally, a shorter dataset with $N = 1000$ images is considered, in order to have a relatively small instance of the problem where full reconstruction in a single batch is still possible. The proposed algorithm selects a subset of 786 views (21.4% less than N), divided into $C = 26$ clusters. Then, MVS is executed separately for each cluster and results are combined into a global 3D model. Figure 2.25 shows a comparison of the point clouds obtained by considering the whole set of images at once (left) and by merging multiple clusters (right). From a quantitative point of view, the full point cloud contains 3.08×10^6 points, which is approximately 7.8% more than the 2.84×10^6 points belonging to the result of the presented framework. This variation is lower than the difference between the input data size, and it is mostly concentrated in ambiguous regions, such as the rotary center with identical trees, where MVS benefits from higher redundancy. Furthermore, increasing the N_{min} parameter always reduces this gap. While ground truth for numerical evaluation in 3D is not available, it can be seen that our divide and conquer approach maintains a good reconstruction quality, while being able to scale up to entire cities, where batch reconstruction is not an option.

Figure 2.24: Clusters of images (left) and corresponding 3D point cloud (right) [43].

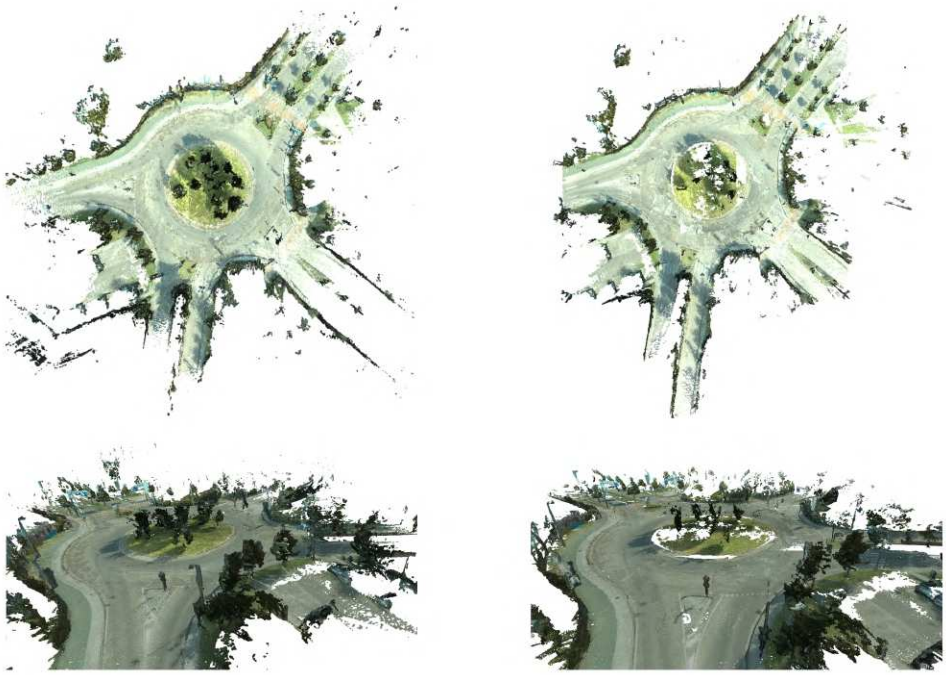


Figure 2.25: Qualitative point cloud comparison between batch reconstruction (left) and merged local clusters computed with the proposed algorithm (right) [43].

2.5 Conclusion

In this chapter, we presented several key improvements towards a scalable and efficient multi-view 3D reconstruction system for autonomous vehicles. Specifically, our contribution to the classical geometry-based pipeline described in Section 2.2 is twofold. Firstly, we revisit the dense reconstruction phase by (i) augmenting the MVS initialization step with keypoints from visual SLAM, (ii) optimizing a novel geometric consistency loss, and (iii) regularizing geometry with a confidence-based global refinement approach (Section 2.3). Secondly, we enable city-scale 3D reconstruction with arbitrary camera configurations in Section 2.4. This is achieved by a view clustering algorithm that builds a set of partially overlapping clusters with shared visibility over the vehicle trajectory, followed by a view selection step that computes the optimal subset of views to reconstruct a local 3D model. The novel contributions are evaluated both on the KITTI dataset and on custom sequences acquired by a real self-driving car, showing better performances compared to state-of-the-art methods.

3

Novel View Synthesis

3.1 Introduction

This chapter focuses on the *novel view synthesis* (NVS) task, where the goal is to render high-quality photorealistic images of a scene from arbitrary viewpoints, given a set of input views. The typical setup is shown in Figure 3.1: an intermediate 3D representation is fit to training data, and it can be queried at test time to generate novel views. At a high level, this problem is similar to multi-view stereo, as it maps calibrated 2D images to their underlying 3D structure. However, NVS algorithms do not compute the explicit geometry of the scene as a mesh or a point cloud, but they aim for a representation that can be rendered effectively. In recent years, this field has been revolutionized by *neural radiance fields* (NeRF) [11], which combine differentiable volumetric rendering with a neural implicit representation to learn both the geometry and the appearance of a 3D scene. A detailed overview of NeRF is provided in Section 3.2, which serves as a theoretical foundation for our original contributions [18, 19]. Then, we present *KeyNeRF* [18] in Section 3.3, a novel approach to speed up NeRF training by focusing on the most informative camera rays and a sparse set of input views. Finally, in Section 3.4 we propose *MVGNeRF* [19]: multi-view geometry constraints can be explicitly enforced when fitting a NeRF, by relying on the 3D reconstruction algorithms from Chapter 2, in order to produce high-fidelity 3D models, along with the renderable representation.

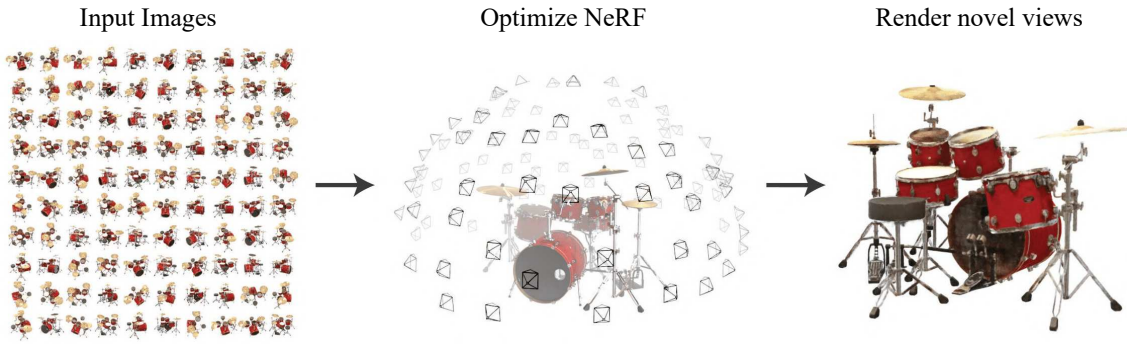


Figure 3.1: Novel view synthesis is the task of optimizing a 3D representation of the scene from a set of calibrated input images, such that other views can be rendered from arbitrary viewpoints [11].

3.2 Neural Radiance Fields

In this section, we provide a detailed literature review of neural radiance fields, covering both the basics in representing (Section 3.2.1) and rendering (Section 3.2.2) scenes with NeRF, how to train such *implicit* representation (Section 3.2.3) and how to extract an *explicit* geometry from it (Section 3.2.4) as well as more recent improvements to the original idea (Section 3.2.5). These concepts lay the foundations for our original contributions [18, 19], that will be presented in the remainder of the chapter.

3.2.1 3D Representation

The set of all visible things in the world can be modeled by the *plenoptic function* [127], which measures the radiance at any point $\mathbf{x} = (x, y, z) \in \mathbb{R}^3$, observed from the viewpoint $\mathbf{d} = (\varphi, \psi) \in \mathbb{S}^2$ at wavelength λ and time t. If we average over the wavelengths of the visible spectrum and assume a static scene, then the following mapping can be established:

$$(\mathbf{c}, \sigma) = f(\mathbf{x}, \mathbf{d}) \tag{3.1}$$

where $\mathbf{c}$ is the RGB color of the point $\mathbf{x}$ as seen from an optical ray with direction $\mathbf{d}$ and σ is its density. This function models a *radiance field*, and it is an implicit *volumetric* representation of a 3D scene, which is approximated as a cloud of tiny colored particles. The key intuition in NeRF [11] is to express the plenoptic function as a *neural network* with learnable parameters θ. In practice, such a network is a simple MLP that is trained to render a set of calibrated training views. Given input images $\mathcal{I}$ with calibration parameters $\mathcal{P}$ from SfM [51], optimal parameters

θ^* are found by minimizing a reconstruction loss $\mathcal{L}$ between the rendered and ground truth images at training viewpoints:

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\mathcal{I}, \mathcal{P}) \tag{3.2}$$

At test time, this 3D representation is then queried with an arbitrary set of camera rays to render images from novel viewpoints.

3.2.2 Volumetric Rendering

Let's consider a generic ray $\mathbf{r}$ that crosses the scene. For a given camera with origin $\mathbf{o} \in \mathbb{R}^3$ and intrinsics $\mathbf{K}$, the direction of a ray through the pixel (u, v) can be computed as follows:

$$\mathbf{d} = \begin{bmatrix} (u - c_u)/f \\ (v - c_v)/af \\ 1 \end{bmatrix} \tag{3.3}$$

Then, each point along such ray can be expressed as $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$. If at distance t the ray hits a particle, we return its color $\mathbf{c}(t)$. However, this notion is probabilistic, and we model the chance that $\mathbf{r}$ hits a particle in a small interval around t as $\sigma(t)dt$, where σ is the volume density. In order to determine the color corresponding to the ray, we also need to know if t is the first hit along the ray. The probability that the ray did not hit anything before t is called *transmittance* $T(t)$, and it can be expressed as a function of σ by exploiting the following physical relationship:

$$T(t + dt) = T(t) \times (1 - \sigma(t)dt) \tag{3.4}$$

Using the Taylor expansion for $T(t)$, Equation 3.4 becomes a differential equation that can be solved by integrating both sides:

$$\begin{aligned} T(t) + T'(t)dt = T(t) - T(t)\sigma(t)dt) &\implies \frac{T'(t)}{T(t)}dt = -\sigma(t)dt \\ &\implies \log T(t) = -\int_{t_0}^{t} \sigma(s)ds \\ &\implies T(t) = \exp\left(-\int_{t_0}^{t} \sigma(s)ds\right) \end{aligned} \tag{3.5}$$

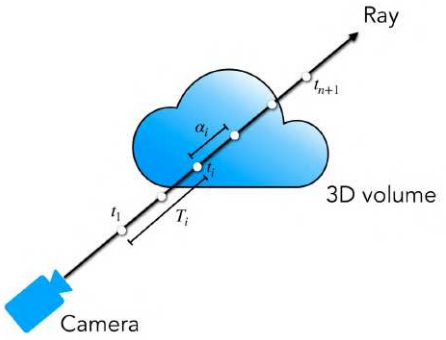


Figure 3.2: Visualization of the volumetric rendering model in NeRF [11] for a given ray through the scene.

In this way, the probability that a ray terminates at t can be expressed as a function of only the volume density σ as $T(t)\sigma(t)dt$. This means that the color corresponding to the ray $\mathbf{r}$ is the expected value computed across a given range of $t \in [t_n, t_f]$:

$$\mathbf{c}(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(t)\mathbf{c}(t)dt \tag{3.6}$$

This nested integral is solved in practice by approximating it with numerical quadrature [128]. The ray is split up into n segments with endpoints $\{t_0, t_1, \ldots, t_{n+1}\}$ and lengths $\delta_i = t_{i+1} - t_i$. By assuming that density and color are roughly constant within each interval, Equation 3.6 can be broken into a sum of analytically tractable integrals:

$$\int_{t_n}^{t_f} T(t)\sigma(t)\mathbf{c}(t)dt \approx \sum_{i=1}^{n} \int_{t_i}^{t_{i+1}} T(t)\sigma_i\mathbf{c}_i dt \tag{3.7}$$

Note that piecewise constant density and color do not imply constant transmittance. Therefore, we need to evaluate at continuous values that can lie partway through an interval. For any $t \in [t_i, t_{i+1}]$:

$$\begin{aligned} T(t) &= \exp\left(\int_{t_n}^{t_i} \sigma_i ds\right) \exp\left(\int_{t_i}^{t} \sigma_i ds\right) \\ &= \exp\left(-\sum_{j=1}^{i-1} \sigma_j\delta_j\right) \exp\left(-\sigma_i(t - t_i)\right) \\ &= T_i \exp\left(-\sigma_i(t - t_i)\right) \end{aligned} \tag{3.8}$$

The first term measures how much light is blocked by all previous segments, while the second term expresses how much light is blocked partway through the current segment. Substituting Equation 3.8 into Equation 3.7 and simplifying:

$$\begin{aligned}\int_{t_n}^{t_f} T(t)\sigma(t)\mathbf{c}(t)dt &\approx \sum_{i=1}^{n} T_i\sigma_i\mathbf{c}_i \int_{t_i}^{t_{i+1}} \exp\left(-\sigma_i(t-t_i)\right) dt \\ &= \sum_{i=1}^{n} T_i\sigma_i\mathbf{c}_i \frac{\exp\left(-\sigma_i(t-t_i)\right)-1}{-\sigma_i} \\ &= \sum_{i=1}^{n} T_i\mathbf{c}_i \exp\left(-\sigma_i(t-t_i)\right) \\ &= \sum_{i=1}^{n} T_i\alpha_i\mathbf{c}_i\end{aligned} \tag{3.9}$$

To sum up, T_i represents how much light is blocked earlier along the ray, while α_i measures how much light is contributed by the segment i, as shown in Figure 3.2. The rendering model for a generic ray $\mathbf{r}$ expressed by Equation 3.9 is trivially differentiable with respect to both $\mathbf{c}$ and σ, thus allowing for end-to-end training of NeRF.

3.2.3 Training Loop

Given a set of input images with calibration parameters, a NeRF representation can be fit to a specific scene by following the procedure in Algorithm 3.1, which is visualized in Figure 3.3. At each training iteration, a batch of B rays is shot from input cameras and the corresponding estimated color is rendered volumetrically with Equation 3.9. The ground truth color values corresponding to the rays are then used in a simple L_2 loss:

$$\mathcal{L} = \sum_{i=1}^{B} ||\hat{\mathbf{c}}(\mathbf{r}_i) - \mathbf{c}(\mathbf{r}_i)||^2 \tag{3.10}$$

After training, the same procedure can be used to generate novel views from arbitrary viewpoints. Camera rays can be shot through the scene and rendered to produce the full image. However, the NeRF formulation presented in Algorithm 3.1 is not sufficient for generating photorealistic renderings of complex, high-resolutions scenes. The seminal paper [11] describes three implementation details to achieve state-of-the-art quality.

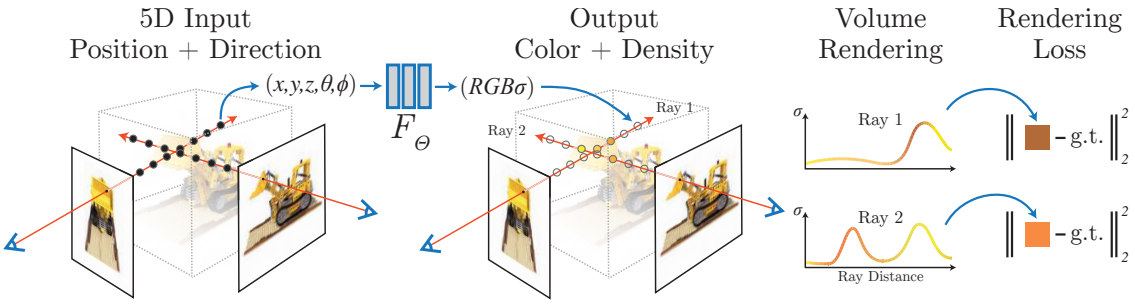


Figure 3.3: NeRF is trained by shooting rays through the scene, sampling a set of points, querying the MLP for color and density, and composing these values volumetrically to optimize a rendering loss [11].

Firstly, the density σ in Equation 3.1 is a function only of the physical point $\mathbf{x}$ and not of the viewing direction $\mathbf{d}$. This allows the MLP to predict multi-view consistent geometry, which does not depend on the camera observing the scene. Practically, the network is split in two stages: the first stage outputs σ and a high-dimensional feature vector $\mathbf{z}$, while the second stage predicts the color $\mathbf{c}$ as a function of $(\mathbf{z}, \mathbf{d})$.

Furthermore, neural networks are known to be biased towards learning low-frequency functions [129]. For novel view synthesis, this results in blurry renderings that fail to capture high-frequency variations in color and geometry. The issue can be solved by mapping the input coordinates to a higher dimensional space [130, 30]. For any $p \in \{x, y, z, \varphi, \psi\}$, the MLP is queried with $\gamma(p) : \mathbb{R} \to \mathbb{R}^{2L}$, which is defined as follows:

$$\gamma(p) = \begin{bmatrix} \sin(2^0 \pi p), \ldots, \sin(2^{L-1} \pi p) \\ \cos(2^0 \pi p), \ldots, \cos(2^{L-1} \pi p) \end{bmatrix} \tag{3.11}$$

Finally, densely evaluating the radiance field at all the query points along each ray is inefficient, since free space and occluded regions are sampled repeatedly, despite not contributing to the final rendering. To this end, the volume is represented hierarchically by optimizing a *coarse* and a *fine* network. The coarse network is evaluated by sampling N_c points uniformly along each ray. The rendering weight $w_i = T_i \alpha_i$ of each point is computed, and a piecewise constant probability distribution is defined by normalizing such weights for each ray. Then, a second set of N_f locations is sampled from this distribution with inverse transform sampling and the fine network is evaluated using all the $N_c + N_f$ points. In this way, more samples are allocated to regions containing more visible content.

Algorithm 3.1 Training Neural Radiance Fields

Input → Images $\mathcal{I} = \{I_i\}$ with calibration parameters $\mathcal{P} = \{(\mathbf{K}_i, \mathbf{R}_i, \mathbf{t}_i)\}$ from SfM
Initialize NeRF parameters θ randomly
for N_{iter} iterations **do**
    Sample a batch of B pixels from input cameras
    Compute rays from pixels (Equation 3.3)
    Sample N points along each ray
    Query the color and density of each point (Equation 3.1)
    Render volumetrically each ray (Equation 3.9)
    Compute the loss $\mathcal{L}$ between ground truth and rendered colors (Equation 3.10)
    Optimize NeRF parameters θ with gradient descent
end for
Output → Trained NeRF θ^* as implicit volumetric 3D representation

3.2.4 Geometry Extraction

The way NeRF represents a 3D scene is both *implicit*, meaning that the underlying geometry must be converted to a dense point cloud or a textured mesh for further processing, and *volumetric*, since there is not a well-defined surface being optimized. The density σ is a probabilistic concept, i.e. $\sigma \approx 0$ means that the point is likely to be empty and $\sigma \to 1$ indicates an occupied location. The typical pipeline to extract an explicit geometry from NeRF is to query the MLP with a dense grid of points and threshold the density at a given value (usually $\sigma = 0.5$). In this way, the 3D scene is represented as a discrete grid of voxels with binary occupancy and the marching cubes [112] algorithm can be used to extract a triangular mesh.

Another approach is to define a smooth camera trajectory across the scene and to render a depth map from a dense set of cameras along this path. Then, depth maps can be back-projected in 3D by following the multi-view fusion procedure in Section 2.2 to obtain a dense point cloud. The expected depth value of a camera ray can be rendered volumetrically by substituting the color $\mathbf{c}$ with the ray termination t in Equation 3.9:

$$d(\mathbf{r}) = \sum_{i=1}^{n} T_i \alpha_i t_i \tag{3.12}$$

This procedure is more popular when computing the geometry of large-scale scenes, while the former approach is the default choice in object reconstruction and bounded scenarios.

Recently, several works [131, 132, 133] have concurrently proposed to predict a signed distance function instead of a volume density, in order to represent a well-defined neural surface.

Figure 3.4: Images acquired from tourists and uploaded to social media [134] exhibit significant appearance changes. Training a NeRF on these pictures requires optimizing a per-image embedding vector [135].

The main advantage is to avoid specifying an arbitrary threshold for σ when converting it to binary occupancy for marching cubes. However, these methods typically trade a smoother geometry for lower quality renderings, thus making them more appealing for 3D reconstruction than for novel view synthesis.

3.2.5 Recent Advances and Applications

The original NeRF formulation [11] has been a breakthrough in vision and graphics for allowing self-supervised 3D scene representation at unprecedented quality. However, it is also slow to render and sensitive to the accuracy of input poses, it relies on the assumption of a static scene with constant lighting and fixed exposure between frames, it does not generalize across scenes, and it assumes dense sampling of cameras in a bounded region of the world. To this date, the NeRF literature has flourished with methods that tackle each of these problems.

The speed of training and inference has been greatly improved by *hybrid scene representations* [136, 137, 138], where a set of scene features is jointly learned with the network parameters and fed into the MLP to produce color and density. The key idea is to store part of the scene information into a hierarchical voxel grid of features that can be queried continuously with trilinear interpolation. In this way, a much smaller MLP is sufficient to decode such information and the whole pipeline in Algorithm 3.1 becomes significantly faster.

When dealing with unconstrained photo collections, such as images scraped online from social media, the same physical location might be affected by multiple appearance changes due to different exposures, lighting changes between day and night, or different weather conditions (see Figure 3.4). To adapt NeRF to these photometric variations, Martin-Brualla *et al.* [135] pioneered the use of *generative latent optimization* [139], in which each image is assigned a latent embedding vector to model image-dependent appearance. This vector is trained jointly

with the MLP parameters, thus allowing for smooth interpolation of appearances at test time.

Another issue in practical scenarios is the accuracy of calibration parameters, since noise and inconsistencies can degrade the visual quality of novel renderings. Some early methods [140, 141, 142] proposed to learn camera poses and intrinsics from scratch, which is an ambiguous problem and hard to initialize correctly. A more recent approach [143, 144] is to refine the camera poses by backpropagating the loss gradients to the parameters estimated from SfM [51]. Following this idea for NeRF-based pose estimation, several neural implicit SLAM systems have been developed as well [145, 146, 147].

All these improvements with respect to the baseline presented in [11] contributed to make NeRF a widespread and mature approach for 3D perception tasks. Its applications nowadays include urban simulation [143, 148], human modeling [149, 150, 151, 152], semantic scene decomposition [153, 154, 155], image super-resolution [156] and visual effects for advertising [157]. We refer the interested reader to a recent survey [158] for more details, while a thorough review of NeRF with sparse input views and explicit geometry as supervision will be presented in Section 3.3 and Section 3.4, respectively.

3.3 Informative Rays Selection for Few-Shot NeRF

This section is based on the original contributions currently under review at [18]:
Orsingher M., Dell'Eva A., Zani P., Medici P., and Bertozzi M., *Informative Rays Selection for Few-Shot Neural Radiance Fields*, International Conference on Computer Vision Theory and Applications (VISAPP), 2024.

3.3.1 Motivation

The lengthy per-scene optimization of NeRF is one of the main limits towards its practical usage, especially in resource-constrained settings. The hybrid representations [136, 137, 138] presented in Section 3.2 tackle this issue by reducing the size of the MLP, which makes it faster to be queried. However, the long training and inference times stem also from the fact that each pixel of each input view must be seen repeatedly until convergence. Given N cameras with M pixels each, an epoch needs $\frac{NM}{B}$ iterations for batch size B and Algorithm 3.1 requires multiple epochs. In this section, we present *KeyNeRF*, a simple yet effective method for training NeRF by focusing on the most informative cameras and pixels, thus reducing both N and M.

Existing few-shot approaches all assume to be given a random set of viewpoints, without control on how such cameras are selected, and propose to regularize the volumetric density learned by NeRF with new loss functions [159, 160] and additional inputs [161, 162, 163, 164, 165], thus introducing complexity in the pipeline. However, in common use cases, such as object scanning from videos acquired by a user with handheld devices, the input data consist of a dense and redundant set of frames with a known acquisition trajectory. Our insight is to better exploit such information in the input views.

Firstly, we select the best input views by finding a minimal set of cameras that ensure scene coverage. Secondly, this initial set is augmented with a greedy algorithm that promotes baseline diversity. Finally, we choose the most informative pixels for each view, in terms of their local entropy in the image. The proposed approach outperforms state-of-the-art methods on standard benchmarks in the considered scenario, while not requiring additional inputs and complex loss functions. Therefore, our contribution to existing literature is threefold:

1. We present a view selection algorithm that starts from the minimal set of cameras covering the scene and iteratively adds the next best view in a greedy way.
2. We propose to sample pixels in a given camera plane by following a probability distribution induced by the local entropy of the image.
3. To the best of our knowledge, our framework is the first few-shot NeRF approach that operates at input level, without requiring additional data or regularization losses.

3.3.2 Related Work

The original formulation of NeRF [11] requires a large set of cameras to converge, thus leading to long training times. For this reason, several methods [165, 159, 166, 164, 161, 162] have been proposed to allow learning radiance fields from few sparse views. All these approaches introduce new loss functions to regularize the underlying volumetric representation. However, such losses might be difficult to balance, and they are in contrast with one of the main advantages of NeRF, which can be trained in a self-supervised way from images with a simple rendering loss. On the other hand, our rays selection procedure is extremely flexible, as it operates directly at input level, and it can be implemented by only changing two lines of any existing NeRF codebase. This means it can be seamlessly integrated with other fast NeRF approaches, since it is orthogonal to improvements in loss functions or field representations [136, 137, 138].

Moreover, most of these works further assume to have additional inputs, such as depth measurements [162, 163] or other pre-trained networks [161, 165, 164]. Specifically, DS-NeRF [162] and DDP-NeRF [163] require sparse depth measurements to guide sampling along each ray and optimize rendered depth. DietNeRF [161] enforces high-level semantic consistency between novel view renderings with pre-trained CLIP embeddings [167], while RegNeRF [165] and DiffusioNeRF [164] maximize the likelihood of a rendered patch according to a given normalizing flow or diffusion model, respectively.

Another shortcoming of the aforementioned few-shot approaches is that they treat all pixels equally for a given input image and sample a random batch at each iteration. Other methods [145, 168] pioneered the use of uncertainty-based sampling of rays and estimate such uncertainty online, which leads to a computational overhead. On the other hand, we propose to compute the local entropy of the image offline a single time and to draw pixels from such distribution, which represents by definition the most informative rays.

3.3.3 Method

We present a framework, based on NeRF [11], for novel view synthesis from a given set of calibrated cameras. We assume to have a dense and redundant set of views, such as the frames of a video acquired by a user for object scanning. Our method, named KeyNeRF, identifies the key information in the given set of views by greedily selecting a subset of relevant cameras and choosing the most informative pixels within such cameras with entropy-based sampling. The proposed approach significantly improves the efficiency of NeRF, while requiring minimal code changes to its implementation. The first step of batch sampling in Algorithm 3.1 is typically implemented with NumPy [169] as follows:

```
pose_idx = np.random.choice(num_poses)
rays_idxs = np.random.choice(num_rays, size = B, p = None)
```

Note that both the camera and the rays to be optimized in the current iteration are drawn uniformly. In KeyNeRF, we simply reduce the set of input views and change the probablity distribution for sampling pixels:

```
pose_idx = np.random.choice(selected_cams)
rays_idxs = np.random.choice(num_rays, size = B, p = entropy)
```

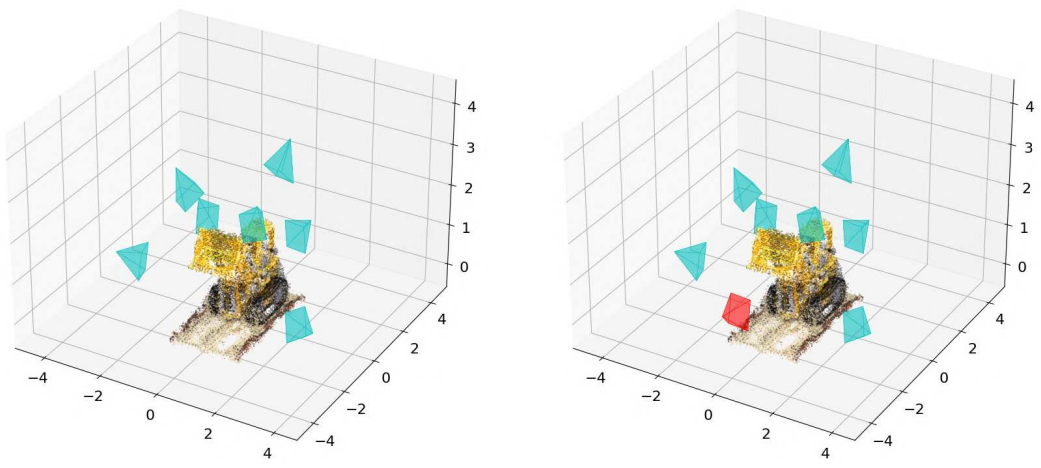


Figure 3.5: Illustration of the view selection procedure [18]. The new camera (red, right) has the most diverse baseline with respect to the set of current cameras (blue, left). A proxy geometry of the scene is shown for reference.

View Selection Algorithm

The goal of a view selection procedure is to sample K views from a dense set of N available cameras ($K \ll N$) for efficient 3D scene representation, while (i) maintaining the visibility of the whole scene and (ii) ensuring diversity within the selected subset. Following Section 2.4 [43], we propose to satisfy the first constraint by solving a simple optimization problem to find the minimal set of cameras that guarantee scene coverage. Then, a greedy algorithm iteratively adds the camera with the most diverse baseline, until all the views have been scheduled.

Scene Coverage Constraint In the first phase, cameras are represented by binary variables $x_i \in \{0, 1\}$ as in [43], and the scene is approximated with a uniform 3D grid of M points within bounds $(\mathbf{p}_{min}, \mathbf{p}_{max})$. We do not assume to have sparse keypoints, but they can be added to the scene approximation if available. Let $\mathbf{A}_j \in \mathbb{R}^N$ be a visibility vector with elements $a_{ij} = 1$ if point j is visible in camera i, 0 otherwise. The following ILP problem is then formulated:

$$\begin{aligned} \min \quad & \sum_{i=1}^{N} x_i \\ \text{s.t.} \quad & \mathbf{A}_j^\top \mathbf{x} > 0 \quad \forall j = 1, \dots, M \end{aligned} \tag{3.13}$$

The set of cameras selected in this way ensures scene visibility, but some regions of interest might not be fully covered with sufficient baseline for 3D understanding (see Figure 3.5, left).

Baseline Diversity Constraint In order to promote baseline diversity, we design a greedy view selection algorithm to choose the next best camera among the available ones with respect to the currently selected set. We generate a $N \times N$ symmetric baseline matrix $\mathbf{B}$, where $b_{ij} = b_{ji}$ is the angle between the optical axes of cameras i and j:

$$b_{ij} = \arccos\left(\frac{\mathbf{z}_i^\top \mathbf{z}_j}{|\mathbf{z}_i| \cdot |\mathbf{z}_j|}\right) \tag{3.14}$$

Then, at each iteration step, until the desired number of cameras has been reached, we add to the selected subset the camera with the highest relative angle with respect to *all* currently selected cameras, as shown in Figure 3.5 (right). Practically, for each remaining view, we query from $\mathbf{B}$ its smallest score against the selected views and add to the subset the camera with the *highest* smallest score. Assume NumPy [169] imported as `np` and let `selected_cams` be the output of the first stage:

```
while remaining_cams:
    sub_matrix = B[remaining_cams][:, selected_cams]
    idx = np.argmax(np.min(sub_matrix, axis = 1))
    selected_cams.append(remaining_cams[idx])
    remaining_cams.remove(remaining_cams[idx])
```

This formulation is different from [43], where matchability is a hard constraint. Moreover, the iterative nature of the greedy procedure induces an implicit ranking on the set of cameras and allows the user to choose flexibly the desired K. This is a significant improvement with respect to [43], where baseline diversity is not explicitly enforced, and a different optimization problem must be solved from scratch for different values of K. We will show in the experiments that any $K \geq K_{min}$ leads to good results, where K_{min} is the cardinality of the minimal scene coverage set. Intuitively, more views progressively improve the performances, with diminishing returns towards the end, when cameras have large overlaps with the current set and do not add relevant information.

Entropy-Based Rays Sampling

At each training iteration, NeRF [11] samples a pose in the dataset and a batch of B pixels from such camera. Typically, rays are sampled with uniform probability from the whole set of $H \times W$ available pixels. However, we observe that not all rays are equally informative about the scene to reconstruct. For example, the background or large textureless regions in the image could

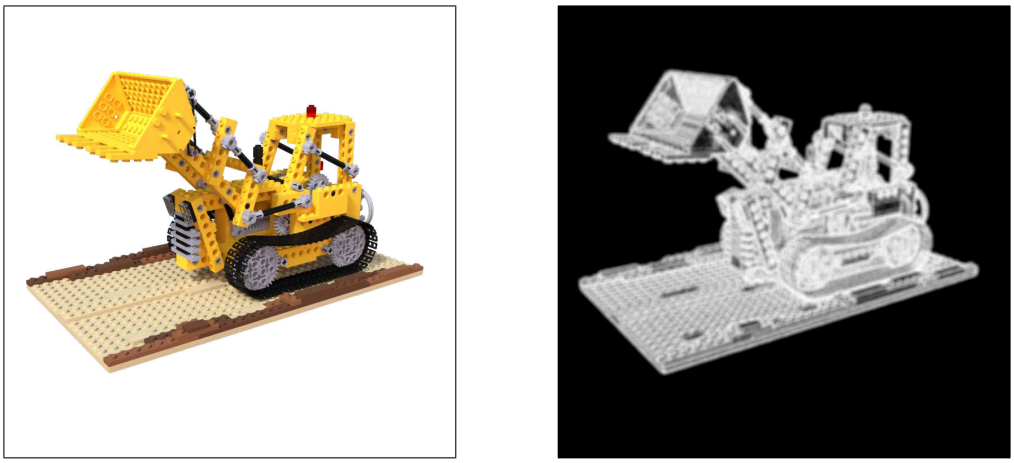


Figure 3.6: Probability distribution over pixels (right) for an example input image (left), induced by the local entropy of the image when sampling a batch of rays for training NeRF [18].

be sufficiently covered with fewer samples, exploiting the implicit smoothing bias of MLPs [170, 129]. We propose to define a probability distribution over pixels and to focus on high-frequency details during training, in order to converge faster, especially in few-shot scenarios. The amount of information of a pixel p can be quantified by computing its local entropy as:

$$e(p) = -\sum_{(u,v)\in\mathcal{W}} h_{uv} \log h_{uv} \tag{3.15}$$

where $\mathcal{W}$ defines a local window the pixel and h is the normalized histogram count. In order to allow random sampling based on such measure, we normalize it to a probability distribution, which is then used as input to `np.random.choice()`. An example is shown in Figure 3.6.

3.3.4 Experiments

Implementation Details

Dataset We perform our experiments on two common benchmarks to validate KeyNeRF on both controlled and real-world scenarios. The Realistic Synthetic 360° dataset was introduced by NeRF [11] and it contains 8 scenes of different objects with diverse materials and complex illumination, rendered by Blender from 400 random viewpoints. Moreover, we randomly select a subset of 8 scenes from the CO3D dataset [171], which contains 18619 real object-centric videos from 50 different categories, acquired by handheld devices. Due to the unavailability of pre-trained checkpoints of baseline methods on such data, a complete evalua-

Category	Sequence ID	N° of images	Resolution (W × H)
Ball	123_14363_28981	202	1062 × 1889
Cake	374_42274_84517	202	1893 × 1064
Hydrant	167_18184_34441	202	703 × 1251
Pizza	586_87341_172687	102	1016 × 1807
Plant	247_26441_50907	198	1065 × 1895
Remote	195_20989_41543	201	679 × 1209
Teddybear	34_1479_4753	202	1066 × 1896
Toaster	416_57389_110765	202	712 × 1267

Table 3.1: Details of the selected scenes in the CO3D subset [171].

tion on this large-scale benchmark would require a huge amount of computational resources, well beyond what we can realistically access. To mitigate this issue, we perform experiments on a subset of 8 diverse scenes and present their details in Table 3.1. The scenes are randomly selected both in terms of object category and individual sequence within each category. The rationale behind the choice of 8 scenes is to establish a real-world equivalent of the synthetic dataset introduced in [11], with different objects, various aspect ratios and noisy camera poses from SfM [51].

PARAMETERS For a given batch size B, at each iteration, we sample $B/2$ rays from the entropy-based distribution and $B/2$ rays at random to ensure full coverage. All the methods are trained for $N_{iter} = 50000$ iterations with $K = 16$ poses. Note that this is a slightly different setup than the typical few-shot scenario, where $K \leq 8$ and training is much longer ($N_{iter} \geq 200000$). We argue that this setup is overlooked in the literature, despite having significant practical relevance. In the common case of object scanning from videos, it is reasonable to assume to have more than 8 frames and the actual goal is training efficiency. However, existing few-shot methods tend to saturate their contributions when $K > 8$ [159], as shown in Table 3.2, while our method shows improved results over a wide range of values of K (see Figure 3.7a). The influence of both the number of poses K and iterations N_{iter} is discussed and ablated in the remainder of this section.

SOFTWARE The training code is based on a reference PyTorch [172] version of NeRF [173]. For the rays selection procedure, we solve the ILP with the OR-Tools library [126] and compute the image entropy with the default implementation in scikit-image [174].

Method	PSNR ↑	LPIPS ↓	SSIM ↑	Avg. ↓
NeRF [11]	24.424	0.132	0.878	0.055
DietNeRF [161]	24.370	0.127	0.878	0.054
InfoNeRF [159]	24.950	0.117	0.884	0.050
KeyNeRF (w/o entropy)	<u>25.568</u>	<u>0.109</u>	<u>0.895</u>	<u>0.046</u>
KeyNeRF (ours)	**25.653**	**0.106**	**0.898**	**0.045**

Table 3.2: Quantitative results on the Blender dataset [11]. Best and second results are **bold** and <u>underlined</u>, respectively.

Method	PSNR ↑	LPIPS ↓	SSIM ↑	Avg. ↓
NeRF [11]	20.708	0.491	0.744	0.128
DietNeRF [161]	19.994	0.511	0.728	0.138
InfoNeRF [159]	20.143	0.576	0.714	0.143
KeyNeRF (w/o entropy)	<u>21.853</u>	<u>0.470</u>	<u>0.759</u>	<u>0.114</u>
KeyNeRF (ours)	**22.183**	**0.463**	**0.762**	**0.109**

Table 3.3: Quantitative results on the CO3D dataset [171]. Best and second results are **bold** and <u>underlined</u>, respectively.

Quantitative Results

Following standard practice [11, 161, 159, 175], we evaluate the proposed approach in terms of the image quality of novel views. For a rendered image I with ground truth values I_{gt}, such quality can be measured by three common metrics:

1. The *Peak Signal-to-Noise Ratio* (PSNR) measures pixelwise errors over all color channels:
$$\text{PSNR}(I, I_{gt}) = -10 \log_{10}(||I - I_{gt}||^2) \tag{3.16}$$

2. The *Structural Similarity Index Measure* (SSIM) [176] gives an estimate of the perceived change in structural information:
$$\text{SSIM}(I, I_{gt}) = \frac{(2\mu_I\mu_{I_{gt}} + C_I)(2\sigma_{I,I_{gt}} + C_{I_{gt}})}{(\mu_I^2 + \mu_{I_{gt}}^2 + C_I)(\sigma_I^2 + \sigma_{I_{gt}}^2 + C_{I_{gt}})} \tag{3.17}$$
where $C_I = (0.01 \cdot L)^2$ and $C_{I_{gt}} = (0.03 \cdot L)^2$, with L the dynamic range of the pixels (i.e. 255 for 8-bit integers). The local statistics (μ, σ) for both images are computed patch-wise and averaged over the entire image. We refer the interested reader to the original paper [176] for more details.

3. The *Learned Perceptual Image Patch Similarity* (LPIPS) [177] uses learned convolutional features and measures the similarity between the activations across L layers of a pre-trained network:

$$\text{LPIPS}(I, I_{gt}) = \sum_{l=1}^{L} ||\varphi_l(I), \varphi_l(I_{gt})||^2 \tag{3.18}$$

While the seminal work [177] presents a detailed comparison between different architectures, we follow [11] and use AlexNet [29] to extract features.

Moreover, as introduced in [175], we report the geometric mean of LPIPS, $\sqrt{1 - \text{SSIM}}$ and $10^{-PSNR/10}$ to combine them in a single metric for easier comparison (reported as Avg. in Table 3.2 and Table 3.3). We compare our KeyNeRF in two different versions (i.e. with and without entropy-based rays sampling) against the original NeRF [11] and two state-of-the-art few-shot methods [161, 159]. Table 3.2 and Table 3.3 show that both versions of KeyNeRF outperform existing approaches on synthetic and real-world data, respectively, while being much simpler to implement and more flexible to integrate with any NeRF backbone. Moreover, note that the concurrent few-shot approaches fall behind the vanilla NeRF on real-world data (see Table 3.3), thus highlighting the complexity of loss weighting in such methods beyond controlled scenarios. On the other hand, KeyNeRF consistently outperforms them, even without entropy-based rays sampling. For the sake of completeness, we also provide a detailed per-scene breakdown of the quantitative results on both datasets in Table 3.4 and Table 3.5.

Ablation Studies

In this section, we analyze the impact of the number of poses K and the number of training iterations N_{iter} on the image quality metrics, as well as the separate role of selecting views and selecting informative rays. We perform such ablations on the Blender dataset [11].

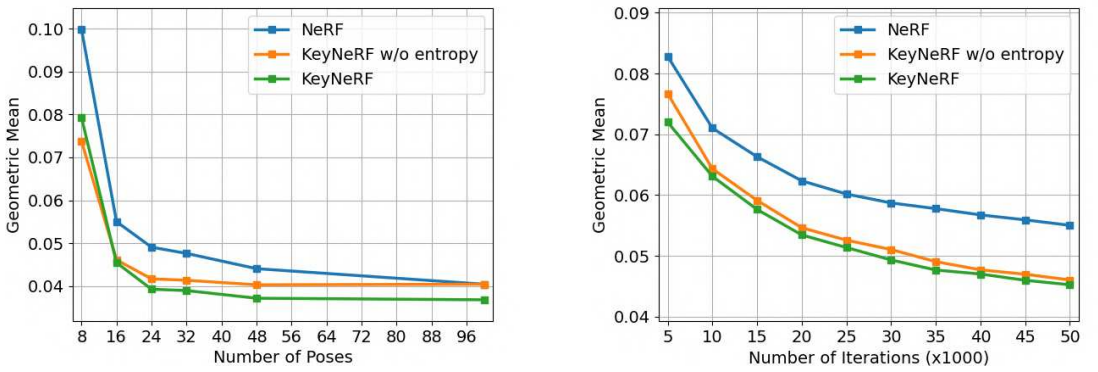
Figure 3.7a shows that the proposed view selection method has more influence for low values of K and progressively decreases, as expected. In order to clearly visualize this difference, we provide a qualitative comparison for $K = 8$ in Figure 3.8 and as a function of the number of poses K in Figure 3.9. It can be seen that the proposed approach converges faster and with better stability. Since the coverage constraint is satisfied optimally by the view selection algorithm, KeyNeRF allows to reconstruct an approximate scene even when $K = 8$. Crucially, our improvement is still significant up to $K = 48$, whereas concurrent few-shot methods only target the lowest end of this spectrum ($K \leq 8$).

Method	Chair	Drums	Lego	Mic	Materials	Ficus	Hotdog	Ship
PSNR ↑								
NeRF [11]	27.331	19.026	25.061	24.457	24.272	21.478	29.748	24.019
DietNeRF [161]	25.339	21.330	25.328	26.671	23.478	22.521	27.180	23.115
InfoNeRF [159]	27.167	21.271	24.522	27.761	24.245	23.170	28.176	23.292
KeyNeRF (w/o entropy)	28.045	**21.637**	27.245	**25.068**	**24.416**	22.060	30.881	25.193
KeyNeRF (ours)	**28.911**	21.545	**27.453**	24.030	24.186	**22.796**	**30.940**	**25.364**
LPIPS ↓								
NeRF [11]	0.099	0.236	0.087	0.118	0.084	0.121	0.071	0.242
DietNeRF [161]	0.141	0.135	0.100	0.073	0.098	0.091	0.117	0,263
InfoNeRF [159]	0.100	0.148	0.094	0.064	0.086	0.077	0.113	0.256
KeyNeRF (w/o entropy)	0.088	**0.129**	0.062	**0.116**	**0.079**	0.106	0.066	0.226
KeyNeRF (ours)	**0.077**	0.135	**0.057**	0.134	0.083	**0.083**	**0.065**	**0.220**
SSIM ↑								
NeRF [11]	0.910	0.814	0.881	0.924	0.894	0.870	0.945	0.792
DietNeRF [161]	0.881	0.867	0.867	0.943	0.885	0.889	0.921	0.772
InfoNeRF [159]	0.906	0.864	0.867	0.950	0.892	0.899	0.923	0.778
KeyNeRF (w/o entropy)	0.921	**0.874**	0.908	**0.927**	**0.898**	0.884	0.949	0.805
KeyNeRF (ours)	**0.929**	**0.874**	**0.912**	0.919	0.895	**0.899**	**0.951**	**0.808**

Table 3.4: Per-scene breakdown of the Blender dataset [11]. Best and second result are **bold** and underlined, respectively.

Method	Ball	Cake	Hydrant	Pizza	Plant	Remote	Teddybear	Toaster
PSNR ↑								
NeRF [11]	20.954	**20.312**	20.917	19.624	20.991	27.101	18.834	16.935
DietNeRF [161]	21.802	20.301	20.152	18.552	20.347	19.573	19.119	**20.108**
InfoNeRF [159]	23.826	20.042	18.078	18.189	21.159	25.449	16.776	17.625
KeyNeRF (w/o entropy)	24.073	19.278	21.016	19.565	**22.708**	28.511	20.697	18.977
KeyNeRF (ours)	**24.235**	19.565	**21.164**	**20.140**	22.326	**29.590**	**20.763**	19.682
LPIPS ↓								
NeRF [11]	0.472	0.635	0.527	0.442	0.641	0.197	0.469	0.549
DietNeRF [161]	0.479	0.639	0.570	0.457	0.616	0.346	0.481	0.507
InfoNeRF [159]	0.467	0.703	0.673	0.575	0.664	0.304	0.530	0.692
KeyNeRF (w/o entropy)	0.437	**0.603**	0.554	0.434	0.610	0.187	**0.416**	0.525
KeyNeRF (ours)	**0.421**	0.624	**0.516**	**0.428**	**0.603**	**0.177**	0.438	**0.502**
SSIM ↑								
NeRF [11]	0.753	0.722	0.605	0.765	0.651	0.929	0.797	0.731
DietNeRF [161]	0.762	0.698	0.604	0.754	0.648	0.847	0.783	0.733
InfoNeRF [159]	0.775	0.710	0.560	0.700	0.652	0.887	0.765	0.665
KeyNeRF (w/o entropy)	0.771	**0.746**	0.627	0.765	**0.678**	**0.961**	0.808	0.723
KeyNeRF (ours)	**0.784**	0.722	**0.634**	**0.773**	0.673	0.944	**0.816**	**0.756**

Table 3.5: Per-scene breakdown of the CO3D dataset [171]. Best and second results are **bold** and underlined, respectively.



(a) Ablation on the choice of K.

(b) Ablation on the choice of N_{iter}.

Figure 3.7: Quantitative comparison between our KeyNeRF and the original NeRF [11] as a function of the number of poses (left) and iterations (right). Lower is better.

Since we mainly focus on training efficiency, Figure 3.7b visualizes the convergence speed in steps of 5000 iterations each. Both versions of KeyNeRF show significant improvements across the whole training runs. Moreover, note how entropy-based sampling of rays is more effective in early iterations and then saturates after around 30000 steps. This confirms that selecting the most informative rays is important, especially with a limited training budget. The quantitative results in Table 3.2 and Table 3.3 underestimate the effect of this component. The lower quantitative impact is due to the fact that entropy-based sampling is most effective in fine-grained details and intricate structures, which are not well captured by numerical metrics. This is shown in Figure 3.10: sampling pixels uniformly discards crucial information, which leads to oversampling textureless areas and undersampling image regions with a lot of details.

Qualitative Results

The performance improvement of the proposed KeyNeRF in terms of rendering quality is visualized in Figure 3.11 for the Blender dataset and in Figure 3.12 for the CO3D dataset. The qualitative comparison against state-of-the-art methods shows that our informative rays selection strategy allows to render novel views with better details, especially in intricate structures such as the bulldozer wheels or the ship mast in Figure 3.11, and less hallucinated geometries (e.g. the teddybear and the hydrant in Figure 3.12). Moreover, our outputs are less blurry and preserve better the original colors of the scene. Finally, these results confirm that both DietNeRF [161] and InfoNeRF [159] tend to saturate their improvements over the original NeRF [11] in the considered setup, while our approach presents significant advantages.

Figure 3.8: Qualitative comparison between choosing poses at random (top row) and using the proposed algorithm (bottom row) in a very few-shot setting ($K = 8$) [18].

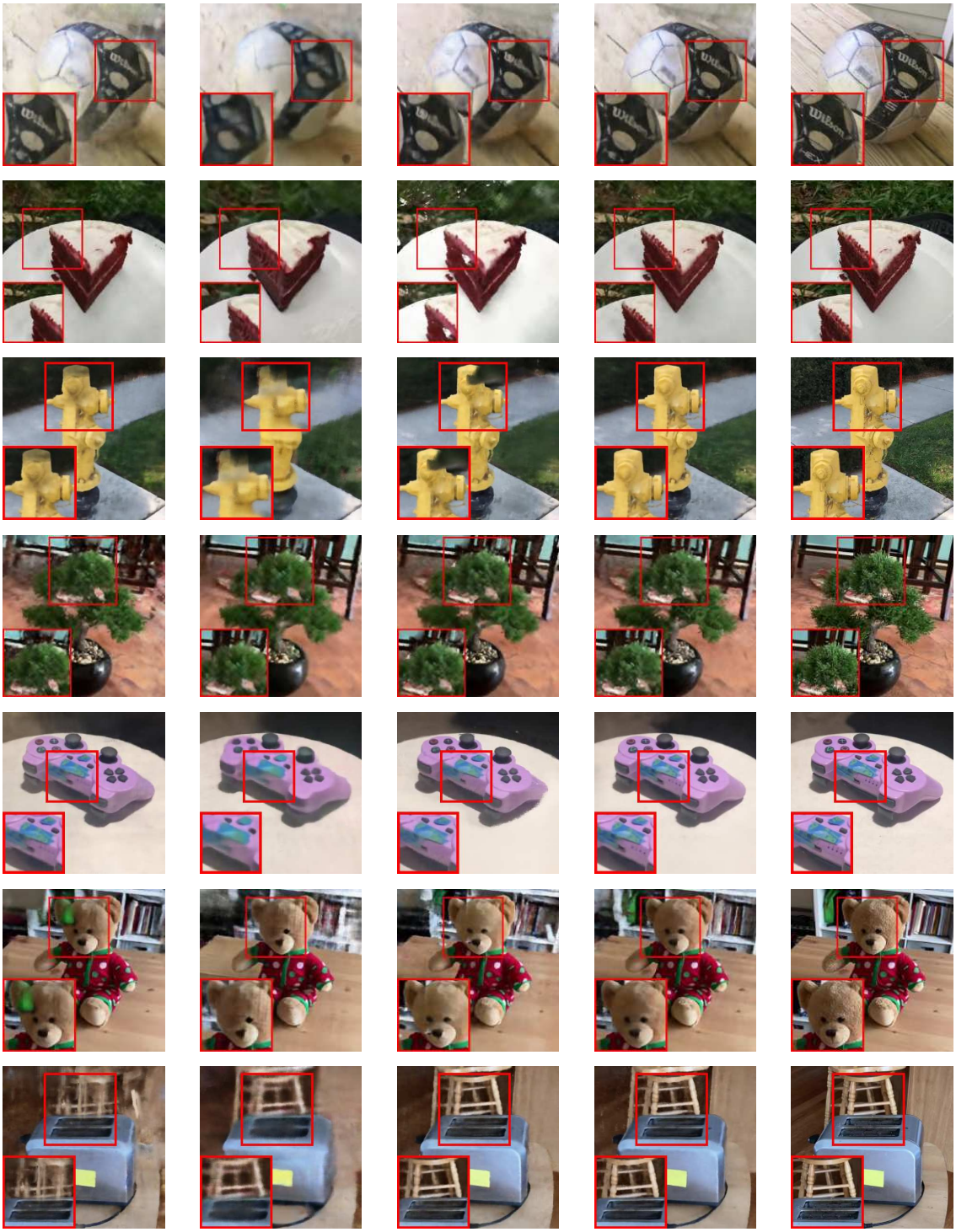
(a) $K = 8$ **(b)** $K = 16$ **(c)** $K = 24$ **(d)** $K = 32$ **(e)** $K = 48$

Figure 3.9: Qualitative comparison between choosing poses at random (top row) and using the proposed algorithm (bottom row), as a function of the number of poses K [18]. Zoom in for a better view.

Figure 3.10: Qualitative comparison between sampling rays at random (top row) and using entropy-based sampling (bottom row) for different frames of the same scene [18]. Zoom in for a better view.

(a) NeRF [11] **(b)** DietNeRF [161] **(c)** InfoNeRF [159] **(d)** Ours **(e)** Ground Truth

Figure 3.11: Qualitative results on the Blender dataset [11]. Zoom in for a better view [18].



(a) NeRF [11] **(b)** DietNeRF [161] **(c)** InfoNeRF [159] **(d)** Ours **(e)** Ground Truth

Figure 3.12: Qualitative results on the CO3D dataset [171]. Zoom in for a better view [18].

3.4 Learning NeRF from Multi-View Geometry

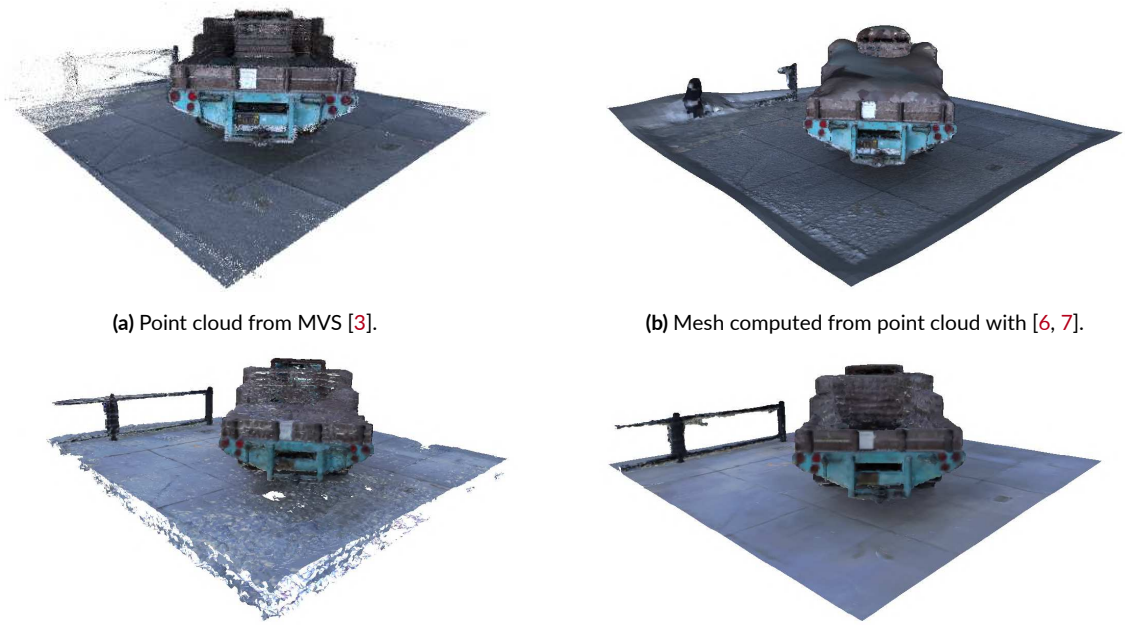
This section is based on the original contributions published in [19]:
Orsingher M., Zani P., Medici P., and Bertozzi M., *Learning Neural Radiance Fields from Multi-View Geometry*, Learning to Generate 3D Shapes and Scenes Workshop at European Conference on Computer Vision (ECCV), 2022.

3.4.1 Motivation

One of the main issues of the NeRF formulation presented in [11] is that the underlying geometry of the scene is not explicitly constrained during training. The only inductive bias towards learning a multi-view consistent representation in Algorithm 3.1 is the dependency of the volume density σ only on the physical point $\mathbf{x}$, but the actual optimization objective is a L_2 loss on the rendered *appearance*. The ability of NeRF to model view-dependent colors leads to an inherent ambiguity between the 3D shape of the scene and its radiance. Without regularization or explicit geometric constraints, it has been shown that this can lead to degenerate solutions where NeRF perfectly explains the training images, but generalizes poorly to novel test views [178]. Furthermore, even when the shape-radiance ambiguity is resolved during training and novel view synthesis generates high-quality renderings, the triangular mesh extracted with marching cubes might be noisy and incorrect, as shown in Figure 3.13c.

In Chapter 2, we presented a classical geometry-based 3D reconstruction framework that computes a 3D model of the scene from a set of input images. The typical output of this pipeline is a *discrete* representation as a dense point cloud, that must be converted to a *continuous* mesh with surface reconstruction algorithms [6, 7], when the downstream task requires such conversion. While point clouds from MVS are generally very accurate, they are also locally sparse, meaning that some areas of the scene might be empty or contain very few points (see Figure 3.13a). This usually happens in textureless regions and non-Lambertian surfaces, where MVS struggles the most. We show in Figure 3.13b that, in this case, the meshing procedure fails to generate the correct geometry and hallucinate incorrect shapes.

In this section, we propose to combine the best of both worlds and present a framework, called *MVGNeRF*, that allows to compute high-fidelity 3D meshes from images (see Figure 3.13d, while retaining the photorealistic novel view synthesis ability of NeRF. The key idea is to leverage pixelwise depths and normals from MVS as pseudo-ground truth for constraining the underlying geometry of NeRF during training. This supervision is softly activated only for rays



(a) Point cloud from MVS [3].

(b) Mesh computed from point cloud with [6, 7].

(c) Mesh extracted from NeRF [11].

(d) Mesh extracted from our framework [19].

Figure 3.13: In complex structures such as the back of the truck, classical algorithms [3, 6, 7] fail by hallucinating incorrect geometry, while the mesh from NeRF [11] is very noisy. Our approach combines them to get a clean 3D model [19].

with a high confidence value, which is computed based on the reprojection error. Differently from recent works that include depth priors in NeRF [162, 163, 179], we show that the joint optimization of normal vectors is crucial to improve the quality of the underlying surface. This is due to the fact that RGB and normals are complementary, meaning that normals can be estimated reliably in textureless regions where photometric consistency fails, while color supervision is effective in textured structures with ambiguous normals.

3.4.2 Related Work

Several works have explored the possibility of introducing geometric priors while optimizing NeRF. DS-NeRF [162] pioneered the idea of exploiting sparse keypoints from SfM and proposed to add a probabilistic depth supervision term in the loss function. Specifically, the depth along each keypoints' ray is modeled as a Gaussian distribution around its estimated 3D position. In this way, the KL divergence [180] between this normal distribution and the one computed from NeRF can be minimized. Following this insight, NerfingMVS [179] and DDP-NeRF [163] trained a monocular depth estimation and completion network, respectively, in order to have dense depth information for NeRF. These methods have shown to reduce both

the number of views required for convergence and the overall processing time. However, training a scene-specific network is a significant computational overhead, and they do not exploit the full geometry of the scene in terms of local surface normals.

Inspired by these approaches, we instead propose to leverage pixelwise depths and normals from a classical 3D reconstruction pipeline as *dense* and geometrically *accurate pseudo*-ground truth. MonoSDF [181] is a concurrent work that shows the importance of including normals as a geometric prior in the context of neural signed distance fields, but it relies on pretrained models for generating such priors. While these models [1, 182] are becoming readily available and generalize zero-shot to novel scenarios, they also output geometry without metric scale, which is recovered by fitting scale and shift parameters with least squares at test time.

3.4.3 Method

We present a framework that generates a triangular mesh of the scene from a set of input images $\mathcal{I}$, by combining classical geometry-based 3D reconstruction [51, 3] with NeRF [11]. The pipeline is visualized in Figure 3.14. Camera poses and calibration parameters $\mathcal{P}$ are computed with SfM [51] and used as additional input for both NeRF [11] and MVS [3]. Then, pixelwise depths $\mathcal{D}$ and normals $\mathcal{N}$ are computed for each image with MVS and used to supervise NeRF training with confidence weights $\mathcal{C}$. At the end of the training process, a 3D mesh is extracted from the density field with marching cubes [12].

Rendering Geometry from NeRF

The color rendering procedure in Equation 3.9 can be extended to render volumetrically any other quantity, including semantics [155], deep features [183] and the underlying geometry that is learned during training. We showed how to compute the expected depth along a ray in Equation 3.12. Similarly, the surface normal for the ray $\mathbf{r}$ can be rendered as follows:

$$\mathbf{n}(\mathbf{r}) = T_i \alpha_i \mathbf{n}_i \tag{3.19}$$

where $\mathbf{n}_i$ is the unit normal vector corresponding to the i-th sample along $\mathbf{r}$. For a given point $\mathbf{x} \in \mathbb{R}^3$ in a density field f_θ, such vector is given by the gradient of the field at that point:

$$\mathbf{n}_i = \frac{\nabla_{\mathbf{x}_i} f_\theta(\mathbf{x}_i)}{||\nabla_{\mathbf{x}_i} f_\theta(\mathbf{x}_i)||} \tag{3.20}$$

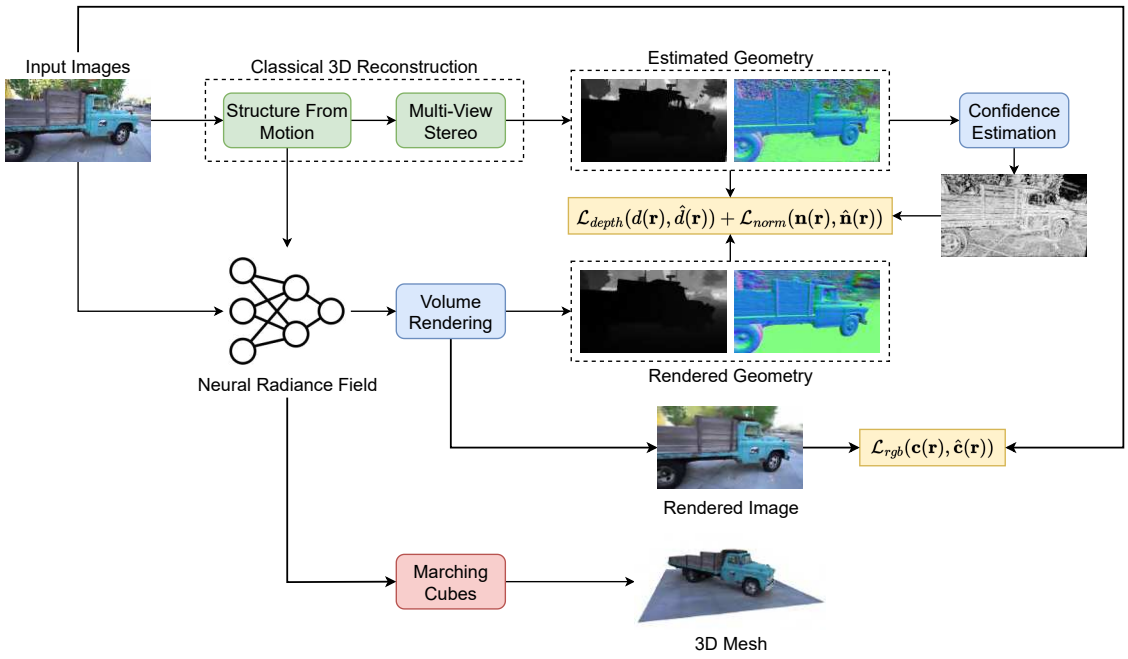


Figure 3.14: An overview of the proposed approach [19]: NeRF training is guided by a confidence-aware geometric *pseudo*-ground truth from classical 3D reconstruction. Pixelwise depths and normals are computed from NeRF [11] with volume rendering and optimized with novel geometry losses.

Modern deep learning frameworks [172] provide an automatic differentiation engine that can be queried to provide the precise gradient of the density field. However, we found in our experiments that a simple central difference approximation is more accurate and efficient. Let Δh be a small step size:

$$\nabla_{\mathbf{x}} f_\theta(\mathbf{x}) \approx \frac{f_\theta(\mathbf{x} + \Delta h) - f_\theta(\mathbf{x} - \Delta h)}{\Delta h} \tag{3.21}$$

Confidence-aware Geometric Loss Functions

At each training iteration in Algorithm 3.1, a random batch of rays $\mathcal{R}$ is sampled from the dataset and differentiable volumetric rendering is used to produce both colors and geometry by integrating along the ray. Then, we propose to optimize NeRF with the sum of three losses:

$$\mathcal{L} = \sum_{\mathbf{r} \in \mathcal{R}} \mathcal{L}_{rgb}(\mathbf{c}(\mathbf{r}), \hat{\mathbf{c}}(\mathbf{r})) + \lambda_{depth} \mathcal{L}_{depth}(d(\mathbf{r}), \hat{d}(\mathbf{r})) + \lambda_{norm} \mathcal{L}_{norm}(\mathbf{n}(\mathbf{r}), \hat{\mathbf{n}}(\mathbf{r})) \tag{3.22}$$

where λ_{depth} and λ_{norm} are weighting parameters to balance the contribution of geometry over appearance. Consistently with the original NeRF formulation [11], the first term is the standard L_2 loss on rendered colors, as in Equation 3.10. Moreover, we guide the optimization procedure by penalizing errors between the rendered geometry and the *pseudo*-ground truth. For both depths and normals, each ray is weighted by the corresponding confidence value to softly activate the supervision only in reliable pixels. We follow Section 2.3 and define a continuous confidence value with Equation 2.9. Then, both $\mathcal{L}_{depth}$ and $\mathcal{L}_{norm}$ are formulated as a Huber loss between the estimated quantity $\hat{x}$ and the reference x, weighted by confidence c:

$$\text{Huber}(x, \hat{x}, c) = \begin{cases} \frac{1}{2} c (x - \hat{x})^2 & \text{if} \quad |x - \hat{x}| < \delta \\ \delta \cdot c (|x - \hat{x}| - \frac{\delta}{2}) & \text{otherwise} \end{cases} \tag{3.23}$$

The choice of the Huber loss over a standard L_2 loss is an additional step towards a more robust optimization. This function is quadratic for small errors and linear for large errors, making it less sensitive to outliers.

3.4.4 Experiments

Implementation Details

Dataset We demonstrate the effectiveness of MVGNeRF on the *Truck* scene from the Tanks & Temples dataset [61], as a proxy for real-world outdoor data. The scene is captured by 250 full HD images with resolution 1920×1080, as well as a high-precision laser scanner for acquiring 3D ground truth. The training set for NeRF is built by randomly selecting 90% of the images, with the remaining 10% being the test set. Similarly, only the training set of NeRF is used as input for classical 3D reconstruction.

Parameters NeRF is optimized for $N_{iter} = 250000$ iterations and a random batch of $B = 1024$ rays is selected at each training step. For each ray, 64 and 128 points are sampled for the coarse and fine stage of hierarchical sampling, respectively. The radiance field is approximated by an 8-layer MLP with 256 neurons each, and the geometric losses are weighted with $\lambda_{depth} = \lambda_{norm} = 0.1$. Finally, during the mesh extraction phase, a uniform grid of 256^3 points is fed to the marching cubes algorithm [12] and the density is thresholded at $\tau = 50$ for generating the binary occupancy field.

Method	PSNR ↑	SSIM ↑	LPIPS ↓	CD $\times 10^{-3}$ ↓
NeRF [11]	**21.2384**	**0.6526**	**0.3819**	2.3823
NeRF [11] w/ depth	20.8911	0.6383	0.4318	<u>1.9701</u>
MVGNeRF (ours)	<u>21.0013</u>	<u>0.6468</u>	<u>0.3971</u>	**1.8865**

Table 3.6: Quantitative results of NeRF with different geometric supervisions. For each metric, best and second results are **bold** and <u>underlined</u>, respectively.

Software The framework has been tested on a single Nvidia V100 GPU with 32 GB RAM, but it can be adapted to run on lower tier devices with less memory. The calibration parameters for NeRF are computed with SfM [184], while the geometric *pseudo*-ground truth is estimated by the MVS algorithm presented in Section 2.3. As for KeyNeRF in Section 3.3, NeRF optimization follows an open-source PyTorch implementation [173], with custom modifications to support our confidence-aware geometric losses. After training, the mesh is extracted with a publicly available marching cubes algorithm [12] from the `PyMCubes` library.

Quantitative Results

In this section, we provide a numerical comparison of our approach against the baseline NeRF [11] and a hybrid version with only pixelwise depth as supervision. We measure the performances of both methods in terms of the resulting 3D geometry and novel view synthesis results. Consistently with existing literature, the image quality of novel renderings is measured with the PSNR, SSIM [176] and LPIPS [177] metrics, as introduced in Section 3.3. Moreover, the Chamfer distance between the point cloud from the laser scanner P_{gt} and the mesh vertices P after marching cubes is computed to quantify the geometric output:

$$\mathrm{CD}(P, P_{gt}) = \frac{1}{|P|} \sum_{x \in P} \min_{y \in P_{gt}} ||x - y||^2 + \frac{1}{|P_{gt}|} \sum_{y \in P_{gt}} \min_{x \in P} ||x - y||^2 \tag{3.24}$$

The quantitative results are shown in Table 3.6, where it can be seen that our approach provides a better 3D geometry, while remaining competitive on the novel view synthesis task. Moreover, note that the supervision of dense depth without normals already improves significantly the quality of the underlying shape, but adding information about the local surface normal leads to the best results. This is consistent with findings in [181] about neural surface reconstruction methods with geometric priors.

Qualitative Results

The improvement in 3D scene representation obtained by MVGNeRF is also shown in qualitative results. Despite the differences in perceptual metrics (see Table 3.6), the novel rendered views from our approach shown in Figure 3.15 match closely the output of basic NeRF. Moreover, Figure 3.16 and Figure 3.17 show the pixelwise depths and normals obtained after volumetric rendering, respectively. It can be clearly seen that the proposed approach produces much smoother results, especially in terms of normal vectors. When compared to the geometric priors used as *pseudo*-ground truth, the geometry rendered from MVGNeRF is consistent even in textureless regions, where the pixels have low confidence and the explicit supervision from MVS is not active. This consistency is confirmed also by the meshes visualized in Figure 3.18. We obtain the cleanest 3D model, without the noise of NeRF [11] and the hallucinated geometry of classical surface reconstruction [6, 7] after MVS.

3.5 Conclusion

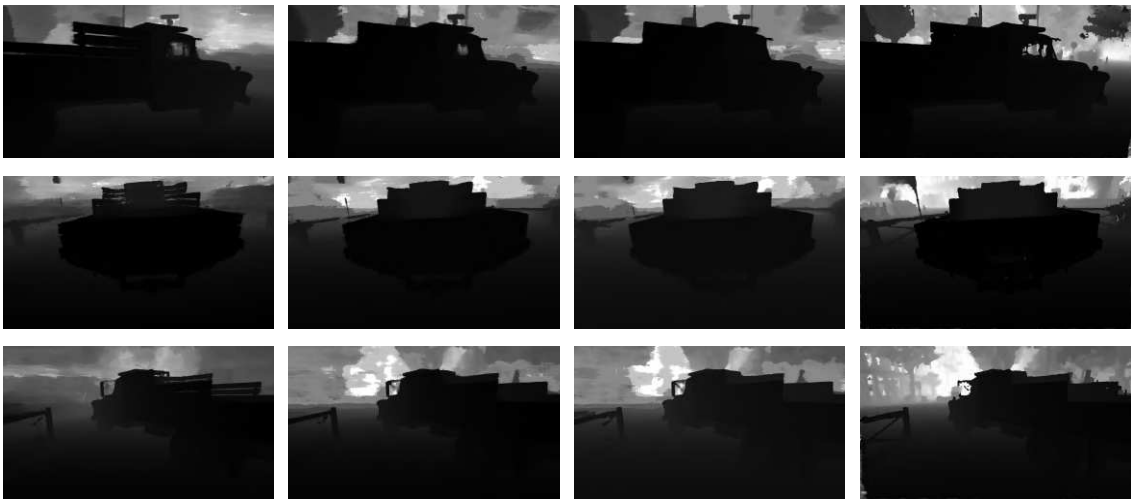
In this chapter, we introduced the task of novel view synthesis and provided a general overview of neural radiance fields in Section 3.2, a learning-based state-of-the-art solution to the problem. Then, we presented two original improvements to the basic formulation [11], in order to tackle some of its main issues: training efficiency, the requirement of a large set of densely sampled views as input, and the lack of multi-view geometry constraints during optimization.

Specifically, in Section 3.3 we proposed a novel method, called KeyNeRF [18], to select informative samples for training a few-shot NeRF. This allows Algorithm 3.1 to focus on relevant information in early iterations, thus significantly speeding up convergence with a limited training budget. Moreover, MVGNeRF [19] was presented in Section 3.4 as a framework that effectively supervises NeRF geometry with classical 3D reconstruction during training, in order to generate cleaner and smoother 3D shapes. Both approaches were evaluated on synthetic and real-world data, showing state-of-the-art performances.

As future work, we plan to explore other ways to define a probability distribution on the image plane for sampling rays in KeyNeRF. Moreover, some recent works [185, 186] proposed to enforce multi-view geometry constraints by warping patches or deep features as an additional loss. This idea could be integrated in MVGNeRF, especially for pixels with low confidence. Finally, it is promising to combine both our contributions in a single strong baseline and to test it on driving data, in order to compare the results with the pipeline described in Chapter 2.

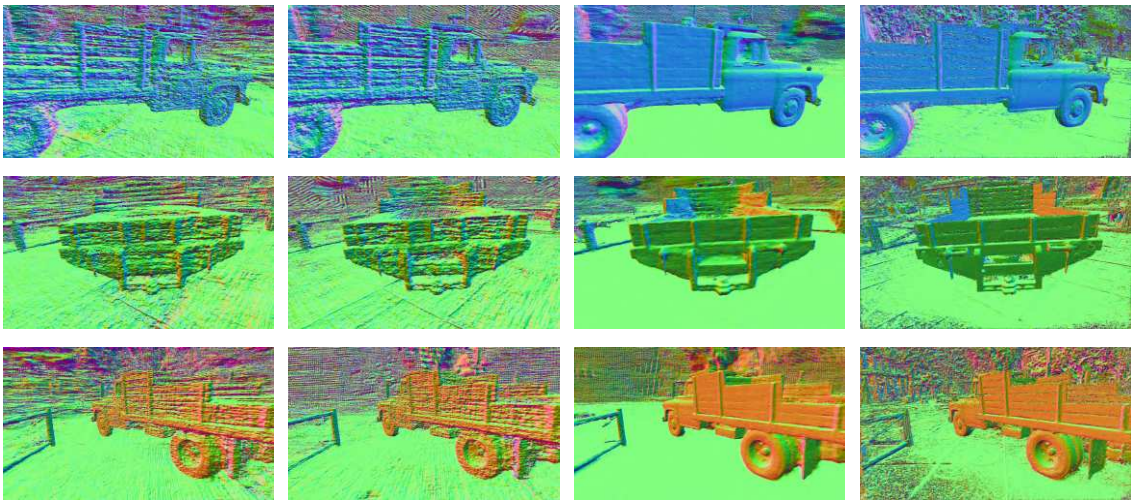
(a) NeRF [11] **(b)** NeRF [11] w/ depth **(c)** Ours **(d)** Ground Truth

Figure 3.15: Qualitative comparison when rendering colors from novel views [19].



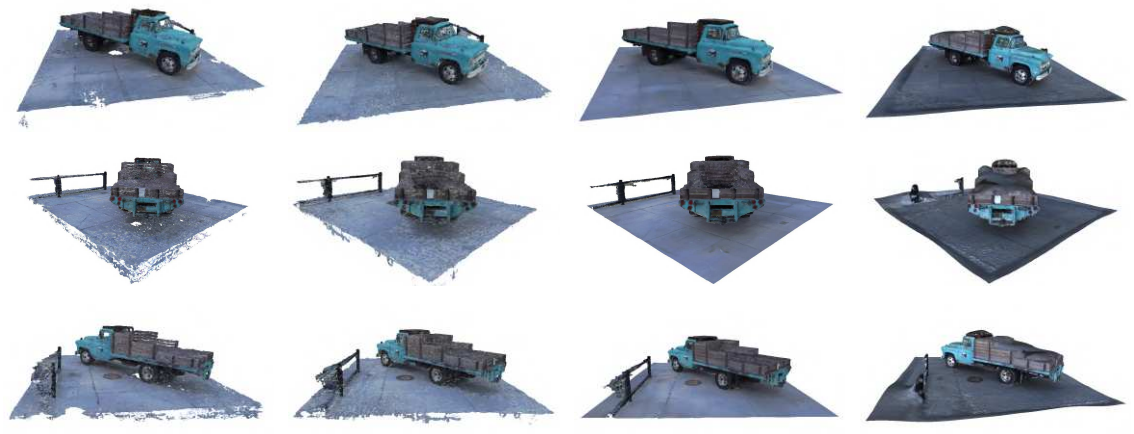
(a) NeRF [11] **(b)** NeRF [11] w/ depth **(c)** Ours **(d)** (Pseudo-)Ground Truth

Figure 3.16: Qualitative comparison when rendering depth from novel views [19].



(a) NeRF [11] **(b)** NeRF [11] w/ depth **(c)** Ours **(d)** (Pseudo-)Ground Truth

Figure 3.17: Qualitative comparison when rendering normals from novel views [19].



(a) NeRF [11] **(b)** NeRF [11] w/ depth **(c)** Ours **(d)** Mesh [6, 7] after MVS

Figure 3.18: Qualitative comparison of the resulting 3D mesh [19]. Our approach removes both the noise of NeRF and the hallucinated geometry of the classical pipeline.

4

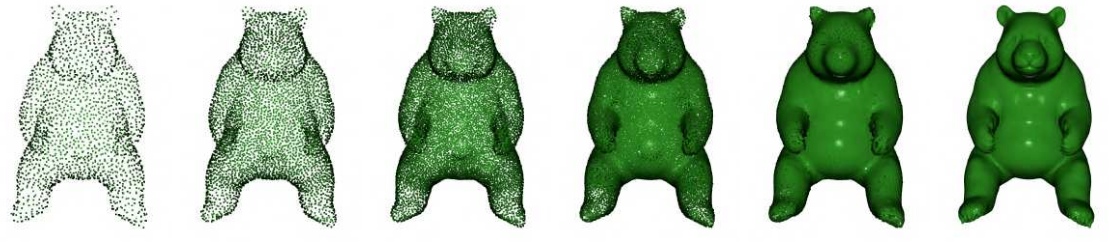
Point Cloud Upsampling

4.1 Introduction

This chapter is based on the original contributions published in [44]:
Dell'Eva A.*, **Orsingher M.***, and Bertozzi M., *Arbitrary Point Cloud Upsampling with Spherical Mixture of Gaussians*, International Conference on 3D Vision (3DV), 2022.
* Equal contribution.

Point clouds are a common way to represent 3D data as an unordered list of points, which can be thought of as discrete samples from the underlying object surface. In recent years, the wide availability of low-cost scanning sensors has driven the research towards 3D point clouds analysis for several applications such as augmented reality, robotics and autonomous driving [187, 188, 189, 190]. However, the sparsity and the noise level in raw data from such sensors pose key challenges in point cloud processing for downstream tasks, such as classification and segmentation. To this end, in this chapter we focus on point cloud upsampling, which consists in generating a dense and uniform set of points from a sparse and noisy input.

Building on pioneering works for neural point processing [15, 16], learning-based methods achieve state-of-the-art results in point cloud upsampling [191, 192, 193, 194] and outperform classical optimization-based techniques [195, 196, 197]. One of the main limitations of current approaches is the tight coupling between the upsampling ratio and the network architec-



(a) Input **(b)** $r = 2.34$ **(c)** $r = 4.93$ **(d)** $r = 9.51$ **(e)** $r = 22.86$ **(f)** Ground Truth

Figure 4.1: Our approach can upsample a sparse input with N points to a high-resolution output with $r \times N$ points. The upsampling ratio $r \in \mathbb{R}$ can be specified arbitrarily at test time, even if the model is trained only with $r = 4$ [44].

ture. Typically, this value must be specified in advance and different models must be re-trained from scratch for different rates. Despite recent efforts on designing flexible networks with a user-defined upsampling factor at test time, existing works still limit its value to be integer and lower than a given bound [198, 199, 200], or reconstruct the whole surface with ground truth normals as an intermediate step [201]. Our main goal is to remove these limitations and to enable arbitrary upsampling from raw point clouds with a unique model, trained a single time, as shown in Figure 4.1. The key intuition of the presented method is to split the upsampling procedure in two steps: (i) the input point cloud is firstly mapped to a probability distribution on a canonical domain, then (ii) an arbitrary number of points are sampled from such distribution and mapped back to the target surface. Inspired by recent works on point cloud autoencoders [202, 203, 204], we propose to use the unit sphere as intermediate representation, and we define a *Spherical Mixture of Gaussians* (SMOG) distribution on such domain. In this way, each input point is associated with a mixture weight and a bivariate Gaussian in spherical coordinates, whose parameters are estimated by a neural network. The inverse mapping from the samples on the unit sphere to the desired shape is implemented by querying the decoder of a Transformer model [30] with an arbitrary number of points, which effectively decouples the network architecture and the upsampling ratio. Therefore, our contributions are threefold:

- We present a novel approach for point cloud upsampling with arbitrary scaling factors, including non-integer values, with a single trained model.
- We propose to learn a mapping from the low-resolution input point cloud to a probability distribution on the unit sphere. This distribution can then be sampled arbitrarily, and the inverse mapping is learned to generate the high-resolution output.
- We design a network architecture fully based on local self-attention, which has proved to learn powerful representations on point cloud data [17, 205, 39].

4.2 Related Work

4.2.1 Canonical Primitives in Point Cloud Auto-Encoders

In the context of point cloud auto-encoders, a common procedure to generate the output is to feed the decoder with a global latent vector encoding the input and a canonical primitive (e.g. a 2D grid [206]), which is deformed to match the target surface. Following the insights in [202], several works use the unit sphere with *uniform* sampling as an intermediate representation [203, 204]. Recently, TearingNet [207] proposed to learn topology-friendly representations by additionally estimating pointwise offsets on the 2D domain. We take a step further by directly learning a mean vector on the unit sphere and 2D variances in spherical coordinates for each point in the input shape. This allows to define a distribution from which an arbitrary number of points are sampled and mapped back to the surface.

4.2.2 Learning-based Point Cloud Upsampling

Point cloud upsampling is an inherently ill-posed problem, since a finite number of samples correspond to many underlying surfaces and viceversa. For this reason, PU-Net [191] pioneered the idea of learning geometric priors from data and outperformed previous classical methods [197, 196, 195]. Building on this seminal idea of learning and expanding multi-scale point features, MPU [208] proposes a patch-based progressive strategy for upsampling at different levels of detail, while PU-GAN [192] casts the upsampling procedure in a generative adversarial framework. PU-GCN [193] introduces several modules built upon graph convolutional network that can be integrated into other architectures, whereas Dis-PU [194] disentangles the upsampling task into two cascaded subnetworks for dense point generation and spatial refinement. Despite showing promising results, all these works require a fixed upsampling ratio r and train different models for varying values of r. This strategy does not adapt to real-world point clouds with different quality and it increases the training time significantly.

4.2.3 Arbitrary Point Cloud Upsampling

Recently, a few works [199, 198, 200] emerged with the goal of decoupling the upsampling ratio and the network architecture, thus achieving flexible upsampling. Meta-PU [200] employs meta-learning to predict the weights of residual graph convolution blocks dynamically for different values of r. However, the model first generates a set of $r_{max} \times N$ points, which are then

downsampled to the desired ratio using *farthest point sampling* (FPS). MAFU [198] and PU-EVA [199] exploit the local geometry of the tangent plane at each point to sample a variable number of candidate points in its neighborhood, but they are limited to *integer* upsampling factors within a predefined range. In addition, the former requires normal vectors information to be trained. On the other hand, our approach is designed to support any value of $r \in \mathbb{R}$. Neural Points [201] is a concurrent work that performs upsampling with unconstrained ratios by first encoding the continuous underlying surface with neural fields and then sampling an arbitrary number of points from it. Their main limitation is the requirement of ground truth normals for training, which are difficult to estimate for real-world noisy inputs. Conversely, our model operates in a discrete-to-discrete way on raw point clouds without normals.

4.2.4 Transformers for Point Clouds

The Transformer model has revolutionized both natural language processing [30] and computer vision [38], thanks to the attention mechanism at its core. Since the Transformer architecture is permutation invariant by design and thus naturally suited for 3D data, early works in the field focused on adapting its modules to point cloud processing [39, 40, 209, 210]. Attention-based methods have achieved state-of-the-art results in many 3D tasks, such as point cloud completion [205], object detection [17] and classification [211]. In this work, we propose to combine the Transformer architecture with an attention-based refinement module for point cloud upsampling. To the best of our knowledge, there is only one concurrent work on this aspect [212]. However, they solely leverage the Transformer encoder, while we also exploit the possibility of querying the Transformer decoder for arbitrary upsampling.

4.3 Method

Denoting by $\mathcal{P} = \{\mathbf{p}_i \in \mathbb{R}^3\}_{i=1}^{N}$ the unordered sparse input point cloud of N 3D points, our objective is to generate an arbitrarily denser point set $\mathcal{Q}_r = \{\mathbf{q}_i \in \mathbb{R}^3\}_{i=1}^{M}$ with $M = r \times N$ points, where $r \in \mathbb{R}$ is the upsampling factor. Note that point cloud upsampling is an ill-posed task, since there is not a single feasible correct output. This means that $\mathcal{Q}$ should represent the same underlying surface $\mathcal{S}$, while not being necessarily a superset of $\mathcal{P}$. To this end, we design a fully attention-based end-to-end network for arbitrary point cloud upsampling, taking advantage of Gaussian mixture sampling and Transformer queries to enable flexible ratios. An overview of our framework is shown in Figure 4.2.

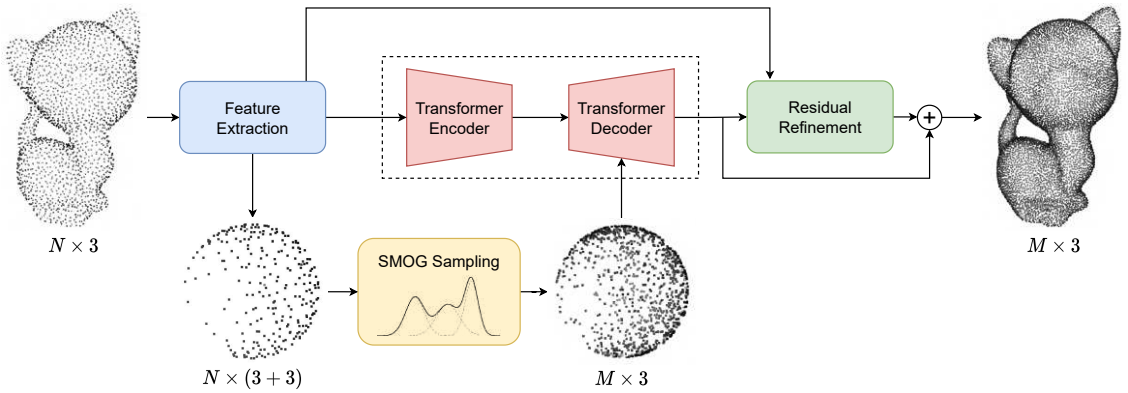


Figure 4.2: A high-level overview: the low-resolution input point cloud with size $N \times 3$ is firstly mapped to a probability distribution on the unit sphere, with a mean vector and covariance matrix associated to each point. Then, this distribution is sampled to produce the high-resolution output with size $M \times 3$, with $M = r \times N$ and r the desired upsampling ratio [44].

4.3.1 Network Architecture

Feature Extraction

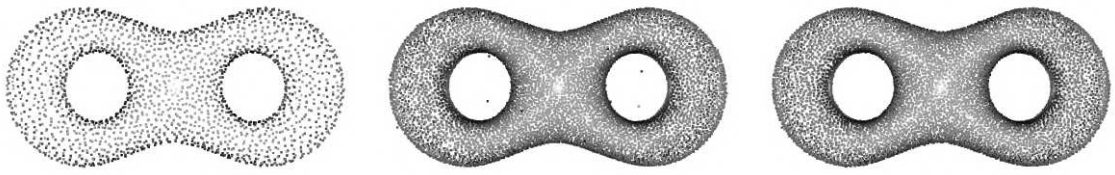
The first step of our pipeline is to extract point-wise features $\mathbf{f}_i \in \mathbb{R}^D$ from the input point cloud. Differently from previous methods [193, 191, 192] that employ either PointNet [15, 16] or DGCNN [213] as backbone, we design a lightweight network based on Point Transformer [39]. For each input point $\mathbf{p}_i \in \mathcal{P}$, its associated feature is computed as:

$$\mathbf{f}_i = \sum_{\mathbf{p}_j \in \mathcal{N}(\mathbf{p}_i)} \rho(\gamma(\beta(\mathbf{p}_i) - \psi(\mathbf{p}_j) + \delta)) \odot (\alpha(\mathbf{p}_j) + \delta) \tag{4.1}$$

where $\mathcal{N}(\mathbf{p}_i)$ is the set of k nearest neighbors of $\mathbf{p}_i$, $\delta = \eta(\mathbf{p}_i - \mathbf{p}_j)$ is the positional encoding and the symbol $\odot$ denotes the element-wise product. In this *Point Transformer Layer* (PTL), the mappings α, β and ψ are simple linear layers, γ and η are two-layers MLP, while ρ is the softmax function. In the context of the classical interpretation of the Transformer model [30], α generates the *values*, β the *queries* and ψ the *keys*.

Transformer Model

The first core intuition of our approach is the possibility of querying the Transformer model [30] with an arbitrary number of points. Inspired by the 3DETR architecture [17], the set of $N \times D$ features produced by our backbone is fed to an encoder to produce a new feature map of di-



(a) Input $(N = 2048)$ **(b)** Coarse Output $(r = 4)$ **(c)** Refined Output $(r = 4)$

Figure 4.3: Visualization of coarse points after the Transformer decoder and the corresponding refined output from the residual refinement module [44].

mension $N \times D$. The Transformer encoder has a single layer with a four-headed attention and an MLP with two layers. These features, along with a set of $r \times N$ queries, are processed by the two-layers Transformer decoder to generate $r \times N$ 3-dimensional Euclidean coordinates, which represent the coarse upsampled point cloud. The queries are obtained by embedding with Fourier positional encoding [130] the points sampled from the unit sphere manifold.

Residual Refinement

The coarse output from the Transformer decoder might be noisy, non-uniform distributed and have several outliers. Motivated by other works [194, 198], we design an attention-based residual refinement with a similar architecture as the feature extractor described previously. In particular, we employ a single *Point Transformer Block* (PTB) [39] composed by a PTL, two linear layers, a ReLU activation function and a residual connection. Differently from Equation 4.1, in this case the PTL takes as input the local features extracted from the backbone, while the positional encodings are computed with the 3D coarse coordinates. The D-dimensional vector produced by the PTB is finally projected to a 3-dimensional space with a two-layers MLP that computes the residual for each point, which is added to the coarse prediction to obtain the refined result. Figure 4.3 visualizes a typical example. It can be seen that he coarse points are already close to the underlying shape, but refinement helps in adjusting outliers.

4.3.2 Spherical Mixture of Gaussians

The second key intuition which underpins our work is to map the input point cloud into a probability distribution on a canonical domain which can be conveniently sampled. To this end, we estimate the parameters of a *Gaussian Mixture Model* (GMM) on the unit sphere from the deep features extracted by the PTL. The choice of this domain over a 2D square is motivated by the fact that the unit sphere is a manifold without boundaries, which avoids the truncation

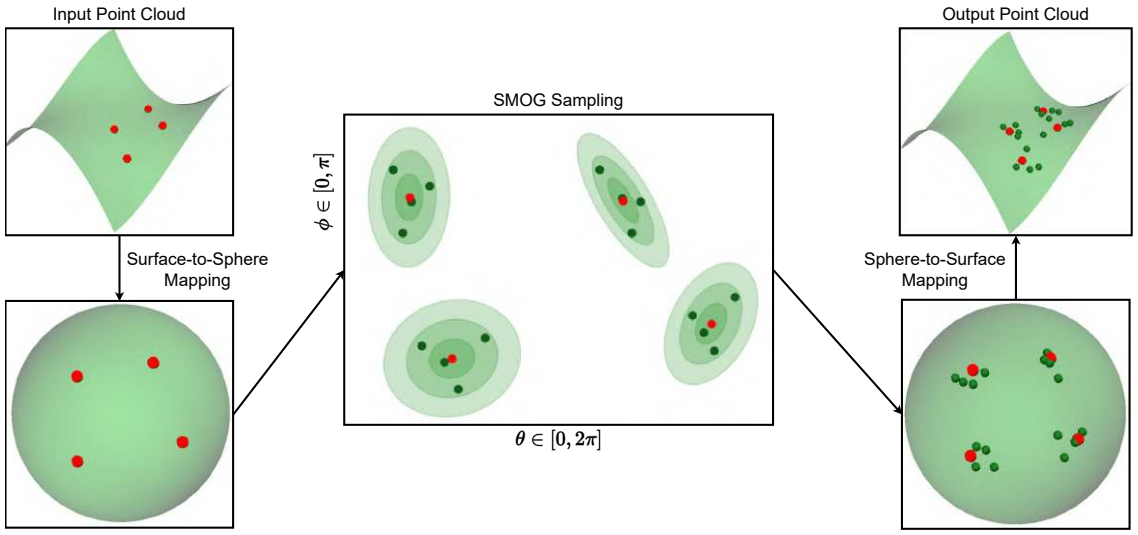


Figure 4.4: The sampling process shown in detail [44]. Red points are the Gaussian means on the unit sphere for each input point, while green points are the arbitrary samples from the SMOG. Red points on the right are just for visualization.

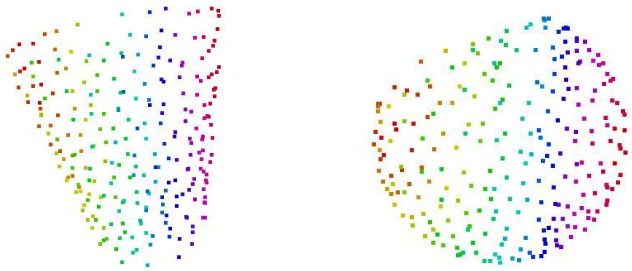
of the Gaussian distributions in the GMM. More specifically, we define a K components GMM $\Gamma = \{w_i, \mu_i, \Sigma_i\}_{i=1}^{K}$, where w_i, μ_i and Σ_i are the mixture weight, mean and covariance of the i-th Gaussian and $K = N$. Therefore, the likelihood of a point $\mathbf{z}$ on the unit sphere is given by a weighted combination of individual components $u_i(\mathbf{z})$:

$$u_\Gamma(\mathbf{z}) = \sum_{i=1}^{N} w_i u_i(\mathbf{z}) \tag{4.2}$$

Each component is weighted equally (i.e. $w_i = \frac{1}{N}$) and its parameters (μ_i, Σ_i) are estimated by a MLP ξ, that predicts the mean vector μ_i in Cartesian coordinates and the covariance matrix Σ_i in spherical coordinates. In particular, Σ_i is built by exploiting the constraint of symmetry:

$$\Sigma_i = \begin{bmatrix} \sigma_\theta^2 & \sigma_{\theta\varphi} \\ \sigma_{\theta\varphi} & \sigma_\varphi^2 \end{bmatrix} \tag{4.3}$$

where σ_θ^2, σ_φ^2 and $\sigma_{\theta\varphi}$ are the outputs of the covariance head of the MLP corresponding to the azimuth angle $\theta \in [0, 2\pi]$ and the elevation angle $\varphi \in [0, \pi]$, respectively. In order to generate a dense output point cloud, we sample an arbitrary number of points from the distribution and learn the inverse mapping from the spherical domain to the target surface. This process is illustrated in Figure 4.4. In the implementation of SMOG sampling, we have to ensure that the covariance matrix is positive semi-definite, in order for it to be well-defined. The simplest way



(a) Input Point Cloud **(b)** Spherical Projection

Figure 4.5: The proposed sphere parameterization learns a meaningful mapping and preserves the shape topology [44].

to impose this condition is to clamp the covariance value $\sigma_{\theta\varphi}$ between the following bounds:

$$-\sigma_\theta\sigma_\varphi \leq \sigma_{\theta\varphi} \leq \sigma_\theta\sigma_\varphi \tag{4.4}$$

To prove this, consider a real symmetric 2×2 matrix:

$$A = \begin{bmatrix} a & b \\ b & c \end{bmatrix} \tag{4.5}$$

Such a matrix is positive semi-definite if and only if its eigenvalues are non-negative. The eigenvalues of a 2×2 matrix can be computed in closed form as the solutions to:

$$\det(A - \lambda I) = (a - \lambda)(c - \lambda) - b^2 = 0 \tag{4.6}$$

We can immediately apply the well-known formula for solving quadratic equations:

$$\lambda_{1,2} = \frac{a + c \pm \sqrt{(a + c)^2 - 4(ac - b^2)}}{2} \tag{4.7}$$

The argument of the square root is always non-negative, as it can be rewritten as the sum of two positive numbers:

$$4b^2 + (a - c)^2 \geq 0 \tag{4.8}$$

Since $a > 0$ and $c > 0$ by construction, the first solution is also guaranteed to be non-negative as the sum of three non-negative components:

$$\lambda_1 = \frac{a + c + \sqrt{(a + c)^2 - 4(ac - b^2)}}{2} \geq 0 \tag{4.9}$$

Let's analyze the second solution:

$$\lambda_2 = \frac{a + c - \sqrt{(a+c)^2 - 4(ac - b^2)}}{2} \geq 0 \tag{4.10}$$

It can be rewritten as follows:

$$a + c \geq \sqrt{(a+c)^2 - 4(ac - b^2)} \tag{4.11}$$

Since both sides are non-negative, we can square them and we can expand the right-hand side:

$$a^2 + 2ac + c^2 \geq a^2 + c^2 - 2ac + 4b^2 \tag{4.12}$$

With simple computations, this becomes:

$$4ac \geq 4b^2 \implies |b| \leq \sqrt{ac} \implies -\sqrt{ac} \leq b \leq \sqrt{ac} \tag{4.13}$$

Note that this finding is confirmed by noting that the correlation value $\rho_{\theta\varphi}$ is bounded between $[-1, 1]$ and by definition:

$$\rho_{\theta\varphi} = \frac{\sigma_{\theta\varphi}}{\sqrt{\sigma_\theta^2 \sigma_\varphi^2}} \implies -1 \leq \frac{\sigma_{\theta\varphi}}{\sqrt{\sigma_\theta^2 \sigma_\varphi^2}} \leq 1 \implies -\sigma_\theta \sigma_\varphi \leq \sigma_{\theta\varphi} \leq \sigma_\theta \sigma_\varphi \tag{4.14}$$

From an implementation point of view, we ensure this condition with the `torch.clamp` function in PyTorch [172]. Even if this naive approach sets the gradients and the related update signal to zero outside the acceptance region, we have found experimentally that very few values lie outside this range during training. An alternative approach for imposing the same constraint is to decompose the covariance matrix as follows:

$$\boldsymbol{\Sigma} = \mathbf{R}(\alpha)\mathbf{D}\mathbf{R}(\alpha)^\top \tag{4.15}$$

where $\mathbf{R}(\alpha)$ is the 2D rotation matrix for a given angle α and $\mathbf{D}$ is a diagonal matrix with variances as diagonal elements. The resulting covariance matrix is symmetric and positive definite as required, with the additional advantage of predicting a continuous value for the angle α as the output of the network, instead of clamping and zeroing the gradients. We have implemented and tested this strategy, but we did not notice any significant improvement. Figure 4.5 shows an example of our surface-to-sphere mapping, which preserves the shape topology.

4.3.3 Loss Function

Let $\widetilde{\mathcal{Q}}_r = \{\widetilde{\mathbf{q}}_i \in \mathbb{R}^3\}_{i=1}^{rN}$ be the coarse prediction generated by the Transformer with upsampling rate r, $\mathcal{Q}_r = \{\mathbf{q}_i \in \mathbb{R}^3\}_{i=1}^{rN}$ the refined prediction and $\mathcal{Y}_r = \{\mathbf{y}_i \in \mathbb{R}^3\}_{i=1}^{rN}$ the ground-truth upsampled point cloud. For training, we adopt a similar strategy as in [201] to define a distance between two generic point clouds $\mathcal{X} = \{\mathbf{x}_i \in \mathbb{R}^3\}_{i=1}^{X}$ and $\mathcal{Z} = \{\mathbf{z}_i \in \mathbb{R}^3\}_{i=1}^{Z}$ as:

$$d_\Pi(\mathcal{X}, \mathcal{Z}) = \frac{1}{X} \sum_{i=1}^{X} \|\mathbf{x}_i - \Pi(\mathbf{x}_i, \mathcal{Z})\|_2^2 \tag{4.16}$$

where $\Pi(\cdot, \cdot)$ is the projection of a 3D point to a point cloud. This term is computed as the weighted combination of the nearest points to $\mathbf{x}$ in $\mathcal{Z}$, with indices $\mathcal{N}(\mathbf{x}, \mathcal{Z})$:

$$\Pi(\mathbf{x}, \mathcal{Z}) = \frac{\sum_{k \in \mathcal{N}(\mathbf{x}, \mathcal{Z})} w_k \mathbf{z}_k}{\sum_{k \in \mathcal{N}(\mathbf{x}, \mathcal{Z})} w_k} \tag{4.17}$$

The weights w_k are given by:

$$w_k = e^{-\alpha \|\mathbf{x} - \mathbf{z}_k\|_2^2}, \; k \in \mathcal{N}(\mathbf{x}, \mathcal{Z}) \tag{4.18}$$

with $\alpha = 10^3$. The corresponding loss is then defined by the following bidirectional sum:

$$\mathcal{L}_\Pi(\mathcal{X}, \mathcal{Z}) = d_\Pi(\mathcal{X}, \mathcal{Z}) + d_\Pi(\mathcal{Z}, \mathcal{X}) \tag{4.19}$$

Our arbitrary upsampling network is trained with the sum of three losses:

$$\mathcal{L} = \mathcal{L}_\Pi(\widetilde{\mathcal{Q}}_4, \mathcal{Y}_4) + \mathcal{L}_\Pi(\mathcal{Q}_4, \mathcal{Y}_4) + \mathcal{L}_{ACD}(\widetilde{\mathcal{Q}}_1, \mathcal{P}) \tag{4.20}$$

where the last term is the *Augmented Chamfer Distance* (ACD) between the reconstructed point cloud and the input, defined for two generic point clouds as follows:

$$\mathcal{L}_{ACD}(\mathcal{X}, \mathcal{Z}) = \max \left\{ \frac{1}{X} \sum_{\mathbf{x} \in \mathcal{X}} \min_{\mathbf{z} \in \mathcal{Z}} \|\mathbf{x} - \mathbf{z}\|_2^2, \frac{1}{Z} \sum_{\mathbf{z} \in \mathcal{Z}} \min_{\mathbf{x} \in \mathcal{X}} \|\mathbf{x} - \mathbf{z}\|_2^2 \right\} \tag{4.21}$$

In practice, each training iteration consists of two forward passes and a single backward pass. During the upsampling forward pass, the Transformer decoder is queried with $4 \times N$ points

sampled from the SMOG and the projection loss components for both the coarse and refined upsampled outputs are computed. On the other hand, in the reconstruction phase, the estimated means of the SMOG components are fed to the decoder and the ACD with respect to the input point cloud is evaluated. This additional term is required to learn the proper positions of the Gaussian means on the unit sphere, which acts effectively as an intermediate probabilistic representation of the input shape. The ablation studies in Section 4.4 prove this insight with numerical evidence.

4.4 Experiments

4.4.1 Implementation Details

Datasets

In order to compare the proposed method with state-of-the-art *fixed-ratio* methods, we employ the challenging PU1K dataset [193], which contains 1147 synthetic 3D models of various shapes, split into 1020 training samples and 127 testing samples. Consistently with existing literature [193, 194, 192, 208], 50 patches are extracted from each training shape with 256 points as input to the model and 1024 points as ground truth ($r = 4$). During testing, 2048 points are sampled from the original mesh with Poisson disk sampling. We employ a similar strategy as in [214] to extract overlapping patches with 256 geodesically close points. The network predicts the upsampled outputs at patch level and the final result is given by combining the overlapping patches with FPS to obtain 8192 points.

Moreover, we evaluate our approach against *flexible-ratio* methods on the widely used PU-GAN dataset [192], which is composed by 147 shapes collected from the PU-Net [191], MPU [208] and Visionair [215] repositories. The same patch-based training and testing procedure is followed with 2048 points as input, but ground truth point clouds with different sizes are generated to adapt to the values of the scaling factor r. Finally, we test the generalization capabilities of our approach on real-world LiDAR point clouds from the KITTI dataset [116].

Parameters

Our framework is implemented in PyTorch [172]. The features dimension is set to $D = 128$. The MLPs in the Transformer model has hidden dimension equal to 64 in the encoder and 128 in the decoder. For the local feature extractor and the refinement module, the number of

neighboring points are set to 32 and $4r$, respectively. The projection components in the loss function are weighted with $\lambda_1 = \lambda_2 = 0.01$, while the reconstruction component weight is set to $\lambda_3 = 1$. In the $\Pi(\cdot, \cdot)$ computation, k is equal to 4. The model is trained with a batch size of 64 for 100K iterations on a single V100 GPU with 32GB of RAM, using random rotation and perturbation as data augmentation techniques to avoid overfitting. The AdamW [28] optimizer is used with a learning rate decay following a cosine schedule [216] from 5e−4 to 1e−6, a weight decay of 0.1 and a gradient L_2 norm clipping of 0.1.

Baselines

For the task of upsampling with a fixed ratio $r = 4$ on the PU1K dataset, we provide quantitative comparison against the state-of-the-art models PU-Net [191], MPU [208], PU-GAN [192], PU-GCN [193] and Dis-PU [194]. On the other hand, the flexible methods MAFU [198], PU-EVA [199] and Neural Points [201] are used as baselines for arbitrary upsampling on the PU-GAN dataset. For a fair comparison, we used the official pre-trained models when available and re-trained the other ones with the official published code. Following previous works, the results are quantitatively evaluated with the Chamfer Distance (CD), Hausdorff Distance (HD) and Point-to-Surface (P2F) distance metrics. The CD is the sum of squared distances between nearest neighbor correspondences of the input and ground truth point clouds, while the HD effectively measures the influence of outliers in the predicted results. On the other hand, the P2F distance is computed against the underlying surface, thus estimating the quality of the upsampled point cloud as an approximation of the real shape. All the metrics are averaged on the whole test set and a lower value indicates better upsampling performance.

4.4.2 Quantitative Results

The quantitative evaluation of the method on the PU1K test set and a fixed upsampling ratio $r = 4$ for point clouds with $N = 2048$ points is presented in Table 4.1. It can be noticed that we achieve the lowest HD value, indicating that APU-SMOG performs upsampling with fewer outliers with respect to state-of-the-art models, as well as the lowest P2F distance, showing a better approximation of the underlying surface. Our approach falls behind Dis-PU [194] in terms of the CD metric by a slight margin. Nevertheless, this quantity measures the consistency of the result with the ground truth *point cloud*, rather than with the *target shape*: two different sets of points sampled from the same mesh have a non-zero Chamfer distance, despite belonging to the same surface by definition.

Method	CD↓	HD↓	P2F μ↓	P2F σ↓
PU-Net [191]	1.155	11.626	4.834	6.799
MPU [208]	0.935	10.298	3.551	5.971
PU-GAN [192]	0.885	16.539	3.717	5.746
PU-GCN [193]	0.584	5.822	2.499	4.441
Dis-PU [194]	**0.511**	<u>4.104</u>	<u>2.013</u>	<u>2.926</u>
Ours	<u>0.528</u>	**2.549**	**1.667**	**2.075**

Table 4.1: Quantitative comparison with state-of-the-art methods on the PU1K dataset and $r = 4$. The units are all 10^{-3} and lower is better. Best and second results are **bold** and <u>underlined</u> [44].

Furthermore, the numerical performances as a function of the ratio r on the PU-GAN test set are provided in Table 4.2. In order to be able to compare against state-of-the-art flexible methods [198, 199], we constrain r to be an integer value smaller than 16, even if we can generate predictions with any $r \in \mathbb{R}$. Our approach achieves the best CD value, as well as the best HD value by a significant margin, along the whole spectrum of ratios. Note that both MAFU [198] and Neural Points [201] have a lower P2F metric since they require ground truth normals at training time, which helps in finding the correct surface. On the other hand, our approach consistently outperforms PU-EVA [199], which is trained from raw point clouds. For completeness, we include the comparison against *fixed-ratio* models as reference. To generate predictions for $r \in \{8, 12, 16\}$, each network is queried iteratively with $r = 4$ and the desired number of points is obtained with FPS.

4.4.3 Ablation Studies

We perform a set of ablation studies in Table 4.3 to evaluate the contribution of each module to our pipeline. The results have been obtained with different versions of our model trained on the PU1K dataset with fixed ratio $r = 4$. In order to measure the influence of the SMOG representation, we replace it with a FoldingNet-like [206] strategy, i.e. sampling the unit sphere uniformly. The quantitative results confirm that our approach is able to generate better predictions, as it adaptively learns a local probability distribution around each point. Furthermore, we investigate the influence of the number of GMM components on the output point clouds, reducing it to $K = N/4$. The numerical evaluation exhibits decent performances, suggesting that the model associates each Gaussian distribution to a local neighborhood of points. However, having a single component for each input point leads to the best results overall.

	Method	$r = 4$			$r = 8$			$r = 12$			$r = 16$		
		CD↓	HD↓	P2F↓	CD↓	HD↓	P2F↓	CD↓	HD↓	P2F↓	CD↓	HD↓	P2F↓
Fixed	PU-GAN [192]	0.274	4.694	1.943	0.489	6.985	2.621	0.233	6.093	2.548	0.209	6.055	2.556
	PU-GCN [193]	0.304	2.656	2.541	0.256	4.175	2.825	0.204	4.157	2.737	0.195	4.176	2.716
	Dis-PU [194]	0.360	5.133	2.868	0.352	7.028	3.338	0.291	6.694	3.258	0.271	6.645	3.240
Flexible	MAFU [198]	0.322	2.116	**1.721**	0.195	2.389	2.037	0.164	2.392	2.034	0.158	2.367	1.971
	NePs [201]	0.368	4.556	1.875	0.254	10.146	**1.928**	0.203	10.018	**1.922**	0.159	9.263	**1.957**
	PU-EVA [199]	0.394	7.676	2.915	0.322	7.951	3.148	0.290	8.191	3.234	0.286	8.390	3.269
	Ours	**0.276**	**1.909**	2.634	**0.194**	**1.628**	2.613	**0.162**	**1.626**	2.635	**0.149**	**1.948**	2.666

Table 4.2: Quantitative comparison with state-of-the-art methods on the PU-GAN dataset [192] and flexible upsampling ratios. The units are all 10^{-3} and lower is better. Best and second results *among flexible methods* are **bold** and underlined. Fixed methods are included as reference [44].

The third row in Table 4.3 proves the effectiveness of the attention-based residual refinement step. The significantly higher HD value indicates that the raw output from the Transformer decoder contains several outliers, which are correctly positioned closer to the surface by this module. Moreover, we train our model without the reconstruction loss and notice a performance drop. This implies that $\mathcal{L}_{ACD}$ is a strong bias towards learning the correct positions of the Gaussian means on the unit sphere. Finally, the advantage of the projection loss for upsampling over ACD is shown in the last row. The remarkable difference in all the metrics justifies the choice of $\mathcal{L}_{\Pi}$ in the final design.

4.4.4 Qualitative Results

Figure 4.6 shows qualitative upsampling results in comparison with state-of-the-art methods. These results have been obtained under the same settings as Table 4.1, namely with all the models trained on the PU1K dataset with fixed upsampling ratio $r = 4$ for input point clouds having size $N = 2048$. Close-up views show that our approach is particularly effective in preserving fine-grained structures, such as the piano's pole and the motorcycle mirror, and disambiguating complex shape (see the bag's handle). It can be noticed that other models fail to distinguish between different details of the surface and tend to merge them together, thus producing noisy point clouds. Moreover, the proposed attention-based residual refinement block generates refined outputs with fewer outliers (e.g. the plane motors). Moreover, we investigate the upsampling results on the PU-GAN dataset [192] against concurrent flexible architectures for different ratios $r \in \{4, 8, 12, 16\}$. Close-up views in Figure 4.7 show that our framework produces cleaner shapes along the whole range of upsampling factors, even when compared with methods requiring ground truth normals at training time [198, 201].

Ablation	CD↓	HD↓	P2F↓
FoldingNet-like [206]	0.558	2.761	1.700
$N/4$ SMOG components	0.564	2.803	1.686
w/o refinement	0.572	8.536	2.182
w/o rec. loss, $\mathcal{L}_{\Pi}$ ups.	0.561	2.843	1.726
w/o rec. loss, $\mathcal{L}_{ACD}$ ups.	0.816	6.459	2.694
Ours	**0.528**	**2.549**	**1.667**

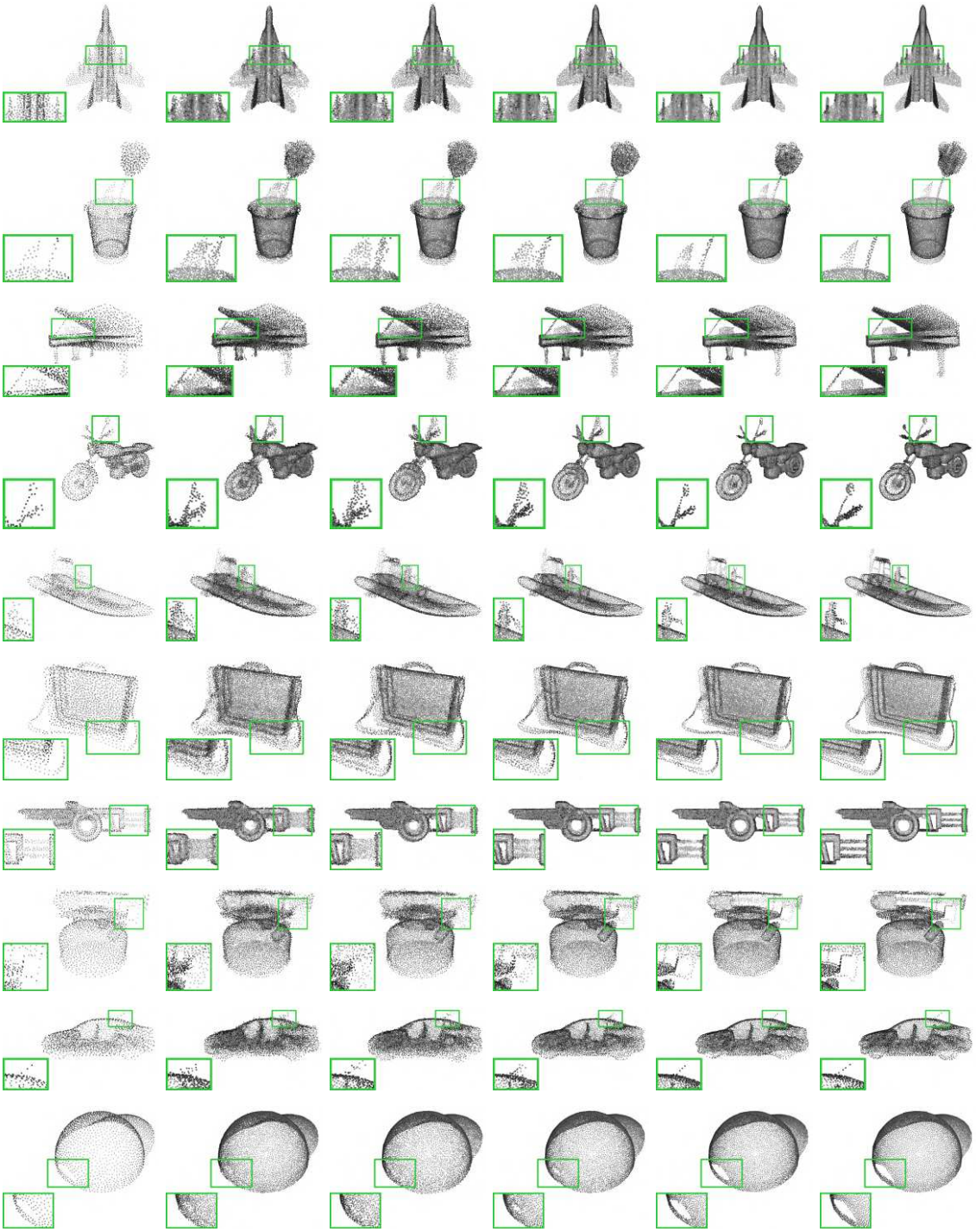
Table 4.3: Quantitative ablation studies. The units are all 10^{-3} and **bold** denotes the best performance [44].

Finally, we present the qualitative results of our approach when the upsampled point clouds are used to extract a continuous mesh, using an open-source implementation of the Poisson surface reconstruction [6, 7] algorithm available in the Open3D library [217]. The outputs of our method allow to reconstruct better shapes with respect to state-of-the-art, as shown in Figure 4.8 (e.g. the index finger in the hand and the human arm in the statue). Furthermore, we provide additional surface reconstruction results as a function of the upsampling ratio in Figure 4.9. In this case, the input point cloud has size $N = 1024$ and the ratios are chosen randomly in a wide range to further demonstrate the ability of handling arbitrary factors $r \in \mathbb{R}$. Note that the ground truth mesh is the actual synthetic shape from the PU1K dataset, while the ground truth point cloud is generated for reference by Poisson disk sampling. The quality of the resulting mesh consistently improves with increasing values of the upsampling ratio, thus proving the effectiveness of our method for the downstream 3D reconstruction task.

4.4.5 Generalization and Robustness

We conduct further experiments to demonstrate the robustness and generalization capabilities of the proposed approach. Firstly, we provide qualitative results on real-world point clouds from the KITTI dataset [116]. This task is particularly challenging, since street-level LiDAR data with noise and occlusions are very different from synthetic training samples. Figure 4.10 shows the generalization power of our approach on different urban elements such as cars, trucks and pedestrians. Additional examples on real-world objects point clouds from the ScanObjectNN dataset [218] are provided in Figure 4.11. In both cases, it can be seen that our model is robust to domain shifts, without any fine-tuning.

Furthermore, since the previous qualitative results in Figure 4.6 and Figure 4.7 have been generated with point clouds having $N = 2048$ points, we show in Figure 4.12 that the predic-



(a) Input **(b)** PU-GAN [192] **(c)** PU-GCN [193] **(d)** Dis-PU [194] **(e)** Ours **(f)** Ground Truth

Figure 4.6: Qualitative comparison with state-of-the-art methods on the PU1K dataset [193]. Inputs with 2048 points (left) are upsampled to 8192 points (right), with upsampling ratio $r = 4$. Details are best viewed when zoomed in [44].

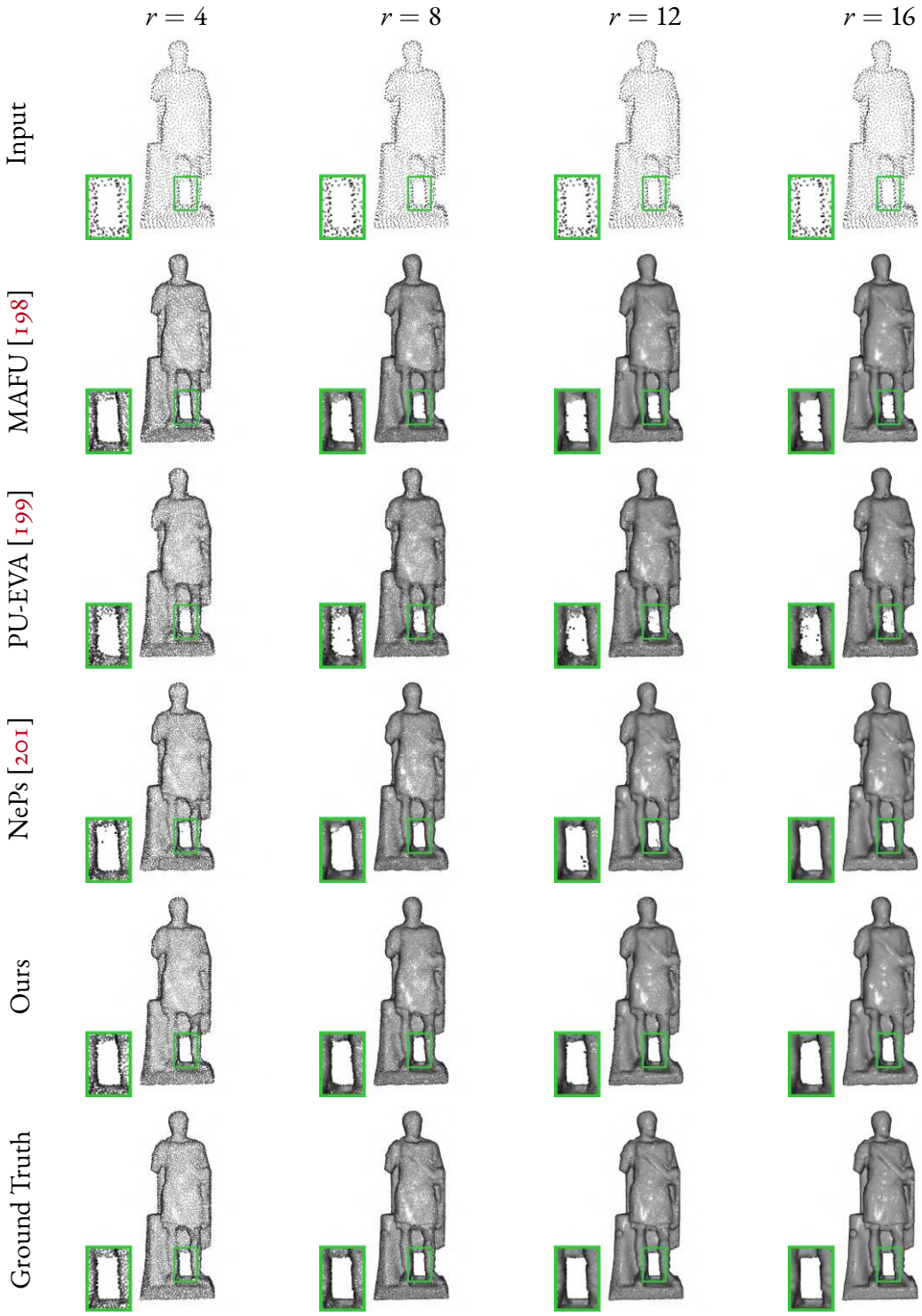
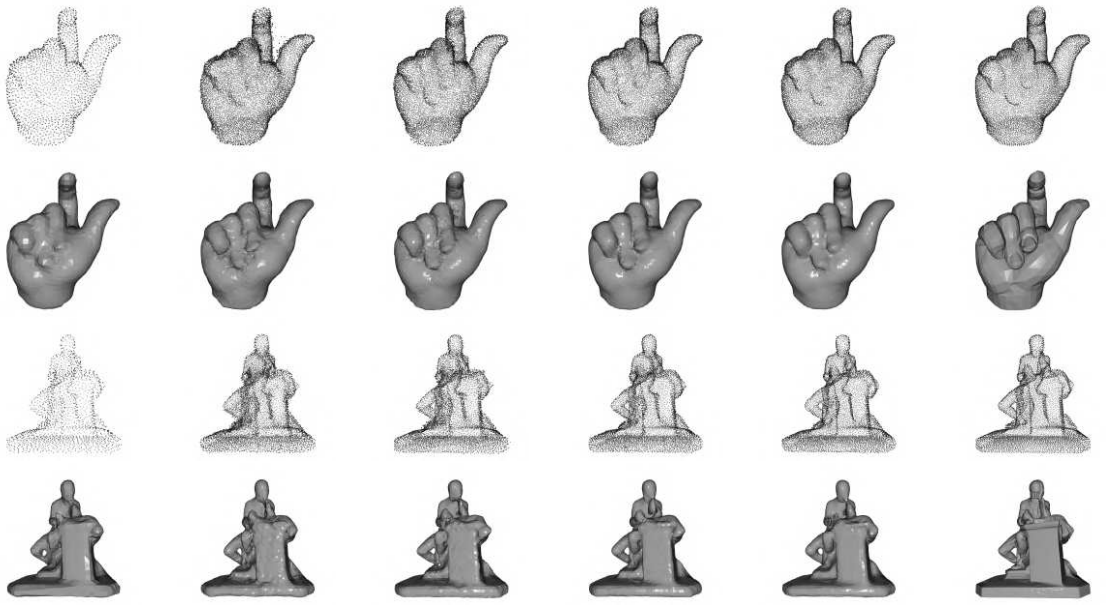
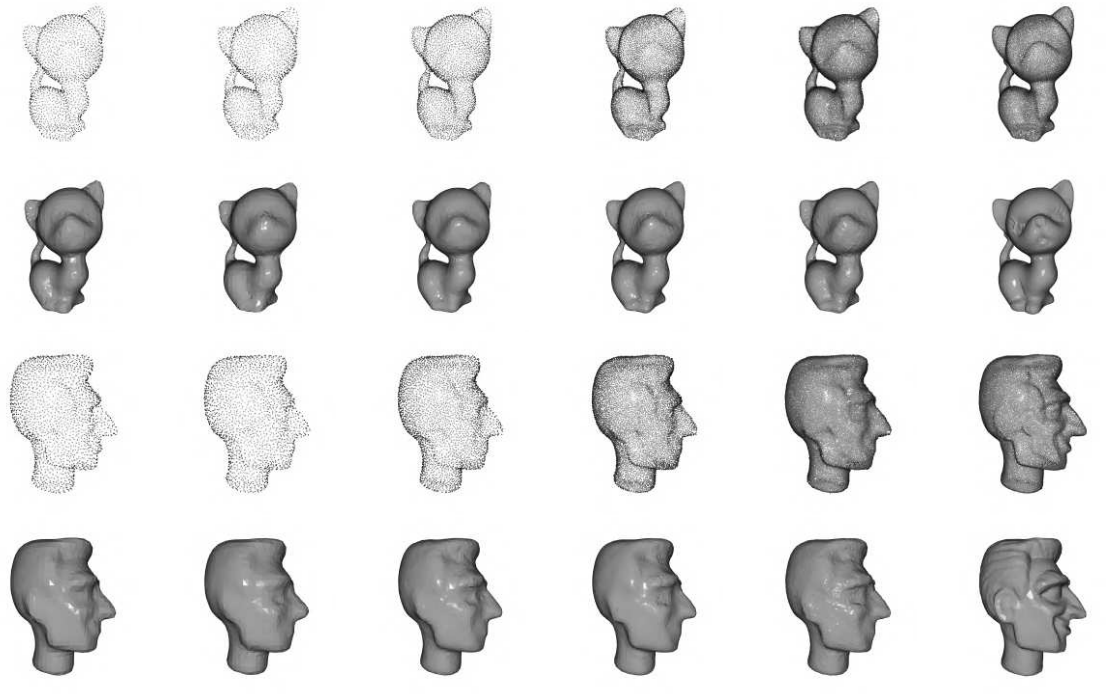


Figure 4.7: Qualitative comparison with state-of-the-art methods on the PU-GAN dataset [192] and flexible upsampling ratios. Inputs with 2048 points are upsampled with $r \in \{4, 8, 12, 16\}$. Details are best viewed when zoomed in [44].



(a) Input **(b)** PU-GAN **(c)** PU-GCN **(d)** Dis-PU **(e)** Ours **(f)** Ground Truth

Figure 4.8: Qualitative comparison when the upsampled point clouds are used to compute a mesh [44].



(a) Input **(b)** $r = 1.98$ **(c)** $r = 3.45$ **(d)** $r = 7.72$ **(e)** $r = 18.11$ **(f)** Ground Truth

Figure 4.9: Qualitative comparison with arbitrary upsampling ratios when computing a mesh [44].

Method	CD↓	HD↓	P2F μ↓	P2F σ↓
Dis-PU [194]	1.392	12.382	4.050	7.216
Ours	1.315	7.883	5.582	4.857

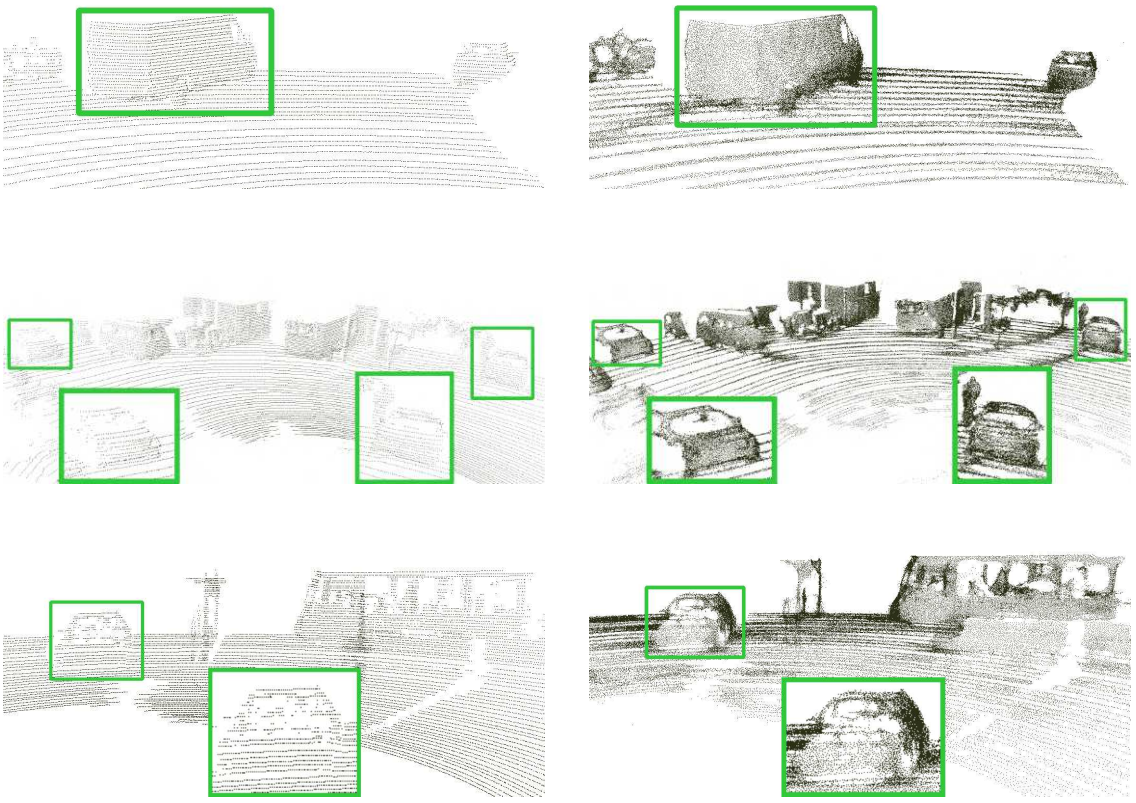
Table 4.4: Quantitative comparison when the input point cloud is corrupted with Gaussian noise ($\sigma = 0.01$) [44].

tions of our method closely follow the underlying surfaces for a wide variety of input sizes. Even for the sparsest input with 256 points, thin structures such as the horse's legs are upsampled correctly. We also study the effect of input point cloud size on the upsampling results from a different point of view. Whereas in Figure 4.12 we show the differences in the upsampled point clouds for a *fixed* ratio $r = 4$, in Figure 4.13 the output size is maintained fixed at $M = 8192$ and the upsampling factor varies in $r \in \{16, 8, 4\}$ for different input sizes. It can be seen that our method achieves satisfying results also for the sparsest input $N = 512$. This robustness, combined with the flexibility of our method, provides a full control on the input and output sizes for real-world upsampling scenarios, where the number of points might need to adapt to computational and bandwidth constraints.

Finally, in order to simulate real noisy point clouds from scanning sensors, Figure 4.14 shows the upsampling results for three different levels of additive Gaussian noise. It can be noticed that the duck shape is successfully maintained for both the clean and the corrupted inputs. We also present a quantitative comparison for with the best concurrent method [194] in Table 4.4, showing that we outperform it in most metrics. The input is corrupted with Gaussian noise having standard deviation $\sigma = 0.01$.

4.5 Conclusion

In this chapter, we presented a novel approach for point cloud upsampling with arbitrary scaling factors. A Transformer-based architecture is designed to decouple the upsampling process in two key steps: (i) firstly, the sparse input is mapped to an intermediate representation as a Spherical Mixture of Gaussians; (ii) then, such distribution is sampled arbitrarily, and the Transformer decoder learns to map each sample back to the surface. The predictions are further improved by an attention-based residual refinement module, which allows to achieve state-of-the-art results on different benchmarks. This strategy enables arbitrary upsampling, since the model is trained a single time with a fixed ratio and it can be queried at test time with any desired value. Future research directions would be to depart from the patch-based training procedure and to learn the weights of the GMM jointly with the Gaussian parameters.



(a) Input Point Cloud **(b)** Output Point Cloud

Figure 4.10: Qualitative upsampling results on the real-world KITTI dataset [116, 44].

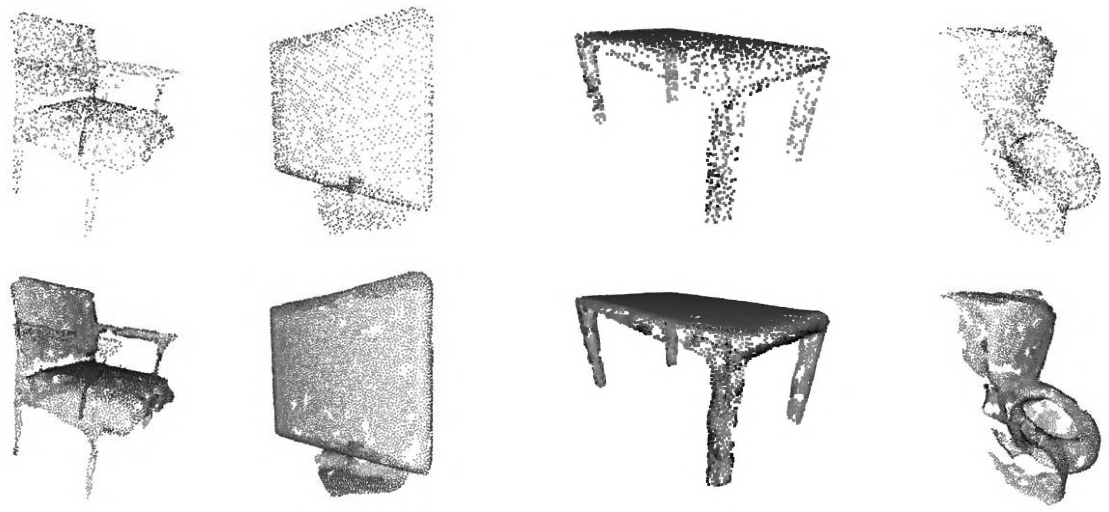
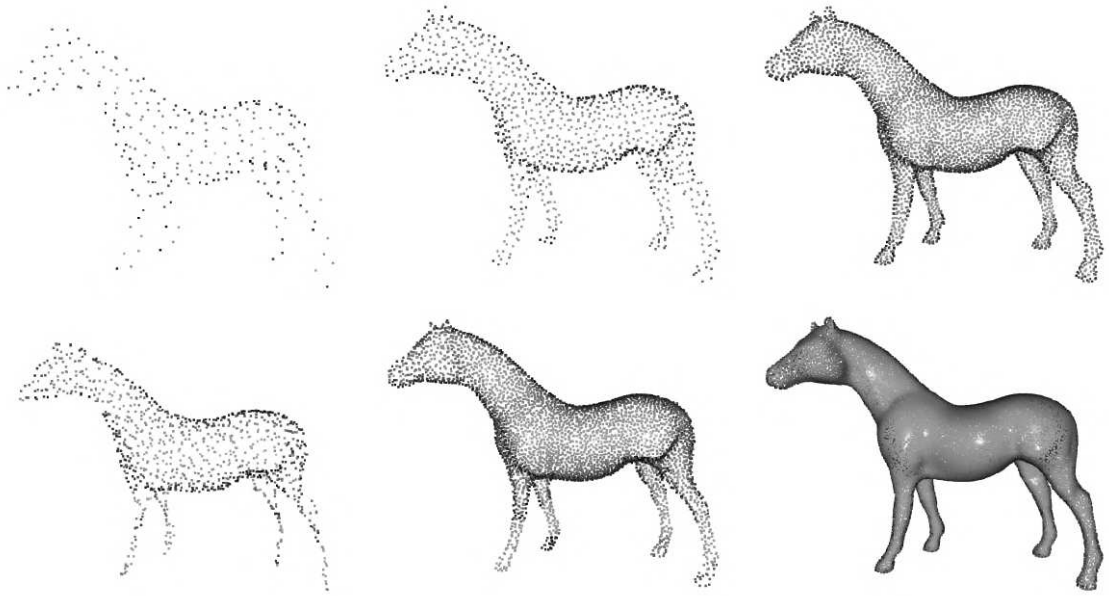
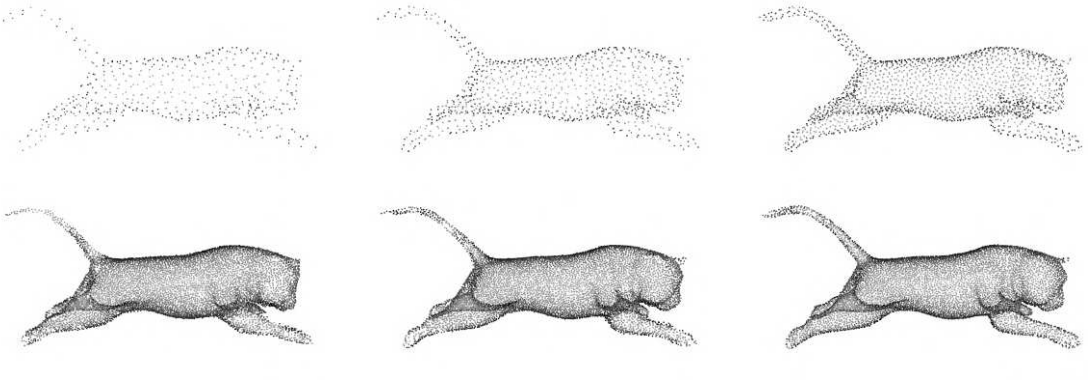


Figure 4.11: Qualitative upsampling results on the real-world ScanObjectNN dataset [218, 44].



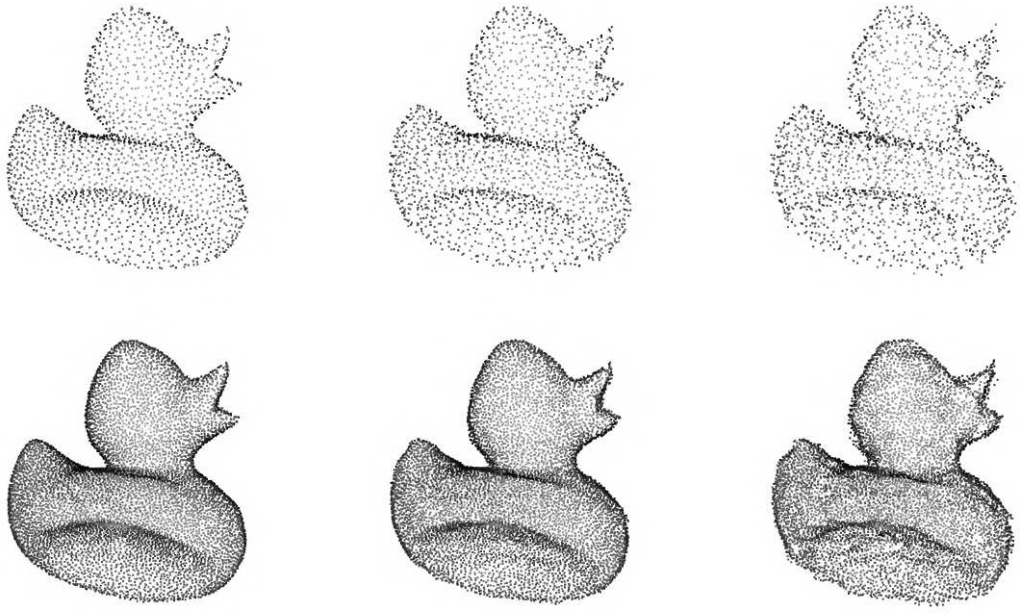
(a) $N = 256$ **(b)** $N = 1024$ **(c)** $N = 4096$

Figure 4.12: Effect of input point cloud size on the upsampling results [44].



(a) 512×16 **(b)** 1024×8 **(c)** 2048×4

Figure 4.13: Effect of input point cloud size on the upsampling results while keeping the output size fixed [44].



(a) $\sigma = 0.00$ **(b)** $\sigma = 0.01$ **(c)** $\sigma = 0.02$

Figure 4.14: Effect of additive Gaussian noise on the upsampling results [44].

5

Neural Network Compression

5.1 Introduction

This chapter is based on the original contributions currently under review at [45]:
Dell'Eva A.*, **Orsingher M.***, Lee Y. M., and Bertozzi M., *Teacher Features-Driven Regularization for Knowledge Distillation*, IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR), 2024.
* Equal contribution.

The widespread diffusion of mobile and wearable devices requires efficient network inference on hardware with limited resources. However, modern deep learning models benefit from large architectures [37, 38, 30, 219, 220] that fail to meet the computational and memory requirement of such devices. In order to bridge this gap, three main neural compression approaches have been proposed: distilling the knowledge from large networks to smaller models [221, 222, 223, 224], pruning redundant parameters [225, 226, 227, 228, 229] and quantizing the weights to low-precision [230].

Knowledge Distillation (KD) is a model compression technique in which a lightweight *student* network is trained to mimic the outputs [221, 224, 231] or the intermediate representations [222, 223, 232, 233] of a much larger *teacher* model. This allows the knowledge learned by the teacher to be effectively transferred to the student, thus achieving a higher accuracy

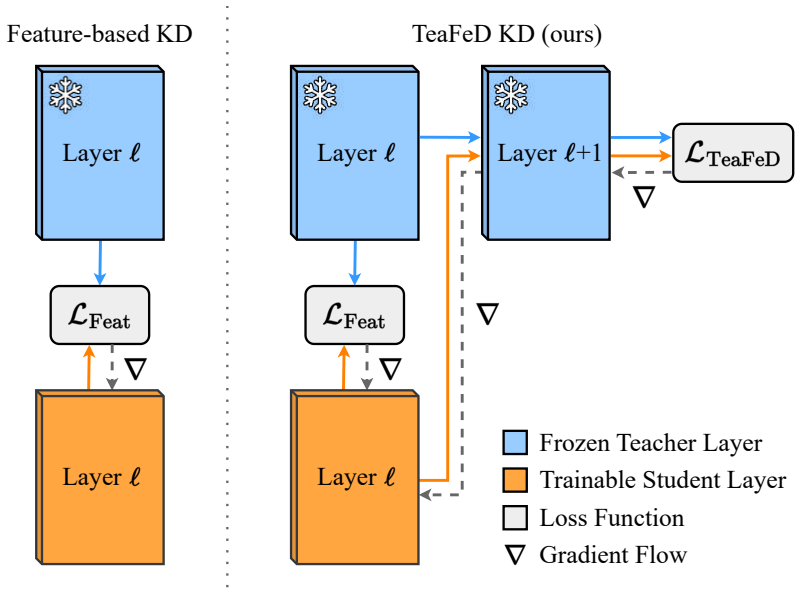


Figure 5.1: Overview of the proposed regularization loss for KD. We inject the student features at layer ℓ to the next *frozen* teacher layer $\ell + 1$ and match them with the corresponding teacher features [45].

than training it from scratch. Existing KD methods fall into two main categories: logit-based [221, 224, 231] and feature-based [222, 223, 232, 233]. KD was originally proposed in [221] for the classification task, as a method to match the output logits of two models with different capacity. The logits are converted to pseudo-probabilities with a softmax with rescaled temperature, and the KL divergence between the two distributions is minimized. Recently, feature-based methods have emerged by following the insights in [222]. The key idea is to align the intermediate representations of the student to those of the teacher, in order to boost the distillation performances.

However, directly matching the feature maps of the teacher is not a sufficient condition for the student to predict the correct output. Moreover, it is a hard optimization constraint to satisfy when the two models have different architectures or capacity. To this end, we propose *TeaFeD*, a novel *Tea*cher *Fe*atures-*D*riven regularization approach for KD, which is visualized in Figure 5.1. The key idea is to train a generic student layer ℓ to produce a latent representation for the next teacher layer $\ell + 1$ that matches the corresponding teacher features. Since network layers typically implement non-injective functions, different features at level ℓ might produce the same output at level $\ell + 1$. Our approach allows for gradients to be propagated from a higher level of abstraction, thus providing more flexibility to the student in learning intermediate representations. The effect of this regularization loss on the features learned by the

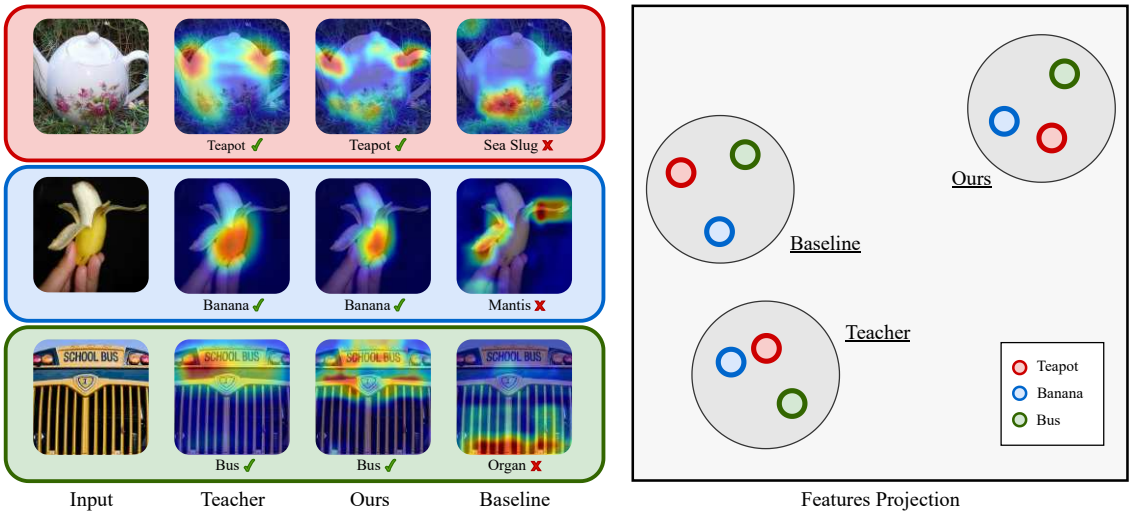


Figure 5.2: Visualization of the representations learned by the student with the baseline method [222] and our approach [45]. UMAP projections [234] (right) show that the baseline produces closer features to the teacher, but fails to classify inputs correctly as it focuses on irrelevant pixels, while our approach matches the teacher's attention. Activation maps are computed with Grad-CAM [235].

student network is shown in Figure 5.2. Even if the direct matching baseline [222] produces closer representations to the teacher (visualized as UMAP projections [234] on the right), it fails to classify correctly the input images. The activation maps on the left, computed with Grad-CAM [235], prove that these errors stem from the focus on irrelevant pixels, while our approach allows the student to concentrate on the same image regions as the teacher.

Furthermore, we present a novel neural compression strategy, called *sparse self-distillation*, that combines network pruning with KD. After the pruning process, the pruned model is typically fine-tuned for a few epochs, in order to partially retrieve the accuracy lost due to the removed parameters. However, in practice, there is a tradeoff between the desired reduction rate and the bound on accuracy loss. To this end, we propose to integrate KD in the pruning and fine-tuning loop, by setting the original unpruned model as the teacher. In this way, the accuracy gap of the student (i.e. the pruned network) can be significantly reduced, and model compression can be pushed further. While this approach works across different network architectures, pruning strategies and distillation methods, we show that the proposed TeaFeD regularization is particularly effective also in this setting.

Therefore, our contributions to existing literature can be summarized as follows:

- We propose a novel approach for feature-based KD that complements direct matching with a teacher features-driven regularization loss.

- We show that relaxing the feature matching constraint at corresponding layers by distilling knowledge from a higher level of abstraction enables the student model to learn more robust and generalizable latent representations.
- We prove the effectiveness of our regularizer on standard benchmarks, showing state-of-the-art results for a wide variety of vision tasks, including image classification, semantic/instance segmentation and object detection.
- We also introduce sparse self-distillation, a neural compression method that combines network pruning with KD and significantly improves the sparsity-accuracy tradeoff.

5.2 Related Work

5.2.1 Logit-based Knowledge Distillation

Logit-based methods distill the knowledge only at output level, by minimizing the divergence between the logits predicted by the teacher and the student model [221, 224, 231, 236, 237]. The seminal KD paper [221] proposed to rescale the temperature parameter in the last softmax layer, in order to smooth the output distribution and focus on negative logits [238]. Recently, DKD [224] showed that reformulating the distillation loss into a weighted sum for positive and negative logits effectively decouples their contributions and further boosts performances. On the other hand, DML [236] employed a mutual learning paradigm to train both models simultaneously, while TAKD [231] introduced an intermediate teacher assistant to reduce the capacity gap between the teacher and the student. Finally, Lin *et al.* [237] presented a multi-level framework to distill logits across instances, batches and categories. Logit-based methods are efficient in terms of both memory and compute, but they typically achieve lower results with respect to state-of-the-art feature-based approaches.

5.2.2 Feature-based Knowledge Distillation

Feature-based methods aim at transferring intermediate representations from the teacher to the student, in order to provide a richer supervision [222, 223, 239, 240, 241, 233, 242, 243, 244, 245, 232, 246, 247, 248, 249, 250]. Following the idea presented in [222] to mimic the teacher features via a simple L_2 loss, several approaches have been developed to improve different aspects of the pipeline. Some works [232, 223, 243] proposed to mask the feature maps,

thus allowing the student to focus on relevant parts of the image. Popular masking strategies include learnable tokens [232], random masks reconstructed with generative layers [223], and fixed separation of background and foreground for object detection [243]. Another approach to alleviate the constraint of direct feature matching is to transform the raw features into attention maps [248, 242, 243, 244] or to compute local statistics, such as pairwise similarities [249] and the Pearson correlation coefficient [246]. Furthermore, cross-level distillation has been explored in [239, 241], with the goal of sharing features across different network layers, thus integrating knowledge from multiple learning stages. Inspired by these methods, we propose a simple approach to complement direct feature matching with a novel regularization term that forces each student layer to learn meaningful features from the subsequent teacher layers.

5.2.3 Neural Network Pruning

The goal of network pruning is to remove redundant parameters from a given model, thus making it faster for inference. *Unstructured* pruning approaches [226, 227, 251, 252, 253] directly zero out specific weights based on their magnitude, without altering the network architecture. Despite being a generic and straightforward strategy, unstructured pruning typically requires dedicated hardware that supports sparse matrix multiplication. On the other hand, *structured* pruning methods [225, 228, 229, 254, 255] physically remove entire channels or layers from the network and allow to exploit standard GPU acceleration. In both cases, the pruning ratio is practically limited by the accuracy lost due to the removed parameters, even after fine-tuning the remaining ones. We propose to push further this sparsity limit by self-distilling the dense (unpruned) model to the sparse (pruned) network, during the pruning and fine-tuning loop.

5.3 Teacher Features-Driven Regularization

In this section, we formally describe and evaluate our proposed teacher features-driven regularization loss. We start by deriving the vanilla logit-based and feature-based KD methods, in order to provide the required technical background. Then, we present our solution to improve the features learned by the student. Finally, we extensively test our strategy on multiple vision tasks and provide a detailed analysis of its main components.

5.3.1 Method

Preliminaries

The fundamental baseline in KD research is the vanilla logit-based method introduced in [221]. Given a model trained on a classification task, the output logits $\mathbf{z} \in \mathbb{R}^C$ can be converted to pseudo-probabilities using the softmax activation as:

$$y_j = \frac{e^{z_j/t}}{\sum_{c=1}^{C} e^{z_c/t}}, \tag{5.1}$$

where z_j and y_j are the logit and probability values for j-th category, respectively. In the KD context, the *temperature* parameter t is often larger than 1, producing softer distributions which alleviate the over-confidence problem in neural networks [25] ($t = 1$ yields the vanilla softmax output). To transfer knowledge from complex to compact models, the original logit-based technique minimizes the KL divergence [180] between teacher and student outputs as:

$$\mathcal{L}_{\text{Log}} = \text{KL}\left(\mathbf{y}^T || \mathbf{y}^S\right) = \sum_{j=1}^{C} y_j^T \log\left(\frac{y_j^T}{y_j^S}\right), \tag{5.2}$$

where the superscripts T and S indicate whether the outputs originate from the teacher or the student, respectively. As a result, the *dark knowledge* is transferred via soft labels from the teacher, improving the prediction of the student.

On the other hand, the distillation approaches based on features are versatile and mostly agnostic with respect to the model architecture, which can vary considerably for different tasks. The basic feature-based method [222] can be formulated as:

$$\mathcal{L}_{\text{Feat}}^{\ell} = \|F_\ell^T - \mathcal{A}_\ell(F_\ell^S)\|_2^2. \tag{5.3}$$

Here, $F_\ell^T \in \mathbb{R}^{C_{T_\ell} \times H_{T_\ell} \times W_{T_\ell}}$ and $F_\ell^S \in \mathbb{R}^{C_{S_\ell} \times H_{S_\ell} \times W_{S_\ell}}$ correspond to the teacher and student feature maps extracted from layer ℓ, while $\mathcal{A}_\ell$ denotes the convolutional adaptation layer needed to align the student features dimensions with those of the teacher. The approach forces the student to directly imitate the teacher features, which may not be optimal if the models' architectures diverge significantly or their capacities are widely different. To address the issue, we introduce a novel feature-based regularization, which employs a straightforward L_2 loss to steer the student toward a more robust solution space.

Overview

Given a teacher network with a structured hierarchy of layers denoted as $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_L\}$, and a parallel student network denoted as $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_L\}$, our TeaFeD method regularizes the learning process at matching layers. At each stage ℓ, the aligned student features $\mathcal{A}_\ell(F_\ell^S)$ are propagated to the corresponding subsequent frozen teacher stage $\mathcal{T}_{\ell+1}$, which provides new features on a higher level of abstraction. Thus, the loss is formulated as:

$$\mathcal{L}_{\text{TeaFeD}}^{\ell} = \|F_{\ell+1}^T - \mathcal{T}_{\ell+1}\left(\mathcal{A}_\ell(F_\ell^S)\right)\|_2^2. \tag{5.4}$$

Due to non-linear mappings and pooling transformations, the teacher layer implementing the loss function is typically non-injective. This suggests that different inputs might produce identical latent representations, allowing for more flexibility in the features derived by the student. Therefore, our penalization term effectively acts as a powerful regularizer, promoting better generalization.

Loss Function

We employ a simple exponentially increasing weighting scheme on the sequence of features for both our loss and the vanilla feature-based loss. The aggregated losses can be expressed as:

$$\mathcal{L}_{\text{Feat}} = \sum_{\ell=1}^{L} \gamma_1^{-\ell}\, \mathcal{L}_{\text{Feat}}^{\ell}. \tag{5.5}$$

$$\mathcal{L}_{\text{TeaFeD}} = \sum_{\ell=1}^{L} \gamma_2^{-\ell}\, \mathcal{L}_{\text{TeaFeD}}^{\ell}, \tag{5.6}$$

Student models are trained with the following cumulative loss:

$$\mathcal{L} = \mathcal{L}_{\text{Task}} + \mathcal{L}_{\text{Log}} + \alpha\, \mathcal{L}_{\text{Feat}} + \beta\, \mathcal{L}_{\text{TeaFeD}}, \tag{5.7}$$

where α and β are hyperparameters to balance the losses and $\mathcal{L}_{\text{Task}}$ represents the task-specific loss (e.g., for classification, the cross-entropy computed between student outputs and ground truth labels). In practice, we set $\alpha = \beta$ and $\gamma_1 = \gamma_2 = \gamma \in (0, 1]$ to halve the number of hyperparameters to be tuned. The core steps of our approach are delineated in Algorithm 5.1.

Algorithm 5.1 TeaFeD Regularization in PyTorch [172]

```
# (img, y): Input and Label
# (S, T): Student and Teacher
# num_layers: Number of Layers
# [adapt]: Adaptation Layers
# (g1, g2, a, b): Loss Hyperparameters
import torch.nn.functional as F
y_S, feat_S = S(img)
y_T, feat_T = T(img)
L_task = F.cross_entropy(y_S, y)
L_log = F.kl_div(y_S, y_T)
L_feat = 0.
L_teafed = 0.
for l in range(num_layers):
  F_S = adapt[l](feat_S[l])
  l2_feat = F.mse_loss(F_S, feat_T[l])                  # Equation 5.3
  l2_teafed = F.mse_loss(T[l+1](F_S), feat_T[l+1])      # Equation 5.4
  L_feat += (g1 ** -l) * l2_feat                        # Equation 5.5
  L_teafed += (g2 ** -l) * l2_teafed                    # Equation 5.6
L = L_task + L_log + a * L_feat + b * L_teafed          # Equation 5.7
```

5.3.2 Experiments

Image Classification

We employ two standard image classification datasets for our experiments. CIFAR-100 [256] consists of 60k images (50k for training and 10k for validation) from 100 classes, with an image resolution of 32×32. TinyImageNet [257] is a subset of ImageNet [258] and comprises 90k images from 200 categories, divided into 80k for training and 10k for validation, with resolution 64×64. In addition to the basic KD method [221], we reproduce various state-of-the-art strategies for comparison, including FitNet [222], CRD [233], SRRL[240], SemCKD [241], ReviewKD [239], and CAT-KD [242]. Note that all the approaches, except for KD itself, incorporate the vanilla KD loss (Equation 5.2). Following [37, 259], we employ standard data augmentation and normalize all images based on channel means and standard deviations. For CIFAR-100, the total training epochs are 240, the batch size is set to 64, and the base learning rate for SGD optimizer to 0.05. The hyperparameters for our loss are configured as $\{\alpha = 20, \gamma = 0.8\}$. For TinyImageNet, the models are trained for 200 epochs, the

Student	ResNet-116	VGG-8	ResNet-8x4	WRN-40-1
	73.70	71.00	73.30	72.17
KD [221]	76.26	73.43	74.66	74.12
FitNet [222]	76.48	73.54	74.46	73.86
CRD [233]	76.83	74.37	75.72	74.44
SRRL [240]	76.79	73.85	75.51	74.78
SemCKD [241]	77.07	74.35	76.00	74.39
CAT-KD [242]	77.39	74.46	**76.78**	75.10
Ours	**77.54**	**75.01**	76.64	**75.18**
Teacher	ResNet-110x2	VGG-13	ResNet-32x4	WRN-40-2
	78.17	75.14	79.21	76.53

Table 5.1: Top-1 test accuracy (%) of homogeneous teacher-student pairs on CIFAR-100 [256], averaged over 4 runs.

Student	VGG-8	ShuffleNetV1	MobileNetV2	ShuffleNetV2
	71.00	71.15	65.94	73.33
KD [221]	72.51	75.27	69.35	75.84
FitNet [222]	72.82	75.36	69.49	75.91
CRD [233]	73.80	75.88	70.05	76.40
SRRL [240]	73.49	75.59	69.96	76.19
SemCKD [241]	75.33	75.78	70.26	77.97
CAT-KD [242]	75.66	75.43	69.84	77.01
Ours	**75.74**	**76.36**	**70.94**	**78.84**
Teacher	ResNet-32x4	VGG-13	WRN-40-2	ResNet-32x4
	79.21	75.14	76.53	79.21

Table 5.2: Top-1 test accuracy (%) of heterogeneous teacher-student pairs on CIFAR-100 [256], averaged over 4 runs.

batch size is 128 and the base learning rate is 0.1, while the settings for the loss parameters are $\{\alpha = 5, \gamma = 0.8\}$. For classification, distillation is performed on the backbone features. The ablation study on losses and features employed can be found in Section 5.3.3.

The classification results are shown in Table 5.1, Table 5.2 and Table 5.3, with evaluations based on Top-1 accuracy. It can be noticed that we consistently outperform other state-of-the-art approaches for nearly every pair on both datasets, including ReviewKD [239] and CAT-KD [242] which require careful hyperparameter tuning for each teacher-student combination. Remarkably, we improve accuracy by a considerable margin in several cases. For example, with the ShuffleNetV2/ResNet-32x4 pair on CIFAR-100, we achieve a 1% accuracy increase compared

Architecture	Homogeneous		Heterogeneous	
Student	VGG-8 55.83	ResNet-8x4 55.53	ShuffleNetV1 58.64	VGG-8 55.83
KD [221]	60.36	58.16	61.25	60.52
FitNet [222]	60.35	58.21	61.47	60.69
CRD [233]	61.72	59.62	63.50	61.81
SRRL [240]	61.41	59.72	62.74	62.02
SemCKD [241]	61.41	57.50	62.53	62.57
CAT-KD [242]	61.52	58.13	63.70	62.59
Ours	**62.24**	**60.28**	**64.21**	**63.75**
Teacher	VGG-13 61.34	ResNet-32x4 64.53	VGG-13 61.34	ResNet-32x4 64.53

Table 5.3: Top-1 test accuracy (%) of multiple teacher-student pairs on TinyImageNet [257], averaged over 3 runs.

to the second-best method. Similarly, in the VGG-8/VGG-13 combination on TinyImageNet, the student surprisingly exceeds even the teacher performance. The outcomes prove that the introduced regularization term facilitates flexible feature learning, enhancing the student robustness regardless of dataset and architecture types.

Object Detection and Instance Segmentation

The experiments are conducted with MMDetection [260] on the COCO2017 dataset [261], which comprises 120k training images and 5k validation images divided in 80 object categories. We compare our approach with recent distillation methods for detectors, including FKD [262], CWD [244], FGD [243], MGD [223], and MasKD [232]. We train the models for 24 epochs with SGD optimizer (momentum = 0.9, weight decay = 10^{-4}). When the student and teacher share the same head structure, we adopt the inheriting strategy [243, 223] to initialize the student with the teacher's neck and head parameters. The hyperparameters are $\{\alpha = 5\times10^{-5}, \gamma = 1\}$ for one-stage detectors and $\{\alpha = 5 \times 10^{-7}, \gamma = 1\}$ for two-stage detectors. The distillation loss is computed on the feature maps extracted from the neck.

The experiments for the object detection task include three different models, namely the one-stage detector RetinaNet [263], the anchor-free one-stage detector RepPoints [264] and the two-stage detector Faster RCNN [265]. The evaluation metrics are the mean bounding box Average Precision (mAP) and the variants AP_S, AP_M, and AP_L, corresponding to small, medium and large object sizes, respectively. In the context of instance segmentation, we focus

	Teacher	Student	mAP	AP_S	AP_M	AP_L
One-stage	RetinaNet ResNeXt101 41.0	RetinaNet-Res50	37.4	20.6	40.7	49.7
		FKD [262]	39.6	22.7	43.3	52.5
		CWD [244]	40.8	22.7	44.5	55.3
		FGD [243]	40.7	22.9	45.0	54.7
		MGD [223]	41.0	23.4	45.3	**55.7**
		MasKD [232]	41.0	22.6	45.3	55.3
		Ours	**41.3**	**23.4**	**45.9**	54.6
Anchor-free	RepPoints ResNeXt101 44.2	RepPoints-Res50	38.6	22.5	42.2	50.4
		FKD [262]	40.6	23.4	44.6	53.0
		CWD [244]	42.0	24.1	46.1	55.0
		FGD [243]	42.0	24.0	45.7	55.6
		MGD [223]	42.3	24.4	46.2	55.9
		MasKD [232]	42.5	**24.9**	46.1	**56.8**
		Ours	**42.9**	24.8	**47.1**	56.3
Two-stage	Cascade Mask RCNN ResNeXt101 47.3	Faster RCNN-Res50	38.4	21.5	42.1	50.3
		FKD [262]	41.5	23.5	45.0	55.3
		CWD [244]	41.7	23.3	45.5	55.5
		FGD [243]	42.0	23.8	46.4	55.5
		MGD [223]	42.1	23.7	46.4	56.1
		MasKD [232]	42.4	24.2	**46.7**	55.9
		Ours	**42.4**	**24.4**	46.3	**56.4**

Table 5.4: Knowledge distillation results for object detection on COCO [261].

	Teacher	Student	mAP	AP_S	AP_M	AP_L
One-stage	SOLO-Res101 3x, MS 37.1	SOLO-Res50-1x	33.1	12.2	36.1	50.8
		FGD [243]	36.0	14.5	39.5	54.5
		MGD [223]	36.2	14.2	39.7	**55.3**
		Ours	**36.4**	**15.3**	**40.1**	55.2
Two-stage	Cascade Mask RCNN ResNeXt101 41.1	Mask RCNN-Res50	35.4	16.6	38.2	52.5
		FGD [243]	37.8	17.1	40.7	56.0
		MGD [223]	38.1	17.1	**41.1**	56.3
		Ours	**38.1**	**17.8**	41.0	**56.5**

Table 5.5: Knowledge distillation results for instance segmentation on COCO [261].

Architecture	Homogeneous	Heterogeneous
Student	PspNet-Res18 69.85	DeepLabV3-Res18 73.20
SKDS [245]	72.70	73.87
CWD [244]	73.53	75.93
MGD [223]	74.10	76.31
Ours	**74.87**	**76.44**
Teacher	PspNet-Res101 78.34	PspNet-Res101 78.34

Table 5.6: Knowledge distillation results for semantic segmentation on Cityscapes [268].

on SOLO [266] and Mask RCNN [267] models, which are evaluated with the mean mask Average Precision metric (mAP). From the results presented in Table 5.4 and Table 5.5, it can be noticed that our method achieves better performances with respect to the other state-of-the-art approaches for both tasks. For instance, our TeaFeD regularization boosts the RepPoints and Mask RCNN students, leading to 4.3 and 2.7 mAP improvement, respectively. Additionally, RetinaNet and SOLO obtain a remarkable 2.8 and 3.1 improvement on the AP_S metric.

Semantic Segmentation

Semantic segmentation is evaluated on Cityscapes dataset [268], consisting of 5k urban scenes images divided in 3k for training, 500 for validation and 1.5k for testing. We compare our method with recent approaches, such as SKDS [245], CWD [244] and MGD [223]. We train the student models with SGD (momentum = 0.9, weight decay = 5×10^{-4}) for 128 epochs with batch size equal to 16. Following [223], our loss is computed only on the last backbone feature and the corresponding parameters are set to $\{\alpha = 2 \times 10^{-5}, \gamma = 1\}$. The experiments are conducted with MMSegmentation [269].

We evaluate the segmentation task on both homogeneous and heterogeneous distillation settings, using mean Intersection over Union (mIoU) as the quantitative metric. Specifically, we employ PspNet-Res101 [270] as the teacher, while PspNet-Res18 [270] and DeepLabV3-Res18 [271] as students. The results in Table 5.6 confirm that our method is the best among the state-of-the-art approaches. In detail, the two students register a 5.02 and 3.24 improvement with respect to the original model trained without distillation, respectively.

Architecture	Homogeneous	Heterogeneous
Student	VGG-8	ShuffleNetV2
	71.00	73.33
$\ell \in \{1, 2, 3\}$	74.55	78.79
$\ell \in \{1, 2\}$	74.49	78.64
$\ell = 4$	73.41	76.74
$\ell = 3$	73.53	76.12
$\ell = 2$	74.11	77.81
$\ell = 1$	74.21	77.27
without $\mathcal{L}_{\text{Log}}$	74.53	78.73
without $\mathcal{L}_{\text{Feat}}$	74.70	78.09
without $\mathcal{L}_{\text{TeaFeD}}$	74.49	78.07
Ours	**75.01**	**78.84**
Teacher	VGG-13	ResNet-32x4
	75.14	79.21

Table 5.7: Ablation studies on different versions of the student for both homogeneous and heterogeneous settings. Layers are numbered in descending order. Ours is equivalent to $\ell \in \{1, 2, 3, 4\}$. Results are averaged over 3 runs.

5.3.3 Ablation Studies

We conduct several ablation studies to assess the contribution of distinct layers and losses on the model predictions. The results in Table 5.7 are obtained by training a homogeneous and a heterogeneous classification network pair on CIFAR-100 with different configurations.

In order to evaluate the influence of layers, we perform distillation while removing them sequentially. Note that, in this context, layers are enumerated from highest to lowest, with $\ell = 1$ corresponding to the last layer of the backbone. Two observations emerge from the accuracy values: firstly, integrating more layers is beneficial for the student, as demonstrated by training the model with only a limited subset; secondly, the final layers contribute more to the network performance, as highlighted by the improved numerical evaluations at $\ell = 1$ and $\ell = 2$. This justifies the choice of applying exponential weighting to the feature-based losses.

Moreover, we investigate the impact of the different losses. The last rows of Table 5.7 confirm that our regularization objective plays a crucial role in the distillation process, as removing it leads to the most significant performance drop. Additionally, the results show that our loss outperforms the vanilla feature-based approach, when evaluated independently. Nonetheless, the best accuracy is achieved through joint training.

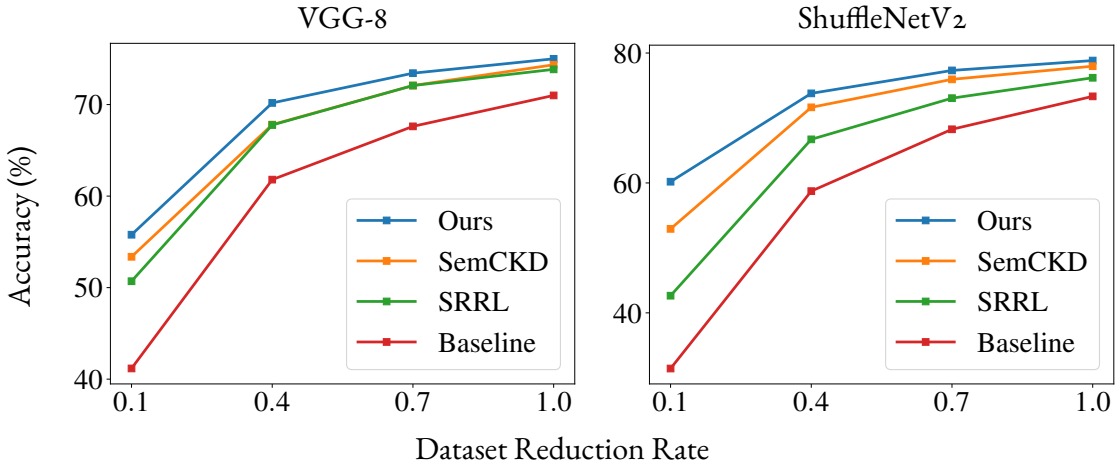


Figure 5.3: Analysis on the data efficiency of different KD methods when trained with different amounts of data [45].

5.3.4 Analysis

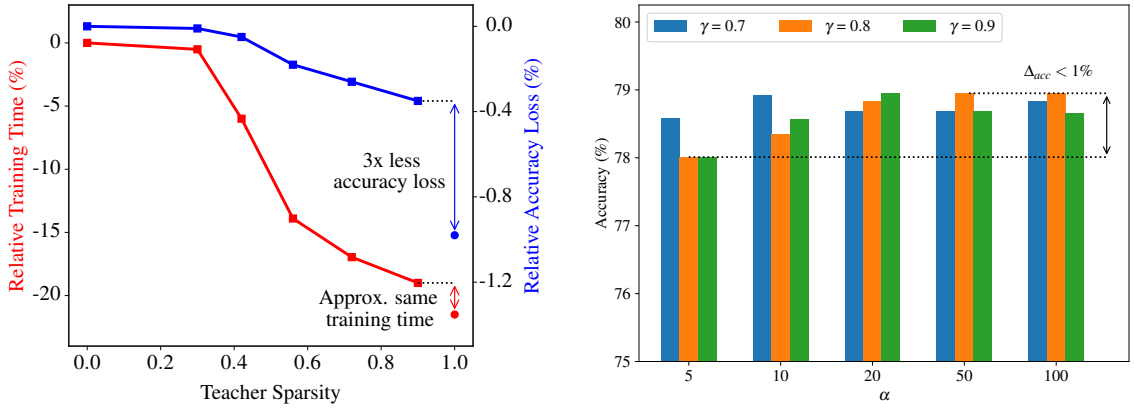
In this section, we provide an extensive analysis of our KD approach in terms of data and training efficiency, sensitivity to hyperparameters, student feature transferability and impact of the adaptation layer. We also present an extension for adapting it to any logit-guided approach. All the experiments are conducted for the classification task on CIFAR-100 [256] for a ShuffleNetV2 student distilled from a ResNet-32x4 teacher, except where stated otherwise.

Data Efficiency

We evaluate the dependency of different distillation approaches [241, 240] on the amount of training data, by reducing the training set of CIFAR-100 [256] to various ratios. Table 5.3 shows that we consistently outperform other methods across the whole spectrum of dataset reductions, for both ShuffleNetV2 and VGG-8 (with VGG-13 as teacher) architectures. This result proves that the proposed regularization loss can transfer knowledge from the teacher more efficiently.

Training Efficiency

While adding our $\mathcal{L}_{\text{TeaFeD}}$ as a learning objective leads to improved classification accuracy, it also increases the overall training time of the student network. We study this aspect by structurally pruning [225] the teacher at multiple sparsity values and analyzing the accuracy-efficiency trade-off. Figure 5.4a compares our results with the baseline feature matching approach [222]. It



(a) Pruning the teacher allows to reduce the training time almost to the baseline, while keeping more accuracy.

(b) Effect of varying the hyperparameters required to tune our regularization loss.

Figure 5.4: Analysis on the training efficiency and hyperparameters sensitivity of the proposed KD approach [45].

can be seen that distilling from a lighter teacher with our KD method allows to achieve approximately the same training time, while losing only 0.35% accuracy, which is almost $3\times$ less than the baseline 0.98% accuracy loss. Moreover, all the intermediate sparsity levels provide a substantial efficiency gain with a bounded performance loss, allowing to adapt the method to custom computational constraints.

Hyperparameters Sensitivity

The proposed approach depends on two main hyperparameters, namely the weights $\gamma \in (0, 1]$ of each feature layer and $\alpha \in \mathbb{R}$ for the feature-based KD losses. We show in Figure 5.4b that the performances of our regularization term are minimally affected by the choice of such parameters. By letting $\gamma \in \{0.7, 0.8, 0.9\}$ and $\alpha \in \{5, 10, 20, 50, 100\}$, the accuracy difference between the best and worst configuration is less than 1%. Moreover, note we avoid cherry-picking a specific set of good hyperparameters, as our choice in Section 5.3.2, i.e. $\{\alpha = 20, \gamma = 0.8\}$, is not even the optimal one.

Features Transferability

As we are interested in which features are learned by the student during the distillation procedure, we also analyze how such features transfer to datasets unseen at training time. To this end, the student trained on CIFAR-100 [256] is used as a frozen representation extractor for images from STL-10 [272] and TinyImageNet [257]. Then, we train a linear classifier on these repre-

Architecture	Homogeneous		Heterogeneous	
Teacher	ResNet-110x2		VGG-13	
Student	ResNet-116		ShuffleNetV1	
CIFAR-100 →	STL-10	TinyImg	STL-10	TinyImg
KD [221]	63.67	29.59	67.70	36.00
FitNet [222]	64.22	30.64	67.94	36.30
SRRL [240]	64.22	31.59	69.15	37.58
SemCKD [241]	66.42	**34.32**	69.17	38.63
CRD [233]	66.11	32.69	69.03	38.66
CAT-KD [242]	65.73	32.75	70.82	39.36
Ours	**66.68**	34.11	**70.86**	**39.52**

Table 5.8: Features transferability from CIFAR-100 [256] to other classification datasets [257, 272]. A linear classifier is trained on features extracted from a frozen student distilled with different KD methods.

sentations and evaluate the accuracy on both datasets. Table 5.8 shows the results against multiple baselines for homogeneous and heterogeneous teacher-student pairs. The regularization effect introduced by our proposed loss allows the student to learn more transferable features with respect to concurrent KD methods, thus proving its effectiveness.

Adaptation Layer

We compare several implementations of the adaptation layer $\mathcal{A}$, which aligns the student features with those of the teacher. The first rows of Table 5.9 suggest that any straightforward single-layer design is beneficial for the knowledge transfer procedure, providing comparable performances. Notably, the highest accuracy is achieved with a 3×3 convolutional layer followed by a batch normalization. This configuration is employed in every experiment across all tasks. Conversely, aligning the features with a three-layer transformation mitigates the distillation effectiveness. We conjecture this is due to the additional capacity provided by the multi-layer structure. In this case, the adaptation module can learn to transform any student representation into any complex teacher representation, preventing the student model from receiving useful feedback and thus learning robust intermediate features.

Layer	Accuracy
Conv 1×1 - BatchNorm	78.75
Conv 3×3 - BatchNorm - ReLU	78.60
Conv 1×1 - Conv 3×3 - Conv 1×1	78.34
Conv 1×1 - Depth-wise Conv 3×3 - Conv 1×1	78.46
Ours (Conv 3×3 - BatchNorm)	**78.84**

Table 5.9: Comparison of adaptation layers, where Conv $N \times N$ indicates a convolutional layer with a $N \times N$. Each convolution in the multi-layer configurations is followed by BatchNorm and ReLU non-linearity.

	Student	Ours	Ours$^+$	Teacher
Homog.	ResNet-116	77.54	77.65	ResNet-110x2
	VGG-8	75.01	74.81	VGG-13
	ResNet-8x4	76.64	76.63	ResNet-32x4
	WRN-40-1	75.18	75.26	WRN-40-2
Heterog.	VGG-8	75.74	75.41	ResNet-32x4
	ShuffleNetV1	76.36	76.16	VGG-13
	MobileNetV2	70.94	70.77	WRN-40-2
	ShuffleNetV2	78.84	78.43	ResNet-32x4

Table 5.10: Top-1 test accuracy (%) of homogeneous and heterogeneous teacher-student pairs on CIFAR-100 [256] obtained with our extended method, denoted as $+$.

Student	KD	DKD	Ours$^+$ w/ KD	Ours$^+$ w/ DKD	Teacher
ShuffleNetV2	75.84	77.02	78.43	78.70	ResNet-32x4
73.33					79.21

Table 5.11: Top-1 test accuracy (%) on CIFAR-100 [256] with our extended method and different logit-based approaches.

Logit-Guided Extension

We extend our method by modifying the design of the loss. In this setting, the features at layer ℓ are propagated to all the corresponding subsequent layers of the teacher, i.e. until the final layer L. Thus, our regularization term is computed on the output logits using Equation 5.2. As presented in Table 5.10, the accuracy values from this strategy are on par with our feature-driven loss, adding no significant advantages. Moreover, the primary limitation of this approach is the extended training time, which increases by 30% compared to our original configuration.

Additionally, in this revisited setup, we train the student with a different logit-based loss, applying DKD [224] to both $\mathcal{L}_{\text{Log}}$ and $\mathcal{L}_{\text{TeaFeD}}$. As expected, the adoption of a more effective training objective on the logits enhances the results, which can be found in Table 5.11. Despite the poor training efficiency, the setting can be leveraged as a pure logit-based distillation method (removing $\mathcal{L}_{\text{Feat}}$), boosting the student trained exclusively with $\mathcal{L}_{\text{Log}}$ when the features are not available. Furthermore, this configuration is adaptable to any logit-based technique.

5.4 Sparse Self-Distillation

5.4.1 Method

The proposed regularization loss allows to train an efficient student model with comparable accuracy with respect to the teacher network. To further enhance the deployment performances of the student, we also present a novel compression strategy, called *sparse self-distillation*, which integrates the distillation procedure in the pruning and fine-tuning loop.

Let S be the student network after KD training, $\mathcal{P}_r$ a generic pruning algorithm [225, 226, 227] with sparsity r and $\hat{S}_r = \mathcal{P}_r(S)$ the resulting model when applying $\mathcal{P}_r$ to S. Typically, $\hat{S}_r$ is fine-tuned for a few epochs with ground truth data and the task-specific loss $\mathcal{L}_{\text{Task}}(\hat{S}_r)$, in order to partially retrieve the accuracy lost during pruning. However, in practice, there is a trade-off between the target sparsity and the bound on accuracy loss, especially for small networks already compressed with KD.

We propose to alleviate this constraint by setting the original unpruned student S as the *teacher* and to self-distill it during the pruning process. This means that $\hat{S}_r$ is fine-tuned with the sum of two loss terms:

$$\mathcal{L} = \mathcal{L}_{\text{Task}}(\hat{S}_r) + \mathcal{L}_{\text{KD}}(\hat{S}_r, S) \tag{5.8}$$

The distillation loss $\mathcal{L}_{\text{KD}}$ forces the sparse student $\hat{S}_r$ to follow the output distribution and the learned representations of the original dense student S. In this way, the sparsity r can be pushed further, leading to the deployment of smaller and faster models with competitive accuracy. Note that this strategy is agnostic to the distillation method implemented by $\mathcal{L}_{\text{KD}}$. However, our regularization loss is particularly effective in this setting, for both structured and unstructured pruning algorithms.

5.4.2 Experiments

We test our sparse self-distillation procedure on the CIFAR-100 classification dataset [256], with different network architectures, including VGG-8, ResNet-116 and MobileNetV2, as well as various distillation algorithms [222, 240, 241]. We also compare the proposed method against the baseline approach of iterative pruning and fine-tuning on ground truth data, without KD. Unstructured pruning is applied with the built-in PyTorch [172] utility, while we rely on the state-of-the-art DepGraph [225] algorithm for structured pruning. In all the experiments, the remaining parameters of the pruned network $\hat{S}_r$ are fine-tuned for 50 epochs. The quantitative results in Table 5.5 show two key results. Firstly, integrating KD within the pruning and fine-tuning loop is consistently better than the baseline approach, for any distillation method. Secondly, our proposed TeaFeD regularization is particularly effective in this setting, outperforming all the other KD losses. Such results find evidence across different architectures and pruning algorithms. This allows to push the network sparsity further, with a much smaller accuracy loss, thus leading to faster models for the inference stage.

5.5 Conclusion

In this chapter, we show that feature-based knowledge distillation can be improved by complementing the direct feature matching baseline with a teacher features-driven regularization loss, which enables the student model to learn robust intermediate representations. We extensively evaluate and ablate the proposed approach, outperforming state-of-the-art distillation methods on several vision tasks and benchmarks. Furthermore, we also introduce a novel neural compression strategy that effectively combines KD with network pruning, in order to improve the sparsity-accuracy tradeoff when deploying the student model on inference devices. The main limitation of our approach is the need for accessing the teacher weights at training time. As future work, we plan to tackle this limitation in order to support more general black-box teachers and to apply our method on non-convolutional architectures.

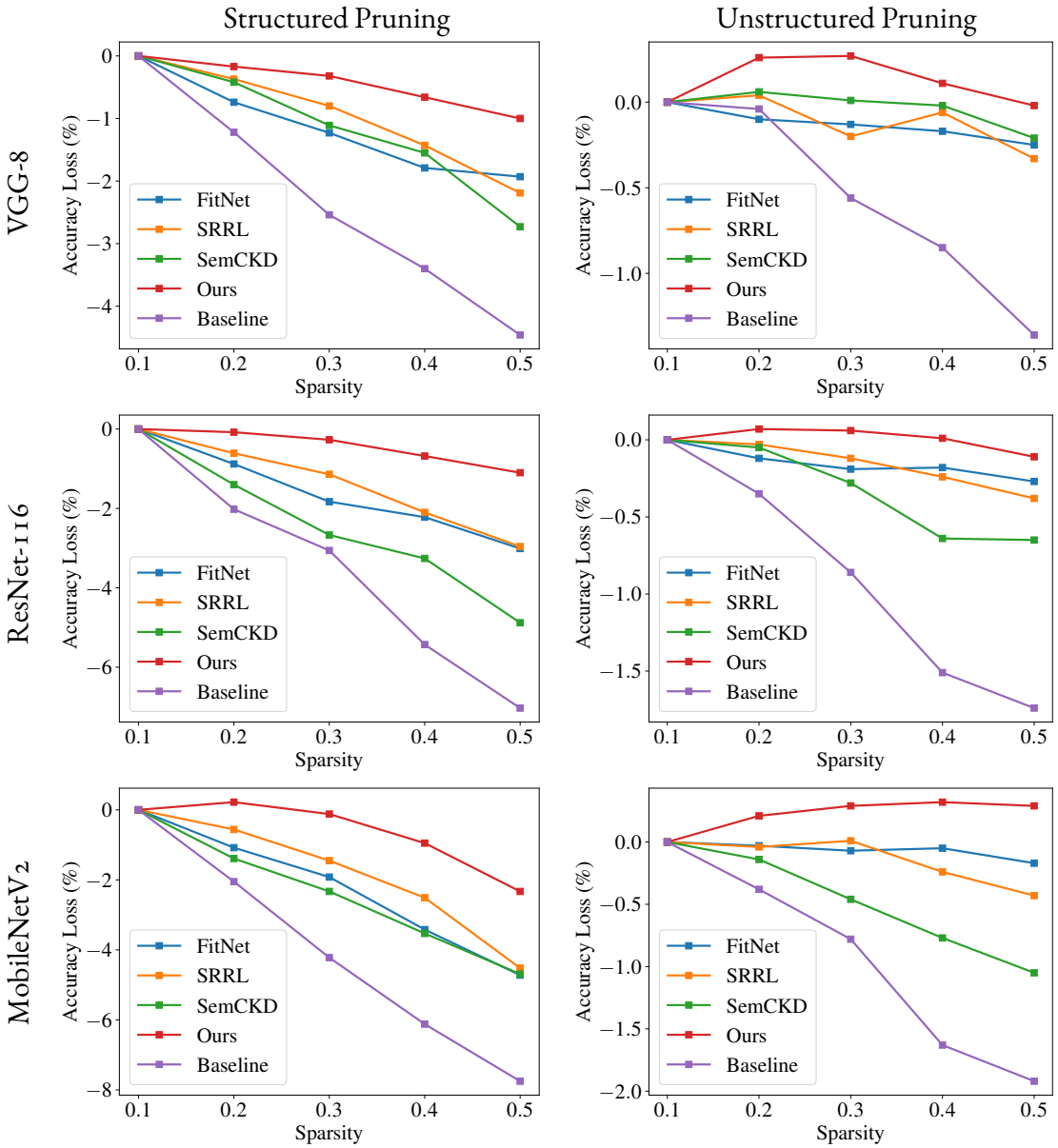


Figure 5.5: Quantitative results of our sparse self-distillation method for image classification on CIFAR-100. The baseline pruning and fine-tuning approach (without KD) is consistently outperformed by integrating KD in the loop. Moreover, our TeaFeD regularization loss provides the lowest accuracy loss across the whole sparsity spectrum [45].

6
Conclusion

In this thesis, we focused on several 3D perception tasks, with the aim of improving the efficiency of both geometry-based and learning-based processing pipelines. The main goal was to develop efficient algorithms to analyze the ever-increasing amount of 3D data from mobile devices, including smartphones, low-cost scanning sensors and autonomous vehicles. We presented several algorithmic advances to push the state-of-the-art in multi-view 3D reconstruction, novel view synthesis and point cloud upsampling, as well as neural compression approaches for the deployment on constrained hardware.

In Chapter 2, we proposed a scalable and efficient multi-view 3D reconstruction system for self-driving cars. We first revisited the classical geometric pipeline by introducing sparse priors from visual SLAM, locally optimizing the depth-normal consistency and globally regularizing the resulting geometry based on estimated confidence. Moreover, we enabled city-scale 3D reconstruction with a view clustering algorithm that builds a set of partially overlapping clusters with shared visibility over the vehicle trajectory, followed by a view selection step that computes the optimal subset of views to reconstruct a local 3D model. Our procedure has a significantly better asymptotic scaling law with respect to concurrent approaches and allows for massive hierarchical parallelization.

In Chapter 3, we improved the state-of-the-art neural radiance fields formulation for novel view synthesis. Specifically, we presented KeyNeRF to select informative samples at training time and speed up the learning process. Given a limited computational budget, we select the optimal camera rays in both 2D and 3D space, thus converging faster to high-quality renderings.

Then, we also designed a framework, called MVG-NeRF to supervise the implicit volumetric field with explicit 3D information from multi-view geometry, in order to generate cleaner and smoother 3D shapes. Extensive evaluation on both synthetic and real-world data confirm the effectiveness of our methods against related work.

In Chapter 4, we introduced a novel approach for point cloud upsampling with arbitrary scaling factors. Our intuition was to decouple the upsampling process in two steps: the sparse input is firstly mapped to an intermediate probabilistic representation, which is then sampled arbitrarily, and a Transformer network learns to map each sample back to the surface. The predictions are further improved by an attention-based residual refinement module, which allows to achieve state-of-the-art results on different benchmarks. This method was the first in the literature to allow 3D upsampling with arbitrary ratios, while not requiring ground truth meshes at training time, which are available only for synthetic data.

In Chapter 5, we showed how to efficiently deploy neural models on target hardware, while keeping their accuracy unharmed. In knowledge distillation, we showed that regularizing the student features with the teacher intermediate representations allows to learn more robust latents, which in turn leads to higher accuracy on downstream tasks. Furthermore, we also introduced a novel neural compression strategy that effectively couples distillation with network pruning, in order to improve the sparsity-accuracy tradeoff at inference time.

To conclude the thesis, we would like to give some insights about open challenges and future work. An interesting research direction would be to leverage the proposed methods for novel view synthesis (Chapter 3) and point cloud upsampling (Chapter 4) into the multi-view 3D reconstruction pipeline of Chapter 2. A possible way would be to combine KeyNeRF and MVGNeRF in a single strong baseline, that can be used to model dynamic objects, reflective materials and occluded areas, which are notoriously difficult to handle with classical geometric methods. Moreover, raw point clouds from sparse radars and LiDARs could be upsampled at arbitrary resolution and provided as priors to either augment or complete the resulting 3D model. Finally, it might be worth to extend the neural compression approaches in Chapter 5 to support non-convolutional architectures, such that the NeRF MLPs and the upsampling Transformer can be efficiently deployed, as well.

References

[1] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler, and V. Koltun, "Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2020.

[2] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 30, no. 2, pp. 328–341, 2007.

[3] J. L. Schonberger and J.-M. Frahm, "Structure-from-motion revisited," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4104–4113.

[4] C. Godard, O. Mac Aodha, M. Firman, and G. J. Brostow, "Digging into self-supervised monocular depth prediction," October 2019.

[5] W. Dong, Y. Lao, M. Kaess, and V. Koltun, "Ash: A modern framework for parallel spatial hashing in 3d perception," *PAMI*, 2022.

[6] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the fourth Eurographics symposium on Geometry processing*, vol. 7, 2006, p. 0.

[7] M. Kazhdan and H. Hoppe, "Screened poisson surface reconstruction," *ACM Transactions on Graphics (ToG)*, vol. 32, no. 3, pp. 1–13, 2013.

[8] M. Pharr, W. Jakob, and G. Humphreys, *Physically based rendering: From theory to implementation*. MIT Press, 2023.

[9] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: Learning 3d reconstruction in function space," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4460–4470.

[10] S. Peng, M. Niemeyer, L. Mescheder, M. Pollefeys, and A. Geiger, "Convolutional occupancy networks," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*. Springer, 2020, pp. 523–540.

[11] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *European conference on computer vision*. Springer, 2020, pp. 405–421.

[12] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *Seminal graphics: pioneering efforts that shaped the field*, 1998, pp. 347–353.

[13] Q. Xu and W. Tao, "Multi-scale geometric consistency guided multi-view stereo," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[14] M. Orsingher, P. Zani, P. Medici, and M. Bertozzi, "Revisiting patchmatch multi-view stereo for urban 3d reconstruction," in *2022 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2022, pp. 190–196.

[15] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[16] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, 2017.

[17] I. Misra, R. Girdhar, and A. Joulin, "An end-to-end transformer model for 3d object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 2906–2917.

[18] M. Orsingher, D. Anthony, P. Zani, P. Medici, and M. Bertozzi, "Informative rays selection for few-shot neural radiance fields," *Under Review*, 2024.

[19] M. Orsingher, P. Zani, P. Medici, and M. Bertozzi, "Learning neural radiance fields from multi-view geometry," *European Conference on Computer Vision, Learning to Generate 3D Shapes and Scenes Workshop*, 2022.

[20] S. Mohapatra, S. Yogamani, H. Gotzig, S. Milz, and P. Mader, "Bevdetnet: bird's eye view lidar point cloud based real-time 3d object detection for autonomous driving," in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2021, pp. 2809–2815.

[21] X. Chen, I. Vizzo, T. Läbe, J. Behley, and C. Stachniss, "Range image-based lidar localization for autonomous vehicles," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 5802–5808.

[22] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 945–953.

[23] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[24] J. L. Schönberger, "Robust Methods for Accurate and Efficient 3D Modeling from Unstructured Imagery," Ph.D. dissertation, ETH Zürich, 2018.

[25] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[26] H. C. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," *Nature*, vol. 293, no. 5828, pp. 133–135, 1981.

[27] D. Nistér, "An efficient solution to the five-point relative pose problem," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 6, pp. 756–770, 2004.

[28] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2019.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[31] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," 2022. [Online]. Available: https://arxiv.org/abs/2212.04356

[32] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*, 2020, https://d2l.ai.

[33] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[34] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[35] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*. Springer, 2014, pp. 818–833.

[36] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[38] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *ICLR*, 2021.

[39] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun, "Point transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 16 259–16 268.

[40] N. Engel, V. Belagiannis, and K. Dietmayer, "Point transformer," *IEEE Access*, vol. 9, pp. 134 826–134 840, 2021.

[41] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[42] Q. Xu and W. Tao, "Planar prior assisted patchmatch multi-view stereo," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[43] M. Orsingher, P. Zani, P. Medici, and M. Bertozzi, "Efficient view clustering and selection for city-scale 3d reconstruction," in *Image Analysis and Processing–ICIAP 2022: 21st International Conference, Lecce, Italy, May 23–27, 2022, Proceedings, Part II*. Springer, 2022, pp. 114–124.

[44] A. Dell'Eva, M. Orsingher, and M. Bertozzi, "Arbitrary point cloud upsampling with spherical mixture of gaussians," in *2022 International Conference on 3D Vision (3DV)*. IEEE, 2022, pp. 465–474.

[45] A. Dell'Eva, M. Orsingher, Y.-M. Lee, and M. Bertozzi, "Teacher features-driven regularization for knowledge distillation," *Currently under review at IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[46] J.-M. Frahm, P. Fite-Georgel, D. Gallup, T. Johnson, R. Raguram, C. Wu, Y.-H. Jen, E. Dunn, B. Clipp, S. Lazebnik *et al.*, "Building rome on a cloudless day," in *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11*. Springer, 2010, pp. 368–381.

[47] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. M. Seitz, and R. Szeliski, "Building rome in a day," *Communications of the ACM*, vol. 54, no. 10, pp. 105–112, 2011.

[48] Y. Furukawa, B. Curless, S. M. Seitz, and R. Szeliski, "Towards internet-scale multi-view stereo," in *2010 IEEE computer society conference on computer vision and pattern recognition*. IEEE, 2010, pp. 1434–1441.

[49] A. Akbarzadeh, J.-M. Frahm, P. Mordohai, B. Clipp, C. Engels, D. Gallup, P. Merrell, M. Phelps, S. Sinha, B. Talton *et al.*, "Towards urban 3d reconstruction from video," in *Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)*. IEEE, 2006, pp. 1–8.

[50] F. Wimbauer, N. Yang, L. von Stumberg, N. Zeller, and D. Cremers, "MonoRec: Semi-supervised dense reconstruction in dynamic environments from a single moving camera," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[51] J. L. Schönberger, E. Zheng, J.-M. Frahm, and M. Pollefeys, "Pixelwise view selection for unstructured multi-view stereo," in *European Conference on Computer Vision*. Springer, 2016.

[52] M. Orsingher, P. Zani, P. Medici, and M. Bertozzi, "Revisiting patchmatch multi-view stereo for urban 3d reconstruction," in *2022 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2022, pp. 190–196.

[53] ——, "Efficient view clustering and selection for city-scale 3d reconstruction," in *International Conference on Image Analysis and Processing*. Springer, 2022, pp. 114–124.

[54] L. Koestler, N. Yang, N. Zeller, and D. Cremers, "Tandem: Tracking and dense mapping in real-time using deep multi-view stereo," in *Conference on Robot Learning*. PMLR, 2022, pp. 34–45.

[55] Z. Teed and J. Deng, "Deepv2d: Video to depth with differentiable structure from motion," *arXiv preprint arXiv:1812.04605*, 2018.

[56] J. Sun, Y. Xie, L. Chen, X. Zhou, and H. Bao, "NeuralRecon: Real-time coherent 3D reconstruction from monocular video," *CVPR*, 2021.

[57] D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superpoint: Self-supervised interest point detection and description," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2018, pp. 224–236.

[58] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, "SuperGlue: Learning feature matching with graph neural networks," in *CVPR*, 2020. [Online]. Available: https://arxiv.org/abs/1911.11763

[59] Y. Yao, Z. Luo, S. Li, T. Fang, and L. Quan, "Mvsnet: Depth inference for unstructured multi-view stereo," *European Conference on Computer Vision (ECCV)*, 2018.

[60] T. Schöps, J. L. Schönberger, S. Galliani, T. Sattler, K. Schindler, M. Pollefeys, and A. Geiger, "A multi-view stereo benchmark with high-resolution images and multi-camera videos," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[61] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun, "Tanks and temples: Benchmarking large-scale scene reconstruction," *ACM Transactions on Graphics*, vol. 36, no. 4, 2017.

[62] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the seventh IEEE international conference on computer vision*, vol. 2. Ieee, 1999, pp. 1150–1157.

[63] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*. Ieee, 2011, pp. 2564–2571.

[64] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*. Springer, 2006, pp. 404–417.

[65] Z. Luo, T. Shen, L. Zhou, S. Zhu, R. Zhang, Y. Yao, T. Fang, and L. Quan, "Geodesc: Learning local descriptors by integrating geometry constraints," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 168–183.

[66] M. Tyszkiewicz, P. Fua, and E. Trulls, "Disk: Learning local features with policy gradient," *Advances in Neural Information Processing Systems*, vol. 33, pp. 14 254–14 265, 2020.

[67] Y. Jin, D. Mishkin, A. Mishchuk, J. Matas, P. Fua, K. M. Yi, and E. Trulls, "Image Matching across Wide Baselines: From Paper to Practice," *International Journal of Computer Vision*, 2020.

[68] J. Heinly, J. L. Schonberger, E. Dunn, and J.-M. Frahm, "Reconstructing the world* in six days*(as captured by the yahoo 100 million image dataset)," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3287–3295.

[69] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[70] P. H. Torr, "An assessment of information criteria for motion model selection," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 1997, pp. 47–52.

[71] R. Gherardi, M. Farenzena, and A. Fusiello, "Improving the efficiency of hierarchical structure-and-motion," in *2010 IEEE computer society conference on computer vision and pattern recognition*. IEEE, 2010, pp. 1594–1600.

[72] D. Crandall, A. Owens, N. Snavely, and D. Huttenlocher, "Discrete-continuous optimization for large-scale structure from motion," in *CVPR 2011*. IEEE, 2011, pp. 3001–3008.

[73] P. Moulon, P. Monasse, and R. Marlet, "Global fusion of relative motions for robust, accurate and scalable structure from motion," in *Proceedings of the IEEE international conference on computer vision*, 2013, pp. 3248–3255.

[74] C. Sweeney, T. Sattler, T. Hollerer, M. Turk, and M. Pollefeys, "Optimizing the viewing graph for structure-from-motion," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 801–809.

[75] K. Wilson and N. Snavely, "Robust global translations with 1dsfm," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part III 13*. Springer, 2014, pp. 61–75.

[76] J. L. Schonberger and J.-M. Frahm, "Structure-from-motion revisited," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4104–4113.

[77] C. Wu, "Towards linear-time incremental structure from motion," in *2013 International Conference on 3D Vision-3DV 2013*. IEEE, 2013, pp. 127–134.

[78] C. Beder and R. Steffen, "Determining an initial image pair for fixing the scale of a 3d reconstruction from an image sequence," in *Joint Pattern Recognition Symposium*. Springer, 2006, pp. 657–666.

[79] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Epnp: An accurate o(n) solution to the pnp problem," *International journal of computer vision*, vol. 81, pp. 155–166, 2009.

[80] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of applied mathematics*, vol. 2, no. 2, pp. 164–168, 1944.

[81] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[82] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[83] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[84] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE robotics & automation magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[85] D. Gálvez-López and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012.

[86] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[87] Q. Xu and W. Tao, "Multi-view stereo with asymmetric checkerboard propagation and multi-hypothesis joint view selection," *arXiv preprint arXiv:1805.07920*, 2018.

[88] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A comparison and evaluation of multi-view stereo reconstruction algorithms," in *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, vol. 1. IEEE, 2006, pp. 519–528.

[89] S. Galliani, K. Lasinger, and K. Schindler, "Massively parallel multiview stereopsis by surface normal diffusion," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[90] E. K. Stathopoulou and F. Remondino, "Multi view stereo with semantic priors," *arXiv preprint arXiv:2007.02295*, 2020.

[91] J. Huang, Z. Gojcic, M. Atzmon, O. Litany, S. Fidler, and F. Williams, "Neural kernel surface reconstruction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 4369–4379.

[92] A. Boulch and R. Marlet, "Poco: Point convolution for surface reconstruction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 6302–6314.

[93] T. Takikawa, J. Litalien, K. Yin, K. Kreis, C. Loop, D. Nowrouzezahrai, A. Jacobson, M. McGuire, and S. Fidler, "Neural geometric level of detail: Real-time rendering with implicit 3D shapes," 2021.

[94] Z. Yang, Y. Chen, J. Wang, S. Manivasagam, W.-C. Ma, A. J. Yang, and R. Urtasun, "Unisim: A neural closed-loop sensor simulator," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 1389–1399.

[95] S. Suo, K. Wong, J. Xu, J. Tu, A. Cui, S. Casas, and R. Urtasun, "Mixsim: A hierarchical framework for mixed reality traffic simulation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 9622–9631.

[96] S. Tan, K. Wong, S. Wang, S. Manivasagam, M. Ren, and R. Urtasun, "Scenegen: Learning to generate realistic traffic scenes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 892–901.

[97] A. Kuhn, C. Sormann, M. Rossi, O. Erdler, and F. Fraundorfer, "Deepc-mvs: Deep confidence prediction for multi-view stereo reconstruction," in *2020 International Conference on 3D Vision (3DV)*, 2020.

[98] M. Rossi, M. E. Gheche, A. Kuhn, and P. Frossard, "Joint graph-based depth refinement and normal estimation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[99] N. Yang, R. Wang, J. Stueckler, and D. Cremers, "Deep virtual stereo odometry: Leveraging deep depth prediction for monocular direct sparse odometry," in *European Conference on Computer Vision (ECCV)*, 2018.

[100] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[101] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *Advances in Neural Information Processing Systems*, 2014.

[102] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao, "Deep Ordinal Regression Network for Monocular Depth Estimation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[103] V. Guizilini, R. Ambrus, S. Pillai, A. Raventos, and A. Gaidon, "3d packing for self-supervised monocular depth estimation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[104] Z. Yang, P. Wang, Y. Wang, W. Xu, and R. Nevatia, "Lego: Learning edge with geometry all at once by watching videos," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[105] B. Li, C. Shen, Y. Dai, A. van den Hengel, and M. He, "Depth and surface normal estimation from monocular images using regression on deep features and hierarchical crfs," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[106] M. Lv, D. Tu, X. Tang, Y. Liu, and S. Shen, "Semantically guided multi-view stereo for dense 3d road mapping," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[107] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele, "Joint bilateral upsampling," *ACM Transactions on Graphics (ToG)*, 2007.

[108] Q. Xu, W. Kong, W. Tao, and M. Pollefeys, "Multi-scale geometric consistency guided and planar prior assisted multi-view stereo," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 4, pp. 4945–4963, 2022.

[109] Z. Xu, Y. Liu, X. Shi, Y. Wang, and Y. Zheng, "Marmvs: Matching ambiguity reduced multiple view stereo for efficient large scale scene reconstruction," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[110] H. Zhan, C. S. Weerasekera, R. Garg, and I. Reid, "Self-supervised learning for single view depth and surface normal estimation," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019.

[111] Z. Yang, P. Wang, W. Xu, L. Zhao, and R. Nevatia, "Unsupervised learning of geometry from videos with edge-aware depth-normal consistency," in *32nd AAAI conference on artificial intelligence*, 2018.

[112] J. Watson, O. M. Aodha, V. Prisacariu, G. Brostow, and M. Firman, "The Temporal Opportunist: Self-Supervised Multi-Frame Monocular Depth," in *Computer Vision and Pattern Recognition (CVPR)*, 2021.

[113] J. H. Lee, M.-K. Han, D. W. Ko, and I. H. Suh, "From big to small: Multi-scale local planar guidance for monocular depth estimation," *arXiv preprint arXiv:1907.10326*, 2019.

[114] P.-H. Huang, K. Matzen, J. Kopf, N. Ahuja, and J.-B. Huang, "Deepmvs: Learning multi-view stereopsis," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[115] H. Zhou, B. Ummenhofer, and T. Brox, "Deeptam: Deep tracking and mapping," in *European Conference on Computer Vision (ECCV)*, 2018.

[116] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *International Journal of Robotics Research*, 2013.

[117] J. Uhrig, N. Schneider, L. Schneider, U. Franke, T. Brox, and A. Geiger, "Sparsity invariant cnns," in *2017 international conference on 3D Vision (3DV)*, 2017.

[118] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proceedings of the European conference on computer vision (ECCV)*, 2018.

[119] V. Guizilini, R. Ambrus, S. Pillai, A. Raventos, and A. Gaidon, "3d packing for self-supervised monocular depth estimation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[120] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," *arXiv preprint arXiv:1903.11027*, 2019.

[121] R. Zhang, S. Li, T. Fang, S. Zhu, and L. Quan, "Joint camera clustering and surface segmentation for large-scale multi-view stereo," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2084–2092.

[122] A. Ladikos, S. Ilic, and N. Navab, "Spectral camera clustering," in *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*. IEEE, 2009, pp. 2080–2086.

[123] M. Mauro, H. Riemenschneider, L. Van Gool, and R. Leonardi, "Overlapping camera clustering through dominant sets for scalable 3d reconstruction," *Proceedings BMVC 2013*, vol. 2013, pp. 1–11, 2013.

[124] M. Mauro, H. Riemenschneider, A. Signoroni, R. Leonardi, and L. Van Gool, "An integer linear programming model for view selection on overlapping camera clusters," in *2014 2nd International Conference on 3D Vision*, vol. 1. IEEE, 2014, pp. 464–471.

[125] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[126] L. Perron and V. Furnon, "Or-tools," Google. [Online]. Available: https://developers.google.com/optimization/

[127] E. H. Adelson, J. R. Bergen *et al.*, "The plenoptic function and the elements of early vision," *Computational models of visual processing*, vol. 1, no. 2, pp. 3–20, 1991.

[128] N. Max, "Optical models for direct volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, no. 2, pp. 99–108, 1995.

[129] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville, "On the spectral bias of neural networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 5301–5310.

[130] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng, "Fourier features let networks learn high frequency functions in low dimensional domains," *NeurIPS*, 2020.

[131] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, and W. Wang, "Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction," *NeurIPS*, 2021.

[132] L. Yariv, J. Gu, Y. Kasten, and Y. Lipman, "Volume rendering of neural implicit surfaces," in *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[133] M. Oechsle, S. Peng, and A. Geiger, "Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction," in *International Conference on Computer Vision (ICCV)*, 2021.

[134] N. Snavely, S. M. Seitz, and R. Szeliski, "Photo tourism: Exploring photo collections in 3d," in *SIGGRAPH Conference Proceedings*. New York, NY, USA: ACM Press, 2006, pp. 835–846.

[135] R. Martin-Brualla, N. Radwan, M. S. M. Sajjadi, J. T. Barron, A. Dosovitskiy, and D. Duckworth, "NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections," in *CVPR*, 2021.

[136] L. Liu, J. Gu, K. Z. Lin, T.-S. Chua, and C. Theobalt, "Neural sparse voxel fields," *NeurIPS*, 2020.

[137] A. Chen, Z. Xu, A. Geiger, J. Yu, and H. Su, "Tensorf: Tensorial radiance fields," in *European Conference on Computer Vision (ECCV)*, 2022.

[138] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Transactions on Graphics (ToG)*, vol. 41, no. 4, pp. 1–15, 2022.

[139] P. Bojanowski, A. Joulin, D. Lopez-Paz, and A. Szlam, "Optimizing the latent space of generative networks," *arXiv preprint arXiv:1707.05776*, 2017.

[140] C.-H. Lin, W.-C. Ma, A. Torralba, and S. Lucey, "Barf: Bundle-adjusting neural radiance fields," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5741–5751.

[141] Z. Wang, S. Wu, W. Xie, M. Chen, and V. A. Prisacariu, "NeRF−−: Neural radiance fields without known camera parameters," *arXiv preprint arXiv:2102.07064*, 2021.

[142] Y. Jeong, S. Ahn, C. Choy, A. Anandkumar, M. Cho, and J. Park, "Self-calibrating neural radiance fields," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5846–5854.

[143] M. Tancik, V. Casser, X. Yan, S. Pradhan, B. Mildenhall, P. Srinivasan, J. T. Barron, and H. Kretzschmar, "Block-NeRF: Scalable large scene neural view synthesis," *arXiv*, 2022.

[144] M. Tancik, E. Weber, E. Ng, R. Li, B. Yi, J. Kerr, T. Wang, A. Kristoffersen, J. Austin, K. Salahi, A. Ahuja, D. McAllister, and A. Kanazawa, "Nerfstudio: A modular framework for neural radiance field development," in *ACM SIGGRAPH 2023 Conference Proceedings*, ser. SIGGRAPH '23, 2023.

[145] E. Sucar, S. Liu, J. Ortiz, and A. Davison, "iMAP: Implicit mapping and positioning in real-time," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.

[146] Z. Zhu, S. Peng, V. Larsson, W. Xu, H. Bao, Z. Cui, M. R. Oswald, and M. Pollefeys, "Nice-slam: Neural implicit scalable encoding for slam," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022.

[147] A. Rosinol, J. J. Leonard, and L. Carlone, "Nerf-slam: Real-time dense monocular slam with neural radiance fields," *arXiv preprint arXiv:2210.13641*, 2022.

[148] K. Rematas, A. Liu, P. P. Srinivasan, J. T. Barron, A. Tagliasacchi, T. Funkhouser, and V. Ferrari, "Urban radiance fields," *CVPR*, 2022.

[149] C.-Y. Weng, B. Curless, P. P. Srinivasan, J. T. Barron, and I. Kemelmacher-Shlizerman, "HumanNeRF: Free-viewpoint rendering of moving people from monocular video," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 16 210–16 220.

[150] K. Park, U. Sinha, J. T. Barron, S. Bouaziz, D. B. Goldman, S. M. Seitz, and R. Martin-Brualla, "Nerfies: Deformable neural radiance fields," *ICCV*, 2021.

[151] Y. Hong, B. Peng, H. Xiao, L. Liu, and J. Zhang, "Headnerf: A real-time nerf-based parametric head model," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[152] R. Li, J. Tanke, M. Vo, M. Zollhofer, J. Gall, A. Kanazawa, and C. Lassner, "Tava: Template-free animatable volumetric actors," 2022.

[153] M. Niemeyer and A. Geiger, "Giraffe: Representing scenes as compositional generative neural feature fields," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[154] C. Wang, M. Chai, M. He, D. Chen, and J. Liao, "Clip-nerf: Text-and-image driven manipulation of neural radiance fields," *arXiv preprint arXiv:2112.05139*, 2021.

[155] A. Kundu, K. Genova, X. Yin, A. Fathi, C. Pantofaru, L. Guibas, A. Tagliasacchi, F. Dellaert, and T. Funkhouser, "Panoptic Neural Fields: A Semantic Object-Aware Neural Scene Representation," in *CVPR*, 2022.

[156] C. Wang, X. Wu, Y.-C. Guo, S.-H. Zhang, Y.-W. Tai, and S.-M. Hu, "Nerf-sr: High-quality neural radiance fields using super-sampling," *arXiv*, 2021.

[157] "Luma labs ai." [Online]. Available: https://lumalabs.ai/

[158] K. Gao, Y. Gao, H. He, D. Lu, L. Xu, and J. Li, "Nerf: Neural radiance field in 3d vision, a comprehensive review," *arXiv preprint arXiv:2210.00379*, 2022.

[159] M. Kim, S. Seo, and B. Han, "Infonerf: Ray entropy minimization for few-shot neural volume rendering," in *CVPR*, 2022.

[160] J. Yang, M. Pavone, and Y. Wang, "Freenerf: Improving few-shot neural rendering with free frequency regularization," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[161] A. Jain, M. Tancik, and P. Abbeel, "Putting nerf on a diet: Semantically consistent few-shot view synthesis," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 5885–5894.

[162] K. Deng, A. Liu, J.-Y. Zhu, and D. Ramanan, "Depth-supervised NeRF: Fewer views and faster training for free," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022.

[163] B. Roessle, J. T. Barron, B. Mildenhall, P. P. Srinivasan, and M. Nießner, "Dense depth priors for neural radiance fields from sparse input views," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022.

[164] J. Wynn and D. Turmukhambetov, "Diffusionerf: Regularizing neural radiance fields with denoising diffusion models," in *ArXiV*, 2023.

[165] M. Niemeyer, J. T. Barron, B. Mildenhall, M. S. M. Sajjadi, A. Geiger, and N. Radwan, "Regnerf: Regularizing neural radiance fields for view synthesis from sparse inputs," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[166] S. Seo, D. Han, Y. Chang, and N. Kwak, "Mixnerf: Modeling a ray with mixture density for novel view synthesis from sparse inputs," in *ArXiV*, 2023.

[167] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763.

[168] X. Pan, Z. Lai, S. Song, and G. Huang, "Activenerf: Learning where to see with uncertainty estimation," in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXIII*. Springer, 2022, pp. 230–246.

[169] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer,

M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[170] S. Ramasinghe, L. E. MacDonald, and S. Lucey, "On the frequency-bias of coordinate-mlps," in *Advances in Neural Information Processing Systems*, 2022.

[171] J. Reizenstein, R. Shapovalov, P. Henzler, L. Sbordone, P. Labatut, and D. Novotny, "Common objects in 3d: Large-scale learning and evaluation of real-life 3d category reconstruction," in *International Conference on Computer Vision*, 2021.

[172] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 8024–8035. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

[173] L. Yen-Chen, "Nerf-pytorch," https://github.com/yenchenlin/nerf-pytorch/, 2020.

[174] S. Van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. Yu, "scikit-image: image processing in python," *PeerJ*, vol. 2, p. e453, 2014.

[175] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan, "Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields," *ICCV*, 2021.

[176] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[177] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 586–595.

[178] K. Zhang, G. Riegler, N. Snavely, and V. Koltun, "Nerf++: Analyzing and improving neural radiance fields," *arXiv preprint arXiv:2010.07492*, 2020.

[179] Y. Wei, S. Liu, Y. Rao, W. Zhao, J. Lu, and J. Zhou, "Nerfingmvs: Guided optimization of neural radiance fields for indoor multi-view stereo," in *ICCV*, 2021.

[180] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[181] Z. Yu, S. Peng, M. Niemeyer, T. Sattler, and A. Geiger, "Monosdf: Exploring monocular geometric cues for neural implicit surface reconstruction," *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[182] A. Eftekhar, A. Sax, J. Malik, and A. Zamir, "Omnidata: A scalable pipeline for making multi-task mid-level vision datasets from 3d scans," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10 786–10 796.

[183] V. Tschernezki, I. Laina, D. Larlus, and A. Vedaldi, "Neural Feature Fusion Fields: 3D distillation of self-supervised 2D image representations," in *Proceedings of the International Conference on 3D Vision (3DV)*, 2022.

[184] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[185] Q. Fu, Q. Xu, Y.-S. Ong, and W. Tao, "Geo-neus: Geometry-consistent neural implicit surfaces learning for multi-view reconstruction," *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[186] F. Darmon, B. Bascle, J. Devaux, P. Monasse, and M. Aubry, "Improving neural implicit surfaces geometry with patch warping," 2022.

[187] Y. Zhao and T. Guo, "Pointar: Efficient lighting estimation for mobile augmented reality," in *European Conference on Computer Vision*. Springer, 2020, pp. 678–693.

[188] D. Cattaneo, M. Vaghi, and A. Valada, "Lcdnet: Deep loop closure detection and point cloud registration for lidar slam," *IEEE Transactions on Robotics*, 2022.

[189] C. Luo, X. Yang, and A. Yuille, "Self-supervised pillar motion learning for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 3183–3192.

[190] K. Yabuuchi, D. R. Wong, T. Ishita, Y. Kitsukawa, and S. Kato, "Visual localization for autonomous driving using pre-built point cloud maps," in *2021 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2021, pp. 913–919.

[191] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng, "Pu-net: Point cloud upsampling network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2790–2799.

[192] R. Li, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng, "Pu-gan: a point cloud upsampling adversarial network," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 7203–7212.

[193] G. Qian, A. Abualshour, G. Li, A. Thabet, and B. Ghanem, "Pu-gcn: Point cloud upsampling using graph convolutional networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 11 683–11 692.

[194] R. Li, X. Li, P.-A. Heng, and C.-W. Fu, "Point cloud upsampling via disentangled refinement," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 344–353.

[195] H. Huang, D. Li, H. Zhang, U. Ascher, and D. Cohen-Or, "Consolidation of unorganized point clouds for surface reconstruction," *ACM transactions on graphics (TOG)*, vol. 28, no. 5, pp. 1–7, 2009.

[196] Y. Lipman, D. Cohen-Or, D. Levin, and H. Tal-Ezer, "Parameterization-free projection for geometry reconstruction," *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3, pp. 22–es, 2007.

[197] H. Huang, S. Wu, M. Gong, D. Cohen-Or, U. Ascher, and H. Zhang, "Edge-aware point set resampling," *ACM transactions on graphics (TOG)*, vol. 32, no. 1, pp. 1–12, 2013.

[198] Y. Qian, J. Hou, S. Kwong, and Y. He, "Deep magnification-flexible upsampling over 3d point clouds," *IEEE Transactions on Image Processing*, vol. 30, pp. 8354–8367, 2021.

[199] L. Luo, L. Tang, W. Zhou, S. Wang, and Z.-X. Yang, "Pu-eva: An edge-vector based approximation solution for flexible-scale point cloud upsampling," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 16 208–16 217.

[200] S. Ye, D. Chen, S. Han, Z. Wan, and J. Liao, "Meta-pu: An arbitrary-scale upsampling network for point cloud," *IEEE Transactions on Visualization and Computer Graphics*, 2021.

[201] W. Feng, J. Li, H. Cai, X. Luo, and J. Zhang, "Neural points: Point cloud representation with neural fields," *arXiv preprint arXiv:2112.04148*, 2021.

[202] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry, "A papier-mâché approach to learning 3d surface generation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 216–224.

[203] R. Li, X. Li, K.-H. Hui, and C.-W. Fu, "SP-GAN:sphere-guided 3d shape generation and manipulation," *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 40, no. 4, 2021.

[204] A.-C. Cheng, X. Li, M. Sun, M.-H. Yang, and S. Liu, "Learning 3d dense correspondence via canonical point autoencoder," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[205] X. Yu, Y. Rao, Z. Wang, Z. Liu, J. Lu, and J. Zhou, "Pointr: Diverse point cloud completion with geometry-aware transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 12 498–12 507.

[206] Y. Yang, C. Feng, Y. Shen, and D. Tian, "Foldingnet: Point cloud auto-encoder via deep grid deformation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 206–215.

[207] J. Pang, D. Li, and D. Tian, "Tearingnet: Point cloud autoencoder to learn topology-friendly representations," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[208] W. Yifan, S. Wu, H. Huang, D. Cohen-Or, and O. Sorkine-Hornung, "Patch-based progressive 3d point set upsampling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5958–5967.

[209] M.-H. Guo, J.-X. Cai, Z.-N. Liu, T.-J. Mu, R. R. Martin, and S.-M. Hu, "Pct: Point cloud transformer," *Computational Visual Media*, vol. 7, no. 2, pp. 187–199, 2021.

[210] K. Mazur and V. Lempitsky, "Cloud transformers: A universal approach to point cloud processing tasks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10 715–10 724.

[211] J. Yang, Q. Zhang, B. Ni, L. Li, J. Liu, M. Zhou, and Q. Tian, "Modeling point clouds with self-attention and gumbel subset sampling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3323–3332.

[212] S. Qiu, S. Anwar, and N. Barnes, "Pu-transformer: Point cloud upsampling transformer," *arXiv preprint arXiv:2111.12242*, 2021.

[213] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *ACM Transactions on Graphics (TOG)*, 2019.

[214] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng, "Ec-net: an edge-aware point set consolidation network," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 386–402.

[215] "Visionair," http://www.infra-visionair.eu/, accessed: 2022-05-20.

[216] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," in *International Conference on Learning Representations*, 2017.

[217] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[218] M. A. Uy, Q.-H. Pham, B.-S. Hua, D. T. Nguyen, and S.-K. Yeung, "Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data," in *International Conference on Computer Vision (ICCV)*, 2019.

[219] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 11 976–11 986.

[220] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[221] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[222] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6550

[223] Z. Yang, Z. Li, M. Shao, D. Shi, Z. Yuan, and C. Yuan, "Masked generative distillation," in *European Conference on Computer Vision*. Springer, 2022, pp. 53–69.

[224] B. Zhao, Q. Cui, R. Song, Y. Qiu, and J. Liang, "Decoupled knowledge distillation," in *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, 2022, pp. 11 953–11 962.

[225] G. Fang, X. Ma, M. Song, M. B. Mi, and X. Wang, "Depgraph: Towards any structural pruning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 16 091–16 101.

[226] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," *Advances in neural information processing systems*, vol. 29, 2016.

[227] V. Sanh, T. Wolf, and A. Rush, "Movement pruning: Adaptive sparsity by fine-tuning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 20 378–20 389, 2020.

[228] Z. You, K. Yan, J. Ye, M. Ma, and P. Wang, "Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[229] L. Liu, S. Zhang, Z. Kuang, A. Zhou, J.-H. Xue, X. Wang, Y. Chen, W. Yang, Q. Liao, and W. Zhang, "Group fisher pruning for practical network compression," in *International Conference on Machine Learning*. PMLR, 2021, pp. 7021–7032.

[230] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," *arXiv preprint arXiv:2103.13630*, 2021.

[231] S. I. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh, "Improved knowledge distillation via teacher assistant," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 04, 2020, pp. 5191–5198.

[232] T. Huang, Y. Zhang, S. You, F. Wang, C. Qian, J. Cao, and C. Xu, "Masked distillation with receptive tokens," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=mWRngkvIki3

[233] Y. Tian, D. Krishnan, and P. Isola, "Contrastive representation distillation," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=SkgpBJrtvS

[234] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.

[235] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.

[236] Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu, "Deep mutual learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4320–4328.

[237] Y. Jin, J. Wang, and D. Lin, "Multi-level logit distillation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 24 276–24 285.

[238] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *International conference on machine learning*. PMLR, 2017, pp. 1321–1330.

[239] C. Pengguang, L. Shu, Z. Hengshuang, and J. Jia, "Distilling knowledge via knowledge review," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[240] J. Yang, B. Martínez, A. Bulat, and G. Tzimiropoulos, "Knowledge distillation via softmax regression representation learning," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=ZzwDy_wiWv

[241] D. Chen, J. Mei, Y. Zhang, C. Wang, Z. Wang, Y. Feng, and C. Chen, "Cross-layer distillation with semantic calibration," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021, pp. 7028–7036.

[242] Z. Guo, H. Yan, H. Li, and X. Lin, "Class attention transfer based knowledge distillation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 11 868–11 877.

[243] Z. Yang, Z. Li, X. Jiang, Y. Gong, Z. Yuan, D. Zhao, and C. Yuan, "Focal and global knowledge distillation for detectors," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 4643–4652.

[244] C. Shu, Y. Liu, J. Gao, Z. Yan, and C. Shen, "Channel-wise knowledge distillation for dense prediction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5311–5320.

[245] Y. Liu, K. Chen, C. Liu, Z. Qin, Z. Luo, and J. Wang, "Structured knowledge distillation for semantic segmentation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 2604–2613.

[246] W. Cao, Y. Zhang, J. Gao, A. Cheng, K. Cheng, and J. Cheng, "Pkd: General distillation framework for object detectors via pearson correlation coefficient," *Advances in Neural Information Processing Systems*, vol. 35, pp. 15 394–15 406, 2022.

[247] B. Heo, J. Kim, S. Yun, H. Park, N. Kwak, and J. Y. Choi, "A comprehensive overhaul of feature distillation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1921–1930.

[248] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," in *ICLR*, 2017. [Online]. Available: https://arxiv.org/abs/1612.03928

[249] F. Tung and G. Mori, "Similarity-preserving knowledge distillation," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1365–1374.

[250] D. Chen, J.-P. Mei, H. Zhang, C. Wang, Y. Feng, and C. Chen, "Knowledge distillation with the reused teacher classifier," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 11 933–11 942.

[251] S. Park, J. Lee, S. Mo, and J. Shin, "Lookahead: A far-sighted alternative of magnitude-based pruning," in *ICLR*, 2020.

[252] N. Lee, T. Ajanthan, S. Gould, and P. H. S. Torr, "A signal propagation perspective for pruning neural networks at initialization," in *ICLR*, 2020.

[253] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," *Advances in neural information processing systems*, vol. 30, 2017.

[254] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *ICLR (Poster)*, 2017.

[255] X. Ding, G. Ding, Y. Guo, and J. Han, "Centripetal sgd for pruning very deep convolutional networks with complicated structure," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4943–4953.

[256] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[257] Y. Le and X. Yang, "Tiny imagenet visual recognition challenge," *CS 231N*, vol. 7, no. 7, p. 3, 2015.

[258] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[259] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proceedings of the British Machine Vision Conference 2016, BMVC*. BMVA Press, 2016.

[260] K. Chen, J. Wang, J. Pang, Y. Cao, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Xu, Z. Zhang, D. Cheng, C. Zhu, T. Cheng, Q. Zhao, B. Li, X. Lu, R. Zhu, Y. Wu, J. Dai,

J. Wang, J. Shi, W. Ouyang, C. C. Loy, and D. Lin, "MMDetection: Open mmlab detection toolbox and benchmark," *arXiv preprint arXiv:1906.07155*, 2019.

[261] T. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," in *ECCV*, ser. Lecture Notes in Computer Science, vol. 8693. Springer, 2014, pp. 740–755.

[262] Z. Shen and E. Xing, "A fast knowledge distillation framework for visual recognition," in *European Conference on Computer Vision*. Springer, 2022, pp. 673–690.

[263] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[264] Z. Yang, S. Liu, H. Hu, L. Wang, and S. Lin, "Reppoints: Point set representation for object detection," in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2019.

[265] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[266] X. Wang, T. Kong, C. Shen, Y. Jiang, and L. Li, "SOLO: Segmenting objects by locations," in *Proc. Eur. Conf. Computer Vision (ECCV)*, 2020.

[267] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[268] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.

[269] M. Contributors, "MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark," https://github.com/open-mmlab/mmsegmentation, 2020.

[270] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2881–2890.

[271] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017.

[272] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 215–223.