MonoNeRD: NeRF-like Representations for Monocular 3D Object Detection

Junkai Xu$^{1,2,*}$ Liang Peng$^{1,2,*}$ Haoran Chen$^{1,2,*}$ Hao Li$^2$
Wei Qian$^2$ Ke Li$^4$ Wenxiao Wang$^3$ Deng Cai$^{1,2}$

$^1$State Key Lab of CAD & CG, Zhejiang University $^2$FABU Inc. $^3$School of Software Technology, Zhejiang University $^4$Fullong Inc
{xujunkai, pengliang, haorancheng}@zju.edu.cn

Abstract

In the field of monocular 3D detection, it is common practice to utilize scene geometric clues to enhance the detector’s performance. However, many existing works adopt these clues explicitly such as estimating a depth map and back-projecting it into 3D space. This explicit methodology induces sparsity in 3D representations due to the increased dimensionality from 2D to 3D, and leads to substantial information loss, especially for distant and occluded objects. To alleviate this issue, we propose MonoNeRD, a novel detection framework that can infer dense 3D geometry and occupancy. Specifically, we model scenes with Signed Distance Functions (SDF), facilitating the production of dense 3D representations. We treat these representations as Neural Radiance Fields (NeRF) and then employ volume rendering to recover RGB images and depth maps. To the best of our knowledge, this work is the first to introduce volume rendering for M3D, and demonstrates the potential of implicit reconstruction for image-based 3D perception. Extensive experiments conducted on the KITTI-3D benchmark and Waymo Open Dataset demonstrate the effectiveness of MonoNeRD. Codes are available at https://github.com/cskkxjk/MonoNeRD.

1. Introduction

Monocular 3D detection (M3D) is an active research topic in the computer vision community due to its convenience, low cost and wide range of applications, including autonomous driving, robotic navigation and more. The key point of the task is to establish reasonable correspondences between 2D images and 3D space. Some works leverage geometrical priors to extract 3D information, such as object poses via 2D-3D constraints. These constraints usually require additional keypoint annotations [10, 21] or CAD models [5, 34]. Other works convert estimated depth maps into 3D point cloud representations (Pseudo-LiDAR) [32, 55, 60]. Depth estimates are also used to combine with image features [58, 33] or generate meaningful bird’s-eye-view (BEV) representations [45, 47, 48, 51], then produce 3D object detection results. These methods have made remarkable progress in M3D.

Unfortunately, current 3D representations transformed by explicit depths have some limitations. First, the lifted features obtained from depth estimates or pseudo-LiDAR exhibit an uneven distribution throughout the 3D space. Specifically, they have high density in close range, but the density decreases as the distance increases. (See top part of Figure 1). Second, the final detection performance heavily depends on the accuracy of the depth estimation, which remains challenging to improve. Thus such representations

Figure 1. Different intermediate 3D representations. Previous depth-based methods generate 3D representations by back-projecting estimated depth maps to 3D space, whereas our method predicts implicit 3D representations, and obtains depth estimates through volume rendering.
cannot produce dense reasonable 3D features for M3D.

Recent researches in Neural Radiance Fields (NeRF) [3, 38, 64] have shown the capability to reconstruct detailed and dense 3D scene geometry and occupancy information from posed images (images with known camera poses). Inspired by NeRF and its follow-ups [20, 25, 59], we re-formulate the intermediate 3D representations in M3D to NeRF-like 3D representations, which can produce dense reasonable 3D geometry and occupancy. To achieve this goal, we combine extracted 2D backbone features and corresponding normalized frustum 3D coordinates to construct 3D Position-aware Frustum features (Section 4.1.1). These 3D features are used to create signed distance fields and radiance fields (RGB color). The signed distance fields encode the distance to the closest surface at every location as a scalar value. This allows us to model scenes implicitly by the zero-level set of the signed distance fields. (Section 3.1). We then adopt volume rendering [35] technique to generate RGB images and depth maps from the signed distance fields and radiance fields (Section 4.1.2), supervising them by original RGB images and LiDAR points (Section 4.2). While the previous depth-based methods generate 3D representations based on predicted depth maps, our method generates depth maps based on 3D representations (Figure 1). It is worth noting that our approach is capable of generating dense 3D occupancy (i.e., volume density) without requiring explicit binary occupancy annotations for individual voxels. Experiments on KITTI [17] 3D benchmark and Waymo Open Dataset [52] show the superiority of our NeRF-like representations in monocular 3D detection.

Our main contributions are threefold:

• We present a novel detection framework, named MonoNeRD, that connects Neural Radiance Fields and monocular 3D detection. It leverages NeRF-like continuous 3D representations to enable accurate 3D perception and understanding from a single image.

• We propose to use volume rendering to directly optimize 3D representations in detection tasks. To the best of our knowledge, our work is the first to introduce volume rendering for 3D detection tasks.

• Extensive experiments on KITTI [17] 3D detection benchmark and Waymo Open Dataset [52] demonstrate the effectiveness of our method, which is competitive with previous state-of-the-art works. This research presents the potential of 3D implicit reconstruction for image-based 3D perception.

2. Related Work

2.1. Monocular 3D Object Detection

Many impressive researches [47, 9, 43] been done in monocular 3D object detection. As the essential information of depth dimension is collapsed in the 2D image, it is an ill-posed but necessary problem of finding an approach to lift the 2D information to 3D space.

Some Monocular 3D Object Detection methods lift 2D to 3D directly by incorporating the geometric relationship between the 2D image plane and 3D space. For example, many methods assume rigidity of objects, aligning 2D keypoints with their 3D counterparts [1, 2, 21, 46]. Some researchers try to build the bridge between 2D and 3D structures by template matching [5, 34]. These methods usually require additional data (e.g., 3D CAD models, object key-point annotations).

Other methods generate intermediate 3D representations to solve the lifting problem. Many approaches [60, 33, 32] convert the estimated depth map (always predicted by an off-the-shelf model) to pseudo-LiDAR point cloud representations, and then feed them to sophisticated LiDAR-based detectors. CaDDN [47] learns categorical depth distributions over pixels to lift camera images into 3D space, constructing BEV representations. Pseudo-Stereo [12] uses some sophisticated modules (e.g., stereo cost volume) to achieve better depth estimation. These methods typically involve estimating depth maps beforehand and subsequently extracting corresponding 3D representations from them. Instead, our method recovers depth maps from the 3D representations and supervises them to improve the detection performance.

2.2. Neural Implicit Representations

Representing 3D geometry with implicit functions has gained popularity in the past few years. Learning-based 3D reconstruction methods [13, 18, 36, 37, 41, 49] and their scene-level counterparts [6, 15, 23, 44] demonstrate compelling results but require 3D input and supervision. Neural Radiance Fields (NeRFs) [38] introduce a new perspective to learn continuous geometry representations from multi-view images only by volume rendering [35]. Due to the low efficiency of original MLP-based NeRF, their grid-based variants [22, 28, 61, 7] have been explored to accelerate the rendering process or save GPU memory. To circumvent the requirement of dense inputs, some works [62, 8, 14, 39] explicitly consider sparse input scenarios and show astonishing results for novel view synthesis, but these representations still need to be optimized per scene. Li et al. [25] propose MINE in which they combine Multiplane Images (MPI) and NeRF for single-image-based novel view synthesis. SceneRF [4] uses posed image sequences for training, alleviating the reliance on datasets. Wimbauer et al. [57] construct a density field to perform both depth prediction and novel view synthesis. Unlike our approach, the works mentioned primarily focus on the Novel View Synthesis task, requiring multi-view or sequenced images with camera poses as supervision, which cannot generalize to un-
seen scenes (except [25, 57]). Instead, we take advantage of volume rendering to optimize 3D feature representations and improve the performance in M3D task.

3. Preliminaries

Here we introduce some essential preliminaries for our idea. We use the same notation as [59] for the concepts of SDF and Volume density. Concerning the space limitation, here we provide a brief overview. Please refer to [38, 59, 35] for more details.

3.1. Signed Distance Function (SDF)

For a given spatial point, Signed Distance Function (SDF) is a continuous function that outputs the point’s distance to the closest surface, whose sign indicates whether the point is inside (negative) or outside (positive) of the surface.

Let \( \mathbb{R}^3 \) represent a 3D space containing some objects, \( \Omega \subset \mathbb{R}^3 \) refers to the occupied part, and \( \mathcal{M} = \partial \Omega \) be the boundary surface. Denote the \( \Omega \) indicator function by \( \mathbbm{1}_\Omega \), and the Signed Distance Function (SDF) by \( d_\Omega \),

\[
\mathbbm{1}_\Omega(x) = \begin{cases} 
1 & \text{if } x \in \Omega \\
0 & \text{if } x \notin \Omega 
\end{cases} 
\]

where \( \cdot \) is the standard Euclidean 2-norm. The underlying surface \( \mathcal{M} \) can be implicitly represented by the zero-level set of \( d_\Omega(\cdot) = 0 \).

3.2. Transform SDF to Density

The volume density \( \sigma : \mathbb{R}^3 \rightarrow \mathbb{R}_+ \) is a scalar volumetric function, where \( \sigma(x) \) can be interpreted as the probability that light is occluded at point \( x \). Previous works et al. [59, 63] suggest modeling the density \( \sigma \) using a certain transformation of a learnable Signed Distance Function (SDF), as follows:

\[
\sigma(x) = \alpha \Psi_\beta(-d_\Omega(x)) 
\]

where \( \alpha, \beta > 0 \) are learnable parameters and \( \Psi_\beta \) is the cumulative distribution function of the Laplace distribution with zero mean and \( \beta \) scale,

\[
\Psi_\beta(s) = \begin{cases} 
\frac{1}{2} \exp\left(\frac{s}{\beta}\right) & \text{if } s \leq 0 \\
1 - \frac{1}{2} \exp\left(-\frac{s}{\beta}\right) & \text{if } s > 0 
\end{cases} 
\]

As \( \beta \) approaches zero, the density \( \sigma \) converges to scaled indicator function of \( \Omega \), means \( \sigma \rightarrow \alpha \mathbbm{1}_\Omega \) for all points \( x \in \Omega \setminus \mathcal{M} \). The benefits of modeling density in this way have been discussed in [59]. We simply set \( \alpha = \beta^{-1} \) in our implementations. Figure 2 depicts an example of how density varies along SDF when a ray passes through an object.

3.3. Volume Rendering

Volume intensity (RGB color) \( c : \mathbb{R}^3 \rightarrow \mathbb{R}^3 \) is a vector volumetric function of spatial position \( x \), where \( c(x) \) represents the intensity or color at a certain point \( x \). Given color \( c \) and density \( \sigma \), the expected color \( C(r) \) of ray \( r(t) = o + td \) (where \( o \) and \( d \) are the origin and direction of the ray respectively) with near and far bounds \( t_n \) and \( t_f \) is:

\[
C(r) = \int_{t_n}^{t_f} T(t) \sigma(r(t)) c(r(t)) dt, 
\]

where \( T(t) = \exp\left(-\int_{t_n}^{t} \sigma(s) ds\right) \) (5)

where \( \sigma \) refers to the volume density. The function \( T(t) \) denotes the accumulated transmittance along the ray from \( t_n \) to \( t_f \), namely, the probability that the ray (or light) travels from \( t_n \) to \( t_f \) without being occluded by any particle. In Section 4.1.2, we describe the details to calculate volume rendering numerical integration in our setup.

4. Method

Overview. The overall framework is illustrated in Figure 3. We follow the network design in LIGA-Stereo [19]. Note that the stereo part of LIGA-Stereo [19] is not applicable for monocular 3D detection (M3D) task. We replace the intermediate 3D feature representations with the proposed
NeRF-like representations while keeping the 2D image backbone and downstream detection module unchanged.

Given an input RGB image $I \in \mathbb{R}^{H \times W \times 3}$, it is passed to a 2D image backbone for extracting image features. Then we use the 2D image features to generate the proposed NeRF-like 3D representations. Specifically, we first build the Position-aware Frustum (See Section 4.1.1), which is transformed to 3D frustum features and SDF frustum features. Such features can be used for obtaining RGB and density features to achieve rendered RGB images and depth maps via Volume Rendering (See Section 4.1.2). They are supervised by RGB loss and depth loss to guide previous features, serving as auxiliary tasks in training. Furthermore, we use grid-sampling on 3D frustum features and density frustum features to build regular 3D voxel features and corresponding density. They are used for forming 3D Voxel Features (See Section 4.1.3), which are fed into the detection modules. We elaborate on each part of our method in the following.

4.1. NeRF-like Representation

We need to clarify the difference between NeRF and the proposed NeRF-like representation. NeRF encodes a scene with a MLP and has to be trained per scene. The proposed method predicts continuous 3D geometry information from the single input RGB image. Similar ideas have been explored in researches about single-image-based novel view synthesis. Please refer to [20, 25] for details.

4.1.1 Position-aware Frustum Construction

To lift 2D backbone features to 3D space without depth, a naive manner is to simply repeat 2D features on different depth planes. However, this way does not contain any position information, causing ambiguous spatial features. To resolve this problem, we propose to generate position-aware frustum features.

Taking an RGB image as input, the 2D image backbone produces image features $F_{\text{image}} \in \mathbb{R}^{H_2d \times W_2d \times C}$, where $H_2d$ and $W_2d$ are the height and width of the 2D image feature map, and $C$ is the number of feature channels. Such 2D image features are mapped to a camera frustum along with corresponding normalized frustum 3D coordinates in a query-based manner. More specifically, given pre-defined near depth $z_n$ and far depth $z_f$, we sample $D$ planes from this depth range under equal depth intervals with random perturbation as described in [38, 25]. Each depth plane consists of frustum 3D position coordinates $p \in [u, v, z]^T \in \mathbb{R}^3$. These coordinates are normalized later. Thus we have 3D positional frustum features $F_{\text{pos}} \in \mathbb{R}^{H_3d \times W_3d \times D \times C}$, where $D$ refers to the number of depth planes and 3 denotes the normalized coordinates. To combine 3D position and original 2D image features, we use three linear layers $f_g, f_k, f_v$ to do the Query-Key-Value mapping and attention calculation (in Figure 4).

$$f_g : F_{\text{pos}} \rightarrow Q \in \mathbb{R}^{H_2d \times W_2d \times D \times C}$$
$$f_k : F_{\text{image}} \rightarrow K \in \mathbb{R}^{H_2d \times W_2d \times C}$$
$$f_v : F_{\text{image}} \rightarrow V \in \mathbb{R}^{H_2d \times W_2d \times C}$$

(6)

Therefore, the Position-Aware Frustum features can be calculated by a softmax function along the depth dimension.

$$F_p = \text{Softmax}(QK, \text{dim}=D)V$$

(7)
4.1.2 Volume Rendering within Frustum

After constructing position-aware frustum features, we try to encode such features in 3D space to obtain more informative 3D features for volume rendering. Towards this goal, we employ two 3D convolution blocks \( f_1 \) and \( f_2 \). \( f_1 \) consists of a three-layer 3D-convolution with kernel size 3 and softplus activation. It takes position-aware frustum features \( F_p \in \mathbb{R}^{H_d \times W_d \times D \times C} \) as input, outputting a same shape feature \( F' \) except the channel number, as follows:

\[
f_1 : F_p \rightarrow F' \in \mathbb{R}^{H_d \times W_d \times D \times (1+C)} \tag{8}
\]

\( i) \) SDF and density features. The first channel feature in \( F' \) is regarded as SDF feature \( F_{sdf} \in \mathbb{R}^{H_d \times W_d \times D \times 1} \). The signed distance fields \( F_{sdf} \) represent the scene geometry of the input image. For any 3D point \( [x, y, z]^T \) in frustum space, we can get its signed distance \( s(x, y, z) \in \mathbb{R}^1 \) by trilinear sampling in \( F_{sdf} \). Furthermore, scene volume density \( \sigma \) (or density frustum \( F_{density} \in \mathbb{R}^{H_d \times W_d \times D \times 1} \)) can be transformed by Equation 3:

\[
\sigma(x, y, z) = \alpha \Psi_\beta(s(x, y, z)) \quad F_{density} = \alpha \Psi_\beta(F_{sdf})
\]

The density will be used for rendering depth maps and weighing the voxel features.

\( ii) \) RGB features. The remaining \( C \) channel feature in \( F' \), called \( F'' \), is then passed to \( f_2 \), which consists of a one-layer 3D-convolution with kernel size 3 and sigmoid activation.

\[
f_2 : F'' \rightarrow F_{rgb} \in \mathbb{R}^{H_d \times W_d \times D \times 3} \tag{10}
\]

The resulting feature is used for approximating the scene radiance field \( F_{rgb} \). We can also get color intensity \( c(x, y, z) \in \mathbb{R}^3 \) by trilinear sampling in \( F_{rgb} \).

Rendering from original view. To save GPU memory, we render downsampled RGB image \( \hat{I}_{low} \) and depth map \( \hat{Z}_{low} \). According to known camera calibration and multiple depth value \( \{z_i | i = 1, \ldots, D\} \), a pixel point \( [u, v]^T \) in \( \hat{I} \) can be back-projected to several 3D point coordinates \( \{[x_i, y_i, z_i]^T | i = 1, \ldots, D\} \). We can calculate the distance between two successive 3D points as follows:

\[
\delta_{x_i, y_i, z_i} = \| [x_{i+1}, y_{i+1}, z_{i+1}] - [x_i, y_i, z_i] \|_2 \tag{11}
\]

Denote the distance map at depth \( z_i \) as \( \delta_{z_i} \in \mathbb{R}^{H_d \times W_d \times 1} \). We can explicitly learn the density map \( \sigma_{z_i} \in \mathbb{R}^{H_d \times W_d \times 1} \) from \( F_{density} \) and color map \( c_{z_i} \in \mathbb{R}^{H_d \times W_d \times 3} \) from \( F_{rgb} \). Then we can render \( \hat{I} \) by:

\[
\hat{I}_{low} = \sum_{i=1}^D T_i (1 - \exp(-\sigma_{z_i} \delta_{z_i})) c_{z_i}, \tag{12}
\]

where \( T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_{z_j} \delta_{z_j}\right) \) denotes the accumulated transmittance map from plane 1 to plane \( i \). Similarly, the low resolution depth map \( \hat{Z}_{low} \in \mathbb{R}^{H_d \times W_d \times 1} \) can be rendered by:

\[
\hat{Z}_{low}(u, v) = \sum_{i=1}^D T_i (1 - \exp(-\sigma_{z_i} \delta_{z_i})) z_i \tag{13}
\]

We simply upsample \( \hat{I}_{low} \) and \( \hat{Z}_{low} \) to original image resolution to get target image \( \hat{I} \in \mathbb{R}^{H \times W \times 3} \) and depth map \( \hat{Z} \in \mathbb{R}^{H \times W \times 1} \). Such recovered RGB images and depth maps will be supervised by RGB loss and depth loss (See Figure 5. Supervision from two RGB images by using volume rendering. We can obtain associated rendered RGB images from frustum density features \( F_{density} \) and frustum radiance fields \( F_{rgb} \) via Equation 12. Under this setting, we do not require explicit depth labels (i.e., LiDAR depths). Instead, the depth can be implicitly learned by calculating RGB reconstruction loss between original and rendered images.)
They are used in training as auxiliary tasks to facilitate the learning process and can be discarded at the inference stage.

**Rendering from other views.** Moreover, we can render from other views as long as such views’ frustum overlaps with the original view’s frustum and their camera calibrations are available. The rendered images can be produced in the same way described above. By calculating the image reconstruction loss from other views’ rendered images, our NeRF-like representations can get extra supervisions. In Figure 5, we give an example of taking the stereo images (left and right images) in KITTI [17] as volume rendering targets. In this way, we can even eliminate the reliance on explicit depth supervisions. Considering the space limitation of the main text, we provided ablation of depth supervision loss items but remove the stereo depth estimation surface in LIGA-Waymo. The overall loss is formulated as:

\[ L = \lambda_{rgb}L_{rgb} + \lambda_{depth}L_{depth} + \lambda_{sdf}L_{sdf} + \lambda_{LIGA}L_{LIGA}, \]

where \( \lambda_{rgb}, \lambda_{depth}, \lambda_{sdf}, \lambda_{LIGA} \) are fixed loss weights. We empirically set all weights to 1 by default. **RGB loss.** The RGB loss consists of two items: Smooth L1 loss \( L_{rgb} \) and SSIM [56] loss \( L_{SSIM} \) which are usually used in image reconstruction:

\[ L_{rgb} = \lambda_{smooth}L_{smooth} + \lambda_{SSIM}L_{SSIM} \]

**Depth loss.** Depth map labels are obtained by projecting associated LiDAR points onto the image plane. The depth loss is L1 loss between ground truth sparse depth map \( Z \) and predicted depth map \( \hat{Z} \),

\[ L_{depth} = \frac{1}{N_{depth}} \sum_{u,v} \|\hat{Z} - Z\|_1 \]

where \( N_{depth} \) denotes the number of valid pixels \((u,v)\) with LiDAR depth.

**SDF loss.** The SDF loss encourages \( F_{sdf} \) to vanish on geometry surface \( \mathcal{M} \),

\[ L_{sdf} = \frac{1}{N_{gt}} \sum_{x,y,z} \|F_{sdf}(x,y,z)\|_2 \]

where \( N_{gt} \) denotes the number of LiDAR points in the current camera view.

**LIGA loss** We denote the original loss in LIGA [19] as \( L_{LIGA} \). We keep the detection, imitation and auxiliary 2D supervision loss items but remove the stereo depth estimation item to fit the monocular setting. The imitation item is removed in all Waymo [52] experiments for convenience.

\[ L_{LIGA-KITTI} = L_{det} + \lambda_{im}L_{im} + \lambda_{2d}L_{2d} \]

\[ L_{LIGA-Waymo} = L_{det} + \lambda_{2d}L_{2d} \]

5. Experiments

5.1. Dataset and Metrics

**KITTI.** The KITTI dataset [17] comprises of 7,481 images in the training \((trainval)\) set and 7,518 images in the testing \((test)\) set with synchronized LiDAR point clouds. In previous works [11], the training images were split into a train set (3,712 samples) and a val set (3,769 samples). We also follow this setting. Results on the KITTI dataset [17] are measured using IoU-based criteria with a threshold of 0.7 to compute averaged precision over 40 recall values \((AP_{[0.7]})\) for the car class in both bird’s-eye-view (BEV) and 3D tasks.
Waymo. The Waymo Open Dataset [52] consists of 798 training sequences and 202 validation sequences. We follow the setting in CaDDN [47] which only considers the front camera and uses the sampled training set (51,564 samples). Results on Waymo val are measured by officially evaluation of the mean average precision (mAP) and the mean average precision weighted by heading (mAPH) with IoU criteria of 0.7 and 0.5. The evaluation is performed on two difficulty levels (Level1 and Level2) and three distance ranges (0 - 30m, 30m - 50m, 50m - ∞).

5.2. Implementation Details

Training details. Our method is implemented with Pytorch [42] framework. We have done all experiments with four NVIDIA 3080Ti (12G) GPUs. The detector is trained using AdamW [30] optimizer with $\beta_1 = 0.9, \beta_2 = 0.999$. The batch size is fixed to 4, with 1 sample on each GPU. On KITTI [17], the input size is fixed to $320 \times 1248$, we train 50 epochs using an initial learning rate of 0.001, and 10 epochs with a reduced learning rate of 0.0001, weight decay is 0.0001. On Waymo [52], the input size is fixed to $640 \times 960$, we train 16 epochs using an initial learning rate of 0.001, and 4 epochs with a reduced learning rate of 0.0001, weight decay is 0.0001.

Hyperparameters. On KITTI [17], we sample $D = 72$ planes within depth range $[z_n, z_f] = [2, 59.6](\text{meter})$ in the frustum space. The voxel grid size is $[2, 59.6] \times [-30.4, 30.4] \times [-3, 1](\text{meter})$ and voxel size is $[0.2, 0.2, 0.2](\text{meter})$ for the depth, width, and height axis in 3D space, respectively. On Waymo [52] the voxel grid range is $[2, 59.6] \times [-25.6, 25.6] \times [-3, 1](\text{meter})$ while all other hyperparameters are the same as KITTI experiments.

Table 1. Comparisons for Car category on KITTI test at IOU threshold 0.7. We obtain the values of other methods from respective papers. We use red to indicate the highest result and blue for the second-highest result. We can see that our method sets a new state of the art.

5.3. Main Results

KITTI. Table 1 shows the main results of MonoNeRD on the KITTI test set. Our method is competitive with previous methods without any extra data. MonoNeRD achieves the best results under the Moderate and Hard settings in both $AP_{BEV}$ and $AP_{3D}$. Compared to DD3D [40], which uses a vast private dataset for depth pre-training, we boost the performance from 22.56/16.34 to 23.46/17.13 under moderate setting, and from 20.03/14.20 to 20.97/15.63 under hard setting for $AP_{BEV}/AP_{3D}$, respectively. As for DID-M3D [43], we exceed it on both moderate and hard settings with 0.7/1.14 $AP_{BEV}$ and 0.84/1.88 $AP_{3D}$.

Waymo. Table 2 shows the main results of on the Waymo val set. MonoNeRD achieves competitive results without using data augmentation techniques. It has a lightweight backbone (ResNet34) compared to other depth-based methods like CaDDN (ResNet101) and DID-M3D (DLA34).

The Results on both KITTI [17] hard and Waymo [52] distance ranges (30m - 50m, 50m - ∞) demonstrate the superiority of our method in handling distant and occluded objects.

5.4. Ablation Studies

Architectural components. We conduct an ablation study on the car category in KITTI val to analyze the effectiveness of the proposed NeRF-like representations. In Table 3, Exp. (a) is the baseline which directly takes features sampled from $F_p$ as voxel representations $V_3d$ for M3D. In Exp. (b), we use more 3D convolution layers to prevent enhancements from complex model structures. Exp. (c) shows that supervision only from a single RGB image is useless and even harmful for detection since a purely single image cannot provide depth information. We can see that sparse depth map labels from LiDAR can significantly improve the per-
performance of the detector from Exp. (d). The effectiveness of \( L_{rgb} \) can be revealed by comparing Exp. (d, e) or Exp. (f, g). Finally, SDF loss brings noticeable performance gains, especially on \( AP_{BEV} \).

```
<table>
<thead>
<tr>
<th>Exp.</th>
<th>Setting</th>
<th>( L_{rgb} )</th>
<th>( L_{depth} )</th>
<th>( L_{sd} )</th>
<th>( AP_{BEV} )</th>
<th>( AP_{3D} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>✓</td>
<td>✓</td>
<td>24.96 / 17.01</td>
<td>19.27 / 13.38</td>
<td>16.89 / 11.73</td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td>✓</td>
<td>✓</td>
<td>25.64 / 17.22</td>
<td>19.75 / 13.65</td>
<td>18.03 / 11.99</td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td>✓</td>
<td>✓</td>
<td>24.83 / 16.92</td>
<td>19.10 / 13.11</td>
<td>16.74 / 11.73</td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td>✓</td>
<td>✓</td>
<td>26.91 / 18.72</td>
<td>20.87 / 14.54</td>
<td>18.57 / 12.60</td>
<td></td>
</tr>
<tr>
<td>(e)</td>
<td>✓</td>
<td>✓</td>
<td>26.46 / 19.44</td>
<td>20.16 / 15.03</td>
<td>17.59 / 12.83</td>
<td></td>
</tr>
<tr>
<td>(f)</td>
<td>✓</td>
<td>✓</td>
<td>27.94 / 20.28</td>
<td>21.44 / 15.32</td>
<td>18.82 / 13.48</td>
<td></td>
</tr>
<tr>
<td>(g)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>29.03 / 20.64</td>
<td>22.03 / 15.44</td>
<td>19.41 / 13.99</td>
</tr>
</tbody>
</table>
```

Table 3. Ablation of network structure and volume rendering loss. “3D Conv.”: 3D convolution blocks as mentioned in Section 4.1.2; “\( L_i \)”: use the loss for supervision; “Exp.”: Experiment.

**Positional aware module.** Based on experiment (g) in Table 3, we replace the query-key-value (QKV) based positional aware module with the concatenation of positional features and back-projected image features. The results show a performance degradation in \( AP_{BEV} \) from 29.03/22.03/19.41 to 28.39/21.60/18.94, and in \( AP_{3D} \) from 20.64/15.44/13.99 to 19.73/15.16/12.94. This experiment demonstrates the effectiveness of the proposed design.

```
<table>
<thead>
<tr>
<th>Exp.</th>
<th>Pos.</th>
<th>( AP_{BEV} )</th>
<th>( AP_{3D} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>QKV</td>
<td>29.03 / 20.64</td>
<td>22.03 / 15.44</td>
</tr>
<tr>
<td>2</td>
<td>Cat.</td>
<td>28.39 / 19.73</td>
<td>21.60 / 15.16</td>
</tr>
</tbody>
</table>
```


**5.5. Visualization**

In Figure 6, we replace the proposed representation with depth-transformed one used in previous works [12, 47] and visualize them for comparison. For depth-transformed category, we first predict the depth map from the input RGB image and apply the same transformation as CaDDN [47] to get its 3D representation. We visualize such representation by back-projecting the estimated depth map to the 3D space. For the NeRF-like representation, we directly visualize the predicted 3D volume density. Our method produces dense and continuous geometry in the distance (as indicated
by the red box area). Due to space constraints, we provide additional quantitative and qualitative results in the Supplementary materials. Occupancy results are best viewed as videos, so we urge readers to view our supplementary video.

6. Conclusion

In this paper, we explore how to optimize the intermediate 3D feature representations implicitly for monocular 3D object detection (M3D) without explicit binary occupancy annotations. We build a bridge between Neural Radiance Fields (NeRF) and 3D object detection, and propose a novel monocular 3D object detection framework, i.e., MonoNeRD. MonoNeRD, which produces continuous NeRF-like Representations for M3D. MonoNeRD treats intermediate 3D representations as SDF-based neural radiance fields and optimizes them using volume rendering techniques. Extensive experiments show that our method sets a new baseline for monocular 3D detection and has the potential to be extended to other 3D perception tasks.

Limitations and future work. The performance of our method is highly dependent on the modeling approaches. We are currently limited by the working with bounds modeling (see Section 3.3), which could fail to predict 3D occupancy when its 2D projected pixel represents sky or other things not within the specified bounds. We believe this is an interesting direction for future research.

Acknowledgement

This work was supported in part by The National Nature Science Foundation of China (Grant Nos: 62273301, 62273302, 62273303, 62036009, 61936006), in part by the Key R&D Program of Zhejiang Province, China (Grant Nos: 62273301, 62273302, 62273303, 62036009, 61936006), in part by the Key R&D Program of Zhejiang Province, China (Grant No: 2022A-240-G).

References


