Improved Point Transformation Methods For Self-Supervised Depth Prediction

Chen Ziwen
Grinnell College
Grinnell, IA
chenziwe@grinnell.edu

Zixuan Guo
Grinnell College
Grinnell, IA
guozixua@grinnell.edu

Jerod Weinman
Grinnell College
Grinnell, IA
jerod@acm.org

***Abstract*—Given stereo or egomotion image pairs, a popular and successful method for unsupervised learning of monocular depth estimation is to measure the quality of image reconstructions resulting from the learned depth predictions. Continued research has improved the overall approach in recent years, yet the common framework still suffers from several important limitations, particularly when dealing with points occluded after transformation to a novel viewpoint. While prior work has addressed the problem heuristically, this paper introduces a z-buffering algorithm that correctly and efficiently handles occluded points. Because our algorithm is implemented with operators typical of machine learning libraries, it can be incorporated into any existing unsupervised depth learning framework with automatic support for differentiation. Additionally, because points having negative depth after transformation often signify erroneously shallow depth predictions, we introduce a loss function to explicitly penalize this undesirable behavior. Experimental results on the KITTI data set show that the z-buffer and negative depth loss both improve the performance of a state of the art depth-prediction network. The code is available at https://github.com/arthurhero/ZbuffDepth and archived at https://hdl.handle.net/11084/10450.**

I. Introduction

With the advent of deep neural networks and large, real-world data sets, it has become possible to learn models that estimate depth from a single image with remarkable accuracy. The common framework for learning to predict depth from a single image involves training on image pairs of a scene with known relative camera positions, either from stereo or egomotion. Models can be trained in a self-supervised fashion (meaning there is no ground truth depth signal) by using one image and the predicted depth to reconstruct the view in the other image. The training should bring the reconstructed image and the true image into agreement.

To facilitate the process, the scene projection from one image is inverted to generate a 3D point cloud, each point being associated with a pixel in the original image. This point cloud is then transformed (by translation and rotation) to the coordinate system of another viewpoint and finally reprojected to synthesize the image of the second view. (See Figure 2.)

The quality of the depth prediction is measured indirectly by comparing the pixels in the reconstructed image to the corresponding pixels in the actual image. If the depth prediction is correct, the two sets of pixels should be similar, despite

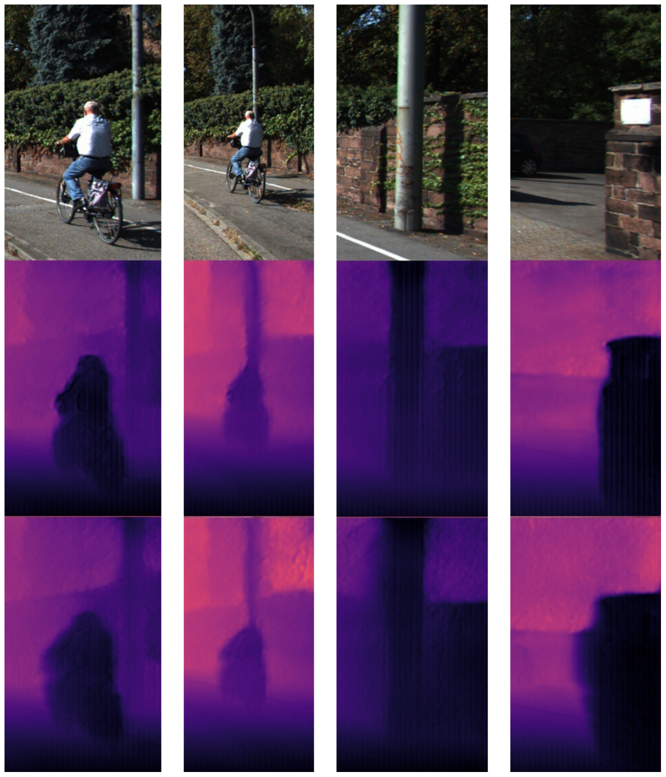

Fig. 1. Comparison of depth prediction results with (second row) and without (third row) a z-buffer in the self-supervised training pipeline. Differences are most pronounced at large depth boundaries where occlusions are most prominent. (Images from the KITTI [1] Eigen test split.)

changes in lighting or viewing angle. Stereo image pairs [2], [3] and video sequences [4], [5], [6], [7] are usually used as the signal source. Common evaluation benchmarks for this task include KITTI [1] and Cityscapes [8].

Several problems can occur during the viewpoint transformation phase of this training paradigm. First and foremost, after the viewing angle is rotated some visible points become occluded because two points project onto the same pixel in the second image (cf. Figure 4). For correct image reconstruction, we need to render only the point closest to the image plane; any other point is occluded by the closest point and thus invisible from the second camera's viewpoint. The z-buffer,

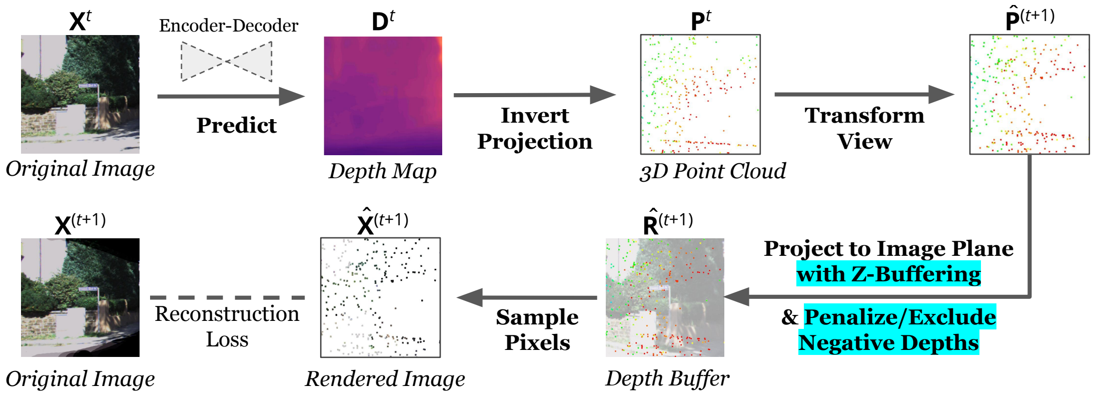

Fig. 2. Overview of our training method under the image reconstruction depth-learning paradigm, with contributions highlighted. Here we use a video sequence with known egomotion as the signal source. $\mathbf{X}$ stands for images, $\mathbf{D}$ for depth map, $\mathbf{P}$ for point cloud, $\mathbf{R}$ is projected (registered) point cloud. Superscript t denotes the image sequence number.

or depth buffer, is a computer graphics workhorse created to solve precisely this problem.

The occlusion problem has only been indirectly addressed by previous work in learning monocular depth estimation. Godard *et al.* [9] propose a minimum reprojection loss, and Gordon *et al.* [10] propose a minimum depth consistency filter to simulate a z-buffer. However, the former solution requires multiple source images and is not guaranteed to eliminate the error, while the latter is still only a heuristic method that can incur both false positives and false negatives. Using a z-buffer to solve the occlusion problem is conceptually easy but challenging to implement, in that it may be non-trivial to parallelize. Without caution, the resulting algorithm might become a training bottleneck. We propose an efficient, parallel approach (implemented in PyTorch) to address the occlusion problem, which can hamper performance around depth discontinuities, as shown in Figure 1. Our method is exact (*i.e.*, does not rely on any heuristics) and can be easily incorporated into any self-supervised learning framework that relies on reprojection.

Another issue that might be fatal for networks trained from scratch is transformed points with negative depth. Having negative depth means that the points are behind the image plane and thus should not be associated with any pixels in the second image. Therefore, they are sure to be excluded from the image reconstruction loss. However, during the initial phase of our training, when the depth prediction is not very accurate, we found that often many points would get abnormally shallow depth and hence negative depth (*i.e.*, behind the image plane) after transformation. Excluding all of those points will make learning inefficient. Most of these points should not in fact have negative depth if they can still be projected to a pixel within the second image boundary. We therefore introduce a simple loss function to penalize such negative depths.

Experimental evaluation (Section IV-C) demonstrates the importance of these techniques using ablation studies, as well as comparison with previous methods. We also show that the best timing for inserting the z-buffer into the training process is not at the beginning, but after an initial training phase.

In summary, our primary contributions include:

- we confirm that proper z-buffering improves depth estimates for reprojection-based training methods,
- we provide an efficient z-buffering algorithm compatible with differentiable deep learning systems, and
- we propose a loss penalizing erroneously shallow (*i.e.*, negative) depth, which also improves performance.

In Section II we situate this work in the context of related works, while Section III details our approach. Section IV provides an experimental evaluation of the proposed methods.

II. Related Works

A. Supervised Monocular Depth Estimation

Historically, estimating depth from a single image involved using hand-crafted features or geometric constraints [11]. The accuracy of those methods often depended on hand-crafted features, which can yield unsatisfactory performance on various scenes if the chosen features are not optimal. Subsequently, research has shifted toward learning-based methods.

Depth estimation using deep convolutional neural networks (DCNNs) has followed on the success of DCNNs in many computer vision tasks such as image classification [12] and image segmentation [13]. Eigen *et al.* [14] first proposed a multi-scale architecture with both a local and global network to generate a refined, high-resolution depth map from lower-resolution depth maps. Their work treats the depth estimation task as a regression problem by minimizing the total pixel-wise error between the network output and the ground-truth depth. Laina *et al.* [15] introduced a fully convolutional network with a residual-connected encoder and upsampling block to increase spatial resolution of the predicted depth map. Liu *et al.* [16] presented a spatial propagation network for predicting an affinity matrix; Chen *et al.* [17] subsequently applied the network to the depth estimation task, which increased the resolution of sparse ground truth depth maps.

To incorporate multi-scale information and high-resolution depth maps, many works adopt a U-net-like architecture with skip connections between encoder and decoder [13]. With the general success of the encoder-decoder architecture, many improvements have been made, such as substituting the encoder with pre-trained networks [18] and enforcing a planar constraint on local patches of the predicted depth output [19].

B. Unsupervised Monocular Depth Estimation

While supervised methods usually offer depth estimations with higher accuracy, the ground truth depth maps used in supervised depth estimation are often subject to spatial inaccuracy due to calibration error and instrument error. In addition, ground truth depth measurements, arising from 3D LiDARs, are usually quite sparse. Therefore, recent research has introduced depth estimation network architectures that learn without depth as an explicit training signal. In this case, the models use image reconstruction as the supervisory signal during the training stage, as described in Section I. The model training process usually takes a temporal series of monocular images and/or pairs of stereo images as input; learning proceeds by enforcing the consistency between an observed image and an image from an alternate viewpoint as reconstructed from the depth prediction (as in Figure 2).

Garg *et al.* [2] proposed the first such unsupervised depth estimation network to achieve performance comparable to supervised networks. It uses an encoder-decoder structure to produce a depth for the left image of the stereo image pair and subsequently utilizes the estimated depth of left image, inter-view displacement and the right image to synthesize the left viewpoint. Then, the reconstruction error between the left image and the synthesized left image is used as the training signal for the network. Using a monocular video sequence with egomotion instead of a stereo image pair, Zhou *et al.* [4] trained a depth estimation network simultaneously with an explainability mask and a pose network. The explainability mask then addresses the occlusion problem by excluding pixels for which the network has low confidence in its depth predictions.

Broadening what is considered for self-consistency, Godard *et al.* [3] proposed a network that enforces the left-to-right and right-to-left consistency with smoothness, reconstruction, and left-right disparity losses. Mahjourian *et al.* [5] presented a network with a video input sequence that enforces 3D geometry consistency within the sequence by estimating egomotion. Godard *et al.* [9] improved their previous architecture [3] by using both the stereo image pair and temporal video sequences as training signals.

Most top-performing unsupervised methods learn to predict depth by enforcing the consistency between actual observations and images reconstructed from depth estimates. Another novel approach by Guizilini *et al.* [7] introduced a velocity supervision loss to solve the scale ambiguity in self-supervised depth estimation networks. Luo *et al.* [20] incorporated a GAN architecture, training the model to synthesize the alternate viewpoint's image and directly calculate depth from the original image and the synthesized image. Because the reconstructed image is not produced by inferring projective geometry, GAN-based methods have no direct need for the occlusion handling methods we propose.

C. Occlusion Handling in Reprojection

When the loss functions used for training involve photometric agreement between a reconstruction and an actual image, accurate renderings will be important. As indicated previously, most early works used all points in the loss function, even if they were occluded in one of the image pairs [3], [5].

More recent work has taken occlusions into account with attempts to mitigate the issue. When two points project to the same pixel, CalibNet by Iyer *et al.* [21] arbitrarily selected only one to contribute to the loss, even though it may be the occluded point. Monodepth2 by Godard *et al.* [9] incorporated a minimum reprojection loss that requires multiple source images. This method assumes that a point occluded in one of the source images might be still visible in others. The images where occlusion happened will likely render higher reconstruction loss due to the erroneously matched pixels. Thus, for each pixel in the original image, Monodepth2 uses only the smallest reconstruction loss from several source images, where occlusion is least likely to have happened. However, this is not applicable when we only have a pair of images (thus only one source image available), as in stereo methods. More importantly, lower loss does not guarantee that the chosen source pixel is visible. PackNet-SfM by Guizilini *et al.* [7] also adopted this approach.

Gordon *et al.* [10] proposed a minimum depth consistency filter to simulate a z-buffer. Using the predicted depth from frame A as a reference, when points from frame B are transformed to coordinate frame A, any point "behind" (having greater depth than) the *predicted depth* will be excluded from the loss calculation. This heuristic method to eliminate occluded points is sensitive to the model's prediction accuracy, which itself is the learning goal.

We propose using a z-buffer to directly address pixel occlusions in the image reconstruction process. Although conceptually easy, it is not "embarrassingly parallel" where multiple points compete for rendering to the same raster location. However, efficiency is not the only concern. Because the z-buffer contents depend on the very depth prediction process being learned, it is likewise potentially sensitive to model accuracy. At the early stages of training, most depth predictions are inaccurate, and thus points may appear occluded due to the prediction error rather than the true scene geometry. Excluding these points from the loss calculation might hamper learning. On the other hand, if we utilize the z-buffer too late, the model might have already mis-fit, so that the z-buffer might not help or else would require additional training epochs. Our experiments will demonstrate there is indeed an optimal middle-ground for including the z-buffer.

In theory, any unsupervised method using image reprojection should benefit from incorporating our methods. Our experiments confirm that the performance of an existing model can indeed be further improved by incorporating the method.

D. Z-Buffer Algorithms

With the z-buffer's longstanding history in computer graphics [22], this work is not the first to demand an efficient algorithm. For graphics rendering, the computation is nearly universally implemented almost entirely in hardware. However, to be used within a deep learning pipeline, the computation must also produce derivatives, which are not readily available in existing hardware and APIs.

A serial algorithm would calculate a projected pixel coordinate for each 3D point. If another point is already stored at that coordinate, the algorithm compares their depths and retains only the point with smaller depth value. In a highly parallelized deep learning pipeline, the speed bottleneck created by such a serial algorithm proves unacceptable.

Work on parallelizing the z-buffer algorithm is nearly as old as the algorithm itself [23]. We provide a few highlights. Forty years ago, Parke [24] investigated three algorithms distributing the "scan conversion" task (*i.e.*, perspective projection) among parallel processors with a distributed z-buffer.

Li and Miguet [25] proposed two complementary parallel z-buffer algorithms for a distributed memory transputer. They note that a naïve approach would distribute *all* scene points to each processor, while dividing the rendering task (portions of the reconstructed image) among processors. To handle large point clouds, their two algorithms examine either the case when scene elements are statically mapped to a processor, with each process potentially contributing to any rendered pixel, or else the case when scene elements are "dynamically redistributed" among a ring of processors. The first approach requires a (tree-shaped) reduction familiar to modern GPU programmers, with a conditional in the merge that checks for the lower z value. Motivated by memory limitations, the second approach distributes work and memory demands by dividing the image into regions and cycling subsets of points through each region for processing and merging.

Renaud [26] similarly concentrated on the difficulty of distributing scene elements to the image regions onto which they project and subsequently balancing the load among SIMD processing elements. Like Li and Miguet, Shen *et al.* [27] also focused on efficiently merging depths from different objects for the same pixels on a PRAM machine.

Rather than focus on the merge step (which introduces conditionals that lower SIMD throughput), our approach admits a race condition among scene elements mapped to the many SIMD processors of modern GPUs. It then iteratively identifies points occluded points that incorrectly won the race. The method (cf. Section III-D) is implementable in modern GPU-based machine-learning libraries, allowing its use in gradient descent frameworks.

The PyTorch3D library [28] offers a differentiable z-buffer for rendering polygon meshes, rather than the point clouds required for this problem. It uses a custom CUDA kernel with atomic instructions, whereas our method is easily expressed in higher-level parallelized primitives of multiple machine learning libraries.

III. Method

In this section, we define the background terms and notation before describing our overarching training method. We then detail solutions to several issues that occur during the point cloud transformation phase of the self-supervised depth learning paradigm, as introduced in Sections I and II.

A. Definitions

In this paper, the notation font for tensors is A, for matrices is $\boldsymbol{A}$, and for scalars is A. The variables used in our training method include $\mathsf{X} \in \mathbb{R}^{3\times h\times w}$ for RGB images, $\mathsf{D} \in \mathbb{R}^{1\times h\times w}$ for depth images, $\mathsf{P} \in \mathbb{R}^{3\times n}$ for point clouds, $\mathsf{R} \in \mathbb{R}^{3\times h\times w}$ for point cloud registration (explained below), $\boldsymbol{E}$ for the relative position matrix between cameras, and $\boldsymbol{M}_{\text{proj}}$ for the projection matrix (i.e., the camera intrinsic matrix).

Image sequence number is indicated using superscripts (*e.g.*, X^t, P^{t+1}), and pixel location is indicated using subscripts (*e.g.*, $\mathsf{X}_{i,j}$, $\mathsf{R}_{i,j}$).

The point cloud registration R is a tensor such that $\mathsf{R}_{i,j}$ is the (x, y, z) coordinates of the point in 3D space corresponding to the pixel at location (i, j) in image X.

The ℓ_1 loss (sum of absolute values) is denoted $\|\cdot\|_1$.

B. Basic Model

Our overarching training method uses image reconstruction (Figure 2), combined with point cloud matching. We first take two images X^t and X^{t+1}, with their relative position $\boldsymbol{E}^t$ known. For stereo data, the images are "left" and "right" pairs with a temporally invariant relative position $\boldsymbol{E}$, but the t, $t+1$ sequence applies more generally to egomotion sequences as well. We train a model to predict depth D^t and D^{t+1} for each of the images and use the known camera intrinsic matrix to inverse project the depths to point clouds P^t and P^{t+1}. We then transform the point clouds to each other's relative position. Using the matrix $\boldsymbol{E}^t$ we have

$$\hat{\mathsf{P}}^t \triangleq \boldsymbol{E}^t \mathsf{P}^{t+1}. \tag{1}$$

($\boldsymbol{E}^t$ is invertible, allowing for the reverse process.) Finally, we project these transformed point clouds onto the image plane, get the resulting pixel coordinates for each point, and sample the pixel color from the other image (*i.e.*, X^{t+1}) to reconstruct the original image from the other viewpoint (*i.e.*, $\hat{\mathsf{X}}^t$).

Although the following loss functions are all written from one image to another, both directions are included in the total loss during training.

A simple point cloud matching loss [3]

$$L_{\text{point}} = \|\hat{\mathsf{R}}^t - \mathsf{R}^t\|_1 \tag{2}$$

ensures the consistency of point clouds predicted for continuous image frames, where R^t is the original registration tensor simply reshaped from P^t. That is, $\hat{\mathsf{R}}^t$ stores the points from $\hat{\mathsf{P}}^t$ that are registered to the pixel locations in X^t using the z-buffering algorithm introduced in Section III-D.

An image reconstruction loss [4], [2]

$$L_{\text{image}} = \|\hat{\mathsf{X}}^t - \mathsf{X}^t\|_1 \tag{3}$$

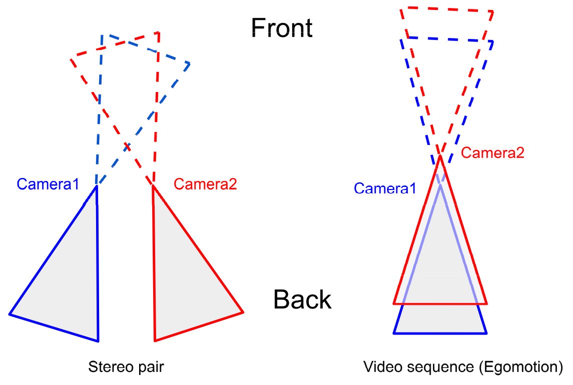

Fig. 3. Improbability of negative depth. It is highly unlikely for points visible to camera 1 to be in frame but have negative depth with respect to camera 2 (i.e., to appear in the grey area of camera 2). With egomotion, such points need to be squeezed between the two cameras (inside the little shaded diamond area in the right figure), which is extremely unlikely to happen in data sets like KITTI [1].

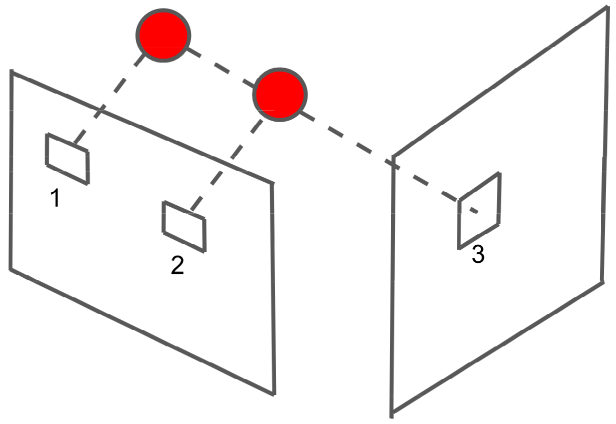

Fig. 4. The occlusion problem: two points in space projecting to different pixels in the original image might be projected to the same pixel on another image. Pixel 1 should not be used to match pixel 3 in the image reconstruction loss.

ensures the similarity between the original image and the reconstructed image from the transformed point cloud. We also include the Structured Similarity (SSIM) Loss used in Godard *et al.* [3] and Mahjourian *et al.* [5]:

$$L_{\text{SSIM}} = \sum_p 1 - \text{SSIM}(\hat{\mathbf{X}}^t_p, \mathbf{X}^t_p), \tag{4}$$

where $\mathbf{X}_p$ here represents a 3×3 image patch. Our overall loss function is

$$L_{\text{total}} = \lambda_1 L_{\text{point}} + \lambda_2 L_{\text{image}} + \lambda_3 L_{\text{SSIM}} + \lambda_4 L_{\text{nd}}, \tag{5}$$

where L_{nd} denotes the "negative depth loss," detailed below in Section III-C. Section IV-B lists the specific relative weights λ_i and other hyperparameter details.

C. Negative In-frame Depth

After a point cloud has been transformed to another viewpoint and reprojected, some points might fall behind the image plane, resulting in a negative depth. Stereo pairs or video sequences have relatively close viewpoints. Thus it is highly unlikely that a point originally projecting to the center of the first image will become invisible in the second image (see Figure 3). When negative depth predictions occur, they signify that the model is producing abnormally or inappropriately shallow depth values.

Let $\mathcal{N}$ be the set of points with negative depths that remain within the image boundary after transformation and projection:

$$\mathcal{N} \triangleq \{(i,j) \,|\, (d_{i,j} < 0) \wedge (0 \leq i < w) \wedge (0 \leq j < h)\}, \tag{6}$$

where (i,j) is the image coordinate of a point, and $d_{i,j}$ is the depth of the point. Our loss term penalizes such points:

$$L_{\text{nd}} = \sum_{(i,j)\in\mathcal{N}} |d_{i,j}|\,. \tag{7}$$

We observe that it is not meaningful to include the in-frame, negative-depth points of $\mathcal{N}$ in the loss calculations for L_{point} (equation 2), L_{image} (equation 3), and L_{SSIM} (equation 4). We are not aware of prior work that filters these points from the calculations (and thus from learning). In our experiments below, when accounting for negative in-frame depth we also exclude points in $\mathcal{N}$ from all losses except L_{nd}, equation 7. We found these conditions to improve performance.

We also found that this loss is particularly helpful during the initial phase of the unsupervised training, especially when the encoder is not pre-trained. If the network tends to predict erroneously shallow depth at the beginning, this loss can help push the depth value toward the correct range. If we do not penalize this erroneous negative depth, but instead only mask them out, then most all the points are discarded and the network is unable to learn.

D. Efficient Z-buffering

Both the L_{point} loss and the L_{image} loss of equation 5 require the correspondence of pixels between two images. As Figure 4 shows, an occluded point in one frame should not correspond to any point in the other frame. Occlusion happens after the point cloud transformation phase because some points inevitably "overlap" with each other by projecting onto the same pixel in the second image. In these cases, we need to choose the point closer to the image plane, as it is the point associated with the visible pixel.

To fundamentally solve the issue, we need to identify the point closer to the image plane when occlusion occurs. Inspired by the z-buffer (or depth buffer) concept from computer graphics, our method achieves efficient processing for all points using parallel computing (cf. Section II-D). We explain Algorithm 1 below. By implementing it in PyTorch, we easily access its differentiable, parallelized operators.

Given a set of points from frame A, we transform them into coordinate frame B. The transformed depth $\mathbf{D}$ has shape $1 \times h \times w$. We vectorize it to $1 \times N$ and calculate the corresponding (i,j) pixel coordinates $\mathbf{C} \in \mathbb{R}^{2\times N}$ on image B for all the points using the camera intrinsic matrix. We

Algorithm 1: Parallel z-buffering algorithm.

```
Input  : $\mathbf{D}$: $1 \times N$, reprojected point depths
         $\mathbf{K}$: $1 \times N$, point raster absolute indices
Output: $\mathbf{V}$, indices for points that should be included
         in the loss.
1  initialize empty $\mathbf{Z}$, same size as $\mathbf{D}$ // Z- buffer
2  $\mathbf{D}_{\text{orig}} \leftarrow \mathbf{D}$
3  $\mathbf{K}_{\text{orig}} \leftarrow \mathbf{K}$
4  repeat
       // Parallel write; race condition
5      $\mathbf{Z}[\mathbf{K}[p]] \leftarrow \mathbf{D}[p]$, for all $0 \leq p < |\mathbf{D}|$
6      create $\mathbf{T}$, same size as $\mathbf{D}$
       // Parallel read of assigned depths
7      $\mathbf{T}[p] \leftarrow \mathbf{Z}[\mathbf{K}[p]]$, for all $0 \leq p < |\mathbf{D}|$
       // Find occluded points in Z-buffer
8      $\mathbf{U} \leftarrow \{p \mid \mathbf{D}[j] < \mathbf{T}[p]\}$ // Parallel compare
9      if $|\mathbf{U}| \neq 0$ then
10         $\mathbf{D} \leftarrow \mathbf{D}[\mathbf{U}]$ // "Contract" to nearer
11         $\mathbf{K} \leftarrow \mathbf{K}[\mathbf{U}]$ // points NOT in Z-buffer
12     end
13 until $|\mathbf{U}| = 0$;
14 create $\mathbf{T}$, same size as $\mathbf{D}_{\text{orig}}$
15 $\mathbf{T}[p] \leftarrow \mathbf{Z}[\mathbf{K}_{\text{orig}}[p]]$ for all $0 \leq p < N$
16 $\mathbf{V} \leftarrow \{p \mid \mathbf{D}_{\text{orig}}[p] = \mathbf{T}[p]\}$ // Visible points
```

employ the "principled mask" [5] to discard the points that fall out of the image frame: masking out the points where the i (j, resp.) are larger than $h-1$ ($w-1$, resp.) or smaller than 0. We also discard points in $\mathcal{N}$, which are within the image frame but have negative depth values (cf. Section III-C).

For each point that remains, we calculate the "absolute indices" $\mathbf{K}$ ($1 \times N$) using the coordinate values from $\mathbf{C}$,

$$k \triangleq j \times w + i. \tag{8}$$

Thus, **points projecting to the same pixel location have the same absolute index.**

Let $\mathbf{Z}$ of shape $1 \times N$ be a tensor to store the correct (visible) depth at each pixel location. Now we assign the elements of $\mathbf{D}$ into $\mathbf{Z}$ using absolute indices $\mathbf{K}$ (line 5 in Algorithm 1). When multiple points project to the same pixel, this operation results in a race condition; we will not know whether the assigned point is visible. This approach stands somewhat in contrast to the prior work reviewed in Section II-D, which carefully folds competing points through conditional assignments. In our algorithm, to compare the values stored in the z-buffer $\mathbf{Z}$ with the original depths $\mathbf{D}$, we fetch back depth values from $\mathbf{Z}$ to a temporary tensor $\mathbf{T}$ (line 7), inverting the assignment. Then we simply compare the values in $\mathbf{D}$ and $\mathbf{T}$ to retain only points with a smaller depth value than stored in $\mathbf{T}$ (line 8). Having selected only those points from $\mathbf{D}$ and $\mathbf{K}$, we repeat the process (overwriting occluded points in the z-buffer) until there is no difference between $\mathbf{D}$ and $\mathbf{T}$.

During our experiments, this process iterates three or four times before all the correct depth values are stored, with only a few points left in the third or fourth rounds.

Finally, we use the correct $\mathbf{Z}$ tensor and compare it with the original $\mathbf{D}$ tensor. The positions where the two values are the same indicate the points that should be included in our losses.

Note the length of $\mathbf{U}$ becomes strictly smaller with each iteration. With a finite number of points, the algorithm is guaranteed to terminate. With this method, we can quickly process all points in parallel, with the iteration limit being the maximum number of occluding points along a ray.

Because all the operations in the algorithm can be implemented with PyTorch tensor methods, the GPU parallelization is straightforward. Moreover, the implementation gains the necessary benefit of providing the derivatives for the learning pipeline. For example, the assignment of line 5 in Algorithm 1 is implemented in PyTorch using `Tensor.index_copy` (or Tensorflow `tf.scatter_nd`), while the read out of lines 7 and 15 use `Tensor.index_select` (or `tf.gather_nd`). Lines 8 and 16 also employ SIMD-parallel comparators followed by select/gather.

IV. Experiments

This section describes the results of our model on standard data sets, comparing against other recent state of the art works and demonstrating the performance improvements of our approach.

A. Data Set

For all training and evaluation we use the KITTI data set [1] with the Eigen [14] training split.

For validation, we sample a random subset of size 500 from the Eigen validation split before training starts. After every 200 steps, we evaluate the model on the validation set, preserving the best model checkpoint so far (using the $\delta < 1.25$ metric).

Finally, for evaluation, we use the standard Eigen [14] test split, which contains 697 samples in total. The ground-truth depth is calculated from the laser point clouds provided by KITTI. The depths are capped at 80 meters; that is, points with ground truth depth exceeding the cap are excluded from test evaluations. We use the crop standard of Garg *et al.* [2], cropping away the upper half of the image and some boundary parts to avoid comparing with inaccurate ground-truth. We evaluate on the best checkpoint identified during training using the validation set.

B. Model Implementation and Training Details

For the depth prediction model, we use the "big-to-small" multi-scale local planar guidance architecture of Lee *et al.* [19]; the encoder is ResNet-50 [12], pre-trained on ImageNet [29], with all weights adjusted during training.

We train for 20 epochs with a batch size of 3 using the Adam optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.999$) and a learning rate of 1E–05.

The images in the KITTI data set vary somewhat in size. To compensate, we crop all training images to 352×1216, aligning the upper-left corner.

We augment the image data by applying small random variations to the colors of the input images given to the depth

TABLE I
PERFORMANCE ON KITTI EIGEN TEST SPLIT. SUPERVISION SIGNAL: D - DEPTH, V - VIDEO SEQUENCE (W/O GROUND-TRUTH EGOMOTION), S - STEREO. $\downarrow$ MEANS LOWER IS BETTER, $\uparrow$ MEANS HIGHER IS BETTER. BEST OVERALL IN **BOLD**. BEST AMONG S IN BLUE. BEST AMONG V IN RED.

Method	Signal	Abs Rel$\downarrow$	Sq Rel$\downarrow$	RMSE$\downarrow$	RMSE log$\downarrow$	$\delta < 1.25 \uparrow$	$\delta < 1.25^2 \uparrow$	$\delta < 1.25^3 \uparrow$
Eigen *et al.*[14]	D	0.203	1.548	6.307	0.282	0.702	0.890	0.958
Zhou *et al.*[4] *	V	0.198	1.836	6.565	0.275	0.718	0.901	0.960
Garg *et al.*[2] †	S	0.169	1.080	5.104	0.273	0.740	0.904	0.962
Mahjourian *et al.*[5] *	V	0.159	1.231	5.912	0.243	0.784	0.923	0.970
Wang *et al.*[6] *	V	0.148	1.187	5.496	0.226	0.812	0.938	0.975
Godard *et al.*[3] *	S	0.114	0.898	4.935	0.206	0.861	0.949	0.976
Gordon *et al.*[10]	V	0.129	0.982	5.230	-	-	-	-
Godard *et al.*[9]	S	0.107	0.849	4.764	0.201	0.874	0.953	0.977
Godard *et al.*[9]	S + V	**0.106**	0.806	4.630	0.193	0.876	0.958	0.980
Guizilini *et al.*[7]	V	0.107	0.802	**4.538**	**0.186**	**0.889**	**0.962**	**0.981**
This work	S	**0.106**	**0.743**	4.707	0.201	0.864	0.949	0.977

* method is trained on Cityscapes data set [8] and fine-tuned on KITTI.
† method is tested with depths capped at 50m, otherwise 80m.

TABLE II
ABLATION STUDIES CONDUCTED ON KITTI EIGEN TEST SPLIT. $\downarrow$ MEANS LOWER IS BETTER, $\uparrow$ MEANS HIGHER IS BETTER.

Negative Depth Loss	Occlusion Handling	Insertion Epoch	Abs Rel$\downarrow$	Sq Rel$\downarrow$	RMSE$\downarrow$	RMSE log$\downarrow$	$\delta < 1.25 \uparrow$	$\delta < 1.25^2 \uparrow$	$\delta < 1.25^3 \uparrow$
-	none	-	0.108	0.776	4.881	0.210	0.854	0.946	0.974
✓	none	-	0.109	0.764	4.790	0.207	0.859	**0.949**	0.975
✓	z-buffer	1	0.108	0.764	4.857	0.207	0.855	0.948	0.976
✓	z-buffer	6	0.109	0.775	4.847	0.211	0.851	0.943	0.974
✓	z-buffer	11	**0.106**	**0.743**	**4.707**	**0.201**	**0.864**	**0.949**	**0.977**
✓	z-buffer	16	0.109	0.770	4.898	0.210	0.854	0.946	0.974
-	z-buffer	11	0.109	0.752	4.808	0.205	0.858	**0.949**	0.976
✓	heuristic [10]	11	0.115	0.860	5.111	0.223	0.846	0.939	0.970

prediction network. However, we sample unmodified pixel colors from the original images during image reconstruction to avoid spurious color mismatches.

The hyperparameters of L_{SSIM} in equation 4 are as given in Mahjourian *et al.* [5]. For our overall loss function in equation 5, we set $\lambda_1 = 0.005$, $\lambda_2 = 10$, $\lambda_3 = 2$ and $\lambda_4 = 2$, which puts each loss term on roughly the same scale and acknowledges the relatively low importance of point cloud matching after the initial training.

In this setup, the model takes about 1.08 seconds to run one training step update (per batch) on an NVIDIA Titan RTX. Including the z-buffer adds only 8 ms to this processing time.

All code and training checkpoints are available.[1][2]

C. Results

Table I lists the evaluation results on the standard Eigen [14] test split compared with several previous methods. All the methods listed are *not* fine-tuned on any ground-truth depth data. This table is not meant to demonstrate our approach to be superior over all—although it does boost performance on some metrics—but instead to demonstrate its comparability with other similar unsupervised methods on the standard metrics, particularly when using the same training signals. In addition, it gives context for the relative scale of improvements.

Rather, the ablation experiments of Table II demonstrate the contributions of our methods with respect to a fixed network architecture. The first row establishes a baseline, using the network trained without the negative depth loss (*i.e.*, $\lambda_4 = 0$) and no occlusion-handling method, explicit or implicit. Importantly, when the negative depth loss is excluded, we also include in-frame negative-depth points from $\mathcal{N}$ in the other losses, as in prior work. The second row demonstrates that incorporating the negative depth loss (and excluding $\mathcal{N}$ from other losses) yields an improvement across all metrics but one. The next group of several rows incorporate the z-buffer for occlusion handling at different points in the training process (*i.e.*, beginning, 25%, 50%, and 75% of the way through). Having found an optimal time to insert the z-buffer—epoch 11 of 20, 50% of the way through the training—the last two rows test the relative contributions of the negative depth loss and the alternative occlusion-handling heuristic proposed by Gordon *et al.* [10].

These results demonstrate that the z-buffer indeed improves the depth prediction results when inserted at the right stage of training. Inserting the z-buffer either too early or too late harms the performance, which is not surprising.

At the early stages of training, most depth predictions are inaccurate, and thus the points appear occluded due to the prediction error rather than the true scene geometry. Excluding the points from the loss calculation hampers learning. On the other hand, if we utilize the z-buffer too late, then truly occluded points might inappropriately factor into the loss function and confuse the model.

Results also show that our use of a z-buffer outperforms

[1]Development repository: https://github.com/arthurhero/ZbuffDepth
[2]Permanent archive: https://hdl.handle.net/11084/10450

the consistency-driven heuristic method of Gordon *et al.* [10], even when added to the training pipeline with the same timing. Their method excludes every transformed point that is "behind" the predicted depth of the target image (cf. Section II-C). Experimental results imply that even when the depth predictions are roughly reliable (around epoch 11), we should not expect points from the original and target frames to agree precisely. Some tiny amount of error can make a valid rotated point fall behind the target depth image, masking it from the loss calculation. With so many extra points excluded from training, the model does not learn as well.

During our experiments, we found that excluding points with negative depth from the loss calculations and including a negative-depth loss is crucial for training shallow networks from scratch. Without the exclusion, the model will be quickly led astray by the erroneous depth information. For pre-trained deep networks, including such points and turning off the negative-depth loss is not fatal, but still harms performance (as shown in Table II).

V. Conclusions

Our work proposes solutions to important issues in the point transformation phase of the image reconstruction training paradigm for self-supervised monocular depth estimation. We experiment on stereo pairs, demonstrating the effectiveness of our methods on the model's performance through ablation studies and comparisons with past literature.

Our results demonstrate that the parallelized z-buffering algorithm rectifies inconsistencies in the loss functions involving reprojections, allowing for improved learning and test performance with a negligible effect on training time.

Moreover, our algorithm can be easily incorporated with other self-supervised approaches to monocular depth prediction that use reprojection and associated metrics.

In addition, we demonstrate that penalizing in-frame points with negative depth, an unlikely situation, can also improve model performance.

Because they are rooted in a general approach rather than a particular network architecture, these procedural changes potentially offer benefits to a wide variety of monocular depth prediction methods.

Acknowledgment

The authors thank the many anonymous reviewers who made suggestions that helped to improve the work.

References

[1] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *Intl. J. Robotics Res.*, vol. 32, pp. 1231–1237, 2013.

[2] R. Garg, V. K. BG, G. Carneiro, and I. Reid, "Unsupervised CNN for single view depth estimation: Geometry to the rescue," in *Proc. ECCV*, 2016, pp. 740–756.

[3] C. Godard, O. Mac Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *Proc. CVPR*, 2017, pp. 270–279.

[4] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," in *Proc. CVPR*, 2017, pp. 1851–1858.

[5] R. Mahjourian, M. Wicke, and A. Angelova, "Unsupervised learning of depth and ego-motion from monocular video using 3D geometric constraints," in *Proc. CVPR*, 2018, pp. 5667–5675.

[6] C. Wang, J. Miguel Buenaposada, R. Zhu, and S. Lucey, "Learning depth from monocular videos using direct methods," in *Proc. CVPR*, 2018, pp. 2022–2030.

[7] V. Guizilini, R. Ambrus, S. Pillai, A. Raventos, and A. Gaidon, "3D packing for self-supervised monocular depth estimation," in *Proc. CVPR*, 2020, pp. 2485–2494.

[8] 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 *Proc. CVPR*, 2016, pp. 3213–3223.

[9] C. Godard, O. Mac Aodha, M. Firman, and G. J. Brostow, "Digging into self-supervised monocular depth estimation," in *Proc. ICCV*, 2019, pp. 3828–3838.

[10] A. Gordon, H. Li, R. Jonschkowski, and A. Angelova, "Depth from videos in the wild: Unsupervised monocular depth learning from unknown cameras," in *Proc. ICCV*, 2019, pp. 8977–8986.

[11] A. Bhoi, "Monocular depth estimation: A survey," *arXiv preprint arXiv:1901.09402*, 2019.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, 2016, pp. 770–778.

[13] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Proc. MICCAI*, 2015, pp. 234–241.

[14] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *Adv. Neural Info. Proc. Sys. (NIPS)*, 2014, pp. 2366–2374.

[15] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, "Deeper depth prediction with fully convolutional residual networks," in *Proc. 3DV*, 2016, pp. 239–248.

[16] S. Liu, S. De Mello, J. Gu, G. Zhong, M.-H. Yang, and J. Kautz, "Learning affinity via spatial propagation networks," in *Adv. Neural Info. Proc. Sys. (NIPS)*, 2017, pp. 1520–1530.

[17] X. Cheng, P. Wang, and R. Yang, "Depth estimation via affinity learned with convolutional spatial propagation network," in *Proc. ECCV*, 2018, pp. 103–119.

[18] I. Alhashim and P. Wonka, "High quality monocular depth estimation via transfer learning," *arXiv preprint arXiv:1812.11941*, 2018.

[19] 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 1907.10326*, 2019.

[20] Y. Luo, J. Ren, M. Lin, J. Pang, W. Sun, H. Li, and L. Lin, "Single view stereo matching," in *Proc. CVPR*, 2018, pp. 155–163.

[21] G. Iyer, R. K. Ram, J. K. Murthy, and K. M. Krishna, "Calibnet: Geometrically supervised extrinsic calibration using 3d spatial transformer networks," in *Proc. IROS*, 2018, pp. 1110–1117.

[22] W. Straßer, "Schnelle kurven-und flächendarstellung auf grafischen sichtgeräten," Ph.D. dissertation, Technischen Universität Berlin, 1974.

[23] H. Fuchs, "Distributing a visible surface algorithm over multiple processors," in *Proc. ACM Annual Conf.*, 1977.

[24] F. I. Parke, "Simulation and expected performance analysis of multiple processor z-buffer systems," in *Proc. SIGGRAPH*, 1980, p. 48–56.

[25] J.-j. Li and S. Miguet, "Z-buffer on a transputer-based machine," in *Proc. Distributed Memory Computing Conf.*, 1991, pp. 315–316.

[26] C. Renaud, "Fast local and global illuminations through a SIMD z-buffer," *Intl. J. Pattern Rec. and Artificial Intell.*, vol. 11, no. 07, pp. 1095–1112, 1997.

[27] H. Shen, J. You, and D. J. Evans, "An efficient parallel algorithm or visible-surface detection in 3d graphics display," *Intl. J. of Comp. Math.*, vol. 67, no. 3-4, pp. 359–371, 1998.

[28] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.-Y. Lo, J. Johnson, and G. Gkioxari, "Accelerating 3D deep learning with PyTorch3D," *arXiv:2007.08501*, 2020.

[29] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. CVPR*, 2009, pp. 248–255.