DI-Fusion: Online Implicit 3D Reconstruction with Deep Priors

Jiahui Huang  Shi-Sheng Huang  Haoxuan Song  Shi-Min Hu*
BNRist, Department of Computer Science and Technology, Tsinghua University, Beijing

Abstract

Previous online 3D dense reconstruction methods struggle to achieve the balance between memory storage and surface quality, largely due to the usage of stagnant underlying geometry representation, such as TSDF (truncated signed distance functions) or surfels, without any knowledge of the scene priors. In this paper, we present DI-Fusion (Deep Implicit Fusion), based on a novel 3D representation, i.e., Probabilistic Local Implicit Voxels (PLIVoxs), for online 3D reconstruction with a commodity RGB-D camera. Our PLIVox encodes scene priors considering both the local geometry and uncertainty parameterized by a deep neural network. With such deep priors, we are able to perform online implicit 3D reconstruction achieving state-of-the-art camera trajectory estimation accuracy and mapping quality, while achieving better storage efficiency compared with previous online 3D reconstruction approaches. Our implementation is available at https://www.github.com/huangjh-pub/di-fusion.

1. Introduction

Online 3D dense reconstruction has made great progress in the past ten years [3, 6, 12, 25, 38, 39, 51], enabling a wide range of applications including augmented reality, robotic navigation and games. Technically most of the previous depth fusion approaches focus on the globally consistent 3D reconstruction with bundle adjustment [3, 12] or loop closure [51] techniques. However, the underlying representation for the 3D scene itself has seldom changed ever since the success of VoxelHashing [39] with Signed Distance Function (SDF) integration on a sparse set of voxels [10]. This leads to the drawback of previous depth fusion systems that often costs a huge amount of memory storage even for moderate-sized 3D scenes. Besides, the geometric quality could be unsatisfactory with non-complete regions or objects [13] due to the uncertainties caused by sensor noise or scan ambiguities such as view occlusions.

On the other hand, recent efforts from the deep geometry learning community have demonstrated the power of implicit geometric representation parameterized by neural networks [8, 34, 40]. By representing the geometry as a continuous implicit function, the underlying shape can be extracted at arbitrary resolution, which introduces more flexibilities. These methods are also efficient since the network structure used to regress implicit function only consists of simple fully connected layers. Another important feature of such deep implicit representation is the capability to encode geometric priors, which enables many applications such as shape interpolation or reconstruction [40]. This capability can be generalized to scene-level by decomposing and encoding the implicit fields in local voxels, leading to high-quality reconstruction agnostic to semantics [4, 24].

The power of such deep implicit representation motivates us to incorporate it into online 3D dense reconstruction systems. By encoding meaningful scene priors with a continuous function, we can achieve improved surface reconstruction as well as accurate camera trajectory. However, several challenges need to be overcome before this new representation can be successfully applied in an online fusion scenario: (1) geometric uncertainty need to be explicitly modeled against sensor noise or view occlusion, (2) an accurate camera tracking formulation based on such an
implicit representation, which is essential for depth fusion, remains unknown yet, and (3) an efficient surface mapping strategy that incrementally integrates new observations directly is also missing.

We hence respond with DI-Fusion, the first online 3D reconstruction system with tracking and mapping modules fully supported by deep implicit representations. To address the above challenges, we first extend the original local implicit grids [4, 24] and adapt it into a novel Probabilistic Local Implicit Voxels (PLIVox), which encodes not only scene geometry but also the uncertainty with one deep neural network. We show that such an additional uncertainty encoding is extremely useful during the online depth fusion. Based on our PLIVox representation, we devise an approximate gradient for solving the camera tracking problem efficiently. Moreover, thanks to our tailored encoder-decoder network design, we are able to perform geometry integration on the domain of latent vectors, achieving high quality surface mapping in an efficient way. We evaluated our approach on public 3D RGB-D benchmark (ICL-NUIM [20], ScanNet dataset [11]), showing state-of-the-art or improved tracking and mapping quality compared to previous representations. We make our implementation publicly available.

2. Related Works

Online 3D Reconstruction and SLAM. The success of KinectFusion [38] inspired a lot of works on online 3D reconstruction. Early efforts mainly focus on efficient data structures to organize discrete voxels or surfels to enable large-scale 3D reconstruction, such as OctreeFusion [53], VoxelHashing [39], Scalable-VoxelHashing [6] and Elastic-Fusion [26, 51]. In order to improve the quality and the global consistency of the reconstructed model, subsequent works turn to bundle adjustment [12, 25, 50] or sub-map multiway registration [9] techniques. Another line of work couples high-level understanding and geometric modeling including dynamic perception [37, 23, 22] or structural regularization [21]. Our work is also relevant to visual SLAM methods [14, 17, 36, 16], which usually perform camera tracking using sparse features or direct intensity data. Readers are referred to [54] for a more comprehensive study.

Learned Probabilistic Reconstruction. The advancement of geometry deep learning has recently encouraged the community to incorporate deep neural networks into reconstruction and tracking techniques for more accurate representation of either sparse features (DeepTrack [19]), monocular depth observations (CNNSLAM [48], CodeSLAM [2]), or object instances (Fusion++ [33], NodeSLAM [47]). The presence of stochastic sensor noise also motivates many works to consider the probabilistic distribution of the underlying geometry, with either hand-crafted models [15, 52] or data-driven ones [28, 49], where the latter additionally incorporate and explicitly model the approximability of the deep networks [46].

Implicit Representation. The use of implicit function for geometric reconstruction can be dated back to [10], where SDF values are stored in a set of occupied voxels describing the surface. However, even though the implicit function is continuous, the simple discretization [6, 39] introduces drawbacks in surface reconstruction quality and memory storage. As an effort to overcome such drawbacks, [29, 32] propose to model the map using Gaussian Process and perform Bayesian map updates incrementally. In contrast, the use of implicit representation has triggered much interest in deep learning community these years [34, 40]. Typical applications include: part segmentation [7], rendering [30, 35], non-linear fitting [45], meta-learning [44], offline reconstruction [4, 24] etc. Nevertheless, to the best of our knowledge, our DI-Fusion is among the first efforts to incorporate such an implicit representation with deep priors into an online 3D fusion framework.

3. Method

Overview. Given a sequential RGB-D stream, our DI-Fusion incrementally builds up a 3D scene based on a novel
Probabilistic Local Implicit Voxels (PLIVox) representation. Different from previous approaches using discrete voxels without any scene priors, our PLIVox is implicitly parameterized by a neural network and encodes useful local scene priors effectively (Sec. 3.1). Based on this representation, we introduce how to robustly perform camera tracking for each frame (Sec. 3.2) and how to perform efficient incremental surface mapping (Sec. 3.3). As a final step mesh can be extracted at arbitrary resolution thanks to the continuous representation, compared to previous approaches which can only reconstruct at a predefined resolution. This overview of our Dil-Fusion is illustrated in Fig. 2.

3.1. PLIVox Representation

The scene reconstructed is sparsely partitioned into evenly-spaced voxels (PLIVoxs), denoted as $V = \{v_m = (c_m, I_m, w_m)\}$, with $c_m \in \mathbb{R}^3$ being voxel centroid, $I_m \in \mathbb{R}^L$ being the latent vector encoding the scene priors and $w_m \in \mathbb{N}$ being the observation weight. For an arbitrary point measurement $x \in \mathbb{R}^3$, we can efficiently query its corresponding PLIVox index $m(x)$ using simple division and rounding operations $m(x) : \mathbb{R}^3 \rightarrow \mathbb{N}^+$. The local coordinate of $x$ in $v_m(x)$ is calculated as $y = \frac{1}{a}(x - c_m(x)) \in [-\frac{1}{2}, \frac{1}{2})^3$, with $a$ being the voxel size.

Probabilistic Signed Distance Function. Different from the previous approaches representing the underlying 3D surface with signed distance function, we represent it using a probabilistic signed distance function, where the output at every position $y$ is not a SDF but a SDF distribution $s \sim p(\cdot | y)$. In this way, the probabilistic signed distance function encodes the surface geometry and geometric uncertainty at the same time. Here we model the SDF distribution as a canonical Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ with $\mu$ and $\sigma$ being the mean and standard deviation respectively. For a more compact representation, we encode the probabilistic signed distance function with a latent vector $l_m$ using an encoder-decoder deep neural network $\Phi$.

Encoder-Decoder Neural Network. The encoder-decoder neural network $\Phi = \{\phi_E, \phi_D\}$ consists of encoder sub-network $\phi_E$ and decoder sub-network $\phi_D$ (Fig. 3) that share weights with all PLIVoxs.

The target of the encoder $\phi_E$ is to convert the measurements from each depth point observation at frame $t$ to observation latent vectors $l_m^t$. Specifically, for all the RGB-D point measurements located in a PLIVox, $\phi_E$ takes the point measurement’s local coordinate $y$ and normal direction $n$, and transforms them to an $L$-dimensional feature vector $\phi_E(y, n)$ using only FC (Fully Connected) layers. Then the feature vectors from multiple points are aggregated to one latent vector $l_m^t$ using a mean-pooling layer (Fig. 3). Then the normal direction $n$ is required to eliminate the orientation ambiguity within each PLIVox so the sign of SDF can be inferred by the network.

For the decoder $\phi_D$, the concatenation of the local coordinate $y$ and the latent vector $l_m$ are taken as input and the output is a 2-tuple $\{\mu_D, \sigma_D\}$, which represents the Gaussian parameters of the probabilistic signed distance function distribution $p(\cdot | y) \sim \mathcal{N}(\mu_D, \sigma_D^2)$ at position $y$. Note that the two latent vectors $l_m^t$ and $l_m$ in $\phi_E$ and $\phi_D$ are different latent vectors. While the observation latent vector $l_m^t$ encodes the RGB-D observations at frame $t$, the geometry latent vector $l_m$ fuses the previous $l_m^t$ and is stored in each PLIVox used for decoding. Both the observation latent vector and the geometry latent vector have the same dimension, and the geometry latent vector can be updated by the observation latent vector as described in Sec. 3.3.

Network Training. We train the $\phi_E$ and $\phi_D$ jointly in an end-to-end fashion, setting $l_m^t = l_m$. We adopt the training strategy similar to Conditional Neural Process (CNP [18]) but extend it to the 3D domain. Specifically, we first build up two training datasets: (1) $S = \{S_m\}$ for encoder, which is a set of tuples $S_m = \{(y_i, n_i)\}$ for each PLIVox $v_m$ with points $y_i$ and $n_i$ sampled from the scene surface; (2) $D = \{D_m\}$ for the decoder, which consists of tuples $D_m = \{(y_i, s_{gt}^i)\}$ where points $y_i$ are randomly sampled within a PLIVox using a strategy similar to [40] with $s_{gt}^i$ being the SDF at point $y_i$. We give more details about how to build up such two training datasets in Sec. 4. During training, we feed $S_m$ to the encoder for latent vector $l_m$ and concatenate the latent vector with each $y_i$ in $D_m$ to obtain predicted SDF mean and standard deviation. The goal of training is to maximize the likelihood of the dataset $D$ for all training PLIVoxs. Specifically, the loss function $L_m$ for each PLIVox $v_m$ is written as:

$$L_m = -\sum_{(y_i, s_{gt}^i) \in D_m} \log \mathcal{N}(s_{gt}^i; \mu_D(y_i, l_m), \sigma_D^2(y_i, l_m)),$$

$$l_m = \frac{1}{|S_m|} \sum_{(y_i, n_i) \in S_m} \phi_E(y_i, n_i).$$

Additionally, we regularize the norm of the latent vector with a $l_2$-loss which reflects the prior distributions of $l_m$. 

Figure 3. The structure of our encoder-decoder neural network $\Phi$. The encoding and decoding process within one PLIVox is shown. The arrow between encoding and decoding denotes incremental latent vector update if the latent vector from last frame is available.
The final loss function $\mathcal{L}$ we used for training is:

$$\mathcal{L} = \sum_{m \in \mathcal{V}} \mathcal{L}_m + \delta \|I_m\|^2.$$  

### 3.2. Camera Tracking

With our PLIVox encoding of the scene priors including both the scene geometry and uncertainty, we propose a frame-to-model [38] camera tracking method. We claim that the learned deep priors have enough information of the 3D scene for an accurate camera pose estimation, without the need for extra sparse features as in [3, 12]. We propose to formulate the probabilistic signed distance function as an objective function for camera pose estimation, with an approximate gradient for the objective function over camera pose, which makes it converge fast enough during the optimization. Furthermore, our network is efficient in decoding the probabilistic signed distance function, leading to fast online tracking performance.

**Tracking.** We denote the RGB-D observation at frame $t$ as $\Omega^t = \{T^t, D^t\}$ with $T^t$ and $D^t$ being the intensity and depth data. Given camera intrinsic parameters, the depth measurement $D^t$ can be re-projected to 3D as point measurements $\mathcal{P}^t = \pi(D^t)$, where $\pi$ is the projection function and $\pi'$ is its inverse. Our goal is to estimate $\Omega^t$’s camera pose $T^t \in SE(3)$ by optimizing the relative pose $T(\xi^t) = \exp((\xi^t)^\top)(\xi^t \in se(3)$ [1]) between $\Omega^t$ and $\Omega^{t-1}$, i.e., $T^t = T^{t-1}T(\xi^t)$. The following objective function is minimized in our system:

$$E(\xi^t) = E_{sdf}(\xi^t) + wE_{int}(\xi^t),$$  

where $E_{sdf}(\xi^t)$ and $E_{int}(\xi^t)$ are the SDF term and intensity term respectively, and $w$ is a weight parameter. The objective function $E(\xi^t)$ can be efficiently optimized using a Gauss-Newton solver.

**SDF Term $E_{sdf}(\xi^t)$.** The goal of our SDF term is to perform frame-to-model alignment of the point measurements $\mathcal{P}^t$ to the on-surface geometry decoded by $\mathcal{V}$. Different from traditional point-to-plane Iterative Closest Point (ICP) methods with projection association, as our map representation is fully backboned by implicit functions, we choose to minimize the signed distance value of each point in $\mathcal{P}^t$ when transformed by the optimized camera pose. Thus we design the objective function as:

$$E_{sdf}(\xi^t) = \sum_{p' \in \mathcal{P}^t} \rho(r(G(\xi^t, p'))),$$  

$$G(\xi^t, p') = T^{t-1}T(\xi^t)p', \quad r(x) = \frac{\mu_D(x, l_m(x))}{\sigma_D(x, l_m(x))},$$  

where $\rho(\cdot)$ is the Huber robust function, $\{\mu_D, \sigma_D\}$ represents the probabilistic signed distance function decoded by the latent vector $l_m$ from the point measurements. Note that $m(x)$ is a discrete function which may lead to discontinuities across PLIVox boundaries, but we remark that the final term can be effectively smoothed via the summation of all the points in $\mathcal{P}^t$.

One important step to optimize the SDF term is the computation of $r(\cdot)$’s gradient with respect to $\xi^t$, i.e. $\frac{\partial r}{\partial \xi^t}$. Note that $\sigma_D$ and $\mu_D$ are the output for decoder $\phi_D$, so $r(\cdot)$ is an highly non-linear composite function of decoder $\phi_D$, which will lead to poor local approximation. We instead propose to treat $\sigma_D$ to be constant during the local linearization, which will efficiently improve convergence during the optimization. Another benefit is that we can control the influence of $\sigma_D$ up to the zeroth-order approximation, achieving robust tracking performance even when $\sigma_D$ prediction is wrong. Specifically, the approximate gradient is computed by:

$$\frac{\partial r}{\partial \xi^t} = \frac{1}{\sigma_D} \frac{\partial \mu_D(\cdot, l_m(x))}{\partial x} (R^{t-1})^\top (T(\xi^t)p')^\odot,$$

where $R^{t-1}$ is the rotation part of $T^{t-1}$, $p^\odot := [x^t, -y^t]$ and $\frac{\partial \mu_D}{\partial x}$ can be efficiently computed using back-propagation through the decoder network. As an empirical proof of Eq (6), we show in Fig. 4 the comparison of convergence property between our method and the gradient decent (GD) method which performs the update step as $\xi^t \leftarrow \xi^t - \alpha \frac{\partial E_{sdf}}{\partial \xi^t}$ using the precise gradient. Our approximation reaches convergence within a few iterations while achieving comparable estimation accuracy compared to GD, demonstrating the effectiveness of our formulation.

**Intensity Term $E_{int}(\xi^t)$.** We also add a photometric error term between the corresponding intensity data (called intensity term) defined as:

$$E_{int}(\xi^t) = \sum_{u \in \Omega} \left(T[u] - T^{t-1}[\pi(T(\xi^t)p'(u, D^t[u]))]\right)^2.$$

where $\Omega$ is the image domain. This intensity term takes effect when the SDF term fails in areas with fewer geometric details such as wall or floor.
3.3. Surface Mapping

After the camera pose of RGB-D observation $O^t$ is estimated, we need to update the mapping from observation $O^t$ based on the deep implicit representation, by fusing new scene geometry from new observations with noise, which is also referred to as geometry integration. Besides, we also describe an optional mesh extraction step that extracts the on-surface watertight mesh with textures for visualization.

**Geometry Integration.** Instead of updating SDF as previous works [10], we propose to update our deep implicit representation in the domain of latent vectors. We perform the geometry integration by updating the geometry latent vector $l_m$ with the observation latent vector $l'_m$ encoded by the point measurements $P^t$. More specifically, we first transform $P^t$ according to $T^t$ and then estimate the normal of each point measurement, obtaining $X^t = \{(x_i, n_i)\}$. In each PLIVox $v_m$, we calculate the point measurements $Y^t_m \subset X^t$ located in such PLIVox and compute the observation latent vector $l'_m = \frac{1}{w_m} \sum_{(y_i, n_i) \in Y^t_m} \phi_E(y, n)$. The geometry latent vector $l_m$ within such PLIVox $v_m$ is then updated as:

$$l_m \leftarrow \frac{l_m w_m + l'_m w'_m}{w_m + w'_m}, \quad w_m \leftarrow w_m + w'_m,$$  

where the weight $w'_m$ is set to the number of points within the PLIVox as $|Y^t_m|$. In this way, our geometry integration is much more efficient than the previous approaches, since we only need to update the latent vectors within a PLIVox but not the SDFs of massive individual voxels as in Voxel-Hashing [39]. The observation latent vector from the forward pass of $\phi_E$ is also very efficient to compute, which makes our geometry integration fast enough during the online surface mapping without latent vector initialization and optimization used in previous auto-decoder methods [4, 40].

Fig. 5 shows a tiny example demonstrating the effect of incremental geometry integration using our method.

In the meantime, thanks to the robust geometry recovered from the learned deep priors, we do not need to perform integration for every frame. Instead, we choose to integrate sparse frames sampled from every $N$ incoming frames, which improve system efficiency largely while still maintaining tracking accuracy.

**Mesh Extraction.** Optionally we can extract the triangle mesh of the scene surface for visualization purposes. Given a desired resolution during the extraction, we divide each PLIVox into equally-spaced volumetric grids and query the SDFs for each grid with the decoder $\phi_D$ using the PLIVox’s latent vector. Then the final on-surface mesh can be extracted with Marching Cubes [31]. Here, to maintain the continuity across PLIVox boundaries, we double each PLIVox’s domain such that the volumetric grids between neighboring PLIVoxs overlap with each other. The final SDF of each volumetric grid is trilinearly interpolated with the SDFs decoded from the overlapping PLIVoxs [24]. For textures, we simply assign the vertices on the extracted mesh with texture colors averaged from the nearest point measurements back-projected from multiple observations.

4. Experiments

4.1. System Implementation

To train an effective encoder-decoder neural network for descriptive deep priors, we build up two training datasets $(S$ and $D$ for encoder and decoder, respectively, *c.f. Sec. 3.1*), on ShapeNet dataset [5] which contains a large variety of 3D shapes with rich local geometry details. We employ the 6 categories of the 3D shapes from the ShapeNet dataset, *i.e.* bookshelf, display, sofa, chair, lamp, and table, and for each category, we sample 100 3D shapes. Each shape is then divided into equally-spaced PLIVoxs. For each PLIVox $v_m$, we randomly sample $n_d = 4096$ $(y_i, s_{gt}^i)$ tuples for $D_m$, and the count of $(y_i, n_i)$ tuple $n_s$ for each $S_m$ is randomly chosen from 16 to 128. We further augment $D_m$ and $S_m$ by adding random jitter for positions $y_i$ and perturbation for normal direction $n_i$ to train the encoder-decoder neural network with enough robustness against depth observation noise.

We set the length of the latent vector as $L = 29$ in both encoder $\phi_E$ and decoder $\phi_D$ sub-networks. The encoder $\phi_E$ contains 5 fully connected (FC) layers, with the layer sizes set as $(6,32,64,256,29)$. The decoder contains 6 FC layers with the layer sizes set as $(32,128,128,128,128,2)$. We choose to use the Adam optimizer for training the encoder-decoder network with an initial learning rate set as $10^{-3}$. For the regularization on the loss function $L$, we set $\delta = 10^{-2}$.

We manage the PLIVoxs using a similar mechanism as [39] and allocate PLIVoxs only when there are enough
Table 1. Comparison of ATE on ICL-NUIM [20] benchmark (measured in centimeters).

<table>
<thead>
<tr>
<th></th>
<th>lr kt0</th>
<th>lr kt1</th>
<th>lr kt2</th>
<th>lr kt3</th>
</tr>
</thead>
<tbody>
<tr>
<td>DVO-SLAM [27]</td>
<td>10.4</td>
<td>2.9</td>
<td>19.1</td>
<td>15.2</td>
</tr>
<tr>
<td>Surfel Tracking [26]</td>
<td>1.7</td>
<td>1.0</td>
<td>2.2</td>
<td>43.2</td>
</tr>
<tr>
<td>TSDF Tracking [42]</td>
<td>4.5</td>
<td>2.1</td>
<td>1.3</td>
<td>12.5</td>
</tr>
<tr>
<td>Ours (w/o Prob)</td>
<td>1.3</td>
<td>1.8</td>
<td>2.6</td>
<td>15.2</td>
</tr>
<tr>
<td>Ours (max)</td>
<td>1.4</td>
<td>2.0</td>
<td>2.4</td>
<td>7.1</td>
</tr>
<tr>
<td>Ours</td>
<td>1.1</td>
<td>1.4</td>
<td>2.6</td>
<td>6.2</td>
</tr>
</tbody>
</table>

Table 2. Comparison of surface error on ICL-NUIM [20] benchmark (measured in centimeters). σ₀ is thresholded to 0.06.

<table>
<thead>
<tr>
<th></th>
<th>lr kt0</th>
<th>lr kt1</th>
<th>lr kt2</th>
<th>lr kt3</th>
</tr>
</thead>
<tbody>
<tr>
<td>DVO-SLAM [27]</td>
<td>3.2</td>
<td>6.1</td>
<td>11.9</td>
<td>5.3</td>
</tr>
<tr>
<td>Surfel Tracking [26]</td>
<td>1.1</td>
<td>0.7</td>
<td>1.0</td>
<td>22.5</td>
</tr>
<tr>
<td>TSDF Tracking [42]</td>
<td>0.6</td>
<td>1.0</td>
<td>0.8</td>
<td>11.7</td>
</tr>
<tr>
<td>Ours (w/o Prob)</td>
<td>0.7</td>
<td>2.0</td>
<td>1.2</td>
<td>4.8</td>
</tr>
<tr>
<td>Ours (max)</td>
<td>0.8</td>
<td>1.8</td>
<td>1.2</td>
<td>4.8</td>
</tr>
<tr>
<td>Ours</td>
<td>0.6</td>
<td>1.5</td>
<td>1.1</td>
<td>4.5</td>
</tr>
</tbody>
</table>

point measurements gathered (16 in our experiments). Our DI-Fusion system is implemented using PyTorch framework [41]. The jacobian matrix of our tracking term \( Eq \) can be efficiently computed with its auto-differentiation.

4.2. Quantitative Evaluations

**Dataset and Metrics.** In this subsection, we demonstrate the effectiveness of DI-Fusion and its core component, i.e., the PLIVox representation, on ICL-NUIM dataset [20], which contains both the ground-truth camera trajectory and the 3D scene geometry to evaluate the accuracy of depth fusion approaches. We adopt the Absolute Trajectory Error (ATE) metric for camera pose estimation and the surface error [51] for 3D surface quality evaluation.

**Baselines.** Since our algorithm does not contain loop closure component, for a fair comparison we choose to compare with previous depth fusion approaches using TSDF or surfel representation with loop closure turned off. For TSDF-based camera tracking, we adopt the frame-to-model ICP tracker implemented in [42], denoted as ‘TSDF tracking’. For surfel-based camera tracking, we adopt the method in [51], denoted as ‘surfel tracking’. Additionally, we compare to DVO-SLAM [27], which performs frame-to-frame camera tracking directly without the aid of any 3D scene representations. Note that both the depth and intensity information are used in all the baselines.

**Results.** As shown in Tab. 1, our approach achieves lower or comparable ATE compared to the baselines. This is mainly due to two reasons: (1) The deep priors learned in our PLIVox provide a continuous underlying surface prediction than TSDF or surfel, which provides smoother gradient estimation for camera tracking, and (2) the uncertainty estimated by our network also effectively filters out noisy RGB-D observations, thus being less prone to large tracking failure. We find it challenging for our method to fit extremely small structures with the current voxel size \( (\alpha = 10cm) \) primarily due to the size limitation of the prior space. Nevertheless, our method is found to be stable and much more memory-efficient than our counterparts. The surface error comparison shown in Tab. 2 verifies the analysis above. Thanks to the learned deep priors provided by our PLIVox representation, our approach reaches state-of-the-art surface error compared to TSDF or surfel-based representations, with much fewer parameters required as shown in the bottom-left inset of Fig. 6.

4.3. Qualitative Results

We compare the visual quality of the reconstructed 3D surfaces between our approach and the other two camera tracking approaches (TSDF tracking and surfel tracking). We implement TSDF tracking with two configurations: (1) ‘TSDF Low-res’ (low resolution) which allocates larger voxels, so that the number of parameters used to represent the geometry roughly matches the storage cost of our approach; (2) ‘TSDF High-res’ (high resolution, also used in the quantitative comparisons in Sec. 4.2) with a much smaller voxel size for a higher quality 3D surface reconstruction. Fig. 6 shows the visual effect comparison on a scan in [20] among different tracking approaches. Our method can achieve a more complete 3D surface reconstruction while using 91.3% and 95.8% fewer parameters than ‘TSDF High-res’ and surfel tracking respectively to represent the entire scene. If the same amount of memory is used as ours, both the camera tracking and the reconstruction quality would be severely hampered for TSDF approaches as shown in ‘TSDF Low-res’ results.

Additionally, we compare the qualitative results between the different approaches on ScanNet [11], which contains large-scale real-world 3D indoor scans captured with a commodity RGB-D camera. As shown in Fig. 7, our approach can output more preferable 3D reconstructions than the other two baselines due to our PLIVox’s learned encoding of scene priors such that the 3D surface can be accurately recovered at the position even without enough observations. Note that we only use the trained weights on ShapeNet dataset for the encoder-decoder network without any subsequent fine-tuning on ScanNet dataset, showing that our representation has a good generalization performance across different scene types. Fig. 8 shows more illustrations of our approach on ScanNet dataset. Please refer to our supplementary video for further demonstrations.

4.4. System Ablations and Analysis

**Probabilistic Modeling.** To evaluate the effect of using probabilistic SDF for tracking, we ignore the decoder
Figure 6. Qualitative comparisons and memory analysis on the lrkt0 sequence of ICL-NUIM [20] dataset. On the top row we show a global view of the reconstructed 3D scene, where the colors represent the normal directions. Close-up looks at the details are visualized at the bottom-right inset (a) (b). We show the evolution of the number of parameters used in each approach at the bottom-left inset.

Figure 7. Qualitative comparisons on ScanNet [11] dataset. The 1st column of every scene shows a global view of our reconstruction with textures applied while detailed close-up comparisons with other baselines are shown in other columns. The close-up views’ camera positions are plotted on the textured reconstruction.

Figure 8. Other qualitative textured results on ScanNet [11] reconstructed online with our method.

branch outputting $\sigma_D$ and perform camera tracking by assuming the standard deviation as a constant 1.0 (denoted as ‘Ours w/o Prob’). As demonstrated by the consistent

Figure 9. Influence of different parameters on ATE benchmark of ICL-NUIM [20].

worse results than ‘Ours’ for both camera trajectory estimation (Tab. 1) and surface error (Tab. 2), we remark that the way we encode the scene priors in a probabilistic manner is beneficial because erroneous fitting is properly down-weighted.
**Intensity Term.** The intensity term used in our camera tracking (Sec. 3.2) also influences the camera pose estimation accuracy. Here we modify our camera tracking with different weights of the intensity term in the objective function (Eq. 4). The ATE curve on the ICL-NUIM sequence with respect to different intensity weights is illustrated in Fig. 9 (left), showing the complementary effect between the two terms we used: Pure SDF tracking (e.g. $w = 0$) tends to fail in places with few geometric features and too large intensity weights (e.g. $w > 4000$) will ignore the reconstructed scene representation, causing increased drift. In practice, setting the weight of the intensity term between 500 and 2000 leads to the best result, in which case the absolute scale of the SDF term $E_{sdf}$ and intensity term $E_{int}$ for accurate camera pose estimation is roughly balanced.

**Latent Vector Update.** One alternative way for the geometry integration is to update the latent vector using the max operator as in [43] instead of what we propose in Eq (8), i.e. $l_m \leftarrow \max(l_m, l_m')$ and we call this baseline ‘Ours (max)’. Tab. 1 and 2 show the average ATE and surface error of this approach, which are consistently worse than using the mean operator. One possible reason would be that the max operator is sensitive to sensor noise, thus leading to spurious reconstructions, while our averaging update could provide smoother reconstructions against sensor noise. Fig. 10 shows a comparison between the two integration schemes.

**Geometry Integration.** The way we perform the geometry integration at sparse frames sampled from every N frames would also influence the surface mapping quality. Fig. 9 (right) shows the average ATE curve for the camera pose estimation along with different frame integration intervals. The average ATE becomes larger along with the increase of integration interval. In contrast, too frequent integration will introduce excessive sensor noise, which confuses the network and deteriorates the mapping quality.

**Voxel Size.** We show one reconstructed scene using different PLIVox sizes in Fig. 11. By shrinking the PLIVox size $a$, the reconstruction quality is boosted with more desirable details. However, the running time and the memory requirement will also increase accordingly. An engineered implementation with a double-layer PLIVox representation can be employed where large voxels are used for real-time tracking and small voxels are for final mapping.

**Timing.** Overall, our system can run online at 10-12Hz on most modern GPU platforms we test. The cost of each component in our system is reported in Fig. 12. Further code optimizations are possible with parallelization, caching, etc., which we remark as future works. The post-processing time for texturing is $\sim$50ms per integrated frame, with the nearest neighbor search accelerated by a k-d tree.

5. Conclusions

**Limitations.** Our approach has three main limitations: (1) The learned prior could not provide an omnipotent fitting of all possible local geometries especially in the case of over-complicated objects. (2) Each PLIVox is independent and the relationships between neighboring PLIVoxes are not considered; therefore the spatial continuity of the reconstructed scene is not guaranteed. (3) No loop closure component has been incorporated to enforce global consistency. We hope to investigate deeply into both the limitations in the future.

In this paper, we present DI-Fusion, which performs online implicit 3D reconstruction with deep priors. With a novel PLIVox representation, our approach effectively incorporates scene priors of both geometry and uncertainty into joint camera tracking and surface mapping, achieving more accurate camera pose estimation and higher-quality 3D reconstruction. We hope that our method can inspire more future efforts for advanced online 3D reconstruction.

**Acknowledgements.** We thank anonymous reviewers for the valuable discussions. This work was supported by the Natural Science Foundation of China (Project No. 61521002), the Joint NSFC-DFG Research Program (Project No. 61761136018), Research Grant of Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology, and grants from the China Postdoctoral Science Foundation (Grant No.: 2019M660646).

---

**Figure 10.** Comparison with alternative integration method using the max operator. Each column shows the actual frames we perform the integration.

**Figure 11.** Reconstruction results with different PLIVox sizes $a$.

**Figure 12.** Timing analysis. The bar length represents the relative time cost of each component in our system. ‘sdf’ and ‘int’ mean the computation related to $E_{sdf}$ and $E_{int}$, respectively.
References


