SceneFactory: A Workflow-centric and Unified Framework for Incremental Scene Modeling

Yijun Yuan, Michael Bleier, Andreas Nüchter

***Abstract*—We present SceneFactory, a workflow-centric and unified framework for incremental scene modeling, that conveniently supports a wide range of applications, such as (unposed and/or uncalibrated) multi-view depth estimation, LiDAR completion, (dense) RGB-D/RGB-L/Mono/Depth-only reconstruction and SLAM. The workflow-centric design uses multiple blocks as the basis for constructing different production lines. The supported applications, i.e., productions avoid redundancy in their designs. Thus, the focus is placed on each block itself for independent expansion. To support all input combinations, our implementation consists of four building blocks that form SceneFactory: (1) tracking, (2) flexion, (3) depth estimation, and (4) scene reconstruction. The tracking block is based on Mono SLAM and is extended to support RGB-D and RGB-LiDAR (RGB-L) inputs. Flexion is used to convert the depth image (untrackable) into a trackable image. For general-purpose depth estimation, we propose an unposed & uncalibrated multi-view depth estimation model (U^2-MVD) to estimate dense geometry. U^2-MVD exploits dense bundle adjustment to solve for poses, intrinsics, and inverse depth. A semantic-aware ScaleCov step is then introduced to complete the multi-view depth. Relying on U^2-MVD, SceneFactory both supports user-friendly 3D creation (with just images) and bridges the applications of Dense RGB-D and Dense Mono. For high-quality surface and color reconstruction, we propose Dual-purpose Multi-resolutional Neural Points (DM-NPs) for the first surface accessible Surface Color Field design, where we introduce Improved Point Rasterization (IPR) for point cloud based surface query.**

We implement and experiment with SceneFactory to demonstrate its broad applicability and high flexibility. Its quality also competes or exceeds the tightly-coupled state of the art approaches in all tasks. We contribute the code to the community[1].

***Index Terms*—Reconstruction, 3D Modelling, SLAM, RGBD, RGB-Lidar**

I. Introduction

SIMULTANEOUS Localization and Mapping (SLAM) plays an important role in robotics. Previous works have made substantial progress for robust localization of robots [1]–[3], including in challenging environments [4], while the mapping of the environment is often sparse. With the aim of producing higher quality maps, dense mapping [5] and 3D reconstruction [6] have been developed. Dense mapping provides a point set as a mapping representation but is much denser. Alternatively, 3D reconstruction [6] produces meshes for compactness and continuity reasons, which is more user friendly.

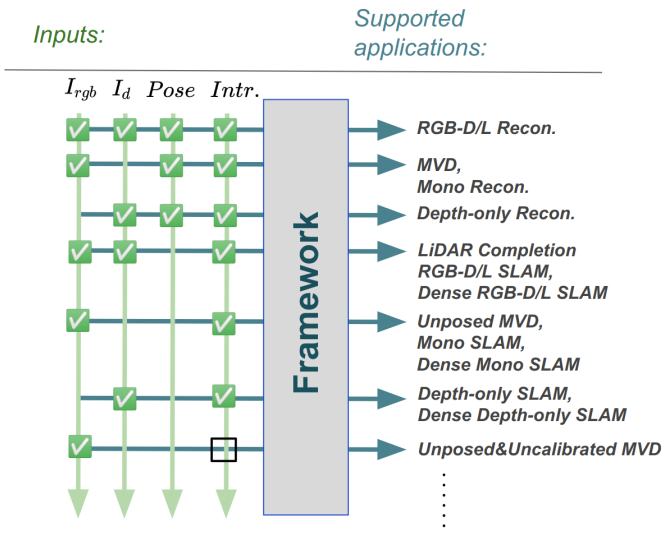


Fig. 1. SceneFactory is workflow-centric and supports a wide range of applications given different input combinations of RGB $\mathbf{I}_{\text{rgb}}$, depth $\mathbf{I}_{\text{d}}$, pose $\mathbf{G}$ and intrinsics θ.

However, we observe that most works construct the whole pipeline in a highly tight-compact design. This restricts the model to a specific application and is difficult to upgrade each submodule. This leads to our first research question: *Can we design a framework that will support all of the production lines?*

First of all, we traverse all possible input cases and look at what the tasks might be. As shown in Fig. 1, given different input combinations of RGB $\mathbf{I}_{\text{rgb}}$, depth $\mathbf{I}_{\text{d}}$, pose $\mathbf{G}$ and intrinsics θ, a number of applications could be run. And our goal at the very beginning is to *support them all* in a single framework.

To achieve this goal, we design our framework in a workflow-centric fashion. The advantage of such a design philosophy is that all tasks are connected by a dependency graph to reduce redundancy. For example, as in our design Fig. 2, one final task usually depends on the output from other tasks. However, the other tasks can also be the final tasks. All subtasks work both independently and together, each block must be completed on its own. So we have four

Yijun Yuan is with Tsinghua University, Beijing, China. Yijun Yuan, Michael Bleier and Andreas Nüchter are with Julius-Maximilians-Universität Würzburg, Germany. Andreas Nüchter is also with the Zentrum für Telematik e.V., Würzburg and currently International Visiting Chair at U2IS, ENSTA, Institut Polytechnique de Paris, France. The research was done at Julius-Maximilians-Universität Würzburg. Contact: {yijun.yuan|michael.bleier|andreas.nuechter}@uni-wuerzburg.de

This work was in parts supported by the Federal Ministry for Economic Affairs and Climate Action (BMWK) on the basis of a decision by the German Bundestag und the grant number KK5150104GM1. We also acknowledge the support by the Elite Network Bavaria (ENB) through the "Satellite Technology" academic program.

[1]https://jarrome.github.io/SceneFactory/

blocks in the workflow: tracking, flexion, depth estimation, and reconstruction.

On the tracking block, we subtract and find the greatest common divisor, Mono SLAM [7], [8]. Mono SLAM considers more complicated conditions than RGB-D and Stereo SLAM due to the difficult depth estimation. Which operates the simplest sensor setup with fewer observation requirements. However, Mono SLAM relies on only sparse keypoints, and thus we also introduce a dense estimation task for dense depth-related applications. The "loose connection" concept directs us to build a dense SLAM where each module focuses on its own task.

When there is no RGB input, but depth input, we exploit the seminal work of [9], called flexion. Flexion image is based on surface curvature, where in each pixel it calculates the difference between two normals from the neighboring pixels. To make it compatible with our vision-based tracking block, we have to rely on trackable images. However, the depth image does not contain SE(3)-invariant features under different viewpoints. So the flexion is applied then to convert the untrackable/unmatchable depth images into trackable images [9].

Then comes depth estimation, which has been widely studied with monocular and multi-view input. While monocular depth [10] naturally loses scale, the reliable methods usually operate multiple views. Such methods are classified as (1) depth-from-video without known poses [11]–[13], (2) multi-view stereo with known poses [14]–[19] and (3) pointmaps without poses and intrinsics [20]. In our depth estimation module, we want to support all of them. For high-reliability, we rely on a low-level vision model, optical flow [21] for correlation search, and use dense bundle adjustment to estimate the dense structure. Here, all camera intrinsics, poses and depths are estimated in one optimization problem.

For use in SLAM sequences, we further introduce *good neighborhood selection* aiming to reduce the above situations. However, even with this selection, the side effect of producing missing regions still exists. An example is in Section V, where there is a hole in the wall. Inspired by MonoSDF [22], that the monocular depth shares point properties with the real depth, we turn to monodepth and find in our scene case that although monodepth is distorted between structures, in-structure coplanarity is preserved. Therefore, we propose to utilize this in-structure property with the ScaleCov model. It relies on the learned deep covariance to fill the clipped locations with regressed scale image. In SLAM-related applications, we additionally scale the depth to global to make the mapping consistent.

Although our depth estimation model fits well into mono SLAM, the model on its own supports unposed & uncalibrated multi-view depth estimation. This also provides a user-friendly application that requires only RGB images.

Afterwards, we perform the 3D reconstruction. Recently, there have been some successful attempts for immersive visualization under 3D reconstruction pipeline, such as ESLAM [23] and NeRF-SLAM [24]. They borrow the NeRF-like training scheme to produce high-quality view synthesis. However, the ray integration naturally suffers from many samples with large space and time costs. Therefore, we turn to another related branch, Neural Surface Light Fields (NSLF), without geometry optimization. Recent research of NSLF has verified the efficiency on SLAM sequences [25]. Because NSLF only learns the surface field, each surface position is not modified. However, the existing online SLF method [25] requires an external SDF model [26] to provide the surface points. Therefore, we propose here dual-purpose multi-resolutional neural points (DM-NPs) to overcome this limitation. Following NSLF-OL [25], which uses a multi-resolutional hash grid for efficient online learning, we create a *multi-resolutional point grid for fast SLF learning*. Moreover, since our point-based representation naturally provides position on surfaces, a straightforward idea is to use point rasterization. However, point rasterization is a dysfunction compared to surface rasterization (as introduced in Section IV-C). To make it work, we propose an *Improved Point Rasterization* which turns out to be really useful compared to the point rasterization of Pytorch3D [27]. Our CUDA implementation is 10 to 50 times faster than Pytorch3D's CUDA implementation.

In summary, as shown in Fig. 1, we intend to design a modular scene modeling method, where each module supports individual applications, while they can be combined into one.

The contributions of this work are:

1) We propose a workflow-centric framework, SceneFactory, that supports *all incremental scene modeling* applications with different combinations under the connection of a dependency graph.
2) We introduce a dual-purpose multiresolution neural points representation for both Surface Light Fields (SLF) and Improved Point Rasterization (IPR). This is (1) *the first surface-gettable SLF model*, (2) *the first to make point rasterization as usable as surface rasterization.*
3) We give a robust depth estimation block with (1) *an unposed & uncalibrated multiview depth estimation model (U^2-MVD)* and (2) *a deep correlation kernel-based depth completion model, ScaleCov*.
4) We capture the *first dense mono SLAM purpose RGB-X dataset* for high-quality monocular reconstruction.

In the following, we first describe the related work on dense monocular SLAM, neural rendering in SLAM and multi-view depth estimation. Then, in three separate sections, we introduce the overall SceneFactory structure, the dual-purpose multi-resolution neural point representation and the unposed & uncalibrated depth estimation. Then, we conduct experiments to thoroughly evaluate the performance of the system. Finally, we conclude this paper and attach the supplementary.

II. Related Works

A. Dense SLAM

Dense SLAMs are the main application of SceneFactory. Originally for reliable reconstruction, dense SLAM systems operate the metric dense depth from laser scanners or RGB-D sensors and focus more on the mapping representation, to approach a high-quality surface reconstruction. When metric depth is involved, traditional fusion methods [28] incrementally update the Truncated Signed Distance Field (TSDF) and then extract afterwards a mesh using Marching Cubes [29].

More recently, neural priors have been used for Neural Implicit Representation [26], [30]. Uni-Fusion's implicit representation supports even more data properties without any training [31]. With the even more recent trend of volumetric rendering, trained neural implicit rendering methods also come into view [25], [32].

However, the application scenarios of scanners and depth sensors are still limited by their low affordability, portability and accessibility. Alternatively, almost everyone has their own monocular camera at hand, e.g., on a mobile phone. Therefore, a lot of work has been done in recent years to explore dense SLAM with a monocular camera [33]–[37]. For Dense RGB SLAM, optimizing an entire sequence of depths is not feasible given the large number of variables involved. To reduce the computational costs of depth estimation, CodeSLAM [33] and DeepFactors [34] optimize the latent codes of depth images. However, the single-frame depth encoder-decoder needs to be pre-trained with similar data sets. And mono-depth naturally loses scale and intrinsic information, usually resulting in distorted structures. Instead of optimizing poses and depths at the same time, Tandem [35] first solves frame poses and then uses the pre-trained MVSNet to recover the dense structure. However, the MVSNets are usually overfitted with the trained camera intrinsic. This is suboptimal for custom cameras and new scenes. On the other hand, DROID-SLAM [36] ensembles dense optical flow modules into the pipeline, and optimizes poses and downsampled inverse depths using dense bundle adjustment. To cope with DROID-SLAM's noisy depth estimation, Sigma-Fusion [37] introduces depth uncertainty into DROID-SLAM's framework and uses TSDF to provide a high-quality dense reconstruction.

Starting in 2022, more researchers realized the high potential of using NeRF for high-quality mapping. Orbeez-SLAM [38] and NeRF-SLAM [39] first embedded NeRF in SOTA-SLAM frameworks. For example, Orbeez-SLAM generates a sparse map using OrbSLAM3 [40] to set up a coarse occupancy grid for on-ray point sampling. NeRF is then applied directly given the tracked poses. Orbeez-SLAM gets high-quality rendering from NeRF. However we find that it also inherits the problem of NeRF, especially for SLAM sequences. It hardly works with non-around-object sequences. While based on Sigma-Fusion, which provides dense depth in nature, NeRF-SLAM addresses the above problem by monitoring depth along with color.

From the design of Orbeez-SLAM [38] and NeRF-SLAM [39], that they have loosely connected pose-and-depth and depth-and-reconstruction relations respectively, we find that the loosely coupling does not just provide high flexibility, but also shows high advances when each sub-task is mature, such as the use of OrbSLAM3 and etc. This point inspires us to make a workflow-centric design that has high expectations for each submodule.

B. Neural Rendering in SLAM

In our view, neural rendering is a topic that has the potential to replace reconstruction for high-quality mapping.

Many 3D reconstruction algorithms rely on marching cubes to extract mesh from explicit [28] or implicit [26], [30], [31]. This is not an efficient update due to the complicated (implicit)-to-explicit-to-mesh steps. While the alternative view synthesis relying on rendering, shows a more direct way to extract visualization from implicit representations. View generation could be super efficient.

The success of neural rendering in SLAM begins with the invention of iMAP [41] and NICE-SLAM [32]. iMAP applies volumetric rendering to MLPs and optimizes poses and MLPs with photometric loss. NICE-SLAM improves the rendering base and replaces the MLPs with hierarchical feature grids. This further improves the surface quality. NICE-SLAM certainly provides a good basis for neural rendering. However, it still has the problems that 1. it is not real-time capable, 2. the color result is of low quality.

Orbeez-SLAM [38] and NeRF-SLAM [39] address these by coupling SOTA-SLAM with SOTA-NeRF, instant-ngp [42]. NSLF-OL [25], on the other hand, focuses only on surface color and produces a real-time neural surface light field model to be used in conjunction with an external real-time reconstruction model.

In our SceneFactory design, we expect high for each module. Thus the reconstruction model does not worry about the inputs, e.g. depth. So NSLF-OL is a good starting point. However, working alongside other model makes SLF's performance highly dependent on the hosting reconstruction algorithm. Therefore, in this work, we also propose to make the SLF model supporting surface extraction in the same representation.

C. Multi-view Depth Estimation

Depth from multi-view is obtainable in multiple ways, i.e., as depth-from-video, multi-view stereo, and pointmaps.

Depth-from-video directly estimates depth images and camera poses from video sequences with known camera intrinsics. DeMoN [11] has firstly utilized deep learning techniques to estimate the depth and motion with a single network. DeepTAM [12] and DeepV2D [13] do not only work on image pairs, but process more images with alternating mapping and tracking modules. However, according to [43], such methods are overfitted to the trained scale and camera parameters, which makes it difficult to generalize to arbitrary real-world applications.

Multi-view stereo estimates 3D geometry from unconstrained images with given intrinsic and extrinsic information. MVS is starting to get a boost with the trend of deep learning. DeepMVS [14], as the first deep network-based method, aggregates sampled patchwise deep features to estimate full image depth. It has demonstrated the high potential of high-quality depth estimation with deep learning. Similarly, MVSNet [15] learns a feature map for whole images and fuses multi-view information into one feature map. Then MVSNet's design serves as mainstream for following deep learning models [16]–[19]. But, as mentioned before, MVS nets overfit the trained camera and scene. This also leads to malfunction in new real scenes.

Pointmaps methods start attracting more attention in 2024. It uses a dense 2D field of 3D points as geometric representation.

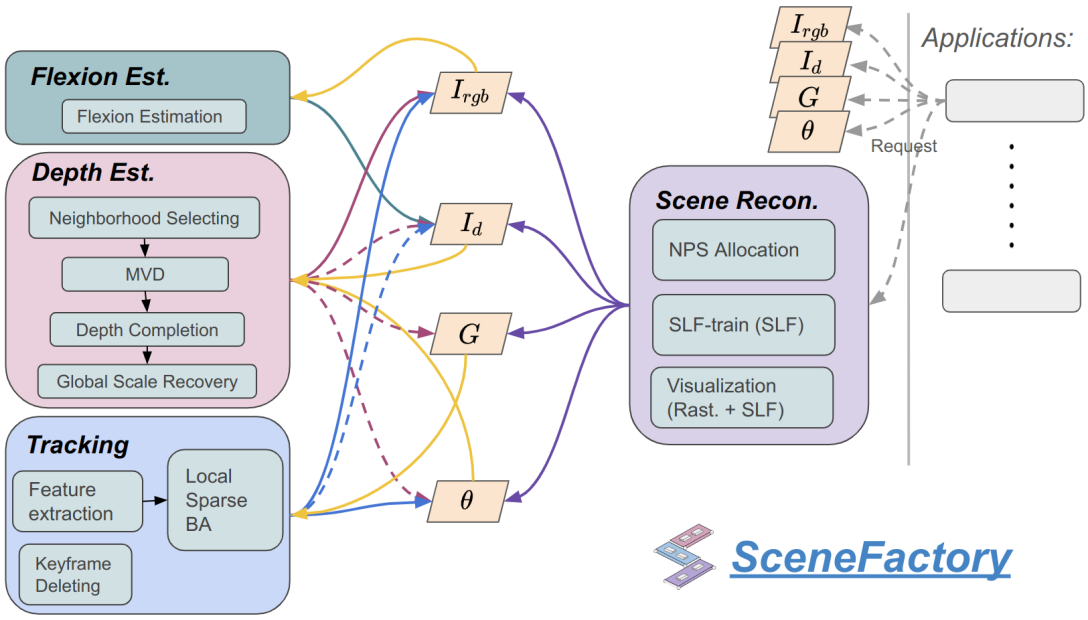


Fig. 2. The dependency graph in SceneFactory. SceneFactory sends requests to its dependent sub-tasks (inputs/blocks). If the dependent sub-tasks are not complete, then each sub-task will call its corresponding dependent sub-tasks. The gray line indicates the requirement of a specific application. The yellow line shows the dependency of input (RGB $\mathbf{I}_{rgb}$, depth $\mathbf{I}_d$, pose $\mathbf{G}$ and intrinsics θ), which is triggered when the input value is None. The green, purple, blue, and pink lines show the dependencies of the Flexion, Depth, Tracking, and Scene Reconstruction blocks. The solid and dotted lines show mandatory and optional dependencies. Inside each block, the functions are applied one after each other, as shown by the black arrows.

Pointmaps are first used in visual localization [44], [45] and monocular 3D reconstruction [46], [47]. A recent pointmaps-based work DUSt3R [20], designs a flexible stereo 3D reconstruction model. DUSt3R unifies different 3D tasks in a groundbreaking way. For the first time, it also provides a user-friendly interface that does not require camera parameters.

Our depth estimation module is also partly inspired by the high-flexibility of the DUSt3R. We want to provide a user friendly interface like DUSt3R. But further, for supporting all above settings.

III. A Unified Framework for Incremental Scene Modeling

SceneFactory supports various combinations of RGB $\mathbf{I}_{rgb}$, depth $\mathbf{I}_d$, pose $\mathbf{G}$ and intrinsics θ as input. We plot the entire pipeline in Fig. 2 to have a brief overview. All of the applications in Fig. 1 find their corresponding parts in this diagram.

SceneFactory consists of four building blocks:

- tracking block to complete pose,
- flexion estimation block to complete the image,
- depth estimation block to estimate and complete the depth,
- scene reconstructing block.

In this section, we introduce each of these blocks respectively (Sections III-A to III-D) and combine them into a whole workflow (Section III-E).

Note that, the last two (depth estimation and scene reconstruction blocks) are our contribution models, we add two more sections (Section IV and Section V) after this framework section for a detailed explanation.

A. Tracking Block

For the tracking block, we use Mono SLAM because it is a greatest common divisor with minimal demands. More specifically, our tracking block is based on DPVO [48], while we generalize it to also support RGB-D/L input.

DPVO is based on sparse correspondences, while the pose $\mathbf{G}$ is optimized with sparse bundle adjustment:

$$\mathcal{L}_{track}(\mathbf{G},\mathbf{P}) = \sum_{(k,j)\in\mathcal{E}} \| \Pi(\mathbf{G}_{ij} \circ \Pi^{-1}(\mathbf{P}_k^{'})) - [\hat{\mathbf{P}}_{kj}^{'} + \delta_{kj}] \|_{\Sigma_{kj}}^2 . \tag{1}$$

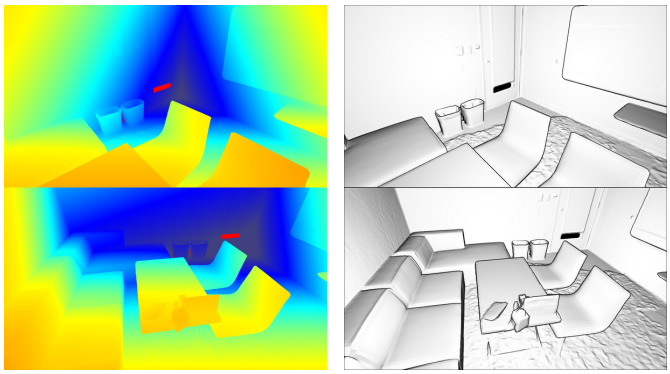
where $\mathcal{E}$ and Π denote the edges and projection, $\mathbf{P}_k^{'}$ is the patch k in image i, $\hat{\mathbf{P}}_{kj}^{'}$ is the center of patch $\mathbf{P}_{kj}^{'}$ in image j, δ_{kj} is the patch update.

The above formulation is efficiently solved with Gauss-Newton. When the depth $\mathbf{I}_d$ is given, we extract patch $\mathbf{P}$ from the corresponding position. We fix the patch from a valid depth pixel and optimize only the poses $\mathbf{G}$ from $\mathcal{L}_{track}$. We choose DPVO for the generalization because it does not require explicit extraction of matches, and thus well fits the sparse observation of the LiDAR. From our experiments we learnt that this generalized version is more accurate by utilizing the metric depth.

B. Flexion Estimation Block

Flexion estimation is required when $\mathbf{I}_{rgb}$ is missing, i.e., depth only is given.

To support depth-only tracking in our RGBD SLAM framework, a feature matching operation is required. However, the depth image does not contain SE3-invariant context for descriptor extraction. This is not trackable in our setting. Therefore, we use the Flexion depth converter [9] to convert



(a) Colorized depth (b) Flexion

Fig. 3. Depth images (left) and their corresponding trackable converted flexion images (right).

the depth image into a flexion image that has the SE3-invariant properties. An example is given in Section III-B. We colorize the depth only for better visualization. We see that flexion images contain a more consistent value, which is more suitable for feature matching [9]. Hence, depth-only applications are able to treat flexion as RGB and serve as RGBD applications as in Fig. 20.

C. Depth Estimation Block

The depth estimation block is mainly our U^2-MVD (in Section V) with intrinsics given for pose-free MVD to support incremental application. It computes the depth for RGB applications, allowing them to work as RGB-D applications.

Note that when no metric depth is specified, tracking frames only have sparse landmarks. Depth estimation is then required when keyframe (KF) and good neighbor frames (NFs) are detected or MVD is called for certain application requests. Once requested, this block will query memory for good neighbor selection (Section V-C), pose-free MVD (Section V-A), depth completion (Section V-B) and scale recovery (Section V-D).

We put the depth estimation after the tracking sequences because the tracker optimizes poses in a local window, and the updated landmarks will affect the scale recovery.

D. Reconstruction Block

The reconstruction block uses our DM-NPs presented in Section IV. This block maintains two threads: 1) NPs allocation and online learning, and 2) visualization.

1) Online-learning thread: A training thread is used to continuously train our SLF model. Similar to the online method, NSLF-OL [25], DM-NPs takes a new keyframe ($\mathbf{I}'_{rgb}$, $\mathbf{I}'_{d}$, $\mathbf{G}'$) and converts it to a colored point cloud $\{(\mathbf{p}_n, \mathbf{c}_n)_{n\in\{1,\cdots\}}\}$. Based on the positions of point cloud, new neural points (NPs) will be allocated (as in Section IV-A1). The input color value $\{\mathbf{c}_n\}_n$ of the point cloud will be utilized to supervise the prediction $\{\mathbf{c}^*_n\}_n$ (as in Section IV-A3).

2) Visualization thread: Another thread for the renderer relies on our Improved Point Rasterization (Section IV-C) for surface extraction, and color prediction (Section IV-A3) on surface. Our interactive GUI receives signals from the user to rotate and move the view camera. Rendering is done in real-time for a first-person view of the scene (Section IV-D).

E. Main Function

SceneFactory's main functions are $EstablishProductLine$ and $Step$ as in Algorithm 1.

Algorithm 1: Main functions

Function $EstablishProductLine$ ($(\mathbf{I}_{rgb}, \mathbf{I}_d, \mathbf{G}, \theta, app)$) **:**
  // Build product line
  $pLine = AssemblingParts(\mathbf{I}_{rgb}, \mathbf{I}_d, \mathbf{G}, \theta, app)$;
  return $pLine$;
End Function
Function $pLine.Step$ ($(\mathbf{I}_{rgb}, \mathbf{I}_d, \mathbf{G}, \theta, app)$) **:**
  // Product line start working part-by-part
  $\mathbf{V}_{inter} \leftarrow pLine.Package(\mathbf{I}_{rgb}, \mathbf{I}_d, \mathbf{G}, \theta, app)$;
  for f_{part} *in pLine.parts* **do**
    $\mathbf{V}_{inter} = f_{part}(\mathbf{V}_{inter})$
  end
  return $pLine.Unpackage(\mathbf{V}_{inter})$;
End Function

When a task is triggered, SceneFactory successively checks the availability of image $\mathbf{I}_{rgb}$, pose $\mathbf{G}$, depth $\mathbf{I}_d$ and intrinsics θ, to prepare the production line. Then, the operation is conducted following Fig. 2.

For example, if $\mathbf{I}_{rgb}$ is missing (depth only), the flexion block (Section III-B) is added to the product line to estimate the trackable image from depth. If SceneFactory is required to solve for the intrinsics without a θ input, it will add intrinsic-free U^2-MVD to the line for θ. If SceneFactory is asked to solve for poses without poses $\mathbf{G}$ as input, it will add a tracker to the line for predicting the pose $\mathbf{G}'$. If the metric depth is missing, the depth estimator block (Section III-C) is added to estimate the depth. While if the metric depth input is sparse, e.g., from a LiDAR, only the depth completion part will be added. When reconstruction is requested by the application, the completed frame parameters with image, depth, pose and intrinsics are required to be fed into the reconstruction block (Section III-D) for online learning and visualization. And the corresponding parts are all added to the product line.

Then the production line treats each step function as a task. If a single task is requested (such as MVD, Completion and etc.), the production line will return the step result as a product. While if sequential application is requested (such as SLAM, Reconstruction and etc.), the production line will conditionally step on each frame for the intermediate and return the product at the end of this production process.

IV. Dual-purposes Multiresolutional Neural Points

In this section, we introduce the dual-purpose representation that simultaneously supports Surface Light Fields (SLF) and point rasterization. This representation is the first SLF method to support *surface querying*.

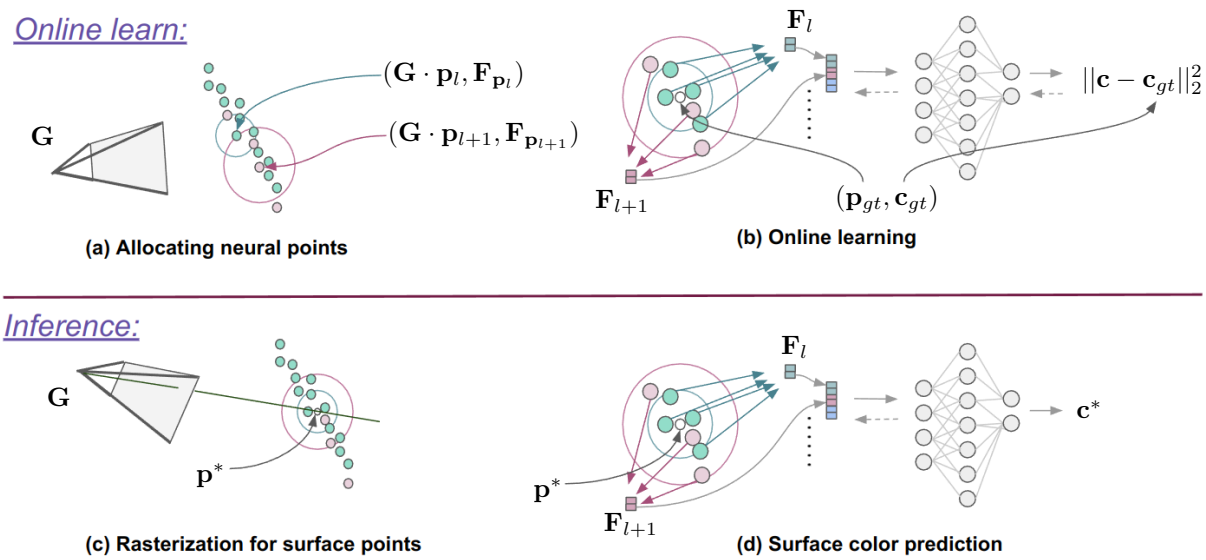


Fig. 4. Illustration of the multiresolution neural points in 2D. The top row indicates the (a) allocating and (b) training during online learning. The bottom row shows the (c) rasterization and (d) color prediction for visualization. We use multiple levels of neural points, for example, green dots for low level and pink dots for higher level. The corresponding circle indicate the resolution of that levels of points.

A. Multiresolutional Neural Points

Intuitively, a neural point is a point with a feature. We denote a neural point as $\mathbf{v} = (\mathbf{F}_v, \mathbf{p}_v)$ with feature $\mathbf{F}_v \in \mathbb{R}^m$ and position $\mathbf{p}_v \in \mathbb{R}^3$.

Then, inspired by instant-ngp [42], we use multiresolution neural points to improve learning efficiency. These are multiple sets of neural points with different densities. We denote them as $\mathbf{V}_a$ the set of $\mathbf{v}$ with resolution/density a. Fig. 4 shows a illustration.

During training, we follow NSLF-OL [25] to feed the set of colored pairs $(\mathbf{p}_i, \mathbf{c}_i)_i$ for coding and training. For each point position, we examine the density in its region and assign neural points. Then, the MLP coding is applied.

1) Neural Points Allocation: For the feedpoint set $\mathbf{Q} = \{\mathbf{p}_i\}_i$, given a set of resolutions $\mathbf{a}$, we downsample to get $\{\mathbf{Q}_a\}_{a\in\mathbf{a}}$. Then, as depicted in Fig. 4 (a), for each point $\mathbf{p}_i$ in $\mathbf{Q}_a$, we check the closest distance to a level neural points, if distance is greater than threshold t_a, the set of neural points is extended by adding this point and assigning it initialized with the feature.

2) Surface Points Encoding: SLF's inference is on the surface, so this coding is an interpolation of the feature. For inference point $\mathbf{p} \in \mathbf{Q}$, we utilize an efficient K-d tree to find its K nearest neighbor ($\{\mathbf{v}_k^a\}_{k\in\{1,\cdots,K\}}$ with distance $\{d_k^a\}_{k\in\{1,\cdots,K\}}$) of inference point from each level as in as in Fig. 4 (b). The a level feature is

$$F_{\mathbf{p}}^a = \sum_{i\in\{1,\cdots,K\}} w_i F_{v_i}^a, \tag{2}$$

where $w_i = exp(-\frac{{d_i}^2}{\sigma})/\sum_j exp(-\frac{{d_j}^2}{\sigma})$.

3) Color Prediction: The color prediction is generated by concatenating features along different levels and decoded with MLP: $f_{\text{MLP}} \circ f_{\text{concat}}((F_{\mathbf{p}}^a)_{a\in\mathbf{a}})$.

B. Mapping via Online Learning

We learn this SLF in an online fashion by continuously feeding data according to [25]. A main thread of this module is to continuously train the SLF. As soon as a new posed image with depth is input to the renderer, it is assigned a trained iteration, it_{trained}, to train in a least-touch strategy.

During the training iteration of a given frame, it randomly samples n_{train} pixels and passes the corresponding point pairs $\{(\mathbf{p}_n, \mathbf{c}_n)_{n\in\{1,\cdots,n_{\text{train}}\}}\}$ to the neural allocation Section IV-A1 and prediction Section IV-A3 for the resulting color $\{\mathbf{c}_n^*\}_n$.

We compute the MSE

$$L_{\text{MSE}} = \sum_{n\in\{1,\cdots,n_{\text{train}}\}} \| \mathbf{c}_n^* - \mathbf{c}_n \|^2 \tag{3}$$

and use the Adam optimizer for stochastic gradient descent optimization of the features of neural points and MLP. In addition, we use the following two strategies to speed up the training.

1) Jump-start Training Strategy: From the experiments, we find that our SLF converges slowly on the first frame, while super fast for the following frames if the first frame gets coarse color. Otherwise, the other frames would also be relatively slow.

We think the problem arises from the chaos of parameters as it is randomly initialized. So we set a $10\times$ higher learning rate for the first 5 iterations and recover then. This way, even the first image converges in a second.

2) Least-trained First Training Strategy: The second training strategy is that after the first frame, when another frame is input, the previous well-trained frame shouldn't have the same chance to be trained as the new one. So an intuitive way is to always train the least trained frame. In this way, all subsequent frames are converged in one second.

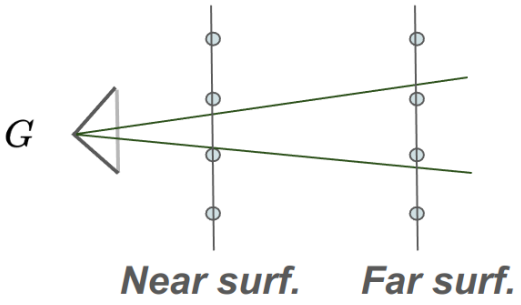


Fig. 5. Near-far-imbalance. The ray penetrates the near surface.

C. Improved Point Rasterization (IPR)

Above we explained the training process where the depth is given. During the inference, we have to estimate the depth. To this end, point rasterization is introduced as in Fig. 4 (c). For simplicity, we will now operate on points in camera space. To render an image with a resolution of $H \times W$, we cast a batch of rays $(\mathbf{o}_i = \mathbf{0}, \mathbf{d}_i \in \mathbb{S}^2)_{i \in \{1,\cdots,HW\}}$ with $(\mathbf{o}, \mathbf{d})$ as the ray source and direction. In implementation, the positions of the neural points are first transformed into NDC space.

Setting camera space point $\mathbf{p} = [X, Y, Z]^T$ as an example, the projected point in NDC is then $\mathbf{p}_{\text{ndc}} = [f_x \frac{X}{Z} + p_x, f_y \frac{Y}{Z} + p_y, \frac{1}{Z}]^T$, where (f_x, f_y, p_x, p_y) are intrinsic parameters. In this way, each ray is directed to $+z_{\text{ndc}}$, i.e., $\mathbf{d}_{\text{ndc}} = [0, 0, 1]^T$. The source of the NDC space ray is obtained by projecting $\mathbf{d}$ accordingly.

Points within a radius r_{ndc} around each ray $(\mathbf{o}_{\text{ndc}} = [x_{\text{ndc}}, y_{\text{ndc}}, z_{\text{ndc}}]^T, \mathbf{d}_{\text{ndc}})$ are then computed using only the (x, y) coordinate.

Thus, for each ray $(\mathbf{o}, \mathbf{d})_i$, we find its K_{ray} nearest neural point neighbors and the distance $\{(\mathbf{p}_{nb,k}, d_{nb,k})\}_{k \in \{1,\cdots,K_{\text{ray}}\}}$ within the radius. The rasterized point is thus

$$\mathbf{p}_{\text{raster}} = \sum_{k \in \{1,\cdots,K\}} w_{nb,k} \mathbf{p}_{nb,k}, \tag{4}$$

where $w_{nb,k} = exp(-\frac{{d_k}^2}{\sigma}) / \sum_j exp(-\frac{{d_j}^2}{\sigma})$. However, unlike mesh rasterization, point rasterization suffers from the following problems:

1) the point distribution in NDC space is distorted. That is, the points near the camera are more sparse while the points far from the camera are more dense compared to Euclidean space. This results in holes in the near regions.
2) the rasterization may contain several layers of points.

To deal with the hole problem, we introduce an adaptive radius to PR. This changes the radius depending on the depth.

Considering that our finest resolution is $r = 0.005m$ and the screen space is at $z = 1$, we expect that each point with a $0.005m$ gap can completely cover a pixel at $z = 1$. The coverage radius should be $l_{\text{coverage}} = \sqrt{2}(r \cdot f_{im})$ where $f_i m$ is the focal length parameter. To simplify the implementation, our CUDA code creates a $l_{\text{coverage}} \times l_{\text{coverage}}$ window for this point.

If $z < 1$, the scenario in Fig. 5 occurs because the points are distorted in the screen space. The ray will then penetrate the near surface. To solve this problem, we increase the coverage area, the coverage length becomes $l_{\text{coverage}} = \sqrt{2}(r \cdot f_{im})/z$.

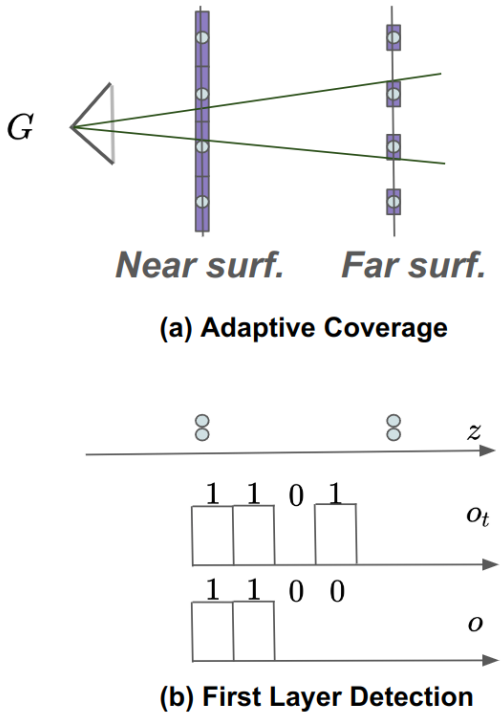


Fig. 6. Improved Point Rasterization. Increasing the coverage area (above) avoids the Near-far-imbalace, cf. Fig Fig. 5 and the first-layer detection keeps only these points (below).

For the second problem, i.e., the problem of multiple layers, we propose to use first-layer detection to keep only the points of the first layer. Given n sorted points along the ascending z-axis, which are traced by a given ray, $\{z_0, \cdots z_n\}$, we compute the occupancy of point z_t.

$$o_t = \begin{cases} 1 & t = 0, \\ z_{t+1} - z_t < th & t > 0. \end{cases} \tag{5}$$

Then, by cumulating production, we get the mask of the first layer points:

$$o_{\{0,\cdots\}} = cumprod(o_{\{0,\cdots\}}). \tag{6}$$

We implement our IPR with CUDA, which is about 20 times faster than Pytorch3d's CUDA implementation (40 FPS vs 2 FPS). We demonstrate the effect of IPR in Suppl.-B.

D. Visualization

This function plays an important role in rasterizing the depth image and in extracting the surface point colors under certain viewpoints. First, we rasterize at the level of neural points with the highest resolution. Then, transforming back to world coordinates with $\mathbf{T}$, the color prediction is obtained as in Fig. 4 (d), by

$$\mathbf{c}_{raster} = f_{MLP} \circ f_{concat}((F^a_{\mathbf{T}\mathbf{p}_{raster}})_{a \in \mathbf{a}}). \tag{7}$$

We follow [25] to create another thread with an interactive GUI to provide a first-person view of the scene.

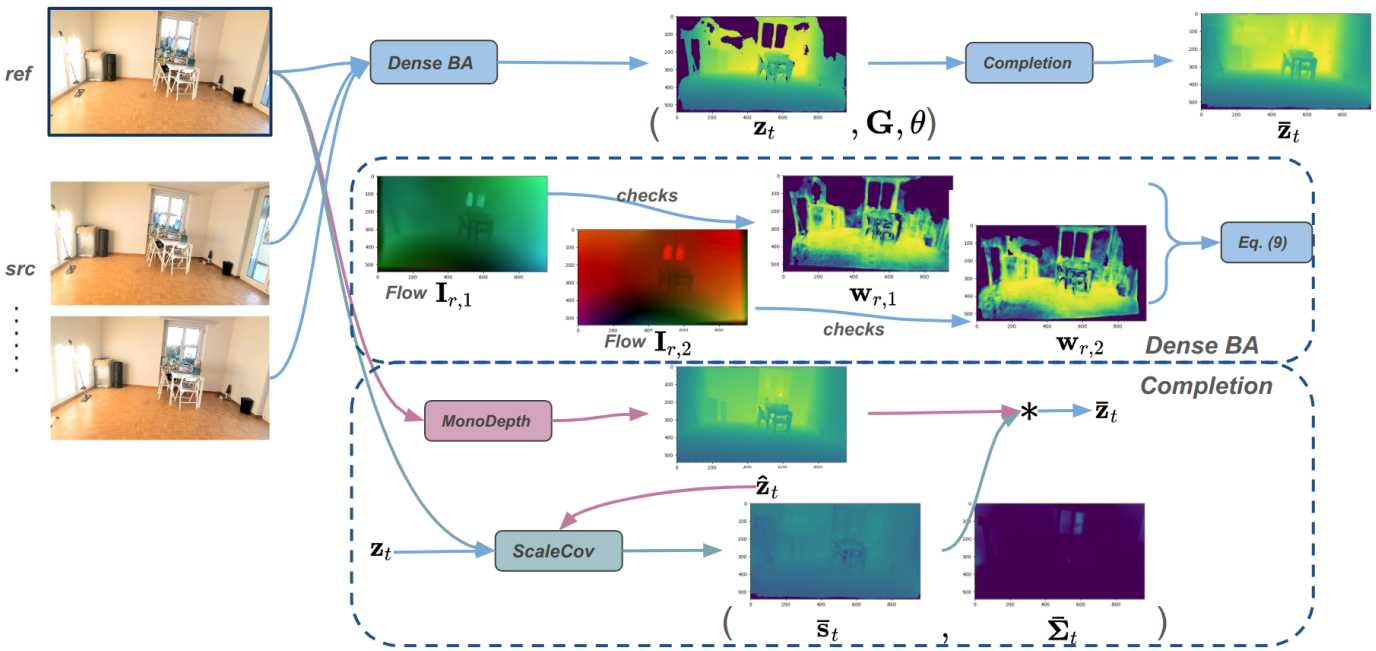


Fig. 7. Depth estimation via unposed & uncalibrated multi-view depth estimation model (U^2-MVD). The overall pipeline is shown.

This thread receives a signal from the user to rotate and move the view camera. The view synthesis is rendered in real-time.

V. Depth Estimation

In this section, we introduce our unposed & uncalibrated multi-view depth estimation model (U^2-MVD) that is depicted as in Fig. 7. As an adjunct to SLAM, we additionally add good neighbor frame selection for more suitable frames.

A. Depth Recovery with Dense Correlation

We acquire dense correspondences from SOTA optical flow estimation model DKMv3 [21] for pixel-wise correspondences $\mathbf{l}_{i,j} \in \mathbb{R}^{H\times W\times 2}$ between the frames i and j. Then the expected correspondence pixel would be $\mathbf{x}_j^* = \mathbf{l}_{i,j} + \mathbf{x}_i$, where $\mathbf{x}_i$ is the pixel coordinate in frame i. However, if false correspondences occur, it will strongly affect the resulting pose and depth. That is, the optical flow cannot be fully trusted.

1) Cross Check: Because single-source matching is risky, we apply cross-checking of correspondences to ensure a high-quality match:

$$\mathbf{w}_{i,j,(u,v)} = \begin{cases} 1 & \text{if } ||\mathbf{x}_{i,(u,v)} - (\mathbf{x}^*_{j,(u,v)} + \mathbf{I}_{j,i,(u,v)})||_2 < 0.5, \\ 0, & \text{otherwise,} \end{cases} \tag{8}$$

where (u, v) is the pixel coordinate in the image.

2) Static Check: We have assumed that our scene is static. However, optical flow is not designed for static scenes. That is, the flow is not constrained by the rigid body. Therefore, unlike previous works [36] that trust the flow, we apply epipolar constraint to filter out the non-rigid flow:

$$\mathbf{w}_{i,j,(u,v)} = \begin{cases} 1 & \text{if } dist_{line}(\mathbf{L}_{j,(u,v)}, \mathbf{x}^*_{j,(u,v)})^2 < 3.84, \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

where $\mathbf{L}_{j,(u,v)}$ is the epipolar line for $\mathbf{x}^*_{j,(u,v)}$, $dist_{line}$ is the distance from point to line. The epipolar line $\mathbf{L}_j$ is obtained by solving the essential matrix with 1000 randomly chosen correspondences.

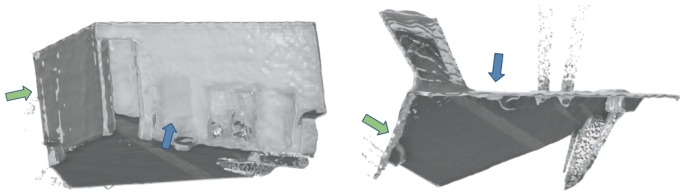
3) In-image Epipole Check: However, since the above static check relies on the epipolar constraint, this check does not work near the epipole, i.e., if the epipole is in the image, the near-pixels will all pass the static check. Therefore, we compute the position of the epipole in the image and filter out the near-pixel correspondences.

4) Dense Bundle Adjustment (DBA): Although the SOTA optical flow is used, from experiments we learnt, the single flow is still too fragile for dense reconstruction use. Therefore, more source frames are usually involved in MVD tasks. In our formulation, we project the depth of the reference frame to all source frames and monitor over dense correspondences between each reference-source pair. In terms of SLAM, we also collect dense matching from the reference (frame t) to neighbor frames ($\mathcal{B}_t$) in a window and compute dense BA to stabilize the inverse depth of reference frame $\mathbf{d}_t$.

Then we follow DROID [36], [49] to denote the poses, the intrinsics and the projection function as $\mathbf{G}$, θ and Π. The cost function of the Dense Bundle Adjustment (DBA) is the sum of the projection errors over all neighbouring frames:

$$\mathcal{L}(\mathbf{G}, \mathbf{d}_t, \theta) = \sum_{j\in\mathcal{B}_t} \| \mathbf{x}^*_j - \Pi(\mathbf{G}_{tj} \circ \Pi^{-1}(\mathbf{x}_t, \mathbf{d}_t, \theta), \theta) \|^2 . \tag{10}$$

This cost function is solved efficiently with Gaussian-Newton. For the detailed formulation, please refer to DROID-SLAM [36], [49]. Eq. (10) directly solves poses, intrinsics and inverse depth at the same time *with only optical flow (dense correspondences) as true value*.



(a) Frontview (b) Topview

Fig. 8. Example for the monocular depth estimation. Green arrow and blue arrow point to two walls.

B. Monocular Depth and Depth Completion

In addition to the DBA, we utilize Metric3Dv2 [50] for monocular depth $\hat{\mathbf{z}}_\mathbf{t}$ on the reference frame t. $\hat{\mathbf{z}}_\mathbf{t}$ guides the DBA from a good start point. While more importantly, we use $\hat{\mathbf{z}}_\mathbf{t}$ to assist in completing the DBA depth with vacancy.

Recent monocular depth methods [10], [50] provide high-quality depth estimation of structures. Please find Fig. 8. In the frontview, we can find a well-predicted shape of the room. However, because the estimation from a single image naturally loses scale and intrinsic information, the relative positions between different structures are distorted. See the top view of Fig. 8, the arrows pointing to the walls are not placed correctly. Besides, monodepth cannot handle vision illusions such as Ames room in Fig. 11, which makes the inter-structure of monodepth theoretically not reliable.

Nevertheless, we still find useful information: the coplanarity is preserved. Which reminds us to extract useful information from **intra-structure**.

Therefore, we intend to utilize a learned covariance function that semantically identifies intra- and extra-structures, e.g., DepthCov [51] that learns deep prior. However, a problem with DepthCov is that it directly regresses depth image with true sparse depth observation. This means that if there is no true sample on certain structures, the depth of such structures will be arbitrarily wrong. An extreme example is with only 1 sample, then the whole depth image will be the same value.

Conversely, we introduce ScaleCov, which regresses scale image for monodepth $\hat{\mathbf{z}}_\mathbf{t}$ given DBA depth $\mathbf{z}_\mathbf{t}$ as observation. ScaleCov does not face this problem, because in the worst case, identical scale image does not ruin the depth structures.

Following DepthCov, we utilize its deep-learned covariance function (kernel function) k to formulate ScaleCov as a Gaussian Process Regression:

$$\mathbf{s}_* = \mathbf{K}_{fn}(\mathbf{K}_{nn} + \sigma_n^2\mathbf{I})^{-1}\mathbf{s}_n, \tag{11}$$

$$\mathbf{\Sigma}_* = \mathbf{K}_{ff} - \mathbf{K}_{fn}(\mathbf{K}_{nn} + \sigma_n^2\mathbf{I})^{-1}\mathbf{K}_{nf}. \tag{12}$$

Where, we sample non-vacancy depth from DBA depth and compute scales $\mathbf{s}_n = \mathbf{z}_{t,n}/\hat{\mathbf{z}}_{t,n}$, and use $\mathbf{K}_{ff}$, $\mathbf{K}_{fn}$ and $\mathbf{K}_{nn}$ to denotes the correlation matrix between query points, query and observations, and observations. $\mathbf{s}_*$ and $\mathbf{\Sigma}_*$ are the queried scale and variance that would be associated with the depth image.

For the finished image we consider the depth to be complete is more trustworthy and let

$$\bar{\mathbf{s}}_t(\mathbf{x}) = \begin{cases} \mathbf{z}_t(\mathbf{x})/\hat{\mathbf{z}}_t(\mathbf{x}), & \text{if } \mathbf{z}_t(\mathbf{x}) > 0 \\ \mathbf{s}_*(\mathbf{x}), & \text{otherwise} \end{cases} \tag{13}$$

$$\bar{\mathbf{\Sigma}}_t(\mathbf{x}) = \begin{cases} \mathbf{0}, & \text{if } \mathbf{z}_t(\mathbf{x}) > 0 \\ \mathbf{\Sigma}_*(\mathbf{x}), & \text{otherwise} \end{cases} \tag{14}$$

where $\mathbf{x}$ is the pixel coordinate.

The final completed depth is

$$\bar{\mathbf{z}}_t = \bar{\mathbf{s}}_t * \hat{\mathbf{z}}_t, \tag{15}$$

where the corresponding pixel-wise variance is $\bar{\mathbf{\Sigma}}_t$.

In addition to the MVD completion shown in Fig. 7, ScaleCov also supports sparse LiDAR depth completion as shown in Fig. 19.

C. Depth Frame Selection

We observe that all previous dense mono SLAMs estimate global geometry using keyframes. It makes sense to estimate depth for keyframes because during the tracking process, the main burden of keyframes is to gain information. The goal is to keep the landmarks in view.

However, we find it problematic to use neighbouring keyframes to support depth recovery. This does not guarantee a good basis for triangulation, especially in dense cases.

Therefore, the good neighbor frames (NF) may not be keyframes (KF). In addition, the tracking model stores non-KF poses as a relative pose to the previous KF. This may not be accurate. When using our depth model, we only input intrinsics and images to estimate the depth without using the poses of the NFs.

Next, we ask the question: "*How to choose a frame that is good for depth estimation?*" Reliable depth estimation requires a reference frame for triangulation. Therefore, our answer is: "*A frame has a good neighborhood for triangulation*".

Since the tracking model provides poses for all frames, we use the relative poses to the reference frame to help verifing. We select neighboring frames according to Algorithm 2, where the relative facing angle and baseline are used.

D. Recover Scale to Global

Since Section V-A is applied without poses, a scale recovery operation is required to adjust to the tracking scale.

While the global scale is hidden in the tracked landmarks or provided poses.

When landmarks are given, the relative scale from local to global is recovered by using RANSAC regression[2] $s_{t,g} = RANSACRegressor(\{\mathbf{z}_t^l(\mathbf{x})/\bar{\mathbf{z}}_t(\mathbf{x}) * s = 1\}_{\mathbf{x}\in\mathcal{P}_t})$, where $\mathbf{z}_t^l$ and $\bar{\mathbf{z}}_t$ are the landmarks from the tracker and the estimated depth from Section V-A, $\mathcal{P}_t$ and $\mathbf{x}$ are the set of landmarks in frame t and pixel coordinate. When landmarks are missing, we adjust depth with the scale difference between solved and provided poses.

[2]https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RANSACRegressor.html

VI. Experiments

The main experiments are conducted on multi-view depth estimation, surface light field and dense SLAM.

A. Settings

1) Implementation Details: We use $l = 3$ levels of neural points with resolution starting at $r_0 = 0.005m$ with a multiplier $m_{reso} = 4$. Our reconstruction model uses the Adam optimizer with $lr = 1e-3$ for both NPs and MLP. The k-nearest neighbor search between the query and NPs is done with a K-d tree, while the rasterization is done with our CUDA-implemented IPR. For the depth estimation model, we rely on DKMv3 [21] for dense correspondences (optical flow), and Metric3Dv2 [50] for monocular depth. We follow DepthCov [51] to build our ScaleCov for depth completion. We extend the monocular tracking model DPVO [48] to RGB-D and RGB-L. Due to the optimization of local windowed frames, we only allow out-of-window (fixed) frames for depth and reconstruction. All experiments are conducted on a PC with an Intel i9-13900KSi9-13900KS CPU and an NVIDIA GeForce 4090 GPU.

2) Datasets: Our experiments mainly rely on datasets for scenes:

a) Replica [41]: Replica has recently become the most widely used synthetic dataset for reconstruction and synthesis of views. It consists of 8 RGB-D sequences for rooms and offices.

b) ScanNet [52]: ScanNet is also widely used for RGB-D reconstruction. However, ScanNet is captured with old camera sensors with high motion blur. Which is harmful for dense matching.

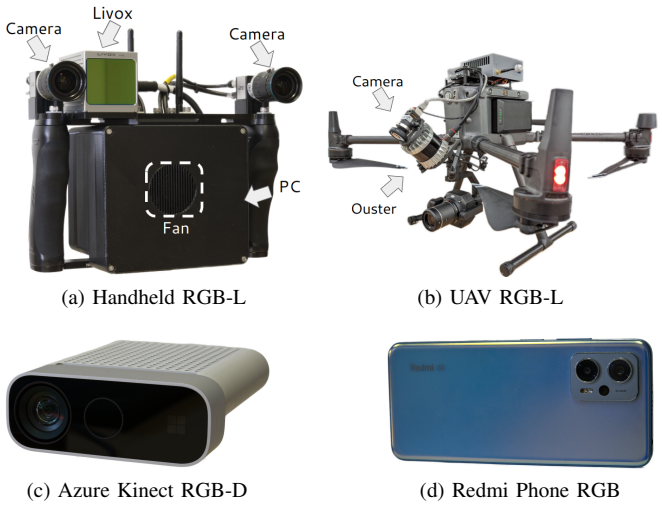
c) KITTI [53]: The KITTI dataset is one of the most famous outdoor datasets with stereo camera, Velodyne laser-scanner and GPS. We use its depth completion data branch for RobustMVD benchmarking.

d) ETH3D [54]: ETH3D is a widely used benchmark in the field of 3D reconstruction. This dataset provides multi-view images with ground truth poses and mesh reconstruction. We use its multi-view benchmark data branch for RobustMVD.

e) DTU [55]: DTU is the most widely used object-level dataset in deep learning-based multi-view stereo work. DTU provides high-quality images with dense point cloud for each object. Although the tabletop object is not the research focus of this paper, we use DTU for RobustMVD.

f) Tanks&Temples [56]: Tanks&Temples captures real indoor and outdoor scenes with high-quality videos. It is mostly used alongside the DTU dataset for generalization testing. We also use it for RobustMVD.

g) Our Datasets: We acquire the first dense mono SLAM purpose RGB-X dataset. Previous RGB-D reconstruction purpose datasets contain large rotations, which is acceptable for RGB-D SLAM even with blur. However, for dense monocular purposes, sufficient parallax between consecutive frames is required. Large rotations and blurry images are detrimental to both tracking and depth estimation. Moreover, recent SOTA dense mono SLAMs only focus on small room- or object-scale reconstruction. With our own datasets we also want to show that the proposed method scales to larger scenes. Therefore, we acquire our own dataset using a motion pattern with good parallax and four different sensor systems, which are shown in Fig. 9 (For more details see Suppl.-C). The datasets ordered from small to large scale are:



(a) Handheld RGB-L (b) UAV RGB-L

(c) Azure Kinect RGB-D (d) Redmi Phone RGB

Fig. 9. Sensors employed for capturing the datasets. See description in Suppl.-C

- Apartment living room using Xiaomi phone (RGB)
- University of Würzburg Robotics hall using a Kinect Azure (RGB-D)
- Veitshöchheim Palace captured from a handheld mapping system (RGB-L)
- University of Wüzburg building complex captured from a UAV of the Center for Telematics (RGB-L)

3) Baselines: We compare our framework to both multi-view depth estimation (MVD) and incremental scene reconstruction (ISR) models.

For MVD, we compare with famous classic COLMAP [57], [58] and Deep learning based methods, such as MVS-Net [15], Vis-MVSSNet [19], MVS2D [59], DeMon [11], DeepV2D [13], Robust MVD baseline [43], DUSt3R [20] and more.

For ISR, we include Dense RGB-D-SLAM SOTAs (NICE-SLAM [32], Vox-Fusion), Dense Mono SLAM SOTAs (NeRF-SLAM, DROID-SLAM, NICER-SLAM), Surface Light Field model (NSLF-OL [25]).

4) Evaluation Metrics: In the MVD test, we follow RobustMVD [43] to use Absolute Relative Error (absrel) and the Inlier Ratio to measure the estimation quality. When ground truth (GT) scale (GT pose) was not given, we follow DUSt3R [20] to align the depth to GT scale with the median.

In the ISR test, our evaluations are mainly for color, geometry, and pose. For color, we use PSNR, SSIM, and LPIPS. For geometry, we use accuracy, completion, and completion ratio. For trajectory, we use ATE RMSE. Before evaluating the geometry and trajectory, we follow NICER-SLAM to align the reconstruction and trajectories with the ICP tool of CloudCompare [60]. For the monocular scenario, scale is also corrected.

TABLE I

MULTI-VIEW DEPTH EVALUATION WITH DIFFERENT SETTINGS: A) CLASSICAL APPROACHES; B) WITH POSES AND DEPTH RANGE, WITHOUT ALIGNMENT; C) ABSOLUTE SCALE EVALUATION WITH POSES, WITHOUT DEPTH RANGE AND ALIGNMENT; D) WITHOUT POSES AND DEPTH RANGE, BUT WITH ALIGNMENT; E) WITHOUT POSES, DEPTH RANGE AND INTRINSICS, BUT WITH ALIGNMENT. (PARENTHESES) DENOTE TRAINING ON DATA FROM THE SAME DOMAIN. THE BEST RESULTS FOR EACH SETTING ARE IN **BOLD**. THE OVERALL BEST ARE UNDERLINED AND BOLDED.

	Methods	GT Pose	GT Range	GT Intrinsics	Align	KITTI rel ↓	KITTI τ ↑	ScanNet rel ↓	ScanNet τ ↑	ETH3D rel ↓	ETH3D τ ↑	DTU rel ↓	DTU τ ↑	T&T rel↓	T&T τ ↑
(a)	COLMAP [57], [58]	✓	×	✓	×	**12.0**	**58.2**	**14.6**	**34.2**	**16.4**	**55.1**	**0.7**	**96.5**	**2.7**	95.0
	COLMAP Dense [57], [58]	✓	×	✓	×	26.9	52.7	38.0	22.5	89.8	23.2	20.8	69.3	25.7	76.4
(b)	MVSNet [15]	✓	✓	✓	×	22.7	36.1	24.6	20.4	35.4	31.4	(1.8)	(86.0)	8.3	73.0
	MVSNet Inv. Depth [15]	✓	✓	✓	×	18.6	30.7	22.7	20.9	21.6	35.6	(1.8)	(86.7)	6.5	74.6
	Vis-MVSSNet [19]	✓	✓	✓	×	**9.5**	**55.4**	**8.9**	**33.5**	**10.8**	**43.3**	**(1.8)**	**(87.4)**	**4.1**	**87.2**
	MVS2D ScanNet [59]	✓	✓	✓	×	21.2	8.7	(27.2)	(5.3)	27.4	4.8	17.2	9.8	29.2	4.4
	MVS2D DTU [59]	✓	✓	✓	×	226.6	0.7	32.3	11.1	99.0	11.6	(3.6)	(64.2)	25.8	28.0
(c)	DeMon [11]	✓	×	✓	×	16.7	13.4	75.0	0.0	19.0	16.2	23.7	11.5	17.6	18.3
	DeepV2D KITTI [13]	✓	×	✓	×	(20.4)	(16.3)	25.8	8.1	30.1	9.4	24.6	8.2	38.5	9.6
	DeepV2D ScanNet [13]	✓	×	✓	×	61.9	5.2	(3.8)	(60.2)	18.7	28.7	9.2	27.4	33.5	38.0
	MVSNet [15]	✓	×	✓	×	14.0	35.8	1568.0	5.7	507.7	8.3	(4429.1)	(0.1)	118.2	50.7
	MVSNet Inv. Depth [15]	✓	×	✓	×	29.6	8.1	65.2	28.5	60.3	5.8	(28.7)	(48.9)	51.4	14.6
	Vis-MVSNet [19]	✓	×	✓	×	10.3	**54.4**	84.9	15.6	51.5	17.4	(374.2)	(1.7)	21.1	65.6
	MVS2D ScanNet [59]	✓	×	✓	×	73.4	0.0	(4.5)	(54.1)	30.7	14.4	5.0	57.9	56.4	11.1
	MVS2D DTU [59]	✓	×	✓	×	93.3	0.0	51.5	1.6	78.0	0.0	(1.6)	(92.3)	87.5	0.0
	Robust MVD Baseline [43]	✓	×	✓	×	**7.1**	41.9	**7.4**	**38.4**	9.0	42.6	**2.7**	82.0	5.0	75.1
	U^2-MVD (Ours)	✓	×	✓	×	23.6	31.8	27.6	21.8	**2.6**	**43.7**	2.82	**81.1**	**2.26**	**89.2**
(d)	DeMoN [11]	×	×	✓	$\|\mathbf{t}\|$	15.5	15.2	12.0	21.0	17.4	15.4	21.8	16.6	13.0	23.2
	DeepV2D KITTI [13]	×	×	✓	med	(3.1)	(74.9)	23.7	11.1	27.1	10.1	24.8	8.1	34.1	9.1
	DeepV2D ScanNet [13]	×	×	✓	med	10.0	36.2	(4.4)	(**54.8**)	11.8	29.3	7.7	33.0	8.9	46.4
	U^2-MVD (Ours)	×	×	✓	med	**2.07**	**81.9**	**3.40**	41.4	**1.89**	**44.8**	**2.77**	**72.0**	**2.30**	**81.4**
(e)	DUSt3R 224-NoCroCo [20]	×	×	×	med	15.14	21.16	7.54	40.00	9.51	40.07	3.56	62.83	11.12	37.90
	DUSt3R 224 [20]	×	×	×	med	15.39	26.69	(5.86)	(50.84)	4.71	61.74	**2.76**	**77.32**	5.54	56.38
	DUSt3R 512 [20]	×	×	×	med	9.11	39.49	(**4.93**)	(**60.20**)	2.91	**76.91**	3.52	69.33	3.17	76.68
	U^2-MVD (Ours)	×	×	×	med	**2.08**	**82.01**	7.19	22.17	**1.87**	50.14	15.61	33.74	**2.46**	**81.00**

B. Effect of Multiview Detph Estimation

One of the most important components of SceneFactory is the depth estimator. We follow the Multiview Depth Estimation (MVD) benchmark, RobustMVD [43], to evaluate the depth result on 5 widely used datasets (KITTI, ScanNet, ETH3D, DTU, and Tanks&Temple).

1) Quantitative Evaluation: Please find Table I the benchmarking result. The metrics are the Absolute Relative Error (rel or absrel) and the Inlier Ratio (τ) with a threshold of 1.03 [43]. Since our model also supports intrinsic-free, which is not listed in [43], we follow DUSt3R [20] to add an additional column (GT Intrinsics) to indicate the requirement of intrinsics besides the whole benchmark. Both of ours and DUSt3R do not require GT pose, depth range, and intrinsics (in setting (e)). To have a better evaluation of our model, we add intrinsics and also test our model in both (c) with GT pose and (d) without GT pose setting.

First of all, in (e) setting (without the input of GT pose, depth range, or intrinsics), on the indoor and outdoor datasets ETH3D, Tanks&Temple and KITTI, our model can already outperform other approaches over all (a-e) folds. On the indoor dataset ScanNet, our scores do not exceed DUSt3R. What's more, our model performs particularly poorly on the object level (DTU dataset). We believe that this is due to the fact that the estimation of intrinsic properties adds additional challenges to the MVD tasks. On ScanNet, the images originally contained strong motion blur, which is also detrimental to the estimation of intrinsics. While on the object level dataset, for example DTU, small intrinsic errors will cause more noticeable estimation errors, sicne the camera is close to the object. Please note when we also input intrinsics (in (d) setting), our model easily achieves overall best result on all datasets over all (a-e) folds. But, when we further input GT pose (in (c) setting), our performance is dropped on KITTI and ScanNet datasets. While on the rest, the performances are still maintained. This is because ETH3D, DTU and T&T's GT poses are captured from the scanning system or COLMAP, which are good enough for multiview depth estimation purpose. However, KITTI's GT pose is from GPS, ScanNet's GT pose is from RGB-D SLAM, BundleFusion [61]. Which is more for tracking and mapping purposes. Therefore, if the GT pose is not available, optimizing both (pose and depth) will certainly exceed optimizing with a non-GT GT pose input.

From the condition that overall BEST in Table I are mainly

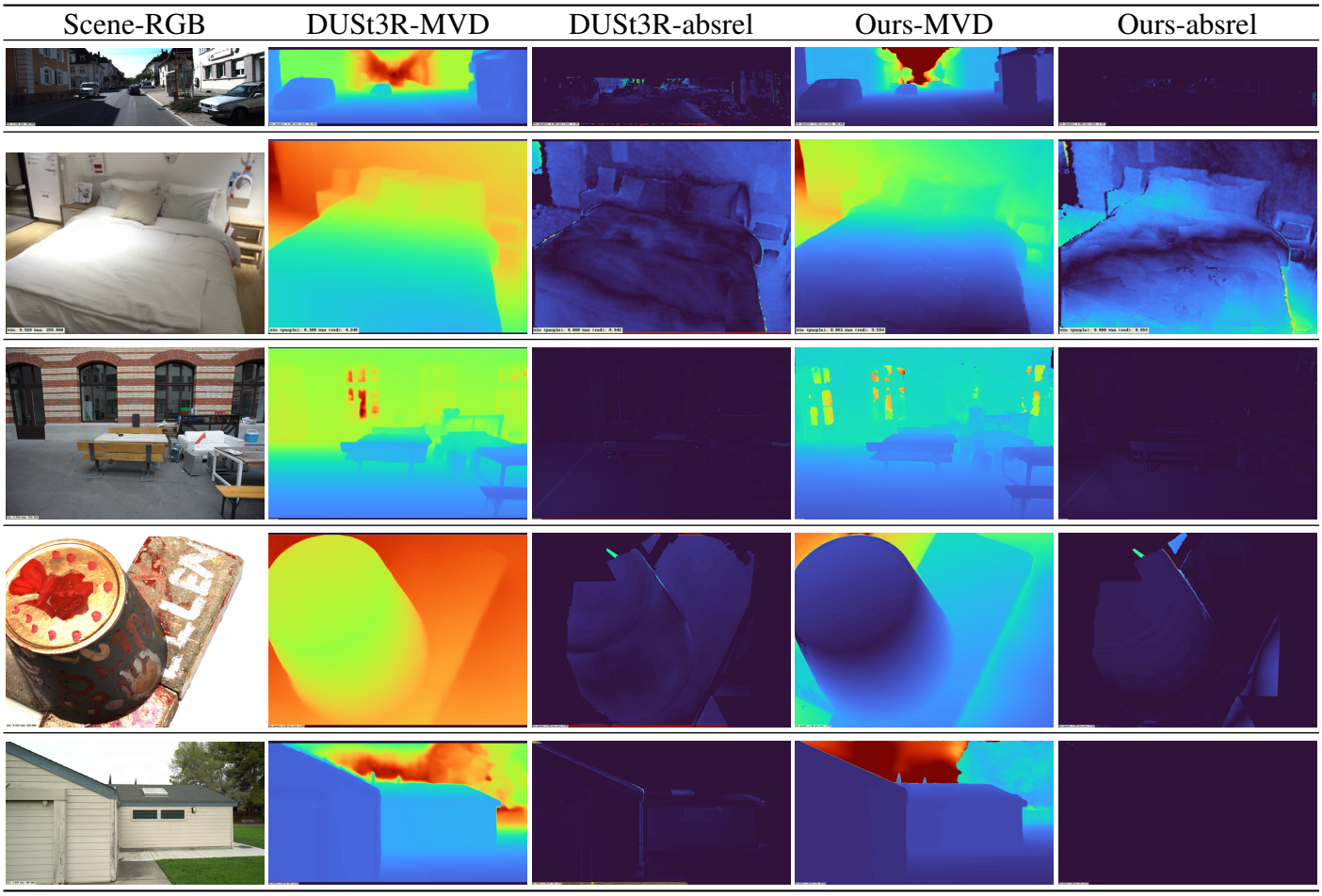


Fig. 10. Result of MVD. Examples from top row to bottom are from the KITTI, ScanNet, ETH3D, DTU and T&T datasets. From left to right are scene image, depth prediction and absolute relative error (absrel) error of DUSt3R, and depth prediction and absrel error of our model. For depth and absrel images, higher values are warmer, lower values are colder colors.

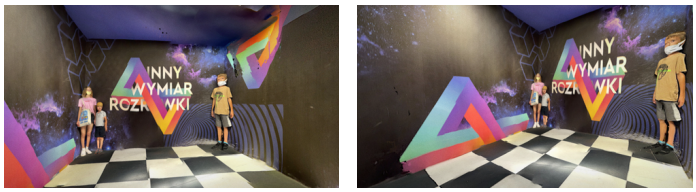
ours, we find our model can flexibly support all input settings, while in a reliable depth estimation performance. This provides us a good basis for our entire SceneFactory pipeline.

2) Qualitative Evaluation: We are particularly interested in the unconstraint setting (e) and show the estimated depth and error in Fig. 10. Where we have attached the qualitative result of the robustMVD of the 5 datasets in Table I sequentially.

Please find in figure that our result has better performance on outdoor and indoor dataset, except for ScanNet. It is consistent with the result in Table I (e) where the intrinsic solving via dense matching of our method is fragile for ScanNet that contains strong motion blur caused by a rather old RGB camera. However, DUSt3R is more optimized for monodepth (because during the benchmarking, DUSt3R does not benefit from more source views). So it is not affected by this issue.

Although our average score on object dataset DTU is lower, it is only because our method fails in some cases. While the robustMVD selected qualitative figure is still better.

3) Demonstration on Custom Data: As we claimed earlier, DUSt3R is not guaranteed to work due to the limitation of the training dataset. In addition, due to the fact that 1) DUSt3R does not improve with increasing number of source images in the RobustMVD benchmark and 2) performs similarly with and without source images. We suggest that the SOTA DUSt3R is mainly based on its strength in monocular depth.



(a) Ames Room (ref frame) (b) Ames Room (src frame)

Fig. 11. MVS input of Ames room. The true shape please find webpage [4].

To verify this, we test the vision illusion that also happens to a human eye vision, the Ames Room[3]. Here we show the data of the Ames Room in Fig. 11. Where we capture the image from Sketchfab resource[4] Ames Room to create the illusion of a distorted room. Where typically shows incorrect relative scales on the left and right corners. DUSt3R, as we suspected, shows the Ames room as a normal room, as in Fig. 12. Our model, on the other hand, truly reproduces the distorted space.

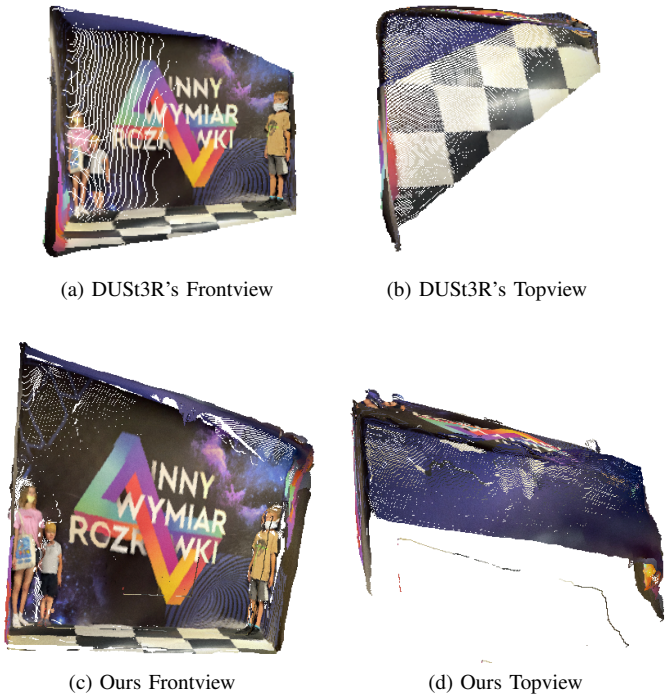
The above tests show that our model not only scores well, but is also more reliable.

[3]https://en.wikipedia.org/wiki/Ames_room

[4]https://sketchfab.com/3d-models/ames-room-iphone-3d-scan-b59a0dcf49de4df2a50104abf3eab7e4

TABLE II
FULL MODEL COMPARISON ON REPLICA SEQUENCES. HEADER INDICATES SCENE NAMES.

		Office0	Office1	Office2	Office3	Office4	Room0	Room1	Room2
NICE-SLAM [32]	PSNR ↑	28.38	30.68	23.90	24.88	25.18	23.46	23.97	25.94
	SSIM ↑	0.908	**0.935**	**0.893**	**0.888**	**0.902**	0.798	0.838	0.882
	LPIPS ↓	0.386	0.278	0.330	0.287	0.326	0.443	0.401	0.315
Surface Light Field methods									
NSLF-OL [25]+DI-Fusion [26]	PSNR ↑	28.59	26.70	21.10	21.89	25.74	23.24	25.68	24.88
	SSIM ↑	0.913	0.879	0.863	0.847	0.893	0.816	0.883	0.888
	LPIPS ↓	0.371	0.497	0.362	0.368	0.401	0.371	0.308	0.330
SceneFactory (Ours)	PSNR ↑	**33.38**	**31.89**	**24.84**	**25.39**	**31.14**	**27.98**	**29.51**	**30.64**
	SSIM ↑	**0.921**	0.893	0.837	0.873	0.893	**0.858**	**0.883**	**0.898**
	LPIPS ↓	**0.167**	**0.267**	**0.208**	**0.129**	**0.178**	**0.111**	**0.149**	**0.145**



(a) DUSt3R's Frontview (b) DUSt3R's Topview

(c) Ours Frontview (d) Ours Topview

Fig. 12. MVD on Ames room. (a) and (b) shows the frontview and the topview of DUSt3R's prediction. Which falsely predicts the distorted space as a rectangular room. (c) and (d) shows the frontview and the topview of our prediction. Which well predicts the real shape.

C. Evaluation on Surface Light Fields (SLF)

In the test above, we generate depth from images. In this subsection, we generate a color image and explore the importance of our surface light field model.

1) Replica Test: To mitigate the effects of other SceneFactory blocks, image, depth, pose and intrinsics $(\mathbf{I}_{rgb}, \mathbf{I}_d, \mathbf{G}, \theta)$ are all available in the dependency graph (Fig. 2).

First, we follow the scene-level SLF model NSLF-OL [25] to test on the Replica dataset. From Table II, our model strongly outperforms the NICE-SLAM and SOTA SLF model, NSLF-OL on all PSNR, SSIM, LPIPS metrics. To make the comparison more obvious, we show images in the Fig. 13, the first two rows are from the Replica dataset. From this, we see that the main advances of our model are that on the one hand, unlike NSLF-OL, our DM-NPs itself can provide a cleaner surface, and on the other hand, our SLF recovers the color better. Also, NSLF-OL only works alongside the surface model, while our SLF model can support surface generation by itself.

TABLE III
SURFACE LIGHT FIELD COMPARISON ON SCANNET SEQUENCES.

		0568	0164
NSLF-OL +Di-Fusion	PSNR ↑	19.09	18.78
	SSIM ↑	0.500	0.595
	LPIPS ↓	0.575	0.551
SceneFactory	PSNR ↑	**19.90**	**21.85**
	SSIM ↑	**0.573**	**0.688**
	LPIPS ↓	**0.536**	**0.444**

2) ScanNet Test: Following the same setup, we continue testing on ScanNet. Please find Fig. 13. The ScanNet image has a lot of blur on the image and more noise in the depth (due to the low-quality capture).

However, our result is still better than NSLF-OL with better quantitative evaluation (Table III). From the image shown in Fig. 13 the row $3, 4$, we see that the structure of NSLF-OL is not well reconstructed. Our result is much clearer.

D. Evaluation of Dense SLAM

In this section, we evaluate SceneFactory's main application, Dense SLAM, in both RGB-D and monocular settings.

TABLE IV
CAMERA TRACKING RESULTS ON THE REPLICA DATASET. ATE RMSE [CM] (↓) IS USED AS THE EVALUATION METRIC.

	rm-0	rm-1	rm-2	off-0	off-1	off-2	off-3	off-4	Avg.
RGB-D input									
NICE-SLAM	1.69	2.04	1.55	0.99	0.90	1.39	3.97	3.08	1.95
Vox-Fusion	0.27	1.33	0.47	0.70	1.11	0.46	**0.26**	0.58	0.65
SceneFactory	**0.20**	**0.12**	**0.14**	**0.17**	**0.07**	**0.13**	0.29	**0.22**	**0.17**
RGB input									
COLMAP	0.62	23.7	0.39	0.33	0.24	0.79	0.14	1.73	3.49
NeRF-SLAM	17.26	11.94	15.76	12.75	10.34	14.52	20.32	14.96	14.73
DIM-SLAM	0.48	0.78	0.35	0.67	0.37	0.36	0.33	0.36	0.46
DROID-SLAM	0.34	**0.13**	0.27	0.25	0.42	0.32	0.52	0.40	0.33
NICER-SLAM	1.36	1.60	1.14	2.12	3.23	2.12	1.42	2.01	1.88
SceneFactory	**0.20**	0.20	**0.15**	**0.20**	**0.12**	**0.25**	**0.29**	**0.22**	**0.20**

1) Replica Test: First, we follow Dense Mono SLAM SOTA, NICER-SLAM, to have a complete test on both color and geometric on Replica dataset.

Fig. 13. Result of Surface Light Field test on Replica and ScanNet dataset.

We show the tracking performance, from Table IV, SceneFactory achieves the best tracking performance overall. While with RGB input, SceneFactory uses Momo-SLAM DPVO, but with RGB-D input, SceneFactory uses our generalized DPVO with RGB-D, which achieves the best performance.

Notably, with RGB and depth as input, NSLF-OL requires an external reconstruction model to provide the surface prediction. While in contrast, **SceneFactory** itself also supports surface. We render input frames for depth and accumulate a point cloud. To fit in the evaluation script of NICER-SLAM, we extract mesh via Screened Poisson Surface Reconstruction for the evaluation. Table V shows the performance of the reconstruction. In addition, to demonstrate the ability to reconstruct surface color, we show the color result in Table V in the view synthesis comparison. The upper rows are with RGB-D input, while the lower rows are with RGB input.

From the tables, we learn that with RGB-D input, **SceneFactory** outperforms all SOTAs by far in both reconstruction and view generation over all scenes by a wide margin. To better demonstrate the quality, we show a qualitative evaluation in Fig. 14. Here our method works best. Please note the highly detailed texture on the quilt.

In the monocular setting with RGB input only, our model cannot outperform the NeRF-like SOTA NICER-SLAM. Please find Fig. 14, our model 1) does not support completion, 2) directly fuses depth images without optimizing the geometry that causes false surface rendering. Which we consider the reason that cannot surpass the NICER-SLAM. However, NICER-SLAM takes ~ 10 **hours train per scene** excluding the preprocessing time that costs more, while our model takes only **minutes**. But as a DBA-based method, compared to our closest model DROID-SLAM, our model still achieves much better scores to a level as useful as the NeRF-like method. We believe this is due to the post-optimization nature of the global geometry for NeRF-like models. While DBA-based methods rely more on a separate solution of the local inverse depth.

TABLE V
RECONSTRUCTION RESULTS ON THE REPLICA DATASET. BEST RESULTS ARE HIGHLIGHTED AS **first**, SECOND, AND THIRD.

		rm-0	rm-1	rm-2	off-0	off-1	off-2	off-3	off-4	Avg.
RGB-D input										
NICE	Acc.[cm]↓	3.53	3.60	3.03	5.56	3.35	4.71	3.84	3.35	3.87
	Comp.[cm]↓	3.40	3.62	**3.27**	4.55	4.03	3.94	3.99	4.15	3.87
	Comp.Ratio[<5cm %]↑	86.05	80.75	**87.23**	79.34	82.13	80.35	80.55	82.88	82.41
Vox-Fusion	Acc.[cm]↓	2.53	1.69	3.33	2.20	2.21	2.72	4.16	2.48	2.67
	Comp.[cm]↓	**2.81**	**2.51**	4.03	8.75	7.36	4.19	**3.26**	**3.49**	4.55
	Comp.Ratio[<5cm %]↑	**91.52**	**91.34**	86.78	**81.99**	82.03	**85.45**	**87.13**	**86.53**	86.59
Ours	Acc.[cm]↓	**1.49**	**1.16**	**1.24**	**1.11**	**0.91**	**1.37**	**1.62**	**1.52**	**1.30**
	Comp.[cm]↓	3.65	2.88	4.31	**1.68**	**2.23**	**3.59**	3.59	4.02	**3.24**
	Comp.Ratio[<5cm %]↑	87.61	90.02	86.83	**93.43**	**90.39**	**86.24**	84.98	85.07	**88.07**
RGB input										
NeRF-S.	Acc. [cm]↓	11.84	10.62	11.86	9.32	14.40	11.54	16.31	11.11	12.13
	Comp. [cm]↓	5.63	5.88	9.22	13.29	10.17	6.95	7.81	5.26	8.03
	Comp. Ratio [%]↑	61.13	68.19	47.85	37.64	56.17	66.20	55.67	61.86	56.84
DROID-S.	Acc. [cm]↓	12.18	8.35	**3.26**	3.01	2.39	5.66	4.49	4.65	5.50
	Comp. [cm]↓	8.96	6.07	16.01	16.19	16.20	15.56	9.73	9.63	12.29
	Comp. Ratio [%]↑	60.07	76.20	61.62	64.19	60.63	56.78	61.95	67.51	63.60
NICER-S.	Acc. [cm]↓	**2.53**	**3.93**	3.40	5.49	3.45	**4.02**	**3.34**	**3.03**	**3.65**
	Comp. [cm]↓	**3.04**	**4.10**	**3.42**	**6.09**	**4.42**	**4.29**	**4.03**	**3.87**	**4.16**
	Comp. Ratio [%]↑	**88.75**	**76.61**	**86.1**	65.19	**77.84**	**74.51**	**82.01**	**83.98**	**79.37**
Ours	Acc. [cm]↓	3.61	4.02	5.53	**2.71**	**2.17**	4.09	4.23	3.69	3.76
	Comp. [cm]↓	6.98	6.76	12.24	6.46	5.59	10.31	7.53	10.46	8.29
	Comp. Ratio [%]↑	74.06	72.59	63.85	**77.80**	75.26	65.56	68.89	69.10	70.89

TABLE VI
NOVEL VIEW SYNTHESIS EVALUATION ON REPLICA DATASET. BEST RESULTS ARE HIGHLIGHTED AS **first**, SECOND, AND THIRD.

		rm-0	rm-1	rm-2	off-0	off-1	off-2	off-3	off-4	Avg.
RGB-D input										
NICE-S.	PSNR ↑	22.12	22.47	24.52	29.07	30.34	19.66	22.23	24.94	24.42
	SSIM ↑	0.689	0.757	0.814	0.874	0.886	0.797	0.801	0.856	0.809
	LPIPS ↓	0.330	0.271	0.208	0.229	0.181	0.235	0.209	0.198	0.233
Vox-F.	PSNR ↑	22.39	22.36	23.92	27.79	29.83	20.33	23.47	25.21	24.41
	SSIM ↑	0.683	0.751	0.798	0.857	0.876	0.794	0.803	0.847	0.801
	LPIPS ↓	0.303	0.269	0.234	0.241	0.184	0.243	0.213	0.199	0.236
Ours	PSNR ↑	**27.72**	**28.86**	**30.17**	**32.59**	**31.39**	**24.44**	**25.34**	**30.47**	**28.87**
	SSIM ↑	**0.850**	**0.872**	**0.899**	**0.915**	**0.892**	**0.837**	**0.867**	**0.887**	**0.877**
	LPIPS ↓	**0.124**	**0.166**	**0.143**	**0.165**	**0.262**	**0.208**	**0.132**	**0.184**	**0.173**
RGB input										
NeRF-S.	PSNR ↑	16.45	19.62	21.17	21.44	20.86	15.49	15.11	18.96	18.64
	SSIM ↑	0.576	0.700	0.754	0.773	0.747	0.731	0.688	0.790	0.720
	LPIPS ↓	0.330	**0.177**	**0.170**	0.335	0.229	0.251	0.282	0.241	0.252
DROID-S.	PSNR ↑	21.41	24.04	22.08	23.59	21.29	20.64	20.22	20.22	21.69
	SSIM ↑	0.693	**0.786**	0.826	**0.868**	**0.863**	**0.828**	0.808	0.819	0.812
	LPIPS ↓	0.329	0.270	0.228	0.232	0.207	0.231	0.234	0.237	0.246
NICER-S.	PSNR ↑	**25.33**	23.92	**26.12**	**28.54**	**25.86**	**21.95**	**26.13**	**25.47**	**25.41**
	SSIM ↑	**0.751**	0.771	0.831	0.866	0.852	0.820	**0.856**	**0.865**	**0.827**
	LPIPS ↓	**0.250**	0.215	0.176	**0.172**	**0.178**	**0.195**	**0.162**	**0.177**	**0.191**
Ours	PSNR ↑	23.12	**24.10**	24.11	26.15	24.68	21.55	22.44	24.42	23.82
	SSIM ↑	0.686	0.749	0.765	0.816	0.852	0.731	0.739	0.789	0.766
	LPIPS ↓	0.353	0.307	0.366	0.338	0.291	0.381	0.346	0.375	0.344

There is no post-optimization involved.

In addition, the DBA-based method mainly relies on triangulation. However, the replica data set is originally acquired for the purpose of dense RGB-D reconstruction, but not for dense mono. This means that the trajectory of the replica sequences inherently does not consider the effect of the interrelation of frames for depth estimation.

To better illustrate this issue, we test on our own dataset for Dense Mono SLAM purpose in large scene, with mostly xy-directional motion.

The recent SOTA NeRF-like Dense SLAM, NICER-SLAM, highly focuses on the repeated encircling capturing of an object/scene. This is because NICER-SLAM theoretically expires on large scale scene: 1) NICER-SLAM works in bounded model, the poses are required to bound the scene (from their open release); 2) NICER-SLAM's hashgrid resolution is preset to 2048 with about 24 GB usage of GPU memory. But this is far too low for a large scene, and it is hard to increase the resolution further. (This is also the reason why the unbounded multi-hashgrid method, NSLF-OL [25], uses SLF instead of NeRF). So as the scale increases, NICER-SLAM's tracking and then mapping will be dysfunctional. Even the new branch, Gaussian blending based dense mono SLAM, MonoGS, is rarely tested at large scale. While SLAM in robotics is usually excircle-like capturing the scene.

Therefore, to better explore the performance, our dataset mainly contains middle- and large-range scenes which are rarely involved by them.

E. Dense Mono SLAM Purpose dataset

The above tests give an overview of the performance of the tracker and the reconstructor.

We collect this dataset because we find that the current RGB-D sequences are all intended for RGB-D tracking or dense RGB-D reconstruction in small scenes. Due to the fact that RGB-D sequences contain real 3D metric depth, the sequence can simply move arbitrarily. For example, in ScanNet and Replica there is a lot of almost purely rotational motion.

On the other hand, monocular sequences datasets are only used for tracking and SfM purposes because no depth ground truth is available. For dense mono SLAM, however, the challenge goes beyond tracking and mapping. Unlike mono SLAM that only estimates sparse points with matching pairs, dense mono SLAM requires a much larger region to cover.

Furthermore, in real-world robotics applications, the scene is usually captured on large scenes and non-repeat capturing which is rarely involved in recent dense mono SLAM SOTAs.

Therefore, we have not found any published dataset that is primarily for Dense Mono SLAM purposes in robotics. To fill this gap, we hereby present the first large-scale RGB-D/L dataset for dense mono SLAM purposes:

- Mid-scale ($\sim 20 \times 10m$) scene (RGB-D) as Fig. 15a,
- Large-scale ($\sim 80 \times 50m$) scene (RGB-L) as Fig. 15b,
- Small-scale ($\sim 5 \times 5m$) scene (RGB) as Fig. 15c,
- Very-large-scale ($\sim 300 \times 200m$) scene (RGB-L) as Fig. 15d

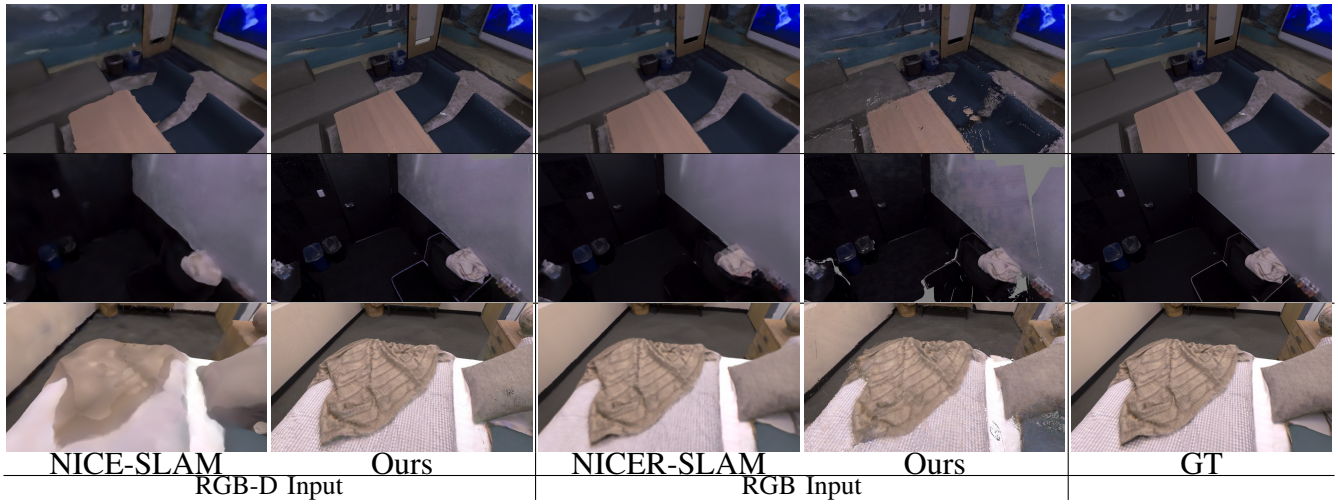
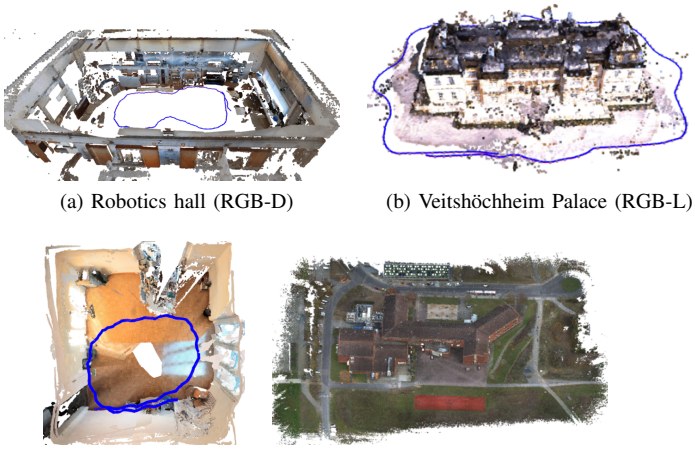


Fig. 14. Qualitative evaluation on the Replica dataset for selected views.



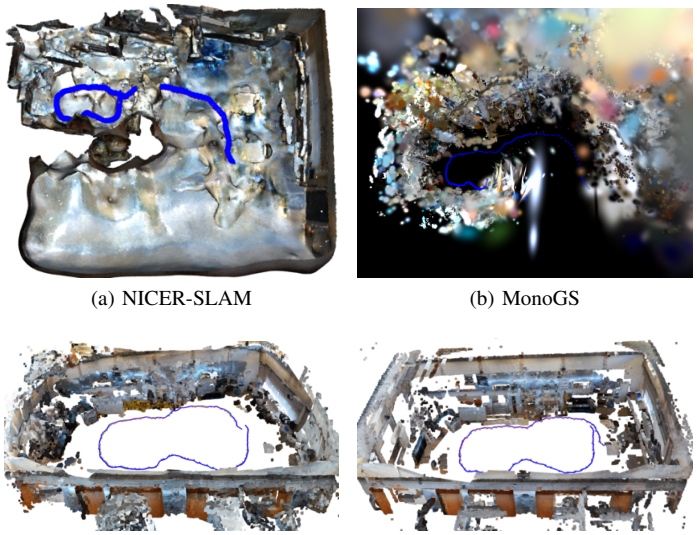
(a) Robotics hall (RGB-D) (b) Veitshöchheim Palace (RGB-L)

(c) Living room (RGB) (d) University campus (RGB-L)

Fig. 15. Our four datasets featuring different ranges.

TABLE VII
TEST ON ROBOTICS HALL OF OUR DATASET.

	NICER.	MonoGS	Ours	Ours (RGB-D)
ATE [m] ↓	2.44	2.17	**0.59**	**0.22**
Acc. [m]↓	1.009	0.724	**0.360**	**0.138**
Comp. [m]↓	1.246	0.837	**0.368**	**0.141**
Comp. Ratio [%]↑	33.11	49.58	**74.84**	**99.13**
PSNR ↑	**19.20**	-	15.13	14.65
SSIM ↑	**0.690**	-	0.442	0.468
LPIPS$_{alex}$ ↓	**0.449**	-	0.628	0.615
Time	8 hours	25 min.	10 min.	5 min.



(a) NICER-SLAM (b) MonoGS

(c) Ours (RGB) (d) Ours (RGB-D)

Fig. 16. Test result (3D maps) on our mid-range scene, i.e., robotics hall. Trajectories are plotted blue.

We capture our mid-range data, Robotics Hall, with an Azure Kinect RGB-D camera. Because of the range limitation of Kinect Depth ($1m - 5m$), we turn to RGB-L for large-range data, Veitshöchheim Palace. For small-range data, living room, we use cell phone for just quick quality demonstration. To explore the extreme, we capture very large-range data, university campus (language center) with an aerial view RGB-L capture.

Figs. 15a to 15c's trajectory is a circle because the target is on the same level. Fig. 15d's drone trajectory is parallel to the ground but the aerial view also well fits the xy-directional translation.

In the previous test, we aligned the surface result with ICP, which becomes challenging in the large scene due to the large partial non-overlap. To standardize the comparison, in our dataset, we use the alignment matrix from trajectory comparison to transform the result reconstruction to ground truth.

1) Mid-scale data: For a handheld camera, since our task is mainly with scene modeling, our facing direction of the camera is inside-out. Therefore, our capture scheme should follow a certain rule, as shown in Fig. 15:

- move in circle in the scene
- viewing direction is perpendicular to the moving direction.

Please find Table VII the result on robotics hall. SceneFactory can easily achieve best tracking and reconstruction than two most new Dense Mono SLAM SOTA, NICER-SLAM and

MonoGS. Moreover, we find NICER-SLAM that works better on image rendering even with much worse tracking and inconsistent scale. To reveal the truth, we visualize the trajectory with reconstruction. Please find Fig. 16, both NICER-SLAM and MonoGS severely suffer severely from scale. This further confirms our speculation: The tightly coupled Dense Mono SLAM SOTAs (NICER-SLAM, MonoGS) models *attend to one thing and lose sight of another*.

Also, the open version of NICER-SLAM requires preprocessing of the data sequence (COLMAP and more), which takes many more hours than the 8h in the table. MonoGS is relatively faster, but the result can hardly be viewed in any non-trained view. Which is, you can only view the mid-range and large-range (next test) in the same trained view. As claimed in NSLF-OL [25], this is not usable in SLAM because SLAM captures contain very sparse view directions.

TABLE VIII
TEST ON SCHLOSS OF OUR DATASET.

	NICER.	MonoGS	Ours	Ours (RGB-L)
ATE [m] ↓	-	12.90	0.31	0.37
Acc. [m]↓	-	7.050	5.074	0.423
Comp. [m]↓	-	19.956	0.215	0.321
Comp. Ratio [%]↑	-	2.6435	95.67	86.54
Time	4 (28%) hours	2 hours	13 min.	9 min.

(a) NICER-SLAM (b) MonoGS

(c) Ours (RGB) (d) Ours (RGB-L)

Fig. 17. Test result (3D map) on our large-range scene, i.e., Veitshöchheim Palace.

2) Large-scale data: For even larger scenes, RGB-D cameras are not applicable, therefore, we turn to Livox LiDAR to provide the metric depth.

We use our handheld RGB-L camera to capture scenes. Since the range is large, our target should be captured outside-in following the same rule as Section VI-E1.

For the larger range scene, from Table VIII, NICER-SLAM fails at 28%, MonoGS shows very large tracking error. This is also revealed by the Fig. 17. MonoGS losses the scale in the middle. In addition the better performance on large-range, the time cost should also be considered. NICER-SLAM takes 4 hours for only 28% without the counting of preprocessing. MonoGS takes 2 hours for the scene. While ours are still in an acceptable range.

3) Aerial-view data test: To better demonstrate the potential, we further expand the test scale to very large. This is too large to capture with simple handheld equipment. So we turn to an airborne device.

The data is collected with a custom-built UAV (DJI Matrice 300 RTK) equipped with a LUCID camera and a co-calibrated OUSTER OS1 laser scanner, flying over a university building that is a former high school.

The Dense SLAM result is demonstrated in Fig. 18. Without using of metric depth. SceneFactory provides a high-quality reconstruction of this very large scene.

F. Exclusive Applications

In addition to the previously demonstrated hot tasks, SceneFactory also supports other usages.

1) LiDAR Completion: Our ScaleCov model supports completion of metric LiDAR depth. Given an RGB and sparse depth image from Livox, ScaleCov regresses the full depth and variance to the client. We provide an example in Fig. 19.

This technique is also utilized in previous test Fig. 17d.

2) Dense Depth-only SLAM: Color sensors tend to be more sensitive to the environment. If a color sensor fails, e.g. due to poor lighting conditions or at night, the depth sensor could play a role instead.

SceneFactory implements depth-only SLAM under the factory lines of depth-flexion and RGB-D SLAM. This first transforms the depth image into a trackable RGB image with depth-flexion. Then the application is used in the same way as our RGB-D SLAM application. Please find Fig. 20 our demo of office2 in Replica dataset, with depth only as input, The reconstruction result is of high quality and smoothness.

VII. CONCLUSION

In this paper, we have introduced a workflow-centric and unified framework for incremental scene modeling, called SceneFactory. Following the structure of a "factory", we have designed "assembly lines" for a wide range of applications, to achieve high flexibility, adaptability and production diversification.

In addition, within the framework, we propose an unposed & uncalibrated multi-view depth estimation model for highly flexible use. We introduce a surface accessible light field design along with an improved point rasterization to enable surface query for the first time.

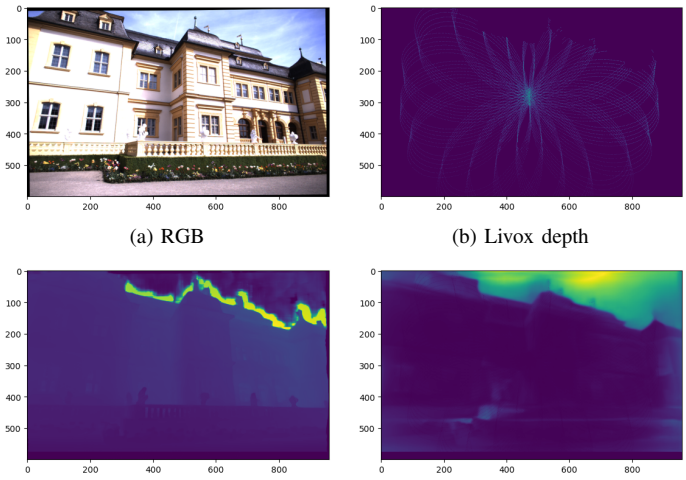
Our experiments show that SceneFactory is highly competitive or even better than compact SOTAs in such applications. The high quality and broad applicability of the design further enhances the progress of our design.

Needless to say, a lot of work remains to be done. In future work, we will focus on adding more applications to the production lines, such as deformable reconstruction, active SLAM, or scene understanding [62].

(a) Aerial image, oblique view ($\sim 300m \times 200m$)

(b) Our result (RGB)

Fig. 18. SceneFactory's Dense Mono SLAM on our very-large range scene, i.e., University campus.



(a) RGB

(b) Livox depth

(c) Completed depth

(d) Depth variance

Fig. 19. Example for the completed LiDAR depth with ScaleCov.

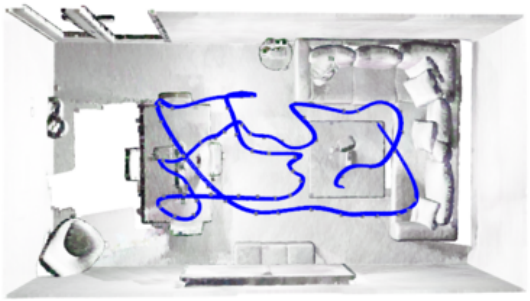


Fig. 20. Depth-only SLAM result on the Replica dataset. Shown without the ceiling for better visualization.

REFERENCES

[1] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European conference on computer vision*. Springer, 2014, pp. 834–849.

[2] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE transactions on robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[3] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, 2021.

[4] K. Ebadi, L. Bernreiter, H. Biggie, G. Catt, Y. Chang, A. Chatterjee, C. E. Denniston, S.-P. Deschênes, K. Harlow, S. Khattak, L. Nogueira, M. Palieri, P. Petráček, M. Petrlík, A. Reinke, V. Krátký, S. Zhao, A.-a. Agha-mohammadi, K. Alexis, C. Heckman, K. Khosoussi, N. Kottege, B. Morrell, M. Hutter, F. Pauling, F. Pomerleau, M. Saska, S. Scherer, R. Siegwart, J. L. Williams, and L. Carlone, "Present and Future of SLAM in Extreme Environments: The DARPA SubT Challenge," *IEEE Transactions on Robotics*, vol. 40, pp. 936–959, 2024.

[5] C. Kerl, J. Sturm, and D. Cremers, "Dense visual slam for rgb-d cameras," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 2100–2106.

[6] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE international symposium on mixed and augmented reality*. Ieee, 2011, pp. 127–136.

[7] Davison, "Real-time simultaneous localisation and mapping with a single camera," in *Proceedings Ninth IEEE International Conference on Computer Vision*. IEEE, 2003, pp. 1403–1410.

[8] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[9] R. Lösch, M. Sastuba, J. Toth, and B. Jung, "Converting depth images and point clouds for feature-based pose estimation," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

[10] W. Yin, C. Zhang, H. Chen, Z. Cai, G. Yu, K. Wang, X. Chen, and C. Shen, "Metric3d: Towards zero-shot metric 3d prediction from a single image," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 9043–9053.

[11] B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy, and T. Brox, "Demon: Depth and motion network for learning monocular stereo," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 5038–5047.

[12] H. Zhou, B. Ummenhofer, and T. Brox, "Deeptam: Deep tracking and mapping," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 822–838.

[13] Z. Teed and J. Deng, "Deepv2d: Video to depth with differentiable structure from motion," in *International Conference on Learning Representations*, 2019.

[14] P.-H. Huang, K. Matzen, J. Kopf, N. Ahuja, and J.-B. Huang, "Deepmvs: Learning multi-view stereopsis," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2821–2830.

[15] Y. Yao, Z. Luo, S. Li, T. Fang, and L. Quan, "Mvsnet: Depth inference for unstructured multi-view stereo," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 767–783.

[16] Y. Yao, Z. Luo, S. Li, T. Shen, T. Fang, and L. Quan, "Recurrent mvsnet for high-resolution multi-view stereo depth inference," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 5525–5534.

[17] J. Yang, W. Mao, J. M. Alvarez, and M. Liu, "Cost volume pyramid based depth inference for multi-view stereo," in *Proceedings of the*

IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 4877–4886.

[18] X. Gu, Z. Fan, S. Zhu, Z. Dai, F. Tan, and P. Tan, "Cascade cost volume for high-resolution multi-view stereo and stereo matching," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 2495–2504.

[19] J. Zhang, S. Li, Z. Luo, T. Fang, and Y. Yao, "Vis-mvsnet: Visibility-aware multi-view stereo network," *International Journal of Computer Vision*, vol. 131, no. 1, pp. 199–214, 2023.

[20] S. Wang, V. Leroy, Y. Cabon, B. Chidlovskii, and J. Revaud, "Dust3r: Geometric 3d vision made easy," *arXiv preprint arXiv:2312.14132*, 2023.

[21] J. Edstedt, I. Athanasiadis, M. Wadenbäck, and M. Felsberg, "Dkm: Dense kernelized feature matching for geometry estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 17 765–17 775.

[22] Z. Yu, S. Peng, M. Niemeyer, T. Sattler, and A. Geiger, "Monosdf: Exploring monocular geometric cues for neural implicit surface reconstruction," *Advances in neural information processing systems*, vol. 35, pp. 25 018–25 032, 2022.

[23] M. M. Johari, C. Carta, and F. Fleuret, "Eslam: Efficient dense slam system based on hybrid representation of signed distance fields," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 17 408–17 419.

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

[25] Y. Yuan and A. Nüchter, "Online learning of neural surface light fields alongside real-time incremental 3d reconstruction," *IEEE Robotics and Automation Letters*, 2023.

[26] J. Huang, S.-S. Huang, H. Song, and S.-M. Hu, "Di-fusion: Online implicit 3d reconstruction with deep priors," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 8932–8941.

[27] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.-Y. Lo, J. Johnson, and G. Gkioxari, "Accelerating 3d deep learning with pytorch3d," *arXiv preprint arXiv:2007.08501*, 2020.

[28] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE international symposium on mixed and augmented reality*. Ieee, 2011, pp. 127–136.

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

[30] Y. Yuan and A. Nüchter, "An algorithm for the se (3)-transformation on neural implicit maps for remapping functions," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 7763–7770, 2022.

[31] ——, "Uni-fusion: Universal continuous mapping," *IEEE Transactions on Robotics*, 2024.

[32] Z. Zhu, S. Peng, V. Larsson, W. Xu, H. Bao, Z. Cui, M. R. Oswald, and M. Pollefeys, "Nice-slam: Neural implicit scalable encoding for slam," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 786–12 796.

[33] M. Bloesch, J. Czarnowski, R. Clark, S. Leutenegger, and A. J. Davison, "Codeslam—learning a compact, optimisable representation for dense visual slam," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2560–2568.

[34] J. Czarnowski, T. Laidlow, R. Clark, and A. J. Davison, "Deepfactors: Real-time probabilistic dense monocular slam," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 721–728, 2020.

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

[36] Z. Teed and J. Deng, "Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras," *Advances in neural information processing systems*, vol. 34, pp. 16 558–16 569, 2021.

[37] A. Rosinol, J. J. Leonard, and L. Carlone, "Probabilistic volumetric fusion for dense monocular slam," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023, pp. 3097–3105.

[38] C.-M. Chung, Y.-C. Tseng, Y.-C. Hsu, X.-Q. Shi, Y.-H. Hua, J.-F. Yeh, W.-C. Chen, Y.-T. Chen, and W. H. Hsu, "Orbeez-slam: A real-time monocular visual slam with orb features and nerf-realized mapping," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 9400–9406.

[39] A. Rosinol, J. J. Leonard, and L. Carlone, "Nerf-slam: Real-time dense monocular slam with neural radiance fields," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 3437–3444.

[40] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, 2021.

[41] E. Sucar, S. Liu, J. Ortiz, and A. J. Davison, "imap: Implicit mapping and positioning in real-time," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 6229–6238.

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

[43] P. Schröppel, J. Bechtold, A. Amiranashvili, and T. Brox, "A benchmark and a baseline for robust multi-view depth estimation," in *2022 International Conference on 3D Vision (3DV)*. IEEE, 2022, pp. 637–645.

[44] E. Brachmann, A. Krull, S. Nowozin, J. Shotton, F. Michel, S. Gumhold, and C. Rother, "Dsac-differentiable ransac for camera localization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 6684–6692.

[45] J. Revaud, Y. Cabon, R. Brégier, J. Lee, and P. Weinzaepfel, "Sacreg: Scene-agnostic coordinate regression for visual localization," *arXiv preprint arXiv:2307.11702*, 2023.

[46] C.-H. Lin, C. Kong, and S. Lucey, "Learning efficient point cloud generation for dense 3d object reconstruction," in *proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[47] J. Wang, B. Sun, and Y. Lu, "Mvpnet: Multi-view point regression networks for 3d object reconstruction from a single image," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 8949–8956.

[48] Z. Teed, L. Lipson, and J. Deng, "Deep patch visual odometry," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[49] A. Hagemann, M. Knorr, and C. Stiller, "Deep geometry-aware camera self-calibration from video," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 3438–3448.

[50] M. Hu, W. Yin, C. Zhang, Z. Cai, X. Long, H. Chen, K. Wang, G. Yu, C. Shen, and S. Shen, "A versatile monocular geometric foundation model for zero-shot metric depth and surface normal estimation," 2024.

[51] E. Dexheimer and A. J. Davison, "Learning a depth covariance function," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 13 122–13 131.

[52] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 5828–5839.

[53] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[54] T. Schops, J. L. Schonberger, S. Galliani, T. Sattler, K. Schindler, M. Pollefeys, and A. Geiger, "A multi-view stereo benchmark with high-resolution images and multi-camera videos," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3260–3269.

[55] R. Jensen, A. Dahl, G. Vogiatzis, E. Tola, and H. Aanæs, "Large scale multi-view stereopsis evaluation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 406–413.

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

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

[58] J. L. Schönberger, E. Zheng, J.-M. Frahm, and M. Pollefeys, "Pixelwise view selection for unstructured multi-view stereo," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III 14*. Springer, 2016, pp. 501–518.

[59] Z. Yang, Z. Ren, Q. Shan, and Q. Huang, "Mvs2d: Efficient multi-view stereo via attention-driven 2d convolutions," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 8574–8584.

[60] R. from http://www.cloudcompare.org/, "Cloudcompare (version 2.11)," 2024.

[61] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt, "Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly

surface reintegration," *ACM Transactions on Graphics (ToG)*, vol. 36, no. 4, p. 1, 2017.
[62] Y. Ming, X. Yang, W. Wang, Z. Chen, J. Feng, Y. Xing, and G. Zhang, "Benchmarking neural radiance fields for autonomous robots: An overview," *arXiv preprint arXiv:2405.05526*, 2024.

SUPPLEMENTARY

A. Good Neighbor Selection

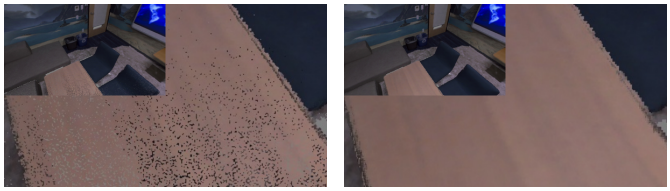
The dense bundle adjustment relies on the selection of good neighbors. We select neighbor frames by filtering the relative poses according to Algorithm 2.

Algorithm 2: Depth neighbor frame selection.

Function `FindGoodNeighbors`$([\mathbf{G}_{nb,1}, \cdots], \mathbf{G}_i), \tau_{nb} = 2)$**:**
  `// select frame by relative pose`
  $ids = []; Ts = []$
  for $\mathbf{G}_j \in [\mathbf{G}_{nb,1}, \cdots]$ **do**
    `// transformation from i to j`
    $\mathbf{T}_{ji} = \mathbf{G}_i^{-1}\mathbf{G}_j$;
    $\mathbf{R}_{ji}, \mathbf{t}_{ji} = \mathbf{T}_{ji}$;
    $l_{baseline} = ||\mathbf{t}_{ji}||_2$;
    $\theta_{facing} = \arccos([0,0,1]\mathbf{R}_{ji}[0,0,1]^T)$;
    if $l_{baseline} > \tau_{baseline}$ ***and*** $\theta_{facing} > \tau_{facing}$ **then**
      $ids.append(j)$; $Ts.append(\mathbf{T}_{ji})$
    end
  end
  if $len(ids) < \tau_{nb}$ **then**
    return $\emptyset$;
  else
    `// return the best` τ_{nb} `XY directional baselines`
    return $SortByBaselineXY(ids, Ts)[:\tau_{nb}]$;
  end
End Function

B. Improved Point Rasterization Ablation Test

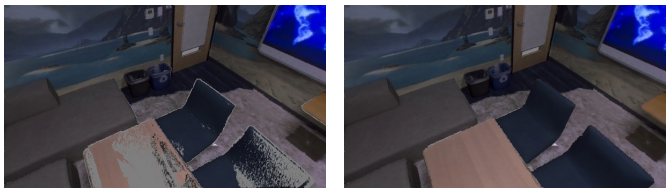
a) Effect of Adaptive Radius: Please note Fig. 21, there is a black hole on the desktop. This is due to the fixed radius of pytorch3d's point rasterization in NDC space. Which would be much more severe if the camera goes even closer. While our IPR does not have this problem.



(a) Pytorch3d Rast. ($\sim 0.57s$) (b) Ours Rast. ($\sim 0.025s$)

Fig. 21. The effect of the Adaptive Radius (left pytorch3d) and our implementation (right) in an arbitrary view.

b) Effect of First-layer filter: Apart from the adaptive radius, our IPR also gets the surface with a layer filter as in Fig. 22. Without the layer filter, the resulting surface points will be in the middle of multiple surface layers, resulting in empty space during color rendering.



(a) w/o Layer filter (b) w/ Layer Filter

Fig. 22. The application of the Layer Filter (right) removed blank space (left).

C. Description of the Sensor Systems

For collecting our own RGB-X dataset, described in Sec. VI-A2, two commercial sensors are employed: the Microsoft Azure Kinect RGB-D sensor and the built-in camera of a Xiaomi Redmi smartphone. Furthermore, two custom-built sensor systems are used for capturing the datasets:

a) Handheld mapping system: Our custom-built handheld mapping device is show in Fig. 9(a). It is based on a commercial camera rig with the sensors mounted to the top bar. The sensors are a Livox AVIA Lidar and two IDS U3-30C0CP global-shutter cameras with a Sony IMX392 2.35 MPixel RGB CMOS sensor. The Lidar sensor has a Field of View (FoV) of $70.4° \times 77.2°$. The cameras are equipped with 4 mm lenses, which results in a similar FoV of $77.3° \times 61.9°$. Below the sensors a 3D printed enclosure is mounted with an embedded PC for data recording and power supply electronics. The cameras and lidar are co-calibrated using a calibration board.

From the system we extract a synchronized RGB-L data stream with Lidar scans and camera images with 10 Hz. While the system features stereo cameras, in this work we focus on mono SLAM. Therefore, only the camera closest to the Lidar sensor is used. The data is captured with the system handheld and an average walking speed of 0.84 $\frac{m}{s}$. During data collection the sensors always points towards the captured object. The trajectory around Veitshöchheim Palace is 251 m long and was captured in 5 min. It consists of 3000 Lidar scans and 3000 RGB images.

b) UAV mapping system: The UAV mapping system is based on the DJI Matrice 300 RTK and is shown in Fig. 9(b). It carries an Ouster OS1-128 Lidar and a LUCID Vision Labs Phoenix PHX032S-CC global-shutter camera with a Sony IMX265 3.2 MPixel RGB CMOS sensor. The Lidar sensor has a FoV of $360° \times 45°$. The camera is used with a 4.5 mm fixed lens with a horizontal FoV of $84.7°$. The co-calibrated camera is mounted directly on top of the Lidar sensor. For the data collection an embedded PC is mounted on top of the UAV. The aerial imagery is captured with an oblique angle and a lawnmower pattern flight trajectory.