Mobile3DRecon: Real-time Monocular 3D Reconstruction on a Mobile Phone

Xingbin Yang, Liyang Zhou, Hanqing Jiang, Zhongliang Tang, Yuanbo Wang, Hujun Bao, *Member, IEEE*, and Guofeng Zhang, *Member, IEEE*

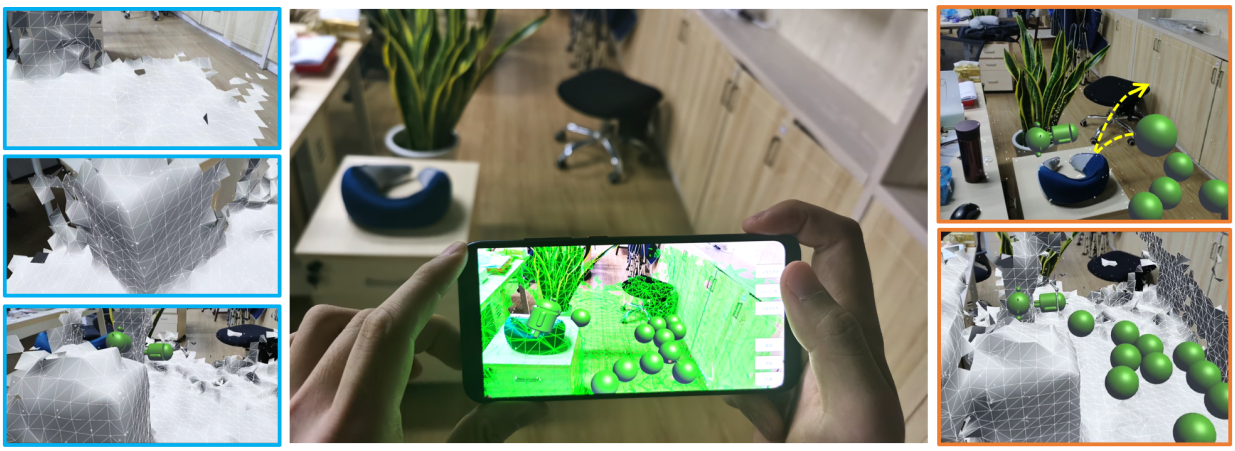

Fig. 1: Examplar case "Indoor office" reconstructed in real-time on MI8 using our Mobile3DRecon system. Note on the left column that surface mesh is incrementally reconstructed online as we navigate through the environment. With the reconstructed 3D environment, realistic AR interactions with the real scene can be achieved, including virtual object placement and collision, as shown on the right column.

Abstract—We present a real-time monocular 3D reconstruction system on a mobile phone, called Mobile3DRecon. Using an embedded monocular camera, our system provides an online mesh generation capability on back end together with real-time 6DoF pose tracking on front end for users to achieve realistic AR effects and interactions on mobile phones. Unlike most existing state-of-the-art systems which produce only point cloud based 3D models online or surface mesh offline, we propose a novel online incremental mesh generation approach to achieve fast online dense surface mesh reconstruction to satisfy the demand of real-time AR applications. For each keyframe of 6DoF tracking, we perform a robust monocular depth estimation, with a multi-view semi-global matching method followed by a depth refinement post-processing. The proposed mesh generation module incrementally fuses each estimated keyframe depth map to an online dense surface mesh, which is useful for achieving realistic AR effects such as occlusions and collisions. We verify our real-time reconstruction results on two mid-range mobile platforms. The experiments with quantitative and qualitative evaluation demonstrate the effectiveness of the proposed monocular 3D reconstruction system, which can handle the occlusions and collisions between virtual objects and real scenes to achieve realistic AR effects.

Index Terms—real-time reconstruction, monocular depth estimation, incremental mesh generation

1 Introduction

Augmented reality (AR) is playing an important role in presenting virtual 3D information in the real world. A realistic fusion of a virtual 3D object with a real scene relies on the consistency in 3D space between virtuality and reality, including consistent localization, visibility, shadow and interaction like physical occlusion between the virtual object and the real scene. To achieve consistent localization in space, a SLAM system can be performed for real-time alignment of virtual object to the real environment. More and more visual-inertial 6DoF odometry systems like VINS-Mobile [19] has been studied to run on mobile phones, which greatly extends the applications of AR in mobile industry. However, most of these works focused on sparse mapping without special consideration on how to reconstruct dense geometrical structures of the scene. A dense surface scene representation is the key to a full 3D perception of our environment, and an important prerequisite for more realistic AR effects including consistency of occlusion, shadow mapping and collision between virtual objects and real environment. More recent works such as KinectFusion [24] and BundleFusion [1] tried to perform 6DoF odometry with dense mapping. However, these works rely on videos augmented with precise depths captured by a RGB-D camera as input, which is currently impossible for most mid-range mobile phones. Besides, due to the huge computation of dense mapping process, most of these systems work on desktop PC or high-end mobile devices. Some works [23, 26] also tried to reconstruct dense surface on mobile phones with monocular camera, but are limited in reconstruction scale due to complexity of both computation and memory. Moreover, these systems cannot gen-

- *Xingbin Yang, Liyang Zhou, Hanqing Jiang, Zhongliang Tang, and Yuanbo Wang are with Sensetime Research. E-mail: {yangxingbin,zhouliyang,jianghanqing,tangzhongliang,wangyuanbo}@sensetime.com.*
- *Hujun Bao and Guofeng Zhang are with the State Key Lab of CAD&CG, Zhejiang University. E-mail: {bao,zhangguofeng}@cad.zju.edu.cn.*
- *Corresponding Author: Guofeng Zhang.*
- *Xingbin Yang, Liyang Zhou, and Hanqing Jiang assert equal contribution and joint first authorship.*

Manuscript received 18 May 2020; revised 26 July 2020; accepted 17 Aug. 2020. Date of publication 21 Sept. 2020; date of current version 3 Nov. 2020. Digital Object Identifier no. 10.1109/TVCG.2020.3023634

erate surface mesh online, which certainly degrates their applications in real-time realistic AR.

To make seamless AR available to more middle-end mobile platforms, this paper introduces a new system for real-time dense surface reconstruction on mobile phones, which we name as Mobile3DRecon. Our Mobile3DRecon system can perform real-time surface mesh reconstruction on mid-range mobile phones with monocular camera we usually have in our pockets, without any extra hardware or depth sensor support. We focus on the 3D reconstruction requirement of realistic AR effects by proposing a keyframe-based real-time surface mesh generation approach, which is essential for AR applications on mobile phones. Our main contributions can be summarized as:

- We propose a multi-view keyframe depth estimation method, which can robustly estimate dense depths even in textureless regions with a certain pose error. We introduce a multi-view semi-global matching (SGM) with a confidence-based filtering to reliably estimate depths and remove unreliable depths caused by pose errors or textureless regions. Noisy depths are further optimized by a deep neural refinement network.

- We propose an efficient incremental mesh generation approach which fuses the estimated keyframe depth maps to reconstruct the surface mesh of the scene online with local mesh triangles updated incrementally. This incremental meshing approach not only provides online dense 3D surface for seamless AR effects on front end, but also ensures real-time performance of mesh generation as back-end CPU module on mid-range mobile platforms, which is difficult for previous online 3D reconstruction systems.

- We present a real-time dense surface mesh reconstruction pipeline with a monocular camera. On mid-range mobile platform, our monocular keyframe depth estimation and incremental mesh updating are performed at no more than 125 ms/keyframe on back end, which is fast enough to keep up with the front-end 6DoF tracking at more than 25 frames-per-second (FPS).

This paper is organized as follows: Section 2 briefly presents related work. Section 3 gives an overview of the proposed Mobile3DRecon system. The monocular depth estimation method and the incremental mesh generation approach are described in Sections 4 and 5 respectively. Finally, we evaluate the proposed solution in Section 6.

2 Related Work

The development of consumer RGB-D cameras such as Microsfot Kinect and Intel RealSense gives rise to a large number of real-time dense reconstruction systems. With an RGB-D camera, a real-time dense reconstruction system simultaneously localizes the camera using iterative closest point (ICP), and fuses all the tracked camera depths into a global dense map represented by TSDF voxels or surfels. An impressive work is KinectFusion [24], which tracks an input Kinect to the ray-casted global model using ICP, and fuses depths into the global map using TSDF. KinectFusion is not able to work for large scale scenes, due to the computation and memory limitations of TSDF voxels. More recent dense mapping works such as BundleFusion [1] used voxel hashing [25] to break through the limitations of TSDF fusion. InfiniTAM [15] proposed a highly efficient implementation of voxel hashing on mobile devices. A more light-weight spatially hashed SDF strategy on CPU for mobile platform was proposed in CHISEL [17]. ElasticFusion [37] uses surfel representation for dense frame-to-model tracking and explicitly handles loop closures using non-rigid warping. Surfel map representation is more suitable for online refinement of both pose and underlying 3D dense map, which is diffcult to handle by TSDF voxel fusion. However, TSDF is more suitable for online 3D model visualization and ray intersection in AR applications, which is hard for surfels. An online TSDF refinement is performed in [1] by voxel deintegration.

Although impressive dense reconstruction quality can be achieved by an RGB-D camera, few mobile phones have been equipped with it today. Besides, most consumer RGB-D cameras are not able to work well in outdoor environments. These limitations encouraged researchers to explore real-time dense reconstruction with monocular RGB cameras which are smaller, cheaper, and widely equipped on a mobile phone. Without input depths, a real-time dense reconstruction system should estimate depths of the input RGB frames, and fuse the estimated depths into a global 3D model by TSDF or surfels. MonoFusion [30] presented a real-time dense reconstruction with a single web camera and MobileFusion [26] proposed a real-time 3D model scanning tool on mobile devices with monocular camera. Both works perform volume-based TSDF fusion without voxel hashing, and therefore can only reconstruct in a limited 3D space. For large-scale scenes, Tanskanen et al. [36] proposed a live reconstruction system on mobile phones, which can perform inertial metric scale estimation while producing dense surfels of the scanned scenes online. Kolev et al. [18] enhanced the pipeline of [36] by introducing a confidence-based depth map fusion method. Garro et al. [3] proposed a fast metric reconstruction algorithm on mobile devices, which solves metric scale estmation problem with a novel RANSAC-based alignment approach in inertial measurement unit (IMU) acceleration space. Schöps et al. [32] estimates sparse depths via motion stereo with a monocular fisheye camera on the GPU of Google's Project Tango Tablets, and integrates the filtered depths by the Tango's volumetric fusion pipeline for large-scale outdoor environments. CHISEL [17] proposed a system for real-time dense 3D reconstruction on a Google's Project Tango. Yang et al. [42] infers online depths by two-frame motion stereo with temporal cost aggregation and semi-global optimization, and fuses depths into an occupany map with uncertainty-aware TSDF, to achieve a 10Hz online dense reconstruction system with the help of Nvidia Jetson TX1 GPU on aerial robots. Very few works achieve real-time outdoor scene reconstruction on mid-range mobile phones with monocular camera.

There are also some offline dense reconstruction works on mobile devices. 3DCapture [23] presented a dense textured model reconstruction system, which starts with an online RGB and IMU data capturing stage followed by an offline post-processing reconstruction. The main reconstruction steps including pose tracking, depth estimation, TSDF depth fusion, mesh extraction and texture mapping, are all done as the post-processing stage on mobile devices. Poiesi et al. [28] described another cloud-based dense scene reconstruction system that performs Structure-from-Motion (SfM) and local bundle adjustment on monocular videos from smartphones to reconstruct a consistent point cloud map for each client, and run periodic full bundle adjustments to align the maps of various clients on a cloud server. Some other works presented real-time dense scene reconstruction with GPU acceleration on desktop PC. For example, Merrell et al. [22] proposed a real-time 3D reconstruction pipeline on PC, which utilizes visibility-based and confidence-based fusion for merging multiple depth maps to online large-scale 3D model. Pollefeys et al. [29] presented a complete system for real-time video-based 3D reconstruction, which captures large-scale urban scenes with multiple video cameras mounted on a driving vehicle.

As depth estimation is a key stage of our proposed reconstruction pipeline, our work is related to a great amount of works on binocular and multi-view stereo, which have been thoroughly investigated in [27, 31, 33, 35]. REMODE [27] carries out a probabilistic depth measurement model for archieving real-time depth estimation on a laptop computer, with the help of CUDA parallelism. For achieving higher quality depth inference from multi-view images, more recent works such as [5, 10–12, 34, 40, 41, 43] employ deep neural networks (DNN) to address depth estimation or refinement problems. However, the generalization performance of most DNN-based methods will be affected by camera pose errors or texturelessness in practical applications. Besides, most of these depth estimation works are not applicable to mobile phones as they do not meet the underlying efficiency requirements by mobile platforms. Online depth from motion on mobile devices has been almost exclusively studied in the literature of dense 3D reconstruction, such as [18, 26, 32, 36, 42]. Due to the limited computing power provided by mobile platforms, sparse depth maps are online estimated in these methods, with historical frames selected as references for stereo. A multi-resolution scheme is used to speed up dense

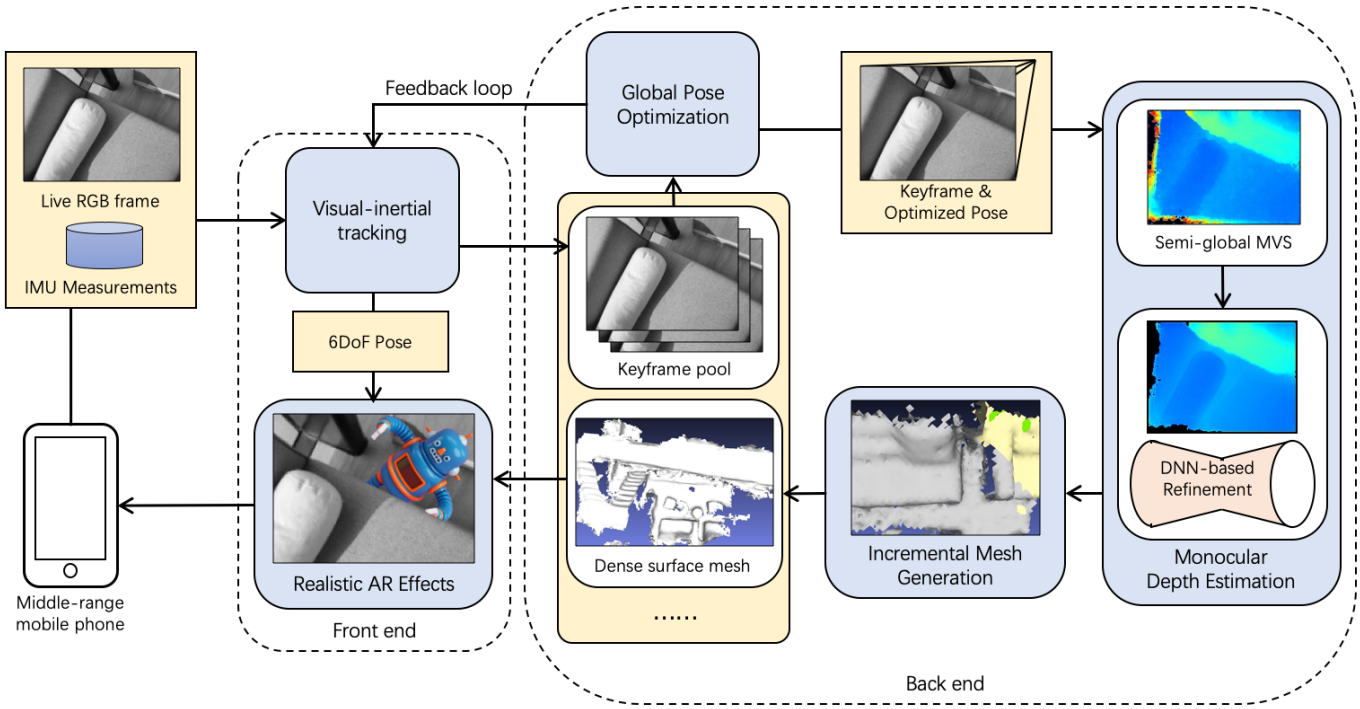

Fig. 2: System framework.

stereo matching in [36], with GPU acceleration. Valentin et al. [39] proposed a low latency dense depth map estimation method using a single CPU on mid-range mobile phones. The proposed method incorporates a planar bilateral solver to filter planar depths while refining depth boundaries, which is more suitable for occlusion rendering purpose. But for our dense reconstruction purpose, accurate depths should be estimated online with limited mobile platform resources, which is indeed a difficult problem for most existing depth estimation works.

Another important stage of our pipeline is surface extraction. Some real-time dense reconstruction works such as [18, 36, 37] only generate surfel cloud. Poisson surface reconstruction [16] should be done as a post-process to extract surface model from surfels. However, AR applications usually require real-time surface mesh generation for interactive anchor placement and collision. Although works such as [26] update TSDF volume in real-time, surface mesh extraction from TSDF is done offline by Marching cubes [21]. [42] uses TSDF only to maintain a global occupancy map instead of 3D mesh. CHISEL [17] proposed a simple incremental Marching cubes scheme, in which polygonal meshes are generated only for part of the scene that need to be rendered. In comparison, our work achieves a truly incremental real-time surface mesh generation to ensure both incremental mesh updating and concise mesh topology, which satisfies the requirements of graphics operations by realistic AR effects to a larger extent.

3 System Overview

We now outline the steps of the proposed real-time scene reconstruction pipeline as in Fig. 2.

As a user navigates into his environment by a mobile phone with a monocular camera, our pipeline tracks 6DoF poses of the mobile phone using a keyframe-based visual-inertial SLAM system, which tracks the 6DoF poses on the front end, while maintaining a keyframe pool on the back end with a global optimization module to refine poses of all the keyframes as feedback to the front end tracking. We use SenseAR SLAM [SenseTime] [1] in our pipeline for pose tracking. Note that any other keyframe-based VIO or SLAM system such as ARCore [Google] can be used at this stage.

[1] http://openar.sensetime.com

After the 6DoF tracking is initialized normally on the front end, for a latest incoming keyframe from the keyframe pool with its globally optimized pose, its dense depth map is online estimated by multi-view SGM, with a part of previous keyframes selected as reference frames. A convolutional neural network follows the multi-view SGM to refine depth noise. The refined key-frame depth map is then fused to generate dense surface mesh of the surrounding environment. Our pipeline performs an incremental online mesh generation, which is more suitable for the requirement of real-time 3D reconstruction by AR application on mobile phone platform. Both the depth estimation and the incremental meshing are done as back-end modules. As a dense mesh is gradually reconstructed at the back end, High level AR applications can utilize this real-time dense mesh and the 6DoF SLAM poses to achieve realistic AR effects for the user on the front end, including AR occlusions and collisions, as shown in Fig. 1. In the following sections, the main steps of our pipeline will be described in detail.

4 Monocular Depth Estimation

With the global optimized keyframe poses from SenseAR SLAM platform, we estimate a depth map for each incoming keyframe online. Some recent works [5, 10, 43] tried to solve multi-view stereo problem with DNN, but they turn out to have generalization problem in practical applications with pose errors, and textureless regions. As can be seen in Fig. 5(a), especially for online SLAM tracking, there are usually inevitable inaccurate poses with epipolar errors more than 2 pixels, which will certainly affect the final multi-view depth estimation results. Instead, our system solves multi-view depth estimation by a generalized multi-view SGM algorithm. Different from the general two-view SGM [7, 8], our proposed SGM approach is more robust because depth noise can be suppressed by means of multi-view costs. Besides, no stereo rectification needs to be done ahead as we compute costs at depth space directly instead of disparity space. Moreover, considering the inevitable depth errors causes by pose errors, textureless region or occlusions, a DNN-based depth refinement scheme is used to filter and refine the initial depth noise. We will show that our scheme of a multi-view SGM followed by a DNN-based refinement performs better than the end-to-end multi-view stereo network.

The whole depth estimation process runs on a single back-end

CPU thread to avoid occupancy of front-end resources used by SLAM tracking or GPU graphics. Since depth estimation is done for each keyframe, we only need to keep up with the frequency of keyframes (almost 5 keyframes-per-second) in order to achieve a real-time performance of depth estimation on mid-range mobile platforms such as OPPO R17 Pro. We will describe our monocular depth estimation in key details.

4.1 Reference Keyframes Selection

For each incoming keyframe, we seek a set of neighboring keyframes as reference frames that are good for multi-view stereo. First, the reference keyframes should provide sufficient parallax for stable depth estimation. This is because in a very short baseline camera setting, a small pixel matching error in image domain can cause a large fluctuation in depth space. In a second aspect, to achieve complete scene reconstruction, current selected keyframe should have as large overlap as possible with the reference ones. In other words, we want to make sure there is enough overlap between the current keyframe and the selected reference ones.

To meet the requirements above, we select those neighboring keyframes far away from the current keyframe, while avoiding too large baseline which may cause low overlap. For simplicity, we use t to denote a keyframe at time t. Therefore, for each keyframe t, we expoit a baseline score between t and another keyframe t' as:

$$w_b(t,t') = \exp\left(-(b(t,t')-b_m)^2/\delta^2\right), \tag{1}$$

where $b(t,t')$ is the baseline between t and t'. b_m is the expectation and δ is standard deviation. We set $b_m = 0.6$m and $\delta = 0.2$ in our experiments. Meanwhile, high overlap should be kept between t and t' to cover as larger common perspective of view as possible for better matching. We compute the angle of optical axis between the two frames. Generally, a larger angle indicates more different perspectives. To encourage a higher overlap, we define a viewing score between t and t' as:

$$w_v(t,t') = \max(\alpha_m/\alpha(t,t'),1), \tag{2}$$

where $\alpha(t,t')$ is the angle between the optical viewing directions between t and t'. The function scores the angles below α_m, which is set to 10 degrees in our experiments.

The scoring function is simply the product of these two terms:

$$S(t,t') = w_b(t,t') * w_v(t,t'). \tag{3}$$

For each new keyframe, we search in the historical keyframe list for reference frames. The historical keyframes are sorted by the scores computed by function (3), in which the top keyframes are chosen as the reference frames for stereo. Too many reference frames are helpful for ensuring accuracy of depth estimation, but will certainly slow down the stereo matching computation, especially on mobile phones. Therefore, we only choose the top two as a trade-off between accuracy and computation efficiency, as shown in the "Indoor stairs" case of Fig. 5(a-b). We compute a final score for each new keyframe t as the sum of the scores between t as its reference keyframes as follows:

$$S(t) = \sum_{t' \in N(t)} S(t,t'), \tag{4}$$

where $N(t)$ denotes the reference keyframes of t. The computed final score will be used as a weight for the following multi-view stereo cost computation.

4.2 Multi-View Stereo Matching

For each new keyframe, we propose to estimate its depth map using an SGM based multi-view stereo approach. We uniformly sample the inverse depth space to L levels. Suppose the depth measurement is bounded to a range from $d_{\min}$ to $d_{\max}$. The l-th sampled depth can be computed as follows:

$$d_l = \frac{(L-1)d_{\min}d_{\max}}{(L-1-l)d_{\min} + l\,(d_{\max}-d_{\min})}, \tag{5}$$

where $l \in \{0,1,2...,L-1\}$, and d_l is the sampled depth at the l-th level. Given a pixel $\mathbf{x}$ with depth d_l on keyframe t, its projection pixel $\mathbf{x}_{t\to t'}(d_l)$ on frame t' by d_l can be calculated by:

$$\mathbf{x}_{t\to t'}(d_l) \sim d_l \mathbf{K}_{t'}\mathbf{R}_{t'}\mathbf{R}_t^{\mathrm{T}}\mathbf{K}_t^{-1}\hat{\mathbf{x}} + \mathbf{K}_{t'}(\mathbf{T}_{t'} - \mathbf{R}_{t'}\mathbf{R}_t^{\mathrm{T}}\mathbf{T}_t), \tag{6}$$

where $\mathbf{K}_{t'}$, $\mathbf{K}_t$ are the camera intrinsic matrices of keyframe t and t'. $\mathbf{R}_{t'}$, $\mathbf{R}_t$ are rotation matrices, and $\mathbf{T}_t$, $\mathbf{T}_{t'}$ are translation matrices respectively. Note that $\hat{\mathbf{x}}$ is the homogeneous coordinate of $\mathbf{x}$.

We resort to a variant of Census Transform (CT) [6] as the feature descriptor to compute patch similarity. Compared to other image similarity measurements such as Normalized Cross Correlation, CT has the characteristic of boundary preservation. Besides, mobile applications considerably benefit from the binary representation of CT. Therefore, our matching cost is determined as follows:

$$\begin{aligned} C(\mathbf{x},l) &= \sum_{t' \in N(t)} w_{t'} CT(\mathbf{x}, \mathbf{x}_{t\to t'}(d_l)) \\ w_{t'} &= S(t') / \sum_{\tau \in N(t)} S(\tau) \end{aligned}, \tag{7}$$

where N is the number of selecting keyframes. $w_{t'}$ is the cost weight of reference frame t', and $S(t')$ is the final score of t'. $CT(\mathbf{x}, \mathbf{x}_{t\to t'}(d_l))$ is the Census cost of the two patches centered at $\mathbf{x}$ and $\mathbf{x}_{t\to t'}(d_l)$ respectively. We use a lookup-table to calculate the hamming distance between two Census bit string. We traverse every pixel corresponding to each slice with label l and compute its matching cost according to Eq. 7. After that, we get a cost volume with size $W \times H \times L$, where W and H are width and height of the frame. Then the cost volume is aggregated, with a Winner-Take-All strategy taken subsequently to obtain the initial depth map.

Compared to the conventional binocular setting [8], we select multiple keyframes as reference to accumulate costs for suppression of depth noise caused by camera pose errors or textureless regions. Although pose errors can be suppressed by multi-view matching to some extent, there are still generally ambiguous matches in repetitive patterns or texture-less regions, which result in noisy depth map. Inspired by the method proposed in [7, 8], we add an additional regularization to support smoothness by penalizing depth labeling changes of pixel neighborhood. Specifically, for image pixel $\mathbf{x}$ with label l, the cost aggregation is done by recursive computation of costs in neighboring directions as:

$$\hat{C}_r(\mathbf{x},l) = C(\mathbf{x},l) + \min\begin{pmatrix} \hat{C}_r(\mathbf{x}-r,l), \\ \hat{C}_r(\mathbf{x}-r,l-1)+P_1, \\ \hat{C}_r(\mathbf{x}-r,l+1)+P_1, \\ \min_i \hat{C}_r(\mathbf{x}-r,i)+P_2 \end{pmatrix} - \min_k \hat{C}_r(\mathbf{x}-r,k), \tag{8}$$

where $\hat{C}_r(\mathbf{x},l)$ is the aggregated cost of $\mathbf{x}$ with label l in neighboring direction r, and $r \in N_r$, which is the neighboring direction set (we use 8 neighborhood). P_1 and P_2 are the penalty values. Since intensity difference usually indicates depth discontinuity, we set $P_2 = -a*|\nabla I_t(\mathbf{x})| + b$. Here $\nabla I_t(\mathbf{x})$ is the intensity gradient of $\mathbf{x}$ in image I_t at keyframe t, which we find out useful for preserving depth boundaries. The value of $\hat{C}_r$ increases along the path, which may lead to extremely large values or even overflow. Therefore, the last term $\min_k \hat{C}_r(\mathbf{x}-r,k)$ is subtracted to avoid permanent increase of the aggregated cost without changing the minimum cost depth level. The aggregated costs of $\mathbf{x}$ at depth label l are recursively computed for all the neighboring directions and summed up. The final cost volume is formed by accumulating each pixel in the image as follows:

$$\hat{C}(\mathbf{x},l) = \sum_{r \in N_r} \hat{C}_r(\mathbf{x},l). \tag{9}$$

The final depth label $\hat{l}(\mathbf{x})$ is given by a Winner-Take-All strategy as:

$$\hat{l}(\mathbf{x}) = \min_l \hat{C}(\mathbf{x},l). \tag{10}$$

In order to get a sub-level depth value, we use parabola fitting to acquire a refined depth level $\hat{l}_s(\mathbf{x})$ as:

$$\hat{l}_s(\mathbf{x}) = \hat{l}(\mathbf{x}) + \frac{\hat{C}\left(\mathbf{x},\hat{l}(\mathbf{x})-1\right) - \hat{C}\left(\mathbf{x},\hat{l}(\mathbf{x})+1\right)}{2\hat{C}\left(\mathbf{x},\hat{l}(\mathbf{x})-1\right) - 4\hat{C}\left(\mathbf{x},\hat{l}(\mathbf{x})\right) + 2\hat{C}\left(\mathbf{x},\hat{l}(\mathbf{x})+1\right)}. \tag{11}$$

We replace l in Eq. (5) with the refined depth level $\hat{l}_s(\mathbf{x})$ to get a more accurate sub-level depth measurement for $\mathbf{x}$. With the sub-level depths, we get an initial depth map D_t^i for each keyframe t, as shown in Fig. 5(b).

On mobile platform, we use NEON instruction set to significantly accelerate the cost volume computation and aggregation optimization. Take OPPO R17 Pro with Qualcomm Snapdragon 710 chip as example, computational speed by NEON acceleration is 2 times faster for cost volume computation and 8 times faster for aggregation, which ensures a real-time performance on mobile phone.

4.3 Depth Refinement

Although our multi-view stereo matching is robust to unstable SLAM tracking to some extent, there are still noisy depths in the initialized depth maps which come from the mistaken matching costs caused by camera pose errors or textureless regions, as seen in Fig. 5(b). Therefore, a depth refinement strategy is introduced as a post-processing step to correct depth noise. The proposed refinement scheme consists of a confidence-based depth filtering and a DNN-based depth refinement, which will be described in more details.

4.3.1 Confidence Based Depth Filtering

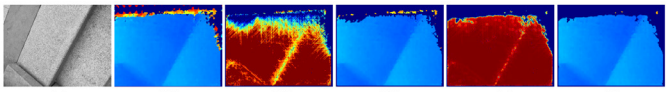

Fig. 3: Confidence-base depth map filtering compared to Drory et al. [2]. (a) A representative keyframe. (b) The depth map estimated by multi-view SGM. (c) The confidence map measured by [2]. (d) The depth map filtered by (c). (e) Our confidence map. (f) Our depth map filtered by (e).

Although the depth maps obtained by our semi-global optimization has good completeness, the depth noise in textureless regions is obvious, which requires confidence measurement for further depth filtering. Drory et al. [2] proposed an uncertainty measurement for SGM, in which they assume the difference between the sum of the lower bound aggregated costs and the minimum of the aggregated costs equals zero. The uncertainty measurement for pixel $\mathbf{x}$ can be expressed as follows:

$$U(\mathbf{x}) = \min_l \sum_{r\in N_r} \hat{C}_r(\mathbf{x},l) - \sum_{r\in N_r} \min_l \left(\hat{C}_r(\mathbf{x},l) - \frac{N_r-1}{N_r} C(\mathbf{x},l)\right). \tag{12}$$

However, this method ignores the neighborhood depth informations when calculating the uncertainty measure $U(\mathbf{x})$, resulting in some isolated noise in the depth map. Considering the fact that neighborhood pixel depths do not change greatly for the correct depth measurements, we calculate a weight $W(\mathbf{x})$ for the uncertainty measurement of $\mathbf{x}$ in its 5×5 local window $\Omega(\mathbf{x})$, which measures the neigboring depth level differences as:

$$\omega(\mathbf{x}) = \frac{1}{|\Omega(\mathbf{x})|} \sum_{\mathbf{y}\in\Omega(\mathbf{x})} \left[\left|\hat{l}_s(\mathbf{x}) - \hat{l}_s(\mathbf{y})\right| > 2\right], \tag{13}$$

where $|\Omega(\mathbf{x})|$ is the number of pixels in local window $\Omega(\mathbf{x})$. We normalize the weighted uncertainty measurement as the final confidence $M(\mathbf{x})$ of $\mathbf{x}$ as:

$$M(\mathbf{x}) = 1 - \frac{\omega(\mathbf{x})U(\mathbf{x})}{\max_{\mathbf{u}\in I_t}(\omega(\mathbf{u})U(\mathbf{u}))}. \tag{14}$$

As can be seen in Fig.3, compared to [2], the proposed confidence-based depth filtering method handles the depth noise more effectively, especially along the depth map borders.

In our implementation, we remove all pixels with confidence lower than 0.15 to get a filtered depth map D_t^f with fewer abnormal noisy depths for each keyframe t, as shown in Fig. 5(c). With this confidence-based filtering, most depth outliers caused by pose errors or textureless regions can be suppressed.

4.3.2 DNN-based Depth Refinement

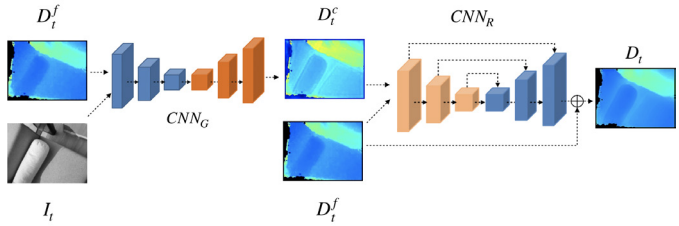

Fig. 4: Depth Refinement Network Flow.

After the confidence-based depth filtering, we employ a deep neural network to refine the remaining depth noise. Our network can be regarded as a two-stage refinement structure, as illustrated in Fig. 4. The first stage is an image-guided sub-network CNN_G which combines the filtered depth D_t^f with the corresponding gray image I_t of keyframe t to reason a coarse refinement result D_t^c. Here, gray image plays a role as guidance for depth refinement. It provides the prior knowledge of object edge and semantic information for CNN_G. With this guidance, the sub-network CNN_G is able to distinguish the actual depth noise to be refined. The second stage is a residual U-Net CNN_R which further refines the previous coarse result D_t^c to get the final refined depth D_t. The U-Net structure mainly contributes to making learning process more stable and overcoming feature degradation problem.

To satisfy our refinement purpose, we follow [4, 9, 14] to exploit three spatial loss functions to penalize depth noise while maintaining object edges. The training loss is defined as:

$$\Phi = \Phi_{edge} + \Phi_{pd} + \lambda\Phi_{vgg}. \tag{15}$$

Φ_{edge} is an edge-maintainance loss, which contains three terms respectively defined as:

$$\begin{aligned} \Phi_{edge} &= \Phi_x + \Phi_y + \Phi_n \\ \Phi_x &= \tfrac{1}{|I_t|}\sum_{\mathbf{x}\in I_t} ln\left(\left|\nabla_x D_t(\mathbf{x}) - \nabla_x D_t^g(\mathbf{x})\right| + 1\right) \\ \Phi_y &= \tfrac{1}{|I_t|}\sum_{\mathbf{x}\in I_t} ln\left(\left|\nabla_y D_t(\mathbf{x}) - \nabla_y D_t^g(\mathbf{x})\right| + 1\right) \\ \Phi_n &= \tfrac{1}{|I_t|}\sum_{\mathbf{x}\in I_t} \left(1 - \eta_t(\mathbf{x}) * \eta_t^g(\mathbf{x})\right) \end{aligned}, \tag{16}$$

where $|I_t|$ is the number of valid pixels in keyframe image I_t. $D_t^g(\mathbf{x})$ represents the ground-truth depth value of pixel $\mathbf{x}$, and $D_t(\mathbf{x})$ is the final refined depth of $\mathbf{x}$. ∇_x and ∇_y denote the Sobel derivatives along the x-axis and y-axis respectively. The depth normal $\eta_t(\mathbf{x})$ is approximated by $\eta_t(\mathbf{x}) = \left[-\nabla_x D_t(\mathbf{x}), \nabla_y D_t(\mathbf{x}), 1\right]$, and $\eta_t^g(\mathbf{x})$ is the ground-truth normal estimated in the same way. Φ_x and Φ_y measure the depth differences along edges. Φ_n measures the similarities between surface normals.

Φ_{pd} is a photometric-depth loss, which minimizes depth noise by forcing the gradient consistency between the gray image I_t and the refined depth D_t as follows:

$$\Phi_{pd} = \frac{1}{|I_t|}\sum_{\mathbf{x}\in I_t}\left(\left|\nabla_x^2 D_t(\mathbf{x})\right| e^{-\alpha|\nabla_x I_t(\mathbf{x})|} + \left|\nabla_y^2 D_t(\mathbf{x})\right| e^{-\alpha|\nabla_y I_t(\mathbf{x})|}\right), \tag{17}$$

where $\alpha = 0.5$ in our experiments. ∇_x^2 and ∇_y^2 are the second derivatives along the x-axis and y-axis respectively. As demonstrated in Eq. (17), Φ_{pd} encourages D_t to be consistent with corresponding gray image I_t in pixel areas with smaller gradients. We use second derivatives to make the refinement process more stable.

Φ_{vgg} is the high-level perceptual loss commonly used in generative adversarial networks like [13, 14]. By minimizing the difference between high-level features, perceptual loss contributes to maintaining global data distribution and avoiding artifacts.

We train the proposed depth refinement network using Demon dataset proposed by University of Freiburg [38] together with our own dataset. We first run our multi-view SGM and confidence-based filtering on Demon dataset to generate a set of initially filtered depth maps and reassemble them with the corresponding ground truth (GT) as training pairs to pre-train the proposed network. The pre-trained refinement model is then finetuned on our own captured sequences, which contains 3,700 training pairs. These pairs consist of 13 indoor and outdoor scenes, which are all captured by OPPO R17 Pro. Each training pair contains an initially filtered depth map and a GT depth map from the embedded ToF sensor. Finally, Adam policy is adopted to optimize the depth refinement network.

Fig. 5(d) shows the depth maps refined by our DNN-based refinement network, which contains less spatial depth noise. We will show in the following section that our DNN-based depth refinement network can also improve the final surface mesh reconstruction results. As further seen in Fig. 10(d-f) and Table 1, our depth refinement network following the multi-view SGM performs better than directly using end-to-end learning-based depth estimation algorithms like [11, 40] in generalization. We also compare the time cost of our scheme with other end-to-end networks. On OPPO R17 Pro, our monocular depth estimation takes 70.46 ms/frame, while MVDepthNet [40] takes 5.91 s/frame. DPSNet [11] is unable to run on OPPO R17 Pro or MI8 because of its complicated network structure. Therefore, applying DNN for depth refinement is a more economical way in time efficiency for a real-time mobile system with limited computing resources.

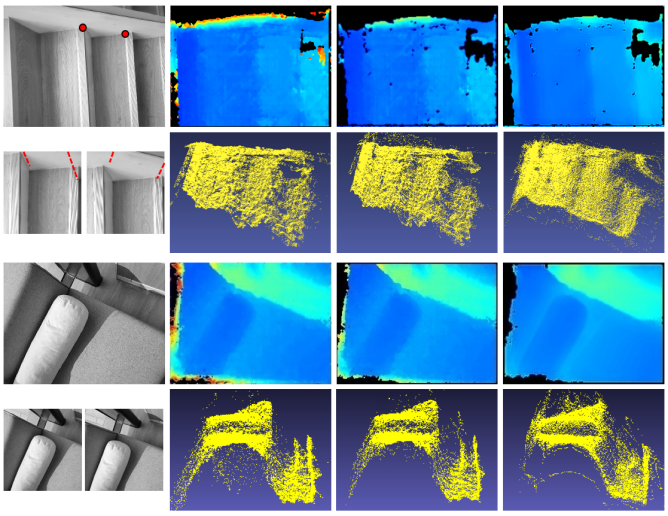

Fig. 5: Our monocular depth estimation results on two representative keyframes from sequences "Indoor stairs" and "Sofa": (a) The source keyframe image and its two selected reference keyframe images. Two representative pixels and their epipolar lines in reference frames of "Indoor stairs" are drawn out to demonstrate certain camera pose errors from 6DoF tracking on front end. (b) The depth estimation result of multi-view SGM and the corresponding point cloud by back-projection. (c) The result after confidence-based depth filtering and its corresponding point cloud. (d) The final depth estimation result after DNN-based refinement with its corresponding point cloud.

5 Incremental Mesh Generation

The estimated depths are then fused simultaneously to generate online surface mesh. Real-time surface mesh generation is required by AR applications for interactive graphics operations such as anchor placements, occlusions and collisions on mobile phones. Although TSDF is maintained and updated online in many real-time dense reconstruction systems such as [26, 42], mesh is however generated offline by these works. Some of the systems like [26] render or interact with the reconstructed model by directly raytracing TSDF on the GPU without meshing, which requires TSDF data to be stored always on GPU. However, with limited computing resources on mid-range mobile phones, dense reconstruction is usually required to run with only CPU on back-end, so as not to occupy resources of front-end modules or GPU rendering. In such cases, an explicit surface meshing should be done on CPU for the realistic AR operations like occlusions and collisions. To achieve an online incremental meshing on CPU, CHISEL [17] generates a polygon mesh for each chunk of the scene, and meshing is only done for the chunks that will participate in rendering. This kind of incremental meshing cannot guarantee consistency and non-redundancy of mesh topology, and is diffcult to handle real-time updating TSDF. In this section, we present a novel incremental mesh generation approach which can update surface mesh in real-time and is more suitable for AR applications. Our mesh generation performs a scalable TSDF voxel fusion to avoid voxel hashing conflicts, while incrementally updating the surface mesh according to the TSDF variation of the newly fused voxels. With this incremental mesh updating, real-time mesh expansion can be done on mobile phones using only a single CPU core. In the following, we will describe the incremental meshing approach in detail.

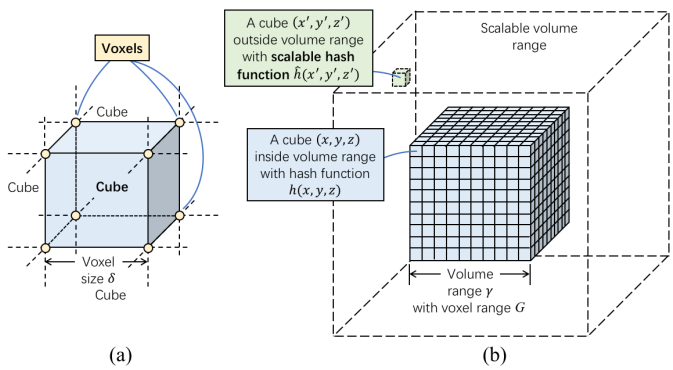

Fig. 6: (a) shows the structure of TSDF cubes, each of which consists of 8 voxels, and each voxel is shared by 6 neighboring cubes. (b) show the purpose of the new scalable hash function. A cube with integer coordinates $(x\ y\ z)$ located inside the volume can be indexed normally the conventional hash function $h(x,y,z)$. But for another cube $(x'\ y'\ z')$ outside the volume, function h will lead to a key conflict. Instead, using the newly proposed scalable hash function $\hat{h}$ will avoid conflict for both cubes.

5.1 Scalable Voxel Fusion

TSDF fusion has demonstrated its effectiveness in literature [1, 24]. Despite of the simplicity of the conventional TSDF, its poor scalability and memory consumption prevented its further applications, such as large scale reconstruction. Besides, the complex computation and large memory requirement made it difficult to achieve real-time performance on mobile phones, especially for outdoor environments. Voxel hashing [25] has been proved to be more scalable for large scale scenes because of its dynamical voxel allocation scheme and lower memory consumption, but it needs to deal with conflicts caused by hash function. Inspired by the above approaches, we propose a new scalable hashing scheme that combines both the simplicity of conventional TSDF voxel indexing and the scalability owned by voxel hashing. The basic idea is to dynamically enlarge the bound of volume whenever a

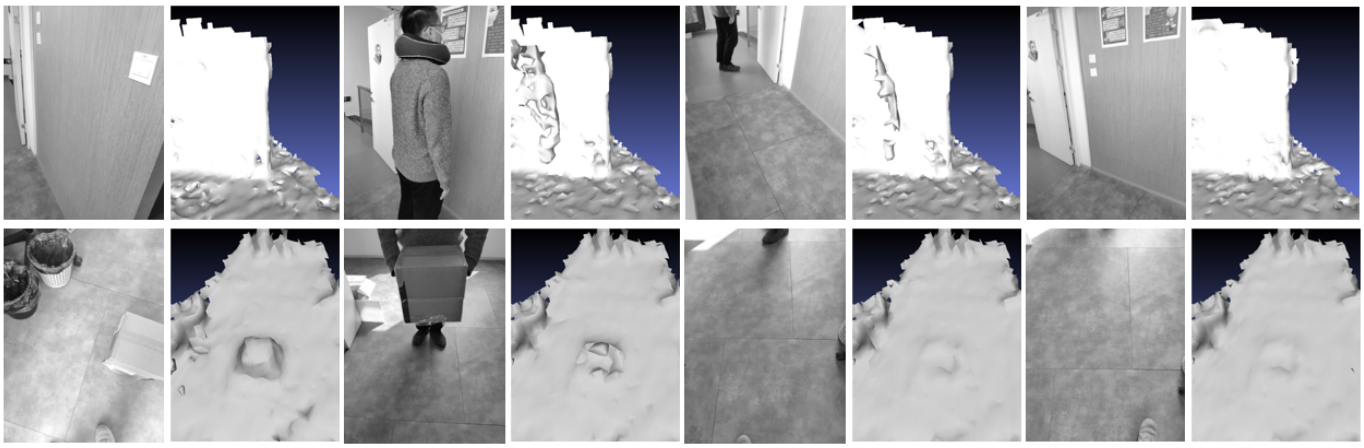

Fig. 7: Illustration of dynamic objects removal on two cases: the first row shows an object removed out of view, and the second row is a pedestrian who walks by, stands for a while and walks away. In both cases, our voxel fusion algorithm can gradually remove the unwanted dynamic object in the reconstructed mesh when it finally goes away.

3D point falls outside the pre-defined volume range. According to this scheme, we don't need to handle voxel hashing conflicts.

5.1.1 Scalable Hash Function

Marching cubes [21] extracts surface from a cube, each of which consists of 8 voxels from TSDF, as shown in Fig. 6(a). In our meshing process, each cube and its associated voxels can be indexed by a unique code generated with a novel scalable hash function.

Suppose we have a 3D volume with pre-defined size γ, each dimension of which has a range $[-\gamma/2, +\gamma/2)$. $G = \gamma/\delta$ is the corresponding volume size of each dimension in voxels, with δ an actual voxel size, such as 0.06 meter. The located cube of a 3D point $\mathbf{V} = (fx\ fy\ fz)$ inside the volume can be indexed by a hash function as follows:

$$h(x,y,z) = g(x) + g(y) * G + g(z) * G^2, \tag{18}$$

where $(x\ y\ z)$ is the lower integer coordinates of $\mathbf{V}$ divided by the voxel size δ, i.e., $(x\ y\ z) = (\lfloor fx/\delta \rfloor\ \lfloor fy/\delta \rfloor\ \lfloor fz/\delta \rfloor)$. $g(i) = i + G/2$, which converts $i \in [-G/2, +G/2)$ to range $[0, G)$.

With Eq. (18), we obtain a unique identifier for a 3D point located inside the pre-defined volume. However, a key conflict will happen when one point falls outside the volume. Suppose we have defined a volume with $G = 5$. A point $\mathbf{V}_a$ inside the volume with coordinates $(g(x)\ g(y)\ g(z)) = (1,1,0)$ will have the same identifier 6 as another point $\mathbf{V}_b$ with coordinates $(6,0,0)$ outside the volume according to Eq. (18), as depicted in Fig. 6(b). To also handle the case outside volume range for better scalability, we propose a new hash function by reformulating the hash function in Eq. (18) into the following form:

$$\begin{aligned} \hat{h}(x,y,z) &= O_C + \hat{g}(x) + \hat{g}(y) * G + \hat{g}(z) * G^2, \\ \hat{g}(i) &= i + GO_G/2, \end{aligned} \tag{19}$$

where O_G is a local offset for larger voxel index range in each dimension, and O_C is a global offset to ensure uniqueness of voxel indexing, which are defined as:

$$\begin{aligned} O_G &= \begin{cases} \lfloor \frac{2\hat{i}}{G} \rfloor + 1 & \hat{i} > 0 \\ \lfloor \frac{-(2\hat{i}+1)}{G} \rfloor + 1 & \hat{i} \le 0 \end{cases} \qquad \hat{i} = \underset{i \in \{x,y,z\}}{\arg\max} |i| \\ O_C &= G^3 \sum_{k=1}^{O_G - 1} k^3 \end{aligned} \tag{20}$$

With the new hash function of Eq. (19), $\mathbf{V}_b$ will have new coordinates $(\hat{g}(x)\ \hat{g}(y)\ \hat{g}(z)) = (11,5,5)$ and a new identifier 286 different from $\mathbf{V}_a$. We avoid voxel index conflict handling by enlarging the indexing range from $[0, G^3)$ to $[0, (GO_G)^3)$ for the case outside the volume, with the help of the local and global offsets. Therefore, unique identifiers are generated by the newly proposed hashing approach for two arbitary different points without confict, which ensures a more efficient way of voxel hashing than conventional hashing scheme. Besides, the reconstruction wouldn't be bounded to the predefined volume size G with the scalability provided by the proposed hash function. Using this scalable hashing scheme, we can expand the real-time reconstruction freely in 3D space without limitation caused by volume range.

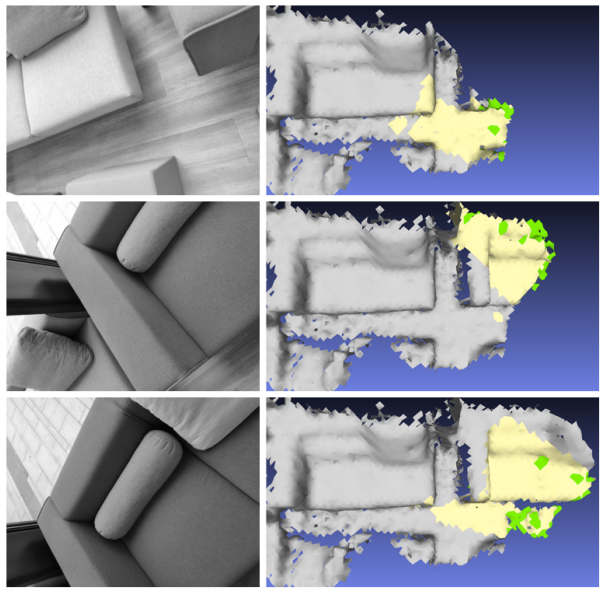

Fig. 8: Illustration of incremental mesh updating on three incoming keyframes. For each keyframe, the triangles colored with light yellow are updated by the current depth map, and the green color indicates the newly generated triangles.

5.1.2 Voxel Fusion with Dynamic Objects Removal

Following our scalable voxel hashing scheme, we integrate our estimated depth measurements into the TSDF voxels where their corresponding global 3D points occupy.

Suppose we have an estimated depth map D_t at time t. For each depth measurement $d \in D_t$ at pixel $\mathbf{x} = (u,v)$, we project it back to get a global 3D space point by $P = \mathbf{M}_t{}^{-1} \rho(u,v,d)$, where $\rho(u,v,d) = \left(\frac{u-c_u}{f_u}d, \frac{v-c_v}{f_v}d, d\right)$ is the back projection function, with (f_u, f_v) the

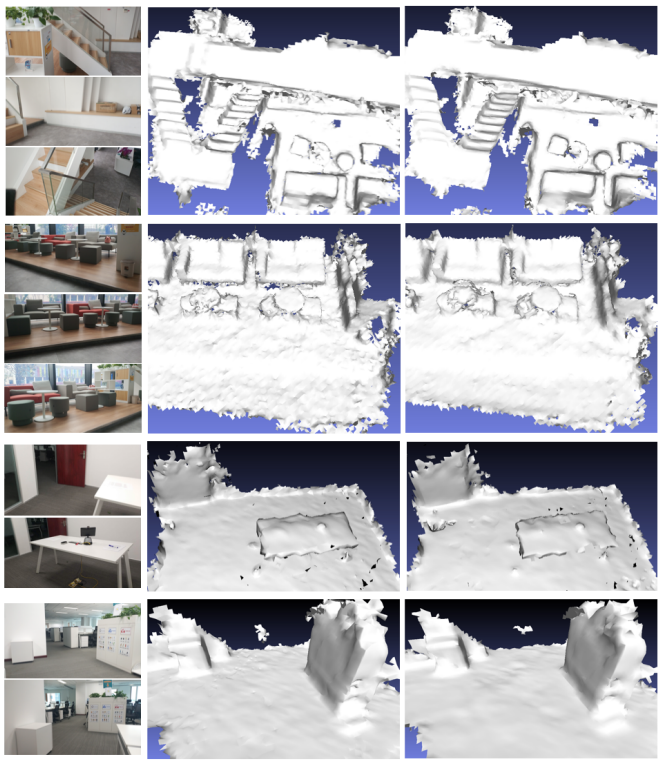

Fig. 9: Our surface mesh generation results of our four experimental sequences "Indoor stairs", "Sofa", "Desktop" and "Cabinet" captured by OPPO R17 Pro. (a) shows some representative keyframes of each sequence. (b) The generated global surface mesh of each sequence without DNN-based depth refinement. (c) Our generated global surface mesh with DNN-based depth refinement.

focal lengths in u and v directions, and (c_u, c_v) the optical center. $\mathbf{M}_t$ is the transformation matrix from global 3D space to local camera space at time t. The hash index of the cube occupied by P is determined by Eq. (19). As illustrated in Fig. 6, the occupied cube has eight associated voxels. Each voxel $\mathbf{V}$ would be a new one when travelled for the first time or be updated as follows:

$$\begin{aligned} T_t(\mathbf{V}) &= T_{t-1}(\mathbf{V}) + d(\mathbf{M}_t\mathbf{V}) - D_t(\pi(\mathbf{M}_t\mathbf{V})) \\ W_t(\mathbf{V}) &= W_{t-1}(\mathbf{V}) + 1 \end{aligned}, \tag{21}$$

where $\pi(x,y,z) = \left(\frac{x}{z}f_u + c_u, \frac{y}{z}f_v + c_v\right)$ is the projection function. For clarity, we rewrite $\pi(\mathbf{M}_t\mathbf{V})$ as $\mathbf{x}$. $d(\mathbf{M}_t\mathbf{V})$ represents the projection depth of $\mathbf{V}$ at local camera space of keyframe t. $D(\mathbf{x})$ is the depth measurement at pixel $\mathbf{x}$. $T_t(\mathbf{V})$ and $W_t(\mathbf{V})$ represent the TSDF value and weight of $\mathbf{V}$ respectively at time t. For a newly generated voxel, $T_t(\mathbf{V}) = d(\mathbf{M}_t\mathbf{V}) - D(\pi(\mathbf{M}_t\mathbf{V}))$ and $W_t(\mathbf{V}) = 1$.

With the TSDF voxel updating method in Eq. (21), we gradually generate or update all the voxels associated with the cubes occupied by the depth measurements from every incoming estimated depth map in real-time. Specifically, we maintain a cube list to record all the generated cubes. For each cube in the list, its associated TSDF voxel hash indices are also recorded, so that two neighboring cubes can share voxels with same hash indices. The isosurface is extracted from the cube list by Marching cubes algorithm [21]. Note that if any associated voxel of a cube is found to be projected outside the depth map border or onto an invalid depth pixel, all the TSDF voxel updates associated with this cube caused by the depth measurement need to be reverted. This rolling-back strategy effectively reduces the probability of broken triangles. Besides, a cube will be removed if the updated TSDF values are lower than zero for all of its 8 voxels, because no triangle should be extracted in that cube.

When users perform real-time reconstruction with some AR applications on mobile phones, there are usually dynamic objects such as walking pedestrians or moved objects, as shown in Fig. 7. These dynamic objects do not follow multi-view geometry prerequisites in temporal times. However, a more complicated case occurs when a pedestrian walks into the front of the camera, stands for a while, and then walks away, as illustrated in the first row of Fig. 7. Multi-view geometry is satisfied when the pedestrian stands still, so that the TSDF voxels are updated to implicitly contain the human body, which will be reflected in the reconstructed surface mesh. Works such as [20,39,42] are customized for static scenes, whose voxel fusion scheme cannot handle such complicated situation well. As far as we know, we are the first work to deal with the influence of dynamic objects on mobile phones with monocular camera.

In addition to the updating of the voxels associated with the current estimated depth measurements, we also project each existing voxel $\mathbf{V}$ to the current frame t for depth visibility checking. If $d(\mathbf{M}_t\mathbf{V}) < D_t(\mathbf{x})$, which means the depth measurement is behind the model, a visibility conflict occurs. It happens when an object enters into the camera view and stand still at some previous frames, and then disappears in the next frames. The voxels fused in the previous frames when the object is almost static, will have conflicts with the next frames when it goes away. Similar situations could also be caused by some broken triangle pieces generated with an inaccurate camera pose. We solve this situation by further updating the TSDF value of voxel $\mathbf{V}$ in the same way as Eq. (21). If $d(\mathbf{M}_t\mathbf{V}) \ll D_t(\mathbf{x})$, the TSDF value will drop quickly below zero, and its associated cubes are invalid. This strategy will make the cubes occupied by the dynamic human disappear quickly. With this visibility checking strategy, users can continue to scan the area occupied by the human body or the moved object after he goes away. The reconstructed foreground will gradually be removed and the background structure will come out soon, as can be seen from the changes of the reconstructed surface meshes shown in Fig.7.

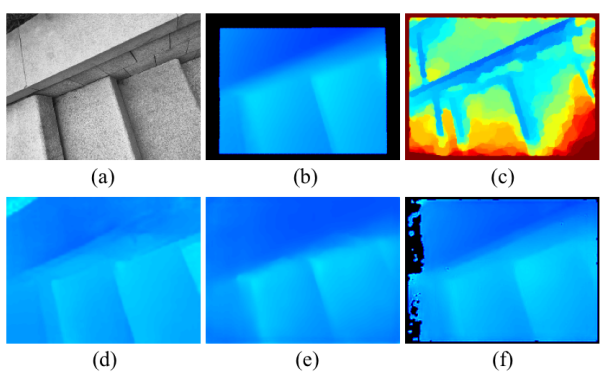

Fig. 10: Comparison of our monocular depth estimation with other state-of-the-art methods: (a) A representative keyframe in sequence "Outdoor stairs" by OPPO R17 Pro. (b) Ground truth ToF depth map by OPPO R17 Pro. (c) REMODE [27]. (d) DPSNet [11]. (e) MVDepthNet [40]. (f) Our Mobile3DRecon.

5.2 Incremental Mesh Updating

Marching cubes [21] is an effective algorithm to extract iso-surface from TSDF volume, which is widely used in dense reconstruction systems like [1,24–26]. However, most of these systems perform surface extraction as a post-process after real-time reconstruction has finished, due to its performance issue caused by frequent interpolation operations. Raycasting technology is employed in [24] to render isosurface for the current frame, but no mesh is actually extracted. Most AR applications require real-time mesh generation which supports incremental updating, especially on mobile phones. We propose an incremental mesh updating strategy, which is particularly suitable for real-time performance on mobile devices. Considering the surface extraction process is done on back-end, we run our incremental mesh generation on

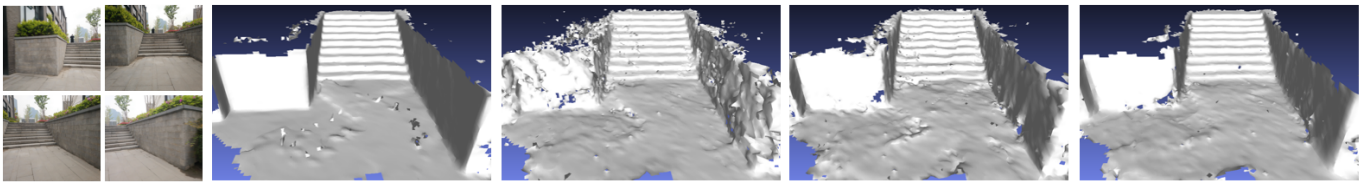

Fig. 11: Comparison of the finally fused surface meshes by fusing the estimated depth maps of our Mobile3DRecon and those of [11, 40] on sequence "Outdoor stairs" by OPPO R17 Pro. (a) Some representative keyframes. (b) Surface mesh generated by fusing ToF depth maps. (c) Mesh by DPSNet [11]. (d) Mesh by MVDepthNet [11]. (e) Our generated surface mesh.

a single CPU thead of the mobile phone, so as not to occupy resources of front-end modules or GPU rendering. The incremental updating strategy is based on the observation that only part of the cubes need to be updated for each keyframe. The iso-surface could be extracted only for these cubes.

In order to know which cubes should participate in surface extraction, a status variable $\chi(\mathbf{V}) \in \{ADD, UPDATE, NORMAL, DELETE\}$ is assigned to each voxel $\mathbf{V}$. If $\mathbf{V}$ is a newly allocated voxel, $\chi(\mathbf{V})$ is set to *ADD*. If the TSDF value $T_t(\mathbf{V})$ is updated for $\mathbf{V}$ at time t, $\chi(\mathbf{V})$ is set to $UPDATE$. If $T_t(\mathbf{V}) \leq 0$ or $W_t(\mathbf{V}) \leq 0$, $\chi(\mathbf{V})$ is set to $DELETE$, which means that $\mathbf{V}$ has to be deleted from the existing voxel list and moved to an empty voxel list for new voxel reallocation. We define a cube as updated if it has at least one associated voxel whose status is $UPDATE$ or *ADD*. After finishing TSDF voxel fusion and depth visibility handling mentioned in Section 5.1.2, we extract mesh triangles only from the updated cubes. If an updated cube already has triangles extracted, these triangles are removed and replaced with the newly extracted ones. After triangle extraction finishes for all the updated cubes, the status variables of the updated voxels are all set to $NORMAL$. If a cube has at least one associated voxel whose status is $DELETE$, the cube is considered as deleted, and its extracted triangles are removed. Fig. 8 illustrates the incremental mesh updating process.

Fig. 9 shows the surface mesh reconstruction results of our four sequences: "Indoor stairs", "Sofa", "Desktop" and "Cabinet". All these sequences are captured and processed in real-time with OPPO R17 Pro, which demonstrates the robustness of our real-time reconstruction system for large-scale and textureless scenes. We also compare the results with and without DNN-based depth refinement. As can be seen in Figs. 9 (b) and (c), our DNN-based depth refinement not only reduce depth noise, but can also eliminate noisy mesh triangles to significantly improve the final surface mesh quality.

6 Experimental Evaluation

In this section, we perform evaluation of our Mobile3DRecon pipeline, which is implemented in C++ code and uses third-party libraries OpenCV 2 for image I/O and Eigen 3 for numerical computation. We report quantitative comparisons as well as qualitative comparisons of our work with the state-of-the-art methods on our experimental benchmark by mid-range mobile phone, showing that our Mobile3DRecon is among the top performers on the benchmark. We also report the time consumption on each stage of our approach to show the real-time performance of our pipeline on mid-range mobile phone. Finally, we show how the method performs in AR applications on some mid-range mobile platforms.

6.1 Quantitative and Qualitative Evaluations

We qualitatively and quantitatively compare our monocular depth estimation approach to other state-of-the-art algorithms on the generated depth maps and surface meshes of the five sequences captured by OPPO R17 Pro. Since OPPO R17 Pro is equipped with a rear ToF sensor, the ToF depth measurements can be used as GT for the quantitative evaluation. In Fig. 10, we compare our estimated depth map against REMODE [27], DPSNet [11] and MVDepthNet [40]. We use the pretrained MVDepthNet model and DPSNet model to generate depth maps for comparisons. Both models are released on their github websites [3] [4], which were trained with Demon dataset [38]. Since DPSNet cannot be run on OPPO R17 Pro with limited computing resources due to its heavy network structure, we run it on a PC for comparison. As shown in Fig. 10, REMODE can ensure certain depth accuracy only in well-textured regions. Our depth estimation performs better in generalization than DPSNet and MVDepthNet, and produce depth measurements with more accurate details. We further give the comparison of the surface meshes fusing our depth maps and the depth maps estimated by other state-of-the-art methods [11, 40] in Fig. 11. Our system performs better than the other works in the finally generated surface structure with less noisy triangles. We can also see from the depth and mesh accuracy evaluation in Table 1 that our Mobile3DRecon reconstruct the scenes with a centimeter-level accuracy on depths and surface meshes, which are competitive in both Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE), even on the "Desktop" and "Cabinet" sequences with textureless regions.

Table 2 gives the time statistics of our method in stages of monocular depth estimation and incremental meshing seperately on two mid-range mobile platforms: OPPO R17 Pro with Qualcomm Snapdragon 710 (SDM710) computing chip and MI8 with Qualcomm Snapdragon 845 (SDM845), both with Android OS. All the time statistics are collected on the two mobile platforms in runtime with 6DoF tracking on front end and other modules such as global pose optimization, monocular keyframe depth estimation and incremental mesh generation on back end. Generally, our Mobile3DRecon performs almost 2 times faster on MI8 than on OPPO R17 Pro because of the more powerful SDM845 computing chip. Note that even with the slower performance on OPPO R17 Pro, our Mobile3DRecon can still achieve real-time, because our monocular depth estimation and incremental mesh generation steps are done for each new keyframe and the reported time consumption on OPPO R17 Pro is fast enought to keep up with the keyframe frequency, which is almost 5 keyframes-per-second in SenseAR SLAM framework.

6.2 AR Applications

With our Mobile3DRecon system integrated on Unity mobile platform, we can achieve real-time realistic AR effects such as occlusions and collisions on various scenes with different geometric structures on OPPO R17 Pro and MI8, which are illustrated in Fig. 12. Note that the example of the "Indoor stairs" shows the interesting effect of the virtual balls rolling down the stairs to verify the physically true interactions between virtual objects and real environment based on our accurate 3D reconstruction. Please refer to the supplementary materials for the complete demo videos. It is worth mentioning that a real-time surface mesh is crucial for simple implementation of occlusion and collision effects on most graphics engines like Unity, which is difficult to fulfill by other 3D representations such as surfels or TSDF volume.

[2] http://cloudcompare.org

[3] https://github.com/HKUST-Aerial-Robotics/MVDepthNet

[4] https://github.com/sunghoonim/DPSNet

Table 1: We report RMSEs and MAEs of the depth and surface mesh results by our Mobile3DRecon and [11, 27, 40] on our five experimental sequences captured by OPPO R17 Pro with ToF depth measurements as GT. For depth evaluation, only the pixels with valid depths in both GT and the estimated depth map will participate in error calculation. For common depth evaluation, only the pixels with common valid depths in all the methods and GT will participate in evaluation. Note that for REMODE, we only take into calculation those depths with errors smaller than 35 cm. For mesh evaluation, we use CloudCompare [2]to compare the mesh results by fusing depths of each method to the GT mesh by fusing ToF depths. For REMODE, we are unable to get a depth fusion result due to its severe depth errors.

Sequences	RMSE/MAE [cm]	REMODE [27]	DPSNet [11]	MVDepthNet [40]	Mobile3DRecon
Indoor stairs	Depth	23.38/18.95	12.48/7.71	10.54/7.82	**7.41/3.98**
	Common depth	24.11/19.15	9.78/6.30	9.25/7.43	**7.11/4.19**
	Mesh	/	6.34/8.67	6.04/8.98	**4.51/4.40**
Sofa	Depth	22.19/14.86	12.82/8.54	11.74/8.01	**9.66/6.10**
	Common depth	24.72/19.78	9.87/6.55	9.27/6.64	**9.19/5.59**
	Mesh	/	5.92/7.20	5.90/8.10	**5.31/5.74**
Outdoor stairs	Depth	19.39/12.91	9.09/6.42	7.46/5.06	**5.69/3.01**
	Common depth	24.57/19.57	7.88/6.03	6.81/5.10	**6.05/3.44**
	Mesh	/	6.22/8.10	5.23/5.10	**4.17/3.86**
Desktop	Depth	25.59/23.39	13.42/9.99	11.14/8.91	**9.45/5.42**
	Common Depth	25.55/23.43	12.51/9.36	10.46/8.49	**9.42/5.41**
	Mesh	/	5.93/9.88	5.76/9.65	**5.58/6.91**
Cabinet	Depth	19.22/16.48	11.89/8.57	**9.58**/7.12	10.43/**6.07**
	Common Depth	18.76/16.07	10.14/7.22	10.46/8.48	**9.54/5.92**
	Mesh	/	5.65/10.96	5.48/11.13	**5.02/7.96**

Table 2: We report detailed per-keyframe time comsumptions (in milliseconds) of our Mobile3DRecon in all the substeps. The time statistics are given on two mobile platforms: OPPO R17 Pro with SDM710 and MI8 with SDM845.

Time [ms/keyframe]	Monocular depth estimation					Incremental mesh generation	Total
	Cost volume computation	Cost volume aggregation	Confidence-based filtering	DNN-based refinement	Total		
OPPO R17 Pro (SDM710)	16.75	28.55	2.26	22.9	70.46	31.13	101.59
MI8 (SDM845)	11.92	17.68	1.1	7.62	38.32	18.89	57.21

Fig. 12: AR applications of Mobile3DRecon on mobile platforms: The first row shows the 3D reconstruction and an occlusion effect of an indoor scene on OPPO R17 Pro. The second and third rows illustrate AR occlusion and collision effects of another two scenes on MI8.

7 Conclusion

We have presented a novel real-time surface mesh reconstruction system which can run on a mid-range mobile phones. Our system allows users to reconstruct the dense surface mesh of the environments with a mobile device with only an embedded monocular camera. Unlike existing state-of-the-art methods which produce only surfels or TSDF volume in real-time, our Mobile3DRecon is unique in that it performs an online incremental mesh generation and is more suitable for achieving seamless AR effects such as occlusions and collisions between virtual objects and real scenes. Due to the limitation of TSDF integration, our dense surface mesh reconstruction is currently unable to keep the reconstructed mesh consistently updated with the changes of the global keyframe poses after bundle adjustment. An online deintegration and reintegration mechanism is preferred in future to make the incremental mesh generation more consistent with global pose optimization and accumulative error compensation. Additionally, how to reasonably handle the limitations of computation and memory resources on mobile platforms when the reconstruction scale becomes larger is a problem worth further studying in our future work.

Acknowledgments

The authors would like to thank Feng Pan and Li Zhou for their kind help in the development of the mobile reconstruction system and the experimental evaluation. This work was partially supported by NSF of China (Nos. 61672457 and 61822310), and the Fundamental Research Funds for the Central Universities (No. 2019XZZX004-09).

References

[1] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt. BundleFusion: Real-time globally consistent 3d reconstruction using on-the-fly surface re-integration. *ACM Transactions on Graphics*, 36(4):1, 2017.

[2] A. Drory, C. Haubold, S. Avidan, and F. A. Hamprecht. Semi-global matching: A principled derivation in terms of message passing. In *German Conference on Pattern Recognition*, pp. 43–53. Springer, 2014.

[3] V. Garro, G. Pintore, F. Ganovelli, E. Gobbetti, and R. Scopigno. Fast metric acquisition with mobile devices. In *Vision, Modeling and Visualization*, pp. 29–36, 2016.

[4] C. Godard, O. Mac Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 270–279, 2017.

[5] 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. *ArXiv Preprint ArXiv:1912.06378*, 2019.

[6] M. Heikkilä, M. Pietikäinen, and C. Schmid. Description of interest regions with local binary patterns. *Pattern Recognition*, 42(3):425–436, 2009.

[7] H. Hirschmüller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 807–814, 2005.

[8] H. Hirschmüller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):328–341, 2007.

[9] J. Hu, M. Ozay, Y. Zhang, and T. Okatani. Revisiting single image depth estimation: Toward higher resolution maps with accurate object boundaries. In *IEEE Winter Conference on Applications of Computer Vision*, pp. 1043–1051, 2019.

[10] P. Huang, K. Matzen, J. Kopf, N. Ahuja, and J. Huang. DeepMVS: Learning multi-view stereopsis. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2821–2830, 2018.

[11] S. Im, H.-G. Jeon, S. Lin, and I. S. Kweon. Dpsnet: End-to-end deep plane sweep stereo. In *International Conference on Learning Representations*, 2019.

[12] J. Jeon and S. Lee. Reconstruction-based pairwise depth dataset for depth image enhancement using CNN. In *European Conference on Computer Vision*, pp. 422–438, 2018.

[13] Y. Jiang, X. Gong, D. Liu, Y. Cheng, C. Fang, X. Shen, J. Yang, P. Zhou, and Z. Wang. Enlightengan: Deep light enhancement without paired supervision. *ArXiv Preprint ArXiv:1906.06972*, 2019.

[14] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pp. 694–711. Springer, 2016.

[15] O. Kahler, V. Prisacariu, C. Ren, X. Sun, P. Torr, and D. Murray. Very high frame rate volumetric integration of depth images on mobile devices. *IEEE Transactions on Visualization and Computer Graphics*, 21(11):1–1, 2015.

[16] M. Kazhdan. Poisson surface reconstruction. In *Eurographics Symposium on Geometry Processing*, 2006.

[17] M. Klingensmith, I. Dryanovski, S. S. Srinivasa, and J. Xiao. Chisel: Real time large scale 3D reconstruction onboard a mobile device using spatially hashed signed distance fields. In *Robotics: Science and Systems*, 2015.

[18] K. Kolev, P. Tanskanen, P. Speciale, and M. Pollefeys. Turning mobile phones into 3D scanners. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[19] P. Li, T. Qin, B. Hu, F. Zhu, and S. Shen. Monocular visual-inertial state estimation for mobile augmented reality. In *IEEE International Symposium on Mixed and Augmented Reality*, pp. 11–21, 2017.

[20] Y. Ling, K. Wang, and S. Shen. Probabilistic dense reconstruction from a moving camera. In *IEEE International Conference on Intelligent Robots and Systems*, pp. 6364–6371, 2018.

[21] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics*, 21(4):163–169, 1987.

[22] P. Merrell, A. Akbarzadeh, W. Liang, P. Mordohai, J. M. Frahm, R. Yang, D. Nister, and M. Pollefeys. Real-time visibility-based fusion of depth maps. In *IEEE International Conference on Computer Vision*, pp. 1–8, 2007.

[23] O. Muratov, Y. V. Slynko, V. V. Chernov, M. M. Lyubimtseva, A. Shamsuarov, and V. Bucha. 3DCapture: 3D reconstruction for a smartphone. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 893–900, 2016.

[24] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, and A. W. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *IEEE International Symposium on Mixed and Augmented Reality*, 2011.

[25] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3D reconstruction at scale using voxel hashing. *ACM Transactions on Graphics*, 32(6CD):169.1–169.11, 2013.

[26] P. Ondruska, P. Kohli, and S. Izadi. MobileFusion: Real-time volumetric surface reconstruction and dense tracking on mobile phones. *IEEE Transactions on Visualization and Computer Graphics*, 21(11):1–1.

[27] M. Pizzoli, C. Forster, and D. Scaramuzza. Remode: Probabilistic, monocular dense reconstruction in real time. In *IEEE International Conference on Robotics and Automation*, 2014.

[28] F. Poiesi, A. Locher, P. Chippendale, E. Nocerino, F. Remondino, and L. Van Gool. Cloud-based collaborative 3D reconstruction using smartphones. In *European Conference on Visual Media Production*, 2017.

[29] M. Pollefeys, D. Nister, J. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S. J. Kim, P. Merrell, et al. Detailed real-time urban 3D reconstruction from video. *International Journal of Computer Vision*, 78(2):143–167, 2008.

[30] V. Pradeep, C. Rhemann, S. Izadi, C. Zach, M. Bleyer, and S. Bathiche. MonoFusion: Real-time 3d reconstruction of small scenes with a single web camera. In *IEEE International Symposium on Mixed and Augmented Reality*, 2013.

[31] D. Scharstein, R. Szeliski, and R. Zabih. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1):7–42, 2001.

[32] T. Schöps, T. Sattler, C. Hne, and M. Pollefeys. Large-scale outdoor 3D reconstruction on a mobile device. *Computer Vision and Image Understanding*, 157:151–166, 2017.

[33] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2006.

[34] V. Sterzentsenko, L. Saroglou, A. Chatzitofis, S. Thermos, N. Zioulis, A. Doumanoglou, D. Zarpalas, and P. Daras. Self-supervised deep depth denoising. In *IEEE International Conference on Computer Vision*, pp. 1242–1251, 2019.

[35] C. Strecha, W. V. Hansen, L. V. Gool, P. Fua, and U. Thoennessen. On benchmarking camera calibration and multi-view stereo for high resolution imagery. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2008.

[36] P. Tanskanen, K. Kolev, L. Meier, F. Camposeco, O. Saurer, and M. Pollefeys. Live metric 3D reconstruction on mobile phones. In *IEEE International Conference on Computer Vision*, pp. 65–72, 2013.

[37] W. Thomas, F. S.-M. Renato, G. Ben, J. D. Andrew, and L. Stefan. ElasticFusion: Real-time dense slam and light source estimation. *The International Journal of Robotics Research*, 35(14):1697–1716, 2016.

[38] 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 *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5038–5047, 2017.

[39] J. Valentin, A. Kowdle, J. T. Barron, N. Wadhwa, M. Dzitsiuk, M. Schoenberg, V. Verma, A. Csaszar, E. Turner, I. Dryanovski, et al. Depth from motion for smartphone AR. *ACM Transactions on Graphics*, 37(6):1–19, 2018.

[40] K. Wang and S. Shen. MVDepthNet: Real-time multiview depth estimation neural network. In *IEEE International Conference on 3D Vision*, pp. 248–257, 2018.

[41] S. Yan, C. Wu, L. Wang, F. Xu, L. An, K. Guo, and Y. Liu. DDRNet: Depth map denoising and refinement for consumer depth cameras using cascaded CNNs. In *European Conference on Computer Vision*, pp. 151–167, 2018.

[42] Z. Yang, G. Fei, and S. Shen. Real-time monocular dense mapping on aerial robots using visual-inertial fusion. In *IEEE International Conference on Robotics and Automation*, 2017.

[43] Y. Yao, Z. Luo, S. Li, T. Fang, and L. Quan. MVSNet: Depth inference for unstructured multi-view stereo. In *European Conference on Computer Vision*, pp. 767–783, 2018.