VIDAR: Data Quality Improvement for Monocular 3D Reconstruction through In-situ Visual Interaction

Han Gao, Yating Liu, Fang Cao, Hao Wu, Fengyuan Xu* and Sheng Zhong

***Abstract*— 3D reconstruction based on monocular videos has attracted wide attention, and existing reconstruction methods usually work in a reconstruction-after-scanning manner. However, these methods suffer from insufficient data collection problems due to the lack of effective guidance for users during the scanning process, which affects reconstruction quality. We propose VIDAR, which visually guides users with the streaming incremental reconstructed mesh in data collection for monocular 3D reconstruction. We propose an incremental mesh extraction algorithm to achieve lossless fusion of streaming incremental mesh data via slice-style management for guidance quality. We also design an incremental mesh rendering algorithm to achieve precise memory reallocation by updating the buffer in a fill-in-the-blank pattern for guidance efficiency. Besides, we introduce several optimizations on data transmission and human-computer interaction to improve the overall system performance. The experiment results on real-world scenes show that VIDAR efficiently delivers high-quality visual guidance and outperforms the non-interactive data collection methods for scene reconstruction.**

I. INTRODUCTION

3D scene reconstruction is essential for perceiving natural environments in augmented/virtual/mixed reality [1]–[3]. With the breakthrough of SLAM technology [4]–[7], vision-based reconstruction technology has been widely developed due to its low cost and ease of use, which obtains the 3D mesh of a target scene with neural networks based on the monocular video collected by a smartphone.

As the proverb says, *Even the cleverest housewife can't cook without rice*. Such a "reconstruction-after-scanning" process lacks timely and reliable scanning guidance for users and therefore blocks users from collecting more complete data. It seriously harms the reconstruction quality and can not be solved only at the algorithmic level. As shown in Fig. 1, it causes three recurring problems because users are confused about determining whether the scanning data covers every region (corresponding to ❶), every view (corresponding to ❷), and every detail (corresponding to ❸). Even if users scan more times, there is not yet an automated technique to merge multiple meshes for better results.

Given that visual guidance is more convenient and intuitive for human users, it's well-suited to provide in-situ feedback on the scanning process, and the reconstructed mesh is undoubtedly one of the easiest to understand and use. As Fig. 1 shows, users are able to continuously see the reconstructed mesh on the screen with the streaming scene data. By comparing the differences between the mesh and the real world, this interactive scanning process can guide them to supplementally collect the missing scene data to improve reconstruction quality.

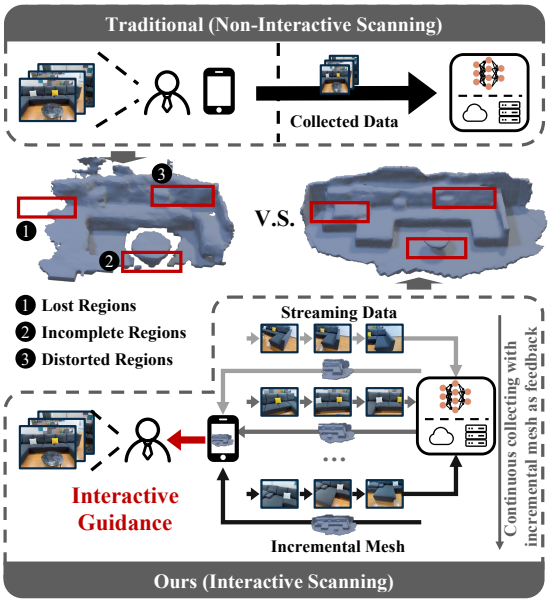


Fig. 1: **Comparison between traditional offline reconstruction process and our interactive process.** By constantly sending the mesh reconstructed on the back-end back to the mobile, users can collect more complete and detailed data based on this in-situ visual guidance, avoiding the three problems that easily occur in the traditional way.

However, two key challenges exist in this design. The *first* challenge is how to achieve precise multi-mesh fusion. A basic idea is to divide the input video into fragments and transfer only the reconstructed mesh of the current fragment each time. However, there are multiple meshes of an area in supplemental data collection. If each mesh is rendered, there will be a problem that the real mesh and the wrong historical mesh exist simultaneously, leading to low reconstruction quality. Therefore, a mesh fusion scheme is needed to ensure the interactively reconstructed results are high-quality and consistent with the real scene.

The *second* challenge is how to achieve efficient incremental rendering. SOTA scene reconstruction methods [8]–[10] use complex neural networks for 3D geometry prediction, which cannot be run directly on smartphones due to high computational resource requirements. We adopt the idea of offloading [11]–[13], which assigns the computation-intensive tasks (e.g., mesh generation) to edge servers with higher performance and the rendering tasks to the mobile

*Fengyuan Xu is the corresponding author.
National Key Lab for Novel Software Technology, Nanjing University, Nanjing 210023, China

side. However, the streaming mesh increases the GPU memory requirements over time and crashes the rendering. Thus, we need to design a mesh rendering strategy to optimize GPU memory management efficiency for smooth reconstruction.

To address these challenges, we propose VIDAR, which guides users to scan higher quality data in situ for monocular 3D scene reconstruction with visual feedback. It is a manner of reconstruction and rendering while scanning and differs from the traditional paradigm. It fundamentally solves the reconstruction quality problem of missing data due to the lack of guidance in data collection. There are two key technologies in VIDAR. *First*, an incremental mesh extraction algorithm. The proposed algorithm utilizes the characteristics of generating meshes in cubes to achieve *slice-style* management and incremental removal and modification of mesh data, ensuring lossless fusion for multiple historical meshes and the current mesh. *Second*, an incremental mesh rendering algorithm. The proposed algorithm takes advantage of the fixed generation pattern of faces in the mesh to perform *fill-in-the-blank* buffer updates, ensuring that the time to update the buffer is only related to the current mesh and reducing significant memory fragmentation.

We additionally introduce several optimizations for system performance, including a tolerance threshold for data transmission, render buffer backup to ensure data synchronization, and interactive prompts for better human-computer interaction. The experimental results on the ScanNet [14] dataset and real-world scenes show that compared to the baseline system, VIDAR enables precise management and lossless fusion to mesh and can reduce the total mesh data volume to 60% with a maximum of 7% quality loss in the case of limited bandwidth. VIDAR enables stable and timely updating of the rendering buffer, only taking 50ms per update on average. VIDAR can render a reconstructed mesh every second to provide instant feedback. To better showcase VIDAR, we have shared video demos on an anonymous website [1].

Our contributions are as follows:

- We seek to improve the 3D reconstruction results in practice from a perspective of data quality of the monocular video collection, which is complementary to algorithm-side optimizations. SOTA reconstruction methods so far cannot address quality issues created by data collectors.
- VIDAR achieves excellent monocular video collection by offering in-situ human-friendly quality feedback to data collectors using smartphones. Thus, a typical smartphone user is able to act as an expert-level collector and shoot videos of desired quality, which is critical when performing large-scale reconstructions in a crowdsourcing approach.
- VIDAR realizes such quality feedback via the visual scene mesh aligned with real-time video contents shown on the smartphone screen. Under the hood, it is supported by proposed new edge-assisted interactive reconstruction architecture, as well as two mobile optimizations to improve the quality and efficiency of visual mesh guidance on how to collect data.
- We implement VIDAR on smartphones with edge/cloud assistance and evaluate it on real-world scenes and users. Experimental results show that VIDAR is able to continuously update at 1Hz the holistic reconstructed model on the smartphone for newly-added video frames and display corresponding visual mesh indications at 20Hz on screen regarding to collectors' real-time location and direction.

[1] https://moss-3drecon.github.io/VIDAR/

II. Related Works

A. MVS-based Scene Reconstruction

The reconstruction of 3D geometry from 2D images is a classical problem in computer vision, and one of the most widely used methods is multi-view stereo (MVS).

MVSNet [15] is a representative learning-based MVS method to reconstruct 3D geometry, which builds a cost volume by traditional plane-sweeping [16] based on features mapped from multiple source images and a reference image and regularizes the cost volume by 3D CNN. DPSNet [17], and MVDepthNet [18] focus on how to build the cost volume. GPMVS [19] sends pairs of images into an encoder-decoder structure for depth estimation, which inspires DeepVideoMVS [9] to introduce ConvLSTM to incorporate more temporal information.

Atlas [20] is the first to directly regress Truncated Signal Distance Field(TSDF) from RGB images, which extracts features with 2D CNN, projects these features to 3D space, and predicts TSDF values by 3D CNN. NeuralRecon [8] predicts TSDF values with coarse-to-fine structure and introduces a 3D gated recurrent unit (GRU) to help fuse local features into global space instead of traditional TSDF fusion methods. TransformerFusion [21] and VoRTX [22] both introduce the Transformer [23] structure to fuse multi-view features. 3DVNet [24] and SimpleRecon [10] combine the advantages of these two types of methods.

Most of the methods mentioned above only run in real-time on desktop GPUs. If we directly deploy them on mobile platforms, the considerable overhead on GPU memory (up to 10+ GB) makes it impractical to view the mesh instantly.

B. Mobile Scene Reconstruction

Some early works on mobile scene reconstruction [25] [26] are implemented on a particular mobile phone prototype, such as Google's Project Tango Tablet. [27]–[29] estimate metric scale with inertial sensors on mobile phones to perform scene reconstruction. Mobile3DRecon [30] introduces a pipeline for real-time volumetric surface reconstruction directly on mobile devices, which utilizes a multi-view semi-global matching method to calculate the initial depth map and a CNN to optimize the depth map noise.

Due to factors such as computing resources, storage resources, and battery energy on the mobile side, mobile scene reconstruction methods sacrifice reconstruction quality and limit the scale of the reconstructed scene.

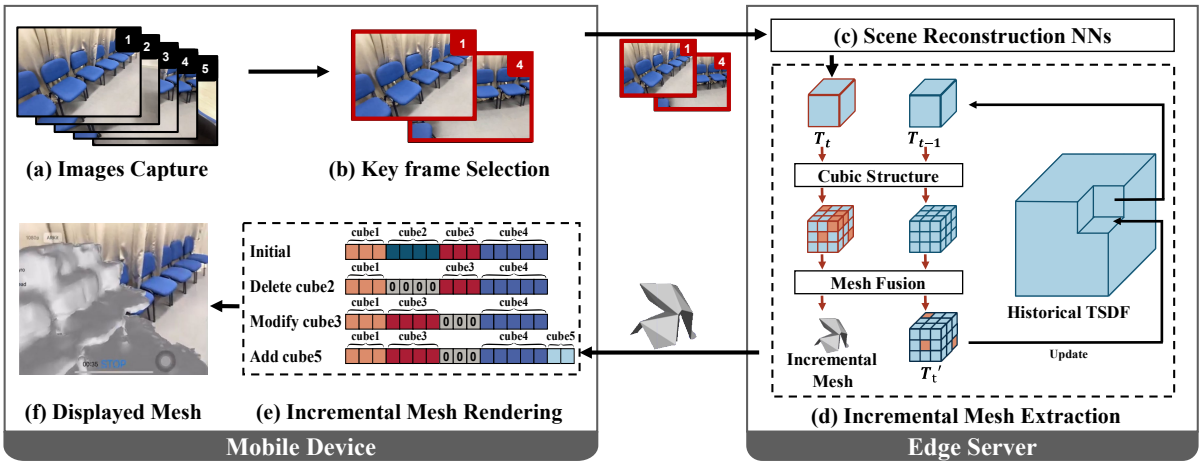


Fig. 2: **System pipeline.** Our motivation is that if users can see the reconstructed mesh all the time through the scanning process, they can spontaneously collect the missing data based on the differences between the reconstructed mesh and the real world. Therefore, we use the incremental reconstructed mesh as a visual interaction guide. The two key problems, like lossless fusion and efficient rendering, are solved by **step d** and **step e**.

III. Design

A. Preliminaries

TSDF is a popular 3D representation used in the field of 3D reconstruction [31], [32]. It divides the 3D space into uniform voxels, and each voxel's value is the distance to the nearest surface, which is truncated to -1 or 1 if it exceeds the distance threshold. Mesh is a classical representation to describe 3D space, which contains a set of vertices and triangular faces connected by these vertices. The Marching Cubes algorithm [33] is a classical way of extracting mesh from TSDF volume. The specific process is that for each voxel at (i,j,k) coordinates, a cube is formed with it and seven other neighbor voxels, placing it in the lower left corner. Based on eight voxels' values, An isosurface is formed with intersecting cube edges, and it is composed of one or multiple faces. The coordinates of these intersection points can be calculated with linear interpolation. Executing like this within each cube will yield the final mesh.

B. Overview

We leverage the rich GPU resources of the edge server to perform computationally intensive tasks. The monocular video captured on the mobile side is transferred to the server for inference via neural networks. The reconstructed mesh is extracted from the output TSDF volume and then transferred back to the mobile side for rendering. Fig. 2 illustrates VIDAR's workflow. VIDAR takes a sequence of RGB frames continuously scanned by the user as input (**step a**) and selects key frames according to the change of camera poses (**step b**). Every N key frames is fed into the scene reconstruction neural networks to predict the TSDF volume [2] (**step c**). Our proposed incremental mesh extraction algorithm manages the multiple historical TSDF volumes in a slice-style manner and obtains incremental updates on the basis of generated mesh (**step d**, discussed in Sec. III-C). Our proposed incremental mesh rendering algorithm utilizes a *fill-in-the-blank* policy to update the mobile GPU's rendering buffer in an incremental manner (**step e**, discussed in Sec. III-D). Finally, users can see the rendered mesh on the screen (**step f**).

[2]It can be any one of the MVS-based scene reconstruction methods, and We adopt NeuralRecon, the SOTA method, here.

C. Incremental Mesh Extraction

The important task for the server is to transfer high-quality mesh as fast as possible. One option is to transfer the whole mesh. Although the mesh quality is assured, the transmission time will increase drastically due to massive repeated data. The 3D mesh compression technique like [34] can not solve the problem fundamentally. Another option is to transfer the generated mesh and integrate it with historical meshes on the mobile side. While it avoids transmission of repeated data, a fusion fault problem occurs because the generated mesh may share the same spatial positions with the historical ones. The direct overlay will cause the rendered mesh to be rugged.

Considering the shortcomings of these two options, the key is to achieve efficient management and precise integration for these meshes. We note that although mesh data consists of unordered vertices and faces, TSDF data can be easily traversed because of the 3D space voxelization, and mesh is also generated by accessing the voxel values of TSDF volumes cube by cube. Thus, we can utilize the cubic structure to split the mesh into slices and locate any face in the mesh based on the coordinates in the TSDF volume. Benefiting from the efficient management scheme, we can identify precisely how the generated mesh has changed relative to the historical meshed, i.e., which vertices and which faces.

Specifically, for a TSDF volume $T \in \mathbb{R}^{H*W*L}$, we assign each cube $c \in T$ for two contributes, including the index c_i and the value c_v. We use the coordinates of the lower left corner on the cube to represent it, i.e. $c_i = (h,w,l)$, and use the eight voxel values of the cube to represent c_v. Based on this, we define incremental mesh M as:

$$M = \langle M_r, M_c \rangle$$

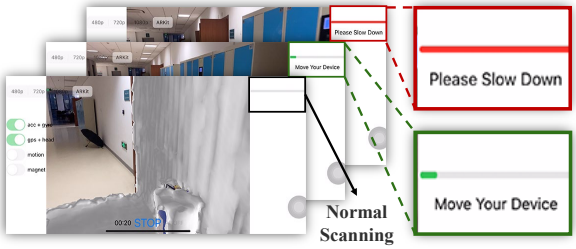


Fig. 3: **Runtime interface with a colored progress bar and text prompts.**

where M_r is the mesh whose vertices and faces are removed, and M_c is the mesh whose vertices and faces change. For M_r, we only need to record corresponding cube indices; for M_c, re-computed vertices and faces both need to be recorded, and the cude indices are also needed for follow-up convenience.

We assume the generated TSDF volume as T_t and retrieve T_{t-1} from the historical TSDF volume. T_t and T_{t-1} share the same spatial coordinates. The output of the fusion algorithm is T_t', which will be put back on the historical TSDF volume. Based on the cubic structure, we sequentially traverse these two TSDF volumes. The cubes at coordinate θ on these two TSDF volumes are noted as c_{t-1}^{θ} and c_t^{θ}. Given that the generated mesh is built from both previous features and new data, we believe that it has a better description of the target scene, so we prefer to trust the new TSDF data when conducting the incremental data. We use a straightforward approach to compare these two cubes, i.e., whether faces exist in this cube. We define the operation function $f(\theta)$ for the cube at coordinate θ as:

$$f(\theta) = \begin{cases} 0 & \texttt{(F,F)} \\ c_{i,t}^{\theta} & \texttt{(T,F)} \\ MC(c_{v,t}^{\theta}) & \texttt{(F,T)} \\ \mathbb{I}(c_{v,t-1}^{\theta} \neq c_{v,t}^{\theta}) * MC(c_{v,t}^{\theta}) & \texttt{(T,T)} \end{cases}$$

where `(F,F)` represents that there are no faces in c_{t-1}^{θ} and c_t^{θ}, and others are in a similar fashion. $MC(\cdot)$ represents the isosurface extraction in the Marching Cubes algorithm. $\mathbb{I}(\cdot)$ is the indicator function. Simply speaking, if there is no face in c_t^{θ}, record the index and add it to M_r. Otherwise, judge whether $c_{v,t-1}$ and $c_{v,t}$ are the same, and if they are different, compute the mesh for $c_{v,t}$.

D. Incremental Mesh Rendering

The crucial task for the mobile side is to achieve efficient and instant rendering for the streamingly received incremental mesh. The rendering process can be simplified to read mesh data from the rendering buffer, which is usually constructed as a linear list. Each element on the buffer stores the coordinates of three vertices in a face. In consideration of the fact that incremental mesh involves many removal and modification operations of historical data, a simple idea is that after new data arrives, the overall mesh is calculated and then fed into the buffer. It is obvious that the growth of the overall mesh will bring a huge processing time overhead.

The key point is how to achieve fast buffer updates and maintain the speed even after multiple operations. We believe that there is a more efficient way to update directly in the buffer rather than calculating from scratch each time. Our basic policy is to utilize the cube indices of M_r and M_c to locate them in the buffer, remove the corresponding historical faces for M_r and M_c (setting these positions as zeros), and append the faces belonging to M_c (add these positions to the tail). It does speed up the update process but brings much memory fragmentation after multiple updates due to massive zero bits, wasting lots of memory resources. We observe that there is a fixed number of faces when extracting the isosurface on each cube, which means that many cubes have the same number of faces. Thus, the optimized policy is to manage the memory in a *fill-in-the-blank* style, i.e., fill the zero bits when a same-sized modified process occurs.

Specifically, we define cube information as c_m. It includes the coordinate of each vertex in all faces v, the count of faces n, and the offset in the buffer δ, which means that faces from δ to $\delta+n$ in the buffer belong to this cube. We conduct a dictionary D mapping from c_i to c_m, which can help us quickly locate the historical mesh. We also build a dictionary L to manage the zeroed positions of the buffer. $L[i]$ records the available offsets whose count of faces is i.

We deal with the incremental mesh in two categories. For each cube on M_r, we can obtain c_m by accessing $D[c_i]$. The positions of the buffer from δ to $\delta+n$ will be zeroed out, and we add δ to $L[n]$ to make it can be reassigned. For each cube on M_c, we obtain c_m in the same way. If the count of faces in the new data is equal to n, we can directly modify v on-site. Otherwise, the operation is similar to M_r. If there is no usable offset in L, we will place the data at the tail.

E. Optimizations

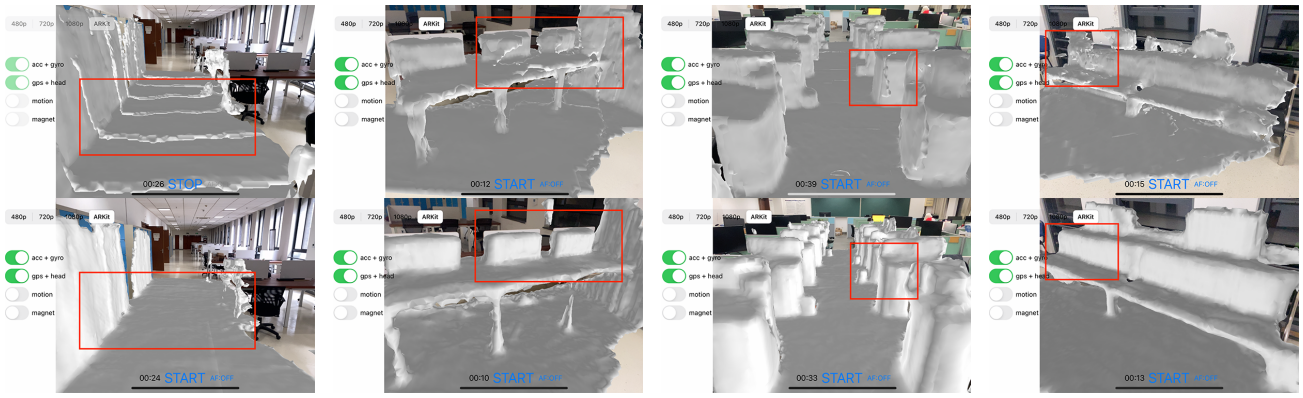
Tolerance Threshold. We introduce a parameter t called "tolerance threshold" to balance quality and latency with limited network bandwidth, which means reducing transferred data by tolerating the fusion quality decrease. Considering the shape of faces in a cube is related to eight voxel values, we observe that the variation of the extracted mesh is slight and hard to see when the change of these voxel values is small. If all voxel value variations between $c_{v,t-1}$ and $c_{v,t}$ are smaller than t, we thought that the changes are not significant and it is no need to re-compute the incremental data.

Rendering buffer Backup. We add an alternative buffer scheme to solve the data synchronization problem, which is caused by the mismatched speed of rendering and generation for incremental mesh. Suppose the GPU is using the data in b_A to render at present, and when a new mesh is received, we update the data in b_B. After finishing the update operation, the GPU render switches to b_B. The above process is performed again each time a new mesh arrives.

Interactive Prompts. Fig. 3 demonstrates our visual-based interaction scheme. The progress bar on the screen shows the proportion of frames collected that can perform an inference process. The user movement monitoring module reminds users whether the scanning speed is reasonable. The greater ratio of uploaded key frames and received mesh on the mobile side means too fast scanning speed. The text

TABLE I: **Quantitative results with the "Baseline(part)" system and VIDAR on ScanNet**. "Mean Data Compression Rate" means the ratio of transferred data with the tolerance threshold t to the same data not using the threshold, which is the average value for all scenes. t helps reduce the incremental data due to being insensitive to the change.

Method	t	3D geometry metrics					2D depth metrics					Mean Data
		Comp ↓	Acc ↓	Recall ↑	Prec ↑	F-score ↑	Abs Rel↓	Abs Diff↓	Sq Rel↓	RMSE↓	$\delta < 1.25$↑	Compression Ratio
baseline	-	0.003	0.044	0.946	0.255	0.398	0.085	0.156	0.058	0.263	0.867	-
ours	0.0	**0.000**	**0.000**	**1.000**	**1.000**	**1.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.992**	**1.000**
	0.1	0.003	0.002	0.931	0.941	0.936	0.004	0.006	0.001	0.032	0.990	0.587
	0.2	0.004	0.004	0.861	0.875	0.868	0.007	0.011	0.002	0.043	0.989	0.512
	0.4	0.005	0.005	0.797	0.813	0.805	0.010	0.016	0.002	0.052	0.987	0.473
	0.8	0.006	0.006	0.775	0.792	0.783	0.010	0.017	0.002	0.055	0.987	0.467



(a) "Corridor" (b) "Desk" (c) "Office" (d) "Bench"

Fig. 4: **Qualitative results on real-world scenes.** The first row is about the "Baseline (part)" system, which transfers the generated mesh and directly fuses it with historical meshes on the mobile side, not utilizing our lossless fusion scheme. Our results are on the second row.

"Please Slow Down" with the warning-colored bar reminds users to slow down the scanning speed. The greater time interval between the last and current key frames means too slow scanning speed, and the text "Move Your Device" will prompt them to collect more views.

IV. Experiments

A. Datasets and Metrics

We use the official test sequences of the ScanNet [14] dataset for the quantitative evaluation. Besides, we capture some real-world scenes to qualitatively demonstrate the performance of VIDAR. We use the 2D depth metrics defined in [35] and 3D geometry metrics defined in [20], which are used by most scene reconstruction methods. We use the original output 3D mesh of the chosen scene reconstruction method as ground truth to evaluate whether the 3D mesh rendered on the mobile side is consistent with the original mesh.

B. Quality Evaluation

We implemented the "Baseline(part)" system as a baseline to evaluate the guidance quality of VIDAR. It applies the conventional Marching Cubes algorithm to extract the generated mesh and then overlays it with the previous mesh data on the screen without using our mesh fusion scheme.

Fig. 4 shows rendered mesh with the "Baseline(part)" system and VIDAR in multiple real-world scenes. The "Baseline(part)" system has obvious grooves at the edges of the rendered mesh, while VIDAR renders a smoothing mesh.

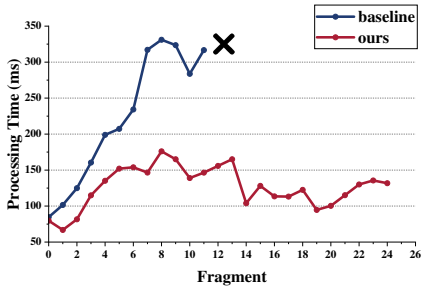


Fig. 5: **Comparison of processing time between the "Baseline(whole)" system and VIDAR for every fragment.** The input frames are divided into fragments and sequentially fed into the system.

In order to better illustrate the performance of VIDAR, we test our fusion scheme on the ScanNet dataset. Table I shows the comparison results. When t is 0.0, the mesh extracted by VIDAR is consistent with the GT mesh. VIDAR outperforms the "Baseline(part)" system in all metrics even when t is taken to 0.8. The "Baseline(part)" system significantly damages the F-score, which is a more comprehensive indicator of mesh quality. Besides, we find that when the value of t is 0.1, the transferred data volume can be reduced to 58.7% of the original data with 6% quality loss (from F-score). Setting a suitable value for t can help balance the quality and latency under limited network bandwidth.

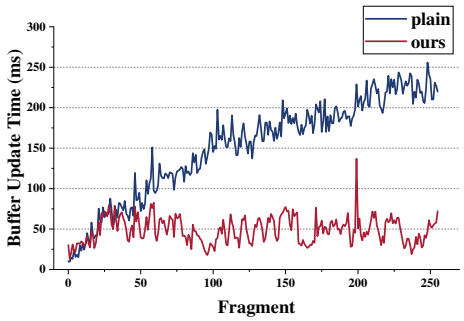
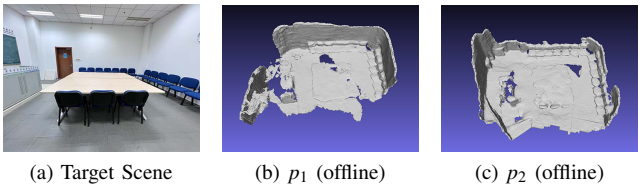
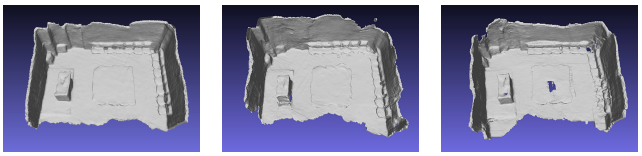


Fig. 6: **Comparison of buffer update time between "plain" algorithm and ours for every fragment.**



(a) Target Scene (b) p_1 (offline) (c) p_2 (offline)



(d) expert (VIDAR) (e) p_1 (VIDAR) (f) p_2 (VIDAR)

Fig. 7: **Qualitative performance when users scan with offline-scanning system and VIDAR.**

C. Efficiency Evaluation

To demonstrate the *data transfer efficiency* of VIDAR, we implemented the "Baseline(whole)" system as a baseline. It integrates the generated TSDF into the historical TSDF volume and extracts the global mesh with the Marching Cubes algorithm, then directly transfers and renders it on the mobile side, without using our incremental transfer scheme.

We tested the "Baseline(whole)" system and ours in a real-world scene to compare the processing time of each fragment. From Fig. 5, we can see that the processing speed of the "Baseline(whole)" system is getting slower and slower, even in the 11th fragment there was a system crash, and the scanning task eventually could not be completed. However, the speed of VIDAR was stable at 100-150ms and completed data collection normally.

To prove the *data rendering effectiveness* of VIDAR, we implemented the "plain" buffer update algorithm, which constructs the buffer from scratch each time a new mesh arrives. Comparative experiments between these two algorithms are conducted on a real-world scene. Fig. 6 shows that the time overhead of the "plain" algorithm shows an upward trend and grows to over 200 ms. However, our update algorithm remains stable within 100 ms (50 ms on average). The overall latency of our update algorithm remains smooth and low over time, which ensures VIDAR renders mesh in time.

Runtime. We tested VIDAR in a number of real-world scenes of varying size and type. Every 9 collected key frames by the smartphone are transferred to the server and then fed into the reconstruction neural networks to generate the incremental mesh. The statistics show that VIDAR can acquire 3.6 key frames each second. The mobile mesh updating time is the time taken from when it starts sending the key frames to when it receives the new incremental mesh, which is about 943ms (1hz). The mobile mesh rendering time is the time to update the rendering buffer, which is about 47ms (20hz). Therefore, VIDAR is able to meet the requirement of instantly displaying the mesh as one scans the scene. VIDAR's average upstream and downstream bit rates are 7.88 and 4.21 Mbps, respectively, which keep the network bandwidth overhead in an acceptable range.

D. Case Study

We invite an expert and two volunteers p_1 and p_2 to reconstruct a real-world scene separately, which demonstrates data collection quality intuitively. The expert has rich experience in 3D scene data collection, and these volunteers are not from the computer science field. The volunteers first reconstruct the scene through the offline scanning process and then reconstruct the same scene using VIDAR. The mesh reconstructed by the expert using VIDAR served as a comparison. For the fairness of the experiment, the key frames selection policy and the reconstruction neural networks used in the traditional method are consistent with what we used in VIDAR. Fig. 7 shows that the offline-scanning reconstruction results of both volunteers are incomplete or even broken. In contrast, the interactive reconstruction results are complete and seem similar to the mesh built by the expert. The results show that with VIDAR, even users without a technical background can reconstruct 3D scene mesh with higher quality more efficiently.

V. CONCLUSIONS

In conclusion, we propose VIDAR to address the data missing problem in situ during users' scanning process for monocular 3D reconstruction. VIDAR guides users to supplement more scene information with the streaming-generated mesh overlayed on real-world images. Our incremental mesh extraction algorithm enables lossless fusion between multiple historical mesh data and current generated mesh for high-quality guidance. Our incremental mesh rendering algorithm achieves instant buffer updates and efficient memory management for instant feedback. The introduction of optimizations ensures system performance and user experience under low bandwidth conditions. We believe VIDAR can improve the reconstruction quality of existing scene reconstruction algorithms in terms of data dimensions.

VI. ACKNOWLEDGEMENTS

This work was supported in part by the National Key R&D Program of China under Grants 2022YFF0604503, in part by NSFC under Grants 62272224, 62341201, 62302207 and 62272215, in part by the Leading Edge Technology Program of Jiangsu Natural Science Foundation under Grant BK20202001, and in part by the Science Foundation for Youths of Jiangsu Province under Grant BK20220772.

REFERENCES

[1] L. F. de Souza Cardoso, F. C. M. Q. Mariano, and E. R. Zorzal, "A survey of industrial augmented reality," *Computers & Industrial Engineering*, vol. 139, p. 106159, 2020.

[2] E. Dincelli and A. Yayla, "Immersive virtual reality in the age of the metaverse: A hybrid-narrative review based on the technology affordance perspective," *The Journal of Strategic Information Systems*, vol. 31, no. 2, p. 101717, 2022.

[3] M. Speicher, B. D. Hall, and M. Nebeling, "What is mixed reality?" in *Proceedings of the 2019 CHI conference on human factors in computing systems*, 2019, pp. 1–15.

[4] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[5] 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.

[6] Apple, "Arkit," https://developer.apple.com/augmented-reality/, 2017.

[7] Google, "Arcore," https://developers.google.com/ar, 2018.

[8] J. Sun, Y. Xie, L. Chen, X. Zhou, and H. Bao, "Neuralrecon: Real-time coherent 3d reconstruction from monocular video," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 15 598–15 607.

[9] A. Duzceker, S. Galliani, C. Vogel, P. Speciale, M. Dusmanu, and M. Pollefeys, "Deepvideomvs: Multi-view stereo on video with recurrent spatio-temporal fusion," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 15 324–15 333.

[10] M. Sayed, J. Gibson, J. Watson, V. Prisacariu, M. Firman, and C. Godard, "Simplerecon: 3d reconstruction without 3d convolutions," in *European Conference on Computer Vision (ECCV)*, 2022, pp. 1–19.

[11] J. Liu, J. Ren, Y. Zhang, X. Peng, Y. Zhang, and Y. Yang, "Efficient dependent task offloading for multiple applications in mec-cloud system," *IEEE Transactions on Mobile Computing*, vol. 22, no. 4, pp. 2147–2162, 2021.

[12] X. Pang, Z. Wang, J. Li, R. Zhou, J. Ren, and Z. Li, "Towards online privacy-preserving computation offloading in mobile edge computing," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 1179–1188.

[13] S. Yue, J. Ren, N. Qiao, Y. Zhang, H. Jiang, Y. Zhang, and Y. Yang, "Todg: Distributed task offloading with delay guarantees for edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 7, pp. 1650–1665, 2021.

[14] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5828–5839.

[15] 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 (ECCV)*, 2018, pp. 767–783.

[16] R. T. Collins, "A space-sweep approach to true multi-image matching," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 1996, pp. 358–363.

[17] S. Im, H.-G. Jeon, S. Lin, and I. S. Kweon, "Dpsnet: End-to-end deep plane sweep stereo," *arXiv preprint arXiv:1905.00538*, 2019.

[18] K. Wang and S. Shen, "Mvdepthnet: Real-time multiview depth estimation neural network," in *International Conference on 3D vision (3DV)*, 2018, pp. 248–257.

[19] Y. Hou, J. Kannala, and A. Solin, "Multi-view stereo by temporal nonparametric fusion," in *International Conference on Computer Vision (ICCV)*, 2019, pp. 2651–2660.

[20] Z. Murez, T. Van As, J. Bartolozzi, A. Sinha, V. Badrinarayanan, and A. Rabinovich, "Atlas: End-to-end 3d scene reconstruction from posed images," in *European Conference on Computer Vision (ECCV)*, 2020, pp. 414–431.

[21] A. Bozic, P. Palafox, J. Thies, A. Dai, and M. Nießner, "Transformerfusion: Monocular rgb scene reconstruction using transformers," *Conference on Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 1403–1414, 2021.

[22] N. Stier, A. Rich, P. Sen, and T. Höllerer, "Vortx: Volumetric 3d reconstruction with transformers for voxelwise view selection and fusion," in *International Conference on 3D Vision (3DV)*, 2021, pp. 320–330.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Conference on Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[24] A. Rich, N. Stier, P. Sen, and T. Höllerer, "3dvnet: Multi-view depth prediction and volumetric refinement," in *International Conference on 3D Vision (3DV)*, 2021, pp. 700–709.

[25] T. Schöps, T. Sattler, C. Häne, and M. Pollefeys, "3d modeling on the go: Interactive 3d reconstruction of large-scale scenes on mobile devices," in *International Conference on 3D Vision (3DV)*, 2015, pp. 291–299.

[26] 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*, vol. 4, no. 1, 2015.

[27] P. Tanskanen, K. Kolev, L. Meier, F. Camposeco, O. Saurer, and M. Pollefeys, "Live metric 3d reconstruction on mobile phones," in *International Conference on Computer Vision (ICCV)*, 2013, pp. 65–72.

[28] K. Kolev, P. Tanskanen, P. Speciale, and M. Pollefeys, "Turning mobile phones into 3d scanners," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 3946–3953.

[29] P. Ondrúška, P. Kohli, and S. Izadi, "Mobilefusion: Real-time volumetric surface reconstruction and dense tracking on mobile phones," *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, vol. 21, no. 11, pp. 1251–1258, 2015.

[30] X. Yang, L. Zhou, H. Jiang, Z. Tang, Y. Wang, H. Bao, and G. Zhang, "Mobile3drecon: real-time monocular 3d reconstruction on a mobile phone," *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, vol. 26, no. 12, pp. 3446–3456, 2020.

[31] 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 *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011, pp. 127–136.

[32] 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, pp. 24:1–24:18, 2017.

[33] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *ACM Special Interest Group on Computer Graphics (SIGGRAPH)*, vol. 21, no. 4, pp. 163–169, 1987.

[34] Google, "Draco 3d data compression," https://google.github.io/draco/, 2017.

[35] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," *Conference on Neural Information Processing Systems (NeurIPS)*, vol. 27, 2014.