# FlashFusion: Real-time Globally Consistent Dense 3D Reconstruction using CPU Computing

Lei Han[*†], Lu Fang[*]
[*]Tsinghua-Berkeley Shenzhen Institute
Shenzhen 518055, P. R. China
Email: fanglu@sz.tsinghua.edu.cn
[†]Hong Kong University of Science and Technology
Hong Kong
Email: lhanaf@connect.ust.hk

*Abstract*—**Aiming at the practical usage of dense 3D reconstruction on portable devices, we propose FlashFusion, a Fast LArge-Scale High-resolution (sub-centimeter level) 3D reconstruction system without the use of GPU computing. It enables globally-consistent localization through a robust yet fast global bundle adjustment scheme, and realizes spatial hashing based volumetric fusion running at $300$Hz and rendering at $25$Hz via highly efficient valid chunk selection and mesh extraction schemes. Extensive experiments on both real world and synthetic datasets demonstrate that FlashFusion succeeds to enable *real-time, globally consistent, high-resolution* ($5mm$)*, and large-scale* dense 3D reconstruction using highly-constrained computation, i.e.,** ***the CPU computing on portable device*****.**

## I. Introduction

Real-time 3D reconstruction, an attractive topic in both computer vision and robotics, has boosted immense practical applications on human-robot interaction, path planning [24], machine perception [16, 23], etc. Newcombe et al. [13] firstly exploits the possibility of high resolution dense 3D reconstruction using a commodity depth sensor [26] combined with Truncated Signed Distance Fields (TSDF) [1] for fusing multiple depth observations from different viewing angles. While KinectFusion [13] works within a limited volume size and requires a Graphics Processing Unit for computing, the high-quality of its reconstruction results and real-time performance demonstrate its potential for a wide range of applications. Latter on, various approaches are proposed addressing the scalability [15, 25], efficiency [11, 9, 17] and global consistency [2] of dense 3D reconstruction problems.

Recently, the state-of-the-art BundleFusion [2] shows superior performance to achieve high resolution and globally consistent real-time 3D reconstruction by utilizing the frame based localization for robustness, and reintegrate frames to correct surfaces on-line when previous poses are updated based on loop closure constraints and bundle adjustment optimizations. Nevertheless, it requires two high-end GPUs for computing. On the other extreme, the state-of-the-art CHISEL [11] merely relies on CPU computing for real-time 3D reconstruction on portable device. However, due to its localization drift and inefficient TSDF fusion, CHISEL can not enable global consistency, and its real-time implementation can merely support $20mm$ voxel resolution. Regardless the remarkable progress, the state-of-art remains an either-or solution: an efficient reconstruction runs on portable devices but fails in robustness and quality [11, 17], or a globally consistent high-quality system yet requires high-end GPUs for real-time performance [2], prohibiting its applications to more general cases like mobile robots or wearable devices.

In this paper, we propose **FlashFusion**, a **F**ast yet **LA**rge-**S**cale **H**igh-resolution (sub-centimeter level) dense 3D reconstruction system without the use of GPU computing. Essentially, FlashFusion contains a globally consistent yet fast localization module, as well as a highly efficient reconstruction module. For the former one, we adopt and further improve our FastGO scheme [7] to optimize the reprejection error of all the feature pairs and achieve globally consistent pose estimation. For the latter one, we investigate the fast implementation of TSDF fusion and mesh extraction using CPU computing. Technical contributions are summarized as follows.

- We improve the robustness of localization by using Huber-norm during pre-integration stage, and further solving it via an on-line fast correction method by exploiting the inherent nature that Huber-norm can be interpreted as an iteratively re-weighted least squares problem, guaranteeing the robust yet efficient globally consistent localization.
- We speed up the TSDF fusion to realize 300Hz implementation by proposing sparse voxel sampling to directly locate the valid voxels that fall into the truncate range of observed surfaces, instead of processing all the voxels that fall into the frustum of camera views. Given that surfaces are continuous and nearby voxels have correlative sdf values, the sparse voxel sampling merely requires the sdf values of 8 corners in each chunk (composing $8 \times 8 \times 8$ voxels) to determine whether it contains valid voxel whose sdf value is smaller than the truncate range. Such chunk filtering step is further shared within keyframes to avoid replicative computations.
- We accelerate the mesh extraction to achieve 25Hz implementation via three stages: 1) an adaptive threshold to identify voxels that may relate to surfaces for each chunk instead of exhaustive estimation. 2) minimizing

the computation of polygon generation by utilizing the feature that only 1 degree of freedom exists for each generated vertex. 3) an efficient look-up-table for each chunk to maintain the address of its neighboring chunks merely, as only the neighboring chunks are accessed in each inter-chunk voxel accessing, assuring efficient voxel accessing even if the chunks are represented sparsely.

Given the above technical contributions, we demonstrate **FlashFusion, the first CPU-based globally consistent real-time 3D reconstruction system on portable devices**. The live demos are available on the project website www.luvision.net/FlashFusion.

## II. Related Work

RGBD based 3D reconstruction have demonstrated their robustness, high-quality and scalability with the development of both powerful processing units like GPU and depth sensors such as Kinect. Generally, the frameworks can be categorized into Voxel or Surfel-based methods, where the former one divides the space into equally distributed 3D grids (voxels) and extract surface from the TSDF field of voxels, and the latter one employs dense surfels to represent the surface and assigns each surfel a range and normal.

Consider the relatively higher universality of voxel-based techniques to fit in with various applications including path planning [24], machine perception [23] and dynamic reconstruction [14], we mainly focus on voxel-based methods in this paper. In this line, Newcombe et al. [13] pioneered the framework of using Truncated Signed Distance Fields (TSDF) to fuse depth observations from a moving RGBD camera with real-time performance on a GPU. Based on it, the extensions in different ways are proposed as follows.

**Scalability**: Nießner et al. [15], Zeng et al. [25] and Whelan et al. [21] tried various technique to make the TSDF-based fusion methods be scalable to large environments, following the similar insight that surfaces are sparse compared with space. More concretely, [21] uses a moving volume to extend the covered area of the fixed TSDF volumes in [13], [15] handles this sparsity using hashing techniques to represent the sparse voxels around the surface, and [25] exploits the Octree data structure to represent a large space volume, where only voxels around surface are represented using a small resolution and empty voxels are represented using a large resolution.

**Global Consistency**: Such feature is of great importance since drift introduced by camera or inertial sensors are inevitable and little camera pose error may introduce significant artifacts in the reconstructed model. [2] investigates it using two high-end GPUs, one for globally consistent localization based on frame registration against all previous frames, the other for TSDF integration and reintegration when loop closures are detected and previous camera poses are updated to correct previously incorrect observations. Considering that [2] solves inconsistency in TSDF fields at the cost of heavy computational demands, [12] addresses it by only reintegrating keyframes instead of all previous frames. Meanwhile, [10] proposes to divide the scenes into submaps and adjust the poses of submaps to maintain global consistency. Compared with keyframe based methods, where the number of keyframes grows with time, the number of submaps in [10] grows only with the size of reconstructed map, yet each submap requires several hundred Megabytes for storage.

**Efficiency**: [9] proposes to optimize the basic data structure in [15] and recasting pipeline, so that the reconstruction pipeline is able to work in real-time on portable GPU devices. To achieve global consistency, a high-end GPU is still necessary as demonstrated in [10]. On the other hand, CPU based dense 3D reconstruction is quite attractive for applications including mobile robots and wearable devices.[17, 11] try to fuse depth observations at frame-rate and output meshes at a backend thread. [17] exploits the benefits of octree data structure while [11] is proposed based on the framework of [15]. Both benefits from the sparse representation of surfaces where less voxels are required to process. However, even with extensive optimizations in multi-threads and SIMD instructions, Klingensmith et al. [11] can only work at a voxel resolution of 2cm for indoor environments while Steinbrücker et al. [17] only handles voxels at a resolution of 5mm for limited areas.

This paper aims at the practical use of 3D reconstruction on portable devices by presenting FlashFusion, a Fast LArge-Scale High-resolution (sub-centimeter level) 3D reconstruction system without the use of GPU computing. The scalability, global consistency and high-resolution characteristics of FlashFusion benefit from the joint realization of robust globally consistent localization and highly efficient TSDF Fusion.

## III. System Architecture

The system architecture of FlashFusion is depicted in Fig. 1, which contains the front-end tracking thread, and the back-end optimization thread and meshing thread. The tracking thread takes raw RGBD data as input, and the drift can be eliminated through a high-precision yet high-efficient loop closure detector: MILD [6], which simply uses ORB features and does not rely on any training process. The back-end threads that relate to the localization and reconstruction are implemented as follows.

Localization – Considering the high demand of efficiency, the frames are divided into keyframe and local frame based on the disparity criteria, i.e., new keyframe is inserted when the average disparity of corresponding features between current frame and its corresponding keyframe is larger than a threshold. Only the pose of keyframes are updated in the global optimization procedure, while the relative transformation between local frame and its preceding keyframe is fixed and the pose of local frame is updated based on the optimized keyframe.

Reconstruction – Resembling [15] and [11], the world is represented as a two level tree, where a cubic of certain voxels are grouped as a chunk and each chunk is spatially-hashed [19] into a dynamic hash map. Given the organization of keyframe and local frame, the reconstruction can be divided into TSDF fusion and mesh extraction. Precisely, TSDF field is updated based on the depth observations for each incoming frame, and all the depth observations from both local frames

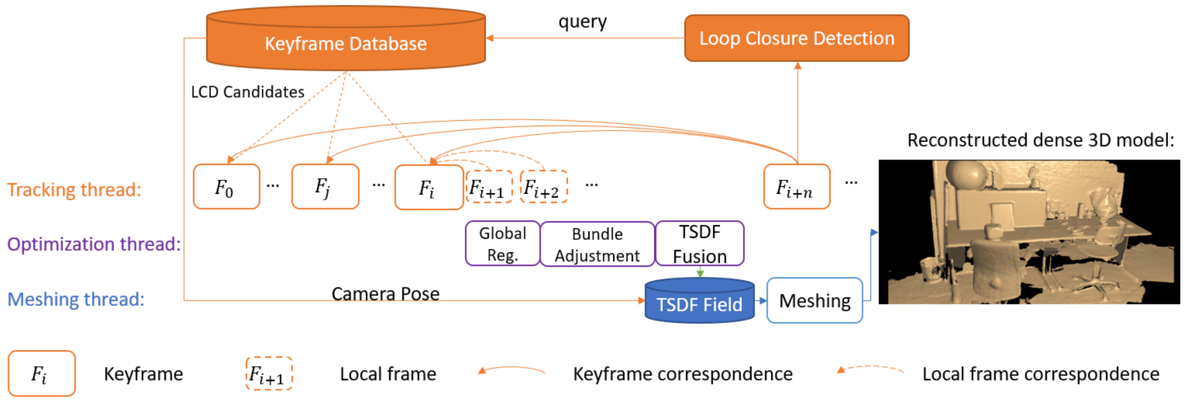


Fig. 1. Framework of FlashFusion.

and keyframes are fused into the TSDF field to ensure the quality of the reconstructed 3D model. The update of meshes, however, is activated only when a new keyframe is inserted. Motivation behind such strategy is that the keyframe and its succedent local frames share a large overlap and there is no need to update meshes every time a local frame is fused in.

## IV. Real-time Globally Consistent Dense 3D Reconstruction using CPU Computing

To elaborate how FlashFusion can enable the global consistency and high efficiency simultenously, this section is organized as follows: a robust localization framework that based on our FastGO scheme [7] is briefly introduced in Sec. IV-A, then Sec. IV-B and Sec. IV-C1 explain the proposed TSDF fusion and mesh extraction methods, finally the reintegration implementation is shown in Sec. IV-D.

### *A. Robust Globally Consistent Localization*

As usual, the frame registration that widely used in pose estimation is discriminated as locally and globally. The local registration indicates frame registration between current frame and its previous keyframe. If the keyframe update criteria meets, current local frame is recognized as a new keyframe and matched against all the previous keyframes, which is global registration. Here the frame registration works via finding the corresponding ORB features (around 1000 ORB features are extracted for each frame) between two frames, saying $F_i, F_j$, and the relative transformation is estimated by minimizing the following cost function on manifold using Lie group [20],

$$E_{i,j}(T_{i,j}|T_{i,j} \in SE3) = \sum_{k=0}^{|C_{i,j}|-1} ||\boldsymbol{p_i^k} - T_{i,j}\boldsymbol{p_j^k}||^2 \qquad (1)$$

where $C_{i,j} = \{(\boldsymbol{p_i^k}, \boldsymbol{p_j^k})|k = 0, 1, \cdots, |C_{i,j}| - 1\}, (\boldsymbol{p_i^k}, \boldsymbol{p_j^k})$ indicates the $k$-th feature correspondence between frame $F_i$ and $F_j$, $\boldsymbol{p_i^k} \in F_i, \boldsymbol{p_j^k} \in F_j$. $T_{i,j}$ is the relative transformation in Euclidean space from $F_j$ to $F_i$.

Given the pair-wise frame correspondence, we minimize the reprojection error of all corresponding keyframe pairs from global registration, so as to achieve a globally consistent pose estimation,

$$E(\boldsymbol{\xi}) = \sum_{i=0}^{N-1} \sum_{j\in\Phi(i)} E_{i,j} = \sum_{i=0}^{N-1} \sum_{j\in\Phi(i)} \sum_{k=0}^{|C_{i,j}|-1} ||\boldsymbol{p_i^k} - T_{i,j}\boldsymbol{p_j^k}||^2, \qquad (2)$$

where $\Phi(i)$ represents the top 5 most similar images that selected via a super-fast yet high-accurate loop closure detection technique MILD [6], as a traverse search [2] is impractical for CPU computing.

Suppose there are $M$ feature correspondences in $E(\boldsymbol{\xi})$, we have:

$$E(\boldsymbol{\xi}) = \boldsymbol{r}(\boldsymbol{\xi})^T \boldsymbol{r}(\boldsymbol{\xi}), \qquad (3)$$

where $r(\boldsymbol{\xi}) = \left[\boldsymbol{r_0^T}, \boldsymbol{r_1^T}, \cdots, \boldsymbol{r_{M-1}^T}\right]$, and $\boldsymbol{r_l} = \boldsymbol{p_i^k} - T_{i,j}\boldsymbol{p_j^k}$ represents the re-projection vector of the $l$-th feature correspondence among the $M$ features. The non-linear Gauss-Newton optimizations are used to solve this problem on Lie manifold in $SE3$ space iteratively:

$$\boldsymbol{\delta} = -(J(\boldsymbol{\xi})^T J(\boldsymbol{\xi}))^{-1} J(\boldsymbol{\xi})^T \boldsymbol{r}(\boldsymbol{\xi}), \qquad (4)$$

where $J(\boldsymbol{\xi})$, of size $3M \times 6(N-1)$, is the Jacobian of $\boldsymbol{r}(\boldsymbol{\xi})$. Instead of computing $J(\boldsymbol{\xi})$ directly, $J(\boldsymbol{\xi})^T J(\boldsymbol{\xi})$ and $J(\boldsymbol{\xi})^T \boldsymbol{r}(\boldsymbol{\xi})$ are computed directly for efficiency since they are more compact:

$$\begin{aligned} J(\boldsymbol{\xi})^T J(\boldsymbol{\xi}) &= \sum_{i=0}^{N-1} \sum_{j\in\Phi(i)} \sum_{k=0}^{|C_{i,j}|-1} (J_{i,j}^k)^T J_{i,j}^k, \\ J(\boldsymbol{\xi})^T \boldsymbol{r}(\boldsymbol{\xi}) &= \sum_{i=0}^{N-1} \sum_{j\in\Phi(i)} \sum_{k=0}^{|C_{i,j}|-1} (J_{i,j}^k)^T \boldsymbol{r_l}, \end{aligned} \qquad (5)$$

where $J_{i,j}^l$ is the Jacobian of $\boldsymbol{r_l}$. As noted in FastGO [7] that $\sum_{k=0}^{|C_{i,j}|-1} (J_{i,j}^k)^T J_{i,j}^k$ and $\sum_{k=0}^{|C_{i,j}|-1} (J_{i,j}^k)^T \boldsymbol{r_l}$ in Eqn. 2 can be efficiently computed with the complexity of $O(1)$ instead of $O(|C_{i,j}|)$, by pre-integrating the second order statistics of feature correspondences from $C_{i,j}$, enabling Eqn. 2 to be solved in real-time using CPU computing.

While FastGO technique leads to superior performance on efficiency, it appears brittle in the presence of outliers. We thus use more robust Huber norm instead of $l_2$ norm in the cost function, which however, requires the weight of each feature pair to be updated based on the latest pose estimations, implying that it cannot be represented using the second-order statistics as implemented in FastGO. A traverse search of all the feature pairs in Eqn. 2 after each Gauss-Newton update would take extensive computations. To solve Huber-norm based minimization in real-time, an on-line correction scheme is adopted by choosing only the top 10 frame pairs whose relative poses change most significantly in all the frame-pairs collected. The insight of such correction is that the re-projection error of features depends on the relative poses of frame-pairs while only a limited number of frame-pairs' relative poses are updated after loop closure detection and global bundle adjustment in practice.

### *B. Efficient TSDF Fusion*

*1) Review of TSDF Representation:* Recall that surface can be represented implicitly using TSDF, where each 3D point in space is mapped to a distance (denoted as Signed Distance Function (SDF)) to its nearest surface, and the SDF values of points on a surface are close to 0. Continuous space is digitalized as voxels, where each voxel stores the SDF of its center point. To efficiently index valid voxels around surfaces, Hashing [15] and Octree [25] based data structures are proposed employing the sparsity of valid voxels. In voxel-hashing based reconstruction systems, $8 \times 8 \times 8$ voxels are organized as chunks. Each chunk is mapped to an address based on the spatial hash function [19]. Depth observations are integrated chunk-wisely using a projective mapping manner: for each voxel, its voxel center is projected onto the 2.5D depth map, the difference between projective distance and depth map reading is recognized as surface distance $d$ and fused into the TSDF field. Since depth readings have different covariance due to the sensor model, observations are fused with different weight $w$ which is determined directly by depth readings:

$$sdf_n = \frac{sdf_o * w_o + sdf_i * w_i}{w_i + w_o}, \quad w_n = w_i + w_o, \tag{6}$$

where $sdf_o$ and $w_o$ indicate the original sdf and weight value, $sdf_i$ and $w_i$ are the incoming observations of sdf and weight, $sdf_n$ and $w_n$ are the newly-updated sdf and weight.

*2) Valid Chunk Selection:* Previously, all the chunks fall into the frustum of the camera view are selected as candidate chunks, which are processed equally during fusion [11]. Then the invalid chunks (the blue grids in Fig. 2) whose voxels have no valid sdf are removed, retaining the valid chunks (the yellow grids in Fig. 2) that are within the truncated band of a surface. Apparently, the valid chunks account for a little portion of the candidate chunks, while most computational resources are wasted on the invalid chunks.

To identify valid chunks efficiently, we propose a sparse voxel sampling scheme by collecting the eight corners of a candidate chunk and checking if the minimum absolute sdf

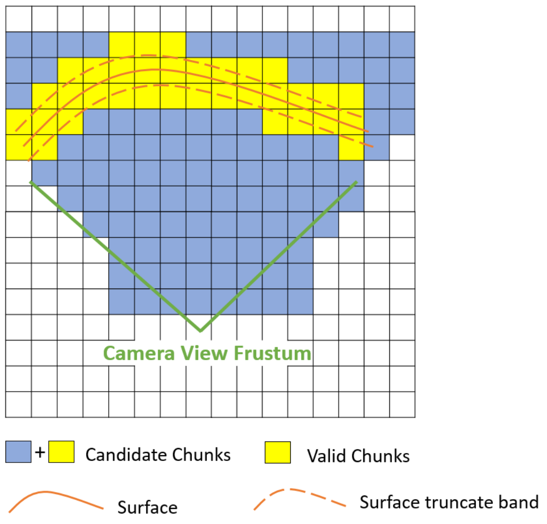


Fig. 2. Illustration of chunk selection, where FlashFusion deals with effective chunks (yellow grids) merely, while CHISEL processes all the candidate chunks (both the blue and yellow grids).

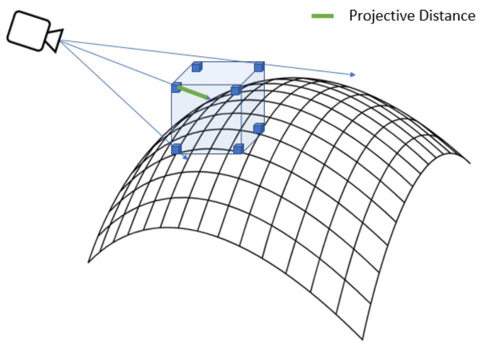


Fig. 3. Illustration of sparse voxel sampling for chunk filtering, which calculates the sdf values of eight corners in the chunk merely.

is larger than a threshold. Insight behind is that surfaces are continuous in general, i.e., if a chunk contains surface inside, there exists a high probability that surfaces may pass at least one of the six faces of this chunk, and at least one of the eight corners can be projected on the surface, as illustrated in Fig. 3. For example, suppose the voxel resolution is $r$ and there are $N_v^3$ voxels in each chunk, if there is one voxel that falls in the truncation of a surface, the sdf value $d_v$ of that voxel should satisfy: $|d_v| < T_{truncation}$. Accordingly, the sdf of a corner voxel $d_c$ would satisfy: $|d_c| \leq |d_v| + \sqrt{2}N_v \times r < T_{truncation} + \sqrt{2}N_v \times r$. On the contrary, if all the corner's sdf values are smaller than $T_{truncation} + \sqrt{2}N_v \times r$, it is improbable that the chunk contains surface inside. While the eight corners can be calculated parallelly using SIMD operations, the computation would be still extensive when the voxel size is smaller than 1cm. Thus, the chunks are firstly examined at a resolution of $20mm$ using sparse voxel sampling, if detected sdf values demonstrate that there may be surface within this large chunk,

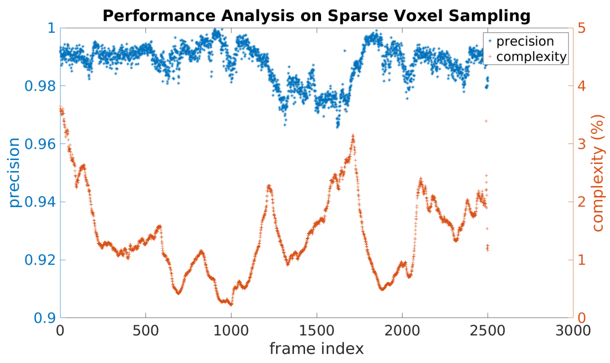


Fig. 4. Performance analysis of our sparse voxel sampling by comparing with conventional candidate chunk selection strategy on fr3office dataset.

we traverse the chunks at a resolution of $5mm$ inside the large chunk to find the truly valid chunks.

In this way, the currently valid chunks can be effectively estimated using the above sparse voxel sampling scheme, while for the previously valid chunks (but currently invalid anymore caused by the object movement), we need to check the stored chunks exhaustively, inducing the linear growth of computation with the size of reconstructed 3D model. To address it, we use a second hash table and hash chunks at a coarse level: every $8 \times 8 \times 8$ chunks are organized as a cube, which is spatially hashed based on its central position and only the existing chunk IDs are stored in the corresponding hash entry. In other words, we firstly select the cubes that fall into the frustum of current camera view, and traverse search the existing chunks stored in the selected cubes to check if they are observed by current camera view.

Fig. 4 presents the experiments on fr3office dataset regarding the performance of our sparse voxel sampling by comparing with conventional candidate chunk selection strategy, where the horizontal axis indexes the frames of sequence, and the left vertical axis denotes the precision of estimated valid chunks, the right vertical axis relates to the computational complexity. Apparently, the acceleration of chunk selection is considerable, retaining more than $98\%$ truly valid chunks under the cost of $2\%$ computational complexity on average.

*3) Keyframe based Optimizations:* The selection of valid chunks applies for keyframes merely, the local frames are then integrated accordingly. In other words, the IDs of selected chunks are stored in each keyframe and can be further reused when the keyframe needs to be de-integrated. As further shown in the experiments that TSDF integration for each frame takes approximately $2ms$ and the valid voxel selection takes $6ms$, the waiver of valid voxel selection for local frames significantly reduces the complexity of TSDF fusion, enabling the reintegration using CPU computing.

Resembling [11, 15], color observations are fused into the TSDF model following the average framework in Eqn. 6. Distinction in FlashFusion is that we store the color values implicitly using the multiplication result (represented by unsigned short for each channel) of color value and weight (the maximum weight is set as 255). In this way, there is

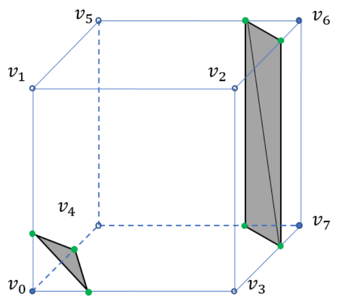


Fig. 5. Illustration of generating polygons using 8 neighboring voxels [4].

no need to calculate the updated color value explicitly for each voxel, while color values can be easily computed in the rendering stage for each generated vertex. To fuse new observations, a single integer addition is enough for the update of the multiplication result while previous methods require 2 additions, 2 multiplications and 1 division to get the updated color value and weight. Note that division requires 38 clock cycles in intel SSE instructions (including latency) while integer addition merely requires 1 clock cycle [5].

## C. Mesh Extraction

In general, given the TSDF fields, we can use incremental marching cubes algorithm [4] to estimate the surfaces represented by triangles (meshes), which however, is impractical to be implemented via CPU computing. For example, the mesh extraction in the state-of-art CHISEL [11] takes around 100ms and 1500ms at the voxel resolution of $20mm$ and $5mm$, respectively. We thus investigate the major computation burdens of mesh extraction, i.e., polygons generation and normal extraction, accomplishing highly accelerated mesh extraction within 60ms at the voxel resolution of $5mm$.

*1) Polygons Generation:* Recall that for each valid voxel $v_0$, its seven neighboring voxels $v_i, i = 1, 2, \cdots, 7$ are extracted composing a cubic $c$ as shown in Fig. 5. Polygons inside this cubic can be calculated given the sdf value of its 8 corners following the classic marching cube algorithm, i.e., vertices merely exist on the edge of $c$ where the sign of the two endpoints' sdf values are different. The exact location $v$ of each vertex is determined by linear interpolation of the two endpoints $v_a, v_b$ based on their sdf values $s_a, s_b$ as follows,

$$\mathbf{v} = \frac{s_a \times \mathbf{v_b} - s_b \times \mathbf{v_a}}{s_a - s_b}. \tag{7}$$

Nevertheless, by examining all the voxels to estimate whether any polygon exists in its neighborhood is improvident. Even though the voxels with small sdf values are closer to surfaces than those with larger sdf values, it is impractical to determine a fixed threshold to filter out voxels with large sdf values, since we are using projective sdf instead of Euclidean sdf. For example, when a surface is observed in parallel to its normal, voxels that close to the surface may have large sdf value. To overcome it, we adopt a dynamic threshold strategy to assign different chunks with different thresholds. When a mesh is created, the maximum absolute sdf value of voxels that contain surfaces is used as threshold. When new observations are introduced and the mesh needs to be updated, voxels with sdf values larger than the threshold are ignored. Given that around $10\%$ meshes are newly created while the other $90\%$ meshes are existing ones that need to be updated at each mesh extraction stage, such strategy successfully achieves 2x acceleration for Polygons Generation.

Moreover, by analyzing the specific structure of the cubic $c$ generated in mesh extraction, we find that the linear interpolation is essentially unnecessary since relative to its origin corner $v_0$, $c$ is a fixed cube whose side length equals to voxel resolution. Since the generated vertices are expected to be on the edge of $c$, allowing one freedom for the position of each vertex, which is linear to the interpolation coefficient: $\frac{s_a}{s_a - s_b}$. The remaining two freedoms merely depend on the edge and can be found through a small look up table.

*2) Normal Extraction:* the normal $n$ of each vertex is computed via the derivative of TSDF field as [13]:

$$\mathbf{n} = \begin{bmatrix} \delta_x \\ \delta_y \\ \delta_z \end{bmatrix} = \begin{bmatrix} s_{i+1,j,k} - s_{i-1,j,k} \\ s_{i,j+1,k} - s_{i,j-1,k} \\ s_{i,j,k+1} - s_{i,j,k-1} \end{bmatrix}, \tag{8}$$

where $i, j, k$ indicates the index of the voxel $v_n$ that is closest to the vertex $v$. Note that $v_n$ can be determined directly in the polygons generation step where $v$ is close to either $v_a$ if $fabs(s_a) < fabs(s_b)$ or $v_b$ if $fabs(s_a) > fabs(s_b)$. Three of the neighbor voxels required for normal estimation of $v_n$ (Eqn. 8 ) are already grouped in the cubic $c$, while the other three ones can be found in the nearby chunks, thus normal can be estimated directly in the mesh extraction step with little computational burden.

The remaining bottleneck of the rendering stage belongs to voxel access which is extensively used, e.g., 7 nearby voxels are accessed in polygons generation for each voxel candidate, and 6 nearby voxels are accessed in normal extraction for each vertex. For inter-chunk voxel access, we need to check the existence of its corresponding chunk firstly and locate the chunk in a large hash table where all chunks are stored. In FlashFusion, a look-up-table is maintained in each chunk when chunks are created or removed. The addresses of its neighbor chunks are stored in the look-up-table thus avoiding the query of the chunk hash map for each voxel in polygons generation step and for each vertex in the normal estimation step.

### D. Re-integration

Benefit from the highly accelerated TSDF fusion and mesh extraction, we can reintegrate depth observations online to realize a realtime 3D reconstruction on CPU computing. At each keyframe, we reintegrate up to 10 previous keyframes. One keyframe may relate to many local frames whereas we only select at most 10 local frames evenly to guarantee that reintegration can be accomplished within certain clock cycles and will not block the whole system pipeline. Insight behinds is: a keyframe links too many local frames may imply that the camera is static or moving slowly, and the consecutive local frames are less informative, thus evenly selecting 10 local frames assures both the efficiency and the diversity.

TABLE I
ACCURACY COMPARISON OF LOCALIZATION ON TUM RGBD DATASET (CM)

| | fr1/desk | fr2/xyz | fr3/office | fr3/nst |
|---|---|---|---|---|
| RGBD SLAM | 2.3 | 0.8 | 3.2 | 1.7 |
| ElasticFusion | 2.0 | 1.1 | **1.7** | 1.6 |
| BundleFusion (on-line) | 1.7 | 1.4 | 2.9 | 1.6 |
| BundleFusion (off-line) | **1.6** | **1.1** | 2.2 | **1.2** |
| FlashFusion | 1.9 | 1.3 | 2.5 | 1.8 |

TABLE II
ACCURACY COMPARISON OF LOCALIZATION ON ICL-NUIM DATASET (CM)

| | kt0 | kt1 | kt2 | kt3 |
|---|---|---|---|---|
| RGBD SLAM | 2.6 | 0.8 | 1.8 | 43.3 |
| ElasticFusion | 0.9 | 0.9 | 1.4 | 10.6 |
| BundleFusion (on-line) | 0.8 | 0.5 | 1.1 | 1.2 |
| BundleFusion (off-line) | **0.6** | **0.4** | **0.4** | **1.1** |
| FlashFusion | 0.7 | 0.8 | 1.1 | 1.4 |

TABLE III
ACCURACY COMPARISON OF SURFACE RECONSTRUCTION ON ICL-NUIM DATASET (CM)

| | kt0 | kt1 | kt2 | kt3 |
|---|---|---|---|---|
| RGBD SLAM | 4.4 | 3.2 | 3.1 | 16.7 |
| FastFusion | 5.6 | 7.5 | 7.0 | 6.6 |
| ElasticFusion | 0.7 | 0.7 | 0.8 | 2.8 |
| InfiniTAM | 1.3 | 1.1 | **0.1** | 2.8 |
| BundleFusion | **0.5** | **0.6** | 0.7 | **0.8** |
| FlashFusion | 0.8 | 0.8 | 1.0 | 1.3 |

## V. EXPERIMENTS

To verify the performance of FlashFusion, extensive experiments are implemented on both synthetic ICL-NUIM [8] dataset (with noise) and real-world TUM RGBD dataset [18], using an Intel Core i7 7700 @3.6GHz CPU. The ICL-NUIM dataset provides ground truth data for both localization and reconstruction, while the TUM RGBD dataset provides ground truth data for localization merely. State-of-the-art including BundleFusion [2], ElasticFusion [22], InfiniTAM [10], RGB-D SLAM [3] , FastFusion [17] and CHISEL [11] are taken for comparisons.

Moreover, the live scanning via an Asus Xtion sensor using FlashFusion that implemented on the CPU computing of the portable tablet Surface Pro[1] is demonstrated, where FlashFusion works at sub-centimeter level (5mm) resolution and achieves global consistency in real-time without the use of GPU computing. The video that contains both live demos as well as the experiments on TUM RGBD dataset is available on the project website[2].

### A. Accuracy

Localization accuracy is evaluated using the rmse of Absolute Trajectory Error (ATE) Sturm et al. [18], as shown

[1]https://www.microsoft.com/en-us/surface
[2]http://www.luvision.net/FlashFusion/

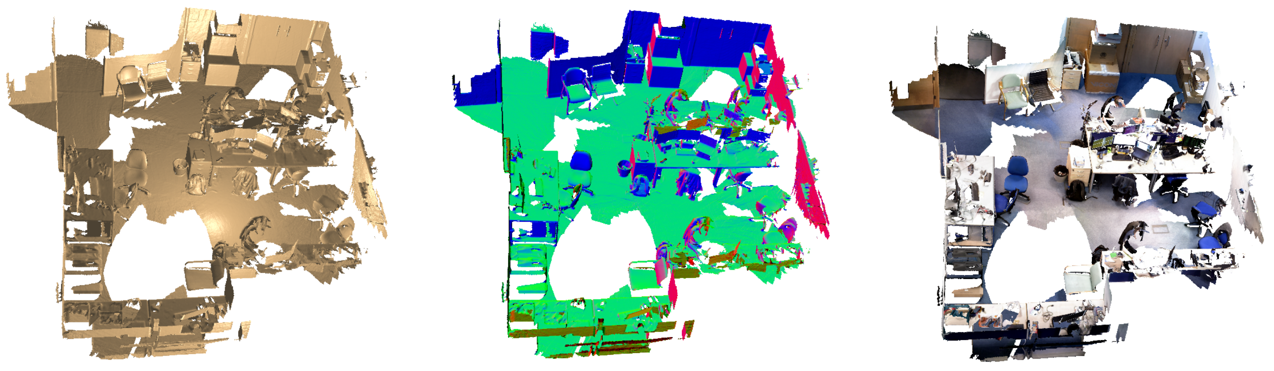
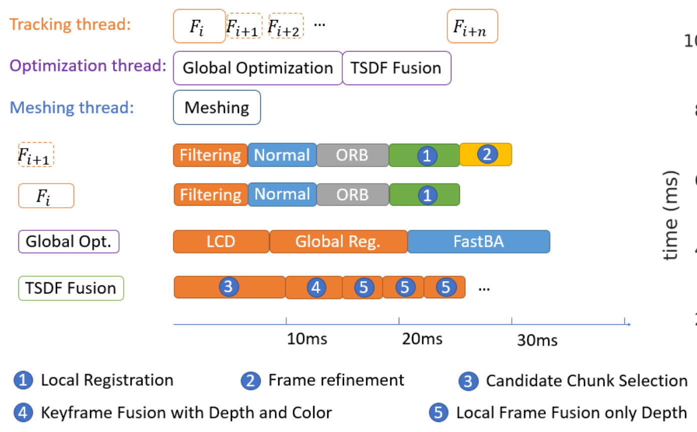
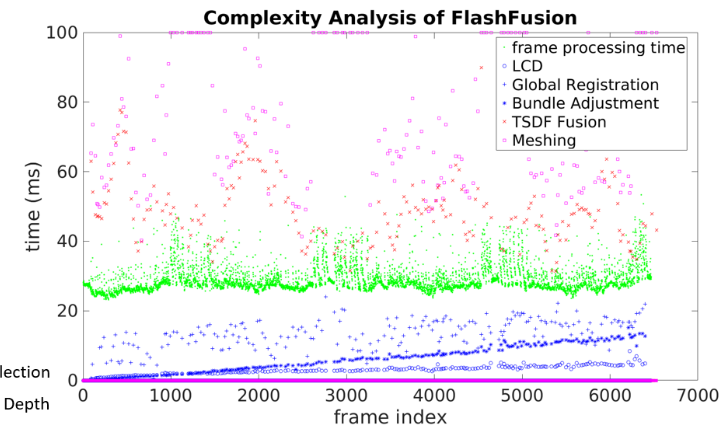

(a) Phong-shaded rendering (b) Normal map (c) Color map

Fig. 6. Qualitative illustration of FlashFusion for fr3office dataset [18], (a) Phong-shaded rendering, (b) normal map, (c) color map.



a. Computing components in FlashFusion



b. Quantized complexity of computing components in FlashFusion

Fig. 7. The thread implementation of FlashFusion and its computational complexity analysis.

in Table I for TUM RGBD datasets and Table II for ICL-NUIM datasets, where a smaller value of ATE implies higher accuracy. Regarding the surface reconstruction accuracy, we compute the differences between the reconstructed meshes and the ground truth 3D model using the SurfReg tool [8], as depicted in Table III for ICL-NUIM datasets.

In general, BundleFusion that adopts two high-end GPU computing performs the best, achieving the highest accuracy in both localization and surface reconstruction, as it matches each keyframe against all the previous keyframes using more robust feature descriptors as well as dense registration. FlashFusion provides relatively lower yet comparable accuracy as BundleFusion, whereas using CPU computing for both localization and reconstruction.

The representative qualitative results of FlashFusion are presented in Fig. 6 for fr3office dataset [18] at 5mm-level resolution, where Fig. 6(a) is the reconstructed model under phong - shaded rendering, Fig. 6(b) is the normal map, and Fig. 6(c) shows its color map. In this dataset, camera moves around the desk and returns to the start point. The procedure of re-integration after loop closure in FlashFusion succeeds to correct the misalignments caused by localization drift, producing a globally consistent 3D model in real-time using CPU computing. More live scanning results can be found in the supplementary video.

## B. *Efficiency*

The detailed computing components in FlashFusion are presented in Fig. 7(a). The tracking thread, serving as the front-end thread for pose estimation, runs at 30Hz constantly. When a new keyframe $F_i$ is inserted, optimization thread will be activated for global pose optimization based on all the previous keyframes. Once the pose of keyframe is determined, valid chunk selection is activated to select chunk candidates to be updated based on current keyframe. When the next keyframe is inserted, $F_i$ and its corresponding local frames are fused into the TSDF field, and then meshing thread starts to update meshes given the updated chunks. All the above computations are accomplished on CPU computing. The reconstructed model including vertices, colors and normals

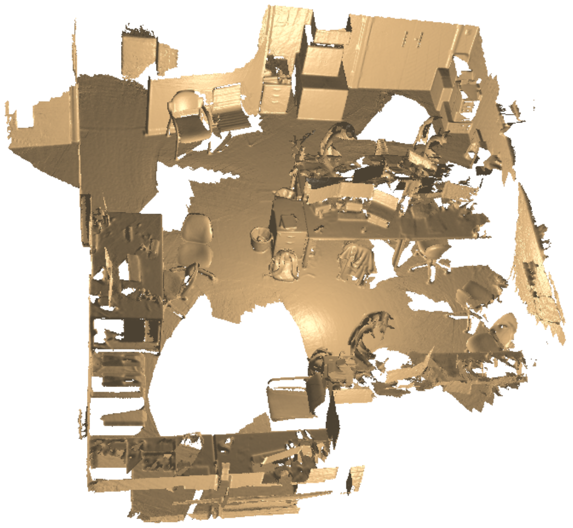


Fig. 8. Reconstructed 3D model of the dyson_lab [22] dataset.

are then transmitted to GPU for visualization merely. To further demonstrate the efficiency of FlashFusion on large scale datasets, quantitative measurements are conducted on the representative dyson_lab [22] dataset which refers the scanning of the whole lab with 6400 frames, as presented in Fig. 7(b). The corresponding qualitative illustration of reconstructed model is shown in Fig. 8. Note that apart from the processing time of each frame, the other 5 components are only invoked at keyframe rate running at back-end threads. Visibly, all the computing components can be accomplished efficiently, guaranteeing the real-time performance of FlashFusion on CPU computing.

As it may be irrational to conduct efficiency comparison between GPU and CPU computing, we compare with state-of-the-art CPU-based systems FastFusion [17] and CHISEL [11] on the fr3office dataset. FastFusion employs an octree data structure where the voxel sizes vary from $20mm$ to $5mm$. It takes FastFusion $20ms$ for TSDF fusion per frame on average while $300ms$ to $1000ms$ for mesh extraction, approximately 6 times slower than FlashFusion. The detailed efficiency comparison that relates to TSDF fusion and mesh extraction under different voxel resolutions ($5mm$, $10mm$ and $20mm$) is presented in Table IV. CHISEL achieves real-time performance at $20mm$ resolution, whereas the computational cost grows significantly with the increase of resolution to prevent the real-time implementation. Instead, FlashFusion enables sub-centimeter level resolution in real-time, i.e., $5mm$ in 300Hz for TSDF fusion and 25Hz for mesh reconstruction, ensuring better performance in applications like VR/AR or mobile robots that need to interact with the environment tightly. Moreover, global consistency that cannot be achieved in CHISEL due to its localization drift and slow TSDF fusion is realized in FlashFusion, as demonstrated in the supplementary video.

TABLE IV
EFFICIENCY COMPARISON BETWEEN CHISEL AND FLASHFUSION (MS)

| | CHISEL | | FlashFusion | |
|---|---|---|---|---|
| | TSDF Fusion | Mesh Extraction | TSDF Fusion | Mesh Extraction |
| $5mm$ | 483 | 1518 | 3.6 | 38.4 |
| $10mm$ | 86 | 312 | 1.1 | 19.7 |
| $20mm$ | 15 | 70 | 0.7 | 6.5 |

## VI. CONCLUSIONS AND LIMITATIONS

In this paper, we present FlashFusion, a CPU-based globally consistent real-time 3D reconstruction system. Based on the proposed technique, TSDF fields can be updated in $300Hz$ at $5mm$ voxel resolution or $900Hz$ at $1cm$ voxel resolution using CPU computing, benefiting the practical applications such as on-board path planning of flying vehicles or robot arms based on TSDF fields, high-resolution dense 3D reconstruction on wearable devices etc.

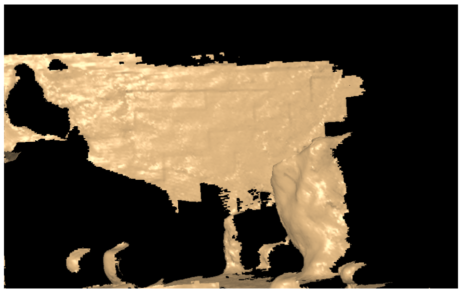


Fig. 9. Artifacts introduced due to the keyframe based integration scheme.

Although the proposed keyframe based fusion technique reduces the complexity of TSDF fusion significantly, it may diminish the quality by inducing minor artifacts at the border of a keyframe as shown in Fig. 9. Since we only select valid voxels for each keyframe and update the TSDF fields of these voxels based on the incoming local frames, surfaces observed in local frames but out the scope of the keyframe are neglected for TSDF fusion. With the incoming of next keyframe, surfaces that belong to both keyframes are fused by $N_1 + N_2$ observations, while surfaces belong to the later keyframe solely are fused by $N_2$ observations ($N_1$ and $N_2$ indicate the number of frames that belong to the first and second keyframe respectively). Artifacts appear due to the inconsistent fusion, which only occur at the border of keyframes and will vanish quickly after more observations are introduced.

For the future work, we will try to solve the limitations reported as well as consider the joint 3D reconstruction and semantic understanding, which may further help robots interact with the real world.

## ACKNOWLEDGMENTS

This work is supported in part by Natural Science Foundation of China (NSFC) under contract No. 61722209 and 61331015.

## References

[1] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312. ACM, 1996.

[2] Angela Dai, Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Christian Theobalt. BundleFusion: Real-time Globally Consistent 3D Reconstruction using On-the-fly Surface Reintegration. *ACM Transactions on Graphics (TOG)*, 36(3):24, 2017.

[3] Felix Endres, Jürgen Hess, Nikolas Engelhard, Jürgen Sturm, Daniel Cremers, and Wolfram Burgard. An evaluation of the RGB-D SLAM system. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1691–1696. IEEE, 2012.

[4] Randima Fernando. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Pearson Higher Education, 2004.

[5] Nadeem Firasta, Mark Buxton, Paula Jinbo, Kaveh Nasri, and Shihjong Kuo. Intel AVX: New frontiers in performance improvements and energy efficiency. *Intel white paper*, 19:20, 2008.

[6] Lei Han and Lu Fang. MILD: Multi-index hashing for appearance based loop closure detection. In *Multimedia and Expo (ICME), 2017 IEEE International Conference on*, pages 139–144. IEEE, 2017.

[7] Lei Han, Lan Xu, Dmytro Bobkov, Eckehard Steinbach, and Lu Fang. Real-time Global Registration for Globally Consistent RGBD SLAM. *http://www.luvision.net/FastGO*, 2018.

[8] Ankur Handa, Thomas Whelan, John McDonald, and Andrew J Davison. A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In *Robotics and automation (ICRA), 2014 IEEE international conference on*, pages 1524–1531. IEEE, 2014.

[9] Olaf Kähler, Victor Adrian Prisacariu, Carl Yuheng Ren, Xin Sun, Philip Torr, and David Murray. Very high frame rate volumetric integration of depth images on mobile devices. *IEEE transactions on visualization and computer graphics*, 21(11):1241–1250, 2015.

[10] Olaf Kähler, Victor A Prisacariu, and David W Murray. Real-time large-scale dense 3D reconstruction with loop closure. In *European Conference on Computer Vision*, pages 500–516. Springer, 2016.

[11] Matthew Klingensmith, Ivan Dryanovski, Siddhartha Srinivasa, and Jizhong Xiao. Chisel: Real Time Large Scale 3D Reconstruction Onboard a Mobile Device using Spatially Hashed Signed Distance Fields. In *Robotics: Science and Systems*, volume 4, 2015.

[12] Robert Maier, Raphael Schaller, and Daniel Cremers. Efficient Online Surface Correction for Real-time Large-Scale 3D Reconstruction. *arXiv preprint arXiv:1709.03763*, 2017.

[13] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.

[14] Richard A Newcombe, Dieter Fox, and Steven M Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 343–352, 2015.

[15] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3D reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (TOG)*, 32(6):169, 2013.

[16] Helen Oleynikova, Alexander Millane, Zachary Taylor, Enric Galceran, Juan Nieto, and Roland Siegwart. Signed distance fields: A natural representation for both mapping and planning. In *RSS 2016 Workshop: Geometry and Beyond-Representations, Physics, and Scene Understanding for Robotics*. University of Michigan, 2016.

[17] Frank Steinbrücker, Jürgen Sturm, and Daniel Cremers. Volumetric 3D mapping in real-time on a CPU. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 2021–2028. IEEE, 2014.

[18] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 573–580. IEEE, 2012.

[19] Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomerantes, and Markus H Gross. Optimized Spatial Hashing for Collision Detection of Deformable Objects. In *Vmv*, volume 3, pages 47–54, 2003.

[20] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[21] Thomas Whelan, Michael Kaess, Maurice Fallon, Hordur Johannsson, John Leonard, and John McDonald. Kintinuous: Spatially extended kinectfusion. 2012.

[22] Thomas Whelan, Stefan Leutenegger, R Salas-Moreno, Ben Glocker, and Andrew Davison. ElasticFusion: Dense SLAM without a pose graph. Robotics: Science and Systems, 2015.

[23] Yu Xiang and Dieter Fox. DA-RNN: Semantic Mapping with Data Associated Recurrent Neural Networks. In *Robotics: Science and Systems*, 2017.

[24] Zhenfei Yang, Fei Gao, and Shaojie Shen. Real-time monocular dense mapping on aerial robots using visual-inertial fusion. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 4552–4559. IEEE, 2017.

[25] Ming Zeng, Fukai Zhao, Jiaxiang Zheng, and Xinguo Liu. A memory-efficient kinectfusion using octree. In *Computational Visual Media*, pages 234–241. Springer, 2012.

[26] Zhengyou Zhang. Microsoft kinect sensor and its effect. *IEEE multimedia*, 19(2):4–10, 2012.