MixedFusion: Real-Time Reconstruction of an Indoor Scene with Dynamic Objects

Hao Zhang and Feng Xu

Abstract—Real-time indoor scene reconstruction aims to recover the 3D geometry of an indoor scene in real time with a sensor scanning the scene. Previous works of this topic consider pure static scenes, but in this paper, we focus on more challenging cases that the scene contains dynamic objects, for example, moving people and floating curtains, which are quite common in reality and thus are eagerly required to be handled. We develop an end-to-end system using a depth sensor to scan a scene on the fly. By proposing a Sigmoid-based Iterative Closest Point (S-ICP) method, we decouple the camera motion and the scene motion from the input sequence and segment the scene into static and dynamic parts accordingly. The static part is used to estimate the camera rigid motion, while for the dynamic part, graph node-based motion representation and model-to-depth fitting are applied to reconstruct the scene motions. With the camera and scene motions reconstructed, we further propose a novel mixed voxel allocation scheme to handle static and dynamic scene parts with different mechanisms, which helps to gradually fuse a large scene with both static and dynamic objects. Experiments show that our technique successfully fuses the geometry of both the static and dynamic objects in a scene in real time, which extends the usage of the current techniques for indoor scene reconstruction.

Index Terms—Scene reconstruction, dynamic reconstruction, single view

1 Introduction

In recent years, real-time indoor scene reconstruction becomes a hot topic in both academia and industry. In academia, with the development of depth recording techniques, 3D point cloud representing the 3D geometry of a real scene can be obtained in real time, which avoids the challenging task of dense 3D reconstruction from color images. With a depth sensor, previous works propose real-time indoor scene reconstruction systems to fuse the point cloud of multiple frames [1], [2], [3]. In industry, the 3D reconstruction of a 360° indoor scene could be directly used or help to generate more 3D content for virtual reality (VR) with high realism. And for augmented reality (AR), it is needed to reconstruct the 3D information of a real scene in real time, to guarantee the rendered virtual objects and virtual information aligning with the real scene. In the newly developed Microsoft Hololens, scanning a real scene is required by many apps.

All previous works for indoor scene reconstruction assume the scene is purely static. However, this assumption may be easily violated in real cases. For example, a scene may contain humans, which, in most cases, do not stay purely still. Or the window of a room is opened, and thus the curtain or other light objects in the room may move with wind. For the former case, it is OK to remove the human, if the scenario is to scan the scene but not the human. However, for the the latter case, it is not reasonable to remove the objects only because they have some small motions. The ideal case for handling moving objects is to align the objects in different poses and fuse their geometries, as well as the static objects. Thus users have the freedom to keep the objects or remove them. Furthermore, if the motions of the dynamic objects are reconstructed, new applications can be enabled, for example, generating realistic dynamic content for viewing and animation. In general, it is important and useful to fuse the geometry of a scene containing both static and dynamic objects, which has not been investigated by any previous works.

Indoor scene reconstruction with dynamic objects is a challenging task. For a pure static scene, we only need to estimate the global camera motion between every consecutive frames before fusing the geometries of these frames. As the global camera motion only contains 6 degree of freedom, it is relative easy to perform the estimation in real time. However, for objects with motions, especially with nonrigid motions, the motion space increases dramatically, and thus real time is difficult to be guaranteed. Furthermore, it becomes more challenging to estimate the global camera motion as it is coupled with local objects motions. If the two kinds of motions cannot be decoupled clearly, neither the static objects nor the dynamic objects can be correctly fused and reconstructed. However, camera motion estimation and segmenting the scene into dynamic and static parts are basically chicken and egg problems. If the camera motion is available, segmentation is easy to be obtained by checking which part in the scene fits the camera motion. On the other hand, if we have a good segmentation, we can directly use the static part to estimate camera motions by traditional ICP.

• *The authors are with the School of Software, Tsinghua University, Beijing 100084, P. R. China, the Key Laboratory for Information System Security, Ministry of Education of China, Beijing 100084, P. R. China, and also with the Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, P. R. China.*
E-mail: zhanghao16@mails.tsinghua.edu.cn, xufeng2003@gmail.com.

Manuscript received 13 July 2017; revised 9 Nov. 2017; accepted 15 Dec. 2017. Date of publication 28 Dec. 2017; date of current version 26 Oct. 2018. (Corresponding author: Feng Xu.)
Recommended for acceptance by K. Zhou.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TVCG.2017.2786233

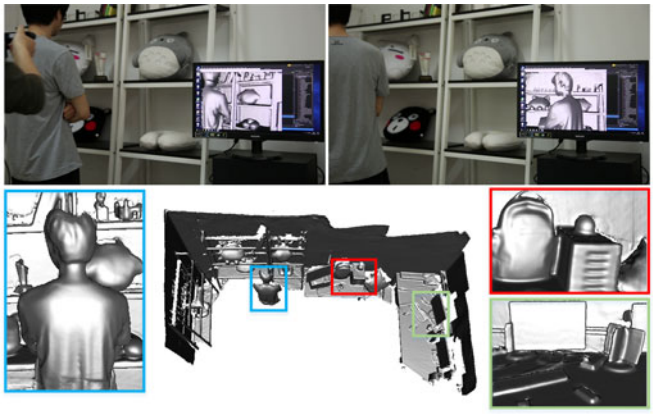


Fig. 1. Our novel real-time system reconstructs scenes with dynamic objects, like human beings. It recovers the 3D surfaces of both the static part (shown in red and green boxes) and the dynamic part (shown in a blue box) simultaneously.

In this paper, we solve the chicken and egg problems by proposing a scene reconstruction technique which allows the existence of dynamic objects (with either rigid or nonrigid motions), like moving human beings (as shown in Fig. 1) and floating curtains. Our technique not only reconstructs the static part of the scene similar to the previous techniques, but also fuses 3D models of the dynamic objects and reconstructs their motions recorded by the sensor. To achieve this, our technique decomposes and reconstructs both the camera motion and the high dimensional nonrigid motions in the scene. And we propose an efficient solution to achieve this in real time.

Our key contribution is a joint global motion estimation and object segmentation method which decouples the global camera motion from other local motions as well as segmenting the scene into dynamic and static objects. In this manner, only the dynamic objects go into the complex nonrigid motion estimation step, which dramatically saves the computation cost. And the static objects are correctly fused as previous works as the camera motion is obtained. For nonrigid motion estimation, we adopt the idea in DynamicFusion [4] to build a node graph to drive dynamic surfaces, and estimate their motions via geometry correspondences. We propose a novel voxel allocation scheme based on a sparse voxel representation to extend the work in handling large scenes with both static and dynamic objects. With a highly optimized implementation on GPU, our whole system performs in real time with a high performance graphic card.

2 Related Work

The technique in this paper aims to reconstruct a scene containing both static and dynamic objects (with rigid or nonrigid motions), so we discuss the static and dynamic 3D reconstruction techniques in this section.

2.1 Real-Time Static Reconstruction

We majorly discuss real-time techniques as our work belongs to this category. With RGB input, real-time 3D reconstruction has been a very challenging task in computer vision, due to the high computation cost for stereo matching and the difficulty in handling low texture regions. Simultaneous localization and mapping (SLAM) is an important technique of this category, aiming to track the motion of the camera and reconstruct the 3D information of a static scene. There are different forms of 3D information for the existing SLAM techniques. Klein and Murray [5] reconstruct sparse 3D points in a scene. Engel et al. [6] and Forster et al. [7] take one step further to achieve semi-dense reconstruction. Also, some works focus on dense reconstruction of the scene. MonoFusion [8] and MobileFusion [9] achieve this goal by volumetric fusion. However, it cannot handle large-scale scenes. With the development of learning techniques, SLAM is also able to reconstruct precise 3D models with semantic information [10], but the categories of scene objects are still very limited.

In recent years, with the development of depth sensing techniques, sensors could directly give dense 3D point cloud in real time. Using the depth sensors, more works achieve real-time dense 3D reconstruction. Keller et al. [11] still use point representation to achieve real-time reconstruction from depth sensors. While KinectFusion [1], [12] uses volumetric representation to achieve the reconstruction of a large scene. Later, voxel hashing [13] is proposed to further improve the effectiveness of volumetric geometry processing. And very high performance system [2] is proposed and released, which achieves real-time performance on mobile devices. Loop closure is also important for reconstructing a large scene, which is considered by Whelan et al. [14]. Recently, Dai et al. [3] achieve not only real-time fusion but also real-time loop closure using online surface re-integration. On top of dense reconstruction, semantic information of the objects in the scene can also be extracted or be used to generate more complete 3D models of the objects. Xu et al. [15] and Zhang et al. [16] make contributions on this point.

2.2 Online Dynamic Reconstruction

Dynamic reconstruction from a single view input is more challenging compared with the static ones. On one hand, only one input image is available for one target scene. To get multi-view information of a scene, a complex multi-view recording system [17], [18], [19] is required. On the other hand, an additional temporal correspondence problem needs to be solved to estimate the motion in the scene. As a consequence, some works require initial geometry templates of the scene [20], [21], [22], others even require to embed a motion prior model, like a skeleton, into the initial template to reconstruct motions [23], [24], [25]. However, these techniques do not meet the requirement of our scenario where we never know what static or dynamic objects will emerge in the scanning process.

Online dynamic reconstruction refers to estimating the scene motion from the online recorded depth, and compensating the motion to fuse the 3D model of the dynamic objects. It does not require pre-processing steps for model acquisition or skeleton embedding, and thus is suitable for our scenario of indoor scene reconstruction with dynamic objects. Liao et al. [26] stitch partial surfaces obtained at multiple frames with the assumption of continuous and predictable motions in the scene. Li et al. [27] use multi-view input to accurately reconstruct loose cloths with nonrigid motions, but the motions should be small and smooth. Recently, Dou et al. [28] gradually fuse multi-frame geometries recorded by a single depth sensor to finally reconstruct a watertight 3D model of an dynamic object, where the nonrigid motions of the object are tracked and compensated in the fusion step.

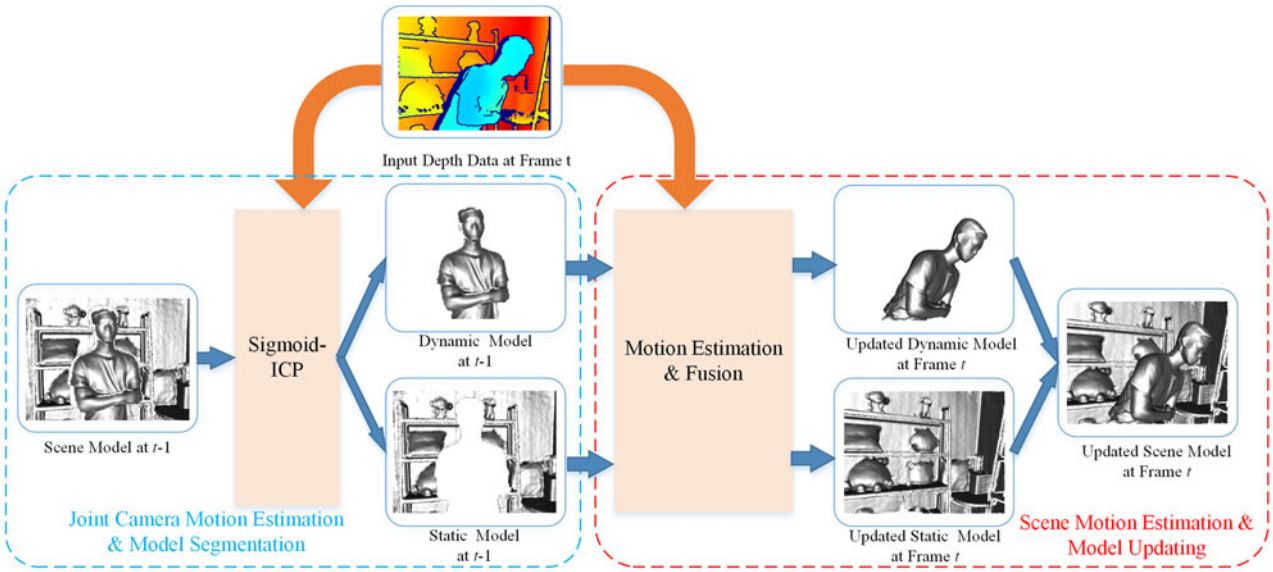


Fig. 2. Overview of our pipeline. The blue box represents our camera pose estimation and model segmentation. The red box represents the optimization of motion field of dynamic objects and the geometry model updating of both static and dynamic objects.

Besides the aforementioned offline techniques, there are also some techniques achieving real-time dynamic reconstruction. The first work is called DynamicFusion [4] which proposes an online nonrigid motion estimation technique and geometry fusion technique to achieve real-time reconstruction with a single depth sensor. Later, VolumeDeform [29] proposes to use SIFT correspondence to improve DynamicFusion, making the system more robust to handle regions with low geometry features. Meanwhile, Guo et al. [30] use shading information to achieve a more robust geometry reconstruction as well as an appearance reconstruction. Even these techniques handle dynamic objects, they assume few objects in a small scene. It is still unclear how to extend these works to handle a large scene with both static and dynamic objects.

3 Overview

The pipeline of our system is illustrated in Fig. 2. In our system, we represent a 3D scene model in a canonical frame, which is assumed to be the camera frame of the first recorded image. And the scene model is called canonical model. Similar to DynamicFusion [4], there is a live model, which is defined in the camera frame of a recorded image, called live frame, and the live model aligns with the input depth map of this frame. To be specific, the live model is obtained by deforming the canonical model with a warping field estimated in the live frame.

When a depth image recorded, the step called *joint camera motion estimation and model segmentation* is executed first, in which the camera motion between the canonical frame and the live frame is estimated and the scene model (both the canonical model and the live model) is separated into two parts, a dynamic model and a static model. The key of this step is the Sigmoid-ICP (S-ICP) algorithm, which aims to accurately estimate camera motions to fit the canonical model to the live recorded depth. As it gradually distinguishes the dynamic part and the static part of the scene in its iterations, only the static part is used to estimate camera motion, while traditional ICP will involve the dynamic part, leading to a wrong estimation. Based on the final accurate camera motion, the two parts of the 3D scene model are segmented by the fitting residual of S-ICP in the live frame.

The second step of our system is the *scene motion estimation and model updating*. In this step, a warping field at current recorded frame, describing the motions of the dynamic model, is estimated and then applied to the canonical model to get the live model. After that, the depth data belonging to the dynamic scene is extracted by the 2D camera projection of the warped dynamic model, and used to update the dynamic model. The static model is also updated by the rest depth representing the static scene. To achieve the updating of both the dynamic and static parts of a scene, a novel mixed voxel allocation scheme is proposed to enable the two updating processes. Finally, the updated live model is warped back to get the updated canonical model, which will be used in the reconstruction of the next frame.

4 Methods

We first introduce some preliminary knowledge and then the two key parts of our system.

4.1 Preliminary

The same with the previous fusion works [1], [4], we use TSDF $S(\mathbf{x})$ to represent the 3D geometry of a scene. The scene is first divided into dense 3D grids and the geometry is represented by $S(\mathbf{x}) = \{\mathrm{v}(\mathbf{x}), \mathrm{w}(\mathbf{x})\}$ defined on the grids, where $\mathbf{x}$ indicates the 3D coordinates of one grid, also noted as a voxel. v and w are the SDF value and the confidence of this value, respectively. The surface mesh M of a scene is implied by the zero-valued surface of TSDF as

$$M = \left\{ (\mathbf{v}, \mathbf{n}) | S(\mathbf{v}) = 0, \mathbf{n} = \frac{\bigtriangledown S(\mathbf{v})}{\| \bigtriangledown S(\mathbf{v}) \|_2} \right\}, \quad (1)$$

where $\mathbf{v}$ indicates 3D mesh vertices and $\mathbf{n}$ indicates the corresponding outer normal.

We use TSDF to represent the canonical model in the canonical frame. Given a warping function $\mathcal{W}$ defining the

motion of the canonical model and the pose change T_g from the canonical frame to a live frame, we calculate the live model $M_{\mathcal{W}} = (\mathbf{V}_{\mathcal{W}}, \mathbf{N}_{\mathcal{W}})$ represented in the live frame

$$M_{\mathcal{W}} = \{(\mathbf{v}_{\mathcal{W}}, \mathbf{n}_{\mathcal{W}}) | \mathbf{v}_{\mathcal{W}} = \mathrm{T}_g \mathcal{W}(\mathbf{v}) \dot{\mathbf{v}}, \mathbf{n}_{\mathcal{W}} = \mathrm{T}_g \mathcal{W}(\mathbf{v}) \dot{\mathbf{n}}\}, \quad (2)$$

where $\mathrm{T}_g = \left[\mathrm{R}_g, \mathbf{t}_g\right]$, $\dot{\mathbf{v}}$ denotes homogeneous vector of $\mathbf{v}$.

4.2 Joint Camera Motion Estimation and Model Segmentation

In this section, we will introduce the S-ICP algorithm at first, which is used to get the accurate camera pose of a live frame when recording a static scene with dynamic objects. Then we introduce how to segment the current reconstructed model into the static part and the dynamic part, where mesh connectivity is used to segment the invisible part of the model in this live frame.

4.2.1 Sigmoid-ICP for Camera Motion Estimation

The key to reconstruct a static scene is camera tracking. After getting the accurate camera pose, live depth data can be transformed back to the canonical frame and fused into the 3D canonical model. It is also very important for reconstructing a scene with both static and dynamic objects, because it is useful not only for the correct fusion of the static scene but also for the segmentation of the scene, which is helpful for motion estimation and geometry reconstruction of the dynamic objects. However, for traditional ICP algorithm, it is difficult to estimate accurate camera poses for a static scene with dynamic objects. The non-rigid motion of the dynamic object prevents a good camera motion estimation. The reason is that the objective of the traditional ICP is to minimize the distance between the corresponding points. This means that if there are some dynamic objects in the scene, the traditional ICP will lean to estimate a wrong camera motion to fit the local motions of the objects. Notice that even though the traditional ICP method can remove some wrong correspondences with the help of a position distance threshold and a normal distance threshold, it cannot remove all the correspondences related to local motions, because local motions may also lead to very small distances in correspondences. Thus extracting the static part of a scene is important for camera motion estimation.

To handle this problem, we propose an iterative manner to jointly solve camera pose estimation and static geometry segmentation. We first build the correspondence between the live model of the previous frame and the live depth of this frame by closest point searching. Unlike traditional ICP, our S-ICP method does not estimate camera motions that minimize the total distances of the correspondences, but minimizes the number of correspondences that do not fit well. This is based on the observation that a correct camera motion will make the static part of a scene fit to the depth but not the dynamic part. As the dynamic part usually takes a small partition of a scene, minimizing the number of unfitted correspondences helps distinguish the two parts and the camera motion is lean to be estimated by the static part only.

To be specific, before the depth data $D(t)$ is fed into the pipeline, bilateral filter is applied to obtain a depth map with reduced noise. Then, all the valid depth values are back projected into the camera coordinate space to get a point cloud. One point is denoted as $\mathbf{V}_d^t(\mathbf{u})$, where $\mathbf{u} = (u, v)$ is in the pixel coordinate. Then the vertex normal $\mathbf{N}_d^t(\mathbf{u})$ is also calculated for each point. Besides, the live model at frame $t-1$: $\left(\mathbf{V}_{\mathcal{W}}^{t-1}, \mathbf{N}_{\mathcal{W}}^{t-1}\right)$ is also obtained as it is a result of the previous frame. Next, the camera motion $\Delta\mathrm{T}_g^t = [\Delta\mathrm{R}_g^t, \Delta\mathbf{t}_g^t]$ between frame $t-1$ to frame t is estimated by aligning $\left(\mathbf{V}_d^t(\mathbf{u}), \mathbf{N}_d^t(\mathbf{u})\right)$ to $\left(\mathbf{V}_{\mathcal{W}}^{t-1}, \mathbf{N}_{\mathcal{W}}^{t-1}\right)$

$$E\left(\Delta\mathrm{T}_g^t\right) = \sum_{\mathbf{u} \in C} F(Dist(\mathbf{u}, \Delta\mathrm{T}_g^t)), \quad (3)$$

where

$$F(Dist(\mathbf{u}, \Delta\mathrm{T}_g^t)) = \begin{cases} 0 & Dist(\mathbf{u}, \Delta\mathrm{T}_g^t) < thr \\ 1 & Dist(\mathbf{u}, \Delta\mathrm{T}_g^t) >= thr \end{cases}. \quad (4)$$

C contains all the pixel coordinates with valid correspondences, which are filtered by the position distance threshold and the normal distance threshold. thr is the threshold to determine whether a correspondence is fitted or not. Similar to [1], the point-to-plane energy is used as the distance of the correspondence defined by $\mathbf{u}$

$$Dist(\mathbf{u}, \Delta\mathrm{T}_g^t) = \left\| \left(\Delta\mathrm{T}_g^t \, \dot{\mathbf{V}}_{\mathcal{W}}^{t-1}(\mathbf{u}) - \mathbf{V}_d^t(\mathbf{u})\right) \cdot \mathbf{N}_{\mathcal{W}}^{t-1}(\mathbf{u}) \right\|_2. \quad (5)$$

However, it is still very difficult to estimate camera motion, because (4) is not differentiable. As we want to achieve real-time performance, we require a differentiable formulation which can be linearized very quickly. To solve this problem, we propose to use a parametric sigmoid function to approximate the original formulation

$$F(Dist) = \frac{1}{1 + exp^{-k(Dist^2 - thr^2)}}, \quad (6)$$

where k is a coefficient to control the slope of the sigmoid function. Then, (3) could be linearly solved. The obtained $\Delta\mathrm{T}_g^t$ will be further used to update $\left(\mathbf{V}_{\mathcal{W}}^{t-1}, \mathbf{N}_{\mathcal{W}}^{t-1}\right)$ and (3) could be minimized again for the next iteration. After the converging of the iterations, we get the estimated $\Delta\mathrm{T}_g^t$ by fusing $\Delta\mathrm{T}_g^t$s in the iterations together. The camera motion T_g^t from canonical frame to the live frame t is finally expressed as

$$\mathrm{T}_g^t = \left[\Delta\mathrm{R}_g^t \mathrm{R}_g^{t-1}, \Delta\mathrm{R}_g^t \mathbf{t}_g^{t-1} + \Delta\mathbf{t}_g^t\right]. \quad (7)$$

Note that the used sigmoid function is actually performing the segmentation of both the depth map and the scene model. Correspondences with $F(Dist(\mathbf{u}, \Delta\mathrm{T}_g^t))$ close to 0 stand for the static scene as they fit the global motion model represented by T_g^t, while correspondences with $F(Dist(\mathbf{u}, \Delta\mathrm{T}_g^t))$ close to 1 stand for the dynamic scene.

4.2.2 Model Segmentation

As formulated in the previous section, (6) could be used to distinguish static and dynamic scene for visible regions. However, there may be some invisible regions which are fused by previous frames, for example, the back side of a human. In this case, mesh connectivity is a useful cue to perform the segmentation as the back side of a human is largely connected to the frontal part and could be segmented as dynamic with the frontal part.

In this section, we propose a model segmentation method based on both the fitting error and the mesh

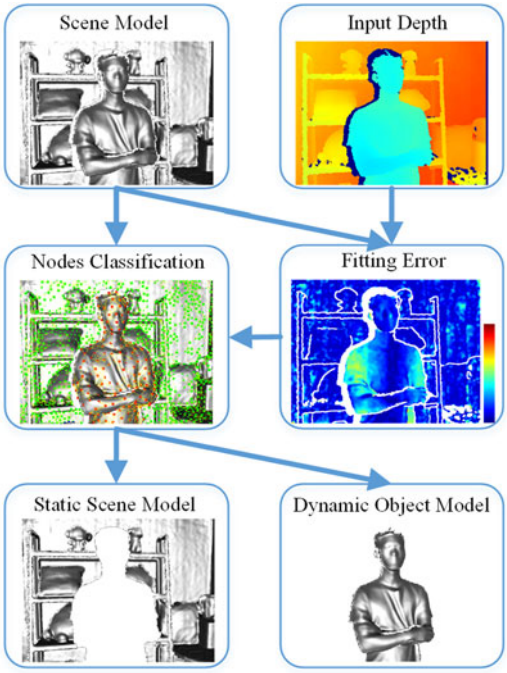


Fig. 3. Illustration of model segmentation. After estimating camera poses, the fitting error map between the reconstructed model and the input depth map is calculated. The mesh connectivity is represented by node graph and all nodes in FOV are classified into two classes, shown in different colors. Then, the scene model can be segmented into a static model and a dynamic model based on the fitting error and the mesh connectivity.

continuity, as shown in Fig. 3. First, we use the result of the previous frame $t-1$ and the camera pose T_g^t obtained by S-ICP to get the live model of this frame t. Then we extract a node set $\mathcal{N}_{FOV}^t$ which covers all the surfaces within the FOV of the camera. The nodes can be considered as a subset of mesh vertices that are uniformly distributed on mesh surfaces and each node controls a set of mesh vertices near the node. A node graph is also constructed to connect neighboring nodes. Details about building nodes and node graphs on mesh surfaces can be found in [4]. The nodes on the invisible surfaces are also in $\mathcal{N}_{FOV}^t$ if they are in the FOV. Second, we use the connectivity of the nodes to classify all the nodes into several classes $\mathcal{N}_{p1}^t, \mathcal{N}_{p2}^t, \ldots, \mathcal{N}_{p\mathrm{m}}^t$. Each class contains nodes with connections and nodes in one class do not have connections with nodes in other classes. Third, we calculate the fitting error of a visible node by the mean fitting errors of the visible surface controlled by the node. Then, we count the number of nodes with large fitting errors (also decided by thr) for each node class, and a class will be selected as dynamic if the number exceeds a threshold. In this case, all nodes in this class, including the invisible nodes, are regarded as dynamic nodes: $\mathcal{N}_D^t = \{\mathbf{V}_{D,1}, \mathbf{V}_{D,2}, \ldots \mathbf{V}_{D,n}\}$. Finally, the dynamic model is extracted as the controlled surface region of all dynamic nodes. The rest surface regions are regarded as the static model.

4.3 Scene Motion Estimation and Model Updating

After camera tracking and model segmentation, we have an accurate camera pose of this live frame and two scene models, the dynamic model and the static model. In this stage, a dense non-rigid ICP algorithm is applied to the dynamic model to estimate the motions that best fit the input depth.

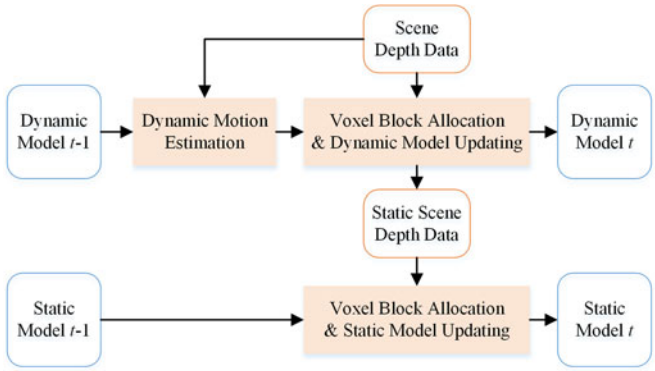


Fig. 4. Pipeline of *scene motion estimation and model updating*. The motion of the dynamic model is estimated first. Next, the dynamic model is updated with the model-based voxel allocation scheme. Then the updated model is used to extract the static depth data, which is used for depth-based voxel allocation and static model updating.

After that, voxel block allocation and geometry fusion are executed for the dynamic model. Next, the updated dynamic model is further used to separate the input depth data into dynamic part and static part. And the static part is used to allocate new static voxel blocks to update the static model. The pipeline of this stage is illustrated in Fig. 4. Note that the voxel allocations for the dynamic model and static model are independently performed with different mechanisms. The former is based on the 3D model while the later is based on the input depth. This joint model and depth based allocation scheme, called mixed voxel allocation, is crucial for the 3D reconstruction and will be illustrated in the following.

4.3.1 Scene Motion Estimation

Since the dynamic model is segmented from the scene model, we follow the framework of [4] to estimate its non-rigid motion represented by a warping function $\mathcal{W}$. To be more specific, we estimate the motion $\mathcal{W}(\mathbf{V}_{D,i})$ of all the dynamic nodes $\mathcal{N}_D^t$ selected in 4.2.2. The motions of the vertices on mesh are obtained by interpolation

$$\mathcal{W}(\mathbf{V}) = SE3(\mathbf{DQB}(\mathbf{V})), \tag{8}$$

where $\mathbf{DQB}(\cdot)$ stands for the dual-quaternion blending, $SE3$ converts a dual-quaternion back to a transformation matrix.

The node motion is estimated by minimizing the energy

$$E(\mathcal{W}) = E_{depth}(\mathcal{W}) + \omega_{smooth} E_{smooth}(\mathcal{W}), \tag{9}$$

where E_{depth} is the point-to-plane energy term

$$E_{depth}(\mathcal{W}) = \sum_{(\mathbf{v},\mathbf{u})\in C} \left\| \left(\mathbf{V}_{\mathcal{W}}(\mathbf{v}) - \mathbf{V}_d^t(\mathbf{u})\right) \cdot \mathbf{N}_{\mathcal{W}}(\mathbf{v}) \right\|_2^2, \tag{10}$$

where C contains all the correspondences between the vertices $\mathbf{v}$ on the dynamic model and the depth data that have the same projection position $\mathbf{u}$ on frame t. $\mathbf{V}_{\mathcal{W}}(\mathbf{v})$ denotes the 3D position of vertex $\mathbf{v}$ warped by the warping field $\mathcal{W}$. The warping field is gradually updated in the minimization of the energy in (9). E_{smooth} is the smooth term

$$E_{smooth}(\mathcal{W}) = \sum_{j\in\mathcal{G}} \sum_{i\in N_j} \left\| \mathcal{W}(\mathbf{V}_{D,i}) \dot{\mathbf{V}}_{D,j} - \mathcal{W}(\mathbf{V}_{D,j}) \dot{\mathbf{V}}_{D,j} \right\|_2^2, \tag{11}$$

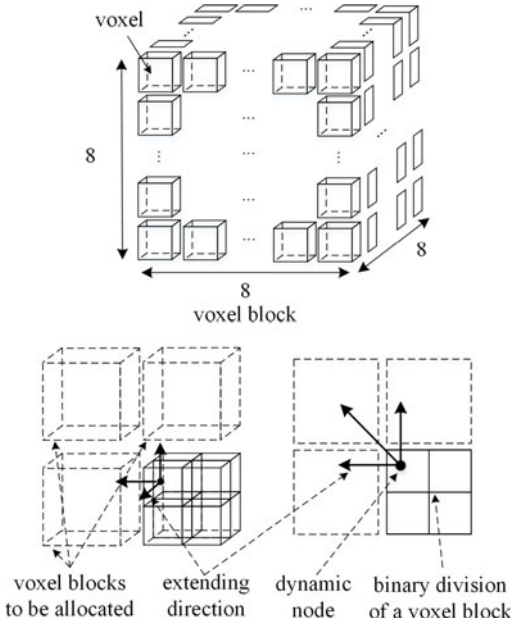


Fig. 5. Mechanism of voxel block allocation for dynamic objects. Voxels are grouped by voxel blocks as shown in the upper part. Voxel blocks near dynamic nodes are allocated for the fusion of the dynamic objects as shown in the lower part.

where $\mathcal{G}$ contains all the dynamic nodes, N_j contains the neighboring dynamic nodes of node j. E_{depth} ensures the warped mesh to be fit to the depth map as close as possible, while E_{smooth} makes the nodes move as rigid as possible. ω_{smooth} is a factor to balance the two terms.

4.3.2 Mixed Voxel Block Allocation

To make an efficient use of memory, we follow the framework of hashing voxel [13] to organize the voxel structure and store the TSDF data. In their method, the memory used for storing the 3D model is allocated by voxel blocks, each of which consists of $8 \times 8 \times 8$ voxels (upper figure of Fig. 5). In this framework, the memory to store voxel blocks is allocated when needed. For pure static scene reconstruction, allocating voxel blocks is performed on the line of sight within a band $[-\mu, \mu]$ for each point $\mathbf{V}_d^t(\mathbf{u})$ in the canonical coordinate [2], [13]. For frame t, they back transform the recorded depth to the canonical frame by the estimated camera pose T_g^t and then perform voxel allocation. However, this method will allocate incorrect voxel blocks for dynamic objects which have additional relative motion to static background. The back transformed depth point may not align with the geometry in the canonical frame, generating wrong geometry in the canonical frame (shown in Fig. 8). To make the allocation correct for dynamic objects, we propose a novel mixed voxel block allocation mechanism.

Our voxel block allocation method includes two mechanisms: one is for dynamic objects and the other is for static objects. The mechanism for dynamic objects is shown in Fig. 5. As mentioned before, voxel block is the basic component for voxel allocation. The goal here is to allocate new voxel blocks that cover new surfaces which may be recorded in the current live frame. The method starts from dynamic node set $\mathcal{N}_D^t$ in the canonical frame. For all the nodes in $\mathcal{N}_D^t$, as they are on the surface fused in previous frames, voxels containing the nodes are all allocated before. The method here is to efficiently allocate the surrounding voxel blocks which have not been allocated. First, every voxel block containing a node is divided into two parts along each axis in canonical coordinate. Thus, there are 8 zones for each voxel block, as shown in the lower figure of Fig. 5. For each dynamic node, we first calculate the zone it falls in and then check whether some adjacent voxel blocks are allocated (The extending direction is illustrated by the solid black arrows in the lower figure of Fig. 5. Thus for one node, there are 7 blocks to be checked). If not, we allocate this voxel block because they are near the node and may contain new surfaces of this frame. This mechanism generates new voxels around the boundary of current dynamic surfaces in the canonical frame. After warping them with the motion of dynamic surfaces to the live frame and updating their TSDF with the input depth, dynamic surfaces will grow with the recording. As voxels are warped with correct motions of dynamic objects, wrong geometry generation is prevented.

For the static object, we can allocate the voxel block following [2]. But before the allocation, we have to extract the static depth from the whole depth data. Otherwise, the depth of non-rigid objects will generate wrong blocks as formulated before. To segment the depth data, we first project the live dynamic model onto the 2D depth map. Then, we ignore all the depth data covered by the projected model. The uncovered depth is regarded as static.

4.3.3 Model Updating

After allocating voxel blocks, TSDF could be updated by depth data. The TSDF $S(\mathbf{x})$ of voxel $\mathbf{x}$ is updated similar with the scheme in [4]

$$S^t(\mathbf{x}) = \begin{cases} [\mathrm{v}'(\mathbf{x}), \mathrm{w}'(\mathbf{x})] & psdf(\mathbf{x}) > -\mu \\ S^{t-1}(\mathbf{x}) & else, \end{cases} \tag{12}$$

where

$$\begin{aligned} \mathrm{v}'(\mathbf{x}) &= \frac{\mathrm{v}(\mathbf{x})_{t-1}\mathrm{w}(\mathbf{x})_{t-1} + min(psdf(\mathbf{x}), \mu)}{\mathrm{w}(\mathbf{x})_{t-1} + 1} \\ \mathrm{w}'(\mathbf{x}) &= min(\mathrm{w}(\mathbf{x})_{t-1} + 1, \mathrm{w}_{max}). \end{aligned} \tag{13}$$

μ is the bandwidth. And $psdf(\mathbf{x})$ is the projective signed distance of voxel $\mathbf{x}$ at the live frame

$$psdf(\mathbf{x}) = \left[\mathbf{K}^{-1} D^t(\mathbf{u}_c)[\mathbf{u}_c^T, 1]^T\right]_z - [\mathbf{x}_t]_z, \tag{14}$$

where $\mathbf{u}_c$ is the pixel of the projection of $\mathbf{x}_t$ on the live depth image, and $[\cdot]_z$ is the z axis coordinate. $\mathbf{x}_t$ is the warped voxel center in the live frame

$$\mathbf{x}_t = \mathrm{T}_g^t \mathcal{W}(\mathbf{x})[\mathbf{x}^T, 1]^T. \tag{15}$$

More details about the fusion method can be found in [4].

5 Results

In this section, we first indicate the performance and parameter settings of our real-time system. Then we demonstrate the effectiveness of the two key components of our method, the S-ICP method, which estimates accurate camera poses for scenes containing dynamic objects and the mixed voxel block allocation method, which correctly allocates voxel blocks for both dynamic and static objects. Next, we

TABLE 1
Average Time Used for Processing One Frame in Our System.

Scene	$T_{S\text{-}ICP}$	T_{MS}	T_{LM}	$T_{VA\&F}$	T_{OP}	T_{total}
Pure Static	6 ms	4 ms	-	3 ms	4 ms	17 ms
Dynamic Object	6 ms	7 ms	20 ms	4 ms	4 ms	41 ms

$T_{S\text{-}ICP}$ is the time used in Sigmoid-ICP, T_{MS} is the time for model segmentation, T_{LM} is the time for scene motion estimation, $T_{VA\&F}$ is the time for voxel allocation and fusion, T_{OP} is the time for other operations, and T_{total} is the total time.

compare our method with InfiniTAM [2] and KinectFusion [1], two typical voxel based scene reconstruction method. DynamicFusion [4], a voxel based nonrigid motion reconstruction method, is also compared with our method. Finally, we present the results obtained on various motion sequences and discuss the limitations of our techniques.

5.1 Performance and Parameters

Our system is running on a computer with a 3.40 GHZ eight core CPU, 16 G RAM and NVIDIA GTX TITAN X graphic card. For pure static scene, our system runs at more than 50FPS. The system drops to 20-30FPS (25FPS on average) when dynamic objects emerge. Details are also shown in Table 1. In case the system runs lower than 25FPS, we drop frames to prevent delay in our system. For voxel block allocation, voxel size is $0.005\ \mathrm{m} \times 0.005\ \mathrm{m} \times 0.005\ \mathrm{m}$ in our system, thus the size of one voxel block is $0.04\ \mathrm{m} \times 0.04\ \mathrm{m} \times 0.04\ \mathrm{m}$. The sampling radius of dynamic node is set to be $0.036\ \mathrm{m}$, a little less than the size of one voxel block.

5.2 Evaluation

Sigmoid-ICP. To test whether the proposed S-ICP can estimate more accurate camera poses for scenes with dynamic objects, we compare S-ICP with traditional ICP and ORB-SLAM on a scene where a person is rotating.

Fig. 6 shows the camera poses of these three methods compared with the ground truth. It is obvious that the camera pose obtained by S-ICP is more accurate than that of traditional ICP and ORB-SLAM. By introducing the sigmoid kernel, our method reduces the influence of the outlier points which have local motions, while neither ICP nor ORB-SLAM considers this. Notice that the result of ICP is similar to that of our S-ICP in the first 20 frames. The reason is that in the first few frames, the accumulated local motion is not that different to the global rigid motion, thus will not affect the rigid motion estimation too much. With more local motion accumulated, ICP becomes worse.

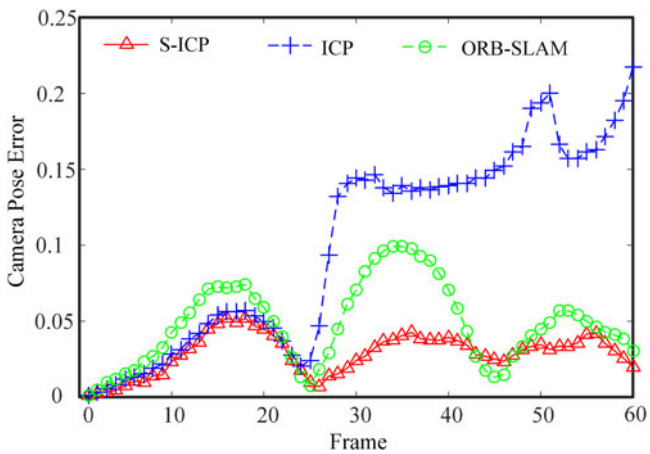


Fig. 6. Camera pose errors of S-ICP, ICP and ORB-SLAM in the first 60 frames of a sequence. The camera pose error is measured by the distance between the 6DOF camera poses(three Euler angles in radians and three displacements) obtained by S-ICP (or ICP or ORB-SLAM) and their ground truth. The ground truth is obtained by ICP performed on the pure static scene data, which is manually segmented. To balance the angle and the displacement, the weights of them are set to 2 and 1 respectively, as Bundlefusion [3] does.

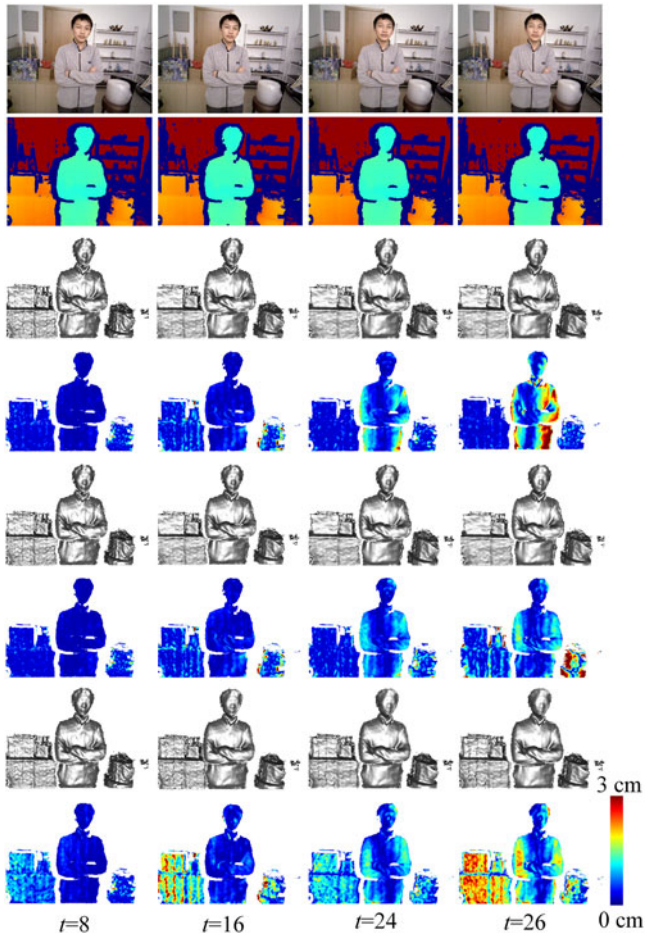


Fig. 7. Fitting errors between the fused models and the input depth for S-ICP, traditional ICP and ORB-SLAM. The first two rows show the reference colors and the input depths of 4 frames. The third and fourth rows are the scene models reconstructed by S-ICP and the errors compared to the input depth. The fifth and sixth rows are the results of traditional ICP, and the seventh and eighth rows are the results of ORB-SLAM. The 4 columns show the results at frame 8, frame 16, frame 24 and frame 26 in a short sequence.

We also show the fitting errors between the rigidly registered models and the input depth in Fig. 7. Ideally, the static part of the scene should generate errors close to 0, while the dynamic part should have large fitting errors. For S-ICP, it is obvious that the errors concentrate on the moving character, as this part has local motions. While for traditional ICP and ORB-SLAM, the errors are distributed on both the static part and the dynamic part, which means the camera motion is not correctly estimated.

Voxel Block Allocation. After the joint camera pose estimation and model segmentation, our method estimates the motion of the dynamic object and then allocates voxel blocks for fusion. The original voxel block allocation method [2] checks all the depth data and allocates new voxel blocks if

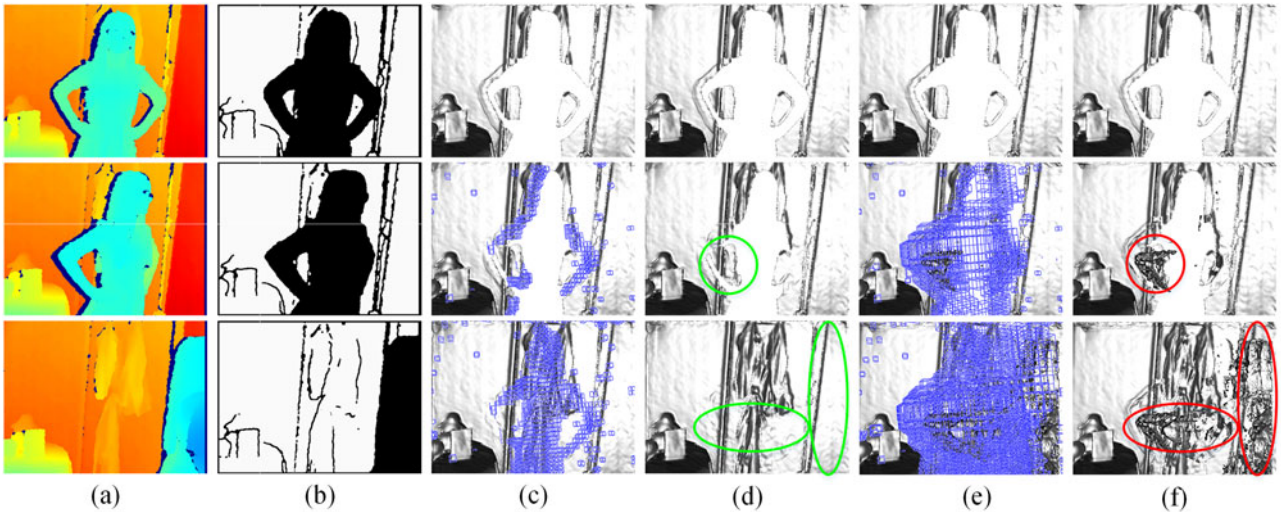


Fig. 8. Validation of our mixed voxel block allocation method. (a)Input depth data of frame 700 (top), 800 (middle) and 1010 (bottom). (b)Depth data for the static scene (The dynamic part is masked off). (c) Our allocated voxel blocks for the static scene (shown as blue cubes), starting from frame 700. (d) The fused geometry results with our voxel block allocation method. (e) and (f) show the allocated voxel blocks and the final geometry of the method in [2], where many wrong voxel blocks are allocated, leading to wrong geometry fusion.

there are no previous blocks aligned with the depth point. This scheme is based on the assumption that the recorded scene is changed due to the camera motion only. However, when there are dynamic objects, camera motion only is not sufficient, thus the original voxel block allocation method will generate some wrong voxels near the depth points of dynamic objects. To fix this problem, in our method, the voxel block allocation is divided into two steps. The first step is to allocate voxel blocks that are adjacent to the dynamic nodes. This is for the dynamic objects. The second step is to project the dynamic model into depth data to exclude dynamic depth data and extract pure static depth data, and then allocate voxel blocks for static objects. Fig. 8 compares our mixed voxel block allocation method with the method in [2]. It is obvious that the reconstructed model of [2] is worse than that of ours (shown in (d) and (f) columns of Fig. 8). By using our mixed voxel block allocation method, the voxel blocks of the occluded blank zone of the static scene model are correctly allocated and the geometry is fused gradually by the depth of the static scene (illustrated in (c) and (d) columns of Fig. 8). However, by using [2], many voxel blocks in the zone are falsely allocated by depth of dynamic objects, resulting in wrong geometry (shown in the (e) and (f) columns of Fig. 8), because [2] does not distinguish the static depth data and the dynamic one.

5.3 Comparison

Here, we compare our method with some existing systems. Fig. 9 is the comparison between our result and that of InfiniTAM [2] and KinectFusion [1], which are the state of the art depth-based system for real-time indoor scene reconstruction. It is obvious that our system can reconstruct a scene with some dynamic objects while InfiniTAM and KinectFusion fail. Please note that the motion of the dynamic object affects the camera motion estimation in InfiniTAM and KinectFusion, which produce artifacts on the static chair and the table regions.

Fig. 10 shows the comparison between our result and that of DynamicFusion [4], which is the state-of-the-art

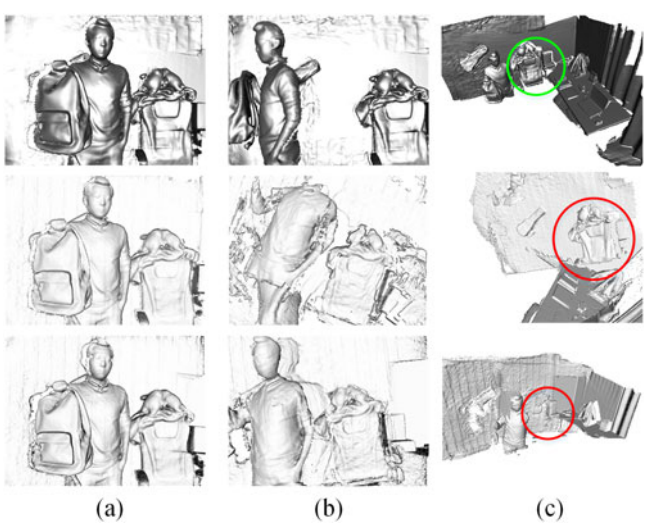


Fig. 9. Comparison of our method with InfiniTAM and KinectFusion. (a) Reconstructed model before objects moving. (b) Reconstructed model when objects moving. (c) Final reconstructed model. The first row shows our results, the second row shows that of InfiniTAM and the third row shows that of KinectFusion.

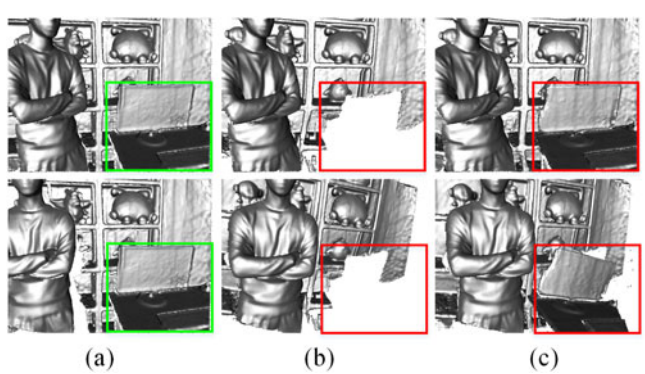


Fig. 10. Comparison of our method with DynamicFusion. (a) Reconstructed model of our method. (b) Reconstructed model of DynamicFusion without depth-based voxel allocation. (c) Reconstructed model of DynamicFusion with depth-based voxel allocation. The first row shows live models while the second row shows canonical models.

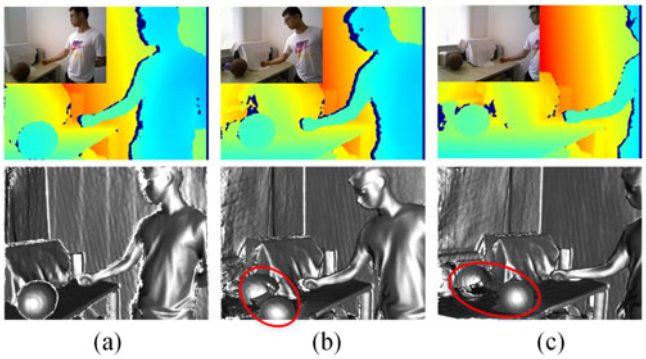


Fig. 11. A failure case where a boy pushes a basketball on the table. (a) The reconstructed model before pushing. (b) After the pushing, the basketball moves on the table. (c) The basketball becomes static again, resulting in two basketball models. The first row shows the input depth and the reference color image. The second row shows the reconstructed results of our system.

depth-based system for real-time non-rigid reconstruction. Our MixedFusion method segments the model in FOV into static and dynamic parts, and we apply dynamic reconstruction only to the dynamic part. For DynamicFusion, the whole scene in FOV is regarded as dynamic. In the original DynamicFusion method, dense voxels are allocated on the boundary of the previous geometry before the reconstruction of the current frame. As a consequence, DynamicFusion cannot reconstruct objects which have a large gap to the previous geometry, which can be seen from (b) of Fig. 10, where the monitor is not fused in the result. We can simply allow DynamicFusion to allocate voxels near the input depth, and thus the monitor can be reconstructed by DynamicFusion. But, the new reconstructed model has wrong orientation and position, which can be seen from the monitor and table in the canonical model in the second row of (c) of Fig. 10. The reason is that the camera pose estimated in DynamicFusion has much drift, leading to wrong orientation and position of new reconstructed models. In this paper, we propose a mixed voxel block allocation method based on sparse voxel representation. And only MixedFusion can reconstruct the whole scene correctly because of more accurate camera pose estimation and segmentation.

5.4 Results

In this section, we show more results of our whole system. Fig. 12 shows some selected results on sequences of various scenes with dynamic objects. More sequential results are presented in the accompanying video. Even though there are some holes and defects on the reconstructed scene model, they diminish gradually with the recording process. In addition, the motions of the dynamic objects are also recorded and their models are refined and completed gradually in the process. These benefits are attributed to two components of our method: the *joint camera motion estimation and model segmentation*, which estimates accurate camera poses and segments the scene model into dynamic and static parts, and the *mixed voxel block allocation*, which allocates correct voxel blocks for dynamic and static parts respectively.

5.5 Limitations

Currently, our technique does not incorporate color information. We agree that color information is important for reconstruction, not only for more accurate camera pose estimation, but also for the realistic rendering of the reconstructed scene. In the future, we will study how to efficiently combine the depth and color information to better track camera motions to achieve loop closure and reconstruct color jointly with geometry. For example, we could implement the method in [3] to handle a 360° scene reconstruction.

For dynamic object reconstruction, similar to [4], we still cannot handle topology changes, as can be seen in Fig. 11. Actually, our work focuses on the combination of dynamic and static reconstructions, but not the individual techniques. As a consequence, our method has the potential to integrate more powerful dynamic reconstruction methods in the future to handle more challenging motions. In addition, the segmentation of our system is based on surface connectivity, and thus it cannot segment dynamic objects if they have large connections with the static ones. To solve this problem, semantic information could be involved to segment different objects in the scene.

6 Conclusion

In this paper, we present a real-time system for indoor scene reconstruction using a single RGBD camera. The key feature

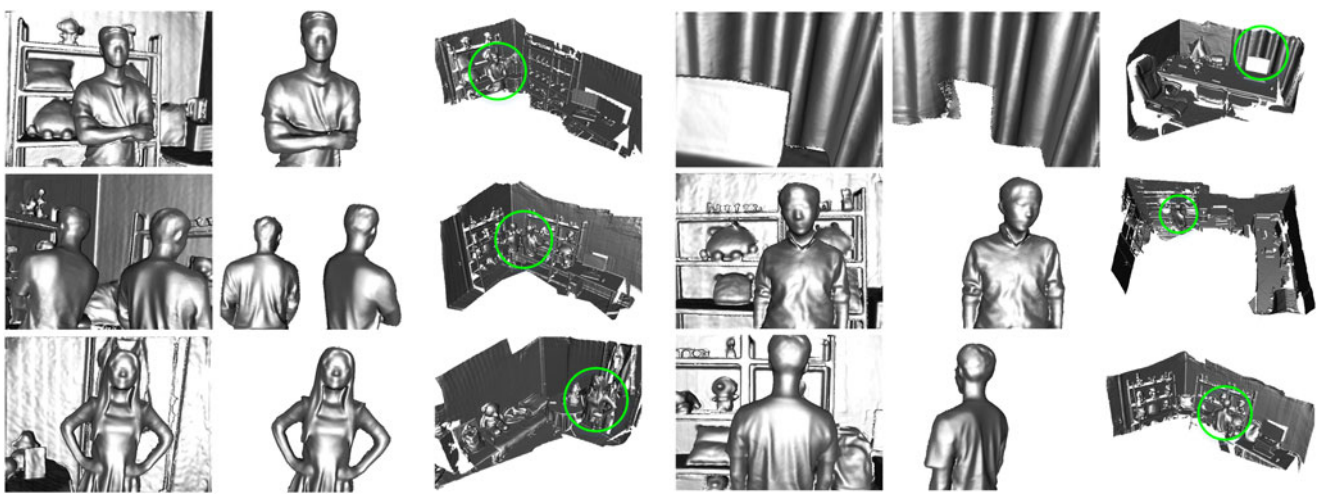


Fig. 12. Results of the reconstructed static scene with dynamic objects. We show 6 results, each of which contains the 3D model in the camera viewpoint, the dynamic object and the whole scene rendered in a god view.

of our technique is that it handles scenes containing dynamic objects. The system achieves camera and scene motion (both rigid and nonrigid motions) reconstruction and high quality fusion of both static and dynamic objects in the scene. With our system, we are able to reconstruct a casual static scenes without requiring of keeping all objects pure static, which enhances the feasibility of scene reconstruction. In addition, our system also records the motion of dynamic objects and reconstructs their 3D models, which could be further used in animation and other related fields. Since our system reconstructs the static and dynamic parts of an indoor scene simultaneously, we believe it achieves a step forward for 3D scene reconstruction.

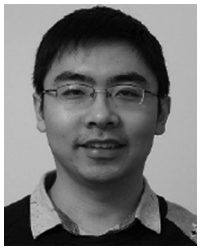
Acknowledgments

This work was supported by the NSFC (No.61671268, 61727808). We wish to express our thanks to the reviewers for their insightful comments. We also would like to thank Kaiwen Guo and Tao Yu for their helpful discussion in this work. Feng Xu is the corresponding author.

References

[1] R. A. Newcombe, et al., "KinectFusion: Real-time dense surface mapping and tracking," in *Proc. 10th IEEE Int. Symp. Mixed Augmented Reality*, 2011, pp. 127–136.
[2] O. Kahler, V. A. Prisacariu, C. Y. Ren, X. Sun, P. H. S. Torr, and D. W. Murray, "Very high frame rate volumetric integration of depth images on mobile device," *IEEE Trans. Vis. Comput. Graph.*, vol. 22, no. 11, pp. 1241–1250, Nov. 2015.
[3] A. Dai, M. Nießner, M. Zollöfer, S. Izadi, and C. Theobalt, "BundleFusion: Real-time globally consistent 3D reconstruction using on-the-fly surface re-integration," *ACM Trans. Graph.*, vol. 36, 2017, Art. no. 24.
[4] R. A. Newcombe, D. Fox, and S. M. Seitz, "DynamicFusion: Reconstruction and tracking of non-rigid scenes in real-time," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2015, pp. 343–352.
[5] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *Proc. 6th IEEE ACM Int. Conf. Mixed Augmented Reality*, 2007, pp. 225–234.
[6] J. Engel, J. Sturm, and D. Cremers, "Semi-dense visual odometry for a monocular camera," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 1449–1456.
[7] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 15–22.
[8] 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 *Proc. IEEE Int. Symp. Mixed Augmented Reality*, 2013, pp. 83–88.
[9] P. Ondrúška, P. Kohli, and S. Izadi, "MobileFusion: Real-time volumetric surface reconstruction and dense tracking on mobile phones," *IEEE Trans. Vis. Comput. Graph.*, vol. 21, no. 11, pp. 1251–1258, Nov. 2015.
[10] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. Kelly, and A. J. Davison, "SLAM++: Simultaneous localisation and mapping at the level of objects," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2013, pp. 1352–1359.
[11] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb, "Real-time 3D reconstruction in dynamic scenes using point-based fusion," in *Proc. Int. Conf. 3DTV-Conf.*, 2013, pp. 1–8.
[12] S. Izadi, et al., "KinectFusion: Real-time 3D reconstruction and interaction using a moving depth camera," in *Proc. 24th Annu. ACM Symp. User Interface Softw. Technol.*, 2011, pp. 559–568.
[13] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3D reconstruction at scale using voxel hashing," *ACM Trans. Graph*, vol. 32, no. 6, 2013, Art. no. 169.
[14] T. Whelan, S. Leutenegger, R. Salas-Moreno, B. Glocker, and A. Davison, "ElasticFusion: Dense slam without a pose graph," in *Robotics: Science and Systems*. Berlin, Germany: Springer, 2015.
[15] K. Xu, et al., "Autoscanning for coupled scene reconstruction and proactive object analysis," *ACM Trans. Graph.*, vol. 34, no. 6, 2015, Art. no. 177.
[16] Y. Zhang, W. Xu, Y. Tong, and K. Zhou, "Online structure analysis for real-time indoor scene reconstruction," *ACM Trans. Graph.*, vol. 34, no. 5, 2015, Art. no. 159.
[17] J. Starck and A. Hilton, "Surface capture for performance-based animation," *Comput. Graph. Appl.*, vol. 27, no. 3, pp. 21–31, 2007.
[18] D. Vlasic, et al., "Dynamic shape capture using multi-view photometric stereo," *ACM Trans. Graph.*, vol. 28, no. 5, 2009, Art. no. 174.
[19] A. Collet, et al., "High-quality streamable free-viewpoint video," *ACM Trans. Graph.*, vol. 34, no. 4, 2015, Art. no. 69.
[20] H. Li, B. Adams, L. J. Guibas, and M. Pauly, "Robust single-view geometry and motion reconstruction," *ACM Trans. Graph.*, vol. 28, no. 5, 2009, Art. no. 175.
[21] M. Zollhöfer, et al., "Real-time non-rigid reconstruction using an RGB-D camera," *ACM Trans. Graph.*, vol. 33, no. 4, 2014, Art. no. 156.
[22] K. Guo, F. Xu, Y. Wang, Y. Liu, and Q. Dai, "Robust non-rigid motion tracking and surface reconstruction using l0 regularization," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 3083–3091.
[23] M. Ye and R. Yang, "Real-time simultaneous pose and shape estimation for articulated objects using a single depth camera," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2014, pp. 2353–2360.
[24] C. Wu, C. Stoll, L. Valgaerts, and C. Theobalt, "On-set performance capture of multiple actors with a stereo camera," *ACM Trans. Graph.*, vol. 32, no. 6, 2013, Art. no. 161.
[25] Y. Chen, Z.-Q. Cheng, C. Lai, R. R. Martin, and G. Dang, "Realtime reconstruction of an animating human body from a single depth camera," *IEEE Trans. Vis. Comput. Graph.*, vol. 22, no. 8, pp. 2000–2011, Aug. 2016.
[26] M. Liao, Q. Zhang, H. Wang, R. Yang, and M. Gong, "Modeling deformable objects from a single depth camera," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2009, pp. 167–174.
[27] H. Li, E. Vouga, A. Gudym, L. Luo, J. T. Barron, and G. Gusev, "3D self-portraits," *ACM Trans. Graph.*, vol. 32, no. 6, 2013, Art. no. 187.
[28] M. Dou, J. Taylor, H. Fuchs, A. Fitzgibbon, and S. Izadi, "3D scanning deformable objects with a single RGBD sensor," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2015, pp. 493–501.
[29] M. Innmann, M. Zollhöfer, M. Nießner, C. Theobalt, and M. Stamminger, "VolumeDeform: Real-time volumetric non-rigid reconstruction," in *European Conference on Computer Vision*. Berlin, Germany: Springer, 2016, pp. 362–379.
[30] K. Guo, F. Xu, T. Yu, X. Liu, Q. Dai, and Y. Liu, "Real-time geometry, albedo and motion reconstruction using a single RGBD camera," *ACM Trans. Graph.*, vol. 36, no. 3, 2017, Art. no. 32.

Hao Zhang received the BS and ME degrees in optical engineering from Beihang University, Beijing, China, in 2013 and 2016, respectively. He is currently working toward the PhD degree in the School of Software of Tsinghua University. His research interests include dynamic reconstruction and motion analysis.



Feng Xu received the BS degree in physics from Tsinghua University, Beijing, China, and the PhD degree in automation from Tsinghua University, Beijing, China in 2007 and 2012. He is currently an assistant professor in the School of Software, Tsinghua University. His research interests include face animation, performance capture, and 3D reconstruction.