Online Structure Analysis for Real-time Indoor Scene Reconstruction

Yizhong Zhang* Weiwei Xu† Yiying Tong‡ Kun Zhou*§

*State Key Lab of CAD&CG, Zhejiang University †Hangzhou Normal University ‡Michigan State University

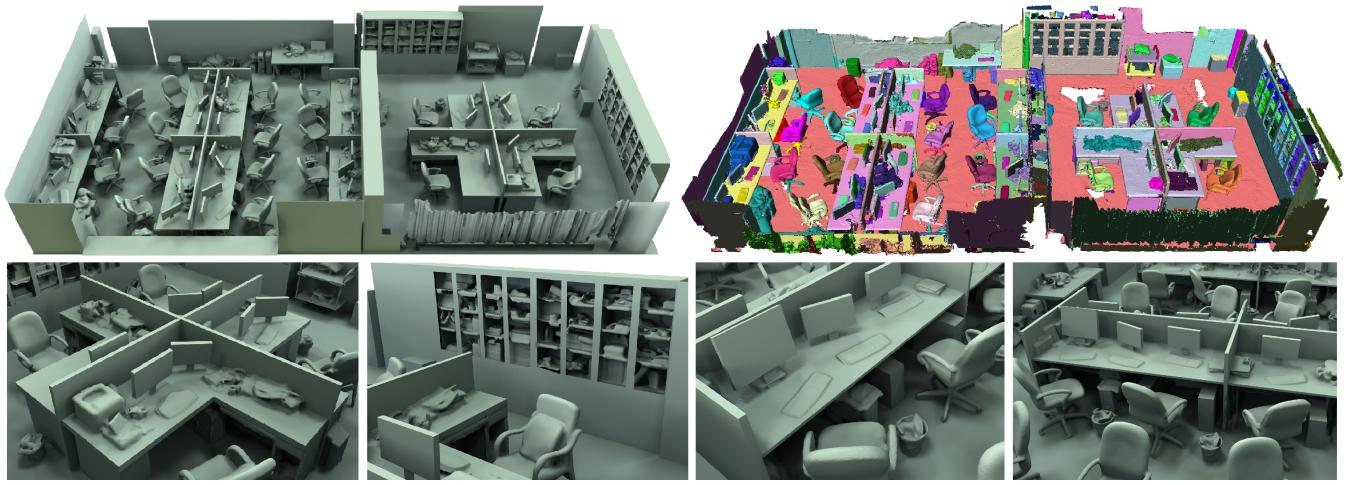


Figure 1: *A lab scene ($100m^2$) reconstructed on-the-fly. The complete scan finished within 70 minutes. Top left: the final reconstructed model by our system, which contains planar regions (polygons) and extracted meshes for separate objects, created and refined progressively in real-time by our online analysis procedure during the scan. Top right: the result of the plane/object labeling procedure, which provides the segmentation of planes and objects for the online analysis. The colors distinguish planar regions and objects by their labels in the volumetric data structure. Bottom: close-up views of the reconstructed scene.*

Abstract

We propose a real-time approach for indoor scene reconstruction. It is capable of producing a ready-to-use 3D geometric model even while the user is still scanning the environment with a consumer depth camera. Our approach features explicit representations of planar regions and non-planar objects extracted from the noisy feed of the depth camera, via an online structure analysis on the dynamic, incomplete data. The structural information is incorporated into the volumetric representation of the scene, resulting in a seamless integration with KinectFusion's global data structure and an efficient implementation of the whole reconstruction process. Moreover, heuristics based on rectilinear shapes in typical indoor scenes effectively eliminate camera tracking drift and further improve reconstruction accuracy. The instantaneous feedback enabled by our on-the-fly structure analysis, including repeated object recognition, allows the user to selectively scan the scene and produce high fidelity large-scale models efficiently. We demonstrate the capability of our system with real-life examples.

*yizhongzhang@zju.edu.cn, kunzhou@acm.org
†weiwei.xu.g@gmail.com
‡ytong@msu.edu
§Corresponding author

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation – Digitizing and Scanning

Keywords: 3D scanning, plane detection, object detection, camera tracking, drifting

1 Introduction

The wide availability of consumer depth cameras has stimulated much research in real-time 3D reconstruction. In particular, the KinectFusion system [Izadi et al. 2011; Newcombe et al. 2011] has achieved great success in generating visually compelling geometries at interactive frame rates during 3D scanning. It enables common users to digitize indoor environments surrounding them, and use the generated models directly in various applications such as augmented reality, autonomous navigation for robots, and interior design.

Several algorithms [Chen et al. 2013; Nießner et al. 2013] have been proposed recently to improve KinectFusion in terms of the reconstruction speed, scalability and quality. The main focus is on designing effective and efficient data structures that can make better use of graphics hardware to record the scanned geometry. The output of these algorithms is "raw" geometry data (e.g., point clouds or implicit surfaces stored within a volumetric data structure), which do not contain any structural information (e.g., planes and objects) needed in typical graphics applications. An offline post-processing step is often mandatory to convert these unstructured geometric data into a "clean" 3D scene with well-defined structures.

In this paper, we propose an approach to reconstructing a meaningful 3D scene model while the user is still scanning an indoor environment with a depth camera. Based on the observation that indoor environments usually consist of many planar surfaces (e.g., floors and walls) and isolated objects (e.g., chairs, cabinets and computer-

s), we perform an online structure analysis on the scanned geometry to extract such planes and objects to form a structured scene. The extracted planes not only fit the scanned geometry well, but also obey the parallelism or orthogonality among planes. The intersection lines of planes are also perfectly aligned. The extracted objects are represented as separate triangular meshes, and repeated objects can also be identified. All computations are performed on the fly and the user can interactively view the reconstructed scene model and thus receive instantaneous feedback during scanning.

The benefits of our approach are four-fold. *First*, our output is a meaningful scene model (planes and object meshes), which can be directly used in a multitude of applications. This saves the offline postprocessing step required in previous techniques. *Second*, performing on-the-fly structure analysis and allowing the user to view the structured model during scanning can guide him/her to scan the scene more efficiently. For instance, when scanning a corner of a room, if the three intersecting planar regions (the floor and two walls) and their boundaries have been extracted, the user would know that the geometric description of the corner is sufficient and move on to other parts of the room. Furthermore, the online analysis can conservatively expand extracted planar regions to automatically cover some unobserved areas. Duplicate object detection can also help avoid repeated scanning of nearly identical objects. *Third*, our online analysis can naturally interact with the user during scanning, which takes advantage of on-the-fly feedback from the user in resolving ambiguities caused by insufficient scanning and results in more accurate analysis results. For example, if the boundaries of an extracted planar region are not accurate enough, or an object detected to be a duplicate of another object scanned earlier does not contain sufficient geometric details, the user can keep scanning the boundaries of the flat region or the object carefully until the results become satisfactory. In contrast, offline analysis methods can only work on the previously scanned sequence, which may not provide enough information to produce desirable results. *Finally*, the global planes extracted by our online analysis can help reduce camera tracking drift during scanning, and thus improve the reconstruction quality when scanning large-scale environments.

We have implemented the approach in a real-time 3D reconstruction system, and experimented on interactive reconstructions of typical indoor environments. We demonstrate that the system can effectively generate meaningful models of entire scenes with well-structured planes and separate object meshes. We also compare our approach with the state-of-the-art algorithms to show the improved reconstruction quality and the reduced camera tracking drift.

1.1 Related Work

3D reconstruction has been a long-standing problem in computer graphics and computer vision. The literature on the topic is vast, so we focus on the most relevant work on 3D scene reconstruction and structure analysis.

3D scene modeling with depth cameras. One major advantage of depth cameras is the ability to capture *dense* depth maps in *real-time*, which leads to their increasing popularity in 3D scene reconstruction. One only needs to gradually fuse the captured depth maps into a unified 3D coordinate system to obtain the final reconstruction result. The seminal work on KinectFusion by Izadi et al. [2011] achieved real-time performance on registering and merging the captured depth maps to a uniform volumetric grid by using a GPU implementation. In order to overcome the size limit of the volumetric grid, moving volume approaches are adopted in [Roth and Vona 2012; Whelan et al. 2012] to swap out voxels not in view of the depth camera from the graphics memory. In [Steinbrucker et al. 2013], a multi-scale octree data structure and dense image alignment are adopted to reconstruct large-scale indoor scenes such as nine rooms along a corridor. Hierarchical spatial data structures and streaming algorithms are developed in [Chen et al. 2013] to extend KinectFusion to large-scale scenes. A recent development in reducing the overhead of the hierarchical data structure is to employ voxel hashing [Nießner et al. 2013].

Although these algorithms can scan visually compelling 3D data in real-time, they still suffer from noises in depth camera output. The accuracy of camera tracking is often imperfect for volumetric fusion and loop closure. Zhou and Koltun performed offline analysis of camera trajectories to achieve dense scene reconstruction [2013], high fidelity texture mapping [2014a], or simultaneous localization and calibration [2014b]. An interactive approach is adopted in [Du et al. 2011] for scene loop closure in depth camera based indoor scene modeling. Our work shows that online structure analysis can significantly improve both depth data fusion and camera tracking, leading to robust high-fidelity reconstruction.

Structure analysis. There are abundant prior structures in typical indoor scene geometry, such as the existence of planar regions and boxes due to the large number of man-made objects in indoor scenes, the parallelism and orthogonality among such planes, and repetition of object shapes. These can be exploited to guide 3D reconstruction or analysis algorithms for improved accuracy.

The Manhattan plane assumption has been used in dense map computation for image-based indoor scene modeling [Furukawa et al. 2009a]. It is also used to estimate the 3D layout of an indoor scene, such as the dimensions and locations of rooms and object boxes [Lee et al. 2009; Lee et al. 2010; Pero et al. 2012]. In outdoor scene modeling, Tomono et al. [2012] proposed to first extract planes from multi-view reconstruction and then propagate the detected planes to the textureless image pixels via graph cut. It can well reconstruct outdoor planar objects, such as roads, sidewalks and building facades. However, these algorithms are designed for image input instead of depth map input. They also rely on the availability of the entire data set of the scene, and thus are not purposed for real-time reconstruction applications.

Due to severe occlusion in cluttered indoor scenes and noises in the captured depth data, it is challenging to directly reconstruct high-quality 3D indoor scene models. Thus, object-level structure analysis is investigated to match the captured objects to the high-quality 3D models in databases for improved reconstruction quality [Nan et al. 2012; Shao et al. 2012; Salas-Moreno et al. 2013], while object repetition is explored to speed up large scale indoor scene reconstruction [Kim et al. 2012; Mattausch et al. 2014]. In contrast, our approach does not rely on a 3D model database, but performs *online* repeated object analysis to simultaneously reduce the scanning burden by providing real-time guidance to the user and improve the reconstruction quality through local volume fusion.

Plane detection has been widely used to reduce the camera tracking drift in 3D scanning [Biber and Strasser 2003; Dou et al. 2013; Ataer-Cansizoglu et al. 2013; Taguchi et al. 2013]. All these methods are developed based on point cloud representations; whereas our design is based on a volumetric grid enhanced with structural information and can thus be seamlessly integrated into the KinectFusion framework to harness the advantage of volumetric depth data fusion. Moreover, in these methods, planes are detected for each individual frame *independently* and used merely for the Iterative Closest Point (ICP) procedure in the current frame; whereas in our case, planar regions are constructed and maintained in a *global data structure* and leveraged in real-time to improve accuracy as well as to reduce the user scanning workload. A concurrent work [Silberman et al. 2014] formulated plane detection and contour completion as optimization problems to provide layout of complete rooms and

partially occluded objects, but the procedure lacks inter-plane relationship analysis and can take one minute for a typical volume.

Polygonal approximation of planar regions. In the captured depth data, the planar boundary is composed of unordered, discrete points. It is necessary to convert them into a compact polygon representation. Such polygonal approximation of discrete points is a well-studied problem in image vectorization and planar region extraction in 3D point cloud data processing [Kolesnikov 2003; Reisner-Kollmann et al. 2013; Li et al. 2011].

In edge-based image vectorization, after sequential tracing of an object boundary, a number of algorithms have been developed to select sparse anchor points in the traced boundary pixels to generate the final polygon, including top-down [Douglas and Peucker 2011; Ballard 1981], bottom-up [Fahn et al. 1989; Latecki and Lakmper 1999], and global optimization methods [Glover and Laguna 1997; Sun and Huang 2000].

Existing works on planar SLAM (simultaneous localization and mapping) frequently use the convex hull technique for planar boundary extraction in fast point-cloud data processing [Biswas and Veloso 2012; Dou et al. 2013]. In contrast, Lee et al. [2012] extracted the boundary polygon of the visible part of a plane by node elimination using object boundary information in the segmented depth image. In [Arikan et al. 2013], alpha-shape is used to detect boundary points from point-cloud data, and local adjacency relations are used to improve the polygon boundary.

The above algorithms assume that the planar region is fully observed and its polygonal approximation is performed to faithfully reconstruct only the visible part. Our algorithm differs in that it deals with incomplete, dynamic scan data, thus it lends itself well to handling online reconstruction. It is also designed to estimate occluded parts of planar regions. For this purpose, we start with a region delineated by the predicted bounding box and plane intersection lines, and use observed data to trim the parts that are verified to be outside the region. This feature is crucial for proper reconstruction, as scan data of indoor scenes typically contain a large number of occluded planar regions.

1.2 Contributions

Compared to existing techniques, the main contributions of our work can be summarized as

- an integral real-time approach to enhancing KinectFusion with crucial structural information, including planes and isolated objects, allowing the user to view a "clean" 3D scene with well-defined structures during the scan;
- a progressive labeling algorithm segmenting incomplete scanned volume data into planar and non-planar regions, and an online analysis algorithm constructing global data structures of planes with polygonal boundaries and non-planar objects;
- and further improvement to reconstruction robustness and accuracy through the recognition of typical planar structures, relationships among planes, and repeated objects based on partial observation.

2 System Overview

Our system extends the voxel Hashing based KinectFusion pipeline [Nießner et al. 2013] to fuse both geometric and structural information, in particular, plane/object labels, to achieve online structure analysis. The volumetric data structure in our system thus stores

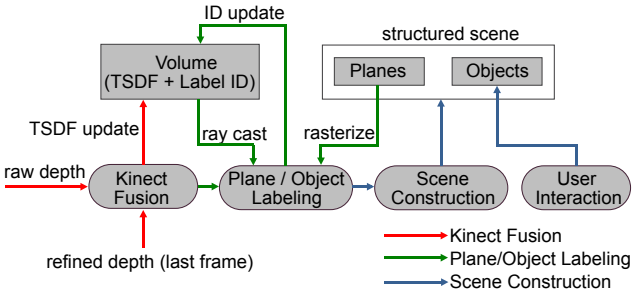


Figure 2: *Pipeline overview.*

the fused signed distance and plane/object label in each voxel. We also design additional plane and object data structures to hold the overall geometry information of labeled planes and objects as well as the mutual relations among planes.

As shown in Fig. 2, the input of our system is the captured depth maps, as in the KinectFusion pipeline. In intermediate processing, we perform depth map fusion and plane/object labeling, and update associated plane and object data structures to record geometry and detected structural relationships. The system outputs refined plane and object geometries as the final reconstruction result.

2.1 System Pipeline

The data processing in our system consists of three major components (see the colored arrows in Fig. 2): (1) a KinectFusion algorithm taking the raw depth image and the refined depth map of the last frame from the global implicit surface representation (i.e., the distance field recorded in the volume) and structured scene as input, and producing the current camera pose through an ICP algorithm; (2) a plane/object labeling algorithm (Sec. 3) that reliably labels the pixels of the raycasting depth map with identifiers (IDs) of planar regions or separate non-planar objects to generate a plane/object ID map by using heuristics for indoor scenes; and (3) a scene construction algorithm (Sec. 4) to construct/update the scene model with the modified plane and object information. User interaction (Sec. 5) is incorporated into the process of object mesh extraction and repeated object detection. The updated labels are fused into the volume to prepare for the processing of the next captured depth map.

KinectFusion is the first stage of our pipeline when processing each frame. When a new frame arrives, we estimate the camera pose by the point-plane ICP between the input depth map and the refined depth map of the last frame. With the estimated camera pose, the raw depth map and color image are fused into the volumetric data structure. Then we perform raycasting also with the camera pose to extract the updated depth map of the current frame using the updated volumetric data.

Plane/object labeling is performed next to update the plane and object labels of volume cells according to the new frame. From the viewpoint of the new frame, a plane/object ID map is also generated by raycasting the volume with the current IDs, in addition to the depth map in the previous stage. The labeling process first attempts to expand the existing planar regions or objects, and then proceeds to cluster pixels into new planes/objects. Separate planes and objects may be merged during the process. The labeling result is finally mapped back into the volume data structure.

Scene construction refines all plane and object data structures according to the labeling result, performs plane-based structure analysis to explore configuration relationships among planes and detect duplicate objects in the scene. Surface meshes for objects are ex-

tracted and added to the scene with very simple user interaction. With the updated volumetric data and structured scene, a refined depth map of the scene at the current camera pose is then generated as the predicted input of the next frame, which will be used for the next ICP-based camera tracking.

Design rationale. A major design goal of our system is to enhance the volumetric grid structure of KinectFusion with structural information and perform plane/object analysis on this volumetric data structure. Such a design has two advantages. First, it allows us to exploit the real-time depth map processing power of voxel hashing in depth fusion. Second, it enables us to fuse the compact dynamic structural information incrementally. As the capture progresses, the plane and object information evolves, and our system updates the label information in the volume to reflect such changes. Therefore, the online structure analysis is essentially a volumetric labeling procedure, since it clusters voxels into planar regions or isolated objects.

We reconstruct planes and objects as the final output because indoor scenes usually consist of planar surfaces and isolated objects. Moreover, planes are essential building blocks of indoor scenes and they are typically orthogonal to or parallel with some other planes. These planes and their mutual relations may be distorted in KinectFusion reconstruction results, especially with large scenes. Our basic assumption is that most of these distortions result from noises in the depth camera and accumulation errors in camera tracking. Thus, we propose to construct a global data structure of planes to rectify these distortions so as to enhance the accuracy of the overall reconstruction and camera tracking. For interactive performance and scalability, we emphasize simplicity in every aspect of the system and compatibility with the GPU architecture.

2.2 Data Structure

Our system maintains three major data structures during scanning, namely volume, plane and object, as shown in Fig. 3.

```
struct Voxel {
    half float sdf;
    ushort id;
    uchar RGB[3];
    uchar weight;
};
```

```
struct Plane {
    ushort id_in_vol;
    list points;
    list trim_points;
    polygon boundary;
};
```

```
struct Object {
    ushort id_in_vol;
    box AABB;
    Mesh* surface;
    ushort copy_id;
    mat4x4 trans;
};
```

Figure 3: *Data structures maintained in our system.*

Volume. We use the volumetric representation in an extended KinectFusion algorithm [Nießner et al. 2013] to represent the entire (unbounded) domain through the zero-crossings of a signed distance function. Additional details on efficient implementation of the truncated signed distance function (TSDF) are in Sec. 6. In each voxel, we store the signed distance to the closest surface point, the color, and the weight for a weighted running average in real-time fusion into the existing structure. In addition, we supplement each voxel with an ID of the plane or object that the corresponding surface point belongs to.

When extracting the surface from the TSDF, a surface point is created at the zero crossing point along a casting ray. Once the zero crossing point is calculated, the associated ID and color is determined by the voxel in which the point resides.

Plane. A planar region is described by a list of interior points in our representation. For simplicity, we also call a planar region simply as a *plane*. We consider the underlying plane as the least-squares

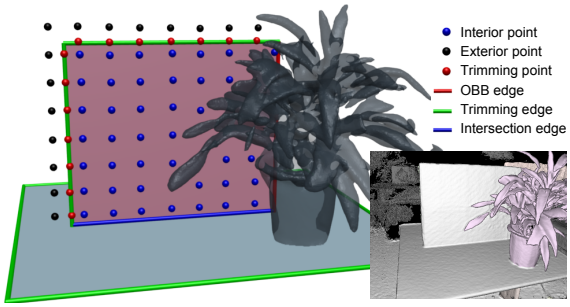


Figure 4: *Data structure of a planar region. As it is possibly occluded by other objects, we predict the shape of the boundary polygon from the partially observed points on the plane.*

fit to the points in the list. The list is augmented at each frame with the points labeled as part of this planar region during the labeling stage. The data structure also maintains a list of trimming points, which delineates some edges of the boundary polygon. The boundary polygon is composed of three possible types of boundary edges, namely, *OBB edge*, *trimming edge* and *plane intersection edge*. The *OBB edge* is formed by the oriented bounding box (OBB) of the entire point list. The *trimming edge* is the part of the boundary, outside of which objects behind the plane have been observed. Such edges are fit from the trimming points. The *plane intersection edge* is formed by the intersection with another planar surface, so it is along the intersection line of the two planes.

Object. A (non-planar) object is encoded implicitly by the voxels sharing its ID in the volumetric data structure. An axis-aligned bounding box (AABB) of the object is maintained to facilitate operations on the object. The bounding box may be expanded with each new frame if additional points outside the AABB are detected to be in this object. The surface mesh of the object can also be extracted explicitly through Poisson surface reconstruction [Kazhdan et al. 2006]. When the object is detected to be a duplicate of a previously detected object, we record the reference ID to the previous object and the relative transformation matrix.

3 Plane/Object Labeling

Our plane/object labeling process achieves segmentation of planes and objects in the volume by labeling on the local raycasting depth map in each frame and mapping the resulting labels back into the global volume. The depth pixels of each new frame are labeled with plane/object IDs, under the guidance of existing label information in the volume. We call the result an *ID map*, which is then merged into the volume data. We adopt the raycasting depth map as input for labeling, since it is smoother and more reliable than the raw depth map. For clarity, we refer to a pixel as a *plane pixel* (*object pixel*, resp.) if it is labeled with a plane (object, resp.) ID.

The initialization of the ID map is concurrent with the raycasting step of the depth map in KinectFusion – we retrieve the ID stored in the voxel hit by the ray from each pixel. However, this ID map is incomplete in most cases, because some newly fused voxels are not covered by existing planes/objects (e.g., the black pixels in the second image at the top left corner of Fig. 5).

Given the raycasting depth map from the fused voxel data, our algorithm assigns plane IDs to pixels before handling object IDs, since objects are often separated by planar regions. Moreover, for either planes or objects, the labeling procedure first examines whether the pixels should be labeled with existing IDs so as to allow the expansion of partially completed scans. Therefore, the labelling order

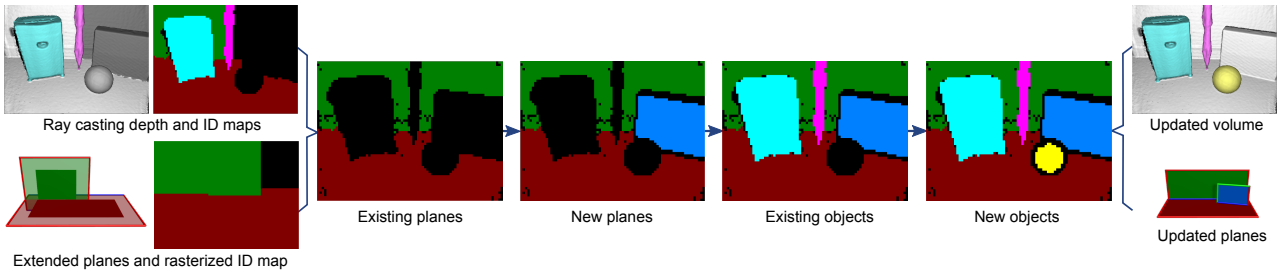


Figure 5: *Plane/object labeling performed on ID map. Starting from raycasting depth and ID maps enhanced by the rasterized ID map of extended planes, we detect existing planes, new planes, existing objects and new objects sequentially. The labeling result is subsequently used to update the volume data and the geometric description of the planes. By rasterizing extended planes, the region growing method can merge two disconnected regions, such as the wall (green segment) separated by the umbrella (pink segment).*

is: existing planes, new planes, existing objects and new objects, as shown in Fig. 5.

3.1 Plane Labeling

Existing planes. In this step, we determine for each pixel on the ID map whether it belongs to an existing plane. The algorithm starts with checking whether the initialized ID map is consistent with the currently captured depth map. Precisely, we unlabel a plane pixel when its distance to the plane indicated by its ID exceeds a preset threshold ($10mm$ in our implementation).

Next, for all non-planar pixels (including unlabeled pixels and object pixels), we test whether they lie on an existing plane and obtain their associated IDs in that case. For efficient implementation, we rasterize the extended polygon of all planar regions visible in the current frame into a temporary depth and plane ID map to serve as references for the labeling process. Details on planar region extension can be found in Sec. 6. The non-planar pixels are scanned in two passes to obtain their plane IDs. In the first pass, we compare the raycasting depth and the rasterized depth of the same pixel. If their depth difference is below a threshold ($10mm$ in our system), we further check the distance of the raycasting point to the corresponding plane and the angle between the normal of the raycasting point and the plane normal. When both are below the respective thresholds ($10mm$ and 3° in our system), we set the label of this pixel with the plane's ID on the rasterized ID map. In the second pass, we expand the labeled area using a breadth-first search. The per-pixel condition is identical to the first pass except that the angle threshold (20°) is greater.

The two passes serve different purposes in labeling. The first pass is to rapidly cover large unlabeled regions. Since the rasterized planar regions are supersets of the currently observed data, we label the pixel with the ID of the plane even if it is not connected to labeled pixels on the raycasting ID map. The second pass is used to grow the labeled area, relying more on spatial adjacency than on the similarity of the normals, since the raycasting normal estimate is usually noisy.

New planes. After pixels of existing planes are labeled on the ID map, we form new planes from the remaining unlabeled pixels and object pixels, using a method similar to the one in [Lee et al. 2012]. The pseudocode of this process is given in Algorithm 1. We choose a region growing method, which is fast and easy to implement, to label new planes for the purpose of online structure analysis, because random sample consensus (RANSAC) and Hough transform based methods [Schnabel et al. 2007; Hulik et al. 2014] have a relatively heavy computational load.

In Line 1 of Algorithm 1, highly non-planar pixels are eliminated from the region growing. The planarity of a pixel $p_{(i,j)}$ is estimated from its four neighboring triangles, formed by the pixel and four neighboring pixels, as described in [Lee et al. 2012]. We denote the unit normal of the triangle formed by the center pixel $p_{(i,j)}$ and two neighboring pixels $p_{(i-1,j)}$ and $p_{(i,j-1)}$ as n_0, and n_1, n_2, and n_3 for the other three neighboring triangles. Then the violation of planarity can be estimated using $\kappa = \sum_{i=0}^{3} |n_i| - \left|\sum_{i=0}^{3} n_i\right|$.

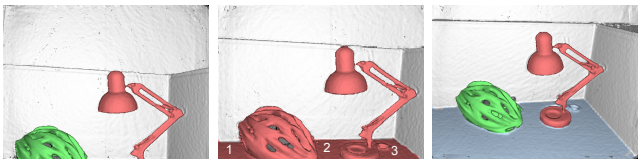
Algorithm 1 Label New Planes

```
1:  Label highly non-planar pixels as "processed"
2:  while pixels without a plane ID and "processed" mark exist do
3:      select seed pixel p_(i,j)
4:      region growing to pixel set P
5:      if (Area(P) > threshold) then
6:          Create new plane with P
7:          if Overlap(P, Object_k pixels) then
8:              Set Object_k unlabeled
9:          end if
10:     else
11:         mark P as "processed"
12:     end if
13: end while
```

This measurement κ is similar to an estimate of the absolute curvature (sum of the absolute values of the principal curvatures), and is only 0 when this neighborhood is flat. If κ is above a threshold (0.025 in our system), or if none of the four neighboring pixels has an observed depth value (no iso-surface point along the ray cast from the pixel), we mark the center pixel as "processed".

Next, we iteratively process all pixels without a plane ID or a "processed" mark. In each iteration, we randomly select a seed pixel from those pixels, and create a temporary plane region for it. Then we use a four-way connected flood fill algorithm to grow the connected region from the seed pixel using breadth-first search. If the neighboring pixel being examined is not labeled with a plane ID yet and the angle difference between its normal and the plane's normal is below a threshold (3° in our system), it is added to the planar region. We set the initial plane normal as the estimated normal of this pixel in the raycasting map, and it is updated to the average normal of all pixels merged into this plane. The flood fill terminates when the plane cannot expand further.

If the area of the labeled planar region exceeds a preset threshold ($0.05m^2$ in our system), a new plane ID is allocated and all of its pixels are labeled with this ID. Otherwise, these pixels are assumed to be part of a non-planar region, and marked as "processed" to prevent repeated testing. This process terminates when no more seed pixels exist.



(a) Two objects labeled (b) Two objects merged (c) Plane detected, two objects relabeled

Figure 6: *Example plane/object conflict resolution. The scene is scanned from top to bottom. (a) the first frame. (b) as the desktop plane is partially observable, it is fragmented (labeled with 1, 2, 3), with each piece treated as parts of potentially non-planar objects, based on the area threshold. This results in a single connected object. (c) as the capturing process continues, the desktop plane is big enough and detected, existing objects on this plane will be labeled from scratch and become individual objects.*

Plane/object conflict resolution. Note that a newly detected plane may contain pixels that are already labeled as non-planar objects. This situation arises if the observed area of a plane did not meet the size requirement and was thus identified as multiple non-planar pieces, as shown in Fig. 6. In this case, we delete these existing non-planar objects using the following procedure. The parts of the objects covered by the new plane will now be merged and identified as a single planar region. The non-planar objects are removed from the global structure, and their pixels outside the planar region will become unlabeled and thus candidates for relabeling in the non-planar object labeling step.

3.2 Object Labeling

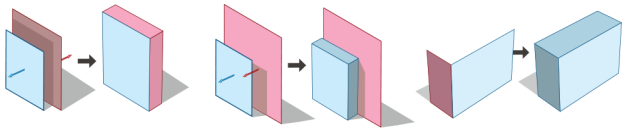
Existing objects. After plane labeling, the depth map is segmented into disconnected subregions. We are now able to label non-planar objects according to the subregions. Similar to the plane labeling procedure, we use breadth-first search on pixels that are already labeled as objects. Since objects can be of any shape, the stopping criterion of flood fill is simply that the difference in depth between the pixel being examined and the neighboring pixel is above a threshold ($3cm$ in our system), or that the latter is already labeled. An example of the existing object labeling result is shown in the existing objects image in Fig. 5.

New objects. We form new objects on the ID map when existing planes, new planes, and existing objects are already labeled. We choose an unlabeled pixel as a seed pixel and perform flood fill, with the same stopping criterion as in existing object labeling. If the number of pixels labeled in the flood fill is big enough to preclude noise-like regions on the map, a new object is detected (for example, see the new objects image in Fig. 5). Otherwise, these pixels are not considered as forming an object and will be "omitted". This process continues until all pixels are labeled or "omitted".

Volume Update. Once the planes and objects are labelled, we can update the volumetric data structure. To do this, we simply write the newly calculated ID of each pixel back into its corresponding voxel, as this voxel index for the pixel was stored during raycasting depth map generation. See, e.g., Fig. 1 (top right) and Fig. 13 (right).

4 Scene Construction

In the scene construction stage, we use the calculated ID map from the labeling stage to amend the structural information. We refine the plane equation of each planar object with new pixels of the same ID, its planar boundary with new trimming points, and each non-planar object with newly observed pixels of the same ID. We then use some heuristics that are applicable to common indoor scenes to enforce configurational relations among planes, such as parallelism or orthogonality. Duplicate objects are also detected in this stage.



(a) thin plate (b) extrusion (c) box

Figure 7: *Rectilinear structure heuristics.*

4.1 Plane Construction

If a plane ID is labeled on any new points in the raycasting map, we append these newly detected points to the existing point list of that plane. To control the memory footprint, we subdivide the planar region into a set of cells using a uniform 2D grid and store only one point (the one that is closest to the cell centroid) inside each cell. Note that the trimming points are treated separately with the same procedure to avoid accuracy degradation of the boundary edges. We then fit the equation of the plane with the updated point list using principal component analysis (PCA).

If some adjacent planar regions have nearly identical equations of the plane, we merge them into a single region. Specifically, we merge their point lists and trimming point lists, and recalculate a combined boundary. When updating the voxels using these planes, they will be reassigned a common ID.

Rectilinear structure heuristics. For each pair or group of planes with no known relationship, we check the conditions of the following rules and trigger them accordingly. In all the following heuristics, we use an error tolerance of 3° for normals being parallel or orthogonal.

- Thin plate: when two planes have opposite normals, overlapping 2D boundary polygons, and small distance ($<10cm$), they form a board structure.
- Extrusion: when two planes have similar normals, overlapping 2D boundary polygons, and small distance ($<10cm$), they form a thin extrusion.
- Box: when all plane boundaries are rectangles, their pairwise intersections form convex 90° dihedral angles, and length differences among parallel intersection edges are smaller than $10cm$, they form a rectangular box.
- Orthogonality: when two planes intersect at a near 90° dihedral angle, they are assumed to be orthogonal to one another.

The effects of these rules are illustrated in Fig. 7. We now elaborate on the processing when each rule is triggered. For *thin plate*, we project the point list and trimming point list of one plane onto the other and recalculate the boundary. The thickness of the thin plate is set to be the distance between these two planes. These planes will then be displayed as a thin plate in our system. For *extrusion*, the thickness of the extrusion is the distance between the two planes. The plane at the front will be displayed as a board attached to the back panel. For *box*, we cluster rectangle pairs satisfying the box condition to form maximal groups. The local coordinate frame of this box is calculated by rotating the coordinate axes to fit normals of these planes, weighted by the area of each plane in the group. Then the box is calculated simply as the oriented bounding box of

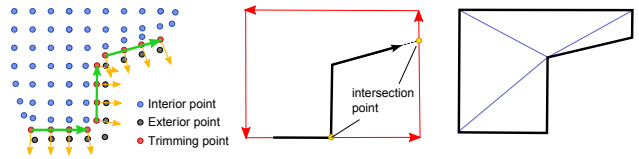


Figure 8: *Calculation of the boundary of a planar region. (a) Calculate the normal of trimming points, then detect trimming edges. (b) Link trimming edges to form curves (polylines), fit OBB boundary to trimming edges, and then calculate intersection point of the curve and OBB boundary. (c) Extract polygon and triangulate it.*

these planes. For *orthogonality*, we consider the planes orthogonal to each other and enforce the dihedral angle to be 90°, similar to the optimization used in detecting boxes, i.e., we use the group of planes to create a local coordinate system and force the planes to be axis-aligned in the local frame; in addition, when the normal of one plane is almost axis-aligned, we force the plane normal to be axis-aligned. Note that the coordinate system is adjusted so that a large number of planes are axis-aligned, as discussed in Sec. 6.

4.2 Plane Boundary Calculation

The goal of this step is to approximate the plane boundary by a polygon. Following the rectilinear structure assumption, we initialize each partially observed planar region to its OBB, which is important in estimating the boundary in the occluded regions (see Fig. 4). Then, we detect *trimming edges* to get a tight polygonal approximation of the planar region according to the visible part of the region. Finally, we snap together edges of adjacent planar regions to remove gaps. Fig. 8 illustrates the general process of polygonal approximation.

Exterior and trimming points. Trimming edges consist of trimming points, which are calculated based on exterior points. As with interior points, only one exterior or one trimming point is recorded within each cell of the plane, so they are much sparser than the pixels. An exterior point is selected from depth pixels predicted to be on the planar region by plane extension, but observed to be off the plane due to depth inconsistency (shown as black points in Fig. 4). More precisely, when the raycasting depth of a pixel is $5cm$ greater than the rasterized depth ($1cm$ if on another plane), it is confirmed to be an exterior point; when the depth is smaller, it is potentially still part of the planar region occluded by other objects. Then, we select trimming points from the *boundary* pixels, i.e., those labeled with the ID of this planar region and adjacent to those with other

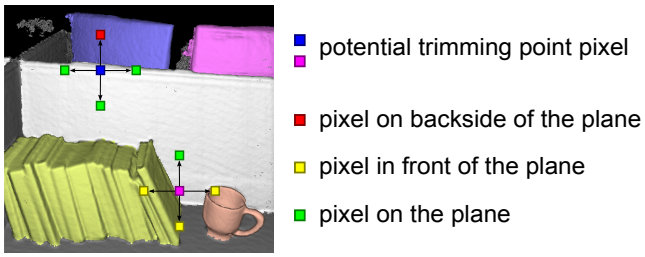
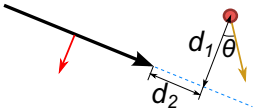


Figure 9: *Trimming point labeling: two potential trimming point pixels shown on the boundary pixels of the highlighted white plane. The blue pixel is tagged as a trimming point because one of its four neighbors is on the backside; whereas the magenta pixel only has off-plane neighbors in front of the plane, so it cannot be confirmed based on the rule.*

IDs, by restricting to those between interior and exterior points to avoid false positives. In our experiments, it suffices to check just the four pixels located at $5cm$ distance (estimated from the depth and the FOV) along axial directions, as illustrated in Fig. 9.

Trimming edges. We fit trimming edges to trimming points, roughly following [Arikan et al. 2013]. We first calculate the in-plane normal of each trimming point (through 3×3 Sobel operator applied to the characteristic function of the region on the interior/exterior grid), then cluster these points into trimming edges, and finally connect these edges to form trimming curves.

As each trimming point can be regarded as a short line segment centered at the point with the estimated in-plane normal, whether it can be concatenated to a line segment is measured by the relation of their end points and their normals. We use a Sobolev-type norm similar to [Cohen-Steiner et al. 2004], $\text{dist} = k_1 d_1 + k_2 d_2 + k_3 \theta$, a weighted average of d_1, the distance to the line, d_2, the distance of its projection to the line segment, and θ, the angle between the normals. In our current implementation, we use $k_1 = 1$, $k_2 = 0.5$ and $k_3 = 3cm$ with θ in radians.



We start clustering the points to trimming edges from an unprocessed seed point, and repeatedly add the closest point and update the edge until no more points are close enough ($<4cm$). Short trimming edges ($<20cm$) are discarded in this step. We then use a K-means-like algorithm to improve the fitting, i.e., we alternately associate each point to the closest edge (if distance $<4cm$) and update each edge according to the associated points, until convergence. The resulting trimming edges are connected to form *trimming curves* (polylines) by snapping pairs of end points (within $10cm$ distance). Note that the curve may contain only one trimming edge or form a loop. Temporal continuity of trimming edges are attained by revising existing trimming edges with updated trimming points.

Polygon creation. After calculating OBB in the local coordinate frame aligned to the longest trimming edge, we insert each trimming curve into the current polygon in the following order: (1) a closed trimming curve; (2) a trimming curve with edges that replace OBB edges; (3) a trimming curve with end points within a preset distance from existing polygon boundary curves ($20cm$ in our system). In the third case, we extrapolate the trimming curve to reach the current polygon, and insert the curve into the polygon edge list to form the closed polygon. The final trimmed polygon is triangulated for rendering. This method is able to create curved boundaries (the desk in Fig. 11), concave boundaries (the cross shaped desk in Fig. 1) or even regions with holes (bookcase in Fig. 1).

4.3 Object Construction

Each object is implicitly encoded in the volumetric data structure through its ID. The only update to each individual object is the recalculation of its bounding box. For objects detected to be merging, we replace their IDs with a single ID. The condition is similar to the merging of planar shapes, i.e., when some of their pixels are adjacent in the ID map with similar depths (difference $< 2cm$), they form a single object, as shown in Fig. 6. We recalculate the AABB of the merged object. If one of the original objects already has a surface mesh in the data structure, we recalculate the surface mesh.

Duplicate object identification. If an object matches an existing object with an extracted surface mesh with a low (partial) matching error, the mesh of the existing object is displaced as a virtual object

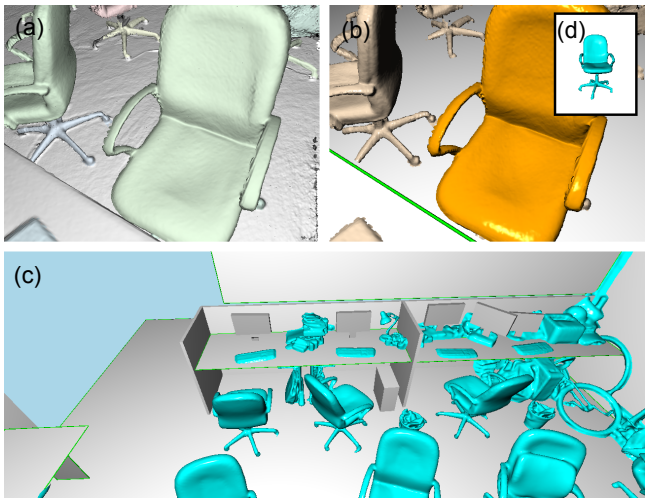


Figure 10: *User interface of our system. (a) Raycasting map; (b) Structure analysis of the current frame; (c) Reconstructed global scene; and (d) Duplicate object hint.*

in our graphical interface, allowing the user to decide whether to keep scanning based on how well the perceived match of the complete shape is. If the user is content with the matching result, we record the ID of the matching object and the transformation matrix between them in the data structure of the recent object. The object will be rendered as the transformed matching object from then on. The matching is calculated by performing ICP between the depth pixels of an object labeled in the current frame and the surface mesh of the existing object. Owing to the availability of the floor information, we found it effective to initialize the ICP by aligning the centers of the objects and rotating one of them by eight different angles around the vertical axis and then pick the best match resulting from these initializations.

Refined depth creation. With all the planes, objects and voxels updated, we are ready to prepare the structurally enriched depth map used for ICP in the next KinectFusion stage. For each pixel on the raycasting depth map, labeled as part of a plane, we calculate the intersection point of the ray from this pixel and that plane. Thus, the more accurate depth value based on this intersection point, instead of the original raycasting depth, is stored into the prediction depth map for the next frame.

5 User Interaction

As an online system, we need user assistance to confirm the structure analysis result in difficult situations, for instance, the correctness of the duplicate object identification result. However, it will be tedious for the user to confirm the result using the mouse/keyboard UI since she/he needs to operate both the depth camera and mouse in such a design. We address this issue by using the Kinect camera itself as an interactive device to interact with the system, and it is implemented based on the detection of persistent camera pose. We also provide instantaneous visual feedback by displaying the KinectFusion output, the analysis result of the current frame, and the online reconstructed overall indoor scene simultaneously.

Visualization. We maintain three views on our graphical interface as shown in Fig. 10: window (a) shows the raycasting result from KinectFusion; window (b) renders the reconstruction in the camera view; and window (c) is the bird's-eye view of the whole reconstructed scene. If a duplicated object is detected, window (d) pops up to indicate the potential duplicate.

In the reconstruction window (b), detected planes and objects are rendered in different colors, and the current *object of interest* is highlighted. The object of interest is defined as the detected object that has the largest number of labeled pixels in the current frame. If a duplicate object is detected, this duplicate will be displayed at the target position in the reconstruction window and the image of this object will be shown explicitly in window (d) as a hint to the user.

The camera orientation of the bird's-eye view is the average sensor orientation tilted to 45° downward, and the camera position is $4m$ behind the average sensor location. It is updated very 500 frames if the average orientation is changed by at least 45° or the position is shifted by at least $1m$.

Object selection. When a duplicate is found for the current *object of interest* and visualized on the screen during the scan process, the user can hold the camera still for one second to enter the *menu mode*. In the menu mode, a 30° *left tilt* of the camera around the optical axis indicates *confirmation* of this duplication, while a 30° *right tilt* indicates *rejection*. Any other movements take the system out of the menu mode. Once rejected, this duplication candidate will not be tested again.

If no duplication is detected, the user can also enter the *menu mode* and *tilt* the Kinect camera to extract the mesh of the current *object of interest*. Objects with generated meshes will serve as candidates for duplication detection. In a common indoor scene, similar but not exactly the same shapes abound, such as chairs of different heights or different types of keyboards. If the user wants to preserve the subtle differences, he/she could reject the duplicate suggestions and extract a new mesh to make it a new type of object in the system.

6 Implementation Details

In this section, we describe some important implementation details of our system.

Voxel hashing. In order to exploit spatial sparsity of the volumetric representation, we use the KinectFusion algorithm based on voxel hashing as described in [Nießner et al. 2013]. The implicit representation of the entire scene uses a TSDF, where $8 \times 8 \times 8$ voxel blocks are allocated in a narrow band around the surface and accessed via a hash table. We extended this method by incorporating an object identifier into each voxel, without incurring additional memory – we change the TSDF variable type to half float, which is considered accurate enough for TSDF according to [Newcombe et al. 2011], keeping the overall voxel size intact (8 bytes). The voxel side length is $8mm$ in our system.

Coordinate system. In a common indoor scene, abundant rectilinear structures exist, in particular, the orthogonality and parallelism among the ground, most walls, and furniture. We maintain a coordinate system maximizing the number of planes aligned to axes. Initially, given the first raw depth frame, the coordinate system is chosen to be centered at the Kinect sensor, with its Y-axis being vertically up (opposite of the Kinect gravity sensor measurement) and its X-axis being the cross product of Y and the optical axis of the camera. We also request the user to hold the camera horizontally facing a vertical plane to obtain a decent initial guess. As planes are detected and updated with each frame, we set the ground to be the lowest horizontal plane (with a vertically up normal direction). Next, we interpret the largest planar region facing the initial optical axis of the camera as a wall. When the sizes of both the ground and wall reach a threshold, we update the coordinate system based on the normals of these two planes. The projection of the current

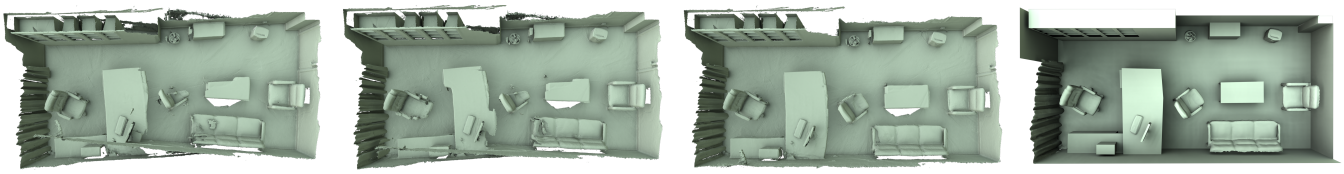


Figure 11: *Office room. From left to right: KinectFusion result; scanned data with our online structure analysis, but without the orthogonality heuristic; scanned data with the orthogonality heuristic; and our reconstructed scene.*

origin on the ground serves as the new origin.

Planar region extension. When labeling existing planes, we rasterize the ID map of each extended plane by offsetting OBB and *intersection* edges along their (in-plane) normals. We do not extend the region across trimming edges, because they are based on actual data. In contrast, the OBB edges are only calculated from currently observed interior points. We also extend over *intersection* edges so that the planes may extend over thin features, such as the partition plates of a bookcase. Note that this extension is only used for the rasterized ID maps but not for the reconstructed planar region geometry. For large flat regions, such as the floor and walls, we extend $40cm$ over each edge, and $4cm$ in other cases.

Surface mesh creation. The surface mesh for an object is constructed by using the Poisson surface reconstruction algorithm [Kazhdan et al. 2006], with the point cloud extracted from the volumetric data as input through raycasting. We use the Poisson surface reconstruction to produce smooth closed surfaces, which represent isolated objects in our online recognition as well as in downstream applications. From each voxel on the faces of the bounding box of the object, we shoot a ray from the center along the negative gradient of the distance field, which is approximately the surface normal. If the ray hits the zero-isosurface and the ID of the voxel belongs to the object, we insert the intersection point to the point cloud. The procedure is implemented on the GPU with one thread per ray, and the points are collected in an array following the same fashion as the voxel block selection in [Nießner et al. 2013].

ID management. In our implementation, we allow different IDs stored in the volumetric data structure to represent the same planar or non-planar object so as to handle merges efficiently. This type of treatment is typical in efficient connected component computation through the union-find algorithm. We simply maintain a table linking the non-unique IDs in voxels with the unique IDs (UIDs) of the planar regions and non-planar objects. With the lookup table, we can simply translate the ID map in raycasting to a UID map. When an object is removed, we set the IDs used for this object to link to a special UID. When several objects are merged into a single object, the IDs of these objects are all linked to the new object UID. When the UID map is stored back into the volume data structure, it is translated back to one of the IDs linked to it. Each time a new planar or non-planar object is created, we allocate a UID to encode it in the volume grid. With this ID management, we avoid frequent updates for IDs. Only the lookup table needs to be maintained.

Weighted ICP. We use point-plane ICP between the raw depth map and the refined raycasting depth map for camera pose estimation. Planar objects are considered to be more accurate than non-planar objects, since planes contain the rectilinear structural information. The larger the planar region is, the more accurate and robust the associated plane equation is. Thus, we assign higher weights for pixels with depth determined by plane equations during the ICP process. The weight is proportional to the area of the planar region.

GPU and CPU. The voxel data is stored on graphics memory, and the KinectFusion stage is run on the GPU. We rasterize the prediction planes using a shader which encodes position, normal and plane ID at each pixel. Then we transfer the raycasting depth and ID map and the rasterized image to the main memory. Planar region and non-planar object data structures are stored in the main memory. We perform planar region and non-planar object labeling and scene update on the CPU. Then we transfer the new ID map and the generated depth map to graphics memory to be stored into voxels and used for ICP in the next frame. It is also possible to port some other operations to the GPU for further performance improvement.

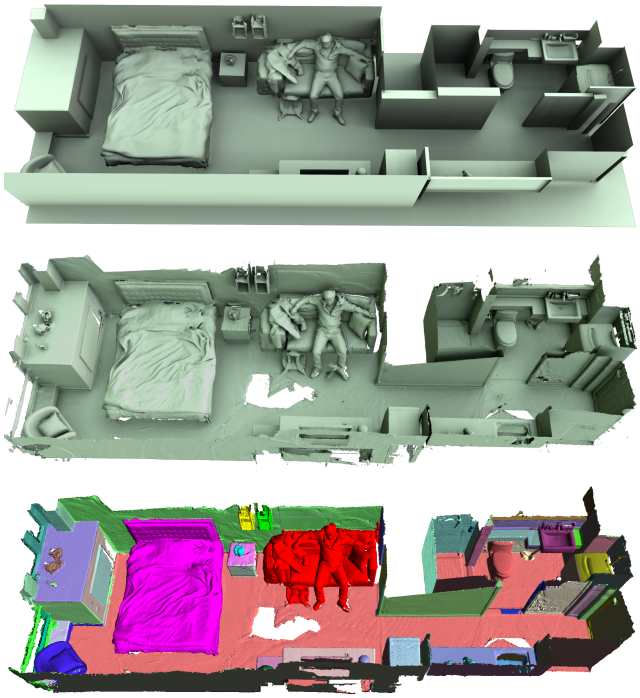


Figure 12: *An apartment scanned in 15 minutes, including a sleeping area, a living area, and a bathroom. From top to bottom: reconstructed scene, KinectFusion with online structure analysis, volume labeling result.*

7 Experimental Results

We implemented our system on a desktop PC with an Intel I5-4430 CPU and an Nvidia GeForce GTX 780 Ti graphics card. It was tested on five indoor scenes: an office, a meeting room, an apartment, a corridor and a lab. See our accompanying video for the online structure analysis procedure.

Performance. The overall performance of our system is about $20fps$, with $35ms$ for the KinectFusion part and $14ms$ for our online analysis per frame, including ray cast ID map ($2ms$), rasterize ID map ($2ms$), plane/object labeling ($2ms$), scene update ($2ms$),

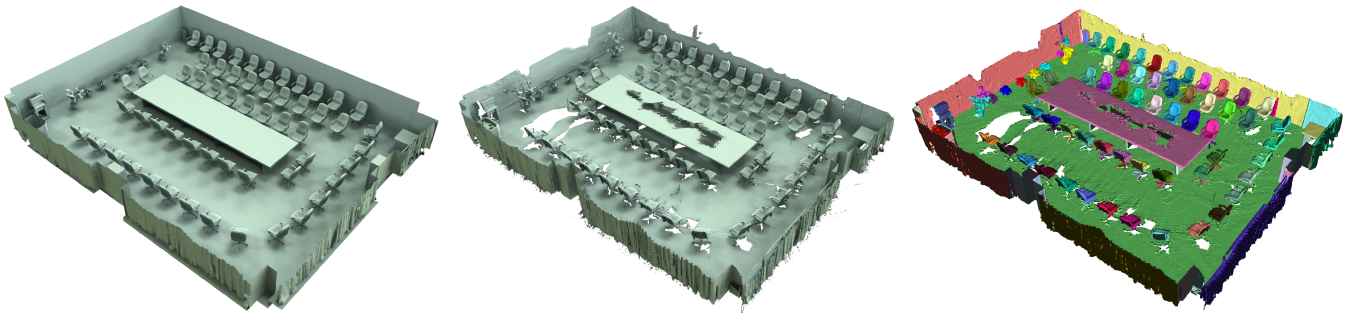


Figure 13: *Meeting room. From left to right: reconstructed scene, KinectFusion with online structure analysis, volume labeling result (one unique color per plane/object).*

volume update ($1ms$) and rendering ($5ms$). The volume structure is stored on the GPU and streamed between CPU and GPU when scanning large-scale scenes. The plane/object data are stored in the CPU memory, thus it does not interfere with the GPU processing of KinectFusion.

Plane structure. Fig. 11 demonstrates the improvement to the reconstruction quality with our plane expansion operation. Due to occlusion, there are holes on the ground and walls of the office scene in the raw data, since those regions might be hard to cover completely while scanning. In contrast, with our plane data, large planar regions, such as walls and the ground, are extended automatically to these missing areas as an extrapolation of the most common shape. This simple procedure is particularly effective for indoor scene reconstruction applications. Another example is the reconstruction of the council table in the middle of the meeting room (Fig. 13). While original ICP based fusion algorithms often suffer from severe drifting for such plane sections with weak textures, our algorithm effortlessly reconstructed it through plane structure analysis. For non-workplace environments, such as the apartment in Fig. 12, which includes a sleeping area, a living area, and a bathroom, the benefits of employing plane structures are also obvious.

Repeated objects. We evaluated how the repeated objects analysis can help to reduce the scanning time. For the meeting room scene, there are 56 chairs of the same shape (see Fig. 13). It would have taken extremely long if we scanned each chair individually. Instead, we only completely scanned one chair as a template, and populated the scene with instances of the template automatically in place of the detected duplicates. With such duplication detection, we were able to scan this room within 40 minutes.

Drifting relief. The existence of planar structures can be leveraged to greatly reduce drifting in the fusion process. As illustrated in Figure 11, the walls of the reconstructed office scenes are faithfully modeled and closed into a seamless loop automatically with our system. The parallelism and orthogonality relations between planes are also well preserved. For comparison, the reconstruction result from direct depth data fusion is shown on the left of Fig. 11.

In Fig. 15, we captured a long corridor by using KinectFusion with voxel hashing [Nießner et al. 2013] at $30fps$. The length of the captured part of the corridor is about 25 meters. Drifting in KinectFusion camera tracking is easily perceptible at this scale. Since our algorithm detects planes explicitly, which helps tremendously in rectifying the camera position, we can greatly reduce drifting without compromising efficiency. The constraints provided by axis aligned planar regions, especially the large ones, can keep the entire scene nearly distortion-free, as shown in the second row of Fig. 15.

Large scale scenes. The scalability of our system is demonstrated on two large scale scenes: a meeting room ($140m^2$) and a lab ($100m^2$). There are numerous types of objects in these scenes, especially the lab scene. Our online structure analysis can identify large planar regions, segment isolated objects, and recognize duplicates successfully to improve the reconstruction quality and efficiency.

Parameters. Aside from data structure related design choices to fit the GPU architecture, all the parameters in our system describe physical quantities. For example, in the plane labeling process, the threshold values to label a point on a plane, describes the maximum distance to the plane and a maximum deviation angle of the normal; in rectilinear structure heuristics, angle threshold is used to identify parallel planes. The values are mainly influenced by the accuracy of the depth sensor, and are thus almost independent of the scene. In fact, we have reported the values of all the parameters used in our system based on Microsoft Kinect depth sensors. All parameters are fixed in our tests, except in one case. The distance threshold is increased from $10mm$ to $30mm$ for labeling existing planes (Sec. 3.1) when scanning the conference room (Fig. 13). The change is induced by the scanning distance. In most cases, the distance of the depth sensor to most planes/objects reaches as low as around 1 meter during the scan, at which distance the noise level of depth map is relatively small. However, to efficiently scan the meeting room, the Kinect sensor remained over 2 meters away from the wall. Note that this change is proportional to the standard deviation error at 1 and 2 meters, as reported in [Khoshelham and Elberink 2012].

Limitations. Our system excels in reconstructing indoor scenes where planes are major shape primitives. In scenes without (many) planes, such as a garden, our system simply reduces to a typical KinectFusion system. Our system will also fail to detect any plane severely occluded by objects so that the detected area is below the threshold, as shown in Fig. 14 (a). Conversely, planarity may be enforced on non-planar shapes, leading to missing near flat features, such as supports of computer displays. Similarly, boundary trimming may remove geometric details, such as chamfers or tiny boundary features. Another issue in labeling is that objects in

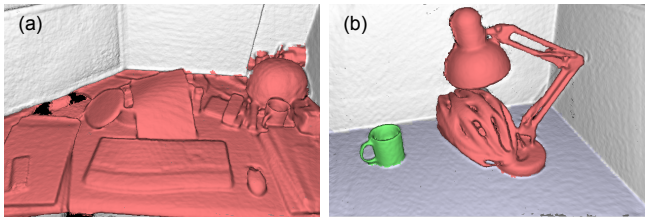


Figure 14: *Failure cases: (a) the top plane of the cluttered desk is undetected, (b) objects in contact are not isolated.*

Figure 15: *Comparison of drift. Top: KinectFusion; Bottom: ours. Note that the ground bends without the plane information.*

contact will be treated as a single one, since the depth continuity criterion alone cannot isolate them at the resolution of the volume data, as shown in Fig. 14 (b). We believe these issues may be alleviated through incorporating matching of detailed templates from a database. Currently, we tested the effectiveness by matching a display template to recover the display supports; extensions are left as future work.

While our system does not introduce explicit loop closure handling, planarity and orthogonality constraints will greatly reduce camera tracking drift and bending of flat regions, resulting in seamless loop closure for scenes with medium-sized loops (e.g., the office in Fig. 11, and even the meeting room in Fig. 13 with a $50m$ long loop) or with large loops that can be decomposed into small loops (e.g., the lab in Fig. 1). For a scene captured with a single extremely large loop, we expect loop closure to remain a problem due to accumulated errors.

8 Conclusion

We have presented an indoor-scene reconstruction system producing structured models in real-time, through an online structure analysis. In contrast to the state-of-the-art systems, we generate, at interactive rates, processed models instead of raw point cloud data, thus enabling operations difficult to perform in the post-processing stage. The global structure gradually refined during the scan effectively reduces camera drift, enhancing both the robustness and accuracy of the reconstruction. Moreover, the prompt feedback with structural information allows the user to skip repeated objects and focus the scan on ambiguous regions.

Our framework for online structure analysis allows other primitive shapes and heuristics to be incorporated, opening up a multitude of opportunities for further research. For future work, we believe loop closure assisted by structure analysis is a worthwhile topic, since such techniques are desirable in high-quality large scene reconstruction. More accurate object segmentation methods could be used to separate objects in contact within cluttered regions. Template matching techniques can also be employed to improve both object segmentation and geometric detail preservation. Texture mapping may also help visually recover the geometric details missing from the captured scene.

Acknowledgements

This research was supported in part by NSF of China (No. 61272305, 61322204), the National Program for Special Support of Eminent Professionals of China, and NSF IIS 0953096.

References

ARIKAN, M., SCHWÄRZLER, M., FLÖRY, S., WIMMER, M., AND MAIERHOFER, S. 2013. O-snap: Optimization-based snapping for modeling architecture. *ACM Trans. Graph. 32*, 1 (Feb.), 6:1–6:15.

ATAER-CANSIZOGLU, E., TAGUCHI, Y., RAMALINGAM, S., AND GARAAS, T. 2013. Tracking an rgb-d camera using points and planes. In *Proc. CDC4CV*, IEEE Workshop.

BALLARD, D. H. 1981. Strip trees: A hierarchical representation for curves. *Commun. ACM 24*, 5, 310–321.

BIBER, P., AND STRASSER, W. 2003. The normal distributions transform: a new approach to laser scan matching. In *Proc. IEEE/RSJ IROS*, IEEE, 2743–2748.

BISWAS, J., AND VELOSO, M. 2012. Planar polygon extraction and merging from depth images. In *Proc. IEEE/RSJ IROS*, 3859–3864.

CHEN, J., BAUTEMBACH, D., AND IZADI, S. 2013. Scalable real-time volumetric surface reconstruction. *ACM Trans. Graph. 32*, 4 (July), 113:1–113:16.

COHEN-STEINER, D., ALLIEZ, P., AND DESBRUN, M. 2004. Variational shape approximation. *ACM Trans. Graph. 23*, 3 (Aug.), 905–914.

DOU, M., GUAN, L., FRAHM, J.-M., AND FUCHS, H. 2013. Exploring high-level plane primitives for indoor 3d reconstruction with a hand-held rgb-d camera. In *Computer Vision - ACCV 2012 Workshops*, vol. 7729. 94–108.

DOUGLAS, D. H., AND PEUCKER, T. K. 2011. *Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature*. John Wiley & Sons, Ltd, 15–28.

DU, H., HENRY, P., REN, X., CHENG, M., GOLDMAN, D. B., SEITZ, S. M., AND FOX, D. 2011. Interactive 3d modeling of indoor environments with a consumer depth camera. In *Proc. of Ubicomp*, 75–84.

FAHN, C.-S., WANG, J.-F., AND LEE, J.-Y. 1989. An adaptive reduction procedure for the piecewise linear approximation of digitized curves. *IEEE Trans. PAMI. 11*, 9, 967–973.

FURUKAWA, Y., CURLESS, B., SEITZ, S. M., AND SZELISKI, R. 2009. Manhattan-world stereo. In *Proc. CVPR*, IEEE, 1422–1429.

FURUKAWA, Y., CURLESS, B., SEITZ, S. M., AND SZELISKI, R. 2009. Reconstructing Building Interiors from Images. In *Proc. ICCV*, IEEE.

GLOVER, F., AND LAGUNA, M. 1997. *Tabu Search*. Kluwer Academic Publishers.

HARTLEY, R. I., AND ZISSERMAN, A. 2004. *Multiple View Geometry in Computer Vision*, second ed. Cambridge University Press.

HULIK, R., SPANEL, M., SMRZ, P., AND MATERNA, Z. 2014. Continuous plane detection in point-cloud data based on 3d hough transform. *Journal of Visual Communication and Image Representation 25*, 1, 86 – 97.

IZADI, S., KIM, D., HILLIGES, O., MOLYNEAUX, D., NEWCOMBE, R., KOHLI, P., SHOTTON, J., HODGES, S., FREEMAN, D., DAVISON, A., AND FITZGIBBON, A. 2011. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proc. of ACM UIST*, 559–568.

KAZHDAN, M., BOLITHO, M., AND HOPPE, H. 2006. Poisson surface reconstruction. In *Proc. SGP*, Eurographics, 61–70.

KHOSHELHAM, K., AND ELBERINK, S. O. 2012. Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors 12*, 2, 1437–1454.

KIM, Y. M., MITRA, N. J., YAN, D.-M., AND GUIBAS, L. 2012. Acquiring 3d indoor environments with variability and repetition. *ACM Trans. Graph. 31*, 6, 138.

KOLESNIKOV, A. 2003. *Efficient Algorithms for Vectorization and Polygonal Approximation*. University of Joensuu.

LATECKI, L. J., AND LAKMPER, R. 1999. Convexity rule for shape decomposition based on discrete contour evolution. *Computer Vision and Image Understanding 73*, 441–454.

LEE, D. C., HEBERT, M., AND KANADE, T. 2009. Geometric reasoning for single image structure recovery. In *Proc. CVPR*, IEEE, 2136–2143.

LEE, D. C., GUPTA, A., HEBERT, M., AND KANADE, T. 2010. Estimating spatial layout of rooms using volumetric reasoning about objects and surfaces. In *NIPS*, 1288–1296.

LEE, T.-K., LIM, S., LEE, S., AN, S., AND OH, S.-Y. 2012. Indoor mapping using planes extracted from noisy rgb-d sensors. In *Proc. IEEE/RSJ IROS*, 1727–1733.

LI, Y., WU, X., CHRYSATHOU, Y., SHARF, A., COHEN-OR, D., AND MITRA, N. J. 2011. Globfit: Consistently fitting primitives by discovering global relations. *ACM Trans. Graph. 30*, 4 (July), 52:1–52:12.

MATTAUSCH, O., PANOZZO, D., MURA, C., SORKINE-HORNUNG, O., AND PAJAROLA, R. 2014. Object detection and classification from large-scale cluttered indoor scans. *Computer Graphics Forum*.

NAN, L., XIE, K., AND SHARF, A. 2012. A search-classify approach for cluttered indoor scene understanding. *ACM Trans. Graph. 31*, 6, 137.

NEWCOMBE, R. A., IZADI, S., HILLIGES, O., MOLYNEAUX, D., KIM, D., DAVISON, A. J., KOHLI, P., SHOTTON, J., HODGES, S., AND FITZGIBBON, A. W. 2011. Kinectfusion: Real-time dense surface mapping and tracking. In *Proc. of IEEE ISMAR*, 127–136.

NIESSNER, M., ZOLLHÖFER, M., IZADI, S., AND STAMMINGER, M. 2013. Real-time 3d reconstruction at scale using voxel hashing. *ACM Trans. Graph. 32*, 6 (Nov.), 169:1–169:11.

PERO, L. D., BOWDISH, J., FRIED, D., KERMGARD, B., HARTLEY, E., AND BARNARD, K. 2012. Bayesian geometric modeling of indoor scenes. In *Proc. CVPR*, IEEE, 2719–2726.

REISNER-KOLLMANN, I., MAIERHOFER, S., AND PURGATHOFER, W. 2013. Reconstructing shape boundaries with multimodal constraints. *Comput. Graph. 37*, 3 (May), 137–147.

ROTH, H., AND VONA, M. 2012. Moving volume kinectfusion. In *British Machine Vision Conference (BMVC)*, 1–11.

SALAS-MORENO, R. F., NEWCOMBE, R. A., STRASDAT, H., KELLY, P. H. J., AND DAVISON, A. J. 2013. Slam++: Simultaneous localisation and mapping at the level of objects. In *Proc. CVPR*, IEEE, 1352–1359.

SCHNABEL, R., WAHL, R., AND KLEIN, R. 2007. Efficient ransac for point-cloud shape detection. *Computer Graphics Forum 26*, 2, 214–226.

SHAO, T., XU, W., ZHOU, K., WANG, J., LI, D., AND GUO, B. 2012. An interactive approach to semantic modeling of indoor scenes with an rgbd camera. *ACM Trans. Graph. 31*, 6 (Nov.), 136:1–136:11.

SILBERMAN, N., SHAPIRA, L., GAL, R., AND KOHLI, P. 2014. A contour completion model for augmenting surface reconstructions. In *Computer Vision–ECCV 2014*. Springer, 488–503.

SNAVELY, N., SEITZ, S. M., AND SZELISKI, R. 2006. Photo tourism: exploring photo collections in 3d. *ACM Trans. Graph. 25*, 3 (July), 835–846.

STEINBRUCKER, F., KERL, C., CREMERS, D., AND STURM, J. 2013. Large-scale multi-resolution surface reconstruction from rgb-d sequences. In *Proc. ICCV*, IEEE, 3264–3271.

SUN, Y.-N., AND HUANG, S.-C. 2000. Genetic algorithms for error-bounded polygonal approximation. *International Journal of Pattern Recognition and Articial Intelligence 14*, 3, 297–314.

TAGUCHI, Y., JIAN, Y.-D., RAMALINGAM, S., AND FENG, C. 2013. Point-plane slam for hand-held 3d sensors. In *Proc. ICRA*, IEEE, 5182–5189.

TOMONO, M. 2012. Image-based planar reconstruction for dense robotic mapping. In *Proc. ICRA*, IEEE, 3005–3012.

TRIGGS, B., MCLAUCHLAN, P., HARTLEY, R., AND FITZGIBBON, A. 2000. Bundle adjustment – a modern synthesis. In *Vision Algorithms: Theory and Practice, LNCS*, Springer Verlag, 298–375.

WHELAN, T., JOHANNSSON, H., KAESS, M., LEONARD, J. J., AND MCDONALD, J. B., 2012. Robust tracking for real-time dense RGB-D mapping with Kintinuous. Technical Report, Sept.

ZHOU, Q.-Y., AND KOLTUN, V. 2013. Dense scene reconstruction with points of interest. *ACM Trans. Graph. 32*, 4 (July), 112:1–112:8.

ZHOU, Q.-Y., AND KOLTUN, V. 2014. Color map optimization for 3d reconstruction with consumer depth cameras. *ACM Trans. Graph. 33*, 4 (July), 155:1–155:10.

ZHOU, Q.-Y., AND KOLTUN, V. 2014. Simultaneous localization and calibration: Self-calibration of consumer depth cameras. In *Proc. CVPR*, IEEE.