A Robust 3D-2D Interactive Tool for Scene Segmentation and Annotation

Duc Thanh Nguyen, Binh-Son Hua*, Lap-Fai Yu, Member, IEEE, and Sai-Kit Yeung, Member, IEEE

Abstract—Recent advances of 3D acquisition devices have enabled large-scale acquisition of 3D scene data. Such data, if completely and well annotated, can serve as useful ingredients for a wide spectrum of computer vision and graphics works such as data-driven modeling and scene understanding, object detection and recognition. However, annotating a vast amount of 3D scene data remains challenging due to the lack of an effective tool and/or the complexity of 3D scenes (e.g. clutter, varying illumination conditions). This paper aims to build a robust annotation tool that effectively and conveniently enables the segmentation and annotation of massive 3D data. Our tool works by coupling 2D and 3D information via an interactive framework, through which users can provide high-level semantic annotation for objects. We have experimented our tool and found that a typical indoor scene could be well segmented and annotated in less than 30 minutes by using the tool, as opposed to a few hours if done manually. Along with the tool, we created a dataset of over a hundred 3D scenes associated with complete annotations using our tool. Both the tool and dataset are available at http://scenenn.net.

Index Terms—Annotation tool, semantic annotation, 3D segmentation, 3D reconstruction, 2D-3D interactive framework

1 INTRODUCTION

High-quality 3D scene data has become increasingly available thanks to the growing popularity of consumer-grade depth sensors and tremendous progress in 3D scene reconstruction research [1], [2], [3], [4], [5], [6]. Such 3D data, if fully and well annotated, would be useful for powering different computer vision and graphics tasks such as scene understanding [7], [8], object detection and recognition [9], and functionality reasoning in 3D space [10].

Scene segmentation and annotation refer to separating an input scene into meaningful objects. For example, the scene in Fig. 1 can be segmented and annotated into chairs, table, etc. Literature has shown the crucial role of 2D annotation tools (e.g. [11]) and 2D image datasets (e.g. [12], [13], [14]) various tasks like semantic segmentation, object detection and recognition [15], [16]. This inspires us for such tasks on 3D scene data. However, segmentation and annotation of 3D scenes require much more effort due to the large scale of the 3D data (e.g. there are millions of 3D points in a reconstructed scene). Development of a robust tool to facilitate the segmentation and annotation of 3D scenes thus is a demand and also the aim of this work. To this end, we make the following contributions:

- We propose an interactive framework that effectively couples the geometric and appearance information from multi-view RGB data. The framework is able to automatically perform 3D scene segmentation.
- Our tool is facilitated with a 2D segmentation algorithm based on 3D segmentation.
- We develop assistive user-interactive operations that allow users to flexibly manipulate scenes and objects in both 3D and 2D. Users co-operate with the tool by refining the segmentation and providing semantic annotation.
- To further assist users in annotation, we propose an object search algorithm which automatically segments and annotates repetitive objects defined by users.
- We create a dataset including over a hundred scenes. All the scenes are fully segmented and annotated using our tool. This dataset will serve as a benchmark for future works in 3D computer vision and graphics.

Compared with existing works on RGB-D segmentation and annotation (e.g. [17], [18]), our tool holds several advantages. First, segmentation and annotation are centralized in 3D and thus free users from manipulating thousands of images. Second, the tool can adapt with either RGB-D images or scene triangular meshes as the input. This enables the tool to handle meshes reconstructed from either RGB-D images [19] or structure-from-motion [20] in a unified framework.

We note that interactive annotation has also been exploited in a few concurrent works, e.g. SemanticPaint in [21] and Semantic Paintbrush in [22]. Compared with those systems, our annotation tool offers a wider range of interactions. In addition, the tool also provides more assistive functionalities, e.g. 3D object search, 2D segmentation.

2 RELATED WORK

RGB-D Segmentation. A common approach for scene segmentation is to perform the segmentation on RGB/D images. Examples of this approach can be found in [17], [18], [23]. The spatial relationships between objects can also be exploited to infer the scene labels. For example, Jia et al. [24] used
object layout rules for scene labeling. The spatial relationship between objects was modeled by a conditional random field (CRF) in [25], [26] and directed graph in [27].

In general, the above methods segment RGB/D images captured from single viewpoints of a 3D scene individually while the segmentation of an individual image may not be reliable and incomplete. Compared with those methods, our tool can achieve more accurate and complete segmentation results with the 3D models of the scene and its objects.

From 2D to 3D. Labeling a 3D scene can be performed by back-projecting the labels obtained on the 2D images of that scene to 3D space. For example, Wang et al. [28] used the labels provided in ImageNet [12] to infer 3D labels. In [3], 2D labels were obtained by drawing polygons.

Labeling directly on images is time consuming. Typically, a few thousands of images need to be handled. It is possible to perform matching among the images to propagate the annotations from one image to another, e.g. [3], but this process is not reliable.

3D Object Templates. 3D object templates can be used to segment 3D scenes. The templates can be organized in holistic models, e.g., [29], [30], [31], [32], or part-based models, e.g., [33]. The segmentation can be performed on 3D point clouds, e.g. [29], [31], [33], or 3D patches, e.g. [32], [30], [34].

Generally speaking, the above techniques require the template models to be known in advance. They do not fit well our interactive system in which the templates can be provided on the fly by users. In our tool, we propose to use shape matching to help users in the segmentation and annotation task. Shape matching does not require off-line training and is proved to perform efficiently in practice [35].

Online Scene Understanding. Recently, there are methods that combine 3D reconstruction and annotation to achieve online scene understanding. For example, Tateno et al. [36] proposed to segment depth images and fuse the segmentations incrementally into a SLAM framework. SemanticPaint developed in [21] allowed users annotate a scene by touching objects. The SemanticPaint was extended to the Semantic Paintbrush [22] for outdoor scenes annotation by exploiting the farther range of a stereo rig.

In both [21] and [22], objects of interest were identified by touching and the corresponding object classes were modeled by CRFs. These methods used voxel-based TSDF representation rather than triangle mesh (as our approach) to represent objects and implicitly assumed that all objects of the same class have similar appearance (e.g. color). Since the CRFs were built upon the reconstructed data, there also assumed the reconstructed data was good enough. However, reconstructed scenes are often incomplete. To deal with this issue, we describe 3D objects using a shape descriptor which is robust to shape variation and occlusion. Experimental results show that our approach works well under noisy data (e.g. broken mesh) and robustly deal with shape deformation while being efficient for practical use.

3 System Overview

Fig. 2 shows the workflow of our tool. The tool includes four main stages: scene reconstruction, automatic 3D segmentation, interactive refinement and annotation, and 2D segmentation.

In the first stage (section 4), the system takes a sequence of RGB-D frames and reconstructs a triangular mesh, called 3D scene mesh. We compute and cache the correspondences between the 3D vertices in the reconstructed scene and the 2D pixels on all input frames. This allows seamless switching between segmentation in 3D and 2D in later steps.

In the second stage (section 5), the 3D scene mesh is automatically segmented. We start by clustering the mesh vertices into supervertices (section 5.1). Next, we group the supervertices into regions (section 5.2). We also cache the results of both steps for later use.

In the third stage (section 6) is designed for users to interact with the system. We design three segmentation refinement operations: merge, extract, and split. After refinement, users can make semantic annotation for objects in the scene.

To further assist users in segmentation and annotation of repetitive objects, we propose an algorithm to automatically search for repetitive objects specified by a template (section 7). We extend the well-known 2D shape context [35] to 3D space and apply shape matching to implement this functionality.

The fourth stage of the framework (section 8) is designed for segmentation of 2D frames. In this stage, we devise an algorithm based on contour matching that uses the segmentation results in 3D to initialize the 2D segmentation.
Fig. 2. Overview of our annotation tool.

4 SCENE RECONSTRUCTION

4.1 Geometry reconstruction

Several techniques have been developed for 3D scene reconstruction. For example, KinectFusion [37] applied frame-to-model alignment to fuse depth information and visualize 3D scenes in real time. However, KinectFusion tends to cause drift where depth maps are not accurately aligned due to accumulation of registration errors over time. Several attempts have been made to avoid drift and led to significant improvements in high-quality 3D reconstruction. For example, Xiao et al. [3] added object constraints to correct misaligned reconstructions. In [38], [4], [39], the input frames were split into small chunks, each of which could be accurately reconstructed. An optimization was then performed to register all the chunks into the same coordinate frame. In many systems, e.g. [40], re-visiting places are used to trigger a loop closure constraint to enforce global consistency of camera poses. In [41], loop closure was enabled by splitting the scene into submaps. The information from every single image frame was accumulated into the submaps and the position and orientation of the submaps were then adjusted accordingly.

In our system, we adopt the method in [4], [19] to calculate camera poses. The triangular mesh then can be extracted using the marching cubes algorithm [42]. The normal of each mesh vertex is given by the area-weighted average over the normals of its neighbor surfaces. We further smooth the resulting normals using a bilateral filter.

4.2 3D-2D Correspondence

Given the reconstructed 3D scene, we align the whole sequence of 2D frames with the 3D scene using the corresponding camera poses obtained from section 4.1. For each 3D vertex, its normal is computed on the 3D mesh and color is estimated as the median of the color of the corresponding pixels on 2D frames.

5 SEGMENTATION IN 3D

After the reconstruction, a scene mesh typically consists of millions of vertices. In this stage, those vertices are segmented into much fewer regions. Directly segmenting millions of vertices is computationally expensive and requires a lot of computer resources. To avoid this, we perform a two-level segmentation. At the first level, we divide the reconstructed scene into a number of so-called supervertices by applying a purely geometry-based segmentation method. At the second level, we merge the supervertices into larger regions by considering both surface normals and colors.

5.1 Graph-based Segmentation

We extend the efficient graph-based image segmentation algorithm of Felzenszwalb et al. [43] to 3D space. We have also tried with normalized cuts [44] and found that graph-based segmentation worked more stably and faster. In addition, graph-based segmentation is often selected for scene segmentation, e.g. in [45]. However, we note that other existing superpixel methods [46] could also be considered for this task.

The graph-based segmentation algorithm operates as follows. Given the scene mesh, a graph is defined in which each node in the graph corresponds to a vertex in the mesh. Two nodes in the graph are linked by an edge if their two corresponding vertices in the mesh are the vertices of a triangle. Let \( V = \{v_i\} \) be the set of vertices in the mesh. The edge connecting two vertices \( v_i \) and \( v_j \) is weighted as

\[
  w(v_i, v_j) = 1 - n_{i}^\top n_{j},
\]

where \( n_i \) and \( n_j \) are the unit normals of \( v_i \) and \( v_j \) respectively.

The graph-based segmenter in [43] employs a number of parameters including a smoothing factor used for noise filtering (normals in our case), a threshold representing the contrast between adjacent regions, and the minimum size of segmented regions. In our implementation, those parameters were set to 0.5, 500, and 20 respectively. However, we also make those parameters available to users for customization.

The graph-based segmentation algorithm results in a set of supervertices \( S = \{s_i\} \). Each supervertex is a group of geometrically homogeneous vertices with similar surface normals. The bottom left image in Fig. 2 shows an example of the supervertices. More examples can be found in Fig. 9.

5.2 MRF-based Segmentation

The graph-based segmentation often produces a large number (e.g. few thousand) supervertices, which could require considerable effort for annotation. To reduce this burden, the
supervertices are clustered into regions via optimizing an MRF model. In particular, for each supervertex \( s_i \in S \), the color and normal of \( s_i \), denoted as \( c_i \) and \( n_i \), are computed as the means of the color values and normals of all vertices \( v \in s_i \). Each supervertex \( s_i \in S \) is then represented by a node \( o_i \) in an MRF. Two nodes \( o_i \) and \( o_j \) are directly connected if \( s_i \) and \( s_j \) share some common boundary (i.e., \( s_i \) and \( s_j \) are adjacent supervertices). Letting \( l_i \) be the label of \( o_i \), the unary potentials are defined as

\[
\psi_1(o_i, l_i) = -\log G_i^c(c_i, \mu_i^c, \Sigma_i^c) - \log G_i^n(n_i, \mu_i^n, \Sigma_i^n),
\]

where \( G_i^c \) and \( G_i^n \) are the Gaussians of the color values and normals of the label class of \( l_i \), and \( \mu_i^c / \mu_i^n \) and \( \Sigma_i^c / \Sigma_i^n \) are the mean and covariance matrix of \( G_i^c / G_i^n \). The means and covariance matrices are computed based on the current labels and updated accordingly during the labeling process.

For the pairwise potentials, we use the Potts model [47]

\[
\psi_2(l_i, l_j) = \begin{cases} -1, & \text{if } l_i = l_j \\ 1, & \text{otherwise.} \end{cases}
\]

Let \( L = \{l_1, l_2, ..., l_{|S|}\} \) be a labeling of the supervertices. The optimal labeling \( L^* \) is determined by

\[
L^* = \arg \min_L \left[ \sum_i \psi_1(o_i, l_i) + \gamma \sum_{i,j} \psi_2(l_i, l_j) \right]
\]

where \( \gamma \) is weight factor that is set to 0.5 in our implementation.

The optimization problem in (4) is solved using the method in [47]. In our implementation, the number of labels was initialized to the number of supervertices; each supervertex was assigned to a different label. Fig. 2 (bottom) shows the result of the MRF-based segmentation. More results of this step are presented in Fig. 9.

6 SEGMENTATION REFINEMENT AND ANNOTATION IN 3D

The automatic segmentation stage can produce over- and under-segmented regions. To resolve these issues, we design three operations: merge, extract, and split.

Merge. This operation is used to resolve over-segmentation. In particular, users identify over-segmented regions that need to be grouped by stroking on them. The merge operation is illustrated in the first row of Fig. 3.

Extract. This operation is designed to handle under-segmentation. In particular, for an under-segmented region, the supervertices composing that region can be retrieved. Users can select a few of those supervertices and use the merge operation to group them to create a new region. The second row of Fig. 3 shows the extract operation.

Split. In a few rare cases, the MRF-based segmentation may perform differently on different regions. This is probably because of the variation of the shape and appearance of objects. For example, a scene may have chairs appearing in a unique color and other chairs each of which composes multiple colors. Therefore, a unique setting of the parameters in the MRF-based segmentation may not adapt to all objects.

To address this issue, we design a split operation enabling user-guided MRF-based segmentation. Specifically, users first select an under-segmented region by stroking on that region. The MRF-based segmentation is then invoked on the selected region with a small value of \( \gamma \) (see (4)) to generate more grained regions. We then enforce a constraint such that the starting and ending point of the stroke belong to two different labels. For example, assume that \( l_i \) and \( l_j \) are the labels of two supervertices that respectively contain the starting and ending point of the stroke. To bias the objective function in (4), \( \psi_2(l_i, l_j) \) in (3) is set to \(-1\) when \( l_i \neq l_j \), and to a large value (e.g., \( 10^9 \)) otherwise. By doing so, the optimization in (4) would favor the case \( l_i \neq l_j \). In other words, the supervertices at the starting and ending point are driven to separate regions. Note that the MRF-based segmentation is only re-executed on the selected region. Therefore, the split operation is fast and does not hinder user interaction. In particular, we have empirically found that the split operation takes much less than a second to process a single segment. We also note that the split operation takes into account the regions users select to re-run the MRF based optimization (with some implied prior) and hence would be less cumbersome than manually selecting supervertices as in the extract operation. The third row of Fig. 3 illustrates the split operation.

As shown in our user study (see Appendix), users mostly perform merge and extract operations. Split operation is only used when extract operation is not able to handle severe under-segmentations but such cases are not common in practice. When all the 3D segmented regions have been refined, users can annotate the regions by providing the object type. Fig. 4 shows an example of using our tool for annotation.

7 OBJECT SEARCH

There may exist multiple instances of an object class in a scene, e.g., the nine chairs in Fig. 5. To support labeling and
annotating repetitive objects, users can define a template by selecting an existing region or multiple regions composing the template. Those regions are the results of the MRF-based segmentation or user refinement. Given the user-defined template, our system automatically searches for objects that are similar to the template. Note that each repetitive object may be composed of multiple regions. For example, each chair in Fig. 5(a) consists of different regions such as the back, seat, legs. Once a group of regions is found to match well with the template, the regions are merged into a single object and recommended to users for verification. This is because the object search operation may propose inappropriate objects. Note that the object search operation is applied only to unlabeled regions. In our implementation, we extend the 2D shape context proposed in [35] to describe 3D objects (section 7.1). Matching objects with the template is performed via comparing shape context descriptors (section 7.2). The object search is then built upon the sliding-window object detection approach [48] (section 7.3).

7.1 Shape Context
Shape context was proposed by Belongie et al. [35] as a 2D shape descriptor and is well-known for many desirable properties such as being discriminative and robust to noise, shape deformation and transformation, and partial occlusions. Those properties fit our need well for several reasons. First, reconstructed scene meshes could be incomplete and contain noisy surfaces. Second, occlusions may also appear due to the lack of sufficient images completely covering objects. Third, the tool is expected to adapt to the variation of object shapes, e.g. chairs with and without arms.

In our work, a 3D object is represented by a set $\mathcal{V}$ of vertices obtained from the 3D reconstruction step. For each vertex $v_i \in \mathcal{V}$, the shape context of $v_i$ is denoted as $s(v_i)$ and represented by the histogram of the relative locations of other vertices $v_j, j \neq i$, to $v_i$. Let $u_{ij} = v_i - v_j$. The relative location of a vertex $v_j \in \mathcal{V}$ to $v_i$ is encoded by the length $||u_{ij}||$ and the spherical coordinate $(\theta, \phi)_{ij}$ of $u_{ij}$. In our implementation, the lengths $||u_{ij}||$ were quantized into
5 levels. To make the shape context $s(v_i)$ more sensitive to local deformations, $||u_{ij}||$ were quantized in a log-scale space. The spherical angles $(\theta, \phi)_{ij}$ were quantized uniformly into 6 discrete values. Fig. 5(a) illustrates the 3D shape context.

The shape context descriptor is endowed with scale invariance by normalizing $||u_{ij}||$ by the mean of the lengths of all vectors. To make the shape context rotation invariant, Kortgen et al. [49] computed the spherical coordinates $(\theta, \phi)_{ij}$ relative to the eigenvectors of the covariance matrix of all vertices. However, the eigenvectors may not be computed reliably for shapes having no dominant orientations, e.g. rounded objects. In addition, the eigenvectors are only informative when the shape is complete while our scene meshes may be incomplete. To overcome this issue, we establish a local coordinate frame at each vertex on a shape using its normal and tangent vector. The tangent vector of a vertex $v_i$ is the one connecting $v_i$ to the centroid of the shape. We found that this approach worked more reliably.

Since a reconstructed scene often contains millions of vertices, prior to applying the object search, we uniformly sample 20,000 points from the scene, which results in objects of 100 – 200 vertices.

7.2 Shape Matching

Comparing (matching) two given shapes $\mathcal{V}$ and $\mathcal{Y}$ is to maximize the correspondences between pairs of vertices on these two shapes, i.e. minimizing the deformation of the two shapes in a point-wise fashion. We define the deformation cost between two vertices $v_i \in \mathcal{V}$ and $y_j \in \mathcal{Y}$ to be the $\chi^2(s(v_i), s(y_j))$ distance between the two corresponding shape context descriptors extracted at $v_i$ and $y_j$ as follow,

$$
\chi^2(s(v_i), s(y_j)) = \frac{1}{2} \sum_{b=1}^{\text{dim}(s(v_i))} \frac{(s(v_i)[b] - s(y_j)[b])^2}{s(v_i)[b] + s(y_j)[b]} \tag{5}
$$

where $\text{dim}(s(v_i))$ is the dimension (i.e. the number of bins) of $s(v_i)$, and $s(v_i)[b]$ is the value of $s(v_i)$ at the $b$-th bin.

Given the deformation cost of every pair of vertices on two shapes $\mathcal{V}$ and $\mathcal{Y}$, shape matching can be solved using the shortest augmenting path algorithm [50]. To make the matching algorithm adaptive to shapes with different number of vertices, "dummy" vertices are added. This enables the matching method to be robust to noisy data and partial occlusions. Formally, the deformation cost $C(\mathcal{V}, \mathcal{Y})$ between two shapes $\mathcal{V}$ and $\mathcal{Y}$ is computed as,

$$
C(\mathcal{V}, \mathcal{Y}) = \sum_{v_i \in \mathcal{V}} \chi^2(s(v_i), s(\pi(v_i))) \tag{6}
$$

where $\hat{\mathcal{Y}}$ is identical to $\mathcal{Y}$ or augmented from $\mathcal{Y}$ by adding dummy vertices and $\pi(v_i) \in \hat{\mathcal{Y}}$ is the matching vertex of $v_i$ determined by using [50].

To further improve the matching, we also consider how well the two shapes correspond. In particular, we first align $\mathcal{V}$ to $\hat{\mathcal{Y}}$ using a rigid transformation. This rigid transformation is represented by a $4 \times 4$ matrix and estimated using the RANSAC algorithm that randomly picks three pairs of correspondences and determines the rotation and translation [51]. We then compute an alignment error,

$$
E(\mathcal{V}, \mathcal{Y}) = \min \left\{ \sqrt{\frac{1}{|\mathcal{V}|} \sum_{i=1}^{|\mathcal{V}|} \epsilon_i^{(Y)}}, \sqrt{\frac{1}{|\mathcal{Y}|} \sum_{i=1}^{|\mathcal{Y}|} \epsilon_i^{(Y)}} \right\} \tag{7}
$$

where

$$
\epsilon_i^{(Y)} = \begin{cases} 
\|\pi(v_i) - T \cdot v_i\|^2 & \text{if } \pi(v_i) \text{ exists for } v_i \in \mathcal{V} \\
\Delta^2 & \text{otherwise}
\end{cases} \tag{8}
$$

and, similarly for $\epsilon_i^{(\mathcal{Y})}$, where $T$ is the rigid transformation matrix and $\Delta$ is a large value used to penalize misalignments. A match is confirmed if: (i) $C(\mathcal{V}, \mathcal{Y}) < \tau_a$ and (ii) $E(\mathcal{V}, \mathcal{Y}) < \tau_s$ where $\tau_s$ and $\tau_a$ are thresholds. In our experiments, we set $\Delta = 2$ (meters), $\tau_s = 0.7$, $\tau_a = 0.4$. We found that the object search method was not too sensitive to parameter settings, and that these settings achieved the best performance.

7.3 Searching

Object search can be performed based on the sliding-window approach [48]. Specifically, we take the 3D bounding box of the template and use it as the window to scan a 3D scene. At each location in the scene, all regions that intersect the window are considered for their possibility to be part of a matching object. However, it would be intractable to consider every possible combination of all regions. To deal with this issue, we propose a greedy algorithm that operates iteratively by adding and removing regions.

The general idea is as follows. Let $\mathcal{R}$ be the set of regions that intersect the window $\mathcal{W}$, i.e. the 3D bounding box of the template. For a region $r \in \mathcal{W} \setminus \mathcal{R}$, we verify whether the object made by $\mathcal{R} \cup \{r\}$ could be more similar to the user-defined template $\mathcal{O}$ in comparison with $\mathcal{R}$. Similarly, for every region $r \in \mathcal{R}$ we also verify the object made by $\mathcal{R} \setminus \{r\}$. These adding and removing steps are performed interchangeably in a small number of iterations until the best matching result (i.e. a group of regions) is found. This procedure is called grow-shrink and described in Algorithm 1.

In our implementation, the spatial strides on the $x-$, $y-$, and $z-$ direction of the window $\mathcal{W}$ were set to the size of $\mathcal{W}$. The number of iterations in Algorithm 1 was set to 10, which resulted in satisfactory accuracy and efficiency (see Section 9.3).

Since a region may be contained in more than one window, it may be verified multiple times in multiple groups of regions. To avoid this, if an object candidate is found in a window, its regions will not be considered in any other objects and any other windows. Fig. 5 illustrates the robustness of the object search in localizing repetitive objects under severe conditions (e.g. objects with incomplete shape). The search procedure may miss some objects. To handle such cases, we design an operation called guided merge. In particular, after defining the template, users simply select one of the regions of a target object that is missed by the object search. The grow-shrink procedure is then applied on the selected region to seek a better match with the template. Fig. 6 shows an example of the guided merge operation.
which aims to fit the boundaries of projected regions to true boundaries in 2D frames. The true boundaries in a 2D frame can be extracted using some edge detector (e.g. the Canny edge detector [52]). Let $\mathcal{E} = \{e_j\}$ denote the set of edge points on the edge map of a 2D frame. Let $U = \{u_i\}$ be the set of contour points of a projected object in that frame. $U$ is then ordered using the Moore neighbor tracing algorithm [53].

The ordering step is used to express the contour alignment problem in a form to which dynamic programming can be applied for efficient implementation.

At each contour point $u_i$, we consider a $21 \times 21$-pixel window centered at $u_i$ (in relative to a $640 \times 480$-pixel image). We then extract the histogram $h_{u_j}$ of the orientations of vectors $(u_i, u_k), u_k \in U$ in the window. The orientations are uniformly quantized into 16 bins. We also perform this operation for edge points $e_j \in \mathcal{E}$. The dissimilarity between the two local shapes at a contour point $u_i$ and edge point $e_j$ is computed as $\chi^2(h_{u_i}, h_{e_j})$ (similarly to (5)).

We also consider the continuity and smoothness of contours. In particular, the continuity between two adjacent points $u_i$ and $u_{i-1}$ is defined as $\|u_i - u_{i-1}\|$. The smoothness of a fragment including three consecutive points $u_i, u_{i-1}, u_{i-2}$ is computed as $\cos(u_i - u_{i-1}, u_{i-2} - u_{i-1})$ where $u_i - u_{i-1}$ and $u_{i-2} - u_{i-1}$ denote the vectors connecting $u_{i-1}$ to $u_i$ and connecting $u_{i-1}$ to $u_{i-2}$ respectively, and $\cos(\cdot, \cdot)$ is the cosine of the angle formed by these two vectors.

Alignment of $U$ to $\mathcal{E}$ is achieved by identifying a mapping function $f : U \rightarrow \mathcal{E}$ that maps a contour point $u_i \in U$ to an edge point $e_j \in \mathcal{E}$ so as to,

$$
\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{[U]} \chi^2(h_{u_i}, h_{f(u_i)}) \\
& + \kappa_1 \sum_{i=2}^{[U]} \|f(u_i) - f(u_{i-1})\| \\
& + \kappa_2 \sum_{i=3}^{[U]} \cos(f(u_i) - f(u_{i-1}), f(u_{i-2}) - f(u_{i-1})) \\
\end{align*}
$$

(9)

The optimization problem in (9) can be considered as the bipartite graph matching problem [50]. However, since $U$ is ordered, this optimization can be solved efficiently using dynamic programming [54]. In particular, denoting $m_{i,j} = \chi^2(h_{u_i}, h_{e_j}), f_i = f(u_i), f_{i,j} = f(u_i) - f(u_j)$, the objective function in (9) can be rewritten as,

$$
F_i = \begin{cases} 
\min_{j \in \mathcal{E}} \{F_{i-1} + m_{i,j} + \kappa_1 \|f_{i-1,j}\| \\
+ \kappa_2 \cos(f_{i-1,j}, f_{i-2,j-1})\}, & \text{if } i > 2 \\
\min_{j \in \mathcal{E}} \{F_{i-1} + m_{i,j} + \kappa_1 \|f_{i-1,j}\|\}, & \text{if } i = 2 \\
\min_{j \in \mathcal{E}} \{m_{i,j}\}, & \text{if } i = 1 
\end{cases}
$$

(10)

where $\kappa_1$ and $\kappa_2$ are user parameters. Empirically, we set $\kappa_1 = 0.1$ and $\kappa_2 = 3.0$ in all of our experiments.

To save the computational cost of (10), for each contour point $u_i$, we consider only its $k$ nearest edge points whose distance to $u_i$ is less than a distance $\delta$ set to 10% of the maximum of the image dimension, e.g., $\delta = 48$ for a $640 \times 480$-pixel.
image. The number of nearest edge points (i.e. \( k \)) was set to 30. Fig. 7(c) shows an example of contour alignment by optimizing (10) using dynamic programming.

We have verified the contribution of the continuity and smoothness in Fig. 8. The results show that, when all the cues are taken into account, the contours are mostly well aligned with the true object boundaries. It is noticeable that the seat of the green chair is not correctly recovered. We have found that this is because the Canny detector missed important edges on the boundaries of the chair. Recently, new edge detectors (e.g. [55]) have been shown to work robustly under severe illumination conditions and in the face of complex texture changes. These edge detectors could be used to enhance the alignment algorithm.

9 Experiments

9.1 Dataset

We created a dataset consisting of over 100 scenes. The dataset includes six scenes from publicly available datasets: the copyroom and lounge from the Stanford dataset [38], the hotel and dorm from the SUN3D [3], and the kitchen and office sequences from the Microsoft dataset [56]. The Stanford and SUN3D dataset also provide registered RGB and depth image pairs. These datasets also include the camera pose data.

In addition to existing scenes, we collected 100 scenes using Asus Xtion and Microsoft Kinect v2. Our scenes were captured from the campus of the University of Massachusetts Boston and the Singapore University of Technology and Design. These scenes were captured from various locations such as lecturer rooms, theaters, university hall, library, computer labs, dormitory, etc. There were about 1,600 object instances of 20 object categories. The objects vary in shape and color. Their sizes also vary significantly; some objects are quite small, e.g. glasses (< 10cm), while others are very large, e.g. beds (about 2m), walls (about 5m). All the scenes and objects were fully segmented and annotated using our tool. The camera pose information is also included. The dataset is available at http://scenenn.net. Fig. 9 shows several of our collected scenes.

9.2 Evaluation of 3D Segmentation

We evaluated the impact of the graph-based and MRF-based segmentation on our dataset. We considered the annotated results obtained using our tool as the ground-truth. To measure the segmentation performance, we extended the object-level consistency error (OCE), the image segmentation evaluation metric proposed in [57] to 3D vertices. Essentially, the OCE reflects the coincidence of pixels/vertices of segmented regions and ground-truth regions. As indicated in [57], compared with other segmentation evaluation metrics (e.g. the global and local consistency error in [58]), the OCE considers both over- and under-segmentation errors in a single measure. In addition, OCE can quantify the accuracy of multi-object segmentation and thus fits our evaluation purpose well.

Table 1 summarizes the OCE of the graph-based and MRF-based segmentation. As shown in the table, compared with the graph-based segmentation, the segmentation accuracy is significantly improved by the MRF-based segmentation. It is also noticeable that the number of supervertices and regions yielded by the segmentation process is much smaller than the number of 3D vertices that were passed in, making them significantly more convenient for the users to work with. The OCE values presented in Table 1 show that automatic segmentation is still not approaching the quality achieved by human beings. Thus, user interactions are necessary. This is for two reasons: first, both the graph-based and MRF-based segmentation aim to segment a 3D scene into homogenous regions rather than semantically-meaningful objects; second, semantic segmentation is user-dependent and subjective [58].

After user interaction, the number of final labels is typically less than a hundred. The number of objects is around 10 to 20 in most of the cases. Note that the numbers of final labels and annotated objects are not identical. This is because there could have been labels whose semantics is not well defined, e.g. miscellaneous items on a table or some small labels appearing as noise in the 3D reconstruction.

We also report the time required for the segmentation and user interactions with the tool in Table 1. As shown in the table, with the assistance of the tool, a complex 3D scene (with millions of vertices) could be completely segmented and annotated in less than 30 minutes, as opposed to approximately
Segmentation by projecting 3D objects into a 2D frame

Using the local shape only (i.e. $\kappa_1 = \kappa_2 = 0$)

Using the local shape and continuity

Using the local shape and smoothness

Using the local shape, continuity, and smoothness

Fig. 8. Illustration of 2D segmentation. First column: segmentation result obtained by projection of 3D regions and overlaid on RGB images. Remaining columns: close-ups of four regions marked in the segmentation result in the first column.
Fig. 9. Results of our captured scenes. From left to right: the result of graph-based segmentation, MRF-based segmentation, and final segmentation and annotation made by user interaction. Please refer to our supplemental document for the results of other scenes.
We used the intersection over union (IoU) metric [13] to determine true detections and false alarms. However, instead of computing the IoU on object bounding boxes as in [13], we computed the IoU at point-level. This is because our aim is not only to localize repetitive objects but also to segment them. In particular, an object \( O \) (a set of vertices) formed by the object search procedure is considered as true detection if there exists an annotated object \( R \) in the ground-truth such that 
\[
\frac{|O \cap R|}{|O \cup R|} > 0.5
\]
where \(| \cdot |\) denotes the area.

The evaluation was performed on every template. The precision, recall, and \( F \)-measure \((= 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}})\) were then averaged over all evaluations. Table 2 shows the averaged precision, recall, and \( F \)-measure of the object search. As shown, the object search can localize 70% of repetitive objects with 69% precision and 65% \( F \)-measure. We have also tested the object search without considering the alignment error, i.e. \( E \) in (7). We have found that, compared with the solely use of shape context dissimilarity score (i.e. \( C \) in (6)), while the augmentation of alignment error could slightly incur a loss of the detection rate (about 2%), it largely improved the precision (from 22% to 69%). This led to a significant increase of the \( F \)-measure (from 30% to 65%).

Experimental results show that, the object search worked efficiently with templates represented by about 200 points. For example, the scene presented in Fig. 5 was completed within 15 seconds with a 150-point template and on a machine equipped by an Intel(R) Core(TM) i7 2.10 GHz CPU and 32 GB of memory. Note that threads can be used to run the object search in the background while users are performing interactions.

### 9.3 Evaluation of Object Search

To evaluate the object search, we collected a set of 45 objects from our dataset. Those objects were selected so that they are semantically-meaningful and their shapes are discriminative. For example, drawers of cabinets were not selected since they were present in flat surfaces which could be easily found in many structures, e.g. walls, pictures, etc. For each scene and each object class (e.g. chair), each object instance in the class was used as the template while the object search was applied to find the remaining objects of the same class.

### 9.4 Evaluation of 2D Segmentation

We also evaluated the performance of the 2D segmentation using the OCE metric. This experiment was conducted on

<table>
<thead>
<tr>
<th>Scene</th>
<th>#Vertices</th>
<th>#Supervertices</th>
<th>OCE</th>
<th>Time (seconds)</th>
<th>#Regions</th>
<th>OCE</th>
<th>Time (seconds)</th>
<th>#Labels</th>
<th>#Objects</th>
<th>Interactive time</th>
</tr>
</thead>
<tbody>
<tr>
<td>copyroom</td>
<td>1,309,421</td>
<td>1,996</td>
<td>0.92</td>
<td>1.0</td>
<td>347</td>
<td>0.73</td>
<td>10.9</td>
<td>157</td>
<td>15</td>
<td>19</td>
</tr>
<tr>
<td>lounge</td>
<td>1,597,553</td>
<td>2,554</td>
<td>0.97</td>
<td>1.1</td>
<td>506</td>
<td>0.93</td>
<td>7.3</td>
<td>53</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>hotel</td>
<td>3,572,776</td>
<td>13,839</td>
<td>0.98</td>
<td>2.7</td>
<td>1433</td>
<td>0.88</td>
<td>17.8</td>
<td>96</td>
<td>21</td>
<td>27</td>
</tr>
<tr>
<td>dorm</td>
<td>1,823,483</td>
<td>3,276</td>
<td>0.97</td>
<td>1.2</td>
<td>363</td>
<td>0.78</td>
<td>7.8</td>
<td>75</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>kitchen</td>
<td>2,557,593</td>
<td>4,640</td>
<td>0.97</td>
<td>1.8</td>
<td>470</td>
<td>0.85</td>
<td>12.2</td>
<td>75</td>
<td>24</td>
<td>23</td>
</tr>
<tr>
<td>office</td>
<td>2,349,679</td>
<td>4,026</td>
<td>0.97</td>
<td>1.7</td>
<td>422</td>
<td>0.84</td>
<td>10.9</td>
<td>69</td>
<td>19</td>
<td>24</td>
</tr>
<tr>
<td>Our scenes</td>
<td>1,450,748</td>
<td>2,498</td>
<td>0.93</td>
<td>1.4</td>
<td>481</td>
<td>0.77</td>
<td>12.1</td>
<td>179</td>
<td>19</td>
<td>30</td>
</tr>
</tbody>
</table>

### Table 2

Performance of the proposed object search.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>( F )-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without alignment error</td>
<td>0.22</td>
<td>0.72</td>
<td>0.30</td>
</tr>
<tr>
<td>With alignment error</td>
<td>0.69</td>
<td>0.70</td>
<td>0.65</td>
</tr>
</tbody>
</table>
the dorm sequence from the SUN3D dataset [3]. The dorm sequence contained 58 images in which the ground-truth labels were manually crafted and publicly available.

We report the segmentation performance obtained by projecting 3D regions into 2D images and by applying our alignment algorithm in Table 3. The impact of the local shape, continuity, and smoothness is also quantified. As shown in Table 3, the combination of the local shape, continuity, and smoothness achieves the best performance. We have visually found the alignment algorithm could make the projected contours smoother and closer to the true edges.

Experimental results show our alignment algorithm worked efficiently. On average, the alignment could be done in about 1 second for a 640 × 480-pixel frame.

### TABLE 3

Comparison of different segmentation methods.

<table>
<thead>
<tr>
<th>Segmentation method</th>
<th>OCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projection</td>
<td>0.57</td>
</tr>
<tr>
<td>Local shape</td>
<td>0.60</td>
</tr>
<tr>
<td>Local shape + Continuity</td>
<td>0.55</td>
</tr>
<tr>
<td>Local shape + Smoothness</td>
<td>0.55</td>
</tr>
<tr>
<td>Local shape + Continuity + Smoothness</td>
<td>0.54</td>
</tr>
</tbody>
</table>

### 10 CONCLUSION

This paper proposed a robust tool for segmentation and annotation of 3D scenes. The tool couples 3D geometric information and multi-view 2D RGB information in an interactive framework. To enhance the usability of the tool, we developed assistive, interactive operations that allow users to flexibly manipulate scenes and objects in both 3D and 2D space. The tool is also equipped with automated functionalities such as scene and image segmentation, and object search.

Along with the tool, we created a dataset of more than 100 scenes. All the scenes were annotated using our tool. The overall performance of the tool depends on the quality of 3D reconstruction. We are currently improving the quality of 3D meshes by recovering broken surfaces and missing object parts. User interactions could provide valuable information for many tasks, e.g. segmentation, annotation, and object search. Enhancing the performance of these tasks based on learning user interaction patterns (e.g. automatically learning templates and parameters) will also be our future work.

### APPENDIX

We have conducted a user study on the effectiveness of our interactive operations (merge, split, extract, and undo). There were 15 users involved in this study. The user study included two tasks (A and B) designed to handle simple and complex scenes. In task A, users were asked to segment a scene with only a few chairs and a table in two minutes. In task B, users were required to segment a complex bedroom scene containing a significant amount of furniture and many small objects in ten minutes. All operations users performed were logged. The statistics are shown in Fig. 11.

![User study statistics](image)

As can be seen, in both segmentation tasks, merge operation dominates, followed by extract and split operation. While a few users prefer split over extract (e.g. user 4, 6, 12, and 15 in task A; user 1 and 6 in task B), in such cases the number of split operation is always very few compared to merge. Note that our tool offers both flexibility in segmenting objects with various sizes as well as efficiency in computation since small segments for extract can be pre-computed.

In addition, task A also shows that merge is the most straightforward operation for new users. Extract and split are more useful when a scene gets more complex, with some coarse segments spread over small objects.
**ACKNOWLEDGMENT**

Binh-Son Hua is supported by the SUTD Digital Manufacturing and Design (DManD) Centre which is supported by the National Research Foundation (NRF) of Singapore.

Lap-Fai Yu is supported by the Joseph P. Healey Research Grant Program provided by the Office of the Vice Provost for Research and Strategic Initiatives & Dean of Graduate Studies of UMass Boston. This research is also supported by the National Science Foundation under award number 1565978. We acknowledge NVIDIA Corporation for graphics card donation.

Sai-Kit Yeung is supported by Singapore MOE Academic Research Fund MOE2016-T2-2-154 and Supported by Heritage Research Grant of the National Heritage Board, Singapore. We acknowledge the support of the SUTD Digital Manufacturing and Design (DManD) Centre which is supported by the National Research Foundation (NRF) of Singapore. This research is also supported by the National Research Foundation, Prime Minister’s Office, Singapore under its IDM Futures Funding Initiative and its Environmental & Water Technologies Strategic Research Programme and administered by the PUB, Singapore National Water Agency. This material is based on research/work supported in part by the National Research Foundation under Virtual Singapore Award No. NRF2015VS-GAA3DCM001-014.

We thank Fangyu Lin and Quang-Hieu Pham for assisting with data capture and Guoxuan Zhang for the early version of the tool. The Lounge scene data in Figure 2 is courtesy of Zhou and Koltun [38].

**REFERENCES**


