SemanticPaint: Interactive 3D Labeling and Learning at your Fingertips

Julien Valentin[1*], Vibhav Vineet[1*], Ming-Ming Cheng[1,4*], David Kim[2], Jamie Shotton[2], Pushmeet Kohli[2], Matthias Nießner[3], Antonio Criminisi[2], Shahram Izadi[2*], Philip Torr[1*]
[1]University of Oxford [2]Microsoft Research Cambridge [3]Stanford University [4]Nankai University

We present a new interactive and online approach to 3D scene understanding. Our system, *SemanticPaint*, allows users to simultaneously scan their environment, whilst interactively segmenting the scene simply by reaching out and touching any desired object or surface. Our system continuously learns from these segmentations, and labels new unseen parts of the environment. Unlike offline systems, where capture, labeling and batch learning often takes hours or even days to perform, our approach is fully *online*. This provides users with continuous *live* feedback of the recognition during capture, allowing them to immediately correct errors in the segmentation and/or learning – a feature that has so far been unavailable to batch and offline methods. This leads to models that are tailored or *personalized* specifically to the user's environments and object classes of interest, opening up the potential for new applications in augmented reality, interior design, and human/robot navigation. It also provides the ability to capture substantial labeled 3D datasets for training large-scale visual recognition systems.

Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Digitizing and Scanning*; I.4.8 [**Image Processing and Computer Vision**]: Scene Analysis—*Range data*

General Terms: Interactive, 3D scene understanding, online, depth camera

Additional Key Words and Phrases: segmentation, 3D features, learning

1. INTRODUCTION

Imagine walking into a room with a wearable consumer depth camera. As you move within the space, the dense 3D geometry of the room is automatically scanned in real-time. As you begin to physically interact with the room, touching surfaces and objects, the 3D model begins to be automatically segmented into semantically meaningful regions, each belonging to a specific category such as wall, door, table, book, cup and so forth. As you continue interacting with the world, a learned model of each category is updated and refined with new examples, allowing the model to handle more variation in shape and appearance. When you start observing new instances of these learned object classes, perhaps in another part of the room, the 3D model will automatically and densely be labeled and segmented, almost as quickly as the geometry is acquired. If the inferred labels are imperfect, you can quickly and interactively both correct the labels and improve the learned model. At the end you rapidly generate accurate, *labeled* 3D models of large environments.

Although simple and intuitive from the user interaction perspective, this type of *scene understanding* scenario has remained challenging since the field of computer vision was first established over 50 years ago [Roberts 1963]. Approaches typically first perform an *offline acquisition phase* where images, RGB-D data, or 3D models are acquired. These are then labeled often using crowd-sourcing techniques e.g. [Russell et al. 2008]. The labeled data is then used for offline *batch* training of generative or discriminative models [Koppula et al. 2011; Silberman et al. 2012]. This is followed by a *test phase* where the learned models are used for labeling semantic parts. It is not unusual for existing systems to take hours or even days to train [Shotton et al. 2006], and even the test phase can often take multiple seconds per image [Shotton et al. 2006; Kohli et al. 2009] due to the use of expensive feature extraction and recognition algorithms. Furthermore, it is hard to know in advance how much labeled training data is required and how varied it should be. Even then, the learned model is not guaranteed to generalize well, and if so the whole process must begin again with further acquisition.

This paper aims to address these issues associated with offline techniques, and bring us closer to realizing the challenging and yet compelling scenario above: rather than offline data collection, labeling, and training, we describe a system that can perform *all* parts of the pipeline, from low-level 3D reconstruction, through to semantic segmentation, training, and testing, interactively (i.e. with the user in the loop) and in an online manner. We provide a simple and *unified* user experience, which even novice users can perform. The user receives continuous and *immediate* feedback of the semantic pipeline during capture. This allows online correction of errors in the segmentation and/or learning. This also leads to 3D semantic models that are tailored or *personalized* specifically to the user's environments and object classes of interest. Our online system allows anyone to interactively acquire semantically labeled 3D models of arbitrary environments in minutes (see Fig. 1 and supplementary video).

Under the hood, a dense 3D model of the environment is captured in real-time by fusing noisy depth maps into an implicit volumetric surface representation. Our system then allows the user to walk up to any object of interest, simply touch and *'paint'* the physical surface, and vocally call out a new or existing object class name. Our method first cleanly segments any touched object from its supporting or surrounding surfaces, using a new volumetric inference technique based on an efficient mean-field approximation. This is lightweight to perform and places the user 'in the loop' during the process,

*Joint first authors

DOI 10.1145/1559755.1559763
http://doi.acm.org/10.1145/1559755.1559763

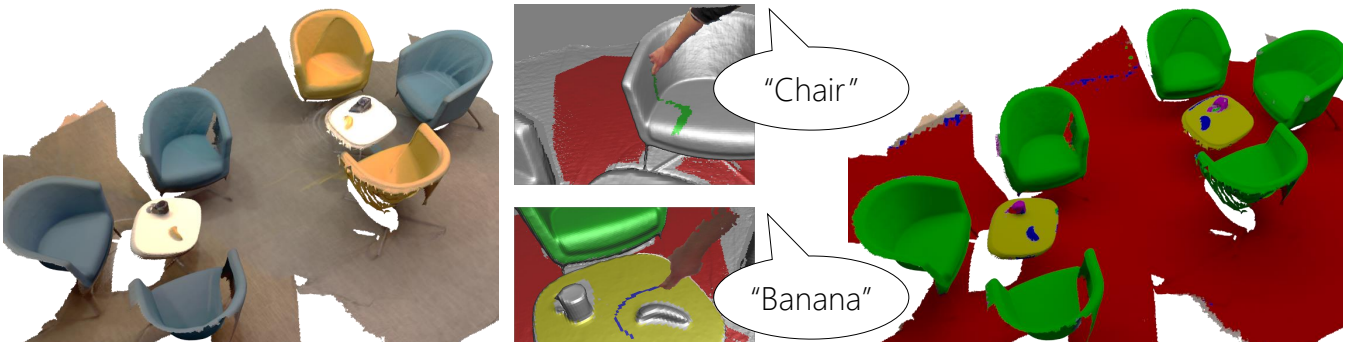

Fig. 1. Our system allows users to quickly and interactively label the world around them. The environment is scanned using a consumer RGB-D camera, and in real-time a volumetric fusion algorithm reconstructs the scene in 3D with additional color data (left). At any point the user can reach out and touch objects in the physical world, and provide object class labels through voice commands (middle). Then, an inference engine propagates these user-provided labels through the reconstructed scene, in real-time. In the meantime, in the background, a new, streaming random forest learns to assign object class labels to voxels in unlabeled regions of the world. Finally, another round of volumetric label propagation produces visually smooth labels over the entire scene (right).

allowing her to focus on objects that are of personal interest, rather than those predefined by application developers or dataset creators.

In the background, a new form of *streaming random forest* is trained and updated as new labeled object examples become available. The decision forest can quickly infer the likelihood that any newly-observed voxel belongs to each object class. The final stage of our pipeline estimates a spatially consistent dense labeling of the voxel reconstruction by again performing mean-field inference over the voxel space but now using the results from the decision forest. This creates a system that can be used to rapidly segment 3D scenes and learn from these labels in an online manner. Furthermore, if after initial labeling, the learned model does not generalize well to new object instances or 3D viewpoints, the user can quickly relabel parts of the scene as necessary and see the improved results almost instantaneously as the learned models are updated online.

All pipeline components (including the features used during learning and inference) work directly on the volumetric data used for reconstruction, as opposed to requiring lossy and potentially expensive conversion of the data to depth image, point-cloud, or mesh-based representations. Additionally, the entire reconstruction, segmentation, classification and filtering pipeline runs in real-time, with the learning occurring in an online manner in the background.

We foresee numerous potential practical applications of our system. For example, for quickly gathering large numbers of labeled 3D environments for training large-scale visual recognition systems such as [Krizhevsky et al. 2012; Silberman and Fergus 2011]; generating personalized environment maps with semantic segmentations to be used for the navigation of robots or partially sighted people; recognizing and segmenting objects for augmented reality games which 'understand' the player's environment; and planning the renovation of a building, automating inventory, and designing interiors [Merrell et al. 2011].

Contributions. Our work builds on the existing body of research on 3D reconstruction, semantic modeling, and scene understanding in the following ways:

First, and foremost, we present a general-purpose 3D semantic modeling system, which fuses live 3D scene geometry as input, allows the user to segment and label surfaces interactively, continuously learns models of object classes, and performs classification and filtering, all in an online manner. This, to our knowledge, is the first time that *all* parts of a 3D semantic pipeline have been shown to work online and mostly in real-time, alongside acquisition. This is combined with a intuitive way for users to label physical objects using a multi-modal interface, which uses the metaphor of painting semantics onto the real-world, combined with speech commands.

To realize this system, we have made specific technical contributions along the way. While secondary, these extend the field of 3D scene understanding further. Specifically we present: (i) an online learning algorithm called streaming random forests that uses reservoir sampling to maintain fixed-length unbiased samples of streaming data; (ii) robust 3D rotation-invariant appearance and geometric features that are computed directly on the volumetric data; and (iii) an efficient mean-field algorithm for performing inference in a dynamically-changing *volumetric* random field model.

Our results demonstrate high-quality object segmentations on varied sequences (see Fig. 1, supplementary video, and Sec. 7). The segmentations are achieved with intuitive user interaction: seconds suffice to label individual objects or correct labels, just by touching surfaces and issuing voice commands, and a reasonably sized room can be scanned in and fully labeled in just a few minutes. Our entire semantic processing pipeline runs at interactive rates on a commodity desktop PC with a single GPU.

2. RELATED WORK

Acquiring 3D models of the real-world is a long standing problem in computer vision and graphics, dating back over five decades [Roberts 1963]. Since then, offline 3D reconstruction techniques have digitized cultural heritage with remarkable quality [Levoy et al. 2000], and given rise to world-scale, Internet-accessible, 3D maps reconstructed using street-side [Pollefeys et al. 2008], aerial [Hirschmuller 2008] and online photo collections [Snavely et al. 2006; Shan et al. 2013]. More recently, *real-time* or online 3D scanning emerged, with the rise of consumer depth cameras and GPU-based algorithms. Methods for real-time dense reconstructions, even over large physical scales, with only a single commodity depth or RGB camera have been demonstrated [Rusinkiewicz et al. 2002; Newcombe et al. 2011; Izadi et al. 2011; Newcombe et al. 2011; Chen et al. 2013; Nießner et al. 2013; Pradeep et al. 2013]. This has given rise to compelling new applications such as live 3D scanning, physically-plausible augmented reality, autonomous robot or vehicle guidance, and 3D fabrication.

Beyond low-level geometry acquisition and reconstruction, researchers have explored how to interpret the content of captured 3D models i.e. the objects or types of surfaces that are present. There is considerable interest and work on scene understanding and semantic modeling dating back as far as the reconstruction algorithm themselves, when primitive 'block world' representations were first devised [Roberts 1963]. Much work has happened in this area since. 2D techniques have been proposed that automatically partition an input RGB image into semantically meaningful regions, each labeled with a specific object class such as ground, sky, building, and so forth (e.g. [Shotton et al. 2006]). Others have focused on geometric reasoning to extract 3D structure from single RGB images (e.g. [Gupta et al. 2010]) or exlicit object detection (e.g. [Yao et al. 2012]). In the remainder we focus on methods that additionally use depth data rather than purely 2D input, but refer the reader to [Xiao 2014] for a review of 2D approaches.

With the advent of affordable depth sensors, there has been growing interest in working with RGB-D input [Silberman and Fergus 2011; Silberman et al. 2012; Couprie et al. 2013; Ren et al. 2012; Kähler and Reid 2013], as well as 3D point clouds [Brostow et al. 2008; Koppula et al. 2011; Anand et al. 2013; Stückler et al. 2013], meshes [Valentin et al. 2013] or voxel representations [Kim et al. 2013; Karpathy et al. 2013; Salas-Moreno et al. 2013; Häne et al. 2013]. Methods have been proposed to break down meshes into semantic parts [Chen et al. 2009], perform dense segmentation of reconstructed scenes [Lin et al. 2013; Sengupta et al. 2013; Ladickỳ et al. 2012; Valentin et al. 2013], even in an online manner [Herbst et al. 2014], localize objects in small scenes [Abdelrahman et al. 2013; Bonde et al. 2013; Karpathy et al. 2013; Lin et al. 2013], or replace objects with synthetic models [Salas-Moreno et al. 2013; Kim et al. 2012; Shao et al. 2012; Nan et al. 2012; Wang et al. 2014]. There has also been a wide body of research on capturing large and compelling datasets which have moved from traditional 2D object images to RGB-D and full 3D scenes [Xiao et al. 2010; Xiao et al. 2013; Geiger et al. 2012].

In the computer graphics literature there has been significant work on automatically segmenting 3D meshes into semantic parts [Chen et al. 2009; Kalogerakis et al. 2010; Shapira et al. 2010; Kim et al. 2013], including incremental depth camera-based methods [Shen et al. 2012]. Most of these methods consider only connected noise-free meshes, and geometric properties, ignoring the appearance. Furthermore, these techniques operate only on single objects, and do not operate in real-time. More recently, there has been relevant work on matching scan data to synthetic 3D model databases [Kim et al. 2012; Nan et al. 2012; Shao et al. 2012], with the aim to replace noisy point clouds with detailed CAD models. These approaches are compelling in that they increase final reconstruction fidelity and exploit repetition of objects to minimize the memory footprint. These systems first perform automatic or interactive segmentation of the scene into constituent parts which are then individually matched to the model database. However, these techniques require a model database to be built and learned offline, and the test-time matching techniques can take seconds to minutes to perform.

[Salas-Moreno et al. 2013] takes this concept a step further, by building an online SLAM system that can recognize objects and update the model live. However, the model database is still captured and generated offline. Only a single object class (chair) is recognized and it is unclear how the system can support larger surfaces such as floors, walls and ceilings. However, this system demonstrates the power of semantic recognition alongside the reconstruction process, improving relocalization, memory efficiency, and loop closure. This type of semantic information has also been explored in the context of bundle adjustment [Fioraio and Di Stefano 2013], and extended to sparse map representations [Ramos et al. 2008; Castle et al. 2007].

For outdoor scene labeling, much of the work has concentrated on classification of images [Brostow et al. 2008; Posner et al. 2009; Ladickỳ et al. 2012]. [Sengupta et al. 2013] generates a dense semantic 3D reconstruction but labeling is performed on the images and projected to the final model, which limits the use of full 3D geometry in their inference. [Häne et al. 2013] perform joint volumetric dense reconstruction and semantic segmentation, using a computationally complex global optimization. [Lin et al. 2013] decomposes outdoor scenes into semantic parts and employs 3D model matching techniques similar to [Kim et al. 2012; Nan et al. 2012; Shao et al. 2012] to create reconstructions from LiDAR data. None of these systems operate in a real-time or in an online manner.

[Silberman and Fergus 2011; Silberman et al. 2012; Couprie et al. 2013; Ren et al. 2012; Kähler and Reid 2013] attempt to label indoor scene images captured using RGB-D sensors. Classification or recognition is performed in image-space, along with 3D priors to aid segmentation. Again these systems fail to exploit full 3D geometry and are the counterpart of image-based segmentation but for RGB-D frames. [Valentin et al. 2013] exploits 3D meshes and geometric and appearance features for improved inference in outdoor and indoor scenes. [Kim et al. 2013] use a voxel-based conditional random field (CRF) for segmentation and occupancy-grid based reconstruction. However, these techniques are not efficient enough to be used in an online system, and operate only on coarse reconstructions.

Our approach differs from these systems in several compelling ways. Firstly our system runs entirely online and interactively, including data capture, feature computation, labeling, segmentation, learning and filtering. Second, our pipeline leads to robust and dense object labels directly on the acquired 3D model. Finally, in our system, the user is 'in the loop' during the labeling and training process, allowing the system to evolve to new object classes in an online fashion, and allowing the user to label a minimal amount and correct any mistakes interactively. This allows the user to rapidly build up models personalized to their spaces and goals.

3. SCENARIO OF USE

Our system offers a new way of capturing, labeling and learning semantic models, all in an online manner. In this section we walk through the main interactive capabilities of our system. Fig. 2 provides one example scenario of use. The user walks into a room with a number of chairs, tables, and smaller objects placed on the tabletops. The user is holding a consumer depth camera, such as a Kinect. Immediately they are able to capture the geometry of room, and generate a dense, globally consistent, 3D model, which can be viewed on a tablet screen or using heads-up displays.

The user can then reach out and touch surfaces in proximity, and issue a voice command to label these objects. Initially, the user points the camera downwards, towards the ground, and puts out their foot, and performs a 'stroke' gesture across the floor. She then issues a voice command – 'floor'. This user annotation then triggers our system to automatically propagate this label across the floor, as shown in Fig. 2 (top row). This leads to a smooth segmentation across the floor (as shown later we use both geometry and appearance cues to perform this segmentation). Now the user defined class 'floor' will be associated with these segmented voxels. The user next walks up to a chair, and again using a simple touch and speech command, annotates it appropriately. Again our algorithm smoothly segments this object, as shown in Fig. 2 (middle row, left). The user can also perform an 'enclose' gesture around smaller objects, where directly touching the surface would be more challenging, as in the

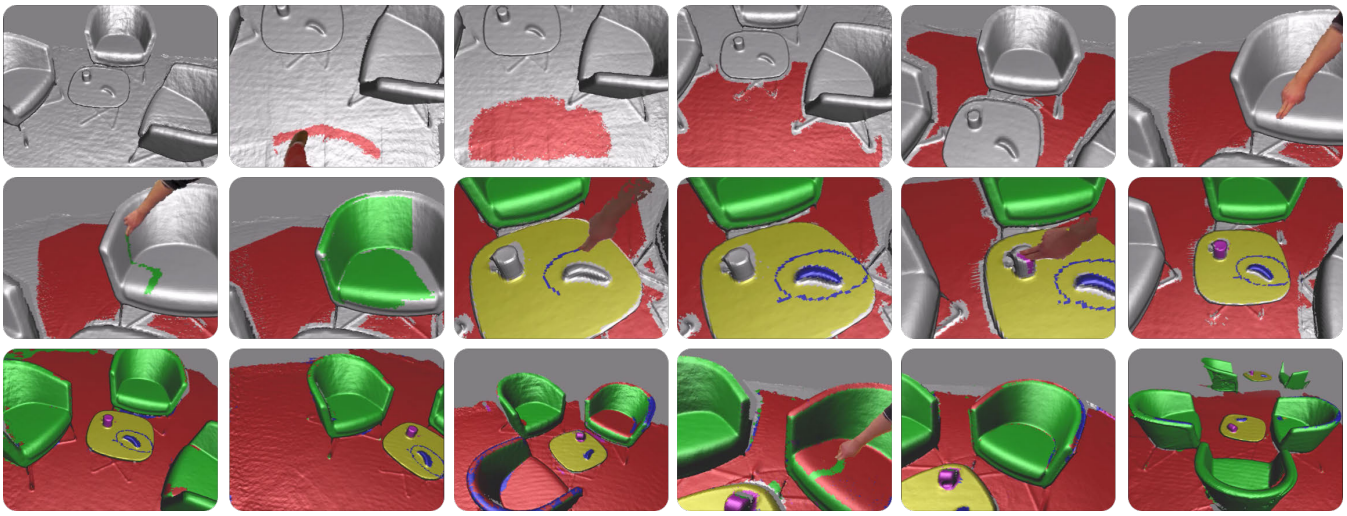

Fig. 2. Interacting with the system to capture geometry, label classes, and use these for online learning. See text for details.

case of the banana in Fig. 2 (middle row, right). At the end of the user annotation stage, the user has labeled the floor, one chair, the table, banana, and mug, and these objects are cleanly segmented in the 3D model, as shown in Fig. 2 (middle row, right). Interacting in this way firmly places the user 'in the loop' and is analogous to successful 2D segmentation tools such as GrabCut [Rother et al. 2004b], but where the interaction and segmentations are performed directly on the physical 3D world.

As new classes are labeled these are added as examples to a new online learning algorithm. When the user says the voice command 'test mode', the system enters a mode where the learning is activated. At this stage all unlabeled parts of the scene are automatically labeled based on the output of the learning. As shown in Fig. 2 (bottom row, left), the two chairs either side of the annotated chair are now automatically labeled correctly. As the user explores a new part of the scene, the floor is also correctly labeled. This includes another table with the same object classes, which is again correctly and automatically labeled. Notice in Fig. 2 (bottom row, middle), the chairs are incorrectly labeled. This is due to the fact that these chairs are actually yellow in color (see Fig. 1), and we use both appearance and geometry cues for learning (and have only trained on blue chairs so far). However, given that our system is fully online, we can correct these labels. Here we first say 'annotation mode' to enter user labeling mode, and again by simply touching the yellow chair, and saying the voice command 'chair', we can add this additional example to the training set. We then say 'training mode' to add the additional example, followed by the voice command 'test mode', which will now activate the learning, but based on this new training example. Note, the ability of our system to correct mistakes is crucial: no learning system is perfect, and our approach allows the user to see immediately where more training data (and thus user interaction) is required and to provide it in a natural way. Finally, we see how all yellow chairs are now also labeled correctly, as shown in Fig. 2 (bottom right) and Fig. 1.

4. SYSTEM PIPELINE

The overall architecture of our system is illustrated in Figure 3, and contains the following main components:

3D model acquisition engine. The first component of our system is 3D model acquisition. This component takes the color and depth image frame stream coming from the RGB-D sensor and fuses it on the GPU to generate a 3D model. We adopt the truncated signed distance function (TSDF) of [Curless and Levoy 1996] to fuse depth images into a 3D volume as in KinectFusion [Newcombe et al. 2011; Izadi et al. 2011]. To handle large-scale scenes we employ the hashed volumetric representation from [Nießner et al. 2013] and use a voxel resolution of 6mm^3.

User interaction. Our system allows for two modes of user interaction: touch-based interaction using hands and feet, and voice based commands. As in [Izadi et al. 2011], we detect touch by looking for large components in the observed depth image that differ from the raycasted model, which are close to the implicit surface. This allows the user to draw strokes on real-world objects and have those strokes appear as labels on screen. As in [Cheng et al. 2014], we use a standard voice recognition system to input labels such as 'table', 'chair', etc., and to recognize commands such as 'annotation mode', 'training mode' or 'test mode'.

Mean-field inference on CRF energy. At the heart of our semantic labeling approach is an efficient mean-field inference engine [Krähenbühl and Koltun 2011] applied to a dynamic conditional random field (CRF) model. The inference algorithm computes a per-voxel approximate posterior distribution over the set of object labels that the user provided. These distributions can be rendered as label maps and provided to the user as feedback. Both the user specified interaction hints and the predictions coming from the classification forest (see below) are integrated into the CRF through unary likelihood functions that operate on individual voxels. Pairwise terms in the CRF model ensure a smooth segmentation and allow the user labels to propagate outwards to object boundaries in the scene. In our application, the unary likelihoods and thus the CRF model changes dynamically from one frame to the next as (i) more data is acquired and the 3D model is updated, and (ii) the user continues to interact and specifies further labels. We show how the mean-field updates can be applied to such dynamic environments, can be implemented on the GPU, and the computational cost can be amortized over multiple video frames to enable an efficient implementation. This ensures

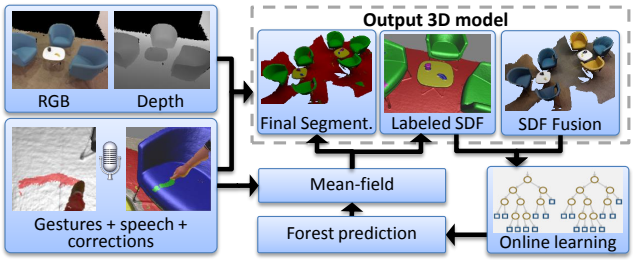

Fig. 3. Overview of 3D semantic modeling pipeline. See text for details.

that super real-time speeds can be maintained (one update of the messages requires ∼6 milliseconds), and as a by-product, results in a visually pleasing 'label propagation' effect.

Streaming random forest classifiers. Our system operates in two modes: training and test. The user can switch between these using voice commands. During the training mode, the labels that result from running the mean-field inference given the user-specified labels are fed into a streaming random forest algorithm that learns to predict likelihoods for assigning object labels to unlabeled parts of the 3D scene. The streaming decision tree algorithm continuously adapts in the background and will process any updates that have been made to the reconstruction volume, including any newly provided user labels. The forest employs a new type of 3D rotation invariant appearance feature that can be efficiently computed directly from the TSDF volume. In test mode, the forest is evaluated in parallel on the GPU for every visible voxel, and the results are used to update the unary likelihoods in the CRF. The mean-field inference then in turn produces a smoothed output to display to the user.

In the next sections, we first detail the volumetric mean-field inference and user interaction in Sec. 5, before describing the online learning in Sec. 6.

5. PAINTING AND SEGMENTING THE WORLD

In this section we describe the volumetric segmentation algorithm used to generate the smooth results that are presented to the user. Our results depend both on labels provided interactively by the user, and on inferences made by the learned classifier (described later in Sec. 6).

All parts of the pipeline directly work with the implicit surface data stored in the volume. Each voxel contains the TSDF (stored as a 32-bit float), a color value (24-bits, in Lab color space), and a weight (8-bits) for averaging distance values. Additionally, we also store class labels associated with: user annotations (8-bits), predictions from the online forest (8-bits), and the results of the mean-field inference before and after classification (two 8-bit values). Finally, the probability distributions from the mean-field inference are also directly stored in the voxels (as a 32-bit float per class). To allow for this additional data per voxel, we exploit the sparse nature of the TSDF using the hashing-based approach of [Nießner et al. 2013].

We formulate the problem of assigning semantic labels to voxels using a pairwise Conditional Random Field (CRF) [Lafferty et al. 2001]. While pairwise CRFs have been widely used for image labeling problems such as image segmentation, stereo and optical flow, a key distinguishing feature of our formulation is its *dynamic* nature that requires a special purpose inference routine. This allows us to deal with a continuously changing underlying 3D model (as more depth frames are fused), new user provided labels, and an on-the-fly trained decision forest predictor based likelihood function.

5.1 Dynamic Conditional Random Field Model

Each voxel i in the 3D reconstruction volume (denoted by $\mathcal{V}$) of the scene is represented by a discrete random variable x_i that represents the semantic class (e.g. floor, wall, table, mug) that the voxel belongs to. Note that the choice and number of labels will depend on the interactive user input. The posterior distribution over the labeling of voxels under the pairwise CRF factorizes into likelihood terms ψ_i defined over individual voxels and prior terms ψ_{ij} defined over pairs of random variables. The posterior is formally written as

$$P(\mathbf{x}|\mathbf{D}_t) \propto \prod_{i\in\mathcal{V}} \psi_i(x_i) \prod_{(i,j)\in\mathcal{E}} \psi_{ij}(x_i, x_j) \tag{1}$$

where $\mathbf{x}$ is the concatenation of the x_i for all $i \in \mathcal{V}$, $\mathbf{D}$ is the volumetric data at time t, and set $\mathcal{E}$ with $i \in \mathcal{V}$ and $j \in \mathcal{V}$ defines the neighborhood system of the random field[1]. To encourage smoother results, we employ a large neighborhood system which densely includes all voxels within a 6cm radius. This can be handled efficiently by our GPU-based volumetric mean-field inference algorithm described below.

An equivalent but perhaps more convenient definition can be reached by taking the negative log of the posterior. This gives the *energy* of the labeling under the CRF as:

$$E_t(\mathbf{x}) = \sum_{i\in\mathcal{V}} \phi_i(x_i) + \sum_{(i,j)\in\mathcal{E}} \phi_{ij}(x_i, x_j) + K \tag{2}$$

where ϕ_i encode the cost of assigning label x_i at voxel i, ϕ_{ij} are pairwise potentials that encourage neighboring (i.e. $(i,j) \in \mathcal{E}$) voxels to take the same label, K is a constant, and the conditioning on the data $\mathbf{D}_t$ is now implicit.

Note that the unary and pairwise terms in (2) will be constantly changing in our dynamic CRF. This is because (i) the volumetric reconstruction is constantly being updated with new observations, and (ii) the user is interacting with the environment and providing new labels. We next provide more details of the particular forms of these dynamic potentials.

Transition-sensitive smoothness costs. For our application, we employ a standard Potts model for the pairwise potentials, defined as:

$$\phi_{ij}(l, l') = \begin{cases} \lambda_{ij} & \text{if } l \neq l' \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

In the 2D segmentation domain, the cost λ_{ij} of assigning different labels to neighboring pixels is generally chosen such that it preserves image edges [Boykov et al. 2001; Rother et al. 2004a]. Inspired from these edge-preserving smoothness costs, we make the label discontinuity cost λ_{ij} dependent on a number of appearance and depth features:

$$\lambda_{ij} = \theta_{\mathbf{p}} e^{-\|\mathbf{p}_i - \mathbf{p}_j\|_2} + \theta_{\mathbf{a}} e^{-\|\mathbf{a}_i - \mathbf{a}_j\|_2} + \theta_{\mathbf{n}} e^{-\|\mathbf{n}_i - \mathbf{n}_j\|_2} \tag{4}$$

where $\mathbf{p}_i$, $\mathbf{a}_i$ and $\mathbf{n}_i$ are respectively the 3D world coordinate position, RGB appearance, and surface normal vector of the reconstructed surface at voxel i, and $\theta_{\mathbf{p}}$ $\theta_{\mathbf{a}}$ and $\theta_{\mathbf{n}}$ are hand-tuned parameters. Note that the surface normals are calculated from the gradient of the TSDF at the zero-crossing. Observe that as the 3D model is updated from one frame to the next, the appearance and surface normals associated with the voxels change. The energy landscape thus continuously changes over time.

[1] We have not listed the data $\mathbf{D}_t$ as an argument in the potential functions ψ_i and ψ_{ij} for the sake of a simpler and uncluttered exposition.

Initialization of the Unary Costs The unary potentials ϕ_i of the CRF model defined above are *initialized* for all $i \in \mathcal{V}$ by specifying a fixed cost that initially encourages all voxels to take the background label:

$$\phi_i(l) \leftarrow \begin{cases} 0 & \text{if } l \text{ is the background label} \\ \theta_{\mathrm{bg}} & \text{otherwise.} \end{cases} \quad (5)$$

Note that each unary can be thought of as a *table* of values (one entry for each $l \in \mathcal{L}$). During operation of our system, the entries in these tables are gradually replaced based on user interactions and predictions from the streaming random forest based classifier that encodes the cost for a voxel being assigned a particular label, as described next.

User 'Paint' Interactions. As illustrated in Fig. 2, our system allows two types of user interaction gestures for labeling the world. The first user interaction supported is a 'paint' (or 'stroke') gesture. The user reaches out, and touches the surface of the object they want to label. Note that the user need not precisely label every voxel belonging to the object, and can instead roughly mark a small set of voxels. We denote the set of voxels that the user has stroked as $\mathcal{H}_{\mathrm{S}}$. The user also specifies a semantic label l_{S} through speech recognition, and can specify an existing label or a new one, in which case the label set $\mathcal{L}$ is enlarged. These labelings are used to update the unary potentials to enforce that these voxels take the specified semantic label. The update is applied to all voxels $i \in \mathcal{H}_{\mathrm{S}}$ as

$$\phi_i(l) \leftarrow \begin{cases} 0 & \text{if } l = l_{\mathrm{S}} \\ \infty & \text{otherwise.} \end{cases} \quad (6)$$

Multiple such labelings with different regions $\mathcal{H}_{\mathrm{S}}$ and labels l_{S} can be provided in sequence. In the case of incorrect propagation due either to wrongly placed user strokes or to problems with the mean-field inference, the user can specify another stroke labeling. This will overwrite any existing labeling and thus update the unary and allow the new label to propagate.

User 'Enclose' Interactions. The second form of user interaction supported is the 'enclose' gesture, where the user reaches out and draws a rough enclosing circle around the object they want to describe. This interaction mode is most useful for small objects such as bananas and pens. Again, voice is used to provide the semantic label l. To obtain an accurate labeling, we follow an approach similar to [Rother et al. 2004a]. First, the user annotation is projected into the current frame's input image. On the CPU, a Gaussian mixture model (GMM) is fit to the colors in the foreground and background regions. Foreground is taken as the interior of the convex hull of the user annotations, and background as the rest of the image. Then we transfer the GMMs to the GPU where a shader computes in parallel the probability

$$P_{\mathrm{E}}(\mathrm{fg}|\mathbf{a}_i) = \frac{P(\mathbf{a}_i|\mathrm{fg})}{P(\mathbf{a}_i|\mathrm{fg}) + P(\mathbf{a}_i|\mathrm{bg})} \quad (7)$$

based on the voxel's foreground (fg) and background (bg) color likelihoods (and assuming uniform priors). This is computed for all voxels i in a bounding volume surrounding the user annotations.

Inference is performed within this bounding volume to estimate which of the two labels (foreground-background) is most likely to be assigned to each voxel. This is used to update the unary cost in the energy defined over the full 3D model as

$$\phi_i(l) \leftarrow \begin{cases} \log P_{\mathrm{E}}(\mathrm{fg}|\mathbf{a}_i) & \text{if } l = \mathrm{fg} \\ \log(1 - P_{\mathrm{E}}(\mathrm{fg}|\mathbf{a}_i)) & \text{if } l = \mathrm{bg} \end{cases} \quad (8)$$

Learned Class Predictions. There is one final source of updates to the unary costs. We describe below in Section 6 how a decision forest is able to learn and make predictions about object classes in newly observed regions of the world that are not hand-labeled. The output of the forest is a prediction of the distribution $P_{\mathrm{F}}(x_i = l \mid \mathbf{D})$ over semantic labels $l \in \mathcal{L}$ at voxel i. For voxels i that have not been hand labeled using either of the interactions described above or label-propagation by mean-field, this distribution is used to update the unary likelihood costs as:

$$\phi_i(l) \leftarrow -\log P_{\mathrm{F}}(x_i = l \mid \mathbf{D}) \, . \quad (9)$$

5.2 Efficient Mean-Field Inference

Given the unary and pairwise terms defined above, the labeling can be propagated through the volume by inferring the optimal labeling $\mathbf{x}$ given the pairwise energy functions. The Maximum a Posteriori (MAP) labeling for pairwise energy functions such as the one defined in equation (2) could be computed using standard graph cut based move making algorithms like α-expansion and $\alpha\beta$-swap. However, these algorithms are intrinsically sequential, and it is hard to tailor them to high throughput architectures like GPUs without significant engineering overheads [Vineet and Narayanan 2008].

Instead, we propose an online volumetric mean-field inference framework that efficiently infers the approximate maximum posterior marginal (MPM) solution of our energy. While based on [Krähenbühl and Koltun 2011], we make two key technical contributions. Firstly we show how such a volumetric energy minimization can be implemented efficiently on the inherently parallel architecture of the GPU. Secondly, we exploit the fact that the energy landscape usually changes only gradually from one frame to the next. This allow us to amortize the optimization cost over multiple frames. This not only enables the system to run at a high frame rate, but also results in a pleasing result to the user as user labels appear to gradually propagate out from the initial strokes until they accurately delineate the objects boundaries based on the energy function (2) of the model.

The mean-field optimization [Krähenbühl and Koltun 2011] proceeds as follows. We introduce a probability distribution $Q(\mathbf{x})$ that approximates the original distribution $P(\mathbf{x})$ (1) under the KL-divergence $D(Q||P)$. Further, we choose a factorized distribution $Q(\mathbf{x})$ such that the marginal of each random variable is independent, i.e. $Q(\mathbf{x}) = \prod_i Q_i(x_i)$. Taking the fixed point solution of the KL-divergence [Koller and Friedman 2009], we obtain the following mean-field update:

$$Q_i^t(l) \leftarrow \frac{1}{Z_i} e^{-M_i(l)} \quad (10)$$

$$M_i(l) = \phi_i(l) + \sum_{l' \in \mathcal{L}} \sum_{j \in \mathcal{N}(i)} Q_j^{t-1}(l') \phi_{ij}(l, l') \quad (11)$$

$$Z_i = \sum_{l \in \mathcal{L}} e^{-M_i(l)} \quad (12)$$

where $l \in \mathcal{L}$ is a label taken by random variable x_i, $Q_i^t(l)$ denotes the marginal probability at iteration t of variable x_i taking label l, $\mathcal{N}_i$ denotes the set of neighbors of i (i.e. $j \in \mathcal{N}_i \Leftrightarrow (i,j) \in \mathcal{E}$), and Z_i normalizes the distribution. After iterating the updates (10) to iteration T, the output MPM estimates can be obtained as

$$x_i^\star = \operatorname*{argmax}_{l \in \mathcal{L}} Q_i^T(l) \, . \quad (13)$$

Given unlimited computation, one might run multiple update iterations until convergence.[2] However, in our online system, we assume that the next frame's updates to the volume (and thus to the energy function) are not too radical, and so we can make the assumption that the Q_i distributions can be temporally propagated from one frame to the next, rather than re-initialized (e.g. to uniform) at each frame. Thus, running even a *single iteration* of mean-field updates per frame effectively allows us to amortize an otherwise expensive inference operation over multiple frames and maintain real-time interactive speeds. Note that this effectively means that the t variable above becomes both the frame number and the mean-field iteration count. Furthermore, this approach results in a smooth propagation of the inferred labels through the world from one frame to the next, providing continuous feedback to the user.

Integration of Forest Predictions. As described above, the output of the streaming random forest (Section 6) is used to update the unary distributions, which will, over several frames, impact the final segmentation that results from the mean-field inference. However, to speed up convergence, we propose an additional step that exploits our temporal propagation of the Q distributions. Rather than simply propagating the Q_i^{t-1}s from the previous frame, we instead provide the next iteration of mean-field updates with a weighted combination of Q_i^{t-1} and the forest prediction $P_{\mathrm{F}}(x_i = l \mid \mathbf{D})$. We thus use

$$\bar{Q}_i^{t-1}(l) = \gamma Q_i^{t-1}(l) + (1-\gamma) P_{\mathrm{F}}(x_i = l \mid \mathbf{D}) \qquad (14)$$

in place of the Q_i^{t-1} in (11), where γ is a weighting parameter. In practice, this step appears to result in considerably quicker updates to the segmentation result given the forest predictions.

5.3 Implementation on the GPU.

To achieve real time performance, we run the mean-field inference on the GPU. Each GPU thread independently computes the $\bar{Q}_i^t$ distribution in parallel for all voxels i based on the previous frame's estimates $\bar{Q}_i^{t-1}$ and the forest probabilities $P_{\mathrm{F}}(x_i \mid \mathbf{D})$. We employ three GPU shaders, which are executed in sequence. The first shader calculates $\bar{Q}$ according to (14). The second shader applies the mean-field update (10) to the class probabilities stored at each voxel. This entails looking up the unary and adding in the pairwise terms for all neighbors of the voxel. While previous methods such as [Krähenbühl and Koltun 2011] have considered fully connected CRFs, a reasonable trade-off between accuracy and speed was achieved by using a radius of 6cm to determine the neighborhood. The third shader finally evaluates the MPM solution using (13) for the current time t.

Each thread is processed independently, and the computation required is proportional to the neighborhood size times the number of classes. We do other optimizations to improve the speed of mean-field inference on the GPU. These take the form of adaptive scheduling of message updates. First, we do not apply the mean-field update for voxels where the probability for a particular class is very high as this is unlikely to change the final labeling. Second, we do not perform the mean-field updates for voxels that lie away from the surface (i.e. outside the truncation region of the TSDF). Finally, we look at the neighbors only if they lie within the truncation region. All these updates are important to enable inference in real time.

[2]If applied sequentially on a fixed energy function, the mean-field inference comes with some convergence guarantees [Krähenbühl and Koltun 2011]. These do not apply to our algorithm as our algorithm is dynamically updated in each frame, but this does not appear to be a problem in practice.

6. LEARNING TO LABEL THE NEW WORLD

The previous section detailed our efficient mean-field inference that, given a set of unary and pairwise potentials, optimizes an energy function to produce spatially-smooth segmentations. We have seen how the unary potentials in (2) are initially encouraged to be background (5), and later are updated based on user interactions (6, 8). This section details how we can learn and infer distributions $P_{\mathrm{F}}(x_i = l \mid \mathbf{D})$ that can be used as another form of unary (9) to allow the CRF inference to automatically predict smooth segmentations for *unlabeled* parts of the world. We apply a new approach called *streaming random forests* to this task. These forests are extremely fast both to train and test, and are capable of learning online from a stream of labeled training voxels, and being updated to correct for mistakes. Another contribution in this section is a new set of features called voxel-oriented patches (VOPs). VOPs can be efficiently computed from the raw TSDF volume and thus avoid the expense of computing an explicit surface reconstruction. They are also designed to be 3D rotation invariant allowing the inference to generalize well across 3D object rotations in the world. The following sections elaborate on our approach.

6.1 Decision Forests

Decision forests [Breiman 2001; Criminisi and Shotton 2013] have proven successful for many applications. Typically trained offline with large datasets, forests can learn to recognize the trained object classes in new scenes. However, we cannot hope to employ such offline learning approaches to our scenario due to the often long training times (typically hours or even days), and our desire to allow interactive updates and corrections to the classifier. We thus turn to online forest learning [Domingos and Hulten 2000; Bifet et al. 2009; Saffari et al. 2009]. While typically less accurate than an offline-trained forest, online learning supports incremental (and thus more interactive) updates given new or improved training data, and can require less memory to train. Our new streaming decision forests framework extends [Saffari et al. 2009] to use reservoir sampling [Vitter 1985]. This allows us to maintaining a fixed-size unbiased sample of all training data seen so far and avoid discarding samples observed early on during training, resulting in faster and more accurate classifiers. Our approach also requires considerably less memory to train.

We first briefly review the standard offline (or 'batch') decision forests (see e.g. [Criminisi and Shotton 2013] for more details), before describing [Saffari et al. 2009], and finally our new streaming decision forest algorithm.

Offline Forests. A forest comprises an ensemble of decision trees. Each tree comprises binary split nodes and leaf nodes. At test time, starting at the root, a left/right decision is made for voxel i according to the evaluation of a binary split function $f(i;\theta) \in \{\mathrm{L}, \mathrm{R}\}$ with learned parameters θ. As described in Sec. 6.3, parameters θ are used to specify the particular features used at this node. According to the result, the left or right child branch is followed, and the process is repeated for each split node encountered, until a leaf node is reached. At the leaf node a stored distribution $P_{\mathrm{F}}(x_i = l \mid \mathbf{D})$ is looked up and used to update the voxel's unary as described above.

Each tree is trained independently (offline) on subsets of the training data. A set $\mathcal{S}_n$ of examples is provided to the root node $n = 0$. Example set $\mathcal{S}$ consists of example pairs (i, l) where i represents the training voxel index, and l is the associated class label provided by user interaction. A set Θ_n of candidate binary split function parameters θ is proposed randomly. Each candidate split induces a partitioning of the set of examples into left and right

subsets, $\mathcal{S}_n^{\mathrm{L}}(\theta)$ and $\mathcal{S}_n^{\mathrm{R}}(\theta)$. We can compute the *information gain* objective as

$$G(\mathcal{S}, \mathcal{S}^{\mathrm{L}}, \mathcal{S}^{\mathrm{R}}) = H(\mathcal{S}) - \sum_{d \in \{\mathrm{L},\mathrm{R}\}} \frac{|\mathcal{S}^d|}{|\mathcal{S}|} H(\mathcal{S}^d) \,, \qquad (15)$$

where $H(\mathcal{S})$ is the Shannon entropy of the distribution of labels l in $\mathcal{S}$. The (locally) optimal parameters can then be obtained as the maximization

$$\theta_n := \operatorname*{argmax}_{\theta \in \Theta_n} G(\mathcal{S}_n, \mathcal{S}_n^{\mathrm{L}}(\theta), \mathcal{S}_n^{\mathrm{R}}(\theta)) \,. \qquad (16)$$

Having found the best split parameters for node n, the tree growing proceeds greedily for the left and right children until a predefined stopping criterion is reached.

Online Random Forests. We next review the online forest algorithm of [Saffari et al. 2009]. Given a current tree structure (initially this will be a single root node), any incoming observations (i, l) are passed down the tree until they reach a leaf node. At each leaf there now exists a set Θ_n of candidate features, and for each feature $\theta \in \Theta_n$, a set of left and right class label statistics are stored. These histogram statistics are sufficient to compute the information gain (15). Based on the output of $f(i; \theta)$, the new observation's label l is used to update the relevant (i.e. left or right) statistics for each θ. If the updated leaf has now seen a large enough number of observations, and if at least one of the candidate features gives a good local optimum of (16), then the leaf will be split, and two child nodes with their statistics created. Hoeffding trees [Domingos and Hulten 2000] work in a similar way, but use a Hoeffding bound to decide whether or not to split.

Streaming Random Forests. We propose a new method called the 'streaming random forest'. This can be seen as an extension of [Saffari et al. 2009] with reservoir sampling. Reservoir sampling [Vitter 1985] is a method of storing an unbiased sample of a fixed maximum size from a stream of data. If the incoming data were i.i.d., then we could of course simply store the first K samples and we would be done. However, in practice our stream of training data is very much not i.i.d., due in part to considerable correlation between one frame and the next. Using a reservoir accumulated over a potentially large temporal window of observations can thus smooth out any imbalance in the distribution of incoming samples and thus improve the quality of the classifier.

A reservoir maintains a list $\mathcal{T}$ of at most K examples, and a count m of the total number of examples observed so far. Given a new labeled observation (i, l), if $m < K$ then the example is simply appended to $\mathcal{T}$. If instead $m \geq K$ and thus the reservoir is full, then a uniform random integer $k \in \{1, \ldots, m\}$ is chosen. If $k \leq K$ then reservoir entry k is replaced by (i, l), otherwise the observation is discarded. Finally, m is incremented, ready for the next observation. Note that the probability of retaining a new sample is $\frac{K}{m}$, and so as m increases we are more likely to discard new observations. However, in the limit of M samples, the probability of any individual sample remaining in the reservoir is exactly uniform at $\frac{K}{M}$.

In [Saffari et al. 2009], at each current leaf node n, it was necessary to store a set of class label statistics at the potential left and right *children* that would result for *every* candidate split function. We propose instead to store a *single* reservoir $\mathcal{R}_n = (\mathcal{T}_n, m_n)$ at each potential *parent* (i.e. current leaf), which saves considerable memory. By storing the count m_n of observations separately, the reservoir allows us to maintain a fixed maximum size set $|\mathcal{T}| \leq K$ of unbiased samples of the incoming observations, where K is a parameter of the method. We show below how the reservoir representation allows us to efficiently compute a good approximation of the information gain (15) from only a small subset of the observations.

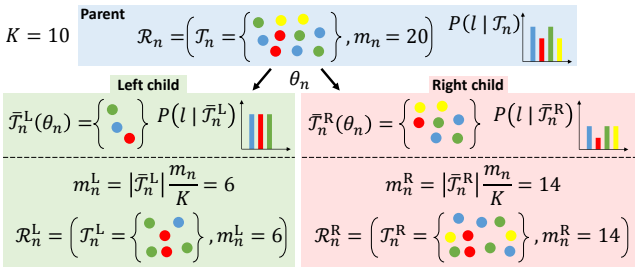

Fig. 4. A toy example of splitting a reservoir. Top: a reservoir $\mathcal{R}_n$ of capacity $K = 10$ that has observed $m_n = 20$ samples. The dots represent examples i and their colors the labels l. The empirical distribution $P(l|\mathcal{T}_n)$ is computed from the labels (colors) in $\mathcal{T}_n$. Bottom, above dashed line: For any setting of the split parameters θ_n, the list $\mathcal{T}_n$ can be efficiently partitioned into two halves, $\bar{\mathcal{T}}_n^d$ for $d \in \{\mathrm{L}, \mathrm{R}\}$. Given these, the corresponding empirical distributions can be computed as before. Bottom, below dashed line: Once the optimal θ_n has been chosen, we can resample to generate child reservoirs: the child counts m_n^d are computed, and then m_n^d samples from $\bar{\mathcal{T}}_n^d$ are drawn and added to the new reservoir $\mathcal{R}_n^d$.

Optimizing the objective. When it is decided to split node n, we must evaluate the objective (15) with the example sets $\mathcal{S}$s now replaced by reservoirs $\mathcal{R}$s. This requires a sweep through reservoir $\mathcal{R}_n$ for each $\theta \in \Theta_n$. This can be performed efficiently, as follows.

The procedure is illustrated in Fig. 4. At the parent n we simply define $|\mathcal{R}_n| = m_n$, and the distribution required to evaluate the entropy $H(\mathcal{R}_n)$ can be calculated by normalizing the histogram of class labels l in $\mathcal{T}_n$. The remaining terms in the objective are $|\mathcal{R}_n^d|$ and $H(\mathcal{R}_n^d)$ for each child $d \in \{\mathrm{L}, \mathrm{R}\}$. These can be computed *without explicitly computing the sub-reservoirs* $\mathcal{R}_n^d$ (cf. 'Splitting the reservoirs' below, and compare above and below the dashed line in Fig. 4). First, the examples $(i, l) \in \mathcal{T}_n$ are partitioned into left and right subsets $\bar{\mathcal{T}}_n^d$ using split function $f(i; \theta)$. We can then efficiently compute the 'effective value' for child count $|\mathcal{R}_n^d|$ as

$$\bar{m}_n^d = |\bar{\mathcal{T}}_n^d| \max(1, \frac{m_n}{K}) \,, \qquad (17)$$

and compute the entropies $H(\mathcal{R}_n^d)$ from $P(l|\bar{\mathcal{T}}_n^d)$, the normalized histogram of labels l in $\bar{\mathcal{T}}_n^d$.

Splitting the reservoirs. The slight extra cost of the above sweep compared to [Saffari et al. 2009] does come with a further benefit beyond reduced memory consumption. Having chosen the optimal θ_n based on (15), rather than throw away the statistics stored at the node (as done in [Saffari et al. 2009]), we can instead split the reservoir $\mathcal{R}_n$ into sub-reservoirs $\mathcal{R}_n^d$. For each of $d \in \{\mathrm{L}, \mathrm{R}\}$, we start with a new, empty sub-reservoir $\mathcal{R}_n^d$ of capacity K. We first compute the partitions $\bar{\mathcal{T}}_n^d$ as described above. To the nearest integer, the number of examples in $\mathcal{R}_n^d$ should be $m_n^d = \lfloor |\bar{\mathcal{T}}_n^d| \max(1, \frac{m_n}{K}) + 0.5 \rfloor$. We thus draw m_n^d random samples (with replacement) from $\bar{\mathcal{T}}_n^d$, and add each into the new reservoir $\mathcal{R}_n^d$ in the standard fashion. Except for rounding errors, this gives us the reservoirs containing unbiased sets of examples for the left and right children.

Advantages of reservoirs. The ability not to throw the statistics away gives us a considerable head start compared to [Saffari et al. 2009] in which each new node must start with an initially empty

set of statistics after being created. This in turn means that the new nodes can themselves be split much sooner with fewer additional observations. Furthermore the reduced memory consumption gives the potential for the algorithm to scale to larger trees. In practice for our scenario, this new approach gives a considerable improvement in accuracy, which is later quantified in experiments on standard classification datasets.

On average, our CPU implementation of the streaming decision forest training process is able to process the addition of 5000 new samples in 30ms per tree. In order not to stall the interactive system while the learner is updating its structure, the learning process is running as a background task, and can make use of all the available CPU cores. Note that we split at most 4 leaves per epoch, as empirically this helps gather better statistics of the split candidates. GPU implementations (e.g. [Sharp 2008]) could drastically speed-up the learning process, and also allow more samples to be processed.

Sampling Training Voxels. In order for the online classifier to update its structure and the distributions stored in the leaves, training samples must be fed to the classifier. We sample at random an equal number of voxels for each class from regions of the volume that have been hand-labeled. Each voxel is assigned its current label as given by the mean-field inference, allowing us to learn both from the explicitly user-labeled regions ($\mathcal{H}_\mathrm{S}$ and $\mathcal{H}_\mathrm{E}$) and also from the labels that have been propagated by the CRF inference.

Given the voxels in the current view frustum, an equal number of random samples (1000 in our implementation) of each class are extracted. Samples are extracted in small batches spaced by constant time steps, except for when the user is interacting with the world.

6.2 Voxel-Oriented Patch Features

The ability of the decision forest to learn to distinguish different object classes depends on the availability of discriminative features. A variety of features have been used with great success in the past. For 2D applications, these have included pixel comparisons [Lepetit and Fua 2006], texton region integrals [Shotton et al. 2006], and invariant descriptors such as SIFT [Lowe 1999] and HOG [Dalal and Triggs 2005]. Features that describe 3D point clouds and meshes have also been proposed, including rotation invariant spin images [Johnson 1997], SfM point-cloud derived features [Brostow et al. 2008], and difference of normals [Ioanou et al. 2012]. Such features are typically designed with certain invariances in mind, including additive photometric invariance, and 2D or 3D rotation invariance. Such invariances can help the classifier by reducing the amount of training data required (though, too much invariance can lead to loss of discriminative power).

For our real-time application, speed is crucial, and given the dynamic nature of the scene, we do not have the time to first extract a mesh or point-cloud before computing features. As one of our key contributions, we thus propose a new type of feature, the *voxel-oriented patch* (VOP) that can be efficiently computed *directly* from the TSDF volume, and are 3D rotation invariant. Our implementation can efficiently handle the computation of features for millions of voxels.

For a voxel i of interest at position $\mathbf{p}_i$, a VOP V_i is extracted as follows; see also Fig. 5. The voxel's normal $\mathbf{n}_i$ is calculated directly from the gradient of the TSDF values, and defines a plane $(\mathbf{p} - \mathbf{p}_i) \cdot \mathbf{n}_i = 0$. Choosing an arbitrary vector on this plane as an initial x-axis, and a third, orthogonal vector as a y-axis, we form an image patch of size $r \times r$ that contains the color values stored in the TSDF on the plane. To reduce the effect of illumination (especially specularities) we employ the CIELab color space, storing the raw

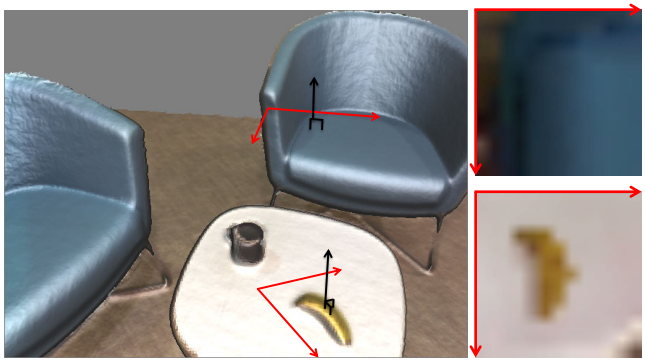

Fig. 5. Illustration of Voxel-Oriented Patches (VOPs). VOPs encode height and color information directly from the TSDF without requiring explicit surface extraction. They are also fully 3D rotation invariant. Note that for illustration purposes the VOP shown above store RGB, which is not the setup used in our system as described in 6.2.

L component as well as $a/(a+b)$ and $b/(a+b)$. To complement the appearance information, each V_i also stores a signed distance to the dominant horizontal surface present in the scene. Note that the resolution of the patch size (i.e. mm per pixel) can be independent of the resolution of the TSDF reconstruction volume; our current implementation uses $r = 13$ with a resolution of 10 mm per pixel.

To achieve rotation invariance, we compute a histogram of intensity gradients in the image patch, and rotate the patch to align with the strongest gradient orientation, in precisely the same way as SIFT [Lowe 1999].

We end up with a VOP V_i that can now store discriminative information about the local appearance around voxel i with full 3D rotation invariance. Examples are given in Figure 5. Note that these features exploit the fact that the TSDF volumetric integration process 'spreads' the color information along the camera ray within the truncation window, such that the voxel i does not need to be precisely aligned with the implicit surface to contain interesting and discriminative information. If the patch intersects regions of empty space (those with a TSDF weight of zero), the relevant pixel in the patch is flagged with an 'invalid' color value, which provides for a weak geometric cue.

A single VOP is computed on the GPU at each voxel being processed by the forest. This is done without having to extract the surface explicitly from the TSDF volume. Note that the forest will evaluate split functions (see below) with different parameters on the same VOP as it descends its trees; the VOP needs only be computed once per voxel for a full evaluation of the forest.

Possible variants of VOPs traverses the volume along the normal direction towards the nearest zero-crossing in the TSDF and then either take the color and the distance along the normal as the VOP pixel value. This was found to be too expensive for our real-time requirements and were not explored further. Other potential variants to explore could make use of the raw TSDF value as a way to describe the geometric properties of the object to learn.

6.3 Split Functions

Our split functions come in three varieties: VOP-based, surface orientation, and world height.

The VOP-based split functions work as follows. Each split node in the forest performs a comparison by either taking the raw value

or pairwise operations between two VOP pixels' colors

$$f(i;\theta) = V_i(x_1, y_1, c_1) \;\; op \;\; V_i(x_2, y_2, c_2) > \tau \qquad (18)$$

where $op \in \{+, -\}$, (x_1, y_1) and (x_2, y_2) specify particular pixels in the VOP, c_1 and c_2 specify color channels, τ is a threshold, and $\theta = (\text{VOP}, op, x_1, y_1, c_1, x_2, y_2, c_2, \tau)$ define the parameters of this VOP-based split function. The second type of split function applies a threshold to the dihedral angle between the surface normal and the world up vector. These are designed to help separate classes such as wall and table which are often flat and textureless, and thus would be difficult to split using a texture or geometry-based feature such as a VOP. The final split function applies a threshold to the world height, allowing the classifier to efficiently deal with 'easy' classes such as the floor.

The choice between the three types of split function, and the parameters θ thereof, is determined during the learning process. Note that the classifier will make the easiest choice available to it. For example, if world height is discriminative for the current training examples, it will use it. If this results in mistakes (for example, the user moves an object from a table to the floor) then the user can correct the labels and the forest can then be updated to use other features that are more appropriate for this object class.

6.4 Efficient test time classification

The learned decision forests are extremely efficient to evaluate at test time. Our GPU-based implementation can handle approximately 17 million voxel classifications per second. Which compares well with the roughly 3-10 million voxels visible the view frustum at once [Nießner et al. 2013]. To maintain interactive rates, we batch the visible voxels randomly and only test one batch at each frame. Each voxel is assigned a flag to say whether it has been classified yet or not. Once all voxels have been processed, the process repeats, applying the forest to the volume which may have been updated in the mean-time.

7. RESULTS

We first present qualitative results, then discuss efficiency of our system, and finally provide quantitative results.

7.1 Qualitative results

Given the interactive and sequential nature of our approach, the accompanying video best demonstrates the capabilities of our system. Fig. 12 shows results from four recorded datasets. In both LIVINGROOM and KITCHEN the user labels one half of the scene interactively. The model is learned online, and tested on the second part of the scene. The DESK sequence is an office table with computer monitor and objects such as phones, and textured objects such as books. After user labeling, the test sequence is based on the user reshuffling the desk, and automatically labeling the objects. BEDROOM is a bed containing multiple objects, which is user labeled. The user returns to this bedroom, from a completely different viewpoint and reconstructs the scene, and objects are automatically segmented using the online learned model. All these examples include a variety of objects, some of which are uniformly colored, whilst others are heavily textured. The video also demonstrates the power of our online approach, in particular allowing for continual correction of the model during segmentation and learning.

Propagation of User Labels. The user strokes the surface of objects in the physical world. Our system interprets such gesture as a paint stroke, and voice input is used to associate an object class label with the corresponding 'touched' voxels. Then, our mean-field inference engine (see Sec. 5) propagates these labels through the reconstructed scene, very efficiently. Thanks to the pairwise potentials (3) the result is a spatially smooth segmentation that adheres to object boundaries. Examples of label propagation are shown in Fig. 6.

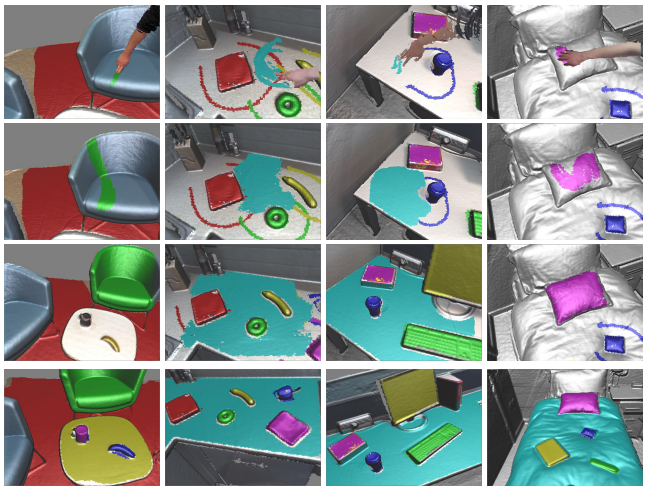

Fig. 6. Label propagation. Our efficient inference engine smoothly propagates class labels from the voxels touched by the user to the rest of the volume. Here we show examples taken from three environments of the coarse user labels (top row) and three time steps (middle three rows) as the mean-field updates are applied over time. The pairwise terms in our energy encourage a smooth segmentation that respects object boundaries. The last row shows the final label propagation results for all hand-labeled objects.

Forest Predictions. Our system learns a streaming random forest classifier in a background CPU thread given the labels provided by the user. At some point, the user selects 'test mode', and the forest starts classifying all voxels. In Fig. 7 we illustrate the resulting intermediate predictions (the 'unaries' in the middle column), and compare them with the final, smoothed result obtained by running the mean-field inference on these unaries (right column). Let us focus on LIVINGROOM in the figure. In the first row, we see an example result in the region used for training the forest. Here, all the chairs are blue, and as expected, the unaries and mean-field results are of very high quality. The second row shows a failure case (highlighted by the arrows) in a region of the environment that was not used for training. The seat of the yellow chair gets confused with the floor, since only a blue chair was used for learning. At this point the user makes a stroke interaction (not shown) to correct the labels of the chair, and the forest is updated. After correction (row three) the chair can be correctly recognized. Note, the ability of our system to correct mistakes is crucial: no learning system is perfect, and our approach allows the user to see immediately where more training data (and thus user interaction) is required and to provide it in a natural way. This final row illustrates the generalization capabilities of our system to previously unavailable viewpoints. Scene 2 shows slightly noisier predictions from the forest, due in part to more challenging lighting conditions and problems with holes in the reconstruction (no user corrections were made in this sequence). The mean-field inference does a good job of smoothing and improving the final result, though some errors do remain that we expect could be corrected through simple further user strokes (an advantage of an online system).

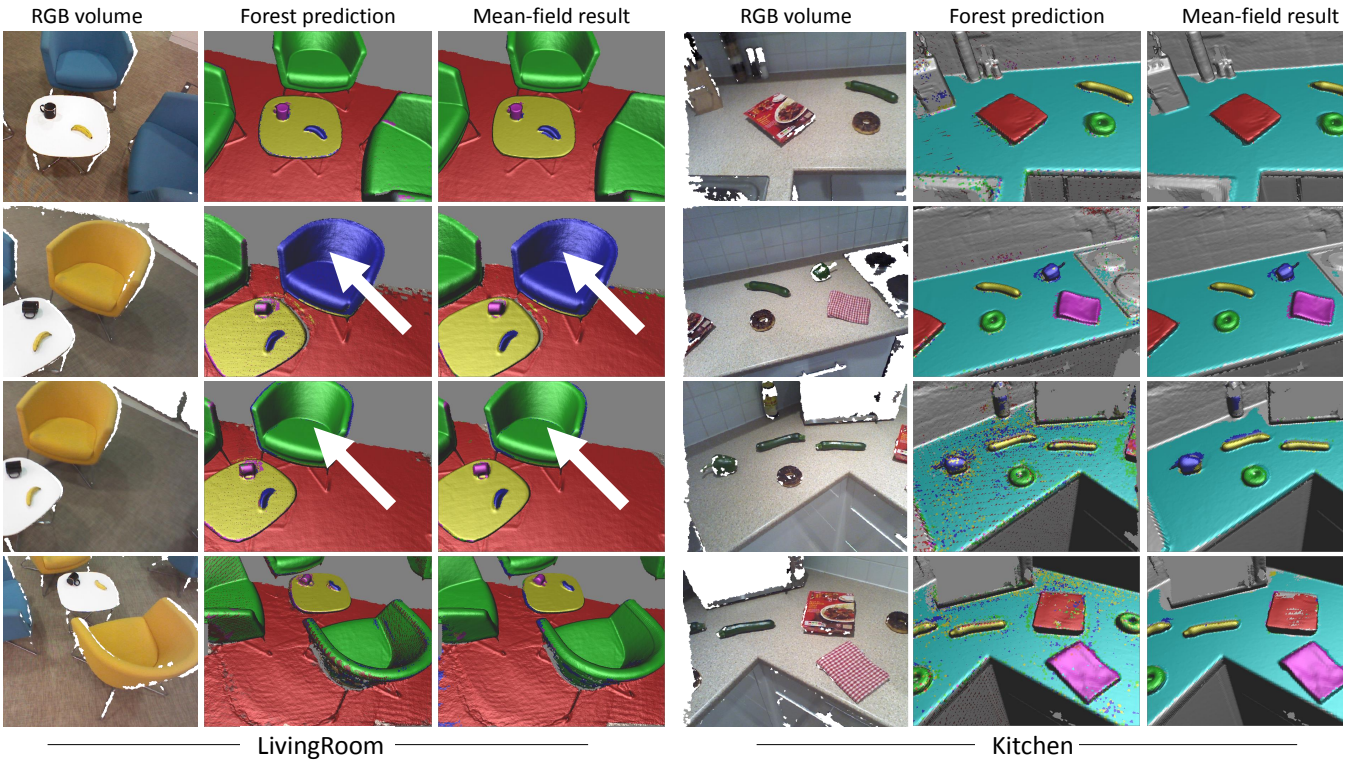

Fig. 7. Forest predictions and final mean-field inference results for two scenes. Our streaming random forest is able to learn to make per-voxel predictions about the object classes present in the scene. Each pixel is classified independently, and so the forest predictions can be somewhat noisy. The mean-field inference effectively smooths these predictions to produce a final labeling output to display to the user. The arrows indicate a region that is initially incorrectly labeled (2nd row), but is successfully corrected (3rd row) by updating the forest based on new user interactions.

Discriminative features. The training data given to the streaming random forest is a transformation of the RGB-D values around each training point. That transformation is usually referred to as a feature. The discriminative power of the feature used to describe these voxels directly impacts the quality of the object segmentation. We compare the proposed VOP feature against fast and established features in the 2D object segmentation literature as well as one widely used 3D feature. The features we compare against are SURF [Bay et al. 2008] (OpenCV implementation), depth probes [Shotton et al. 2011], difference of mean color of two randomly sampled boxes [Shotton et al. 2006], color probe (similar to the depth probe, but in the RGB image) and SPIN images [Johnson 1997] (PCL implementation). Fig. 8 illustrates the qualitative classification results obtained by the aforementioned features. It has to be noted that we also tried to compare against PCL's implementation of PPF [Drost et al. 2010]. For each scene, we extracted the model of each object (mug, duvet cover, etc.). The approach failed to correctly detect most objects, even in the point clouds from which models were extracted from. Consequently, we did not include that method in the comparisons.

7.2 Computational Efficiency.

The inherently volumetric nature of our approach parallelizes well on modern GPU architectures. For our experiments we employed an Nvidia Titan with 6GB of RAM, although the system works on lower-end setups for smaller scenes. We provide approximate system timings in Table I. Although the timings change as a function of the number of visible voxels and resolution, in all tests we performed we observed interactive frame rates. Numbers are provided for 6mm voxel resolution. Note that the forest *learning* runs asynchronously in a background thread. This thread continuously samples new labeled training data from the current view frustum and updates itself. This ensures an up-to-date forest is available for classification whenever the user requests it.

Table I. Approximate system timing. Despite small fluctuations we observed consistently good, interactive frame rates.

Component	Reconst.	Update forest	Sampling	Mean-field	Forest Eval.
Timing	20ms	30ms	140ms	2-10ms	5ms

Reconst. = reconstruction, Eval. = evaluation.

7.3 Quantitative results

System Components. We first evaluate the accuracy of our main system components, namely the user segmentation, streaming random forest and the mean-field filtering of the forest predictions, based on the sequences introduced earlier. For each sequence, a series of RGB-D keyframes were hand-labeled with object segmentations. Keyframes were selected to ensure full coverage of the scene. Examples of these ground-truth images are shown in Fig. 9. These ground-truth images are then projected and aggregated onto the underlying TSDF, and then back-projected to all the views of each sequence. This generates a total of 4176 frames for LIVINGROOM, 12346 frames for KITCHEN, 7583 frames for BEDROOM, and 8916

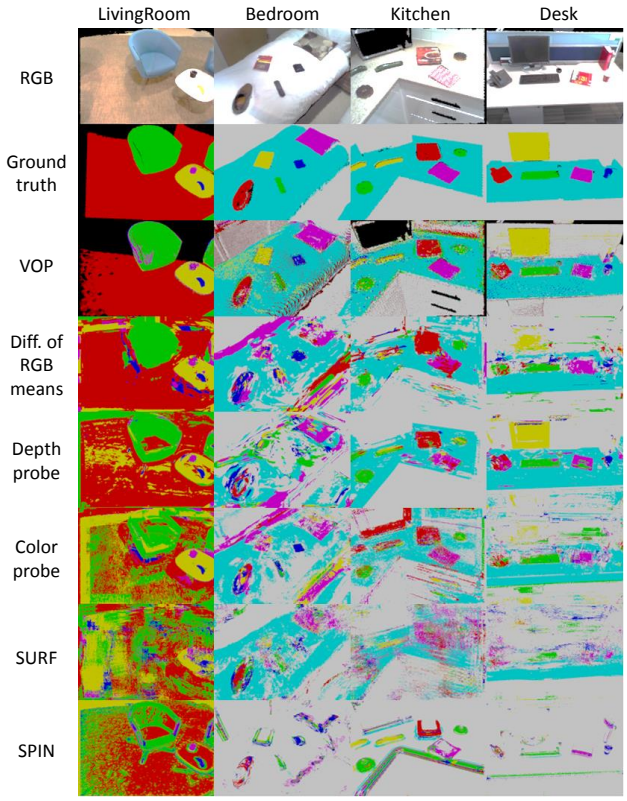

Fig. 8. Quantitative results for the proposed VOP and baseline features on different scenes. One can observe that the proposed VOP feature leads to qualitatively better results than all the baseline approaches.

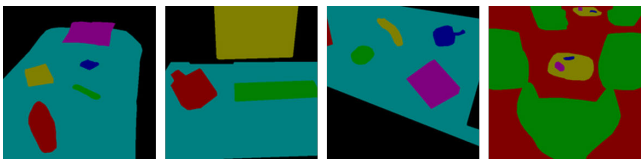

Fig. 9. Example ground truth labels. From left to right: BEDROOM, DESK, KITCHEN and LIVINGROOM.

Table II. Evaluation of the different components of the system on different scenes. The measure corresponds to the percentage of correctly classified pixels.

Component	LivingRoom	Bedroom	Kitchen	Desk	Average
User Interaction	99.35%	97.61%	96.09%	97.73%	97.7%
Forest prediciton	94.57%	88.31%	82.58%	90.29%	88.94%
Final Inference	96.26%	95.19%	90.69%	95.55%	94.42%

frames for DESK, of which roughly a third is used for test, and the rest for training.

Table II shows the percentage of correctly classified pixels for each component. It can be observed that the accuracy of all components is high and that the mean-field filtering provides for a good improvement on top of the predictions made by the forest.

Voxel-Oriented Patch Features. Using the same dataset, we evaluate the discriminative power of the proposed VOP feature against the baseline features listed in Sec. 7.1. For all these baseline features, we set the neighborhood from which data are sampled to be similar to the neighborhood from which VOP sample data from. Tab. III shows these comparisons. One can observe that the proposed VOP feature provides for a substantial margin of discriminative power compared to the different baseline methods.

Table III. Evaluation of the proposed feature descriptor against the baseline descriptors. The measure used is the percentage of correctly classified pixels. Note that overall VOP is also much better than the baseline methods under the class average precision and intersection over union measures.

Feature	LivingRoom	Bedroom	Kitchen	Desk	Average
VOP	**94.57%**	**88.31%**	82.58%	**90.29%**	**88.94%**
Diff. of RGB means	80%	71.84%	76.29%	73.42%	75.39%
Depth probe	77.54%	61.79%	**84.9%**	68.9%	73.06%
Color probe	56.39%	65.68%	60.77%	60.74%	60.9%
SURF	43.74%	67.12%	57%	58.13%	56.5%
SPIN	58.77%	43.22%	48.41%	36.1%	46.63%

Streaming Random Forests. To evaluate the contribution of our new streaming random forest (SRF) learning algorithm, we compare it against two well known online decision forest algorithms: online random forest (ORF) [Saffari et al. 2009] and Hoeffding trees (HT) [Domingos and Hulten 2000]. As an object can be trained in a sequential fashion, our system inherently has to deal with non-IID data. In order to perform quantitative evaluations, we use the dataset of [Lai et al. 2011] which contains 300 objects organized into 51 categories. Each of these objects is spun around a turntable at constant speed. One revolution of each object is recorded from 3 different points of view using a RGB-D camera. To generate an online and non-IID setting, each category is added sequentially. For each object and each viewpoint, a consecutive segment of one third of the images of each object is kept for test purposes and the rest is used for training. All the learning methods have been evaluated over an increasing number of object classes, where the classes present and their order vary for each configuration. Fig. 10 show the corresponding results which demonstrate that the proposed learning algorithm outperforms the baseline methods regardless the number of classes used. Note that for these experiments, we use the difference of mean color of two randomly sampled boxes as a feature [Shotton et al. 2006].

8. DISCUSSION

We have demonstrated a practical system which allows a user to interactively segment and label the surrounding environment in real-time. The labeling happens both explicitly through user interaction with the physical objects, and implicitly through the decision forest's ability to infer class labels in unlabeled parts of the scene.

Applications. We foresee numerous potential practical applications of our system.

Our system can be used to quickly gathering large numbers of labeled 3D environments for training large-scale visual recognition systems such as [Krizhevsky et al. 2012; Silberman and Fergus 2011]. Large-scale offline machine learning systems are currently attempting to solve the more general object recognition problem in both 2D [Krizhevsky et al. 2012] and 3D [Silberman and Fergus 2011]. But this requires vast amounts of labeled training data. Despite progress in crowd-sourcing, data collection and labeling, especially at the pixel/voxel level, remains difficult and expensive, especially for 3D environments. We envisage that our system could be deployed to rapidly capture large numbers of labeled 3D envi-

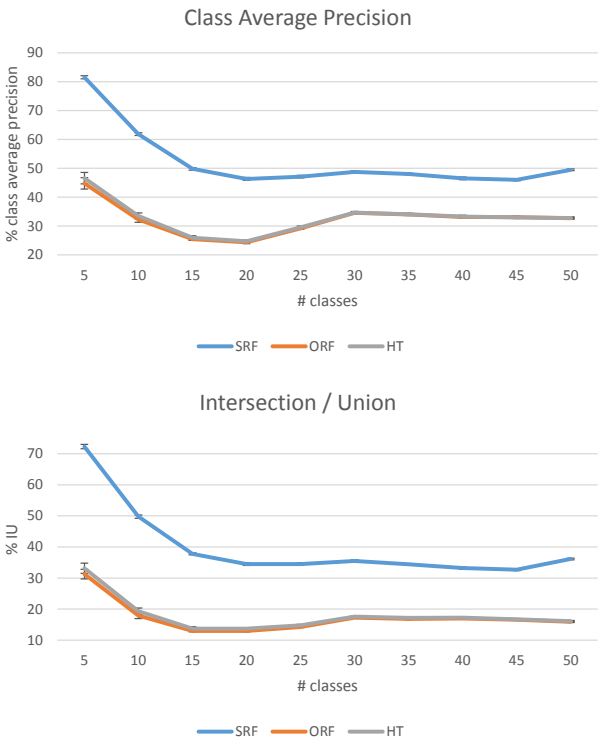

Fig. 10. Comparison of the proposed streaming random forest (SRF) algorithm against online random forest (ORF) and Hoeffding trees (HT) on the dataset described in 7.3. For each configuration of the classes, each learner contains 3 trees and the results have been averaged over 10 folds. These figures demonstrate that the proposed method comfortably outperforms all the baselines.

ronments and thus hasten the availability of more general-purpose recognition systems.

Personalized environment maps with known object class segmentations could be used for way-finding and navigation, either for robots or end users (e.g. the partially sighted). For example, you could ask your robot to find and bring you objects by semantically breaking down the world (e.g. 'bring me the cup by the chair in the living room'). Furthermore, if such a model was maintained and updated over time, finding lost objects could be as simply as uttering a few words (e.g. 'where have my gloves gone?!'). Augmented reality gaming could be a rich source of application. Imagine quickly scanning in and labeling your living room, and then associating object classes with aspects of the game. For example, game characters to sit on chairs or pick up and drink from a cup. Such augmented reality scenarios could be expanded for planning the renovation of a building, automating inventory, and designing interiors [Merrell et al. 2011].

Limitations. Despite very encouraging results, which we hope will inspire future work, our system clearly has limitations. So far we have only demonstrated simple real-world scenes, and it is clear that our system would struggle with more complex scenes. As shown in Fig. 11 there are currently many failure cases.

One of the main issues comes in our use of appearance data. While the combination of appearance and geometry adds a great deal of discriminative power, the use of RGB data comes at a cost.

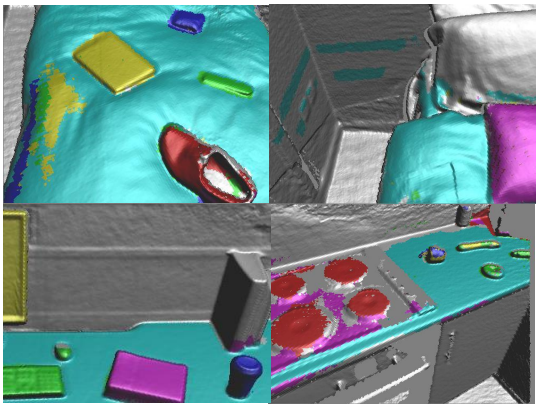

Fig. 11. Example failure cases, including: incorrect labels for segmented objects e.g. mouse and keyboard (bottom left), as well as misclassification of background voxels (top left, top right, and bottom right). Also 'bleeding' of labels around the edges of objects.

Careful calibration of an RGB camera with respect to the depth sensor is required, as well as temporal synchronization between the two. Spatial or temporal misregistration between the sensors can cause misclassification (or 'bleeding') at object boundaries, as shown in Fig. 11. In addition, our learning is not robust to strong illumination changes across different viewpoints of an object. As shown in Fig. 11 (top left) this can result in misclassification of certain labels as the viewpoint of the camera changes. Using more samples at different time intervals could help during the learning phase. However, an interesting avenue of future work would be to focus purely on geometric features.

Furthermore, while our method segments textured objects (i.e. non-uniformly colored objects), especially with strong geometric features, it cannot currently scale to segmentation of more complex foreground objects or backgrounds. Our learning is also very much local, and using global context, such as the relationship of objects to the ground plane could provide strong priors for classification.

Another challenge in the learning is to robustly support a 'background' class. Currently when we add a new training example, we sample voxels associated with the segmented object, as well as other 'background' voxels (that are unlabeled). This can lead to issues in the classification if a particular object is first labeled as background and then annotated by the user. Additionally, learning a background class for varied scenes is also challenging, as any new unobserved geometry and/or appearance will confuse the classification. As shown in Fig. 11 (bottom right) the oven has not be learned as part of the background, and is therefore misclassified. Clearly here the interactive nature of our system, could allow the user to quickly label the oven or re-learn to ensure it is added to the background class.

In terms of interaction, our system currently uses a voice command to switch between annotation, training and test modes. We plan an extension where both the learning and forest predictions are always turned on. This will require considerable care to avoid 'drift' in the learned category models: the feedback loop would mean that small errors could quickly become amplified. We also believe additional modes of interaction such as voice priors (e.g. 'walls are vertical'), as well as more intelligent sampling of training examples would improve results. Finally, algorithmic parameters such as pairwise weights are currently set at compile time (these are cross-validated and common across all datasets shown). Given

a small training set (perhaps with boot-strapping), more reliable settings could be automatically selected online.

9. CONCLUSIONS

We have presented a system that allows a user to interactively segment and label an environment quickly and easily. A real-time algorithm reconstructs a 3D model of the surrounding scene as the user captures it. The user can interact with the world by painting on surfaces and using voice commands to provide object class labels. A GPU-enabled mean-field inference algorithm then propagates the user's strokes through a volumetric random field model representing the scene. This results in a spatially smooth segmentation that respects object boundaries. In the background, the propagated labels are used to build a classifier, using our new streaming decision forests training algorithm. Once trained, the forest can predict a distribution over object labels for previously unseen voxels. These predictions are finally incorporated back into the 3D random field, and mean-field inference provides the final 3D semantic segmentation to the user. Our method generates dense 3D models that are broken down into semantic parts in just minutes.

Our system hopefully brings us closer to a future where people can create semantic models of their environments in lightweight ways, which can then be used in a variety of interactive applications, from robot guidance, to aiding partially sighted people, to helping us find objects and navigate our worlds, or experience new types of augmented realities. It thus helps computers not only reason about space around them, but gives the geometry they observe semantic meaning. Our system also hopefully moves us closer to the vision of *life-long learning*: where semantic models adapt and extend to new object classes online, as users continuously interact with the world. With the increased power of mobile devices, coupled with ultra-mobile depth cameras, we hope such online semantic modeling tools will become more commonplace in our future lives.

REFERENCES

ABDELRAHMAN, M., AONO, M., EL-MELEGY, M., FARAG, A., FERREIRA, A., JOHAN, H., LI, B., LU, Y., MACHADO, J., PASCOAL, P. B., AND TATSUMA, A. 2013. SHREC13: Retrieval of objects captured with low-cost depth-sensing cameras. In *Proc. Eurographics Workshop*.

ANAND, A., KOPPULA, H. S., JOACHIMS, T., AND SAXENA, A. 2013. Contextually guided semantic labeling and search for three-dimensional point clouds. *The International Journal of Robotics Research 32,* 1, 19–34.

BAY, H., ESS, A., TUYTELAARS, T., AND GOOL, L. V. 2008. Surf: Speeded up robust features", computer vision and image understanding. In *Proc. CVIU*.

BIFET, A., HOLMES, G., PFAHRINGER, B., KIRKBY, R., AND GAVALDÀ, R. 2009. New ensemble methods for evolving data streams. In *Proc. SIGKDD*.

BONDE, U., BADRINARAYANAN, V., AND CIPOLLA, R. 2013. Multi scale shape index for 3D object recognition. In *Scale Space and Variational Methods in Computer Vision*. Springer, 306–318.

BOYKOV, Y., VEKSLER, O., AND ZABIH, R. 2001. Fast approximate energy minimization via graph cuts. *IEEE Trans. PAMI 23,* 11.

BREIMAN, L. 2001. Random forests. *Machine Learning 45,* 1.

BROSTOW, G. J., SHOTTON, J., FAUQUEUR, J., AND CIPOLLA, R. 2008. Segmentation and recognition using structure from motion point clouds. In *Proc. ECCV*.

CASTLE, R. O., GAWLEY, D., KLEIN, G., AND MURRAY, D. W. 2007. Towards simultaneous recognition, localization and mapping for hand-held and wearable cameras. In *Proc. ICRA*.

CHEN, J., BAUTEMBACH, D., AND IZADI, S. 2013. Scalable real-time volumetric surface reconstruction. *ACM TOG 32,* 4, 113.

CHEN, X., GOLOVINSKIY, A., AND FUNKHOUSER, T. 2009. A benchmark for 3D mesh segmentation. *ACM TOG 28,* 3, 73.

CHENG, M.-M., ZHENG, S., LIN, W.-Y., VINEET, V., STURGESS, P., CROOK, N., MITRA, N., AND TORR, P. 2014. ImageSpirit: Verbal guided image parsing. *ACM TOG*.

COUPRIE, C., FARABET, C., NAJMAN, L., AND LECUN, Y. 2013. Indoor semantic segmentation using depth information. *arXiv:1301.3572*.

CRIMINISI, A. AND SHOTTON, J. 2013. Decision forests for computer vision and medical image analysis. *Springer*.

CURLESS, B. AND LEVOY, M. 1996. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 303–312.

DALAL, N. AND TRIGGS, B. 2005. Histograms of oriented gradients for human detection. In *Proc. CVPR*.

DOMINGOS, P. AND HULTEN, G. 2000. Mining high-speed data streams. In *Proc. SIGKDD*.

DROST, B., ULRICH, M., NAVAB, N., AND ILIC, S. 2010. Model globally, match locally: Efcient and robust 3d object recognition. In *Proc. CVPR*.

FIORAIO, N. AND DI STEFANO, L. 2013. Joint detection, tracking and mapping by semantic bundle adjustment. In *Proc. CVPR*.

GEIGER, A., LENZ, P., AND URTASUN, R. 2012. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *Proc. CVPR*.

GUPTA, A., EFROS, A. A., AND HEBERT, M. 2010. Blocks world revisited: Image understanding using qualitative geometry and mechanics. In *ECCV*.

HÄNE, C., ZACH, C., COHEN, A., ANGST, R., AND POLLEFEYS, M. 2013. Joint 3D scene reconstruction and class segmentation. In *Proc. CVPR*.

HERBST, E., HENRY, P., AND FOX, D. 2014. Toward online 3-d object segmentation and mapping. In *IEEE International Conference on Robotics and Automation (ICRA)*.

HIRSCHMULLER, H. 2008. Stereo processing by semiglobal matching and mutual information. *Trans. PAMI 30,* 2, 328–341.

IOANOU, Y., TAATI, B., HARRAP, R., AND GREENSPAN, M. 2012. Difference of normals as a multi-scale operator in unorganized point clouds. In *Proc. 3DIMPVT*.

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. UIST*. 559–568.

JOHNSON, A. 1997. Spin-images: A representation for 3-d surface matching. Ph.D. thesis, Robotics Institute, Carnegie Mellon University.

KÄHLER, O. AND REID, I. 2013. Efficient 3D scene labeling using fields of trees. In *Proc. ICCV*.

KALOGERAKIS, E., HERTZMANN, A., AND SINGH, K. 2010. Learning 3D mesh segmentation and labeling. *ACM TOG 29,* 4, 102.

KARPATHY, A., MILLER, S., AND FEI-FEI, L. 2013. Object discovery in 3D scenes via shape analysis. In *Proc. ICRA*.

KIM, B.-S., KOHLI, P., AND SAVARESE, S. 2013. 3D scene understanding by voxel-CRF. In *Proc. ICCV*.

KIM, V. G., LI, W., MITRA, N. J., CHAUDHURI, S., DIVERDI, S., AND FUNKHOUSER, T. 2013. Learning part-based templates from large collections of 3D shapes. *ACM TOG*.

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

KOHLI, P., LADICKY, L., AND TORR, P. H. S. 2009. Robust higher order potentials for enforcing label consistency. *IJCV*.

KOLLER, D. AND FRIEDMAN, N. 2009. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

KOPPULA, H. S., ANAND, A., JOACHIMS, T., AND SAXENA, A. 2011. Semantic labeling of 3D point clouds for indoor scenes. In *Proc. NIPS*.

KRÄHENBÜHL, P. AND KOLTUN, V. 2011. Efficient inference in fully connected CRFs with Gaussian edge potentials. In *NIPS*.

KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. 2012. ImageNet classification with deep convolutional neural networks. In *Proc. NIPS*.

LADICKÝ, L., STURGESS, P., RUSSELL, C., SENGUPTA, S., BASTANLAR, Y., CLOCKSIN, W., AND TORR, P. H. 2012. Joint optimization for object class segmentation and dense stereo reconstruction. *IJCV 100,* 2, 122–133.

LAFFERTY, J., MCCALLUM, A., AND PEREIRA, F. C. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

LAI, K., BO, L., REN, X., , AND FOX, D. 2011. A large-scale hierarchical multi-view rgb-d object dataset. In *Proc. ICRA*.

LEPETIT, V. AND FUA, P. 2006. Keypoint recognition using randomized trees. *IEEE Trans. PAMI*.

LEVOY, M., PULLI, K., CURLESS, B., RUSINKIEWICZ, S., KOLLER, D., PEREIRA, L., GINZTON, M., ANDERSON, S., DAVIS, J., GINSBERG, J., ET AL. 2000. The digital Michelangelo project: 3D scanning of large statues. In *Proc. SIGGRAPH*. ACM.

LIN, D., FIDLER, S., AND URTASUN, R. 2013. Holistic scene understanding for 3D object detection with RGBD cameras. In *Proc. ICCV*.

LIN, H., GAO, J., ZHOU, Y., LU, G., YE, M., ZHANG, C., LIU, L., AND YANG, R. 2013. Semantic decomposition and reconstruction of residential scenes from LiDAR data. *ACM TOG 32,* 4.

LOWE, D. G. 1999. Object recognition from local scale-invariant features. In *Proc. ICCV*.

MERRELL, P., SCHKUFZA, E., LI, Z., AGRAWALA, M., AND KOLTUN, V. 2011. Interactive furniture layout using interior design guidelines. *ACM TOG*.

NAN, L., XIE, K., AND SHARF, A. 2012. A search-classify approach for cluttered indoor scene understanding. *ACM TOG 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. 2011. KinectFusion: Real-time dense surface mapping and tracking. In *Proc. ISMAR*.

NEWCOMBE, R. A., LOVEGROVE, S. J., AND DAVISON, A. J. 2011. DTAM: Dense tracking and mapping in real-time. In *Proc. ICCV*.

NIESSNER, M., ZOLLHÖFER, M., IZADI, S., AND STAMMINGER, M. 2013. Real-time 3D reconstruction at scale using voxel hashing. *ACM TOG 32,* 6.

POLLEFEYS, M., NISTÉR, D., FRAHM, J., AKBARZADEH, A., MORDOHAI, P., CLIPP, B., ENGELS, C., GALLUP, D., KIM, S., MERRELL, P., ET AL. 2008. Detailed real-time urban 3D reconstruction from video. *IJCV 78,* 2.

POSNER, I., CUMMINS, M., AND NEWMAN, P. 2009. A generative framework for fast urban labeling using spatial and temporal context. *Autonomous Robots 26,* 2-3, 153–170.

PRADEEP, V., RHEMANN, C., IZADI, S., ZACH, C., BLEYER, M., AND BATHICHE, S. 2013. Monofusion: Real-time 3D reconstruction of small scenes with a single web camera. In *Proc. ISMAR*.

RAMOS, F., NIETO, J., AND DURRANT-WHYTE, H. 2008. Combining object recognition and SLAM for extended map representations. In *Experimental Robotics*. Springer, 55–64.

REN, X., BO, L., AND FOX, D. 2012. RGB-(D) scene labeling: Features and algorithms. In *Proc. CVPR*.

ROBERTS, L. G. 1963. Machine perception of three-dimensional solids. Ph.D. thesis, Massachusetts Institute of Technology.

ROTHER, C., KOLMOGOROV, V., AND BLAKE, A. 2004a. GrabCut - interactive foreground extraction using iterated graph cuts. *ACM TOG 23,* 3.

ROTHER, C., KOLMOGOROV, V., AND BLAKE, A. 2004b. Grabcut: Interactive foreground extraction using iterated graph cuts. In *ACM Transactions on Graphics (TOG)*. Vol. 23. ACM, 309–314.

RUSINKIEWICZ, S., HALL-HOLT, O., AND LEVOY, M. 2002. Real-time 3D model acquisition. *ACM TOG 21,* 3, 438–446.

RUSSELL, B. C., TORRALBA, A., MURPHY, K. P., AND FREEMAN, W. T. 2008. Labelme: a database and web-based tool for image annotation. *International journal of computer vision 77,* 1-3, 157–173.

SAFFARI, A., LEISTNER, C., SANTNER, J., GODEC, M., AND BISCHOF, H. 2009. On-line random forests. In *IEEE ICCV Workshop*.

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

SENGUPTA, S., GREVESON, E., SHAHROKNI, A., AND TORR, P. H. 2013. Urban 3D semantic modelling using stereo vision. In *Proc. ICRA*.

SHAN, Q., ADAMS, R., CURLESS, B., FURUKAWA, Y., AND SEITZ, S. M. 2013. The visual turing test for scene reconstruction. In *Proc. 3DTV*.

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 TOG 31,* 6, 136.

SHAPIRA, L., SHALOM, S., SHAMIR, A., COHEN-OR, D., AND ZHANG, H. 2010. Contextual part analogies in 3D objects. *IJCV 89,* 2-3, 309–326.

SHARP, T. 2008. Implementing decision trees and forests on a gpu. In *ECCV*. Springer, 595–608.

SHEN, C.-H., FU, H., CHEN, K., AND HU, S.-M. 2012. Structure recovery by part assembly. *ACM TOG 31,* 6, 180.

SHOTTON, J., FITZGIBBON, A., COOK, M., SHARP, T., FINOCCHIO, M., MOORE, R., KIPMAN, A., AND BLAKE, A. 2011. Real-time human pose recognition in parts from single depth images. In *Proc. CVPR*.

SHOTTON, J., WINN, J., ROTHER, C., AND CRIMINISI, A. 2006. TextonBoost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *Proc. ECCV*.

SILBERMAN, N. AND FERGUS, R. 2011. Indoor scene segmentation using a structured light sensor. In *Proc. ICCV Workshop*.

SILBERMAN, N., HOIEM, D., KOHLI, P., AND FERGUS, R. 2012. Indoor segmentation and support inference from RGBD images. In *Proc. ECCV*.

SNAVELY, N., SEITZ, S. M., AND SZELISKI, R. 2006. Photo tourism: exploring photo collections in 3D. *ACM TOG 25,* 3.

STÜCKLER, J., WALDVOGEL, B., SCHULZ, H., AND BEHNKE, S. 2013. Dense real-time mapping of object-class semantics from RGB-D video. *Journal of Real-Time Image Processing*, 1–11.

VALENTIN, J. P., SENGUPTA, S., WARRELL, J., SHAHROKNI, A., AND TORR, P. H. 2013. Mesh based semantic modelling for indoor and outdoor scenes. In *Proc. CVPR*.

VINEET, V. AND NARAYANAN, P. 2008. Cuda cuts: Fast graph cuts on the gpu. In *CVPR Workshops*. IEEE, 1–8.

VITTER, J. S. 1985. Random sampling with a reservoir. *ACM TOMS 11,* 1.

WANG, Y., FENG, J., WU, Z., WANG, J., AND CHANG, S.-F. 2014. From low-cost depth sensors to cad: Cross-domain 3d shape retrieval via regression tree fields. In *European Conference on Computer Vision (ECCV)*.

XIAO, J. 2014. A 2D + 3D rich data approach to scene understanding. Ph.D. thesis, Massachusetts Institute of Technology.

XIAO, J., HAYS, J., EHINGER, K. A., OLIVA, A., AND TORRALBA, A. 2010. SUN database: Large-scale scene recognition from abbey to zoo. In *Proc. CVPR*.

XIAO, J., OWENS, A., AND TORRALBA, A. 2013. SUN3D: A database of big spaces reconstructed using sfm and object labels. In *Proc. ICCV*.

YAO, A., GALL, J., LEISTNER, C., AND VAN GOOL, L. 2012. Interactive object detection. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 3242–3249.

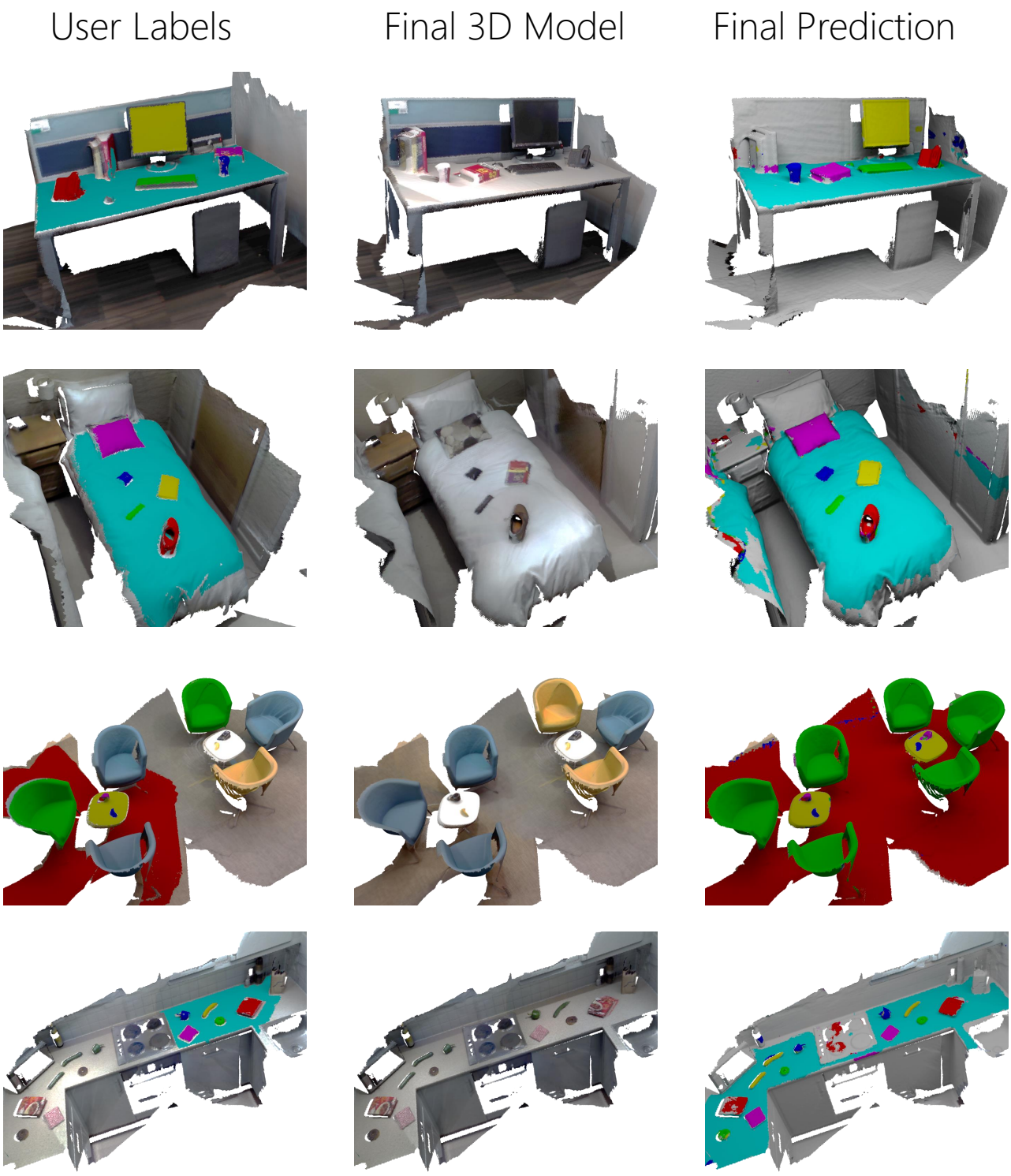

Fig. 12. Final results from our four datasets. From top to bottom: DESK, BEDROOM, LIVINGROOM and KITCHEN. Left column shows user specified labels. Middle column shows final textured 3D model. Right column shows final prediction after classification and inference.