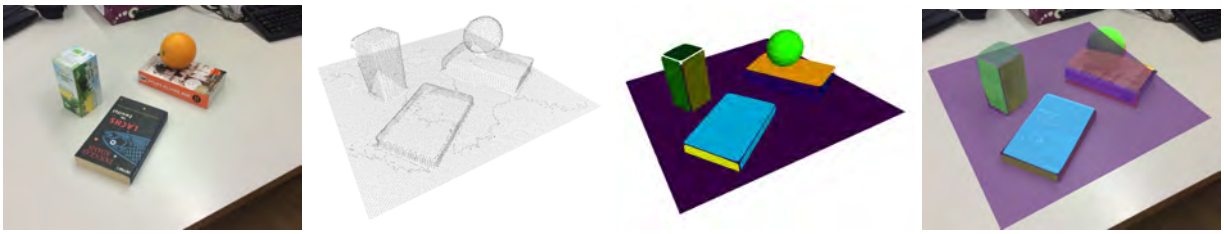
# Semantic Segmentation of Geometric Primitives in Dense 3D Point Clouds

Ana Stanescu* Graz University of Technology
Philipp Fleck† Graz University of Technology
Dieter Schmalstieg‡ Graz University of Technology
Clemens Arth§ AR4 GmbH



(a) (b) (c) (d)

Figure 1: Workflow from image and point cloud data to semantically labeled primitives used in a real-world application - (a) Input RGB image from camera, (b) Dense 3D point cloud as captured by structure sensor, (c) Detection of semantic primitives using RANSAC and SVM, (d) Overlay of primitives in a live-view augmented reality application.

## Abstract

This paper presents an approach to semantic segmentation and structural modeling from dense 3D point clouds. The core contribution is an efficient method for fitting of geometric primitives based on machine learning. First, the dense 3D point cloud is acquired together with RGB images on a mobile handheld device. Then, RANSAC is used to estimate the presence of geometric primitives, followed by an evaluation of their fit based on classification of the fitting parameters. Finally, the approach iterates over successive frames to optimize the fitting parameters or replace a detected primitive by a better fitting one. As a result, we obtain a semantic model of the scene consisting of a set of geometric primitives. We evaluate the approach on an extensive set of scenarios and show its plausibility in augmented reality applications.

**Index Terms:** Computing methodologies—Computer Vision—Computer vision tasks—Scene Understanding; Computing methodologies—Mixed / Augmented Reality; Computing methodologies—Machine learning approaches—Kernel methods—Support vector machines;

## 1 Introduction

Bringing semantic meaning to the world around us using computer vision (CV) and machine learning (ML) is increasingly relevant for practical applications, since mobile devices with depth sensors are becoming widely available. The availability of high-quality dense 3D point clouds with registered RGB images allows for entirely new levels of understanding our environment and opens new ways to interact with it, especially for augmented reality (AR) applications.

Modeling real environments with polygonal meshes and other geometric primitives is often referred to as *structural modeling*. 3D depth sensors acquire the approximate shape of the underlying scene in the form of a dense 3D point cloud. These point clouds have to be further processed to generate meshed surfaces. Despite a high computational effort, a meshed surface usually has no semantic meaning. However, our everyday environment largely consists of very few geometric primitives, mostly planes, boxes, cylinders and spheres. This prior knowledge can support mesh generation in a significant way. If a point cloud can be transformed into a set of geometrically meaningful entities, AR interactions, like placing virtual objects or automatic highlighting of object in the environment become feasible.

Semantic modeling from real-world assemblies is supported in some commercial modeling products, but usually requires substantial manual interaction. For example Curvsurf[1] requires manual labeling of 3D point clouds. Based on the fit of a primitive, parts of the geometric primitive are meshed, and the points are replaced. Although this approach gives reasonable results, choosing geometric primitives, scaling them and finally aligning them in 3D is tedious. Clearly, the ability to automatically determine geometric primitives in dense 3D point clouds has high potential for scientific and commercial exploitation.

In the proposed method, we acquire dense 3D point clouds and RGB images with a handheld mobile device. Using the points and their approximate surface normal, we use random sample consensus (RANSAC) to check for the existence of particular geometric primitives, and we estimate their parameters. These parameters and other key attributes of the fitted primitive are evaluated using a set of binary support vector machines (SVM), to determine the quality of the fit. Upon positive evaluation, the corresponding parts of the point cloud are marked as assigned, and the primitive label can be forwarded to the next frame. However, a negative evaluation by the SVM can lead to replacing an ill-fitting primitive with a different one. In this paper, we consider *planes*, *spheres* and *cylinders*, which we found to be integral parts of everyday environments.

## 2 Related work

In recent years, structural modeling by fitting geometric primitives has been investigated in various contexts. Examples include robot grasping, collision detection [2, 14, 22], architectural modeling [10–12, 25], 3D reconstruction [17], and scene denoising and compression [17]. Extensions to AR [16, 23] place a stronger emphasis on real-time aspects and online acquisition of input data. Most methods build on RANSAC [1] or Hough transforms [3] for robust primitive fitting.

Early works on fitting geometric primitives like planes, spheres, cylinders, cones and tori include [7, 14, 17], to some extent also investigating their contextual relations. All these approaches are not designed for real-time performance and are applied on a carefully

*e-mail: ana.stanescu@icg.tugraz.at
†e-mail: philipp.fleck@icg.tugraz.at
‡e-mail: schmalstieg@icg.tugraz.at
§e-mail: clemens@ar4.io

[1] Curvsurf: http://www.curvsurf.com/

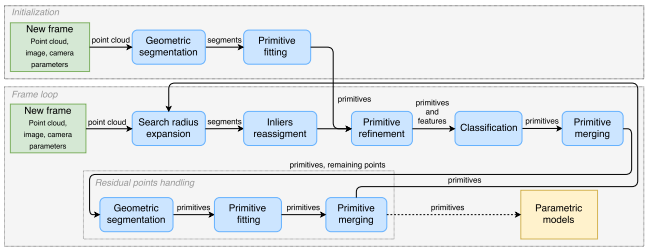



Figure 2: Overview of the proposed algorithm. Our approach is subdivided into an *initialization* and a *frame loop* part, which handle the startup and runtime phase properly. More information about the individual parts are given in Sec. 3 through Sec. 5, respectively.

acquired 3D point clouds. In contrast, approaches using real-time scene segmentation [2, 22] are used in mobile robotics.

With the increasing popularity of deep learning, recent approaches for semantic segmentation of 3D data use convolutional deep belief networks [24], convolutional neural networks (CNN) [8] and CNN together with conditional random fields (CRF) [20]. These methods use concrete object classes and require a large amount of data for training. Despite recent advances, a generalized formulation for objects consisting of geometric primitives is still missing.

Recent progress in simultaeneous localization and mapping (SLAM) provides a real-time alternative to offline acquisition of 3D point models. Point clouds from SLAM are popular for investigating semantic segmentation in real-time, for example, for planes [9] or planes, spheres and cylinders [13].

In this work, we apply an approach based on dense and noisy point clouds. We use a segmentation-fitting-refinement pipeline [13, 14], but applied over multiple frames to refine detected primitives. We make use of ML to decide whether a shape has to be discarded or kept at runtime. ML techniques such as SVM [4, 10] and probabilistic graphical models [6, 26] have been used for structural modeling of geometric primitives in the past. However, in our case, SVM classification is used to discard outdated models. In contrast to previous work [14], our method is designed to improve the structural modeling over time. Our approach therefore closes a gap in research, and we will show that it is particularly useful in an AR context.

## 3 Overview

In the following, we collectively refer to the 3D point cloud, the related RGB image and the camera pose as a *frame*. We use the terms *cluster* or *segment* for a set of 3D points, and we assume a 3D representation for all objects, unless otherwise noted. The point clouds used in this work span a volume of approximately one cubic meter. The working volume is a restriction of our acquisition setup, and not an intrinsic restriction of the employed algorithms.

The goal of our method is to achieve a segmentation of a 3D scene into geometric primitives from a *stream* of frames. The models need to be improved over time and adapt to the newly discovered geometry, as the acquisition device moves through the scene. This scenario is relevant when using an AR system relying on SLAM, where new parts of the scene are discovered opportunistically, and the user expects quick system response.

While we do not rely on a consistent relationship between individual 3D points, we assume the point clouds are dense and already registered in a global coordinate system. The scenarios are static, due to the nature of the sensors, and using SLAM implies that a significant amount of noise is present. We assume indoor scenes consisting of objects that can be approximated with the geometric primitives supported in our system, but otherwise make no further assumptions about the structure of the scene.

The proposed algorithm is shown in Fig. 2. In the *initialization* phase, we segment the 3D points in the first cloud based on the orientation of their normals. As a result, we obtain segments $C = \{c_1, ..., c_n\}$ which are coherent w.r.t. their normal orientation. We subsequently apply a fitting phase, considering all types of 3D primitives for each segment, choosing only a single primitive which fits best. For this stage, we incorporate the number of inliers and the surface normal orientation, obtaining a set of primitives, $P = \{p_1, ..., p_n\}$. As this is the most computationally expensive part of our algorithm, a full point cloud segmentation and fitting is not repeated for the rest of the modeling process.

For all subsequent frames, the procedure is outlined in the *frame loop* in Fig. 2. When a new frame is processed, the labels of the primitives are propagated to the next frame, and points are assigned to the previously detected primitives. More details are described in Section 4.5.

The subsequent refinement of each primitive $p_k$ is performed by means of non-linear optimization (see Section 4.3). Afterwards, based on features describing the quality of the fit, SVM is used to decide whether $p_k$ has an invalid or a valid fit. In the former case, the primitive is discarded, and its inliers are released to the set of unassigned 3D points. At the same time, primitives are merged together based on proximity and on the similarity of the parameters (further described in Sec 4.4).

At this stage of the algorithm, there exists a subset of 3D points that do not belong to any primitive. These 3D points are again segmented, and primitives are fitted to obtain a smaller set of new primitives $P_{new}$. The new primitives are merged to the already detected ones, $P = P \cup P_{new}$; we refer to this step as *residual points handling* in Fig 2 (more details to be found in Section 4.6.)

## 4 Primitive Fitting based on RANSAC

The robustness of RANSAC suggests a naive approach to structural modeling, which iteratively fits shapes to the point cloud, removes the inliers and then repeats the procedure. In scenarios with high outlier rates and many different objects present, the computational expenses of running a naive version of RANSAC increase significantly. RANSAC derivatives for higher outlier rates exist, but their performance heavily depends on the domain, noise level, *etc*. We avoid this problem by applying a pre-segmentation step.

### 4.1 Scene segmentation

Initialization. Segmentation of the scene using normal orientation is performed based on criteria constraining the angles between the normals of a 3D point and its surrounding 3D points, thereby delimiting the segmentation onto objects and surfaces. Color has been successfully employed in 3D object segmentation methods [6]. However, as man-made objects in indoor scenes present high variations on color and illumination, we do not use color information.

The surface normals are calculated by locally fitting a plane to a 3D point's nine nearest neighbors, and taking this plane's orientation as the normal for the 3D point. Previous work [19] introduced two operators to create a segmentation of the 3D point cloud, denoted by $\phi$ and $\Gamma$.

The normal analysis operators are used in a graph-based connected component clustering phase. A threshold is applied to the values of the operators, using 0.96 for $\phi$ and 0.0013 for $\Gamma$. These thresholds have been empirically chosen for the expected noise levels in the used datasets. After identifying the points describing edges that separate segments, the edge-points are removed from the point cloud, which is further transformed into an undirected graph. The graph edges between 3D points represent their proximity. A connected component algorithm is run on the graph, obtaining object segments. All segments containing fewer points than 0.05% of the entire point cloud are not considered further, as they are likely just noise.

Remaining Points Handling At a later stage of our algorithm, we employ segmentation again, however, on the remaining points only. This time both convex and concave regions are penalized, using a different operator, which is also based on the orientation of the surface normals. This operator takes into account the smallest angle between neighboring normals [22].

If the angle between the normals is smaller than a threshold, a surface normal edge region has been found. The threshold is set to 0.8, which corresponds to around 36.9 degrees. We set a low threshold, so that noise does not introduce edges erroneously. The edges are removed, and the rest of the points are clustered. Clusters with fewer points than 0.04% of the whole point cloud are not considered further, in a similar way to the first segmentation method. The threshold is lower, because smaller clusters should also be considered in order to find smaller surfaces as well.

The two segmentation criteria have different purposes. While the first one more coarsely segments the point cloud into convex objects in the main part of the algorithm, the second one is aimed at a finer segmentation in the *residual point handling* step. This coarse-to-fine strategy increases chances that primitives that have been previously overlooked are finally identified.

### 4.2 Primitive fitting

A plane, a sphere and a cylinder are fit to each point cluster. For this purpose, we employ a variant of RANSAC, M-estimator SAmple and Consensus (MSAC) [21]. We use existing MATLAB functions for fitting primitives to point clouds. Such functionality of fitting different primitive types with RANSAC in a point cloud is also provided by the Point Cloud Library [15]. However, the methods provided by PCL only allow a one-time fit of a particular primitive type to a point cloud; they do not allow incremental fitting or adjustment over time.

Spheres or cylinders with a diameter larger than 80% of the span of the point cloud bounding box are discarded, as any large enough cylinder or sphere can resemble a plane at a high noise level. Another check is performed regarding the distribution of the points on the shape surface. The shape surface distribution of the inliers is compared to a discrete uniform distribution of points on the surface with the Hellinger distance. If this distance is below a threshold of 0.65, we assume the fit to be invalid. As the Hellinger distance takes values between zero and one, the latter being the maximal distance, this threshold only excludes extreme cases.

The final decision on the choice of the primitive is taken based on the number of inliers, for which the normals on the point cloud surface do not deviate from the normals on the primitive surface by more than 18 degrees. This threshold is empirically chosen to account for noise in the point cloud. This strategy includes the curvature of the point cloud surface into the decision [17].

### 4.3 Primitive refinement

After having estimated the primitive parameters, we proceed to a refinement step by performing non-linear optimization. The fit is improved based on the estimated model and its inliers. This optimization step accounts for non-linearity caused by distortions or noise introduced by the sensor. Using the Ceres[2] solver, we initialize the optimization with the already estimated primitive parameters, and then find the minimum iteratively, using the Levenberg-Marquard algorithm. The cost function is defined as the squared distance from the points to the 3D parametric model, modified by the Huber loss function. The loss function slightly dampens the influence of points with larger squared distances, making the fitting process more robust to outliers.

The refinement is applied to all detected primitives. Then, the inliers are reassigned based on their distance to the new model.

[2]Ceres-Solver: `http://ceres-solver.org/`

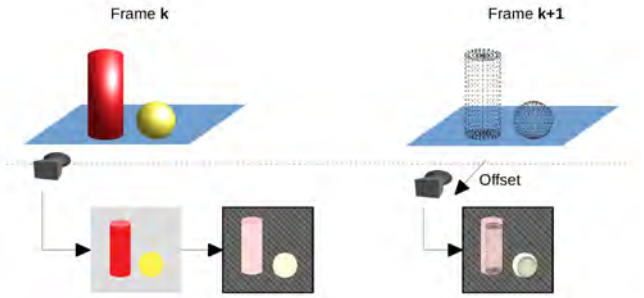



Figure 3: Label propagation and search radius expansion between two consecutive frames: The inliers of the primitives detected in frame $k$ are projected onto the camera plane. Then, a mask of the convex hulls of the inliers projections is created. The mask is used to define the inlier candidates for each primitive in frame $k+1$ by projecting the new points onto the camera plane from the pose of frame $k+1$, moved backwards by an offset for increasing the search region.

### 4.4 Primitive merging

Because of occlusions or noise, parts of the scene might be over-segmented, and certain primitives might be detected multiple times, raising support from different clusters of inliers. Those clusters need to be merged, and the merging strategy differs for the individual primitive types.

For planes, we define a *plane segment* as the convex hull of the inliers of a plane, projected onto the plane. In order to merge two plane segments, not only their orientations need to be similar, but the distance between them has to be small as well. We implement a minimal 3D point-to-polygon distance [18]. The spheres are merged based on the distance between their centers and their radii. In the case of cylinders, their axis orientations, their radii, and the distance between the line segments determined by their axes are compared.

### 4.5 Propagation and search radius expansion

We aim to propagate the labels of the detected primitives to the next frame, without needing to refit the primitives again. Unfortunately, the stream of point clouds usually does not include explicit correspondences between individual points. We propose projecting each primitive's class label onto the image plane [19] and using this projection as a cone-shaped region of interest for finding inlier candidates in the next frame. The projection area of the convex hull is also slightly increased by artificially moving the camera position backwards by a fixed offset, assuming there is little optical flow in-between the frames. The points inside the 3D search cone resulted from the projected convex hull are potential inliers. This procedure is sketched in Figure 3. The set of inliers is recalculated based on the distance from the candidates to the model. After this step, the set of primitives and inliers have been identified in a new frame.

### 4.6 Residual points handling

At each frame, there are points that remain unassigned to a model left from previous steps from the algorithm. This might either be due to their assignment to very small clusters, to their lack of fit to any primitive, or to the fact that corresponding primitives were discarded by the classifiers as invalid.

We repeat the segmentation-primitive fitting steps described in Sections 4.1 and 4.2 in these regions. Here, the segmentation method is different, not only concerning the edge criterion, but also because small clusters that were previously ignored are taken into consideration as well. The resulting primitives are merged with the already found primitives.

## 5 Parameter classification using SVM

During acquisition, objects are usually only partially visible to the sensor setup (*i.e.,* the setup only captures a part of the scene). In

certain cases, especially at the beginning of a scanning session, this leads to the detection of wrong primitives, as only little data is available. Therefore, we need a criterion to decide when to discard misfits at a later stage, such that a refitting step can be rerun in this particular region.

Deep learning would likely be able to handle such a challenging decision, but training requires vast amounts of largely noise-free training data. Such databases are difficult to obtain without excessive cost. We therefore decided in favor of SVM as a more traditional classification technique, which can be applied on a standard CPU hardware with very manageable training effort. We use an SVM set that, from five features describing the fit of a model, evaluates whether a primitive should be discarded.

Characterizing primitives Each primitive fit is described by five features, which are inferred from the points assigned to the primitive. A sample $s_k$ is obtained as a 5D vector
$s_k = \{\#inliers,\ rmse,\ \#inliers\_nde,\ mnde,\ hellinger\_dist\}$.

1. **#inliers**: the number of inliers of a primitive. Primitives with over 10.000 inliers are not classified at all, as they are considered reliable.
2. **rmse**: the root mean square error, *i.e.,* mean distance from the inliers to the fitted model, normalized by the maximum allowed distance.
3. **#inliers_nde**: the number of inliers coherent with the shape curvature. Checking whether the orientation of the normals of a primitives inliers agrees with the orientation of the normals on the surface of the primitive is a criterion for the goodness of the fit [14, 17]. We perform this check, counting how many inliers' normals deviate by $< 18°$ from the normals in the points, obtained by the projection of the inliers on the model surface.
4. **mnde**: the mean normal deviation error, *i.e.,* the mean of the cosines of the deviation angles between the surface normals at the position of the inliers and the primitive's normals.
5. **hellinger_dist**: the Hellinger distance as a uniformity measure of the point distribution on the shape surface. As large cylinders or spheres can approximate planar regions only locally, the distribution of the inliers on their surface is an indicator of their fit. We calculate this in terms of the Hellinger distance between the discrete 2D distribution of the inliers on the primitive surface and a discrete 2D uniform distribution on the primitive surface.

Training SVM sets We generate training data by artificially creating a copy of a scene with color-coded primitives and aligning it with the real-world point cloud. We lay out a scene on scale paper and recreate it in Blender. After alignment, we transfer the labels of individual primitives, which are color coded, to the points closely falling onto these primitives. Cylinder caps are labeled as planes, as the expected behavior of the algorithm is to identify cylinder caps as plane models. Examples are shown in Figure 4.

The samples are generated by running the algorithm with an update mechanism in a two-frame cycle. We perform a complete segmentation and fitting on each second frame, followed by a frame update and primitive refinement. Finally, we compute the features for each primitive. This procedure is equivalent to restarting the algorithm at each second frame. We employ this restarting scheme to ensure that is sufficient variation and little interdependence between the detected primitives.

For learning a particular primitive, we force-fit said primitive to the data, while ignoring other primitives during this step. The fitted model is compared to the type of the real underlying primitive by counting inliers whose color labels are consistent with the desired primitive type. If the two types are the same, then the label of the sample is set to one, otherwise it is set to zero. We repeat this step for all types of primitives, generating a large set of positive and negative samples.

From the labeled dataset, the SVM learns to classify correctly and incorrectly fitted primitives. The values of the features are scaled to the interval [0,1] with respect to the maximum value per feature.

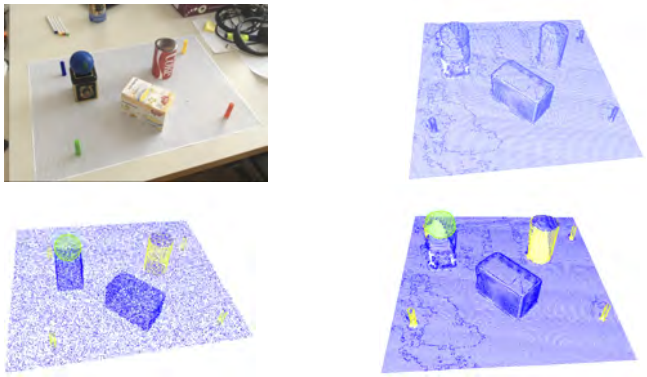


Figure 4: Top left: real world dataset image. Top right: unlabeled real world dataset point cloud. Bottom left: real world dataset point cloud after labels transfer. Bottom right: label transfer between artificial and real datasets.

| Classifier | Kernel | Feature set | # Training Samples | # Support Vectors | Accuracy |
|---|---|---|---|---|---|
| Plane | rbf | {1, 3, 4, 5} | 3008 | 507 | 79.85% |
| Sphere | poly. | {1, 2, 3, 4} | 1491 | 30 | 97.57% |
| Cylinder | rbf | {1, 2, 3, 4, 5} | 1582 | 258 | 89.3% |

Table 1: SVM classifiers properties and accuracy.

## 6 Experimental evaluation

In our evaluation, we first trained SVM sets and evaluated their performance, in order to assess the complexity of the problem and the suitability of certain feature combinations. Then we applied the learned SVM sets in our algorithm on a set of test scenarios.

### 6.1 Acquisition setup

Since our approach is aimed to work on portable devices, we test it on an Apple iPad equipped with a *structure.io*[3] sensor. This setup provides registered point clouds along with each frame. The sensor uses infrared structured light for retrieving the depth map, and the point back-projection and registration is done locally on the tablet. The datasets encompass between 100 and 300 frames, with point clouds of up to 30.000 points per frame.

### 6.2 SVM performance

For each primitive type, we perform five-fold cross-validation and train the binary SVM. We experimented with radial basis kernels, third-degree polynomial kernels and linear SVM, as well as with six configurations of features, selecting subsets of the five available features to find the best combination of features for a particular primitive type. The following configurations of features were evaluated:

$\{1,2,3,4,5\}$ $\{3,4,5\}$ $\{1,3,4,5\}$ $\{1,3,4\}$ $\{1,2,3,4\}$ $\{3,4\}$

The best-performing configurations identified for the individual primitives were chosen for our system. Details about the configurations and the classifiers are listed in Table 1. The best performing configurations lead to around 79% accuracy for planes, 97% for spheres and 89% for cylinders in the five-fold cross-validation. The

[3]Structure-IO Sensor: `https://structure.io/`

| Dataset | Primitive | Precision | Recall | Dataset | Primitive | Precision | Recall |
|---|---|---|---|---|---|---|---|
| Dataset #1 | Planes | 0.874 | 0.997 | Dataset #5 | Planes | 0.224 | 1.000 |
| | Spheres | – | – | | Spheres | 0.959 | 0.321 |
| | Cylinders | 0.750 | 0.747 | | Cylinders | 0.255 | 0.851 |
| | Total | 0.820 | 0.886 | | Total | 0.281 | 0.623 |
| Dataset #2 | Planes | 0.976 | 0.940 | Dataset #6 | Planes | 0.914 | 0.841 |
| | Spheres | 1 | 1 | | Spheres | 1 | 0.993 |
| | Cylinders | 0.720 | 0.714 | | Cylinders | 0.752 | 0.755 |
| | Total | 0.928 | 0.901 | | Total | 0.901 | 0.849 |
| Dataset #3 | Planes | 0.928 | 0.995 | Dataset #7 | Planes | 0.797 | 0.878 |
| | Spheres | – | – | | Spheres | – | – |
| | Cylinders | 0.743 | 0.741 | | Cylinders | 0.624 | 0.716 |
| | Total | 0.850 | 0.882 | | Total | 0.780 | 0.865 |
| Dataset #4 | Planes | 0.910 | 0.919 | Dataset #8 | Planes | 0.772 | 0.814 |
| | Spheres | 1 | 1 | | Spheres | 1 | 0.989 |
| | Cylinders | – | – | | Cylinders | 0.770 | 0.719 |
| | Total | 0.899 | 0.926 | | Total | 0.788 | 0.821 |
| **Total** | **Planes** | **0.796** | **0.921** | | | | |
| | **Spheres** | **0.994** | **0.859** | | | | |
| | **Cylinders** | **0.623** | **0.752** | | | | |

Table 2: Results of experiments in terms of average precision and recall for each dataset and overall.

number of support vectors give an indication about the complexity of the problem solved by the SVM. While the number of support vectors is low for spheres, it is considerably higher for cylinders.

For planes, almost 17% of the samples are required. We attribute this to a large variation in the samples used for training, and a yet incomplete set of suitable features to describe planes properly. Investigating other features to describe the quality of the fit of a primitive could improve the method.

### 6.3 Semantic modeling results

Using the trained SVM set, we tested our algorithm on a total of eight scenes with known object parameters. The performances obtained are listed in Table 2. Despite the decent performance of the SVM for cylinders in our five-fold cross-evaluation, cylinders show the poorest performance in practice. The reason is likely that cylinders can reasonably well approximate both spheres and parts of planes, leading to ambiguities in certain regions. Spheres reach a high precision, which is attributed to a sphere's specific curvature, which makes it harder to mistake a sphere for a cylinder and even harder to mistake for a plane. Some exemplary images from our algorithm on the datasets are shown in Figures 8.

In Figure 7, the average-filtered precision and recall achieved when running the algorithm on each individual dataset is depicted. The reference models are created by manually adjusting the primitives fit to the last frame of each dataset. One can see that both precision and recall are low in the beginning and steeply increase over the first few frames, as the point clouds get denser, and new geometry is revealed. Precision tends to stay at the same value or to slightly decrease. The algorithm tries to fit different primitives to newly discovered or freed areas in the point cloud, while some of these fitted primitives are not correct. Recall increases as expected, reflecting the successful retrieval of the primitives over time.

The effect of the set of SVM is exemplified with selected frames of dataset #3, shown in Figure 5.

(Frame #4) (Frame #10) (Frame #14)

Figure 5: Frames from dataset #3 with superimposed detected primitives, showing plane primitives discarded by the SVM and correctly replaced by refitting a cylinder in later frames.

(Frame #70) (Frame #91) (Frame #93) (Frame #104)

Figure 6: Frames from dataset #5 with superimposed detected primitives, showing the unstable behaviour of the algorithm when dealing with challenging shapes.

## 7 Discussion

### 7.1 Datasets

Because of the large variation of related approaches mentioned in Section 2, it is challenging to compare the results of the proposed approach to results of existing methods. Many methods [10, 14, 17, 25] are designed to work on a single frame, without iteratively updating the detection in a stream scenario. Some methods segment objects without assigning them to a certain class [2, 19], while many ML-based methods perform an instance-based segmentation [8, 20, 24], rather than structural modeling using geometric primitives. The most similar approach to ours [13] focuses on sparse point clouds, which are not suitable for the segmentation step employed in this work, making them difficult to compare. We do not think that a comparison to a segmentation method using different classes [10, 14, 16, 17, 23, 25] would be meaningful.

As there are no publicly available suitable dense point cloud datasets with labeled 3D primitives, we were forced to create our own set of test sequences. As a result, comparing the performance of our approach to others in the literature remains challenging. To improve this situation, we will release our datasets to the public for further research.

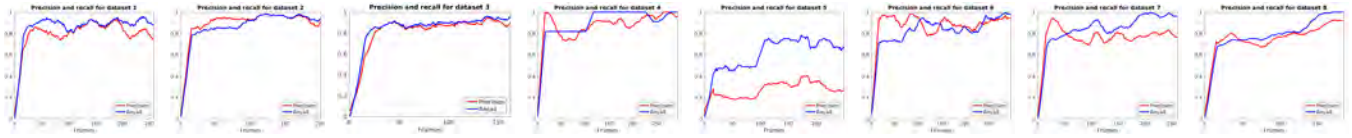
### 7.2 Interpretation

Upon closer investigation, the more regular certain shapes are, the easier it becomes to represent them as geometric primitives. When the geometry becomes more complex, the algorithm tries to model it with the known primitives, and it becomes locally unstable in-between frames, oscillating between primitive types. This behavior is shown in Figure 6, running on dataset #5. The lowest precision and recall performance is achieved, because the shapes of the two bottles are ambiguous and challenging to model with the primitives. In this case, the algorithm discards and refits shapes to the bottles, as all types partially fit, a fact that is reflected in the precision. A solution for this instability would require adding more primitive types or employing a hybrid method [14] that models object details as meshes.

Like any other sensor based on structured light, our sensor has difficulties dealing with reflections or transparent objects. This can also be seen in Figure 6, where the neck of the bottle from dataset 5 is not captured by the sensor due to its transparency. A thorough analysis on the capabilities of the sensor is described in the literature [5].

In datasets #2, #7 and #8, the edges of the stacked books are partly merged together into the same plane. This behavior can be explained by the lack of context, which forces the algorithm to rely only on the geometry. A possible improvement for these cases is the addition of contextual relations or specific constraints between primitives [9].

One drawback of our current implementation is its inability to model small objects. This shortcomiung is mainly caused by the segmentation step based on the orientation of surface normals. When modeling small spheres and cylinders, even with a locally smooth surface, the angles between their normals are above the acceptable thresholds for performing reliable segmentation. The effect is that inliers are categorized as cluster edges instead of surfaces. Consequently, those points are not taken into consideration for primitive



(Dataset #1) (Dataset #2) (Dataset #3) (Dataset #4) (Dataset #5) (Dataset #6) (Dataset #7) (Dataset #8)

Figure 7: Average-filtered precision and recall of the primitive fits in datasets #1-#8 over time.

(Dataset #1) (Dataset #2) (Dataset #3) (Dataset #4) (Dataset #5) (Dataset #6) (Dataset #7) (Dataset #8)

Figure 8: Examples of fit primitives in datasets #1-#8, with various object configurations.

fitting. A possible improvement for small primitives is the use of adaptive thresholds for the geometric segmentation, dependent on cluster scale and expected noise level. Such an adaptive thresholding will likely improve the overall performance of the algorithm, as the segmentation step has a crucial influence on the detection of primitives.

Although our current implementation is running close to real-time on desktop hardware, we admit that it cannot deliver interactive rates on mobile devices yet. This is mainly a software engineering limitation, owed to a heterogeneous setup of experimental tools without proper integration and optimization. Nothing in our implementation precludes an optimized version which is real-time capable on mobile hardware.

## 8 Conclusion

In this paper, we present an approach for structural modeling of indoor static scenes using planes, spheres and cylinder as geometrical primitives. The method is capable of inferring the missing geometry from point clouds in an incremental way. Outdated fits are discarded automatically by an SVM classifier, improving the modeled structure of the scene over time.

Being suited for a stream of point clouds coming from a mobile device, the system outputs a compact representation of the scene as a set of parametric models, ready to be further used by AR applications. With this thought in mind, further work includes improving the algorithm for real-time performance on mobile devices.

## Acknowledgments

The authors wish to thank Rafael Roberto for his efforts. This work was partially funded by FFG grant 859208.

## References

[1] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In *Readings in computer vision*, pp. 726–740. Elsevier, 1987.

[2] D. Holz, S. Holzer, R. B. Rusu, and S. Behnke. Real-time plane segmentation using rgb-d cameras. In *Robot Soccer World Cup*, pp. 306–317. Springer, 2011.

[3] P. V. Hough. Method and means for recognizing complex patterns, Dec. 18 1962. US Patent 3,069,654.

[4] J. Huang and S. You. Detecting objects in scene point cloud: A combinational approach. In *3DTV*, pp. 175–182. IEEE, 2013.

[5] M. Kalantari and M. Nechifor. Accuracy and utility of the structure sensor for collecting 3d indoor information. *Geo-spatial information science*, 19(3):202–209, 2016.

[6] H. S. Koppula, A. Anand, T. Joachims, and A. Saxena. Semantic labeling of 3d point clouds for indoor scenes. In *NIPS*, pp. 244–252, 2011.

[7] Y. Li, X. Wu, Y. Chrysathou, A. Sharf, D. Cohen-Or, and N. J. Mitra. Globfit: Consistently fitting primitives by discovering global relations. In *ACM Transactions on Graphics (TOG)*, vol. 30, p. 52. ACM, 2011.

[8] J. McCormac, A. Handa, A. Davison, and S. Leutenegger. Semanticfusion: Dense 3d semantic mapping with convolutional neural networks. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 4628–4635. IEEE, 2017.

[9] T. Nguyen, G. Reitmayr, and D. Schmalstieg. Structural modeling from depth images. *TVCG*, 21(11):1230–1240, 2015.

[10] S. Ochmann, R. Vock, R. Wessel, and R. Klein. Automatic reconstruction of parametric building models from indoor point clouds. *Computers & Graphics*, 54:94–103, 2016.

[11] S. Oesau, F. Lafarge, and P. Alliez. Indoor scene reconstruction using feature sensitive primitive extraction and graph-cut. *ISPRS Journal of Photogrammetry and Remote Sensing*, 90:68–82, 2014.

[12] F. Remondino, D. Lo Buglio, N. Nony, and L. De Luca. Detailed Primitive-Based 3d Modeling of Architectural Elements. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pp. 285–290, July 2012. doi: 10.5194/isprsarchives-XXXIX-B5-285-2012

[13] R. Roberto, H. Uchiyama, J. P. Lima, H. Nagahara, R. Taniguchi, and V. Teichrieb. Incremental structural modeling on sparse visual slam. *IPSJ Transactions on Computer Vision and Applications*, 9(1):5, 2017.

[14] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz. Close-range scene segmentation and reconstruction of 3d point cloud maps for mobile manipulation in domestic environments. In *IROS*, pp. 1–6. IEEE, 2009.

[15] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, May 9-13 2011.

[16] R. F. Salas-Moreno, B. Glocken, P. H. Kelly, and A. J. Davison. Dense planar slam. In *ISMAR*, pp. 157–164. IEEE, 2014.

[17] R. Schnabel, R. Wahl, and R. Klein. Efficient ransac for point-cloud shape detection. In *Computer graphics forum*, vol. 26, pp. 214–226. Wiley Online Library, 2007.

[18] P. Schneider and D. H. Eberly. *Geometric tools for computer graphics*. Elsevier, 2002.

[19] K. Tateno, F. Tombari, and N. Navab. Real-time and scalable incremental segmentation on dense slam. In *IROS*, pp. 4465–4472. IEEE, 2015.

[20] L. P. Tchapmi, C. B. Choy, I. Armeni, J. Gwak, and S. Savarese. Segcloud: Semantic segmentation of 3d point clouds. *arXiv preprint arXiv:1710.07563*, 2017.

[21] P. H. Torr and A. Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *CVIU*, 78(1):138–156, 2000.

[22] A. Ückermann, C. Elbrechter, R. Haschke, and H. Ritter. 3d scene segmentation for autonomous robot grasping. In *IROS*, pp. 1734–1740. IEEE, 2012.

[23] J. Ventura and T. Hollerer. Online environment model estimation for augmented reality. In *ISMAR*, pp. 103–106. IEEE, 2009.

[24] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1912–1920, 2015.

[25] J. Xiao and Y. Furukawa. Reconstructing the worlds museums. *IJCV*, 110(3):243–258, 2014.

[26] X. Xiong and D. Huber. Using context to create semantic 3d models of indoor environments. In *BMVC*, pp. 1–11, 2010.