Joint Classification and Contour Extraction of Large 3D Point Clouds

Timo Hackel, Jan D. Wegner, Konrad Schindler

Photogrammetry and Remote Sensing, ETH Zürich – firstname.lastname@geod.baug.ethz.ch

Abstract

We present an effective and efficient method for point-wise semantic classification and extraction of object contours of large-scale 3D point clouds. What makes point cloud interpretation challenging is the sheer size of several millions of points per scan and the non-grid, sparse, and uneven distribution of points. Standard image processing tools like texture filters, for example, cannot handle such data efficiently, which calls for dedicated point cloud labeling methods. It turns out that one of the major drivers for efficient computation and handling of strong variations in point density, is a careful formulation of per-point neighborhoods at multiple scales. This allows, both, to define an expressive feature set and to extract topologically meaningful object contours.

Semantic classification and contour extraction are interlaced problems. Point-wise semantic classification enables extracting a meaningful candidate set of contour points while contours help generating a rich feature representation that benefits point-wise classification. These methods are tailored to have fast run time and small memory footprint for processing large-scale, unstructured, and inhomogeneous point clouds, while still achieving high classification accuracy. We evaluate our methods on the semantic3d.net benchmark for terrestrial laser scans with $> 10^9$ points.

Keywords: Semantic Classification, Scene Understanding, Point Clouds, LIDAR, Contour Detection, big data

1. Introduction

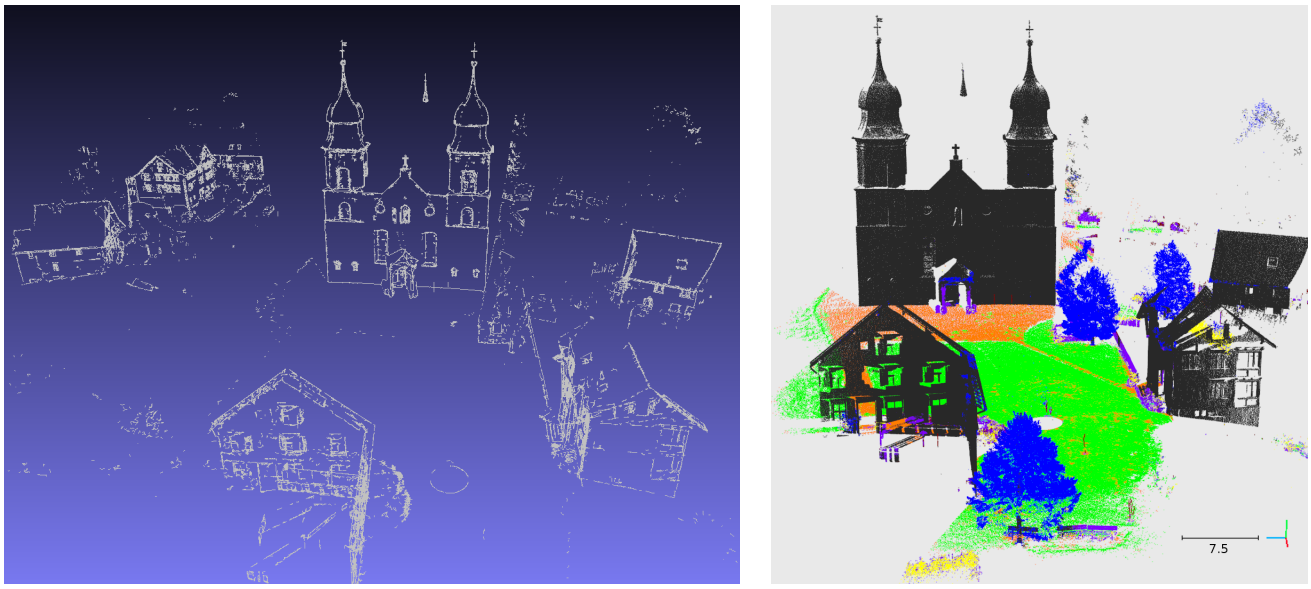
With advanced measurement technologies, like terrestrial laser scanners, it takes fairly little effort and time to collect millions to billions of 3D points in various environments and for many different applications. Yet, large collections of 3D data are hard to process due to their lack of structure. This is especially true in natural scenes, where not only the captured data itself is challenging but also the many possible variations of objects within the scene.

Most applications avoid working directly on these challenging data sets by transferring point clouds into different representations, such as triangle meshes or more CAD-like structures, e.g. [35, 22, 51]. The most common CAD model is Constructive Solid Geometry, where parametric 3D solids are fitted to the data as shape primitives. Such an approach quickly reaches its limits in complex scenes, where large portions of the geometry are not covered by the primitive library. A more generic approach is to extract contours directly in point clouds and use them as representation (Fig. 1(a)). At first glance this step might appear of little advantage, since edges can also be extracted from a triangle mesh or parametric model. But, arguably, it is advantageous to already know the contours as an input for mesh triangulation or surface fitting, so as to avoid modelling errors that frequently occur around contours. Especially mesh triangulation is known to smooth out edges.

Intuitively, contours should thus be detected *before* surface reconstruction to support segmentation. Interactive modelling systems in graphics and vision [41] and in commercial mapping [43, 11] actually operate in this way. In practice it is challenging to detect contours in point clouds. The main reasons are not only the unstructured data but also the unclear definition of what constitutes a contour. While high curvature values play an important role there are also other factors, which cannot be neglected. For instance, the curvature where two building walls meet can in some cases be quite a bit lower than on rough surfaces, e.g. meadows. Obviously, a human operator uses additional contextual cues to identify what constitutes a contour. This situation, where one must exploit diffuse knowledge that is hard to make explicit and formalise, calls for statistical machine learning.

Much the same can be said of semantic segmentation, i.e., the task to assign each individual data point a semantic class label (Fig. 1(b)). Early work on semantic segmentation was concerned with airborne data. The point data was converted to a 2.5D range image (rastered height field), so that it can be treated with image processing techniques [18, 14]. Newer methods, especially those concerned with terrestrial datasets consisting of multiple scans, follow a more general approach and work directly on 3D points [6, 53, 12, 48]. Very recently, Convolutional Neural Networks (CNN) have shown promising results also for 3D data, following their success in image interpretation and speech recognition [26, 38, 36].

It is easy to see that semantic segmentation and contour detection are related, both because contours often coincide with object boundaries and because the geometric properties of contours may vary depending on the object class (c.f. "breaklines" on natural terrain or on man-made objects). In this paper we extend preliminary work [16, 15] on contour detection and semantic classification by showing ex-



(a) (b)

Fig. 1. Examples of (a) extracted contours and (b) point-wise semantic classification; *gray:* buildings, *orange:* man made ground, *green:* natural ground, *yellow:* low vegetation, *blue:* high vegetation, *purple:* hard scape, *pink:* cars

perimentally that it is beneficial to couple the two tasks and solve them jointly.

2. Related Work

Contour detection is closely related to edge and boundary detection in 2D images, which is still an active field of research, in spite of long-standing classics like the Canny edge detector [4]. Supervised machine learning has brought about important advances in 2D contour detection, where recent algorithms approach human performance [21, 36, 52].

One option to construct 3D lines in the multi-view setting is to triangulate them from 2D line segments extracted in images [34, 30, 17]. The outcome are mostly short, disconnected, straight line segments in 3D that can certainly support certain modelling tasks, but are a long way from connected, topologically meaningful wireframe. In many cases point clouds are not generated via multi-view matching or the source images are not available; such that contour detection must start directly in 3D space. Early work in computer graphics has confirmed the intuition that 3D contours can be identified better with more complex feature sets, which go beyond simple curvature values [31]. Our contour detector relies on a hypothesize-and-verify strategy: an overcomplete set of line candidates is extracted via shortest path search (based on point-wise features), followed by a global pruning step that selects those candidates that fulfill appropriate topological conditions like long-range connectivity. The technically most related work we are aware of comes from the fields of medical imaging [46, 45] and topographic mapping [44, 28, 47]. A recurrent finding in these works is that the local evidence, i.e., the likelihood that an individual point of a line lies on a contour, is best learned from a large set of labeled training data (rather than guessed and hand-coded).

Early work on semantic point cloud segmentation transformed the points (recorded from airborne platforms) into other representations such as regular raster height maps, in order to simplify the problem and benefit from the comprehensive toolbox of image processing functions [18, 24, 14, 32]. Recent work follows a more generic approach that can also deal with true 3D data. Learning and prediction operate directly on 3D points [5, 6, 29, 53], such that one can also process data which cannot be reduced to height maps in a straight-forward manner, in particular terrestrial data generated from multiple imaging or scan positions, and mobile mapping data [48, 9].

Training a good model requires an expressive feature set. A large number of 3D point descriptors has been developed, which typically encode geometric properties within the point's neighborhood, like surface curvature, surface normal orientation, e.t.c.. Popular descriptors are for example spin images [19], fast point feature histograms (FPFH) [33] and signatures of histograms (SHOT) [42]. One drawback of these rich descriptors is their high computational cost. While computation time is not an issue for small point sets (e.g., sparse key points), it is a crucial bottleneck when *all* points in a large point cloud shall be classified. A faster alternative if working on range images instead of 3D point clouds is the NARF operator that is a widely used for key point extraction and description in the robotics community [40, 39]. In order to achieve robustness against viewpoint changes, it explicitly models object contour information. A computationally cheaper alternative for full 3D point cloud features is derived from the structure tensor of a point's neighbourhood [8], and the point distribution in oriented (usually vertical) cylinders [27, 48]. Neural networks (usually of the deep, convolutional network flavour)

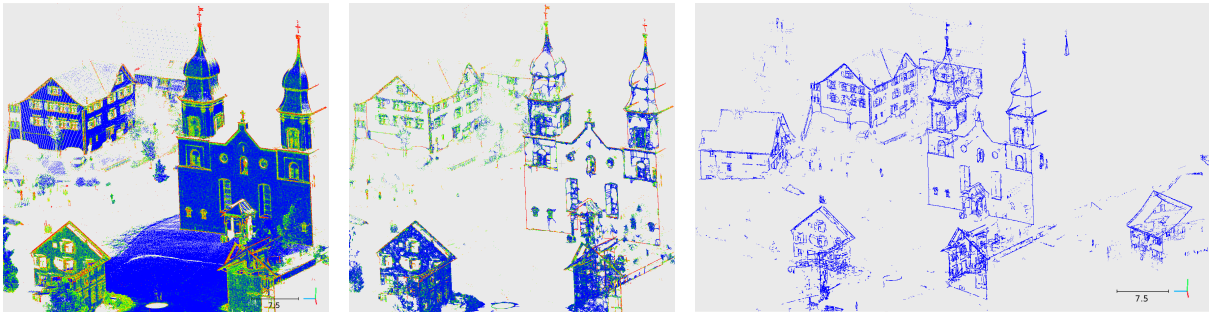


Fig. 2. Illustration of our contour detection pipeline. (left) A binary classifier predicts point-wise contour scores (red: high contour probability, blue: low contour probability); (middle) Seed points with high contour scores are linked into an over-complete graph of contour candidates; candidates are re-scored with another round of classification; (right) candidates are pruned to an optimal set of contours by MRF inference.

offer the possibility to completely avoid heuristic feature design and feature selection. They are at present immensely popular in 2D image interpretation, recently deep learning pipelines have been proposed for voxel grids [23, 50, 26] and RGB-D images [38], too. These deep learning techniques inherently capture appearance based features as well as geometric object properties. Yet, they are computationally demanding and have so far been limited to comparatively small point clouds.

Our solution is in some sense related in that we aim to use correlations between object classes and the geometric properties of contours. In both cases the hope is that mutual feedback between semantics and geometry will benefit the overall interpretation. Our approach is simpler and less elegant, but much more efficient to compute and more generally applicable in case of limited training data.

3. Approach

Requirements for our processing pipeline are efficiency in terms of both run-time and memory footprint, such that the algorithm can be applied to point clouds of realistic size. In fact, the bottleneck in terms of efficiency are the large number of 3D nearest-neighbor queries to construct the neighborhood structure for feature and contour computation. We build on the following, simple key insight to gain speed: to (approximately) characterise a larger neighbourhood, it is sufficient to use a proportionally smaller subset of points. It may seem that this strategy sacrifices accuracy for speed. But the computational savings are so large that one can side-step scale selection and instead exhaustively compute features over a large range of scales. Empirically, this multi-scale coverage *increases* performance. For details, refer to Section 4.

For contour detection, we start with computing per-point contour probabilities with a binary semantic classifier, to distinguish between points on contours and non-contours (Fig. 2). Here we are not trying to strictly detect semantic or geometric transactions. Instead for us contours are more abstract and ill defined. In our framework users can define contours by providing a training set for the semantic classifier.

Second, it tries to find regularly spaced points with high likelihoods, and link them into an over-complete graph of candidate contours. In a final step, an optimal subset of candidates is extracted with a higher-order random field, as described in section 5. This procedure is in line with recent computer vision and medical imaging research [46, 28].

Please note that we explicitely integrate information from semantic multiclass classification into the contour classification problem. The reason for this is, that labeling ground truth contours in 3D space is tedious work. Thus the training set is smaller than the entire semantic3D.net training set used to train the initial per-point, multi-class classifier. Therefore, we extend the feature set of the semantic classifier for contour detection with the class conditional probabilities of an additional multi-class semantic classifion stage. For the multiclass classification contour information is used only implicitly by integrating contour features into the feature set without an additional classification stage.

4. Semantic Classification

Semantic classification of large point sets is a computationally demanding problem. The main bottleneck turns out to be the computation of the point neighborhoods, which is required to compute geometric features. We approximate neighborhoods with multi-scale pyramids, so that small volumes can be described with a small number of neighbors at a fine scale and large volumes with a small number of neighbors at a coarse scale. The feature set described in section 4.2 extends the work of [48] to multi-scale, and adds new features that aim at contour extraction. Details of our training routine are given in section 4.3.

4.1. Neighborhood approximation

Typically, search methods are either of geometrical nature, like radius search, or select the k nearest neighbours. From a conceptual point of view radius search has advantages because a constant radius guarantees that always the same volume is covered. Yet, radius search is impractical for large radii in very dense point clouds due to its high computational cost. Another disadvantage of radius search is the varying number of neighbours (in the worst case none), which can be a problem for further computations. The alternative, k nearest neighbors, can be seen as an adaptive version, in which the radius is increased until

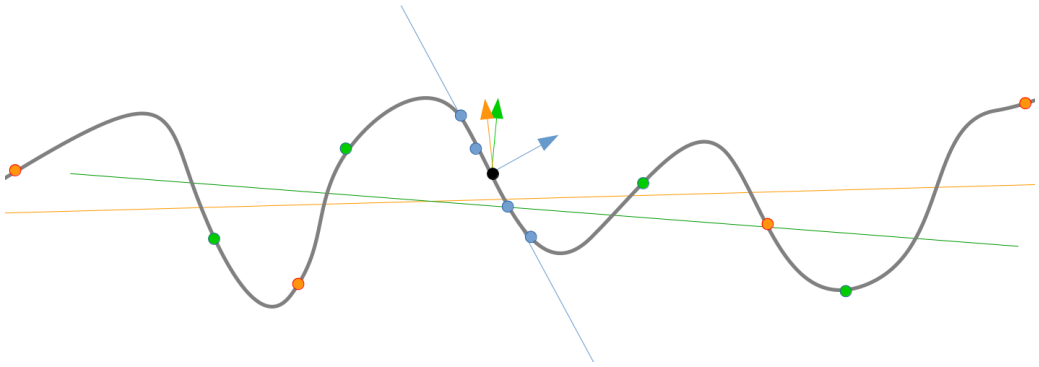


Fig. 3. Sampling a point cloud at different scales (denoted by colors) reveals different properties of the underlying surface – here different surface normals (colored arrows).

it includes exactly k neighbors. For constant point density, both search strategeis give the same output.

Here, we prefer k nearest neighbors over radius search, and approximate constant point density in sufficiently dense parts of the point cloud by performing uniform down-sampling with a voxel grid filter. A range of different voxel sizes are used for voxel grid filtering, so as to enable efficient neighbour search across different scales [3]. The resulting set of search trees forms a multi-scale pyramid, similar in spirit to a discrete image scale space. It has been shown that features extracted with a small neighborhood of $k = 10$ are more discriminative than larger neighborhoods [49]. We apply this insight to our problem and use $k = 10$ to search within each of the scales of our pyramid. While fine scales capture detailed information of the surface, coarser scales cover a larger search volume by selecting $k = 10$ representative samples. This strategy greatly improves the computational efficiency. Another beneficial effect is the reduced size of the search trees. Uniform down-sampling reduces the point cloud size, and hence also the search tree size, which significantly speeds up search queries especially for coarse scales[1].

4.2. *Feature extraction*

Our set of features is designed for purely point-based classification, given points and their nearest neighbors. We refrain from using colour or intensity information, which is not always available, and empirically did not improve the classification in our tests. We work with local neighborhood $\mathcal{P}^N$ of point $\mathbf{p}$ because for large point clouds it becomes quickly infeasible to generate features based on much larger neighborhoods ("neighbors of neighbors") due to memory and computational limitations. We follow [48] and use 3D features based on eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$ and corresponding eigenvectors $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ of the covariance tensor $C = \frac{1}{k} \sum_{i \in \mathcal{P}} (\mathbf{p}_i - \overline{\mathbf{p}})(\mathbf{p}_i - \overline{\mathbf{p}})^\top$, where $\mathcal{P}$ is the set of k nearest neighbors and $\overline{\mathbf{p}} = \mathrm{med}_{i \in \mathcal{P}}(\mathbf{p}_i)$ is its medoid. The original feature set covers object boundaries only weakly. For this reason we introduce two additional features. Our first addition to the feature set is the normalized first order moment around the largest eigenvector, which we use to identify boundaries between surfaces with different point density.

$$O = \frac{\mathsf{m}_\uparrow^2}{\mathsf{m}_\Uparrow} \qquad \text{with} \qquad \begin{aligned} \mathsf{m}_\uparrow &= \sum_{n \in \mathcal{P}_i^N} \langle \mathbf{p}_n - \mathbf{p}_i, \mathbf{e}_2 \rangle \ , \\ \mathsf{m}_\Uparrow &= \sum_{n \in \mathcal{P}_i^N} \langle \mathbf{p}_n - \mathbf{p}_i, \mathbf{e}_2 \rangle^2 \ . \end{aligned} \tag{1}$$

Strictly, the momentum around the axis $\mathbf{e}_1$ is a 2D moment in the $\mathbf{e}_2\mathbf{e}_3$ plane. However, on smooth surfaces (with an object boundary) there is little variation along the $\mathbf{e}_3$ axis so that we neglect $\mathbf{e}_3$ to obtain a 1D moment. This feature is especially suited to identify occluding and occluded edges following the nomenclature of [7], which defines occluding edges as edges, which cast a shadow and dubs edges caused by a shadow occluded edges.

Another property of edges are different orientations of the separated surfaces. In order to identify points on different surfaces, we use the direction of the first eigenvector, expected to point in the direction of the possible contour, and split the neighborhood into two disjoint sets on either side of the contour: $d_{i,n} = \langle\ p_n - p_i,\ \mathbf{e}_2\ \rangle$. The first set $\mathcal{P}^+$ consists of points with a distance $d_{i,n} \geq 0$, while the remaining points are collected in $\mathcal{P}^-$. We take the medoid within each of these subsets and use their scalar product as additional feature:

$$\begin{aligned} &\mathcal{P}^- = \{\mathbf{p}_n \in \mathcal{P}^N | \langle \mathbf{p}_n, \mathbf{e}_2 \rangle < 0\} \ , \ \mathcal{P}^+ = \mathcal{P}^N \setminus \mathcal{P}^- \\ &R = \left| \left\langle \underset{n \in \mathcal{P}^-}{\mathrm{med}} (\mathbf{n}_n), \underset{n \in \mathcal{P}^+}{\mathrm{med}} (\mathbf{n}_n) \right\rangle \right| . \end{aligned} \tag{2}$$

Furthermore we introduce a line feature, where we again make use of the distance of neighboring points to the eigenvector $d_{i,n}$. Points are divided into "near" neighborhood $\mathcal{C}_{\mathsf{near}}$ of the $\lceil \alpha \cdot N \rceil$ closest points to the tangent and a "far" neighborhood $\mathcal{C}_{\mathsf{far}}$ consisting of the remaining points. For each point we examine the local surface variation $\gamma_n = \lambda_3 / (\lambda_1 + \lambda_2 + \lambda_3)$. The ratio between the average surface

[1] Conceptually similar alternatives like hierarchical octrees [10] exist, but are not investigated here.

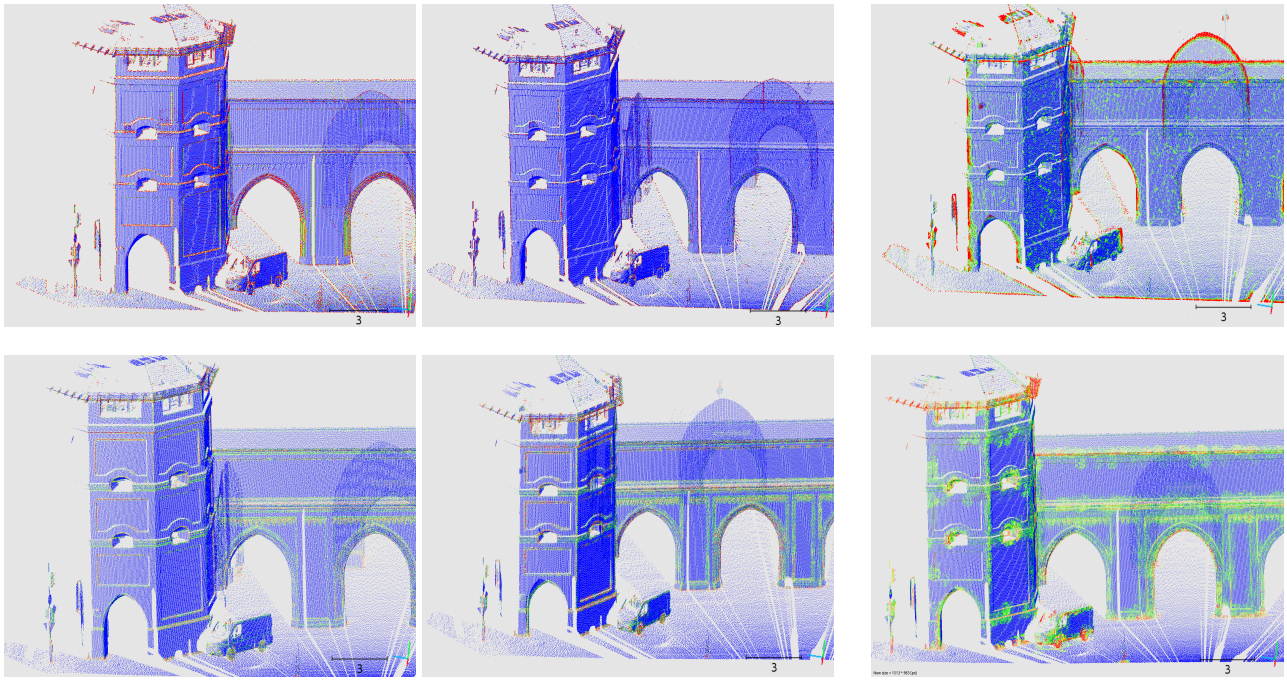


Fig. 4. *Top Row:* Feature values for O *Bottom Row:* Feature values for $1 - R$ on example point cloud computed at 3 different scales: $\{0.025$ m, 0.05 m, 0.4 m$\}$ from left to right.

covariance	Sum	$\lambda_1 + \lambda_2 + \lambda_3$
	Omnivariance	$(\lambda_1 \cdot \lambda_2 \cdot \lambda_3)^{\frac{1}{3}}$
	Eigenentropy	$-\sum_{i=1}^{3} \lambda_i \cdot \ln(\lambda_i)$
	Anisotropy	$(\lambda_1 - \lambda_3)/\lambda_1$
	Planarity	$(\lambda_2 - \lambda_3)/\lambda_1$
	Linearity	$(\lambda_1 - \lambda_2)/\lambda_1$
	Surface Variation	$\lambda_3/(\lambda_1 + \lambda_2 + \lambda_3)$
	Sphericity	λ_3/λ_1
	Verticality	$1 - \lvert\langle[0\,0\,1], \mathbf{e}_3\rangle\rvert$
contour	1st order	O
	2nd order	$\mathsf{m}_{\Uparrow}$
	surface orientation	R
	line feature	Q
height	Vertical range	$z_{\max} - z_{\min}$
	Height below	$z - z_{\min}$
	Height above	$z_{\max} - z$

Table 1. Our basic feature set consists of geometric features based on eigenvalues of the local structure tensor, moments around the corresponding eigenvectors, as well as features computed in vertical columns around the point. Features in the center row are explicitly designed to capture contour information.

variations of the two subsets serves as feature.

$$Q = \frac{\text{mean}_{n \in \mathcal{C}_{\text{near}}}(\gamma_n)}{\text{mean}_{n \in \mathcal{C}_{\text{far}}}(\gamma_n)} \,. \tag{3}$$

The local structure tensor is computed from $k = 10$ points (current point plus 9 neighbors) over 9 scale levels, corresponding to voxels of side-length $[0.025, 0.05, 0.1, 0.2, 0.4, 0.8, 1.6, 3.2, 6.4]$ m. The feature set is summarised in table 1. A visualization of some features at different scales is given in Fig. 4.

4.3. Classification and training

For each point $\mathbf{p}_i \in \mathcal{P}$ we compute a feature vector x_i and train a discriminative learner to predict the class probabilities $p(c_i|x_i)$. By taking the $\arg\max_{c_i} \left(p(c_i|x_i)\right)$ we obtain the most likely class for point $\mathbf{p}_i$. We deploy a Random Forest classifier, which is directly applicable to multi-class problems, by construction delivers probabilities, and has been shown to yield good results in reasonable time on large point clouds [6, 49]. We run grid search with 5-fold cross validation to find the best parametrization of the Random Forest; in our case 50 trees and a depth of approximately 30, with the Gini-index as splitting criterion.

Training on large data sets in 3D space is challenging due to memory restrictions. Furthermore, for polar recordings like those of laser scanners or cameras, the class frequency distribution depends heavily on the instrument's position in the scene. Surfaces near the sensor and perpendicular to the ray direction will contribute much more training samples than distant or slanted objects of the same physical dimensions, hence the distribution learned from a few scans can be severely biased. To resolve this issue in 3D space, a simple solution is to weight errors depending on their distance and angle to the sensor, but to do so the sensor position for every point must still be known. We follow a more pragmatic approach; training data is uniformly down-sampled with voxel grid filtering, so that the number of training samples on objects are approximately independent of the instrument position. As a side effect the number of training samples in densely sampled regions is reduced. Consequently, the training set can cover a larger region (in object space)

with the same memory footprint, which is preferrable to ensure the classifier generalises well.

In order to increase computational efficiency at test time, we also scan the Random Forest to identify features that are never used and therefore do not influence the result. These features are not extracted from the test data. Usually almost all covariance-based features are used. Most importantly, the classifier uses at least some features from each scale level. Since the most expensive computation (even if it is done analytically) is to solve for eigenvector and eigenvalues once per scale, the gains from feature selection are negligible for the basic features from Table 1.

Importantly, the memory footprint at test time is small, and independent of the dataset size, since there is no need to store feature vectors. The features x_i of each point $\mathbf{p}_i \in \mathcal{P}$ can be computed on the fly and discarded after classification.

5. Contour Extraction

Our contour extraction is a three-stage approach in the spirit of [46, 28, 13]. *(i)* We define what we understand as contour by providing labeled training data and use this data to train a discriminative classifier as described in Section 4. This discriminative learner is used to predict class probabilities for a point being a contour or non-contour (background), which we use as low-level evidence for identifying points, which are likely to be a contour; *(ii)* We transition from a set of individual points to connected contour segments by selecting regularly spaced points with high contour scores as seed points, and link them into a connected graph as described in Section 5.1; *(iii)* Lastly, a (higher-order) MRF on the over-complete graph of putative contour segments helps to filter out false positives, Section 5.2.

5.1. Contour candidate generation

An important property of contours is their connectivity over large distances. We follow a hypothesize-and-verify strategy in order to discover long-range interactions. We first generate an over-complete set of contour candidates, and in a second step prune the set to optimal size. This strategy delivers a contour graph in form of many linked line segments, similar to wire-frame models, which do not only consist of line segments but are a collection of linked models, including line segments, b-splines or circles.

We generate contour candidates by selecting points of high contour probabilities $p(c_i = 0|\mathbf{x}_i)$ as seed points, and connecting with 3D shortest path search.

Voxel-grid non-maxima suppression

We uniformly sample seed points in the point cloud, and reduce that initial set with a conservative heuristic for Non-Maxima Suppression (NMS). Non-uniform sampling would risk missing the true contour segment, unless shortest paths are computed in a large neighbourhood around each seed point; which would, in turn, inflate the set of incorrect contour candidates and complicate the classification in Section 5.2. For uniform sampling we once again rely on a (sparse) voxel grid (Fig. 5), with $s = 0.1$ $[m]$ spacing.[2] In a first stage we perform simple thresholding $p(c_i = 0|\mathbf{x}_i) \geq 0.5$ on the most likely point per voxel, so that we find an initial set of potential seeds. In a next step, seed points are pruned that have a neighbour with higher contour likelihood in $\mathcal{P}^N$, unless they are close to the tangent of the contour, assumed to be the eigenvector $\mathbf{e}_1$ of the covariance tensor at the neighbor with the highest likelihood. Finally, points are also removed if they have a more likely neighbour along the tangent within a distance of $< 0.5 \cdot s$.

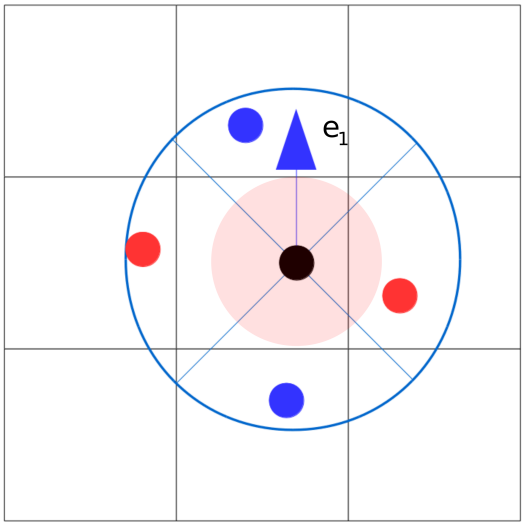


Fig. 5. Voxel-Non-Maxima Suppression: For each voxel the point with the highest probability is stored. If nearby (blue circle) neighboring seed points have a smaller probability they get suppressed (red points) unless they are located in the direction of the expected contour $\mathbf{e}_1$ and are not too close (red circle).

Graph construction

In a next step a neighborhood graph is generated in order to extract shortest paths between seed points. At this point one must take care to correctly handle anisotropic variations of the point density, most notably large differences between horizontal and vertical point spacing, which stem from the polar scanning of slanted surfaces, c.f. Fig. 6.

Our target is to keep the number of connections, and consequently the memory footprint of the neighborhood graph, small. For this reason we split the creation of the neighborhood graph into two tasks. First, a k-nearest neighbor graph with a small $k_1 = 5$ neighborhood is extracted, which is able to connect uniformly sampled areas but fails in the anisotropic case. Second, disconnected regions are identified and connected. For this purpose a larger neighborhood $k_2 = 50$ is computed for each point. These neighbors

[2] Throughout this section, a number of model parameters need to be chosen. These are mostly distances in metric object coordinates, which are easy to derive for a given application. Values quoted refer to terrestrial laser scans of settlement areas.

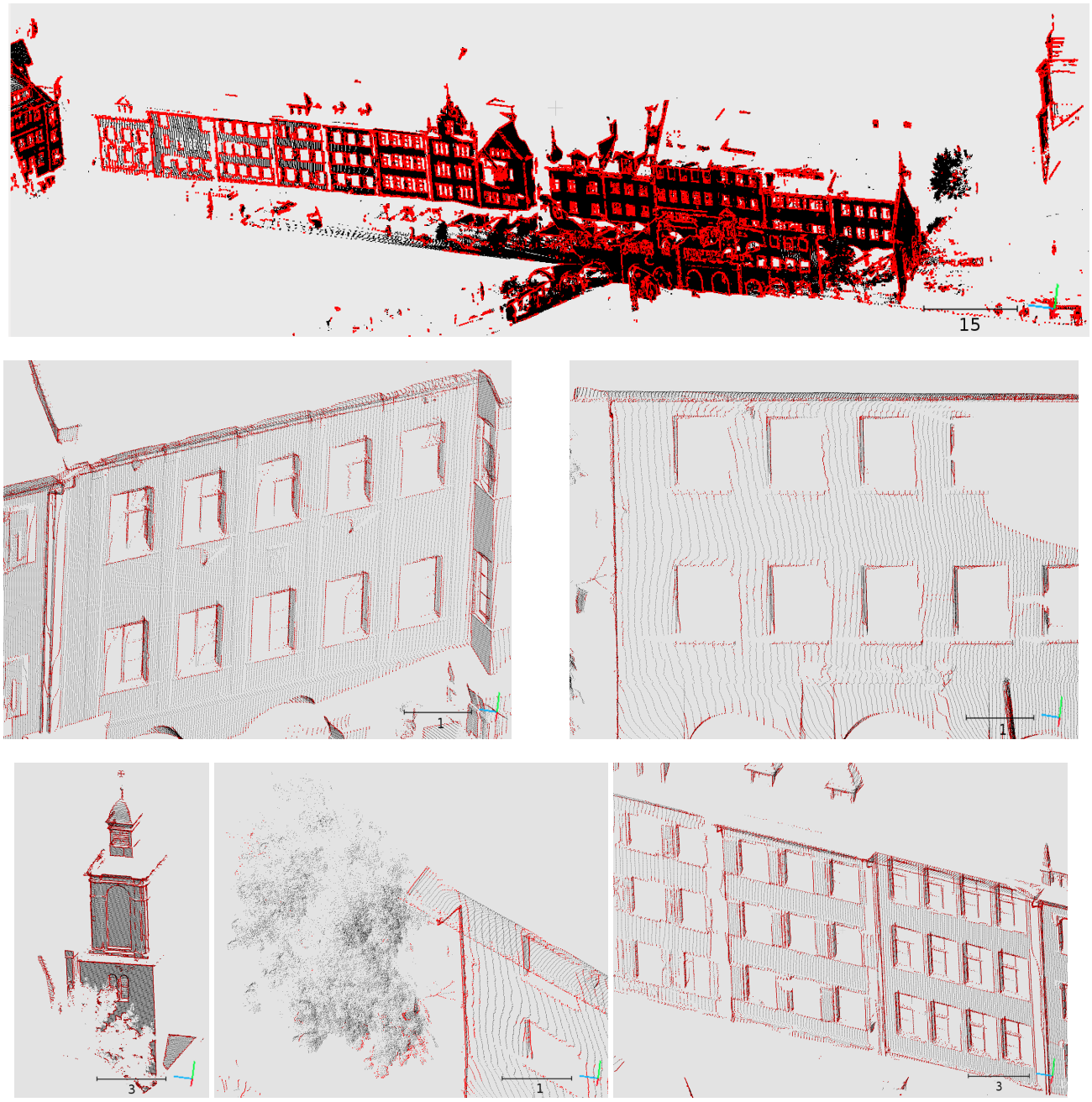


Fig. 6. “Market Square” Laser-scan. While the horizontal point density decreases rapidly due to grazing scan rays, the vertical density decreases much slower and “scan lines” become visible on the facades on the left. Various zooms are given. Please note that we suppressed responses on vegetation in our training set, so that we intentionally do not want to detect contours in tree crowns.

are then connected by using their precomputed k_1 neighborhood and reduced to a minimum spanning tree (MST) with Prim's algorithm. The MST edges are then merged with the edges obtained by k_1. Candidate contours are extracted within this neighborhood graph by linking each seed point with a maximum of $\beta = 15$ neighboring seed points in a radius $r_{\text{link}} = 0.85\,\text{m}$ via Dijkstra shortest paths.

The graph edge costs depend on the contour likelihoods of the segment's endpoints, and on the segment's length (to avoid unjustified short-cuts):

$$e_{ij} = 2 - \frac{p(c_i|\mathbf{x}_i) + p(c_j|\mathbf{x}_j)}{2} - \frac{d^*}{\|\mathbf{p}_i - \mathbf{p}_j\|}\,, \quad (4)$$

with $\|\mathbf{p}_i - \mathbf{p}_j\|$ the Euclidean length of the edge between $\mathbf{p}_i$, $\mathbf{p}_j$, and d^* the smallest distance from either of the two endpoints $\mathbf{p}_i$, $\mathbf{p}_j$ to all other points. In practice it is unnecessary to perform shortest path search in the full graph containing also nodes with a large distance. Instead, we run it only within a local neighborhood $2 \cdot r_{\text{link}}$. We note that more sophisticated methods exist to construct an optimal graph, for instance using ant colony optimization [46]. In our experience shortest path search is sufficient and yields results of similar quality, while being a lot faster to compute.

5.2. Contour edge labeling

The extracted set of contour candidates is designed for high recall, therefore it is over-complete and contains many false positives. To prune false positives from the contour graph we set up a MRF, where contour scores and spatial arrangement of contour candidates are used for graph labeling. The connectivity of the graph naturally leads to higher-order terms that suppress short isolated contours, minimise the number of free endpoints and encourage closed contours.

Unary Term

Standard MRFs consist of a unary term and one or several prior terms. Our formulation follows this design. The unary term encodes properties of the individual contour candidates, such as statistics over the point wise contour scores or the shape of the candidate. We develop a new descriptor, coined *cumulative shape context*, which adapts the ideas of the original shape context (SC, [1]) to our problem. The main idea of SC is to encode relative positions $\mathbf{v}_{ij} = \mathbf{p}_j - \mathbf{p}_i$ of a point $\mathbf{p}_i$ and its neighbors $\mathbf{p}_j$ in a polar histogram is directly applicable to 3D space. However, contours do not consist of a single point $\mathbf{p}_i$ but of many, therefore all points on a contour should be considered in order to obtain a discriminative descriptor of a putative contour's shape. An additional requirement for our descriptor is scale and rotation invariance. To achieve scale invariance and at the same time simplify the descriptor to 1D, we discard relative distance and encode only directions, by normalising the vectors. Rotation invariance is achieved by projecting onto the canonical direction $\mathbf{v}_{se} = \mathbf{p}_e - \mathbf{p}_s$ defined by the start point $\mathbf{p}_s$ and end point $\mathbf{p}_e$ of the contour candidate.

Information from all points along the contour candidate is merged by simply summing their individual histograms into a single one: we visit every point $\mathbf{p}_i$ in turn, and generate a vector v_{ij} to each other point of the contour:

$$v_{ij} = \left\| \left\langle \frac{\mathbf{v}_{se}}{\|\mathbf{v}_{se}\|}, \frac{\mathbf{v}_{ij}}{\|\mathbf{v}_{ij}\|} \right\rangle \right\| \;\; \forall\, i,j \in \{s,\ldots,e\} : i \neq j\;. \quad (5)$$

v_{ij} is a 1D value and is added to a histogram with 5 equally spaced bins. To encode overall straightness / curvature of the contour candidate, we compute the same histogram again, but this time normalised by projection onto tangent (first eigenvector $\mathbf{e}_{1,i}$):

$$w_{ij} = \left\| \left\langle \mathbf{e}_{1,i}, \frac{\mathbf{v}_{ij}}{\|\mathbf{v}_{ij}\|} \right\rangle \right\| \;\; \forall\, i,j \in \{s,\ldots,e\} : i \neq j\;, \quad (6)$$

The bin counts of both histograms $\mathbf{z}_1(v_{ij})$ and $\mathbf{z}_2(w_{ij})$ form the first part of our feature vector to discriminatively learn the unary term. Moreover, the feature vector also includes the contour likelihoods $p(c_i|\mathbf{x}_i)$ along the sequence of points $\mathbf{p}_i$ along the contour segment, again in the form of two discrete distributions. The first one quantises the class probabilities into a global histogram with 5 bins centered at the probabilities $\{0.1,\ 0.3,\ 0.5,\ 0.7,\ 0.9\}$. The second one retains some spatial information, by chopping the contour sequence into 5 segments with equal number of points, and computing their mean probabilities. Given these features $\mathbf{z}_i$ we again employ a random forest to learn the probabilities $p(g|\mathbf{z})$ that a candidate segment is part of a true contour ($g_i = 1$) or a false alarm ($g_i = 0$), and pass the log-likelihoods $h_i = -\log p(g_i|\mathbf{z}_i)$ as unary term into the MRF.

Markov Random Field

By definition, contours should be long connected line features rather than independent, short segments. We treat the contour candidates as variables (nodes) in a graph and encourage long chains of contour segments with a simple Potts-type prior, which aims to to suppress segments that cannot be connected, and encourage segments that close a gap between their start and end nodes, linking up other candidates into longer contours ("start" and "end" are used for linguistic convenience, the graph is undirected). Each contour candidate $\mathbf{l}_i$ defines a clique ℓ_i that also includes all other candidates with which it shares either the start node or the end node. Denoting by $\mathcal{L}_i^s$ the set of contours that connect to $\mathbf{l}_i$ at the start node, and by $\mathcal{L}_i^e$ those which meet $\mathbf{l}_i$ at the end node, we have

$$\ell_i = \begin{cases} \gamma & g_i = 1 \;\textbf{AND}\; \forall\, \mathbf{l}_j \in \mathcal{L}_i^s, \mathcal{L}_i^e : g_j = 0 \\ & \text{(isolated contour)} \\ \delta & g_i = 1 \;\textbf{AND}\; (\exists\, \mathbf{l}_j \in \mathcal{L}_i^s : g_j = 1 \;\textbf{XOR}\; \exists\, \mathbf{l}_j \in \mathcal{L}_i^e : g_j = 1) \\ & \text{(continuation only on one side)} \\ 0 & g_i = 1 \;\textbf{AND}\; \exists\, \mathbf{l}_j \in \mathcal{L}_i^s : g_j = 1 \;\textbf{AND}\; \exists\, \mathbf{l}_j \in \mathcal{L}_i^e : g_j = 1 \\ & \text{(continuation on both sides)} \\ 0 & g_i = 0 \\ & \text{(candidate is not a contour)} \end{cases} \quad (7)$$

Method	$\overline{IoU}$	OA	$t[s]$	IoU_1	IoU_2	IoU_3	IoU_4	IoU_5	IoU_6	IoU_7	IoU_8
Our	0.494	0.850	38421	0.911	0.695	0.328	0.216	0.876	0.259	0.113	0.553
HarrisNet	0.623	0.881	unknown	0.818	0.737	0.742	0.625	0.927	0.283	0.178	0.671
DeepSegNet	0.516	0.884	unknown	0.894	0.811	0.590	0.441	0.853	0.303	0.190	0.050
Image Based	0.391	0.745	unknown	0.804	0.661	0.423	0.412	0.647	0.124	0.	0.058

Table 2. Semantic3d benchmark results on the full data set with a training set of $\sim 1.7 \cdot 10^9$ points and a test set of $\sim 2.3 \cdot 10^9$ points.

Method	$\overline{IoU}$	OA	$t[s]$	IoU_1	IoU_2	IoU_3	IoU_4	IoU_5	IoU_6	IoU_7	IoU_8
Our	0.542	0.862	1800	0.898	0.745	0.537	0.268	0.888	0.189	0.364	0.447
DeepNet	0.437	0.772	64800	0.838	0.385	0.548	0.085	0.841	0.151	0.223	0.423
Image Based	0.384	0.740	unknown	0.726	0.73	0.485	0.224	0.707	0.050	0.0	0.15

Table 3. Semantic3d benchmark results on the reduced data set with the same training set as the full challenge and a test set of $\sim 8 \cdot 10^7$ points.

These cliques are higher-order terms over a variable number of segments, but they can be computed efficiently, since they only depend on direct neighbors. Higher-order render MRF inference expensive, fortunately it turns out that satisfactory minima of the energy

$$E = \sum_i h_i + \sum_i \ell_i. \tag{8}$$

can be found with the Iterated Conditional Modes (ICM) algorithm [2], which we use for its computational efficiency [20].

Our graph is constructed such that overlapping candidates are not penalized. Overlaps are needed to correctly recover T-junctions and crossings that do not coincide with a seed point. They are easily resolved in post-processing, since the overlapping segments share the same vertices.

5.3. Postprocessing

The desired output of the contour detector is a wireframe with few vertices linked by long, straight segments, which can be further processed with conventional CAD software. To get closer to such a minimal representation, while at the same time reducing measurement noise, we employ line simplification. Following [25] we separate the simplification of junctions from the simplification of edges. Junction simplification is implicitly handled in our framework, via the MRF prior (Sec. 5.2). For contour simplification between junctions, we found that the classic Douglas-Peuker algorithm yields good results [37].

6. Experiments

We have conducted experiments on the *semantic3d.net* semantic 3D labeling challenge (`www.semantic3d.net`). Since contour annotations are not available for that dataset, we also test on a smaller set of terrestrial laser scans with manually labeled contours [3].

[3]The same dataset has alreadt been used in [16].

The *semantic3d.net* benchmark provides terrestrial laser scans from a variety of different urban and rural scenes, as well as hand-labeled ground truth for 8 classes. The point clouds are large: they have been recorded with state-of-the-art terrestrial instruments and have a realistic (i.e., high) point density. The training set consists of $\approx 1.7 \cdot 10^9$ points, and the test set of $\approx 2.3 \cdot 10^9$ points[4].

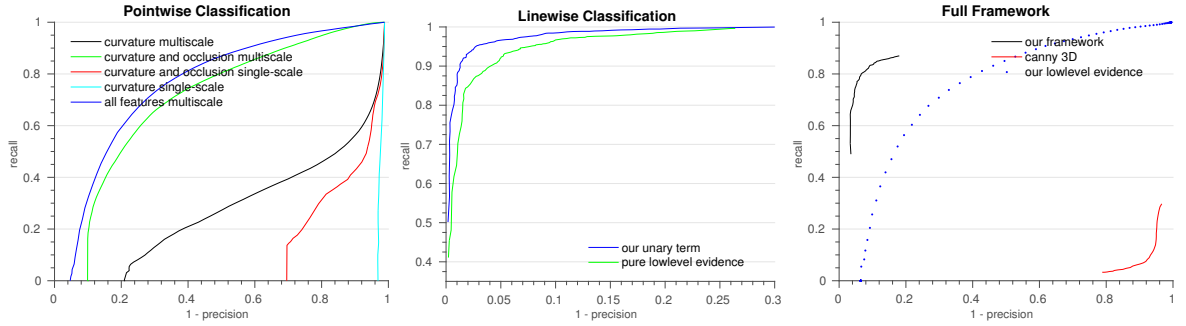
The qualitatively similar, but smaller scan data set from [15] also features high-density terrestrial scans from various geographic locations, and includes ground truth contour dense annotations for one scan, which is exclusively used for evaluation of the entire framework as well as 14 sparsely labeled point clouds, of which five were used for training while the rest was used for the evaluation of point-wise contour evidence.

Our software is implemented in *C++* using the *Point Cloud Library* (`pointclouds.org`), *FLANN* (`github.com/mariusmuja/flann`) for nearest-neighbor search and the *ETH Random Forest Templates* (`https://edit.ethz.ch/igp/photogrammetry/downloads/rforest.zip`) for discriminative learning. All experiments were run on a standard desktop machine with Intel Xeon E5-1650 CPU (hexa-core, 3.5 GHz) and 64 GB RAM. Timings always refer to that hardware configuration. We do parallelize across multiple cores with the help of *OpenMP* (`http://openmp.llvm.org`), but note that a significant speed-up would still be possible with extreme parallelism on the GPU.

6.1. Experiments on the Semantic3d.net Benchmark

We reduce the very large number of training samples in two ways: *(i)* The sub-sampling as described in section 4.3 is applied with a spacing of 1cm; *(ii)* We distort the class frequencies so that the largest class has at maximum $4\times$ the number of training samples of the smallest class, by ran-

[4]A reduced test set with $\approx 79 \cdot 10^6$ points is also available, to allow submissions from computationally very demanding algorithms. We run on both versions for completeness



(a) (b) (c)

Fig. 7. Precision-Recall curves for contour detection. Please note, each stage requires different ground truth, so values are not comparable across diagrams. See text for details.

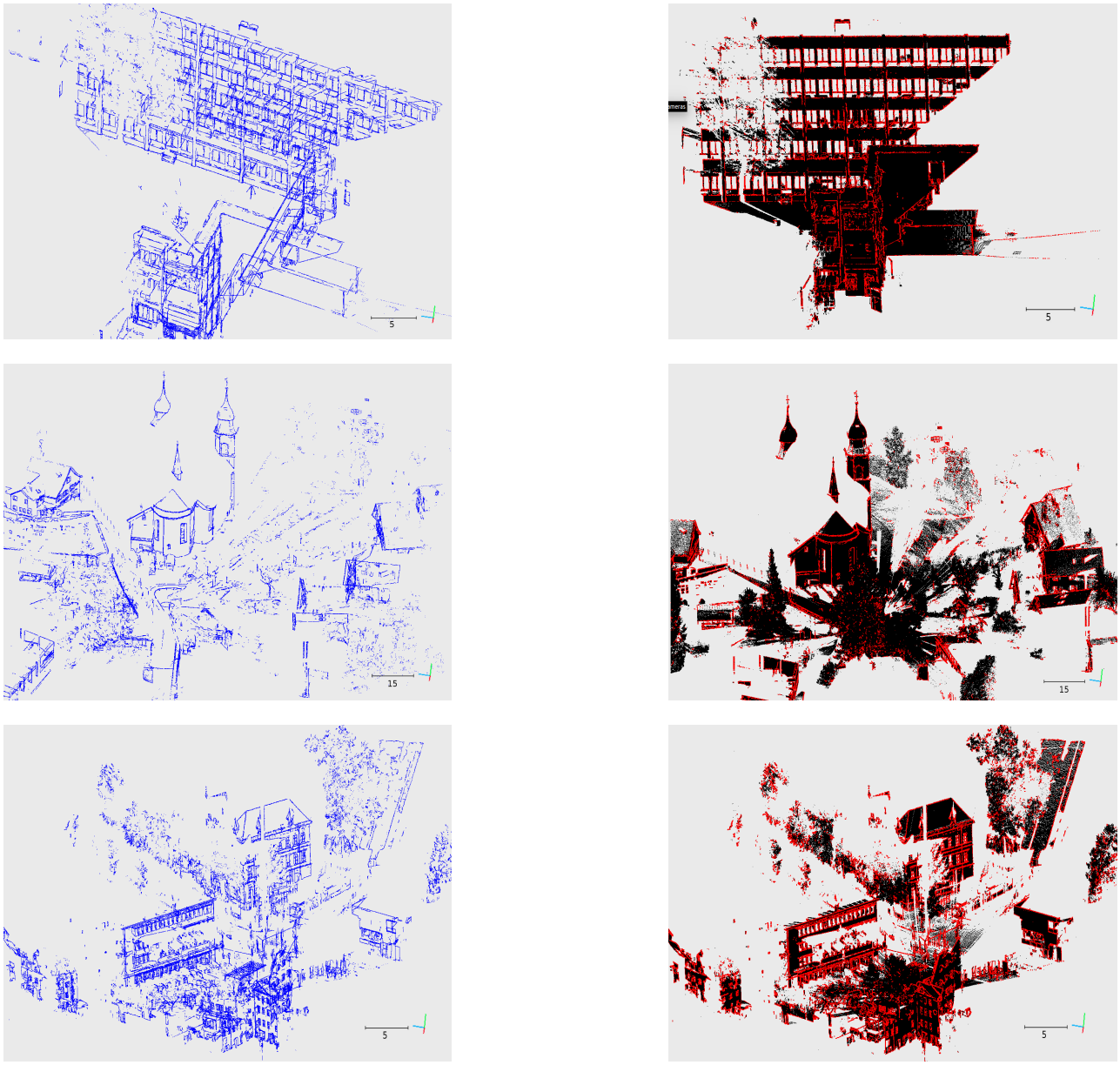


Fig. 8. Visual evaluation 1 to 3: *left column:* detected points on contour (blue); *right column:* original point cloud (black) with detected contour (red).

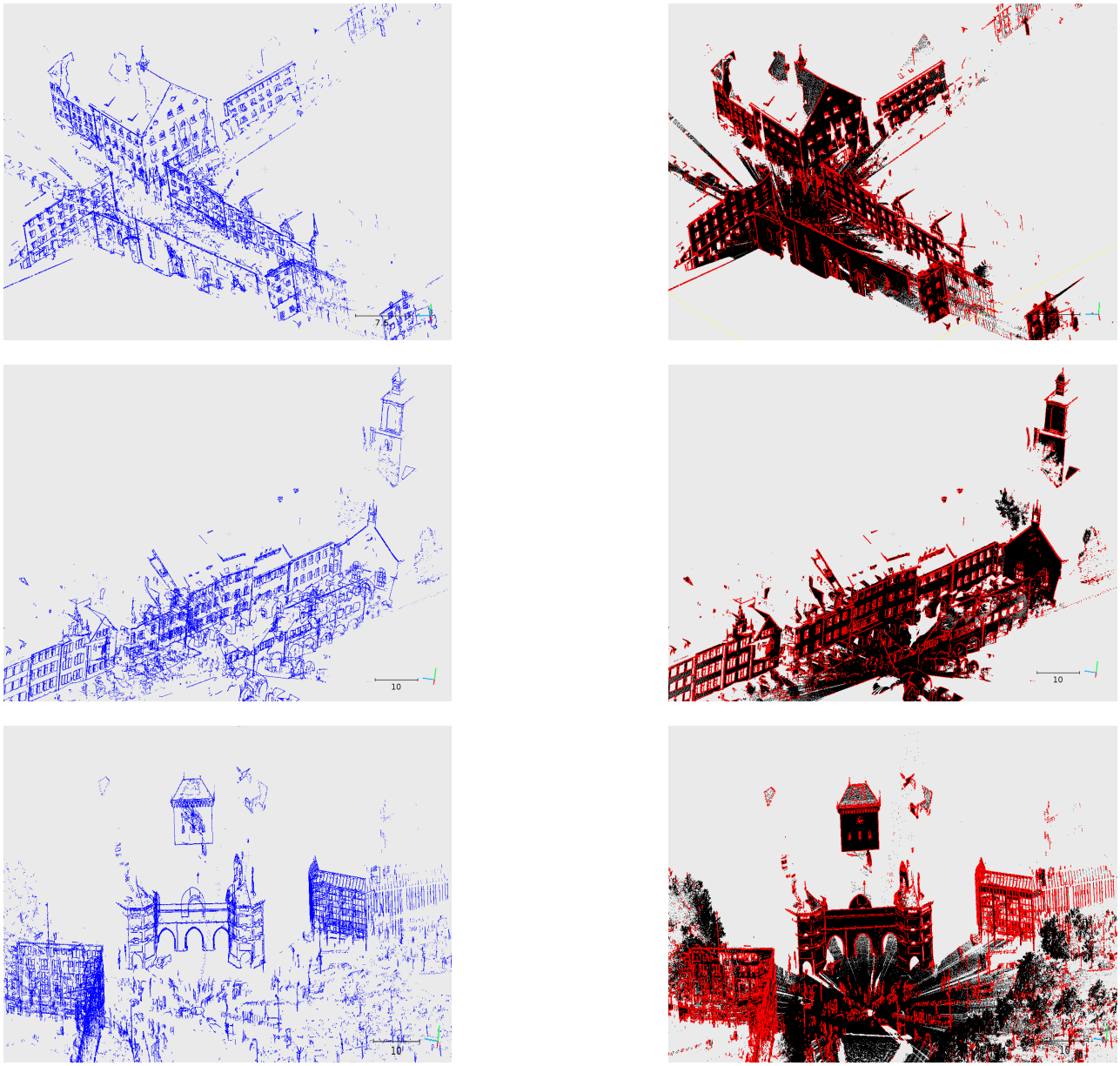


Fig. 9. Visual evaluation 4 to 6: *left column:* detected points on contour (blue); *right column:* original point cloud (black) with detected contour(red).

domly picking training samples per class, which leads to a total of 10^6 samples for training.

The depth of the Random Forest is estimated with Grid Search and 5-Fold Cross-Validation over a parameter range from 30 to 60 with a stride length of 3. In general more trees yield slightly better results, but also linearly increase the computational cost. The number of trees was empirically set to 40 and the Gini index was used as splitting criterion for interior tree nodes. Training and grid search took less than one day for the 1 million training samples. The time given is the time needed to classify the test set and includes loading data from hard disk, feature extraction, classification and saving of results.

The main metric of the benchmark is Intersection over Union (IoU), and its average $\overline{IoU}$ over all classes. IoU is defined as follows:

$$IoU_i = \frac{c_{ii}}{c_{ii} + \sum_{i \neq j} c_{ij} + \sum_{i \neq k} c_{ki}}, \tag{9}$$

where c_{ij} is the count of class i being predicted as j. IoU is a a more sensitive error measure that does not saturate as quickly as alternative metrics. It gives all classes equal weight independent of their point count, and decreases faster than the F_1-score or average class accuracy as the number of false positives (type-1 errors) or false negatives (type-2 errors) increases. Additionally, the overall accuracy (OA) and the runtime for the complete test set are given as auxiliary measures.

Table 2 shows the results of our method on the full data set, while table 3 compares our method with other baselines on the reduced data set. Recently, there have been submis-

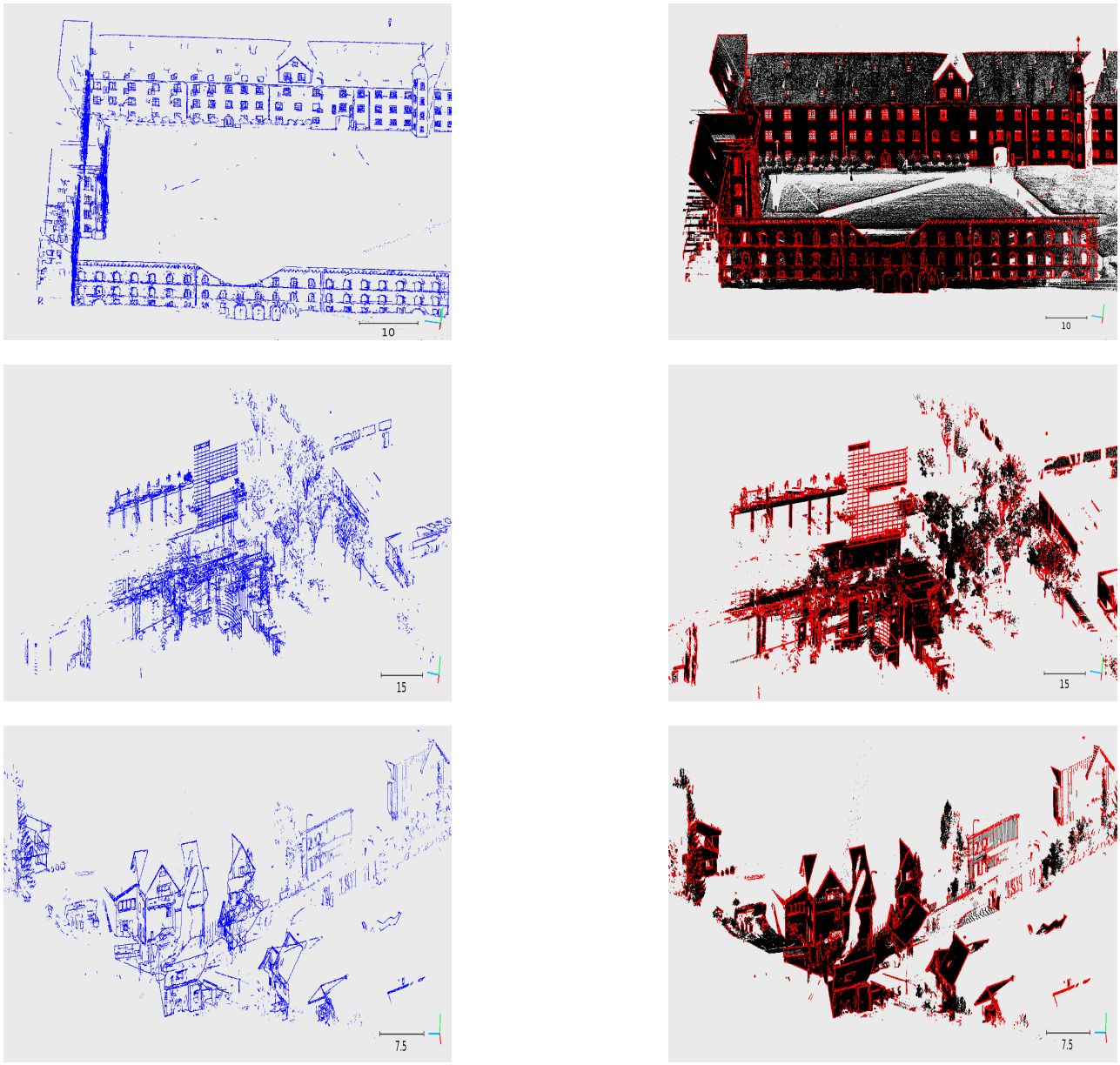


Fig. 10. Visual evaluation 7 to 9: *left column:* detected points on contour (blue); *right column:* original point cloud (black) with detected contour (red).

sion based on deep neural networks (HarrisNet, DeepNet) with a higher $\overline{IoU}$, which comes with high computational costs and their methods are not published, yet. Nevertheless, we reach higher $\overline{IoU}$ and faster runtimes than competitors based on traditional machine learning, as shown in [16]. We note that our method not only achieves good quantitative results, but also offers practically viable runtimes even with the current prototype software: classifying new point clouds takes less than 3 minutes per 10 million points.

6.2. Contour Extraction

A database of 16 laser scans of urban scenes in different countries were used for qualitative evaluation of the contour extraction framework. One scan was manually labeled to serve as ground truth, with $101'614$ points on contours and $7'681'061$ background points. The only baseline for which we could find a 3rd-party implementation is the Canny-style 3D edge detector in the *Point Cloud Library*. In order to facilitate a meaningful comparison, the *tree* and *clutter* classes are excluded. We are forced to do this to avoid a complete failure of the baseline, which otherwise generates huge amounts of false positives on those classes. Making the task easier by excluding the worst distactors is a significant bias *against* our integrated method: it has access to the object class likelihoods, precisely *because* it jointly addresses contours and semantic segmentation.

6.2.1. Pointwise Classification

In a first experiment the influence of semantic classification on the detection of contours was tested with a focus on different feature sets. We also evaluated the performance of multi-scale feature sets versus single-scale features. We trained our Random Forest on $48,964$ positive

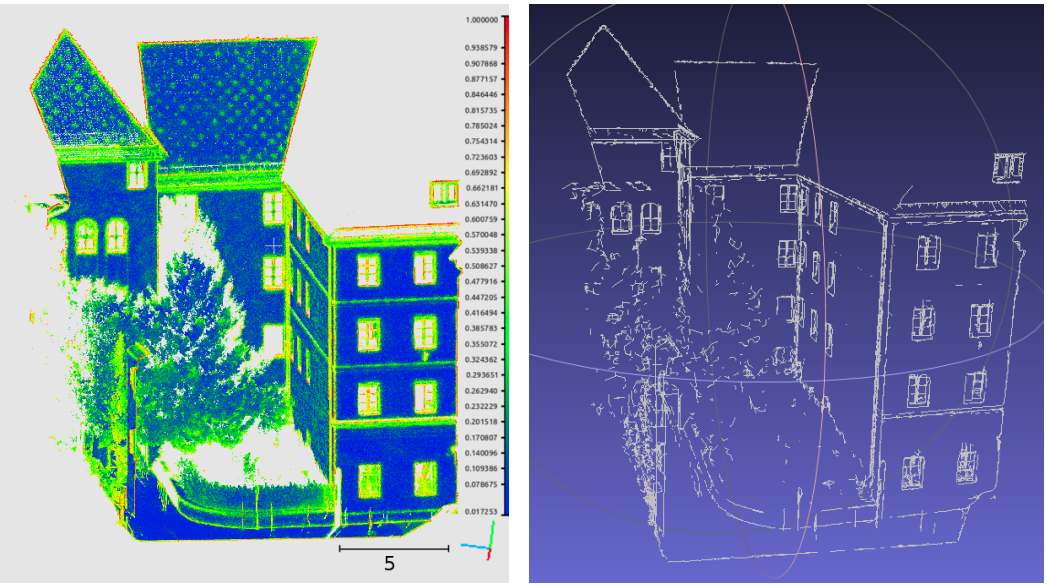


Fig. 11. Point wise score and final result.

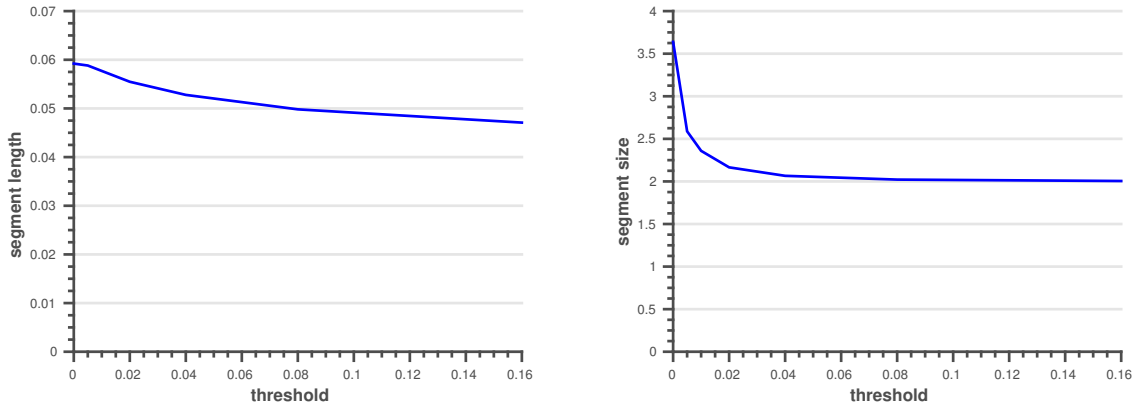


Fig. 12. The statistics show how the average length of contour segments changes by applying douglas peucker. Both graphs show that the length decreases slowly while the number of points on the contour is reduced significantly, which is a desired behavior. The threshold is the parameter of the Douglas-Peucker-algorithm.

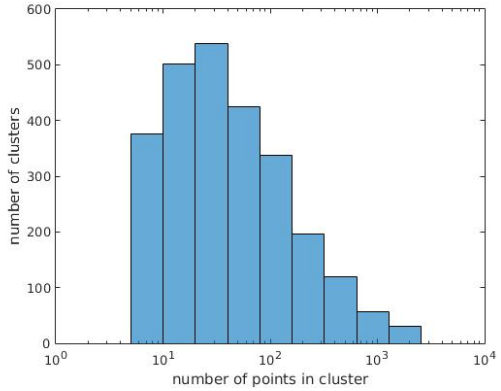
and $85,853$ negative examples, captured in different cities, and use precision-recall curves for the evaluation. The performance of the different feature sets is shown in Fig. 7a. Our feature subsets cover the features used in related methods, which either use curvature as feature or combine curvature with occlusion features. In particular, single-scale curvature and occlusion form the basis of RGB-D edge detection in [7], while multi-scale curvature is the descriptor used by [31]. In our comparison, we train a Random Forest for each of the feature subsets, which can be expected to yield at least as good results as a single threshold per feature dimension. This experiment shows that single-scale detection does not perform well on large and complex point clouds. Multi-scale curvature alone still misses many clear and unambiguous contours, particularly along occlusion boundaries. Our full feature set performs better than the best feature subset and significantly better than the baseline method.

6.2.2. *Linewise Classification*

We also assess the performance of the unary term for contour segment classification. The classifier was trained on a set of 1862 true contour candidates and 832 negatives. Note that each of these candidates consists of multiple points. We again used grid search with the same settings as in the previous experiment for training. The evaluation was performed on a test set with 2227 positive and 1013 negative samples. Here, we compare against the naive baseline feature set consisting only the per-point contour likelihoods $p(c_i = 0|x_i)$, where we use the histograms from the full feature vector. The results are shown in Fig. 7b. It can be seen that adding information about the shape of contour candidates improves the identification of actual contours up to $\approx$20 percent points, especially in the high-precision regime.

6.2.3. *Full Framework*

We evaluate our complete contour detector against the 3D Canny of [7], which is the only published competitor in recent literature, and the only method for which we found an



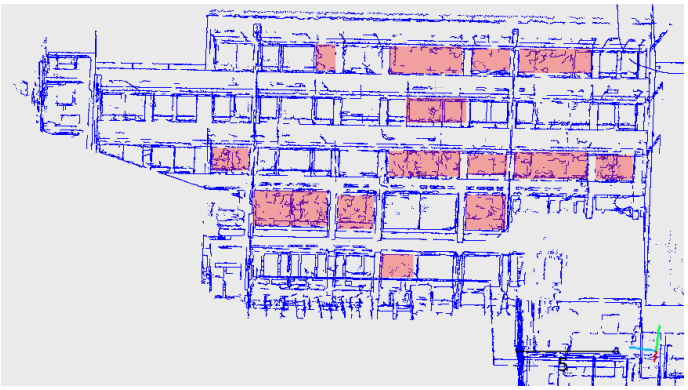


Fig. 13. *left)* Histogram over points per contour segments, not broken up at junctions. *right)* A failure case: if objects with strong geometric structure did not appear in the training set, then they tend to get high contour scores. Red highlights windows with sun blinds. Sun blinds were not covered in our training set and cause many false positives.

Terrestrial	All features			no contour features		
Laser Scans	Recall	Precision	IoU	Recall	Precision	IoU
Man made terrain	0.8758	0.9433	0.9083	0.8757	0.9627	0.9171
Natural terrain	0.8809	0.8921	0.8864	0.9064	0.8774	0.8916
High vegetation	0.9129	0.8102	0.8585	0.8086	0.9254	0.8631
Low vegetation	0.6496	0.4021	0.4967	0.6356	0.5364	0.5818
Buildings	0.9592	0.9878	0.9733	0.9726	0.9813	0.9769
Remaining hard scape	0.7879	0.4878	0.6025	0.8419	0.4743	0.6068
Scanning artefacts	0.5127	0.4595	0.4847	0.3995	0.5039	0.4457
Overall accuracy		0.9215			0.9187	
$\overline{IoU}$-score		0.6514			0.646	

Table 4. Quantitative results for terrestrial laser scans.

accessible implementation (in *PCL*). This method requires "organised point clouds", which are essentially the same as height maps, i.e. they have a 2.5D grid structure. We convert individual scans to "organised clouds" by cube-mapping, noting that 3D Canny in the proposed form is not applicable if the sensor position is unknown, or when the point cloud has been recorded dynamically from a moving platform.

At this point the evaluation faces a subtle, but important problem: the Canny baseline does not return line segments, but only a list of all points that form part of a line. For this reason, our evaluation is only based on point-wise scores. However, our full framework is designed to extract contours in form of a graph, and (during NMS and shortest path search) intentionally discards many point that are not needed to trace the contour. We emulate this subsampling of the points along detected contours by ignoring all false negatives with lie within 3 cm of either a true positive or an already counted false negative. With this filtering we account for the dropped points. Gaps larger than 6 cm still count as false negatives, and alarms where there is no ground truth contour still count as false negatives. As a consequence of the filtering, the graph in Fig. 7c is not a true precision-recall curve over the complete, original point cloud, but it displays the most meaningful comparison we could come up with. One can see that the proposed per-contour scores based on contour likelihoods and shape consistently beat the pure likelihoods , and that our higher-order context model (including the connectivity prior) greatly outperforms the discriminatively trained low-level evidence. The latter already works dramatically better than the 3D Canny, although we made our best effort to tune the baseline for maximum performance.

The final output of our contour detector is a graph of contour edges which connect vertices that are 3D points from the original point cloud as shown in (Fig. 11). In that graph, the average length of a single contour is 187 points, respectively 3.7 m in metric world units for our specific urban outdoor scan data. The median is 28 points, respectively 0.5 m. The full histogram of the extracted contours' length is also shown in Fig. 13. For post-processing we split up the graph at junctions and smooth individual segments with the Douglas Peucker algorithm. The effect on the segment length between two junctions is shown in (Fig. 12). It can be seen that line simplification has only a small impact on the segment length but a large impact on the average number of vertices and quickly converges to 2, the minimum number of vertices.

Additionally, several point clouds with in total more than $3 \cdot 10^8$ points were evaluated visually. Representative results are shown in Fig. 8 to Fig. 10.

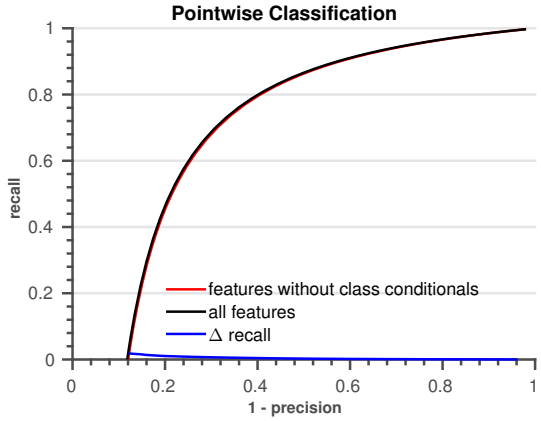


Fig. 14. Precision Recall for pointwise contour labeling.

6.2.4. *Failure cases*

Errors of our detector are mainly caused by two effects: *(i)* if there are independent contours with a small distance ($< s$) the framework will hallucinate erroneous connections between them; *(ii)* if the classifier encounters unusual point configurations dissimilar to those seen in the training data, it tends to produce false positives.

6.3. *Impact of contour features on multi-class classification*

In order to show the impact of contours on the performance of a multi class classifier we perform an experiment based on the data set of [16]. The training set consists of $\sim 2 \cdot 10^8$ points with 7 classes, while the test set consists of $\sim 3 \cdot 10^8$ points. We train two different classifiers. The first version contains all features designed for contour detection and multi class classification. The second version uses only the subset of features, which have not been explicitly designed for contour detection. This includes $\mathcal{O}$, R, Q and $\mathsf{m}_{\Uparrow}$. We again use grid search and 5-fold cross-validation to estimate the depth of the Random Forest and train with Gini-impurity as splitting criterion. On this data set the full feature set performs slightly better than the one without contour features; $\sim 0.5\%$ in $\overline{IoU}$ amd $\sim 0.3\%$ in Overall Accuracy. This shows that multi-class classification slightly benefits from features, which encode contour information.

6.4. *Impact of multi-class classification on contours*

Recall that we have labeled contours (to train the contour classifier) only for a relatively small set of data compared to the size of the dataset used for multi-class classification, because labeling contours in 3D space is tedious work. In order to still make use of the information contained in the entire multi-class training data, the feature vector of the contour detection was extended with class conditional probabilities of the multi-class classifier. The assumption is that (i) these additional probabilities from the full, much larger training data set might help the contour classifier to better generalize and that (ii) transitions between specific classes (e.g., roof and facade) might be a strong hint at contours.

To verify the benefit of adding probabilities from the multi-class classifier, we perform an experiment similar to Fig.7a using the data sets from section 6.2.1. It turns out that, in fact, contour detection benefits only slightly from the multi-class classifier. Small performance improvement by few percent points is recognizable in the high precision region in Fig. 14, whereas the gain is marginal in low precision regions. Multi-class scores from the full training dataset seem to add only few extra evidence in our case. This outcome shows that, although rather small, the training dataset seems to be sufficiently large to implicitly learn semantic classes for contour detection even without any direct feedback from the multi-class classifier. Although it is only a binary classification problem, the contour detector learns to neglect potential contours on vegetation, e.g. trees, while firing on contours at building edges.

7. Conclusion

We have described a novel approach for semantic classification and contour extraction, which is able to handle large data sets of unstructured point clouds very efficiently while achieving state-of-the-art performance. Because contours are ill-defined and not easy to explicitly describe with a set of ad-hoc rules, we prefer to also cast contour detection as a machine learning problem and let the classifier learn relevant contour evidence directly from the data. Contour detection benefits significantly from prior point-wise semantic classification. Our methods are fast enough for point clouds of realistic size. Moreover, they reach practically interesting accuracies: precision *and* recall of semantic segmentation are $> 85\%$ for most major object classes, which is well above the often quoted usability threshold of 80%. And contour detection reaches $\approx 85\%$ recall at 90% precision, which also is a healthy basis for subsequent interative modelling in a CAD system. We thus hope to soon see widespread deployment of automatic interpretation methods in point cloud processing software.

References

[1] Belongie, S., Malik, J., Puzicha, J., 2002. Shape matching and object recognition using shape contexts. IEEE transactions on pattern analysis and machine intelligence 24 (4), 509–522.

[2] Besag, J., 1986. On the statistical analysis of dirty pictures. Journal of the Royal Statistical Society. Series B (Methodological), 259–302.

[3] Brodu, N., Lague, D., 2012. 3d terrestrial lidar data classification of complex natural scenes using a multi-scale dimensionality criterion: Applications in geomorphology. ISPRS Journal of Photogrammetry and Remote Sensing 68, 121–134.

[4] Canny, J., 1986. A computational approach to edge detection. IEEE Transactions on pattern analysis and machine intelligence (6), 679–698.

[5] Charaniya, A. P., Manduchi, R., Lodha, S. K., 2004. Supervised parametric classification of aerial lidar

data. In: Computer Vision and Pattern Recognition Workshop, 2004. CVPRW'04. Conference on. IEEE, pp. 30–30.

[6] Chehata, N., Guo, L., Mallet, C., 2009. Airborne lidar feature selection for urban classification using random forests. International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences 38 (Part 3), W8.

[7] Choi, C., Trevor, A. J., Christensen, H. I., 2013. Rgb-d edge detection and edge-based registration. In: Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on. IEEE, pp. 1568–1575.

[8] Demantké, J., Mallet, C., David, N., Vallet, B., 2011. Dimensionality based scale selection in 3d lidar point clouds. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 38 (Part 5), W12.

[9] Dohan, D., Matejek, B., Funkhouser, T., 2015. Learning hierarchical semantic segmentations of lidar data. In: 3D Vision (3DV), 2015 International Conference on. IEEE, pp. 273–281.

[10] Elseberg, J., Borrmann, D., Nüchter, A., 2013. One billion points in the cloud–an octree for efficient processing of 3d laser scans. ISPRS Journal of Photogrammetry and Remote Sensing 76, 76–88.

[11] Eos Systems Inc., Photomodeler, 2017. http://www.photomodeler.com/index.html.

[12] Golovinskiy, A., Kim, V. G., Funkhouser, T., 2009. Shape-based recognition of 3d point clouds in urban environments. In: Computer Vision, 2009 IEEE 12th International Conference on. IEEE, pp. 2154–2161.

[13] Guo, Y., Kumar, N., Narayanan, M., Kimia, B., 2014. A multi-stage approach to curve extraction. In: European Conference on Computer Vision. pp. 663–678.

[14] Haala, N., Brenner, C., Anders, K.-H., 1998. 3d urban gis from laser altimeter and 2d map data. International Archives of Photogrammetry and Remote Sensing 32, 339–346.

[15] Hackel, T., Wegner, J. D., Schindler, K., 2016. Contour detection in unstructured 3d point clouds. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1610–1618.

[16] Hackel, T., Wegner, J. D., Schindler, K., 2016. Fast semantic segmentation of 3d point clouds with strongly varying density. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Prague, Czech Republic 3, 177–184.

[17] Hofer, M., Maurer, M., Bischof, H., 2015. Line3d: Efficient 3d scene abstraction for the built environment. In: German Conference on Pattern Recognition. Springer, pp. 237–248.

[18] Hug, C., Wehr, A., 1997. Detecting and identifying topographic objects in imaging laser altimeter data. International archives of photogrammetry and remote sensing 32 (3 SECT 4W2), 19–26.

[19] Johnson, A. E., Hebert, M., 1999. Using spin images for efficient object recognition in cluttered 3d scenes. IEEE Transactions on pattern analysis and machine intelligence 21 (5), 433–449.

[20] Kappes, J., Andres, B., Hamprecht, F., Schnorr, C., Nowozin, S., Batra, D., Kim, S., Kausler, B., Lellmann, J., Komodakis, N., et al., 2013. A comparative study of modern inference techniques for discrete energy minimization problems. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1328–1335.

[21] Konishi, S., Yuille, A. L., Coughlan, J. M., Zhu, S. C., 2003. Statistical edge detection: Learning and evaluating edge cues. IEEE Transactions on Pattern Analysis and Machine Intelligence 25 (1), 57–74.

[22] Lafarge, F., Mallet, C., 2012. Creating large-scale city models from 3d-point clouds: a robust approach with hybrid representation. International journal of computer vision 99 (1), 69–85.

[23] Lai, K., Bo, L., Fox, D., 2014. Unsupervised feature learning for 3d scene labeling. In: Robotics and Automation (ICRA), 2014 IEEE International Conference on. IEEE, pp. 3050–3057.

[24] Maas, H.-G., 1999. The potential of height texture measures for the segmentation of airborne laserscanner data. In: Fourth international airborne remote sensing conference and exhibition/21st Canadian symposium on remote sensing. Vol. 1. pp. 154–161.

[25] Mackaness, W. A., Mackechnie, G. A., 1999. Automating the detection and simplification of junctions in road networks. GeoInformatica 3 (2), 185–200.

[26] Maturana, D., Scherer, S., 2015. Voxnet: A 3d convolutional neural network for real-time object recognition. In: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on. IEEE, pp. 922–928.

[27] Monnier, F., Vallet, B., Soheilian, B., 2012. Trees detection from laser point clouds acquired in dense urban areas by a mobile mapping system. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 25, 245–250.

[28] Montoya-Zegarra, J. A., Wegner, J. D., Ladickỳ, L., Schindler, K., 2014. Mind the gap: modeling local and global context in (road) networks. In: German Conference on Pattern Recognition. Springer, pp. 212–223.

[29] Niemeyer, J., Wegner, J. D., Mallet, C., Rottensteiner, F., Soergel, U., 2011. Conditional random fields for urban scene classification with full waveform lidar data. In: Photogrammetric Image Analysis. Springer, pp. 233–244.

[30] Ok, A. O., Wegner, J. D., Heipke, C., Rottensteiner, F., Soergel, U., Toprak, V., 2012. Matching of straight line segments from aerial stereo images of urban areas. ISPRS Journal of Photogrammetry and Remote Sensing 74.

[31] Pauly, M., Keiser, R., Gross, M., 2003. Multi-scale feature extraction on point-sampled surfaces 22 (3), 281–289.

[32] Rottensteiner, F., Briese, C., 2002. A new method for building extraction in urban areas from high-resolution lidar data. International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences 34 (3/A), 295–301.

[33] Rusu, R. B., Marton, Z. C., Blodow, N., Holzbach, A., Beetz, M., 2009. Model-based and learned semantic object labeling in 3d point cloud maps of kitchen environments. In: Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on. IEEE, pp. 3601–3608.

[34] Schmid, C., Zisserman, A., 1997. Automatic line matching across views. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. IEEE, pp. 666–671.

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

[36] Shen, W., Wang, X., Wang, Y., Bai, X., Zhang, Z., 2015. Deepcontour: A deep convolutional feature learned by positive-sharing loss for contour detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3982–3991.

[37] Shi, W., Cheung, C., 2006. Performance evaluation of line simplification algorithms for vector generalization. The Cartographic Journal.

[38] Song, S., Xiao, J., 2016. Deep sliding shapes for amodal 3d object detection in rgb-d images. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 808–816.

[39] Steder, B., Rusu, R. B., Konolige, K., Burgard, W., 2010. Narf: 3d range image features for object recognition. In: Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS). Vol. 44.

[40] Steder, B., Rusu, R. B., Konolige, K., Burgard, W., 2011. Point feature extraction on 3D range scans taking into account object boundaries. In: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA).

[41] Taylor, C. J., Debevec, P. E., Malik, J., 1996. Reconstructing polyhedral models of architectural scenes from photographs. In: European Conference on Computer Vision. Springer, pp. 659–668.

[42] Tombari, F., Salti, S., Di Stefano, L., 2010. Unique signatures of histograms for local surface description. In: European Conference on Computer Vision. Springer, pp. 356–369.

[43] Trimble, Inpho Geo-Modeling Module, 2016. http://www.trimble.com/imaging/inpho.aspx?tab=Geo-Modeling_Module.

[44] Tupin, F., Maitre, H., Mangin, J.-F., Nicolas, J.-M., Pechersky, E., 1998. Detection of linear features in sar images: Application to road network extraction. IEEE transactions on geoscience and remote sensing 36 (2), 434–453.

[45] Türetken, E., Benmansour, F., Fua, P., 2012. Automated reconstruction of tree structures using path classifiers and mixed integer programming. In: Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on. IEEE, pp. 566–573.

[46] Türetken, E., González, G., Blum, C., Fua, P., 2011. Automated reconstruction of dendritic and axonal trees by global optimization with geometric priors. Neuroinformatics 9 (2-3), 279–302.

[47] Wegner, J. D., Montoya-Zegarra, J. A., Schindler, K., 2015. Road networks as collections of minimum cost paths. ISPRS Journal of Photogrammetry and Remote Sensing 108, 128–137.

[48] Weinmann, M., Jutzi, B., Mallet, C., 2013. Feature relevance assessment for the semantic interpretation of 3d point cloud data. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences 5, W2.

[49] Weinmann, M., Urban, S., Hinz, S., Jutzi, B., Mallet, C., 2015. Distinctive 2d and 3d features for automated large-scale scene analysis in urban areas. Computers & Graphics 49, 47–57.

[50] Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J., 2015. 3d shapenets: A deep representation for volumetric shapes, 1912–1920.

[51] Xiao, J., Furukawa, Y., 2014. Reconstructing the worlds museums. International journal of computer vision 110 (3), 243–258.

[52] Xie, S., Tu, Z., 2015. Holistically-nested edge detection. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1395–1403.

[53] Yao, W., Hinz, S., Stilla, U., 2011. Extraction and motion estimation of vehicles in single-pass airborne lidar data towards urban traffic analysis. ISPRS Journal of Photogrammetry and Remote Sensing 66 (3), 260–271.