POINT CLOUD SEGMENTATION USING RGB DRONE IMAGERY

Marc WuDunn, James Dunn, and Avideh Zakhor

Department of Electrical Engineering and Computer Sciences, University of California, Berkeley
{mwudunn, jhdunn, avz}@berkeley.edu

ABSTRACT

In recent years, the ubiquity of drones equipped with RGB cameras has made aerial 3D model generation significantly more cost effective than traditional aerial LiDAR-based methods. Most existing aerial 3D point cloud segmentation approaches use geometric methods and are tailored to 3D LiDAR data. In this paper, we propose a pipeline for semantic segmentation of 3D point clouds obtained via photogrammetry from aerial RGB camera images. Our basic approach is to directly apply deep learning segmentation methods to the very RGB images used to create the point cloud itself, followed by back-projecting the pixel class in segmented images onto the 3D points. This is a particularly attractive solution, since deep learning methods for image segmentation are more mature and advanced as compared to 3D point cloud segmentation. Furthermore, GPU engines for 2D image convolutions are likely to result in higher processing speeds than could be achieved using 3D point cloud data. We demonstrate our segmentation approach on two RGB Drone image datasets captured in Alameda, California, and compare its performance with manually labelled ground truth data. We use F1 and Jaccard similarity coefficient scores to show that our methodology outperforms existing methods such as PointNet++ and commercially available packages such as Pix4D.

***Index Terms*—** UAV Imaging, Drone, Occlusion, Photogrammetry, Semantic Segmentation, Point Cloud

1. INTRODUCTION

The semantic understanding of 3D point clouds provides valuable information to many disciplines that model outdoor environments. Urban planning, wireless communications engineering, and wildfire abatement are applications aided immensely by information on the structure and classification of environmental objects. The advent of increasingly accessible and accurate methods of surveying environments, combined with powerful computational methods for representing and processing surveyed data, have led to a large body of research into the task of semantic segmentation, producing a variety of approaches to this problem [1]. These approaches differ in complexity, accuracy, and data collection requirements. Two increasingly powerful tools that aid in this task are a) photogrammetry, which creates structure from 2D image collections [2], and b) deep networks, capable of accurately segmenting images that form the basis for 3D structure [3]. Combined, these tools provide a simple yet powerful method of generating and semantically segmenting 3D point clouds.

In this paper, we present a pipeline for the generation of semantically-segmented 3D point clouds from aerial RGB imagery of outdoor environments. By combining commercially-available photogrammetry software with open-source state-of-the-art deep net frameworks pre-trained on large datasets, our pipeline is simple, accurate, and accessible. The basic idea behind our approach is to apply deep learning methods to segment the very RGB images that are used to create the 3D point cloud, and to back-project the segmented pixels onto the points in the point cloud. In doing so, we develop geometric techniques to deal with occlusion issues. This pipeline is applied to aerial imagery collected by a commercial UAV of an urban outdoor scene, segmenting objects based on the classes 'building' and 'vegetation'. For applications such as fire prevention, the distance between vegetation and building is an important quantity to estimate.

A large body of research has investigated semantic segmentation of 3D point clouds. Early learning-based methods used principle component analysis to segment based on learned spatial distributions of points [4]. However, the associated eigendecomposition is computationally expensive. Recent state-of-the-art methods utilize deep learning applied to different forms of input data. [5, 6, 7, 8] demonstrated the application of a deep net directly to point clouds for the task of segmentation, while [9, 10] applied such geometric methods to aerial point clouds. However, purely geometric methods struggle with the fine details of outdoor imagery unless extremely dense LiDAR data is available. Collection of such data is often prohibitively expensive and time-consuming. Networks that jointly learn from geometric and color data have been proposed [11, 12, 13], as well as methods that flatten 3D surfaces to apply 2D CNNs [14, 15]. These approaches also assume availability of dense 3D data. Our work most closely resembles [16, 17], in which 2D segmentation methods are combined with photogrammetry reconstruction. However, [16, 17] mesh, texture-map, and create a synthetic view of the scene, which is then segmented and back-projected, thus introducing additional computationally complex steps.

This paper is organized as follows. Section 2 describes our methodology, including photogrammetry processing, 2D image segmentation, projection of segmented pixels to 3D points, and a geometric method for occlusion detection. Section 3 describes data collection and presents qualitative and quantitative results of our pipeline applied to aerial imagery. In Section 4, we summarize our contributions and discuss future improvements to this work.

2. METHODOLOGY

In this section, we provide an overview of our proposed pipeline. A block diagram for this pipeline is shown in Figure 1.

2.1. Photogrammetry

We use the Pix4D software suite [18] to generate a dense cloud and calculate the internal and external parameters for camera calibration

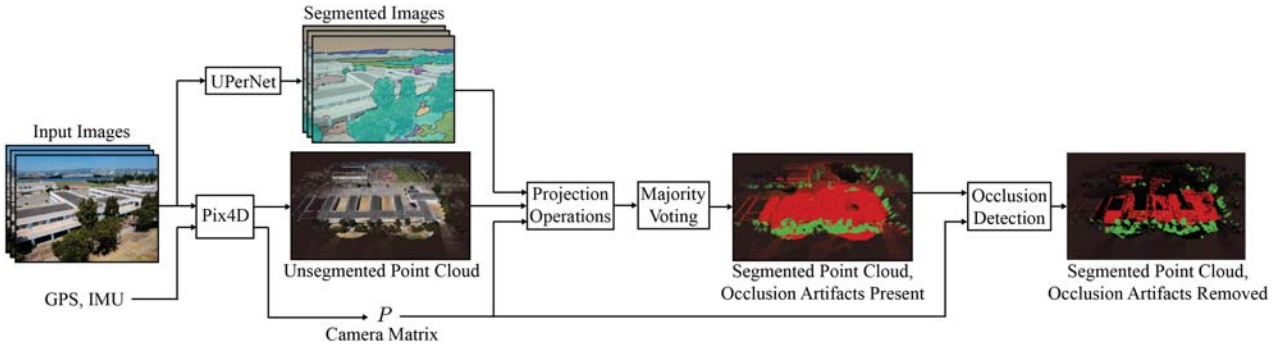


Fig. 1: Block diagram of proposed pipeline.

from our collection of input images. Pix4D additionally provides a set of camera pose matrices as output based on the internal and external camera parameters calculated for each input image. This matrix will be used to convert the 3D coordinates of a point in the generated dense cloud to the 2D coordinate of a pixel in an undistorted input image.

2.2. Segmentation of Undistorted RGB Images

We segment RGB images with UPerNet [19], a framework based on a 50-layer ResNet [20] convolutional neural net architecture. The UPerNet framework was chosen for its flexibility in semantically segmenting scenes based on object, material, or texture. We use existing models pre-trained by the UPerNet authors on the ADE20K dataset [20]. We pass the undistorted images produced by Pix4D to the segmentation network after downsampling the resolution by a factor of 4. The output of the segmentation network is a 2D array corresponding to the pixel dimensions of the input image, with each element containing a class label for the pixel. The ADE20K output classes fall into multiple categories, which we condense into 'building', 'vegetation', and 'other'. An output from the 2D image segmentation network is shown in Figure 2.

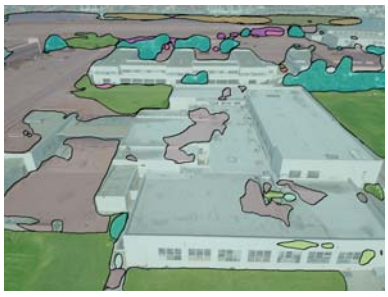


Fig. 2: 2D segmentation for a single image of Dataset 1.

2.3. Projection of 3D Points to Classified 2D Pixels

We next project each 3D coordinate in the generated point cloud to the 2D coordinate of all segmented images using the camera pose matrices provided by Pix4D, and retrieve the pixel classes corresponding to the 2D coordinates.

Given the set of segmented 2D images $\mathbf{I} = \{I_1, I_2, ..., I_m\}$, where each image I_m consists of a list of pixel coordinates $\{(u_1, v_1), (u_2, v_2), ..., (u_n, v_n)\}$, a list of corresponding pixel classes $\{c_1, c_2, ..., c_n\}$, and a corresponding camera pose matrix P_m; and, given a point cloud $C = \{(x_1, y_1, z_1), (x_2, y_2, z_2), ..., (x_o, y_o, z_o)\}$, we determine $(u_{m,o}, v_{m,o})$ for each point (x_o, y_o, z_o) in C and each image I_m via that image's camera matrix P_m:

$$(u_{m,o}, v_{m,o}) = P_m \cdot (x_o, y_o, z_o) \tag{1}$$

If $(u_{m,o}, v_{m,o})$ is within the pixel coordinate range of the image, it is a valid projection. We then index to find the corresponding pixel class $c_{m,o}$ for that coordinate, and add it to a stored dictionary of pixel classes per 3D point. We take the majority vote of all c_m for each point, and assign this value as the final point class. A fundamental limitation of this projection method is as follows: 3D points which are occluded from the perspective of a 2D image map to the same pixel on that image if the points lie along the same ray of projection. This leads to highly inaccurate point class assignment, as any two points along the ray of projection are assigned to the same class. For example, with aerial imagery, the large amount of ground occluded by a building could erroneously be classified as building.

2.4. Occlusion Detection

When projecting pixel classes in captured images onto the point cloud, multiple 3D points with different distances might be affiliated with a given pixel. The purpose of occlusion detection is to choose the 3D point that is closest to the camera image under consideration. This way, more distant 3D points corresponding to the same pixel will not be assigned a class from that pixel, since they

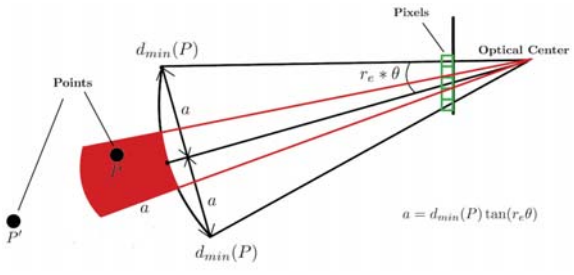


Fig. 3: Two points P and P' that project to a pixel (i_P, j_P). One point (P) is 'unoccluded' while the other point P' is 'occluded'.

are occluded by closer 3D points. An example of a point cloud segmentation in which occlusion is not taken into account is shown in Figure 1, where a large number of points are mis-classified.

A standard approach for detecting occlusions is to trace a ray through the scene and check for intersection within a 3D bounding shape. However, tracing rays can be quite costly from a computational point of view, especially for several hundred images. Rather, we iterate over each captured image, and in each iteration we project all 3D points in the point cloud onto the image and compare projective distances.

In what follows, we describe our approach for determining whether a given 3D point P is visible in a given image. Let d_P denote the projective distance of P with respect to the camera optical center, and (i_P, j_P) denote the pixel upon which P projects onto. We project each 3D point in the point cloud using the camera pose matrix, filtering out the points that lie outside of the camera image. We then round the resulting coordinates of the projection to the nearest integer (i, j) values; this corresponds to assigning each point projection to the nearest pixel in the camera image. Next, we compute the minimum projective distance $d_{min}(i, j)$ for all 3D points projected onto the pixel coordinates (i, j). This results in a matrix $\mathbf{A}_{min}$ of the minimum projective distances, whose size is the same as the size of the camera image.

The most obvious way to determine whether P is occluded is to directly compare d_P with the associated entry in A_{min} for pixel (i_P, j_P). However, the extremely high resolution of the camera imagery coupled with the sparsity of the point cloud can result in many pixels having only a small number of 3D points projected onto them. This would render the comparison between d_P and the associated minimum value in A_{min} meaningless, and could result in an inaccurate occlusion detection. To circumvent this problem, we instead compare d_P with the distances of a set of 3D points $S(i_P, j_P)$ that project to the 'vicinity' of (i_P, j_P), where vicinity is defined as a circle of radius r_e. We declare P as 'unoccluded' if its projective distance d_P is 'close' to the minimum projective distance of all 3D points in set $S(i_P, j_P)$. We denote the 3D point in the set $S(i_P, j_P)$ with such minimum distance as P_{min}, and its associated distance with $d_{min}(P)$. Roughly speaking, we need to determine whether d_P is close enough to $d_{min}(P)$, and if it is, we can declare P to be unoccluded. We perform this check for each 3D point in the point cloud whose projection lies in the bounds of the given camera image.

The radius r_e varies with the projective distance d_P of the point, since at a point further away from the camera image, the physical distance between two rays passing between neighboring pixels increases. Thus, we propose the following relationship for r_e:

$$r_e \propto \frac{1}{d_p} \tag{2}$$

Figure 3 provides a visualization of how to determine whether d_p is close enough to $d_{min}(P)$. In particular, the figure shows a side view of the camera pixels and the optical center. The red pixels depict the pixels within radius r_e of pixel (i_P, j_P). The black spherical cone (SC) in the figure depicts the extrusion of the circle of radius r_e in the image plane outward by the distance $d_{min}(P)$, with the apex of the SC corresponding to the optical center of the camera. Similarly, the red SC in the figure corresponds to the extrusion of the SC associated with pixel (i_P, j_P), again with the apex of the SC at the optical center of the camera. If we denote the angle between the rays that pass through successive pixels as θ, then the angles for the red and black SC are θ and $2r_e\theta$ respectively. The quantity a in the figure is the base radius of the black SC, and is given by $a = tan(r_e\theta)d_{min}(P)$. We construct the red zone in Figure 3 by extruding the red SC beyond the black SC by an amount a, and declare all points that fall into the red zone to be unoccluded. Intuitively, this makes sense since the lateral dimension of the red zone should be approximately proportional to the base radius of the black SC. In a way, as the distance $d_{min}(P)$ increases, we need to be more 'lenient' in declaring P as unoccluded. To summarize, we declare P to be unoccluded if:

$$d_p - d_{min}(P) > d_{min}(P)\tan(r_e\theta) \tag{3}$$

Though the qualitative results are acceptable for this method, the running time is fairly high. Specifically, on an Intel Core i5-8600K CPU and for a point cloud with 6 million points, it requires roughly 10 minutes per camera image. Though this method can be done in parallel, we opt to improve efficiency by implementing an approximation to the above method. Rather than computing r_e as a function of d_P for each point P, we discretize r_E into n effective radii spread uniformly throughout the range of the projective distances in $\mathbf{A}_{min}$. Next, we perform a block-reduce operation on $\mathbf{A}_{min}$ for each discretized effective radius to obtain n matrices $\mathbf{A}_1, \mathbf{A}_2, ..., \mathbf{A}_n$. For each 3D point P, we then simply index into the matrix k whose effective radius is closest to the projective distance of the point, and check whether Equation 3 holds for the projective distance at the corresponding coordinate in $\mathbf{A}_k$. While this may lead to some artifacts as a result of the spatial discretization, in practice we do not find this to be a significant problem. However, by using this approximation we achieve a significant speed-up, with running time on the order of seconds per image.

	Building			Vegetation		
	Ours	Pix4D	PointNet++	Ours	Pix4D	PointNet++
Precision	0.90	**0.98**	0.76	**0.72**	0.42	0.47
Recall	**0.90**	0.48	0.90	**0.91**	0.87	0.47
Jaccard	**0.82**	0.48	0.69	**0.64**	0.41	0.31
F1	**0.90**	0.64	0.82	**0.79**	0.58	0.47

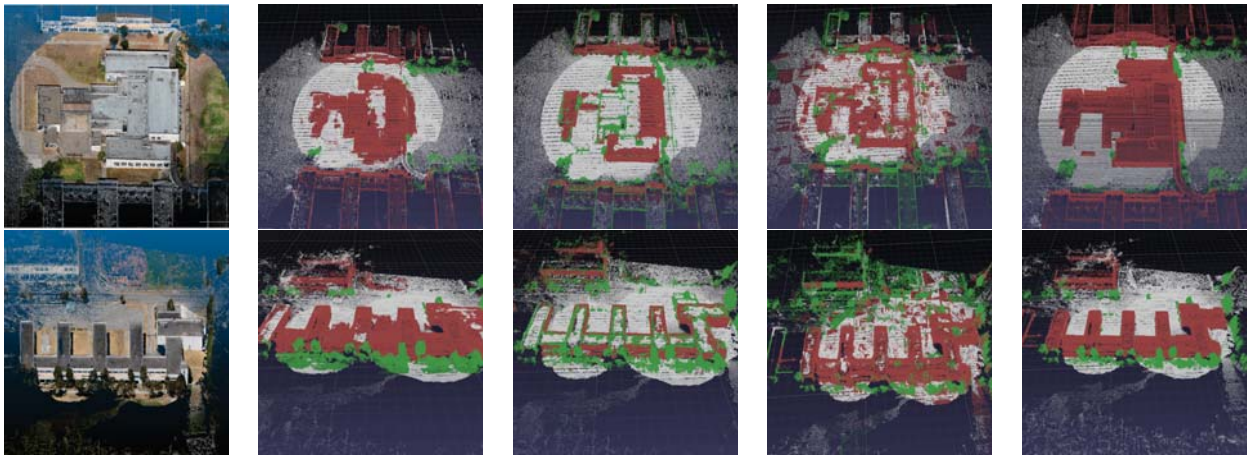
Table 1: Precision, Recall, F1 scores and Jaccard Similarity Coefficient scores for our methodology on Dataset 1.

	Building			Vegetation		
	Ours	Pix4D	PointNet++	Ours	Pix4D	PointNet++
Precision	0.90	**0.99**	0.75	**0.83**	0.25	0.22
Recall	**0.85**	0.62	0.73	0.55	**0.90**	0.63
Jaccard	**0.77**	0.58	0.60	**0.49**	0.24	0.19
F1	**0.87**	0.76	0.74	**0.66**	0.39	0.33

Table 2: Precision, Recall, F1 scores and Jaccard Similarity Coefficient scores for our methodology on Dataset 2.

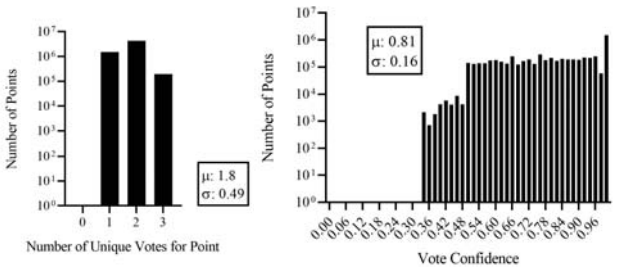
3. EXPERIMENTAL RESULTS

We collect 2 datasets, each consisting of several hundred images of a singular building in Alameda, California. Images are captured using the DJI Spark drone with a 12MP RGB camera, 25mm lens, and f/2.6 aperture. The buildings are captured in a circular pattern, with a 70% overlap between successive images. Images are then processed using Pix4D as described in Section 2.1 to generate a dense cloud with 6 million points. Pix4D is also used to generate a point



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

Fig. 4: (a) The example point clouds. Classification results from (b) our method. (c) Pix4D. (d) PointNet++. (e) The ground truth. Green: Vegetation, Red: Building, White: Other



(a) (b)

Fig. 5: (a) Histogram of number of unique votes per point. (b) Histogram of winning vote's confidence per point.

classification for each of these points , which we use as a comparison to the results of our method. For the occlusion detection, we find that using a single block-reduce size of (10,10) for a 12MP camera image works well for our purposes. We use the full image dataset to generate the dense point cloud, but only use 1 in 5 of the dataset images for segmentation, to improve runtime. Because we use a subset of photogrammetry images for segmentation, a percentage of 3D points may not be mapped to any 2D pixels in the segmented images. We find this to be $\sim$7000 points (0.12%) for Dataset 1 and $\sim$34100 points for Dataset 2 (0.31%). These are small enough to have a negligible impact on the results. An example point cloud is shown in Figure 4a, with the resulting predictions from our method, Pix4D, and PointNet++ in Figures 4b, 4c, and 4d respectively, and the ground truth in Figure 4e. Qualitatively, for both datasets Pix4D and PointNet++ leave a large number of vegetation points unclassified and misclassify a large number of building points as vegetation points.

To evaluate the performance of our method, we compute the F1 score and Jaccard Similarity Coefficient score for the two classes: 'building' and 'vegetation'. For comparison, we also compute the scores from the output of Pix4D's classifier and for PointNet++ trained on the Semantic3D dataset. Note that for these classifiers, we combine the LAS classifications of 'high', 'medium', and 'low vegetation' into a singular 'vegetation' class, as our work is not focused on distinguishing between the different types of vegetation. The results are shown in Tables 2 and 1 for Datasets 1 and 2, respectively. Comparing the accuracy scores, our method significantly outperforms both Pix4D and PointNet++ in detecting vegetation points for both Jaccard and F1 scores on both datasets, significantly outperforms both classifiers in detecting building points for Dataset 2, and modestly outperforms both classifiers in detecting building points for Dataset 2.

Additionally, we evaluate confidence metrics for points in our classified dataset. Figure 5a shows a distribution of the number of unique classes ('vegetation', 'building', 'other') that a point has been assigned by segmented pixels from separate images. This represents how likely the classified pixels among all images mapping to a given point agree with one another. The mean of this value is 1.8. We also show a histogram of 'vote confidence' in Figure 5b. We define the confidence of a point as the ratio of the number of majority classifications to the number of total classifications. This represents how 'strong' the majority class is relative to other assigned classes. This distribution has a mean of 0.81, showing that there is often a strong majority among the classes assigned to a point.

4. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a pipeline for the generation and semantic segmentation of 3D point clouds entirely from a collection of 2D imagery. By combining 2D RGB semantic segmentation and photogrammetry to accomplish this task, we avoid the need for LiDAR or other depth information from the environment. We address a fundamental limitation of this approach, occlusion, with a novel occlusion detection method. Results are favorable compared to state-of-the-art semantic segmentation built-in to an existing photogrammetry software suite. Future improvements to this work include a confidence score for point classifications, calculated with a Bayesian method on the list of classes for each point. This would likely improve results compared to the current majority vote of pixel classes corresponding to the point.

5. REFERENCES

[1] Yuxing Xie, Jiaojiao Tian, and Xiao Xiang Zhu, "A review of point cloud semantic segmentation," 2019.

[2] J. L. Schönberger and J. Frahm, "Structure-from-motion revisited," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 4104–4113.

[3] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 3431–3440.

[4] M. Carlberg, P. Gao, G. Chen, and A. Zakhor, "Classifying urban landscape in aerial lidar using 3d shape analysis," in *2009 16th IEEE International Conference on Image Processing (ICIP)*, Nov 2009, pp. 1701–1704.

[5] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," 2016.

[6] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," 2017.

[7] Loïc Landrieu and Martin Simonovsky, "Large-scale point cloud semantic segmentation with superpoint graphs," *CoRR*, vol. abs/1711.09869, 2017.

[8] Alexandre Boulch, "Convpoint: Continuous convolutions for point cloud processing," *Computers Graphics*, vol. 88, pp. 24 – 34, 2020.

[9] Lyne P. Tchapmi, Christopher B. Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese, "Segcloud: Semantic segmentation of 3d point clouds," 2017.

[10] E. Özdemir and F. Remondino, "Classification of aerial point clouds with deep learning," *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLII-2/W13, pp. 103–110, 2019.

[11] Vishakh Hegde and Reza Zadeh, "Fusionnet: 3d object classification using multiple data representations," 2016.

[12] Ji Hou, Angela Dai, and Matthias Nießner, "3d-sis: 3d semantic instance segmentation of rgb-d scans," 2018.

[13] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz, "SPLATNet: Sparse lattice networks for point cloud processing," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2530–2539.

[14] Hao Pan, Shilin Liu, Yang Liu, and Xin Tong, "Convolutional neural networks on 3d surfaces using parallel frames," *ArXiv*, vol. abs/1808.04952, 2018.

[15] Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou, "Tangent convolutions for dense prediction in 3d," 06 2018, pp. 3887–3896.

[16] A. Boulch, B. Le Saux, and N. Audebert, "Unstructured point cloud semantic labeling using deep segmentation networks," in *Proceedings of the Workshop on 3D Object Retrieval*, Goslar, DEU, 2017, 3Dor '17, p. 17–24, Eurographics Association.

[17] Alexandre Boulch, Joris Guerry, Bertrand Saux, and Nicolas Audebert, "Snapnet: 3d point cloud semantic labeling with 2d deep segmentation networks," *Computers Graphics*, vol. 71, 12 2017.

[18] Pix4D SA, "Pix4d," .

[19] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun, "Unified perceptual parsing for scene understanding," in *European Conference on Computer Vision*. Springer, 2018.

[20] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba, "Scene parsing through ade20k dataset," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.