Normal Estimation for Pointcloud using GPU based Sparse Tensor Voting

Ming Liu, François Pomerleau, Francis Colas, Roland Siegwart
Autonomous Systems Lab, ETH Zurich, Switzerland
[ming.liu, francois.pomerleau, francis.colas]@mavt.ethz.ch, rsiegwart@ethz.ch

***Abstract*—Normal estimation is the basis for most applications using pointcloud, such as segmentation. However, it is still a challenging problem regarding computational complexity and observation noise. In this paper, we propose a normal estimation method for pointcloud using results from tensor voting. Comparing with other approaches, we show it has smaller estimation error. Moreover, by varying the voting kernel size, we find it is a flexible approach for structure extraction as well. The results show that the proposed method is robust to noisy observation and missing data points as well. We use a GPU based implementation of Sparse Tensor Voting, which enables realtime calculation.**

I. Introduction

ESTIMATION of surface normal is one of the fundamental problems for pointcloud analysis. It is still a challenging problem for the case of real data from mobile robotics system, because of the following major reasons:

- *Unreliable observations*: the unreliability is multi-fold. In figure 1, we show a cropped part of pointcloud observed from an indoor semi-structured environment. Outliers, shadow points, non-uniform distributions, missing points etc. can be seen all over the place. These unreliable observations make data modeling subtle.
- *Application criteria*: since estimation results of pointcloud are used differently for applications, in order to fit various requirements or constraints, several compromises may be demanded.
- *Computational complexity*: we could see from figure 1 that the number of points are usually huge. Though subsampling is usually performed, the computational cost is still a bottleneck for most applications.

Respecting with these three major difficulties, we tackle the segmentation problem based on tensor voting framework [1] using a parallel implementation.

A. Contributions

We address the following two aspects in this paper:

- Surface normal estimation using sparse tensor voting. We compare two different algorithms in terms of precision and complexity. We show that the proposed algorithm is flexible in structure extraction.
- A GPU implementation of sparse tensor voting. We show significant improvement in performance comparing with CPU implementation.

This work was supported by the EU FP7 project NIFTi (contract # 247870)

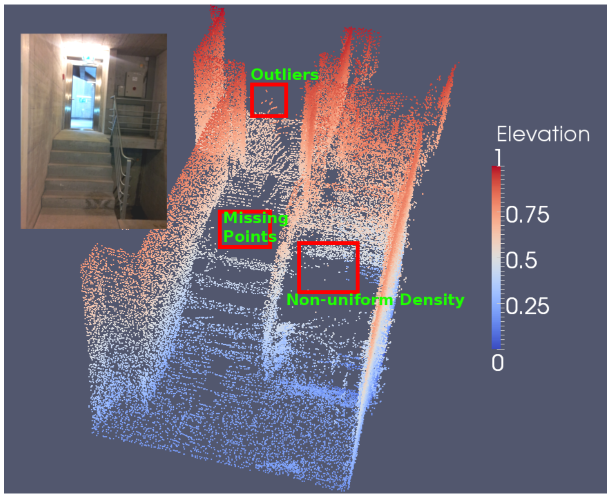


Fig. 1. Clip of a typical pointcloud and common observation noises.

B. Arrangement

The remainder of this paper is arranged as follows. We start with introducing related works. In section III, we compare tensor voting based normal estimation algorithms with a related common algorithm. The parallel implementation is presented in IV, followed by experiments on real dataset in section V. At last, we draw conclusions and introduce our vision for future work.

II. Related Work

A. Range Image and Pointcloud

Pointcloud is sometimes also referred as "range image". Usually these two terminologies are not distinguished. We understand range images are the 3d data which can be considered to be captured by a single scan, namely a grabbed *frame* from 3D range finders. On the other hand, point cloud is usually more complex. It can be combined with several registered scans. It means that the structure can not be directly reprojected to a 2d observation plane without losing information.

Several works regarding range image segmentation have been proposed. These works are based on different features, such as edge, projected as from 2d images [2], [3], [4], [5], [6], [7]. Then computer vision techniques are applied to process the data in 2D. The main drawback of these works is that they rely on almost-clean dense representations of the target models, which is not the case for most robotics applications. Moreover, the back-forth projection between 2d and 3d representation is time consuming. The ideal case is that all scans are with 0-noise and uniformly distributed points. For typical robotics applications, since the range finder is mounted on a moving platform and surveillant the work space from different viewpoints, these problems are however inevitable. Taking the neighbourhood of certain points into account is the key technique to deal with these issues. The conducted missing information can be possibly obtained by local parametrical modeling (e.g. surface model [8], [9]), non-parametrical regression [10], [11], [12], and other local embedding techniques [13]. Regarding propagation, they are usually broadcast over field [14], structured grid [15], [16], or over unstructured graph [17] etc.

B. Normal based Segmentation

Surface normal is a local consistent feature. Therefore it is widely used for pointcloud analysis. Regarding segmentation, one early work by Pulli et al [7] aims at segmenting range images into homogeneous regions, by decomposing x- and y-components of the normal vectors. It assumes perfect dense point clouds and the resulting algorithm only deal with segmentation in 2.5D. Normal estimation can also be based on local constrained least square modeling [18]. These normal estimation results often lead to clustering or segmentation of pointcloud, such as by an initial segmentation in normal space, then refines in distance space [19]. Teutsch et al presented a clustering algorithm for subset segmentation [20], which targets at segmentation of point clouds without plane-assumption. [21] introduced an incremental way to model different clusters by using both angular and distance constraints. For further references, [22] gave a recent report on different criteria for surface normal estimation of 3D range data.

C. Tensor voting

Tensor voting [1] is originated in computer vision. It has been extended to several applications related to segmentation [23], [24]. Through these works, tensor voting has shown its importance in reconstructing missing structures and local information registration [25], [26]. We consider it is one of the most important algorithms for structural analysis, because it is extraordinary performance in its tolerance to noise and missing data, its consistency for local information and intuitive extraction of evidence saliency etc. Nevertheless, the computational cost of Tensor voting is high. The original algorithm has complexity $O(N^2)$. We propose a parallel computation frame stimulated by [27]. Comparing with this existing work, our algorithm is more optimized considering the advanced calculation characteristics of CUDA, in order to improve calculation efficiency, e.g. using coalesced memory access, avoiding atomic operation and implementing online tensor split etc.

III. Sparse Tensor Voting and Normals

In this section we introduce the novel approach to estimate surface normals based on pointcloud directly, which enables a smooth and more precise estimation result comparing with parametrical algorithms. We starting with introducing a widely cited normal estimation algorithm, then describe the proposed approach. The discussion of the relation and difference between two methods is shown in the end.

A. Normal estimation

1) PCA based algorithm using k-NearestNeighbour (kNN): Principle component analysis (PCA) is used for PCL [28], which we take for main comparison. We name the algorithm that combining direction vectors to kNN and PCA analysis as kNN-PCA in this paper.

In spite of several variations, the kNN-PCA algorithm can be summarized as follows. It aims at finding the solution for normal vector $\overrightarrow{n}$ by using the eigenvector corresponding to the minimum eigenvalue of covariance matrix $\mathbb{C}$, expressed as:

$$\mathbb{C} = \frac{1}{k}\sum_{i=1}^{k}\xi_i(p_i-\bar{p})(p_i-\bar{p})^T \tag{1}$$

where k is the number of considered nearest neighbours; ξ_i is weighting factor for i-th neighbour; p_i's are kNN points and $\bar{p}$ is the mean of all k neighbours. ξ_i's are commonly equal. We could imagine that for highly noisy data and structural extraction, kNN-PCA is not feasible, because it relies much on local characteristics, which is fragile against noise or missing data.

2) Tensor Voting: Tensor Voting [1] is a computational framework used for structural extraction based on saliency of basic evidences. It originated in computer vision problems. King extended its application regarding pointcloud based terrain modeling and proposed an optimized stick voting field [26]. Following a generic pipeline described in [29], we construct sparse ball voting fields $eyes(3)$, and broadcast it through each neighboring point by a decay function:

$$k(d,\sigma) = e^{-\frac{d^2}{\sigma^2}} \tag{2}$$

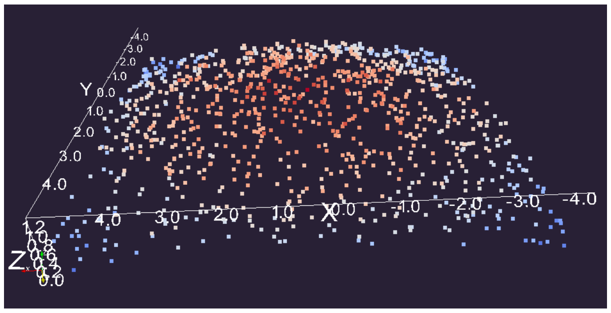
where d is the Euclidean distance between the voter and votee, and σ is a selected kernel size. For robotics applications, the kernel size can be chosen as the size of the navigation footprint. In this work, we omit dense voting process, which is often performed after sparse voting. Nevertheless it is a powerful tool for further structural inference such as topological segmentation.

The collected votes by each point is also a tensor containing the neighbouring structural information. The eigen decomposition of the resulting 3×3 tensor T can be formulated as:

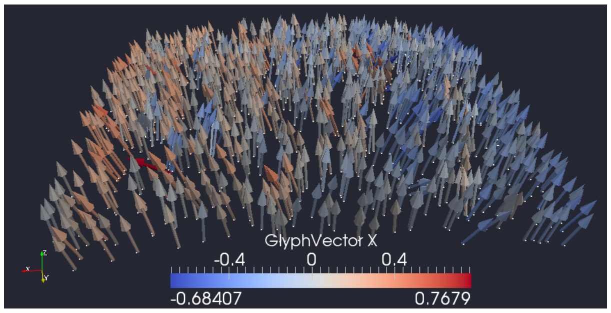
$$\begin{aligned} T &= \lambda_1\hat{e_1}\hat{e_1}^T + \lambda_2\hat{e_2}\hat{e_2}^T + \lambda_3\hat{e_3}\hat{e_3}^T \\ &= (\lambda_1-\lambda_2)\hat{e_1}\hat{e_1}^T + && \text{(stick component)} \\ &(\lambda_2-\lambda_3)(\hat{e_1}\hat{e_1}^T + \lambda_2\hat{e_2}\hat{e_2}^T) + && \text{(plate component)} \\ &\lambda_3(\hat{e_1}\hat{e_1}^T + \hat{e_2}\hat{e_2}^T + \hat{e_3}\hat{e_3}^T) && \text{(ball component)} \end{aligned} \tag{3}$$

where λ_i's are eigenvalues sorted in decreasing length sequence, $\hat{e_i}$'s are the corresponding eigenvectors. We could see that the stick saliency can be represented by $\lambda_1-\lambda_2$, which is highlighted by colormap shown in figure 2(c) and 4(c). The stick saliency for each point indicates how confident that a point can be considered as lying on a plane. The corresponding tensor, indicating the plane, is characterized by the normal direction of the local plane $\hat{e_1}$.

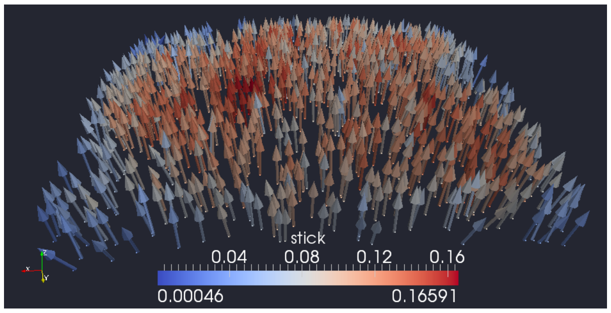
3) Surface normal estimation evaluation: The comparison of the aforementioned two algorithms is shown in figure 2. The tests are performed on a simulated surface given by figure



(a) Raw simulated noisy data



(b) Estimated surface normals by kNN-PCA



(c) Estimated surface normals by Sparse Tensor Voting

- Surface Equation:

$$\begin{cases} F = 1 - 0.02x^2 - 0.02y^2 - z + \omega = 0 \\ \omega \sim N(0, 0.1) \end{cases}$$

- Normal direction ground-truth:

$$\frac{\partial F}{\partial x, y, z} \Rightarrow \overrightarrow{\mathbf{n_{gt}}} = (-0.04x\,, -0.04y\,, -1)^T$$

(d) Setups

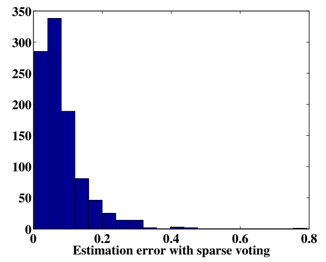
Fig. 2. Normal estimation

2(d). The colormap in figure 2(a) shows the elevation of points in z direction. The comparison between the direct normal estimation by kNN-PCA and sparse tensor voting is subtle, because the weighting factors are different for every points in the neighbourhood of a point, affected by kernel size σ. In order to have a fair comparison, we perform sparse voting first, then calculate the equivalent k by the mean number of voters per votee.

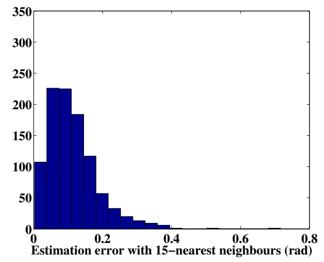
We can consider the kernel function as a probability density function (PDF), since the integral tends to a constant $\sigma\sqrt{\pi}$ at infinity. A numeric method is used to accumulate the function value starting from 0, approximating the integral of this PDF. We use third quartile [30] of the kernel function for statistical coverage, which leads to the sphere volume within 0.78σ. We consider the equivalent number of nearest neighbours by

$$n_{3rd-quartile} = \frac{\pi(0.78\sigma)^2}{\pi(3\sigma)^2} \cdot E(N_\sigma)$$

where S is the area of the surface, $E(N_\sigma)$ is the mean number of voters per votee, 217.3 points. We take $n_{3rd-quartile} = 15$ points. The histograms of the angular errors for both algorithms are shown in figure 3. They depict that the proposed



(a) Angular Error Histogram by Tensor Voting



(b) Angular Error Histogram PCA over kNN

Fig. 3. Error Analysis

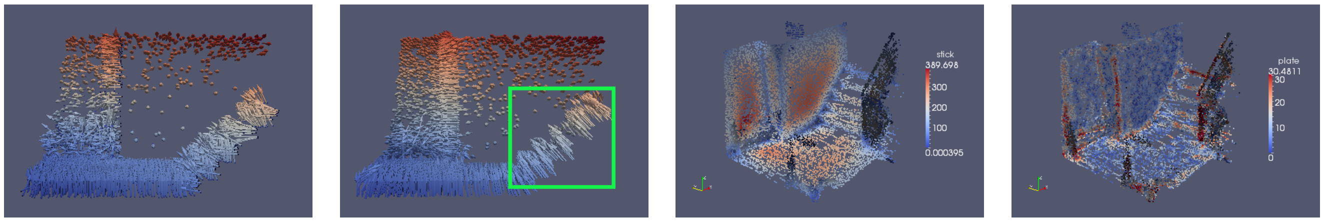
algorithm using sparse tensor voting is more reliable then kNN-PCA.

4) Complexity: The analysis of time complexity is shown in table I. It indicates that in general the complexity of Tensor Voting is higher than kNN-PCA. To alleviate the computation time, we use a GPU implementation described in section IV.

5) Discussion: Similar as kNN-PCA, Tensor Voting smoothes out noise by eigen decomposition as well. Therefore the two algorithms are inherently the same. The difference of performance comes from the definition of weighting factors of 1. They are introduced by the decay function (2). Moreover, by allowing this dynamic weighting of the related neighbours, the non-uniform distribution has less impact on the result. Another benefit is that by tuning kernel size σ, plane structures can be loosely defined. It is very useful to smoothing regions with dynamics. For example, if stairs are traversable for a rescue robot, they can also be selectively assumed to be a plane, so that connected stairs can be clustered as one topological node. This concept will greatly help the path planning using topological maps. The example from a real dataset by varying kernel size is shown in figure 4. We could see that by varying

TABLE I
TIME COMPLEXITY OF NORMAL CALCULATION

Method	Complexity	Parameters
kNN-PCA	$O(N \log N) + N \times (O(k \log N) \times O(c_3))$	k: number of considered neighbours;
kNN Tensor Voting	$O(N \log N) + N \times (O(k \log N) \times O(c_3))$	c_3: complexity of eigen decomposition of a 3×3 matrix;
Naive Tensor Voting (CPU)	$O(N^2) + N \times O(c_3)$	N: number of data points.
Naive Tensor Voting (GPU)	$O(h_{GPU}) + O(N) + N \times O(c_3)$	h_{GPU}: Overhead for GPU assignment and memory copy;



(a) Surface Normal estimation by tensor voting $\sigma = 0.1m$ (b) Surface Normal estimation by tensor voting $\sigma = 0.5m$ (c) Plane (stick) saliency by Sparse Voting $\sigma = 0.5m$ (d) Edge (plate) saliency by Sparse Voting $\sigma = 0.5m$

Fig. 4. Different results by varying kernel size σ and saliency extraction by sparse voting

the kernel size σ, different results can be obtained. With smaller σ, the estimation will be more constrained in local area; on the other hand, greater σ can get the normal in a larger area, e.g. the stairs in green rectangle can be considered as a complete plane. This feature will facilitate the execution of topological planning for instance. Additionally, we can also obtain the plane and edge saliency of the points as shown in figure 4(c) and 4(d). The conducted results will shown in our further report.

IV. SPARSE TENSOR VOTING AND TENSOR SPLIT ON GPU

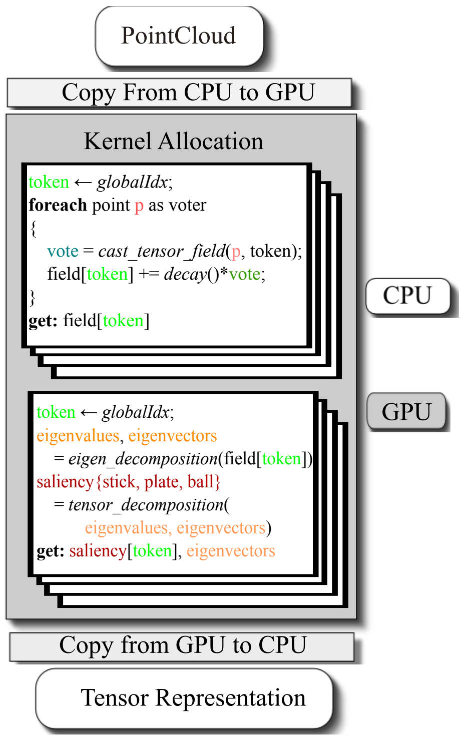
We summarize the parallel tensor voting framework in this section. At the same time, we address several technique details which directly related to the performance.

A. Structure Overview

Comparing with [27], we use iteration of voter's as base loop instead of votee based iteration. We considered the following two reasons:

1) The final output is gathered information by each votee. When use voter based iteration as base loop, the generation of output needs non-coalesced access of memory space. It is extremely inefficient for most GPU hardware comparing to coalesced access [31].
2) Because of the non-coalesced access, atomic operations are required, which is again a performance blocker for GPU computation. It is reported to be thousands of times slower than direct cycle [32]. The proposed voter based loop will alleviated this by direct shared memory access.

The sparse voting kernel is depicted as figure 5. There are two main blocks executed on GPU lying in the middle part of figure 5, which are designed for tensor field propagation and tensor split respectively. The tensor split algorithm is using orthogonality constraints proposed in [33].



```
token ← globalIdx;
foreach point p as voter
{
    vote = cast_tensor_field(p, token);
    field[token] += decay()*vote;
}
get: field[token]
```

```
token ← globalIdx;
eigenvalues, eigenvectors
    = eigen_decomposition(field[token])
saliency{stick, plate, ball}
    = tensor_decomposition(
        eigenvalues, eigenvectors)
get: saliency[token], eigenvectors
```

Fig. 5. Algorithm overview for GPU based sparse tensor voting

B. Implementation

In order to improve the performance, we used several conducted techniques. The comparison by applying these techniques (mean of 5 runs) are shown in table II, where we executed sparse tensor ball voting with the same kernel size for 14K points. We start with the result of an optimized CPU code, followed by the naive GPU implementation [27]. Then by applying different techniques, we reach to the proposed

Method	Exe. time (ms)	Speed-up	Accu. speed-up	Comment
CPU	728479	/	/	optimized code
naive GPU	11165.6	65.24x	65.24x	using Atomic operations
apply fast math	2287.4	4.88x	318.45x	e.g. __fsqrt_rn
using voter loop	1119.4	2.04x	650.72x	
using shared memory	1063.6	1.05x	684.85x	

TABLE II
PERFORMANCE GAIN BY OPTIMIZATION

algorithm. We could see that the proposed framework, which uses voter loop instead of voter loop greatly improves the calculation time. Further results will be shown in the next section.

V. Experiments and Discussion

A. Effect of Kernel size and GPU performance

The size of the voting kernel σ will greatly affect the computational complexity. A greater σ leads to quadratically more points to vote. By varying the size of σ, we show the execution time of sparse tensor voting and tensor split in figure 6. The dataset is the stair pointcloud shown in figure 1 which contains 14K points. Please note that for standard tensor voting, points within the range of 3 times σ are considered. The number of votes is illustrated in the lower figure of 6. It

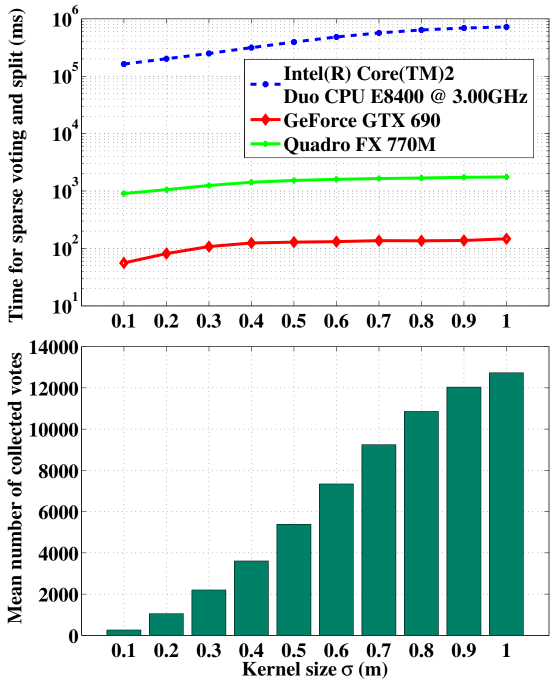
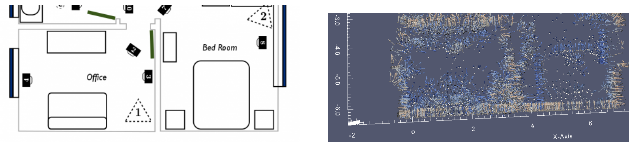


Fig. 6. The upper figure shows the comparison of execution time for different implementation. The lower figure show different number of votes by changing kernel size σ.

depicts the results on CPU and two different types of GPU's. We could see that the GPU implementation is superior to CPU in terms of calculation speed. Moreover, since the allocation of GPU computation is in unit of blocks, the increment of computational time is less various than CPU for large number of points.

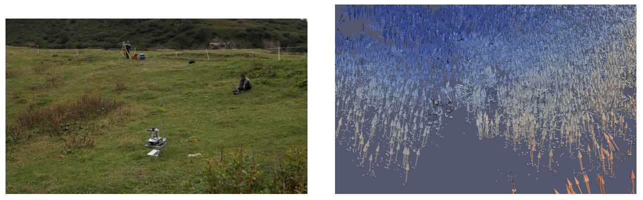
B. Normal estimation results on datasets

Several tests for normal estimation using the proposed algorithm are carried out. We show the qualitative results based on "Apartment" and "Mountain Plain" from [34] in figure 7 and 8 respectively.



(a) Overview of the area (b) Estimated Surface Normal

Fig. 7. Estimated normal for a typical apartment environment, 63.9K points, taking 645 ms, $\sigma = 0.2m$, considering mean 275 neighbours' votes.



(a) Overview of the area (b) Estimated Surface Normal

Fig. 8. Estimated normal for a typical outdoor field environment, 40.8K points, taking 337 ms, $\sigma = 1.0m$, considering mean 2113 neighbours' votes.

VI. Conclusion and Future work

In this work, we present a surface normal estimation algorithm using sparse tensor voting. It allows flexible structure extraction by tuning the size of voting kernel. Considering the complexity of tensor voting, we propose a GPU implementation. It shows significant improvement in computation time. We will carry out extended analysis such as pointcloud segmentation based on the existing results in our future work.

References

[1] G. Medioni, M. Lee, and C. Tang, *A computational framework for segmentation and grouping*. Elsevier Science, 2000, vol. 1.
[2] M. Pilu and R. Fisher, "Part segmentation from 2d edge images by the mdl criterion," *Image and vision computing*, vol. 15, no. 8, pp. 563–573, 1997.
[3] M. Wani and B. Batchelor, "Edge-region-based segmentation of range images," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 16, no. 3, pp. 314–319, 1994.

[4] D. Zhao and X. Zhang, “Range-data-based object surface segmentation via edges and critical points,” *Image Processing, IEEE Transactions on*, vol. 6, no. 6, pp. 826–830, 1997.

[5] Y. Alshawabkeh, N. Haala, and D. Fritsch, “Range image segmentation using the numerical description of the mean curvature values,” in *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences. ISPRS Congress*, 2008, p. 533.

[6] H. Iddamsetty, “Segmentation of range images for modeling of large outdoor scenes,” 2003.

[7] K. Pulli and M. Pietikäinen, “Range image segmentation based on decomposition of surface normals,” in *Proceedings of the Scandinavian conference on image analysis*, vol. 2, 1993, pp. 893–893.

[8] A. Leonardis, A. Gupta, and R. Bajcsy, “Segmentation of range images as the search for geometric parametric models,” *International Journal of Computer Vision*, vol. 14, no. 3, pp. 253–277, 1995.

[9] G. Yu, M. Grossberg, G. Wolberg, and I. Stamos, “Think globally, cluster locally: A unified framework for range segmentation,” in *International Symposium on 3D Data Processing, Visualization and Transmission*, 2008.

[10] L. Faivishevsky and J. Goldberger, “A nonparametric information theoretic clustering algorithm,” in *ICML*. Citeseer, 2010, pp. 351–358.

[11] T. Melzer, “Non-parametric segmentation of als point clouds using mean shift,” *Journal of Applied Geodesy*, vol. 1, no. 3, p. 159, 2007.

[12] J. Roca-Pardiñas, H. Lorenzo, P. Arias, and J. Armesto, “From laser point clouds to surfaces: Statistical nonparametric methods for three-dimensional reconstruction,” *Computer-Aided Design*, vol. 40, no. 5, pp. 646–652, 2008.

[13] P. Felzenszwalb and D. Huttenlocher, “Efficient graph-based image segmentation,” *International Journal of Computer Vision*, vol. 59, no. 2, pp. 167–181, 2004.

[14] M. Johnson-Roberson, J. Bohg, M. Björkman, and D. Kragic, “Attention based active 3d point cloud segmentation,” *IROS 2010*, 2010.

[15] H. Woo, E. Kang, S. Wang, and K. Lee, “A new segmentation method for point cloud data,” *International Journal of Machine Tools and Manufacture*, vol. 42, no. 2, pp. 167–178, 2002.

[16] B. Douillard, J. Underwood, N. Kuntz, V. Vlaskine, A. Quadros, P. Morton, and A. Frenkel, “On the segmentation of 3d lidar point clouds,” in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2798–2805.

[17] A. Golovinskiy and T. Funkhouser, “Min-cut based segmentation of point clouds,” in *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*. Ieee, 2009, pp. 39–46.

[18] E. Castillo and H. Zhao, “Point cloud segmentation via constrained nonlinear least squares surface normal estimates,” Technical Report CAM09-104, Computational and Applied Mathematics Department, University of California Los Angeles, Tech. Rep., 2009.

[19] D. Holz, S. Holzer, R. Rusu, and S. Behnke, “Real-time plane segmentation using rgb-d cameras$\star$,” in *Proc. of the 15th RoboCup International Symposium*, 2011.

[20] C. Teutsch, E. Trostmann, and D. Berndt, “A parallel point cloud clustering algorithm for subset segmentation and outlier detection,” in *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, vol. 8085, 2011, p. 8.

[21] K. Klasing, D. Wollherr, and M. Buss, “Realtime segmentation of range data using continuous nearest neighbors,” in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 2431–2436.

[22] K. Klasing, D. Althoff, D. Wollherr, and M. Buss, “Comparison of surface normal estimation methods for range sensing applications,” in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 3206–3211.

[23] Y. Dumortier, I. Herlin, and A. Ducrot, “4-d tensor voting motion segmentation for obstacle detection in autonomous guided vehicle,” in *Intelligent Vehicles Symposium, 2008 IEEE*. IEEE, 2008, pp. 379–384.

[24] J. Jia and C. Tang, “Inference of segmented color and texture description by tensor voting,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 6, pp. 771–786, 2004.

[25] H. Schuster, “Segmentation of lidar data using the tensor voting framework,” *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 35, pp. 1073–1078, 2004.

[26] B. King, “Range data analysis by free-space modeling and tensor voting,” Ph.D. dissertation, RENSSELAER POLYTECHNIC INSTITUTE, 2009.

[27] C. Min and G. Medioni, “Tensor voting accelerated by graphics processing units (gpu),” in *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, vol. 3. IEEE, 2006, pp. 1103–1106.

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

[29] P. Mordohai and G. Medioni, “Tensor voting: a perceptual organization approach to computer vision and machine learning,” *Synthesis Lectures on Image, Video, and Multimedia Processing*, vol. 2, no. 1, pp. 1–136, 2006.

[30] J. Glosup, “Statistical modelling with quantile functions,” *Technometrics*, vol. 43, no. 4, pp. 488–489, 2001.

[31] NVIDIA, Advanced CUDA Webinar, “Memory Optimizations.” [Online]. Available: {http://developer.download.nvidia.com/CUDA/training/NVIDIA_GPU_Computing_Webinars_CUDA_Memory_Optimization.pdf}

[32] Steven Robertson, “CUDA atomics: a practical analysis.” [Online]. Available: {http://strobe.cc/cuda_atomics/}

[33] A. Edelman, T. Arias, and S. Smith, “The geometry of algorithms with orthogonality constraints,” *Arxiv preprint physics/9806030*, 1998.

[34] Autonomous Systems Lab, ETH Zürich, “ASL Datasets Repository.” [Online]. Available: {http://http://projects.asl.ethz.ch/datasets/doku.php}