Normal estimation for pointcloud using GPU based sparse tensor voting
Ming Liu, François Pomerleau, Francis Colas, Roland Siegwart

To cite this version:
Ming Liu, François Pomerleau, Francis Colas, Roland Siegwart. Normal estimation for pointcloud using GPU based sparse tensor voting. IEEE International Conference on Robotics and Biomimetics (Robio), 2012, Guangzhou, China. 10.1109/ROBIO.2012.6490949. hal-01142736

HAL Id: hal-01142736
https://hal.archives-ouvertes.fr/hal-01142736
Submitted on 15 Apr 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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)

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 $\vec{n}$ by using the eigenvector corresponding to the minimum eigenvalue of covariance matrix $C$, expressed as:

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

where $k$ is the number of considered nearest neighbours; $\xi_i$ is weighting factor for $i$-th neighbour; $p_i$’s are kNN points and $\bar{p}$ is the mean of all $k$ neighbours. $\xi_i$’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:

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

where $d$ is the Euclidean distance between the voter and votee, and $\sigma$ 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 $3 \times 3$ tensor $T$ can be formulated as:

$$T = \lambda_1 \tilde{e}_1 \tilde{e}_1^T + \lambda_2 \tilde{e}_2 \tilde{e}_2^T + \lambda_3 \tilde{e}_3 \tilde{e}_3^T$$

$$= (\lambda_1 - \lambda_2) \tilde{e}_1 \tilde{e}_1^T + (\lambda_2 - \lambda_3) (\tilde{e}_1 \tilde{e}_1^T + \tilde{e}_2 \tilde{e}_2^T) + \lambda_3 (\tilde{e}_1 \tilde{e}_1^T + \tilde{e}_2 \tilde{e}_2^T + \tilde{e}_3 \tilde{e}_3^T)$$

(3)

where $\lambda_i$’s are eigenvalues sorted in decreasing length sequence, $\tilde{e}_i$’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 2(a). Raw simulated noisy data (b) Estimated surface normals by kNN-PCA (c) Estimated surface normals by Sparse Tensor Voting

Surface Equation:
\[
\begin{aligned}
F &= 1 - 0.02x^2 - 0.02y^2 - z + \omega = 0 \\
\omega &\sim N(0, 0.1)
\end{aligned}
\]

Normal direction ground-truth:
\[
\frac{\partial F}{\partial x, y, z} \Rightarrow \hat{n}_{gt} = (-0.04x, -0.04y, -1)^T
\]

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 $0.78\sigma$. 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 algorithm using sparse tensor voting is more reliable than 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 $\sigma$, 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

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 $\sigma$. 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.
# TABLE I

<table>
<thead>
<tr>
<th>Method</th>
<th>Complexity</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>kNN-PCA</td>
<td>$O(N \log N) + N \times (O(k \log N) \times O(c_1))$</td>
<td>$k$: number of considered neighbours;</td>
</tr>
<tr>
<td>kNN Tensor Voting</td>
<td>$O(N \log N) + N \times (O(k \log N) \times O(c_2))$</td>
<td>$c_1$: complexity of eigen decomposition of a $3 \times 3$ matrix; $c_2$: number of data points.</td>
</tr>
<tr>
<td>Naive Tensor Voting (CPU)</td>
<td>$O(N^2) + N \times O(c_3)$</td>
<td>$N$: number of data points.</td>
</tr>
<tr>
<td>Naive Tensor Voting (GPU)</td>
<td>$O(h_{GPU}) + O(N) + N \times O(c_3)$</td>
<td>$h_{GPU}$: Overhead for GPU assignment and memory copy;</td>
</tr>
</tbody>
</table>

Fig. 4. Different results by varying kernel size $\sigma$ and saliency extraction by sparse voting

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].

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...
TABLE II
PERFORMANCE GAIN BY OPTIMIZATION

<table>
<thead>
<tr>
<th>Method</th>
<th>Exec. time (ms)</th>
<th>Speed-up</th>
<th>Accu. speed-up</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>728479</td>
<td>/</td>
<td>/</td>
<td>optimized code</td>
</tr>
<tr>
<td>naive GPU</td>
<td>11165.6</td>
<td>65.24x</td>
<td>65.24x</td>
<td>using Atomic operations</td>
</tr>
<tr>
<td>apply fast math</td>
<td>2287.4</td>
<td>4.88x</td>
<td>318.45x</td>
<td>e.g. _sqrt_m</td>
</tr>
<tr>
<td>using voter loop</td>
<td>1119.4</td>
<td>2.04x</td>
<td>650.72x</td>
<td></td>
</tr>
<tr>
<td>using shared memory</td>
<td>1063.6</td>
<td>1.05x</td>
<td>684.85x</td>
<td></td>
</tr>
</tbody>
</table>

V. EXPERIMENTS AND DISCUSSION
A. Effect of Kernel size and GPU performance

The size of the voting kernel $\sigma$ will greatly affect the computational complexity. A greater $\sigma$ leads to quadratically more points to vote. By varying the size of $\sigma$, 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 $\sigma$ are considered. The number of votes is illustrated in the lower figure of 6. It 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.

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


