Real-time planar segmentation of depth images: from 3D edges to segmented planes.

Hani Javan Hemmat, Egor Bondarev, Peter H.N. de With
Eindhoven University of Technology, Department of Electrical Engineering, Eindhoven, The Netherlands.

Abstract. Real-time execution of processing algorithms for handling depth images in a 3D data framework is a major challenge. More specifically, considering depth images as point-clouds and performing planar segmentation requires heavy computation, because available planar-segmentation algorithms are mostly based on surface normals and/or curvatures, and consequently, do not provide real-time performance. Aiming at the reconstruction of indoor environments, the spaces mainly consist of planar surfaces, so that a possible 3D application would strongly benefit from a real-time algorithm. In this paper, we introduce a real-time planar-segmentation method for depth images avoiding any surface-normal calculation. First, we detect 3D edges in a depth image and generate line-segments between the identified edges. Second, we fuse all the points on each pair of intersecting line-segments into a plane candidate. Third and finally, we implement a validation phase to select planes from the candidates. Furthermore, various enhancements are applied to improve the segmentation quality. The GPU implementation of the proposed algorithm segments depth images into planes at the rate of 58 fps. Our pipeline-interleaving technique increases this rate up to 100 fps.

With this throughput rate improvement, the application benefit of our algorithm may be further exploited in terms of quality and enhancing the localization.

Keywords: Real-time planar segmentation, 3D-edge detection, depth images, 3D reconstruction, GPU implementation.

Address all correspondence to: Hani Javan Hemmat, Eindhoven University of Technology, Department of Electrical Engineering, Signal Processing Systems, Den Dolech 2, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.; Tel: +31 40 247 44 41; Fax: +31 40 247 45 67; E-mail: h.javan.hemmat@tue.nl

List of Figures

1 Introduction

Multiple studies in 3D reconstruction and perception have been recently triggered due to the appearance of low-cost depth-sensors capable of sensing volumetric environments in real-time. Currently, a wide range of applications in robotics, healthcare, and surveillance are profiting from depth images enabled by such sensors. Depth images along with their corresponding visual data combined into RGB-D frames, are exploited in 3D models of environments. With the emerging low-cost sensors like the Xtion and Kinect, multiple 3D reconstruction applications have been designed. However, reconstructing 3D indoor environments remains challenging due to cluttered spaces, extensive variability and the real-time constraints. Besides this, understanding the geometry of surrounding structures is becoming increasingly important for indoor applications. Since on the average, up to 95% of indoor structures consist of planar surfaces,[1] fast and accurate detection of such geometry features is essential for quality and functionality aspects of 3D applications, e.g. interaction, decreasing model size (decimation), enhancing localization, mapping and semantic 3D reconstruction.

Multiple planar segmentation algorithms have been proposed for point cloud datasets. Considering depth images as point clouds and performing planar segmentation requires heavy computation, because available planar segmentation algorithms are mostly based on surface normals and/or curvatures. More specifically, such algorithms utilize region growing,[2,3] 3D Hough Transform,[4] RANdom SAmple Consensus (RANSAC), and a combination of Hough and RANSAC for multiple resolutions[5] on the complete set of data,[6] or on pre-selected points using 3D feature descriptors.[7] RANSAC is utilized to detect planes with a reliable set of inliers.[8,9] It is applied iteratively to a depth image in order to detect multiple planes,[10] or applied region-wise to connect regions.[11] However, points belonging to the same segment do not always connect and therefore extensive post-processing is necessary to integrate under-segmented regions. In the 3D Hough-transform method, each plane is typically described in the object space along with a corresponding point in the parameter space. Hough results are combined with a voting mechanism of detection across the sequence of incoming frames to increase the detection robustness.[12] However, similar to RANSAC, post-processing steps are required for Hough-based methods to merge neighbouring segments of which the Hough parameters do not considerably deviate. Other popular segmentation techniques are based on surface normals and/or curvatures extraction,[13] linear fitting and Markov chain Monte Carlo.[14] These algorithms are also computationally expensive and challenging for real-time performance. In such algorithms, a real-time segmentation of planes is normally achieved by sacrificing the image quality.[15] Another segmentation technique is the region growing method, which is an efficient technique to extract planes from depth images. However, besides the already mentioned evaluation of surface normals and curvature estimates,[13] the chosen segmentation primitives and growth criterion play a critical role for the algorithm complexity. Introducing high-level features (line or curve blocks) as segmentation primitives instead of individual pixels, reduces the amount of processed data,[16] but still does not deliver real-time performance. Briefly, these techniques typically introduce computationally expensive algorithms or they suffer robustness. Although vanishing points and their roles in plane detection of 2D/3D color images have been indicated in the literature,[17,18] it is almost impossible to utilize them for depth images due to the following reasons. First of all, it is required to cover a relatively large area in an image to detect vanishing points, which is not the case for depth images targeted in this paper (e.g. urban scenes vs. the Kinect indoor depth-images). Furthermore, we need to detect lines to find vanishing points

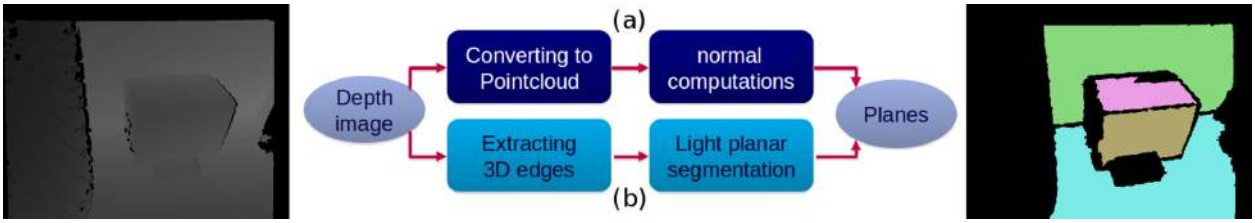

Fig 1 Two alternative approaches to deal with depth images in order to perform planar segmentation: (a) as point cloud, (b) as 2D images containing 3D information.

in a depth image. It is very challenging due to noisy data involved in depth images, especially on edges, that play a definitive role in line detection. Besides this, occlusions occurring in indoor scenes decrease the chance of having continuous lines in a scene. Therefore, probability of finding parallel lines in indoor images is very low.

As illustrated in Figure 1, a straightforward way to extract planes out of a depth image is to conventionally convert it to a point cloud and then apply an appropriate algorithm. In this paper, as an alternative to this approach, we propose a real-time planar segmentation algorithm, which directly deploys the depth images as 2D frames containing 3D information. More specifically, the proposed algorithm involves high-level region growing based on a geometrical proposition stating that each pair of intersecting lines lies on a plane. To this end, first, edge contours should be detected in order to extract surfaces bounded between the 3D edges.[19,20] Several algorithms for edge detection in intensity images have been developed and extensively discussed in the literature. However, edge-detection algorithms which are designed for intensity images, have a poor performance when applied to depth images due to several reasons. First, an edge in intensity images is defined as a significant change in gray-level value modeled as a jump edge, whereas in depth images, corner and curved edges should also be extracted.[21] Second, in depth images, spatial distribution of range data can be irregular, which requires operator adaptivity with respect to shape and size of the objects[22] in the scene. Finally, traditional edge-detection operators for intensity images such as Prewitt, Sobel and Canny are designed for normal visual images with normal contrast and therefore, perform poorly in highly noisy and blurred edges of depth images.[23] An edge-detection algorithm specific for depth images has been developed in an early study by detecting residues of morphological operations.[24] In a later work, straight lines in 3D space have been detected as depth edges by using a (2+2)–D Hough space.[25] Wani and Batchelor[26] have studied spatial curves to design edge masks for edge detection. This methodology has been further investigated based on the scan-line approximation technique.[27] Another line-segmentation technique[27] has been proposed for range images, which provides edge-strength measurements to categorize edge types.

We propose an algorithm where after detecting 3D edges of a depth image, it searches for line-segments between the opposite edges and then merges the detected line-segments into planes, thereby facilitating planar segmentation. The key is that the algorithm is capable of extracting 3D edges from depth images in real-time, particularly the components for corner and curved edges are enhanced in the execution speed. This high speed is obtained by utilizing concurrency and parallel structures to the highest extent. The interleaved GPU-based implementation of the proposed algorithm is capable of handling VGA-resolution depth images at a high frame rate up to 100 fps.

This paper is structured as follows. Section 2 describes the methods used in 3D edge detection and plane-extraction components of the proposed algorithm in detail. Experimental results are

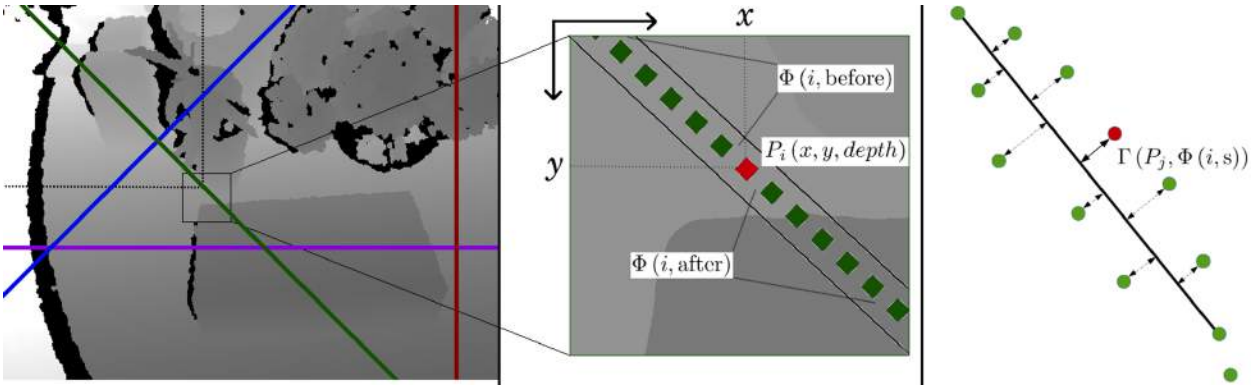

Fig 2 Illustration of the definitions: (a) a sample depth image with four randomly chosen 1-pixel-wide strings in various directions, (b) a magnified region of a sample string, and (c) the distance of a point to the line connecting two ends of its neighbourhood in 3D.

presented and discussed in Section 3 and Section 4, respectively. Finally, Section 5 concludes the paper.

2 Real-time planar segmentation

The proposed planar segmentation algorithm for depth images is based on two intrinsic properties of planes in a 3D environment:

Property 1 *Each planar surface inside a depth frame is bounded between its surrounding 3D edges.*

Property 2 *Based on a geometrical proposition, each pair of lines crossing each other in a joint 3D point, establish a 3D plane.*

As indicated by Property 1, we should search the areas in between 3D edges in order to find a planar surface in a depth image. According to Property 2, extracting the crossing line-segments in a depth image can lead us to identification of planar surfaces.

The proposed algorithm segments planar surfaces, based on the mentioned properties, in three principal steps. In the first step, all 3D edges in a depth image are extracted. In the second step, the algorithm searches for the line-segments lying between each pair of opposite edges of the supposed 3D planar region. In the final step, the identified line-segments are merged to planar areas. The merged line-segments are divided into various groups according to their connectivity. Each group of crossing line-segments is a planar-surface candidate. Furthermore, an extra validation step is performed to verify each candidate as a planar surface. In the following sub-sections, we focus on the methods exploited in each step in detail.

2.1 Step 1: Edge detection

In order to detect 3D edges in a depth image, the algorithm scans the image in 1-pixel strings in four directions (vertically, horizontally, left- and right-diagonally) as illustrated in Figure 2-a. Note that the string may be part of a curve in 3D, while in 2D still looks like a line. Edge detection is performed based on changes of depth values in a string. At each scan, four different types of edges are detected: *jump*, *extremum*, *corner*, and *curved*. The jump-edges result from occlusions or holes in depth images. The extremum edges include the local-minima or -maxima in a depth image. The

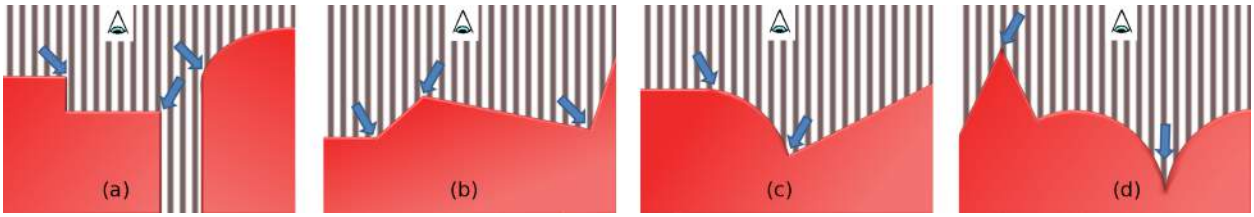

Fig 3 Various types of 3D edges in a depth image: (a) *jump-edges*, where there is a gap between two surfaces and/or in case of existence of a hole; (b) *corner-edges*, where two planar surfaces meet each other; (c) situations that planar surfaces join non-planar ones and *curved-edges* emerge; and finally (d) *extremum edges* due to local-maxima or -minima. In all the images, vertical lines indicate distance between surfaces and the depth-sensor camera-plane.

corner-edges emerge where two planes meet each other, and the curved-edges are resulting from intersection of actual planar and non-planar surfaces. Figure 3 depicts the mentioned types of 3D edges in the imaginary depth images. Although these various types of edges may have an overlap, we need to extract them all to cover all possible 3D edges in a depth image.

The following definitions formulate detection criterion for each edge type.

Definition 2.1 Point $P_i\,(x, y, depth)$ is the i^{th} 3D point on the current 1-pixel-wide string of each scan direction in a given depth image. Actually, it is the pixel located in column x and row y of the depth image as illustrated in Figure 2-b. In the remainder of this paper, we use the abbreviated notation of P_i instead of $P_i\,(x, y, depth)$.

Definition 2.2 Parameter Th_f determines a user-defined value indicating a threshold for the feature f, which can be any of the features used in the definitions (e.g. Th_{slope} is the basis for comparison of line slopes).

Definition 2.3 Parameter $\Phi_n\,(i, \mathrm{s})$ defines a set of n neighbours on a (subset of) string located on side $\mathrm{s} \in \{before, after\}$ of point P_i on the current 1-pixel-wide string for any of the mentioned scan directions as depicted in Figure 2-b. The set size n is a user-defined value. We occasionally use Φ to briefly represent $\Phi_n\,(i, \mathrm{s})$ in this paper, and similarly, Φ_{before} and Φ_{after} instead of $\Phi_n\,(i, before)$ and $\Phi_n\,(i, after)$, respectively.

Definition 2.4 Parameter $Sl\,(\mathrm{a}, \mathrm{b})$ is the slope of the line passing through points a and b in the 3D space.

Definition 2.5 Parameter $\overline{Sl}\,(\Phi)$ is the average slope of the lines passing through each pair of consecutive points in Φ.

Definition 2.6 Parameter $\Gamma\,(P_j, \Phi\,(i, \mathrm{s}))$, as shown in Figure 2-c, represents the signed distance of P_j to the line passing through P_{i-n} and P_{i+n} in the 3D space. The unit for this parameter is consistent with the unit utilized for depth images.

Jump edge. The 3D points P_i and P_j are located on the opposite sides of a *jump-edge* if they meet the following condition

$$|P_i.depth - P_j.depth| > Th_{jump}\ . \tag{1}$$

The term $P_i.depth$ represents only the depth value component of 3D pixel P_i, which represents the distance to the sensor plane. This equation describes two points located on a jump edge if their depth difference is more than a user-defined threshold as illustrated in Figure 3-a. The threshold can have fixed setting or alternatively be a function that returns a corresponding threshold value

according to the depth value of a point (distance-aware threshold[28,29]). The unit for the Th_{jump} is consistent with the unit utilized for depth images.

Corner edge. The *corner-edge*, as shown in Figure 3-b, is explained based on the following definition.

Definition 2.7 Binary flag $F_{equal-slope}(\Phi)$ determines whether all the lines passing through each consecutive pair of points in the neighbourhood Φ have the same slope or not.

Equation (2) formulates this definition according to a user-defined value of Th_{slope}. The binary "1"-value for the flag means that all the points of Φ are located on a line (and they may lie on a planar surface, accordingly). Similar to the jump-edge, the threshold here can be either a fixed number or a distance-aware function, alternatively. Equation (2) is specified by

$$F_{equal-slope}(\Phi) = \begin{cases} 1 & \forall\,(P_i, P_{i\pm1}) \text{ and } (P_j, P_{j\pm1}) \mid P_i, P_{i\pm1}, P_j, P_{j\pm1} \in \Phi \\ & |Sl(P_i, P_{i\pm1}) - Sl(P_j, P_{j\pm1})| < Th_{slope}\ ; \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The term $(P_i, P_{i\pm1})$ in Equation 2 indicates a pair of consecutive points on a 1-pixel-wide string in either (P_i, P_{i+1}) or (P_i, P_{i-1}) forms.

A point P_i is located on a *corner-edge* if the following conditions (3) and (4) are met:

$$\{F_{\textit{equal-slope}}\,(\Phi\,(i, \textit{before})) \quad \text{AND} \quad F_{\textit{equal-slope}}\,(\Phi\,(i, \textit{after}))\} = 1\ , \tag{3}$$

$$\left|\overline{Sl}\,(\Phi\,(i, \textit{before})) - \overline{Sl}\,(\Phi\,(i, \textit{after}))\right| > Th_{slope}\ . \tag{4}$$

Equation (3) checks if both segments, before and after the point P_i are line-segments, and Equation (4) ensures that each of the line-segments has a different slope (they are not located on the same line).

Curved edge. The *curved-edge*, depicted in Figure 3-c, is specified according to the following definition.

Definition 2.8 Binary flag $F_{straight}(\Phi)$ indicates whether all points of the neighbourhood Φ are located on a straight line or not.

This definition is formulated as Equation (5) based on a user-defined value of $Th_{straight}$. Similar to previous edges, the threshold here can either be a constant or a distance-aware setting. The "1"-value for the $F_{straight}(\Phi)$ means that points of Φ are located on a line (and may lie on a planar surface, accordingly). The difference between the Equation (2) and Equation (5) is that the former is more sensitive on boundaries between two planar surfaces, while the latter performs better in case a plane meets a non-planar surface. Equation (5) is specified by

$$F_{straight}(\Phi) = \begin{cases} 1 & \forall\, P_j \in \Phi: \quad |\Gamma(P_j, \Phi)| \le Th_{straight}\ ; \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Ponint P_i is located on a *curved-edge*, if condition (6) is met, given by

$$\{F_{straight}\,(\Phi_{before}) \quad \text{XOR} \quad F_{straight}\,(\Phi_{after})\ \} = \ 1\ . \tag{6}$$

The XOR operation ensures that *one and only one* side is a planar surface.

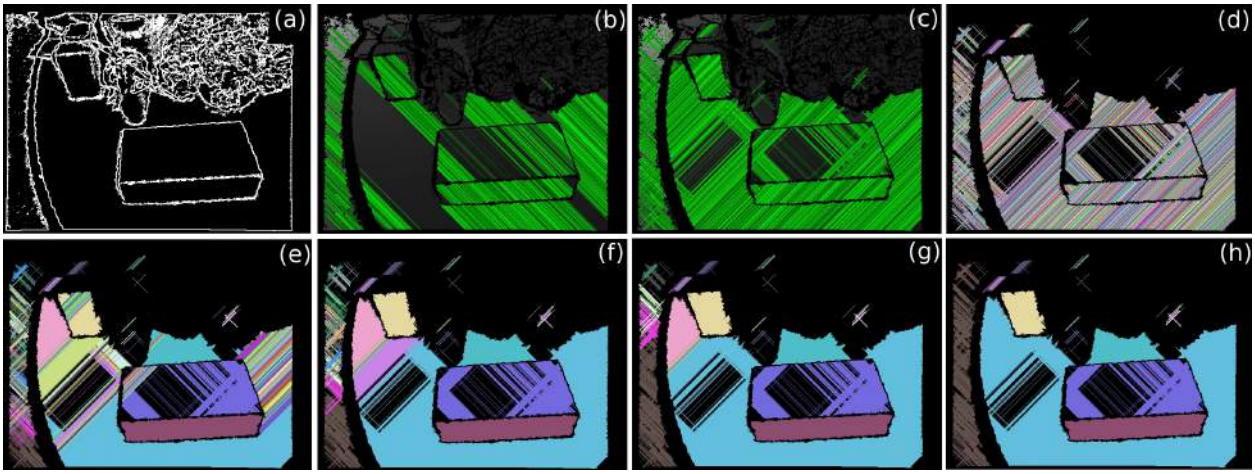

Fig 4 Planar segmentation of an example depth image from 3D edges to planar surfaces: (a) 3D edges detected on a given depth image; (b) left-diagonally detected line-segments between the 3D edges; (c) right-diagonally detected line-segments added (the vertical and horizontal lines are not depicted for the sake of clarity); (d) segmentation process which begins by individual labelling each line-segment; (e) to (g) merging process of intersecting line-segments sharing the same label; (h) final outcome of the planar segmentation process (Note that involving the vertical and horizontal strings can improve the result. Besides this, the proposed enhancement methods improving the final outcome have not been applied here).

Extremum edge. As shown in Figure 3-d, each point P_i, which is a local-minimum or -maximum in terms of its depth value is located on an *extremum-edge*.

In order to formulate this definition, we utilize two flags, as defined in Equation (7) and Equation (8), indicating if a point is a local-minimum or -maximum in its neighbourhood, respectively. Equation (7) is given by

$$F_{min}(P_i, \Phi) = \begin{cases} 1 & \forall\, P_j \in \Phi : \quad P_i.depth \leq P_j.depth \times En_{extremum}\ ; \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

And Equation (8) is specified by

$$F_{max}(P_i, \Phi) = \begin{cases} 1 & \forall\, P_j \in \Phi : \quad P_i.depth \geq P_j.depth \times En_{extremum}\ ; \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

In the previous equations, the multiplicative parameter $En_{extremum}$ is a coefficient to handle noise occurring in depth images and can be either a constant or a distance-aware function. Point P_i is located on an *extremum-edge* if the following condition is satisfied

$$\{F_{min}(P_i, \Phi) \quad \text{OR} \quad F_{max}(P_i, \Phi)\} \ = \ 1\ . \tag{9}$$

Evidently, only one of the flags can be true at the same time.

2.2 Line-based plane detection

Extracting 3D edges out of a depth image enables us to perform the second step of finding planes located in between the edges as illustrated in Figure 4. To perform the planar segmentation, we

first extract all the (1-pixel-wide) string-segments bounded in between the edges. This step commences with scanning all the strings in all four directions (vertically, horizontally, left- and right-diagonally). Then, for each of these string-segments, we evaluate whether it is a line-segment or not. Hence, after the test we maintain only line-segments in each direction from the string segments (Figure 4-b and -c). After finding all the line-segments (Figure 4-d), the algorithm attempts to merge the points on each pair of intersecting lines into a plane candidate, according to Property 2. Figure 4, in particular subfigures 4-e to 4-h illustrate the merging process. This step segments a depth image into its plane candidates. However, the resulting candidates need validation, which is discussed in the following section.

2.3 Plane validation

In order to improve the segmentation outcome, we perform several validation checks. First of all, we evaluate each plane candidate in terms of its curvature in a 3D space (for instance as a point cloud). Second, due to occlusions, a planar surface can be detected as various disconnected planar segments. Therefore, a merging process is needed to coalesce these apart segments into one actual plane. Finally, we evaluate the resulting segments in terms of their size. Based on different criteria, users may prefer to discard any planar segment, that has a relatively small number of points. Hence, we process the detected planes further in order to reject diminutive planes.

The three mentioned stages of validation are performed based on the following definitions.

Definition 2.9 Eigenvalues for plane Ψ in a 3D space are defined by λ_0, λ_1, and λ_2, where $\lambda_2 < \lambda_1 < \lambda_0$.

Definition 2.10 Parameter T_k, where $k \in \{0, 1\}$ is the curvature ratio for a plane and it is defined as the ratios of the Eigenvalues of the plane, which equal λ_2/λ_k for both $k = 0$ and $k = 1$.

The values T_0 and T_1 for a plane represent the ratios of the plane height to both its length and width, respectively. A planar surface typically has a quite small ratio values in terms of these aspects.[30] In other words, the smaller the values of T_0 and T_1 are for a plane, the less curvature the surface has (the flatter the surface is).

Definition 2.11 Parameter $Th_{curvature\_k}$, where $k \in \{0, 1\}$ represents the threshold value for T_k in order to validate a surface as a plane.

Definition 2.12 Vector $\overrightarrow{\mathbf{n}}(\Psi)$ is the normal vector of plane Ψ which equals to $\overrightarrow{\nu_2}$, which is the smallest Eigenvector of the plane.

Definition 2.13 Vector $\overrightarrow{\gamma}(\Psi_a, \Psi_b)$ is the vector connecting the center of mass of plane Ψ_a to its corresponding point of plane Ψ_b.

Curvature validation. Each plane candidate is converted to the corresponding point cloud in order to evaluate its curvature metrics (T_k). The Eigenvalues for each candidate are calculated by means of the Principal Component Analysis (PCA) method. A plane candidate is considered a valid plane if both its curvature values T_0 and T_1 are less than the user-defined thresholds $Th_{curvature\_0}$ and $Th_{curvature\_1}$, respectively. Equation (10) formulates the curvature-validation process as a condition, which should be satisfied by each valid plane and it is specified by

$$\{(T_0 \leq Th_{curvature\_0}) \text{ AND } (T_1 \leq Th_{curvature\_1})\} = 1 \,. \tag{10}$$

Merging separate segments. Due to holes and occlusions in a depth image, different segments of an actual plane may be detected as separate disconnected planes. In order to merge the separate segments, we evaluate the conditions (11) and (12) for each pair of plane segments Ψ_a and Ψ_b, given by

$$\overrightarrow{\mathbf{n}}(\Psi_a) \times \overrightarrow{\mathbf{n}}(\Psi_b) \approx \mathbf{0} \; ; \tag{11}$$

$$\overrightarrow{\mathbf{n}}(\Psi_a) \cdot \overrightarrow{\gamma}(\Psi_a, \Psi_b) \approx 0 \quad \text{AND} \quad \overrightarrow{\mathbf{n}}(\Psi_b) \cdot \overrightarrow{\gamma}(\Psi_a, \Psi_b) \approx 0 \; . \tag{12}$$

If the above conditions are met, the two separated segments are merged into a single plane. If the condition represented by Equation (11) becomes true, it means that the separated planes Ψ_a and Ψ_b are parallel to each other (their normal vectors are parallel). Likewise, the condition described by Equation (12) is satisfied when both separated planes Ψ_a and Ψ_b lie on the same planar surface. In other words, both normals are perpendicular to the vector, that connects the center of masses of the planes and this happens only if both segments lie on the same planar surface.

Size validation. A valid plane is required to contain a certain minimum number of points according to user requirements. Performing this test, the segmentation quality is improved, due to the removal of the small segments that cannot be merged into any other larger planes. The straightforward implementation of this condition means that each valid plane Ψ should satisfy

$$\Psi.\mathit{size} \geq Th_{\mathit{size}} \; , \tag{13}$$

where parameter $\Psi.\mathit{size}$ indicates the number of points located on plane Ψ.

In addition to all these post-processing enhancements, a pre-processing step is also performed to improve segmentation results even further. After extracting all the 3D edges of a depth image, we apply a morphological closing filter (i.e. a dilation followed by an erosion) in order to obtain more smooth and connected edges.

3 Evaluation results

We have applied the proposed algorithm to a collection of datasets and this section reports the evaluation results in detail. The dataset collection and the corresponding results are publicly available at `http://vca.ele.tue.nl/demos/fpsdi/fpsdi.html` . Figure 5 depicts several snapshots of the datasets and the corresponding results for both the 3D-edge detection and planar segmentation algorithms.

3.1 Datasets

In order to evaluate the proposed algorithm, we have prepared a rich collection of datasets to cover all various sorts of edges (jump, corner, curved, and extremum edges) in several indoor scene situations. The depth images of all the datasets have been captured via Kinect as a depth-sensor. (XBox 360 Kinect, version 1.0, VGA-resolution 480x640 pixels). Figure 5-a shows color images of the five samples chosen from the dataset collection. The color images at the top row are depicted instead of the depth images, since they give a better scene orientation.

Fig 5 Five examples of datasets containing various types of 3D edges and the corresponding outcomes: (a) color images of each scene, (b) extracted 3D edges, and (c) planar surfaces (Note that the merging and size-validation phases have not been incorporated in the visual results).

3.2 Real-time implementation

The proposed algorithm has been initially designed to maximally benefit from parallel computing. Each component of the algorithm has a minimum dependency on other components (inter-component independence). Besides this, the same approach is followed inside each component (intra-component independence). This intrinsic independence of the algorithm components has enabled us to implement it to be best suited for multi-core and many-core architectures. The OpenMP (OMP) has been utilized in order to implement the multi-threaded version running on multi-core CPUs. Besides this, we have used the Compute Unified Device Architecture (CUDA) for the many-core implementation of the proposed algorithm to be executed on GPU platforms. All the CPU-related results reported in this section, have been obtained utilizing a PC with a CPU of Intel® Xeon(R) W3550 @3.07GHz (4 cores) and 20 GB of RAM for both single- and multi-threaded versions. A CUDA-enabled 'GeForce GTX 480' VGA card with 480 cores and 1.5 GB of RAM has been used for the GPU-related experiments. In order to provide more precise timing-information for GPU implementation, results for both the kernel and wrapper (kernel as well as memory transfers) are reported.

3.3 Edge detection result

In this section, we present detailed results for the various modules of the edge-detector algorithm applying to depth images. To detect all mentioned types of edges, we perform the edge-detector modules independently for each edge type. These modules provide corresponding edge masks per edge type. The final mask is generated by combining all the edge masks together. Figure 6 shows three different scenes with various types of edges, for which the resulted edge-masks and the corresponding final masks are depicted. For the sake of better visual perception, only the color image of each scene is provided. We have found that the optimal results in terms of both quality and efficiency are obtained with the following settings. For jump edges, $Th_{jump} = 40$, for corner

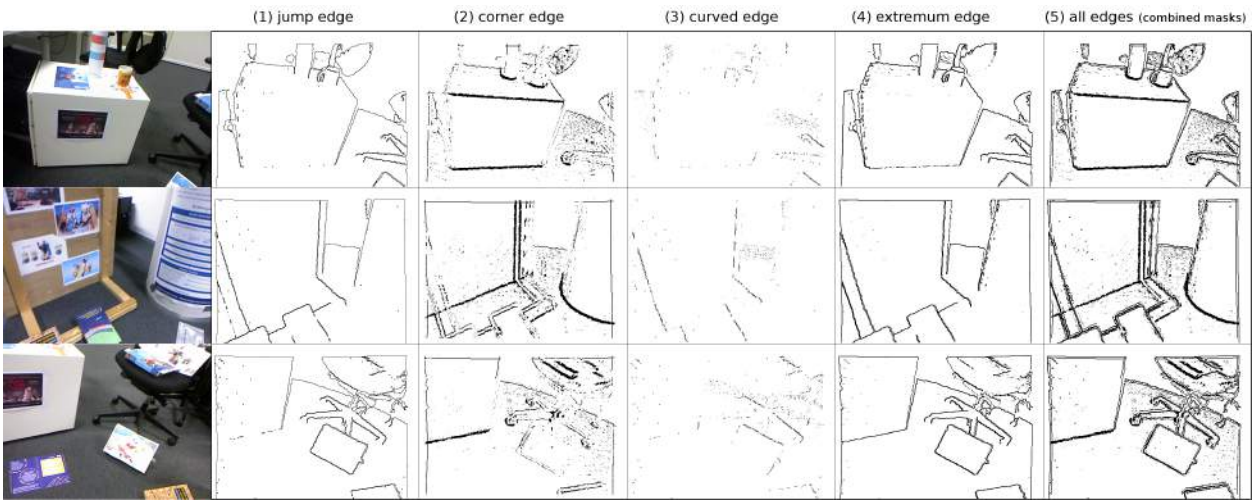

Fig 6 Three sample scenes and the corresponding edge-masks. Columns 1 to 4 show the independent edge-masks per edge type for each scene. The combined edges, as the final masks, are depicted in column 5.

Table 1 Execution time of the single-threaded, multi-threaded, and GPU-based implementations of various types of 3D-edge detectors applying on 20 datasets of depth images.

3D-edge detector	CPU (ms) single-threaded	CPU (ms) multi-threaded	speedup	GPU (ms) wrapper	speedup	GPU (ms) kernel	speedup
jump	3.27	1.24	2.64	1.07	3.06	0.08	42.1
extremum	6.31	2.41	2.62	1.11	5.68	0.15	40.8
corner	856	248	3.45	8.22	104	7.25	118
curved	250	95.5	2.62	2.74	91.3	1.89	132
all	**998**	**323**	**3.09**	**10.41**	**95.9**	**9.40**	**106**

edges, $Th_{slope} = 0.08$ and $n = 16$, for curved edges, $Th_{straight} = 4$ and $n = 4$, and finally for the extremum edges, $En_{extremum} = 0.98$ and $n = 2$.

Table 1 summarizes the average execution time for various implementations of the proposed algorithm. The results have been obtained according to the mentioned settings. Besides this, Figure 7 and Table 2 provide comparisons of the different results obtained by the 3D-edge detection algorithm (final mask) for different values of the number-of-neighbours (n in Φ_n) parameter. Among the various 3D-edge detectors, the corner- and curved-edge detectors are more sensitive to the number-of-neighbours parameter in terms of both quality and timing, compared to the jump- and extremum-edge detectors.

3.4 Planar segmentation result

This section presents evaluation of the planar segmentation algorithm applied to the dataset collection. Figure 5-c depicts corresponding segmented planes for the sample scenes. A summary of whole planar segmentation pipeline including edge-detection and plane-detection is shown by Table 3.

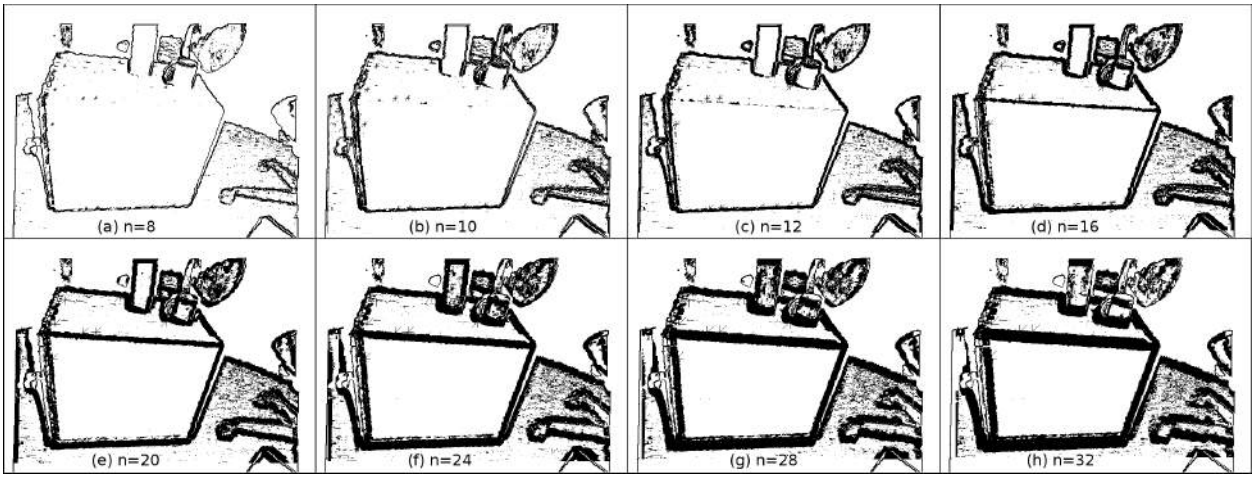

Fig 7 Effect of the setting value n (number of neighbours) on resulting 3D edges (final mask).

Table 2 Average execution time of the single-threaded, multi-threaded, and GPU-based implementations of 3D-edge detector (final mask) for different values assigned as number of neighbours.

Number of neighbours	CPU (ms) single-threaded	CPU (ms) multi-threaded	speedup	GPU (ms) wrapper	speedup	GPU (ms) kernel	speedup
32	2727	840	3.25	27.5	99	26.2	104
28	2424	835	2.90	26.9	90	25.0	97
24	2149	704	3.05	23.1	93	21.7	99
20	1831	592	3.09	20.0	91	18.8	98
16	1537	482	3.19	16.6	93	15.2	101
12	1208	410	2.95	12.8	95	11.6	105
10	969	344	2.82	10.9	89	9.7	100
8	802	295	2.72	9.0	89	7.9	102
		avg.	**3.00**	**avg.**	**92**	**avg.**	**101**

Table 3 Average execution time of planar segmentation pipeline for the single-threaded, multi-threaded, and GPU-based implementations.

Algorithm	CPU (ms) single-threaded	CPU (ms) multi-threaded	speedup	GPU (ms) wrapper	speedup	GPU (ms) kernel	speedup
edges	998	323	3.09	10.41	95.9	9.40	106
planes	103	68.7	1.50	8.80	11.72	7.84	13.2
full planar segmentation	1102	391	2.81	18.2	60	17.2	64

Table 4 Comparing the obtained 3D edges and planes to the corresponding groundtruth data: average similarity based on SSIM (Structural SIMilarity) and Data-to-Model Coverage (D2MC) metrics. The D2MC metric determines the amount of pixels in model image which have been covered by data image.

Similarity metric	3D edges	Planes
mSSIM (%)	97.3	97.8
D2MC (%)	99.6	98.7

4 Discussion

In this section, we discuss the results reported in the previous section in detail. First, we briefly assess the quality of the extracted 3D edges and planes. Then, we focus on detailed quantitative aspects of the proposed algorithms. And finally, we discuss the planar segmentation as a dual layer pipeline, consisting of the edge-detection and plane-extraction layers.

4.1 Qualitative assessment

As shown by Table 4, the resulting images of 3D-edge detection and planar-segmentation algorithms are 97.3% and 97.8% similar to the groundtruth, respectively, based on the mean Structural SIMilarity (mSSIM) metric. Besides this, 99.6% of groundtruth pixels are found and represented by the resulting images of 3D-edge detection algorithm and 98.7% for planar-segmentation algorithm according to the Data-to-Model Coverage (D2MC) metric.

As depicted in Figure 5, the proposed 3D edge-detector algorithm is capable of extracting the 3D edges in the presented scenes and segmenting the planar surfaces, accordingly. In the first row, five different scenes are depicted. The corresponding edges are shown in the second row. And finally, the third row contains the resulting planar surfaces for each scene. There are various planes detected regardless of both their size and their relative pose to the sensor.

As illustrated by Figure 6, there is a considerable amount of overlap between various kinds of 3D edges. Among them, the corner edge generates the most dominant part of the final edge mask. Besides this, the jump edge also plays a prominent role in generating the final edge mask. For all presented cases, the extremum edge has a huge overlap with the jump edge. This observation can be explained by the zero values for any hole or gap in depth images, which turns the points adjacent to holes into local maxima. And finally, the curved edges have the smallest share in the final edge mask, which is expected for indoor scenes, where most of the objects are planar surfaces.[1] In all cases, we have filtered the depth values by a threshold of 2.5 meters to reduce the effect of the Kinect-sensor noise prior to supplying it to the 3D-edge detector. This setting of this threshold is considerably influenced by the Kinect-sensor noise pattern.[28,29]

4.2 Quantitative assessment

3D-edge detection

According to Table 1, the amount of time needed to extract 3D edges is varying and depending on the type of edges. It varies from a few milliseconds for jump and extremum edges to hundreds of milliseconds for corner and curved edges. For the suggested settings, on the average, around 1 second is needed to extract all the 3D edges of a depth frame by a single-threaded implementation.

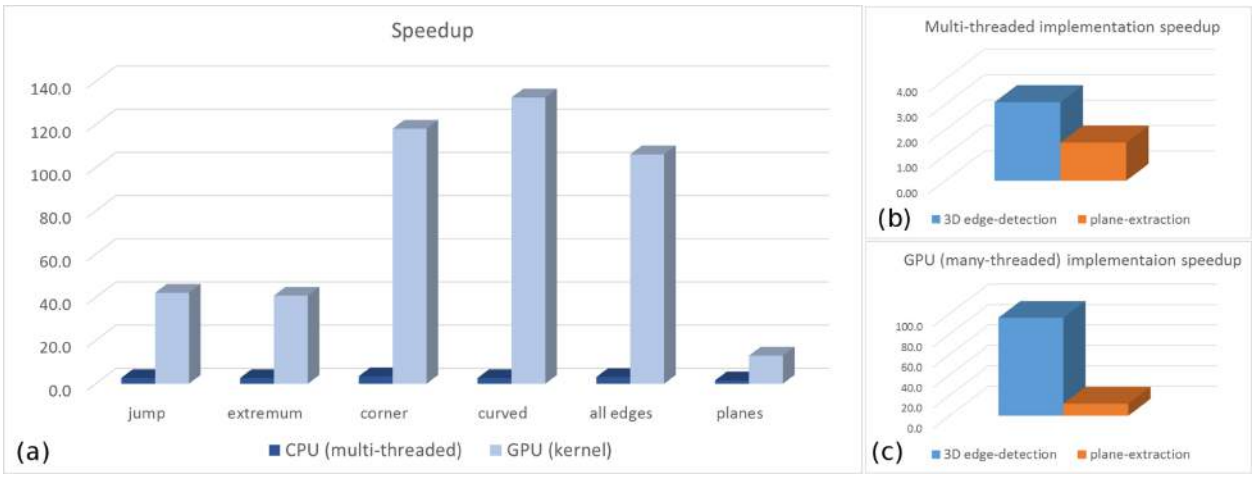

Fig 8 Speedup gained via multi-threaded and GPU-based implementations for edge-detector and plane-extractor algorithms: (a) multi-threaded versus GPU-based implementations, (b) and (c) comparing 3D edge detection to plane extraction for multi-threaded and GPU-based implementations, respectively.

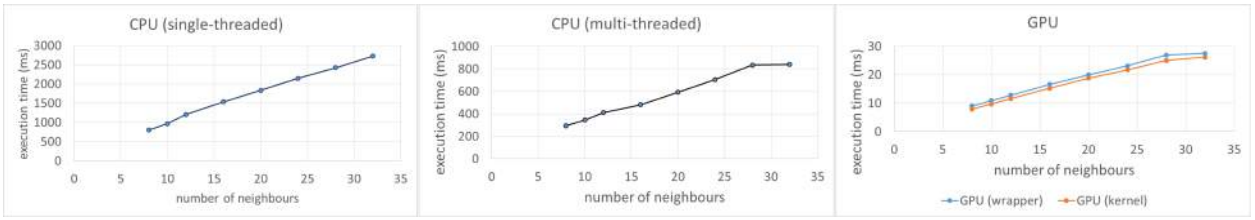

Fig 9 Effect of the setting-variable n on execution time of various implementations of the 3D edge detector algorithm.

A multi-threaded implementation improves the execution time to 323 ms, while exploiting a GPU-based many-threaded approach enables us to reach over 100 fps (9.40 ms per frame). Although the multi-threaded implementation establishes approximately a speedup factor of 3 for various edge types, the obtained speedup is varying per edge type for GPU-based implementation. As illustrated in Figure 8, the largest speedup factor emerges for the curved-edge case, namely 132 followed by 118 for the corner-edge case. Despite the implementation of both the curved and corner edges are very similar (according to their definitions), there is a noticeable difference between their obtained speedups. This difference is due to the proposed settings in which the value of n has been differently assigned per each edge type. Compared to the single-threaded implementation, the GPU version of the jump and extremum edges executes 42 and 41 times faster, respectively. The complete pipeline of the 3D edge detection is able to achieve a speedup factor of more than 100.

The proposed 3D edge detector output is generally sensitive to the setting of parameter n, which represents the number of neighbours. This sensitivity on the parameter n is mostly visible in terms of both functionality and obtained execution time. As shown by Table 2 and Figure 9, there is a linear relation between the value of n and the obtained execution time of the 3D edge detector. Considering the resulting edges depicted in Figure 7, there is an optimum value for n ($n = 16$) in terms of both quality and speed. For any value larger than this specific n, there is no edge added anymore and only the current edges become undesirably thick at the expense of a higher execution time. Moreover, the edge mask does not cover all the edges for the values smaller than the actual value of n. A reason for this observation originates from the resolution of the applied depth sensor

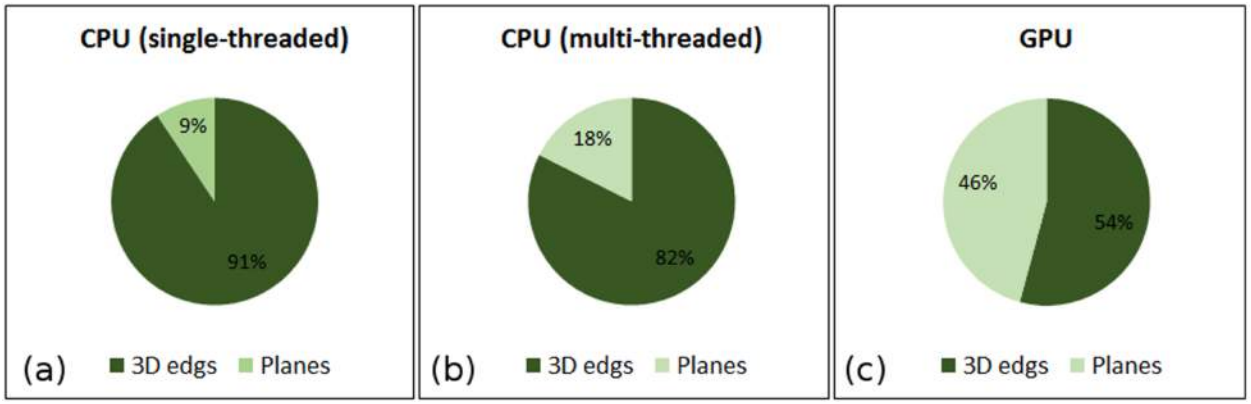

Fig 10 Percentage of execution time required by each layer of the complete planar segmentation pipeline for various implementations: (a) single-threaded, (b) multi-threaded and (c) GPU-based implementations.

(Kinect) to capture the dataset. The detector algorithm is expected to have a different optimum n for any other depth sensor.

Plane detection

Detecting planes based on 3D edges needs less computation compared to 3D edge detection itself. As presented in Table 3, on the average, 103 ms is needed to detect planes of a depth image based on the extracted 3D edges. By exploiting multi-threaded and many-threaded (GPU) implementations, speedup factors of 1.5 and 12 are achieved, respectively.

Planar segmentation as a pipeline

Planar segmentation of depth images is implemented as a dual-layer pipeline, consisting of 3D edge detection and plane extraction. As shown in Table 3, on the average, it takes 1102 ms to segment a depth image into its planes for a single-threaded implementation. A multi-threaded implementation decreases this execution time to 391 ms per frame (speedup factor of ≈ 3). And finally, this pipeline can produce planar segments in 17 ms per depth image by employing a GPU-based implementation (speedup factor of ≥ 60). As illustrated by Figure 10, the difference in execution time between the two layers of the pipeline is too high for both the single-threaded and multi-threaded CPU-based implementations to be interleaved by ratio of 1-to-10 and 1-to-4.5, respectively. However for a GPU-based implementation, this imbalance in the ratio is close enough to a 1-to-1 ratio and enables us to still execute the interleaved pipeline, on the average, resulting in a fps $>$100 (9.4 ms per interleaved cycle).

Due to the different amount of dependencies inside each layer of the pipeline, the level of concurrency differs between the two layers. As shown in Figure 8 (b) and (c), the 3D edge detection algorithm reaches a higher speedup factor in both multi-threaded and GPU-based implementations by namely 2 and 8 times, respectively, when compared to the plane-extraction algorithm.

As depicted in Figure 11, despite the parallel structure of the proposed algorithms, both the 3D edge detection and plane-extraction algorithms are content-dependent in terms of the execution time. Although the execution-time patterns are similar between all implementations of each algorithm, the variations in execution times are different for various datasets. For the 3D edge detection algorithm, the difference of execution time between some datasets varies up to 200% and 100% for

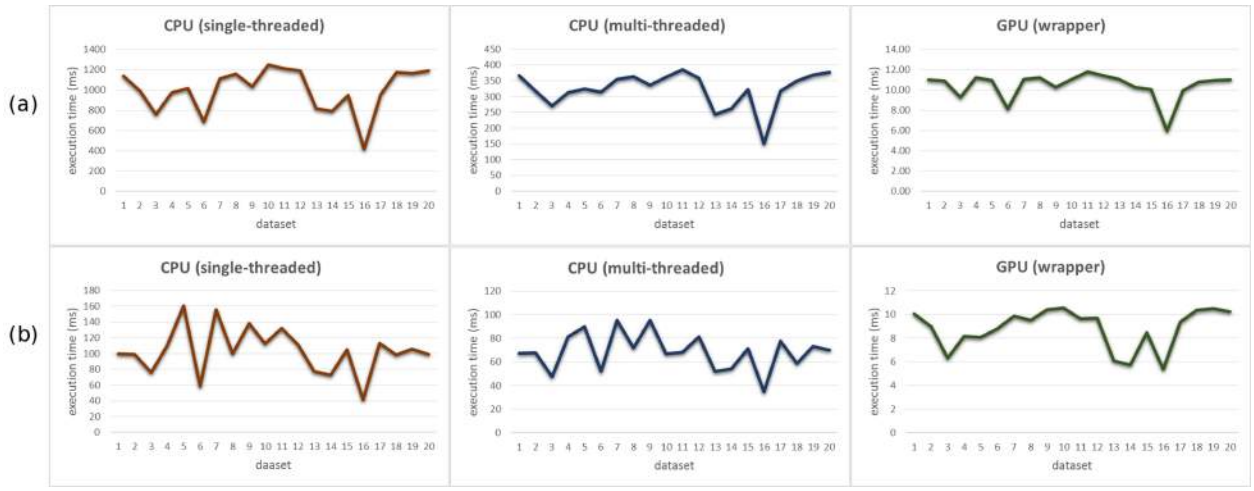

Fig 11 Execution time per each benchmark of the dataset collection for various implementations: (a) 3D edge detector, (b) plane extractor.

the CPU and GPU-based implementations, respectively. For the plane-extraction algorithm, this difference is up to 300% and 100% in some datasets for the CPU and GPU-based implementations, respectively.

5 Conclusion

In this paper, we have introduced a real-time planar segmentation algorithm, which enables plane detection in depth images avoiding any normal-estimation calculation. First, the proposed algorithm searches for 3D edges in a depth image and then finds the line-segments located in between of these 3D edges. Second, the algorithm merges all the points on each pair of the intersecting line-segments into a plane candidate. The developed 3D edge detection algorithm considers four different types of edges: jump, corner, curved and extremum edges. For each of those edges, we have defined the corresponding thresholds and a number-of-neighbours parameter. For the planar segmentation algorithm, we designed three quality-enhancing properties, i.e. curvature and size validation and merging of separate segments.

Because of the two previous algorithms, i.e. the proposed the 3D edge detection and the plane extraction, the implementation of the complete system leads to a dual-layer execution architecture. This enables a fast execution of both algorithms in parallel. By exploiting the GPU-based implementation, on the average, 3D edges are detected in 9.4 ms and planes are extracted in 7.8 ms. Therefore, the planar segmentation pipeline is capable of segmenting planes in a depth image with a rate of 58 fps. Utilizing pipeline-interleaving techniques for the proposed implementation further increases the rate up to 100 fps.

Acknowledgments

This research has been performed within the PANORAMA project, co-funded by grants from Belgium, Italy, France, the Netherlands, the United Kingdom, and the ENIAC Joint Undertaking.

References

1 R. B. Rusu, *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Computer Science department, Technische Universitaet Muenchen, Germany (2009).

2 T. Rabbani, F. A. van den Heuvel, and G. Vosselman, "Segmentation of point clouds using smoothness constraint," in *Proc. ISPRS Comssission V Symp. Image Engineering and Vision Metrology*, 248–253 (2006).

3 J. Xiao, J. Zhang, B. Adler, H. Zhang, and J. Zhang, "3D point cloud plane segmentation in both structured and unstructured environments.," *Robotics and Autonomous Systems* **61**(12), 1641–1652 (2013).

4 D. Borrmann, J. Elseberg, K. Lingemann, and A. Nüchter, "The 3D Hough Transform for plane detection in point clouds: A review and a new accumulator design," *3D Research* **2**(2), 1–13 (2011).

5 B. Oehler, J. Stueckler, J. Welle, D. Schulz, and S. Behnke, "Efficient multi-resolution plane segmentation of 3d point clouds," in *Proc. of Int. Conf. on Intelligent Robotics and Applications (ICIRA)*, S. Jeschke, H. Liu, and D. Schilberg, Eds., *Lecture Notes in Computer Science* **7102**, 145–156, Springer (2011).

6 R. Schnabel, R. Wahl, and R. Klein, "Efficient RANSAC for Point-Cloud Shape Detection," *Computer Graphics Forum* **26**(2), 214–226 (2007).

7 R. B. Rusu, N. Blodow, and M. Beetz, "Fast Point Feature Histograms (fpfh) for 3D Registration," in *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*, 3212–3217 (2009).

8 N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor Segmentation and Support Inference from RGBD Images," in *Proc. European Conf. Computer Vision (ECCV)*, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, Eds., *Lecture Notes in Computer Science* **7576**, 746–760, Springer (2012).

9 T.-k. Lee, S. Lim, S. Lee, S. An, S.-y. Oh, and S. Member, "Indoor Mapping Using Planes Extracted from Noisy RGB-D Sensors," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 1727–1733 (2012).

10 K.-M. Lee, P. Meer, and R.-H. Park, "Robust adaptive segmentation of range images," *IEEE Trans. Pattern Analysis and Machine Intelligence (TPAMI)* **20**(2), 200–5 (1998).

11 L. Silva, O. Bellon, and P. Gotardo, "A global-to-local approach for robust range image segmentation," in *Proc. IEEE Int. Conf. Image Processing (ICIP)*, 773–776 (2002).

12 R. Hulik, M. Spanel, Z. Materna, and P. Smrz, "Continuous plane detection in point-cloud data based on 3d hough transform," *Journal of Visual Communication and Image Representation* **25**(1), 86–97 (2013).

13 D. Holz and S. Behnke, "Fast Range Image Segmentation and Smoothing Using Approximate Surface Reconstruction and Region Growing," in *Proc. Int. Conf. Intelligent Autonomous Systems*, 61–73 (2012).

14 C. Erdogan, M. Paluri, and F. Dellaert, "Planar Segmentation of RGBD Images Using Fast Linear Fitting and Markov Chain Monte Carlo.," in *Proc. of Conf. on Computer and Robot Vision (CRV)*, 32–39, IEEE (2012).

15 D. Holz, S. Holzer, R. B. Rusu, and S. Behnke, "Real-Time Plane Segmentation Using RGB-D Cameras," in *Proc. of RoboCup 2011: Robot Soccer World Cup XV*, **7416**, 306–317, Springer Berlin Heidelberg (2012).

16 X. Jiang, H. Bunke, and U. Meier, "High-level feature based range image segmentation," *Image and Vision Computing* **18**(10), 817–22 (2000).

17 F. A. Andal, G. Taubin, and S. Goldenstein, "Vanishing point detection by segment clustering on the projective space.," in *ECCV Workshops (1)*, K. N. Kutulakos, Ed., *Lecture Notes in Computer Science* **6554**, 324–337, Springer (2010).

18 J. C. Bazin, P. Y. Laffont, I. Kweon, C. Demonceaux, and P. Vasseur, "An original approach for automatic plane extraction by omnidirectional vision," *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings* , 752–758 (2010).

19 A. Harati, S. Gächter, and R. Y. Siegwart, "Fast range image segmentation for indoor 3D-SLAM," in *Proc. IFAC Symp. on Intelligent Autonomous Vehicles*, 475–480 (2007).

20 R. Hulik, V. Beran, M. Spanel, P. Krsek, and P. Smrz, "Fast and accurate plane segmentation in depth maps for indoor scenes," *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems* , 1665–70 (2012).

21 S. A. Coleman, S. Suganthan, and B. W. Scotney, "Gradient operators for feature extraction and characterisation in range images," *Pattern Recognition Letters* **31**(9), 1028–40 (2010).

22 S. Suganthan, S. A. Coleman, and B. W. Scotney, "Using Dihedral Angles for Edge Extraction in Range Data," *Journal of Mathematical Imaging and Vision* **38**(2), 108–18 (2010).

23 A. Hoover, G. Jean-Baptiste, X. Jiang, P. J. Flynn, H. Bunke, D. B. Goldgof, K. Bowyer, D. W. Eggert, A. Fitzgibbon, and R. B. Fisher, "An experimental comparison of range image segmentation algorithms," *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)* **18**(7), 673–89 (1996).

24 R. Krishnapuram and S. Gupta, "Edge detection in range images through morphological residue analysis," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 630–2 (1992).

25 P. Bhattacharya, H. Liu, A. Rosenfeld, and S. Thompson, "Hough-transform detection of lines in 3-D space," *Pattern Recognition Letters* **21**(9), 843–9 (2000).

26 M. A. Wani and B. G. Batchelor, "Edge-region-based segmentation of range images," *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)* **16**(3), 314–319 (1994).

27 X. Jiang and H. Bunke, "Edge Detection in Range Images Based on Scan Line Approximation," *Computer Vision and Image Understanding* **73**(2), 183–99 (1999).

28 H. Javan Hemmat, E. Bondarev, G. Dubbelman, and P. de With, "Improved ICP-based Pose Estimation by Distance-aware 3D Mapping," in *Proc. Int. Conf. on Computer Vision Theory and Applications (VISAPP)*, 360–367 (2014).

29 H. Javan Hemmat, E. Bondarev, and P. de With, "Exploring Distance-Aware Weighting Strategies for Accurate Reconstruction of Voxel-Based 3D Synthetic Models," in *Proc. Int. Conf. on MultiMedia Modeling (MMM)*, 412–423 (2014).

30 C. M. Brown, "Principal Axes and Best-Fit Planes with Applications," *Technical report TR7, University of Rochester, Computer Science Department* (1976).

Hani Javan Hemmat received his B.Sc. degree in Computer Engineering from Amirkabir University of Technology (Tehran Polytechnic) in 2002. In 2006, he received the M.Sc. degree in Computer Architecture from the Computer Engineering department of the Sharif University of

Technology (Tehran, Iran). His research interests include 3D reconstruction, robotics, parallel processing, hardware-software co-design, and design of hardware/software architectures for real-time implementation. Since 2012, He researches on multi-modal fusion and 3D reconstruction at Technical University of Eindhoven.

Egor Bondarev Egor Bondarev received his MSc degree in robotics and informatics from the State Polytechnic University, Belarus Republic, in 1997. In 2009 he has obtained his PhD degree in Computer Science at Eindhoven University of Technology (TU/e), The Netherlands in the domain of real-time systems with a focus on performance predictions of component-based systems on multiprocessor architectures. Currently, he is an Assistant Professor at the Video Coding Architectures group, TU/e, focusing on such research areas as multi-modal sensor fusion, photorealistic 3D reconstruction of environments and SLAM systems. Egor Bondarev is involved in several European research projects and currently he is a TU/e project leader in the PANORAMA and OMECA projects, both addressing challenges of ultra-high definition UHD and 3D image processing.

Peter H. N. de With graduated in electrical engineering (MSc., ir.) from Eindhoven University of Technology and received his PhD degree from University of Technology Delft, The Netherlands. From 1984 to 1997, he worked for Philips Research Eindhoven, where he worked on video compression and chaired a cluster for programmable TV architectures as senior TV Systems Architect. From 1997 to 2000, he was full professor at the University of Mannheim, Germany, Computer Engineering, and chair of Digital Circuitry and Simulation. From 2000 to 2007, he was with LogicaCMG in Eindhoven as a principal consultant Technical SW and distinguished business consultant. He was also part-time professor at the University of Technology Eindhoven, heading the chair on Video Coding and Architectures. In the period 2008 to 2010, he was VP Video (Analysis) Technology at CycloMedia Technology. Since 2011, he has been an assigned full professor at Eindhoven University of Technology and appointed scientific director Care and Cure Technology and theme leader Smart Diagnosis in the University Health program. He is a national and international expert in video surveillance for safety and security and has been involved in multiple EU projects on video analysis, featuring object and behavior recognition, and also surveillance projects with the Harbor of Rotterdam, Dutch Defense, Bosch Security Systems, TKH-Security, etc. He is board member of DITSS and R&D advisor to multiple companies. He is Fellow of the IEEE, has (co-) authored over 300 papers on video coding, analysis, architectures, 3D processing and their realization. He is the (co-) recipient of multiple papers awards of the IEEE CES, VCIP and Transactions papers and the EURASIP Signal Processing journal paper award.