Real-time planar segmentation of depth images

Journal of Electronic Imaging

JElectronicImaging.org

Real-time planar segmentation of depth images: from three-dimensional edges to segmented planes

Hani Javan Hemmat
Egor Bondarev
Peter H. N. de With

Journal of Electronic Imaging 24(5), 051008 (Sep/Oct 2015)

Real-time planar segmentation of depth images: from three-dimensional edges to segmented planes

Hani Javan Hemmat,* Egor Bondarev, and Peter H. N. de With
Eindhoven University of Technology, Department of Electrical Engineering, Signal Processing Systems, Den Dolech 2, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract. Real-time execution of processing algorithms for handling depth images in a three-dimensional (3-D) 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 3-D application would strongly benefit from a real-time algorithm. We introduce a real-time planar segmentation method for depth images avoiding any surface normal calculation. First, we detect 3-D 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. [DOI: 10.1117/1.JEI.24.5.051008]

Keywords: real-time planar segmentation; three-dimensional edge detection; depth images; three-dimensional reconstruction; GPU implementation.

Paper 15379SSP received May 14, 2015; accepted for publication Sep. 23, 2015; published online Oct. 22, 2015.

1 Introduction

Multiple studies in three-dimensional (3-D) 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, health care, 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 3-D models of environments. With the emerging low-cost sensors like the Xtion and Kinect, multiple 3-D reconstruction applications have been designed. However, reconstructing 3-D 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 3-D applications, e.g., interaction, decreasing model size (decimation), enhancing localization, mapping, and semantic 3-D 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] 3-D 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 preselected points using 3-D 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 postprocessing is necessary to integrate undersegmented regions. In the 3-D 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, postprocessing steps are required for Hough-based methods to merge neighboring 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

*Address all correspondence to: Hani Javan Hemmat, E-mail: h.javan.hemmat@tue.nl

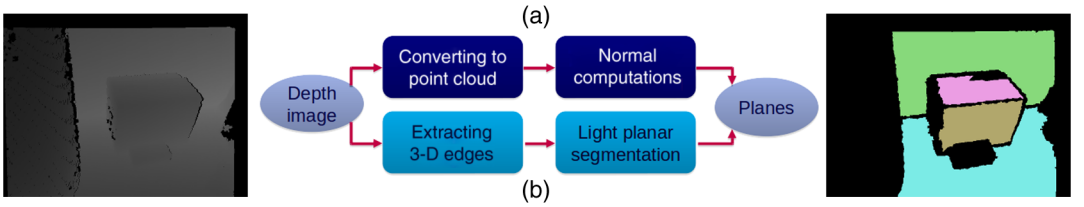

Fig. 1 Two alternative approaches to deal with depth images in order to perform planar segmentation: (a) as point cloud and (b) as two-dimensional images containing three-dimensional (3-D) information.

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 two-dimensional (2-D)/3-D 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 versus the Kinect indoor depth images). Furthermore, we need to detect lines to find vanishing points 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 Fig. 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 2-D frames containing 3-D 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 3-D 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 3-D 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 3-D 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 3-D 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 3-D edge detection and plane extraction components of the proposed algorithm in detail. Experimental results are presented and discussed in Secs. 3 and 4, respectively. Finally, Sec. 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 3-D environment.

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

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

As indicated by property 1, we should search the areas in between 3-D 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 3-D 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 3-D 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

sections, we focus on the methods exploited in each step in detail.

2.1 *Step 1: Edge Detection*

In order to detect 3-D edges in a depth image, the algorithm scans the image in 1-pixel strings in four directions (vertically, horizontally, left diagonally, and right diagonally) as illustrated in Fig. 2(a). Note that the string may be part of a curve in 3-D, while in 2-D, it 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 corner edges emerge where two planes meet each other, and the curved edges are resulting from intersection of actual planar and nonplanar surfaces. Figure 3 depicts the mentioned types of 3-D 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 3-D edges in a depth image.

The following definitions formulate detection criterion for each edge type.

Definition 2.1: Point $P_i(x, y, \text{depth})$ is the i'th 3-D 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 Fig. 2(b). In the remainder of this paper, we use the abbreviated notation of P_i instead of $P_i(x, y, \text{depth})$.

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

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

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

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

2.1.1 *Jump edge*

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

$$|P_i.\text{depth} - P_j.\text{depth}| > \text{Th}_{\text{jump}}. \quad (1)$$

The term P_i.depth represents only the depth value component of 3-D 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

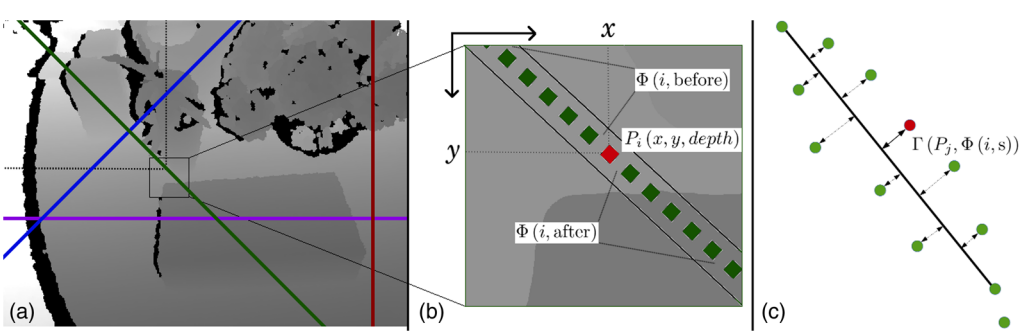

Fig. 2 Illustration of the definitions: (a) a sample depth image with four randomly chosen 1-pixel-wide strings in various directions, (b) a magnified region of a sample string, and (c) the distance of a point to the line connecting two ends of its neighborhood in 3-D.

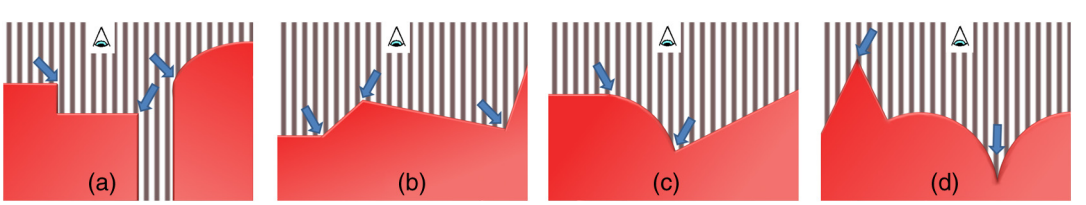

Fig. 3 Various types of 3-D edges in a depth image: (a) jump edges, where there is a gap between two surfaces and/or in case of existence of a hole, (b) corner edges, where two planar surfaces meet each other, (c) situations that planar surfaces join nonplanar ones and curved edges emerge, and finally (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.

user-defined threshold as illustrated in Fig. 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 $\mathrm{Th}_{\mathrm{jump}}$ is consistent with the unit utilized for depth images.

2.1.2 *Corner edge*

The corner edge, as shown in Fig. 3(b), is explained based on the following definition.

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

The term $(P_i,P_{i\pm1})$ in Eq. (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 [Eqs. (3) and (4)] are met:

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

$$|\overline{\mathrm{Sl}}[\Phi(i,\text{before})]-\overline{\mathrm{Sl}}[\Phi(i,\text{after})]|>\mathrm{Th}_{\mathrm{slope}}.\tag{4}$$

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

2.1.3 *Curved edge*

The curved edge, depicted in Fig. 3(c), is specified according to the following definition.

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

This definition is formulated as Eq. (5) based on a user-defined value of $\mathrm{Th}_{\mathrm{straight}}$. Similar to the previous edges, the threshold here can either be a constant or a distance-aware setting. The "1" value for $F_{\text{straight}}(\Phi)$ means that points of Φ are located on a line (and may lie on a planar surface, accordingly). The difference between Eqs. (2) and (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 nonplanar surface. Equation (5) is specified by

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

Point P_i is located on a curved edge if Eq. (6) is satisfied, given by

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

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

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

Equation (2) formulates this definition according to a user-defined value of $\mathrm{Th}_{\mathrm{slope}}$. 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

2.1.4 *Extremum edge*

As shown in Fig. 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 Eqs. (7) and (8), indicating if a point is a local minimum or maximum in its neighborhood, respectively. Equation (7) is given by

$$F_{\min}(P_i,\Phi)=\begin{cases}1 & \forall\ P_j\in\Phi\colon\ P_i.\mathrm{depth}\le P_j.\mathrm{depth}\times\mathrm{En}_{\mathrm{extremum}};\\ 0 & \text{otherwise.}\end{cases}\tag{7}$$

And Eq. (8) is specified by

$$F_{\max}(P_i,\Phi)=\begin{cases}1 & \forall\ P_j\in\Phi\colon\ P_i.\mathrm{depth}\ge P_j.\mathrm{depth}\times\mathrm{En}_{\mathrm{extremum}};\\ 0 & \text{otherwise.}\end{cases}\tag{8}$$

In the previous equations, the multiplicative parameter $\mathrm{En}_{\mathrm{extremum}}$ 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:

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

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

2.2 *Line-Based Plane Detection*

Extracting 3-D edges out of a depth image enables us to perform the second step of finding planes located in between the edges as illustrated in Fig. 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 diagonally, 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

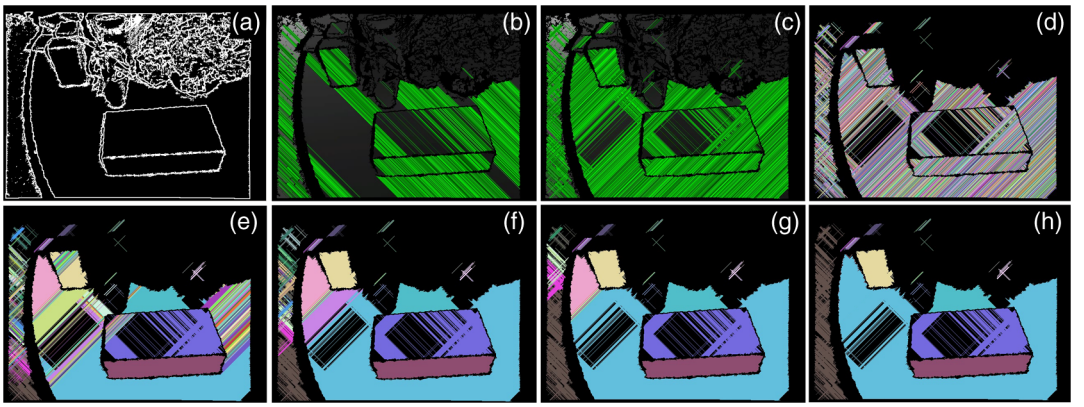

Fig. 4 Planar segmentation of an example depth image from 3-D edges to planar surfaces: (a) 3-D edges detected on a given depth image, (b) left diagonally detected line segments between the 3-D edges, (c) right diagonally detected line segments added (the vertical and horizontal lines are not depicted for the sake of clarity), (d) segmentation process that begins by individually labeling 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).

only line segments in each direction from the string segments [Figs. 4(b) and 4(c)]. After finding all the line segments [Fig. 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, Figs. 4(e)–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 3-D 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 separate 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 3-D 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 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 $\text{Th}_{\text{curvature}k}$, 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 $\vec{\mathbf{n}}(\Psi)$ is the normal vector of plane Ψ, which equals to $\vec{\nu}_2$, which is the smallest eigenvector of the plane.

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

2.3.1 *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 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 $\text{Th}_{\text{curvature}0}$ and $\text{Th}_{\text{curvature}1}$, respectively. Equation (10) formulates the curvature validation process as a condition that should be satisfied by each valid plane, and it is specified by

$$[(T_0 \leq \text{Th}_{\text{curvature}0}) \quad \text{AND} \quad (T_1 \leq \text{Th}_{\text{curvature}1})] = 1. \quad (10)$$

2.3.2 *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 Eqs. (11) and (12) for each pair of plane segments Ψ_a and Ψ_b, given by

$$\vec{\mathbf{n}}(\Psi_a) \times \vec{\mathbf{n}}(\Psi_b) \approx \mathbf{0}, \quad (11)$$

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

If the above conditions are met, the two separated segments are merged into a single plane. If the condition represented by Eq. (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 Eq. (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.

2.3.3 *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

$$\Psi.\text{size} \geq \text{Th}_{\text{size}}, \tag{13}$$

where parameter Ψ.size indicates the number of points located on plane Ψ.

In addition to all these postprocessing enhancements, a preprocessing step is also performed to improve segmentation results even further. After extracting all the 3-D 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 in Ref. 31. Figure 5 depicts several snapshots of the datasets and the corresponding results for both the 3-D 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 480×640 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.

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 multicore and many-core architectures. The OpenMP has been utilized in order to implement the multithreaded version running on multicore 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 W3550 with 3.07 GHz (4 cores) and 20 GB of RAM for both single- and multithreaded 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 applied to depth

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

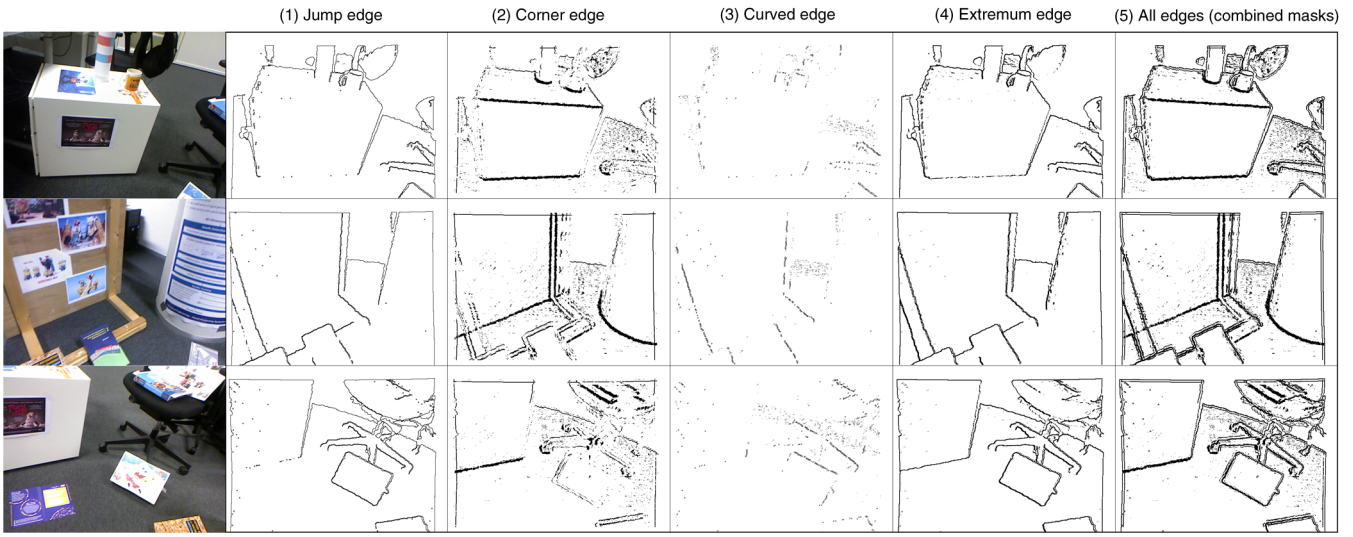

Fig. 6 Three sample scenes and the corresponding edge masks. Columns 1 to 4 show the independent edge masks per edge type for each scene. The combined edges, as the final masks, are depicted in column 5.

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 resulting 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, $\mathrm{Th}_{\mathrm{jump}} = 40$, for corner edges, $\mathrm{Th}_{\mathrm{slope}} = 0.08$ and $n = 16$, for curved edges, $\mathrm{Th}_{\mathrm{straight}} = 4$ and $n = 4$, and finally for the extremum edges, $\mathrm{En}_{\mathrm{extremum}} = 0.98$ 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, Fig. 7 and Table 2 provide comparisons of the different results obtained by the 3-D edge detection algorithm (final mask) for different values of the number-of-neighbors (n in Φ_n) parameter. Among the various 3-D edge detectors, the corner- and curved edge detectors are more sensitive to the number-of-neighbors 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 in Table 3.

4 Discussion

In this Sec. 3, we discuss the results reported in the previous section in detail. First, we briefly assess the quality of the extracted 3-D 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 in Table 4, the resulting images of 3-D edge detection and planar-segmentation algorithms are 97.3 and 97.8% similar to the ground truth, respectively, based on the mean

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

3-D edge detector	CPU (ms) single-threaded	CPU (ms) multithreaded	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**

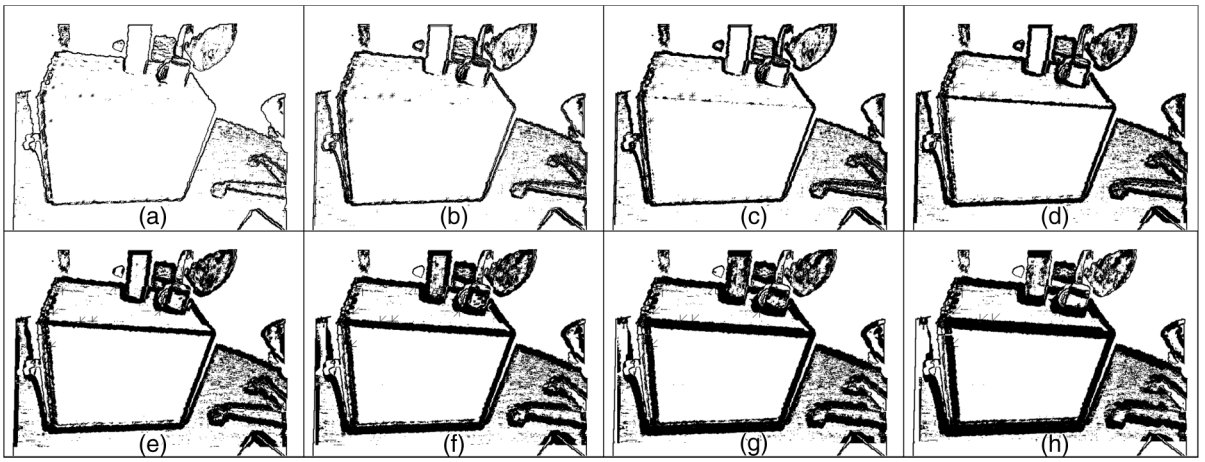

Fig. 7 Effect of the setting value n (number of neighbors) on resulting 3-D edges (final mask): (a) $n = 8$, (b) $n = 10$, (c) $n = 12$, (d) $n = 16$, (e) $n = 20$, (f) $n = 24$, (g) $n = 28$, and (h) $n = 32$.

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

Number of neighbors	CPU (ms) single-threaded	CPU (ms) multithreaded	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**

structural similarity metric. Besides this, 99.6% of ground truth pixels are found and represented by the resulting images of 3-D edge detection algorithm and 98.7% for planar segmentation algorithm according to the data-to-model coverage metric.

As depicted in Fig. 5, the proposed 3-D edge detector algorithm is capable of extracting the 3-D 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 Fig. 6, there is a considerable amount of overlap between various kinds of 3-D edges. Among them, the corner edge generates the most dominant part of the final

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

Algorithm	CPU (ms) single-threaded	CPU (ms) multithreaded	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 3-D edges and planes to the corresponding ground truth data: average similarity based on structural similarity (SSIM) and data-to-model coverage (D2MC) metrics. The D2MC metric determines the amount of pixels in model image that have been covered by data image.

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

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 m to reduce the effect of the Kinect-sensor noise prior to supplying it to the 3-D edge detector. This setting of this threshold is considerably influenced by the Kinect-sensor noise pattern.[28,29]

4.2 *Quantitative Assessment*

4.2.1 *Three-dimensional edge detection*

According to Table 1, the amount of time needed to extract 3-D 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, ~1 s is needed to extract all the 3-D edges of a depth frame by a single-threaded implementation. A multithreaded 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 multithreaded 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 Fig. 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 being 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 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 3-D edge detection is able to achieve a speedup factor of more than 100.

The proposed 3-D edge detector output is generally sensitive to the setting of parameter n, which represents the number of neighbors. This sensitivity on the parameter n is mostly visible in terms of both functionality and obtained execution time. As shown in Table 2 and Fig. 9, there is a linear relation between the value of n and the obtained execution time of the 3-D edge detector. Considering the resulting edges depicted in Fig. 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 (Kinect) to capture the dataset. The detector algorithm is expected to have a different optimum n for any other depth sensor.

4.2.2 *Plane detection*

Detecting planes based on 3-D edges needs less computation compared to 3-D 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 3-D edges. By exploiting multithreaded and many-threaded (GPU) implementations, speedup factors of 1.5 and 12 are achieved, respectively.

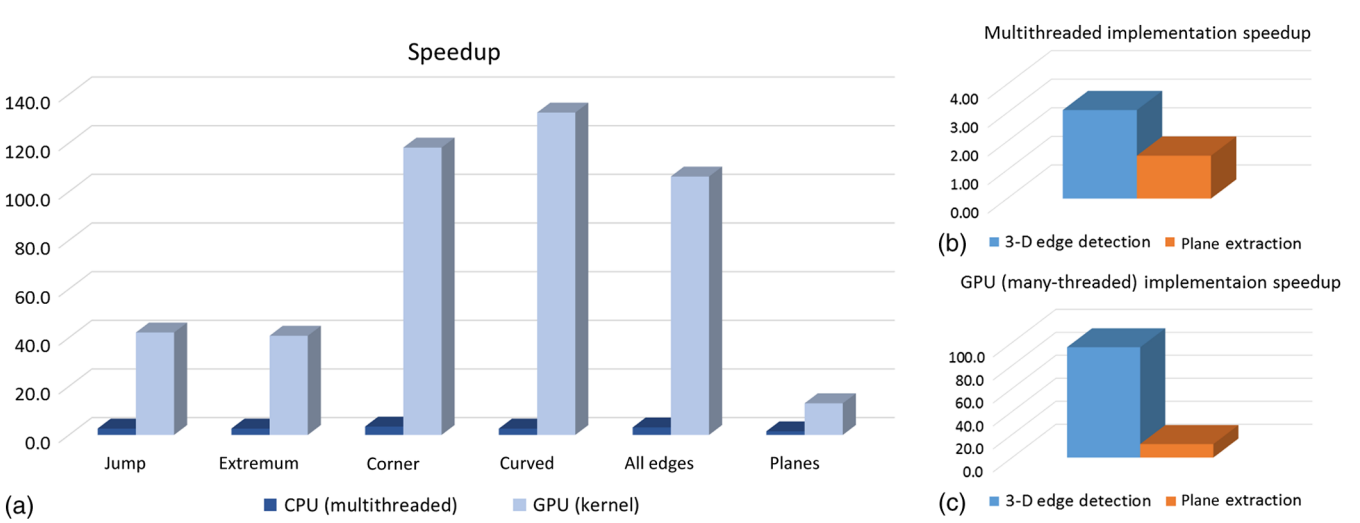

Fig. 8 Speedup gained via multithreaded and GPU-based implementations for edge detector and plane extractor algorithms: (a) multithreaded versus GPU-based implementations, (b) and (c) comparing 3-D edge detection to plane extraction for multithreaded and GPU-based implementations, respectively.

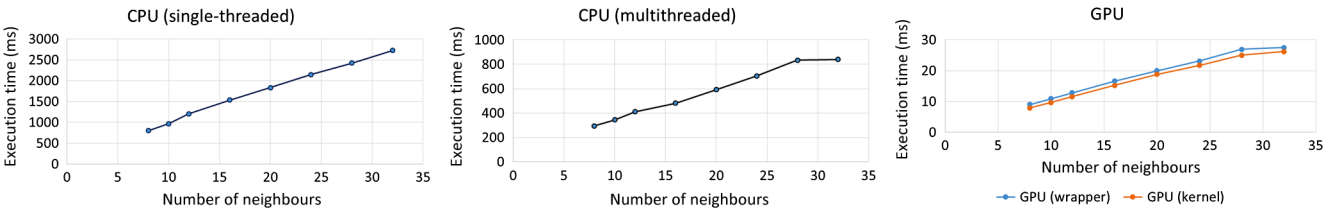

Fig. 9 Effect of the setting variable *n* on execution time of various implementations of the 3-D edge detector algorithm.

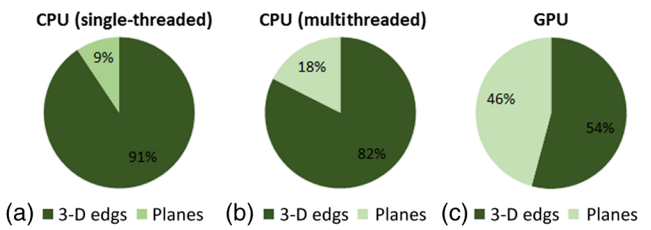

Fig. 10 Percentage of execution time required by each layer of the complete planar segmentation pipeline for various implementations: (a) single-threaded, (b) multithreaded, and (c) GPU-based implementations.

4.2.3 *Planar segmentation as a pipeline*

Planar segmentation of depth images is implemented as a dual-layer pipeline, consisting of 3-D edge detection and plane extraction. As shown in Table 3, on an average, it takes 1102 ms to segment a depth image into its planes for a single-threaded implementation. A multithreaded 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 in Fig. 10, the difference in execution time between the two layers of the pipeline is too high for both the single-threaded and multithreaded CPU-based implementations to be interleaved by ratios 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 an average, resulting in an 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 Figs. 8(b) and 8(c), the 3-D edge detection algorithm reaches a higher speedup factor in both multithreaded and GPU-based implementations by namely 2 and 8 times, respectively, when compared to the plane extraction algorithm.

As depicted in Fig. 11, despite the parallel structure of the proposed algorithms, both the 3-D 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 3-D edge detection algorithm, the difference of execution time between some datasets varies up to 200% and 100% for 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 3-D edges in a depth image and then finds the line segments located in between these 3-D edges. Second, the algorithm merges all the points

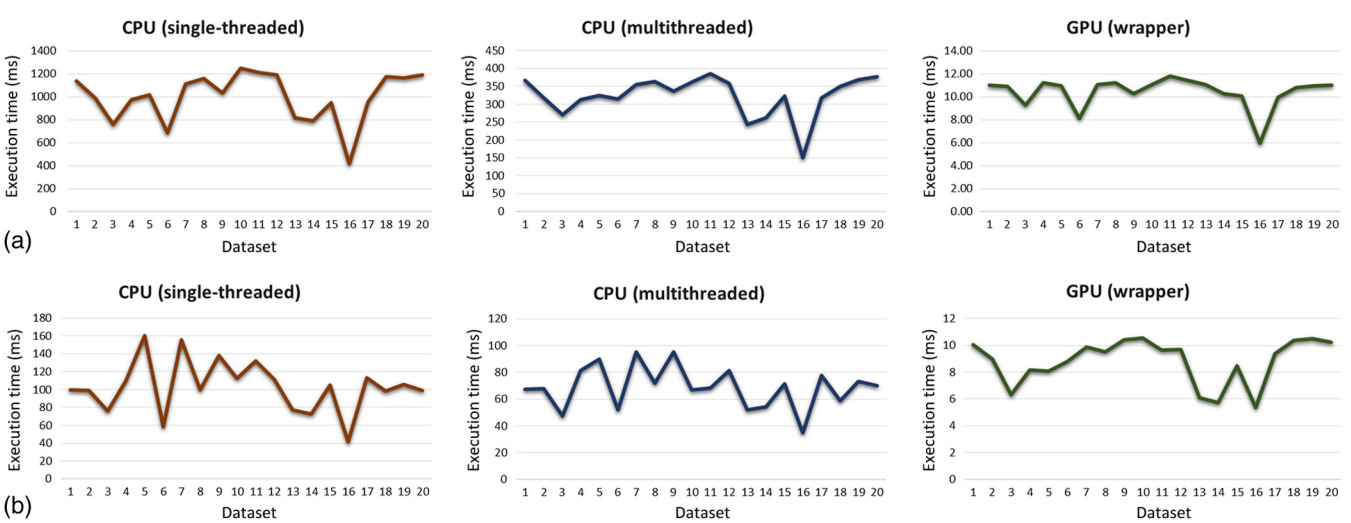

Fig. 11 Execution time per each benchmark of the dataset collection for various implementations: (a) 3-D edge detector and (b) plane extractor.

on each pair of intersecting line segments into a plane candidate. The developed 3-D 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-neighbors 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 3-D edge detection and 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 an average, 3-D 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, cofunded 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 Commission V Symp. Image Engineering and Vision Metrology*, pp. 248–253 (2006).
3. J. Xiao et al., "3D point cloud plane segmentation in both structured and unstructured environments," *Rob. Auton. Syst.* **61**(12), 1641–1652 (2013).
4. D. Borrmann et al., "The 3D Hough transform for plane detection in point clouds: a review and a new accumulator design," *3D Res.* **2**(2), 1–13 (2011).
5. B. Oehler et al., "Efficient multi-resolution plane segmentation of 3D point clouds," *Lec. Notes Comput. Sci.* **7102**, 145–156 (2011).
6. R. Schnabel, R. Wahl, and R. Klein, "Efficient RANSAC for point-cloud shape detection," *Comput. Graph. 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*, pp. 3212–3217 (2009).
8. N. Silberman et al., "Indoor segmentation and support inference from RGBD images," *Lec. Notes Comput. Sci.* **7576**, 746–760 (2012).
9. T.-k. Lee et al., "Indoor mapping using planes extracted from noisy RGB-D sensors," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 1727–1733 (2012).
10. K.-M. Lee, P. Meer, and R.-H. Park, "Robust adaptive segmentation of range images," *IEEE Trans. Pattern Anal. Mach. Intell.* **20**(2), 200–205 (1998).
11. L. Silva, O. Bellon, and P. Gotardo, "A global-to-local approach for robust range image segmentation," in *Proc. of IEEE Int. Conf. on Image Processing*, pp. 773–776 (2002).
12. R. Hulik et al., "Continuous plane detection in point-cloud data based on 3D Hough transform," *J. Vis. Commun. Image Represent.* **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. on Intelligent Autonomous Systems*, pp. 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*, pp. 32–39, IEEE (2012).
15. D. Holz et al., "Real-time plane segmentation using RGB-D cameras," *Lec. Notes Comput. Sci.* **7416**, 306–317 (2012).
16. X. Jiang, H. Bunke, and U. Meier, "High-level feature based range image segmentation," *Image Vis. Comput.* **18**(10), 817–822 (2000).
17. F. A. Andaló, G. Taubin, and S. Goldenstein, "Vanishing point detection by segment clustering on the projective space," *Lec. Notes Comput. Sci.* **6554**, 324–337 (2010).
18. J. C. Bazin et al., "An original approach for automatic plane extraction by omnidirectional vision," in *IEEE/RSJ 2010 Int. Conf. on Intelligent Robots and Systems*, pp. 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*, pp. 475–480 (2007).
20. R. Hulik et al., "Fast and accurate plane segmentation in depth maps for indoor scenes," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 1665–1670 (2012).
21. S. A. Coleman, S. Suganthan, and B. W. Scotney, "Gradient operators for feature extraction and characterisation in range images," *Pattern Recognit. Lett.* **31**(9), 1028–1040 (2010).
22. S. Suganthan, S. A. Coleman, and B. W. Scotney, "Using dihedral angles for edge extraction in range data," *J. Math. Imaging Vis.* **38**(2), 108–118 (2010).
23. A. Hoover et al., "An experimental comparison of range image segmentation algorithms," *IEEE Trans. Pattern Anal. Mach. Intell.* **18**(7), 673–689 (1996).
24. R. Krishnapuram and S. Gupta, "Edge detection in range images through morphological residue analysis," in *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 630–632 (1992).
25. P. Bhattacharya et al., "Hough-transform detection of lines in 3-D space," *Pattern Recognit. Lett.* **21**(9), 843–849 (2000).
26. M. A. Wani and B. G. Batchelor, "Edge-region-based segmentation of range images," *IEEE Trans. Pattern Anal. Mach. Intell.* **16**(3), 314–319 (1994).
27. X. Jiang and H. Bunke, "Edge detection in range images based on scan line approximation," *Comput. Vis. Image Underst.* **73**(2), 183–199 (1999).
28. H. Javan Hemmat et al., "Improved ICP-based pose estimation by distance-aware 3D mapping," in *Proc. Int. Conf. on Computer Vision Theory and Applications*, pp. 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*, pp. 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).
31. H. Javan Hemmat, E. Bondarev, and P. de With, "Demonstration: real-time planar segmentation of depth images," 2015, http://vca.ele.tue.nl/demos/fpsdi/fpsdi.html (16 October 2015).

Hani Javan Hemmat received his BSc degree in computer engineering from Amirkabir University of Technology (Tehran Polytechnic) in 2002. In 2006, he received his MSc degree in computer architecture from the Computer Engineering Department of the Sharif University of Technology, Tehran, Iran. His research interests include three-dimensional (3-D) reconstruction, robotics, parallel processing, hardware-software co-design, and design of hardware/software architectures for real-time implementation. Since 2012, he has researched multimodal fusion and 3-D reconstruction at Technical University of Eindhoven.

Egor Bondarev received his MSc degree in robotics and informatics from the State Polytechnic University, Belarus Republic, in 1997. In 2009, he obtained his PhD in computer science at Eindhoven University of Technology (TU/e), The Netherlands, in the domain of real-time systems. Currently, he is an assistant professor at TU/e, focusing on multimodal sensor fusion, photorealistic 3-D reconstruction and SLAM systems. He is involved in several European research projects and as the TU/e project leader in the PANORAMA and OMECA projects.

Peter H. N. de With graduated in electrical engineering (MSc, ir.) from TU/e and received his PhD from University of Technology Delft, The Netherlands. He has had several positions in academia and industry: in Philips Research Eindhoven working on video compression (1984–1997), a full professor at the University of Mannheim, Germany (1997–2000), in LogicaCMG in Eindhoven as a principal consultant and also a part-time professor at TU/e (2000–2007), as VP video (analysis) technology at CycloMedia Technology (2008–2010). Since 2011, he has been an assigned full professor at TU/e. He is a national and international expert in video surveillance for safety and security and has been involved in multiple EU projects. He is a board member of DITSS and R&D advisor to multiple companies. He is a fellow of IEEE and has co-authored over 300 papers. He is the co-recipient of multiple papers awards.