Real-Time Detection of Planar Regions in Unorganized Point Clouds

Frederico A. Limberger, Manuel M. Oliveira*

Universidade Federal do Rio Grande do Sul
Instituto de Informática - PPGC - CP 15064
91501-970 - Porto Alegre - RS - BRAZIL

Abstract

Automatic detection of planar regions in point clouds is an important step for many graphics, image processing, and computer vision applications. While laser scanners and digital photography have allowed us to capture increasingly larger datasets, previous techniques are computationally expensive, being unable to achieve real-time performance for datasets containing tens of thousands of points, even when detection is performed in a non-deterministic way. We present a deterministic technique for plane detection in unorganized point clouds whose cost is $O(n \log n)$ in the number of input samples. It is based on an efficient Hough-transform voting scheme and works by clustering approximately co-planar points and by casting votes for these clusters on a spherical accumulator using a trivariate Gaussian kernel. A comparison with competing techniques shows that our approach is considerably faster and scales significantly better than previous ones, being the first practical solution for deterministic plane detection in large unorganized point clouds.

Keywords: Hough transform, Real-time plane detection, Unorganized point clouds

---

*Corresponding author. Tel.: +55 51 3308 6821; fax: +55 51 3308 7308.
Email addresses: `falimberger@inf.ufrgs.br` (Frederico A. Limberger), `oliveira@inf.ufrgs.br` (Manuel M. Oliveira)
URL: `http://www.inf.ufrgs.br/~falimberger` (Frederico A. Limberger), `http://www.inf.ufrgs.br/~oliveira` (Manuel M. Oliveira)

1. Introduction

Automatic plane detection in point clouds is a key component in many graphics, image processing, and computer vision applications. These include, among others, model reconstruction for reverse engineering [1, 2, 3, 4, 5], camera calibration [6], object recognition [7, 8], augmented reality [9, 10], and segmentation [11, 12]. The recent popularization of laser scanners has led to an increasingly growth in the sizes of the available datasets, and point clouds containing tens of millions of samples are now commonplace. Software applications like *SynthExport* [13] and *Photosynth* [14] also allow us to extract point clouds from large collections of digital images. However, existing techniques for detecting planar regions in point clouds are computationally expensive and do not scale well with the size of the datasets. For performance improvement, they often exploit non-deterministic strategies, such as working on a randomly-selected sub-set of the original samples. While this can reduce execution time, these techniques are still unable to achieve real-time performance even on datasets containing just tens of thousands of points. More importantly, their results depend on the selected sample sub-sets and, therefore, there is no guarantee that all relevant planes will be detected, or that such results will be consistent across multiple executions.

We present an efficient technique to perform deterministic plane detection in unorganized point clouds whose cost is $O(n \log n)$ in the number of input samples. Our approach scales well with the size of the datasets, is robust to the presence of noise, and handles point clouds with different characteristics in terms of dimensions and sampling distributions. While the actual running times depend on specific features of the dataset (*e.g.*, the number of planar regions), our technique is several orders of magnitude faster than previous ones. For instance, it processes an entire point cloud with 20-million samples (Bremen dataset) in just 2.1 seconds on a typical PC. In contrast, efficient versions of RANSAC can take from 12 minutes to more than 2 hours to process the same dataset, while the Randomized Hough transform takes 42.8 seconds to process only 10% of the samples.

Our technique is based on a robust and fast algorithm to segment point clouds into approximately planar patches, even in the presence of noise or irregularly distributed samples. For this, we use a subdivision procedure to refine an octree and cluster groups of approximately coplanar samples. We use the identified clusters to obtain an efficient Hough-transform voting scheme by casting votes for each of these clusters (instead of for individual samples) on a spherical accumulator. For voting, we use a Gaussian kernel centered at the cluster's best fitting plane, which takes into account the cluster's variances. In this sense, our approach extends the kernel-based voting scheme proposed by Fernandes and Oliveira [15] using a trivariate Gaussian distribution defined over spherical coordinates (θ, ϕ, ρ). While, at first, plane detection in unorganized point clouds might seem as an immediate extension of line detection in images, the lack of explicit neighborhood information among samples imposes significant challenges, requiring new clustering and accumulation-management strategies.

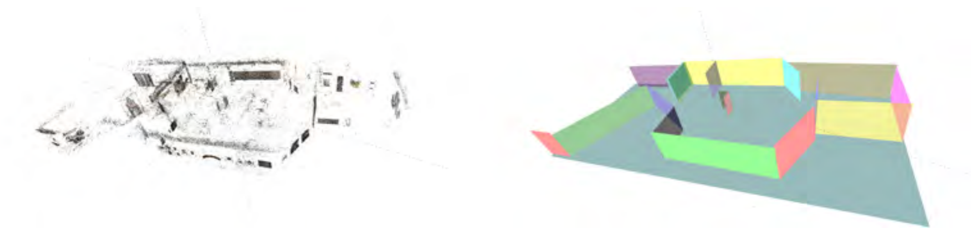

Fig. 1: Example of plane detection using our technique. (left) Museum dataset: point cloud consisting of 179,744 samples obtained from a set of photographs using SynthExport and Photosynth. (right) Planes automatically detected by our technique in just 0.025 seconds on a 3.4 GHz PC. They were manually resized to better represent the original model.

Fig. 1 shows an example of planar regions detected using our technique. The point cloud shown on the left consists of 179,744 samples obtained from a set of photographs taken inside a museum. The samples were extracted using SynthExport [13] and Photosynth [14]. The image on the right shows the planes detected by our technique in just 0.025 seconds on a 3.4 GHz PC, and illustrates the effectiveness of our approach.

The **contributions** of this paper include:

- An $O(n \log n)$ deterministic Hough-transform-based technique for detecting planar regions in unorganized point clouds (Section 3). Our solution is robust to noise, and to sampling distributions. It is a few orders of magnitude faster and scales significantly better than existing approaches. A software implementation of our technique handles datasets with up to 10^5 points in real time on a typical PC;

- A fast Hough-transform voting strategy for planar-region detection (Section 3.3). Our solution uses a robust segmentation strategy to identify clusters of approximately coplanar samples. Votes are cast for clusters as opposed to for individual samples, greatly accelerating the detection process.

2. Related work

The most popular techniques to detect planes in point clouds are the *Hough transform*, *RANSAC*, and *region growing*. This section discusses these algorithms and their various optimizations intended to accelerate plane detection in point clouds.

2.1. Hough Transform

The Hough transform (HT) [16, 17] is a feature-detection technique. For any given input sample, it casts a vote for each instance of the feature one wants to detect that could possibly contain that sample. The votes are accumulated over all samples, and the detected features correspond to the ones with most votes. The time and space complexity of the algorithm both depend on the discretization used for the accumulator, whose dimensionality varies with the number of parameters used to describe the features to be detected. For instance, plane detection requires a 3-D accumulator to represent the three parameters that characterize a plane.

The Hough transform was introduced by Paul Hough [16] for the detection of lines in images. Today, the universally used version of the HT is the *generalized Hough transform* (GHT) proposed by Duda and Hart [17], which replaced the slope-intercept with an angle-radius parameterization based on the normal equation of the line (1):

$$\rho = x\ cos(\theta) + y\ sin(\theta). \tag{1}$$

Here, x and y are the coordinates of a sample pixel, ρ is the distance from a line (passing through the pixel) to the origin of image's coordinate system, and θ is the angle between the normal of the line and the x-axis. This parameterization naturally extends to 3-D, supporting plane detection in the (θ, ϕ, ρ) Hough Space:

$$\rho = x\ cos(\theta)sin(\phi) + y\ sin(\phi)sin(\theta) + z\ cos(\phi). \tag{2}$$

In (2), x, y and z are the Cartesian coordinates of the samples, $\theta \in [0°, 360°)$ and $\phi \in [0°, 180°]$ are the polar coordinates of the plane's normal vector, and $\rho \in \mathbb{R}_{\geq 0}$ is the distance from the plane to the origin of the coordinate system.

The *Standard Hough transform* (SHT) for plane detection uses (2) and iterates over each sample in the point cloud casting votes in the accumulator for all possible planes passing through that sample. More specifically, for given x, y and z coordinates, it iterates over all combinations of θ and ϕ, computing the value of the parameter ρ (2) and casting a vote at the corresponding accumulator cell (or bin). To make the computation feasible, one needs to discretize the θ and ϕ parameter values (defining angular steps). Thus, the computational cost of the SHT is $O(|P|N_\theta N_\phi)$, where $|P|$ is the number of points in the point cloud P, and N_θ and N_ϕ are the number of bins in the discretization of the θ and ϕ angles, respectively.

Given the high computational cost of the SHT, many techniques have been proposed to accelerate its voting procedure. *Common to most of these techniques is the focus on reducing the execution time by using a subset of the points in P, as opposed to designing new algorithms that effectively reduce the asymptotic cost of the voting process.* Next, we briefly review these strategies.

The *Probabilistic Hough transform* (PHT) [18] randomly selects m points ($m < |P|$) and uses them, instead of the entire point cloud, for voting. Since m is a percentage of $|P|$, the asymptotic cost is still $O(|P|N_\theta N_\phi)$. The PHT needs to find an optimal value for m to achieve good results. Small values tend to cause some relevant planes not to be detected, while large values do not result in significant reduction in execution time. As opposed to the SHT, the PHT is not deterministic.

Finding the optimal value for m is not a simple task, as it depends on many characteristics of the point cloud. To overcome this difficulty, the *Adaptive Probabilistic Hough transform* (APHT) [19] monitors the accumulator during the voting procedure. As stable structures emerge, they are stored in a list of potential maximum cells and only this list needs to be monitored. Since the process is adaptive, there is no need for an initial m value. The algorithm ends when the list of potential peaks becomes stable (*i.e.*, when the list does not change for a few iterations). The APHT is sensitive to noise, as the choice of the points is probabilistic and may lead to the detection of planes not present in the dataset. Its asymptotic cost is the same as SHT's.

The *Progressive Probabilistic Hough transform* (PPHT) [20] tries to avoid the influence of random noise by only detecting structures whose number of votes exceeds a threshold defined as a percentage of the total number of votes. Once a structure has been detected, the votes from all samples that support it are removed from the accumulator. Like the previous techniques, PPHT is non-deterministic and its asymptotic cost is the same as SHT's.

The Randomized Hough transform (RHT) [21] reduces the SHT's voting-processing time by exploiting the fact that a plane can be defined by three non-collinear points. The technique randomly selects groups of three non-collinear points and casts a single vote to the accumulator cell corresponding to the plane. This strategy significantly reduces the voting cost. However, the technique is non-deterministic and does not scale well with the size of the point cloud. Among all previous HT-based techniques for plane detection, the RHT is by far the fastest one.

2.1.1. Other Hough Transform Variants

Vosselman et al. [22] proposed a two-step procedure for the Hough transform, exploiting the connectivity of point clouds acquired with laser scanners to calculate the normal vectors of the points. This way, each sample casts a single vote. This approach is not as fast as the RHT, and its use with unorganized point clouds requires estimating surface normals. Bauer and Polthier [23] use the Radon transform (continuous form of the HT) to detect planes on a structured or unstructured grid. They use a subdivided icosahedron, with a Hamiltonian path over the edge graph, to represent the parameter space in order to search for all connected components. This technique requires the use of a grid and its performance is similar to the SHT. More recently, Ogundana et al. [24] used an optimized model for tridimensional sparse matrix to accumulate votes. They propose a robust peak detection algorithm using connected-component labeling and weighted average. Ogundana et al. also showed a Hough transform optimization specific to detect parallel planes, replacing the default accumulator by an unidimensional array, since the planes have the same orientation. Nguyen et al. [25] estimate normals in range images and map such normals to a sphere (a Gauss map) to define plane orientations. Optimization is then used to segment patches of coplanar samples in the range image. The authors demonstrated their technique on simple box-like and polyhedral shapes.

2.2. RANSAC

Another important algorithm for performing shape detection is the Random Sample Consensus (RANSAC) [26]. It performs plane detection iteratively by randomly choosing three points, calculating the plane defined by them, and counting how many points (in the dataset) lie on this plane within some tolerance threshold. The number of points found is called the *score of the plane*. The algorithm stops when it reaches stability, based on a low probability of finding a plane with higher score than the previous ones. RANSAC's computational cost for detecting a single plane is then given by $(I\,(E + |P|\,F)) = O(I\,|P|)$, where I is the number of iterations required to detect a plane, E is the cost of estimating a plane from three points, and F is the cost of checking whether a point lies on a plane. While being robust to noise, RANSAC's random nature makes it non-deterministic. Depending on the choice of its parameter values, the algorithm may detect planes that are not representative of the original dataset. This is illustrated in Fig. 2 for a point cloud corresponding to a cube. The dark gray plane was detected by RANSAC and is one of many planes that could be detected by the algorithm.

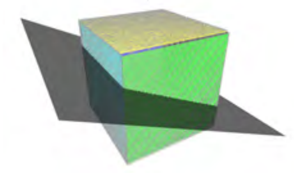

Fig. 2: A point cloud representing the faces of a cube, shown in color. Depending on the used parameters, RANSAC may detected spurious planes, such as the one shown in dark gray, that do not represent the original dataset.

Schnabel et al. [27] introduced an optimization to RANSAC using an octree to establish spatial proximity among samples. In their approach, point selection is performed inside each node, and the score function only tests a local subset of the samples. Since spatial proximity does not guarantee coplanarity, the technique needs to estimate normals for the samples, and the shapes have to be properly sampled. While this approach can significantly accelerate RANSAC, it inherits RANSAC's limitations, and its performance is still far from real time for datasets with a few hundred thousand samples.

2.3. Surface Growing

The third class of techniques used to identify planes in point clouds is surface growing [28, 29, 30, 31] – the 3-D analogue of region growing in images. These techniques perform a local search to identify and expand regions with the same range of characteristics. Surface growing methods require information about the neighbors of each sample, not being directly applicable to unorganized point clouds, which lack explicit connectivity information.

Recently, Deschaud and Goulette [32] proposed an algorithm to detect planes in unorganized point clouds using filtered normals and voxel growing. Their approach assigns a normal to each point through a normal-estimation procedure and uses a voxel-growing algorithm based on these normals. Like other surface-growing techniques, it is slow.

In contrast to the techniques described in this section, our approach is deterministic, does not require connectivity information nor information about sample normals, and its cost is $O(n \log n)$ in the number of samples.

3. Efficient Plane Detection in Point Clouds

This section presents the details of our technique. It has been inspired by the efficient Kernel-based Hough transform (KHT) for straight line detection introduced by Fernandes and Oliveira [15]. However, when dealing with unorganized point clouds, the lack of explicit neighborhood information among samples (which is available for images) requires new and efficient clustering and accumulation-management strategies. In a nutshell, our technique performs a fast and robust octree-based segmentation of approximately coplanar clusters of samples. We then use the identified clusters to perform a Hough-transform voting procedure where votes are cast by clusters (as opposed to by individual samples) on a spherical accumulator. For voting, we use a trivariate Gaussian kernel defined over spherical coordinates (θ, ϕ, ρ) and centered at each cluster's best fitting plane. Peak detection is then performed on the resulting accumulator. Algorithm 1 summarizes the technique.

Algorithm 1 Plane Detection (3-D KHT) Algorithm

```
Require: P {point cloud}
1: $nodes \leftarrow Clustering(P)$ {cluster approximately coplanar samples}
2: for each $n$ in $nodes$ do
3:     $q \leftarrow$ kernel($n$) {kernel estimation ($q$ stores the kernel parameters)}
4:     $accumulator \leftarrow accumulator + voting(q)$ {voting}
5: end for
6: sort accumulator cells by voting importance in descending order
7: iterate over accumulator detecting cells not adjacent to already inspected
   ones {peak detection}
```

3.1. Clustering of Approximately Coplanar Samples

Clustering of approximately coplanar samples is key to our technique as it optimizes the voting procedure, which is the Hough transform's bottleneck. For unorganized point clouds, no neighborhood information among samples is available. For efficiency, we perform clustering by spatial subdivision. For this, we have compared the advantages of using kd-trees and octrees. Kd-trees provide little control over the dimensions of the nodes. According to our experience, subdividing the kd-tree cells using the centroid of the samples tends to lead to thin cells that do not capture the shapes of the planes in the dataset. In contrast, all nodes at a given level of an octree have 1/8 of the size of its parent node and better capture the structure of the planes. Moreover, the costs of creating and manipulating a kd-tree are higher than for an octree. For these reasons, we have chosen an octree as spatial-subdivision data structure. It has proven to be a good choice both in terms of efficiency and quality of the results.

The clustering procedure starts with a root node that includes the entire point cloud, which is then recursively subdivided to refine the octree. Except when the entire point cloud is just a plane, searching for planes in the initial level(s) of the octree often lends to less effective computations. Thus, the procedure only checks for approximate coplanarity among samples after a certain level of the octree has been reached, thus minimizing processing time. According to our experience, starting checking for approximately sample coplanarity at level 4 of the octree provides a good compromise between performance and robustness, and produces good segmentation in practice. The more detailed the point cloud, the more subdivisions are required, as nodes must be small enough to contain only approximately coplanar samples. If no further subdivision is required for an octree node, that branch stops and the node is stored as a cluster. If, on the other hand, the number of samples inside the octree node is smaller than a threshold, the node is marked as not containing a cluster (of approximately coplanar samples).

The procedure for clustering approximately coplanar samples is presented in Algorithm 2. It uses descriptive statistics to analyze the data and calculate the variances associated with the point-cloud distribution. For this, we use principal component analysis (PCA).

Algorithm 2 Clustering

```
Require: n {current node of the octree }
    s {settings to cluster data: $s_{mp}$, $s_{level}$, $s_\alpha$, $s_\beta$ }
Symbols
    $n_{samples}$ {samples in the current octree node }
    $n_{level}$ {level of the current octree node}
    $n_{coplanar}$ {are the samples in current node approximately coplanar?}
    $n_{children}$ {children of the current octree node}
    $s_{ms}$ {minimum number of samples required in a cluster}
    $s_{level}$ {first octree level for checking for approximate coplanarity}
    $s_\alpha$ {relative tolerance associated with plane thickness }
    $s_\beta$ {relative tolerance associated with plane isotropy }
1: $n_{coplanar} \leftarrow false$
2: if $size(n_{samples}) < s_{ms}$ then
3:     return
4: end if{octree subdivision step}
5: if $n_{level} > s_{level}$ then
6:     $\Sigma_{(x,y,z)} \leftarrow cov(n_{samples})$ {covariance matrix in $(x, y, z)$ space}
7:     $(V_{xyz}, \Lambda_{xyz}) \leftarrow eigen(\Sigma_{(x,y,z)})$ {eigen-decomposition}
       {approximate coplanarity test }
8:     if $(\lambda_2 > s_\alpha \lambda_1)$ and $(s_\beta \lambda_2 > \lambda_3)$ then
9:         $n_{coplanar} \leftarrow true$
10:        return
11:    end if
12: end if
13: $n_{children} \leftarrow children(n)$ {initialize the node's eight children}
14: for each $p$ in $n_{samples}$ do
15:     put $p$ in respective child node
16: end for
17: for each $c$ in $n_{children}$ do
18:     call Clustering($c$,$s_{mp}$, $s_{level}$, $s_\alpha$, $s_\beta$) {recursive call for node $c$}
19: end for
```

Since the eigenvalues of the covariance matrix associated to the set of samples inside an octree node represent the proportions of the variances of the sample distribution inside that cell, they can be used to filter out clusters that could not represent planes. In order to check whether a set of

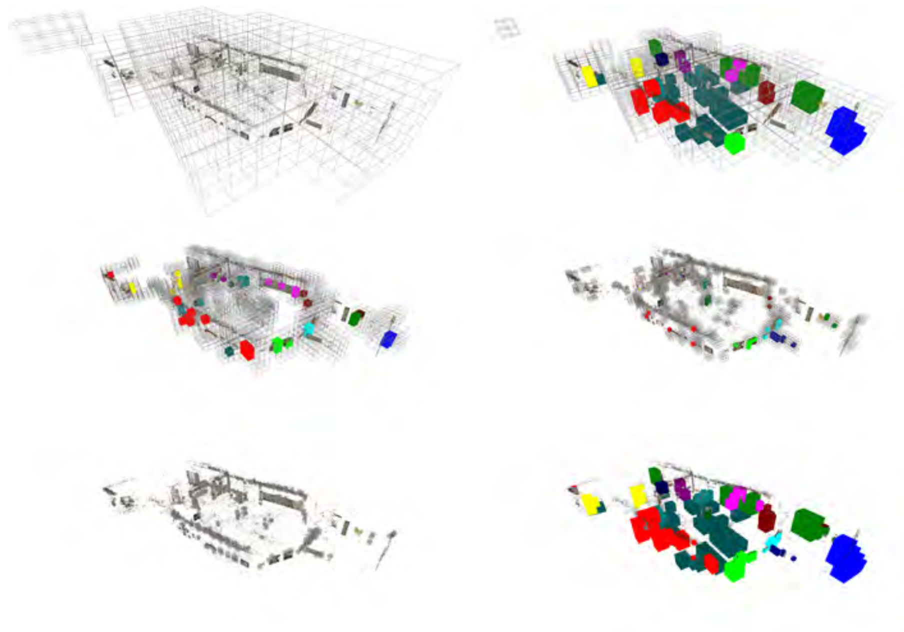

Fig. 3: Adaptive octree refinement and sample clustering for the Museum dataset using Algorithm 2. From left to right, top to bottom, the first five images show the 6th, 7th, 8th, 9th, and 10th levels of the octree. The image at the bottom right shows all nodes at different octree levels containing coplanar samples. Note that once a planar patch is found the subdivision stops for that branch. Each color represents one detected plane, whose reconstructions are shown in Fig. 1.

samples is approximately coplanar, two conditions are verified: the cluster *thickness* and its degree of *isotropy* (the technique should avoid detecting lines and thin elongated clusters as planes). Thus, let Λ and V represent, respectively, the eigenvalues and the eigenvectors of a cluster's covariance matrix Σ. These eigenvalues are non-negative and we sort them in ascending order so that $\lambda_i \leq \lambda_{i+1}$. A test for approximate sample coplanarity can be obtained by checking if $(\lambda_2 > s_\alpha \lambda_1)$ and $(s_\beta \lambda_2 > \lambda_3)$, where s_α and s_β are scaling factors defining relative tolerances for the acceptable amount off-plane displacement (*i.e.*, noise) and degree of sample anisotropy on the cluster. According to our experience, $s_\alpha = 25$ and $s_\beta = 6$ produce good results and have been used for all examples shown in the paper. The recursive subdivision performed by Algorithm 2 stops when the current octree cell is considered to contain approximate coplanar samples (*i.e.*, $n_{coplanar} = true$, line 9) or the cell contains less than a minimum number of samples (s_{ms}, in line 2). Small number of samples tend to provide less reliable estimates for the variances of the samples. In our experience, $s_{ms} = 30$ provides a good threshold for large point clouds.

Once an octree cell is considered to contain an approximately coplanar sample cluster, least-squares is used for plane fitting [33] after discarding samples at a distance bigger than $\tau/10$ from the plane passing by the centroid of the cluster and whose normal is given by the eigenvector with smallest eigenvalue of Σ. τ is the current octree-node edge length.

Fig. 3 illustrates the adaptive octree refinement and sample clustering for the Museum dataset using Algorithm 2. From left to right, top to bottom, the first five images show the 6th, 7th, 8th, 9th, and 10th levels of the octree, respectively. The image at the bottom right shows the octree nodes containing approximately coplanar samples. Note that these nodes might be at different levels of the octree. A single color has been assigned to each plane, even when it spans different levels of the octree. This is possible by keeping track of the clusters who voted for the individual planes.

3.2. Computing Gaussian Trivariate Kernels for Cluster Voting

Let K be a cluster of approximately coplanar samples stored in an octree node, with covariance matrix Σ, and centroid $\mu = (\mu_x, \mu_y, \mu_z)^T$ (Fig. 4). Also let $V = \{\vec{v_1}, \vec{v_2}, \vec{v_3}\}$ be the unit eigenvectors of Σ and let $\Lambda = \{\lambda_1, \lambda_2, \lambda_3\}$ be their respective eigenvalues, so that $\lambda_i \leq \lambda_{i+1}$. The equation of the plane π passing though μ and with normal $\vec{n} = \vec{v}_1 = (n_x, n_y, n_z)^T$ is given by:

$$Ax + By + Cz + D = n_x x + n_y y + n_z z - (n_x\mu_x + n_y\mu_y + n_z\mu_z) = 0 \quad (3)$$

Using (2), one can rewrite (3) using spherical coordinates as:

$$\begin{aligned} &\rho = -D = \mu_x\ n_x + \mu_y\ n_y + \mu_z\ n_z = \sqrt{p_x^2 + p_y^2 + p_z^2}, \\ &\theta = \arctan\left(\frac{p_y}{p_x}\right), \qquad \phi = \arccos\left(\frac{p_z}{\rho}\right), \end{aligned} \quad (4)$$

where $\rho \in \mathbb{R}_{\geq 0}$, $\theta \in [0°, 360°)$, $\phi \in [0°, 180°]$ and $\vec{p} = (p_x, p_y, p_z)^T = \rho\vec{n}$. For θ calculation, if the angle between $\vec{n}$ and $\vec{\mu}$ is bigger than 90°, we reverse $\vec{n}$'s sense (*i.e.*, multiply it by -1). When voting in an accumulator indexed by (θ, ϕ, ρ), the vote distribution is based on the uncertainties associated with each cluster's best-fitting plane π (*i.e.*, the cluster's variances σ_ϕ^2, σ_θ^2, and

σ_ρ^2). A cluster with small variances concentrates its votes in a small region of the accumulator, while a cluster with large variances spreads its votes over a large region, like in the KHT [15].

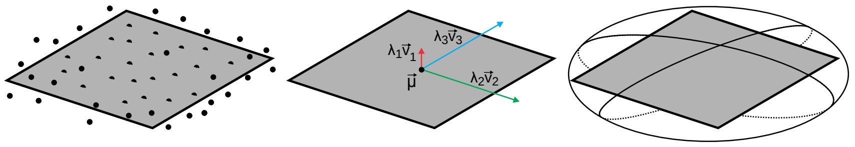

Fig. 4: Samples approximating a planar region in a point cloud, shown with its best-fitting plane (left). Eigenvectors of the covariance matrix Σ associated to the sample distribution (center). Ellipsoid defined by the eigenvalues and eigenvectors of Σ (right).

Algorithm 3 Computing $\Sigma_{(\theta,\phi,\rho)}$ and the Gaussian kernel voting threshold

```
Require: $\Sigma_{(x,y,z)}$ {covariance matrix with respect to x, y and z coordinates}
 1: $J \leftarrow Jacobian()$ {defined in (6)}
 2: $\Sigma_{(\theta,\phi,\rho)} \leftarrow J\,\Sigma_{(x,y,z)}\,J^T$ {cov. matrix in $(\theta,\sigma,\rho)$ space from $\Sigma_{(x,y,z)}$}
 3: $\sigma_\rho^2 \leftarrow \sigma_\rho^2 + \varepsilon$ {add a small value to avoid zero variance}
 4: $(V_{\theta\phi\rho}, \Lambda_{\theta\phi\rho}) \leftarrow eigen(\Sigma_{(\theta,\phi,\rho)})$ {eigen-decomposition of $\Sigma_{(\theta,\phi,\rho)}$}
 5: $\lambda_{\theta\phi\rho\_min} \leftarrow smallestEigenvalueIn(\Lambda_{\theta\phi\rho})$ {smallest eigenvalue}
 6: $V_{\theta\phi\rho\_min} \leftarrow Eigenvector(\lambda_{\theta\phi\rho\_min})$ {normalized eigenvector of $\lambda_{\theta\phi\rho\_min}$}
 7: $std\_dev \leftarrow sqrt(\lambda_{\theta\phi\rho\_min})$ {standard deviation}
 8: $g_{min} = Gaussian(2 \times std\_dev \times V_{\theta\phi\rho\_min})$ {threshold value for voting}
```

The variances and covariances defined in the (θ, ϕ, ρ) space can be estimated from the covariance matrix $\Sigma_{(x,y,z)}$ defined in Euclidean space, using first-order uncertainty propagation analysis [34] as:

$$\Sigma_{(\theta,\phi,\rho)} = \begin{pmatrix} \sigma_\rho^2 & \sigma_{\rho\phi} & \sigma_{\rho\theta} \\ \sigma_{\rho\phi} & \sigma_\phi^2 & \sigma_{\phi\theta} \\ \sigma_{\rho\theta} & \sigma_{\phi\theta} & \sigma_\theta^2 \end{pmatrix} = J\Sigma_{(x,y,z)}J^T = J \begin{pmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{xz} \\ \sigma_{xy} & \sigma_y^2 & \sigma_{yz} \\ \sigma_{xz} & \sigma_{yz} & \sigma_z^2 \end{pmatrix} J^T, \quad (5)$$

where J is the Jacobian matrix:

$$J = \begin{pmatrix} \frac{\partial\rho}{\partial p_x} & \frac{\partial\rho}{\partial p_y} & \frac{\partial\rho}{\partial p_z} \\ \frac{\partial\phi}{\partial p_x} & \frac{\partial\phi}{\partial p_y} & \frac{\partial\phi}{\partial p_z} \\ \frac{\partial\theta}{\partial p_x} & \frac{\partial\theta}{\partial p_y} & \frac{\partial\theta}{\partial p_z} \end{pmatrix} = \begin{pmatrix} n_x & n_y & n_z \\ \frac{p_x p_z}{\sqrt{w}\rho^2} & \frac{p_y p_z}{\sqrt{w}\rho^2} & -\frac{\sqrt{w}}{\rho^2} \\ \frac{-p_y}{w} & \frac{p_x}{w} & 0 \end{pmatrix}. \quad (6)$$

(p_x, p_y, p_z) are defined in (4), $\vec{n} = (n_x, n_y, n_z)^T$, and $w = p_x^2 + p_y^2$.

3.3. Cluster Voting using 3-D Gaussian Distributions

Once we have computed the variances and covariances associated with θ, ϕ and ρ ($\Sigma_{(\theta,\phi,\rho)}$), the votes are cast in the spherical accumulator using a trivariate Gaussian distribution. For the multivariate non-degenerate case, *i.e.*, when the covariance matrix Σ is symmetric and positive definite, its probability density function is given by [35]

$$p(\mathbf{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{k/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^t\Sigma^{-1}(\mathbf{x}-\mu)\right), \tag{7}$$

where $|\Sigma|$ is the determinant of Σ. Considering the trivariate case (*i.e.*, $k = 3$), letting $\vec{\delta} = \mathbf{x} - \mu$ be the displacement with respect to the center, and since the votes are already centered at the best-fitting parameters (θ, ϕ, ρ), this equation can be rewritten as

$$p(\vec{\delta}, \Sigma) = \frac{1}{15.7496\,|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}\;\vec{\delta}^t\;\Sigma^{-1}\;\vec{\delta}\right). \tag{8}$$

Casting votes for a given accumulator cell requires two matrix-vector multiplications and one exponentiation, since the determinant of the covariance matrix and its inverse need to be calculated only once per cluster. While the values of σ_θ^2 and σ_ϕ^2 are never zero, the value of σ_ρ^2 will be zero if all samples in the cluster are exactly coplanar. In such a case, $\Sigma_{(\theta,\phi,\rho)}$ becomes singular, and voting should be done using a bivariate Gaussian kernel defined over (θ, ϕ). We avoid the need to handle such a special case by adding a small value ε to σ_ρ^2 (*e.g.*, $\varepsilon = 0.001$, see line 3 in Algorithm 3). Thus, voting can always use a trivariate Gaussian kernel, without affecting the results.

As planes become more horizontal (*i.e.*, when ϕ approaches 0 or 180 degrees) the variance relative to θ tend to become large, since at the poles the parameter θ does not affect the orientation of a plane. As a result, the amount of votes in individual accumulator cells near the poles tend to be smaller than in voted cells in other regions of the accumulator. Although this does not affect the correct detection of peaks around the poles, sorting the detected planes taking into account only the amount of votes would lead to always finding vertical planes before horizontal ones.

When estimating the importance of a cluster (and consequently the importance of its votes), one should consider other properties besides its number of samples. Aspects, such as area coverage should be given greater importance as sampling density varies with object distance to the scanner. Thus, we weight votes from a cluster by the relative size of its octree node with respect to the size of the octree root (in our system, all octree cells, including the root, are cubic cells). Votes are then weighted using

$$w_{cluster(i)} = w_a \left(\frac{node_{size}}{octree_{size}} \right) + w_d \left(\frac{|C_i|}{|P|} \right), \tag{9}$$

where $node_{size}$ and $octree_{size}$ are, respectively, the edge length of the octree node containing the cluster and the edge length of the root node. $|C_i|$ is the number of samples in the cluster i and $|P|$ is the total number of samples in the point cloud. w_a and w_d are the weights associated with *relative area* and *relative number of samples*, such that $w_a + w_d = 1$. According to our experience, spatial coverage should be favored over number of samples. While we have used $w_a = 0.75$ and $w_d = 0.25$ in all examples shown in the paper, even $w_a = 1$ and $w_d = 0$ produce good results in practice.

The voting procedure uses a spherical accumulator indexed by (θ, ϕ, ρ), which is described in Section 3.4. Starting at the center of the 3-D Gaussian kernel representing the position, orientation, and uncertainties of the best-fitting plane for a given cluster, the voting procedure iterates away from the kernel's center up to two standard deviations storing votes in the accumulator's cells. This provides a 95.4% assurance that the selected region of the parameter space receiving votes covers the true plane. Sampling is performed in the accumulator at steps of $\Delta\theta$, $\Delta\phi$ and $\Delta\rho$. The number of votes that a cluster casts in a given accumulator cell a is obtained by multiplying the weight of the cluster's vote (9) by the evaluation of (8) for the cell's $(\theta_a, \phi_a, \rho_a)$ parameter values (*i.e.*, for $\vec{\delta} = (\theta_a, \phi_a, \rho_a) - (\mu_\theta, \mu_\phi, \mu_\rho)$).

3.4. The Spherical Accumulator

While a 2-D array provides a good accumulator for the detection of lines in images, Borrmann et al. [36] have demonstrated that the use of a 3-D accumulator array for plane detection may compromise the result of the HT. They have shown that since polar regions must have fewer cells than equatorial regions, the use of a full 3-D array may result in cells improperly receiving less votes than others. To overcome this problem, Borrmann et al. have proposed a spherical accumulator called the *accumulator ball*, whose illustration is provided in Fig. 5.

While Borrmann et al. cast votes in a conventional way (*i.e.*, they cast one vote for each possible plane passing through each sample in 3-D), we vote for

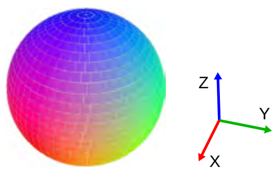

Fig. 5: A discrete representation of the 3-D spherical accumulator, showing the individual cells for a given value of the parameter ρ. The colors represent normal directions. The coordinate axes on the right indicate directions of the normal vector components.

each entire cluster at once. Thus, we also need to cast votes for cells adjacent to the one that represents the cluster's best-fitting plane. This is required to account for the uncertainty resulting from the variances in the cluster's sample positions. We use Borrmann et al.'s spherical accumulator, but normalize the azimuthal angle $\theta \in [0°, 360°)$ to $[0, 1)$, as its discretization varies with the elevation angle ϕ (see Fig. 5). For any value of ϕ, the θ index for actually accessing the spherical accumulator is obtained as $\theta_{index} = round(\theta\, nc(\phi))$, where $nc(\phi)$ is the number of accumulator cells for the elevation angle ϕ: $nc(\phi) = 2 \times length(\phi_{vector}) \times sin(\phi)$. At the poles, we set $nc(\phi) = 1$.

For identifying adjacent cells, θ_{index} must be incremented/decremented using modular arithmetic. The ϕ_{index} must be between 0 and the size of the array (ϕ_{max}), which varies with the accumulator discretization. If incrementing/decrementing ϕ_{index} should exceed its lower or upper bounds, the sign of its increment is reversed to guarantee the wrapping around the sphere. Finally, ρ_{index} must be between 0 and ρ_{max}, which depends on the point cloud size. If ρ_{index} exceeds the limit of ρ_{max}, the voting process stops; if, however, it assumes a negative value, θ_{index} and ϕ_{index} are recalculated for angles θ and ϕ in the opposite sense along the same direction.

Votes in a HT 3-D accumulator tend to be sparsely distributed. Thus, during initialization, we only allocate space for the angular discretization of θ and ϕ. The third dimension (ρ) is allocated as needed during the voting procedure. Therefore, if a range of ρ values are only required around certain values of θ and ϕ, they will only be allocated there. This strategy lends to considerable memory savings, allowing such memory to be used towards better angular discretization, resulting in more accurate detections. Fig. 6 illustrates the data structures used for implementing the spherical accumulator: a 1-D array with size ϕ_{max} represents the discretization of the elevation angle ϕ. Each element of this array contains a pointer to another 1-D array representing the discretization of the azimuthal angle (θ) at that specific elevation. In turn, each element of a θ array stores a pointer to yet another 1-D ρ array that stores the number of votes cast to cells indexed by (θ, ϕ, ρ). Note that the arrays at this last level are only allocated if necessary.

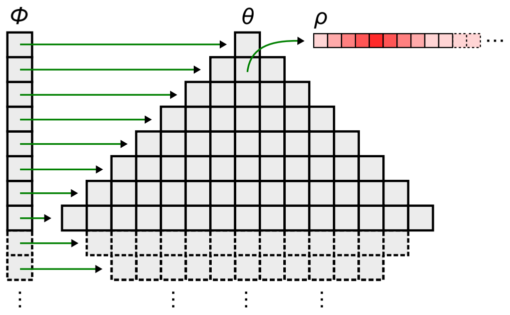

Fig. 6: The spherical accumulator and its representation in memory (the bottom half is just indicated). The angular discretization (θ, ϕ) behaves like a sphere. The indexing of the azimuthal angle (θ) uses modular arithmetic (*i.e.*, it wraps around). Each (θ, ϕ) cell has a points to a vector (allocated as needed during the voting process) storing the actual votes associated with the distances from the origin (ρ), thus covering all possible orientations and positions.

The accumulator discretization can be adjusted by choosing the number of ϕ cells, since the number of θ cells is automatically calculated to represent the same proportion of the arc length discretization. The number of cells used for radial discretization (ρ) is adjusted according to size of the point cloud to emphasize either performance or precision.

Accumulator structures used for HT store a discrete number of votes, which is true even for the KHT [15]. In our solution, a vote represents a plane and the uncertainty associated to its exact location and orientation. Thus, our accumulator uses floats instead of integers to reduce the influence of rounding errors. This improves the accuracy of our solution allowing the accumulation of fractional votes, at the expense of a small increment (approximately 4%) in the execution time. Since our solution already achieves real-time performance for relatively large datasets, this additional cost is not perceived by the users.

3.5. Peak Detection

The last stage of a Hough-transform consists of detecting peaks of votes in the accumulator. We optimize this process by using a 1-D auxiliary array (AA) to store the (θ,ϕ,ρ) coordinates of accumulator cells as they receive votes for the first time during the voting procedure. Only cells in this array need to be inspected for peak detection. As demonstrated in the KHT [15], low-pass filtering the accumulator smooths the voting map, consolidating adjacent peaks. Therefore, before peak detection is actually performed, we apply a low-pass filter to the accumulator cells whose indices have been stored in AA. For this, we use a 3-D isotropic kernel comprised of seven cells (the central one and its six-connected neighborhood in 3-D). This topology was used to handle the discretization of the accumulator, which has a singularity at the poles. The filtered results are stored in the corresponding cells of AA, thus avoiding the need for an extra copy of the accumulator. The kernel weights should satisfy $w_c, w_n > 0$, $w_c + 6w_n = 1$, and $w_c \geq w_n$, where w_c and w_n are the weights of the central and neighbor cells, respectively. Although various combinations of w_c and w_n values produce good results in practice, according to our experience the use of $w_c = 0.2$ and $w_n = 0.133$ seem to provide the best compromise between peak consolidation and smoothness.

For peak detection, the cells in AA are sorted in decreasing order with respect to the stored filtered values. Iterating over this sorted array, the algorithm inspects each peak candidate and checks if the corresponding accumulator cell has already been visited. If not, the cell is chosen as a peak and its (up to) 26 neighbors are tagged as *visited*. If the cell has already been visited, their neighbors are also tagged as *visited*. This procedure guarantees that only true peaks will be selected to represent the output planes.

The number of votes cast by a plane on the cells of a spherical accumulator decreases as one moves from the equator to the poles. This is illustrated in Fig. 7(a), which compares the distribution of votes cast by a cluster as it is rotated around the origin. The color scale indicates the number of cast votes, while the thumbnail image on its right shows the best fitting planes corresponding to the rotated cluster. Note that the number of votes cast on cells around the poles are significantly smaller than the ones near the equator. Figs. 7(b) and 7(c) illustrate this behavior, for a fixed value of the parameter ρ. Fig. 7(b) shows two versions of the rotated point cloud: one near the equator and the other near the north pole. The noise in the point cloud lends to some uncertainty on the plane's orientation, which is represented by a cone of normals around the normal of the best-fitting plane (shown in red). On the equator, such uncertainty causes some votes to be cast in a small θ and ϕ neighborhood around the (θ, ϕ, ρ) coordinates of the best fitting plane. There, equal angular steps in θ and in ϕ correspond to arc lengths of equal sizes, resulting in an isotropic Gaussian kernel in the (θ, ϕ)

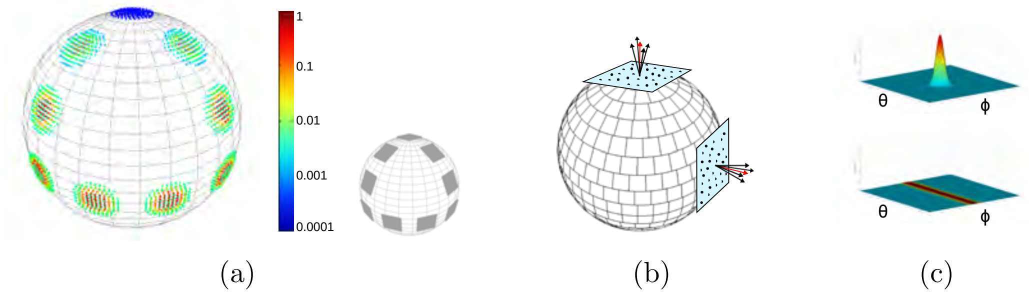

Fig. 7: The number of votes cast by a cluster as it is rotated varies with the position on the spherical accumulator (a). The color scale indicates the number of votes, while the thumbnail image on its right shows the best-fitting planes corresponding to the rotated clusters. (b) On the equator, the uncertainty on the plane orientation lends to votes on a small isotropic neighborhood in the (θ, ϕ) subspace. At (next to) a pole, the same uncertainty on the plane orientation lends to a small uncertainty in the ϕ dimension, but to a big uncertainty in the θ dimension, as θ can range from 0 to 360 degrees. (c) isotropic (top) and truncated anisotropic (bottom) Gaussian kernels in the (θ, ϕ) subspace associated to the cluster near the equator and near the pole, respectively.

subspace. Such a Gaussian is illustrated on the top portion of Fig. 7(c). Near a pole, on the other hand, the uncertainty on the plane's normal lends to a small uncertainty in the parameter ϕ, but to a huge uncertainty in the parameter θ, as at the pole the value of θ varies from 0 to 360 degrees. This results in a highly anisotropic Gaussian kernel in the (θ, ϕ) subspace, as shown by the truncated kernel at the bottom of Fig. 7(c). This wider and lower Gaussian covers a large θ neighborhood, but the voting procedure constrains voting to values of at least g_{min}. g_{min} is obtained by evaluating Eq. 8 for $\vec{\delta} = 2 \times std\_dev \times V_{\theta\phi\rho\_min}$ (see line 8 of Algorithm 3). This explains the smaller number of votes per cell as a cluster approaches a pole.

Let C_1 and C_2 be two clusters with the same number of samples and same variances, C_1 located near the equator, and C_2 near a pole. Considering only the number of votes, C_1 would always be detected earlier, as its peak is higher than C_2's. To retrieve the detected planes based on how representative they are for a scene, as opposed to just on the heights of their associated peaks, *the list of detected planes is sorted based on the sum of the weights (9) of all clusters that voted for each plane* π_i:

$$w_{sum}(\pi_i) = \sum_{voted\ for\ \pi_i} w_{cluster}. \tag{10}$$

3.6. Algorithm Complexity

The overall steps of our plane-detection technique can be summarized in Algorithm 4. The cost of constructing an octree for a set of $|P|$ samples is $O(|P| \log_8 |P|)$. Checking whether a cluster C_i with $|C_i|$ samples (inside an octree node) is approximately coplanar requires computing its covariance matrix $\Sigma_{(x,y,z)}$ and comparing its eigenvalues (lines 7 and 8 in Algorithm 2). These operations are performed in time $O(|C_i|)$. Since this operation is used to decide if a node needs to be subdivided, it has to be performed in all nodes of the octree, resulting in a total cost of $O(|P| \log_8 |P|)$. Transforming $\Sigma_{(x,y,z)}$ to the (θ, ϕ, ρ) space requires computing a Jacobian matrix and multiplying three 3×3 matrices (see 5 and 6), which has cost $O(1)$. Computing the eigenvalues of $\Sigma_{(\theta,\phi,\rho)}$ costs $O(1)$. These operations (lines 5 and 6 in Algorithm 4) are executed once per cluster, whose number is bounded by $O(|P|)$. Each cluster C_i casts votes over a total v_{C_i} cells. Note that since in a spherical accumulator each cell represents a (set of) plane orientation(s), each cluster only votes for a relatively small number of cells. Thus, typical numbers for v_{C_i} vary from 20 to 50, depending on the distribution of samples in the cluster, and on the resolution of the accumulator. Let B be the total number of accumulator bins that received some votes. Since there are $O(|P|)$ clusters and each cluster votes for a finite number of cells, $B = O(|P|)$. Filtering any given accumulator cell is performed in $O(1)$ time, for a total cost of $O(|P|)$. Sorting the B voted cells is accomplished in $O(|P| \log |P|)$, and peak detection has cost $O(|P|)$. Thus, *the overall time complexity of the algorithm is* $O(|P| \log |P|)$. By restricting the maximum number of levels of the octree, one also restricts the number of clusters, making the time complexity linear in the number of samples: $O(|P|)$. While this could accelerate the process, it may make it harder to detect small (approximately) planar patches.

3.7. Space Complexity

The amount of memory required by our 3-D Kernel-based Hough transform (3-D KHT) consists basically of the octree (used to store point-cloud samples and indexes), the voting map, the 1-D auxiliary array (AA), and the trivariate kernels. Except for the root, which stores the samples, each octree node only stores (integer) indexes for the sample array. Since each sample stores its (x, y, z) coordinates as doubles, the memory required for the octree is given by $3 \times 8 \times |P|$ bytes for the samples themselves, and $4 \times |P| (\log_8 |P| - 1)$ bytes for the remainder of the octree. The voting map, in turn, depends on the discretization of the Hough space (θ, ϕ, ρ) and on how

Algorithm 4 Algorithm Summary and Asymptotic Cost

```
Require: P {point cloud}
 1: Octree generation; $\{O(|P| \log_8 |P|)\}$
 2: for each octree node do
 3:   Compute cluster covariance matrix $\Sigma_{(x,y,z)}$; $\{O(|C_i|)\}$
 4:   if cluster is approximately co-planar then
 5:     Transform covariance matrix $\Sigma_{(x,y,z)}$ to $(\theta, \phi, \rho)$ space; $\{O(1)\}$
 6:     Compute eigenvalues of $\Sigma_{(\theta,\phi,\rho)}$; $\{O(1)\}$
 7:     Cast cluster votes and update the auxiliary array $(AA)$; $\{O(1)\}$
 8:   end if
 9: end for
10: Filter accumulator cells pointed by $AA$, storing result in $AA$; $\{O(|P|)\}$
11: Sort $AA$; $\{O(|P| \log |P|)\}$
12: Iterate over sorted $AA$ detecting peaks; $\{O(|P|)\}$
```

many cells receive votes (only voted cells are allocated in memory). Each accumulator cell consists of one float for storing its votes, and one boolean to indicate if it has already been inspected by the peak-detection procedure. Each AA cell stores one double and two shorts for the indexes for θ, ϕ, and ρ, respectively, and one float for the filtered peak value. Finally, a trivariate kernel stores a 3×3 covariance matrix and the (θ, ϕ, ρ) Hough coordinates of the best fitting plane, resulting in 12 doubles per cluster. While voting could be performed in parallel on-the-fly as we reach the octree leaf nodes, we have not implemented concurrency control mechanisms for accessing the accumulator, and voting is performed in a serial fashion (see Algorithm 4). *The space complexity of the algorithm is then* $O(|P| \log_8 |P|)$. Similarly to time complexity, restricting the maximum level of the octree would make the space complexity linear in the number of samples: $O(|P|)$.

4. Results

We have implemented our technique in C++, using OpenMP to parallelize the octree generation, and *dlib* [37] to compute eigenvalue decompositions. We used OpenGL to render the detected planes. We have used this implementation to automatically detect planes in a large number of unorganized point clouds, and compared its performance against the state-of-the-art approaches: the Randomized Hough transform (RHT) and two efficient versions of RANSAC. Since *surface-growing* techniques are not as fast as RHT and RANSAC, and require information about neighbor samples (see Section 2.3), they were not included in our performance comparisons.

To evaluate the accuracy of our approach, we created a point cloud (Box) by sampling the faces of a cube centered at the origin and with a side of 400 units. Each face of the cube contains 160,801 samples, to which we added 2.5% of uniformly-distributed noise (*i.e.*, using a uniform distribution of noise values ranging from 0 to 2.5% of the side of octree root node). This point cloud is shown in Fig. 8(a) and was also used for the RANSAC experiment shown in Fig. 2. An unfolded slice of the spherical accumulator displaying the six peaks detected by our technique is shown in Fig. 8(b). Four of these peaks are equally distributed on the central line (equator) of the accumulator, while gray indicates zero votes. Such peaks correspond to the lateral faces of the cube. The two additional peaks are at the poles (shown as the blue lines on top and at the bottom of the gray region), and correspond to the top and bottom faces of the cube. The detected planes are shown in Fig. 8(c). Note that only six planes were found as our trivariate Gaussian kernel naturally handles the noise in the dataset. We have also rotated the point cloud by arbitrary amounts and around arbitrary axes and verified that our technique accurately detected the six planes in all cases (Fig. 8(d)).

To evaluate the robustness of our technique to missing samples and noise, we downsampled a noiseless version of the Box dataset to 48,000 points and added 1% of Gaussian noise (*i.e.*, using a normal distribution with $\sigma = 1\%$ of the side of octree root node). For this experiment, each face of the cube corresponds to three discontinuous stripes of samples covering approximately 60% of its original area. The resulting point cloud is illustrated in Fig. 8(e) with six colored squares representing the most important detected planes. As in the previous experiment, the point cloud was rotated by arbitrary amounts around arbitrary axes, consistently producing the same results.

For performance comparisons, we used the optimized RANSAC technique (and code) of Schnabel et al. [27], and the RANSAC implementation for plane detection in point clouds available in the *Point Cloud Library* (PCL) v1.7 [38], a modern C++ library for 3-D point-cloud processing. For RHT, we used the implementation by Borrmann et al. [36]. These implementations proved to be the most efficient ones for plane detection using RANSAC and RHT, respectively. All experiments were performed on an Intel i7-2600 3.4 GHz CPU with 16 GB of RAM. The codes for the four compared techniques were compiled for 64 bits to exploit the full potential of the hardware. Fig. 9

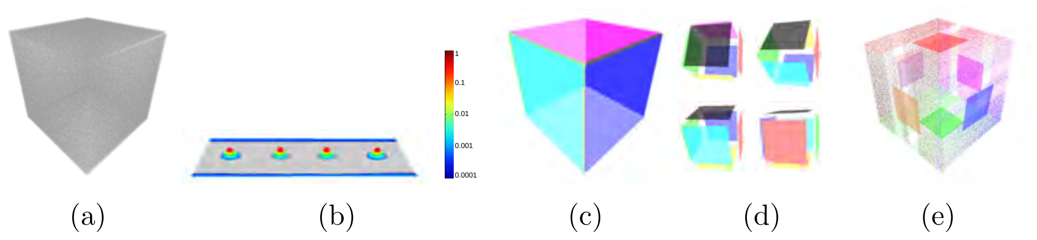

Fig. 8: Box dataset. (a) A point cloud representing a cube centered at the origin. Each face consists of 160,801 points with 2.5% of uniformly-distributed noise. (b) A 3-D visualization of an unfolded slice of the accumulator representing all pairs (θ, ϕ) for one value of ρ after the voting procedure. There are six detected peaks: four equally distributed on the gray region represent the lateral faces of the cube, plus two at the poles (shown as the blue lines) corresponding to the top and bottom faces. (c) Reconstructed planes from the peaks detected by our approach. (d) Detected planes (with random colors) after rotating the point cloud by 20, 40, 60 and 80 degrees around the x-axis. (e) Downsampled version of the cube, where each face is covered by three stripes of samples covering approximately 60% of its original area. The squares indicate the detected planes.

shows the datasets used for performance comparisons. On the right, it also shows the most representative planes detected by our approach (for the more complex examples, some planes are not shown to avoid cluttering the images). These datasets include a computer desk (*Computer*), a room (*Room*), a set of façades from a city block (*Utrecht*), the interior of a museum (*Museum*), and façades of some buildings in the city of Bremen (*Bremen*). The *Computer* dataset is from [36]. The other point clouds were extracted from sets of photographs using SynthExport [13] and Photosynth [14]. They were chosen because they span a large range of parameters, varying in number of points, sampling rate, occupied volume, and number of detectable planes.

Table 1 presents detailed information about the experiments performed with our technique on each dataset. These times were computed by averaging the results of 50 executions. It shows that our approach processes the *Computer* dataset with its 68K samples in approximately 22 milliseconds, and the *Museum* dataset, which contains 179K samples, in 25 milliseconds. For larger point clouds (e.g., *Bremen*, with 20 million samples) the octree creation (column 6) dominates the time of 3-D KHT. Still, our technique processes the Bremen dataset 353 faster than Schnabel et al.'s RANSAC technique [27], and 3,586 times faster than PCL's RANSAC [38]. The available implementation of the RHT could not handle the entire dataset. Working on the full dataset, our technique is still 20 times faster than the RHT working on a subset containing only 10% of the original samples (Table 2). Table 1 also shows that even though the octree might segment large coplanar structures into several clusters, such clusters vote for the same regions in the accumulator, resulting in the detection of the right planes. In our experiments, all datasets were processed in their original scales.

Table 1: Data on the experiments performed with our technique. Number of samples in the point clouds, numbers of detected clusters, number of samples used in the voting procedure, octree-generation time, voting time, peak-detection time, and number of detected planes.

Point Cloud			Octree			Time (sec)			Result
Name	Size	Bounding Box	Clusters	Used Points	Rate(%)	Clustering	Voting	Peaks	Planes
Computer	68 852	$1.3 \times 3.0 \times 0.9$	119	30 630	44.48	0.005	0.009	0.008	8
Room	112 586	$29.2 \times 14.5 \times 3.1$	339	66 682	59.22	0.009	0.02	0.012	40
Utrecht	160 256	$75.8 \times 75.8 \times 37.3$	393	92 839	57.93	0.024	0.005	0.011	38
Museum	179 744	$72.4 \times 132.8 \times 23.1$	232	121 943	67.84	0.013	0.007	0.005	21
Box	964 806	$409.9 \times 409.9 \times 409.9$	144	584 028	60.53	0.054	0.008	0.015	6
Bremen	20 332 246	$110.2 \times 379.3 \times 84.6$	7 489	17 929 145	88.18	2.05	0.033	0.022	202

Table 2: Performance comparison of our approach (3-D KHT) against RHT, Schnabel et al.'s RANSAC [27], and PCL's RANSAC [38] for various datasets. The entries of the table show the execution times (in seconds) of the four techniques for these datasets. (*) The RHT was computed with a simplified version of Bremen dataset containing only 2 million samples, because the available implementation did not support larger inputs.

	Computer	Room	Utrecht	Museum	Bremen
3-D KHT	0.022	0.041	0.040	0.025	2.105
RHT	0.121	6.313	2.814	11.960	42.824*
RANSAC [27]	0.340	0.774	0.919	1.200	745.055
RANSAC [38]	0.424	3.293	15.412	302.610	7,531.010

Table 2 compares the performance of our technique to RHT and RANSAC. These results show that our approach is one to four orders of magnitude faster than the competing ones. Although RHT and RANSAC are relatively fast on small datasets containing low noise and just a few planar structures (e.g., *Computer*), they are not as efficient on bigger and noisier datasets (*e.g.*, *Bremen*). Our approach can efficiently handle both large and noisy datasets.

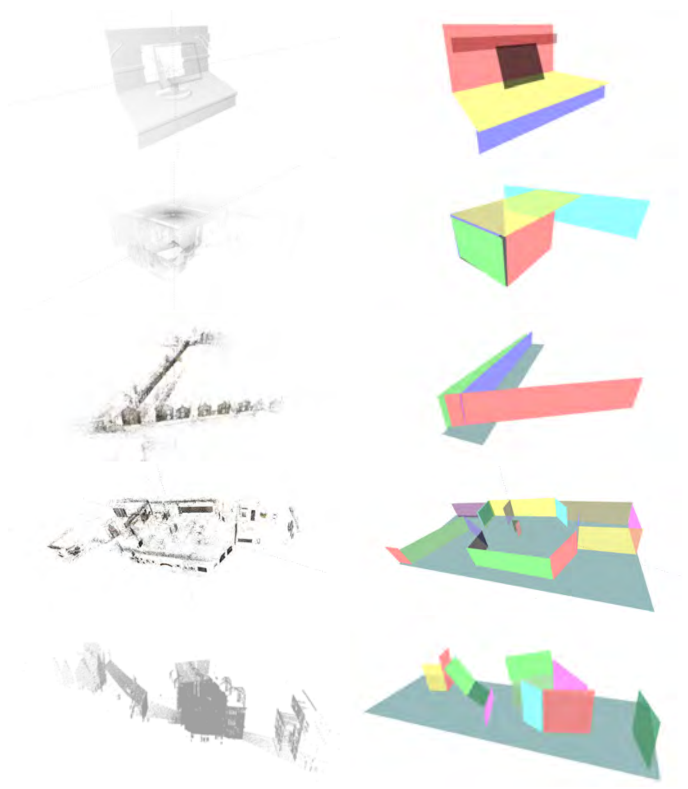

Fig. 9: Datasets used for performance comparison. Point clouds (left) and the most representative planes detected by our technique (right). From top to bottom, the datasets are: *Computer* (from [36]), *Room*, *Utrecht*, *Museum*, and *Bremen*. Their numbers of samples are shown in Table 1. The detected planes were resized for better visualization. For all datasets, the accumulator discretization was obtained using $\phi_{max} = 30$ and $\rho_{max} = 300$. The threshold s_{ms} was set to 30. s_{level} was chosen for each point cloud as 2, 4, 5, 6 and 7 (top to bottom), as at these levels the ratio between the sizes of the octree cells and the sizes of the planar patches inside them is approximately constant, which lends to good detection accuracy. These levels can be detected automatically by checking the sample variances of the detected planes.

While 3-D KHT and RANSAC use the same number of parameters (less than RHT), the 3-D KHT is less dependent on them. This is because our approach performs adaptive clustering based on relative measurements of the samples' variances, instead of using specific thresholds. Since the running times of the these algorithms are affected by the selected parameters, we chose values that optimize the execution times of each individual algorithm. Fig. 10 compares the planes detected by the three techniques on two datasets. The results are similar, but our technique is significantly faster (Table 2).

Rotating the input point cloud may result in a different sample distribution in the octree cells. This, however, should have no impact in the detection of large planes. The detection of small planar patches requires a minimum number of samples in a cell (*i.e.*, s_{ms}), and might be affected positively or negatively by the rotation (similar to the original point cloud case).

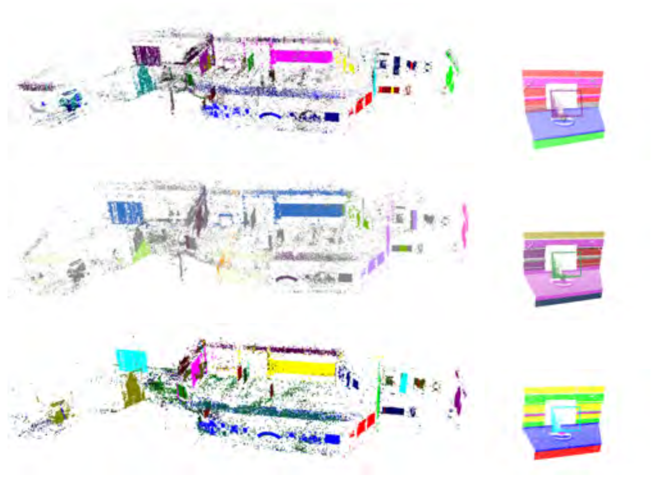

Fig. 10: Plane detection for two datasets using our technique (top), RHT (middle), and RANSAC (bottom). The results are similar, but our technique is significantly faster.

Limitations: Like in all other Hough-transform techniques for plane detection, the position and orientation of the detected planes is constrained by the accumulator discretization. This can be improved by refitting the detected planes using only the inlier samples. If the amount of noise inside the octree nodes is considerably high, our approach might not be able to detect a plane there. This would be a restriction for all techniques, Also, if the samples in an octree node are left-overs from its neighbors, our technique may fit a spurious plane through these samples. However, according to our experience, starting the approximate-coplanarity check after the third level of the octree subdivision tends to avoid this problem.

5. Conclusion and Future Work

We have presented an $O(n \log n)$ Hough-transform technique to perform deterministic plane detection in unorganized point clouds. Our approach uses a fast and robust algorithm to segment clusters of approximately coplanar samples, and casts votes for individual clusters, instead of for individual samples, on a spherical accumulator. For this, we use a trivariate Gaussian kernel that models the uncertainty about the position and orientation of the plane represented by the cluster.

While previous approaches for plane detection have basically resorted to randomly selecting a subset of the samples as a way to reduce execution time, we have undertaken the more fundamental strategy of designing an efficient algorithm with lower asymptotic cost.

Probabilistic approaches are good at finding the first few best planes. However, as the points that lie on these planes are removed, the amount of noise relative to the number of left samples tend to increase. Thus, the odds of finding additional relevant planes in the resulting point cloud tend to decrease. In contrast, our approach scans the entire point cloud without removing partial information, thus keeping the inliners/outliers ratio constant.

Our experiments have shown that our approach is several orders of magnitude faster than existing (non-deterministic) techniques for plane detection in point clouds, such as RHT and RANSAC, and scales better with the size of the datasets. It is also robust to noise and to irregularly-distributed samples. As such, it has the potential to enable a new range of applications that require fast detection of planar features on large datasets.

Our approach can be further optimized using a more efficient subdivision procedure. The use of concurrency control mechanisms for accessing the accumulator would allow voting to be performed in parallel.

Acknowledgments

This work was sponsored by CNPq-Brazil grants 131002/2012-0, 482271/2012-4, and 308936/2010-8, and FAPERGS PQG 10/1322-0.

References

[1] G. Vosselman, E. Dijkman, 3d building model reconstruction from point clouds and ground plans, Int. Arch. of Photogrammetry and Remote Sensing (2001) 37–43.

[2] R. Kaucic, R. Hartley, N. Dano, Plane-based projective reconstruction, in: Proceedings of Eighth IEEE International Conference on Computer Vision, Vol. 1, 2001, pp. 420–427 vol.1.

[3] F. Tarsha-Kurdi, T. Landes, P. Grussenmeyer, Hough-transform and extended ransac algorithms for automatic detection of 3d building roof planes from lidar data, International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, ISPRS 3 (2007) 407–412.

[4] H. Huang, C. Brenner, M. Sester, 3d building roof reconstruction from point clouds via generative models, in: Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS '11, ACM, 2011, pp. 16–24.

[5] H. Fuchs, Z. M. Kedem, S. P. Uselton, Optimal surface reconstruction from planar contours, Commun. ACM 20 (10) (1977) 693–702.

[6] B. Triggs, Autocalibration from planar scenes, in: Proceedings of the 5th European Conference on Computer Vision, Vol. 1 of ECCV '98, Springer-Verlag, 1998, pp. 89–105.

[7] C. A. Rothwell, A. Zisserman, D. A. Forsyth, J. L. Mundy, Planar object recognition using projective shape representation, International Journal of Computer Vision 16 (1) (1995) 57–99.

[8] M. Peternell, T. Steiner, Reconstruction of piecewise planar objects from point clouds, Computer-Aided Design 36 (2003) 333–342.

[9] G. Simon, A. W. Fitzgibbon, A. Zisserman, Markerless tracking using planar structures in the scene, in: Proceedings of IEEE and ACM International Symposium on Augmented Reality, 2000, pp. 120–128.

[10] D. Chekhlov, A. P. Gee, A. Calway, W. Mayol-Cuevas, Ninja on a plane: Automatic discovery of physical planes for augmented reality using visual slam, in: Proc. of the IEEE and ACM ISMAR, 2007, pp. 1–4.

[11] J. M. Biosca, J. L. Lerma, Unsupervised robust planar segmentation of terrestrial laser scanner point clouds based on fuzzy clustering methods, ISPRS J. of Photogrammetry and Remote Sensing 63 (1) (2008) 84–98.

[12] X. Ning, X. Zhang, Y. Wang, M. Jaeger, Segmentation of architecture shape information from 3d point cloud, in: Proc. 8th Int. Conf. Virtual Reality Continuum and Its Applications in Industry, 2009, pp. 127–132.

[13] C. Hausner, Synthexport, http://synthexport.codeplex.com/ (2010).

[14] Photosynth, Microsoft, 2008.

[15] L. A. F. Fernandes, M. M. Oliveira, Real-time line detection through an improved hough transform voting scheme, Pattern Recognition 41 (1) (2008) 299–314.

[16] P. Hough, Method and means for recognizing complex patterns (1962).

[17] R. O. Duda, P. E. Hart, Use of the hough transformation to detect lines and curves in pictures, Commun. ACM 15 (1) (1972) 11–15.

[18] N. Kiryati, Y. Eldar, A. M. Bruckstein, A probabilistic hough transform, Pattern Recogn. 24 (4) (1991) 303–316.

[19] A. Ylä-Jääski, N. Kiryati, Adaptive termination of voting in the probabilistic circular hough transform, IEEE Trans. Pattern Anal. Mach. Intell. 16 (9) (1994) 911–915.

[20] J. Matas, C. Galambos, J. Kittler, Progressive probabilistic hough transform, in: Proc. of the British Machine Vision Conf., 1998, pp. 256–265.

[21] L. Xu, E. Oja, P. Kultanen, A new curve detection method: randomized hough transform (rht), Pattern Recogn. Lett. 11 (5) (1990) 331–338.

[22] G. Vosselman, B. G. H. Gorte, G. Sithole, T. Rabbani, Recognizing structure in laser scanner point clouds, Intern. Archives of Photogrammetry, Remote Sensing and Spatial Info. Sciences, vol. 46 (2004) 33–38.

[23] U. Bauer, K. Polthier, Detection of planar regions in volume data for topology optimization, in: Proceedings of the 5th Int. Conference on Advances in Geometric Modeling and Processing, 2008, pp. 119–126.

[24] O. O. Ogundana, C. R. Coggrave, R. L. Burguete, J. M. Huntley, Automated detection of planes in 3-d point clouds using fast hough transforms, Optical Engineering 50 (5) (2011) 053609.

[25] H. H. Nguyen, J. Kim, Y. Lee, N. Ahmed, S. Lee, Accurate and fast extraction of planar surface patches from 3d point cloud, in: Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication, ACM, 2013, pp. 84:1–84:8.

[26] M. A. Fischler, R. C. Bolles, Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography, Commun. ACM 24 (6) (1981) 381–395.

[27] R. Schnabel, R. Wahl, R. Klein, Efficient ransac for point-cloud shape detection, Computer Graphics Forum 26 (2) (2007) 214–226.

[28] M. A. Fischler, R. C. Bolles, Perceptual organization and curve partitioning, IEEE Trans. Pattern Anal. Mach. Intell. 8 (1) (1986) 100–105.

[29] P. J. Besl, R. C. Jain, Segmentation through variable-order surface fitting, IEEE Trans. Pattern Anal. Mach. Intell. 10 (2) (1988) 167–192.

[30] D. S. Chen, A data-driven intermediate level feature extraction algorithm, IEEE Trans. Pattern Anal. Mach. Intell. 11 (7) (1989) 749–758.

[31] J. Poppinga, N. Vaskevicius, A. Birk, K. Pathak, Fast plane detection and polygonalization in noisy 3d range images, in: IEEE/RSJ International Conf. on Intelligent Robots and Systems, 2008, pp. 3378–3383.

[32] J.-E. Deschaud, F. Goulette, A fast and accurate plane detection algorithm for large noisy point clouds using filtered normals and voxel growing, in: Proc. 3DPVT, 2010.

[33] C. M. Shakarji, Least-squares fitting algorithms of the nist algorithm testing system, Journal of Research of the National Institute of Standards and Technology 103 (6) (1998) 633–641.

[34] L. Parratt, Probability and experimental errors in science: an elementary survey, Science Editions, Wiley, 1961.

[35] Y. L. Tong, The Multivariate Normal Distribution, Springer, 1990.

[36] D. Borrmann, J. Elseberg, K. Lingemann, A. Nüchter, The 3d hough transform for plane detection in point clouds: A review and a new accumulator design, 3D Res. 2 (2) (2011) 32:1–32:13.

[37] dlib, dlib c++ library, `http://dlib.net/` (n.d).

[38] R. B. Rusu, S. Cousins, 3d is here: Point cloud library (pcl), in: International Conference on Robotics and Automation, 2011, pp. 1–4.