Estimating Surface Normals in a Point Cloud

Surface normals are important properties of a geometric surface, and are heavily used in many areas such as computer graphics applications, to apply the correct light sources that generate shadings and other visual effects.

Given a geometric surface, it’s usually trivial to infer the direction of the normal at a certain point on the surface as the vector perpendicular to the surface in that point. However, since the point cloud datasets that we acquire represent a set of point samples on the real surface, there are two possibilities:

- obtain the underlying surface from the acquired point cloud dataset, using surface meshing techniques, and then compute the surface normals from the mesh;
- use approximations to infer the surface normals from the point cloud dataset directly.

Here the latter is used , that is, given a point cloud dataset, directly compute the surface normals at each point in the cloud.

The problem of determining the normal to a point on the surface is approximated by the problem of estimating the normal of a plane tangent to the surface, which in turn becomes a least-square plane fitting estimation problem.

The solution for estimating the surface normal is therefore reduced to an analysis of the eigenvectors and eigenvalues (or PCA – Principal Component Analysis) of a covariance matrix created from the nearest neighbors of the query point. More specifically, for each point p_i, we assemble the covariance matrix C as follows:

$$\mathcal{C} = \frac{1}{k}\sum_{i=1}^{k} \cdot (\boldsymbol{p}_i - \overline{\boldsymbol{p}}) \cdot (\boldsymbol{p}_i - \overline{\boldsymbol{p}})^T, \ \mathcal{C} \cdot \vec{\mathrm{v}_j} = \lambda_j \cdot \vec{\mathrm{v}_j}, \ j \in \{0, 1, 2\}$$

Where k is the number of point neighbors considered in the neighborhood of p_i,   represents the 3D centroid of the nearest neighbors, lambda_j is the j-th eigenvalue of the covariance matrix, and v_j the j-th eigenvector.

$\overline{\boldsymbol{p}}$

The eigenvectors v_j form an orthogonal frame, corresponding to the principal components of P_k. If $0 \le \lambda_0 \le \lambda_1 \le \lambda_2$, the eigenvector v_0 corresponding to the smallest eigenvalue λ_0 is therefore the approximation of $+n=\{n_x,n_y,n_z\}$ or $-n$.

In general, because there is no mathematical way to solve for the sign of n, the orientation of the normal computed via Principal Component Analysis (PCA) as shown above is ambiguous,and not consistently oriented over a point cloud dataset P.

The solution to this problem is trivial if the viewpoint v_p is in fact known. To orient all normals n_i consistently towards the viewpoint, they need to satisfy the equation: $n_i \cdot (v_p - p_i) > 0$.

In situations where the viewpoint information is not available, the problem is a bit more difficult to solve. One possible solution is modeling the consistency as a graph optimization problem.

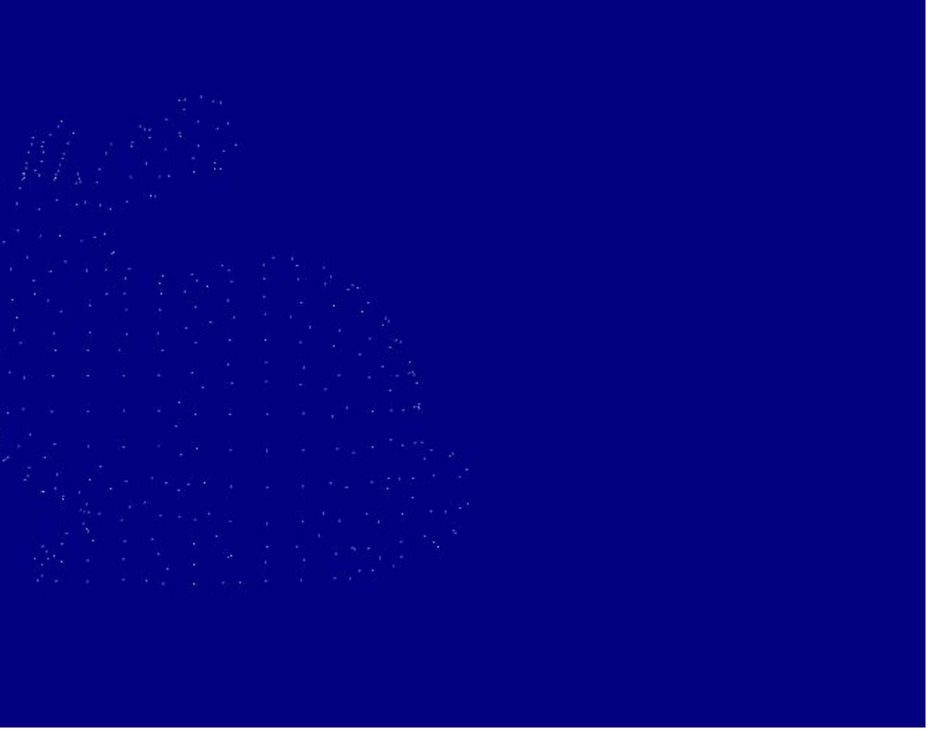
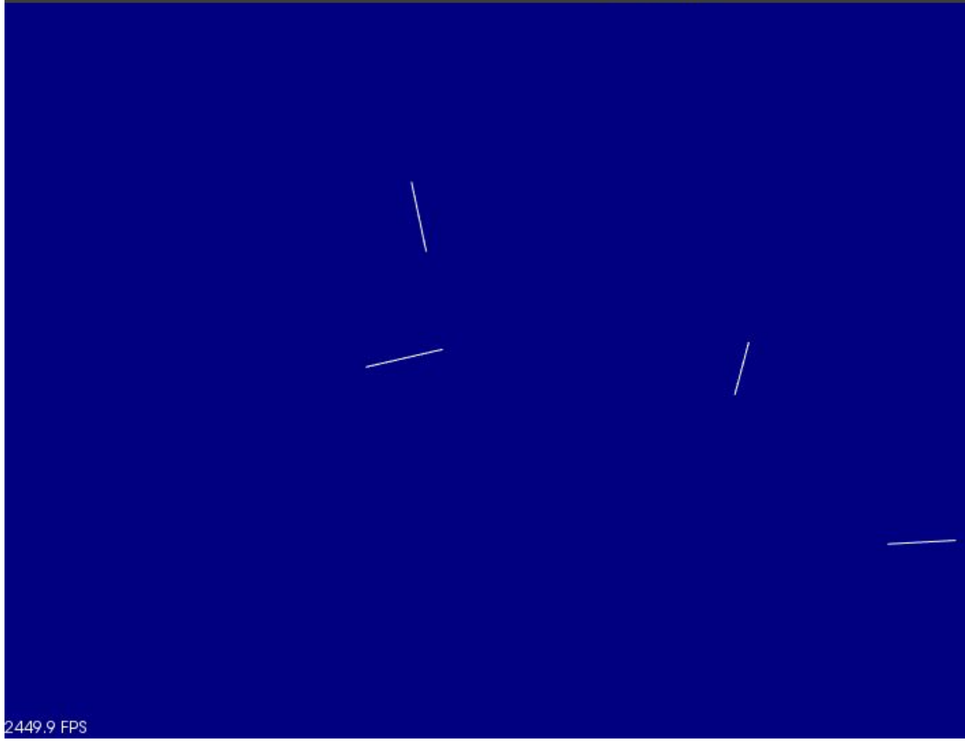
The key idea is to consider that two data points p_i and p_j belonging to a smooth surface and geometrically close, need to have their normals consistently oriented, that is:

$n_i \cdot n_j \approx 1$

In general, this assumption holds true for densely sampled datasets representing smooth surfaces. Each point $p_i \in P$ is therefore modeled as a node in a graph, with edge costs being set to the Euclidean distances between the points. This graph formulation that tends to connect neighbors is found to be the Euclidean Minimum Spanning Tree (EMST). Starting from a random node encapsulating p_i that is guaranteed to have a correct normal orientation n_i, the orientation of its adjacent nodes is propagated by traversing the minimal spanning tree of the graph and changing the sign of each normal which satisfies the equation:

$$n_i \cdot n_j < 0 \Rightarrow n_j = -n_i$$

Setting the cost of graph edges to Euclidean distances can sometimes lead to failures in the propagation orientation. A better edge cost e_g is therefore formulated as: $e_g = 1 - |n_i \cdot n_j|$. The proposed cost is non-negative and has the property that its value is small for unoriented nearly parallel normals, therefore enabling a better traversal of the minimal spanning tree for a consistent normal propagation.

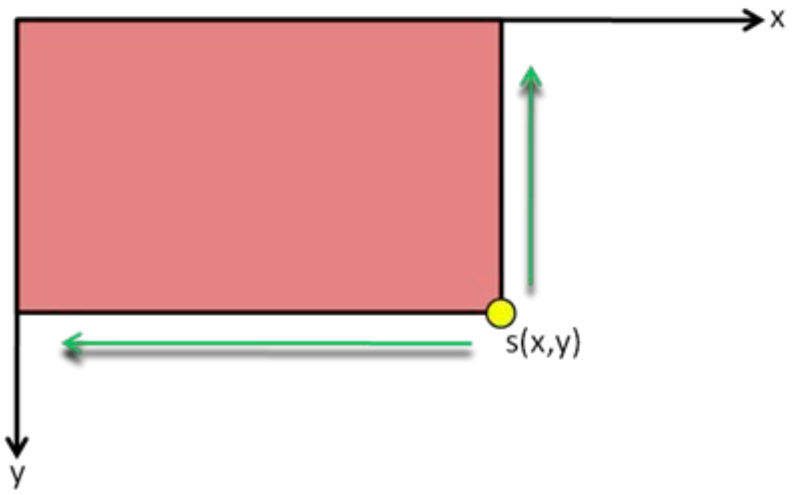
Left figure is the point cloud image of a bunny and right image is the normal at some points.

Method of Integral Images

Method of integral images can be used to speed up the calculation of covariance matrix.

The Integral Image is used as a quick and effective way of calculating the sum of values (pixel values) in a given image – or a rectangular subset of a grid (the given image).

When creating an Integral Image, we need to create a Summed Area Table. In this table, if we go Any point (x,y) in this table contains the sum of all the pixel values above, to the left and including the original pixel value of (x,y) itself:



What is really good about the Summed Area Table, is that we are actually able to construct it with only one pass over of the given image. The value in the Summed Area Table at (x,y) is simply calculated by:

$$s(x,y) = i(x,y) + s(x-1,y) + s(x,y-1) - s(x-1,y-1)$$

That is, we get the original pixel value i(x,y) from the image, and then we add the values directly above this pixel, and directly left to this pixel from the Summed Area Table at s(x-1, y) and s(x, y-1). Finally, we subtract the value directly top-left of i(x,y) from the Summed Area Table – that is, s(x-1, y-1).The task of calculating the sum of pixels in some rectangle which is a subset of the original image/matrix can be done in constant time. To do this, we use the equation,

$$i(x',y') = s(A) + s(D) - s(B) - s(C)$$

Where A,B,C,D are corners of the sub-matrix in order from top-left to bottom-right.

Now covariance matrix can be calculated using 9 integral images.

Point Feature Histogram

The goal of the PFH formulation is to encode a point’s k-neighborhood geometrical properties using a multi-dimensional histogram of values and thus use it as a local descriptor.

This highly dimensional hyperspace provides an informative signature for the feature representation, is invariant to the 6D pose of the underlying surface, and copes very well with different sampling densities or noise levels present in the neighborhood.

It attempts to capture as best as possible the sampled surface variations by taking into account all the interactions between the directions of the estimated normals. The resultant hyperspace is thus dependent on the quality of the surface normal estimations at each point.

For each point, its k neighbours within radius r are considered.

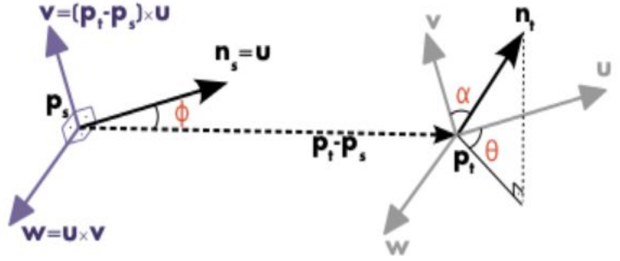
The final PFH descriptor for a point is computed as a histogram of relationships between all pairs of points in the neighborhood, and thus has a computational complexity of O(k*k)for each point.

To compute the relative difference between two points p_i and p_j and their associated normals n_i and n_j, we define a fixed coordinate frame at one of the points.

$$\mathrm{u} = \boldsymbol{n}_s$$

$$\mathrm{v} = \mathrm{u} \times \frac{(\boldsymbol{p}_t - \boldsymbol{p}_s)}{\|\boldsymbol{p}_t - \boldsymbol{p}_s\|_2}$$

$$\mathrm{w} = \mathrm{u} \times \mathrm{v}$$



Using the above uvw frame, the difference between the two normals n_s and n_t can be expressed as a set of angular features as follows:

$$\alpha = \mathrm{v} \cdot \boldsymbol{n}_t$$

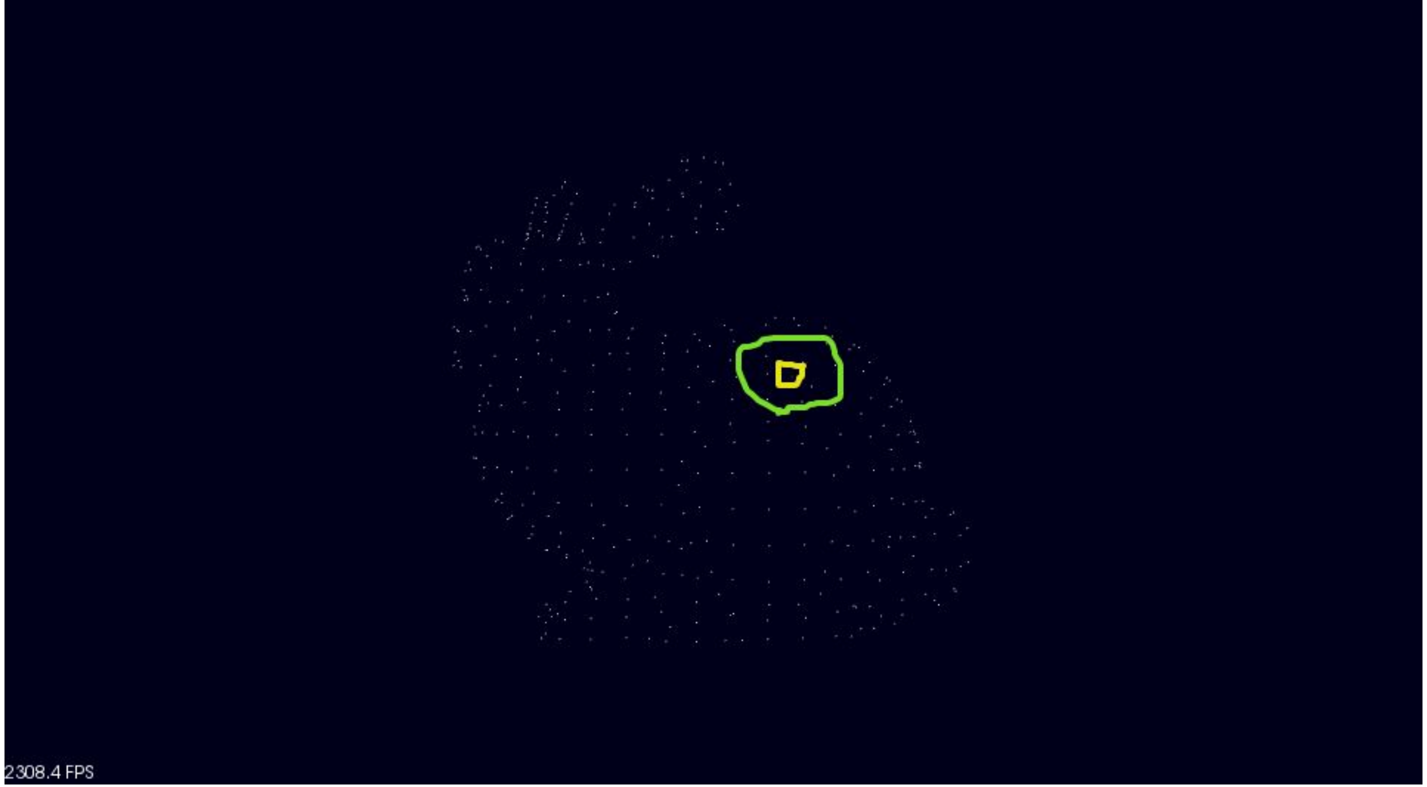
$$\phi = \mathrm{u} \cdot \frac{(\boldsymbol{p}_t - \boldsymbol{p}_s)}{d}$$

$$\theta = \arctan(\mathrm{w} \cdot \boldsymbol{n}_t, \mathrm{u} \cdot \boldsymbol{n}_t)$$

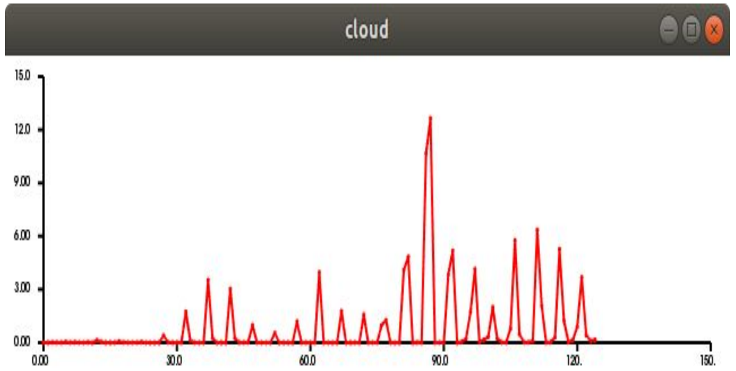
where d is the Euclidean distance between the two points ps and pt. The quadruplet (alpha, phi, theta, d) is computed for each pair of two points in k-neighborhood, therefore reducing the 12 values (3 for each x,y,z and normal) of the two points and their normals to 4.

To create the final PFH representation for the query point, the set of all quadruplets is binned into a histogram. The binning process divides each feature's value range into b subdivisions, and counts the number of occurrences in each subinterval. Since three out of the four features presented above are measure of the angles between normals, their values can easily be normalized to the same interval on the trigonometric circle. A binning example is to divide each feature interval into the same number of equal parts, and therefore create a histogram with b^4 bins in a fully correlated space. In this space, a histogram bin increment corresponds to a point having certain values for all its 4 features.

In some cases, the fourth feature, d, does not present an extreme significance for 2.5D datasets, usually acquired in robotics, as the distance between neighboring points increases from the viewpoint. Therefore, omitting d for scans where the local point density influences this feature dimension has proved to be beneficial.



This is a point cloud and the point feature histogram of the marked point is shown in the next slide.



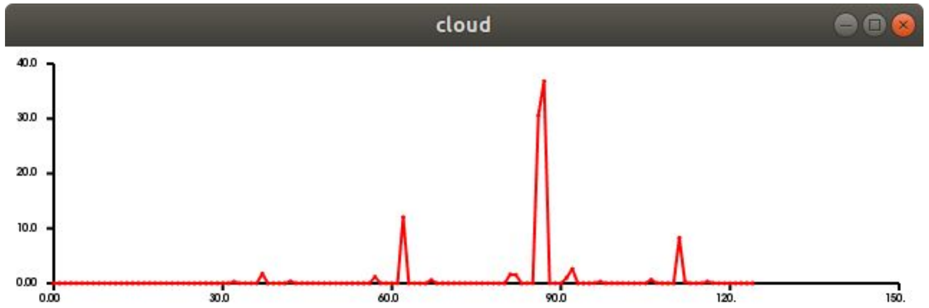


Fig. The y-axis represents the % of neighbour point pairs and the x-axis represents the value of the angular triplet features. The left figure is point feature histogram of the marked point as shown in the previous slide when neighbours within a radius of 0.1 cm are considered while the right figure is point feature histogram of the marked point as shown in the previous slide when neighbours within a radius of 0.05 cm are considered. On increasing the radius, more number of points are considered which leads to values in other bins as well resulting in reduced peak value (since its %)while the peak at around 90 is maintained.

Fast Point Feature Histogram

The theoretical computational complexity of the Point Feature Histogram for a given point cloud P with n points is O(nk^2), where k is the number of neighbors for each point p in P. For real-time or near real-time applications, the computation of Point Feature Histograms in dense point neighborhoods can represent one of the major bottlenecks.

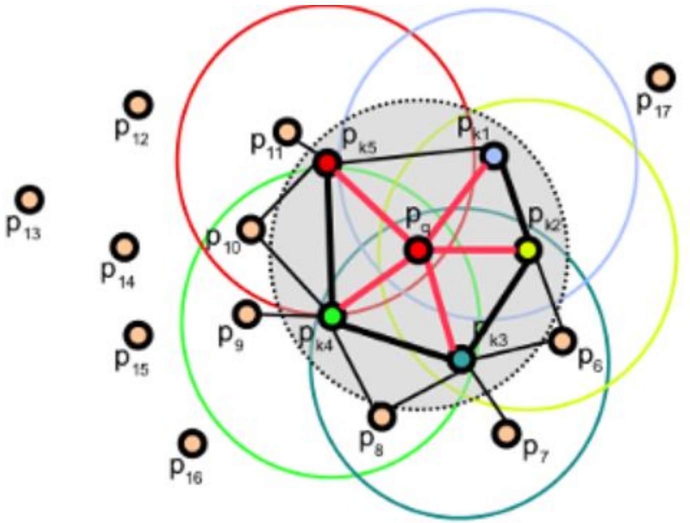
Thus Fast Point Feature Histogram was introduced which reduces the computational complexity of the algorithm to O(nk), while still retaining most of the discriminative power of the PFH.

To simplify the histogram feature computation, we proceed as follows:

- In a first step, for each query point p_q a set of tuples alpha, phi, theta between itself and its neighbors are computed as described in Point Feature Histograms (PFH) descriptors - this will be called the Simplified Point Feature Histogram (SPFH).
- In a second step, for each point its k neighbors are re-determined, and the neighboring SPFH values are used to weight the final histogram of p_q (called FPFH) as follows:

$$FPFH(\mathbf{p}_q) = SPFH(\mathbf{p}_q) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{\omega_k}\cdot SPFH(\mathbf{p}_k)$$

where the weight omega_k represents a distance between the query point p_q and a neighbor point p_k in some given metric space, thus scoring the (p_q, p_k) pair, but could just as well be selected as a different measure if necessary. To understand the importance of this weighting scheme, the figure in next slide presents the influence region diagram for a k-neighborhood set centered at p_q.

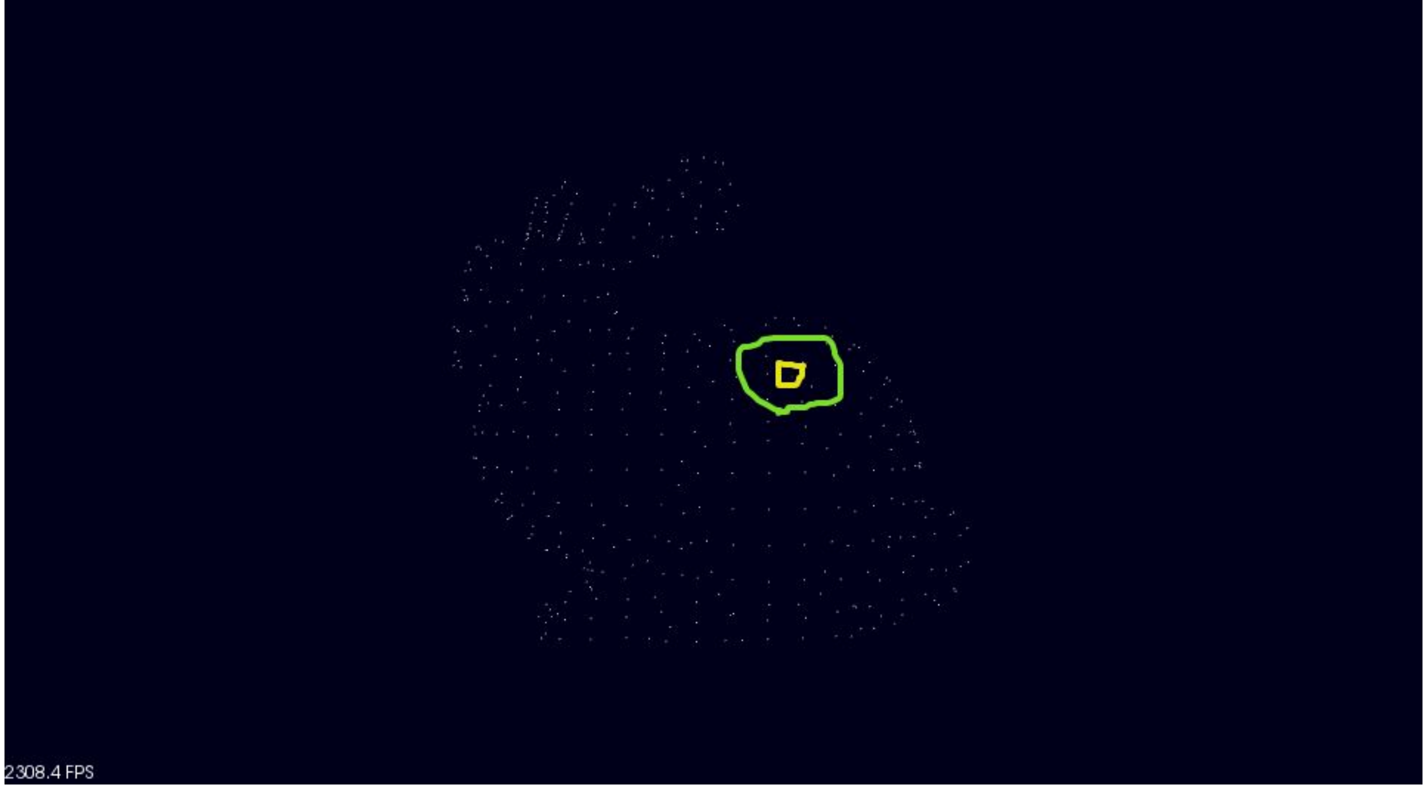


Thus, for a given query point p_q, the algorithm first estimates its SPFH values by creating pairs between itself and its neighbors (illustrated using red lines). This is repeated for all the points in the dataset, followed by a re-weighting of the SPFH values of pq using the SPFH values of its p_k neighbors, thus creating the FPFH for p_q. The extra FPFH connections, resultant due to the additional weighting scheme, are shown with black lines. As the diagram shows, some of the value pairs will be counted twice (marked with thicker lines in the figure).

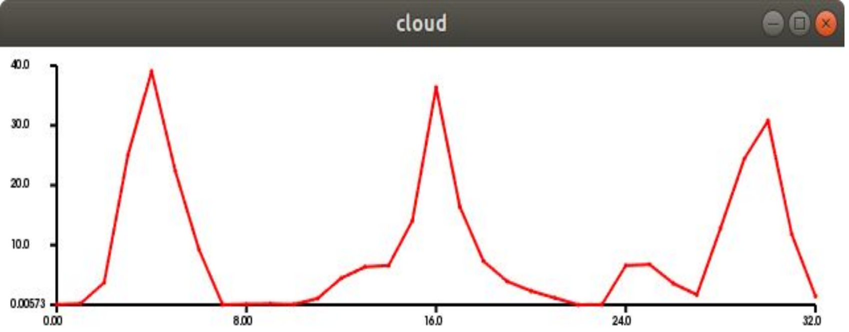
Differences between PFH and FPFH

The main differences between the PFH and FPFH formulations are summarized below:

1. the FPFH does not fully interconnect all neighbors of p_q as it can be seen from the figure, and is thus missing some value pairs which might contribute to capture the geometry around the query point;
2. the PFH models a precisely determined surface around the query point, while the FPFH includes additional point pairs outside the r radius sphere (though at most 2r away);
3. because of the re-weighting scheme, the FPFH combines SPFH values and recaptures some of the point neighboring value pairs;
4. the overall complexity of FPFH is greatly reduced, thus making possible to use it in real-time applications;
5. the resultant histogram is simplified by decorrelating the values, that is simply creating d separate feature histograms, one for each feature dimension, and concatenate them together.



This is a point cloud and the point feature histogram of the marked point is shown in the next slide.



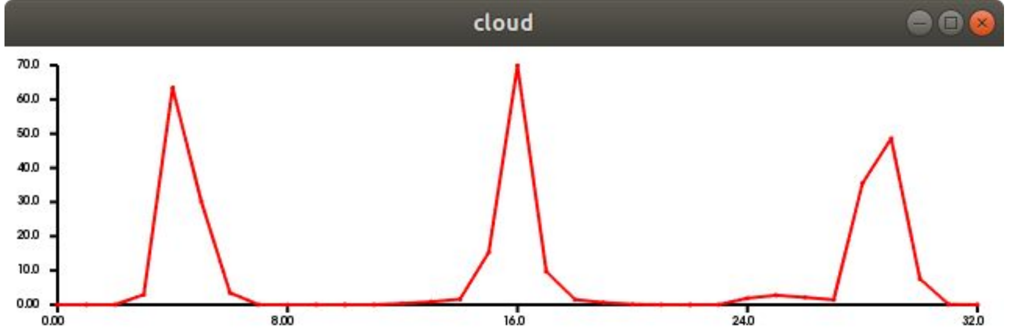


Fig. The y-axis represents the % of neighbour point pairs and the x-axis represents the value of the angular triplet features. The left figure is point feature histogram of the marked point as shown in the previous slide when neighbours within a radius of 0.1 cm are considered while the right figure is point feature histogram of the marked point as shown in the previous slide when neighbours within a radius of 0.05 cm are considered. On increasing the radius, more number of points are considered which leads to values in other bins as well resulting in reduced peak value (since its %)while the peaks still exist.

ViewPoint Feature Histogram

Viewpoint Feature Histogram descriptor is a novel representation for point clusters for the problem of Cluster (e.g., Object) Recognition and 6DOF Pose Estimation

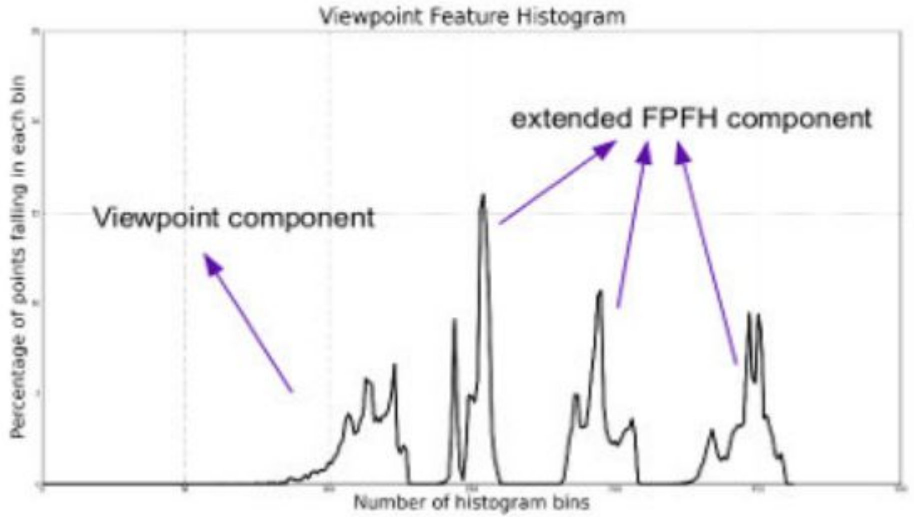
The Viewpoint Feature Histogram (or VFH) has its roots in the FPFH descriptor . VFH is made by leveraging the strong recognition results of FPFH, but to add in viewpoint variance while retaining invariance to scale.

VFH extends the FPFH to be estimated for the entire object cluster and computes additional statistics between the viewpoint direction and the normals estimated at each point by mixing the viewpoint direction directly into the relative normal angle calculation in the FPFH.

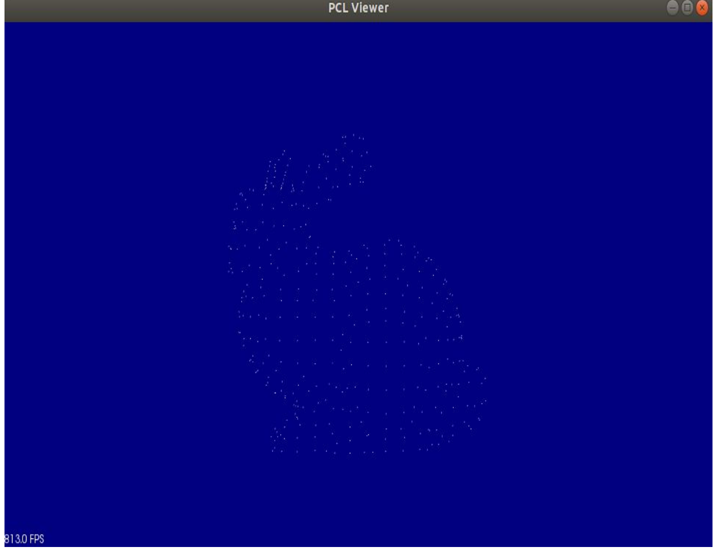
The viewpoint component is computed by collecting a histogram of the angles that the viewpoint direction makes with each normal. Note, we do not mean the view angle to each normal as this would not be scale invariant, but instead we mean the angle between the central viewpoint direction translated to each normal. The second component measures the relative pan, tilt and yaw angles as described in Fast Point Feature Histograms (FPFH) descriptors but now measured between the viewpoint direction at the central point and each of the normals on the surface.

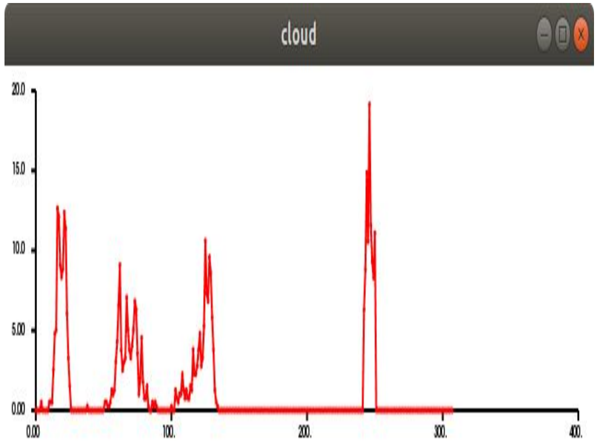
The new assembled feature is therefore called the Viewpoint Feature Histogram (VFH). The figure below presents this idea with the new feature consisting of two parts:

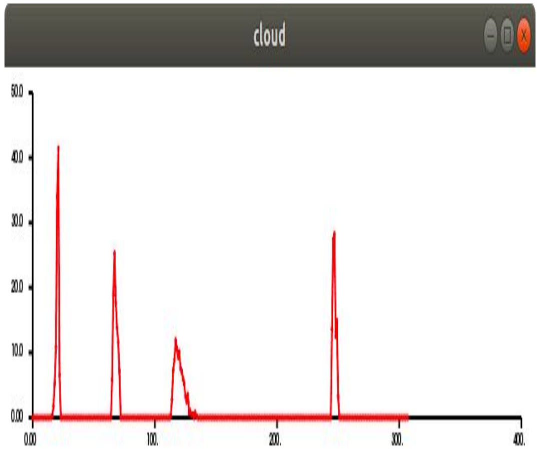
1. a viewpoint direction component and
2. a surface shape component comprised of an extended FPFH



The major difference between the PFH/FPFH descriptors and VFH, is that for a given point cloud dataset, only a single global VFH descriptor will be estimated, while the resultant PFH/FPFH data will have the same number of entries as the number of points in the cloud(one feature histogram per point).







Leftmost figure is the point cloud of a bunny, the 2 histograms from left to right are the VFH of the bunny pcd in order of increasing radius for search of neighbours.

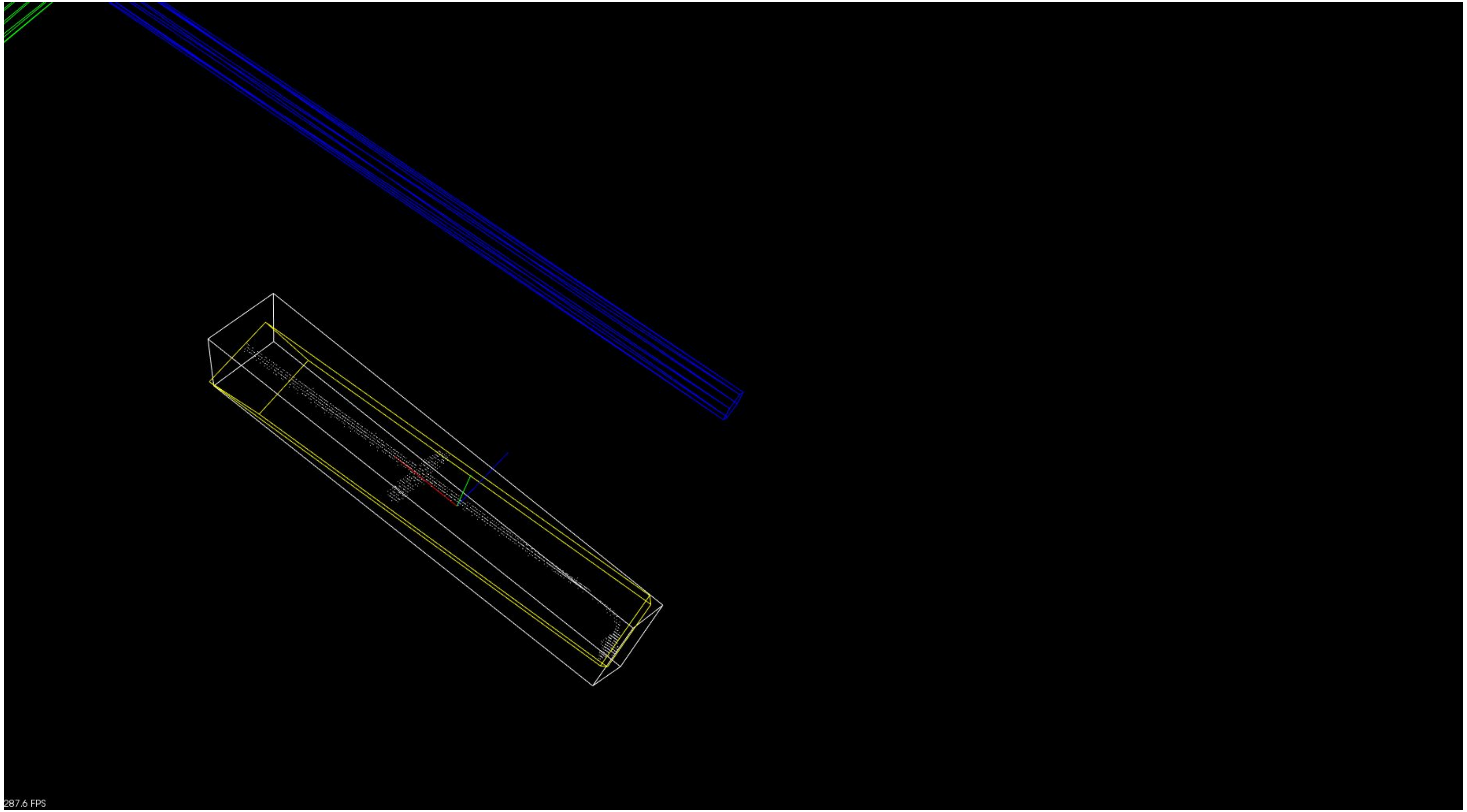
Moment of inertia and eccentricity based descriptors

It is a descriptor based on eccentricity and moment of inertia.

The idea goes like this:

First of all the covariance matrix of the point cloud is calculated and its eigenvalues and vectors are extracted. You can consider that the resultant eigen vectors are normalized and always form the right-handed coordinate system (major eigen vector represents X-axis and the minor vector represents Z-axis). On the next step the iteration process takes place. On each iteration major eigen vector is rotated. Rotation order is always the same and is performed around the other eigen vectors, this provides the invariance to rotation of the point cloud. Henceforth, we will refer to this rotated major vector as current axis.

For every current axis moment of inertia is calculated. Moreover, current axis is also used for eccentricity calculation. For this reason current vector is treated as normal vector of the plane and the input cloud is projected onto it. After that eccentricity is calculated for the obtained projection.



Here AABB is yellow, OBB is greyish white. You can also see the eigen vectors(the smaller three axes).

Normal Aligned Radial Features

The Normal Aligned Radial Feature (NARF) keypoint detector has two major characteristics:

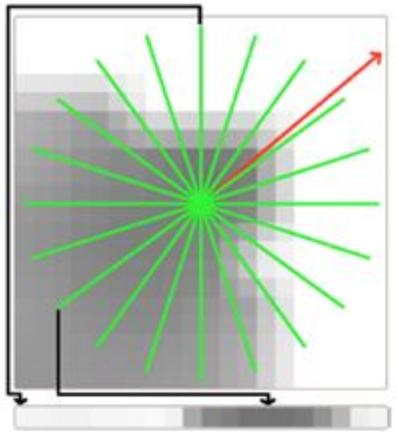
NARF extracts keypoints in areas where the direct underlying surface is stable and the neighborhood contains major surface changes.This causes NARF keypoints to be located in the local environment of significant geometric structures and not directly on them.According to the authors this characteristic leads to a more robust point descriptor computation as the normal at that point can be computed more robustly.

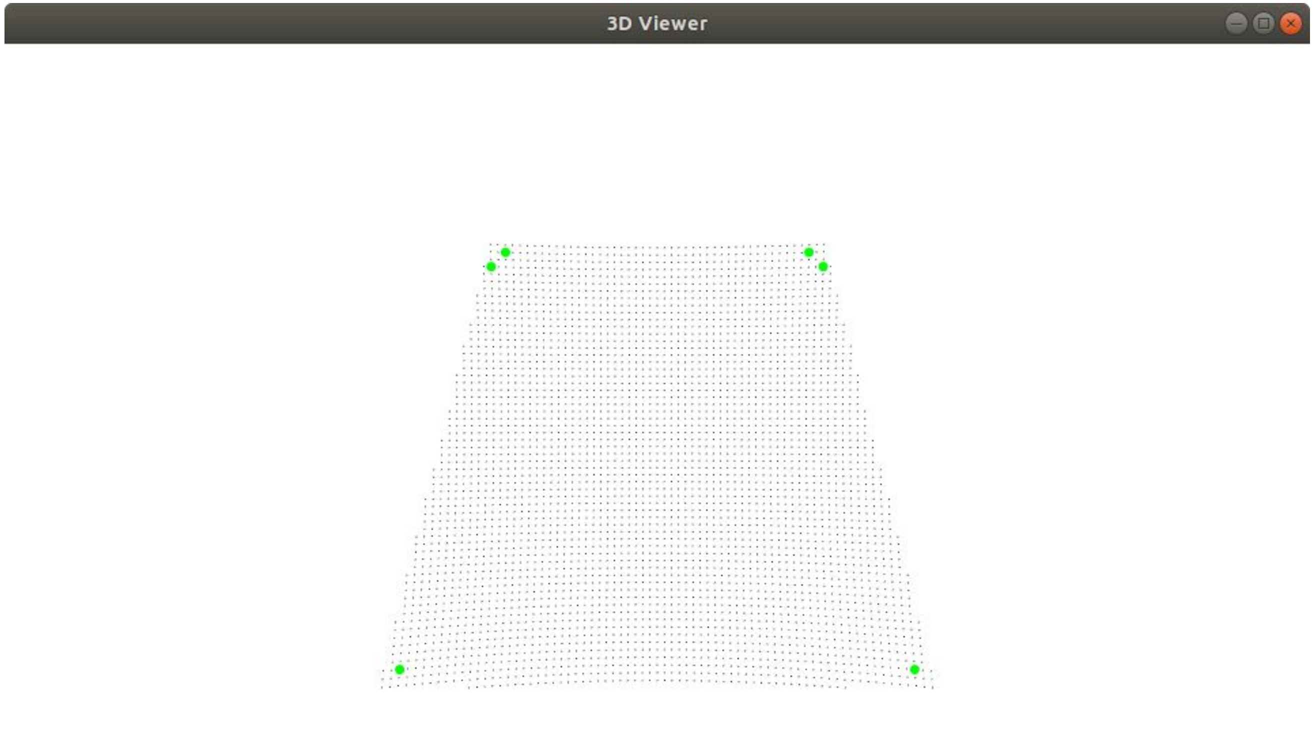
NARF takes object borders into account, which arise from view dependent non-continuous transitions from the foreground to the background.Thus, the silhouette of an object has a profound influence on the resulting keypoints.

The point cloud is transformed into a range image to perform a heuristic-based detection of object borders. The range values within a local neighborhood of size s around every image point p are ordered by their 3D distances to p.

How it works:

- Iterate over all interest points in the range image *RI*.
- For each point *Pi* create a small image patch by looking at it along its normal. The normal is the Z-axis of the image patch's local coordinate system where *Pi* is at (0,0). The Y-axis is the world coordinate system Y-Axis. The X-axis aligns accordingly. All neighbours within the radius *r* around *Pi* are transferred into this local coordinate system.
- A star pattern with *n* beams is projected on the image patch. For each beam a score in [-0.5,0.5] is calculated. Beams have a high score if there are lots of intensity changes in the cells lying under the beam. This is calculated by comparing each cell with the next adjacent one. Additionally cells closer to the center contribute to the score with a higher weight (2 in the middle, 1 at the edge).
- Finally the dominant orientation of the patch is calculated to make it invariant against rotations around the normal.





There is a rectangular point cloud in space whose NARF keypoints are detected at corners

References

- http://pointclouds.org/documentation
- http://mediatum.ub.tum.de/doc/800632/941254.pdf
- https://github.com/PointCloudLibrary/pcl/wiki/Overview-and-Comparison-of-Features
- https://www.willowgarage.com/sites/default/files/icra2011_3dfeatures.pdf