Organized Segmentation
Alex Trevor, Georgia Institute of Technology
PCL TUTORIAL @ICRA’13
Overview

• Motivation
• Connected Component Algorithm
• Planar Segmentation & Refinement
• Euclidean Clustering
• Timing Results
• Applications
• Demos (Time Permitting)
• Q & A
Segmentation is used for many applications
- Object Detection & Recognition
- Semantic Mapping
- Obstacle avoidance

Goal is to make a fast, general approach focused on RGB-D data

We provide an open-source implementation in PCL
Motivation

- RANSAC takes a lot of time, which is okay for scanning lasers, but not for RGB-D sensors
- Convex hulls poorly represent the shape of many surfaces
- Key idea: we can exploit the organized structure to solve both issues
Organized Point Clouds

- RGB-D sensors can produce organized point clouds by back-projecting RGB onto the depth image.
- Result is an image-like structure, but with additional information channels such as X, Y, Z coordinates.
- Neighboring pixels can be accessed in constant time.
- We can augment this with other densely computed features such as surface normals.
- Key advantage is that keeping NaN points enables us to maintain organization.

**Monochrome Image Point:**
\[ p = \{i\} \]

**RGB Image Point:**
\[ p = \{r, g, b\} \]

**RGBD Point:**
\[ p = \{r, g, b, x, y, z\} \]

**RGBD+Normal Point:**
\[ p = \{r, g, b, x, y, z, n_x, n_y, n_z\} \]
Normal Estimation

- Surface normals can be useful for many segmentation tasks
- We use Integral Image Normal Estimation of Holzer et. al
- Enables real-time surface normal computation

Connected Components

- Well known algorithm from computer vision
- Can be applied to organized point cloud segmentation
- Requires a comparison function to determine if pixels should be part of the same segment or not
Connected Components

- Well known algorithm from computer vision
- Can be applied to organized point cloud segmentation
- Requires a comparison function to determine if pixels should be part of the same segment or not
Connected Components

- Well known algorithm from computer vision
- Can be applied to organized point cloud segmentation
- Requires a comparison function to determine if pixels should be part of the same segment or not
Connected Components

• Well known algorithm from computer vision

• Can be applied to organized point cloud segmentation

• Requires a comparison function to determine if pixels should be part of the same segment or not
Connected Components

- Well known algorithm from computer vision
- Can be applied to organized point cloud segmentation
- Requires a comparison function to determine if pixels should be part of the same segment or not
Connected Components

- Well known algorithm from computer vision
- Can be applied to organized point cloud segmentation
- Requires a comparison function to determine if pixels should be part of the same segment or not
Connected Components

- Well known algorithm from computer vision
- Can be applied to organized point cloud segmentation
- Requires a comparison function to determine if pixels should be part of the same segment or not
Connected Components

- Well known algorithm from computer vision
- Can be applied to organized point cloud segmentation
- Requires a comparison function to determine if pixels should be part of the same segment or not
Connected Components

- Well known algorithm from computer vision
- Can be applied to organized point cloud segmentation
- Requires a comparison function to determine if pixels should be part of the same segment or not
• An additional pass is required to merge labels in some cases

• The union-find algorithm can be used to do this efficiently
Boundary Detection

- Organized data makes it easy to find the boundary of segmented regions using standard contour tracing.
- Provides the actual boundary shape, rather than a convex shape.
- Example convex hull timing (80k pts): 0.012 sec.
- Example boundary timing (80k pts): 0.000048 sec.
Boundary Detection

- Organized data makes it easy to find the boundary of segmented regions using standard contour tracing.

- Provides the actual boundary shape, rather than a convex shape.

- Example convex hull timing (80k pts): 0.012 sec

- Example boundary timing (80k pts): 0.000048 sec
Boundary Detection

• Organized data makes it easy to find the boundary of segmented regions using standard contour tracing

• Provides the actual boundary shape, rather than a convex shape

• Example convex hull timing (80k pts): 0.012 sec

• Example boundary timing (80k pts): 0.000048 sec
Boundary Detection

- Organized data makes it easy to find the boundary of segmented regions using standard contour tracing.

- Provides the actual boundary shape, rather than a convex shape.

- Example convex hull timing (80k pts): 0.012 sec

- Example boundary timing (80k pts): 0.000048 sec
Boundary Detection

- Organized data makes it easy to find the boundary of segmented regions using standard contour tracing.

- Provides the actual boundary shape, rather than a convex shape.

- Example convex hull timing (80k pts): 0.012 sec.

- Example boundary timing (80k pts): 0.000048 sec.
Boundary Detection

- Organized data makes it easy to find the boundary of segmented regions using standard contour tracing

- Provides the actual boundary shape, rather than a convex shape

- Example convex hull timing (80k pts): 0.012 sec

- Example boundary timing (80k pts): 0.000048 sec
Boundary Detection

- Organized data makes it easy to find the boundary of segmented regions using standard contour tracing.
- Provides the actual boundary shape, rather than a convex shape.
- Example convex hull timing (80k pts): 0.012 sec.
- Example boundary timing (80k pts): 0.000048 sec.
Boundary Detection

- Organized data makes it easy to find the boundary of segmented regions using standard contour tracing.

- Provides the actual boundary shape, rather than a convex shape.

- Example convex hull timing (80k pts): 0.012 sec

- Example boundary timing (80k pts): 0.000048 sec
Boundary Detection

- Organized data makes it easy to find the boundary of segmented regions using standard contour tracing

- Provides the actual boundary shape, rather than a convex shape

- Example convex hull timing (80k pts): 0.012 sec

- Example boundary timing (80k pts): 0.000048 sec
Comparison Functions

• Different comparison functions can be used for different segmentation tasks
• Allows us to easily modify the approach for different applications
• These comparison functions have the form:

\[ C(P(x_1, y_1), P(x_2, y_2)) = \begin{cases} 
  \text{true if similar} \\
  \text{false otherwise}
\end{cases} \]
Planar Segmentation

Point Representation:
\[ p = \{x, y, z, n_x, n_y, n_z, n_d\} \]
\[ n_d = \{x, y, z\} \cdot \{n_x, n_y, n_z\} \]

Distance Metrics:
\[ \text{dist}_{normal}(p_1, p_2) = p_{1n} \cdot p_{2n} \]
\[ \text{dist}_{range}(p_1, p_2) = |p_{1n_d} - p_{2n_d}| \]

Comparison Function:
\[ C(p_1, p_2) = \begin{cases} 
\text{true} & \text{if } (\text{dist}_{normal} < \text{thresh}_{normal}) \\
& \text{&& } (\text{dist}_{range} < \text{thresh}_{range}) \\
\text{false} & \text{otherwise} 
\end{cases} \]
/** brief Compare points at two indices by their plane equations. True if the angle between the normals is less than the angular threshold, and the difference between the d component of the normals is less than distance threshold, else false 
* \param idx1 The first index for the comparison 
* \param idx2 The second index for the comparison 
*/

virtual bool compare (int idx1, int idx2) const
{
    float threshold = distance_threshold_; 
    if (depth_dependent_)
    {
        Eigen::Vector3f vec = input_->points[idx1].getVector3fMap ();

        float z = vec.dot (z_axis_); 
        threshold *= z * z; 
    }
    return (fabs ((*plane_coeff_d_)[idx1] - (*plane_coeff_d_)[idx2]) < threshold) 
        && (normals_->points[idx1].getNormalVector3fMap ().dot (normals_->points[idx2].getNormalVector3fMap () ) > angular_threshold_ ) ;
}
```cpp
// Segment planes
pcl::OrganizedMultiPlaneSegmentation<PointT, pcl::Normal, pcl::Label> mps;
mps.setMinInliers (10000);
mps.setAngularThreshold (0.017453 * 2.0); // 2 degrees
mps.setDistanceThreshold (0.02); // 2cm
mps.setInputNormals (normal_cloud);
mps.setInputCloud (cloud);
std::vector<pcl::PlanarRegion<PointT>> regions;
mps.segmentAndRefine (regions);

for (size_t i = 0; i < regions.size (); i++)
{
    Eigen::Vector3f centroid = regions[i].getCentroid ();
    Eigen::Vector4f model = regions[i].getCoefficients ();
    pcl::PointCloud<PointT> boundary_cloud;
    boundary_cloud.points = regions[i].getContour ();
    printf("Centroid: (%f, %f, %f)\n Coefficients: (%f, %f, %f, %f)\n Inliers: %d\n",
        centroid[0], centroid[1], centroid[2],
        model[0], model[1], model[2], model[3],
        boundary_cloud.points.size ());
}
```

- See PCL’s Organized Segmentation Demo for more:
  pcl_organized_segmentation_demo
Refinement of planar regions

- Normals near surface edges can be noisy
- We can address this issue with an additional two passes
- Instead of comparing neighboring pixels values, we compare to the planar model of neighboring pixels
Refinement of planar regions

Distance Metric:

\[ \text{dist}_{\text{ptp}}(p, eqn) = |n_x * x + n_y * y + n_z * z + n_d| \]

Comparison Function:

\[ C(p_1, p_2) = \begin{cases} \text{false} \text{ if } (p_1 \notin \text{refine_labels} \\
\text{&& } p_2 \notin \text{refine_labels} \\
\text{|| } \text{dist}_{\text{ptp}}(p_1, eqn(p_2)) > \text{thresh}_{\text{ptp}} \\
\text{true otherwise} \end{cases} \]

- Refinement requires two additional passes to extend regions
/** \ brief Compare two neighboring points 
* \param[in] idx1 The index of the first point.
* \param[in] idx2 The index of the second point.
*/

virtual bool
compare (int idx1, int idx2) const
{
    int current_label = labels_\->points[idx1].label;
    int next_label = labels_\->points[idx2].label;

    if (!(\*refine_labels_)[current_label] && !(\*refine_labels_)[next_label])
        return (false);

    const pcl::ModelCoefficients& model_coeff = (*models_)[(*label_to_model_)[current_label]];

    PointT pt = input_\->points[idx2];
    double ptp_dist = fabs (model_coeff.values[0] * pt.x +
                           model_coeff.values[1] * pt.y +
                           model_coeff.values[2] * pt.z +
                           model_coeff.values[3]);

    // depth dependent
    float threshold = distance_threshold_;
    if (depth_dependent_)
    {
        Eigen::Vector3f vec = input_\->points[idx1].getVector3fMap ();

        float z = vec.dot (z_axis_);
        threshold *= z * z;
    }

    return (ptp_dist < threshold);
}
Using Color Information

- We can easily segment on color and geometric information at little additional cost.
- We can use the same comparison as we did for planes, plus an additional constraint that neighboring pixels have similar color.
Euclidean Cluster Extraction

• Use planar regions extracted in the first pass as a mask
• Simply compare euclidean distance between neighboring points

\[
C(P_1, P_2) = \begin{cases} 
\text{false} & \text{if } (L(P_1) \in \text{exclude\_labels}) \\
\quad \text{and } (L(P_2) \in \text{exclude\_labels}) \\
\quad \text{and } d_{\text{euclidean}}(P_1, P_2) > d_{\text{thresh}} \\
\text{true} & \text{otherwise}
\end{cases}
\]
// Segment Objects
pcl::PointCloud<PointT>::CloudVectorType clusters;

if (use_clustering_ && regions.size() > 0)
{
    std::vector<bool> plane_labels;
    plane_labels.resize(label_indices.size(), false);
    for (size_t i = 0; i < label_indices.size(); i++)
    {
        if (label_indices[i].indices.size() > 10000)
        {
            plane_labels[i] = true;
        }
    }
}

euclidean_cluster_comparator_->setInputCloud(cloud);
euclidean_cluster_comparator_->setLabelRegionIndices(labels);
euclidean_cluster_comparator_->setExcludeLabelIndices(plane_labels);
euclidean_cluster_comparator_->_setDistanceThreshold(0.01f, false);

pcl::PointCloud<pcl::Label> euclidean_labels;
std::vector<pcl::PointIndices> euclidean_label_indices;
pcl::OrganizedConnectedComponentSegmentation<PointT, pcl::Label> euclidean_segmentation(euclidean_cluster_comparator_);
euclidean_segmentation.setInputCloud(cloud);
euclidean_segmentation.segment(euclidean_labels, euclidean_label_indices);

for (size_t i = 0; i < euclidean_label_indices.size(); i++)
{
    if (euclidean_label_indices[i].indices.size() > 10000)
    {
        pcl::PointCloud<PointT> cluster;
        pcl::copyPointCloud(*cloud, euclidean_label_indices[i].indices, cluster);
        clusters.push_back(cluster);
    }
}
Timing Results

- Timing results were computed on a 2.6 GHz Intel Core i7 CPU
- Point Clouds from the publicly available TUM RGBD Dataset were used:
  - fr1_desk
  - fr1_floor
  - fr2_pioneer_slam
### Timing Results

#### Planar Segmentation:

<table>
<thead>
<tr>
<th></th>
<th>fr1 desk</th>
<th>fr1 floor</th>
<th>fr2 pioneer_slam</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal Estimation</td>
<td>21.56 ± 2.07ms</td>
<td>22.98 ± 1.93ms</td>
<td>21.97 ± 3.91ms</td>
</tr>
<tr>
<td>Plane Segmentation</td>
<td>26.13 ± 3.19ms</td>
<td>21.28 ± 3.09ms</td>
<td>23.17 ± 5.62ms</td>
</tr>
<tr>
<td>Frame Callback</td>
<td>33.51 ± 3.13ms</td>
<td>33.62 ± 2.71ms</td>
<td>33.98 ± 4.03ms</td>
</tr>
<tr>
<td>Callback Rate</td>
<td>29.83 Hz</td>
<td>29.73 Hz</td>
<td>29.42 Hz</td>
</tr>
</tbody>
</table>

#### Planar Segmentation with refinement:

<table>
<thead>
<tr>
<th></th>
<th>fr1 desk</th>
<th>fr1 floor</th>
<th>fr2 pioneer_slam</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal Estimation</td>
<td>21.91 ± 2.36ms</td>
<td>23.99 ± 2.57ms</td>
<td>22.00 ± 3.86ms</td>
</tr>
<tr>
<td>Plane Segmentation+Refinement</td>
<td>32.55 ± 3.29ms</td>
<td>29.83 ± 3.38ms</td>
<td>29.53 ± 6.34ms</td>
</tr>
<tr>
<td>Frame Callback</td>
<td>35.79 ± 3.20ms</td>
<td>34.08 ± 3.23ms</td>
<td>35.74 ± 4.35ms</td>
</tr>
<tr>
<td>Callback Rate</td>
<td>27.93 Hz</td>
<td>29.34 Hz</td>
<td>27.97 Hz</td>
</tr>
</tbody>
</table>
• Segmentation approach has been used in a mapping application
• Planar segments from multiple images are merged in a map frame
• Technique also works on stereo data, as this is also organized
• Has been applied to ground segmentation on outdoor wide-baseline stereo
• Segmentation & Refinement steps are used
Organized Edge Extraction

- Similar organized extraction techniques can be used to extract edges
- Supports multiple edge types:
  - Occluding Edges
  - Occluded Edges
  - High Curvature Edges
- pcl_openni_organized_edge_detection