Fast Segmentation of Organized Point Cloud Data

- First step: calculate surface normals
- Use Integral Image Normal Estimation
- Takes about 70 ms per frame (covariance matrix method), or 30-40 ms (simple gradient method)
Connected Components

- Next, do connected component labeling
- Implementation allows various different comparisons between neighboring pixels
- Different comparison functions allow different types of segmentation: planes, objects, color blobs, etc
Connected Components

- Next, do connected component labeling
- Implementation allows various different comparisons between neighboring pixels
- Different comparison functions allow different types of segmentation: planes, objects, color blobs, etc
Connected Components

- Next, do connected component labeling
- Implementation allows various different comparisons between neighboring pixels
- Different comparison functions allow different types of segmentation: planes, objects, color blobs, etc
Connected Components

- Next, do connected component labeling
- Implementation allows various different comparisons between neighboring pixels
- Different comparison functions allow different types of segmentation: planes, objects, color blobs, etc
Connected Components

- Next, do connected component labeling
- Implementation allows various different comparisons between neighboring pixels
- Different comparison functions allow different types of segmentation: planes, objects, color blobs, etc
Connected Components

- Next, do connected component labeling
- Implementation allows various different comparisons between neighboring pixels
- Different comparison functions allow different types of segmentation: planes, objects, color blobs, etc
Connected Components

• Next, do connected component labeling

• Implementation allows various different comparisons between neighboring pixels

• Different comparison functions allow different types of segmentation: planes, objects, color blobs, etc
Multi Plane Segmentation

- Comparison function for plane segmentation:
  \[ n_1 \cdot n_2 < \theta_{thresh} \]
  \[ |d_1 - d_2| < d_{thresh} \]

- Fit planes to regions with > min_inliers
Boundary Detection

- Organized data makes it easy to find the boundary of segmented regions
- Trace outer contour of a region
- 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
- Trace outer contour of a region
- 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
- Trace outer contour of a region
- 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
- Trace outer contour of a region
- 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
- Trace outer contour of a region
- 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

• Trace outer contour of a region

• 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
- Trace outer contour of a region
- 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
- Trace outer contour of a region
- Example convex hull timing (80k pts): 0.012 sec
- Example boundary timing (80k pts): 0.000048 sec
Organized data makes it easy to find the boundary of segmented regions.

Trace outer contour of a region.

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

Example boundary timing (80k pts): 0.000048 sec.
Refinement of planar regions

- Normals near surface edges can be noisy
- We can address this issue with an additional pass
- Instead of comparing neighboring pixels values, we compare to the planar model of neighboring pixels
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.
Using Boundaries and Holes

- Since we have organized data, we can also notice missing / NAN data
- This can be useful for detecting small shiny objects which may not give good depth returns
Robust to Lighting Conditions

- Comparison functions that don’t use color data work fine in the dark!
Timing Improvements

• Fair comparison is not so easy due to the large number of parameters.

• For this comparison, I chose to use Iterative RANSAC as used in my previous mapping work, 1000 iterations

• Clusters RANSAC planes to separate coplanar disjoint regions (e.g. multiple tables)

• Computes convex hull, since the MPS gives a boundary

• Data set is kinect data from the “yesterday’s sushi” area, 508 frames

• Timings are reported for plane extraction + hull / boundary computation only, so this is on top of normal estimation
Timing Improvements

- MultiPlaneSegmentation, PlaneCoefficientComparator, SegmentAndRefine:
  - mean = 0.0332 seconds, stddev = 0.0023
- RANSAC (1000 iterations, dominant plane, 2.5cm voxelized cloud):
  - mean = 0.32 seconds, stddev = 0.34
- RANSAC (1000 iterations, all planes in the scene, 2.5cm voxelized cloud):
  - mean = 1.52 seconds, stddev = 1.79
- RANSAC (1000 iterations, all planes in the scene, 1cm voxelized cloud):
  - mean = 6.18 seconds, stddev = 6.62
Code Example

```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!
Multi-Frame Segmentation: Planar Polygon Fusion

• Considering multiple frames is more interesting than single views

• Given a set of frames and sensor locations (from odom, AMCL, etc) we can fuse planar surfaces across views