Downsampling

- Downsampling enable faster work
- Methods can be divided into …
  - Voxel Grid-based method
  - Farthest point sampling(FPS)-based method
  - Normal Space sampling-based method

Voxel Grid Downsampling

- Using regular voxel grid to create a uniformly downsampled pointcloud
- Each voxel retains single representative value of bucketed pointcloud
  - Center of voxel
  - Centroid of points
  - Closest point to centroid
- Both PCL and Open3D utilize centroid

Voxel Grid Downsampling in code

Open3D

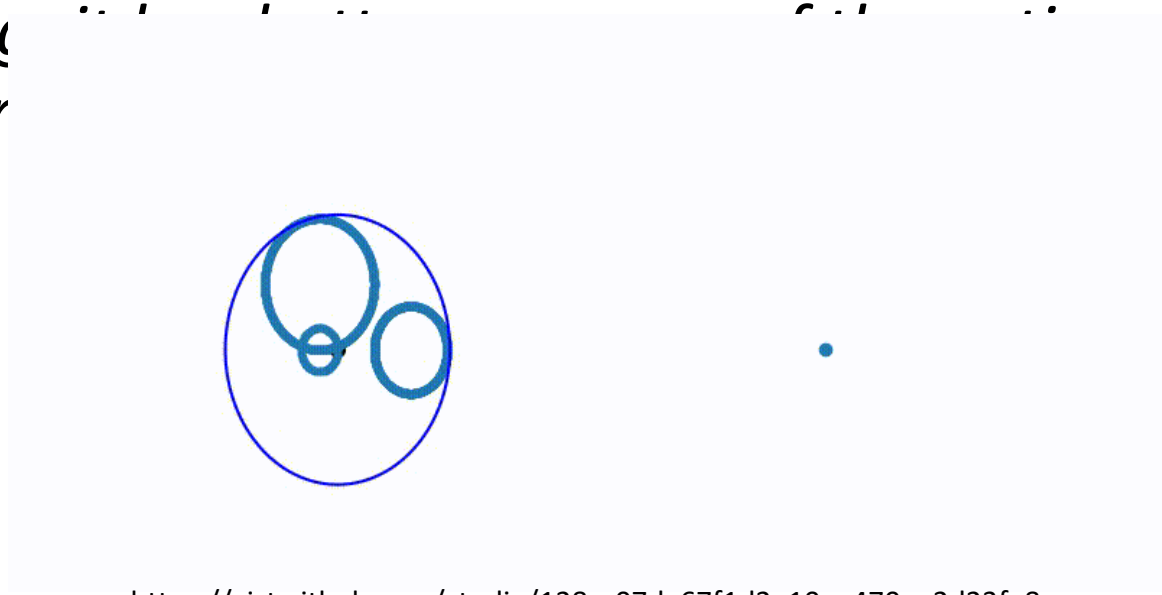
```python
import open3d as o3d
pcd = o3d.io.read_point_cloud("PATH/TO/POINTCLOUD.pcd")
downpcd = pcd.voxel_down_sample(voxel_size=0.05)
```

PCL

```cpp
#include <iostream>
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
#include <pcl/filters/voxel_grid.h>

int main()
{
    pcl::PCLPointCloud2::Ptr cloud (new pcl::PCLPointCloud2 ());
    pcl::PCLPointCloud2::Ptr cloud_filtered (new pcl::PCLPointCloud2 ());

    pcl::PCDReader reader;
    reader.read ("PATH/TO/PCD", *cloud);

    // Create the filtering object
    pcl::VoxelGrid<pcl::PCLPointCloud2> sor;
    sor.setInputCloud (cloud);
    sor.setLeafSize (0.01f, 0.01f, 0.01f);
    sor.filter (*cloud_filtered);

    return 0;
}
```

Farthest Point Sampling (FPS)

- Given pointcloud, choose a subset of points such that new point i is the most distant point from $(i-1)$ points
- Utilized in PointNet++, in sampling layer
- *Compared with random sampling*



https://gist.github.com/ctralie/128cc07da67f1d2e10ea470ee2d23fe8

Normal Space Sampling

- Introduced in “Efficient Variants of the ICP Algorithm (01’ 3DIMPVT)”
- Construct a histogram of surface normal, and uniformly sample points from all bins until desired number of point is reached.

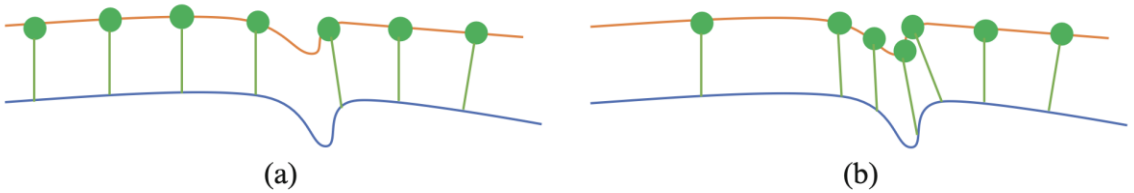


Fig. 2.3 Comparison between uniform sampling and NSS. (**a**) Uniform sampling. (**b**) NSS

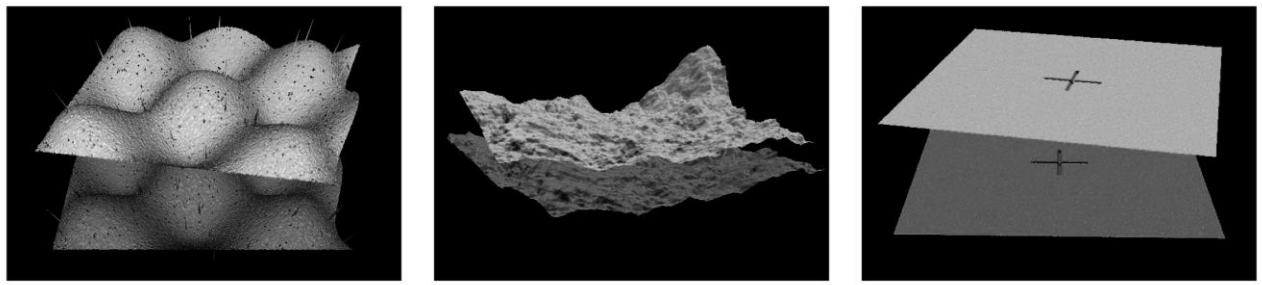
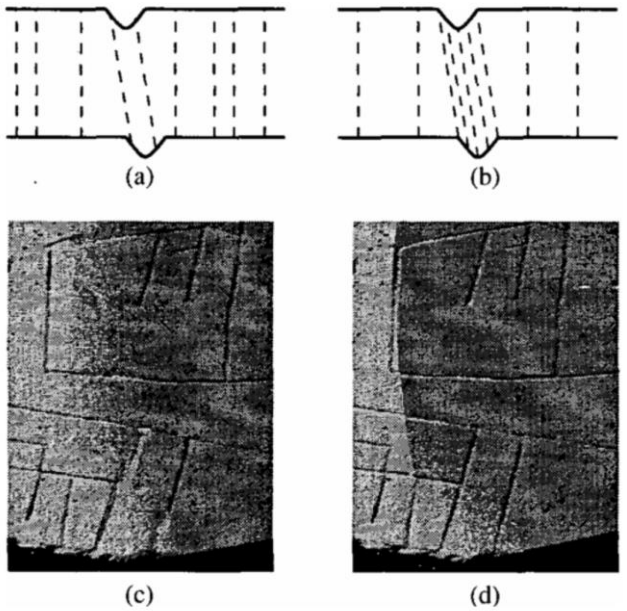
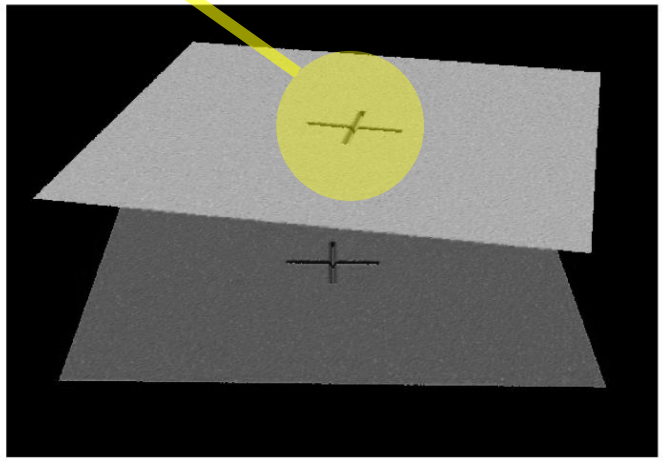


Figure 1: Test scenes used throughout this paper.



Normal Space Sampling

- “choosing points such that the distribution of normals among selected points is as large as possible. The motivation for this strategy is the observation that for certain kinds of scenes (such as our “incised plane” data set) **small features of the model are vital to determining the correct alignment**. A strategy such as *random sampling* will often select only a few samples in these features, which leads to an inability to determine certain components of the correct rigid-body transformation. Thus, one way to improve the chances that enough constraints are present to determine all the components of the transformation is to **bucket the points according to the position of the normals in angular space, then sample as uniformly as possible across the buckets**.”



(c) Incised plane

Finding Surface Normal

- Find neighbor points and create a 3x3 covariance matrix
- apply PCA to it, and select eigenvectors with the smallest eigenvalue

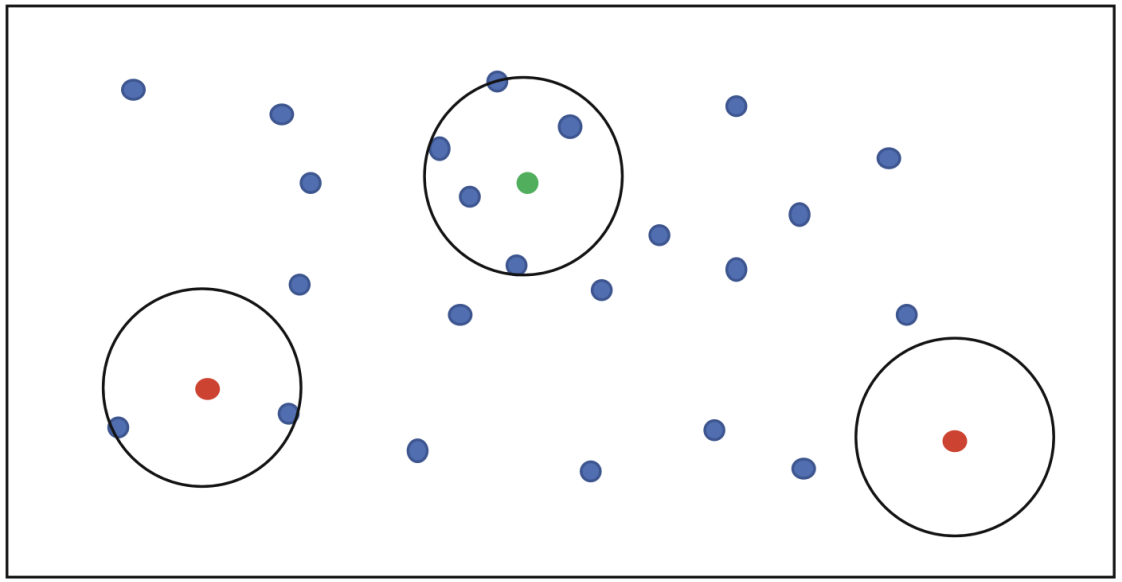
$$\mathcal{C} = \frac{1}{k}\sum_{i=1}^{k} \cdot (\boldsymbol{p}_i - \overline{\boldsymbol{p}}) \cdot (\boldsymbol{p}_i - \overline{\boldsymbol{p}})^T,\ \mathcal{C} \cdot \vec{\mathbf{v}_j} = \lambda_j \cdot \vec{\mathbf{v}_j},\ j \in \{0, 1, 2\}$$

Noise Removal

- Radius Outlier Removal
- Statistical Outlier Removal

Radius Outlier Removal

- For each point, find its neighbor inside radius (sphere)
- If the number of neighbor is lower than the threshold, it is outlier
- Available both in PCL and Open3D



Radius Outlier Removal

Open3D

```python
import open3d as o3d
pcd = o3d.io.read_point_cloud("PATH/TO/POINTCLOUD.pcd")
cl, ind = pcd.remove_radius_outlier(nb_points=16, radius=0.05)
```

PCL

```cpp
#include <iostream>
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
#include <pcl/filters/voxel_grid.h>

int main()
{
    pcl::PCLPointCloud2::Ptr cloud (new pcl::PCLPointCloud2 ());
    pcl::PCLPointCloud2::Ptr cloud_filtered (new pcl::PCLPointCloud2 ());

    pcl::PCDReader reader;
    reader.read ("PATH/TO/PCD", *cloud);

    pcl::RadiusOutlierRemoval<pcl::PointXYZ> outrem;
    // build the filter
    outrem.setInputCloud(cloud);
    outrem.setRadiusSearch(0.8);
    outrem.setMinNeighborsInRadius (2);
    outrem.setKeepOrganized(true);
    // apply filter
    outrem.filter (*cloud_filtered);

    return 0;
}
```

Statistical Outlier Removal

- For each point, compute the mean distance from it to all its neighbors
- By modeling distances as Gaussian distribution, get mean / std. dev.
- All points whose mean distances are outside the interval is outlier

$$\mu = \frac{1}{nk}\sum_{i=1}^{n}\sum_{j=1}^{k} d_{ij}, \quad \sigma = \sqrt{\frac{1}{nk}\sum_{i=1}^{n}\sum_{j=1}^{k}(d_{ij}-\mu)^2}.$$

$$\frac{1}{k}\sum_{j=1}^{k} d_{ij} > \mu + c\sigma \quad \text{or} \quad \frac{1}{k}\sum_{j=1}^{k} d_{ij} < \mu - c\sigma,\ c \in \mathbb{R}^{+}.$$

Statistical Outlier Removal

Open3D

```python
import open3d as o3d
pcd = o3d.io.read_point_cloud("PATH/TO/POINTCLOUD.pcd")
cl, ind = pcd.remove_statistical_outlier(nb_neighbors=20, std_ratio=2.0)
```

PCL

```cpp
#include <iostream>
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
#include <pcl/filters/voxel_grid.h>

int main()
{
    pcl::PCLPointCloud2::Ptr cloud (new pcl::PCLPointCloud2 ());
    pcl::PCLPointCloud2::Ptr cloud_filtered (new pcl::PCLPointCloud2 ());

    pcl::PCDReader reader;
    reader.read ("PATH/TO/PCD", *cloud);

    pcl::StatisticalOutlierRemoval<pcl::PointXYZ> sor;
    sor.setInputCloud (cloud);
    sor.setMeanK (50);
    sor.setStddevMulThresh (1.0);
    sor.filter (*cloud_filtered);

    return 0;
}
```