Einführung

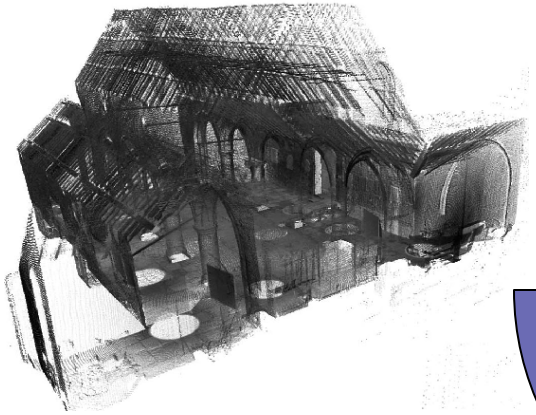
Point Cloud Library

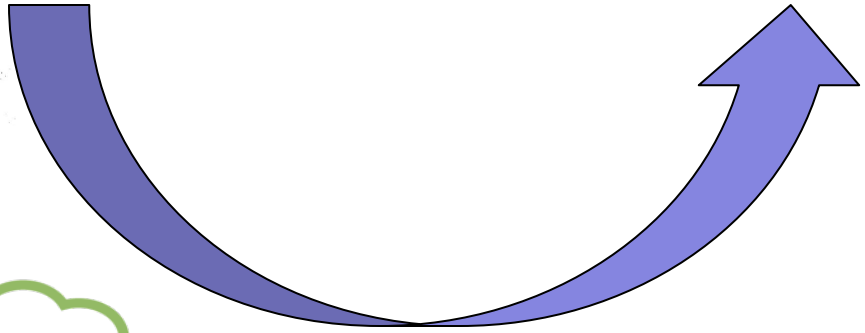
Inhalt

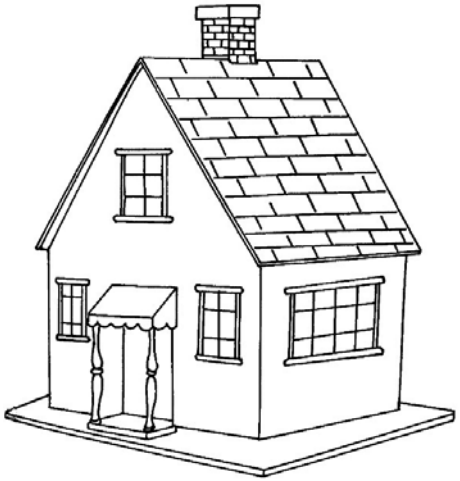
1. Was ist PCL?
2. Was kann PCL?
3. Wie funktioniert PCL?
4. Module
5. Klassen
6. Klassen-/Funktionsverkettung
7. Point Cloud Data (PCD)
8. Workflow
9. Beispiele

1. Was ist PCL?

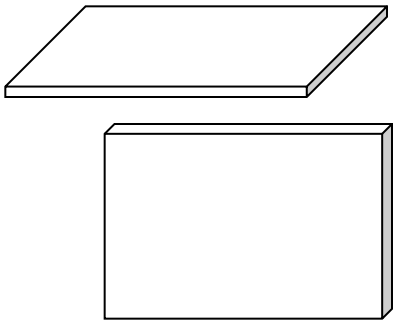
- Open-Source-Framework zur 3D/4D Punktwolken-/Geometrienprozessierung
- C++ Libraries
- Plattformunabhängig
- Standalone
- Ansammlung umfangreicher Module/Libraries







pcl



Quelle: http://pointclouds.org/about/#open

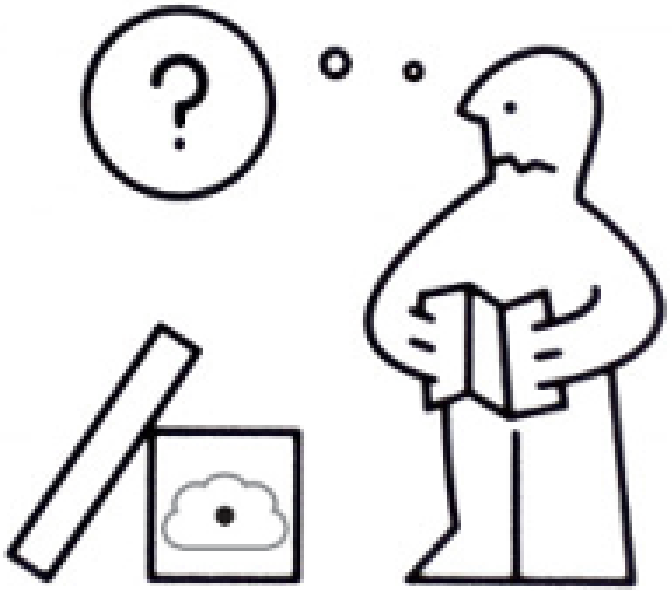
An Entwicklung beteiligt

Quelle: http://pointclouds.org/about/#open

Finanzielle Unterstützer

2. Was kann PCL?

- Filterung
- Oberflächenkonstruktion
- Schätzverfahren
- Model fitting
- Erkennung von Ausreißern
- Rauschminderung
- Segmentierung
- Objekterkennung
- Visualisierung
- Punktwolkenregistrierung



3. Wie funktioniert PCL?

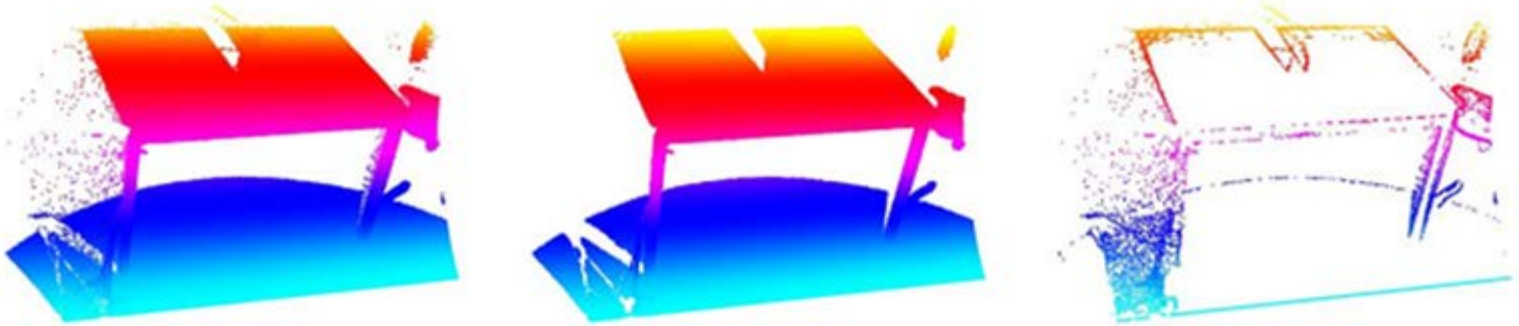
- Allgemeingültige und wiederverwendbare Prozesse in Moulen mit diversen Klassen/Funktionalitäten
- Verwendung gängiger APIs und Libraries:
  - Eigen (Matrizen, Vektoren, Lineare Algebra)
  - FLANN (Fast Library for Approximate Nearest Neighbors)
  - Qhull (konvexe Hülle, Delaunay Triangulation, Voronoi)
  - OpenMP (Thread Multiprocessing)
  - Usw.

- Einfacher Programmablauf:

  1. Prozessobjekt erstellen
  2. Input setzen
  3. Parameter setzen
  4. Ausführen

```
pcl::StatisticalOutlierRemoval<pcl::PointXYZ> f;
f.setInputCloud (input_cloud);
f.setMeanK (50);
f.setStddevMulThresh (1.0);
f.filter (output_cloud);
```

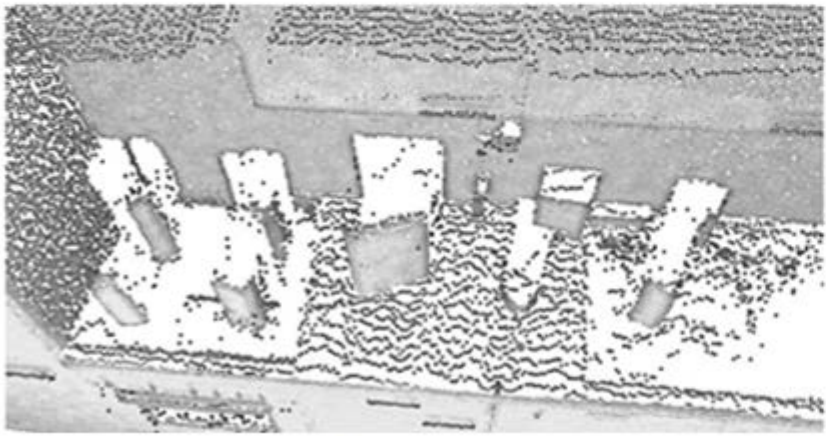
1
2
3
3
4

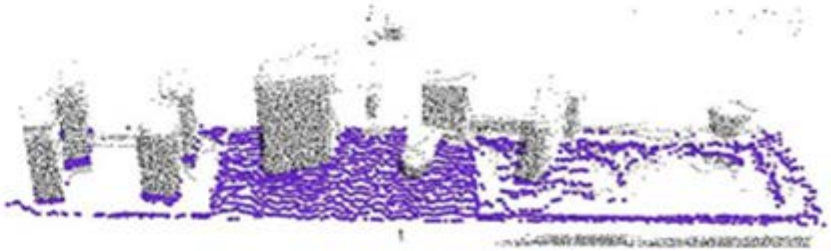


Beispiel für Ausreißerentfernung mit k=50 Nachbarschaften

```
pcl::SACSegmentation<pcl::PointXYZ> f;
f.setInputCloud (input_cloud);
f.setModelType (pcl::SACMODEL_PLANE);
f.setMethodType (pcl::SAC_RANSAC);
f.setDistanceThreshold (0.01);
f.segment (output_cloud);
```

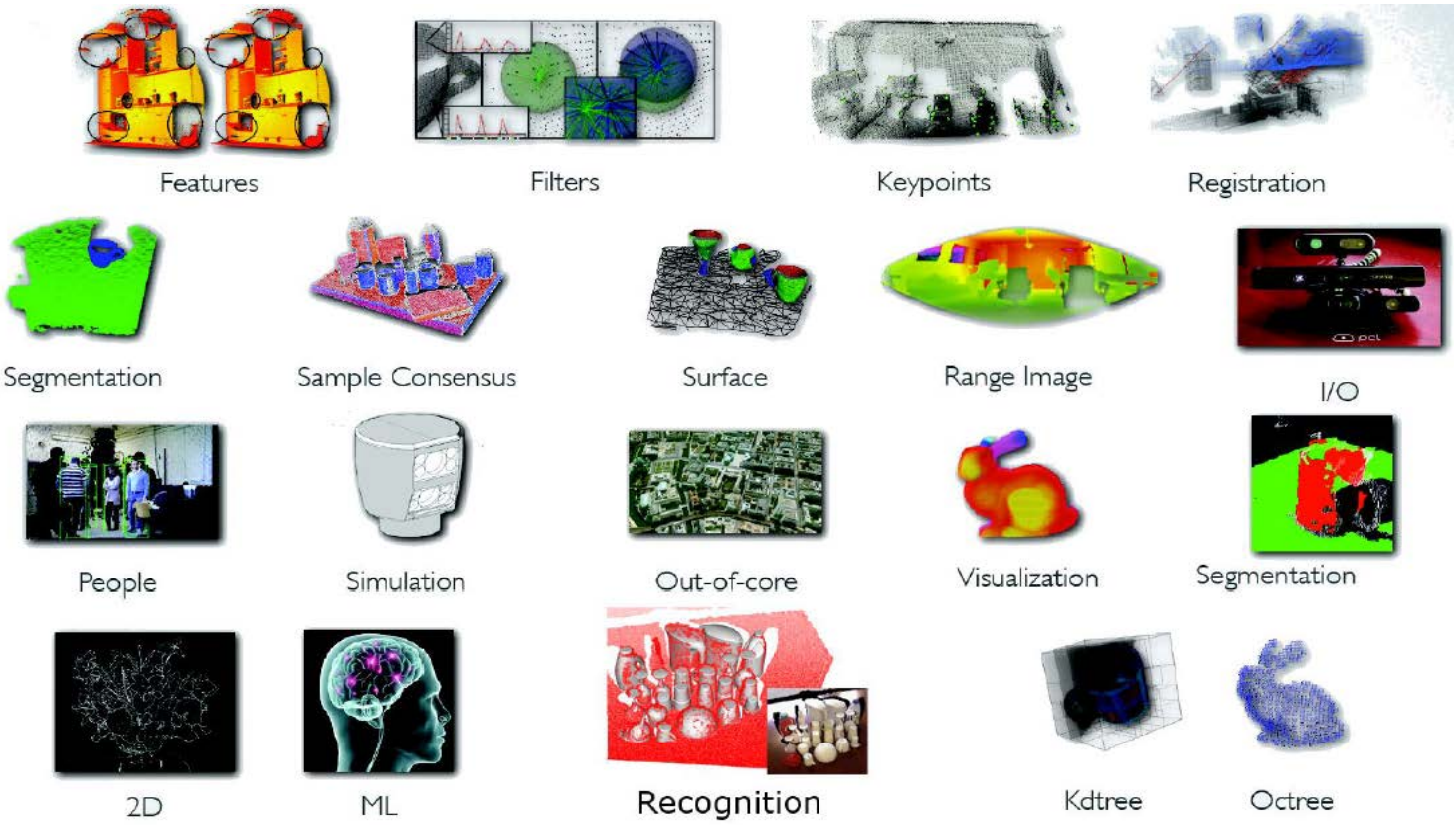
1
2
3
3
3
4





Segmentierung/Ebenenerkennung mit Ransac (1cm Toleranz)

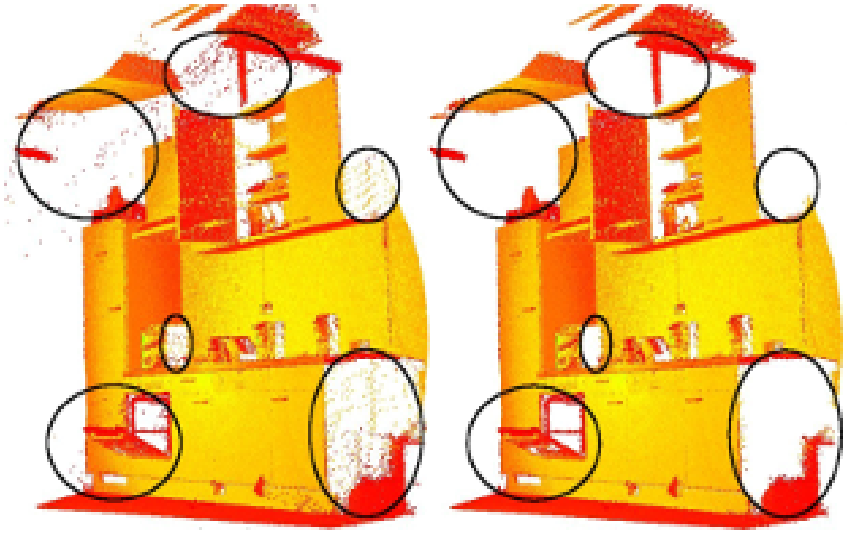
4. Module

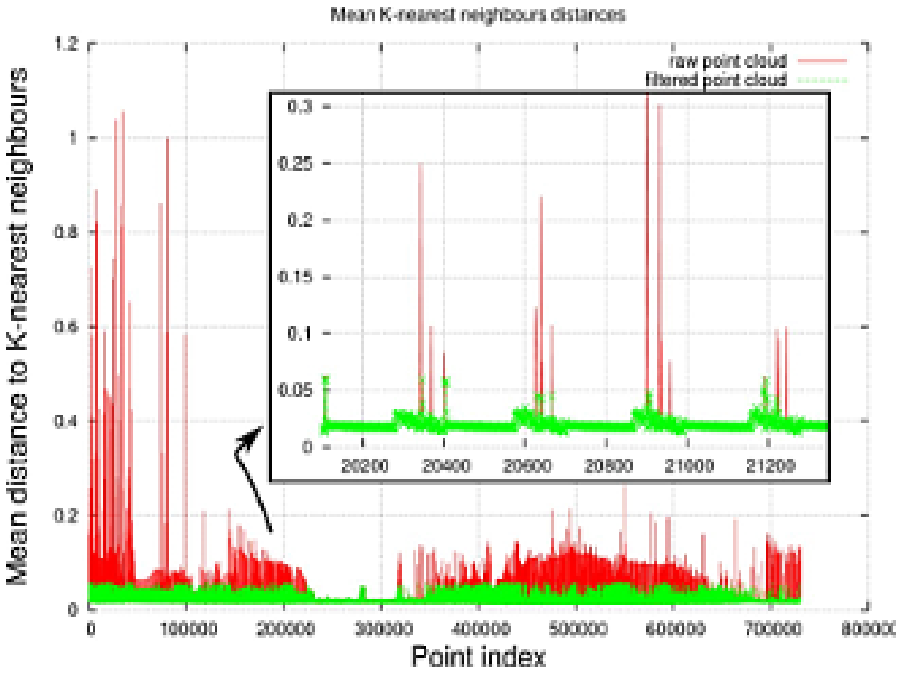

Features
Filters
Keypoints
Registration
Segmentation
Sample Consensus
Surface
Range Image
I/O
People
Simulation
Out-of-core
Visualization
Segmentation
2D
ML
Recognition
Kdtree
Octree

- **Common**:
  - Grundlagen
  - Gemeinsam genutzte Datenstruktur und Methoden
  - PointCloudClass
  - Punkttypen
  - Distanzberechnung
  - etc

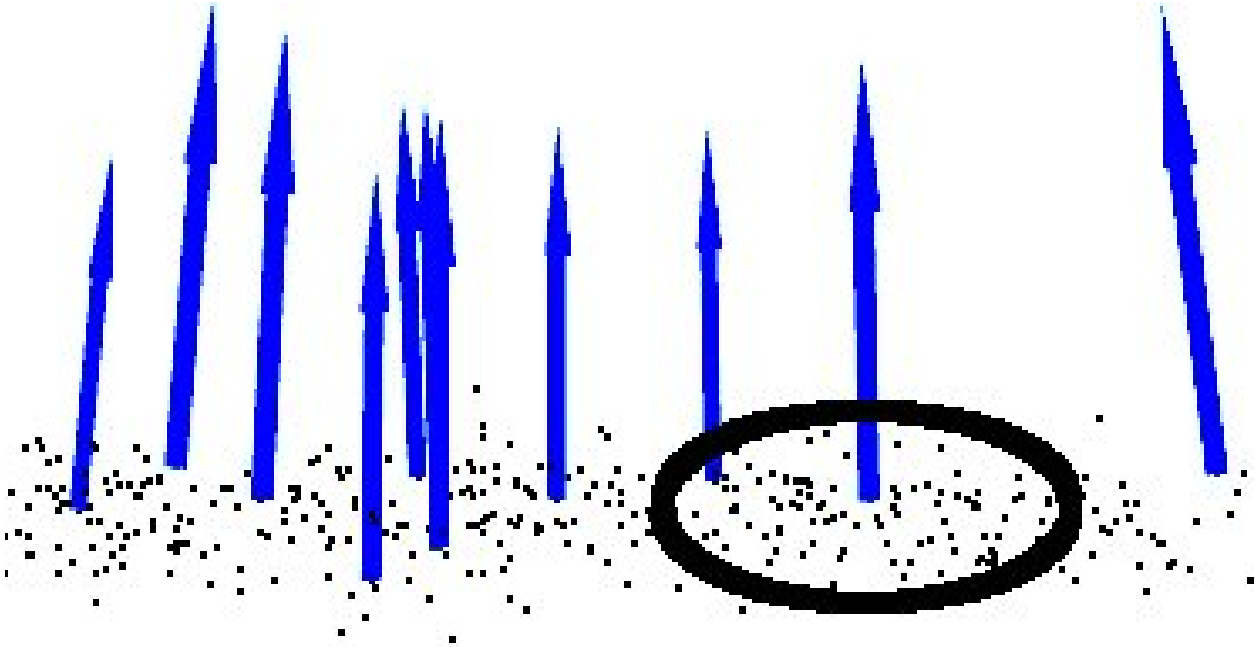
- **IO:**
  - Input/Output – Lesen/Schreiben der Punktwolke in Point Cloud Data

- **Filter**:
  - Statistische Ausreißerverfahren

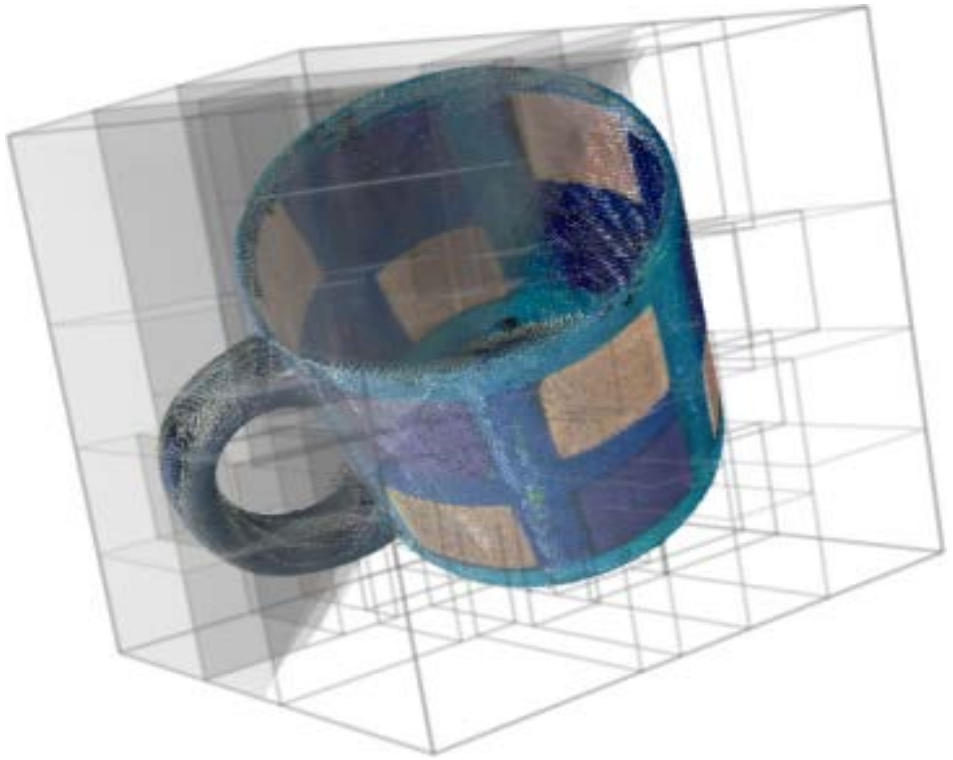




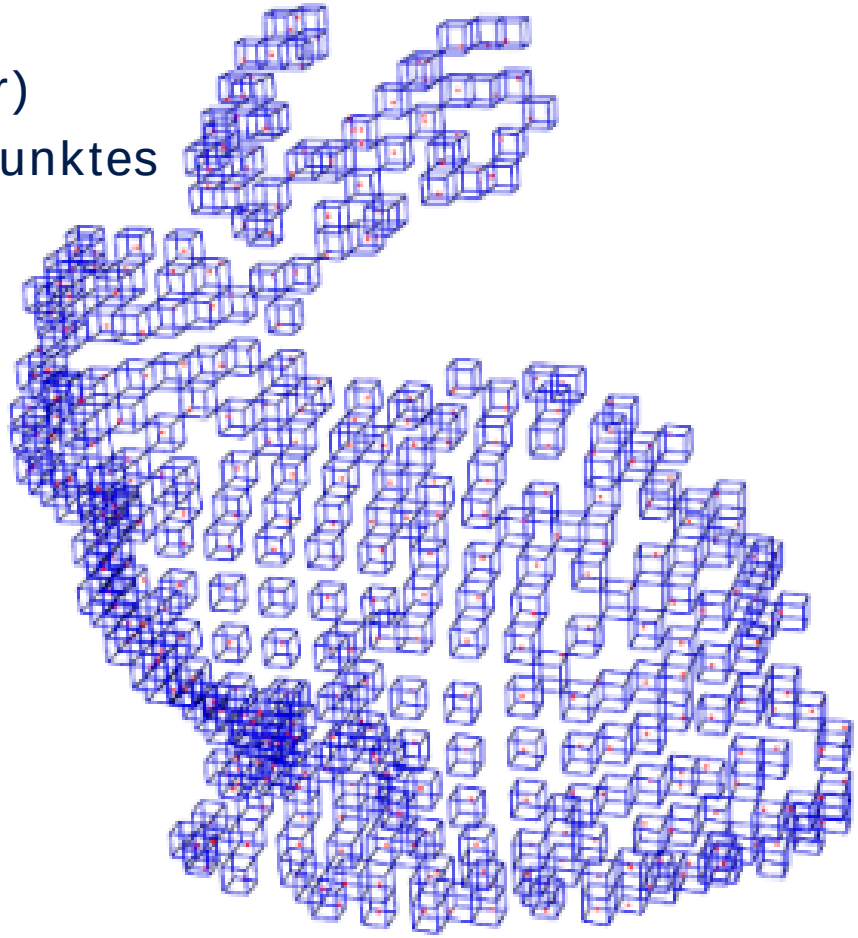
- **Features:**
  - Schätzfunktionen
  - Normalen-, Krümmungsberechnung
  - K-Nachbarschaften



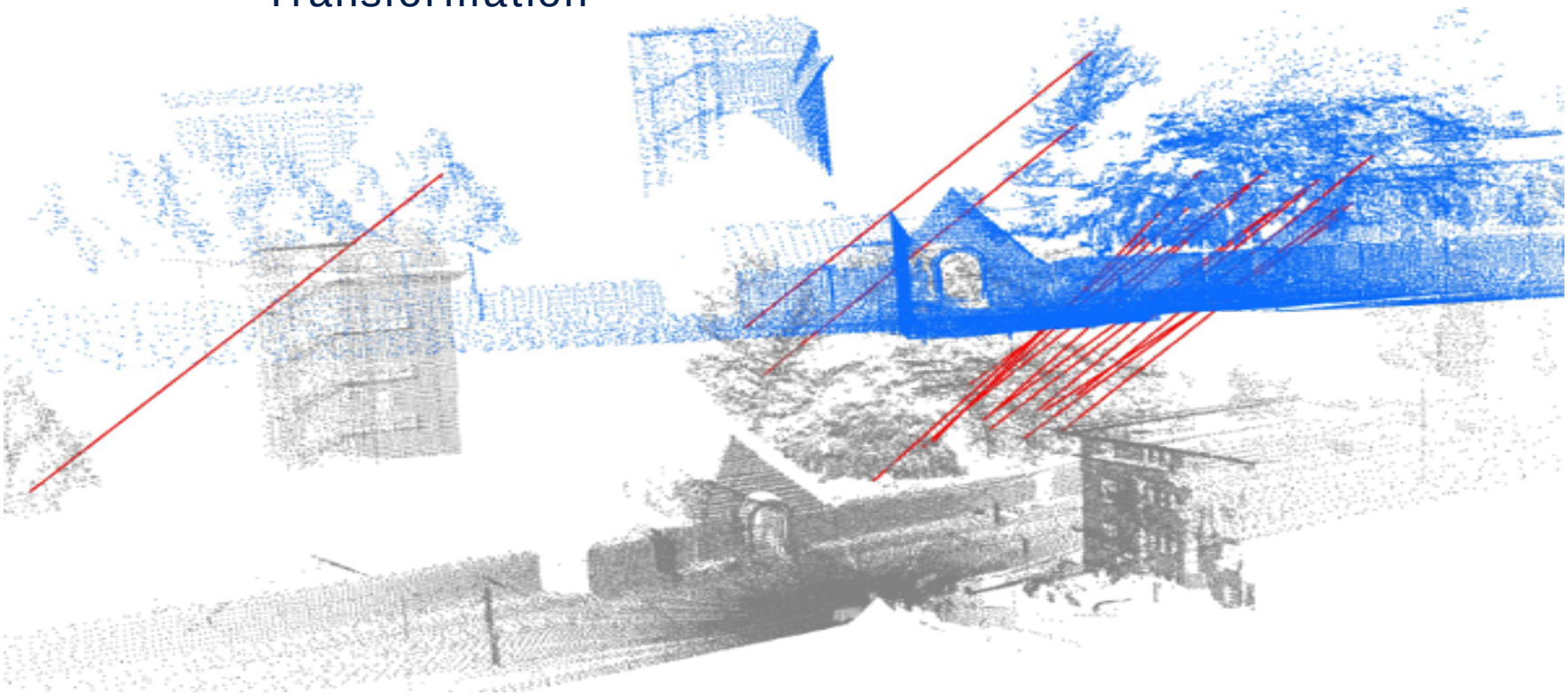
- **KDTree**:
  - Nachbarschaftssuche
  - K-dimensionaler Baum zur Raumpartitionierung
  - Splittet entlang der Dimension
  - Binärbaum



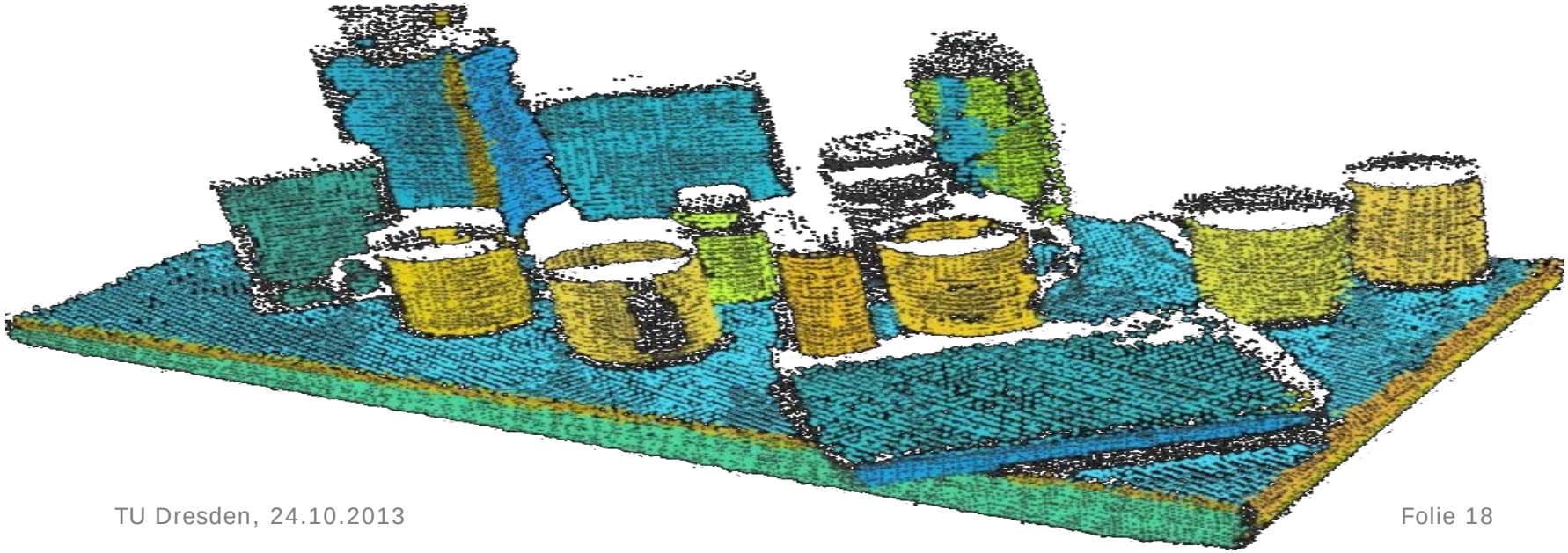
- **OCTree**
  - 8er Baum (nicht binär)
  - Splittet entlang des Punktes
  - Ebenso Raumteilung



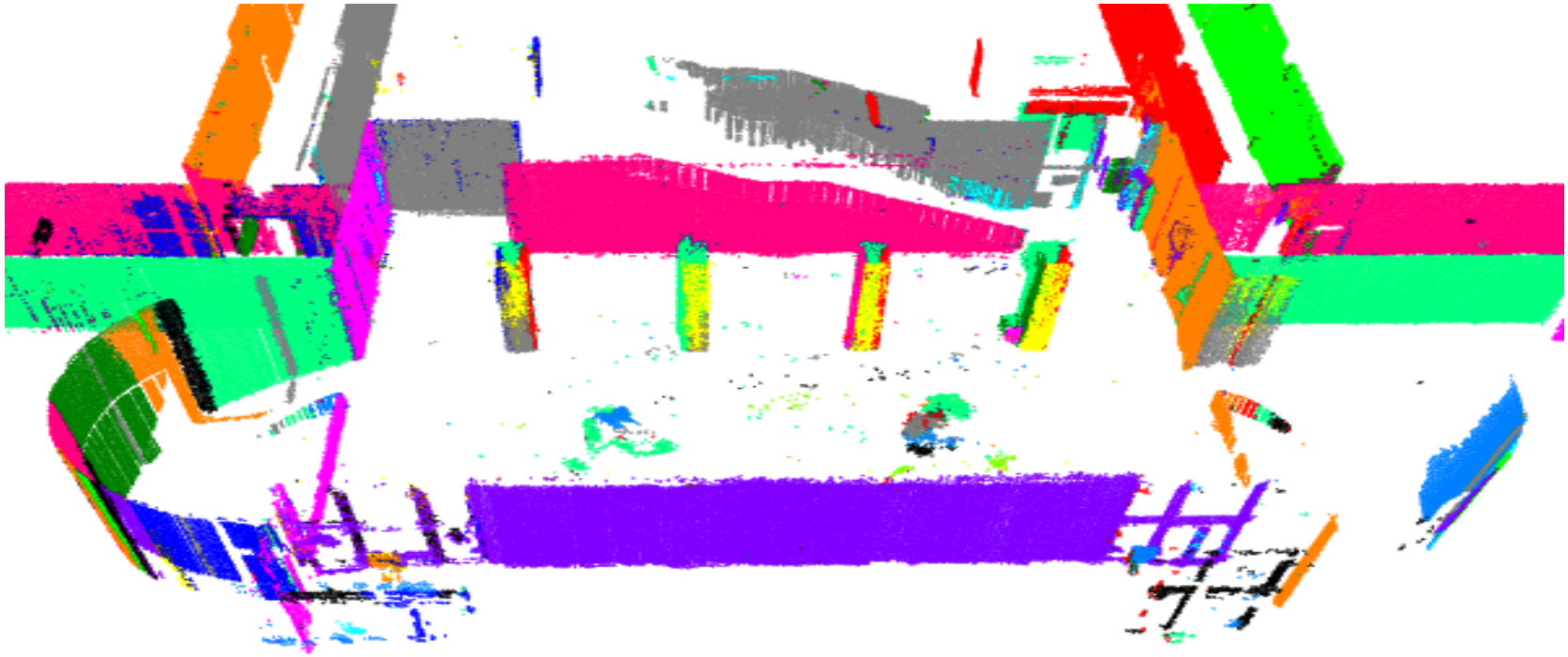
- **Registration**:
  - Punktwolken verbinden
  - Transformation

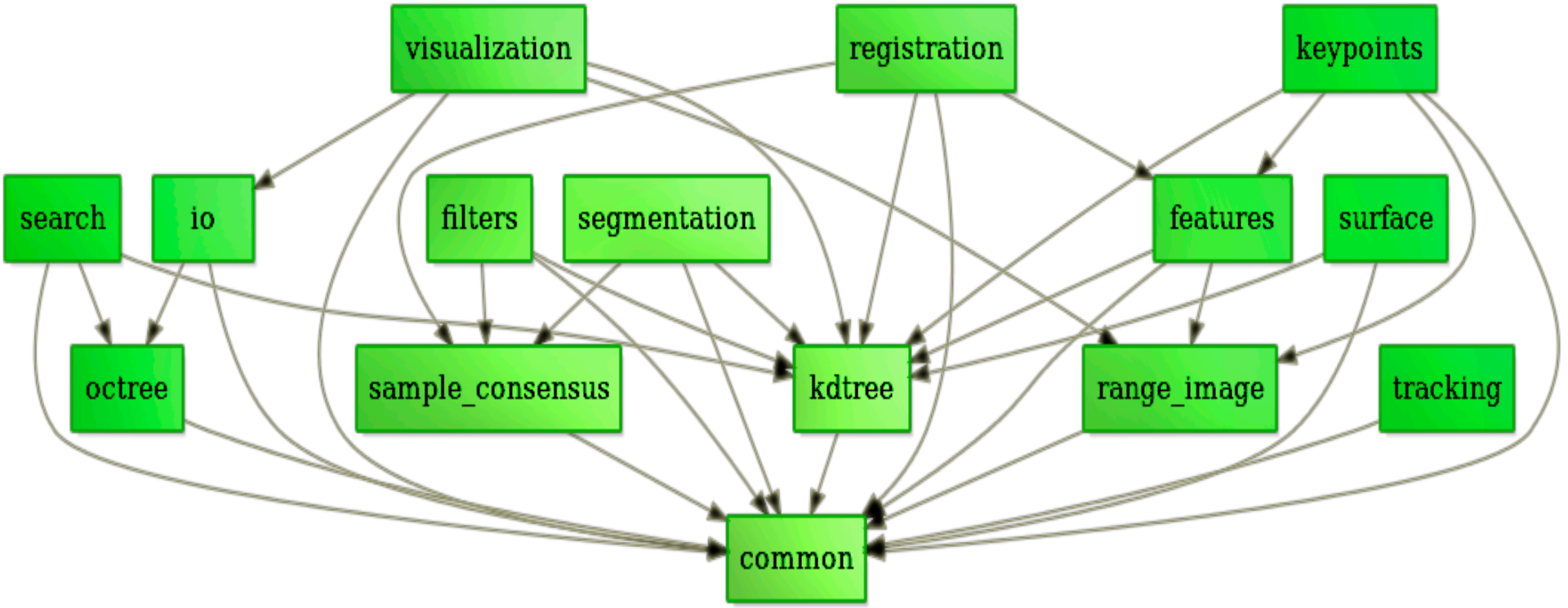


- **Sample Consensus:**
  - Template Objekte für Abgleich
  - Auffinden geometrischer Formen (Zylinder, Kugel etc.)
  - Z.B. mit Hilfe von RANSAC
  - Modeldetektion + Parameter
  - (Plane Fitting: Türen, Wände



- **Segmentation:**
  - Clusterbildung
  - Auffinden isolierter Regionen
  - Geeignet zur Vorprozessierung




visualization
registration
keypoints
search
io
filters
segmentation
features
surface
octree
sample_consensus
kdtree
range_image
tracking
common

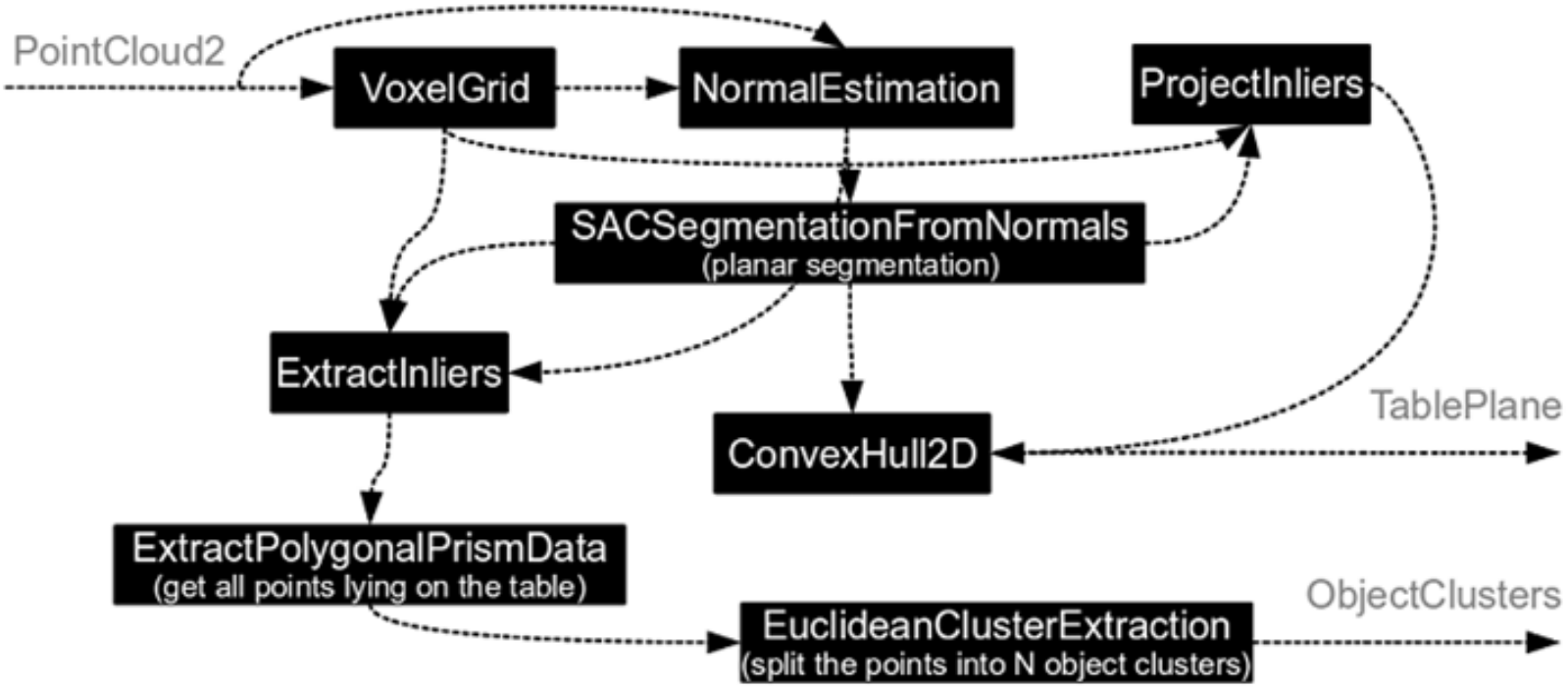
5. Klassen

- VoxelGrid
  - Grid über Punktwolke -> mehrere Punkte durch Schwerpunkt ersetzen
- OutlierRemoval
  - Ausreißer über durchschnittliche Entfernungen finden
- SACSegmentation
  - Detektieren von Modellen (RANSAC)

- EuclideanClusterExtraction
  - Clustersuche
- BoundaryEstimation
  - Oberflächenpunkte in Abhängigkeit vom Winkelkriterium finden
- ConcaveHull

- PlaneWithPlaneIntersection
  - Schnittgeraden finden
- LineWithLineIntersection
  - Schnittpunkte finden

6. Klassen-/Funktionenverkettung

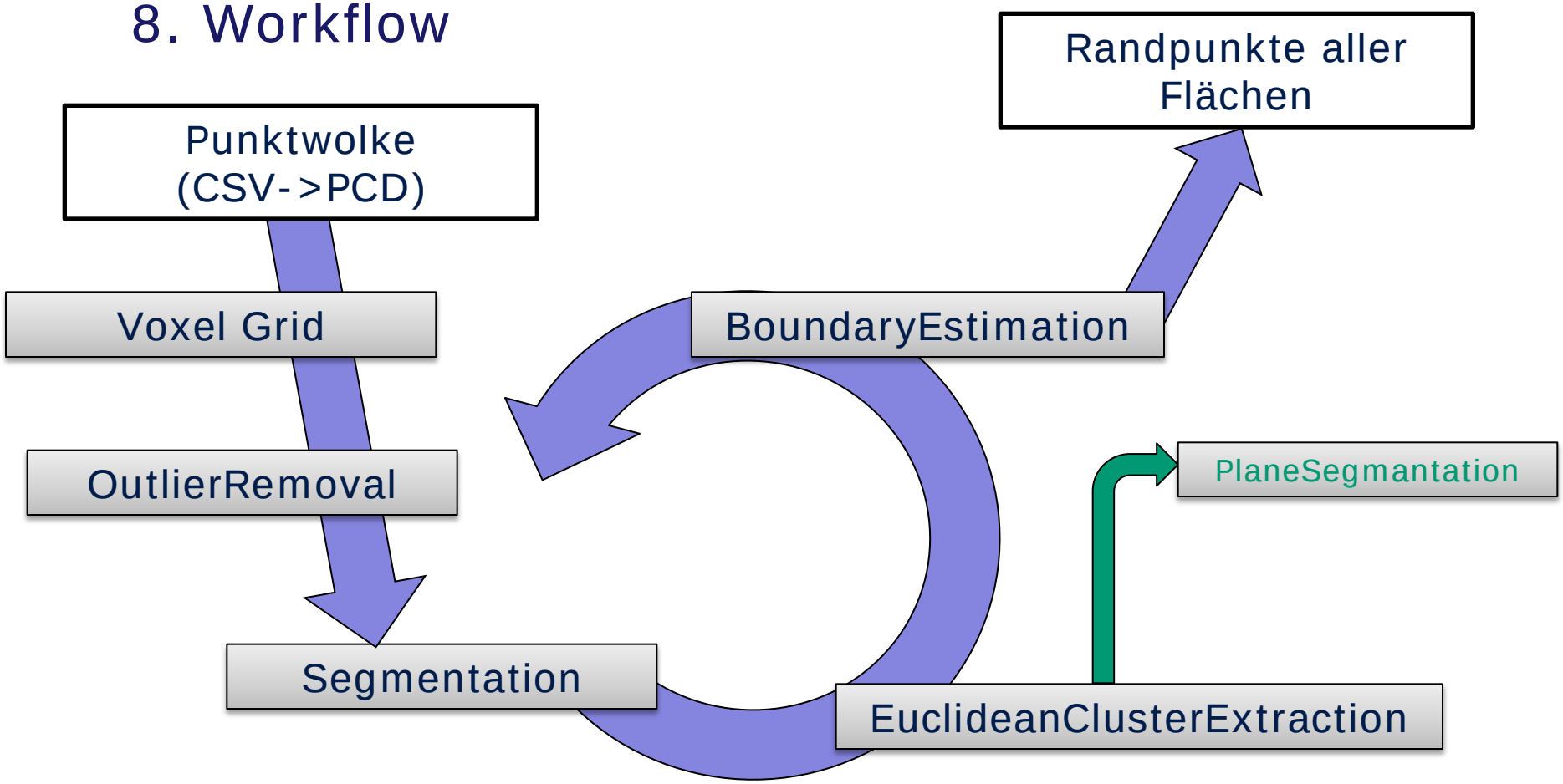


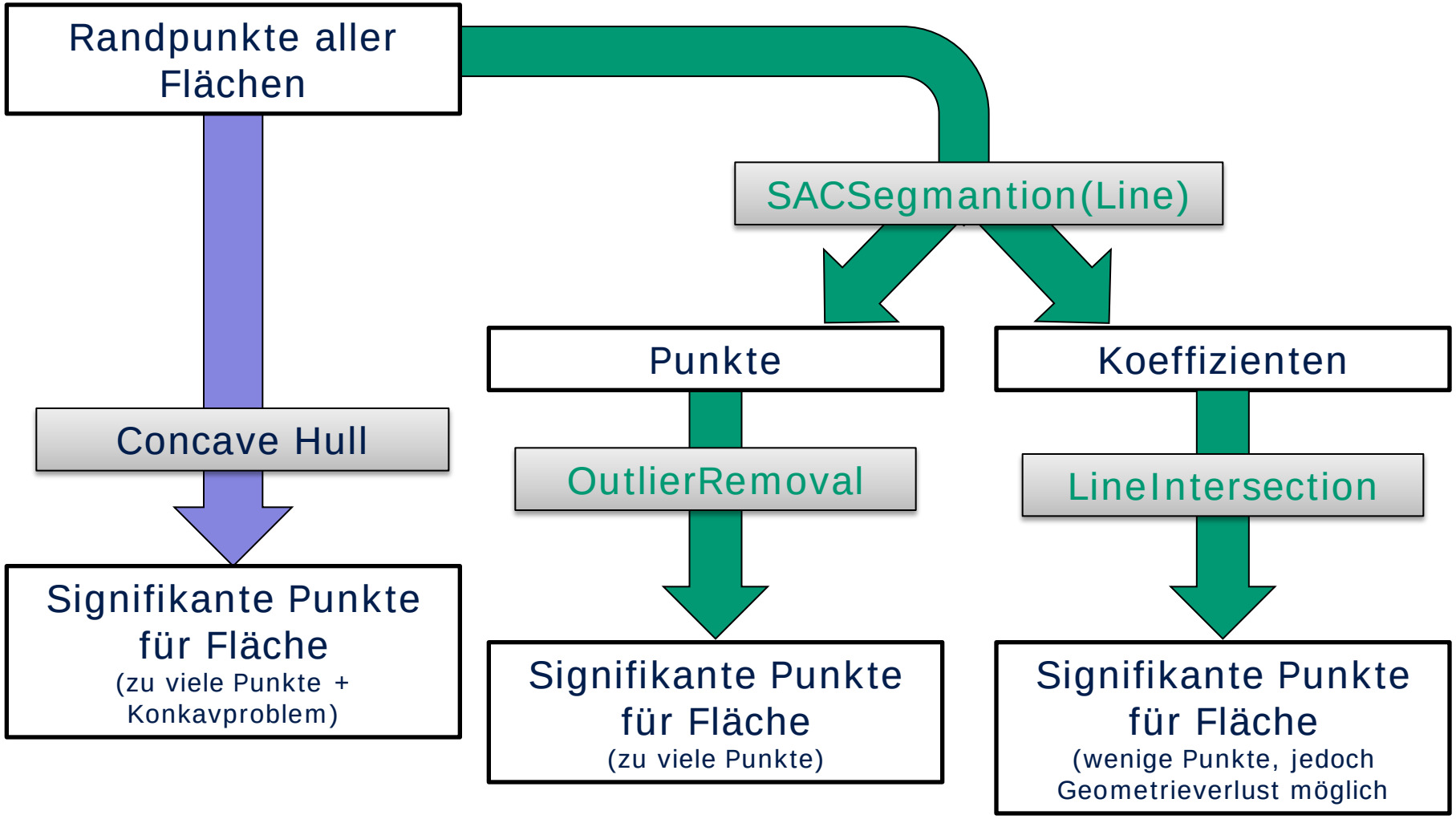
7. Point Cloud Data (PCD)

```
# .PCD v.7 - Point Cloud Data file format
VERSION .7
FIELDS x y z rgb
SIZE 4 4 4 4
TYPE F F F F
COUNT 1 1 1 1
WIDTH 213
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 213
DATA ascii
0.93773 0.33763 0 4.2108e+06
0.90805 0.35641 0 4.2108e+06
0.81915 0.32 0 4.2108e+06
0.97192 0.278 0 4.2108e+06
0.944 0.29474 0 4.2108e+06
0.98111 0.24247 0 4.2108e+06
0.93655 0.26143 0 4.2108e+06
0.91631 0.27442 0 4.2108e+06
```

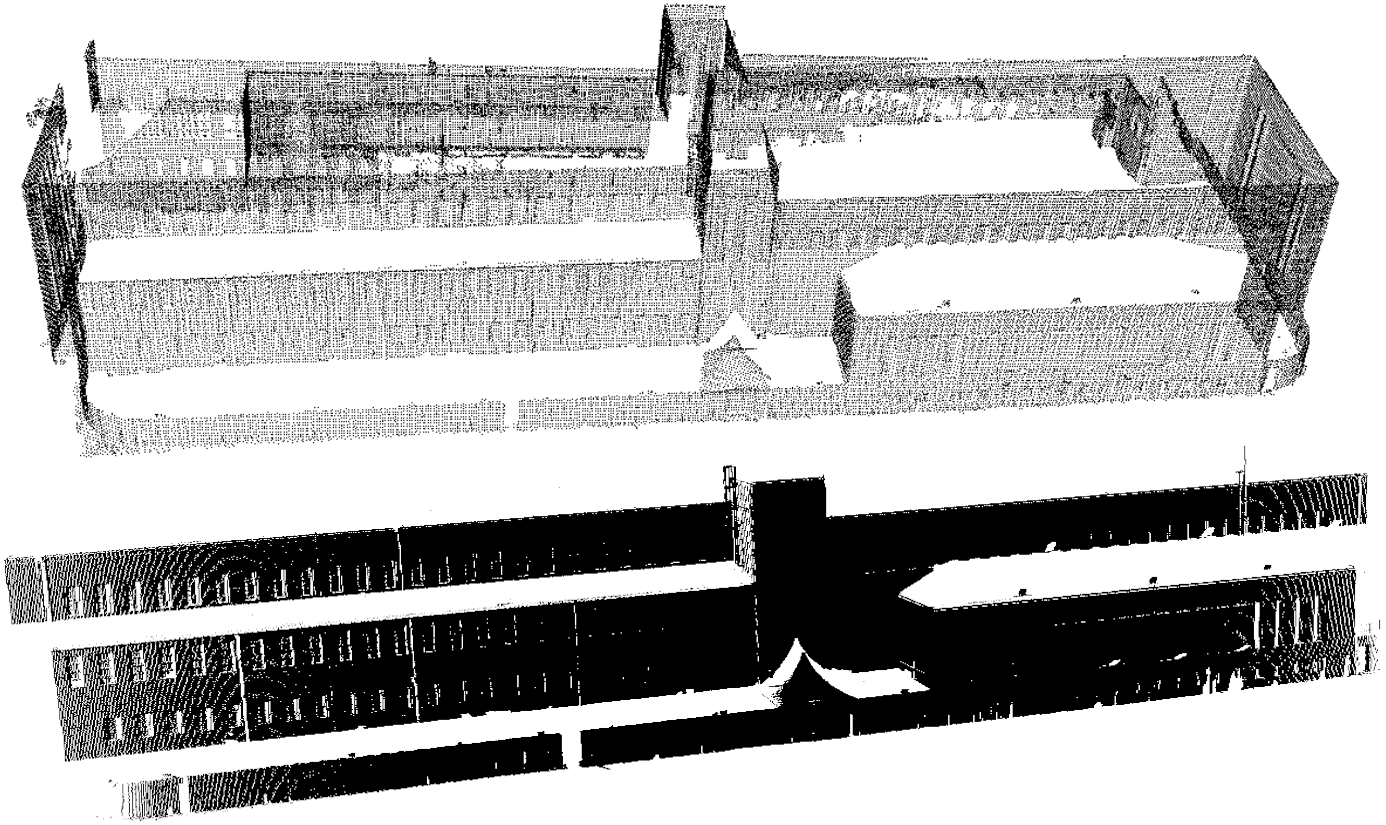
- FIELDS → X Y Z / X Y Z RGB / X Y Z Normale1, Normale2, Normale3
- SIZE → Dimensionsgröße (char, short, float, double)
- TYPE → Unsigned, Signed, Float
- COUNT → Elemente/Dimension
- WIDTH → Unorganisiert (Gesamtpunkte) / Organisiert (Punkte/Zeile)
- HEIGHT → Punktanzahl in Höhe bei organisiert / 1 unorganisiert
- VIEWPOINT → Aufnahmewinkel - Translation(xyz) + Drehung (wxyz)
- POINTS → Gesamtpunktanzahl
- DATA → Format (ascii, binär)

8. Workflow


Randpunkte aller Flächen
Punktwolke (CSV->PCD)
Voxel Grid
BoundaryEstimation
OutlierRemoval
PlaneSegmantation
Segmentation
EuclideanClusterExtraction


Randpunkte aller Flächen
SACSegmantion(Line)
Punkte
Koeffizienten
Concave Hull
OutlierRemoval
LineIntersection
Signifikante Punkte für Fläche
(zu viele Punkte + Konkavproblem)
Signifikante Punkte für Fläche
(zu viele Punkte)
Signifikante Punkte für Fläche
(wenige Punkte, jedoch Geometrieverlust möglich

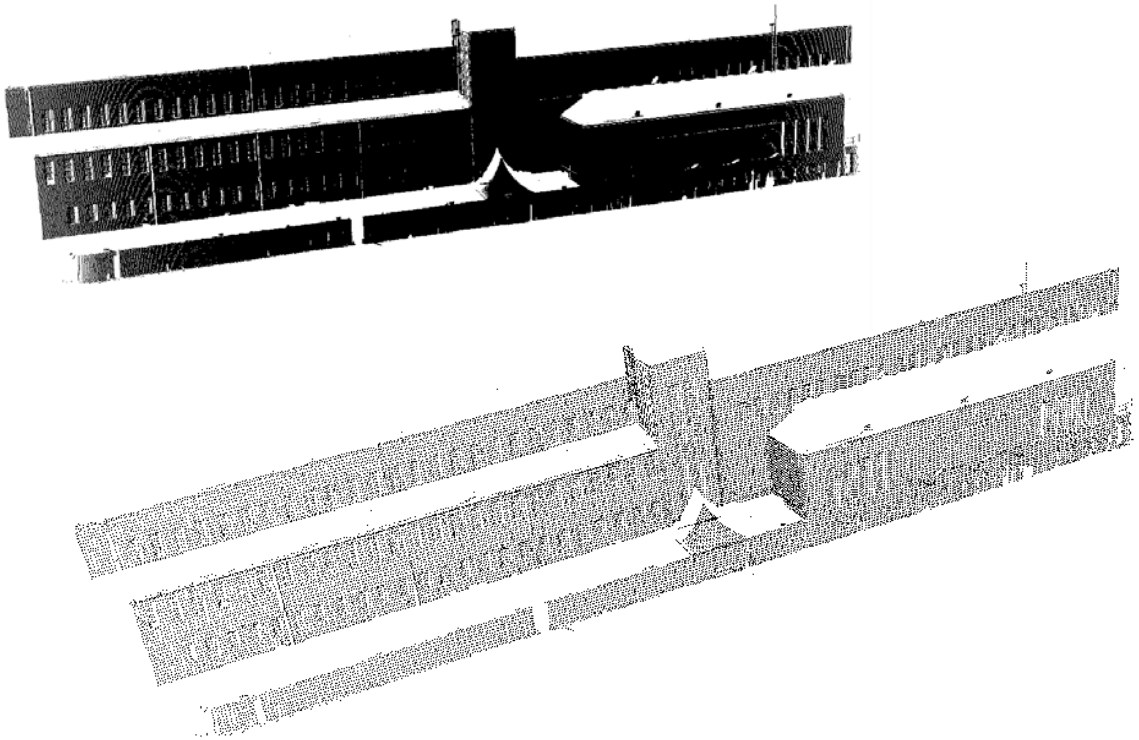
9. Beispiele



Punktwolke (CSV->PCD)

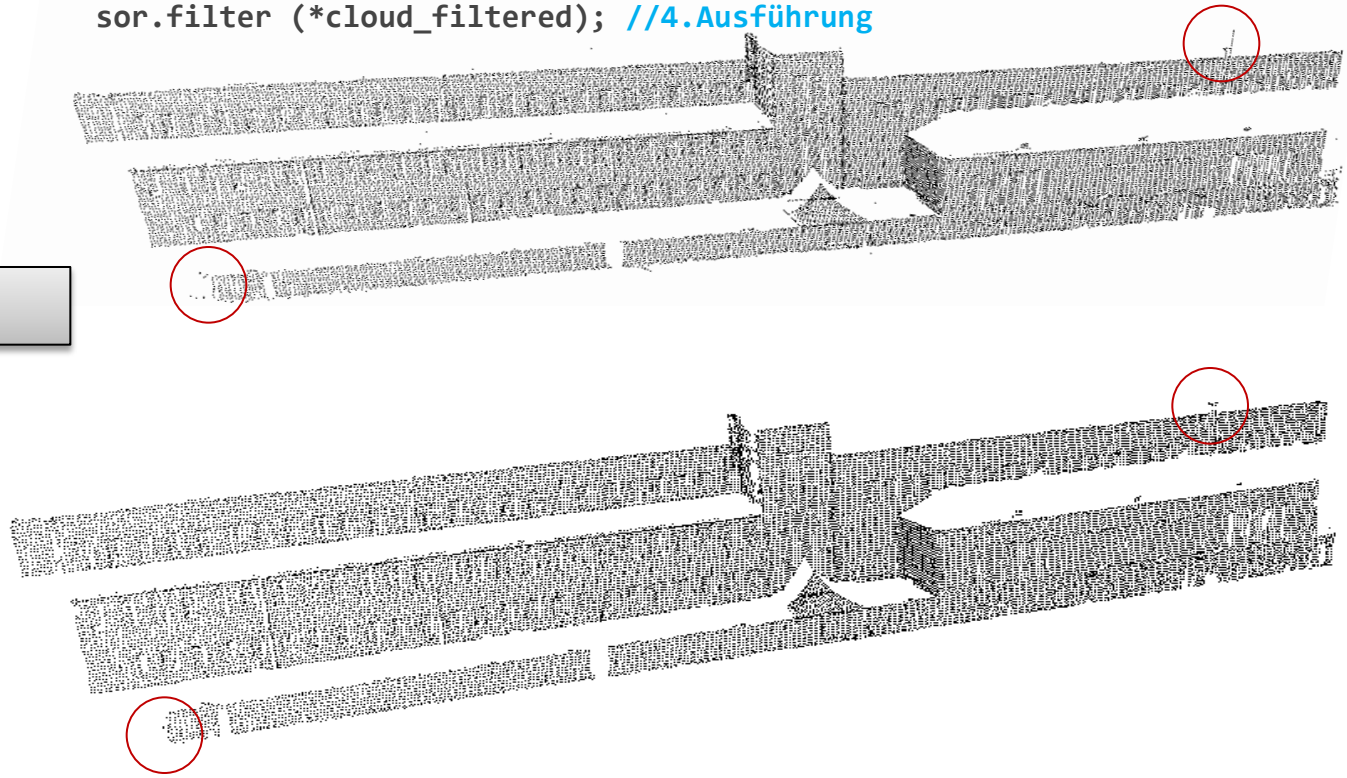
Voxel Grid

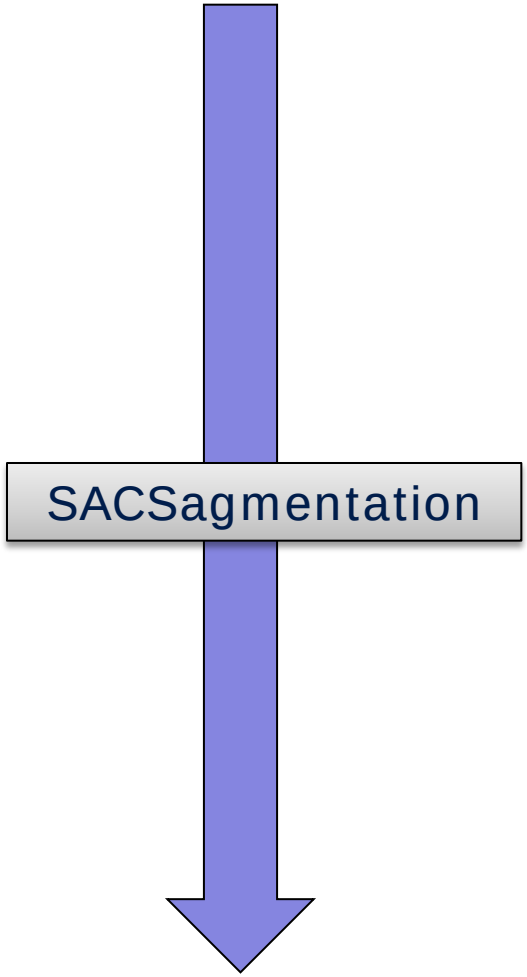
```
pcl::VoxelGrid<sensor_msgs::PointCloud2> sor; //1. Objekt erstellen
sor.setInputCloud (cloud_input); //2. Input setzen
sor.setLeafSize (0.25f, 0.25f, 0.25f); //3. Parameter setzen
sor.filter (*cloud_output); //4. Ausführen
```



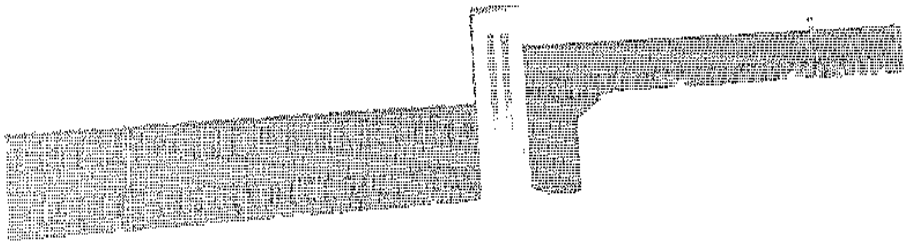
OutlierRemoval

```
pcl::StatisticalOutlierRemoval<pcl::PointXYZ> sor; //1. Objekt erstellen
sor.setInputCloud (cloud); //2. Input setzen
sor.setMeanK (50); //3.Parameter setze, Nachbaranzahl + Schwellenwert
sor.setStddevMulThresh (1.0);
sor.filter (*cloud_filtered); //4.Ausführung
```



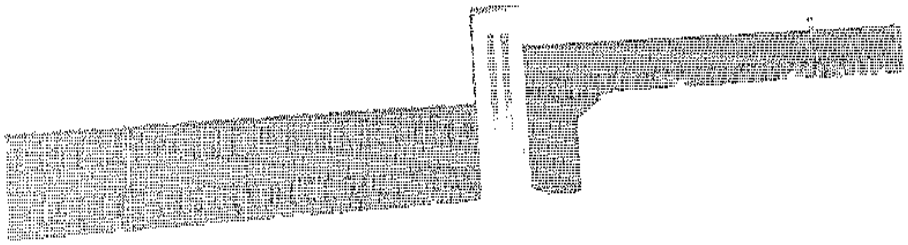



```
pcl::SACSegmentation<pcl::PointXYZ> seg; //1. Objekt erstellen
//Input wird später hinzugefügt
//3.Parameter setzen, Modelltyp, Methode, Schwellenwert
seg.setModelType (pcl::SACMODEL_PLANE);
seg.setMethodType (pcl::SAC_RANSAC);
seg.setDistanceThreshold (0.5);
...
```

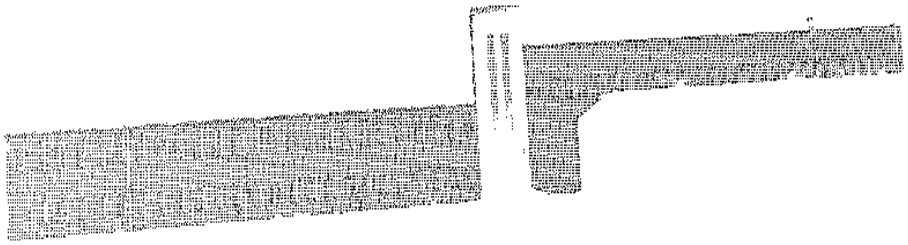


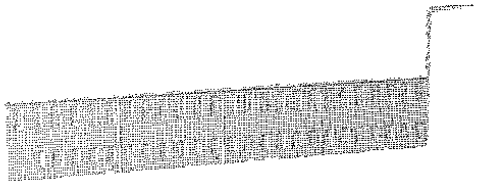
SACSagmentation

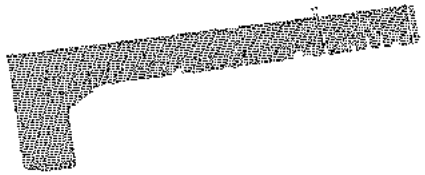
```
...
nr_points = (int) cloud ->points.size ();
while (cloud ->points.size () > 0.01 * nr_points)
            {

           seg.setInputCloud (cloud); //2.Input setzen
           seg.segment (*inliers, *coefficients); //4.Ausführen
 }
```

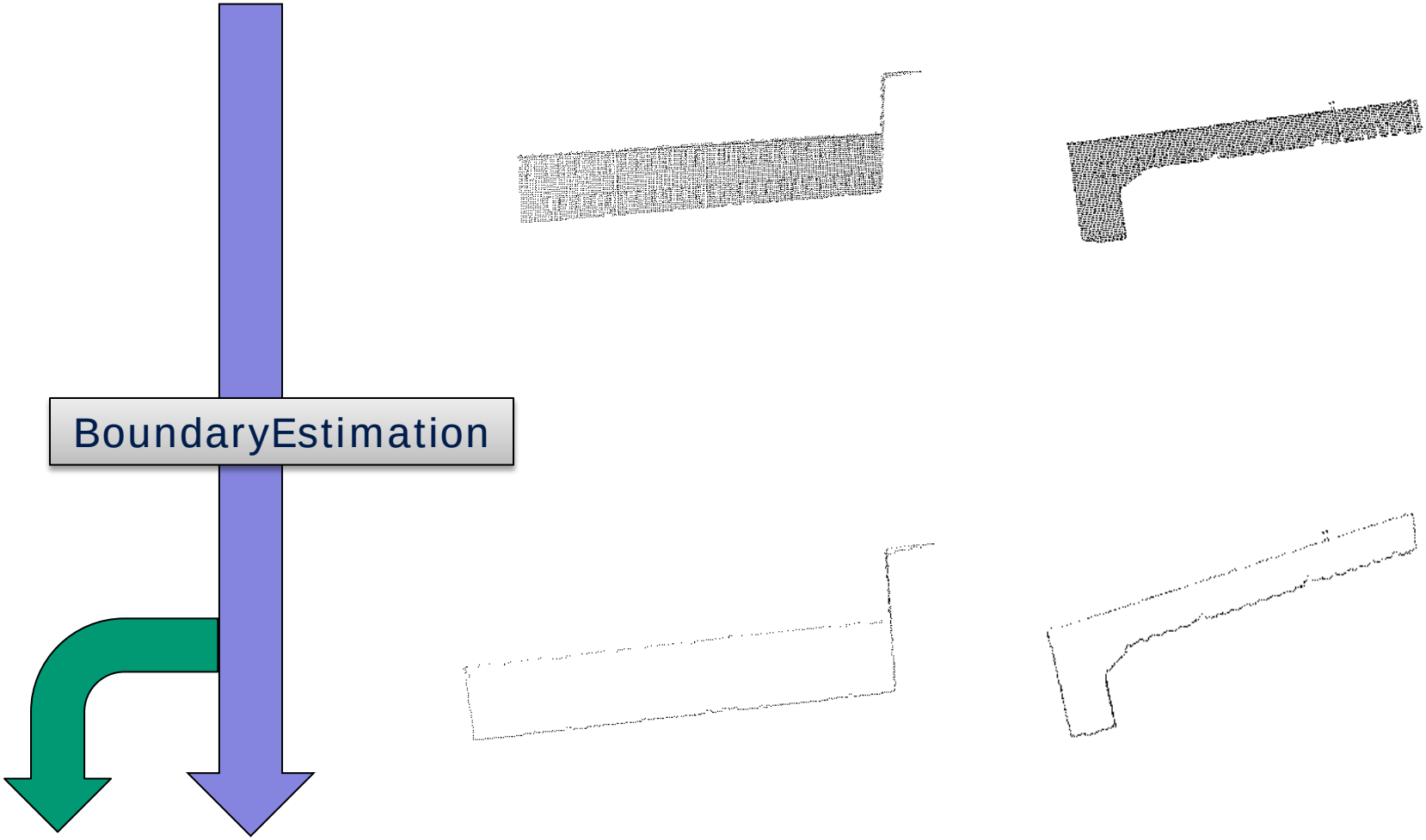


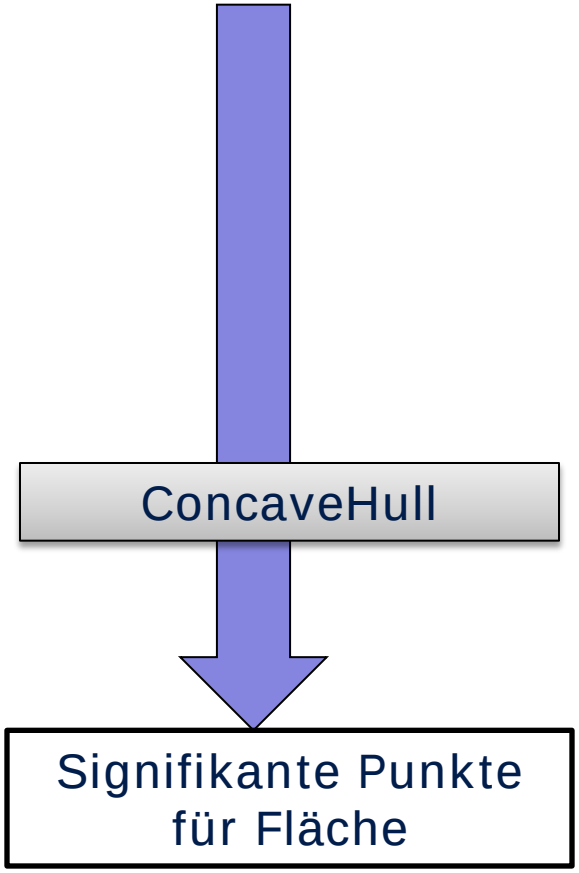
ClusterExtraction

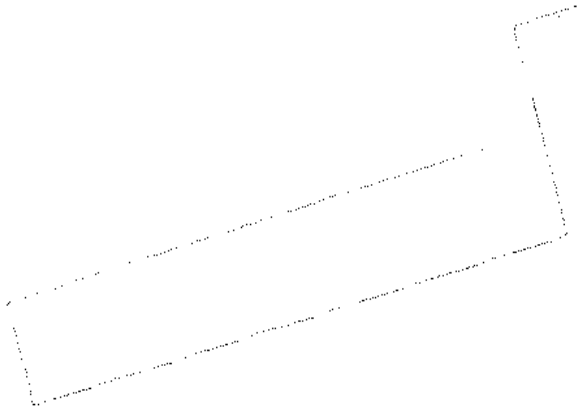


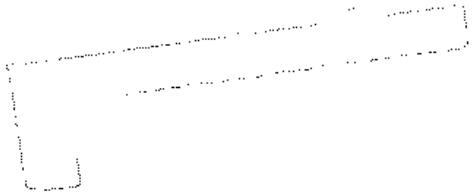


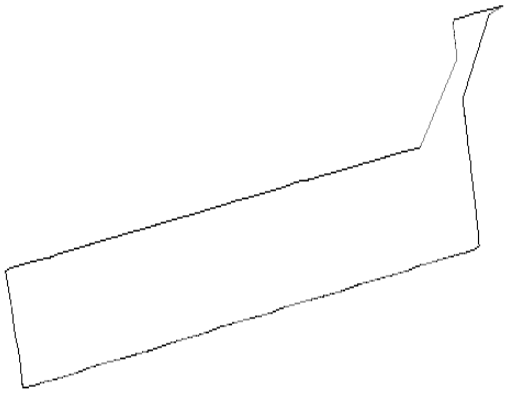


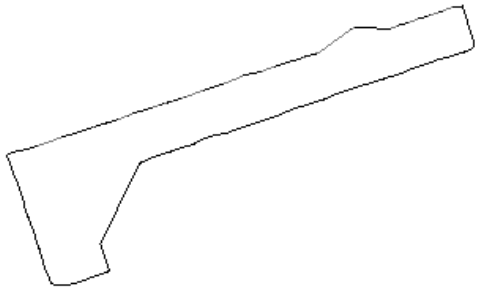

BoundaryEstimation


ConcaveHull
Signifikante Punkte für Fläche

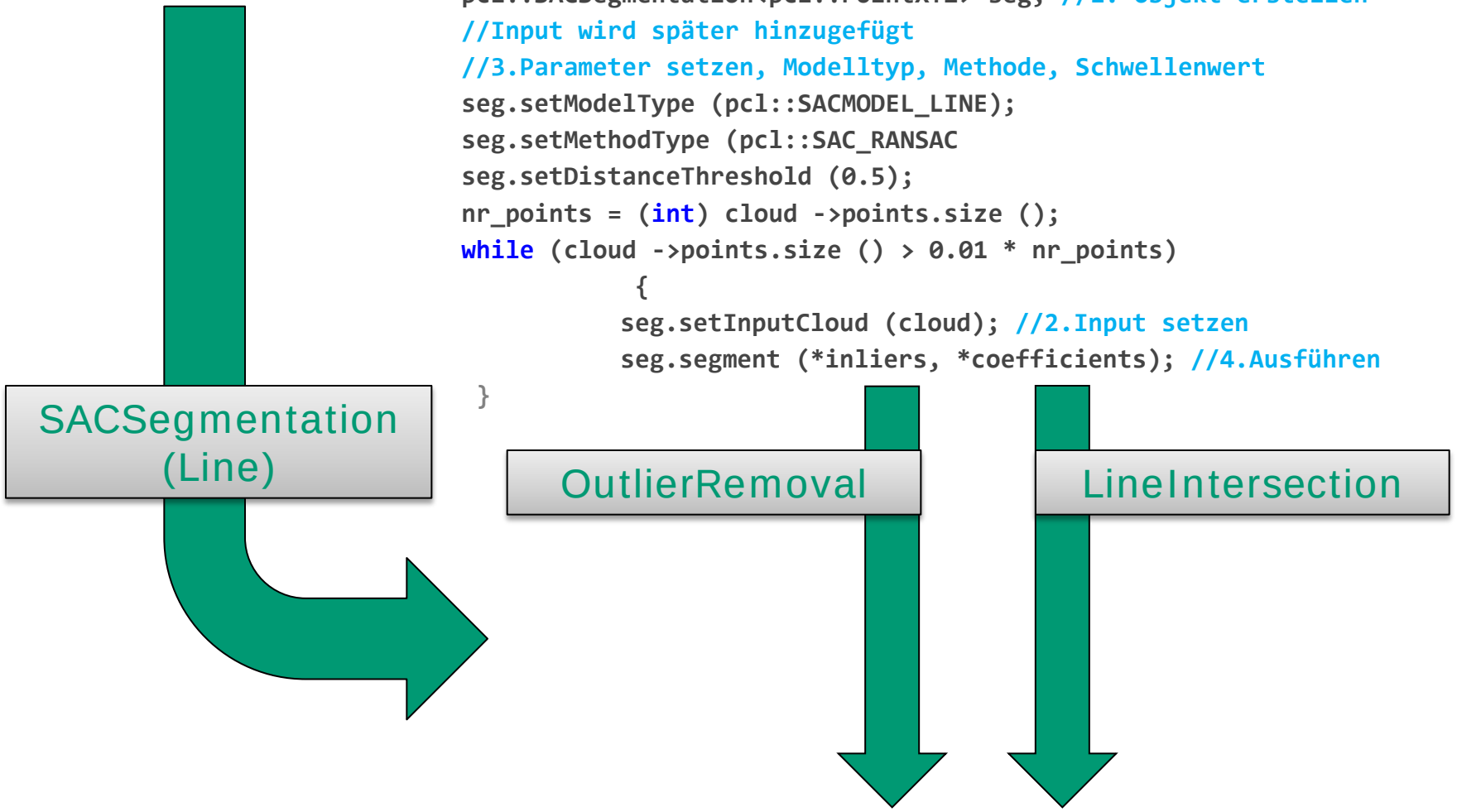






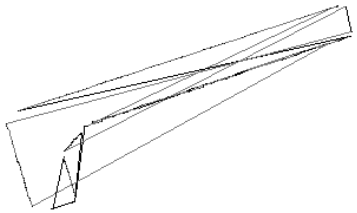


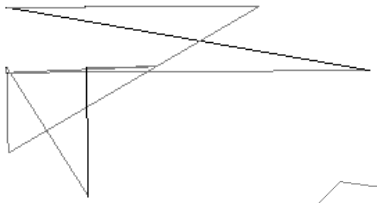
```
pcl::SACSegmentation<pcl::PointXYZ> seg; //1. Objekt erstellen
//Input wird später hinzugefügt
//3.Parameter setzen, Modelltyp, Methode, Schwellenwert
seg.setModelType (pcl::SACMODEL_LINE);
seg.setMethodType (pcl::SAC_RANSAC
seg.setDistanceThreshold (0.5);
nr_points = (int) cloud ->points.size ();
while (cloud ->points.size () > 0.01 * nr_points)
            {
           seg.setInputCloud (cloud); //2.Input setzen
           seg.segment (*inliers, *coefficients); //4.Ausführen
 }
```

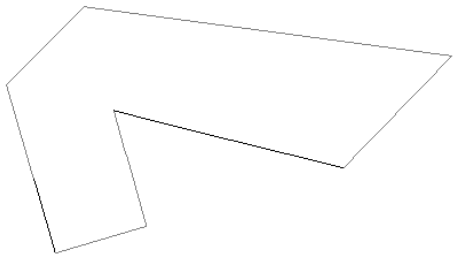


OutlierRemoval

LineIntersection



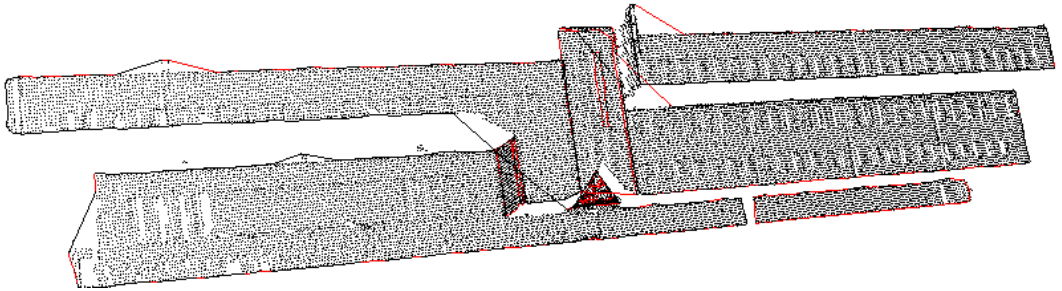




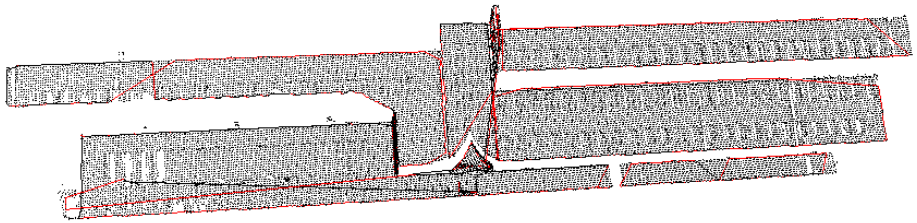
Signifikante Punkte für Fläche

Signifikante Punkte für Fläche

ConcaveHull Variante



SACSegmentation Punkte-Variante



SACSegmentation Koeffizienten-Variante