**CHAPTER 1**

**INTRODUCTION**

**1.1 Introduction to Point Clouds**

A point cloud is a set of data points in space. Point clouds are generally produced by 3D scanners, which measure a large number of points on the external surfaces of objects around them. As the output of 3D scanning processes, point clouds are used for many purposes, including to create 3D CAD models for manufactured parts, for metrology and quality inspection, and for a multitude of visualization, animation, rendering and mass customization applications.

****

**Figure 1.1** *A Point Cloud Image of a Cat*

**1.1.1 Alignment and Registration**

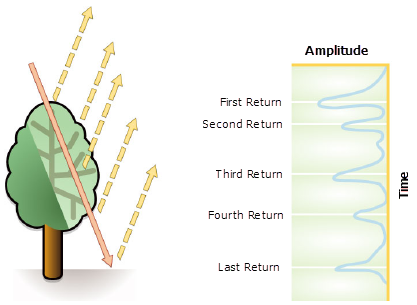
Point clouds are often aligned with 3D models or with other point clouds, a process known as point set registration. For industrial metrology or inspection using industrial computed tomography, the point cloud of a manufactured part can be aligned to an existing model and compared to check for differences. Geometric dimensions and tolerances can also be extracted directly from the point cloud.

**1.2 LIDAR**

Light Detection And Ranging (LIDAR) is an optical remote-sensing technique that uses laser light to densely sample the surface of the earth, producing highly accurate x,y,z measurements. Lidar, primarily used in airborne laser mapping applications, is emerging as a cost-effective alternative to traditional surveying techniques such as photogrammetry. Lidar produces mass point cloud datasets that can be managed, visualized, analyzed, and shared.

The major hardware components of a lidar system include a collection vehicle (aircraft, helicopter, vehicle, and tripod), laser scanner system, GPS (Global Positioning System), and INS (Inertial Navigation System). An INS system measures roll, pitch, and heading of the lidar system.

Lidar is an active optical sensor that transmits laser beams toward a target while moving through specific survey routes. The reflection of the laser from the target is detected and analyzed by receivers in the lidar sensor. These receivers record the precise time from when the laser pulse left the system to when it is returned to calculate the range distance between the sensor and the target.

****

**Figure 1.2** *Return of Lidar pulses from an Object(Tree)*

The point data is post-processed after the lidar data collection survey into highly accurate geo-referenced x,y,z coordinates by analyzing the laser time range, laser scan angle, GPS position, and INS information.

Post-processed spatially organized lidar data is known as LIDAR-Point Cloud Data. The initial point clouds are large collections of 3D elevation points, which include x, y, and z, along with additional attributes such as GPS time stamps. The specific surface features that the laser encounters are classified after the initial lidar point cloud is post-processed. Elevations for the ground, buildings, forest canopy, highway overpasses, and anything else that the laser beam encounters during the survey constitutes point cloud data.

**1.2.1 LIDAR Laser Returns**

Laser pulses emitted from a lidar system reflect from objects both on and above the ground surface: vegetation, buildings, bridges, and so on. One emitted laser pulse can return to the lidar sensor as one or many returns. The first returned laser pulse is the most significant return and will be associated with the highest feature in the landscape like a treetop or the top of a building. The first return can also represent the ground, in which case only one return will be detected by the lidar system.

Multiple returns are capable of detecting the elevations of several objects within the laser footprint of an outgoing laser pulse. The intermediate returns, in general, are used for vegetation structure, and the last return for bare-earth terrain models.The last return will not always be from a ground return. For example, consider a case where a pulse hits a thick branch on its way to the ground and the pulse does not actually reach the ground. In this case, the last return is not from the ground but from the branch that reflected the entire laser pulse.

**1.2.3 LIDAR Point Attributes**

Additional information is stored along with every x, y, and z positional value. The following lidar point attributes are maintained for each laser pulse recorded: intensity, return number, number of returns, point classification values, points that are at the edge of the flight line, RGB (red, green, and blue) values, GPS time, scan angle, and scan direction.

**1.2.4 LIDAR Sensors**

As opposed to passive sensors that detect energy naturally emitted from an object, LIDAR uses active sensors which emit their own energy source for illumination. The energy source hits objects and the reflected energy from the objects is detected and measured by sensors. LIDAR is an example of active sensor and uses LASER principle. Distance to the object is determined by recording the time between transmitted and backscattered pulses and by using the speed of light to calculate the distance traveled.

**1.3 Segmentation**

The objective of segmentation on point clouds is to spatially group points with similar properties into homogeneous regions. Segmentation is a fundamental issue in processing point clouds data acquired by LIDAR and the quality of segmentation largely determines the success of information retrieval. Unlike the image or TIN model, the point clouds do not explicitly represent topology information. As a result, most existing segmentation methods for image and TIN have encountered two difficulties. First, converting data from irregular 3-D point clouds to other models usually leads to information loss; this is particularly a serious drawback for range image based algorithms. Second, the high computation cost of converting a large volume of point data is a considerable problem for any large scale LIDAR applications. Here we investigate the strategy to develop LIDAR segmentation methods directly based on the concept of Supervoxels.

**CHAPTER 2**

**LITERATURE SURVEY**

**2.1 SEMANTIC LABELLING OF URBAN POINT CLOUD DATA by A.M.Ramiya , Rama Rao Nidamanuri , R Krishnan**

**Contribution:**

This paper uses a new object oriented methodology for semantic labelling of urban point cloud data. In addition to the geometrical information from LIDAR, they have used the spectral information for labelling of the point cloud. The coloured point cloud was segmented using colour based region growing algorithm to produce 3D segments.

**Merits:**

This method is able to label the points with an accuracy of 94% for ground classes and 82% for non-ground classes.

**Demerits:**

Classification stage requires better geometric features for improving the accuracy.

**2.2 OBJECT-ORIENTED SEMANTIC LABELLING OF SPECTRAL–SPATIAL LIDAR POINT CLOUD FOR URBAN LAND COVER CLASSIFICATION AND BUILDINGS DETECTION by Anandakumar M. Ramiya, Rama Rao Nidamanuri and R Krishnan**

**Contribution:**

This paper integrates geometric information from LiDAR point cloud data and spectral information from multispectral images for object-oriented labelling. Multiple classifier system is then applied on the features extracted from the segments for classification and also for reducing the subjectivity involved in the selection of classifier and improving the precision of the results.

**Merits:**

Accuracy of about 85% has been achieved for the classification of non-ground urban land covers.

**Demerits:**

Ambiguity arises during discriminating buildings of different shapes due to lack of adequate geometric features.

**2.3 A FRAMEWORK FOR AUTOMATIC MODELING FROM POINT CLOUD DATA by Charalambos Poullis, Member, IEEE**

**Contribution:**

A novel unsupervised clustering algorithm separates each dataset into clusters based on a hierarchical statistical analysis of the points’ geometric properties. The boundaries extracted for each cluster are refined using a fast energy minimization with graph-cuts.

**Merits:**

Uses fast boundary refinement process for the extraction of boundary points of segments.

**Demerits:**

It can handle only linear boundaries. Non-linear boundaries are considered piecewise linear.

**2.4 A METHODOLOGY FOR AUTOMATED SEGMENTATIONAND RECONSTRUCTION OF URBAN 3-D BUILDINGS FROM ALS POINT CLOUDS by Dong Chen, Liqiang Zhang, P. Takis Mathiopoulos, *Senior Member, IEEE*, and Xianfeng Huang**

**Contribution:**

In this paper, a methodology which allows automated and efficient reconstruction of three-dimensional (3-D) geometric building models from an Airborne Laser Scanning (ALS) point cloud is introduced and its performance is analyzed and evaluated. An improved random sample consensus (RANSAC) algorithm is proposed to segment the rooftop primitives, i.e., the planar patches that constitute rooftops, of each building or group of connected buildings.

**Merits:**

The algorithm successfully maintains topological consistency among primitives and avoids under- and over-segmentation with high efficiency.

**Demerits:**

Segments only planar primitives. The method is not designed to process non planar primitives, e.g., spheres, cylinders, and cones.

**2.5 A SUPERVOXEL-BASED SPECTRO-SPATIAL APPROACH FOR 3D URBAN POINT CLOUD LABELLING by Anandakumar M. Ramiya, Rama Rao Nidamanuri and Krishnan Ramakrishnan**

**Contributions:**

This paper proposes a novel 3D object based classification framework for labelling urban LIDAR point cloud using a computer vision technique called Supervoxels. Supervoxels are generated by over-segmenting the coloured point cloud using the voxel-based cloud connectivity algorithm in the geometric space. The segments are classified by different machine learning techniques based on several spectral and geometric features extracted from the segments. All the points within a labelled segment are assigned the same segment label.

**Merits:**

An overall classification accuracy of 90% is achieved by this method.

**Demerits:**

There are some cases in which there is a misclassification when the point density is low.

**CHAPTER 3**

**SYSTEM ANALYSIS**

**3.1 EXISTING SYSTEM**

3D Point Cloud Segmentation is the process of classifying point clouds into multiple homogeneous regions, the points in the same region will have the same properties. The segmentation is challenging because of high redundancy, uneven sampling density, and lack explicit structure of point cloud data. This problem has many applications in robotics such as intelligent vehicles, autonomous mapping and navigation. Many authors have introduced different approaches and algorithms.

The existing methodologies that have been suggested for the segmentation of 3D point cloud data can be categorized into five classes: Edge based methods, Region based methods, Attributes based methods, Model based methods, and Graph based methods.

**3.1.1 Edge based methods:**

Edges describe the characteristics about the shape of objects. Edge based methods detect the boundaries of several regions in the point clouds to obtain segmented regions. The principle of these methods is locate the points which have rapid change in the intensity.Although edge based methods allow fast segmentation but they have accuracy problems because all of them are very sensitive with noise and uneven density of point clouds, situations that commonly occur in point cloud data.

**3.1.2 Region based methods:**

Region based methods use neighborhood information to combine nearby points that have similar properties to obtain isolated regions and consequently find dissimilarity between the different regions. Region based methods are more accurate to noise than edge based methods. But they have problem with over or under segmentation and determining region borders accurately. We divide region based methods into two categories: seeded-region (or bottom-up) methods and unseeded-region (or top-down) methods.

Seeded-region method start the segmentation process by choosing a number of seed points, then from these points, each region will grow by adding neighbour points if they satisfy certain criterion or compatibility thresholds. Seeded-region approaches are highly dependent on selected seed points. Inaccurate choosing seed points will affect the segmentation process and can cause under or over segmentation.

Unseeded-region methods, on the contrary, based on the top-down approach. First, all points are grouped into one region. Then the subdivision process starts to divide it into smaller regions. As long as a chosen figure of merit for fitting is higher than a threshold, region subdivision is continued. The main difficulty of unseeded-region methods are to decide where and how to subdivide.

**3.1.3 Attributes based methods:**

Attributes based methods are robust approaches based on clustering attributes of point cloud data. These methods include two separate steps. The first step is attribute computation, in the second step, point clouds will be clustered based on the computed attributes. The clustering methods offer flexibility in accommodating spatial relation and attributes to incorporate different cues into the segmentation process. A limitation of these approaches is they are highly dependent on the quality of derived attributes. The attributes of point cloud data should be computed precisely to produce the best separation among different classes.

**3.1.4 Model based methods:**

Model based methods use geometric primitive shapes (e.g. sphere, cone, plane, and cylinder) for grouping points. The points which have the same mathematical representation are grouped as one segment. *Fischer et al.* introduced a well known algorithm called RANSAC (RANdom SAmple Consensus). RANSAC is a robust model and is used to detect mathematical features like straight lines, circles, etc. This method is now the state of the art for model fitting. In 3D point cloud segmentation, many subsequent works have inherited this as initial algorithm. The main limitation of these methods is their inaccuracy when dealing with different point cloud sources.

**3.1.5 Graph based methods:**

Graph based methods consider the point clouds in terms of a graph. A simple model is each vertex corresponds to a point in the data and the edges connect to certain pairs of neighboring points. Many works on graph based methods are cast into a probabilistic inference model such as Conditional Random Fields (CRF).To compare with other methods, graph based methods can segment complex scenes in point cloud data include noise or uneven density with better results. However, these methods are usually cannot run in real time. Some of them may need offline training step or require special co-registered sensors and camera system.

**3.2 PROPOSED SYSTEM**

The system that is proposed in this model is a hybrid method from most of all the existing methods. It is clear from the previous researches that accuracy of 3D point cloud data processing increases by implementing Supervoxel concept. The proposed system uses the region growing model for clustering the Voxels by implementing Octree/ k-d tree method. The adjacency graph (a graph based approach) for supervoxels is maintained, which is used extensively to generate the supervoxels and to determine the adjaceny among the resulting supervoxels. And for supervoxel creation from the clustered voxel cloud it uses a Voxel Cloud Connectivity Segmentation (VCCS) algorithm, which is also a region growing approach.

To maintain efficiency, VCCS does not search globally, but rather only considers points within the neighbourhood. Supervoxels maintain adjacency relations in voxelized 3D space. Hence for creating meaningful segments from the clustered supervoxels a bottom-up approach named Constrained Planar Cuts (CPC) algorithm is used for segmenting the objects from the scene.

This supervoxel based 3D CPC segmentation approach can yield fairly high classification accuracy with less computational demand.

By using the extracted output point cloud data we use Region growing to merge the points that are close enough in terms of the smoothness constraint and produce the Clusters.

Minimum Cut based Segmentation performs bipartite segmentation and divided into two parts.  Each edge has a weight value according to the distance separating them. The greater the distance, the more likely the edge will be cut. This can be evaluated by supervoxel segmentation and compared.

**CHAPTER 4**

**SYSTEM DESIGN**

**4.1 SYSTEM ARCHITECTURE**

The architecture of the proposed system is given as follows. This architecture will provide the overall representation of the system.

Supervoxel Creation &Clustering

Preprocessing

DSM

Octree

Outdoor Scene

Voxels

Voxel

3D Point Cloud

DTM Filter

VCCS

Supervoxels

Segmentation using Region Growing, Min cut, LCCP ,CPC

Evaluation Of Supervoxel Segmentation

Segmented point cloud data

**Figure 4.1 :** *Outline of the approach used in this paper. Pictorial representations are given for the input stage of the workflow.*

A brief outline of the methodological framework is given in Figure 4.1. The proposed architecture includes five main stages: Data Pre-processing, Voxel Clustering, Supervoxel Creation, Segmentation and Evaluation of different segmentations.

**CHAPTER 5**

**SYSTEM MODULES**

**5. MODULES**

The proposed system consists of five modules which are as follows.

1. Data Pre-Processing

2. Voxel Clustering

3. Supervoxel Creation

4. Segmentation

5. Evaluation

**5.1 Data Pre-Processing**

LIDAR scans typically generate point cloud datasets of varying point densities. Especially in Indoor 3D point cloud data such varying densities results in performance degradation during Voxel creation. Additionally, measurement errors lead to sparse outliers which corrupt the Supervoxel results even more.

**5.1.1 Noise Removal**

These above errors(noise) complicates the estimation of local point cloud characteristics such as surface normals or curvature changes, leading to erroneous values, which in turn might cause point cloud registration failures for Segmentation. Some of these irregularities can be solved by performing a statistical analysis on each point’s neighborhood, and trimming those which do not meet a certain criteria. The Statistical Outlier Removal (SOR) Filter is based on the computation of the distribution of point to neighbors distances in the input dataset. For each point, we compute the mean distance from it to all its neighbors.

Raw 3D Point Cloud Data

Progressive Morphological Filter

Non-Ground 3D Point Cloud Data

**(a)** *Outdoor Dataset*

**Figure 5.1 :** *Dataflow diagram for Pre-Processing*

In simple terms, it computes first the average distance of each point to its neighbors (considering k nearest neighbors for each - k).Then it rejects the points that are farther than the average distance plus a number of times the standard deviation.

**5.1.2 DTM Filter**

The Digital Terrain Model (DTM) represents the bare ground surface without any objects like plants and buildings. In order to generate a 3D point cloud of Non-Ground classes, measurements from non-ground features such as buildings, vehicles, and vegetation have to be classified and extracted from the given input 3D point cloud data. In this paper, a Progressive Morphological Filter (DTM Filter) was used to detect non-ground LIDAR measurements which was developed by *K.Zhang et al*.By gradually increasing the window size of the filter and using elevation difference thresholds, the measurements of vehicles, vegetation, and buildings are preserved, while ground data points are removed.

Local raw airborne LIDAR data and generate a minimum surface grid

Morphological Filtering

Filtered terrain surface model

Increase the size of the filter and determine the elevation difference threshold

No

Size\_of\_filter > MAX window size

Yes

Non-ground points

Ground points (DTM)

**Figure 5.2 :** *Outline of the approach used in this paper to extract the non-ground points from the given input 3D point cloud.*

Our areas of interest in urban city modelling are objects like vegetation, buildings, roofs, vehicles, etc., except ground classes.

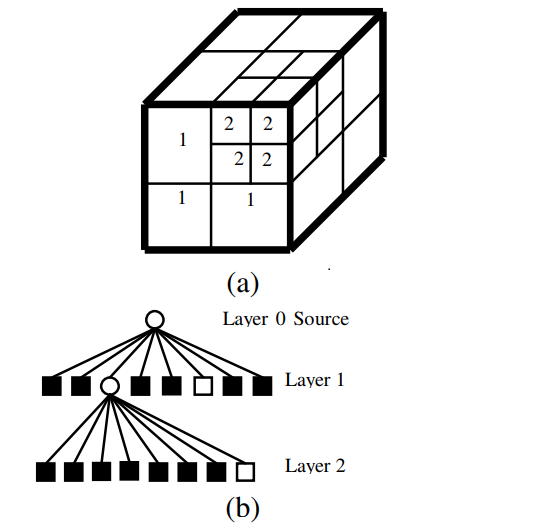
**5.2 VOXEL**

The continuous 3D space containing the point cloud can be discretized by the volume based on the density and distance of the point in the LIDAR point cloud. Each discrete unit in 3D-space is called a Volumetric Element (voxel). One voxel represents a 3D pixel in the 3D space.

**5.2.1 VOXEL CLUSTERING**

Voxels are generated from the 3D points based on a structuring algorithm such as Octree or k-d tree methods, which group points based on neighbourhood and spatial property. An Octree-structure based split-and-merge method for organizing lidar point cloud into clusters of 3D planes is used here which was introduced by *M.Wang et al*.

An Octree is a tree-based data structure for managing sparse 3-D data. Each internal node has exactly eight children. The method is hierarchically splitting the point cloud set on the octree structure until the points contained in each sub-node are coplanar, or say distributed in a 3D plane or less than 3 points.



**Figure 5.3:** *Octree Structure (a): Divided sub-spaces (b): Tree representation*

The Octree structure uses spatial partitioning and neighbor search within point cloud data to cluster the voxels. The neighbouring 3D planes with similar attribute are merged after splitting to form larger planes. Each 3D plane is termed as ‘voxels’ which contains a cluster of voxel.

**5.3 SUPERVOXEL CREATION**

The voxels are further grouped based on geometrical and spectral relationships among them to create super-voxels.

Supervoxels were created using the Voxel Cloud Connectivity Segmentation (VCCS) (*Papon et al. 2013*), which over-segments a 3D point cloud into an adjacency graph of supervoxels (a 3D analog of superpixels). During the implementation of VCCS for supervoxel creation, voxels were created using the octree data structuring algorithm. The spatial relationship among voxels was established by generating an adjacency graph based on 26 neighbourhood connectivities of the voxel. From the voxelized space, seed voxels were selected based on the chosen seed resolution, these having been grown to form supervoxels. These seed voxels were evenly distributed in space on the voxel grid.

Supervoxel feature vector initialization

Seed Voxels Selection

Voxelized Space

Distance computation

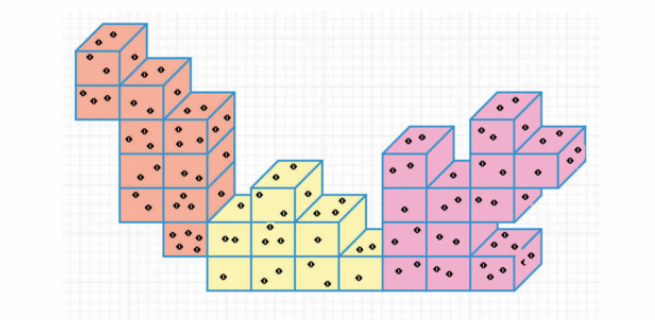
Centroid Updation & Stabilization

Geometrically constrained & disjoint Supervoxels

**Figure 5.4 :** *Dataflow Diagram for VCCS Algorithm*

Supervoxels have two important properties; they are evenly distributed across the 3D space, and they cannot cross boundaries unless the underlying voxels are spatial connected. The former is accomplished by seeding supervoxels directly in the cloud, rather than the projected plane, while the latter uses an octree structure which maintains adjacency information of leaves. Supervoxels maintain adjacency relations in voxelized 3D space; specifically, 26-adjacency- that is neighboring voxels are those that share a face, edge, or vertex.

The adjacency graph of supervoxels (and the underlying voxels) is maintained efficiently within the octree by specifying that neighbors are voxels within R\_voxel of one another, where R\_voxel specifies the octree leaf resolution. This adjacency graph is used extensively for both the region growing used to generate the supervoxels, as well as determining adjacency of the resulting supervoxels themselves.



**Figure 5.5 :** *Graphical illustration of Voxels and Supervoxels. Voxels are discrete units containing many points (voxel). A group of voxels forms Supervoxels (different colors represent different Supervoxels).*

**5.3.1 SUPERVOXEL CLUSTERING**

Voxel Cloud Connectivity Segmentation(VCCS) is a region growing method which incrementally expands supervoxels from a set of seed points distributed evenly in space on a grid with resolution R\_seed. To maintain efficiency, VCCS does not search globally, but rather only considers points within R\_seed of the seed center. Additionally, seeds which are isolated are filtered out by establishing a small search radius R\_search around each seed and removing seeds which do not have sufficient neighbor voxels connected to them.

For each seed voxel, we initialized the supervoxel feature vector consisting of spatial, colour, and normal estimates of points within the voxel. The distance is computed from the supervoxel centres to the adjacent voxels within a limited search space using the equation.

where Dc is the colour distance in the CIELab space (Connolly and Fleiss 1997) normalized by constant m; Ds is the spatial distance normalized by the seed resolution R\_seed; is the distance in the FPFH (Fast Point Feature Histogram) space using the histogram intersection kernel (Hik) (*Rusu 2009*); and λ, μ, ε are the parameters controlling the colour, spatial distance, and geometric similarity, respectively, between the voxels. We gave equal weightage to spatial distance and colour, with distance in the FPFH space having a slightly higher weight. The voxels were assigned to their neighbouring supervoxel centres based on the distance measure. Once all the voxels were assigned to the supervoxel centres, their centroids were updated. The process was carried out iteratively until the centroids stabilized.

This algorithm uses a local region growing variant of k-means clustering to generate individual supervoxels i= (i , i , Ni), with centroid i , normal vector i, and edges to adjacent supervoxels e ∈ Ni .The supervoxels thus created were geometrically constrained and disjoint in the 3D space.

**5.4 3D Segmentation using Constrained Planar Cuts :**

We implemented the recently proposed Constrained Planar Cuts (CPC)-based segmentation (*M.Schoeler et al.*) for creating meaningful clusters from the LIDAR Point Cloud data coupled with spectral data. The CPC algorithm uses a bottom-up method for segmenting 3D point clouds into functional parts which does not require supervision and achieves equally good results in processing RGB-D datasets.This segmentation approach relies on local concavities as an indicator for inter-part boundaries. i.e. the convexity/concavity between two neighbouring patches to create realistic segments (objects). It employs a novel locally constrained geometrical boundary model which uses greedy cuts through a local concavity graph. Only planar cuts are considered and evaluated using a cost function, which rewards cuts orthogonal to concave edges. Additionally, a local clustering constraint is applied to ensure the partitioning only affects relevant locally concave regions.

Establish Geometrical Connectivity between neighbouring supervoxels

Identify Local connectivity and cuts between adjacent patches

Supervoxels

Compare & Merge two neighbouring Supervoxels if satisfies Convexity criteria

Segmented Supervoxels

Compute Centroid & Normals for individual Supervoxels

**Figure 5.6 :** *Dataflow Diagram for CPC Algorithm*

**5.4.1 Local concavity evidence extraction :**

Seed points for the segmentation are initialized using a regular grid which samples the occupied space uniformly using an adjacency-octree structure. Segments are expanded from the seed points, governed by a similarity measure calculated in a feature space consisting of spatial extent, color, and normal difference.

Once we have the supervoxel adjacency graph, we use the Locally Convex Connected Patches (LCCP) algorithm to label edges in the graph as either convex or concave. Considering two adjacent supervoxels with centroids at 1, 2 and normals 1, 2 we treat their connection as convex if1 ∙  - 2 ∙  ≥ 0 ,

With



Likewise, a connection is concave if 1 ∙  - 2 ∙ < 0

The Supervoxels S1 and S2 are said to be connected if they satisfy theequation

CCb = (1 - 2) ∙ > 0 ˅ (β < βthresh)

where CCb defines the basic convexity criteria and β denotes the angle between the normals n1 and n2. To reduce errors in segmentation a threshold, βthresh, was used to merge concave surfaces with low curvature values. The segments thus created were object primitives, which better describes the object geometry.

**5.4.2 Semiglobal Partitioning :**

To make use of the concavity information we will now introduce a recursive algorithm for partitioning parts which can cut convex edges as well. Beginning with the concave/convex-labeled supervoxel adjacency graph, we search for euclidean splits which maximize a scoring function. In this work we use a planar model, but other boundary models, such as constrained paraboloids are possible as well. In each level we do one cut per segment from the former level. All segments are cut independently, that is, other segments are ignored. Cuts do not necessarily bi-section segments (as most graph cut methods), but as we cut in euclidean space, can split into multiple new segments with a single cut. This also allows us to use evidence from multiple scattered local concavities from different parts to induce and refine a globally optimal combined cut.

**5.4.2.1 Euclidean edge cloud :**

An object shall be cut at edges connecting supervoxels. Consequently, we start by converting the adjacency graph into a Euclidean Edge Cloud (EEC), where each point represents an edge in the adjacency graph. The point-coordinate is set to the average of the supervoxels it connects (1,2). Additionally, the points maintain the direction of the edge together with the angle α between the normals of both supervoxels (1, 2)

cos-1 (1. 2)



We will use α < 0 to describe convex edges and α > 0 to denote concavities. The EEC has the advantage of efficiently storing the edge information and bridging the gap between the abstract adjacency graph representation and the euclidean boundary model.

**5.4.2.2 Geometrically constrained partitioning :**

To find the planes for cutting we use a locally constrained, directionally weighted sample consensus algorithm and apply it on the edge cloud as follows. While canonical RANSAC treats points equally, here we extend it with Weighted RANSAC, allowing each point to have a weight. Points with high positive weights encourage RANSAC to include them in the model, whereas points with low or negative weights will penalize a model containing them. All points are used for model scoring, while only points with weights ωi > 0 are used for model estimation. We normalize the score by the number of inliers in the support region, leading to a scale-invariant scoring. With Pm being the set of points which lie within the support region (i.e. within a distance below a predefined threshold τ of the model m ) and |x| denoting the cardinality of set x , the score can thus be calculated using the equation:



Weighted RANSAC will favor the split along as many concave boundaries as possible. To deal with such cases, we use Directional Weighted RANSAC as follows. Let m denote the vector perpendicular to the surface of model m and i  the ith  edge direction calculated from the above equations. To favor cutting edges with a plane that is orthogonal to the edge, we add a term to the scoring of concavities,

The notation · refers to the dot-product and |x| to cardinality or absolute value. The idea behind the equation (ti) is that convexities should always penalize regardless of orientation, whereas concavities hint at a direction for the cutting. Due to perpendicular vectors |1. 1| and |1. 2| the directional concavity weights for the cut in B are almost decreased to zero.

**5.4.2.3 Locally constrained cutting:**

The last step of this algorithm introduces locally constrained cutting of segments. In this algorithm, concavities separating several parts can sometimes leads to cases where regions with strong concavities induce a global cut which will split off a convex part of the object. To prevent this kind of over-segmentation we constrain our cuts to regions around local concavities as follows.

Given the set of edge-points Pm located within the support region of a model, we start with a euclidean clustering of all edge-points using a cluster threshold equal to the seed-size of the supervoxels. Using nPm⊂ Pm to denote the set of points in the n-th cluster, we modify above equation to operate on the local clusters instead of Pm:



As this operation is too expensive to be employed at each model evaluation step of the RANSAC algorithm, we only apply it to the highest scoring model. Only edges with a cluster-score nSm  ≥ Smin  will be cut.

This whole cutting procedure is repeated recursively on the newly generated segments and terminates if no cuts can be found which exceed the minimum score Smin or if the segment consists of less than Nmin supervoxels.

**5.4.3 Region Growing**

The purpose of the said algorithm is to merge the points that are close enough in terms of the smoothness constraint. Thereby, the output of this algorithm is the set of clusters, were each cluster is a set of points that are considered to be a part of the same smooth surface. The work of this algorithm is based on the comparison of the angles between the points normals.

It needs to be done because the region begins its growth from the point that has the minimum curvature value. The reason for this is that the point with the minimum curvature is located in the flat area (growth from the flattest area allows to reduce the total number of segments).

So we have the sorted cloud. Until there are unlabeled points in the cloud, algorithm picks up the point with minimum curvature value and starts the growth of the region. This process occurs as follows:

* The picked point is added to the set called seeds.
* For every seed point algorithm finds neighbouring points.
  + - * + Every neighbour is tested for the angle between its normal and normal of the current seed point. If the angle is less than threshold value then current point is added to the current region.
        + After that every neighbour is tested for the curvature value. If the curvature is less than threshold value then this point is added to the seeds.
        + Current seed is removed from the seeds.

If the seeds set becomes empty this means that the algorithm has grown the region and the process is repeated from the beginning. You can find the pseudocode for the said algorithm below.

Inputs:

* Point cloud = \{P\}
* Point normals = \{N\}
* Points curvatures = \{c\}
* Neighbour finding function \Omega(.)
* Curvature threshold c_{th}
* Angle threshold \theta_{th}

Initialize:

* Region list {R}\leftarrow{\O}
* Available points list \{A\}\leftarrow\{1,...,|P|\}

**5.4.4 Color Based Region Growing:**

This algorithm is based on the same concept as the pcl::RegionGrowing  . There are two main differences in the color-based algorithm. The first one is that it uses color instead of normals. The second is that it uses the merging algorithm for over- and under- segmentation control. After the segmentation, an attempt for merging clusters with close colors is made. Two neighbouring clusters with a small difference between average color are merged together. Then the second merging step takes place. During this step every single cluster is verified by the number of points that it contains. If this number is less than the user-defined value than current cluster is merged with the closest neighbouring cluster.

**5.4.5 Minimum Cut Based Segmentation:**

This algorithm makes a binary segmentation of the given input cloud. Having objects center and its radius the algorithm divides the cloud on two sets: foreground and background points (points that belong to the object and those that do not belong).

For the given point cloud algorithm constructs the graph that contains every single point of the cloud as a set of vertices and two more vertices called source and sink. Every vertex of the graph that corresponds to the point is connected with source and sink with the edges. In addition to these, every vertex (except source and sink) has edges that connect the corresponding point with its nearest neighbours.

Algorithm assigns weights for every edge. There are three different types of weight. Let’s examine them:

* + - * First of all it assigns weight to the edges between clouds points. This weight is called smooth cost and is calculated by the formula:

**Smooth cost=**

Here dist is the distance between points. The farther away the points are, the more is probability that the edge will be cut.

* + - * Next step the algorithm sets data cost. It consists of foreground and background penalties. The first one is the weight for those edges that connect clouds points with the source vertex and has the constant user-defined value. The second one is assigned to the edges that connect points with the sink vertex and is calculated by the formula:

**Background Penalty = **

Here distanceToCenter is the distance to the expected center of the object in the horizontal plane:

**distanceToCenter =**

Radius that occurs in the formula is the input parameter for this algorithm and can be roughly considered as the range from objects center outside of which there are no points that belong to foreground (objects horizontal radius).

After all the preparations the search of the minimum cut is made. Based on an analysis of this cut, cloud is divided on foreground and background points.

**5.5 Evaluation:**

Here we evaluate the different supervoxel segmentation approaches on LIDAR dataset, we use the segmentation such as Region Growing, Color based Region Growing, Minimum Cut based Segmentation, Locally Convex connected Patches(LCCP) and Constrained Planar Cuts(CPC).

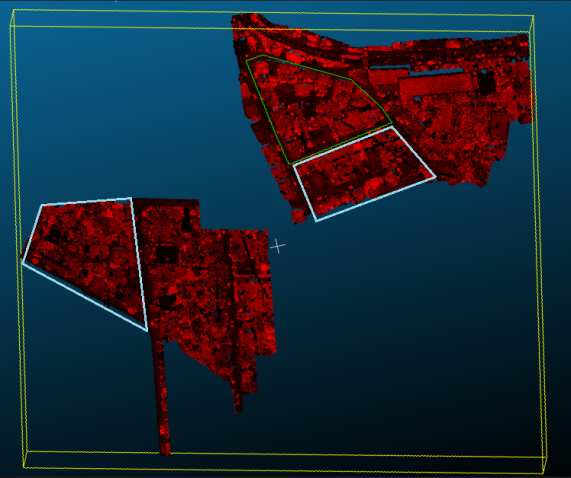
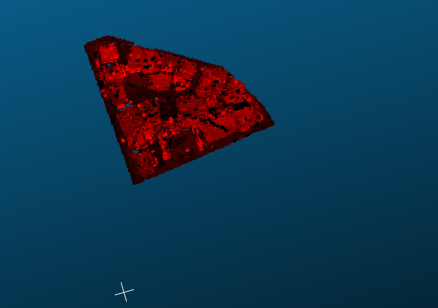
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Results** | **Region Growing** | **Minimum Cut Based** | **LCCP** | **CPC** |
| Advantages | Choose multiple criteria at same time | Best to get foreground as accurately | Faster than minimum cut segmentation | Efficiently stores the edge information |
| Accuracy | 70% | 90% | 82% | 89% |
| Disadvantages | Sensitive to noise | High Computational cost | It requires high end Processor | It too requires high end processor |

**CHAPTER 6**

**DATASETS USED**

**6.1. VAIHINGEN DATASET**

This data set was collected over the city of Vaihingen, Germany during August 2008 and was provided to us by the ISPRS as part of the ISPRS test project on urban classification and 3D building reconstruction (Cramer 2010). The lidar data were collected using a Leica ALS50 at a flying altitude of 500 m above ground. There was a simultaneous capture of the spectral information using an Integraph/ZI DMC. The multi-spectral images, captured in the green, red, and near-infrared parts of the electromagnetic spectrum, have a spatial resolution of 5 cm and radiometric resolution of 11 bits.

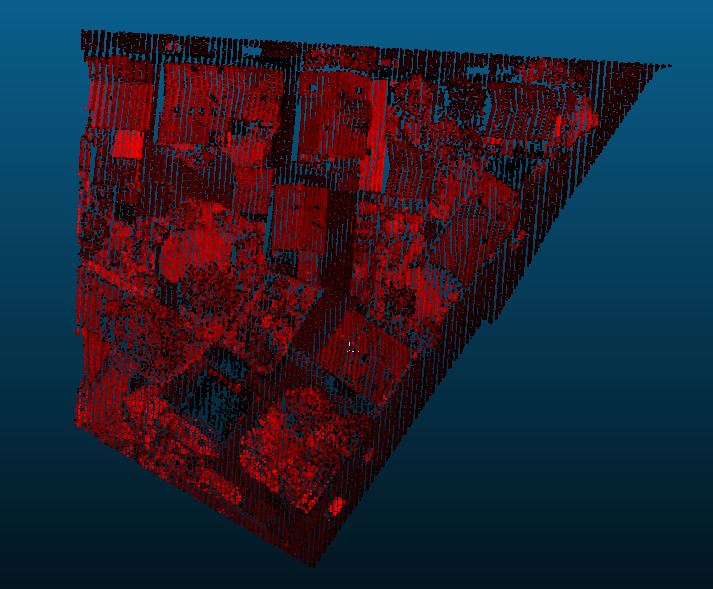
 

1. **(b)**

**Figure 6.1 :***Outdoor - Vaihingen dataset*

*(a): Original dataset downloaded from the isprs site.*

*(b): Study area-1(cropped) from the original dataset.*

****

**(c)**

**Figure 6.1 :***Outdoor - Vaihingen dataset*

*(c): Study area-2(cropped) from the original dataset.*

****

**Figure 6.1.1 :***Outdoor - Vaihingen dataset - Orthophoto image*

**CHAPTER 7**

**SYSTEM SPECIFICATION**

**7.1 HARDWARE REQUIREMENTS**

|  |  |
| --- | --- |
| * CPU SPEED * CPU TYPE | : 3.1 GHz  : 64 bit |
| * STORAGE | : 200 GB |
| * OS | : Microsoft Windows 10 64 bit |
| * RAM | : 8 GB |

**7.2 SOFTWARE REQUIREMENTS**

* Cloud Compare v2.10
* CMake v3.10.2
* Point Cloud Library (PCL) v1.8
* Microsoft Visual Studio 2013

**7.2.1 CLOUD COMPARE v2.10**

CloudCompare is a 3D point cloud processing software (such as those obtained with a laser scanner). It can also handle triangular meshes and calibrated images.

CloudCompare provides a set of basic tools for manually editing and rendering 3D points clouds and triangular meshes. It also offers various advanced processing algorithms, among which methods for performing:

* Projections (axis-based, cylinder or a cone *unrolling*, ...)
* Registration (ICP, ...)
* Distance computation (cloud-cloud or cloud-mesh nearest neighbor distance, ..)
* Statistics computation (spatial Chi-squared test, ...)
* Segmentation (connected components labeling, front propagation based, ...)
* Geometric features estimation (density, curvature, roughness, geological plane orientation, ...)

CloudCompare can handle unlimited scalar fields per point cloud on which various dedicated algorithms can be applied (smoothing, gradient evaluation, statistics, etc.). A dynamic color rendering system helps the user to visualize per-point scalar fields in an efficient way. Therefore, CloudCompare can also be used to visualize N-D data.

**7.2.2 CMake v3.7.1**

CMake is a cross-platform free and open-source software for managing the build process of software using a compiler-independent method. It supports directory hierarchies and applications that depend on multiple libraries. It is used in conjunction with native build environments such as make, Apple's Xcode, and Microsoft Visual Studio. It has minimal dependencies, requiring only a C++ compiler on its own build system.

**7.2.3 Point Cloud Library (PCL) v1.8**

Point Cloud Library (PCL) is an open-source library of algorithms for point cloud processing tasks and 3D geometry processing, in three-dimensional computer vision. The library contains algorithms for feature estimation, surface reconstruction, 3D registration, model fitting, and segmentation. It is written in C++ and released under the BSD license.

**7.2.4 Microsoft Visual Studio 2013**

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as web sites, web apps, web services and mobile apps. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight. It can produce both native code and managed code.

Visual Studio includes a code editor supporting IntelliSense (the code completion component) as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a code profiler, forms designer for building GUI applications, web designer, class designer, and database schema designer. It accepts plug-ins that enhance the functionality at almost every level—including adding support for source control systems (like Subversion) and adding new toolsets like editors and visual designers for domain-specific languages or toolsets for other aspects of the software development lifecycle (like the Team Foundation Server client: Team Explorer).

Visual Studio supports 36 different programming languages and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML and CSS. Support for other languages such as Python, Ruby, Node.js, and M among others is available via plug-ins. Javaand J# were supported in the past.

**CHAPTER 8**

**SYSTEM IMPLEMENTATION**

**8.1 Pre-Processing**

LIDAR scans typically generate point cloud datasets of varying point densities. Especially in 3D point cloud data with varying densities results in performance degradation during Voxel creation.This complicates the estimation of local point cloud characteristics such as surface normals or curvature changes, leading to erroneous values, which in turn might cause point cloud registration failures for Segmentation.Some of these irregularities can be solved by performing pre-processing on each point’s neighborhood, and trimming those which do not meet a certain criteria.

**Pre-Processing – Progressive Morphological Filter**

**Step 1:** Local raw airborne LIDAR data and generate a minimum surface grid.

*Repeat*

**Step 2:** Do Morphological Filtering

**Step 3:** Get the Filtered terrain surface model

**Step 4:** Increase the size of the filter and determine the elevation difference threshold.

*Until Size\_of\_filter > MAX window size*

**Step 5:** Non-ground points are extracted.

**8.2 Supervoxel Clustering**

Supervoxels were created using the Voxel Cloud Connectivity Segmentation (VCCS) which over-segments a 3D point cloud into an adjacency graph of supervoxels (a 2D analog of superpixels). During the implementation of VCCS for supervoxel creation, voxels were created using the octree data structuring algorithm.

**Supervoxel Clustering – VCCS Algorithm**

**Step 1:** Load the input voxel cloud based on the input argument.

**Step 2:** Initialize the data structure(Multi-map) to extract the Supervoxels.

*Repeat*

**Step 3:** Extract the Supervoxel adjacency list (in the form of a Multi-map of label adjacencies) for each seed-voxel.

*Until all the voxels are processed.*

**Step 4:** Iterate through the Multi-map, to create a point cloud of the centroids of each Supervoxel's neighbours.

**Step 5:** Create a Supervoxel graph using a drawing helper function to draw a star polygon mesh of the Supervoxel centroid to all of its neighbors centroids.

**8.3 CPC Segmentation**

We implemented the recently proposed Constrained Planar Cuts (CPC)-based segmentation for creating meaningful clusters from the LIDAR Point Cloud data coupled with spectral data. The CPC algorithm uses a bottom-up method for segmenting 3D point clouds into functional parts which does not require supervision and achieves equally good results in processing RGB-D datasets.

**Segmentation – CPC Algorithm**

*Repeat*

**Step 1:**  Identify Local connectivity and cuts between adjacent patches.

**Step 2:** Establish Geometrical Connectivity between neighbouring Supervoxels.

**Step 3:** Compute Centroid & Normals for individual Supervoxels.

**Step 4:** Compare two neighbouring Supervoxels for defining Connectivity between them.

**Step 5:** Merge two Supervoxels if satisfies Convexity criteria.

*Until all Supervoxels are processed.*

**8.4 Region Growing:**

The purpose of the said algorithm is to merge the points that are close enough in terms of the smoothness constraint. Thereby, the output of this algorithm is the set of clusters, were each cluster is a set of points that are considered to be a part of the same smooth surface. The work of this algorithm is based on the comparison of the angles between the points normals.

**Segmentation – Region Growing**

**While**  **** is not empty **do**

* + Current region 
  + Current seeds 
  + Point with minimum curvature in





\{A\}\leftarrow\{A\}\setminus P_{min}

**for** i=0 to **size** ( \{S_c\} ) **do**

Find nearest neighbours of current seed point

\{B_c\}\leftarrow\Omega(S_c\{i\})

**for** j=0 to **size** ( \{B_c\} ) **do**

Current neighbour point P_j\leftarrow B_c\{j\}

**If** \{A\} contains P_j and cos^{-1}(|(N\{S_c\{i\}\},N\{S_c\{j\}\})|)<\theta_{th} **then**

\{R_c\}\leftarrow\{R_c\}\cup P_j

\{A\}\leftarrow\{A\}\setminus P_j

**If** c\{P_j\}<c_{th} **then**

\{S_c\}\leftarrow\{S_c\}\cup P_j

**end if**

**end if**

Add current region to global segment list \{R\}\leftarrow\{R\}\cup\{R_c\}

**end while**

**Return** \{R\}

*Clusters are created*

**8.5 Minimum Cut Based Segmentation:**

This algorithm makes a binary segmentation of the given input cloud. Having objects center and its radius the algorithm divides the cloud on two sets: foreground and background points (points that belong to the object and those that do not belong).

**Segmentation- Min Cut based**

*Repeat*

**Step 1**: Every vertex of the graph that corresponds to the point is

connected with source and sink with the edges.

**Step 2:** In addition to these, every vertex (except source and sink) has

edges that connect the corresponding point with its nearest

neighbours.

**Step 3:** Algorithm assigns weight for each edges.

**Step 4**: The farther away the points are, the more is probability that the edge

will be cut.

**Step 5:**  Next step the algorithm sets data cost.

**Step 6:** After all the preparations the search of the minimum cut is made.

Foreground & Background points

**CHAPTER 9**

**CONCLUSION AND FUTURE WORKS**

This paper proposes the use of Constrained Planar Cuts (CPC) Algorithm for segmentation of LIDAR Point Clouds. This method uses local concavities as an indicator for inter-part boundaries. This criterion is efficient to compute and generalizes well across different object classes.

We used voxel cloud connectivity for creating supervoxels. The supervoxels created using the cloud connectivity algorithm were combined to form segments. The segments can be classified with different machine learning algorithms based on various geometrical and spectral features. The proposed methodology was tested on four different data sets of varying point densities.

The results indicate that the supervoxel based 3D CPC segmentation approach can yield fairly high classification accuracy with less computational demand. Especially, when the point density is high, the proposed algorithm can give accurate classification results with low computational demand. The main advantage of this method is that by employing planar cuts, it reduces the computational complexity of Segmentation when compared to LCCP.

To achieve better classification accuracy and thereby better point cloud labelling, it is important to choose the best features representing the segments. In this study, we extracted one of the geometrical features from the segments created. With the help of those extracted features we can classify and label the objects (viz., roof, tree…) with high accuracy. An important aspect is that the extracted features should be fed into a proper classifier to achieve better and desired accuracy.

**APPENDIX**

**A.SAMPLE CODE**

**Sample code – Digital Terrain Model Filter**

pcl::ProgressiveMorphologicalFilter<pcl::PointXYZ>pmf;

pmf.setInputCloud (cloud);

pmf.setMaxWindowSize (20);

pmf.setSlope (1.0f);

pmf.setInitialDistance (0.5f);

pmf.setMaxDistance (3.0f);

pmf.extract (ground->indices);

pcl::ExtractIndices<pcl::PointXYZ>extract; *// Create the filtering object*

extract.setInputCloud (cloud);

extract.setIndices (ground);

extract.filter (\*cloud\_filtered);

**Sample code – VCCS Supervoxel Clustering**

std::multimap<uint32\_t,uint32\_t>:: iteratorlabel\_itr=supervoxel\_adjacency.begin ();

for ( ; label\_itr!=supervoxel\_adjacency.end (); ) { *//First get the label*

uint32\_t supervoxel\_label=label\_itr->first;

*//Now get the supervoxel corresponding to the label*

pcl::Supervoxel<PointT>::Ptrsupervoxel=supervoxel\_clusters.at (supervoxel\_label);

*//Now we need to iterate through the adjacent supervoxels and make a point cloud*

PointCloudTadjacent\_supervoxel\_centers;

std::multimap<uint32\_t,uint32\_t>::iteratoradjacent\_itr=supervoxel\_adjacency.equal\_range (supervoxel\_label).first;

for ( ; adjacent\_itr!=supervoxel\_adjacency.equal\_range (supervoxel\_label).second; ++adjacent\_itr) {

pcl::Supervoxel<PointT>::Ptrneighbor\_supervoxel=supervoxel\_clusters.at (adjacent\_itr->second);

adjacent\_supervoxel\_centers.push\_back (neighbor\_supervoxel->centroid\_); }

*// Now we make a name for this polygon and Move iterator forward to next label*

label\_itr=supervoxel\_adjacency.upper\_bound (supervoxel\_label); }

**Sample code – LCCP 3D Segmentation**

template <typename PointT> void

pcl::LCCPSegmentation<PointT>::calculateConvexConnections (SupervoxelAdjacencyList& adjacency\_list\_arg){

bool is\_convex;

EdgeIterator edge\_itr, edge\_itr\_end, next\_edge;

boost::tie (edge\_itr, edge\_itr\_end) = boost::edges (adjacency\_list\_arg);

for (next\_edge = edge\_itr; edge\_itr != edge\_itr\_end; edge\_itr = next\_edge){

next\_edge++; *// next\_edge iterator is neccessary, because removing an edge //invalidates the iterator to the current edge*

uint32\_t source\_sv\_label = adjacency\_list\_arg[ boost::source ( ) ];

uint32\_t target\_sv\_label = adjacency\_list\_arg[boost::target ( )];

float normal\_difference;

is\_convex = connIsConvex (source\_sv\_label, target\_sv\_label, normal\_difference);

adjacency\_list\_arg[\*edge\_itr].is\_convex = is\_convex;

adjacency\_list\_arg[\*edge\_itr].is\_valid = is\_convex;

adjacency\_list\_arg[\*edge\_itr].normal\_difference = normal\_difference; }}

**Sample code – CPC 3D Segmentation**

template <typename PointT> void

class CPCSegmentation : public LCCPSegmentation<PointT>{

public: CPCSegmentation ();

virtual ~CPCSegmentation ();

void segment ();

inline void setCutting () {

max\_cuts\_ = max\_cuts;

min\_segment\_size\_for\_cutting\_ = cutting\_min\_segments;

min\_cut\_score\_ = cutting\_min\_score;

use\_local\_constrains\_ = locally\_constrained;

use\_directed\_weights\_ = directed\_cutting;

use\_clean\_cutting\_ = clean\_cutting }  
inline void setRANSACIterations (const uint32\_t ransac\_iterations) {

ransac\_itrs\_ = ransac\_iterations;

}

**Cluster Extraction**

pcl::PointCloud<pcl::PointXYZL>::

Ptr cloud(new pcl::PointCloud<pcl::PointXYZL>);

std::vector<pcl::PointCloud<pcl::PointXYZL>::Ptr> clouds;

reader.read(file, \*cloud);

for (int i = 0; i < 32; i++){

pcl::PointCloud<pcl::PointXYZL>::

Ptr cloud2(new pcl::PointCloud<pcl::PointXYZL>);

cloudss.push\_back(cloud2);}

for (int i = 0; i < cloud->points.size(); i++){

int label = cloud->points[i].label;

clouds[label%32]->points.push\_back(cloud->points[i]);

pcl::io::savePCDFile(file + "\_\_" + std::to\_string(i) + ".pcd", \*clouds[i],true);}

**Region Growing**

pcl::RegionGrowing<pcl::PointXYZ, pcl::Normal> reg;

reg.setMinClusterSize (50);

reg.setMaxClusterSize (1000000);

reg.setSearchMethod (tree);

reg.setNumberOfNeighbours (30);

reg.setInputCloud (cloud);

//reg.setIndices (indices);

reg.setInputNormals (normals);

reg.setSmoothnessThreshold (3.0 / 180.0 \* M\_PI);

reg.setCurvatureThreshold (1.0);

std::vector <pcl::PointIndices> clusters

reg.extract (clusters);

std::cout << "Number of clusters is equal to " << clusters.size () << std::endl;

std::cout << "First cluster has " << clusters[0].indices.size () << " points." << endl;

std::cout << "These are the indices of the points of the initial" <<

std::endl << "cloud that belong to the first cluster:" << std::endl;

int counter = 0;

while (counter < clusters[0].indices.size ())

{

std::cout << clusters[0].indices[counter] << ", ";

counter++;

if (counter % 10 == 0)

std::cout << std::endl;

}

**Color Based Region Growing:**

// kd-tree object for searches.

pcl::search::KdTree<pcl::PointXYZRGB>::Ptr kdtree(new pcl::search::KdTree<pcl::PointXYZRGB>);

kdtree->setInputCloud(cloud);

// Color-based region growing clustering object.

pcl::RegionGrowingRGB<pcl::PointXYZRGB> clustering;

clustering.setInputCloud(cloud);

clustering.setSearchMethod(kdtree);

// Here, the minimum cluster size affects also the postprocessing step:

// clusters smaller than this will be merged with their neighbors.

clustering.setMinClusterSize(100);

// Set the distance threshold, to know which points will be considered neighbors.

clustering.setDistanceThreshold(10);

// Color threshold for comparing the RGB color of two points.

clustering.setPointColorThreshold(6);

// Region color threshold for the postprocessing step: clusters with colors

clustering.setRegionColorThreshold(5);

std::vector <pcl::PointIndices> clusters;

clustering.extract(clusters);

**Minimum Cut based Segmentation**

// Min-cut clustering object.

pcl::MinCutSegmentation<pcl::PointXYZ> clustering;

clustering.setInputCloud(cloud);

// Create a cloud that lists all the points that we know belong to the object

// (foreground points). We should set here the object's center.

pcl::PointCloud<pcl::PointXYZ>::Ptr foregroundPoints(new pcl::PointCloud<pcl::PointXYZ>());

pcl::PointXYZ point;

point.x = 100.0;

point.y = 100.0;

point.z = 100.0;

foregroundPoints->points.push\_back(point);

clustering.setForegroundPoints(foregroundPoints);

// Set sigma, which affects the smooth cost calculation. It should be

// set depending on the spacing between points in the cloud (resolution).

clustering.setSigma(0.05);

// Set the radius of the object we are looking for.

clustering.setRadius(0.20);

// Set the number of neighbors to look for. Increasing this also increases

// the number of edges the graph will have.

clustering.setNumberOfNeighbours(20);

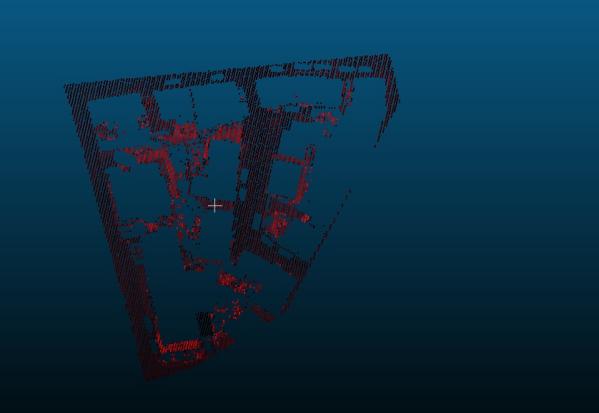
// Set the foreground penalty. It is the weight of the edges

clustering.setSourceWeight(0.6);

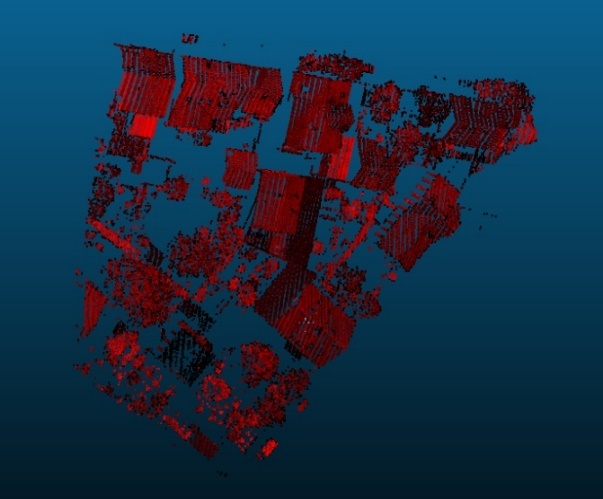
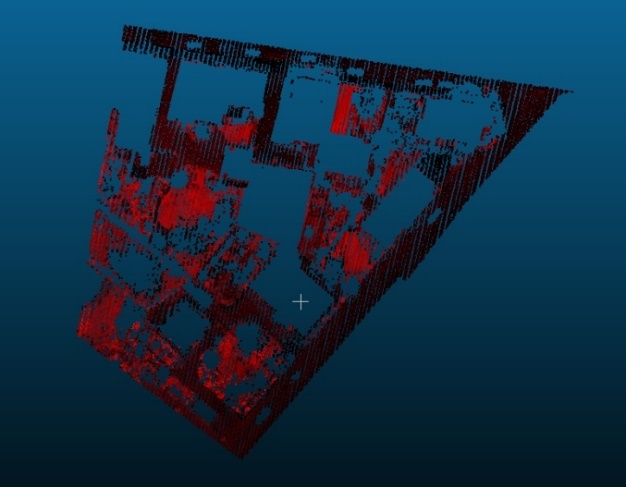
std::vector <pcl::PointIndices> clusters;clustering.extract(clusters);std::cout << "Maximum flow is " << clustering.getMaxFlow() << "." << std::endl;

**B. SCREENSHOTS**

***Vaihingen dataset Pre-Processing:***



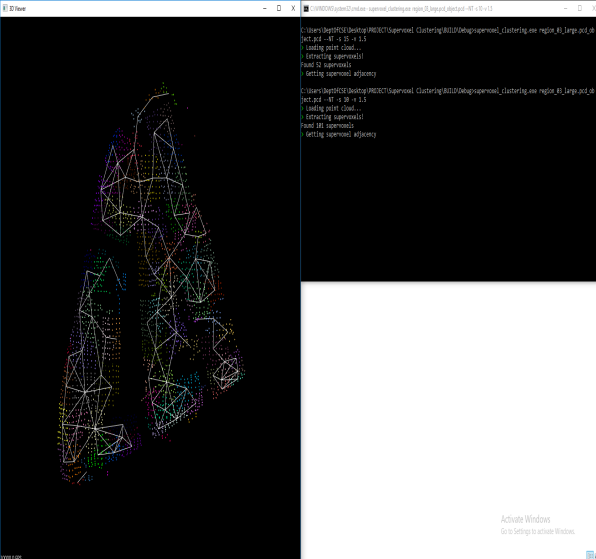
*(a): Noise removed from the point cloud (b): Processed point cloud after noise removal*



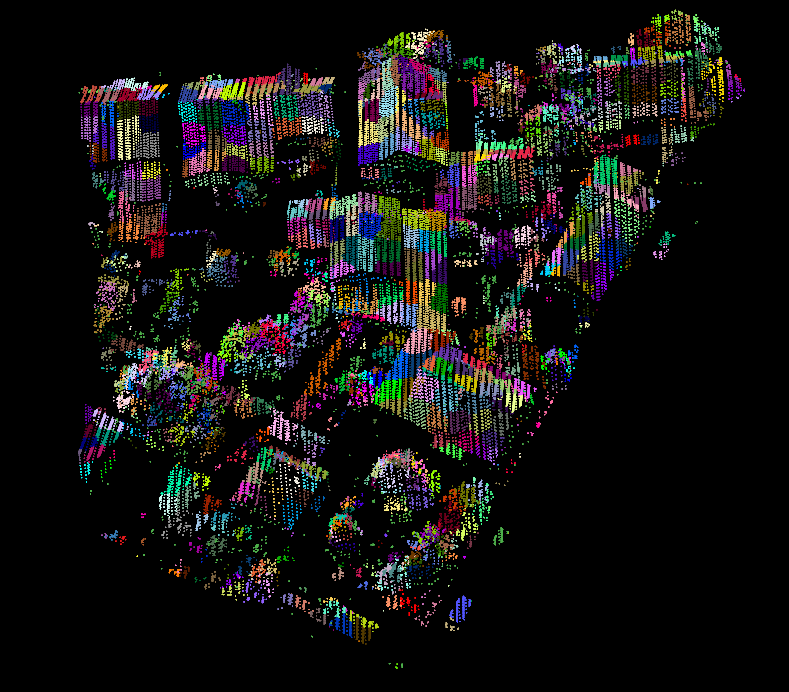
*(c): Noise removed from the point cloud (d): Processed point cloud after noise removal*

*In the above screenshots (a)(b),(c)(d) corresponds to Study Area-1,2 & 3 respectively*

**Supervoxel Creation**



**(a)**



**(b)**

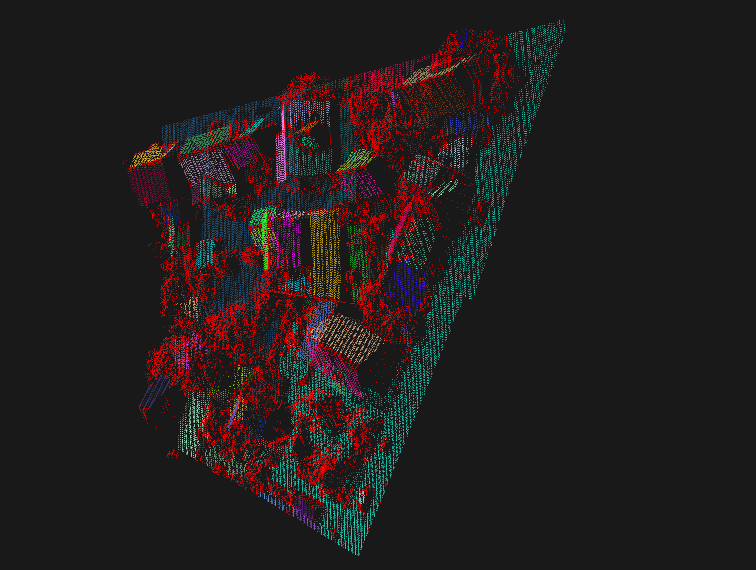
*Vaihingen dataset Supervoxel Creation:*

*(a):Study Area-1 R\_Seed=10 & R\_Voxel=1.5*

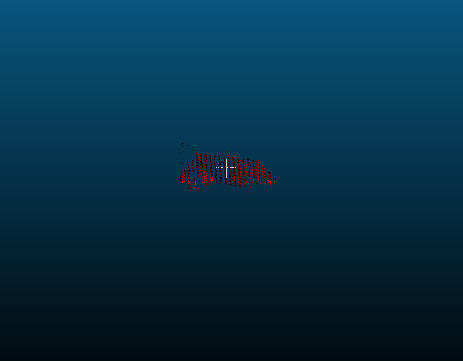
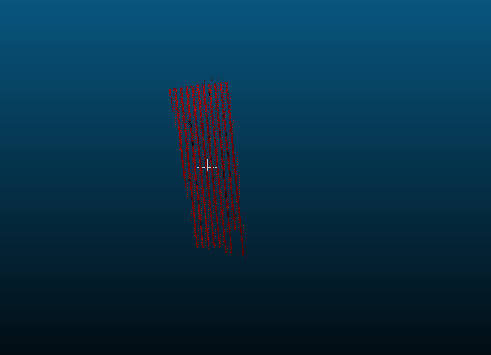
*(b): Study Area-2 R\_Seed=4 & R\_Voxel=0.75*

**Evaluation of Supervoxel Segmentation:**

**Region Growing:**

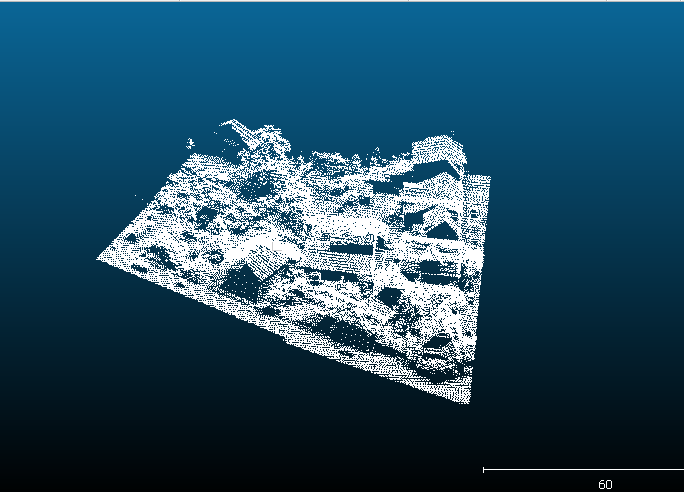
****

**Color Based Region Growing**

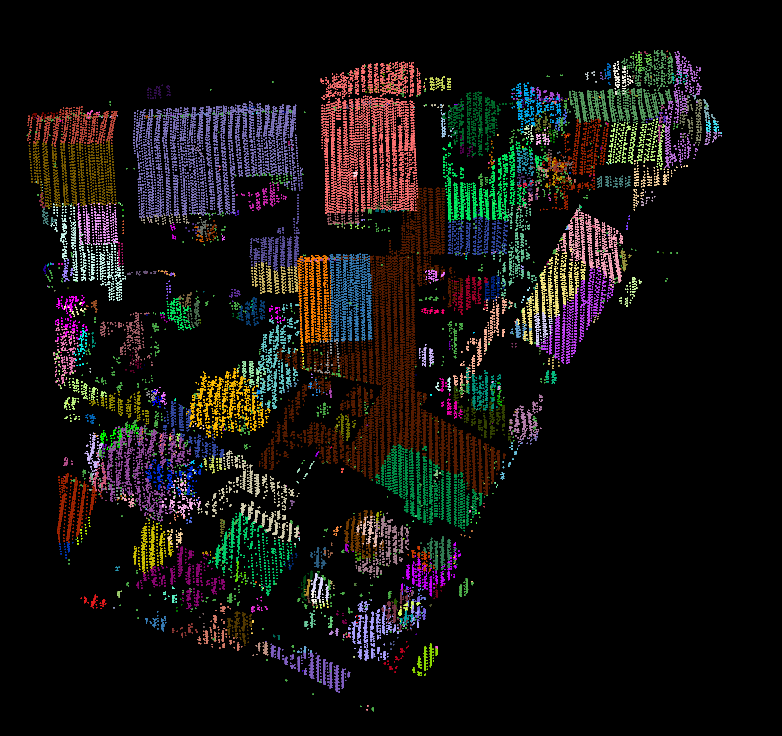
**  **

**(a) (b) (c)**

**Minimum Cut based Segmentation**

****

**LCCP Segmentation**



*Vaihingen dataset LCCPSegmentation:*

*Study Area-R\_Seed=3.75 & R\_Voxel=0.5*

**CPC Segmentation**



*Vaihingen dataset CPCSegmentation:*

*Study Area-R\_Seed=10 & R\_Voxel=1.5*

**REFERENCES**

1. Hanbury.” How do superpixels affect image segmentation?” Progress in Pattern Recognition, Image Analysis and Applications, Springer 2008.
2. A.Farhadi and M. A. Sadeghi,”Phrasal recognition”. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(12):2854–2865, Dec. 2013.
3. Anandakumar M. Ramiya, Rama Rao Nidamanuri & Ramakrishnan Krishnan, “A supervoxel-based spectro-spatial approach for 3D urban point cloud labelling”, International Journal of Remote Sensing, 2016.
4. Anandakumar M. Ramiya, Rama Rao Nidamanuri & Ramakrishnan Krishnan, “Segmentation based building detection approach from LiDAR point cloud”, Egyptian Journal of Remote Sensing and Space Science, 2016.
5. Anandakumar M. Ramiya, Rama Rao Nidamanuri & Ramakrishnan Krishnan, “Object-oriented semantic labelling of spectral–spatial LiDAR point cloud for urban land cover classification and buildings detection”, Geocarto International (Taylor & Francis group), 2015.
6. Anandakumar M. Ramiya, Rama Rao Nidamanuri & Ramakrishnan Krishnan, “Semantic labelling of urban point cloud data”, The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 2014.
7. Anh Nguyen, Istituto Italiano di Tecnologia, Bac Le, Ho Chi Minh City University of Science,”3D point cloud segmentation: A survey”,6th International Conference on Robotics, Automation and Mechatronics (RAM), 2013.
8. Charalambos Poullis, Member, IEEE, “A Framework for Automatic Modeling from Point cloud Data”, IEEE Xplore, 2013.
9. SC Stein, F Wörgötter, M Schoeler, J Papon, T Kulvicius ,” Convexity based object partitioning for robot applications”, IEEE International Conference on Robotics and Automation (ICRA), 2014.
10. Dong Chen, Liqiang Zhang, P.Takis Mathiopoulos, Senior Member IEEE, Xianfeng Huang, “A Methodology for Automated Segmentation and Reconstruction of Urban 3-D Buildings from ALS Point Clouds”, IEEE, 2014.
11. Filin, S. and Pfeifer, N. “Segmentation of airborne laser scanning data using slope adaptive neighbourhood”, ISPRS Journal of Photogrammetry and Remote Sensing. 2006.
12. Golovinskiy, A., V. G. Kim, and T. Funkhouser, “Shape-Based Recognition of 3D Point Clouds in Urban Environments.” IEEE 12th International Conference on Computer Vision, Kyoto, September 29–October 2009.
13. J. Behley, V. Steinhage, and A. Cremers. “Laser-based segment classification”. In Intelligent Robots and Systems (IROS), IEEE 2013.
14. J. Papon, A. Abramov, M. Schoeler, and F. Worgotter, ”Voxel cloud connectivity segmentation-supervoxels for point clouds”. In Computer Vision and Pattern Recognition (CVPR), IEEE 2013.
15. Jin, X., and C. H. Davis. 2005. “Automated Building Extraction from High-Resolution Satellite Imagery in Urban Areas Using Structural, Contextual, and Spectral Information.” EURASIP Journal on Advances in Signal Processing 2196–2206. doi:10.1155/ASP.2005.2196
16. Jun Wang, Jie Shan, Geomatics Engineering, School of Civil Engineering, Purdue University, “SEGMENTATION OF LIDAR POINT CLOUDS FOR BUILDING EXTRACTION”, ASPRS Annual Conference, 2009.
17. Keqi Zhang, Shu-Ching Chen, Member, IEEE, Dean Whitman, Mei-Ling Shyu, Member, IEEE, Jianhua Yan, and Chengcui Zhang, Student Member, IEEE,” A Progressive Morphological Filter for Removing Non ground Measurements From Airborne LIDAR Data”, IEEE transactions on geoscience and remote sensing, VOL. 41, NO. 4, APRIL 2003.
18. Lari Z, Habib A, Kwak E, “An adaptive approach for segmentation of 3D laser point cloud”. International archives of the photogrammetry, remote sensing and spatial information sciences; Calgary, Canada. Vol. XXXVII-5/W12. 2011.
19. Markus Schoeler, Jeremie Papon and Florentin Worgotter, Bernstein Center for Computational Neuroscience (BCCN), “Constrained Planar Cuts - Object Partitioning for Point Clouds”, IEEE Xplore, 2016.
20. Mayer H,” Automatic object extraction from aerial imagery – a survey focusing on buildings”. Comput Vis Image Und. 74: 138–149. 1999.
21. P. Arbelaez, B. Hariharan, C. Gu, S. Gupta, L. Bourdev, and J. Malik. “Semantic segmentation using regions and parts”. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
22. Sampath A, Shan J. “Segmentation and reconstruction of polyhedral building roofs from aerial LIDAR point clouds”. IEEE Trans Geosci Remote Sens. 2010,
23. J Papon, M Schoeler,” Semantic pose using deep networks trained on synthetic RGB-D”, IEEE International Conference on Computer Vision (ICCV), 2015.
24. Stein, S. C., M. Schoeler, J. Papon, and F. Worgotter.. “Object Partitioning Using Local Convexity.” IEEE Conference on Computer Vision and Pattern Recognition, Columbus OH, June 23–28, 2014.
25. Vapnik V et al,”The Nature of Statistical Learning Theory”, New York: Springer-Verlag. 1995
26. Wang M et al,”Lidar dat Segmentation and Classification based on Octree Structure”,Photogrammetric Engineering and Remote Sensing Journal, April 2010.

.