Geometrical Segmentation of Point Cloud Data by Spectral Analysis

Sergey Alexandrov

Publisher: Dean Prof. Dr. Wolfgang Heiden

University of Applied Sciences Bonn-Rhein-Sieg,
Department of Computer Science

Sankt Augustin, Germany

September 2014

Technical Report 02-2014

Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

ISSN 1869-5272

UNIVERSITY OF APPLIED SCIENCES BONN-RHEIN-SIEG

Abstract

Department of Computer Science

Master of Autonomous Systems

Geometrical Segmentation of Point Cloud Data by Spectral Analysis

by Sergey Alexandrov

A principal step towards solving diverse perception problems is segmentation. Many algorithms benefit from initially partitioning input point clouds into objects and their parts. In accordance with cognitive sciences, segmentation goal may be formulated as to split point clouds into locally smooth convex areas, enclosed by sharp concave boundaries. This goal is based on purely geometrical considerations and does not incorporate any constraints, or semantics, of the scene and objects being segmented, which makes it very general and widely applicable.

In this work we perform geometrical segmentation of point cloud data according to the stated goal. The data is mapped onto a graph and the task of graph partitioning is considered. We formulate an objective function and derive a discrete optimization problem based on it. Finding the globally optimal solution is an NP-complete problem; in order to circumvent this, spectral methods are applied. Two algorithms that implement the divisive hierarchical clustering scheme are proposed. They derive graph partition by analyzing the eigenvectors obtained through spectral relaxation. The specifics of our application domain are used to automatically introduce cannot-link constraints in the clustering problem. The algorithms function in completely unsupervised manner and make no assumptions about shapes of objects and structures that they segment. Three publicly available datasets with cluttered real-world scenes and an abundance of box-like, cylindrical, and free-form objects are used to demonstrate convincing performance.

Preliminary results of this thesis have been contributed to the International Conference on Autonomous Intelligent Systems (IAS-13).

Contents

List of Figures

List of Algorithms

Abbreviations

CL	**C**annot-**L**ink
NURBS	**N**on-**U**niform **R**ational **B**-**S**pline
OSD	**O**bject **S**egmentation **D**atabase
PCA	**P**rincipal **C**omponent **A**nalysis
PCL	**P**oint **C**loud **L**ibrary
RGB-D	**R**ed, **G**reen, **B**lue, and **D**epth
RSD	**R**adius-based **S**urface **D**escriptor
SSC	**S**pectral **S**upervoxel **C**lustering
SVC	**S**pectral **V**oxel **C**lustering
TSDF	**T**runcated **S**igned **D**istance **F**unction
SVM	**S**upport **V**ector **M**achine

Chapter 1

Introduction

Perception of objects and their surroundings is one of the key prerequisites for a general-purpose service robot. Effective navigation and manipulation are impossible without understanding how the environment is composed of permanent structures (walls, ceilings, floors), furniture (tables, counters), and objects of daily use. A successful assistive robot should be able to derive such knowledge from its sensory inputs reliably and autonomously. Designing a perception pipeline that starts with raw sensor data and produces a semantic map of the environment is a challenging task that is yet to be solved.

An early stage in the perception pipeline consists of parsing the observed domain into parts; the subsequent stages try to reason and assign semantic meaning to them. For example, object recognition and pose estimation are concerned with putting names on instances of objects and figuring out their positions and orientations in the world. One of the popular approaches makes use of so-called *global descriptors* [AMT+12]. These descriptors are calculated for object candidates; in order to obtain them the scene has to be segmented beforehand. Another higher-level task that relies on segmentation is scene understanding, i.e. determining the structure classes of objects (walls, furniture, props) and support relations between them.

In this thesis we address the problem of geometrical segmentation of point cloud data by means of spectral analysis. Below the three whys are answered to motivate and explain why this type of input data, this particular task, and this solution paradigm were chosen.

1.1 Why Point Clouds?

3D perception is an emerging trend in robotics thanks to the availability of cheap RGB-D cameras. Their ability to directly measure the geometry of a scene allows to sidestep the difficulties associated with structured 3D interpretation inherent to purely image-based approaches. Algorithms that leverage this information were shown to be superior to conventional monocular camera-based approaches in many areas. This includes object recognition [BRF12], especially under heavy occlusions and clutter [TS12], scene understanding [GAM13], and people tracking [MM14], to name a few. A lot of ongoing research is concerned with the development of efficient algorithms to manipulate and analyze point clouds. A large-scale open-source library of such algorithms is freely available [RC11].

1.2 Why Geometrical Segmentation?

Segmentation is a principal step towards solving diverse perception problems, be that object recognition or scene understanding. Many algorithms benefit from initially partitioning the input point cloud into *geometrically meaningful* regions. Of course, this notion is ambiguous and depends on the application at hand, however often it is desirable to obtain results most closely matching with the human perception. In cognitive science, the *minima rule* suggests that all negative minima of the principal curvatures form boundaries between objects and their parts [HS97]. Therefore, the segmentation goal may be formulated as splitting point cloud into locally smooth convex areas, enclosed by sharp concave boundaries.

The stated goal is based only on the geometrical considerations. Indeed, the concepts of smoothness, curvature, convexity, and concavity are purely geometrical. Only point coordinates in 3D space are required to reason about them. Furthermore, the goal does not incorporate any constraints, or semantics, of the scene and objects being segmented. This makes it very general and widely applicable.

1.3 Why Spectral Analysis?

Spectral analysis or, more specifically, *spectral relaxation*, is a technique used to approximate globally optimal solutions to certain types of NP-complete optimization problems. Geometrical segmentation of point cloud data according to the minima rule may be formulated as an instance of such a problem.

A simpler way to treat NP-complete problems is to exploit heuristics and construct algorithms that make local greedy decisions. They usually run fast, but are not guaranteed to find the optimal solution, even approximately. In the context of geometrical segmentation, *region-growing* is a simple and popular algorithm that implements this approach [RvdHV06, HB12, SWS$^+$14]. Starting from initial seeds, it merges neighboring points with similar normals or curvatures, but do not expand regions over sharp edges.

In practice, however, region-growing does not necessarily produce desired results. Consider a point cloud in Figure 1.1a. A human observer would parse the cloud into four objects: two books and two boxes. Due to the limitations of the sensing technology, the boundary between the books is not sharp in the bottom-right corner, as could be observed in the curvature plot in Figure 1.1b. Although the boundary between the books is very sharp for the most part, a typical region-growing algorithm will propagate through the breach, yielding under-segmentation as shown in Figure 1.1c.

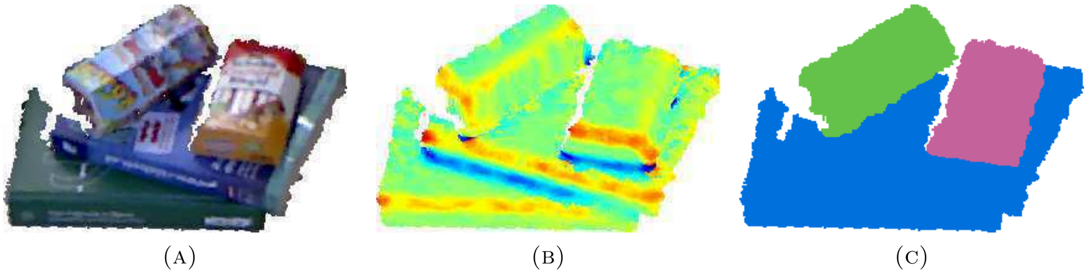

(A) (B) (C)

FIGURE 1.1: Failure case for a region-growing algorithm. (a) Colored input point cloud. (b) Curvatures estimated using PCA with 1.5 cm radius, colors vary from blue for negative curvature, through green for flat regions, to red for positive curvature. (c) Under-segmented output of a region-growing algorithm due to the boundary weakness.

In order to match up with the capabilities of the human visual perception and avoid making poor myopic decisions, a robust point cloud segmentation algorithm should have a certain degree of global awareness. This significantly complicates the problem, but the spectral methods allow to keep it tractable.

1.4 Thesis Outline and Contributions

The contributions of this thesis are:

1. Formulation of point cloud segmentation as a graph partitioning problem;
2. A method to construct a graph from the input point cloud;
3. A recursive algorithm for partition recovery from spectral data;
4. Novel domain-specific utility functions that allow unsupervised segmentation without a prescribed number of segments;
5. Labeling of several publicly available point cloud datasets.

The rest of the thesis is organized as follows. Chapter 2 introduces the main theoretical concepts relevant to the thesis. Chapter 3 summarizes the related work in the domains of RGB-D data segmentation and constrained spectral clustering. In Chapter 4 the geometrical segmentation algorithms are presented. Chapter 5 is concerned with evaluation metrics, datasets, and results. Chapter 6 concludes the thesis and outlines the possible directions of future research.

1.5 Publication

Preliminary results of this thesis have been contributed to the 1^{st} International Workshop on 3D Robot Perception with Point Cloud Library, held in conjunction with 13^{th} International Conference on Autonomous Intelligent Systems (IAS-13):

- Sergey V. Alexandrov and Rainer Herpers. Geometrical Segmentation of Point Cloud Data using Spectral Clustering. In *13th International Conference on Autonomous Intelligent Systems (IAS) – 1st International Workshop on 3D Robot Perception with Point Cloud Library*, Padova and Venice, Italy, July 2014

Chapter 2

Background

This chapter introduces the main theoretical concepts relevant to the thesis. In the first section we address the topics related to the point cloud data. Two different acquisition methods and the implications on the properties of the data are discussed. Several preprocessing methods are described as well. In the second section we introduce the terminology and important properties of graphs and other associated mathematical objects. In the third section an extensive review of the spectral graph clustering theory is given. It involves formulation of main partitioning objectives and associated optimization problems, as well as explanation of the ideas behind spectral relaxation and partition recovery.

2.1 Point Cloud Acquisition and Preprocessing

A *point cloud* is a collection of points that provide a discrete non-parametric representation of a 3D surface. The points have three Cartesian coordinates that define their positions with respect to some fixed coordinate frame.

2.1.1 Acquisition

There exist several ways to obtain point cloud data. Firstly, one may use various range imaging sensors that are capable of measuring distances. Secondly, if a parametric or a dense model of a surface is available, a point cloud could be sampled from this model.

2.1.1.1 Range Imaging Sensors

Range imaging sensors operate according to a number of different physical principles, structured light and time-of-flight being most widely used in the robotics context [SJP11]. These sensors deliver 2D images that encode distances from scene points to a fixed point (usually the optical center of the acquisition device). Using known calibration parameters, it is possible to project these points into 3D space, thereby obtaining a point cloud.

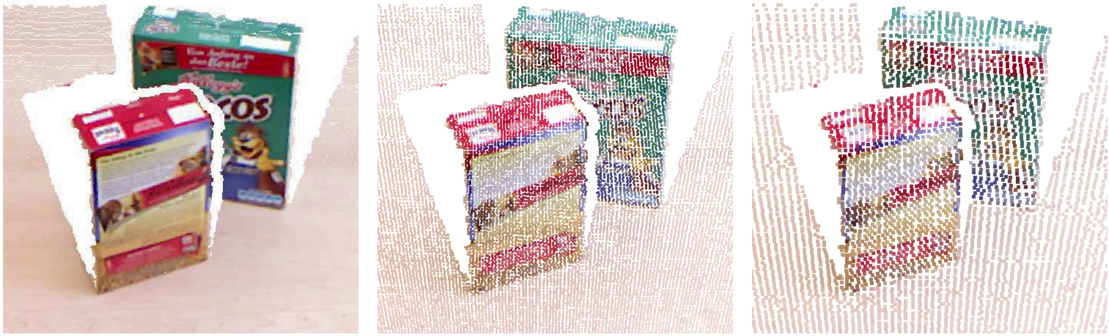
An important property of point cloud data acquired this way is that they represent a surface as observed from a single viewpoint. This implies that the objects and structures in the foreground occlude the background, making the surfaces there partially unobservable. Despite the fact that the points have 3D coordinates, they do not capture the complete 360° view of a scene and are oftentimes called $2\frac{1}{2}$D.

2.1.1.2 Model Sampling

Recent advances in point cloud registration and surface reconstruction make it possible to perform dense 3D mapping of the environment in real time [NIH$^+$11, WMK$^+$12]. Tools like KinectFusion track the motion of an RGB-D camera with respect to the model built so far, simultaneously extending it with the newly acquired data. The model is typically represented with the volumetric truncated signed distance function (TSDF). Using a marching cubes type algorithm it is relatively easy to render the model into a mesh or a point cloud.

The resulting point cloud is truly three-dimensional because it incorporates observations from multiple viewpoints. Depending on the number and arrangement of the incorporated views, it is still possible that some surfaces are occluded. However, it is reasonable to assume that in the region of interest the objects and structures were observed from multiple viewpoints and thus their models are complete.

In contrast with $2\frac{1}{2}$D point clouds, there is no one-to-one mapping of the cloud and the depth image. This makes the conventional 2D image-based processing methods inapplicable.



(A) Original point cloud (B) Voxel size 6 mm (C) Voxel size 12 mm

Figure 2.1: Point cloud voxelization. The number of points in the original cloud is 96405. The number of points after downsampling is 18363 and 5162 respectively.

2.1.2 Preprocessing

Many different kinds of preprocessing can be applied to point cloud data. In the context of this thesis of the main interest are downsampling and computation of basic geometric features of a surface.

2.1.2.1 Downsampling

The number of points in a cloud may be huge. A scene captured by a Kinect camera at standard resolution contains over 300000 points. This amount of data may be hard to process, especially if (nearly) real-time performance is required. Therefore, it is often desirable to downsample point cloud data. Its amount should be reduced, yet the quality should be retained as much as possible.

Voxelization is a simple method that may be used to downsample a point cloud. A grid is superimposed on 3D Euclidean space; each of its' cells is a unit cubic volume called *voxel*. The points of the original cloud that belong to the same voxel are approximated by their centroid. By varying the resolution of the grid, different degrees of downsampling may be achieved, as exemplified in Figure 2.1.

Besides from downsampling the data, voxelization serves two additional purposes. Firstly, a certain degree of smoothing is achieved by suppressing high-frequency noise in point coordinates. Secondly, the grid induces adjacency relations between voxels. Indeed, two voxels may share a face, an edge, or a corner; they are assumed to be adjacent is such case. This makes neighborhood computation a cheap and straightforward procedure.

2.1.2.2 Normal and Curvature Estimation

A point, when viewed in isolation, is described by its Euclidean coordinates. Two additional attributes—*normal orientation* and *curvature*—may be attached when it is considered as a part of the surface, from which it was sampled. The former describes the plane, tangential to the surface at the point; the latter captures the amount of variation in the surface around the point. These attributes are of an importance for various point cloud visualization and processing tasks; shading and rendering, denoising, feature detection, and segmentation are just a few of them.

There exist a number of approaches to normal estimation [KAWB09]. The classical method consists of approximating the tangential plane by means of the first order 3D plane fitting and is based on the theory of Principal Component Analysis (PCA). Given a point $\mathbf{p}_0$ and the set of its neighbors $\mathcal{P} = \{\mathbf{p}_1, \dots, \mathbf{p}_k\}$, the covariance matrix $C \in \mathbb{R}^{3\times 3}$ is defined as

$$C = \frac{1}{k+1} \sum_{i=0}^{k} (\mathbf{p}_i - \bar{\mathbf{p}}) \cdot (\mathbf{p}_i - \bar{\mathbf{p}})^\top , \tag{2.1}$$

where $\bar{\mathbf{p}}$ is the centroid of $\mathcal{P} \cup \{\mathbf{p}_0\}$. The eigenvectors of this matrix form an orthogonal coordinate frame and correspond to the principal components of $\mathcal{P}$. The eigenvector associated with the minimal eigenvalue thus defines the normal of the plane fitted to the neighborhood.

A few different ways to compute the curvature are available. Rusu [Rus09] proposed a simple method which makes use of the eigendecomposition involved in normal computation. He noted that the eigenvalues λ_j of the covariance matrix C approximate surface variations around the point $\mathbf{p}_0$. Therefore, the ratio between the minimum eigenvalue λ_0 and the sum of all eigenvalues approximates the curvature. Formally, the curvature σ is given by

$$\sigma = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2}. \tag{2.2}$$

2.2 Graphs and Associated Mathematical Objects

In this section we introduce the terminology and basic definitions from the graph theory. We also define a family of graph Laplacians that turn to be essential for the spectral methods discussed later.

2.2.1 Terminology and Definitions

A *graph* G is a mathematical abstraction which represents a collection of objects and links between them. It consists of two sets $(\mathcal{V}, \mathcal{E})$ called its *vertex set* and its *edge set* [Tru13].

The vertex set $\mathcal{V}$ is a finite non-empty set. Its elements represent inter-connected objects and are called *vertices*. The number of vertices in the set $n = |\mathcal{V}|$ is called the *order* of the graph.

The edge set $\mathcal{E}$ is a finite (possibly empty) set. Its elements represent links connecting pairs of objects and are called *edges*. An edge is defined by unordered pair $\{v, u\}$, where v and u are vertices of G. The number of edges in the set $m = |\mathcal{E}|$ is called the *size* of the graph.

Two vertices v and u are *adjacent* if there exists an edge $\{v, u\} \in \mathcal{E}$ that *joins* them. We will say that this edge is *incident* to both v and u and call these vertices *neighbors*. The set of all neighbors of a vertex v is called its *neighborhood* and is denoted by $\mathcal{N}(v)$.

A *weighted graph* is a graph for which a function $\omega : \mathcal{E} \to \mathbb{R}$ is defined that assigns a weight to each edge. In this thesis we will always deal with weighted graphs. We will denote the weight of an edge $\{v_i, v_j\} \in \mathcal{E}$ as w_{ij}. In the context of graph clustering research it is standard to assume that the weights are positive. In the exposition of this chapter we also make this assumption.

The total weight of the edges incident to a vertex v is called the *degree* of v

$$deg\left(v\right) = \sum_{u \in \mathcal{N}(v)} \omega\left(v, u\right). \tag{2.3}$$

Arranging vertex degrees into a diagonal matrix gives the *degree matrix* D of a graph. The degree of a cluster $\mathcal{C}$ is the total weight of the edges both of whose end-points are inside $\mathcal{C}$

$$deg\left(\mathcal{C}\right) = \left|\left\{u \in \mathcal{N}\left(v\right) | v, u \in \mathcal{C}\right\}\right|. \tag{2.4}$$

The *volume* of a graph is given by

$$vol\,(G) = \sum_{v \in \mathcal{V}} deg\,(v) = trace\,(D)\,, \tag{2.5}$$

where $trace\,(\cdot)$ is the sum of diagonal elements of a matrix. Similarly, the volume of a cluster $\mathcal{C}$ is the sum of the degrees of its vertices

$$vol\,(\mathcal{C}) = \sum_{v \in \mathcal{C}} deg\,(v)\,. \tag{2.6}$$

The edge structure of a graph is described by the *adjacency matrix* $A \in \mathbb{R}^{n \times n}$ where

$$A_{ij} = \begin{cases} w_{ij} & \text{if } \{v_i, v_j\} \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}. \tag{2.7}$$

A *partition* $\pi^k = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$ of the vertices of a graph is defined by k disjoint, non-empty subsets of vertices, each one denoted by $\mathcal{C}_i$, such that $\bigcup_{1..k} \mathcal{C}_i = \mathcal{V}$. The set of all possible k-way partitions of the graph is denoted as Π^k. In the special case where $k = 2$, a partition $\pi^2 = \{\mathcal{C}, \mathcal{V} \backslash \mathcal{C}\}$ is called a *bipartition*, or a *cut*. $\mathcal{V} \backslash \mathcal{C}$ is called the *complement* of $\mathcal{C}$ and is denoted as $\bar{C}$ for simplicity.

Cut size is the sum of the weights of the edges crossing the cut

$$Cut\,\left(\mathcal{C}, \bar{\mathcal{C}}\right) = \sum_{v \in \mathcal{C}, u \in \bar{\mathcal{C}}} \omega\,(v, u)\,. \tag{2.8}$$

This is generalized for an arbitrary k-way partition as

$$Cut\,\left(\pi^k\right) = \frac{1}{2} \sum_{i=1}^{k} Cut\,\left(\mathcal{C}_i, \bar{\mathcal{C}_i}\right). \tag{2.9}$$

A partition π^k of a graph may be encoded by the k-cluster *indicator matrix* $X \in \mathbb{R}^{n \times k}$. The columns $x_1 \ldots x_k$ of this matrix are called the *indicator vectors* and are defined as

$$x_j\left(v_i\right) = \begin{cases} 1 & \text{if } v_i \in \mathcal{C}_j \\ 0 & \text{otherwise} \end{cases} \qquad (i = 1, \ldots, n;\ j = 1, \ldots k)\,. \tag{2.10}$$

These vectors are orthonormal to each other (i.e. $X^\top X = I$).

2.2.2 Graph Laplacians

We will follow the terminology of von Luxburg [vL07] and distinguish between three types of graph Laplacians: unnormalized, symmetric, and random walk.

We will state several properties of different graph Laplacians that are important for the following exposition without proofs. An interested reader may consult von Luxburg [vL07].

2.2.2.1 Unnormalized graph Laplacian

Using the adjacency and degree matrices defined in the previous section, the unnormalized graph Laplacian matrix is given by

$$L = D - A. \tag{2.11}$$

Proposition 2.1. *Properties of L:*

1. *L is symmetric and positive semi-definite;*
2. *The smallest eigenvalue of L is 0, the corresponding eigenvector is the constant one vector $\mathbf{1}$;*
3. *L has n non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \cdots \leq \lambda_n$;*
4. *For every vector $x \in \mathbb{R}^n$*

$$x^\top L x = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} \left(x_i - x_j\right)^2 . \tag{2.12}$$

2.2.2.2 Normalized graph Laplacians

Two matrices are known in the literature under the name of ”normalized graph Laplacian”. Von Luxburg [vL07] proposes to denote the first as L_{sym}

$$L_{sym} = D^{-1/2} L D^{-1/2}, \tag{2.13}$$

because it is symmetric, and the second as L_{rw}

$$L_{rw} = D^{-1} L, \tag{2.14}$$

because it is related to the random walks on a graph.

2.3 Spectral Graph Clustering

Data often have a certain underlying structure. Given a similarity measure over the data, cluster analysis is concerned with automatic identification of this structure. The data are partitioned into clusters, so that the elements in the same cluster are similar, and the elements that belong to different clusters are dissimilar. In other words, *clustering* is an unsupervised approach to grouping related data elements.

Graph clustering is concerned with partitioning the data that has been mapped onto a graph. The mapping models data elements as vertices, and similarity relationships between them as edges (or lack thereof). Based on the edge structure, graph clustering seeks to partition the vertices into a number of subgraphs (or clusters). The partitioning should possess a property that the majority of edges are intra-cluster, and relatively few inter-cluster edges exist.

The research in graph clustering is very active and a number of distinct methodologies exist [Sch07]. They could be roughly divided into *partitioning* and *hierarchical* categories. The former try to split the vertices into a given number of clusters, usually by shuffling individual vertices between clusters and evaluating a given objective function until stopping criteria are met. The latter form clusters gradually, starting from either a cluster assignment where all vertices belong to one cluster, or each vertex belongs to its own cluster. *Divisive* algorithms proceed by dividing large clusters and *agglomerative* by uniting small clusters.

Spectral graph clustering is a particular class of clustering algorithms that are characterized by the use of eigendecomposition of the graph Laplacian matrices. Spectral clustering has many fundamental advantages and often outperforms ”traditional” clustering algorithms [vL07]. A review of spectral methods for graph clustering was contributed recently by Nascimento and de Carvalho [NdC11]. Furthermore, a comprehensive self-contained introduction to spectral clustering was presented by von Luxburg [vL07].

In the remainder of this section we formalize the objectives of graph clustering, demonstrate how attempts to find desirable partitions yield discrete optimization problems, and present spectral relaxation, an instrument that allows to effectively solve such problems.

2.3.1 Graph Partitioning Objectives

We began this section by informally defining the goal of clustering (and graph clustering in particular) as trying to group similar data elements together and dissimilar data elements apart. This could be formalized with an *objective function*—a function that measures the degree in which a given partition satisfies the goal. Objective functions, sometimes also called *fitness measures*, may serve either for identification of clusters, or for choosing between alternative clusterings. Below we formally state and discuss a number of common graph partitioning objectives. The generalized (k-way) variants are presented, though historically some of them were first formulated for the bipartitioning problems.

2.3.1.1 Minimum Cut Objective

Minimum cut objective seeks to find a partition where the sum of the weights of the inter-cluster edges is minimum. This sum is called cut size, as was discussed in Section 2.2.1. The problem of minimizing the cut size admits efficient algorithms to find optimal solutions, especially in the bipartitioning ($k = 2$) case [SW97]. However, it was observed [WL93] that the optimal partitions often involve singular clusters (i.e. clusters consisting of a single vertex), which is an undesirable property.

2.3.1.2 Ratio Cut Objective

In contrast to the minimum cut objective, *ratio cut* expresses the cut size relative to the sizes of the clusters; this promotes more "balanced" partitions. Formally, ratio cut objective function is defined as

$$RatioCut\left(\pi^{k}\right)=\sum_{i=1}^{k}\frac{Cut\left(\mathcal{C}_i,\bar{\mathcal{C}}_i\right)}{|\mathcal{C}_i|} \tag{2.15}$$

and is a subject to minimization.

2.3.1.3 Ratio Association Objective

Ratio association objective function is given by

$$RatioAssociation\left(\pi^{k}\right)=\sum_{i=1}^{k}\frac{deg\left(\mathcal{C}_i\right)}{|\mathcal{C}_i|}. \tag{2.16}$$

This objective counts the sum of the weights of the intra-cluster edges relative to the sizes of the clusters, and is therefore a subject to maximization.

2.3.1.4 Normalized Cut Objective

Normalized cut objective is similar to ratio cut, however it uses cluster volumes instead of sizes to normalize the cut size. Formally the objective function is defined as

$$NCut\left(\pi^{k}\right)=\sum_{i=1}^{k}\frac{Cut\left(\mathcal{C}_i,\bar{\mathcal{C}}_i\right)}{vol\left(\mathcal{C}_i\right)} \tag{2.17}$$

and is a subject to minimization.

2.3.1.5 Normalized Association Objective

The *normalized association* objective is given by

$$NAssociation\left(\pi^{k}\right)=\sum_{i=1}^{k}\frac{deg\left(\mathcal{C}_i\right)}{vol\left(\mathcal{C}_i\right)} \tag{2.18}$$

and is a subject to maximization. It could be shown that maximizing it is equivalent to minimizing the normalized cut objective.

2.3.1.6 Min-Max Cut Objective

This objective aims to minimize inter-cluster similarities while maximizing intra-cluster similarities. Formally the objective function is given by

$$MinMaxCut\left(\pi^k\right) = \sum_{i=1}^{k} \frac{Cut\left(\mathcal{C}_i, \bar{\mathcal{C}}_i\right)}{deg\left(\mathcal{C}_i\right)}. \tag{2.19}$$

Ding et al. [DHZ$^+$03] used the random graph model to demonstrate that the min-max cut objective has much stronger preference to balanced partitions than ratio or normalized cuts.

2.3.2 Discrete Optimization Problems

Given a problem that admits a large finite number of solutions and an objective function that assigns a score to each solution, the task of choosing the one with the best score is called a *discrete optimization problem*. Depending on the type of the objective function, the goal may be either to minimize or maximize the function value. In this subsection we use some of the previously defined objective functions to formulate graph clustering as discrete optimization problems.

2.3.2.1 Ratio Association Maximization

The ratio association objective function can be rewritten using the k-cluster indicator matrix X from Equation 2.10 and the graph adjacency matrix A. It is easy to see that $RatioAssociation\left(\mathcal{C}_i, \bar{\mathcal{C}}_i\right) = \mathbf{x}_i^\top A \mathbf{x}_i$, which is the element at the intersection of i^{th} row and column of $X^\top AX$. Consequently,

$$RatioAssociation\left(\pi^k\right) = \sum_{i=1}^{k} \mathbf{x}_i^\top A \mathbf{x}_i = trace\left(X^\top AX\right). \tag{2.20}$$

Ratio association measures the fitness; it is desirable to find a partition that maximizes it over the set of all possible k-way partitions of the graph. This corresponds to the following discrete optimization problem:

$$\begin{array}{ll} \max\limits_{X \in \mathbb{R}^{n \times k}} & \mathit{trace}\left(X^\top A X\right) \\ \text{subject to} & X^\top X = I \\ & X \text{as defined in Equation 2.10} \end{array} . \tag{2.21}$$

2.3.2.2 Ratio Cut Minimization

Ratio cut measures the cost of a partition and therefore is a subject to minimization. Similarly to ratio association, this may be formulated as a matrix trace minimization problem. To enable this, the indicator matrix elements have to be redefined as

$$x_{ij} = \begin{cases} 1/\sqrt{|\mathcal{C}_j|} & \text{if } v_i \in \mathcal{C}_j \\ 0 & \text{otherwise} \end{cases} . \tag{2.22}$$

Like the original indicator matrix, this modified version also defines a k-way partitioning of a graph and possesses the same two important properties. Firstly, its columns are orthogonal to each other. Secondly, its elements assume one of at most $k+1$ different possible values in the set $\left\{0, \frac{1}{\sqrt{|\mathcal{C}_1|}}, \dots, \frac{1}{\sqrt{|\mathcal{C}_k|}}\right\}$. Using Proposition 2.1 it could be shown that $\mathit{RatioCut}\left(\mathcal{C}_i, \bar{\mathcal{C}}_i\right) = \mathbf{x}_i^\top L \mathbf{x}_i$, which is the element at the intersection of i^{th} row and column of $X^\top L X$ [vL07]. It follows that

$$\mathit{RatioCut}\left(\pi^k\right) = \sum_{i=1}^{k} \mathbf{x}_i^\top L \mathbf{x}_i = \sum_{i=1}^{k} \left(X^\top L X\right)_{ii} = \mathit{trace}\left(X^\top L X\right). \tag{2.23}$$

Therefore, the problem of minimizing the ratio cut objective over the set of all possible k-way partitions of the graph corresponds to the following discrete optimization problem:

$$\begin{array}{ll} \min\limits_{X \in \mathbb{R}^{n \times k}} & \mathit{trace}\left(X^\top L X\right) \\ \text{subject to} & X^\top X = I \\ & X \text{as defined in Equation 2.22} \end{array} . \tag{2.24}$$

2.3.2.3 Normalized Cut Minimization

The normalized cut objective function can also be rewritten using a variant of the indicator matrix where

$$x_{ij} = \begin{cases} 1/\sqrt{vol\left(\mathcal{C}_j\right)} & \text{if } v_i \in \mathcal{C}_j \\ 0 & \text{otherwise} \end{cases} . \tag{2.25}$$

Its minimization transforms to the following discrete optimization problem:

$$\begin{aligned} \min_{X \in \mathbb{R}^{n \times k}} \quad & trace\left(X^\top L X\right) \\ \text{subject to} \quad & X^\top D X = I \\ & X \text{as defined in Equation 2.25} \end{aligned} . \tag{2.26}$$

As was noted before, this problem is equivalent to the maximization of the normalized association objective.

2.3.3 Spectral Relaxation

The discrete optimization problems presented in the previous subsection are NP-complete [HK92, SM00]. In order to circumvent this, the problems may be relaxed by discarding the discreteness conditions and allowing the elements of the indicator matrices X to take arbitrary real values.

Consider the optimization problem derived from the ratio cut objective (Equation 2.24). Its relaxed version

$$\begin{aligned} \min_{X \in \mathbb{R}^{n \times k}} \quad & trace\left(X^\top L X\right) \\ \text{subject to} \quad & X^\top X = I \end{aligned} \tag{2.27}$$

aims to minimize the matrix trace, subject only to orthogonality of column vectors. In the lack of the discreteness constraint the Rayleigh–Ritz theorem can be applied [L9̈7]. According to it, the real-valued solution is given by a matrix assembled of the eigenvectors of L. Each of the eigenvectors $\mathbf{v}_2, \ldots, \mathbf{v}_{k+1}$ that correspond to the smallest nonzero eigenvalues $\lambda_2, \ldots, \lambda_{k+1}$ of the eigenproblem

$$L\mathbf{v} = \lambda\mathbf{v} \tag{2.28}$$

is a real-valued approximation of the corresponding discrete cluster indicator vector.

A discrete cluster indicator vector explicitly defines a 2-way graph partition by assigning each vertex either of two fixed discrete values. Its real-valued approximation does not have this property; additional processing is required to recover the partition that it encodes. This section is concluded with an overview of two approaches to do so.

2.3.3.1 Iterative 2-way Partitioning

This approach may be traced back to the work of Hagen and Kahng [HK92]. They proposed a simple recursive procedure that consists of the following steps:

1. Given a weighted graph, construct the unnormalized Laplacian;
2. Compute the eigenvector $\mathbf{v}$ corresponding to the smallest nonzero eigenvalue;
3. Analyze the elements of $\mathbf{v}$ to find a suitable graph split;
4. If certain stopping criteria are not met yet, partition the graph into two subgraphs;
5. Repeat the procedure for each subgraph.

Of a particular interest is the method used in step 3 to derive the split based on the eigenvector $\mathbf{v}$. Four different heuristics were proposed:

1. Split based on the sign of the elements of $\mathbf{v}$;
2. Partition around the median of $\mathbf{v}$;
3. Sort $\mathbf{v}$ to obtain a linear ordering, then determine the largest eigengap between two consecutive elements and partition around it;
4. Sort $\mathbf{v}$ to obtain a linear ordering, then determine the splitting index that yields the best ratio cut objective value.

According to the authors, the fourth method subsumes the other three in that it finds a solution that is better than, or equal to, the other heuristics.

2.3.3.2 Direct k-way Partitioning

An alternative approach seeks to obtain a k-way partition by simultaneously considering all computed eigenvectors. It embeds the graph vertices in the k-dimensional space spanned by the eigenvectors and performs k-means clustering of vertices in that space. A few variants exist which differ in the kind of graph Laplacian matrix that they use [SM00, NJW02]. The deficit of this approach is that the prescribed number of clusters has to be known beforehand.

Chapter 3

State of the Art

3.1 Segmentation of RGB-D Data

Segmentation of surfaces and objects in 3D has been an active research field for several decades. The early work targeted at the sensory output of laser range scanners or structured light based instruments, often exploiting image-like—"$2\frac{1}{2}$D"—character of such data [HJBJ$^+$96]. Segmentation of unstructured point clouds was also considered. For example, Rabbani et al. [RvdHV06] presented a method to segment point clouds into smooth connected areas using region-growing approach. Recently the advent of cheap RGB-D sensors pushed the researches to reuse and combine ideas from color and range image segmentation.

Mishra et al. [MSA12] proposed a strategy to segment "simple" objects from the RGB-D input data. They define a "simple" object to be a compact region enclosed by a boundary (depth or contact). Central to their method is a concept of probabilistic boundary edge map. This map is obtained by applying an edge detector to a color image supplied by an RGB-D camera. The detected edges are classified as either being a part of object boundary or not by leveraging the depth data. Next they define a procedure that given a boundary map and a point in it finds a closed contour that may correspond to an object that contains that point. They select points by applying border ownership concept and a fixation strategy. In the end they have multiple (possibly duplicating contours) which are filtered to come up with a final set of segmented objects. In the work of Potapova et al. [PZV12] similar techniques are used to perform attention-driven segmentation of more cluttered scenes. Instead of selecting attention points based on the border ownership concept, they compute a saliency map of the scene and pick points using a winner-take-all neural network. Both methods are inherently single-view. Furthermore, the latter uses the height of a point above the supporting plane in the saliency map computation, therefore implicitly assuming table-top scenes.

Silberman et al. [SHKF12] presented a work on structured 3D scene interpretation. Given an input from an RGB-D camera they strive to obtain physical scene prarse, i.e.. to interpret major surfaces, objects, and establish support relations between them. In order to classify objects and surfaces and infer their relations, they start by breaking the scene into regions. They start with an over-segmentation produced by applying the watershed algorithm to Pb boundary map derived from a color image (see Arbeleaz). This produces around 1000-2000 regions that have to be merged to come up with objects. Next they iteratively merge pairs of regions with minimum boundary strength. The strengths are predicted by a trained boosted decision tree classier which relies on both 2D (color) and 3D (geometric) features. The latter include differences in surface orientations and depth differences. Given the final segmentation they proceed by reasoning about support relations between them.

Similar is the work of Gupta et al. [GAM13]. They also target at cluttered indoor scenes which they try to segment semantically. Their algorithm generalizes the gPb-ucm approach of Arbelaez by incorporating the depth information in the boundary detection process. This is followed by interative region merging based on the average strengths of the boundaries as predicted by an SVM classifiers trained of a combination of depth and color features.

The work of Jia et al. [JGSC13] is concerned with 3D scene interpretation. Their application scenario is table-top scenes, however they apply their method (with a limited success) to more general indoor scenes. They split the scene into individual objects, fit bounding boxes and then apply volumetric reasoning to infer support relations and evaluate stability. By incorporating physical constraints they are able to improve scene segmentation by encouraging segmentation hypotheses that are physically plausible. They have an energy function that gauges the segmentation in terms of volumetric consistency, support relations and overall physical stability of the system. They interweave the two processes: take object segments, fit bounding box and evaluate support

Katz et al. [KKBS13] address the problem of robotic manipulation of a pile of unknown objects using autonomous interactive perception. In their work object segmentation and detection is interleaved with manipulation to allow progressive removal of objects from the table. They use an RGB-D camera to segment the scene into so-called object facets—approximately smooth circumscribed surfaces. The input depth images are processed to extract a map of depth discontinuities and a estimate normals. The normals are encoded as a color image, and depth edges are subtracted from it. The standard color-based mean-shift segmentation algorithm is applied to this image to extract facets.

Goron et al. [GMLB12] describe a system for segmentation of box-like and cylindrical objects in cluttered table-top scenes. They remove the supporting plane and split the cloud into Euclidean clusters. Within clusters they compute normals, curvatures, and Radius-Based Surface Descriptor (RSD), which are used to determine the type of surface a point belongs. Then they fit 2D lines and circles models to the projections of the clusters onto the supporting plane. From the fitted models a complete object is recovered using region growing approach.

A framework for segmenting unknown objects was presented by Richtsfeld et al. [RZV12]. Their system over-segments the input point cloud into surface patches and fit either plane or NURBS (Non-Uniform Rational B-Spline). Using annotated training data, they train an SVM classifier to predict if two patches belong to the same object based on geometrical and visual features. Finally they construct a graph of the surface patches and assign edge weights based on the output of the SVM. Finally, a greedy graph-cut algorithm is used to find a partitioning of this graph. The follow-up work of Morwald et al. [MRP$^+$13] concentrates on improving parametric representation of surface patches. They incorporate the model of sensor noise and refine patch boundaries using B-Spline curves, which leads to improved segmentations.

Ückermann et al. [UHR12] presented an algorithm for 3D segmentation of cluttered scenes that can work in real-time. Their approach is model-free and is capable of segmenting objects of unknown shapes. In the first step they compute edge image using the input depth image and segment an image into connected surface patches using a region growing method. In the second step they merge patches into object regions by constructing a directed adjacency graph and applying probabilistic composition with a set of rules. and stability, merge or split segments to improve the score.

Karpathy et al. [KMFF13] focus on identifying portions of a 3D scene that could correspond to objects. Conversely to many other works, they consider dense 3D environment maps, represented by meshes and obtained by preprocessing a set of RGB-D frames using the KinectFusion algorithm. Multiple segmentations of a scene are obtained by running the greedy graph-based algorithm of Felzenszwalb and Huttenlocher [FH04] with different sets of parameters. Then they reason about certain geometric properties of found segments without attaching any semantic labels.

The most similar is a work contemporary to ours of Stein et al. [SWS+14]. They present a bottom-up approach for point cloud segmentation into object parts. The cloud is oversegmented into surface patches and a graph is built. They classify edges into convex and cocave, and apply region growing to identify locally convex connected subgraphs which represent object parts.

3.2 Constrained Spectral Clustering

The performance of a clustering algorithm in general—and spectral clustering in particular—may be improved by incorporating prior knowledge about the data in their design. The most common form of such knowledge are instance-level constraints that either require or deny a link between data elements. These are called *must-link* and *cannot-link* constraints respectively. Depending on the way these restraints are used, constrained spectral clustering approaches may be classified into two groups: the ones that integrate the constraints in the adjacency matrix, and the ones that integrate the constraints in the optimization criterion.

Lu and Carreira-Perpiñán [LCP08] demonstrated one of the first results of incorporating constraints into the spectral clustering method. They do so by modifying the affinity matrix of the graph. They observe that only changing the weights of the constraint edges has a negligidble effect on the resulting clustering unless the number of consraint edges is very large. Therefore, they proposed to propagate the constraints over the affinity matrix. The propagation is based on Gaussian process formulation. They demonstrated good results, however admit that the method does not take full advantage of cannot-link constraints and also needs a more natural extension to the k-way partitioning case.

Sharma et al. [SvLH10] proposed a constrained spectral clustering algorithm for supervised segmentation of 3D shapes. Their formulation admits both cannot-link and must-link pair-wise constraints. Constraints are used to modify the edge structure of the graph and thus fall in the first category as well. They compute a low-dimensional embedding of the data based on the spectrum of the unnormalized Laplacian and propagate the consraints in that space.

In the work of Wang et al. [WQD12] the constraints are used to restrict the feasible solution space. They accept both hard and soft constraints and encode them in a special constraints matrix. Given a cluster indicator matrix, it is easy to quantify how well the constraints are satisfied. This measure is lower-bounded by some value, and this condition is appended to the stardard spectral clustering optimization problem. Solving it is split into two steps. In the first step a set of candidate feasible solutions are obtained using the Karush–Kuhn–Tucker theorem. By manipulating the KKT conditions they arrive at a generalized eigenproblem, which could be solved deterministically in polynomial time. In the second step all of these solutions are evaluated and the one with the best objective function value is selected.

Another form of constrained spectral clustering, often referred to as signed spectral clustering, has emerged recently from the research in the area of network analysis. That formulation admits negative edge weights. Kunegis et al. [KSL$^+$10] introduced a signed Laplacian matrix and proved that it is positive-semidefinite. Then they derived an extension of the ratio cut objective, optimizing which translates into trace minimization of signed Laplacian matrix. Unfortunately, their derivation is only valid for 2-way partitioning. This weakness was pointed out by Chiang et al. [CWD12].

Chapter 4

Approach

In this chapter we present our two proposed approaches to geometrical segmentation of point cloud data. Both methods share a common pipeline; the data are downsampled and mapped onto a graph, which is then partitioned using the ideas from spectral graph clustering. Despite this structural similarity, the approaches differ in how they implement each stage of the pipeline. In the following we shall first describe Spectral Supervoxel Clustering (SSC), a method that makes use of a mid-level representation of point clouds with supervoxels. It will be followed by the description of Spectral Voxel Clustering (SVC), a method that operates directly on voxelized point clouds and is more computationally demanding.

4.1 Spectral Supervoxel Clustering

In the first approach we apply ideas from spectral graph clustering to group a mid-level representation of a point cloud–supervoxels–into geometrically meaningful regions. A high-level view of the flow of the clustering algorithm is presented in Figure 4.1. The three main stages consist of over-segmentation of an input point cloud via supervoxel clustering, followed by construction of a graph, and concluded by spectral clustering of this graph. The stages are described in great detail in the following subsections.

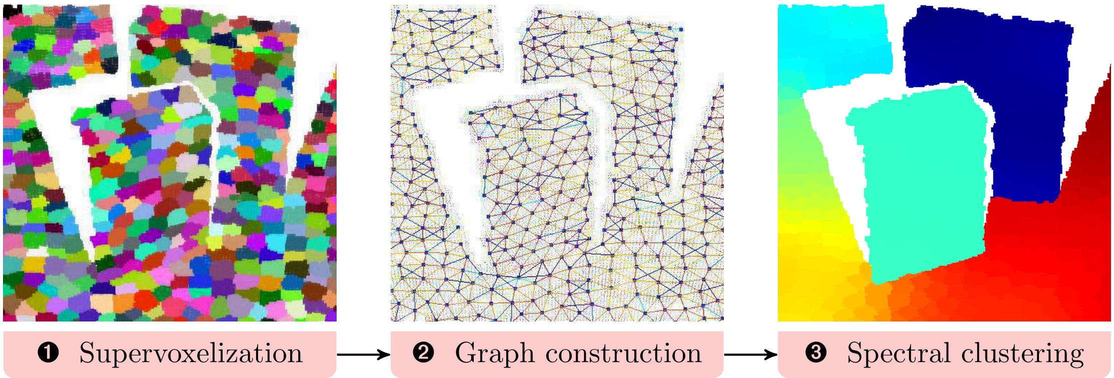


Figure 4.1: The three main steps of the Spectral Supervoxel Clustering pipeline.

4.1.1 Supervoxelization

The purpose of this step is to construct an approximation of the surfaces represented by the point cloud. As discussed in Section 2.1.2.1, voxelization may be used to significantly reduce the size of a point cloud, simultaneously achieving a certain degree of smoothing and inducing a neighborhood system. Being a purely volumetric filter, however, voxelization does not take into account the topology of the surfaces from which the points were sampled. This means that the points that fall into the same volume unit will inevitably be merged even if they belong to distinct surfaces. Consequently, voxelization introduces undesirable artifacts when the grid resolution surpasses a certain level. Therefore, a different strategy has to be emloyed to further downsample and approximate the point cloud.

Supervoxelization is a process of grouping together similar voxels in order to produce an over-segmentation of the point cloud. The idea emerged and became popular in the contex of image segmentation, where pixels are clustered into small regions based on local low-level features such as brightness, color, or texture [AMFM11]. Likewise, voxels may be grouped based on the local geometrical features such as distances and normals. These groups will consist of voxels that are close to each other and have similar normals. Therefore, supervoxels represent nearly-planar patches of a surface. It is desirable that supervoxels do not violate object boundaries, have uniform size, and are regularly distributed.

The supervoxelization algorithm of Papon et al. [PASW13] is used in our work[1]. It

---

[1] The code is freely distributed as a part of the Point Cloud Library (`https://github.com/PointCloudLibrary/pcl`).

implements a variant of local k-means clustering that works directly on voxelized point clouds, as compared to other existing methods [WGB12] that operate in the projected image plane and are only suitable for RGB-D input. The algorithm allows the user to indirectly control the average size and number of supervoxels via the seed resolution parameter. Figure 4.2 demonstrates a point cloud and several over-segmentations obtained with different seed resolutions.

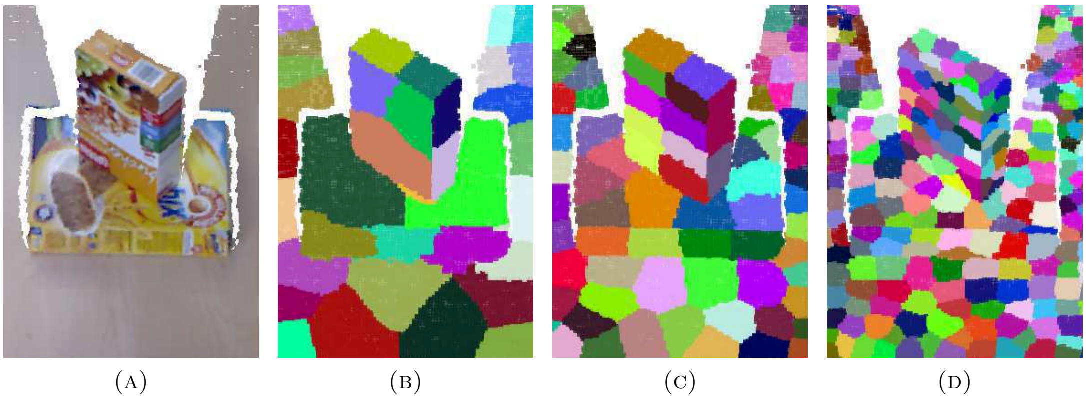

(A) (B) (C) (D)

FIGURE 4.2: Point cloud supervoxelization. (a) Original point cloud. (b, c, d) Supervoxels produced with different seed resolutions (16 cm, 8 cm and 4 cm respectively).

We experimented with different values for the seed resolution parameter and observed that the smaller the resolution, the faster the algorithm works. Moreover, small seed resolutions yield small supervoxels, which better capture object boundaries. On the other hand, the amount of supervoxel is large when the seed resolution is small. We found the seed resolution 3 cm to be a good tradeoff.

4.1.2 Graph Construction

In this step a graph representation of the supervoxelized point cloud is constructed. The vertex set of the graph represents supervoxels, and the edges define topological adjacency relations between them. Finally, edge weights reflect geometrical similarity between supervoxels.

4.1.2.1 Vertex and Edge Sets

A supervoxel is modeled by a vertex with two properties: centroid position and normal orientation. The centroid is computed by averaging the positions of the voxels that belong to the supervoxel, and the normal orientation is estimated using PCA (see Section 2.1.2.2). Figure 4.3b demonstrates centroids and normals superimposed on the supervoxels. In the following we will denote the centroid position and normal orientation associated with a graph vertex v_i as $\mathbf{c}_i$ and $\mathbf{n}_i$ respectively.

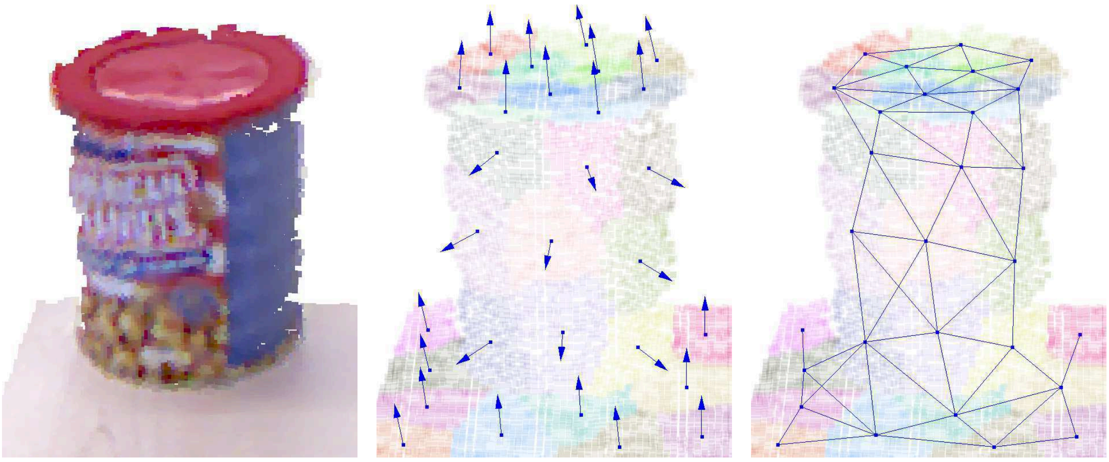

(A) (B) (C)

FIGURE 4.3: Supervoxel graph construction. (a) Original point cloud. (b) Vertices model supervoxels' centroids and normal orientations. (c) Edges model adjacency relations between supervoxels.

Adjacency edges are established between pairs of supervoxels that have a common border. In other words, two supervoxels are considered to be adjacent if some of their voxels are neighbors, as illustrated in Figure 4.3c. Therefore, the adjacency edges define the topology of the over-segmented surface. It is important to stress that proximity in Euclidean sense does not necessarily imply adjacency between supervoxels.

4.1.2.2 Edge Convexity Heuristic

According to the minima rule formulated in Section 1.2, concave and convex boundaries should be treated differently. Therefore, we need to distinguish between locally convex and concave connections between supervoxels. Several different heuristics to detect convexity may be found in the literature. For example, Karpathy et al. [KMFF13] say that a vertex v_i is relatively convex to v_j if the normal $\mathbf{n}_i$ points away from the centroid $\mathbf{c}_j$, thus indicating that the surface is curving outwards. If both v_i is relatively convex to v_j and vice versa, then the connection between them is considered convex. Formally, this could be described with

$$Convex\left(v_i, v_j\right) = \begin{cases} \text{true} & \text{if } \left(\mathbf{n}_j - \mathbf{n}_i\right) \cdot \left(\mathbf{c}_j - \mathbf{c}_i\right) > 0 \\ \text{false} & \text{otherwise} \end{cases}. \tag{4.1}$$

Lai et al. [LHMR09] consider the plane passing through the centroid $\mathbf{c}_j$, parallel to the normal $\mathbf{n}_i$. If the normal $\mathbf{n}_j$ lies on the opposite side of the plane to the centroid $\mathbf{c}_i$, then the connection is convex. Formally, this could be written as

$$Convex\left(v_i, v_j\right) = \begin{cases} \text{true} & \text{if } \left(\mathbf{d} - \left(\mathbf{d} \cdot \mathbf{n}_i\right)\mathbf{n}_i\right) \cdot \mathbf{n}_j > 0 \\ \text{false} & \text{otherwise} \end{cases}, \tag{4.2}$$

where $\mathbf{d} = \left(\mathbf{c}_j - \mathbf{c}_i\right)$ is the vector joining the centroids.

Our experiments indicate that in the absolute majority of cases the two heuristics output the same result. Taking into account that Equation 4.1 is slightly cheaper to evaluate, we decided to use it in our algorithm.

4.1.2.3 Edge Weighting Function

Given a pair of adjacent supervoxels, the edge weight is chosen to reflect the likelihood of them belonging to the same object. Difference in normal orientations and convexity type are the two cues used to compute it. Formally,

$$\omega\left(v_i, v_j\right) = \begin{cases} 1 & \text{if } Convex\left(v_i, v_j\right) \\ \phi\left(\left\|\mathbf{n}_i - \mathbf{n}_j\right\|\right) & \text{otherwise} \end{cases}, \tag{4.3}$$

where $\phi\left(\cdot\right)$ is a radial basis function. Typical choices are the Gaussian $e^{-\frac{\left\|\mathbf{n}_i - \mathbf{n}_j\right\|^2}{\sigma}}$ and inverse quadratic $\frac{1}{1+\sigma\left\|\mathbf{n}_i - \mathbf{n}_j\right\|^2}$ functions. Both have a single free parameter and behave similarly. The empirical study of Grady and Jolly [GJ08] suggests that the latter performs better, which is reconfirmed by our experiments. Therefore, we use inverse quadratic function in our algorithm.

4.1.2.4 Edge Validity Heuristic

In certain rare cases an adjacency edge is established between supervoxels that belong to two geometrically disjoint surfaces which happen to touch in a single point. An example is given in Figure 4.4, where the top horizontal surface of the left box touches with the front vertical surface of the middle box in a single point. Edges like this need to be identified and removed, because the convexity heuristic is not applicable and edge weight calculation does not make sense.

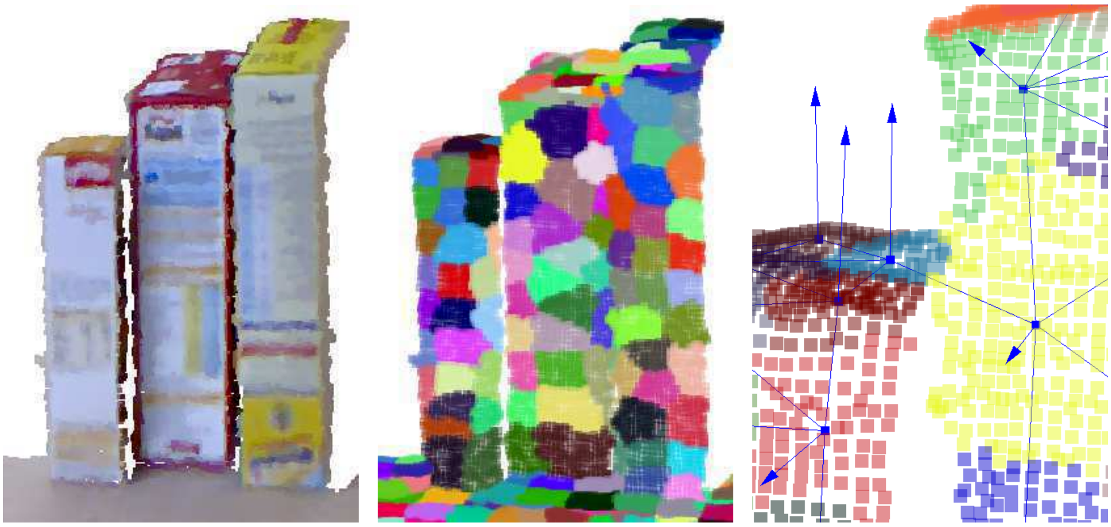

(A) (B) (C)

Figure 4.4: Invalid adjacency edges. (a) Original point cloud. (b) Supervoxels. (c) Close-up view of the region where the left and the middle boxes touch and an invalid adjacency edge is established.

In a work contemporary to ours, Stein et al. [SWS+14] propose a *sanity criterion* to identify and invalidate such connections. Building on the premise that supervoxels approximate linear patches of the surface, they consider the line of intersection between the planes given by these patches $\mathbf{s} = (\mathbf{n}_i \times \mathbf{n}_j)$. They define the angle

$$\vartheta_{ij} = \min\left(\measuredangle\left(\mathbf{d}, \mathbf{s}\right), \measuredangle\left(\mathbf{d}, -\mathbf{s}\right)\right), \tag{4.4}$$

where $\mathbf{d}$ is the vector joining the centroids, $\measuredangle(\cdot,\cdot)$ denotes the measured angle between two vectors, and minimum in introduced to account for the fact that the direction of $\mathbf{s}$ is not important. They observe that singular configurations tend to occur when ϑ is smaller than a certain threshold, i.e. when $\mathbf{d}$ and $\mathbf{s}$ are nearly parallel. The threshold depends on the angle between normals $\beta_{ij} = \measuredangle(\mathbf{n}_i, \mathbf{n}_j)$ and has the form of a sigmoid

$$\vartheta_{\text{thresh}}(\beta_{ij}) = \vartheta_{\text{thresh}}^{\max} \cdot \left(1 + \exp\left\{-a \cdot (\beta_{ij} - \beta_{\text{off}})\right\}\right)^{-1}, \tag{4.5}$$

where the values $\vartheta_{\text{thresh}}^{\max} = 60°$, $\beta_{\text{off}} = 25°$, and $a = 0.25$ were derived experimentally. Summarizing, the heuristic is written as

$$Sane(v_i, v_j) = \begin{cases} \text{true} & \text{if } \vartheta_{ij} > \vartheta_{\text{thresh}}(\beta_{ij}) \\ \text{false} & \text{otherwise} \end{cases}. \tag{4.6}$$

According to our observations, this heuristic is overly pessimistic, i.e. it tends to invalidate many "sane" edges. We used the point clouds from the OSD dataset (Section 5.1) to collect a large amount of experimental data about adjacency edges. Each point cloud was supervoxelized and adjacency edges between supervoxels were labeled as intra- or inter-cluster based on the ground truth segmentation. Figure 4.5 presents an empirical distribution of edges and the decision boundary of the sanity criterion defined in Equation 4.6. Small blue and large red dots represent intra- and inter-cluster edges (according to the ground truth annotations) respectively. The position of a dot is defined by β and ϑ angles. We found the parameter values $\vartheta_{\text{thresh}}^{\max} = 57°$, $\beta_{\text{off}} = 33°$, and $a = 0.3$ to provide a tighter boundary and consequently used them in our algorithm.

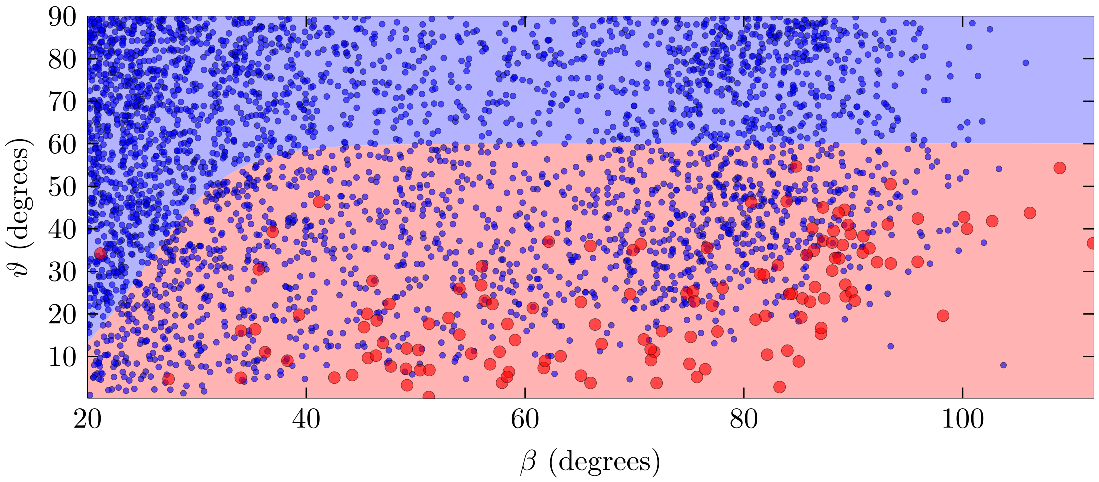


FIGURE 4.5: Convex edges and sanity criterion decision boundary. Small blue dots represent intra-cluster edges and large red dots represent inter-cluster edges.

4.1.2.5 Cannot-Link Constraint Heuristic

We observe that concave edges with large differences in normal orientations almost certainly connect supervoxels that belong to different objects. Such edges may be regarded as *cannot-link* (CL) constraints for the clustering problem or otherwise used in the partition recovering procedure as will be detailed in Section 4.1.3.3. We define the cannot-link contraint heuristic as

$$CL\left(v_i, v_j\right) = \begin{cases} \text{true} & \text{if}\, \omega_{ij} < \omega_{thresh} \\ \text{false} & \text{otherwise} \end{cases} . \tag{4.7}$$

Our experiments indicate that 45° difference is a reasonable threshold.

4.1.3 Spectral Clustering

In this step the constructed graph is partitioned into disjoint regions to fullfill the segmentation goal. This section details how the spectral graph clustering machinery, introduced in Section 2.3 in a general form, is adapted to serve the purposes of partitioning the supervoxel graph into geometrically meaningful regions.

4.1.3.1 Partitioning Objective

The ultimate segmentation goal is to split the input point cloud into objects and their parts along the concave boundaries. This could be restated in terms of the supervoxel graph, onto which the point cloud data were mapped in the previous steps. Adjacent supervoxels that are either parallel or relatively convex to each other should be grouped into the same cluster. Conversely, relatively concave supervoxels should be assigned to different clusters.

This goal could be restated once again in terms of the edge weights. The first subgoal requires the total weight of the cut edges to be small. Indeed, the weighting function assigns large values to convex and flat edges; avoiding cutting such edges keeps the total weight small. The second subgoal requires the total weight of the edges that remain uncut to be large.

Unfortunately, the collection of functions and optimization problems presented in Section 2.3 does not include the desired objective. The closest match are the ratio and normalized cut objectives; they implement the first subgoal and also promote balanced partitions. The second subgoal is not accounted for explicitly, however in practice these work well enough.

It has been noted that using both ratio and normalized cut objectives yields similar results. It has also been noticed that the spectral properties of the normalized Laplacian are less favorable, because the spread of the eigenspace is typically smaller than for the unnormalized Laplacian constructed from the same graph. In our experiments we did not observe a noticeable difference in performance. Furthermore, the results were always almost identical.

4.1.3.2 Spectral Relaxation

The discrete optimization problem that arises from the partitioning objective is relaxed into a real-valued problem to allow for efficient solution using eigendecomposition. Its first k eigenvectors form a real-valued approximation of the indicator matrix. A concrete example is presented below to expose the problems associated with spectral relaxation. Consider the point cloud with books and boxes from Figure 1.1. We supervoxelized it, constructed a graph, and solved the eigenproblem from Equation 2.28. Each eigenvector associates a real-valued coefficient to every supervoxel. This is used to visualize eigenvectors in the top row of Figure 4.6. Supervoxels are colored according to their associated coefficients (from dark blue for the smallest to dark red for the largest). The bottom row presents scatter plots of eigenvector elements, sorted in nondecreasing order.

The first eigenvector approximates a discrete indicator vector very closely. Indeed, its coefficients assume two distinct values with a large gap and highlight the left box. The second eigenvector indicates the right box, though the gap is smaller and there are more than two distinct values that its coefficients assume. In the third and fourth eigenvectors coefficients vary smoothly and there is no evident gap. On the other hand, there seems to be a vivid border between the boxes in the third eigenvector. Intuitively, one can see that the information needed to derive the desired partition is contained in the eigenvectors, however recovering it automatically without knowing the desired number of segments is not a trivial task.

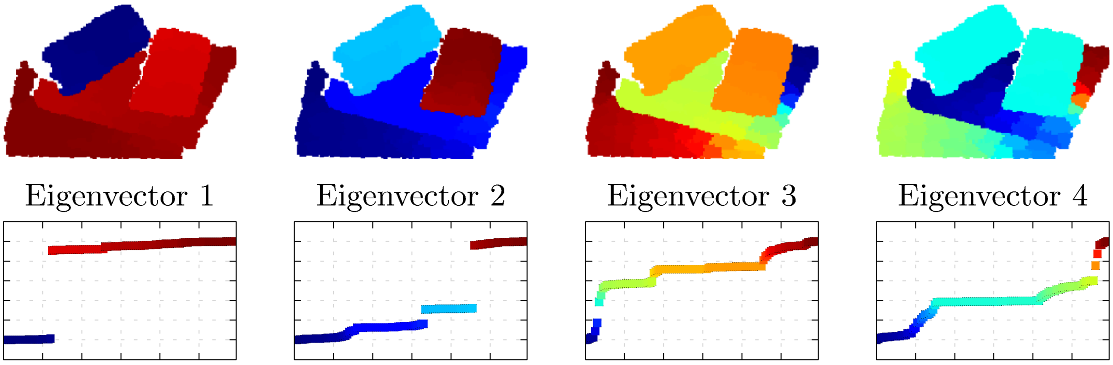


Figure 4.6: First four eigenvectors, computed for the point cloud from Figure 1.1. Each eigenvector is scaled so that its' elements are in $[0, 1]$. *Top Row*: Supervoxels are painted according to their corresponding elements in eigenvectors. *Bottom Row*: Elements of eigenvectors sorted in nondecreasing order.

In the lack of information about the number of segments, the choice of the appropriate number k of eigenvectors is not trivial. Several experiments (as detailed in Section 5.3.2) were performed to find a reasonable number.

4.1.3.3 Graph Partitioning Algorithm

Section 2.3.3.1 introduced the idea of basic iterative algorithm for deriving a graph partition from the spectral data. Our approach is based on it, however is different in a number of ways, as detailed below.

Algorithm 1 outlines the Spectral Supervoxel Clustering procedure. As an input it accepts a supervoxel graph G, which becomes the first subgraph in the processing queue $\mathcal{Q}$. (Note that a graph is its own subgraph.) The outer loop pops a subgraph from the queue, constructs the unnormalized graph Laplacian matrix, and solves the eigenproblem. In contrast with the method of Hagen and Kahng [HK92], where a single eigenvector is used, we compute multiple eigenpairs (see Section 5.3.2 for the experimental determination of the number of eigenpairs that have to be computed). In the inner loop the subgraph is recursively bipartitioned based on the computed eigenvectors. When a (sub)subgraph can not be split any further based on the previously computed eigenvectors, it is pushed to the queue Q; in later iterations a new eigenproblem will be build and solved for it.

An utility function is defined that assigns a fitness value to each graph bipartition. It is used both to find the best cut, as well as to decide if the cut is good enough to be

Algorithm 1 Spectral Supervoxel Clustering

```
function SpectralSupervoxelClustering($G$)
    $\mathcal{C} \leftarrow \emptyset$                                ▷ $\mathcal{C}$: final set of clusters
    $\mathcal{Q} \leftarrow \{G\}$                              ▷ $\mathcal{Q}$: queue of subgraphs for processing
    while $sizeof(\mathcal{Q}) > 0$ do
        $S \leftarrow pop(\mathcal{Q})$
        $L \leftarrow$ BuildLaplacian($S$)
        $\lambda_1 \dots \lambda_k, \mathbf{v}_1 \dots \mathbf{v}_k \leftarrow$ EigenDecomposition($L$)
        $\mathcal{A} \leftarrow \{S\}$                          ▷ $\mathcal{A}$: queue of "active" subgraphs
        $i \leftarrow 0$                                 ▷ $i$: cut counter
        while $sizeof(\mathcal{A}) > 0$ do
            $C \leftarrow pop(\mathcal{A})$
            $cut \leftarrow$ FindBestCut($C, \mathbf{v}_1 \dots \mathbf{v}_k$)
            if CutUtility($C, cut$) $> threshold$ then
                $C_1, C_2 \leftarrow$ CutSubgraph($C, cut$)
                $\mathcal{A} \leftarrow \mathcal{A} \cup \{C_1, C_2\}$
                $i \leftarrow i + 1$
            else
                if $i > 0$ then
                    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{C\}$
        if $i = 0$ then
            $\mathcal{C} \leftarrow \mathcal{C} \cup \{S\}$
    return $C$
```

committed. The standard approach is to use the clustering objective function as an utility function. Hagen and Kahng [HK92] used ratio cut, whereas Shi and Malik [SM00] used normalized cut. In contrast, we propose an utility function that takes into account the specifics of our application domain, the cannot-link constraints introduced in Section 4.1.2.5. We favor partitions where the total weight of the cut edges is small, but the number of CL edges among them is large. Formally, the utility function is

$$Utility\left(\mathcal{C}, \bar{\mathcal{C}}\right) = \frac{\left|\left\{CL\left(v, u\right), v \in \mathcal{C}, u \in \bar{\mathcal{C}}\right\}\right|}{Cut\left(\mathcal{C}, \bar{\mathcal{C}}\right)}. \tag{4.8}$$

Algorithm 2 presents the procedure used to find the best graph bipartition from the spectral data contained in a collection of eigenvectors. It is based on the observation that each eigenvector induces a *linear ordering* of the graph vertices. Therefore, by selecting a splitting vertex one obtains a bipartition of the graph. Furthermore, iterating over the graph vertices in the order prescribed by an eigenvector, one could find the splitting vertex which induces a cut of the highest utility.

Algorithm 2 Find Best Cut

```
function FindBestCut($C, \mathbf{v}_1 \dots \mathbf{v}_k$)
    $cut_i, cut_v, cut_s \leftarrow 0, 0, 0$                ▷ $cut$: best found cut (index, vertex, score)
    for $e$ in $[1 \dots k]$ do
        $w \leftarrow 0$                                      ▷ $w$: total weight of the current cut
        $cl \leftarrow 0$                                     ▷ $cl$: total number of CL edges cut
        for $v$ in $argsort\left(\mathbf{v}_e\right)$ do
            for $u$ in $\mathcal{N}\left(v\right)$ do
                if $\{v, u\} \not\subset C$ then
                    continue
                if $v_e\left(v\right) < v_e\left(u\right)$ then
                    $w \leftarrow w + \omega\left(v, u\right)$
                    if $CL\left(u, v\right)$ then
                        $cl \leftarrow cl + 1$
                else
                    $w \leftarrow w - \omega\left(v, u\right)$
                    if $CL\left(u, v\right)$ then
                        $cl \leftarrow cl - 1$
            if $\frac{cl}{w} > cut_s$ then
                $cut_i, cut_v, cut_s \leftarrow e, v, \frac{cl}{w}$
    return $cut_i, cut_v$
```

Note that sometimes while splitting the graph may be broken into more than two components.

4.1.4 Constrained Spectral Clustering

The previous section detailed the procedure of spectral supervoxel clustering. The cannot-link constraints played an important role in it, however they were exploited only in the process of partition recovery. Here we propose an alternative approach to supervoxel clustering, where the cannot-link constraints are directly incorporated into the objective function and, therefore, participate in spectral relaxation.

It could be observed that the edges marked as cannot-link constraints are virtually discarded from the graph clustering problem. Indeed, the weights of such edges are very small, and their contribution the cut size is unnoticeable. Consequently, they have no effect on ratio or normalized cut objectives that are used in spectral clustering. However, an edge with small weight is semantically different from a lack of edge. Lack of edge between two supervoxels provides no information, whereas a small weight edge suggests that the two supervoxels are geometrically connected, but are likely to belong to different clusters. Therefore, it is desirable to adjust the objective functions so that this information is leveraged during spectral relaxation.

We make use of the ideas from the domain of community network analysis, where graph edges model friend and enemy relations. The former are assigned positive weights, whereas the latter get negative weights. Unfortunately, none of the graph Laplacian matrices presented in Section 2.2.2 may be used in the spectral relaxation procedure, because they lose the property of being positive semi-definite. To circumvent this problem, Kunegis et al. [KSL+10] proposed the signed Laplacian matrix and a set of associated signed objects.

The *signed vertex degree* is defined as

$$\bar{deg}\left(v\right) = \sum_{u \in \mathcal{N}(v)} \left|\omega\left(v, u\right)\right|, \tag{4.9}$$

and the *signed degree matrix* $\bar{D}$ is obtained by arranging signed vertex degrees into a diagonal matrix. Further, *positive* and *negative adjacency matrices* A^{+} and A^{-} are defined. The former contains only the positive coefficients of the adjacency matrix. The latter contains absolute values of only the negative coefficients of the adjacency matrix. It follows that $A = A^{+} - A^{-}$. Consequently, *positive* and *negative cut sizes* that count the total weight of all positive or all negative edges crossing a cut are defined as

$$Cut^{+}\left(\mathcal{C}, \bar{\mathcal{C}}\right) = \sum_{v_i \in \mathcal{C}, v_j \in \bar{\mathcal{C}}} A^{+}_{ij} \tag{4.10}$$

and

$$Cut^{-}\left(\mathcal{C}, \bar{\mathcal{C}}\right) = \sum_{v_i \in \mathcal{C}, v_j \in \bar{\mathcal{C}}} A^{-}_{ij}. \tag{4.11}$$

Both are united into the *signed cut size*

$$SignedCut\left(\mathcal{C}, \bar{\mathcal{C}}\right) = 2 \cdot Cut^{+}\left(\mathcal{C}, \bar{\mathcal{C}}\right) + Cut^{-}\left(\mathcal{C}, \mathcal{C}\right) + Cut^{-}\left(\bar{\mathcal{C}}, \bar{\mathcal{C}}\right). \tag{4.12}$$

Finally, the *signed ratio cut* objective is

$$SignedRatioCut\left(\mathcal{C}, \bar{\mathcal{C}}\right) = SignedCut\left(\mathcal{C}, \bar{\mathcal{C}}\right) \left(\frac{1}{|\mathcal{C}|} + \frac{1}{|\bar{\mathcal{C}}|}\right). \tag{4.13}$$

Minimization of the signed ratio cut leads to partitions where the two clusters are connected by few positive edges and contain few negative edges within them.

The signed Laplacian matrix is defined as

$$\bar{L} = \bar{D} - A. \tag{4.14}$$

Similarly, signed versions of the normalized Laplacians in Equations 2.13 and 2.14 could be obtained by substituting the signed degree matrix. Kunegis et al. [KSL$^+$10] proved that these matrices are positive semi-definite. Furthermore, they demonstrated that the problem of minimizing the signed ratio cut can be reformulated as a trace minimization problem for a quadratic form involving the signed Laplacian. Unfortunately, their derivation is only valid for 2-way partitioning case. Chiang et al. [CWD12] have formally proved that it can not be extended to k-way partitioning.

Our approach that implements constrained spectral supervoxel clustering is same as Algorithm 1, with two minor differences. Firstly, instead of building the standard graph Laplacian, its signed version is constructed. The edges marked by the cannot-link constraint heuristic are assigned a fixed negative weight ω_{CL} (-1.0 in our case, see Section 5.3.3 for experiments). Secondly, only a single eigenvector is computed in the eigendecomposition step due to the reasons mentioned earlier. The remaining steps, including cut utility computation and determination of the best cut, are the same.

4.2 Spectral Voxel Clustering

In our second approach—Spectral Voxel Clustering (SVC)—the same basic pipeline is used, however in certain aspects the method is significantly different. In this section we describe SVC step-by-step, focusing our attention on the new ideas.

4.2.1 Voxelization

The purpose of this step is to downsample the input point cloud and induce a neighborhood system. The procedure was described in Section 2.1.2.1, furthermore in Section 4.1.1 we argued that the voxelization may yield undesirable artifacts when the resolution is too large. According to our experiments, the resolution of 8 mm presents a good trade-off between data compression and quality loss.

4.2.2 Graph Construction

In this step a graph representation of the voxelized point cloud is constructed. Voxels become graph vertices, and edges define geometrical adjacency between them. Similarly to how this is done in SSC, edge weights reflect geometrical similarity between voxels.

4.2.2.1 Vertex and Edge Sets

A voxel is modeled by a vertex with three properties: position, normal orientation, and curvature. The latter two are estimated jointly using PCA (see Section 2.1.2.2).

Adjacency edges are established between pairs of neighboring voxels. Recall that voxels are the cells of a regular grid. We use 18-neighborhood system, i.e. the system where the cells sharing an edge or a face are considered to be neighbors.

4.2.2.2 Edge Convexity Heuristic

The edge convexity heuristic used in supervoxel graphs (see Equation 4.1) could be applied in voxel graphs as well. However, in certain situations it does not produce desirable results. Voxels are significantly smaller than supervoxels and may fall exactly on a concave boundary between surfaces. A pair of such voxels will have similar normals and the edge connecting them will be considered convex. Consequently, a graph will contain a convex edge within a concave boundary, which is not desirable. This issue is addressed by introducing the concept of *voxel convexity*.

A voxel is considered to be convex if it is relatively convex to the majority of its direct neighbors. Formally, this is defined as

$$VoxelConvex\left(v_i\right) = \begin{cases} \text{true} & \text{if } \sum_{v_j \sim v_i} sign\left(\mathbf{d}_{ij} \cdot \mathbf{n}_j\right) > 0 \\ \text{false} & \text{otherwise} \end{cases}, \tag{4.15}$$

where $\mathbf{d}_{ij}$ is the vector joining voxels v_i and v_j. With this definition, a graph edge is called convex if both voxels that it connects are convex, and concave otherwise.

4.2.2.3 Edge Weighting Function

Given a pair of adjacent voxels, the edge weight is chosen to reflect the geometrical similarity between them. There are two factors—difference in normal orientations and curvature—that contribute to it. Formally,

$$\omega\left(v_i, v_j\right) = \begin{cases} 1 & \text{if } Convex\left(v_i, v_j\right) \\ \phi_n\left(\left\|\mathbf{n}_i - \mathbf{n}_j\right\|\right) \cdot \phi_c\left(c_i c_j\right) & \text{otherwise} \end{cases}, \tag{4.16}$$

where c_i and c_j are the voxel curvatures, and $\phi_n(\cdot)$ and $\phi_c(\cdot)$ are radial basis functions used to balance the contributions of the two factors. Similarly to SSC, inverse quadratic functions are utilized here.

4.2.3 Spectral Clustering

The constructed voxel graph is partitioned into segments in this step. Spectral relaxation of ratio cut minimization problem is exploited, however compared with the supervoxel graph case, a different utility function is used to derive graph partition from the relaxed solution. The utility function (Equation 4.8) is based on the concept of cannot-link edges, which only emerges at the level of supervoxels. Indeed, when individual voxels are considered, a concave edge does not necessarily imply that the voxels that it connects belong to two different objects. A large difference in normal orientations can as well be due to noisy surface approximation or due to noise in estimated normals themselves.

In addition to edge weights, we introduce a quantity called *edge potential*. Given two connected graph vertices v and u and a set of eigenvectors $\mathbf{v}_1 \ldots \mathbf{v}_k$ obtained through eigendecomposition of the graph Laplacian, the i^{th} potential of the edge between these vertices is defined as the absolute value of the difference of their corresponding elements in the i^{th} eigenvector

$$\phi_i\left(v, u\right) = \left|v_i\left(v\right) - v_i\left(u\right)\right|. \tag{4.17}$$

Similarly to the cut size that sums up the weights of the inter-cluster edges, the i^{th} *cut potential* is defined as

$$\Phi_i\left(\mathcal{C},\bar{\mathcal{C}}\right) = \sum_{v\in\mathcal{C},u\in\bar{\mathcal{C}}} \phi_i\left(v,u\right). \tag{4.18}$$

The larger the cut potential, the more salient the cut is. Recall the example with books and boxes in Figure 4.6. The 1^{st} potential of the cut splitting the small left box from the rest of the point cloud is large, because the elements of the eigenvector $\mathbf{v}_1$ corresponding to the box are all close to zero, whereas the elements corresponding to the other objects are close to one. Conversely, the 3^{rd} potential of the same cut is rather small, because the corresponding elements of the eigenvector $\mathbf{v}_3$ have similar values.

Armed with this definition, we introduce a new utility function that assigns a fitness value to a graph bipartition derived from the i^{th} eigenvector. The function is a ratio between cut potential and cut size, and is formally defined as

$$Utility\left(\mathcal{C},\bar{\mathcal{C}}\right) = \frac{\Phi_i\left(\mathcal{C},\bar{\mathcal{C}}\right)}{Cut\left(\mathcal{C},\bar{\mathcal{C}}\right)}. \tag{4.19}$$

The procedure of SVC clustering is the same as in Algorithm 1, except to the function that determines the best cut. It is different because of the different utility function, and is presented in Algorithm 3.

Algorithm 3 Find Best Cut for SVC

```
function FindBestCut2($C, \mathbf{v}_1 \dots \mathbf{v}_k$)
    $cut_i, cut_v, cut_s \leftarrow 0, 0, 0$        ▷ cut: best found cut (index, vertex, score)
    for $e$ in $[1 \dots k]$ do
        $w \leftarrow 0$        ▷ $w$: total weight of the current cut
        $p \leftarrow 0$        ▷ $p$: total potential of the current cut
        for $v$ in $argsort\left(\mathbf{v}_e\right)$ do
            for $u$ in $\mathcal{N}\left(v\right)$ do
                if $\{v, u\} \not\subset C$ then
                    continue
                if $v_e\left(v\right) < v_e\left(u\right)$ then
                    $w \leftarrow w + \omega\left(v, u\right)$
                    $p \leftarrow p + \phi_e\left(v, u\right)$
                else
                    $w \leftarrow w - \omega\left(v, u\right)$
                    $p \leftarrow p - \phi_e\left(v, u\right)$
            if $\frac{p}{w} > cut_s$ then
                $cut_i, cut_v, cut_s \leftarrow e, v, \frac{p}{w}$
    return $cut_i, cut_v$
```

Chapter 5

Experiments and Evaluation

In this chapter the results of experimental evaluation of our work are reported. We start by introducing an array of publicly available segmentation datasets, among which several were selected for qualitative and quantitative evaluation. Then a number of evaluation metrics are presented. Next we describe a set of experiments that were performed to determine reasonable values for the parameters of our system. Finally, the results of a thorough evaluation are reported.

5.1 Datasets

A number of RGB-D object segmentation datasets are publicly available; three of them were used to perform evaluations in this work. Below we describe these datasets and discuss their differences. For the sake of completeness we also mention a few other datasets that were not considered in this work.

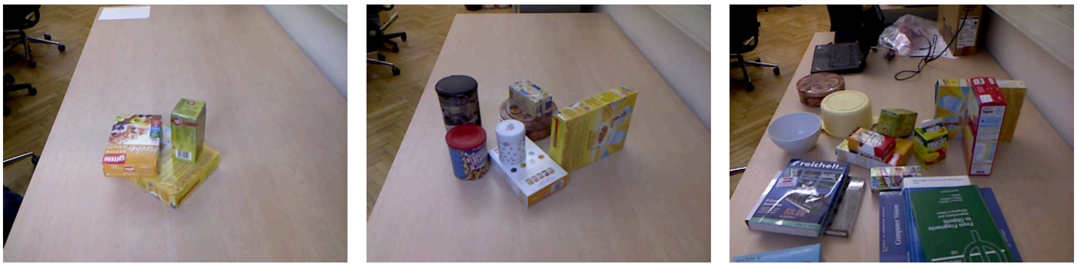
Object Segmentation Database

Object Segmentation Database (OSD[1]) was recorded by the Vision for Robotics group at Vienna University of Technology and subsequently used to evaluate the methods of Richtsfeld et al. [RZV12], Ückermann et al. [UHR12], and Stein et al. [SWS+14]. The

---

[1] `http://www.acin.tuwien.ac.at/?id=289`, retrieved on May 26, 2014.

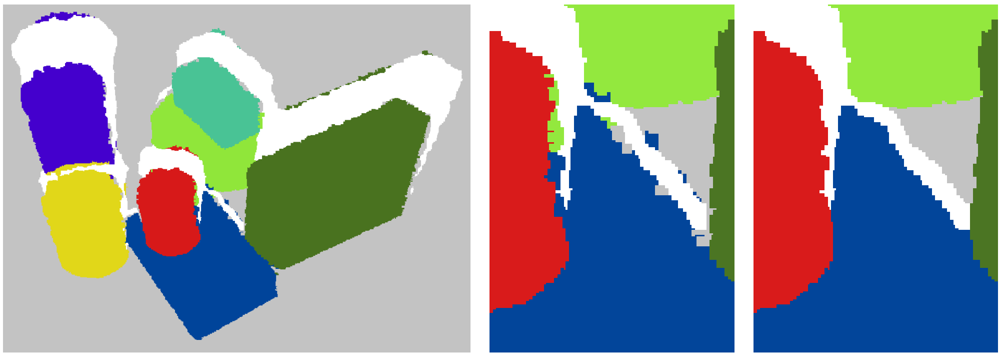
dataset includes point clouds of 111 different scenes captured from a single viewpoint using an RGB-D camera. The scenes feature a number of different box-like and cylindrical objects residing on a table. Several scenes are shown in Figure 5.1.



(A) Low complexity (B) Medium complexity (C) High complexity

FIGURE 5.1: Sample scenes of varying complexity from the OSD dataset. The scenes are composed of a table with a number of box-like and cylindrical objects on top.

All point clouds in the dataset are supplied with ground truth annotations. The authors derived these annotations from the color images of the scenes, presumably using automatic segmentation tools. Due to slightly imprecise calibration between RGB and depth sensors of the camera, object boundaries in color and range images do not coincide exactly. Therefore, the labels derived from color images tend to jump over the geometric boundaries of the objects, as displayed in Figures 5.2a and 5.2b. This is undesirable since we are interested in evaluating the performance of geometrical segmentation algorithms. For this reason we manually rectified the ground truth annotations to obtain geometrically consistent labelings, as exemplified in Figure 5.2c.



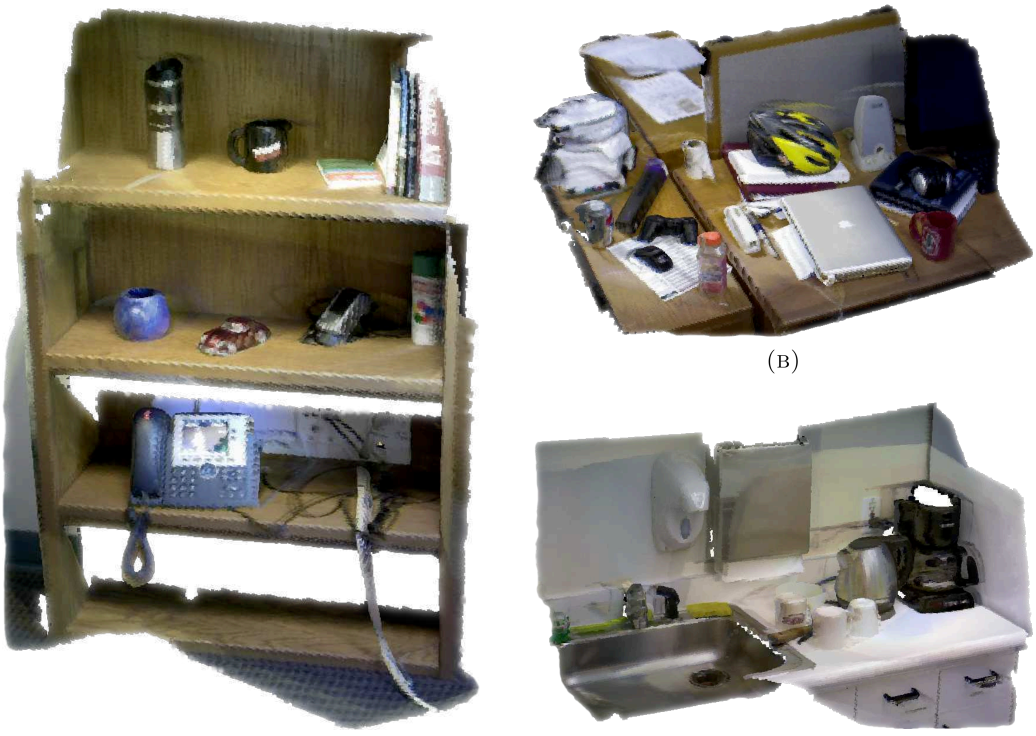
(A) (B) (C)

FIGURE 5.2: Imprecise ground truth annotations in the OSD dataset. (a) The scene from Figure 5.1b, colors according to ground truth annotation. (b) Close-up view of the central region; a considerable number of points on the edges are mislabeled. (c) Rectified ground truth annotation.

In the majority of scenes the objects occlude each other and the table they reside upon. Often an occluded object appears to be split in several disconnected parts. For example, in Figure 5.2a a small patch of the table surface in the center of the image is disconnected from the rest of the table. The point cloud data themselves do not provide enough evidence that both are parts of the same object. Therefore, it is unreasonable to expect a segmentation algorithm to assign them the same label, unless prior knowledge about scene composition and object shapes is used. In order to not penalize our algorithm for something it is not designed to do, we changed the ground truth annotations so that geometrically disconnected parts of the same object are assigned different labels.

KinFu

The KinFu dataset[2] was presented by Karpathy et al. [KMFF13] in their work on object discovery. The dataset contains point clouds with dense 3D reconstructions of 58 scenes. They were produced from multiple RGB-D frames using an open-source implementation of Kinect Fusion. Figure 5.3 demonstrates several sample scenes.



(A) (B) (C)

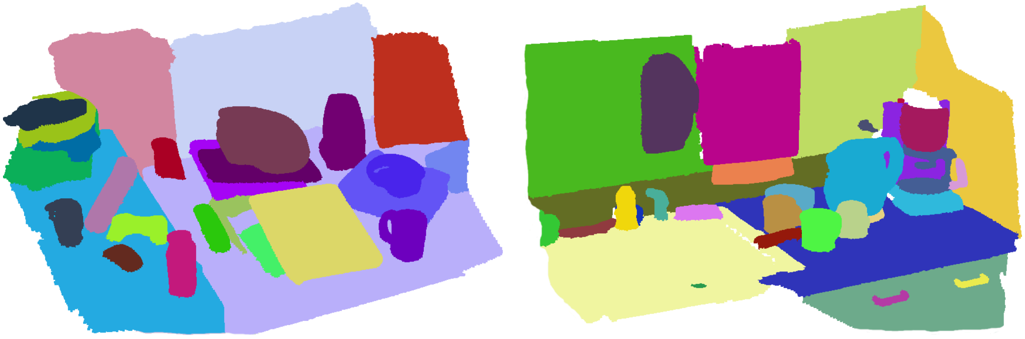
Figure 5.3: Sample scenes from the KinFu dataset. The scenes present complex real-world environments with a wide variety of objects.

---

[2] `http://cs.stanford.edu/people/karpathy/discovery`, retrieved on May 26, 2014.

This dataset is different from OSD in several important ways. The scenes are more complex and present unmanipulated real-world environments found in common households and offices. In the vast majority of scenes there are multiple supporting surfaces. A wide variety of objects with both simple (e.g. cups, books) and complex (e.g. telephone, coffee machine) shapes are present.

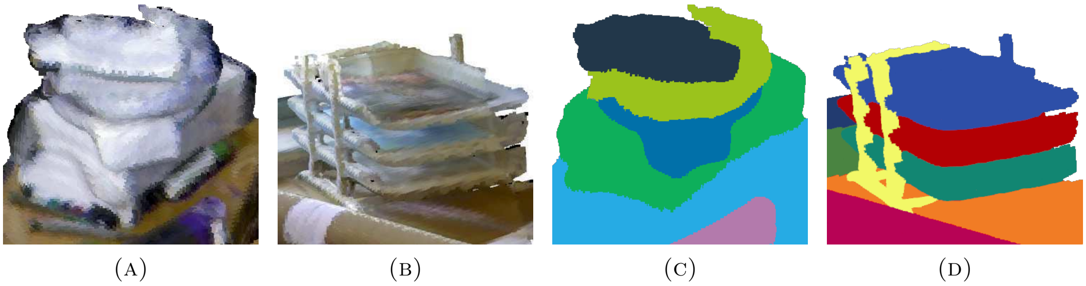
The KinFu dataset lacks ground truth annotations. In order to allow quantitative evaluation of our algorithm using this dataset, we performed manual labeling. Owing to the fact that this is a very tedious and time-consuming procedure, we annotated only 10 of the scenes. The ground truth annotations for the scenes shown in Figures 5.3b and 5.3c are presented in Figure 5.4.



(A) (B)

FIGURE 5.4: Our ground truth annotations for the KinFu scenes shown in Figures 5.3b and 5.3c.

Admittedly, we had to make a number of arbitrary choices between possible ground truth annotations. Figures 5.5a and 5.5b demonstrate two such cases. The first one contains an unknown object of highly irregular shape. We split it into parts as shown in Figure 5.5c, however depending on the desired level of detail, some of its parts may be either united, or split even further. The second figure contains a letter tray. This is a complex object that consists of multiple parts. Again, depending on the desired level of detail, individual trays may be considered as "atomic" objects (as is done in Figure 5.5d), or complex objects consisting of bottoms and walls.

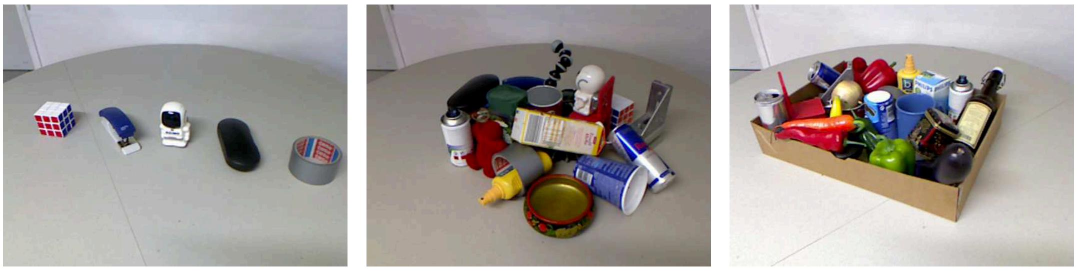


(A) (B) (C) (D)

Figure 5.5: Ambiguity of ground truth annotations in the KinFu dataset. (a) An unknown object of irregular shape. (b) A letter tray consisting of multiple parts. (c, d) One of a number of conceivable ground truth annotations for the objects (a) and (b).

Table Objects Database

Table Objects Database (TOD[3]) was used for object segmentation evaluation in the work of Potapova et al. [PZV12]. The dataset includes 89 different point clouds captured from a single viewpoint using an RGB-D camera. There are three types of scenes: with isolated free-standing objects, with occluded objects, and with objects placed in a box. In contrast with the OSD dataset, object shapes are not limited to box-like and cylindrical. There are also fruits and vegetables, office supplies, and toys. Several scenes are shown in Figure 5.6.



(A) Free-standing (B) Occluded (C) In a box

Figure 5.6: Sample scenes of different types from the TOD dataset. The scenes are composed of a table with a number of objects lined up, piled up, or put in a box.

The dataset contains ground truth annotations, however these are given in the form of polygonal areas in the color image plane. Projecting them onto the point clouds results in imprecise and often inconsistent annotations from the geometrical viewpoint. Therefore, we use this dataset only for qualitative evaluation of our algorithms.

---

[3] `https://repo.acin.tuwien.ac.at/tmp/permanent/TOSD.tar.gz`, retrieved on May 11, 2013.

Other Datasets

A number of other publicly available datasets with hand-labeled ground truth annotations exist. The RGB-D Object Dataset (ROD[4]) was recorded by the Robotics and State Estimation group at University of Washington. It contains multi-view images (the total of 250 thousand images) of scenes with a very limited degree of occlusion. This makes this dataset too simple and not interesting for our purposes. The NYU-Depth V2 dataset[5] was recorded by the Vision Learning Graphics lab team at New York University. The dataset consists of 1449 RGB-D images with 464 different indoor scenes, however it is primary targeted for scene understanding.

5.2 Evaluation Metrics

A number of methodologies for quantitative evaluation of segmentation and clustering algorithms were proposed in the literature. These include Segmentation Covering [AMFM11] and its unweighted version [HEH10], Positive/Negative scores [UHR12], Under-Segmentation Error [LSK+09], Probabilistic Rand index and its normalized version [UH05], Variation of Information [Mei05], and some others [Sch07]. This abundance is due to the inherent ill-posedness of the segmentation problem. We use the first two metrics because they provide results which convey perceptual meaning and were also used in recent works on point cloud segmentation. In addition, we introduce a pair of new metrics that measure the degree of over- and under-segmentation. These metrics are formally defined below; a few example segmentations are provided to establish a relation between numbers and intuitive perception of the segmentation quality.

5.2.1 Segmentation Covering

Consider a ground truth annotation $\mathcal{G} = \{\mathcal{G}_1, \dots \mathcal{G}_m\}$ consisting of m segments and a computed partition $\mathcal{S} = \{\mathcal{S}_1, \dots, \mathcal{S}_n\}$ consisting of n segments. The *overlap* between a pair of segments $\mathcal{G}_i$ and $\mathcal{S}_j$ is

$$\mathit{Overlap}\,(\mathcal{G}_i, \mathcal{S}_j) = \frac{|\mathcal{G}_i \cap \mathcal{S}_j|}{|\mathcal{G}_i \cup \mathcal{S}_j|}, \tag{5.1}$$

---

[4] `http://www.cs.washington.edu/rgbd-dataset/dataset.html`, retrieved on May 26, 2014.

[5] `http://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html`, retrieved on May 26, 2014.

where $|\cdot|$ denotes the number of points in a segment. Using this definition, the *weighted covering* of ground truth annotation by computed segmentation is

$$WCovering\left(\mathcal{G}, \mathcal{S}\right) = \frac{1}{N} \sum_{\mathcal{G}_i \in \mathcal{G}} |\mathcal{G}_i| \cdot \max_{\mathcal{S}_j \in \mathcal{S}} Overlap\left(\mathcal{G}_i, \mathcal{S}_j\right), \tag{5.2}$$

where N denotes the total number of points in the point cloud. In addition, Hoiem et al. [HEH10] define the *unweighted covering* as

$$UWCovering\left(\mathcal{G}, \mathcal{S}\right) = \frac{1}{|\mathcal{G}|} \sum_{\mathcal{G}_i \in \mathcal{G}} \max_{\mathcal{S}_j \in \mathcal{S}} Overlap\left(\mathcal{G}_i, \mathcal{S}_j\right). \tag{5.3}$$

The score of 1.0 signifies that the computed segmentation is identical (i.e. perfectly covers) the ground truth segmentation. These metrics have been used in a number of recent works to evaluate the performance of point cloud segmentation algorithms [SHKF12, GAM13, SWS^{+}14].

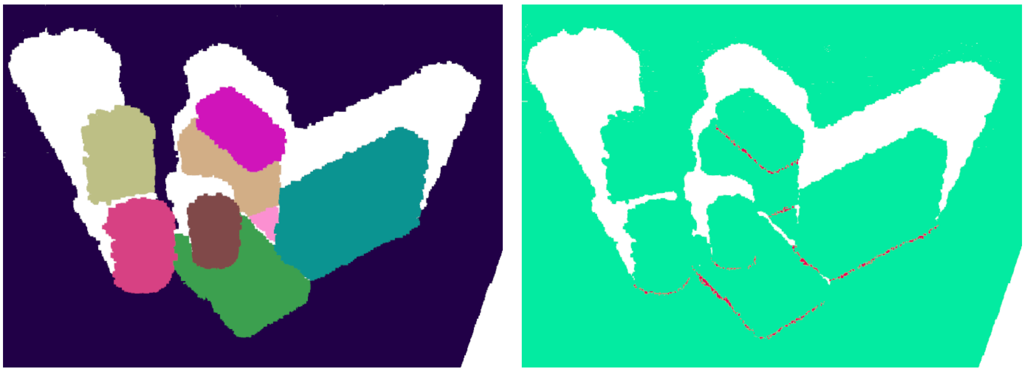
5.2.2 Under/Over Segmentation

Weighted and unweighted covering summarize the performance in a single perceptually meaningful number. However, they are dominated by large segments and do not inform whether the point cloud is under- or over-segmented. To address these problems, we introduce a second pair of metrics, *under/over segmentation*.

For each pair $\{\mathcal{G}_i, \mathcal{S}_j\}$ the number of points in their intersection is computed. Then a linear assignment problem is solved to find a one-to-one assignment with the largest total sum of intersection points. The number of ground truth segments in $\mathcal{G}$ without a match gives *under-segmentation*, whereas the number of unassigned segments in $\mathcal{S}$ gives *over-segmentation*.

5.2.3 Examples

Consider a relatively complex scene from Figure 5.1b. Figures 5.7a and 5.8 show several segmentations of different quality and Table 5.1 presents the values of evaluation metrics computed for them.



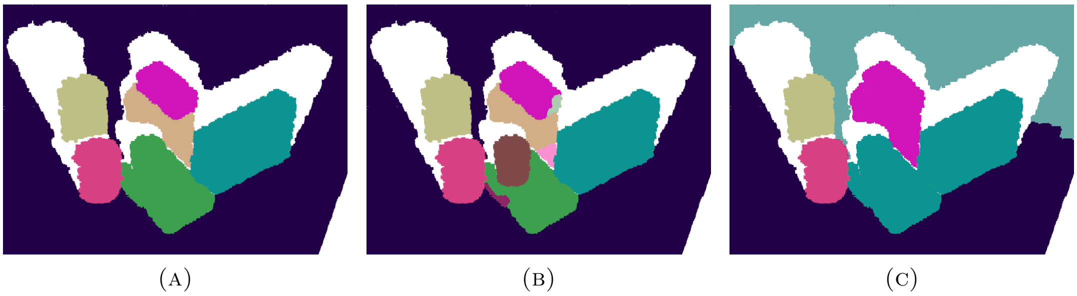
(a)

(b)

Figure 5.7: Perfect segmentation of the scene from Figure 5.1b. (a) Segmentation labels. (b) Difference with the ground truth annotation; correctly labeled points are painted green, mislabeled points are painted red.

The segmentation in Figure 5.7a may be regarded as perfect; it properly captures all the objects and their boundaries. One can see that the values of weighted and unweighted coverings are 0.988 and 0.965 respectively. This small deviation from 1.0 is due to the fact that there is a certain ambiguity in ground truth labeling when individual points at the object boundaries are considered. Figure 5.7b demonstrates which points were mislabeled according to the ground truth. Apparently, these are only a few boundary points with disputable affiliation, and therefore such covering scores should be considered as extremely good.

Figure 5.8a presents a result with mild under-segmentation. The two boxes in the center and the large round box and the table were not properly split. The weighted and unweighted covering values are 0.955 and 0.761 respectively; it signifies that the unweighted covering metric is more sensitive to under-segmentation. The weighted covering is almost not affected because of the normalization with segment sizes. The under-segmented regions are relatively small compared to the supporting table, therefore their influence is small. Figure 5.8a shows a result with mild over-segmentation. The pink and green boxes are erroneously split into two parts. One can observe almost perfect covering scores, which suggests that that they do not accurately capture small degrees of over-segmentation. Finally, Figure 5.8c demonstrates an example of a very bad segmentation. The table is over-segmented into two large parts, whereas some of the objects are erroneously grouped together. Both covering scores reflect this, however we must admit that even for this extremely bad segmentation the unweighted score is around 50%.



(a) (b) (c)

Figure 5.8: Segmentations of different quality of the scene from Figure 5.1b. (a) Mild under-segmentation. (b) Mild over-segmentation. (c) Extremely bad result with both under- and over-segmented regions.

In contrast with the covering scores that capture the overall performance in terms of the numbers of mislabeled points, the under- and over-segmentation scores give precise estimation of how many ground truth objects were not properly identified and how many extra segments were detected. These numbers could be seen in Table 5.1.

	5.7a	5.8a	5.8b	5.8c
Weighted covering	0.988	0.955	0.986	0.701
Unweighted covering	0.965	0.761	0.957	0.525
Under-segmentation	0	2	0	4
Over-segmentation	0	0	2	1

Table 5.1: Evaluation metrics computed for the segmentations shown in Figures 5.7a, 5.8a, 5.8b, and 5.8c.

5.3 Parametrization

The algorithms presented in the previous chapter have a number of free parameters that have an effect on the system performance. In this section we describe a series of experiments that were carried out to test hypothesis about the parameters and determine reasonable default values.

5.3.1 Cannot-Link Threshold

The cannot-link constraints introduced in Section 4.1.2.5 play a crucial role in the partitioning algorithm as they influence both selection of a cut and decision when to stop splitting. Concave edges become CL edges when the angle between the normals of their corresponding supervoxels exceed a certain threshold. Small threshold will result in too many CL edges and, consequently, we expect a large degree of over-segmentation. Conversely, a large threshold will result in a small number of CL edges and under-segmentation. We used the labeled scenes in the KinFu dataset to find the angle threshold that yields the best trade-off.

Figure 5.9 presents the relation between cannot-link threshold value and the performance of the algorithm in terms of the evaluation metrics introduced in Section 5.2. We observe that small threshold leads to an extremely high degree of over-segmentation (up to 50 extra segments; the plot is cropped at 25 for readability). The degree of over-segmentation drops as the threshold increases, whereas the under-segmentation score raises. Furthermore, in terms of the weighted and unweighted covering we observe a flat peak around 45°. Consequently, this value was chosen as the default, because it gives the best overall performance according to all metrics.

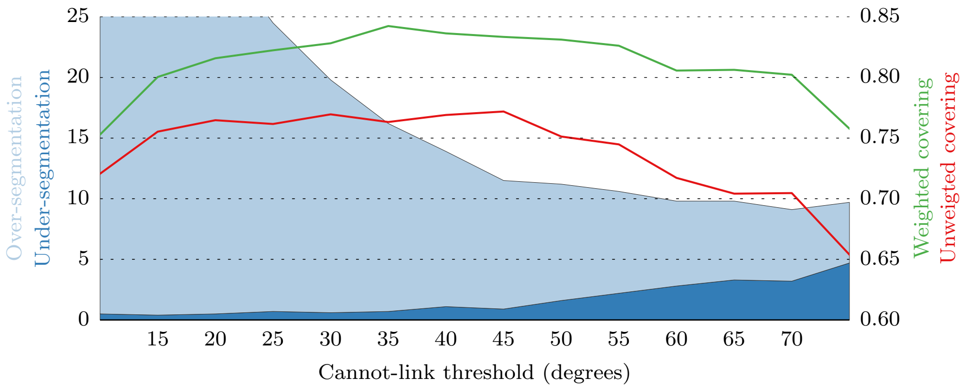


FIGURE 5.9: Performance of SSC with different settings of the cannot-link threshold. The four evaluation metrics are visualized simultaneously using twin axes. The left axis and the regions filled with shades of blue demonstrate the under- and over-segmentation scores. The right axis and the red and green charts display the weighted and unweighted covering. In order to improve the readability, in this plot the over-segmentation values above 25 are not visualized.

5.3.2 Number of Eigenvectors to Compute

From the standpoint of the theoretical derivation in Section 2.3.3, each eigenvector is an approximation of an indicator vector of some cluster. Therefore, in order to identify all objects in a point cloud, it is necessary to compute at least as many eigenvectors as there are objects. In unsupervised segmentation this number is not known a priori. Consequently, a large enough default value that can accommodate both simple and complex scenes has to be selected.

In practice, however, we observe that a single eigenvector often ”indicates” multiple objects (cf. Figure 4.6). This suggests that it may be sufficient to compute a small fixed number of eigenvectors that is less than the actual number of objects. A set of experiments have been performed to test this hypothesis and select a reasonable number.

Correlation with the Scene Complexity

Two groups of scenes were selected. The first group contained 28 simple scenes from the OSD dataset with 4 to 6 ground truth segments. The second group contained 3 complex scenes from the KinFu dataset with 30 to 35 ground truth segments. We ran the SSC algorithm in a supervised mode. This mode has a modified FindBestCut procedure; it allows the user to intervene and manually select an eigenvector that is to be used to cut the graph. We always selected the smallest eigenvector that induced a desirable cut (i.e. a cut that agrees with the ground truth), but not necessarily with the best utility score. In this manner we arrived at segmentations that match ground truth, using the least possible number of eigenvectors.

For both groups the total number of 87 cuts were performed, and for each cut the number of eigenvector used was recorded. The histogram in Figure 5.10 presents the distributions for the first (red) and the second (blue) groups. We observe that in all cases it was not necessary to compute as many eigenvectors as there were objects. Furthermore, the eigenvectors with small numbers were used more often. We conclude that the number of eigenvectors that are required to successfully segment a scene correlates with the scene complexity, yet is much smaller than the actual number of objects present.

Selection of the Default Number of Eigenvectors to Compute

Two types of experiments were performed to determine a reasonable default number of eigenvectors to compute. In both cases the labeled scenes of the KinFu dataset were segmented and evaluation metrics were computed. In the first experiment we used a variant of SSC where a single eigendecomposition of the full graph is performed. The subgraphs created in the process of partitioning are pushed to the result array immediately without an additional round of spectral clustering. This means that during the first (and only) eigendecomposition a sufficiently large number of eigenvectors have

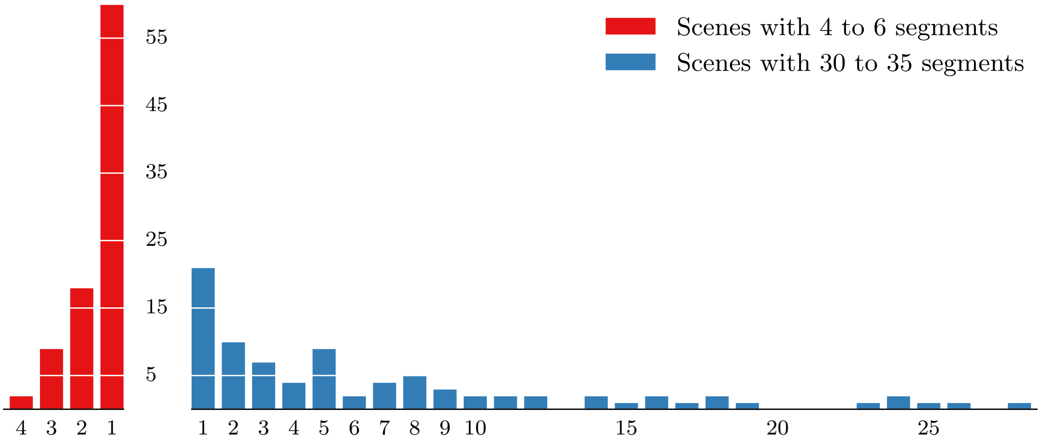


FIGURE 5.10: Distribution of cuts along eigenvectors. Left part (red) is for the simple scenes from the OSD dataset with 4 to 6 segments. Right part (blue) is for the complex scenes from the KinFu dataset with 30 to 35 segments.

to be computed. Figure 5.11 demonstrates the relation between the performance and the number of computed eigenvectors. We observe that increasing this number leads to a steady improvement of the both covering scores and under- and over-segmentation. The most significant improvement happens in the first five eigenvectors; this reinforces the conclusions drawn in the previous experiments. In the second experiment the standard variant of SSC with recursive eigendecompostions was used. Here only the first few eigenvectors give a noticeable effect. We chose the number of 10 eigenvectors as a reasonable trade-off between performance and computational burden.

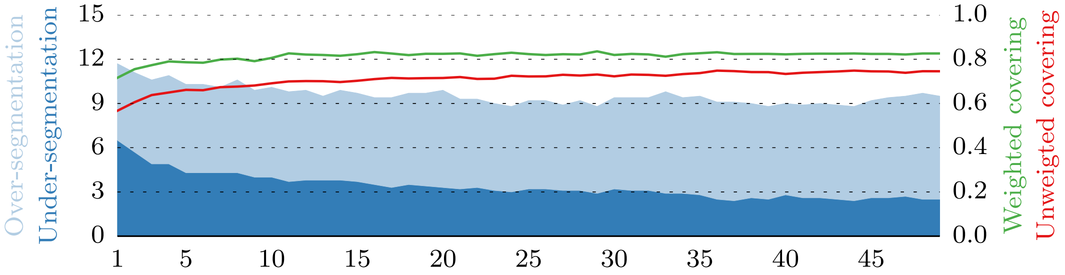


FIGURE 5.11: Performance of SSC with single eigendecomposition of full graph with different settings for the number of eigenvectors to compute. The four evaluation metrics are visualized simultaneously using twin axes. The left axis and the regions filled with shades of blue demonstrate the under- and over-segmentation scores. The right axis and the red and green charts display the weighted and unweighted covering.

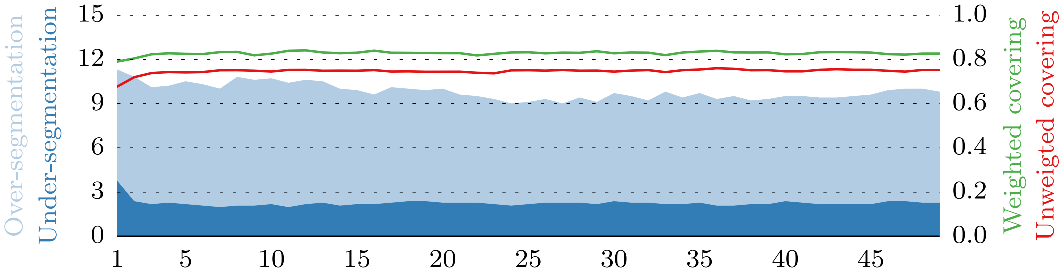


FIGURE 5.12: Performance of SSC with recursive eigendecomposition of subgraphs with different settings for the number of eigenvectors to compute. The four evaluation metrics are visualized simultaneously using twin axes. The left axis and the regions filled with shades of blue demonstrate the under- and over-segmentation scores. The right axis and the red and green charts display the weighted and unweighted covering.

5.3.3 Cannot-Link Edge Weight in Constrained Clustering

In the constrained spectral clustering approach, described in Section 4.1.4, the cannot-link edges are assigned a fixed negative weight ω_{CL}. This makes sure that such edges have an influence on the objective function. More specifically, in signed ratio cut (Equations 4.12 and 4.13) there are special terms that count the total weight of negative edges within clusters. The larger ω_{CL}, the more influence negative edges have on the objective function. Conversely, when ω_{CL} is zero, signed ratio cut is reduced to ratio cut.

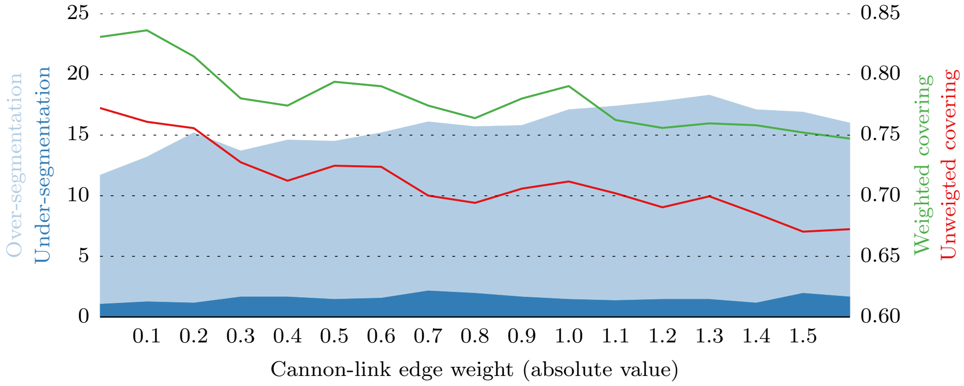


FIGURE 5.13: Performance of constrained SSC with different settings of the cannot-link edge weight. The four evaluation metrics are visualized simultaneously using twin axes. The left axis and the regions filled with shades of blue demonstrate the under- and over-segmentation scores. The right axis and the red and green charts display the weighted and unweighted covering.

A set of experiments were performed to find out a good value for ω_{CL}. The labeled scenes from the KinFu dataset were segmented using constrained SSC with different settings for cannot-link edge weight. The results are visualized in Figure 5.13. We observe that the performance steadily degrades both in terms of under- and over- segmentation and weighted and unweighted covering as the absolute value of cannot-link edge weight is increased.

5.4 Results

The performance of the SSC algorithm on the "testing" part of the OSD dataset is summarized in Figure 5.14. In the first 55 scenes (low and medium complexity) the number of ground truth segments is between 3 and 10, and in the last 11 scenes (high complexity) it is significantly larger, 20 to 30. The weighted covering is about 0.98 on average, with only a single score dropping below 0.9. Stein et al. [SWS+14] reported mean weighted covering 0.87 on this dataset, however we must admit that it is not directly comparable with our result because we use different ground truth annotations.

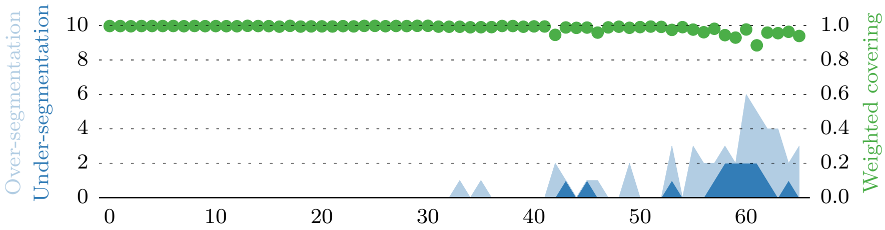
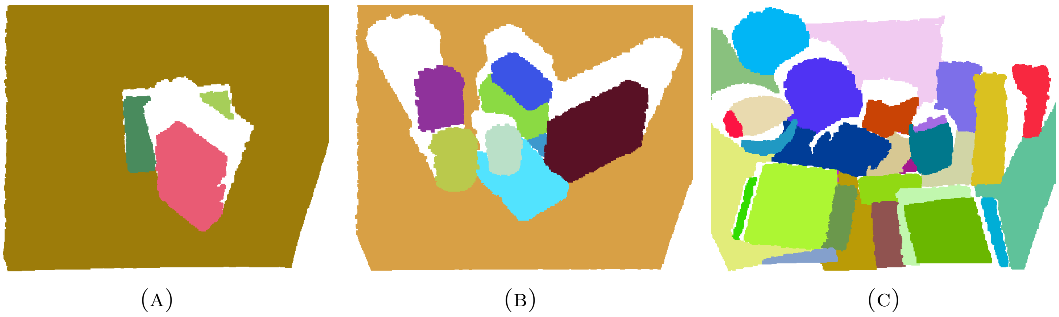


Figure 5.14: Performance of SSC on the 66 test scenes from the OSD dataset. Green scatterplot shows weighted segmentation covering. Dark blue chart shows the number of unmatched ground truth segments (under-segmentation). Light blue chart shows the number of unassigned computed segments (over-segmentation).

We observe that among the first—simple—scenes there were only a few cases of minor under- or over-segmentation. In the complex scenes there is consistently a certain degree of over-segmentation and occasional under-segmentation. Figure 5.15 presents examples of segmentations of the OSD scenes from Figure 5.1 for visual evaluation.

The performance of constrained SSC is very similar and is summarized in Figure 5.16. We observe that like the standard SSC it has no problems segmenting relatively simple scenes. In more complex scenes in shows slightly better performance in terms of the over-segmentation scores.

The SSC algorithm performs reasonably well on much more complex scenes from the KinFu dataset. Using the annotated subset of scenes we obtained 0.83 mean weighted



(A) (B) (C)

FIGURE 5.15: Segmentation of the sample OSD scenes from Figure 5.1.

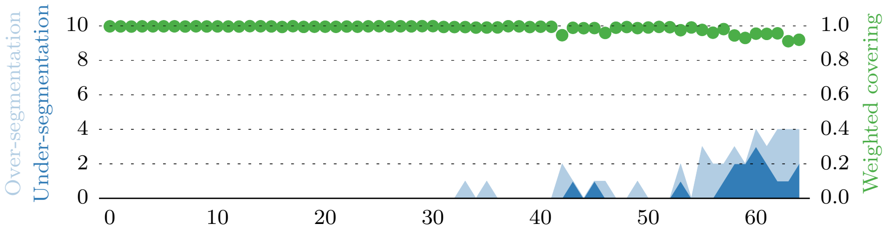
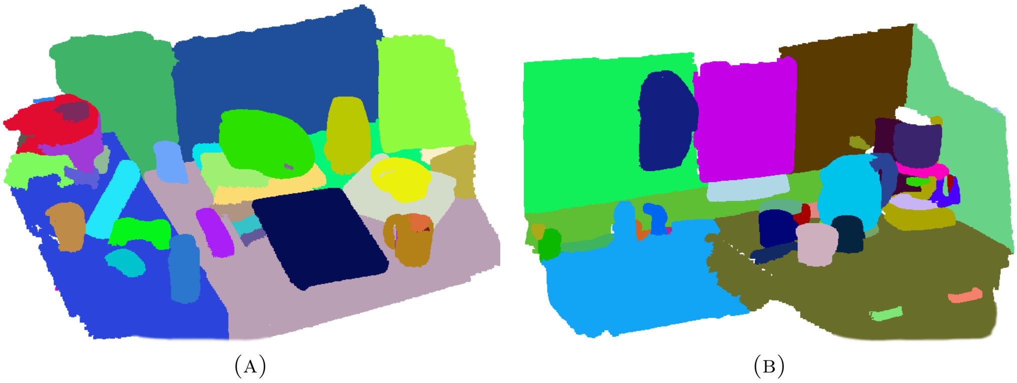


FIGURE 5.16: Performance of constrained SSC on the 66 test scenes from the OSD dataset. Green scatterplot shows weighted segmentation covering. Dark blue chart shows the number of unmatched ground truth segments (under-segmentation). Light blue chart shows the number of unassigned computed segments (over-segmentation).



(A) (B)

FIGURE 5.17: Segmentation of the sample KinFu scenes from Figures 5.3b and 5.3c.

covering and 2/9 mean under/over segmentation. Figure 5.17 demonstrate segmentations of two sample scenes from Figures 5.3b and 5.3c.

From the results we observe that the algorithm tends to over-segment objects with concave curved surfaces like bowls and mugs. Also segmentations are not necessarily meaningful for complex objects consisting of many fine parts.

Complete segmentation of a typical scene from the OSD dataset takes about 0.5 s on a modern Intel Core i7 laptop. Approximately half of the time is spent on supervoxelization, and the rest is spent on solving generalized eigenproblems. The time needed for the other steps, such as graph construction or utility computation, is negligible.

Chapter 6

Conclusions

6.1 Conclusions

The problem of point cloud data processing has come to the foreground in the recent years. Among the fundamental tasks is segmentation of such data based on local low-level geometric cues. Several approaches that address this task were developed and evaluated in the present work.

The task of geometrical segmentation consists of partitioning point clouds into locally smooth convex regions, enclosed by sharp concave boundaries. This formulation is supported by the universal minima rule and is in accordance with human perception. It is model-free and is based only on simple local geometric features. Despite this fact, we argued that segmentation algorithms that make strictly local decisions might produce poor results in certain cases. For example, the region-growing algorithm suffers from weak boundaries between objects. To overcome this problem, it is necessary that a segmentation algorithm has some degree of global awareness. In this thesis we proposed two such algorithms. They make use of the ideas from spectral graph clustering and attempt to produce globally optimal partitions of point cloud data.

In the first approach, Spectral Supervoxel Clustering, a mid-level representation of point clouds with supervoxels is used. Supervoxels and the geometrical relations between them are mapped onto a graph, and the segmentation task is formulated as a problem of optimizing the ratio cut objective function over the set of all possible partitions of the graph. We use spectral relaxation to efficiently approximate the optimal solution of this NP-complete problem. In order to recover the exact solution from its real-valued approximation, we introduced a recursive clustering algorithm driven by a novel utility function. It takes into account the cannot-link edges identified by a heuristic function and allows to perform segmentation in unsupervised manner, without the need to know the number of segments beforehand. A variation of this algorithm, where the cannot-link edges are used to constrain the optimization problem was presented as well.

The second approach, Spectral Voxel Clustering, is similar in spirit, however uses voxels as the data elements that are subject to clustering. The procedure of recovering a graph partition from the output of spectral relaxation is driven by a new utility function, which relies on the concept of edge potential.

Two sources of point cloud data were identified: direct sensing with range cameras and sampling from a complete dense model. Many state-of-the-art segmentation algorithms make use of the image-like structure of the point clouds obtained using the first method, and therefore are not suitable for processing of the second type of point clouds. In contrast, our proposed approaches make no assumptions about the structure of the point cloud data, which makes them more universally applicable.

The algorithms were evaluated on three publicly available datasets. Nearly perfect results were obtained in the scenes where the objects have box-like or cylindrical form and are piled on a table. The performance on unmanipulated real-world scenes with various complex objects is also very promising.

6.2 Future Work

The algorithms developed in this thesis may be improved and extended in a number of ways. Below we summarize the foreseeable directions of future work.

Improved partition recovery. Our proposed approach is based on recursive graph bipartitioning. The FindBestCut function (see Algorithm 2) is designed to find a single splitting vertex along the linear ordering induced by an eigenvector. The vertices that have smaller corresponding elements in eigenvector are put into the first part, the rest is put into the second part. However, in some cases we observed that a better bipartition could be obtained if two splitting points were allowed, i.e.

the vertices in between them constituted one cluster, and the rest were put into the other. An algorithm that admits such bipartitions may produce better results at the cost of being more computationally demanding.

Other ways of incorporating constraints. In Section 4.1.4 we presented one possible way of incorporating cannot-link constraints in the graph partitioning problem via modification of the graph Laplacian. Alternatively, one may explore using constraint propagation [LCP08] or imposing restrictions on the feasible solution space [WQD12]. Additionally, soft constraints may be considered.

Supervoxelization improvement. The supervoxelization algorithm per se is not a subject of this thesis, however the quality of supervoxels play important role in the overall success of the SSC approach. Therefore, improvement of the supervoxelization algorithm may lead to better segmentation results.

Edge validity heuristic. In our work an adaptation of a hand-crafted heuristic proposed by Stein et al. [SWS^{+}14] was used to discard spurious edges between geometrically disjoint surfaces. Even with our modification, the heuristic is too pessimistic and discards large numbers of valid edges. We expect that application of machine learning techniques may output a heuristic with a tighter decision boundary.

Physical stability reasoning. One may go beyond purely geometrical considerations and perform simple reasoning about physical stability of the segmented scene. This may be implemented as a post-processing step or directly incorporated in the partitioning algorithm.

Incorporation of the color information. The RGB-D cameras output both depth and color information. Considering the success of color-based segmentation algorithms, it is not unreasonable to expect that incorporation of color information in the otherwise purely geometrical segmentation algorithm may be beneficial. The results reported in the literature so far are contradictory; in some cases a better performance was observed [BRF12], in other cases the authors did not observe any improvement when adding color information [KMFF13]. This suggests that the problem is worth investigating.

Bibliography

[AMFM11] Pablo Arbeláez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour Detection and Hierarchical Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):898–916, 2011.

[AMT$^+$12] Aitor Aldoma, Zoltan-Csaba Marton, Federico Tombari, Walter Wohlkinger, Christian Potthast, Bernhard Zeisl, Radu Bogdan Rusu, Suat Gedikli, and Markus Vincze. Point Cloud Library: Three-Dimensional Object Recognition and 6 DOF Pose Estimation, 2012.

[BRF12] Liefeng Bo, Xiaofeng Ren, and Dieter Fox. Unsupervised Feature Learning for RGB-D Based Object Recognition. In *Proc. of ISER*, 2012.

[CWD12] Kai-Yang Chiang, Joyce Jiyoung Whang, and Inderjit S. Dhillon. Scalable Clustering of Signed Networks using Balance Normalized Cut. In *Proc. of CIKM*, 2012.

[DHZ$^+$03] Chris Ding, Xiaofeng He, Hongyuan Zha, Ming Gu, and Horst D. Simon. A MinMaxCut Spectral Method for Data Clustering and Graph Partitioning. Technical report, Lawrence Berkeley National Laboratory, 2003.

[FH04] Pedro Felzenszwalb and Daniel Huttenlocher. Efficient Graph-Based Image Segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[GAM13] Saurabh Gupta, Pablo Arbeláez, and Jitendra Malik. Perceptual Organization and Recognition of Indoor Scenes from RGB-D Images. In *Proc. of CVPR*, 2013.

[GJ08] Leo Grady and Marie-Pierre Jolly. Weights and Topology: a Study of the Effects of Graph Construction on 3D Image Segmentation. In *Proc. of MICCAI*, 2008.

[GMLB12] Lucian Cosmin Goron, Zoltan-Csaba Marton, Gheorghe Lazea, and Michael Beetz. Robustly Segmenting Cylindrical and Box-like Objects in Cluttered Scenes using Depth Cameras. In *Proc. of ROBOTIK*, 2012.

[HB12] Dirk Holz and Sven Behnke. Fast Range Image Segmentation and Smoothing using Approximate Surface Reconstruction and Region Growing. In *Proc. of IAS*, 2012.

[HEH10] Derek Hoiem, Alexei A. Efros, and Martial Hebert. Recovering Occlusion Boundaries from an Image. *International Journal of Computer Vision*, 91(3):328–346, 2010.

[HJBJ$^+$96] Adam Hoover, Gillian Jean-Baptiste, Xiaoyi Jiang, Patrick J. Flynn, Horst Bunke, Dmitry B. Goldgof, Kevin Bowyer, David W. Eggert, Andrew Fitzgibbon, and Robert B. Fisher. An Experimental Comparison of Range Image Segmentation Algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(7), 1996.

[HK92] Lars Hagen and Andrew B. Kahng. New Spectral Methods for Ratio Cut Partitioning and Clustering. *Computer-aided design of integrated circuits and systems*, 1992.

[HS97] D Hoffman and M Singh. Salience of Visual Parts. *Cognition*, 63(1), 1997.

[JGSC13] Zhaoyin Jia, Andy Gallagher, Ashutosh Saxena, and Tsuhan Chen. 3D-Based Reasoning with Blocks, Support, and Stability. In *Proc. of CVPR*, 2013.

[KAWB09] K. Klasing, D. Althoff, D. Wollherr, and M. Buss. Comparison of Surface Normal Estimation Methods for Range Sensing Applications. *Proc. of ICRA*, 2009.

[KKBS13] Dov Katz, Moslem Kazemi, Andrew Bagnell, and Anthony Stentz. Clearing a Pile of Unknown Objects using Interactive Perception. In *Proc. of ICRA*, 2013.

[KMFF13] Andrej Karpathy, Stephen Miller, and Li Fei-Fei. Object Discovery in 3D scenes via Shape Analysis. In *Proc. of ICRA*, 2013.

[KSL+10] Jérôme Kunegis, Stephan Schmidt, Andreas Lommatzsch, Jurgen Lerner, Ernesto W De Luca, and Sahin Albayrak. Spectral Analysis of Signed Graphs for Clustering, Prediction and Visualization. In *Proc. of ICDM*, 2010.

[L9̈7] H. Lütkepohl. *Handbook of Matrices.* Wiley, 1997.

[LCP08] Zhengdong Lu and M.A. Carreira-Perpinan. Constrained Spectral Clustering Through Affinity Propagation. In *Proc. of CVPR*, 2008.

[LHMR09] Yu-Kun Lai, Shi-Min Hu, Ralph Robert Martin, and Paul Rosin. Rapid and Effective Segmentation of 3D Models using Random Walks. *Computer Aided Geometric Design*, 26(6), 2009.

[LSK+09] Alex Levinshtein, Adrian Stere, Kiriakos N. Kutulakos, David J. Fleet, Sven J. Dickinson, and Kaleem Siddiqi. TurboPixels: Fast Superpixels Using Geometric Flows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2290–2297, 2009.

[Mei05] Marina Meila. Comparing Clusterings – An Axiomatic View. In *Proc. of ICML*, 2005.

[MM14] Matteo Munaro and Emanuele Menegatti. Fast RGB-D People Tracking for Service Robots. *Autonomous Robots*, 2014.

[MRP+13] Thomas Morwald, Andreas Richtsfeld, Johann Prankl, Michael Zillich, and Markus Vincze. Geometric Data Abstraction using B-splines for Range Image Segmentation. In *ICRA*, 2013.

[MSA12] Ajay K Mishra, Ashish Shrivastava, and Yiannis Aloimonos. Segmenting "Simple" Objects Using RGB-D. In *Proc. of ICRA*, 2012.

[NdC11] Mariá C.V. Nascimento and André C.P.L.F. de Carvalho. Spectral Methods for Graph Clustering – A Survey, 2011.

[NIH+11] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew Davison, Pushmeet Kohli, Jamie Shotton, Steve

Hodges, and Andrew Fitzgibbon. KinectFusion: Real-time Dense Surface Mapping and Tracking, 2011.

[NJW02] Andrew Ng, M. Jordan, and Y. Weiss. On Spectral Clustering: Analysis and an Algorithm. In T G Dietterich, S Becker, and Z Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14 of *Advances in Neural Information Processing Systems*, pages 849–856. MIT Press, MIT Press, 2002.

[PASW13] Jeremie Papon, Alexey Abramov, Markus Schoeler, and Florentin Wörgötter. Voxel Cloud Connectivity Segmentation - Supervoxels for Point Clouds. In *Proc. of CVPR*, 2013.

[PZV12] E. Potapova, Michael Zillich, and Markus Vincze. Attention-driven Segmentation of Cluttered 3D Scenes. In *Proc. of ICPR*, 2012.

[RC11] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *Proc. of ICRA*, 2011.

[Rus09] Radu Bogdan Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments.* PhD thesis, 2009.

[RvdHV06] T Rabbani, F A van den Heuvel, and G Vosselman. Segmentation of Point Clouds using Smoothness Constraint. In *Symponsium on Image Engineering and Vision Metrology*, 2006.

[RZV12] Andreas Richtsfeld, Michael Zillich, and Markus Vincze. Implementation of Gestalt Principles for Object Segmentation. In *21st International Conference on Pattern Recognition ICPR*, 2012.

[Sch07] Satu Elisa Schaeffer. Graph Clustering. *Computer Science Review*, 1(1):27–64, 2007.

[SHKF12] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor Segmentation and Support Inference from RGBD Images. In *Proc. of ECCV*, pages 1–14, 2012.

[SJP11] Jan Smisek, Michal Jancosek, and Tomas Pajdla. 3D with Kinect. In *Proc. of ICCV*, pages 1154–1160, 2011.

[SM00] Jianbo Shi Jianbo Shi and Jitendra Malik. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8), 2000.

[SvLH10] Avinash Sharma, Etienne von Lavante, and Radu Horaud. Learning Shape Segmentation Using Constrained Spectral Clustering and Probabilistic Label Transfer. In *ECCV*, pages 743–756, 2010.

[SW97] Mechthild Stoer and Frank Wagner. A Simple Min-Cut Algorithm, 1997.

[SWS$^+$14] Simon Stein, Florentin Wörgötter, Markus Schoeler, Jeremie Papon, and Tomas Kulvicius. Convexity Based Object Partitioning for Robot Applications. In *Proc. of ICRA*, 2014.

[Tru13] Richard J. Trudeau. *Introduction to Graph Theory.* Courier Dover Publications, 2013.

[TS12] Federico Tombari and Luigi Di Stefano. Hough Voting for 3D Object Recognition under Occlusion and Clutter. *IPSJ Transactions on Computer Vision and Applications*, 4, 2012.

[UH05] R. Unnikrishnan and M. Hebert. Measures of Similarity. *2005 Seventh IEEE Workshops on Applications of Computer Vision (WACV/MOTION'05) - Volume 1*, 1, 2005.

[UHR12] Andre Uckermann, Robert Haschke, and Helge Ritter. Real-Time 3D Segmentation of Cluttered Scenes for Robot Grasping. In *Proc. of Humanoids*, 2012.

[vL07] Ulrike von Luxburg. A Tutorial on Spectral Clustering. *Statistics and Computing*, 17(4), 2007.

[WGB12] David Weikersdorfer, David Gossow, and Michael Beetz. Depth-Adaptive Superpixels. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, number 1, pages 2087–2090, 2012.

[WL93] Zhenyu Wu and Richard Leahy. An Optimal Graph Theoretic Approach to Data Clustering: Theory and its Application to Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101–1113, 1993.

[WMK$^+$12] Thomas Whelan, John McDonald, Michael Kaess, Maurice Fallon, Hordur Johannsson, and John J Leonard. Kintinuous: Spatially Extended Kinect-Fusion. *csnuimie*, 2012.

[WQD12] Xiang Wang, Buyue Qian, and Ian Davidson. On Constrained Spectral Clustering and its Applications. In *Data Mining and Knowledge Discovery*, 2012.