Fakultät für Mathematik, Informatik und Naturwissenschaften
Lehr- und Forschungsgebiet für Informatik VIII (Computer Vision)
Prof. Dr. Bastian Leibe

Bachelor Thesis

Superpixel Segmentation Using Depth Information

David Stutz

September 2014

Primary reviewer: **Prof. Dr. Bastian Leibe**
Secondary reviewer: **Prof. Dr. Thomas Seidl**

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The term superpixel was introduced by Ren and Malik in 2003 [RM03] and is used to describe a group of pixels similar in color or other low-level features. The concept of superpixels is motivated by two important aspects, which have been adapted widely. Firstly, pixels do not represent natural entities but are merely a result of discretization [RM03]. And secondly, the number of pixels prevents many algorithms from being feasible [RM03]. At this point, Ren and Malik introduce superpixels as more natural entities – grouping pixels which perceptually belong together[1].

The task of segmenting an image into superpixels is widely referred to as oversegmentation or superpixel segmentation. While recent algorithms are explicitly designed to generate superpixel segmentations, others were initially intended for classical image segmentation or clustering. Overall, superpixels may have different properties which first of all impose a visual difference. Figure 1.1 presents the running examples of this thesis, each image oversegmented using a different superpixel algorithm.

[1] Of course, "perceptually belong[ing] together", or "perceptually meaningful" as Veksler et al. [VBM10] put it, cannot be used as mathematical definition. Mostly this refers to similarity in color and spatial consistency [VBM10].

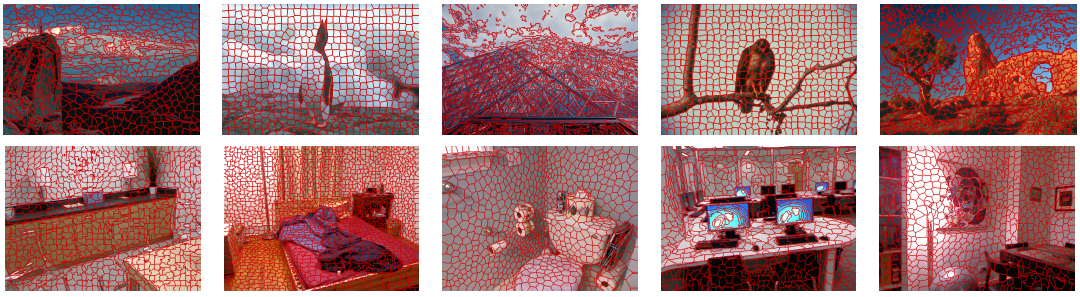

Figure 1.1: The running examples of this thesis will be used to introduce and evaluate several superpixel algorithms. Top: images taken from the Berkeley Segmentation Dataset [AMFM11] oversegmented into roughly 600 superpixels. Bottom: images taken from the NYU Depth Dataset [SHKF12] oversegmented into roughly 840 superpixels.

Since their introduction, superpixels have actively been used for a wide range of applications: for example as pre-processing step for classical segmentation [RM03, RE07], as mid-level cues used for tracking [WLYY11], as pre-processing step for graph-based stereo matching [ZHMB11b], or for semantic segmentation [GAM13]. In addition, numerous superpixel algorithms have been proposed. Although it may depend on the application which of these algorithms is preferable, most authors agree on the following basic requirements for superpixels [LSK+09, ASS+10, LTRC11]:

- Superpixels should respect object boundaries;
- Superpixels should be generated as efficiently as possible;
- Superpixels should not lower the achievable performance[2] of subsequent processing steps.

Additional requirements may include compactness, especially in regions where no object boundaries are present [LSK+09, SFS12], as well as connectivity [LSK+09].

Until 2012, superpixel algorithms were designed to oversegment images with respect to the requirements stated above. However, with rising availability of low-cost RGB-D cameras, algorithms utilizing depth information become increasingly interesting for applications. Furthermore, extending the idea of superpixel segmentation to the temporal domain of videos is receiving increased attention. Approaches using depth information to oversegment videos [WSC13] have recently been proposed. While some superpixel algorithms can naturally be extended to depth or video data, others may be constrained to color images or need to be adapted. One of the latter, although easily generalized to process video data [vdBRB+13], is the approach proposed by Van den Bergh et al. [vdBBR+12], called **SEEDS**[3]. As **SEEDS** represents a superpixel algorithm showing both excellent runtime as well as state-of-the-art performance, the focus of this thesis lies in integrating depth information into **SEEDS**. Our goal is to preserve the low runtime while increasing performance by utilizing depth information.

Another focus of this thesis lies in a comparative assessment of all superpixel algorithms providing source code. We find this necessary as only few research is devoted to comparing and evaluating the rising number of approaches within a consistent framework. To the best of our knowledge these are [SFS12, NP12] and [ASS+12]. We base our experiments on both the Berkeley Segmentation Dataset [AMFM11] as well as the NYU Depth Dataset [SHKF12]. While the Berkeley Segmentation Dataset is commonly used to assess superpixel algorithms, performance on the NYU Depth Dataset may be of particular interest for specific applications. In particular, the scenes captured by the NYU Depth Dataset provide a fresh contrast to the natural images from the

[2] Here and in the following, performance refers to the quality of a superpixel segmentation rather than its runtime.

[3] **SEEDS** stands for "Superpixels Extracted via Energy-Driven Sampling" [vdBBR+12], see chapter 3 for details.

Berkeley Segmentation Dataset. Additionally, it provides depth images allowing us to evaluate additional algorithms requiring depth information. Furthermore, widely used evaluation measures are not defined consistently throughout the literature, as for example the Undersegmentation Error [NP12, ASS+12]. Another commonly used measure is Boundary Recall [MFM04]. However, evaluation based on these two measures only may give a highly biased impression such that an assessment using additional measures is necessary. Finally, details concerning the available implementations, as well as the parameter selection is often omitted within the corresponding publications. Overall, it is difficult to determine both the state-of-the-art as well as approaches suited for specific applications. In our opinion, this is highly unsatisfactory.

1.1 Contributions

The main contributions of this thesis are:

- An implementation of **SEEDS** including a variant providing a compactness parameter [vdBBR+13] and several extensions utilizing depth information.
- A thorough evaluation of several superpixel algorithms on the Berkeley Segmentation Dataset and the NYU Depth Dataset using an extended version of the Berkeley Segmentation Benchmark providing additional measures to evaluate superpixel algorithms.

The extended Berkeley Segmentation Benchmark is made publicly available together with our implementation of **SEEDS**.

1.2 Outline

The thesis is organized as follows: After this short introduction, chapter 2 reviews the literature on superpixel algorithms. Chapter 3 gives a thorough introduction to **SEEDS** as well as another superpixel algorithm used as baseline, called **SLIC**[4] [ASS+10]. Then, chapter 4 discusses several approaches to superpixel segmentation utilizing depth information. This includes a detailed discussion of integrating depth information into **SLIC**. Chapter 5 represents the main part of this thesis and discusses several extensions of **SEEDS** to depth. We present several variants of **SEEDS** using depth information and focus on two of them for evaluation. The datasets and measures used for evaluation are discussed in chapter 6. Chapter 7 presents experimental results of several superpixel algorithms. Parameters are optimized individually on training/validation sets before the different algorithms are compared on the corresponding test sets with regard to their qualitative and quantitative performance as well as runtime. Chapter 8 gives a short conclusion, a brief outlook and presents ideas for future work in this area.

[4] **SLIC** stands for "Simple Linear Iterative Clustering", see chapter 2 for details.

Chapter 2

Related Work

Research related to superpixel algorithms has seen a substantial growth during the last few years (see figure 2.1). Consequently, the literature on this topic is quite extensive. In this chapter, we try to cover the literature **to the best of our knowledge** and give a brief introduction to all superpixel algorithms evaluated in chapter 7. We divide all algorithms presented in this chapter into those applicable to color images, those utilizing depth information and those used to oversegment point clouds. Furthermore, we review the literature devoted to comparing these approaches as well as suitable datasets and benchmarks.

2.1 Superpixel Segmentation

Throughout the literature, superpixel algorithms are often categorized as either graph based methods or gradient ascent methods [ASS+12]. Both expect a color image $I : \underline{W} \times \underline{H} \rightarrow \mathbb{R}^3$, $\underline{W} = \{1,\dots,W\}$, with width W, height H and $N = WH$ pixels to be given and aim to generate a superpixel segmentation

$$S = \{S_1,\dots,S_K\} \quad \text{with} \quad S_k \subseteq \underline{W} \times \underline{H}. \tag{2.1}$$

For this superpixel segmentation to be valid, we demand the superpixels S_i to represent connected components [LSK+09]. Additionally, we enforce the superpixels to be disjoint and resemble the whole image:

$$S_i \cap S_j = \emptyset \quad \text{and} \quad \bigcup_{S_i \in S} S_i = \underline{W} \times \underline{H}. \tag{2.2}$$

Then, we can define the function $s : \underline{W} \times \underline{H} \rightarrow \underline{K}$ assigning a unique superpixel to each pixel $x_n \in \underline{W} \times \underline{H}$.

Graph based approaches construct an undirected, weighted graph $G = (V,E)$ with $V = \underline{N}$ where each vertex n corresponding to pixel x_n is connected to the pixel's four

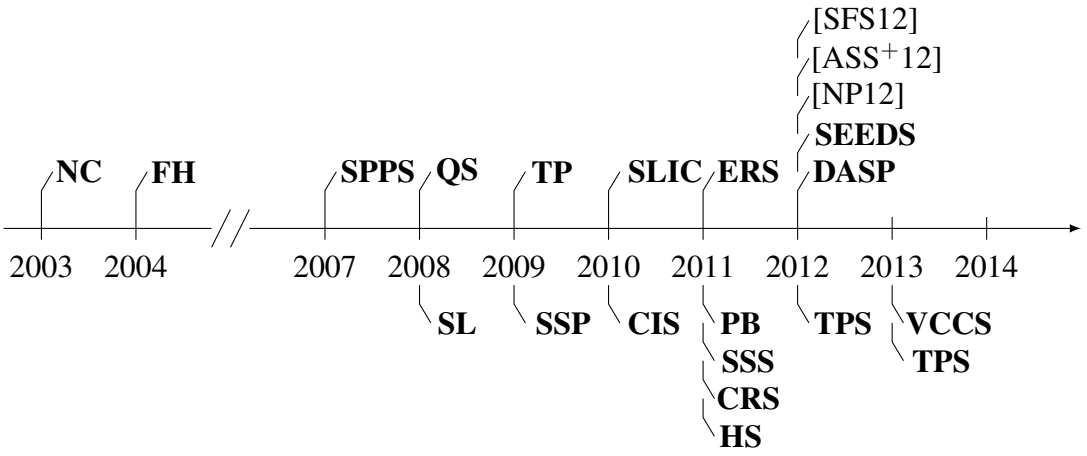

Figure 2.1: Timeline of proposed superpixel, supervoxel and temporal supervoxel algorithms. See sections 2.1, 2.2 and 2.3 for the used abbreviations. In this context, three publications devoted to the comparison of superpixel algorithms are shown, see section 2.4.

direct neighbors[1]. The weights $w_{n,m}$ are interpreted as elements of a weight matrix

$$\boldsymbol{w} = (w_{n,m})_{n,m=1}^{N} \tag{2.3}$$

where $w_{n,m}$ measures the similarity of neighboring pixels x_n and x_m and we set $w_{n,m} = 0$ for $(n,m) \notin E$. Typically, an energy based on the graph G is proposed and optimized using graph cuts or similar techniques.

Gradient ascent methods are commonly built around an energy as well, however, these methods are not based on a graph structure. Instead, they use a variety of different approaches to optimize the proposed energy in order to generate a superpixel segmentation. For example, clustering based approaches define the superpixel center of superpixel S_i to be given by

$$\mu(S_i) = \frac{1}{|S_i|} \sum_{x_n \in S_i} x_n \quad \text{and} \quad I(S_i) = \frac{1}{|S_i|} \sum_{x_n \in S_i} I(x_n). \tag{2.4}$$

Then, pixels are assigned to superpixels based on their similarity to the superpixel centers.

Although the categorization into graph based and gradient ascent methods appears to be very coarse, a finer categorization lies not in the focus of this thesis. In the following we briefly introduce all superpixel algorithms evaluated in chapter 7. The algorithms are sorted according to the year of their publication. Figure 2.1 shows a timeline of all superpixel algorithms.

NC – Superpixels from Normalized Cuts [RM03]. The Normalized Cuts algorithm was originally proposed in 2000 by Shi and Malik [SM00] for the task of classical image segmentation. In 2003, Ren and Malik [RM03] used Normalized Cuts as integral

[1] This resembles a 4-connected graph. However, some approaches and applications use a 8-connected graph, instead, where each pixel x_n is connected to its diagonal neighbors as well.

Algorithm 2.1 The superpixel algorithm **FH** proposed in [FH04].

Input: undirected, weighted graph $G = (V, E)$
Output: superpixel segmentation S
1. sort E by increasing edge weight
2. let S be the superpixel segmentation where each pixel is its own superpixel
3. **for** $k = 1$ **to** $|E|$
4. let (n, m) be the k^{th} edge
5. **if** $s(x_n) \neq s(x_m)$
6. **then if** $w_{n,m}$ is sufficiently small compared to $MInt(S_{s(x_n)}, S_{s(x_m)})$
7. **then** merge superpixels $S_{s(x_n)}$ and $S_{s(x_m)}$
8. **return** S

component for the very first superpixel algorithm. As graph based approach, the algorithm successively adds graph cuts in order to oversegment the image where each graph cut minimizes a global criterion called Normalized Cut:

$$NCut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)} \tag{2.5}$$

where $A, B \subseteq V$ are the two segments resulting from the cut and

$$cut(A,B) = \sum_{n \in A} \sum_{m \in B} w_{n,m}, \tag{2.6}$$

$$assoc(A,V) = \sum_{n \in A} \sum_{m \in V} w_{n,m}. \tag{2.7}$$

As shown in [SM00], this criterion can be minimized by discretizing the second smallest eigenvalue corresponding to the generalized eigenvalue problem

$$(\boldsymbol{d} - \boldsymbol{w})y = \lambda \boldsymbol{d} y \tag{2.8}$$

where $\boldsymbol{d}$ is a diagonal matrix with $d_{n,n} = \sum_{m \in V} w_{n,m}$. To obtain superpixels, this criterion can be applied recursively or additional eigenvectors (third smallest eigenvector etc.) can be used [RM03].

FH – Felzenswalb & Huttenlocher [FH04]. Proposed in 2004 by Felzenswalb and Huttenlocher, this is another graph based approach summarized in algorithm 1. For $A \subseteq V$, we define $MST(A)$ to be the set of edges corresponding to the minimum spanning tree of A within the graph $G = (V, E)$ and define

$$Int(A) = \max_{(n,m) \in MST(A)} \{w_{n,m}\}, \tag{2.9}$$

$$MInt(A,B) = \min\{Int(A) + \frac{\tau}{|A|}, Int(B) + \frac{\tau}{|B|}\} \tag{2.10}$$

Algorithm 2.2 The superpixel algorithm **QS** proposed in [VS08].

Input: color image I
Output: superpixel segmentation S
1. **for** $n = 1$ **to** N
2. initialize $t(x_n) = \mathbf{0}$
3. **for** $n = 1$ **to** N
4. // $N_R(x_n)$ is the set of all pixels in the local neighborhood of size $R \times R$ around pixel x_n:
5. calculate $p(x_n) = \sum_{x_m \in N_R(x_n)} \exp\left(\frac{-d(x_n,x_m)^2}{(2/3)R}\right)$
6. **for** $n = 1$ **to** N
7. set $t(x_n) = \arg\max_{x_m \in N_R(x_n): p(x_m) > p(x_n)} \{p(x_m)\}$
8. // t maps each pixel to its neighbor x_m with highest $p(x_m)$ if $p(x_m) > p(x_n)$;
9. // t can be interpreted as forest, where all pixels x_n with $t(x_n) = \mathbf{0}$ are roots.
10. derive superpixel segmentation S from t
11. **return** S

where τ is a threshold parameter and $MInt(A,B)$ is called the minimum internal difference of components A and B. Starting from an initial superpixel segmentation where each pixel forms its own superpixel, the algorithm processes all edges sorted by increasing edge weight. Whenever an edge connects two different superpixels, these are merged if the edge weight is small compared to the minimum internal difference.

QS – QuickShift [VS08]. Proposed four years later in 2008, **QS** can be categorized as gradient ascent method and is a mode-seeking algorithm. In general, a mode-seeking algorithm starts from a Parzen density estimate $p(x_n)$ for all pixels and each pixel is assigned to a mode by following the density $p(x_n)$ upwards, that is in the direction of the gradient [VS08]. In particular, **QS** pre-computes $p(x_n)$ for all pixels using a Gaussian kernel. In practice, the distance $d(x_n,x_m)$ used within the Gaussian kernel consists of a color term and a spatial term:

$$d(x_n,x_m) = \alpha \|I(x_n) - I(x_m)\|_2 + \|x_n - x_m\|_2 \tag{2.11}$$

where α weights the influence of the color term. Subsequently, each pixel x_n gets assigned to the pixel $x_m \in N_R(x_n) = \{x_m : \|x_n - x_m\|_\infty \leq \lfloor R/2 \rfloor\}$ such that $p(x_m) > p(x_n)$ or is left unassigned. These assignments correspond to the found modes, representing the final superpixels. This procedure is summarized in algorithm 2.

TP – Turbopixels [LSK+09]. Proposed in 2009, Turbopixels is an algorithm inspired by active contours. After placing initial superpixel centers on a regular grid with step size R, each superpixel is grown based on an evolving contour. The contour is implemented as level set[2] of a smooth function $\Psi : \mathbb{R}^2 \times [0,\tau) \to \mathbb{R}^2$. Evolution is

[2]That is, the superpixel boundaries are interpreted as the zero-level set of the function Ψ (all points where Ψ equals zero). See [OF03] for details on level set methods.

Algorithm 2.3 The superpixel algorithm **TP** proposed in [LSK+09].

Input: color image I, number of superpixels K
Output: superpixel segmentation S
1. // R can be derived from the image size $W \times H$ and the number of superpixels K:
2. place superpixel centers on a regular grid with step size R
3. // All pixels x_n where $\Psi(x_n) > 0$ are unassigned.
4. initialize $\Psi^{(0)}$
5. **repeat**
6. compute v_I and v_B
7. evolve the contour by computing $\Psi^{(T+1)}$
8. update assigned pixels
9. $T \leftarrow T+1$
10. **until** all pixels are assigned
11. // A superpixel segmentation can be derived from the contour given by Ψ:
12. derive S from Ψ
13. **return** S

formally defined by the equation

$$\Psi_t = -v\|\nabla\Psi\|_2 \tag{2.12}$$

where $\nabla\Psi$ denotes the gradient of Ψ, Ψ_t is the time derivative of Ψ and the speed v describes the future evolution of the contour. In practice, Ψ will be the signed euclidean distance of each pixel to the contour [LSK+09] and the evolution is carried out using a first order discretization such that the contour in iteration $(T+1)$ is given by

$$\Psi^{(T+1)} = \Psi^{(T)} - v_I v_B \|\nabla\Psi^{(T)}\|\Delta t \tag{2.13}$$

where v_I depends on the image content, while v_B ensures that superpixels do not overlap, see [LSK+09] for details. In each iteration $(T+1)$, pixels x_n for which $\Psi^{(T)}(x_n) > 0$ are considered unassigned. Iteratively, the superpixels are grown by computing the speeds v_I and v_B and evolving the contour according to equation (2.13) until all pixels are assigned. A summary is given in algorithm 3.

SLIC – Simple Linear Iterative Clustering [ASS+10]. Proposed in 2010, this algorithm is often used as baseline, for example in [FCT+14, PASW13, vdBBR+12] and [CMM13], and is particular interesting because of its simplicity. **SLIC** implements a local K-means clustering to generate a superpixel segmentation with K superpixels. Therefore, **SLIC** can be categorized as gradient ascent method. The algorithm is discussed in detail in chapter 3.

CIS – Constant Intensity Superpixels [VBM10]. Proposed by Veksler et al. in 2010, this is a graph based method defined on grayscale images only. However, the below description is easily extended to color images. Initially, the image is covered by

Algorithm 2.4 The greedy algorithm used to maximize the energy $E(\hat{G})$ to obtain Entropy Rate Superpixels [LTRC11].

Input: undirected, weighted graph $G = (V, E)$
Output: superpixel segmentation S
1. initialize $M = \emptyset$
2. **for each** edge $(n,m) \in E$
3. // Let $\hat{G}$ denote the graph $\hat{G} = (V, M \cup \{(n,m)\})$:
4. choose edge $(n,m) \in E$ yielding the largest gain in the energy $E(\hat{G})$
5. **if** $\hat{G}$ has no cycles and $\hat{G}$ contains less or equal than K connected components
6. **then** $M \leftarrow M \cup \{(n,m)\}$
7. // The superpixel segmentation is given by the connected components in $\hat{G}$:
8. derive superpixel segmentation S from $\hat{G}$
9. **return** S

overlapping squares such that each pixel is covered by several squares. Each square represents a superpixel and each pixel can get assigned to one of these squares. An energy of the form

$$E(S) = \sum_{n \in V} \sum_{m \in V} w_{n,m} \psi_{n,m}(s(x_n), s(x_m)) + \sum_{n \in V} \theta_n(s(x_n)) \tag{2.14}$$

is minimized using α-expansion [BVZ01]. In particular, θ_n defines a data term which is simplified[3] given by

$$\theta_n(i) = \begin{cases} |I(x_n) - I(S_i)| & \text{if } x_n \text{ is assigned to superpixel } S_i \\ 0 & \text{else} \end{cases}. \tag{2.15}$$

Further, $\psi_{n,m}$ is a Potts Model given by

$$\psi_{n,m}(i,j) = \begin{cases} 1 & \text{if } i \neq j \\ 0 & \text{else} \end{cases} \tag{2.16}$$

The weights $w_{n,m}$ for neighboring pixels x_n and x_m are set to

$$w_{n,m} = \lambda \exp\left(-\frac{(I(x_n) - I(x_m))^2}{2\|x_n - x_m\|_2 \sigma^2} \right) \tag{2.17}$$

where λ is a controllable parameter and σ is set to the overall variance within the image.

[3] Actually, instead of using $I(S_i)$ within the data term, the color of the center pixel of the initial square S_i is used and fixed. However, this requires discussing an additional term ensuring that this center pixel is assigned to superpixel S_i as well, see [VBM10].

$s(x_n)$	$s(x_m)$	(2.23)	(2.24)	(2.25)
0	0	0	$w_{n,m}$	0
0	1	$w_{n,m}$	$w_{n,m}$	$w_{n,m}$
1	0	$w_{n,m}$	$w_{n,m}$	$w_{n,m}$
1	1	0	0	$w_{n,m}$

Table 2.1: The data term $\psi_{n,m}(s(x_n), s(x_m))$ of the energy in equation (2.22) depends on the labels of pixels x_n and x_m as well as on the corresponding strips, see equations (2.23), (2.24) and (2.25).

ERS – Entropy Rate Superpixels [LTRC11]. This algorithm is another graph based method and was proposed in 2011. An objective function based on the entropy rate of a random walk on the graph $\hat{G} = (V, M)$ with $M \subseteq E$ is proposed:

$$E(\hat{G}) = H(\hat{G}) + \lambda B(\hat{G}). \tag{2.18}$$

This energy is maximized subject to the constraint that the number of connected components in $\hat{G}$ is lower or equal to the desired number of superpixels K. Here, $H(\hat{G})$ refers to the entropy rate of the random walk, while $B(\hat{G})$ defines a balancing term. We use $w_n = \sum_{m \in V} w_{n,m}$ to define

$$H(\hat{G}) = -\sum_{n \in V} \frac{w_n}{\sum_{m \in V} w_m} \sum_{m \in V} p_{n,m} \log(p_{n,m}) \tag{2.19}$$

where the probabilities $p_{n,m}$ represent the transition probabilities of the random walk and are calculated as

$$p_{n,m} = \begin{cases} \frac{w_{n,m}}{w_n} & \text{if } n \neq m \text{ and } (n,m) \in M \\ 0 & \text{if } n \neq m \text{ and } (n,m) \notin M \\ 1 - \frac{\sum_{m=1}^{N} w_{n,m}}{w_n} & \text{if } n = m \end{cases}. \tag{2.20}$$

In practice the weights $w_{n,m}$ are defined using the L_1 color distance between pixels x_n and x_m and then using a Gaussian kernel to obtain similarities [LTRC11]. The balancing term favors superpixels of approximately the same size:

$$B(\hat{G}) = -\sum_{i=1}^{K} \frac{|S_i|}{N} \log\left(\frac{|S_i|}{N}\right). \tag{2.21}$$

Starting from an initial superpixel segmentation where each pixel forms its own superpixel, the algorithm greedily adds edges to merge superpixels, see algorithm 4.

PB – Superpixels via Pseudo Boolean Optimization [ZHMB11a]. Proposed in 2011, this algorithm is comparable to **CIS** and is therefore categorized as graph based. First, the image is covered by overlapping vertical and horizontal strips such that each pixel is covered by exactly two vertical and two horizontal strips. This way, considering only the horizontal strips, each pixel is either labeled 0 or 1. Then, an energy

similar to (2.14) is used:

$$E(S) = \sum_{n \in V} \sum_{m \in V} \psi_{n,m}(s(x_n), s(x_m)) + \sum_{n \in V} \theta_n(s(x_n)) \tag{2.22}$$

where the data term θ_n is set to zero. The smoothing term is based on the following considerations. Numbering the horizontal strips such that $H_i \subseteq V$ is covered halfway by $H_{i+1} \subseteq V$ and considering neighboring pixels x_n and x_m such that x_n lies above or at the same horizontal line as x_m, three cases are possible:

$$x_n \in H_i \cap H_{i+1} \quad \text{and} \quad x_m \in H_i \cap H_{i+1}, \tag{2.23}$$

$$x_n \in H_i \cap H_{i+1} \quad \text{and} \quad x_m \in H_{i+1} \cap H_{i+2}, \tag{2.24}$$

$$x_n \in H_{i+1} \cap H_{i+2} \quad \text{and} \quad x_m \in H_{i+2} \cap H_{i+3}. \tag{2.25}$$

Including two possible labels per pixel, there are twelve cases to consider. Table 2.1 shows the resulting smoothing term for each case where the weights $w_{n,m}$ are calculated using a Gaussian kernel and the L_1 color distance, similar to **ERS**. The energy is optimized using max-flow. The final superpixel segmentation can be derived from the vertical and horizontal labels.

CRS – Contour Relaxed Superpixels [MCG11, CMM13][4]. This approach, proposed in 2013, represents a statistical approach to the task of superpixel segmentation where the image I is assumed to be a result of multiple stochastic processes. Actually, the value $I_c(x_n)$ of pixel x_n in channel c is thought to be an outcome of a stochastic process described by the parameter set $\theta_{s(x_n),c}$. Then, using $\boldsymbol{\theta} = \{\theta_{1,1}, \ldots, \theta_{K,3}\}$, the superpixel segmentation S maximizing the probability $p(S, \boldsymbol{\theta} | I)$ is searched for. Applying Bayes' theorem and omitting the normalization, the energy is given by

$$E(S) = p(I|S, \boldsymbol{\theta}) p(S, \boldsymbol{\theta}) \tag{2.26}$$

where $p(S, \boldsymbol{\theta})$ is set to $p(S, \boldsymbol{\theta}) = \kappa p(S)$ with κ constant as the parameters $\boldsymbol{\theta}$ are considered deterministic parameters. Then, an EM-style optimization scheme is applied. The parameters $\boldsymbol{\theta}$ are estimated using maximum likelihood considering the superpixel segmentation S to be constant, followed by optimizing S while holding $\boldsymbol{\theta}$ constant. In practice, $p(S)$ is modeled using a Gibbs Random Field (e.g. see [Bis06]) where for each pixel x_n a clique of size two for each direct or diagonal neighbor is considered. Then, $p(S)$ can be written as

$$p(S) = \kappa' \exp(-N_e C_e - N_v C_v) \tag{2.27}$$

where N_e is the number of direct neighbors of x_n having a different label than $s(x_n)$, N_v is the number of diagonal neighbors of x_n having a different label than $s(x_n)$ and C_e

[4]We first found this approach as presented in [CMM13]. However, the approach was already introduced in 2011 in [MCG11]. As both publications offer a thorough description of the superpixel algorithm, we will use [CMM13] as primary reference.

Algorithm 2.5 The algorithm to maximize the energy given in equation (2.26) to obtain Contour Relaxed Superpixels [CMM13].

Input: color image I, number of superpixels K
Output: superpixel segmentation S
1. // The step size R can be derived from the image size $W \times H$ and K:
2. initialize S as regular grid with step size R
3. initialize $\boldsymbol{\theta}$ using sufficient statistics (e.g. Gaussian)
4. **for** $t = 1$ **to** T
5. // Originally, the image is traversed multiple times using different directions to avoid a directional bias [CMM13]:
6. **for** $n = 1$ **to** N
7. **if** x_n is a boundary pixel
8. // This can be evaluated by taking $\boldsymbol{\theta}$ as constant; Conrad et al. suggest to minimize the negative logarithm of (2.29) instead:
9. **then** assign x_n to the label maximizing (2.29)
10. **return** S

and C_v are the corresponding costs. Further, the probability $p(I|S,\boldsymbol{\theta}) = p(I|S)$ can be expressed as

$$p(I|S) = \prod_{S_i \in S} \prod_{x_n \in S_i} \prod_{c=1}^{3} p(I_c(x_n)|\theta_{i,c}). \tag{2.28}$$

when assuming the stochastic processes to be statistically independent. The optimization is carried out as follows. In each iteration, all boundary pixels x_n are considered to change their label. Assuming $\boldsymbol{\theta}$ to be constant, pixel x_n is assigned to the label maximizing

$$E(S) = \kappa\kappa' \exp(-N_e C_e - N_v C_v) \prod_{S_i} \prod_{x_m \in S_i} \prod_{c=1}^{3} p(I_c(x_m)|\theta_{i,c}) \tag{2.29}$$

where the first product runs over all superpixels S_i to which pixel x_n may be assigned. This is summarized in algorithm 5 and it becomes apparent that **CRS** can be categorized as gradient ascent method. The implementation of **CRS** models the stochastic processes as being Gaussian and uses

$$|S_i|, \quad I_c(S_i) \quad \text{and} \quad \sum_{x_m \in S_i} I_c(x_m)^2 \tag{2.30}$$

as sufficient statistics. The compactness of the obtained superpixel segmentation can be controlled by additionally using the pixel coordinates as channels. A compactness parameter β controls the weighting of the costs based on these additional channels.

SEEDS – Superpixels Extracted via Energy-Driven Sampling [vdBBR+12]. Proposed in 2013, this algorithm can be categorized as gradient ascent method and is the main focus of this thesis. The algorithm is discussed in detail in chapter 3 and approaches to generalize the algorithm to use depth information are reviewed in chapter 5.

TPS – Topology Preserved Superpixels [TFC12]. **TPS** aims to generate a superpixel segmentation representing a regular grid topology, that is the superpixels can be arranged in an array where each superpixel has a consistent, ordered position [MPW+08]. Given an edge map

$$p : \underline{W} \times \underline{H} \to [0,1], x_n \mapsto p(x_n) \tag{2.31}$$

defining the probability of an edge being present at pixel x_n, the algorithm proceeds in three steps. Firstly, a set of pixels are chosen as initial grid positions. This is done on a regular grid with horizontal step size R_h and vertical step size R_v given as

$$R_v \approx \sqrt{\frac{KH}{W}} \quad \text{and} \quad R_h \approx \frac{K}{R_v}. \tag{2.32}$$

Let $\hat{\mu}_1, \ldots, \hat{\mu}_{K'}$ denote these positions (to obtain K superpixels, K' of these grid positions are needed). Secondly, the positions are moved towards maximum edge positions by choosing

$$\hat{\mu}_i = \arg \max_{x_n \in N_R(\hat{\mu}_i)} \{p(x_n) \exp\left(\frac{\|x_n - \hat{\mu}_i\|_2}{2\sigma^2}\right)\} \tag{2.33}$$

where $N_R(\hat{\mu}_i)$ defines a local search region around the position $\hat{\mu}_i$. Finally, these grid positions define an undirected graph based on their relative positions. Neighboring positions are connected by the shortest path calculated on the undirected, weighted graph with weights

$$w_{n,m} = \frac{1}{p(x_n) + p(x_m)} \tag{2.34}$$

for neighboring pixels x_n and x_m. The shortest path is computed using Dijkstra's algorithm (e.g. see [CLRS09]). The superpixels are then given by the enclosed regions.

There are several more superpixel algorithms which are not evaluated in the course of this thesis as no source code is available.These are:

SPPS – Superpixels using Pairwise Pixel Similarities [RE07]. The approach described by Rohkohl and Engel is based on an initial superpixel segmentation of the image into a regular hexagonal grid. Pixels are exchanged between superpixels based on color similarity. **SPPS** is a gradient ascent method where in each iteration, all boundary pixels are considered to change their label.

SL – Superpixel Lattices [MPW+08]. This algorithms is similar to **TPS** in that in attempts to create a regular grid of superpixels. Based on an edge map, the image is

successively partitioned using vertical and horizontal paths. The paths can be found using different approaches: either based on graph cuts, or using dynamic programming. Therefore, the algorithm cannot be categorized as either graph based method or gradient ascent method.

SSP – Superpixels from Strong Paths [DM09]. The work of Drucker and MacCormick aims to provide an efficient method to generate superpixel segmentations. Therefore, it makes use of dynamic programming to compute minimum cost paths on a given edge map or similar feature maps. Although the authors do not give an explicit energy, the algorithm can be categorized as gradient ascent method.

SSS – Structure Sensitive Superpixels [ZWW+11]. Zeng et al. propose a superpixel algorithm based on Lloyd's algorithm and a custom geodesic distance. This way, the superpixels adapt to the underlying image content such that superpixels in highly textured regions tend to be smaller, while superpixels within homogeneous regions are bigger.

HS – Homogeneous Superpixels [PM11]. **HS** by Perbet and Maki resembles a graph based algorithm utilizing Markov Clustering. Based on a Markov graph, that is a undirected, weighted graph where all edges of a given vertex are positive and sum to one, the algorithm alternates an expansion and an inflation step which are carried out on the corresponding weight matrix.

In [ASS+12], Achanta et al. evaluate an additional algorithm able to oversegment images by computing watershed lines, see [VS91] for details. This approach is extended in [AMFM09]: Arbeláez et al. propose an extension called Oriented Watershed Transform to generate an oversegmentation based on an edge map provided by the contour detector described in [AMFM11]. The superpixels within this oversegmentation are iteratively merged based on their similarity in order to form a hierarchy of segmentations. While we do not discuss this approach in detail, it is commonly used as pre-processing step [SHKF12, RBF12] and therefore worth mentioning in the context of superpixel algorithms.

2.2 Superpixel Segmentation using Depth Information

After reviewing the literature on superpixel segmentation of color images, we discuss the literature on superpixel algorithms using depth information. At this point it is necessary to distinguish algorithms creating supervoxels[5] and algorithms creating superpixels by utilizing depth information as additional cue. The former operates directly on voxels, while the latter will be applied to color images and use the additional depth information to improve the generated superpixel segmentation. In this section we review Depth-Adaptive Superpixels [WSC13] as the only algorithm generating superpixel segmentations by utilizing depth information.

[5]If voxel denotes a volume pixel, a supervoxel is a group of voxels. This means, a supervoxel is the generalization of a superpixel to an additional dimension. As discussed in section 2.3, this dimension may either be given by depth information or by the temporal domain of a video.

DASP - Depth-Adaptive Superpixels [WSC13] (see also [Wei14]). Proposed in 2012, this is the first algorithm utilizing depth information to improve the generated superpixel segmentation. Although similar to **SLIC**, this algorithm heavily relies on depth information for choosing the initial superpixel centers and the actual K-means clustering. This approach will be reviewed in detail in chapter 4.

2.3 Supervoxel Segmentation

As mentioned above, supervoxels describe groups of volume pixels, short voxels. Again, it is necessary to differentiate between supervoxels in three dimensional space and supervoxels using the temporal domain of a video as third dimension, which we call temporal supervoxels. As stated in [RJRO13] it is not sufficient to treat the temporal domain of a video as additional spatial dimension such that the above naming seems appropriate. Temporal supervoxel algorithms are discussed in [RJRO13, CWF13] as well as [FCT+14]. As this thesis focusses on superpixel algorithms using depth information, we only review the literature on supervoxel algorithms, which is limited to one approach proposed in [PASW13].

VCCS – Voxel-Cloud Connectivity Segmentation [PASW13]. This is the first algorithm generating supervoxels by oversegmenting a point cloud. **VCCS** is similar to **SLIC** and **DASP**. Based on a 26-adjacency graph constructed from a voxelized point cloud, local K-means clustering is applied to form supervoxels. However, in contrast to **SLIC**, connectivity is ensured by using breadth-first search as basis for K-means clustering. This approach is introduced in detail in chapter 4.

2.4 Comparison and Evaluation

There are only few publications devoted to the comparison of existing superpixel algorithms in a consistent framework: to the best of our knowledge these are [SFS12, ASS+12] and [NP12]. Figure 2.1 shows these works in relation to the superpixel algorithms discussed in the previous sections. As can be seen, these works cannot include some of the recently proposed algorithms like **SEEDS**, **DASP** or **VCCS**. Furthermore, some of the older algorithms are partly not included either. Finally, these publications do not cover superpixel segmentation using depth information or supervoxel segmentation of point clouds. Although most publications proposing new superpixel algorithms include a brief comparison to other state-of-the-art methods, these evaluations cover only parts of the literature. Furthermore, implementations of error measures vary across publications [NP12].

2.5 Datasets and Benchmarks

Superpixel algorithms are usually evaluated utilizing the Berkeley Segmentation Dataset [AMFM11]. Achanta et al. [ASS+12] additionally use the Microsoft Research Cambridge Dataset [SWRC09], short MSRC. For evaluating supervoxel algorithms (or superpixel algorithms using depth information), the NYU Depth Dataset [SHKF12] was used by Papon et al. [PASW13]. Weikersdorfer et al. [WGB12] used a custom dataset[6] comprising eleven RGB-D images to evaluate **DASP**. The Berkeley Segmentation Dataset includes a benchmark, which we refer to as Berkeley Segmentation Benchmark, implementing common measures used to evaluate segmentation algorithms and contour detectors. For evaluating superpixel algorithms commonly used measures include the Undersegmentation Error [LSK+09, LTRC11] and the Achievable Segmentation Accuracy [LTRC11]. Additionally, Schick et al. [SFS12] proposed a compactness measure. Both the Berkeley Segmentation Dataset as well as the NYU Depth Dataset are discussed in detail in chapter 6. In addition, chapter 6 includes a thorough discussion of all used evaluation measures.

[6] Available at `https://github.com/Danvil/dasp`.

Chapter 3

Superpixel Segmentation

We devote this chapter to reviewing two state-of-the-art algorithms to generate superpixel segmentations from color images: **SEEDS** [vdBBR+12] and **SLIC** [ASS+10]. While **SEEDS** represents the main focus of this thesis, **SLIC** is introduced for two reasons. Firstly, it offers a natural extension to depth and secondly, it is commonly used as baseline and especially popular because of its simplicity. In addition, other approaches like **DASP** [WGB12] and **VCCS** [PASW13], discussed in chapter 4, have strong similarities to **SLIC**.

3.1 SEEDS

SEEDS is a gradient ascent method which, in contrast to other methods as **SLIC** or **TP** [LSK+09], starts with an initial superpixel segmentation and iteratively refines it to maximize an energy [vdBBR+12]. As the actual implementation differs from the algorithm proposed by Van den Bergh et al., we first discuss the algorithm in detail before introducing the theoretical background.

3.1.1 Algorithm

The algorithm begins with grouping pixels into blocks of size $w^{(1)} \times h^{(1)}$ (e.g. 2×2, 2×3, 3×2, ...). These blocks are then further arranged in groups of 2×2. This can be applied recursively to form a hierarchy of blocks such that blocks at level l have size $w^{(l)} \times h^{(l)}$ with

$$w^{(l)} = w^{(1)} \cdot 2^{l-1} \quad \text{and} \quad h^{(l)} = h^{(1)} \cdot 2^{l-1}. \tag{3.1}$$

For a total of L levels, the blocks at level L represent the initial superpixels. Therefore, the number of superpixels is calculated as

$$K = \left\lfloor \frac{W}{w^{(L)}} \right\rfloor \cdot \left\lfloor \frac{H}{h^{(L)}} \right\rfloor. \tag{3.2}$$

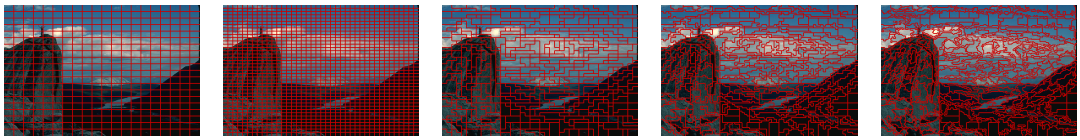

Figure 3.1: Using $w^{(1)} = 3$, $h^{(1)} = 2$ and $L = 4$ which results in 400 superpixels, from left to right: the block segmentation at level $l = 4$; the block segmentation at level $l = 3$; the superpixel segmentation after performing block updates at level $l = 3$; the superpixel segmentation after performing block updates at level $l = 2$ and the superpixel segmentation before performing pixel updates, that it after performing block updates at level $l = 1$.

This means that the minimum block size $w^{(1)} \times h^{(1)}$ and the number of levels L need to be derived from the desired number of superpixels K and the image size $W \times H$.

The initial superpixel segmentation is refined by exchanging blocks or pixels between neighboring superpixels. Blocks are exchanged based on their similarity to the respective superpixel expressed as histogram intersection. Therefore, let $B_i^{(l)} \subseteq \underline{W} \times \underline{H}$ be a block of pixels at level $1 \leq l < L$ and S_j be a superpixel. Further, let $h_{B_i^{(l)}}$ denote the color histogram of block $B_i^{(l)}$ such that $h_{B_i^{(l)}}(q)$, $1 \leq q \leq Q$, is the fraction of pixels within $B_i^{(l)}$ assigned to bin q, with Q being the total number of bins. Then, the similarity of block $B_i^{(l)}$ and superpixel S_j is expressed as the intersection [BOV03] of the corresponding color histograms:

$$\cap(h_{B_i^{(l)}}, h_{S_j}) = \sum_{q=1}^{Q} \min\{h_{B_i^{(l)}}(q), h_{S_j}(q)\}. \tag{3.3}$$

Furthermore, the similarity of a pixel x_n and the superpixel S_j can be quantified by

$$h_{S_j}(h(x_n)) \tag{3.4}$$

where $h(x_n) \in \{1, \ldots, Q\}$ denotes the histogram bin of pixel x_n. Given this framework, algorithm 6 describes the basic idea of **SEEDS**: At each level, including the pixel level ($l = 0$), a new superpixel segmentation is proposed by moving blocks and pixels between neighboring superpixels. The proposed superpixel segmentation is accepted according to the similarity measures discussed above. The process of exchanging blocks between neighboring superpixels is referred to as block updates, while exchanging pixels is called pixel updates. Figure 3.1 illustrates the block hierarchy as well as block updates.

As alternative to pixel updates based on color histograms, Van den Bergh et al. [vdBBR+12] propose so called mean pixel updates. The similarity of pixel x_n and superpixel S_j is then expressed as euclidean distance in color space:

$$d(x_n, S_j) = \|I(x_n) - I(S_j)\|_2. \tag{3.5}$$

Algorithm 3.6 The basic algorithm of **SEEDS**.

Input: color image I, block size $w^{(1)} \times h^{(1)}$, number of levels L, histogram size Q
Output: superpixel segmentation S

1. initialize the block hierarchy and the initial superpixel segmentation S
2. // Initialize histograms for all blocks and superpixels:
3. **for** $l = 1$ **to** L
4. **for each** block $B_i^{(l)}$ // For $l = L$ these are the initial superpixels.
5. initialize histogram $h_{B_i^{(l)}}$
6. // Block updates:
7. **for** $l = L-1$ **to** 1
8. **for each** block $B_i^{(l)}$
9. let S_j be the superpixel $B_i^{(l)}$ belongs to
10. **if** a neighboring block belongs to a different superpixel S_k
11. **then if** $\cap(h_{B_i^{(l)}}, h_{S_k}) > \cap(h_{B_i^{(l)}}, h_{S_j - B_i^{(l)}})$
12. **then** $S_k \leftarrow S_k \cup B_i^{(l)}$ and $S_j \leftarrow S_j - B_i^{(l)}$
13. // Pixel updates:
14. **for** $n = 1$ **to** N
15. let S_j be the superpixel x_n belongs to
16. **if** a neighboring pixel belongs to a different superpixel S_k
17. **then if** $h_{S_k}(h(x_n)) > h_{S_j}(h(x_n))$
18. **then** $S_k \leftarrow S_k \cup \{x_n\}$ and $S_j \leftarrow S_j - \{x_n\}$
19. **return** S

Each pixel x_n gets assigned to the superpixel minimizing the above distance, see algorithm 7. We use **SEEDSmp** to refer to **SEEDS** using mean pixel updates.

As compact and smooth superpixels may be preferable [SFS12], a smoothing term can be integrated into pixel updates. Therefore, we consider a local neighborhood around each pixel pair (x_n, x_m) belonging to different superpixels S_j and S_k, respectively. Let $N_1(x_n, x_m)$ denote the 3×4 or 4×3 neighborhood around the pixel pair (x_n, x_m). If the number of pixels $N_{n,m,k}$ within $N_1(x_n, x_m)$ belonging to superpixel S_k is greater than the number of pixels $N_{n,m,j}$ belonging to superpixel S_j, pixel x_n is more likely to belong to superpixel S_k as well[1] [vdBBR+12]. The criterion for pixel updates given in line 17 of algorithm 6 changes to

$$N_{n,m,k} \cdot h_{S_k}(h(x_n)) > N_{n,m,j} \cdot h_{S_j}(h(x_n)). \tag{3.6}$$

For mean pixel updates as given in algorithm 7, line 7 changes to

$$\frac{d(x_n, S_k)}{N_{n,m,k}} < \frac{d(x_n, S_j)}{N_{n,m,j}}. \tag{3.7}$$

[1] In practice, the pixel to be exchanged is not considered within the counts $N_{n,m,k}$ and $N_{n,m,j}$.

Algorithm 3.7 As alternative to pixel updates as described in algorithm 6, Van den Bergh et al. [vdBBR+13] propose mean pixel updates. This variant of **SEEDS** is referred to as **SEEDSmp**.

Input: color image I, initial superpixel segmentation S
Output: superpixel segmentation S

```
for k = 1 to K
    initialize mean I(S_k)
// Mean pixel updates:
for n = 1 to N
    let S_j be the superpixel x_n belongs to
    if a neighboring pixel belongs to a different superpixel S_k
      then if d(x_n, S_k) < d(x_n, S_j)
              then S_k ← S_k ∪ {x_n} and S_j ← S_j − {x_n}
return S
```

An alternative smoothing term [vdBBR+13] uses the pixel coordinates to redefine the distance given in equation (3.5) as

$$d(x_n, S_j) = \|I(x_n) - I(S_j)\|_2 + \beta\|x_n - \mu(S_j)\|_2. \tag{3.8}$$

where β controls the compactness of the generated superpixels. Note that both smoothing terms can also be combined. With **SEEDSmp*** we refer to **SEEDS** using mean pixel updates and the alternative smoothing term of equation (3.8).

3.1.2 Complexity

In the following we give a detailed analysis of the complexity of **SEEDS**. For initialization, we first run over all pixels to compute the corresponding histogram bins and then compute the histograms for level $l = 1$. Computing the histogram bins for each pixel can be done in $O(N)$ operations using uniform binning or in $O(3N + 255Q)$ operations using non-uniform binning based on integral channels[2]. Together, this requires

$$3N + 255Q + \left\lfloor \frac{W}{w^{(1)}} \right\rfloor \cdot \left\lfloor \frac{H}{h^{(1)}} \right\rfloor \cdot w^{(1)} \cdot h^{(1)} = O(4N + 255Q) \tag{3.9}$$

operations. The histograms at level $1 < l \leq L$ can be computed as the sum of the corresponding histograms at level $(l-1)$: four blocks at level $(l-1)$ form a block at level l. Then the initialization of the histograms for all levels $1 < l \leq L$ requires

$$\sum_{l=2}^{L} 4 \cdot \left\lfloor \frac{W}{w^{(l)}} \right\rfloor \cdot \left\lfloor \frac{H}{h^{(l)}} \right\rfloor \cdot Q \approx \sum_{l=2}^{L} 4 \cdot \frac{N}{w^{(l)} h^{(l)}} \cdot Q \tag{3.10}$$

[2]For each 8-bit channel of the color image I, the corresponding integral channel represents an array where the i^{th} entry, $0 \leq i \leq 255$, counts the number of pixels whose color in this channel is equal or smaller than i. Then, non-uniform binning chooses the bins such that approximately the same number of pixels fall in each bin.

operations. Considering $w^{(1)} \geq 2$ and $h^{(1)} \geq 2$, equations (3.9) and (3.10) are at most linear in the number of pixels: $O(4QN)$.

For block updates at level $1 \leq l < L$ we consider the worst case: each block has four neighbors belonging to different superpixels. Then, we need to compute the histogram intersection five times: four times for the neighboring superpixels, once for the current superpixel. Overall, for block updates at all levels $1 \leq l < L$, this results in

$$\sum_{l=1}^{L-1} 5 \cdot \left\lfloor \frac{W}{w^{(l)}} \right\rfloor \cdot \left\lfloor \frac{H}{h^{(l)}} \right\rfloor \cdot Q \approx \sum_{l=1}^{L-1} 5 \cdot \frac{N}{w^{(l)} h^{(l)}} \cdot Q \tag{3.11}$$

operations. Hence, block updates at all levels $1 \leq l < L$ are linear in N when considering the histogram size Q to be a constant: $O(5QN)$.

Considering pixel updates as in algorithm 6, for each pixel we need to calculate the similarity to five different superpixels: $O(5N)$ operations. Considering mean pixel updates as in algorithm 7, computing the superpixel centers can be done in $O(N+K)$ operations[3]. Further, in the worst case, $O(5N)$ operations are necessary to perform mean pixel updates.

Overall, the algorithm has linear runtime in the number of pixels. However, for block updates, the number of histogram bins plays an important role. In addition, both block and pixel updates are in practice iterated more than once, such that the number of iterations T influences the runtime as well: $O(QTN)$. While this describes the worst case, we note that in practice, the runtime is also influenced by the number of levels L and the block size $w^{(1)} \times h^{(1)}$.

3.1.3 Theoretical Background

We follow [vdBBR+12] to derive the energy which Van den Bergh et al. aim to maximize using algorithm 6. The energy comprises a color term $H(S)$ and a smoothing term $B(S)$:

$$E(S) = H(S) + \lambda B(S). \tag{3.12}$$

where λ is a balancing parameter. In practice, λ is not adaptable as the smoothing term introduced in equation (3.6) does not allow to adapt its importance.

The color term favors superpixels with color histograms concentrated in a single bin. Therefore, a color term of the form

$$H(S) = \sum_{S_j \in S} \sum_{q=1}^{Q} h_{S_j}(q)^2 \tag{3.13}$$

[3] In practice, for each superpixel S_k the sums and the normalization needed to compute the means $I(S_k)$ and $\mu(S_k)$ are stored separately, resulting in $O(N)$ operations for initialization.

is used. As of equation (3.13), $H(S)$ reaches its maximum if and only if each superpixel histogram is concentrated in a single bin [vdBBR$^+$12]. As alternative, Van den Bergh et al. refer to an entropy based color term[4]. Another possible color term is given by

$$H(S) = \sum_{S_j \in S} \max_q \{h_{S_j}(q)\}, \tag{3.14}$$

favoring histograms concentrated in a single bin as well.

The boundary term is based on superpixel histograms. For each pixel x_n, the corresponding superpixel histogram g_{x_n} counts the number of pixels $x_m \in N_R(x_n)$ belonging to each superpixel $1 \leq k \leq K$. Then, the boundary term is given by

$$B(S) = \sum_{n=1}^{N} \sum_{k=1}^{K} g_{x_n}(k)^2. \tag{3.15}$$

This boundary term favors regular superpixels with smooth borders: When considering the superpixel histograms of all pixels within a specific superpixel, the superpixel histograms of pixels lying at the border will never be concentrated in one bin. However, with a more compact form of the superpixel, more pixels will have a uniformly labeled neighborhood, maximizing $B(S)$ [vdBBR$^+$12].

To implement a hill climbing algorithm maximizing the energy $E(S)$, Van den Bergh et al. proof the following propositions:

Proposition 1. *Let S_j, S_k be superpixels similar in size, and the size of block $B_i^{(l)} \subseteq S_j$ be much smaller: $|B_i^{(l)}| \ll |S_j| \approx |S_k|$. Let S denote the current superpixel segmentation and S_p denote the proposed superpixel segmentation where block $B_i^{(l)}$ is moved to superpixel S_k. If the color histogram $h_{B_i^{(l)}}$ is concentrated in one bin, it holds:*

$$\cap(h_{B_i^{(l)}}, h_{S_k}) \geq \cap(h_{B_i^{(l)}}, h_{S_j - B_i^{(l)}}) \Leftrightarrow H(S_p) \geq H(S) \tag{3.16}$$

where $h_{S_j - B_i^{(l)}}$ is the color histogram computed over the pixels in $S_j - B_i^{(l)}$.

Proposition 2. *Let x_n be a pixel belonging to superpixel S_j. Let S denote the current superpixel segmentation and S_p denote the proposed superpixel segmentation where x_n is moved to superpixel S_k. Further, let M_{x_n} be the set of all pixels x_m for which $x_n \in N_R(x_m)$, then it holds:*

$$\sum_{x_m \in M_{x_n}} (g_{x_m}(k) + 1) \geq \sum_{x_m \in M_{x_n}} g_{x_m}(j) \Leftrightarrow G(S_p) \geq G(S). \tag{3.17}$$

Algorithm 6 is intended to maximize $E(S)$ via hill climbing: In each step, a new superpixel segmentation is proposed by exchanging blocks or pixels and the proposed superpixels segmentation is accepted if the energy increases.

[4]That is, $H(S) = -\sum_{S_j \in S} \sum_{q=1}^{Q} h_{S_j}(q) \log(h_{S_j}(q))$ where the histogram h_{S_j} is interpreted as discrete probability distribution.

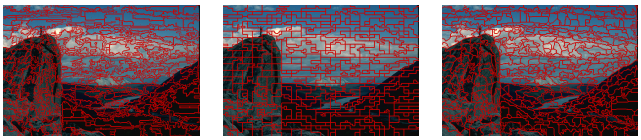

Figure 3.2: An illustration of the poor superpixel segmentations obtained by directly maximizing the proposed energy $E(S)$ given by equation (3.13). From left to right: the superpixel segmentation obtained after maximizing $E(S)$ directly using both block updates as well as pixel updates; the superpixel segmentation obtained after performing block updates at level $l = 1$ to directly maximize the energy $E(S)$; and the superpixel segmentation obtained after using block updates to optimize the energy $E(S)$ directly while using the pixel updates described in algorithm 6.

3.1.4 Discussion

Although the implementation provided by Van den Bergh et al. performs well, the implementation lacks relation to the theoretical background discussed in the previous section. Firstly, proposition 1 assumes the superpixel sizes to be comparable and the histogram $h_{B_i^{(l)}}$ to be concentrated in a single bin. Arguing that this holds in 93% of the cases [vdBBR$^+$12], this can merely be seen as heuristic rather than mathematical accurate. Secondly, our experiments show that the energy given in equation (3.12) is unsuited for superpixel segmentation in the first place. Even though, an implementation directly maximizing $E(S)$ through hill climbing may be inefficient, figure 3.2 presents superpixel segmentations obtained from a naive implementation. These experiments indicate that the idea of color histograms being concentrated in a single bin has to be relaxed in order to obtain proper superpixel segmentations. Thirdly, considering the smoothing term of the energy in equation (3.12), the weight λ cannot be adapted. Although this could be fixed by adapting equations (3.6) and (3.7), our experiments indicate that using the alternative smoothing term of equation (3.8) results in more compact and regular superpixels. Furthermore, mean pixel updates which are reported to give superior performance [vdBBR$^+$13] do not conform to the theoretical background introduced in the previous section. Overall, figure 3.3 shows the running examples oversegmented using our implementation with mean pixel updates as well as the alternative smoothing term of equation (3.8). In conclusion, we find the theoretical background of **SEEDS** as provided by Van den Bergh et al. [vdBBR$^+$12] unsuited in the light of the provided implementation and our experiments. However, a novel formulation of **SEEDS** lies not within the scope of this thesis.

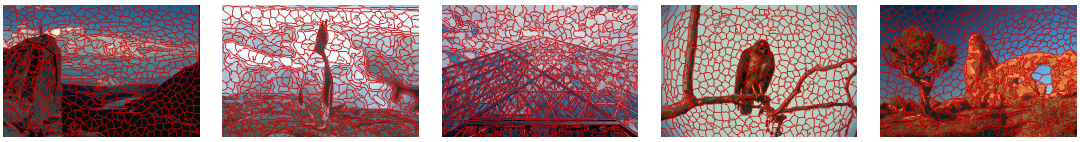

Figure 3.3: The running examples oversegmented into exactly 600 superpixels using our implementation of **SEEDS** using mean pixel updates and the alternative smoothing term of equation (3.8) which we refer to as **SEEDSmp***. Further examples can be found in section 7.2.1 and appendix B.

3.1.5 Implementation Details

Our implementation of **SEEDS** closely follows algorithm 6. The histograms are precomputed for all blocks across all levels. This can be done efficiently by accumulating histograms of level $(l-1)$ to form the histograms in level $1 < l \leq L$. The original implementation, as well as our implementation, both use non-uniform binning, that is the bin size is adapted to the colors present within the image. Then, the histograms only need to be updated after exchanging blocks or pixels. When using mean pixel updates as in algorithm 7, the superpixel centers need to be computed just before starting pixel updates. In practice the individual terms of equation (3.8) can also be normalized in order to choose the weighting parameter β more easily.

3.2 SLIC

SLIC is a gradient ascent method growing superpixels from initial superpixel centers using color similarity and spatial proximity. In particular, **SLIC** performs local K-means clustering where the search space for each superpixel is restricted to a local neighborhood around its center. The approach is easily implemented and adapted to custom needs. In addition, it can naturally be extended to depth or video data and is commonly used for comparison. We revisit the algorithm in detail and give a brief discussion.

3.2.1 Algorithm

The algorithm can be described as local K-means clustering in a five dimensional space comprising pixel coordinates and color [ASS+10], see algorithm 8. The superpixel centers are initialized on a regular grid with step size R. In practice, the centers are then moved to locations with low gradient magnitude to avoid placing superpixel centers at strong edges. The step size of the regular grid can be computed as

$$R = \left\lfloor \frac{WH}{K} \right\rfloor. \tag{3.18}$$

Then, each pixel x_n gets assigned to the nearest superpixel with respect to the following distance:

$$d(x_n, S_j) = \|I(x_n) - I(S_j)\|_2 + \frac{\beta}{R}\|x_n - \mu(S_j)\|_2 \tag{3.19}$$

where β is a parameter controlling the compactness. However, in contrast to classical K-means clustering, for each superpixel only pixels in a $2R \times 2R$ neighborhood around the superpixel's center are of interest. Afterwards, the superpixel centers are updated according to the new assignment. This procedure is iterated until convergence or for a maximum number of T iterations. Finally, superpixels do not necessarily represent connected components such that **SLIC** needs to enforce connectivity after clustering.

Algorithm 3.8 SLIC is implemented as local K-means clustering. Here, local means that for each superpixel only pixels in a $2R \times 2R$ window around the superpixel's center are of interest. After clustering, **SLIC** needs to enforce connectivity.

Input: color image I, number of superpixels K
Output: superpixel segmentation S
1. initialize superpixel centers on a regular grid with step size R
2. move centers to low-gradient magnitude positions
3. **repeat**
4. **for** $k = 1$ **to** K
5. **for each** pixel x_n in a $2R \times 2R$ neighborhood around $\mu(S_k)$
6. **if** x_n is unassigned
7. **then** $S_k \leftarrow S_k \cup \{x_n\}$
8. **else if** $d(x_n, S_k) < d(x_n, S_{s(x_n)})$
9. **then** $S_k \leftarrow S_k \cup \{x_n\}$ and $S_{s(x_n)} \leftarrow S_{s(x_n)} - \{x_n\}$
10. **until** nothing changes // or maximum number of iterations reached.
11. enforce connectivity
12. **return** S

3.2.2 Theoretical Background

In general, K-means clustering minimizes an energy given by

$$E(S) = \sum_{S_j \in S} \sum_{n=1}^{N} a_{n,j} d(x_n, S_j)^2 \tag{3.20}$$

where $a_{n,j} = 1$ if and only if pixel x_n is assigned to superpixel S_j, else $a_{n,j} = 0$. However, **SLIC** relaxes this energy in that the search space for each superpixel is restricted to a $2R \times 2R$ neighborhood around $\mu(S_j)$. Therefore, for fixed $\mu(S_j)$, the energy can be rewritten as

$$E(S) = \sum_{S_j \in S} \sum_{x_n \in N_{2R}(\mu(S_j))} a_{n,j} d(x_n, S_j)^2. \tag{3.21}$$

Additionally, the above energy is not minimized subject to the constraint that the superpixels S_j are connected components. Therefore, the last step of enforcing connectivity may actually increase the energy.

3.2.3 Complexity

While classical K-means clustering has complexity $O(NKT)$, **SLIC** runs linear in the number of pixels because of the restricted search space: For each superpixel, only pixels in a $2R \times 2R$ neighborhood around the superpixel's center are considered. Therefore, **SLIC** has complexity $O(TN)$.

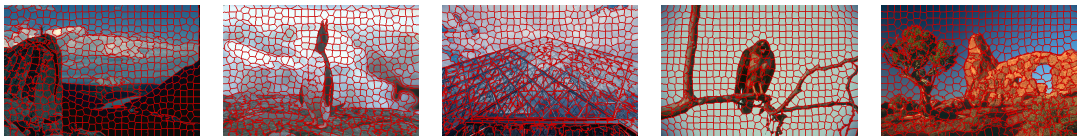

Figure 3.4: Superpixel segmentations with roughly 600 superpixels of the running examples generated by the original implementation of **SLIC**. Further examples can be found in section 7.2.1 and appendix B.

3.2.4 Discussion

One advantage of **SLIC** is its simplicity and efficiency [ASS+12]. There are several implementations available: the original implementation by Achanta et al. [ASS+10], an implementation as part of the VLFeat Library [VF08] and a parallel GPU implementation by Ren and Reid [RR11]. Further, **SLIC** can easily be adapted by changing the distance in equation (3.19). For example, Schick et al. improve the balancing between color and spatial term by changing equation (3.19) to

$$d(x_n, S_j) = (1-\beta)\|I(x_n) - I(S_j)\|_2 + \frac{\beta}{R}\|x_n - \mu(S_j)\|_2. \tag{3.22}$$

On the other hand, **SLIC** only generates a valid superpixel segmentation after enforcing connectivity. This means, that the algorithm cannot be aborted after an arbitrary number of iterations. In addition, Van den Bergh et al. [vdBBR+13] report lower performance with more iterations which may be due to the generated "stray labels" [vdBBR+13] – small groups of pixels not connected to their corresponding superpixel. This is also confirmed by Tang et al. [TFC12] reporting only a small decrease in performance when using only $T = 1$ iteration. However, considering an approach similar to **VCCS** [PASW13] (see section 4.3), breadth-first search could be used as basis for K-means clustering in order to avoid stray labels. To sum up, figure 3.4 shows the running examples oversegmented using the original implementation of **SLIC**.

Chapter 4

Supervoxel Segmentation and Superpixel Segmentation using Depth

The focus of this thesis is the integration of depth information into **SEEDS** [vdBBR$^+$12] as we believe that depth may serve as additional cue to identify object boundaries and improve the generated superpixel segmentation. Some of the superpixel algorithms discussed in section 2.1 can easily be extended to use depth information, for example **CRS** [CMM13] and **SLIC** [ASS$^+$10]. Graph based approaches can also be extended to use depth information by adapting the weights accordingly, for example **ERS** [LTRC11] and **PB** [ZHMB11a]. An extension of **SLIC** based on depth is discussed in the following section. However, there are more sophisticated approaches available: **DASP** [WGB12] places the initial superpixel centers depending on the depth, and with **VCCS** [PASW13], Papon et al. provide an approach able to oversegment point clouds. We revisit both approaches in detail as they will serve as additional baselines and present state-of-the-art algorithms using depth as an integral component.

4.1 SLIC using Depth Information

SLIC is built around a distance function $d(x_n, S_j)$ as defined in equation (3.19). As this distance already uses pixel coordinates, we can add an additional term to introduce depth:

$$d(x_n, S_j) = \|I(x_n) - I(S_j)\|_2 + \frac{\beta}{R}\|x_n - \mu(S_j)\| + \gamma\|D(x_n) - D(S_j)\|_2 \tag{4.1}$$

where γ is an additional weighting term and $D : \underline{W} \times \underline{H} \to \mathbb{R}$ represents a depth image. The depth of a superpixel S_j is defined as $D(S_j) = \frac{1}{|S_j|}\sum_{x_m \in S_i} D(x_m)$. This distance aims to discourage superpixels crossing boundaries including a difference in depth.

Instead of using pixel coordinates and depth as separate information, we can project the pixel coordinates x_n into three-dimensional space to obtain the corresponding 3D point coordinates which we denote by $P(x_n) \in \mathbb{R}^3$. The distance in equation (4.1)

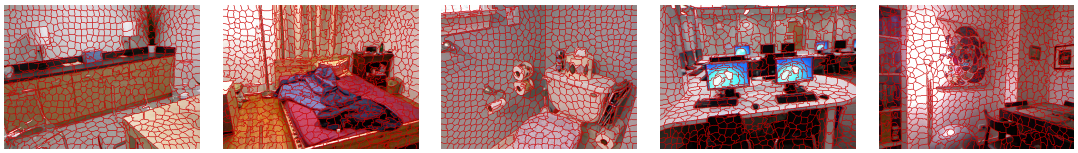

Figure 4.1: Superpixel segmentations with roughly 840 superpixels of the running examples generated using **SLIC3D**. Further examples can be found in section 7.2.1 or appendix B.

changes to

$$d(x_n, S_j) = \|I(x_i) - I(S_j)\|_2 + \beta\|P(x_n) - P(S_j)\|_2 \tag{4.2}$$

with $P(S_j) = \frac{1}{|S_j|}\sum_{s_m \in S_j} P(x_m)$. Connectivity can still be enforced on the image plane. We implemented this variant of **SLIC**, in the following referred to as **SLIC3D**, as additional baseline.

4.1.1 Discussion

Because of **SLIC**'s good performance, we do not expect a significant increase in performance while slightly increasing the runtime due to the extended distance in equation (4.2). However, we are able to observe a visual difference in the generated superpixels as, depending on the value of β, **SLIC3D** enforces compactness within three dimensional space such that the superpixels appear to reflect the underlying three dimensional structure. Figure 4.1 shows the generated superpixel segmentations of the running examples.

4.2 DASP

The approach proposed by Weikersdorfer et al. [WGB12] is based on an important observation. Usually, we choose the number of superpixels such that we obtain superpixels comparable in size to the smallest objects we are interested in [WGB12]. Consequently, the density of superpixels is higher at surfaces farther away from the image plane [WGB12], that is with increasing depth the superpixels get smaller on average. Nevertheless, we want the superpixels to be distributed uniformly on surfaces at any depth. This is achieved by defining a superpixel density on the depth image. We follow [WGB12] and consider a disk of radius R centered at pixel x_n in depth $D(x_n)$. Using the model of a pinhole camera [FP02], the radius of this disk projected onto the image plane is given by

$$r(x_n) = \frac{f}{D(x_n)}R \tag{4.3}$$

where f denotes the focal length of the camera. As noted in [WGB12], this equation can only be applied in cases where the disk is parallel to the image plane. Otherwise, a

Algorithm 4.9 The easiest way to sample superpixel centers form the superpixel density $p(x_n)$ is given by random sampling. For each pixel, a number in the range $[0,1]$ is randomly chosen and compared to the probability $p(x_n)$ of x_n being an initial superpixel center. For details how the desired number of superpixels is met we refer to the implementation[1].

Input: color image I, density $p(x_n) \propto \frac{1}{A(x_n)}$
Output: superpixel centers $\mu(S) = \{\mu(S_1), \ldots, \mu(S_K)\}$
1. initialize $\mu(S) = \emptyset$
2. **for** $n = 1$ **to** N
3. $r \leftarrow random([0,1])$
4. **if** $r < p(x_n)$
5. **then** $\mu(S) \leftarrow \mu(S) \cup \{x_n\}$
6. **return** $\mu(S)$

projective transformation would be necessary which Weikersdorfer et al. approximate using an affine transformation. First, the gradient $\nabla D(x_n)$ of the depth image is estimated using finite differences. Then, the area of the projected disk can be expressed as

$$A(x_n) = \frac{r(x_n)^2 \pi}{\sqrt{\|\nabla D(x_n)\|_2^2 + 1}} \tag{4.4}$$

and the superpixel density at pixel x_n is proportional to $\frac{1}{A(x_n)}$. A detailed derivation can be found in [Wei14]. In practice, the provided implementation of **DASP** offers several methods to sample the initial superpixel centers from this density, one of which is given by random sampling, summarized in algorithm 9. Note that the above mechanism can also be applied to custom densities or densities based on different information, see [Wei14] for details.

Similar to **SLIC**, the algorithm alternates between the assignment step where each pixel is assigned to the nearest superpixel and the update step where the superpixel centers are updated. The used distance is given by

$$d(x_n, S_j) = \|I(x_n) - I(S_j)\|_2 + \beta \|P(x_n) - P(S_j)\|_2 + \gamma \left(1 - N(x_n)^T N(S_j)\right) \tag{4.5}$$

where β and γ are parameters weighting the corresponding terms and $N(x_n)$ denotes the normal corresponding to pixel x_n which we assume to be normalized. Again, we use $N(S_j) = \frac{1}{|S_j|} \sum_{x_m \in S_j} N(x_m)$ and the normals are estimated using depth gradients. The term $N(x_n)^T N(S_j)$ represents the cosine of the angle between the normals $N(x_n)$ and $N(S_j)$. Here, the intuition is that pixel x_n and superpixel S_j lie on the same planar surface if this angle is small. On the other hand, the cosine of this angle reaches zero if the normals are orthogonal to each other.

Figure 4.2: Superpixel segmentations with roughly 840 superpixels of the running examples generated by **DASP**. Further examples are available in section 7.2.1 and appendix B.

4.2.1 Discussion

In contrast to **SLIC3D**, the approach presented above not only uses 3D point coordinates and normal information but also adapts the size of superpixels according to the given depth. This way we expect to get performance comparable to superpixel algorithms like **SLIC** and **SEEDS**, however, with a lower number of superpixels. Additionally, by using normal information, **DASP** may recognize additional boundaries not apparent when using color and 3D point coordinates only. Figure 4.2 shows the running examples oversegmented using **DASP**.

4.3 VCCS

Although **VCCS** may be applied to arbitrary point clouds as for example generated from multiple RGB-D cameras or laser scans [PASW13], we assume the point cloud to be created from a single image with corresponding depth image. Note that this results in a dense and ordered point cloud[2]. The point cloud is voxelized using a given voxel resolution R_v such that an 26-adjacency graph can be defined, see [Rus09] for details.

The idea of **VCCS** is similar to **SLIC** in that it performs local K-means clustering within the voxelized point cloud. However, **VCCS** ensures connectivity by applying breadth-first search as basis for clustering. After placing supervoxel centers at a regular grid with step size[3] R, also called supervoxel resolution[4], we first need to remove supervoxel centers which have no voxels in their immediate neighborhood. If a given supervoxel center would lie on a planar surface, then the following equation yields the number of voxels within a specified search radius R_s:

$$\frac{R_s^2 \pi}{R_v^2} \tag{4.6}$$

where R_v^2 describes the ground area of a single voxel. All supervoxel centers having less voxels in the sphere of radius R_s are discarded (see the implementation in the Point

[2]That is, the points are accessible by their pixel coordinates within a two dimensional array.

[3]Note that this step size is applied in three dimensional space.

[4]Actually, Papon et al. [PASW13] call R seed resolution where a seed corresponds to a supervoxel center. This naming is often used for superpixel centers as well [ASS+10].

Algorithm 4.10 VCCS uses K-means clustering based on breadth-first search beginning at the supervoxel centers to assign each voxel to a supervoxel. This way, **VCCS** ensures that the supervoxels represent connected components within the 26-adjacency graph derived from the voxelized point cloud. The below algorithm can easily be adapted to return a supervoxel segmentation instead of a superpixel segmentation.

Input: voxelized point cloud, supervoxel resolution R, search radius R_s
Output: superpixel segmentation S

```
1.  place supervoxel centers on a regular grid with step size R
2.  discard unnecessary supervoxel centers based on the search radius R_s
3.  move supervoxel centers to low gradient magnitude positions
4.  for t = 1 to T // T is the maximum depth.
5.      for k = 1 to K
6.          perform one step breadth first search for supervoxel S_k
7.          update supervoxel center
8.  derive superpixel segmentation S by backprojecting the supervoxels
9.  return S
```

Cloud Library[5] [RC11] for details). Afterwards, the supervoxel centers are moved to low gradient magnitude positions similar to **SLIC**.

According to [PASW13], **VCCS** performs clustering in a 39 dimensional space comprising color, 3D point coordinates and Fast Point Feature Histograms [RBB09], short FPFH. However, the implementation available as part of the Point Cloud Library suggests that instead of FPFH features, point normals are used. Additionally, our experiments suggest that using FPFH features is quite expensive regarding runtime. Thus, we focus on the version as provided by the Point Cloud Library. The used distance function is given by

$$d(x_n, S_j) = \|I(x_n) - I(S_j)\|_2 + \beta\|P(x_n) - P(S_j)\|_2 + \gamma\left(1 - N(x_n)^T N(S_j)\right) \tag{4.7}$$

where β and γ are parameters weighting the spatial and normal terms.

The clustering process is different to the local K-means clustering used by **SLIC** in that it implements a breadth-first search beginning at the supervoxel centers. As for **SLIC**, the pixels being of interest for a particular supervoxel S_j are limited by the maximum depth T of the breadth-first search. In practice, the maximum depth T is set to

$$T = 1.8 \cdot \frac{R}{R_v}. \tag{4.8}$$

VCCS is, slightly simplified, described in algorithm 10.

[5]The Point Cloud Library, implemented in C++, is mainly aimed to provide state-of-the-art computer vision algorithms running on point clouds: `http://pointclouds.org/`.

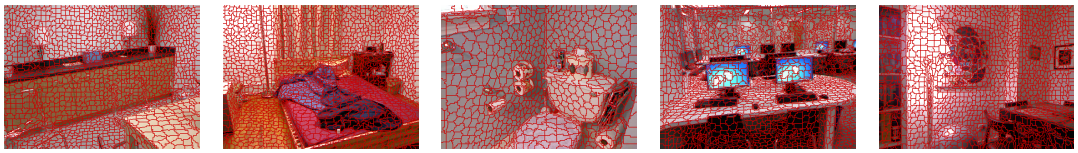

Figure 4.3: The running examples oversegmented into roughly 840 superpixels using **VCCS**. Further examples can be found in section 7.2.1 and appendix B.

4.3.1 Discussion

While the approaches discussed in section 4.1 and 4.2 generate superpixel segmentations using depth as additional cue, **VCCS** creates supervoxels directly in three dimensional space. This is advantageous when considering input from multiple RGB-D cameras describing the same scene or laser scans [PASW13]. Furthermore, the algorithm can be applied to unordered point clouds. Additionally, in contrast to **SLIC**, **VCCS** does not need to enforce connectivity afterwards.

On the other hand, being dependent on a point cloud can also have major disadvantages when aiming to oversegment images for which depth information is available separately. For example **SLIC3D** or **DASP** can easily be applied to images with incomplete depth information by falling back to a color only mode. As we discuss in chapter 6, this is often the case when grabbing raw images from RGB-D cameras like the Microsoft Kinect. In such a case we are not able to generate a complete point cloud such that **VCCS** cannot be applied to the whole image. However, note that these considerations differ depending on the application. Furthermore, we note that the comparison of superpixel algorithms with supervoxel algorithms is difficult by design.

As the distance used for clustering is equal to the distance used by **DASP**, we cannot expect to observe increased performance due to different features. However, in contrast to **SLIC3D** and **DASP**, **VCCS** avoids stray labels and respects the underlying three dimensional surface directly by oversegmenting a point cloud in the form of a 26-adjacency graph. Although **SLIC3D** and **DASP** would be able to detect boundaries within the depth image, depending on the parameter β, these boundaries can also be ignored. Due to the breadth-first search utilized by **VCCS** such cases are handled automatically. Overall, we expect improved performance for a lower number of superpixels when compared to other approaches. **VCCS** applied to the running examples is shown in figure 4.3.

Chapter 5

SEEDS using Depth Information

This chapter covers the main topic of this thesis and presents extensions of **SEEDS** [vdBBR+12] utilizing depth information. In particular, we aim to preserve the framework given by **SEEDS**, which can best be described as the combination of block and pixel updates, while improving the generated superpixel segmentation. As described in sections 5.1 and 5.2, integrating depth information into pixel updates can be done similar to **SLIC3D** and **DASP** [WGB12]. However, in our opinion, block updates are responsible for the excellent runtime of **SEEDS** and therefore we discuss several approaches to improve block updates using depth as well.

5.1 3D Pixel Updates

Similar to **SLIC3D**, it is possible to integrate 3D point coordinates into mean pixel updates by changing the distance in equation (3.5) to

$$d(x_n, S_j) = \|I(x_n) - I(S_j)\|_2 + \beta\|P(x_n) - P(S_j)\|_2 \tag{5.1}$$

where β can be interpreted as parameter controlling compactness of superpixels within three dimensional space. Using mean pixel updates with the above distance function is referred to as **SEEDS3D**.

5.1.1 Discussion

Although simple, experiments in chapter 7 show that usage of 3D point coordinates for pixel updates slightly increases performance. This observation can be explained by considering that pixel updates have the greatest influence on the final superpixel segmentation. Therefore, depending on the parameter β, the above pixel updates are capable of adapting the initial superpixel segmentation to the underlying three dimensional structure. This includes boundaries within the depth image as well as more difficult boundaries between objects in highly cluttered scenes where the above distance prevents superpixels to cover multiple objects and enforces compactness on the

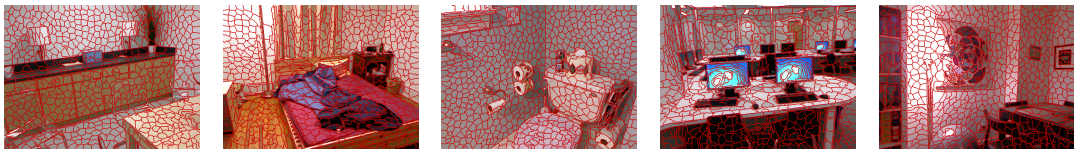

Figure 5.1: The running examples oversegmented into exactly 840 superpixels using **SEEDS3D**. Further examples are available in section 7.2.1 and appendix B.

object's surface. In contrast to **VCCS** [PASW13], however, when using a small β, superpixels may still cover multiple objects even if these objects are not connected in the sense of a voxelized point cloud. Furthermore, the block updates remain unaffected such that the complexity remains unchanged. Figure 5.1 shows the running examples oversegmented using **SEEDS3D**.

5.2 3D Pixel Updates using Normal Information

Similar to **DASP**, normal information can be integrated into pixel updates by adapting equation (5.1) to

$$d(x_n, S_j) = \|I(x_n) - I(S_j)\|_2 + \beta\|P(x_n) - P(S_j)\|_2 + \gamma \arccos\left(N(x_n)^T N(S_j)\right) \quad (5.2)$$

where β is a compactness parameter and γ controls the influence of normal information. Instead of computing the arc-cosine of the dot product $N(x_n)^T N(S_j)$, a distance similar to the one utilized by **DASP** can be used. This variant of **SEEDS3D** is in the following referred to as **SEEDS3Dn**.

The problem of estimating the normal $N(x_n)$ for the point $P(x_n)$ based on a set of points $P(x_{n_1}), \ldots, P(x_{n_k})$ within its local neighborhood is discussed in detail in [Rus09] and several implementations are available as part of the Point Cloud Library [RC11]. For example, using the covariance matric C of the points $P(x_n), P(x_{n_1}), \ldots, P(x_{n_k})$, the normal $N(x_n)$ can be estimated by computing the eigenvalues and eigenvectors of C. This corresponds to a first order plane fit. As we can base normal estimation on an ordered point cloud constructed from the depth image D, normals can be computed efficiently using integral images, see [HRD+12]. In contrast, **VCCS** uses least squares plane fits to compute normals within an unordered point cloud and **DASP** estimates normals purely based on the depth gradient ∇D.

5.2.1 Discussion

While normal estimation can be done quite efficiently and results in eligible normal estimates, these are still quite noisy and lack accuracy especially in highly cluttered scenes and at object borders, see figure 5.2. Additionally, normals estimated for regions where depth information has been added artificially due to incomplete depth

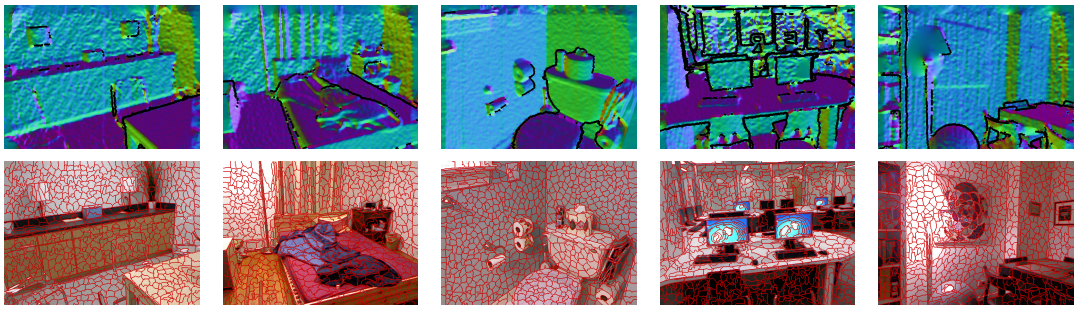

Figure 5.2: Top: estimated normals for the running examples color coded – black indicates that no normal could be estimated for this particular pixel. Bottom: the running examples oversegmented into exactly 840 superpixels using **SEEDS3Dn**.

images are not reliable such that normal information should only be used in addition to 3D point coordinates and color.

As experiments in chapter 7 show, using normal information for pixel updates results in a negligible performance increase when the normal information is weighted sufficiently low. Whether this is due to the noisy normal computation or the fact that superpixel algorithms using color and 3D point coordinates only already perform well needs to be verified on different datasets. Overall, the running examples oversegmented using **SEEDS3Dn** are shown in figure 5.2.

5.3 Mean-Based Block Updates

Color histograms offer a suitable representation for blocks of pixels. However, as approaches like **SLIC** [ASS+10], **DASP** and **VCCS** successfully apply the concept of superpixel centers to superpixel segmentation, it may be beneficial to use similar representations for block updates as well. Consequently, each block $B_i^{(l)}$ is represented by its center and its similarity to a superpixel S_j can be expressed as distance:

$$d(B_i^{(l)}, S_j) = \|I(B_i^{(l)}) - I(S_j)\|_2 + \beta\|P(B_i^{(l)}) - P(S_j)\|_2. \tag{5.3}$$

The block centers can be pre-computed during initialization, see algorithm 11.

The above concept can also be extended to use normal information. In practice, this amounts to computing the blocks mean color and position as well as its orientation. In the simplest form, the orientation $N(B_i^{(l)})$ of block $B_i^{(l)}$ is given by the average of the orientations $N(x_n)$ of the corresponding pixels such that the similarity of block $B_i^{(l)}$ and superpixel S_j can be expressed as

$$\begin{aligned} d(B_i^{(l)}, S_j) =& \|I(B_i^{(l)}) - I(S_j)\|_2 + \beta\|P(B_i^{(l)}) - P(S_j)\|_2 \\ &+ \gamma \arccos\left(N(B_i^{(l)})^T N(S_j)\right). \end{aligned} \tag{5.4}$$

Algorithm 5.11 After defining a distance $d(B_i^{(l)}, S_j)$ between blocks and superpixels based on their respective centers, block updates can be performed without relying on color histograms. Of course, both approaches can also be combined resulting in a higher runtime

Input: color image I, block size $w^{(1)} \times h^{(1)}$, number of levels L
Output: superpixel segmentation S

```
1.  initialize the block hierarchy and the initial superpixel segmentation S
2.  // Initialization of block/superpixel centers:
3.  for l = 1 to L
4.      for each block B_i^(l) at level l // For l = L these are the initial superpixels.
5.          compute I(B_i^(l)) and P(B_i^(l))
6.  // Mean based block updates:
7.  for l = L − 1 to 1
8.      for each block B_i^(l) at level l
9.          let S_j be the superpixel B_i^(l) belongs to
10.         if a neighboring block belongs to a different superpixel S_k
11.           then if d(B_i^(l), S_k) < d(B_i^(l), S_j)
12.                   then S_k ←S_k ∪ B_i^(l) and S_j ←S_j − B_i^(l)
13. mean pixel updates // Note that the superpixel centers are already available.
14. return S
```

This is the natural extension of **SEEDS3Dn** to block updates. Together, the position $P(B_i^{(l)})$ and the orientation $N(B_i^{(l)})$ can be interpreted as plane such that the orientation could also be estimated using a plane fit.

5.3.1 Discussion

Unfortunately, block centers appear to be a poor representation for blocks of pixels. For example consider the running examples in figure 5.3. Choosing a block crossing a strong boundary between different objects, a color histogram is able to capture both parts of the block by having two modes. However, the block center may capture information not present within the block. In contrast, superpixels computed after performing block updates at level $l = 1$ appear to be homogeneous in color and cover mostly one object such that using superpixel centers is appropriate for pixel updates. Therefore, using mean position and color as representation of superpixels is appropriate only when given a sound initial superpixel segmentation and consequently blocks of pixels are better represented by color histograms. The same considerations apply for averaging normals. Therefore, we used the covariance matrix to estimate the orientation of the blocks and superpixels. However, after each update, the orientation of the corresponding superpixels need to be recomputed. Although this approach is comparably inefficient, our experiments show that even after computing the optimal

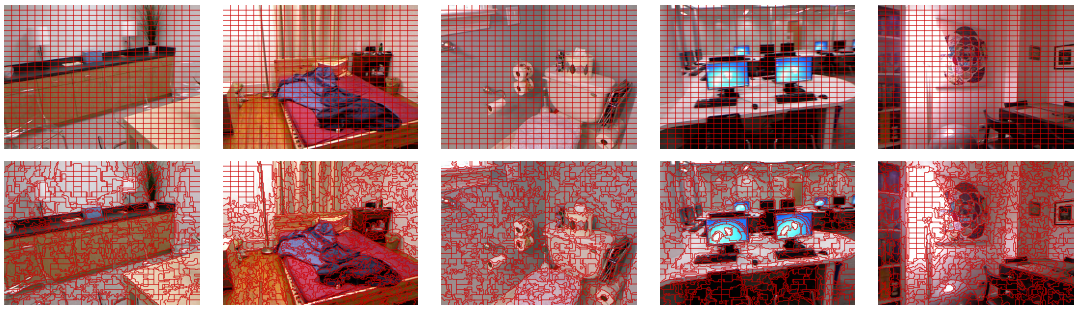

Figure 5.3: Top: Block segmentations at level $(L-1)$. Bottom: superpixel segmentations before performing pixel updates.

orientation in the least squares sense, using normal information does not increase performance. Furthermore, computing block centers at lower levels, where each block contains only few pixels, is highly influenced by outliers and noise. Overall, these considerations imply that mean based block updates perform poorly when compared to the original block updates in algorithm 6. This can also be interpreted as motivation of superpixel growing as done by **SLIC**, **DASP** and **VCCS**.

5.4 Block Updates Based on Normal Histograms

As color histograms appear to work well as representations for blocks of pixels, it appears to be advantageous to integrate depth information in the form of histograms. Depth histograms, however, do not offer an increase in performance. Therefore, we would like block updates to capture normal information instead. This can be accomplished using histograms of normal orientation which we refer to as normal histograms.

Given a point normal $N(x_n)$ with components $N_1(x_n), N_2(x_n), N_3(x_n)$, the orientation can be split up into the polar angle ϕ between normal and the y-axis, and the azimuthal angle θ between normal and x-axis, see figure 5.4. The angles are binned to form a two dimensional histogram and the similarity of a block and a superpixel is then expressed as the intersection of these histograms. This can be weighted and used in combination with color histograms.

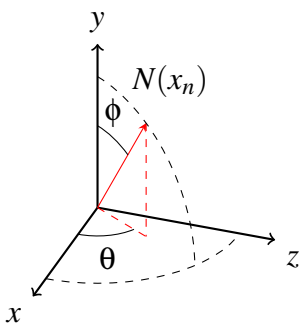

Figure 5.4: Point normals are binned using the polar angle ϕ between normal and y-axis and the azimuthal angle θ between normal and x-axis. In practice, the normal is always oriented towards the viewpoint such that $\phi, \theta \in [0, 180°]$.

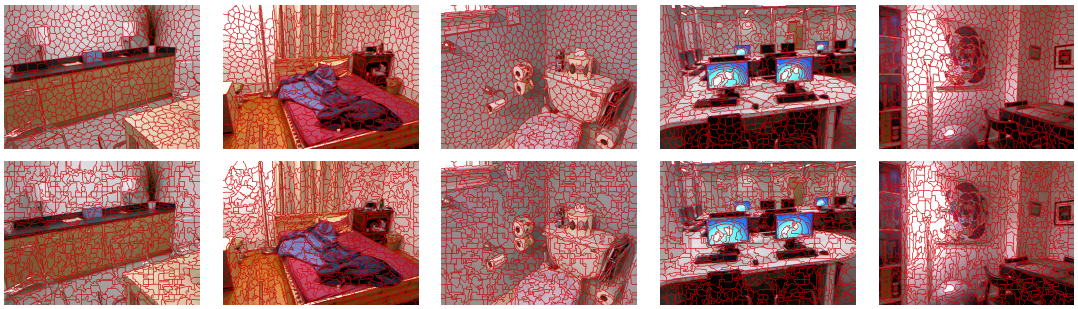

Figure 5.5: Top: the running examples oversegmented using normal histograms for block updates. Bottom: superpixel segmentations with exactly 840 superpixels of the running examples generated using normal histograms for both block updates and pixel updates.

5.4.1 Discussion

Considering the binning of the normals, we came to the conclusion that an odd number of bins per angle should be used. This is based on the following case: We consider a normal with $\phi = 90^\circ$ and $\theta = 90^\circ$. As this normal is not estimated perfectly, the angles would lie slightly below or above the 90° such that the actual bin depends on the noise and the estimation method which should be avoided to get an adequate representation of surfaces parallel to the image plane.

Overall, experiments show that this extension does not improve performance while additionally increasing runtime. In our opinion, the main reasons for this observation are the following. Firstly, as stated in section 5.2, a superpixel segmentation obtained by running **SEEDS** based purely on color achieves state-of-the-art performance leaving only little room for improvement. Secondly, the most difficult regions within images taken from the NYU Depth Dataset [SHKF12] contain plenty of small objects – often the scenes are highly cluttered. In such settings, the estimated normals tend to be highly unrepresentative and noisy. In contrast, in regions of nearly planar surfaces, where we expect normal information to be of help, oversegmentations based on color only usually yield high performance. In conclusion, we find that normal histograms are not worth their computation time. Figure 5.5 shows the running examples oversegmented using normal histograms. As pixel updates have not been adapted, the results differ only slightly from those presented in figure 5.1. In contrast, when using pixel updates based on color and normal histograms similar to those in algorithm 6, the superpixel segmentations appear to have low quality.

Chapter 6

Datasets and Benchmarks

In this chapter we discuss two popular datasets for evaluating segmentation algorithms: the Berkeley Segmentation Dataset [AMFM11] and the NYU Depth Dataset [SHKF12]. Furthermore, we discuss several measures used to assess superpixel algorithms which have, in the course of this thesis, been implemented as extension of the Berkeley Segmentation Benchmark which is provided as part of the Berkeley Segmentation Dataset by Arbeláez et al. [AMFM11].

6.1 Berkeley Segmentation Dataset

The first version of the Berkeley Segmentation Dataset, introduced in [MFTM01] and referred to as BSDS300, comprises 300 images split up into a training set of 200 images and a test set of 100 images. The second version [AMFM11], referred to as BSDS500, adds 200 additional images forming a new test set while the old test set is used as validation set instead. Per image, at least five ground truth segmentations are available, each annotated by a different person, see for example figure 6.1. Therefore, the ground truth segmentations are highly varying and reflect the difficult nature of segmenting natural images. Overall, this results in a fair evaluation of superpixel algorithms as for each superpixel segmentation a suitable ground truth segmentation can be chosen or the average performance over the ground truth segmentations can be

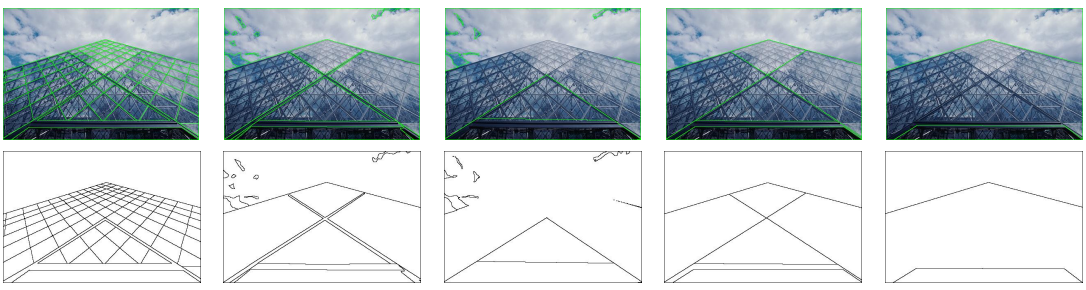

Figure 6.1: Ground truth segmentations of one of the running examples as provided by the BSDS500.

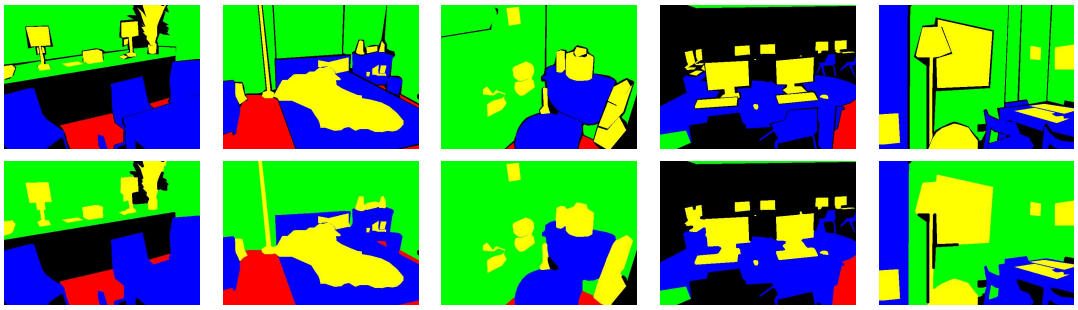

Figure 6.2: The NYUV2 provides a high variety of different labels. In [SHKF12], Silberman et al. grouped these labels to obtain a total of four so called structure labels. Top: the structure labels as provided by the NYUV2. Bottom: the structure labels after removing thin unlabeled regions in between larger labeled regions. Black indicates unlabeled pixels. Overall, this illustrates why the dataset is considered harder in comparison to the natural images from the BSDS500.

computed. In practice, concerning all measures based on ground truth segmentations presented in this chapter, for each image we choose the ground truth segmentation resulting in the best value and subsequently average over all images. However, note that some of the measures provided as part of the Berkeley Segmentation Benchmark offer natural extensions to multiple ground truth segmentations, as for example the Probabilistic Rand Index [UPH07], see appendix A.

6.2 NYU Depth Dataset

The first version of the NYU Depth Dataset, in the following referred to as NYUV1, was introduced in [SF11] by Silberman et al. The dataset comprises 2284[1] labeled frames taken from video sequences showing six different indoor scenes. The video sequences were taken by a calibrated Microsoft Kinect [SF11] such that depth information is accessible. The frames were annotated using the LabelMe tool [RTMF08] resulting in around 1000 classes which are highly redundant [SF11].

The second version of the NYU Depth Dataset [SHKF12], NYUV2, adds further variety in the form of indoor scenes from different cities as well as commercial accommodations. The dataset comprises 1449 labeled frames with a total of 894 classes [SHKF12]. Every instance of a particular class within a single frame gets a unique instance label. Following Ren and Bo [RB12] we combine class labels and instance labels to generate ground truth segmentations compatible with the Berkeley Segmentation Benchmark. Although Silberman et al. state that the annotators were instructed to provide a dense labeling of the scenes, the images often contain small unlabeled regions in between larger labeled regions, see figure 6.2. Therefore, based on code provided by Ren and Bo [RB12], we remove these small background regions. Fur-

[1] Although, Silberman et al. mention 2347 labeled frames [SF11], we found that, actually, the dataset comprises only 2284 labeled frames.

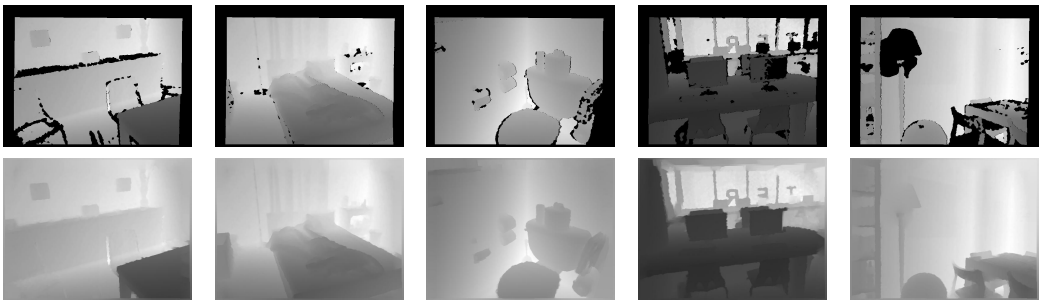

Figure 6.3: Top: the raw depth images corresponding to the running examples where black regions indicate that no depth information is available. Bottom: the pre-processed depth images corresponding to the running examples.

thermore, bad lighting as well as cluttered scenes account for the difficulty of the NYUV2. However, this also represents an appropriate contrast to the natural images of the BSDS500.

The NYUV2 includes both the raw depth images as well as a pre-processed version. The provided raw depth images have been aligned with the corresponding color images, however, the region covered by the depth images is significantly smaller and reflective surfaces such as mirrors and windows cause holes within the depth images. Therefore, Silberman et al. provide a pre-processed version of the depth images where these regions have been filled artificially. This is illustrated in figure 6.3, showing both the raw depth images as well as the pre-processed depth images of the running examples. Furthermore, the provided images have been undistorted resulting in a small white border around the actual images. Therefore, we crop the images of size 640×480 to obtain images of size 608×448 without white borders. Finally, we choose 200 images to form a training set and 400 images to form a test set both including most of the available indoor scenes[2].

6.3 Extended Berkeley Segmentation Benchmark

Most measures provided as part of the Berkeley Segmentation Benchmark are unsuited for evaluating superpixel algorithms, see appendix A. Therefore, in this section, we discuss most of the available measures used to evaluate superpixel algorithms. These have, except for Boundary Recall, in the course of this thesis, been implemented into the framework of the Berkeley Segmentation Benchmark.

[2]Our split is based on a training/test split provided by Ren and Bo [RB12]. However, to speed up evaluation, we reduced the size of the sets to 200 and 400, respectively. The images have been chosen such that almost all scenes are represented.

6.3.1 Boundary Recall

Boundary Recall is part of the Precision-Recall-Framework introduced in [MFM04] which is discussed in detail in appendix A. Let S be a superpixel segmentation and G be a ground truth segmentation, then we define:

True Positives, $TP(S,G)$: Boundary pixels in G for which there is a boundary pixel in S in range r.

False Negatives, $FN(S,G)$: Boundary pixels in G for which there is no boundary pixel in S in range r.

where r defines a tolerance parameter controlling the allowed deviation from the ground truth. In practice, r is set to 0.0075 times the image diagonal[3]. The Boundary Recall $Rec(S,G)$ is the fraction of all boundary pixels within the ground truth segmentation G which are correctly detected within the superpixel segmentation S, that is

$$Rec(S,G) = \frac{|TP(S,G)|}{|TP(S,G)| + |FN(S,G)|}. \tag{6.1}$$

As we want superpixels to respect the boundaries within the image, a high Boundary Recall is desirable.

6.3.2 Undersegmentation Error

The Undersegmentation Error describes the leakage or "bleeding" [LSK+09] of a superpixel with respect to a specific ground truth segment. In [LTRC11] and [VBM10], the Undersegmentation Error is defined as

$$UE(S,G) = \frac{1}{N}\left(\sum_{G_i \in G}\left(\sum_{S_j \cap G_i \neq \emptyset} |S_j|\right) - N\right). \tag{6.2}$$

This equation computes the total amount of leakage for all ground truth segments and normalizes the sum by the number of pixels[4]. We note that the normalization used above may not be sufficient. For example consider a superpixel segmentation with a single superpixel covering the whole image. Then, if the ground truth consists of more than two segments, the Undersegmentation Error will be greater than one. In addition, as stated by Neubert and Protzel [NP12] as well as Achanta et al. [ASS+10] this definition has another serious disadvantage. Consider a large superpixel covering one ground truth segment perfectly except for some pixels. In the above equation,

[3] This is the default value as used by the Berkeley Segmentation Benchmark and we chose not to change this value to keep our results comparable to other publications.

[4] We note that equation (6.2) uses a slightly different normalization when compared to the Undersegmentation Error as used by Levinshtein et al. in [LSK+09].

such superpixels receive high penalties [NP12]. Achanta et al. adapt equation (6.2) to tolerate a small amount of leakage per superpixel:

$$UE(S,G) = \frac{1}{N}\left(\sum_{G_i \in G}\left(\sum_{|S_j \cap G_i| > B} |S_j|\right) - N\right) \tag{6.3}$$

where B is the corresponding tolerance parameter. To avoid an additional parameter, we implemented the Undersegmentation Error as proposed by Neubert and Protzel:

$$UE(S,G) = \frac{1}{N}\sum_{G_i \in G}\sum_{S_j \cap G_i \neq \emptyset} \min\{|S_j \cap G_i|, |S_j - G_i)\}. \tag{6.4}$$

where for each superpixel only the smaller part is considered leakage.

6.3.3 Achievable Segmentation Accuracy

When using superpixels as pre-processing step, we want the performance of subsequent steps to be as far as possible unaffected [LTRC11]. Of course, as we inevitably loose information, this is not possible. However, we would like to quantify the accuracy achievable by subsequent steps, as for example classical segmentation. Achievable Segmentation Accuracy labels superpixels according to their underlying ground truth segments and counts the correctly labeled pixels [LTRC11]:

$$ASA(S,G) = \frac{1}{N}\sum_{S_j \in S} \max_{G_i}\{|S_j \cap G_i|\}. \tag{6.5}$$

Therefore, Achievable Segmentation Accuracy represents an upper bound on the accuracy achievable by a subsequent segmentation step [LTRC11].

6.3.4 Compactness

Schick et al. [SFS12] propose a compactness measure for superpixels based on the isoperimetric quotient. Given a superpixel S_j, the perimetric quotient relates the area $A(S_j)$ of the superpixel to the area of a circle with the same perimeter $U(S_j)$:

$$\frac{4\pi A(S_j)}{U(S_j)^2}. \tag{6.6}$$

As the circle represents the most compact form, the perimetric quotient measures the compactness of the superpixel, reaching 1 if and only if the superpixel has the shape of a circle. The proposed compactness measure considers the perimetric quotient of all superpixels weighted by their area:

$$CO(S) = \frac{1}{N}\sum_{S_j \in S} |S_j| \frac{4\pi A(S_j)}{U(S_j)^2}. \tag{6.7}$$

Although we assume superpixels to represent connected components, in practice, this will not always be the case. Therefore, we need to enforce connectivity before being able to compute the perimeter of the superpixels. Then, the above compactness measure captures the notion of spatial coherence within superpixels. The need of being able to measure compactness is also supported by Ren and Malik who introduce the concept of superpixels as "local" and "coherent" [RM03].

6.3.5 Sum-of-Squared Error

To measure the quality of the superpixel segmentation without being dependent on a ground truth, we propose to use the Sum-of-Squared Error as used for clustering evaluation as well:

$$SSE(S) = \frac{1}{N} \sum_{S_j \in S} \sum_{x_n \in S_j} d(x_n, S_j)^2 \tag{6.8}$$

where $d(x_n, S_j)$ can be an arbitrary distance. In our case the euclidean distance in color space is suitable:

$$d(x_n, S_j) = \|I(x_n) - I(S_j)\|_2. \tag{6.9}$$

By coloring each pixel according to the corresponding superpixel's mean color, the superpixel segmentation is interpreted as reconstruction of the original image and the Sum-of-Squared Error measures the reconstruction error.

6.3.6 Explained Variation

Another measure not depending on a ground truth segmentation is the Explained Variation. The Explained Variation quantifies how well the color variation within the image is captured by the superpixel segmentation [FCT+14]:

$$EV(S) = \frac{\sum_{S_j \in S} (I(S_j) - \mu)^2}{\sum_{n=1}^{N} (I(x_n) - \mu)^2} \tag{6.10}$$

where μ is the mean color of the image. The information carried by an image is primarily defined by variation in color or intensity. Thus, Explained Variation measures the fraction of information captured by the superpixel segmentation.

Chapter 7

Evaluation and Comparison

After having discussed many superpixel algorithms, some of them in detail, we are about to examine their performance in terms of the measures described in the previous chapter. In particular, we are interested in the performance increase when using depth information as additional cue. Furthermore, we compare our implementation of **SEEDS** [vdBBR$^+$12], and its extensions to depth, to the original implementation as well as other state-of-the-art approaches.

To ensure a fair comparison, in the first step, we choose the optimal parameters for each algorithm with respect to Boundary Recall and Undersegmentation Error on the validation set of the BSDS500 and the training set of the NYUV2. Then, we use the test sets of both datasets to conduct a thorough comparison based on all additional measures.

7.1 Evaluation

Due to time constraints, we are not able to search the full parameter space of each superpixel algorithm. Therefore, we base our experiments on parameter values which are reported to work well throughout the literature and examine the influence of variations from these values. The experiments presented in this section are results obtained on the validation set of the BSDS500 and the training set of the NYUV2 and are also available in tabular form in appendix B.

We choose to optimize the parameters of all superpixel algorithms with respect to Boundary Recall and Undersegmentation Error as these are the most commonly used measures throughout the literature. We plot both Boundary Recall and Undersegmentation Error against the number of superpixels. While Boundary Recall represents the fraction of boundary pixels within a ground truth segmentation which are correctly detected by the superpixel segmentation, Undersegmentation Error measures the leakage of superpixels with respect to the ground truth segmentation. Both on the BSDS500, depicted in blue, as well as on the NYUV2, depicted in red, we can expect superpixel algorithms to reach nearly perfect Boundary Recall, that is 100% of the boundary pix-

τ	$A_{\min}$	*Rec*	*UE*	*K*
50	5	1	$2.69 \cdot 10^{-2}$	1,080
50	**10**	**0.99671**	**0.030257**	**757**
50	25	0.99	$3.73 \cdot 10^{-2}$	533
100	**5**	**0.9922**	**0.035088**	**660**
100	**10**	**0.98974**	**0.037503**	**446**
100	**25**	**0.98255**	**0.042693**	**286**

τ	$A_{\min}$	*Rec*	*UE*	*K*
50	**10**	**0.99489**	**0.078983**	**1812**
50	**25**	**0.99271**	**0.091453**	**1261**
50	**50**	**0.987**	**0.10616**	**911**
100	10	0.99	0.1	1,079
100	**25**	**0.98222**	**0.1096**	**762**
100	**50**	**0.97519**	**0.12007**	**565**

Table 7.1: Some of the evaluation results of **FH** for the BSDS500 (left) and the NYUV2 (right). $A_{\min}$ denotes the minimum size of the superpixels enforced in a post-processing step and τ denotes the threshold of equation (2.9). Further, *Rec* denotes the Boundary Recall, *UE* denotes the Undersegmentation Error and *K* the number of superpixels. The parameters chosen for the final comparison are highlighted.

els within the ground truth segmentation are correctly detected. However, with about 10%, Undersegmentation Error on the NYUV2 is usually higher than on the BSDS500 where it is possible to reach an Undersegmentation Error of only 2%. We revisit all superpixel algorithms as introduced in chapter 2 while placing a focus on **SEEDS**.

NC – Superpixels from Normalized Cuts [RM03]. The implementation[1] as used in [MREM04] and [Mor05] is provided by Mori. The algorithm is based on an edge map computed by the contour detector described in [AMFM11]. There are several parameters available, however, as of the high runtime we choose to use the algorithm as provided. As we do not change any parameters, results are available for comparison in the next section.

FH – Felzenswalb & Huttenlocher [FH04]. Felzenswalb and Huttenlocher provide a C implementation[2] of their algorithm. However, the algorithm does not allow to control the number of superpixels directly. Instead, the implementation offers mainly two parameters: the threshold τ of equation (2.9) and the minimum superpixel size $A_{\min}$ which is enforced in a post-processing step. Evaluating **FH** imposes a challenge in that the number of superpixels depends on both, the threshold and the minimum superpixel size. Therefore, we evaluated the algorithm for $\tau \in \{15, 25, 50, 100, 150\}$ and $A_{\min} \in \{5, 10, 25, 50\}$. The results were sorted by the number of superpixels and optimal parameters where chosen as to obtain roughly the desired number of superpixels. Parts of the results are shown in table 7.1 where the final parameters are highlighted. However, the number of superpixels strongly depends on the images such that the average number of superpixels on the test sets used for the comparison may be different.

QS – QuickShift [VS08]. The implementation[3] of **QS** is part of the VLFeat Library [VF08]. The algorithm does not offer direct control over the number of superpixels. The provided parameters are: the weight α controlling the color influence in equation (2.11), the kernel size R used to compute the density $p(x_n)$ and the maximum distance $d_{\max}$ which controls the maximum distance between two pixels x_n and x_m such

[1] Available at `http://www2.cs.sfu.ca/~mori/research/superpixels/`.

[2] Available at `http://cs.brown.edu/~pff/segment/`.

[3] Available at `http://www.vlfeat.org/overview/quickshift.html`.

R	$d_{\max}$	Rec	UE	K
1	**7**	**0.99238**	**0.027951**	**911**
1	**9**	**0.98342**	**0.035874**	**540**
1	**10**	**0.97704**	**0.040115**	**451**
3	**7**	**0.97085**	**0.037697**	**424**
3	9	0.96	$4.35 \cdot 10^{-2}$	316
3	**10**	**0.95031**	**0.046707**	**265**

R	$d_{\max}$	Rec	UE	K
1	**7**	**0.98678**	**0.08052**	**1708**
1	**9**	**0.9694**	**0.10059**	**950**
1	**10**	**0.95891**	**0.11026**	**727**
2	**7**	**0.97378**	**0.091721**	**1138**
2	9	0.95	0.11	730
2	10	0.94	0.12	600
3	7	0.95	0.11	753
3	**9**	**0.93183**	**0.12117**	**554**
3	10	0.92	0.13	480

Table 7.2: Some of the evaluation results of **QS** for the BSDS500 (left) and the NYUV2 (right). All results were obtained with $\alpha = 0.75$. Further, R is the kernel size, $d_{\max}$ is the maximum distance, Rec denotes the Boundary Recall, UE denotes the Undersegmentation Error and K the number of superpixels. The parameters chosen for the final comparison are highlighted.

that pixel x_n may be assigned to pixel x_m during mode seeking. Unfortunately, all three parameters have influence on the number of superpixels. Our approach is similar to the one presented for **FH**: We evaluated the algorithm for $\alpha \in \{0.25, 0.5, 0.75\}$, $R \in \{1,2,3,5,7\}$ and $d_{\max} \in \{5,7,9,10\}$ and sorted the results by the number of superpixels. Overall, table 7.2 shows a part of the results where we fixed $\alpha = 0.75$ and the final parameters are highlighted. We note that the number of superpixels may vary depending on the images.

TP – Turbopixels [LSK+09]. The implementation[4] provided by the authors allows to choose the number of superpixels desired. However, no other parameters are available. Results are available as part of the comparison in the next section.

SLIC – Simple Linear Iterative Clustering. [ASS+10]. We evaluated the implementation[5] provided by Achanta et al. and the implementation available as part of the VLFeat Library. To avoid confusion and following [NP12], we call the original implementation **oriSLIC**, while referring to the implementation of the VLFeat Library as **vlSLIC**. Both implementations allow to control the number of superpixels. Addi-

[4] Available at `http://www.cs.toronto.edu/~babalex/research.html`.
[5] Available at `http://ivrg.epfl.ch/research/superpixels`.

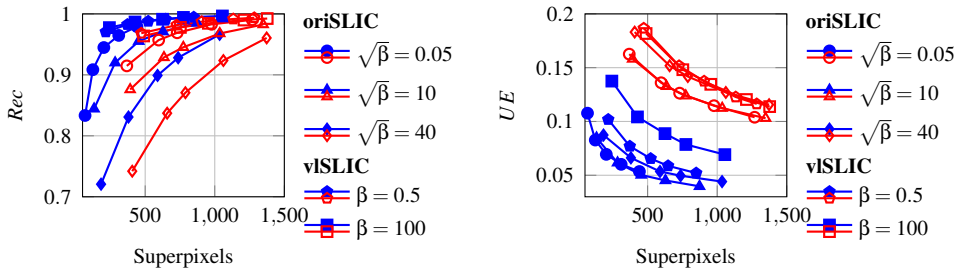

Figure 7.1: Evaluation results of **oriSLIC** and **vlSLIC** for the BSDS500 (blue) and the NYUV2 (red). The algorithm was evaluated for different values of $\sqrt{\beta}$ and β.

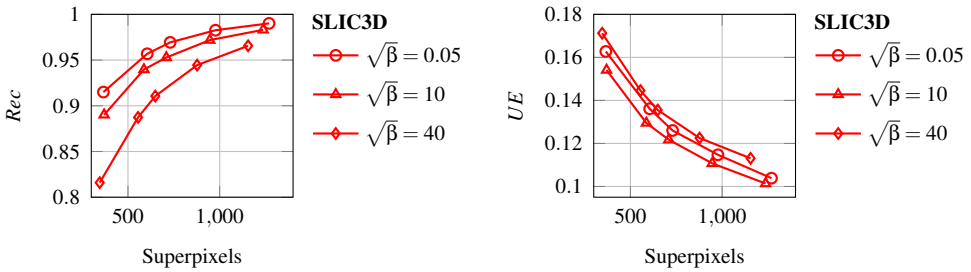

Figure 7.2: Evaluation results of **SLIC3D** for the NYUV2 (red). Different values for $\sqrt{\beta}$ with β from equation (4.2) are shown.

tionally, the compactness is controllable by adapting $\sqrt{\beta}$ and β, respectively, with β from equation (3.19). Figure 7.1 illustrates the influence of the compactness parameter on the achieved Boundary Recall and Undersegmentation Error. Especially on images from the NYUV2, compactness needs to be traded off against Boundary Recall. Due to time constraints and to keep the final comparison uncluttered, we concentrate on **oriSLIC** and choose $\sqrt{\beta} = 10$ for the BSDS500 and $\sqrt{\beta} = 0.05$ for the NYUV2.

SLIC3D. We implemented **SLIC3D** based on the original implementation as to keep **SLIC3D** comparable to **oriSLIC**. However, we found that performance is nearly equal to **oriSLIC**, see figure 7.2. Nevertheless, in contrast to **oriSLIC**, we choose $\sqrt{\beta} = 10$. This change in the parameter $\sqrt{\beta}$ may be due to the additional depth information such that enforcing compactness on the three dimensional surfaces is beneficial. However, this may also be due to the normalization which we kept equal to the one used for **oriSLIC**.

CIS – Constant Intensity Superpixels [VBM10]. The implementation[6] of **CIS** is based on grayscale images and edge maps[7] and offers indirectly control over the number of superpixels by choosing the size R of the overlapping squares. In practice these

[6] Available at `http://www.csd.uwo.ca/~olga/Projects/superpixels.html`.

[7] The implementation allows to adjust the weights by providing edge maps. We chose to use gradients in x and y direction in order to adjust the vertical and horizontal weights, however, do not use edge maps to adjust the diagonal weights.

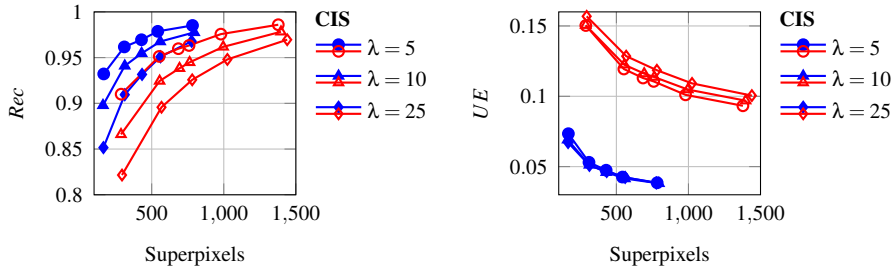

Figure 7.3: Evaluation results of **CIS** for the BSDS500 (blue) and the NYUV2 (red). Different values for λ from equation (2.17) are shown.

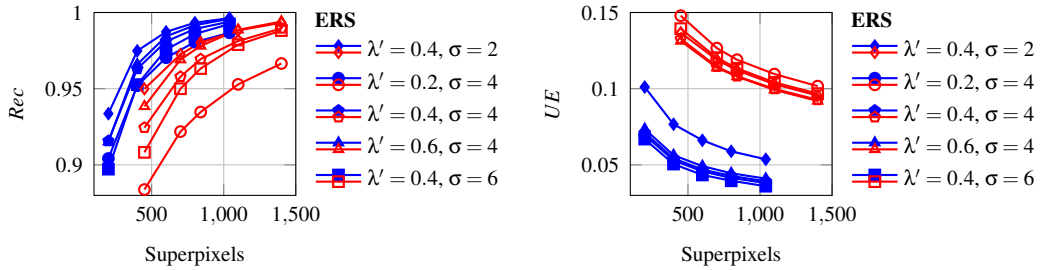

Figure 7.4: Evaluation results of **ERS** for the BSDS500 (blue) and the NYUV2 (red). Different values for λ' and σ are shown.

squares are placed on a regular grid with step size $\frac{R}{2}$. However, we note that the number of computed superpixels is persistently lower than the number of desired superpixels. Figure 7.3 shows the results of **CIS** for different choices of λ from equation (2.17) and suggests that choosing $\lambda < 5$ results in increased performance, however, due to time constraints, we choose $\lambda = 5$ (Veksler et al. choose $\lambda = 10$ as default value).

ERS – Entropy Rate Superpixels [LTRC11]. The implementation[8] of **ERS** offers direct control over the number of superpixels. Note that **ERS** ensures that the desired number of superpixels is met exactly. In addition the implementation allows to adapt the balancing parameter λ of equation (2.18) and the parameter σ used within the Gaussian Kernel to convert color distances to similarities. However, Lui et al. use an automatic procedure to choose λ based on a user-chosen parameter λ', see [LTRC11] for details. Unfortunately, the parameters λ' and σ need to be traded off against each other to simultaneously optimize Boundary Recall and Undersegmentation Error. Figure 7.4 shows the results for different choices of λ' and σ. Especially on images from the BSDS500, parameters resulting in a high Boundary Recall, show the highest Undersegmentation Error, therefore we choose $\lambda' = 0.4$ and $\sigma = 4$. These parameters result in reasonable performance and are comparable to the values used in [LTRC11].

[8] Available at `https://sites.google.com/site/seanmingyuliu/home/research_segmentation`.

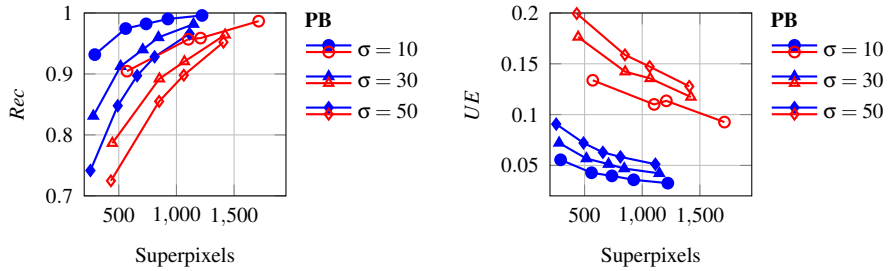

Figure 7.5: Evaluation results of **PB** for the BSDS500 (blue) and the NYUV2 (red). Different values for σ were evaluated. For $\sigma \to 1$, the number of superpixels reaches more than triple the number of desired superpixels while not further improving performance.

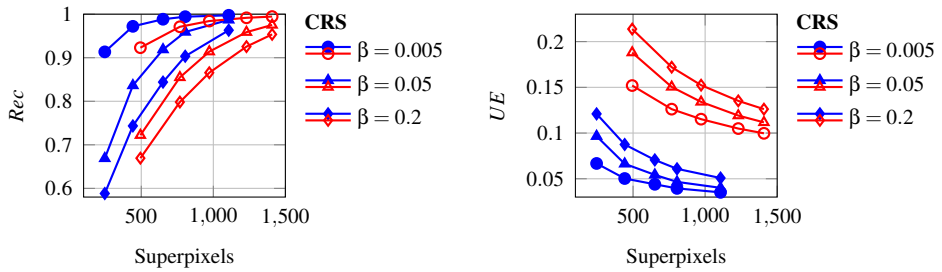

Figure 7.6: Evaluation results of **CRS** for the BSDS500 (blue) and the NYUV2 (red). Different values of the compactness parameter β are shown.

PB – Superpixels via Pseudo-Boolean Optimization [ZHMB11a]. The implementation[9] offers two approaches for obtaining a superpixel segmentation from the created graph: max-flow and elimination. We found that, overall, the max-flow based **PB** performs better with respect to both Boundary Recall and Undersegmentation Error. The implementation does allow to choose the parameter σ used within the Gaussian kernel to convert color distances to similarities. Furthermore, **PB** offers indirect control over the number of superpixels by choosing the height and width of the horizontal and vertical strips. Figure 7.5 shows results for different values of σ. Although not shown in the figure, we note that with $\sigma \to 1$, the number of superpixels reaches more than the triple of the desired number of superpixels. Therefore, we exclude results for $\sigma < 10$. We choose $\sigma = 10$ and note that even then, for images from the NYUV2, the number of superpixels oversteps the number of desired superpixels.

CRS – Contour Relaxed Superpixels [CMM13]. The implementation[10] provided by Conrad et al. offers indirect control over the number of superpixels by adapting the step size of the initial superpixel segmentation. Additionally, the implementation offers to adapt the compactness in the means of the parameter β. Our experiments show that the clique costs C_v and C_e of equation (2.29) do not have significant influence on the performance. In practice, the diagonal clique cost is derived as

$$C_v = \frac{C_e}{\sqrt{2}} \tag{7.1}$$

and we set $C_e = 0.1$. Additionally, we use $T = 2$ iterations as the number of iterations has only negligible influence on the performance. Figure 7.6 shows the Boundary Recall and Undersegmentation Error for different values of β. We trade off compactness for both Boundary Recall and Undersegmentation Error and choose $\beta = 0.005$.

SEEDS – Superpixels Extracted via Energy-Driven Sampling [vdBBR+12]. Beneath our implementation, referred to as **reSEEDS**, the original implementation[11],

[9] Available at `http://yuhang.rsise.anu.edu.au/yuhang/misc.html`.

[10] Available at `http://www.vsi.cs.uni-frankfurt.de/research/current-projects/superpixel-segmentation/`.

[11] Available at `http://www.mvdblive.org/seeds/`.

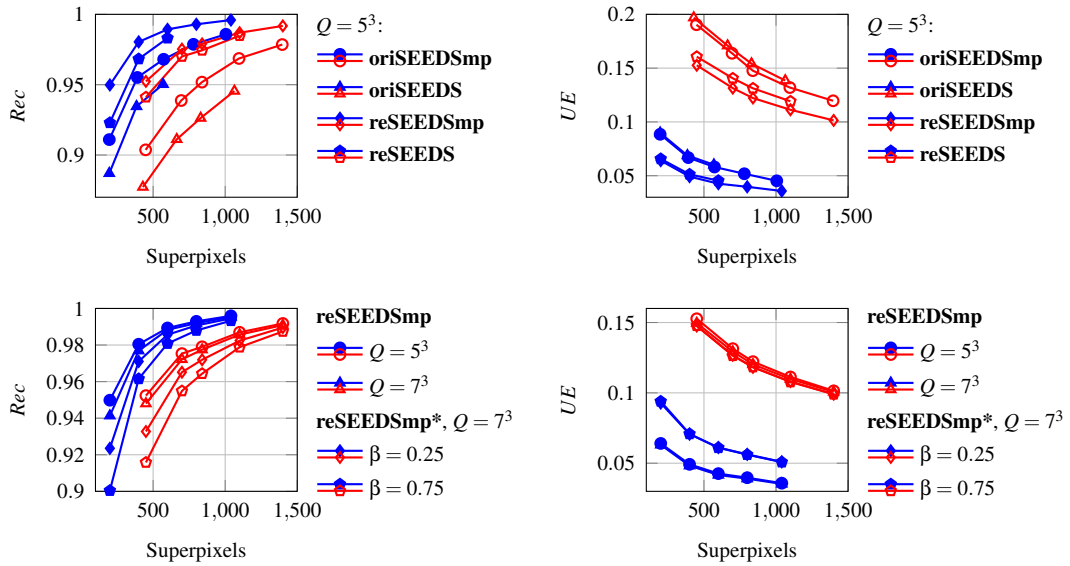

Figure 7.7: Evaluation results of **oriSEEDS** and **reSEEDS** for the BSDS500 (blue) and the NYUV2 (red). Top: the advantage of using mean pixel updates for both implementations. Bottom: the influence of the histogram size Q on the performance and the effect of increasing the compactness parameter β of equation (3.8). All results were obtained using $T = 2$.

referred to as **oriSEEDS**, is available. Both perform $2T$ iterations of pixel updates and offer to use the original smoothing term of equation (3.6). As this smoothing term does not increase runtime[12] and experiments suggest that it increases performance, see appendix B, we include the smoothing term in all our experiments. We evaluated several variants of our and the original implementation: **oriSEEDSmp** refers to the original implementation using mean pixel updates and **reSEEDSmp** refers to our implementation using mean pixel updates. Then, for $T = 2$, the advantage of using mean pixels updates as well as the influence of the histogram size Q is shown in figure 7.7. We note that for $Q \geq 3^3$, performance changes only slightly such that Q can be chosen small if runtime is crucial. An additional variant of our implementation is given by using the alternative smoothing term of equation (3.8), which we refer to as **reSEEDSmp***[13]. We note, that performance decreases slightly when increasing the compactness parameter β of equation (3.8) and that **reSEEDSmp** outperforms **reSEEDSmp*** when choosing β to be significantly greater than zero. Overall, the influence of Q and the advantage of using mean pixel updates can also be observed for the original implementation, see appendix B, however, our implementation outperforms the original one. For both implementations, to minimize Undersegmentation Error, we choose $Q = 7^3$ and

[12] Actually, the original implementation offers a hard coded implementation of the smoothing term to use 3×4 and 4×3 neighborhoods, respectively. In contrast, our implementation provides a generalized implementation allowing to use $N_R(x_n, x_m)$ for arbitrary R.

[13] Note that in practice the spatial term of equation (3.8) is normalized by the width and height of the image.

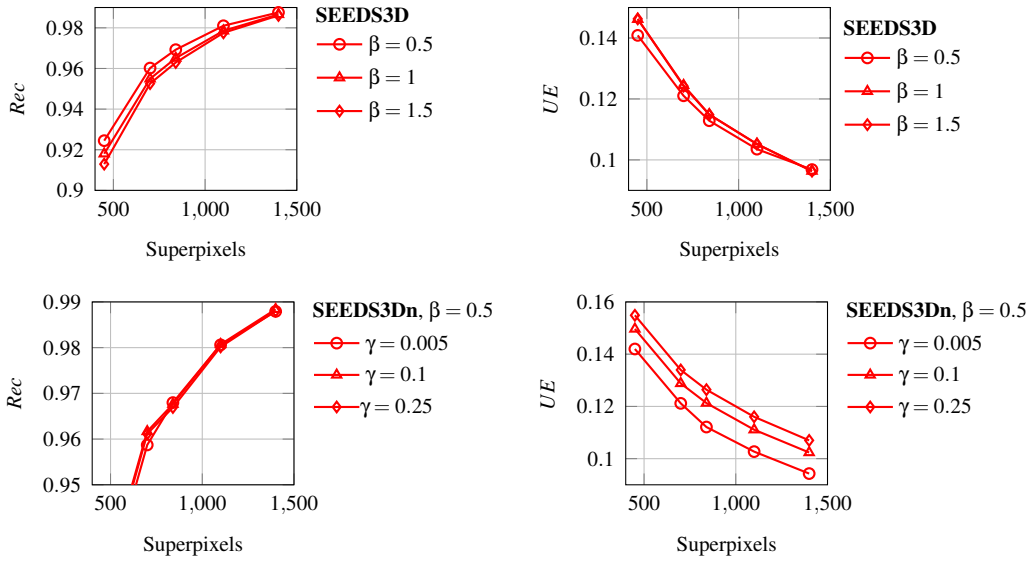

Figure 7.8: Evaluation results of both **SEEDS3D** and **SEEDS3Dn** for the NYUV2. Different values of β and γ have been evaluated.

for **reSEEDSmp*** set $\beta = 0.25$.

SEEDS3D. We introduced two additional variants of **SEEDS**: **SEEDS3D** using 3D point coordinates for pixel updates and **SEEDS3Dn** additionally using normal information. As of equation (5.2), there are two additional parameters to be tweaked: β, weighting the influence of 3D point coordinates, and γ, weighting the normal information[14]. Figure 7.8 shows the performance for several choices of β and γ. We observe that normal information should be used sparingly in order not to decrease performance. Overall, we conclude that using normal information offers only negligible performance increase such that we concentrate on **SEEDS3D** for comparison.

DASP – Depth-Adaptive Superpixels [WGB12]. The implementation[15] published by Weikersdorfer et al. offers control over the number of superpixels and additionally provides several methods for sampling superpixel centers from the superpixel density. As all evaluated methods show no significant improvement over random sampling, we keep the sampling method constant to reduce the number of parameters. Additionally, the implementation offers control over the weights β and γ of equation (4.5). Figure 7.9 shows the results for different values of β and γ. Note the influence of γ on the number of superpixels. Furthermore, we observe that increasing β may worsen Boundary Recall while improving Undersegmentation Error. Overall, we choose $\beta = 0.05$ and $\gamma = 1.5$.

[14] In practice, concerning the used distance function, the color term is normalized by $\sqrt{3} \cdot 255$ as we use 8 bit channels, the spatial term is normalized by the maximum and minimum 3D point coordinates and the normal term is normalized by π.

[15] Available at `https://github.com/Danvil/dasp`.

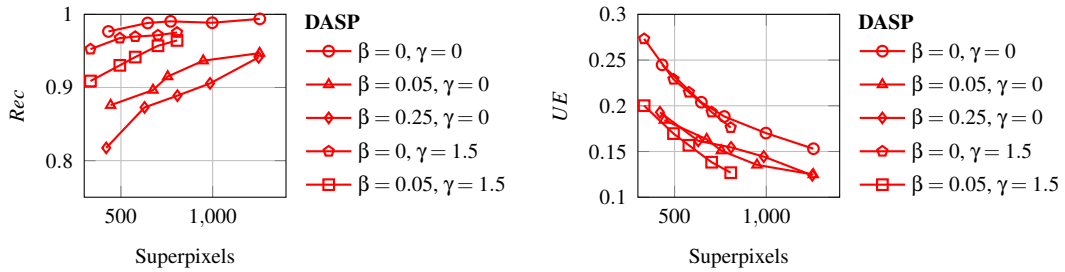

Figure 7.9: Evaluation results of **DASP** for the NYUV2. The performance for different values of β and γ is shown.

TPS – Topology Preserved Superpixels [TFC12]. An implementation[16] of **TPS** is available from the authors. As edge detector, the implementation uses the contour detector described in [AMFM11]. No parameters are provided, however, the number of superpixels is controllable. Results are available during comparison in the next section.

VCCS [PASW13]. An implementation of **VCCS** is provided as part of the Point Cloud Library[17] [RC11]. As **VCCS** operates directly on point clouds, it does not offer exact control over the number of superpixels. However, the average number of superpixels computed can be controlled by adapting the step size R. Furthermore, the user is allowed to adapt the weights β and γ of equation (4.7), weighting the spatial and normal term, respectively. At this point we note that 3D point coordinates are already taken into account as **VCCS** performs breadth-first search on the voxelized point cloud. Consequently, small changes of β have little influence. Figure 7.10 summarizes the results. We choose $\beta = 0$ and $\gamma = 1$ to optimize both Boundary Recall and Undersegmentation Error.

[16] Available as code of [FCT+14] here: `https://sites.google.com/site/huazhufu/`.

[17] Available at `http://pointclouds.org/documentation/tutorials/supervoxel_clustering.php`.

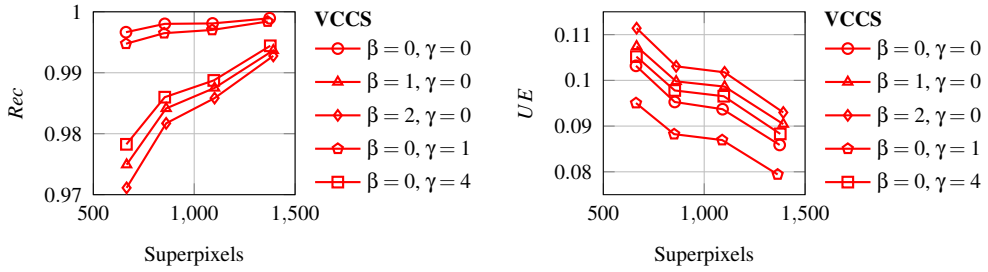

Figure 7.10: Evaluation results of **VCCS** for the NYUV2 showing the influence of β and γ.

7.2 Comparison

The comparison of the superpixel algorithms of the previous section is split into several parts. The first part investigates the visual appearance and quality of the generated superpixel segmentations. Then, a quantitative comparison is given using the measures introduced in chapter 6. Finally, we consider the runtime of the superpixel algorithms in order to give a complete impression of each superpixel algorithm.

7.2.1 Qualitative

The visual appearance of a superpixel segmentation is primarily determined by the regularity and the compactness of the superpixels. While compactness refers to the form of the superpixels, regularity corresponds to the positioning and the size of the superpixels. Together, regular and compact superpixels are regarded as visually appealing. Based on the test sets of the BSDS500 and the NYUV2, figure 7.12 shows the superpixel segmentations generated by several variants of **SEEDS**, while figure 7.11 shows the superpixel segmentations generated by the remaining superpixel algorithms not depending on depth and figure 7.13 shows the superpixel segmentations obtained from algorithms utilizing depth information. Note that these images only resemble a subset of the test sets such that more images need to be analyzed to come to a final conclusion, see appendix B. Furthermore, the parameters have been chosen to result in optimal Boundary Recall and Undersegmentation Error while the appearance was secondary.

NC. Superpixels from Normalized Cuts are relatively smooth when compared to other approaches, however, tend to be highly elongated in order to adhere to boundaries. Mostly, strong boundaries are met well, but smaller objects and details are missed.

FH. Superpixels generated by **FH** cannot be described as regular or compact. However, they are able to adapt even to small details. This leads to a good overall boundary adherence. In addition, we often observe collapsed superpixels within relatively large superpixels.

QS. Similar to **FH**, **QS** generates superpixels irregular in size and shape. However, superpixels are able to capture little details. Furthermore, we notice that the number of superpixels adapts to the image content in that more superpixels are found at strongly textured regions.

TP. A strong focus on compactness often results in poor boundary adherence. This can be observed when considering superpixels generated by **TP**. In particular, objects or details smaller than the initial regular grid are rarely captured well. However, when increasing the number of superpixels, boundary adherence may be combined with very compact superpixels.

SLIC is one of the few algorithms offering direct control over the compactness. Therefore, using **SLIC** may result in visually appealing, regular and compact superpixels. However, the compactness needs to be traded off against Boundary Recall and

Figure 7.11: Qualitative results for several evaluated superpixel algorithms illustrated on two images form the BSDS500 and three images from the NYUV2. From top to bottom: **NC**, **FH**, **QS**, **TP**, **oriS-LIC**, **CIS**, **ERS**, **PB**, **CRS**, **TPS**. For images from the BSDS500, approximately 600 superpixels are shown; for images from the NYUV2 approximately 840 images are shown. Further examples can be found in appendix B.

Figure 7.12: Qualitative results for all implementations of **SEEDS** illustrated on two images form the BSDS500 and three images from the NYUV2. From top to bottom: **oriSEEDS**, **oriSEEDSmp**, **reSEEDS**, **reSEEDSmp**, **reSEEDSmp***. For images from the BSDS500, approximately 600 superpixels are shown; for images from the NYUV2 approximately 840 images are shown. Further examples can be found in appendix B.

Undersegmentation Error. As we optimized the parameters with respect to Boundary Recall and Undersegmentation Error, we get relatively irregular superpixels at nearly uniformly colored regions but on the other hand are able to capture small details. This gets apparent on images from the NYUV2.

SLIC3D. Superpixels generated by **SLIC3D** are similar to those generated by **SLIC**. On some images, the superpixels appear relatively compact on nearly uniformly colored regions while on others we can observe highly irregular superpixels. Overall, the superpixels are able to adapt to comparably small details and lighting changes.

CIS. As **CIS** uses gray scale images only, some boundaries are inevitable missed. Additionally, **CIS** gets easily challenged by small details as well as bad lighting. However, the superpixels appear to be relatively compact and regular when compared to other approaches not offering direct control over the compactness. Some of these disadvantages may be resolved when extending **CIS** to use color images.

ERS. Superpixels generated by **ERS** show a good boundary adherence. This is partly due to the large variation in size: **ERS** superpixels are able to adapt their size to capture small details. At nearly uniformly colored regions, the superpixels tend to be irregular and are not visually appealing anymore.

Figure 7.13: Qualitative results for superpixel algorithms utilizing depth. From top to bottom: **SLIC3D**, **SEEDS3D**, **SEEDS3Dn**, **DASP**, **VCCS**. For images from the BSDS500, approximately 600 superpixels are shown; for images from the NYUV2 approximately 840 images are shown. Further examples can be found in appendix B.

PB. Superpixels generated by **PB** appear to be unfinished and of poor quality. Often, the superpixels tend to be too regular and compact such that even strong boundaries are not captured well. In particular, this can be observed for images taken from the NYUV2. Even strong changes in color do not affect the initial superpixel segmentation given by a regular grid.

CRS. Similar to **SLIC**, **CRS** offers direct control over the compactness. As we optimized the parameters with respect to Boundary Recall and Undersegmentation Error, the generated superpixels do not appear compact, especially on images from the NYUV2. However, this enables superpixels to capture small details and achieve reasonable boundary adherence. Increasing the number of superpixels as well as the compactness parameter will result in visually more appealing superpixels with similar performance.

SEEDS. Independent of the implementation, using **SEEDS** without mean pixel updates (that is **oriSEEDS** and **reSEEDS** for the original and our implementation, respectively) results in highly irregular superpixels which are mostly not compact. However, especially on images from the BSDS500, the superpixel segmentations show good boundary adherence. On images from the NYUV2, the original implementation shows poor boundary adherence compared to our implementation. Using mean pixel

updates (that is **oriSEEDSmp** and **reSEEDSmp** for the original and or implementation, respectively) results in improved boundary adherence. The superpixels are able to capture small details at the cost of regularity and compactness. At this point, using the alternative smoothing term of equation (3.8), referred to as **reSEEDSmp***, generates more compact as well as regular superpixels. Nevertheless, these superpixels are able to adapt to small details and boundaries.

SEEDS3D. Superpixels generated by **SEEDS3D** appear to be similar to those obtained from **reSEEDSmp***. However, the superpixels appear to resemble the underlying three dimensional structure better. Additionally, the superpixels are adaptive to small changes in color as well as lighting changes. Unfortunately, we observe collapsed superpixels especially at nearly uniformly colored regions. **SEEDS3Dn** appears to worsen the results, at least concerning the visual quality of the superpixels. As of the negligible performance improvement of **SEEDS3Dn** over **SEEDS3D**, we further concentrate on **SEEDS3D**.

DASP. Superpixels generated by **DASP** are overall not visually appealing. However, note that **DASP** allows to adjust the compactness of the superpixels. In addition, the algorithm is not robust to noise within the depth image. This can be observed at regions where depth has been filled, especially towards the borders. Often strong boundaries are not captured well, although the superpixels are allowed to vary in size and shape.

TPS. Superpixels generated by **TPS** should resemble a regular topology. However, often the superpixel segmentation appears to be of poor quality as we are able to think of better topology preserving superpixel segmentations [TFC12] adhering to more of the strong boundaries present in the images. On images from the NYUV2 this may be due to the fact that the underlying contour detector [AMFM11] has been trained on the BSDS500, however, even on images from the BSDS500 the quality is relatively poor.

VCCS. As **VCCS** is designed to work directly on point clouds, the generated supervoxels can best be observed on the oversegmented point clouds. Figure 7.13 merely shows the backprojection of the oversegmented point cloud. This means, that the effect of voxelization is still visible. As we chose to optimize Boundary Recall and Undersegmentation Error, the supervoxels are irregular and not compact. However, this can be controlled by the provided compactness parameter. The supervoxels align well with most of the visible boundaries, and are able to capture small details.

7.2.2 Quantitative

The quantitative evaluation is based on the measures provided in section 6.3. Although Boundary Recall and Undersegmentation Error may give a good idea of the performance of an algorithm, additional measures are necessary to complement the impression. Therefore, we use Achievable Segmentation Accuracy, Compactness, Sum-of-Squared Error as well as Explained Variation. We note that due to the prohibitive runtime of **NC**, the results obtained on the smaller validation set of the BSDS500 and training set of NYUV2 are used. Unfortunately, we were not able to evaluate **CIS** and

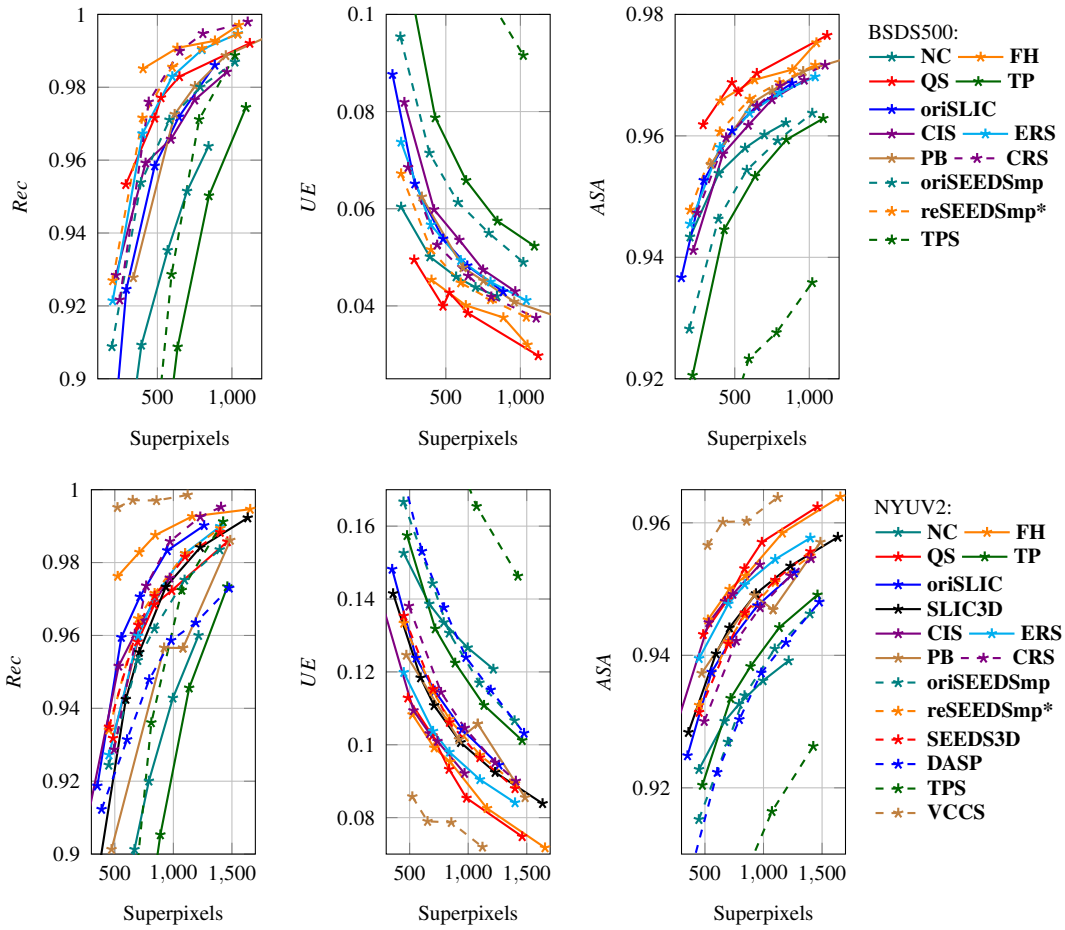

Figure 7.14: Final comparison of all superpixel algorithms with respect to Boundary Recall (*Rec*), Undersegmentation Error (*UE*) and Achievable Segmentation Accuracy (*ASA*) on the BSDS500 and the NYUV2. Note that for visualization purposes only a small part of the range of all measures is shown, for exact numbers we refer to appendix B.

TPS with respect to the Sum-of-Squared Error and Explained Variation. All results are also available in tabular form in appendix B.

Boundary Recall, Undersegmentation Error, Achievable Segmentation Accuracy – Figure 7.14. First of all, we observe that our implementation of **SEEDS**, **reSEEDS**, outperforms the original implementation, **oriSEEDS**, with respect to all three measures. Although, our implementation uses the alternative smoothing term of equation (3.8), our experiments in the previous sections suggest that this can be generalized. Furthermore, our implementation is comparable to **oriSLIC**, outperforming **oriSLIC** on the BSDS500 and lying head-to-head on the NYUV2. On the BSDS500, our implementation is consequently outperformed only by **FH**. In contrast, on the NYUV2, **VCCS** and also **ERS** outperform our implementation as well. The observed performance increase when using **SEEDS3D** or **SLIC3D** is negligible suggesting that using depth information does not further improve performance at this level. This suggests

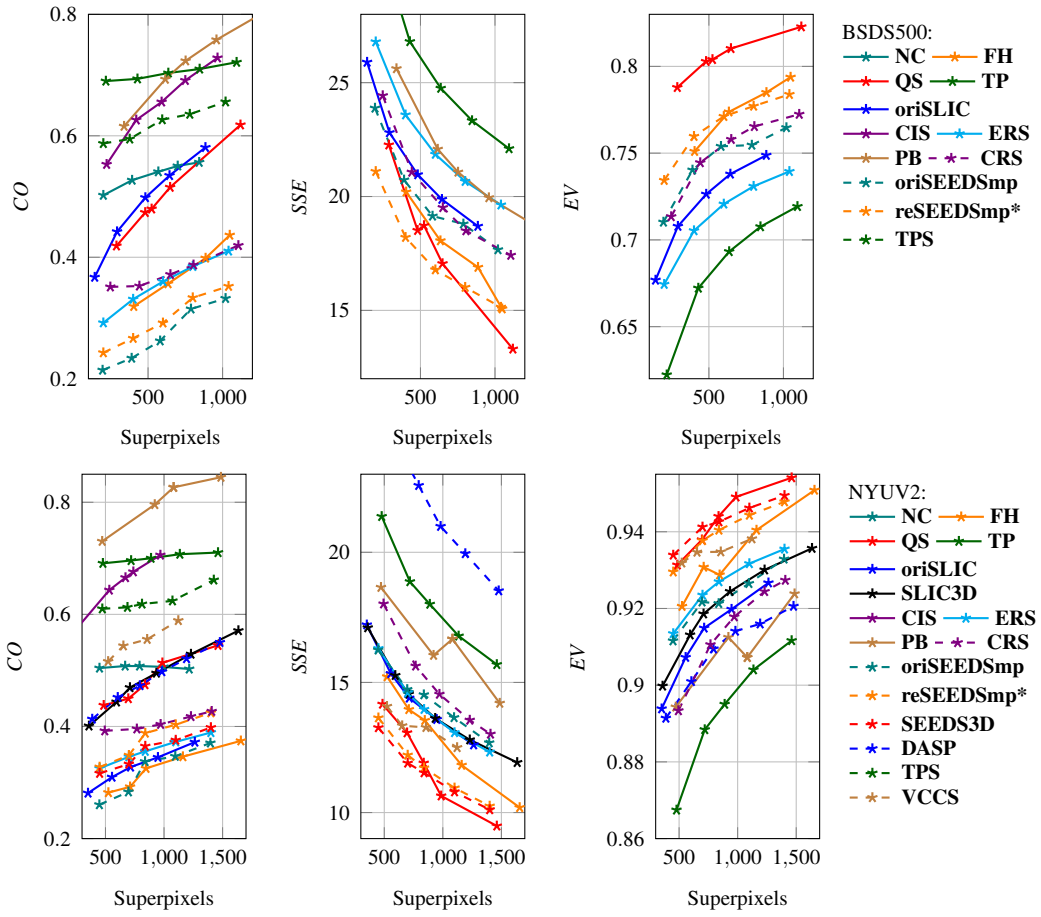

Figure 7.15: Final comparison of all superpixel algorithms with respect to Compactness, the Sum-of-Squared Error and Explained Variation on the BSDS500 and the NYUV2. Note that for visualization purposes only a part of the range of all measures is shown. We refer to appendix B for details.

that the excellent performance achieved by **VCCS** is a result of oversegmenting point clouds instead of images with depth information.

On the BSDS500, **FH** performs excellent, often lying ahead together with **reSEEDSmp*** and **QS**. On the NYUV2, **FH** gets outperformed only by **VCCS**, offering excellent performance with respect to all measures and always being slightly ahead of the other algorithms. Further, we are able to identify solid performance of **CIS** as well as **CRS** lying close to **oriSLIC** and **reSEEDSmp***. Concerning Undersegmentation Error and Achievable Segmentation Accuracy on the NYUV2, **CIS** even outperforms **oriSLIC** and **reSEEDSmp***. While **PB** offers adequate performance on the BSDS500, it performs poorly on the NYUV2. This is in accordance with the qualitative results discussed above. The performance of **NC** and **TP** is mostly not comparable to the other algorithms. Surprisingly, **DASP** performs poor as well, especially concerning Boundary Recall where it is outperformed by **NC**. Finally, **TPS** performs worst.

Overall, we conclude that it is appropriate to use Boundary Recall and Underseg-

mentation Error to choose parameters as Achievable Segmentation Accuracy presents similar results to those obtained by Undersegmentation Error. Furthermore, the state-of-the-art, at least on the BSDS500, is hard to identify. Although **FH** outperforms most of the algorithms, approaches like **oriSLIC**, **SEEDS** – especially our implementation –, **ERS**, **CRS**, **QS** and **CIS** are highly competitive. However, the NYUV2 offers a different point of view: clearly, **VCCS** outperforms all other algorithms.

Compactness – Figure 7.15. As already discussed in the course of the qualitative comparison, **TP**, **CIS**, **PB** and **TPS** provide the most compact superpixels on both datasets. Surprisingly, **QS** provides relatively compact superpixels. In our opinion this is the result of many small superpixels in highly textured regions. **SEEDS** provides poor compactness, even though the alternative smoothing term is used. On the other hand, **reSEEDSmp*** as well as **oriSLIC** and **CRS** provide a compactness parameter which may be used to trade Boundary Recall and Undersegmentation Error for Compactness. Another important aspect is the compactness of superpixels from **VCCS** and **DASP**. In both cases the used compactness measure is misplaced as it does only measure compactness on the image plane, while both algorithms enforce compactness within three dimensional space. The same is true for **SEEDS3D** and **SLIC3D** showing poor compactness due to the choice of β.

Sum-of-Squared Error, Explained Variation – Figure 7.15. Surprisingly, these measures do not fully support the conclusion drawn based on the previous measures. For example, **VCCS** is outperformed by **QS**, **FH**, **reSEEDSmp*** and **SEEDS3D**. In particular, **QS** shows excellent performance. In our opinion, this corresponds to the observation made in the qualitative section that more superpixels are generated in highly textured regions. Furthermore, **reSEEDSmp*** shows competitive performance and the influence of using depth in the form of **SEEDS3D** is visible. On the other hand, some of the earlier statements are supported. For example, our implementation of **SEEDS** continuously outperforms the original implementation as well as **oriSLIC**. Furthermore, **TP** and **DASP** perform rather poorly with respect to both measures.

7.2.3 Runtime

As of the previous section, several of the superpixel algorithms are comparable in performance. Therefore, other aspects need to be considered in order to choose the appropriate superpixel algorithm for specific applications. One important property of a superpixel algorithm is its runtime. We note that images from the BSDS500 have $N = 481 \cdot 321 = 154401$ pixels while images from the NYUV2 have $N = 608 \cdot 448 = 272384$ pixels. The results presented in this section where obtained on a 64 bit machine equipped with a Intel Core i7-3770 @ 3.4GHz, a Nvidia GeForce GTX 680 and 16GB RAM. To the best of our knowledge, none of the evaluated algorithms makes use of multi-threading or the graphics card. The timings are given in seconds and represent the average runtime over all images from the test sets of the BSS500 and the NYUV2.

Runtime – Figure 7.16 and table 7.3. We concentrate on algorithms able to provide runtime below $500ms$ as these become interesting for applications requiring efficiency

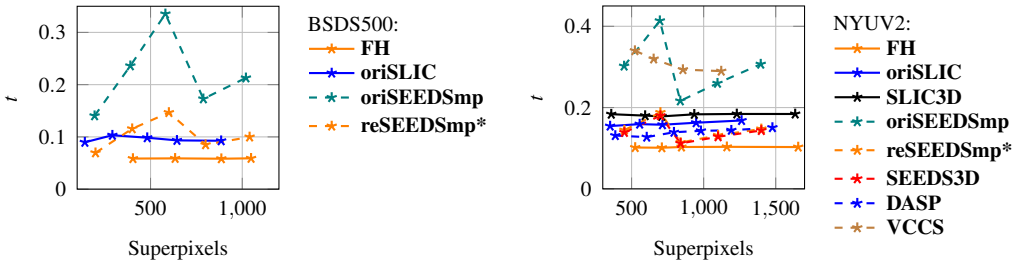

Figure 7.16: Final comparison of the runtime of those superpixel algorithms requiring less than 500ms per image on the BSDS500 and the NYUV2 in seconds. Note that the runtime of **SEEDS** is in practice also influenced by the number of levels L and the block size $w^{(1)} \times h^{(1)}$ resulting in the observed zig-zag-pattern, see section 3.1.2.

and those running in realtime, see figure 7.16. The runtime of the remaining algorithms is shown in table 7.3. We can observe that our implementation of **SEEDS** is consequently faster than the original implementation and comparable in runtime to **oriSLIC** and **DASP**. In addition, we note that the runtime of most of the algorithms is not significantly depending on the number of superpixels. Overall, **FH** can be identified as fastest algorithm, closely followed by our implementation of **SEEDS**, **oriSLIC** and **DASP**. Further we notice that **VCCS** shows comparable runtime to the original implementation of **SEEDS**. However, all algorithms lie above 50ms, especially on the NYUV2, which corresponds to a maximum of $\frac{1}{0.05} = 20Hz$ excluding any additional processing which would be necessary for realtime applications.

As we concluded that none of the algorithms is able to run above $20Hz$, we conducted further experiments including **oriSLIC**, **oriSEEDS**, **reSEEDS** and **SEEDS3D** to identify the minimum possible runtime and the corresponding performance of these implementations. Therefore, figure 7.17 shows the performance and runtime when using only $T = 1$ iteration for the above algorithms while using the results for **FH** from above. Additionally, we reduced the histogram size to $Q = 3^3$. We point out, that our implementation of **SEEDS** using mean pixel updates is able to run at $\sim 30ms$ on images from the BSDS500, while offering state-of-the-art performance. Our implementation is outperformed only by **FH** offering runtime comparable to **oriSLIC** of $\sim 60ms$ per image. Further, we observe that **SEEDS3D** is slightly outperformed by **SEEDSmp*** while providing similar runtime. Finally, **oriSEEDS** performs worst regarding both performance and runtime.

Dataset	K_d	**NC**	**QS**	**TP**	**CIS**	**ERS**	**CRS**	**TPS**
BSDS500	400	250.51	2.2255	5.8005	3.7002	1.0317	0.5808	43.776
	600	332.23	1.0113	5.671	3.7586	1.0568	0.70345	44.659
NYUV2	450	646.72	4.3859	14.182	6.0505	1.973	0.72337	59.273
	700	804.6	1.5681	14.29	6.7361	2.0522	0.90168	59.476

Table 7.3: Final comparison of the runtime of several superpixel algorithms requiring more than 500ms per image on the BSDS500 and the NYUV2 in seconds.

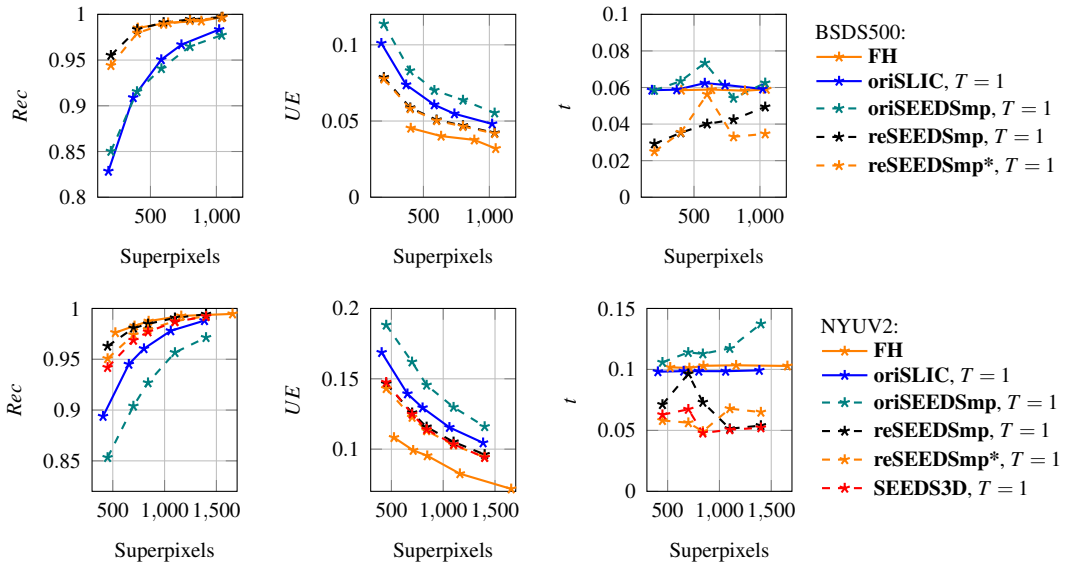

Figure 7.17: Final comparison of the runtime of **SLIC** [ASS+10] and **SEEDS** [vdBBR+12] using $T = 1$ iteration on the BSDS500 and the NYUV2 in seconds. **FH** is shown for comparison only, no changes were made to increase its runtime.

7.2.4 Discussion

We conclude this chapter by assembling the above insights in order to redefine the state-of-the-art in superpixel segmentation with regard to qualitative and quantitative performance as well as runtime. Additionally we consider the available implementations and their ease of use. In particular, we find it important whether the number of superpixels is controllable. Considering all of the above algorithms and their performance, the number of superpixels can be interpreted as lower bound on the performance. Therefore, the number of superpixels offers the most obvious measure of performance and simplifies the integration of superpixels as pre-processing step. Furthermore, for more than $\sim 1000 - 2000$ superpixels, the performance of all evaluated algorithms appears to saturate quickly such that the choice of algorithm is more and more influenced by performance-independent aspects. Nevertheless, the reduction from $N > 10^5$ pixels to $K > 1000$ superpixels enables a significant increase in runtime for subsequent steps.

First of all, **FH** outperforms most of the algorithm on both dataset and provides considerable runtime. However, considering visual quality and the implementation of **FH**, other algorithms are preferable. Especially as the number of superpixels is not controllable and superpixels are highly irregular in size and shape. **FH** may still be used as baseline, however, we find it difficult to choose the available parameters in order to obtain results comparable to other algorithms as the number of computed superpixels is highly varying. This difficulty may also be problematic for applications as the optimal parameters are depending on the actual image content. On the NYUV2,

on the other hand, **VCCS** shows excellent performance, often lying slightly ahead concerning Boundary Recall, Undersegmentation Error and Achievable Segmentation Accuracy. Furthermore, **VCCS** allows to adjust the compactness. However, we find it difficult to choose the voxel resolution and the seed resolution in order to obtain the desired number of superpixels. In practice, this results in problems similar to **FH**. In addition, when aiming to oversegment images, complete depth information as well as the intrinsic camera parameters are required. Another algorithm providing excellent performance on both datasets is **QS**. However, **QS** has the same problems as **FH**: superpixels are highly irregular and the number of superpixels is not controllable as well as strongly depending on the image content.

As of the above considerations, the choice of **SLIC** as baseline in many publications [FCT+14, PASW13, vdBBR+12, CMM13] was appropriate for the last few years as it is easy to use, allows to control the number of superpixels as well as the compactness and in most cases shows state-of-the-art performance and considerable runtime. However, based on the results presented in this chapter, we propose to use our implementation of **SEEDS**, especially **reSEEDSmp***, as new (or additional) baseline for future research. **reSEEDSmp*** clearly outperforms most of the available algorithms, especially when considering runtime. In addition, the number of superpixels is controllable and met exactly. Furthermore, we encourage the use of **ERS** and **CRS**. Both show comparable performance and offer to control the number of superpixels directly – **CRS** also allows to control the compactness. Additionally, **ERS** ensures that the number of superpixels is met exactly.

Chapter 8

Conclusion

As result of this bachelor thesis, we present a new implementation of **SEEDS**, a superpixel algorithm proposed by Van den Bergh et al. [vdBBR+12]. Additionally, we implement an extension proposed in [vdBBR+13] using pixel coordinates as smoothing term to obtain compact and regular superpixels. Furthermore, we experimented with several extensions of **SEEDS** using depth information. In conclusion, we favor two of these approaches: **SEEDS3D** using 3D point coordinates to exchange pixels between superpixels of an initial superpixel segmentation; and **SEEDS3Dn** using normal information as additional cue. As baseline we used another superpixel algorithm, especially popular because of its simplicity, called **SLIC** [ASS+10]. In the course of the thesis we extended **SLIC** to use 3D point coordinates, as well, in order to compare its performance to **SEEDS3D**.

In the course of our experimental evaluation, we implemented most of the available measures commonly used to evaluate superpixel algorithms as extension to the popular Berkeley Segmentation Benchmark [AMFM11]. Among these are the Undersegmentation Error measuring the leakage of superpixels with respect to a ground truth segmentation, as well as a compactness measure proposed by Schick et al. [SFS12]. Utilizing the extended Berkeley Segmentation Benchmark, we compared most of the available superpixel algorithms on the Berkeley Segmentation Dataset [AMFM11] and the NYU Depth Dataset [SHKF12].

In conclusion, we summarize these experiments in three important observations. Firstly, our implementation of **SEEDS** is able to compete with all of the evaluated superpixel algorithms concerning both performance and runtime. Actually, providing near realtime, while achieving state-of-the-art performance, it is one of the fastest algorithms available (competing only with the approach proposed in [FH04], called **FH**, the original implementation of **SEEDS** and **SLIC**). Additionally, the implementation is easy to use: the parameters are easily interpreted and the number of superpixels can be chosen by the user and is met exactly.

Secondly, depth information is not as easily integrated into **SEEDS** as into other superpixel algorithms like **SLIC**. Nevertheless, we provide two implementations offering slightly increased performance when utilizing depth information. Based on ad-

ditional experiments conducted with different algorithms using depth information, we are able to conclude that depth information does not provide a significant increase in performance. This conclusion is based on several considerations concerning the NYU Depth Dataset. To begin with, **SEEDS** does achieve excellent performance even without depth information, leaving only little room for improvement. Additionally, scenes or regions where **SEEDS** fails mostly show highly cluttered scenes with bad lighting. In these cases, the provided depth information tends to be noisy or incomplete such that using depth information in the form of 3D point coordinates or point normals does not provide sufficient information to further improve performance.

Finally, many superpixel algorithms provide state-of-the-art performance. However, as these algorithms are comparable in performance, other aspects become important: the availability of implementations, ease of use, visual quality and especially runtime. Unfortunately, only few algorithms are available through popular open source computer vision libraries. Ease of use may refer to several properties as for example the provided parameters or whether the number of superpixels is controllable. In our opinion, the latter is an important property of a superpixel algorithm as the number of superpixels provides a lower bound on the achievable performance. And although a compactness measure is available [SFS12], visual quality is difficult to measure. In contrast, runtime is measurable and may be an important property, especially when considering realtime applications. Based on these considerations, we propose to use our implementation of **SEEDS** as baseline for future research as it provides all of the discussed quality characteristics.

8.1 Outlook and Future Work

As brief outlook, we see superpixel algorithms at a stage ready to be used by a great variety of computer vision applications. Actually, in our opinion, the field of superpixel algorithms appears to be saturated, that is we do not expect further research concerning novel superpixel algorithms to result in groundbreaking new approaches both increasing performance as well as decreasing runtime. However, we see a deficit concerning supervoxel algorithms. To date, there is only one algorithm ready to be used to oversegment point clouds. Furthermore, suitable benchmarks as well as annotated datasets are missing.

Future work based on our implementation of **SEEDS** includes an integration into OpenCV [Bra00], one of the most popular open source computer vision libraries, to increase accessibility as well as – if applicable – a variant adapting the number and size of superpixels according to depth information. Additionally, due to time constraints, we were not able to conduct several interesting experiments. For example we started to implement an extension of the superpixel algorithm proposed in [CMM13] using depth information, however, were not able to include any experimental results. Further, we plan to use our implementation of **SEEDS** as basis for both classical segmentation as well as semantic segmentation.

Appendices

Appendix A

Berkeley Segmentation Benchmark

Arbeláez et al. [AMFM11] provide a benchmark implemented in MatLab together with the BSDS500. Although most of the provided measures are not suitable for evaluating superpixel algorithms, they provide a framework which can easily be extended to evaluate the quality of superpixel segmentations. Note that these measures are not intended to assess superpixel algorithms but are usually used to evaluate classical segmentation algorithms or contour detectors. This chapter discusses the provided measures.

A.1 Precision-Recall Framework

Introduced in [MFM04], the Precision-Recall Framework is used to evaluate contour detectors given a ground truth segmentation. By treating region boundaries of a segmentation as contours, the framework can be applied to evaluate segmentation algorithms as well [AMFM11]. The framework relies on the following definitions (we follow [NP12]): Let G be a ground truth segmentation and S be a given segmentation to evaluate. Then, define:

True Positives, $TP(S,G)$**:** Boundary pixels in G for which there is a boundary pixel in S in range r.

False Positives, $FP(S,G)$**:** Boundary pixels in S for which there is no boundary pixel in G in range r.

False Negatives, $FN(S,G)$**:** Boundary pixels in G for which there is no boundary pixel in S in range r.

The definitions already solve the problem of boundary correspondence[1] as discussed in [MFM04] by allowing a small displacement defined by the range r.

[1] The boundary correspondence problem describes the problem of finding the boundary pixel in the segmentation S corresponding to a given boundary pixel in the ground truth G, see [MFM04].

With the above definitions we can define the Boundary Precision $Pre(S,G)$ and the Boundary Recall $Rec(S,G)$ of a segmentation S with respect to a ground truth G:

$$Pre(S,G) = \frac{|TP(S,G)|}{|TP(S,G)| + |FP(S,G)|}; \tag{A.1}$$

$$Rec(S,G) = \frac{|TP(S,G)|}{|TP(S,G)| + |FN(S,G)|}. \tag{A.2}$$

Precision relates the number of True Positives to the total number of boundary detections in S (note that $TP(S,G) \cap FP(S,G) = \emptyset$ by definition). Recall, on the other hand, relates the number of True Positives to the total number of boundary pixels in G and therefore quantifies the fraction of boundary pixels in G which are correctly detected in S. The requirement that superpixels should adhere object boundaries relates to a high Boundary Recall. However, due to the oversegmentation, Boundary Precision is an unsuited measure for evaluating superpixel algorithms. This is because we encourage superpixel boundaries within objects, this means we typically get a high number of False Positives.

A.2 Variation of Information

Variation of Information was initially used for cluster comparison [Mei05] and is defined based on the entropy and mutual information of a segmentation S and a ground truth segmentation G:

$$H(S) = -\sum_{S_j \in S} \frac{|S_j|}{N} \log\left(\frac{|S_j|}{N}\right), \tag{A.3}$$

$$I(S,G) = \sum_{S_j \in S} \sum_{G_i \in G} \frac{|S_j \cap G_i|}{N} \log\left(\frac{|S_j \cap G_i|}{N} \frac{N}{|S_j|} \frac{N}{|G_i|}\right). \tag{A.4}$$

Then, the variation of information is given as

$$VI(S,G) = H(S) + H(G) - 2I(S,G). \tag{A.5}$$

As shown in [Mei03], $VI(S,G)$ is a metric. However, Variation of Information is not meaningful when evaluating superpixel algorithms. This is because the segment sizes S_j and G_i are not comparable in size. Even if a superpixel S_j perfectly matches the underlying ground truth segment G_i, the intersection $S_j \cap G_i$ is usually only a small subset of G_i: $|S_j \cap G_i| \ll |G_i|$.

A.3 Probabilistic Rand Index

The Rand Index was originally proposed in [Ran71] and the Probabilistic Rand Index was proposed in [UPH07]. The Probabilistic Rand Index is defined as

$$PRI(S,G) = \frac{1}{\binom{N}{2}} \sum_{n \neq m} (c_{n,m} p_{n,m} + (1 - c_{n,m})(1 - p_{n,m})) \tag{A.6}$$

where $c_{n,m}$ describes the random event where the pixels x_n and x_m have the same label in S and G with probability $p_{n,m}$. For a detailed interpretation we refer to [UPH07]. This measure is implemented as shown in the appendix of [UPH07]:

$$c_{n,m} = 1 \Leftrightarrow s(x_n) = s(x_m); \tag{A.7}$$

$$p_{n,m} = 1 \Leftrightarrow g(x_n) = g(x_m); \tag{A.8}$$

where $g(x_m)$ denotes the label of pixel x_m within the ground truth G. As stated in [UPH07], this definition can easily be extended to the case of multiple ground truth segmentations $G_1, \ldots, G_5$. Now we consider S to be a superpixel segmentation. Then we usually find multiple pixel pairs having the same label in the ground truth, but different labels in the superpixel segmentation. Therefore, the Probabilistic Rand Index is unsuited for evaluating superpixel algorithms.

A.4 Segmentation Covering

The Segmentation Covering is a measure based on the overlap of two segments S_j and G_i from a superpixel segmentation S and a ground truth G [MFM04]:

$$O(S_j, G_i) = \frac{|S_j \cap G_i|}{|S_j \cup G_i|}. \tag{A.9}$$

Note that $O(S_j, G_i)$ equals one if and only if S_j and G_i are equal. Using the overlap we can define the Segmentation Covering of G by S:

$$Cov(S \to G) = \frac{1}{N} \sum_{G_i \in G} |G_i| \max_{S_j \in S} \{O(G_i, S_j)\}. \tag{A.10}$$

Similarly, we define the Segmentation Covering of S by G as $Cov(G \to S)$. Consider S to be a superpixel segmentation. Then, for the overlap we usually have

$$O(S_j, G_i) \ll 1 \tag{A.11}$$

for most of the superpixels S_j. Therefore, both $Cov(S \to G)$ and $Cov(G \to S)$ are inappropriate for evaluating superpixel algorithms.

Appendix B

Evaluation – Images, Figures and Tables

Due to the large amount of evaluation data, appendix B is available separately at

`http://davidstutz.de/`.

Bibliography

[AMFM09] P. Arbeláez, M. Maire, C. Fowlkes, and J. Malik. From contours to regions: An empirical evaluation. In *Computer Vision and Pattern Recognition, Conference on*, pages 2294–2301, Miami, Florida, June 2009.

[AMFM11] P. Arbeláez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *Pattern Analysis and Machine Intelligence, Transactions on*, 33(5):898–916, May 2011.

[ASS+10] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. SLIC superpixels. Technical report, École Polytechnique Fédérale de Lausanne, Lusanne, Switzerland, June 2010.

[ASS+12] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. SLIC superpixels compared to state-of-the-art superpixel methods. *Pattern Analysis and Machine Intelligence, Transactions on*, 34(11):2274–2281, November 2012.

[Bis06] C. Bishop. *Pattern Recognition and Machine Learning*. Springer Verlag, New York, 2006.

[BK14] J. Borovec and J. Kybic. jSLIC: Superpixels in ImageJ. In *Computer Vision Winter Workshop*, 2014.

[BOV03] A. Barla, F. Odone, and A. Verri. Histogram intersection kernel for image classification. In *Image Processing, International Conference on*, volume 3, pages 513–516, Barcelona, Spain, September 2003.

[Bra00] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. `http://opencv.org/`.

[BVZ01] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *Pattern Analysis and Machine Intelligence, Transactions on*, 23(11):1222–1239, November 2001.

[CLRS09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, 2009.

[CMM13] C. Conrad, M. Mertz, and R. Mester. Contour-relaxed superpixels. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, volume 8081 of *Lecture Notes in Computer Science*, pages 280–293. Springer Berlin Heidelberg, 2013.

[CWF13] J. Chang, D. Wei, and J. W. Fisher. A video representation using temporal superpixels. In *Computer Vision and Pattern Recognition, Conference on*, pages 2051–2058, Portland, Oregon, June 2013.

[DM09] F. Drucker and J. MacCormick. Fast superpixels for video analysis. In *Motion and Video Computing, Workshop on*, pages 1–8, Snowbird, Utah, December 2009.

[FCT+14] H. Fu, X. Cao, D. Tang, Y. Han, and D. Xu. Regularity preserved superpixels and supervoxels. *Multimedia, Transactions on*, 16(4):1165–1175, June 2014.

[FH04] P. F. Felzenswalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *Computer Vision, International Journal of*, 59(2), 2004.

[FP02] D. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, New Jersey, 2002.

[GAM13] S. Gupta, P. Arbeláez, and J. Malik. Perceptual organization and recognition of indoor scenes from RGB-D images. In *Computer Vision and Pattern Recognition, Conference on*, pages 564–571, Portland, Oregon, June 2013.

[HRD+12] S. Holzer, R. B. Rusu, M. Dixon, S. Gedikli, and N. Navab. Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images. In *Intelligent Robots and Systems, International Conference on*, pages 2684–2689, Vilamoura, Portugal, October 2012.

[JTB+11] V. Jain, S. C. Turaga, K. L. Briggman, M. N. Helmstaedter, W. Denk, and H. S. Seung. Learning to agglomerate superpixel hierarchies. In *Neural Information Processing Systems, Conference on*, pages 648–656. Curran Associates, December 2011.

[KAWB09] K. Klasing, D. Althoff, D. Wollherr, and M. Buss. Comparison of surface normal estimation methods for range sensing applications. In *Robotics and Automation, International Conference on*, pages 3206–3211, Kobe, Japan, May 2009.

[LSK+09] A. Levinshtein, A. Stere, K. N. Kutulakos, D. J. Fleet, S. J. Dickinson, and K. Siddiqi. TurboPixels: Fast superpixels using geometric

flows. *Pattern Analysis and Machine Intelligence, Transactions on*, 31(12):2290–2297, December 2009.

[LTRC11] M. Y. Lui, O. Tuzel, S. Ramalingam, and R. Chellappa. Entropy rate superpixel segmentation. In *Computer Vision and Pattern Recognition, Converence on*, pages 2097–2104, Providence, Rhode Island, June 2011.

[MCG11] R. Mester, C. Conrad, and A. Guevara. Multichannel segmentation using contour relaxation: Fast super-pixels and temporal propagation. In *Image Analysis*, volume 6688 of *Lecture Notes in Computer Science*, pages 250–261. Springer Berlin Heidelberg, 2011.

[Mei03] M. Meilă. Comparing clusterings by the variation of information. In *Learning Theory and Kernel Machines*, volume 2777 of *Lecture Notes in Computer Science*, pages 173–187. Springer Berlin Heidelberg, 2003.

[Mei05] M. Meilă. Comparing clusterings: an axiomatic view. In *Machine Learning, International Conference on*, pages 577–584, Bonn, Germany, 2005.

[MFM04] D. Martin, C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *Pattern Analysis and Machine Intelligence, Transactions on*, 26(5):530–549, May 2004.

[MFTM01] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Computer Vision, International Conference on*, volume 2, pages 416–423, Vancouver, Canada, July 2001.

[Mor05] G. Mori. Guiding model search using segmentation. In *Computer Vision, International Conference on*, volume 2, pages 1417–1423, Beijing, China, October 2005.

[MPW+08] A. P. Moore, S. J. D. Prince, J. Warrell, U. Mohammed, and G. Jones. Superpixel lattices. In *Computer Vision and Pattern Recognition, Conference on*, pages 1–8, Anchorage, Alaska, June 2008.

[MPW10] A. P. Moore, S. J. D. Prince, and J. Warrell. Lattice cut - constructing superpixels using layer constraints. In *Computer Vision and Pattern Recognition, Conference on*, pages 2117–2124, San Francisco, California, June 2010.

[MREM04] G. Mori, X. Ren, A. A. Efros, and J. Malik. Recovering human body configurations: combining segmentation and recognition. In *Computer Vision and Pattern Recognition, Conference on*, volume 2, pages 326–333, Washington, D.C., June 2004.

[NP12] P. Neubert and P. Protzel. Superpixel benchmark and comparison. In *Forum Bildverarbeitung*, Regensburg, Germany, November 2012.

[OF03] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer Verlag, New York, 2003.

[PASW13] J. Papon, A. Abramov, M. Schoeler, and F. Wörgötter. Voxel cloud connectivity segmentation - supervoxels for point clouds. In *Computer Vision and Pattern Recognition, Conference on*, pages 2027–2034, Portland, Oregon, June 2013.

[PM11] F. Perbet and A. Maki. Homogeneous superpixels from random walks. In *Machine Vision and Applications, Conference on*, pages 26–30, Nara, Japan, June 2011.

[Ran71] W. M. Rand. Objective criteria for the evaluation of clustering methods. *American Statistical Association, Journal of the*, 66(336):846–850, 1971.

[RB12] X. Ren and L. Bo. Discriminatively trained sparse code gradients for contour detection. In *Advances in Neural Information Processing Systems*, volume 25, pages 584–592. Curran Associates, 2012.

[RBB09] R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (FPFH) for 3D registration. In *Robotics and Automation, International Conference on*, pages 3212–3217, Kobe, Japan, May 2009.

[RBF12] X. Ren, L. Bo, and D. Fox. RGB-(D) scene labeling: Features and algorithms. In *Computer Vision and Pattern Recognition, Conference on*, pages 2759–2766, Providence, Rhode Island, June 2012.

[RC11] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *Robotics and Automation, International Conference on*, Shanghai, China, May 2011.

[RE07] C. Rohkohl and K. Engel. Efficient image segmentation using pairwise pixel similarities. In *Pattern Recognition*, volume 4713 of *Lecture Notes in Computer Science*, pages 254–263. Springer Berlin Heidelberg, 2007.

[RJRO13] M. Reso, J. Jachalsky, B. Rosenhahn, and J. Ostermann. Temporally consistent superpixels. In *Computer Vision, International Conference on*, pages 385–392, Sydney, Australia, December 2013.

[RM03] X. Ren and J. Malik. Learning a classification model for segmentation. In *Computer Vision, International Conference on*, pages 10–17, Nice, France, October 2003.

[RR11] C. Y. Ren and I. Reid. gSLIC: a real-time implementation of SLIC superpixel segmentation. Technical report, University of Oxford, Oxford, England, 2011.

[RTMF08] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. LabelMe: A database and web-based tool for image annotation. *Computer Vision, International Journal of*, 77(1-3):157–173, 2008.

[Rus09] R. B. Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Technische Universität München, Munich, Germany, 2009.

[SF11] N. Silberman and R. Fergus. Indoor scene segmentation using a structured light sensor. In *Computer Vision Workshops, International Conference on*, pages 601–608, Barcelona, Spain, November 2011.

[SFS12] A. Schick, M. Fischer, and R. Stiefelhagen. Measuring and evaluating the compactness of superpixels. In *Pattern Recognition, International Conference on*, pages 930–934, Tsukuba, Japan, November 2012.

[SHKF12] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from RGBD images. In *Computer Vision, European Conference on*, volume 7576 of *Lecture Notes in Computer Science*, pages 746–760. Springer Berlin Heidelberg, 2012.

[SM00] J. Shi and J. Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, Transactions on*, 22(8):888–905, August 2000.

[SWRC09] J. Shotton, J. Winn, C. Rother, and A. Criminisi. TextonBoost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *Computer Vision, International Journal of*, 81(1):2–23, 2009.

[TFC12] D. Tang, H. Fu, and X. Cao. Topology preserved regular superpixel. In *Multimedia and Expo, International Conference on*, pages 765–768, Melbourne, Australia, July 2012.

[TL10] J. Tighe and S. Lazebnik. SuperParsing: Scalable nonparametric image parsing with superpixels. In *Computer Vision, European Conference on*, volume 6315 of *Lecture Notes in Computer Science*, pages 352–365. Springer Berlin Heidelberg, 2010.

[UPH07] R. Unnikrishnan, C. Pantofaru, and M. Hebert. Toward objective evaluation of image segmentation algorithms. *Pattern Analysis and Machine Intelligence, Transactions on*, 29(6):929–944, June 2007.

[VBM10] O. Veksler, Y. Boykov, and P. Mehrani. Superpixels and supervoxels in an energy optimization framework. In *Computer Vision, European Conference on*, volume 6315 of *Lecture Notes in Computer Science*, pages 211–224. Springer Berlin Heidelberg, 2010.

[vdBBR+12] M. van den Bergh, X. Boix, G. Roig, B. de Capitani, and L. van Gool. SEEDS: Superpixels extracted via energy-driven sampling. In *Computer Vision, European Conference on*, volume 7578 of *Lecture Notes in Computer Science*, pages 13–26. Springer Berlin Heidelberg, 2012.

[vdBBR+13] M. van den Bergh, X. Boix, G. Roig, B. de Capitani, and L. van Gool. SEEDS: Superpixels extracted via energy-driven sampling. *Computing Research Repository*, abs/1309.3848, 2013.

[vdBRB+13] M. van den Bergh, G. Roig, X. Boix, S. Manen, and L. van Gool. Online video seeds for temporal window objectness. In *Computer Vision, International Conference on*, pages 377–384, Sydney, Australia, December 2013.

[VF08] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. `http://www.vlfeat.org/`, 2008.

[VS91] L. Vincent and P. Soille. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *Pattern Analysis and Machine Intelligence, Transactions on*, 13(6):583–598, June 1991.

[VS08] A. Vedaldi and S. Soatto. Quick shift and kernel methods for mode seeking. In *Computer Vision, European Conference on*, volume 5305 of *Lecture Notes in Computer Science*, pages 705–718. Springer Berlin Heidelberg, 2008.

[Wei14] D. Weikersdorfer. *Efficiency by Sparsity: Depth-Adaptive Superpixels and Event-based SLAM*. PhD thesis, Technische Universität München, Munich, Germany, 2014.

[WGB12] D. Weikersdorfer, D. Gossow, and M. Beetz. Depth-adaptive superpixels. In *Pattern Recognition, International Conference on*, pages 2087–2090, Tsukuba, Japan, November 2012.

[WLYY11] S. Wang, H. Lu, F. Yang, and M.-H. Yang. Superpixel tracking. In *Computer Vision, International Conference on*, pages 1323–1330, Barcelona, Spain, November 2011.

[WSC13] D. Weikersdorfer, A. Schick, and D. Cremers. Depth-adaptive supervoxels for RGB-D video segmentation. In *Image Processing, International Converence on*, pages 2708–2712, Melbourne, Australia, September 2013.

[YLY14] F. Yang, H. Lu, and M.-H. Yang. Robust superpixel tracking. *Image Processing, Transactions on*, 23(4):1639–1651, April 2014.

[ZHMB11a] Y. Zhang, R. Hartley, J. Mashford, and S. Burn. Superpixels via pseudo-boolean optimization. In *Computer Vision, International Conference on*, pages 1387–1394, Barcelona, Spain, November 2011.

[ZHMB11b] Yuhang Zhang, R. Hartley, J. Mashford, and S. Burn. Superpixels, occlusion and stereo. In *Digital Image Computing Techniques and Applications, International Conference on*, pages 84–91, Noosa, Australia, December 2011.

[ZWW+11] G. Zeng, P. Wang, J. Wang, R. Gan, and H. Zha. Structure-sensitive superpixels via geodesic distance. In *Computer Vision, International Conference on*, pages 447–454, Barcelona, Spain, November 2011.