Superpixels and Polygons using Simple Non-Iterative Clustering

Radhakrishna Achanta and Sabine Süsstrunk
School of Computer and Communication Sciences (IC)
École Polytechnique Fédérale de Lausanne (EPFL)
Switzerland
{radhakrishna.achanta,sabine.susstrunk}@epfl.ch

Abstract

We present an improved version of the Simple Linear Iterative Clustering (SLIC) superpixel segmentation. Unlike SLIC, our algorithm is non-iterative, enforces connectivity from the start, requires lesser memory, and is faster. Relying on the superpixel boundaries obtained using our algorithm, we also present a polygonal partitioning algorithm. We demonstrate that our superpixels as well as the polygonal partitioning are superior to the respective state-of-the-art algorithms on quantitative benchmarks.

1. Introduction

Image segmentation continues to be a challenge that attracts both domain specific and generic solutions. To avoid the struggle with semantics when using traditional segmentation algorithms, researchers lately diverted their attention to a much simpler and achievable task, namely that of simplifying an image into small clusters of connected pixels called *superpixels*. Superpixel segmentation has quickly become a potent pre-processing tool that simplifies an image from, potentially, millions of pixels to about two orders of magnitude fewer, clusters of similar pixels.

After their introduction [27], several applications such as object localization [14], multi-class segmentation [15], optical flow [22], body model estimation [24], object tracking [35], and depth estimation [37] took advantage of superpixels. For these applications, superpixels are commonly expected to have the following properties [7, 18]:

- Tight region boundary adherence.
- Containing a small cluster of similar pixels.
- Uniformity; roughly equally sized clusters.
- Compactness; limiting the degree of adjacency.
- Computational efficiency.

One of the most promiment superpixel segmentation algorithms is the Simple Linear Iterative Clustering algorithm (SLIC) [7], which satisfies these criteria and is very efficient in terms of computation and memory requirements. Despite its widespread use, SLIC suffers from a few shortcomings. It requires several iterations for the centroids to converge. It uses a distance map of the same size as the number of input pixels, which amounts to significant memory consumption for image stacks or video volumes. Lastly, SLIC enforces connectivity only as a post-processing step.

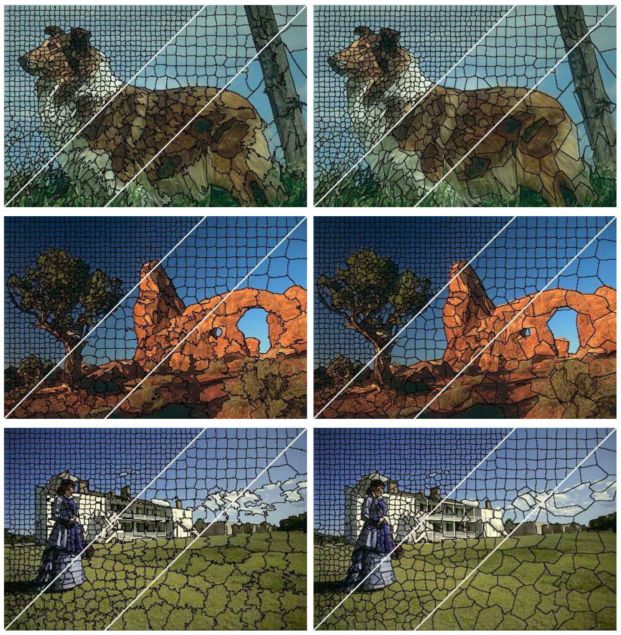


Figure 1. Images on the left show SNIC segmentation for three different superpixel sizes. Images on the right show the corresponding SNICPOLY polygonal partitioning.

In this paper, we first present an improved version of the SLIC algorithm that overcomes the above-mentioned limitations: (1) our algorithm runs in a single iteration; (2) it does not use a distance map and therefore requires less memory; and, (3) our algorithm enforces connectivity explicitly from the start. In addition, our algorithm improves (4) the computational efficiency, (5) memory consumption, and (6) segmentation quality. Since our algorithm is a non-iterative version of the SLIC algorithm, we call it *Simple Non-Iterative Clustering* (SNIC).

Second, we propose a polygonal segmentation algorithm called SNICPOLY, which uses SNIC superpixel segmentation as the basis. Polygonal segmentation of images has been shown [12] to be especially suited for applications that deal with images containing geometric or man-made structures [9]. An example of SNIC segmentation and SNICPOLY polygonal partitioning is presented in Fig. 1.

We compare SNIC to the state-of-the-art in superpixel segmentation [13, 16, 17, 18, 28, 30] and the polygonal partitioning to the corresponding state-of-the-art [12]. In both cases, our algorithms perform better quantitatively as well as in terms of computational efficiency.

The rest of the paper is as follows. We review other segmentation algorithms in Section 2, describe the SNIC algorithm in Section 3, and explain how we obtain polygonal partitioning of the image using SNIC in Section 4. In Section 5, we compare SNIC and our polygonal segmentation with the existing algorithms, and conclude the paper with Section 6.

2. Background

In this section, we review SLIC [7] and related algorithms. For the sake of completeness we also review other state-of-the-art techniques. More reviews of superpixel techniques can be found in [7, 25].

2.1. SLIC

The Simple Linear Iterative Clustering Algorithm [7], or SLIC, is one of the most prominent superpixel segmentation algorithms. It owes its success as a pre-processing algorithm to its simplicity, its computational efficiency, and its ability to generate superpixels that satisfy the requirements of good boundary adherence and limited adjacency. SLIC performs a localized k-means optimization in the five-dimensional CIELAB color and image space to cluster pixels, starting from seeds chosen on a regular grid. SLIC has only two input parameters in practice - the required number of superpixels and a compactness factor, which determines how compact the superpixels are. The authors of SLIC also introduce a version that does not require the compactness parameter as input because it automatically sets its value to the maximum color distance within a superpixel.

Owing to its widespread use, variants of SLIC have been introduced. Li and Chen [17] present Linear Spectral Clustering (LSC) that projects the five-dimensional space of spatial and color coordinates to a ten-dimensional space before performing the k-means clustering. Liu et al. [19] on the other hand present manifold-SLIC (MSLIC) that projects the same five-dimensional space to a two-dimensional space before performing clustering. The authors of both these methods show quantitative improvements in segmentation quality, but the underlying limitations of the SLIC algorithm, such as multiple iterations and lack of explicit connectivity, remain unchanged.

2.2. Graph-based algorithms

Some of the other prominent approaches for image segmenation rely on pixel graphs. The Normalized cuts algorithm (NCUTS) [28] creates superpixels by recursively computing normalized cuts for the pixel graph. Felzenszwalb and Huttenlocher [13] propose a minimum spanning tree based segmentation approach (MST). Their algorithm progressively joins components until a stopping criterion is met, which prevents the spanning tree from covering the whole image, is met. Moore et al. [23] generate superpixels (SLAT) by finding the shortest paths that split an image into vertical and horizontal strips. Similarly, Zhang et al. [36] create superpixels (SPBO) by applying horizontal and vertical graph-cuts to overlapping strips of an image. Instead of finding cuts on an image, Veskler et al. [32], generate superpixels by stitching together overlapping image patches. More recently, Liu et al. [18] present another graph-based approach (ERS) that connects subgraphs by maximizing the entropy rate of a random walk.

2.3. Non-graph-based algorithms

There are several other algorithms that are not graph-based. The watershed algorithm (WSHED) [33] accumulates similar pixels starting from local minima to find *watersheds*, i.e., lines that separate segments. The mean shift algorithm (MSHIFT) [10] iteratively locates local maxima of a density function in color and image plane space. Pixels that lead to the same local maximum belong to the same segment. Quick shift (QSHIFT) [31] creates superpixels by seeking local maxima like MSHIFT but is more efficient in terms of computation. The Turbopixels algorithm (TURBO) [16] generates superpixels by progressively dilating pixel seeds located at regular grid centers using a level-set approach. SEEDS [30] generates superpixels by iteratively improving an initial rectangular approximation of superpixels using coarse to fine pixel exchanges with neighboring superpixels.

2.4. Summary

MST, MSHIFT, and WSHED are traditional segmentation algorithms, they do not aim for uniformly-sized compact segments. The rest are considered superpixel algorithms - they aim to generate clusters of segments that have uniform sizes and a small number of adjacent segments. Of these, NCUTS, SLAT, TURBO, SLIC, and SPBO exhibit more compactness, i.e., a higher ratio of the area of a superpixel to its perimeter. MST, SLIC, SEEDS, and LSC are the fastest in computation. TURBO, SLIC, ERS, and SEEDS allow the user control over the number of output segments. This last property of superpixels has become important because it lets the user choose the size of the superpixels based on the needs of the application.

2.5. Polygonal partitioning

In a small deviation from the theme of superpixels, Duan and Lafarge [12] present a method (CONPOLY), which partitions the image into uniformly-sized convex polygons instead of creating superpixels of arbitrary shape. Such partitioning finds use in applications such as surface reconstruction [9] and object localization [29]. The authors of CONPOLY detect preliminary line segments using Line Segment Detector [34]. They build a Voronoi tessellation that conforms to these line segments, and then homogenize the generated polygons with additional partitions. The resulting algorithm is computationally efficient and shows good boundary adherence properties.

In this paper, we describe a method to perform polygonal partitioning of images using SNIC segmentation as the starting point. Our polygonal partitioning method SNICPOLY outperforms CONPOLY on standard benchmarks as well as in computational efficiency.

3. Simple non-iterative clustering (SNIC)

Our algorithm clusters pixels without the use of the k-means iterations, while explicitly enforcing connectivity from the start. In this section, we describe our algorithm in relation to SLIC [7].

3.1. Distance measure

Like SLIC, we also initialize our centroids with pixels chosen on a regular grid in the image plane. The affinity of a pixel to a centroid is measured using a distance in the five-dimensional space of color and spatial coordinates. Our algorithm uses the same distance measure as SLIC. This distance combines normalized spatial and color distances. With spatial position $\mathbf{x} = [x\ y]^T$ and CIELAB color $\mathbf{c} = [l\ a\ b]^T$, the distance of the kth superpixel centroid $C[k]$ to the jth candidate pixel is given by:

$$d_{j,k} = \sqrt{\frac{\|\mathbf{x}_j - \mathbf{x}_k\|_2^2}{s} + \frac{\|\mathbf{c}_j - \mathbf{c}_k\|_2^2}{m}}, \qquad (1)$$

where s and m are the normalizing factors for spatial and color distances, respectively. For an image of N pixels, each of the K superpixels is expected to contain N/K pixels. Assuming a square shape of a superpixel, the value of s in Eq. 1 is set to be $\sqrt{N/K}$. The value of m, also called the *compactness factor*, is user-provided. A higher value results in more compact superpixels at the cost of poorer boundary adherence, and vice versa.

3.2. Evolution of centroids

In each k-means iteration, SLIC evolves a centroid by computing the average of all pixels that are closest to it in terms of d and, therefore, have the same label as the centroid. In this manner, SLIC requires several iterations for the centroids to converge.

Starting from the initial centroids, our algorithm uses a priority queue to choose the next pixel to add to a cluster. The priority queue is populated with candidate pixels that are 4 or 8-connected to a currently growing superpixel cluster. Popping the queue provides the pixel candidate that has the smallest distance d from a centroid.

Each new pixel that is added to a superpixel is used to perform an online update of the corresponding centroid value. Thus, unlike SLIC, which requires multiple k-means iterations to update the centroids, we update the centroids in a single iteration.

The online updating of the centroids is quite effective because redundancies in natural images usually result in adjacent pixels being quite similar. The centroids therefore converge quickly, as demonstrated in Fig. 2.

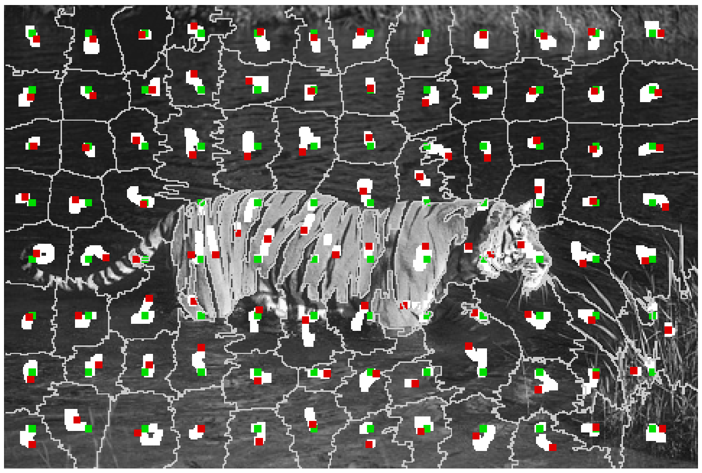
3.3. Efficient distance computation

SLIC achieves computational efficiency by restricting the distance computations within square regions of area $2s \times 2s$ that are centered around the K centroids. The size of the square regions is conservatively chosen to ensure that there is some overlap between the squares of neighboring centroids on the image plane. So, each pixel is reachable by the nearest centroids even after the centroids get displaced from their original position on the image plane during the k-means iterations.

Since pixel connectivity is not explicitly enforced in such k-means based clustering, pixels that may not belong to the final superpixel but lie in the $2s \times 2s$ regions are visited nonetheless and the distance d is computed for them. Although the overlapping square restriction drastically reduces the number of distances to be computed, redundant computations are unavoidable.

We only compute distances to pixels that are 4 or 8-connected to the currently growing cluster in order to create elements to populate the queue. Therefore, even compared to a single iteration of SLIC, our algorithm computes fewer distances. A natural consequence of enforcing connectivity is also that we do not need to impose any spatial restrictions on distance computation like SLIC does. The queue contains far fewer elements than N, so it uses less memory than SLIC that requires a memory of size N to store distances.

Through the use of a priority queue and online averaging for updating the centroids, we thus obtain SNIC, which has the following advantages over SLIC:



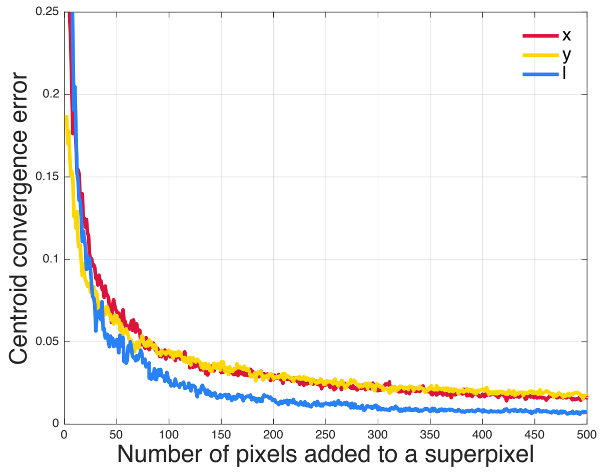

Centroid convergence error
Number of pixels added to a superpixel
x
y
l
0
0.05
0.1
0.15
0.2
0.25
50
100
150
200
250
300
350
400
450
500

Figure 2. Effectiveness of the online update. The left image shows 100 spatial centroids that start at the position shown by green squares and drift during convergence to the position shown by red squares. The intermediate positions occupied are shown in white. The right image shows the plot of the average change, i.e., residual error, of the x, y, and l centroids w.r.t their previous values over the 100 superpixels. As seen, within adding the first 50 pixels to a superpixel, the errors sufficiently drop down, i.e., the centroids converge.

- Connectivity enforced explicitly from the start.
- No need for multiple k-means iterations.
- Fewer pixel visits and distance computations.
- Lower memory requirements.

The pseudo-code for the algorithm is presented in Algorithm 1 and is explained below.

Algorithm 1 SNIC segmentation algorithm

```
Input: Input image $I$, $K$ initial centroids $C[k] = \{\mathbf{x}_k, \mathbf{c}_k\}$
    sampled on a regular grid, color normalization factor $m$
Output: Assigned label map $L$
 1: Initialize $L[:] \leftarrow 0$
 2: for $k \in [1, 2, ...K]$ do
 3:     Element $e \leftarrow \{\mathbf{x}_k, \mathbf{c}_k, k, 0\}$
 4:     Push $e$ on priority queue $Q$
 5: end for
 6: while $Q$ is not empty do
 7:     Pop $Q$ to get $e_i$
 8:     if $L[\mathbf{x}_i]$ is 0 then
 9:         $L[\mathbf{x}_i] = k_i$
10:         Update centroid $C[k_i]$ online with $\mathbf{x}_i$ and $\mathbf{c}_i$
11:         for Each connected neighbor $\mathbf{x}_j$ of $\mathbf{x}_i$ do
12:             Create element $e_j = \{\mathbf{x}_j, \mathbf{c}_j, k_i, d_{j,k_i}\}$
13:             if $L[\mathbf{x}_j]$ is 0 then
14:                 Push $e_j$ on $Q$
15:             end if
16:         end for
17:     end if
18: end while
19: return $L$
```

3.4. Algorithm

The initial K seeds $C[k] = \{\mathbf{x}_k, \mathbf{c}_k\}$ are obtained as for SLIC on a regular grid over the image. Using these seed pixels, K elements $e_i = \{\mathbf{x}_i, \mathbf{c}_i, k, d_{i,k}\}$ are created, wherein each label k is set to one unique superpixel label from 1 to K, and each distance value $d_{i,k}$, representing the distance of the pixel from the kth centroid, is set to zero. A priority queue Q is initialized with these K elements. When popped, Q always returns the element e_i whose distance value $d_{i,k}$ to the kth centroid is the smallest.

While Q is not empty, the top-most element is popped. If the pixel position on the label map L pointed to by the element is unlabeled, it is given the label of the centroid. The centroid value, which is the average of all the pixels in the superpixel, is updated with this pixel. In addition, for each of its 4 or 8 neighbors that have not been labeled yet, a new element is created, assigning to it the distance from the connected centroid and the label of the centroid. These new elements are pushed on the queue.

As the algorithm executes, the priority queue is emptied to assign labels at one end and populated with new candidates at the other. When there are no remaining unlabeled pixels to add new elements to the queue and the queue has been emptied, the algorithm terminates.

4. SNIC-based polygonal partitioning

The polygonal partitioning we perform relies on the boundaries generated by the SNIC superpixel segmentation. Each superpixel results in one polygon. The polygons are created taking care that adjacent superpixels share the same polygon edges. To create polygons from the initial segmen-

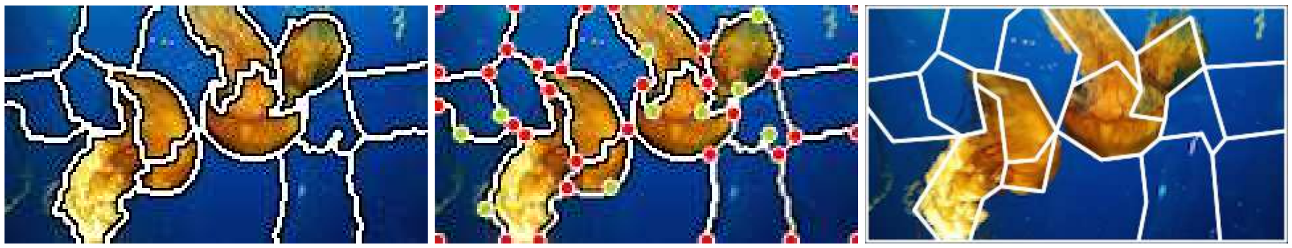

(a) SNIC segmentation (b) Choosing vertices (c) Polygon partitioning

Figure 3. A visual explanation of the polygon formation steps. (a) Initial segmentation using SNIC. (b) Initial vertices, in red, are chosen to be pixels that touch at least three different segments, at least two segments and the borders of the image, or, are image corners. The additional vertices, in green, are obtained by the Douglas-Peucker algorithm [11] algorithm. (c) After merging vertices that are too close, we obtain polygons by joining the remaining vertices with line segments.

tation, we take the following steps:

1. **Contour tracing:** The closed path along the boundary of each superpixel is traced using a standard contour tracing algorithm [26]. This generates an ordered sequence of pixel positions along superpixel boundaries.

2. **Initial vertices:** Since adjacent superpixels share boundaries, some common vertices are chosen. All pixel positions along the boundary paths that touch at least three superpixels or at least two superpixels and the image borders are taken as initial shared vertices (Fig. 3b). In addition these, the corners of the image are taken to be vertices.

3. **Additional vertices:** Now we simplify the path segment between two vertices. For each path segment between two vertices, we add new vertices using the Douglas–Peucker algorithm [11]. This simplifies the path segment from several pixel positions to a few polygon vertices.

4. **Vertex merging:** Depending on the superpixel size, vertices that are deemed to be very close to each other according to a threshold (one-tenth of the expected superpixel radius) are assigned a common vertex. This common vertex is chosen to be the one with the highest image gradient magnitude.

5. **Polygon generation:** Finally, polygons are obtained by joining the vertices obtained so far with straight line segments (Fig. 3c).

After creating the polygons, we relabel the pixels based on the polygonal borders. The entire process of creating polygons and assigning new labels takes only 20% more time than the initial SNIC segmentation. This makes our polygonal partitioning process SNICPOLY faster then CONPOLY [12]. As a note, although we rely on SNIC superpixels, the polygonal partioning algorithm presented in this section can also generate polygons for superpixels obtained using a different algorithm.

Unlike CONPOLY [12] though, some of our polygons can be non-convex, especially for natural images. If convex polygons or triangles are necessary for an application [9], it is possible to add edges inside the non-convex polygons to make them convex.

5. Experiments

We compare SNIC[1] to SLIC [7] as well as several state-of-the-art superpixel methods: NCUTS [28], MST [13], TURBO [16], SEEDS [30], ERS [18], and LSC [17]. We use the implementations available online for all methods [1, 2, 3, 4, 5, 6]. Simultaneously, we compare SNICPOLY against the state-of-the-art CONPOLY [12].

The benchmarking is done on the Berkeley 300 dataset [21] taking into consideration both the color and grayscale groundtruth images for the range of 50 to 2000 superpixels. The quantitative comparisons can be viewed in Fig. 5 and Fig. 6. A visual comparison of SNIC and SNICPOLY with the state-of-the-art is provide in Fig. 4.

5.1. Under-segmentation error

Under-segmentation error measures the overlap error that occurs when a superpixel is compared with the ground truth segment occupying the same location. Neubert and Protzel [25] observed that the computation of under-segmentation error as presented in TURBO [16] and SLIC [7] penalizes the overlapping on both sides when a superpixel straddles a ground truth boundary.

They propose the Corrected Under-Segmentation Error (CUSE), which corrects for this error. Its value is computed for each superpixel S_k as:

$$CUSE = \frac{1}{N}\sum_{k=1}^{K} |G^{max}(S_k) \cup S_k - G^{max}(S_k)|, \quad (2)$$

where $G^{max}(S_k)$ returns the ground truth segment with which superpixel S_k overlaps the most and $|.|$ is the abso-

[1]Our source code can be found at: http://ivrl.epfl.ch/research/snic_superpixels

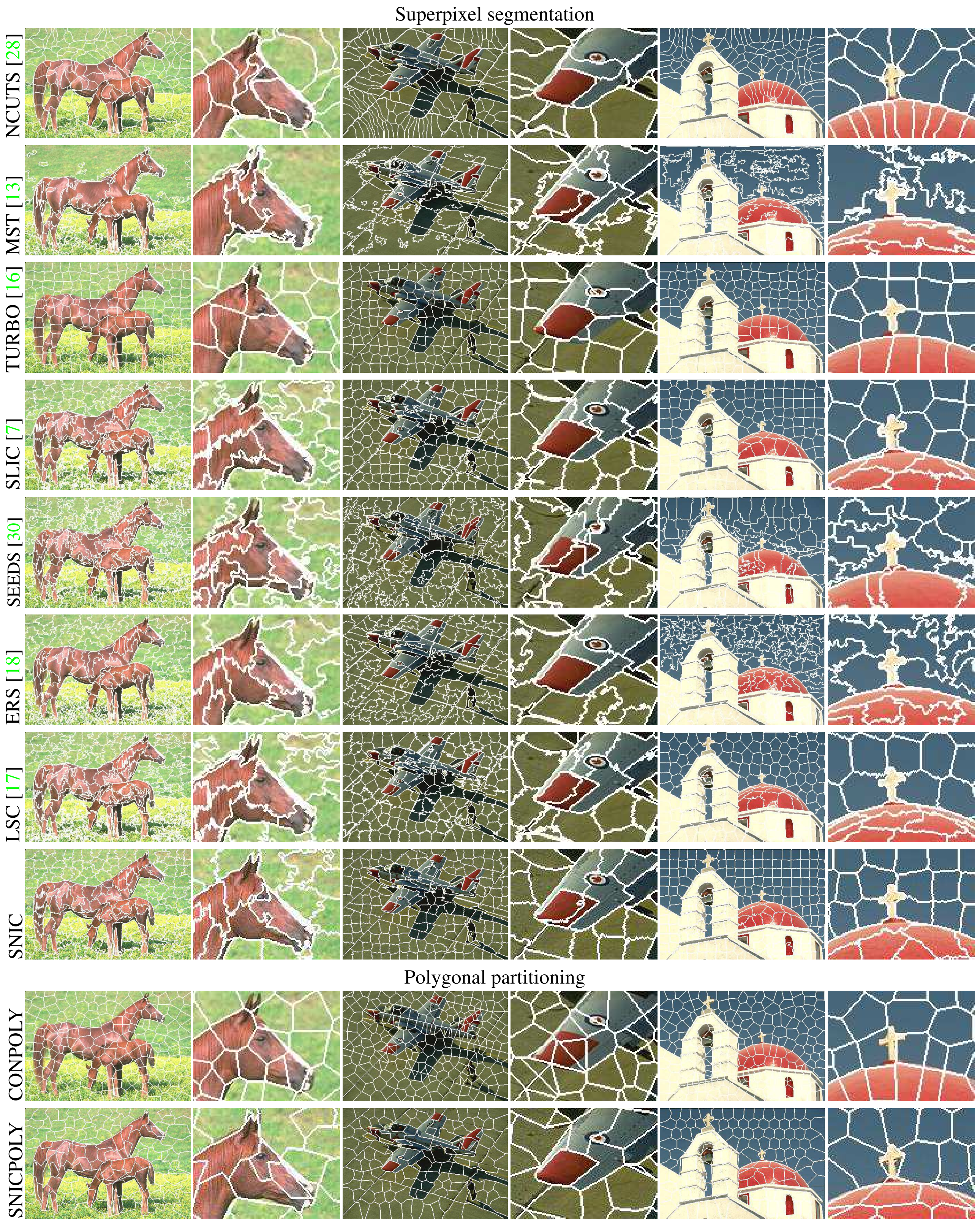


Figure 4. Visual comparison of SNIC superpixels and polygonal partitioning against state-of-the-art algorithms. In the zoomed-in regions, note how SNIC adapts well to each region of the image according to its local structure. SNIC results are generated with $m = 10$. Note: the polygon boundaries of CONPOLY [12] and SNICPOLY appear aliased because discrete pixel labels are used to draw them.

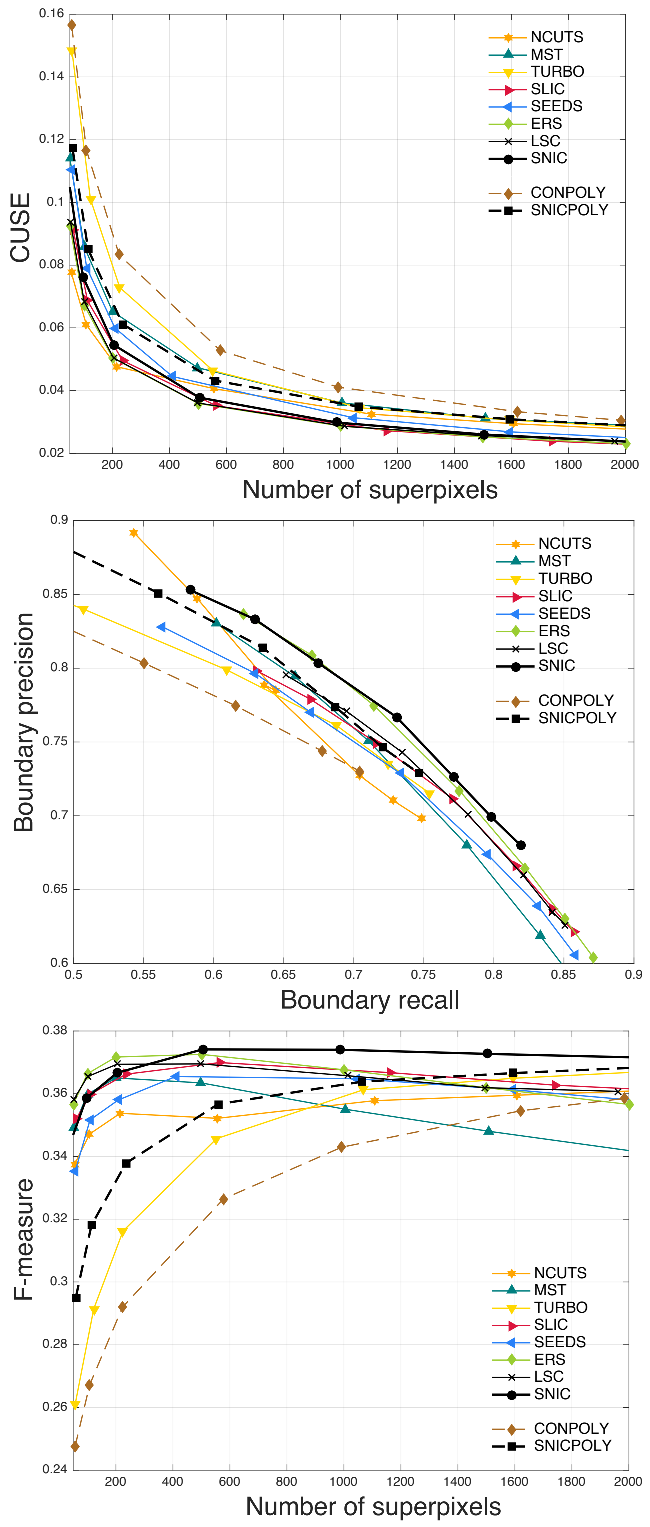


Figure 5. Comparison of SNIC and SNICPOLY with state-of-the-art methods. (Top) The under-segmentation error (CUSE) of SNIC is one of the lowest of all algorithms. (Middle) SNIC shows a better precision-versus recall performance, as well as (Bottom) a better F-measure than the state-of-the-art. Likewise, SNICPOLY significantly outperforms the state-of-the-art polygonal partitioning method CONPOLY [12].

lute value operator. CUSE is plotted against the number of superpixels in Fig. 5 (top plot).

5.2. Boundary recall and precision

To assess the performance of any detection technique, the two values of recall and precision are considered together [8, 20]. Recall is the ratio of the true positives to the sum of true positives and false negatives, while precision is the ratio of true positives to the sum of true and false positives. Recall or precision considered alone can be misleading since it is possible to have a very high recall with extremely poor precision and vice versa.

Let $B^S[i]$ and $B^G[i]$ be, respectively, the superpixel and ground truth boundary maps of the same dimensions as the input image. In these maps, the value at a pixel position $\mathbf{x}_i$ is 1 if it is a boundary pixel and 0 otherwise. We compute the number of true positives as:

$$TP = \sum_{i=1}^{N} \mathbb{1}_{j \in \mathcal{N}(i,\epsilon)} (B^G[i] \times B^S[j]), \tag{3}$$

where $\mathcal{N}$ is the $\epsilon \times \epsilon$ neighborhood around pixel position $\mathbf{x}_i$. The function $\mathbb{1}$ returns 1 if a superpixel boundary overlaps with the ground truth boundary pixel within the neighborhood $\mathcal{N}$ and 0 otherwise. We use $\epsilon = 2$ for our evaluations like the state-of-the-art [7, 18, 30, 25].

False positives is the number of superpixel boundary pixels in the ϵ neighborhood that are not true positives, i.e., do not have a groundtruth pixel in the vicinity. We compute its value as:

$$FP = \sum_{i=1}^{N} \left[1 - \mathbb{1}_{j \in \mathcal{N}(i,\epsilon)} (B^G[i] \times B^S[j])\right] \tag{4}$$

The sum of true positives and false negatives is the number of all boundary pixels, i.e.,

$$TP + FN = \sum_{i=1}^{N} (B^G[i]) \tag{5}$$

following the definition of $B^G[i]$. Therefore, using Eq. 3 and Eq. 4, we can compute $Recall = TP/(TP+FN)$ and $Precision = TP/(TP+FP)$. In Fig. 5 we show plots for Precision versus Recall (center plot), and F-measure versus the number of superpixels (bottom plot).

5.3. Computational efficiency

SNIC visits all pixels only once, except for those at the borders of the superpixels. Thus, the number of visits is N plus a value that is dependent on the number of desired superpixels K. The priority queue, when implemented using a heap data structure, is known to have logarithmic complexity for pushing and popping elements.

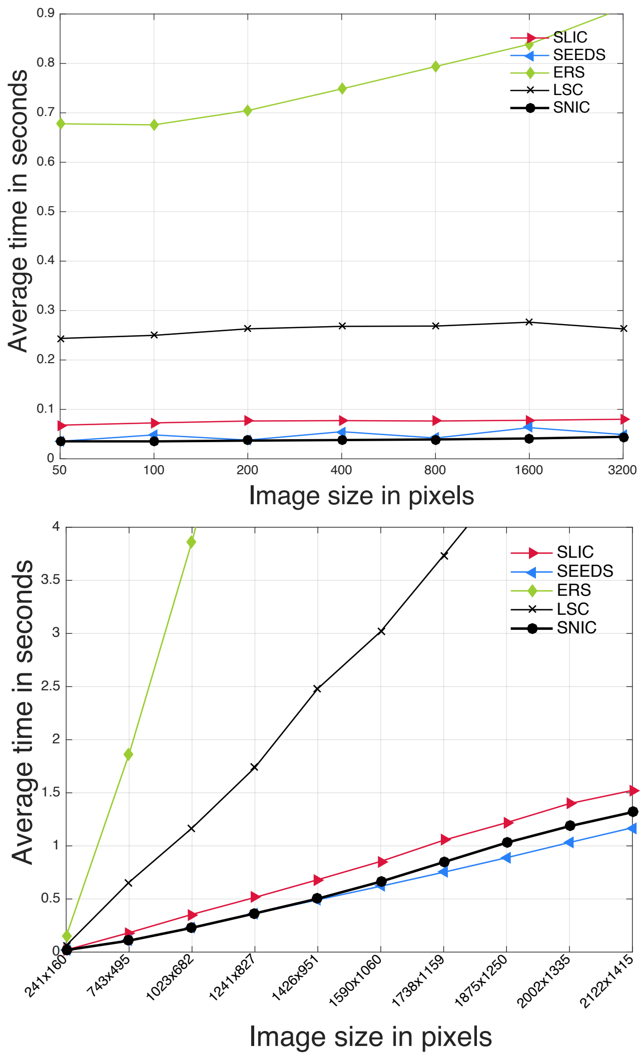


Figure 6. Comparison of speed in seconds. The top plot compares computation times against different number of superpixels averaged over a 100 images of size 321×481. The bottom plot compares the average computational time against linearly increasing image sizes. Both the plots show that SNIC exhibits linear complexity in practice and is also the fastest method after SEEDS. Note: SNICPOLY and CONPOLY speed curves (not plotted) lie quite close to the curves of SNIC and SLIC, respectively.

.

In our case, the size of the priority queue starts at K, increases while more elements are pushed than popped, and then reduces to zero. Since the number of elements on the queue is much smaller than the total number of pixels N, the influence of the priority queue on the computational complexity is not very pronounced. To test this, we compare the speed of the SNIC and the fastest of the other methods (see Fig. 6). All the algorithms tested are using C or C++ code provided by the authors [2, 3, 4, 6] and run on the same machine with 16 GB RAM and 2.6 GHz Intel Core i7 processor.

In the upper plot of Fig. 6, we compare average computation time taken by the algorithms against change in the number of superpixels. In the lower plot of Fig. 6, we compare the average computation time taken by the algorithm as the image size increases. Both these plots show that, the computational time is nearly linear in the number of pixels N in the image, i.e. SNIC exhibits $O(N)$ complexity in practice, which is similar in complexity to SLIC [7] and LSC [17].

5.4. Discussion of results

Referring to the plots in Fig. 5, the under-segmentation error (CUSE) of SNIC is low and compares with the best. In the precision-recall curve, SNIC shows the best performance, convincingly proving that SNIC adheres best to all object boundaries in the ground truth (high recall) but at the same time to only the true object boundaries (high precision). This fact is also confirmed by the F-measure plot, where SNIC clearly outperforms all the other methods compared with including SLIC. Interestingly, even though there are more recent algorithms, Fig. 5 and Fig. 6 show that SLIC continues to remain competitive in terms of quality and efficiency.

SNICPOLY polygon partitioning shows similarly encouraging performance. The CUSE values for SNICPOLY are significantly lower than CONPOLY [12]. In both the precision-recall and F-measure plots, the curves of SNICPOLY are significantly better than those of CONPOLY. In terms of computational efficiency, SNIC is the fastest algorithm after SEEDS. SNIC is faster than SLIC and LSC, the variant of SLIC we compared with (Fig. 6).

6. Conclusion

We introduce SNIC, an improved version of the well-known SLIC superpixel segmentation algorithm. Our algorithm retains the desirable properties of SLIC, namely computational efficiency, simplicity of implementation and use, and control over the number and compactness of superpixels. At the same time it overcomes the limitations of SLIC: our algorithm SNIC is non-iterative, explicitly enforces connectivity, is computationally cheaper, uses lesser memory, and yet outperforms SLIC on quantitative benchmarks. The resulting algorithm is a simplification of the original SLIC algorithm. We also present a polygonal partitioning algorithm that relies on SNIC superpixel boundaries. Like SNIC, the polygonal partitioning algorithm SNICPOLY also outperforms the state-of-the-art in terms of segmentation quality and speed.

References

[1] http://cs.brown.edu/ pff/segment/. 5

[2] http://ivrl.epfl.ch/research/superpixels. 5, 8

[3] http://jschenthu.weebly.com/projects.html. 5, 8

[4] https://github.com/akanazawa/collective-classification/tree/master/segmentation. 5, 8

[5] http://www.cs.toronto.edu/ babalex/research.html. 5

[6] http://www.mvdblive.org/seeds/. 5, 8

[7] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274—2282, 2012. 1, 2, 3, 5, 6, 7, 8

[8] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 33(5):898–916, May 2011. 7

[9] A. Bódis-Szomorú, H. Riemenschneider, and L. V. Gool. Superpixel meshes for fast edge-preserving surface reconstruction. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2011–2020, June 2015. 2, 3, 5

[10] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, May 2002. 2

[11] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973. 5

[12] L. Duan and F. Lafarge. Image partitioning into convex polygons. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3119–3127, June 2015. 2, 3, 5, 6, 7, 8

[13] P. Felzenszwalb and D. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision (IJCV)*, 59(2):167–181, September 2004. 2, 5, 6

[14] B. Fulkerson, A. Vedaldi, and S. Soatto. Class segmentation and object localization with superpixel neighborhoods. In *International Conference on Computer Vision (ICCV)*, 2009. 1

[15] S. Gould, J. Rodgers, D. Cohen, G. Elidan, and D. Koller. Multi-class segmentation with relative location prior. *International Journal of Computer Vision (IJCV)*, 80(3):300–316, 2008. 1

[16] A. Levinshtein, A. Stere, K. Kutulakos, D. Fleet, S. Dickinson, and K. Siddiqi. Turbopixels: Fast superpixels using geometric flows. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2009. 2, 5, 6

[17] Z. Li and J. Chen. Superpixel segmentation using linear spectral clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1356–1363, June 2015. 2, 5, 6, 8

[18] M.-Y. Liu, O. Tuzel, S. Ramalingam, and R. Chellappa. Entropy rate superpixel segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. 1, 2, 5, 6, 7

[19] Y.-J. Liu, C.-C. Yu, M.-J. Yu, and Y. He. Manifold slic: A fast method to compute content-sensitive superpixels. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 2

[20] D. Martin, C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Transactions on Pattern Analysis Machine Intelligence (PAMI)*, 26(5):530–549, 2004. 7

[21] 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 *IEEE International Conference on Computer Vision (ICCV)*, July 2001. 5

[22] M. Menze and A. Geiger. Object scene flow for autonomous vehicles. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 1

[23] A. Moore, S. Prince, J. Warrell, U. Mohammed, and G. Jones. Superpixel Lattices. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2008. 2

[24] G. Mori. Guiding model search using segmentation. In *IEEE International Conference on Computer Vision (ICCV)*, 2005. 1

[25] P. Neubert and P. Protzel. Superpixel benchmark and comparison. In *Proc. of Forum Bildverarbeitun, Regensburg, Germany*, 2012. 2, 5, 7

[26] I. Pitas. *Digital Image Processing Algorithms and Applications*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 2000. 5

[27] X. Ren and J. Malik. Learning a classification model for segmentation. In *IEEE Conference on Computer Vision (CVPR)*, 2003. 1

[28] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 22(8):888–905, Aug 2000. 2, 5, 6

[29] X. Sun, C. M. Christoudias, and P. Fua. Free-shape polygonal object localization. In *European Conference on Computer Vision (ECCV)*, pages 317–332, 2014. 3

[30] M. Van den Bergh, X. Boix, G. Roig, and L. Van Gool. SEEDS: Superpixels extracted via energy-driven sampling. *International Journal of Computer Vision*, 111(3):298–314, 2015. 2, 5, 6, 7

[31] A. Vedaldi and S. Soatto. Quick shift and kernel methods for mode seeking. In *European Conference on Computer Vision (ECCV)*, 2008. 2

[32] O. Veksler, Y. Boykov, and P. Mehrani. Superpixels and supervoxels in an energy optimization framework. In *European Conference on Computer Vision (ECCV)*, 2010. 2

[33] L. Vincent and P. Soille. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analalysis and Machine Intelligence*, 13(6):583–598, 1991. 2

[34] R. G. von Gioi, J. Jakubowicz, J. M. Morel, and G. Randall. Lsd: A fast line segment detector with a false detection control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(4):722–732, April 2010. 3

[35] S. Wang, H. Lu, F. Yang, and M.-H. Yang. Superpixel tracking. In *IEEE International Conference on Computer Vision (ICCV)*, Nov 2011. 1

[36] Y. Zhang, R. Hartley, J. Mashford, and S. Burn. Superpixels via pseudo-boolean optimization. In *IEEE International Conference on Computer Vision (ICCV)*, Nov 2011. 2

[37] C. L. Zitnick and S. B. Kang. Stereo for image-based rendering using image over-segmentation. *International Journal of Computer Vision (IJCV)*, 75:49–65, October 2007. 1