This is a modified version of paper published in Journal of Electronic Imaging 26(6), 061609 (Nov/Dec 2017)

Convex Constrained Meshes for superpixel segmentations of images

Jeremy Forsythe[a,*], **Vitaliy Kurlin**[b]
[a]Vienna University of Technology, Favoritenstr. 9-11 / E186, A-1040 Vienna, Austria
[b]Department of Computer Science, University of Liverpool, Liverpool L69 3BX, UK

Abstract. We consider the problem of splitting a pixel-based image into convex polygons with vertices at subpixel resolution. Edges of resulting polygonal superpixels can have any direction and should adhere well to object boundaries. We introduce a Convex Constrained Mesh that accepts any straight line segments and outputs a complete mesh of convex polygons without small angles and with approximation guarantees for the given lines. Experiments on the Berkeley Segmentation Dataset BSD500 show that the resulting meshes of polygonal superpixels outperform other polygonal meshes on boundary recall and pixel-based SLIC and SEEDS superpixels on undersegmentation errors.

Keywords: superpixel over-segmentation, line segment detection, convex polygonal mesh, constrained triangulation.

*Corresponding author jforsythe@cg.tuwien.ac.at

1 Introduction: Motivation, Problem Statement and Applications to Superpixels

This extended version of the conference paper[1] proves approximation guarantees for Convex Constrained Meshes (CCM) in Theorem 8 and affine invariance of optimal meshes in Theorem 10.

1.1 Motivation for Splitting Images into Convex Polygons at Subpixel Resolution

The traditional statement of the image over-segmentation problem is to group square-based pixels into superpixels that adhere well to object boundaries in the image. The boundaries of these pixel-based superpixels consist of short horizontal and vertical edges restricted to a given grid.

Since images represent a spatially continuous world, we argue that the segmentation problem should be solved in terms of functions defined over a *continuous image domain*, not over a discretization such as a regular grid. Due to anti-aliasing filtering, grayscale values across a real image edge rarely drop from 255 (white) to 0 (black), but change gradually over 2-3 pixels [2, Fig. 1]. Hence a real edge between objects is often not along pixel boundaries and should be considered in the infinite family of line segments that can have any slope and endpoints with real coordinates.

Fig 1 Left: image 106 from the Oxford Buildings Dataset[3] with 600K pixels. **Right**: 2159 CCM superpixels.

The next harder task is to extend line segments to a complete mesh whose polygons can be considered as superpixels at subpixel resolution. The brute-force approaches based on constrained triangulations often lead to polygons with small angles, because detected line segments can be almost parallel in real-life images containing long thin structures, which motivates the following.

Over-segmentation problem for polygonal superpixels at subpixel resolution.

Split a given pixel-based image into convex polygons such that

(1.1a) all internal angles of the polygons are larger than a reasonable lower bound and

(1.1b) the edges of the polygons well approximate all boundaries of objects in the image.

(1.1c) a reconstruction with colored polygons is similar to the original image as in Fig. 1.

The convexity requirement is motivated by the fact that many pixel-based superpixels have irregular shapes and zigzag boundaries that follow horizontal and vertical lines of a pixel grid.

1.2 Refining the Line Segment Detector and Building a Convex Constrained Mesh (CCM)

To build a mesh satisfying conditions (1.1a)–(1.1c) above, we have chosen the Line Segment Detector[4] (LSD), which outputs edges in grayscale images at subpixel resolution with theoretical guarantees. Since LSD edges are allowed to have intersections, we need to convert the LSD output into a properly embedded planar graph consisting of non-intersecting straight edges in the plane.

The CCM algorithm takes as **input** any grayscale image, consists of the following big stages below and **outputs** a mesh of convex polygons satisfying conditions (1.1a)–(1.1c) in the over-segmentation problem by Theorem 8, which was announced in Ref. 1 without proof.

Section 4: refinement of the Line Segment Detector (LSD) output with approximation guarantees.

Section 5: extension of the line segments to a mesh of convex polygons with no small angles.

The above stages of the CCM algorithm require the following parameters.

• Min_Distance (default 3 pixels) is an approximation tolerance of LSD edges by CCM edges.

• Min_Angle (default 20 degrees) is the minimum angle between adjacent edges in a final mesh.

1.3 Contributions and Applications to Superpixel Segmentations at Subpixel Resolution

The Convex Constrained Meshes are applied to segmenting images into polygonal superpixels. Traditional superpixel algorithms associate every square-based pixel to one of superpixels. This discrete output often contains superpixels with holes or several connected components.

A Convex Constrained Mesh outputs connected convex polygonal superpixels by construction. We introduce optimal constant colors for any polygonal mesh over a given pixel-based image in Definition 9. The resulting colored mesh can be rendered at any higher resolution, see Fig. 2.

Fig 2 **Left**: input. **Middle**: 275 Voronoi cells,[5] $nRMS \approx 10.2\%$. **Right**: 246 CCM superpixels, $nRMS \approx 4.48\%$.

The resulting colored mesh well approximates an original image by using much fewer convex polygons instead of square-based pixels. Fig. 2 shows that only 246 convex polygons approximate the original 512×512 image with a small reconstruction error $nRMS$ introduced in Definition 9.

Here is a summary of the contributions to the state-of-the-art for image over-segmentation.

• The over-segmentation problem is studied at subpixel resolution with the extra restrictions on convexity, small angles and approximations, see conditions (1.1a)–(1.1c) in subsection 1.1.

• The LSD refinement in section 4 can be applied to any collection of potentially intersecting segments and outputs a planar straight line graph with approximation guarantees by Theorem 5.

• The Convex Constrained Meshes can be built without angles smaller than $\text{Min\_Angle} \leq 20.7°$ so that the original line segments are within the Min_Distance-offset of the mesh by Theorem 8.

• The CCMs outperform other polygonal superpixels on the Reconstruction Error (justified by Theorem 10) and Boundary Recall from the Berkeley Segmentation Database,[6] see section 6.

Section 2 discusses the related past work on superpixels. Section 3 introduces key concepts necessary for proofs of main Theorems 5 and 8. Section 4 resolves intersections and small angles of any line segments. Section 5 extends any straight line planar graph to a mesh of convex polygons. Section 6 evaluates CCM superpixels on 500 images from the Berkeley Segmentation Database.

2 Related Work on Superpixels, Line Segment Detection and Polygonal Meshes

2.1 Traditional Superpixels consisting of Square-based Pixels restricted to a Pixel Grid

A pixel-based image is represented by a lattice L whose nodes are in a 1–1 correspondence with all pixels, while all edges of L represent adjacency relations between pixels. Usually each pixel is connected to its closest 4 or 8 neighbors. The seminal *Normalized Cuts* algorithm by Shi and Malik[7] finds an optimal partition of L into connected components, which minimizes an energy taking into account all nodes of L. The Entropy Rate Superpixels (ERS) of Lie et al.[8] minimize the entropy rate of a random walk.

The *Simple Linear Iterative Clustering* (SLIC) algorithm by Achanta et al.[9] forms superpixels by k-means clustering in a 5-dimensional space using 3 colors and 2 coordinates per pixel. The search is restricted to a neighborhood of a given size, so the complexity is $O(kmn)$, where n and m are the numbers of pixels and iterations. The clustering approach was further developed by Li and Chen,[10] by Giraud et al.[11] in SCALP superpixels, by Buyssens et al.[12] in Eikonal superpixels.

SEEDS (Superpixels Extracted via Energy-Driven Sampling) by Van den Bergh et al.[13] seems the first superpixel algorithm to use a *coarse-to-fine optimization*. The colors of all pixels within each fixed superpixel are put in bins, usually 5 bins for each color channel. Each superpixel has the associated sum of deviations of all bins from an average bin within the superpixel. This sum is maximal for a superpixel whose pixels have colors in one bin. SEEDS iteratively maximizes the sum of deviations by shrinking or expanding superpixels. Another energy function is similarly optimized from coarse to finer levels by Yao et al.[14] in Coarse-to-Fine superpixels.

Royer et al.[15] introduced convexity constraints in a dicrete setting for superpixels consisting of square-based pixels. Waterpixels by Machairas et al.[16] used a spatially regularized gradient for a

tradeoff between the superpixel regularity and the adherence to object boundaries.

The key limitation of pixel-based superpixels is their restriction to a given pixel grid. Any diagonally oriented object will have a zigzag superpixel boundary consisting of only short horizontal or vertical edges, because other directions are not allowed. Hence measuring lengths of diagonal objects using pixel-based superpixels can have a multiplicative error up to $\sqrt{2}$. The only way to measure lengths more accurately is to allow superpixels whose edges may have any direction. Accurate measurements of boundaries are needed in many applications from satellites to medicince.

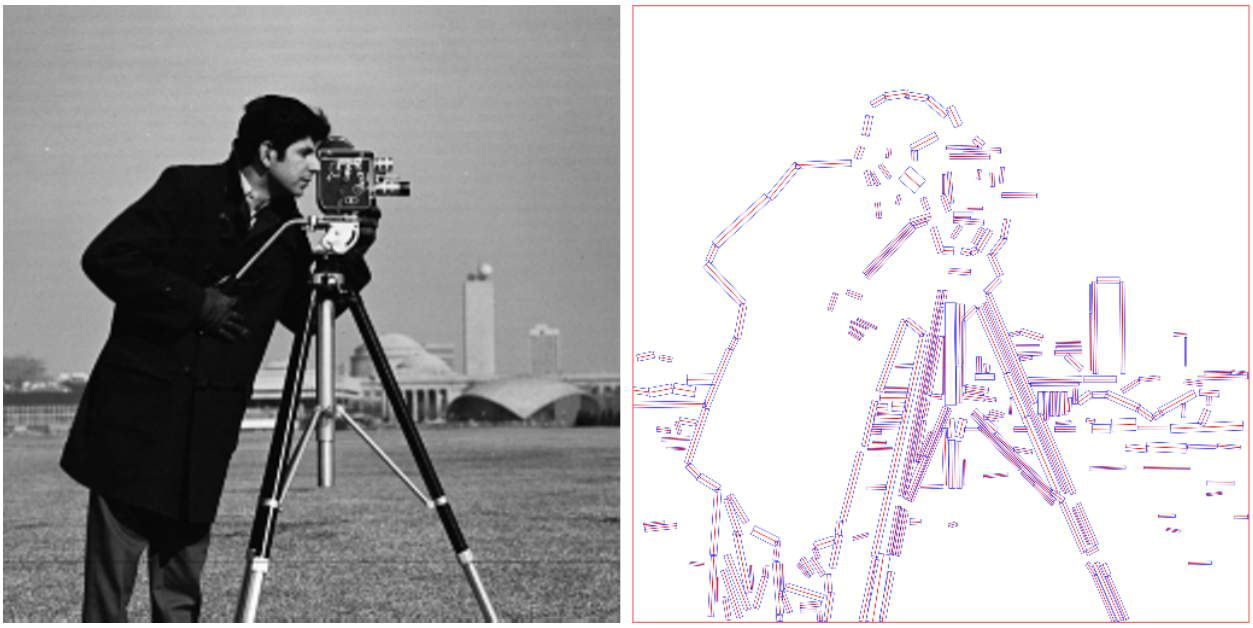

Fig 3 Left: input. **Right**: 259 LSD red middle segments in blue rectangles before the refinement in Section 4.

2.2 The Line Segment Detector (LSD) and Shewchuk Triangulations without Small Angles

The Line Segment Detector by Grompone von Gioi et al.[44] finds edges at subpixel resolution in grayscale images. The output consists of thin blue rectangles whose long middle red lines are detected line segments, see Fig. 3. Adjacent pixels with similar gradients are clustered and every resulting cluster is approximated by a rectangle which might have any position on the plane. The main LSD parameter is a tolerance τ for angles between gradients of adjacent pixels.

Fig 4 Top: 589 Voronoi superpixels (boundary mesh and reconstructed image) have $nRMS \approx 9.22\%$. **Bottom**: 416 CCM superpixels (boundary mesh and reconstructed image) have the smaller error $nRMS \approx 6.32$ from Definition 9.

We use the state-of-the-art Triangle C++ software,[17] which outputs Shewchuk triangulations satisfying a large lower bound for all angles. A Convex Constrained Mesh extends Shewchuks triangulation to a general mesh of convex polygons for the same $\mathrm{Min\_Angle} = 20^\circ$ as in Ref. 17.

2.3 Polygonal superpixels based on Voronoi meshes

Duan and Lafarge[5] used Voronoi meshes to make a step change by introducing the first polygonal superpixels at subpixel resolution. These *Voronoi superpixels* are obtained by splitting an image into Voronoi cells whose centers are chosen along LSD edges. If there are several LSD edges that are near parallel and close to each other, only one of them is included into a final Voronoi mesh. Fig. 4 shows that the resulting Voronoi mesh misses long thin structures such as legs of a camera tripod. So the LSD edges were considered as soft constraints without proven guarantees yet.

3 Key Definitions: Planar Graphs and Polygonal Meshes

This section defines concepts that are necessary for the later algorithms and theorems. Definition 1 introduces a *PL complex* that formally covers all the essential cases including

- the input for the CCM algorithm (a set of points and constrained line segments);
- any PSLG (a planar straight line graph) as defined in Ref. 17;
- Steiner Delaunay[18] triangulations and Shewchuk triangulations;[17]
- the final output of the CCM algorithm (a Convex Constrained Mesh CCM).

Definition 1. *[18, Def 2.8] A* (PL) complex C *is a finite set of vertices, edges and polygons (faces) such that*

- *if* C *contains an edge* e*, then* C *contains both endpoints of* e*;*
- *the boundary of any face is a union of edges from* C*;*
- *edges from* C *can meet only at their common endpoint;*
- *faces from* C *can share only common edges and vertices.*

The domain $|C| \subset \mathbb{R}^2$ *of the PL complex* C *is the area covered by all vertices, edges and faces of* C*. If the complex* C *has no faces, we call* C *a* Planar Straight Line Graph *(PSLG)[17] or (briefly) a graph.*

A PL complex C can consist of disconnected line segments. Definition 2 extends C using extra vertices (called *Steiner* points) to a full triangulation T. All edges of C (possibly subdivided in T) will be called *constrained*, all other edges of T are *unconstrained*.

Definition 2. *[18, 2.15] A* Steiner constrained triangulation *of a PL complex C is a PL complex* $\mathrm{SDT}(C)$ *such that*

(2a) $\mathrm{SDT}(C)$ *has only triangular faces and covers* $|C| \subset \mathbb{R}^2$,

(2b) if all edges of the given complex C are opaque, then for any triangle T the open circumdisk of T covers no vertices of C visible from the interior of the triangle T.

Condition (2a) means that all faces of C are subdivided into triangles. An edge of C can be subdivided into shorter edges and the domain of $\mathrm{SDT}(C)$ may not be convex. Condition (2b) guarantees that $\mathrm{SDT}(C)$ contains only nice triangles, e.g. a quadrilateral in Fig. 5 should be split by a shorter diagonal rather than a longer one.

If a PL complex C is a finite set of points, then $\mathrm{SDT}(C)$ is a classical Delaunay triangulation, which is dual to the Voronoi mesh in Definition 3 below. Let d be the Euclidean distance. For a set $S \subset \mathbb{R}^2$, let $d(p, S) = \inf\{\, d(p, q) \,:\, q \in C \,\}$ be the distance from a point p to S.

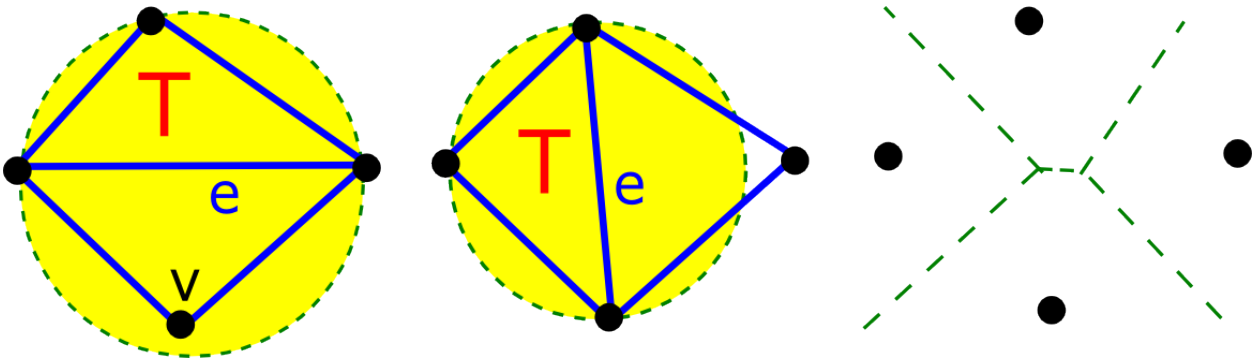

Fig 5 Left: the yellow circumdisk of T contains a vertex v. **Middle**: the circumdisk of T contains no vertices, both triangles belong to $\mathrm{SDT}(C)$. **Right**: the top and bottom Voronoi cells are adjacent, so their centers are connected by e.

Definition 3. *For a set of points $C = \{p_1, \dots, p_n\}$, the* Voronoi cell $V(p_i) = \{q \in \mathbb{R}^2 \,:\, d(q, p_i) \le d(q, C - p_i)\}$ *is the closed neighborhood of p_i consisting of points $q \in \mathbb{R}^2$ that are non-strictly closer to the* site p_i *than to other points of $C - p_i$. The splitting of the plane into Voronoi cells $V(p_1) \cup \dots \cup V(p_n)$ is called the* Voronoi mesh*, see Fig. 5.*

Then a Delaunay triangulation $\mathrm{DT}(C)$ *on the vertex set C has*

- *an edge between p_i, p_j if the faces $V(p_i) \cap V(p_j) \ne \emptyset$,*
- *a triangle on p_i, p_j, p_k if $V(p_i), V(p_j), V(p_k)$ share a point.*

Definition 4 formally introduces the main concept of a Convex Constrained Mesh (CCM).

Definition 4. *Let G be a planar straight line graph with all angles between adjacent edges at least $\varphi \le 60°$. A* Convex Constrained Mesh $\mathrm{CCM}(G)$ *is a piecewise linear complex such that*

(4a) $\mathrm{CCM}(G)$ *has convex polygons with angles* $\ge \mathrm{Min\_Angle} = \arcsin\left(\dfrac{1}{\sqrt{2}} \sin\dfrac{\varphi}{2}\right)$*;*

(4b) the graph G is covered by the edges of the Convex Constrained Mesh $\mathrm{CCM}(G)$*.*

Any Shewchuk triangulation is an example of a Convex Constrained Mesh. However, Definition 4 allows general meshes of any convex polygons without intersections and with no small angles. We build CCM by converting the LSD output into a planar graph G (Section 4) without self-intersections and then by extending G into a polygonal mesh without small angles.

4 Refinement of the Line Segment Detector to Resolve Intersections and Small Angles

The input for the LSD algorithm[4] is a grayscale image. The output is an unordered set of thin rectangles in the plane. We consider only the long middle lines of these rectangles as the red constrained edges and also add the boundary edges of the whole image, see Fig. 3.

The resulting red segments may intersect each other, go outside the boundary of the image or form small angles. Subsections 4.1–4.3 describe how to refine the LSD output and get a graph G without angles smaller than a given bound φ. We define the *strength* of a line segment S as $+\infty$

if S is on the boundary of the image, else the *strength* is the length of S. We apply each of the refinements to line segments ordered by their strength. So the order of refinements is not random and depends only on line segments that were detected by LSD. In each of subsections 4.1–4.3 below all the listed steps are performed for every pair of line segments from the LSD output.

4.1 Extending Segments to Line-Segment Intersections

If an endpoint v of one segment is very close to another segment, then a Shewchuk triangulation will have many tiny triangles at the vertex v to avoid small angles. The steps below resolve this singular case by inserting a proper intersection, see Algorithm 1.

Algorithm 1 Extending segments to line-segment intersections in Steps (4.1a)–(4.1d)

Input: Line segments $S_i \in S$
Output: Refined line segments S_R

```
for all S_1 ∈ S do
    Take the infinite line L(S_1)
    Find all intersection points p on segments S_2 ∈ S intersecting the two rays L(S_1) − S_1
    Take S_2 whose p is closest to S_1
    if d(p, S_1) < Min_Distance then
        Find acute angle θ between L(S_1) and S_2.
        if θ ≥ φ then
            Extend S_1 to p
        end if
        if θ < φ then
            Find a point q such that d(q, S_2) = Min_Distance
            if q exists then
                Shorten S_1 to q
            else
                Remove S_1
            end if
        end if
    end if
end for
S_R ← S
```

Step (4.1a) For any straight segment S_1, we take the infinite line $L(S_1)$ through S_1, see Fig. 6. We

find all segments S_2 intersecting the two rays $L(S_1) - S_1$ (the line $L(S_1)$ with S_1 removed).

Step (4.1b) Among all intersection points of the two rays $L(S_1) - S_1$, we find the segment S_2 whose intersection p with $L(S_1) - S_1$ is closest to S_1. This choice of p guarantees that if we extend S_1 to p, then no new intersections with other segments are created. Steps (4.1c) and (4.1d) below work similarly for the intersection point closest to another endpoint of S_1.

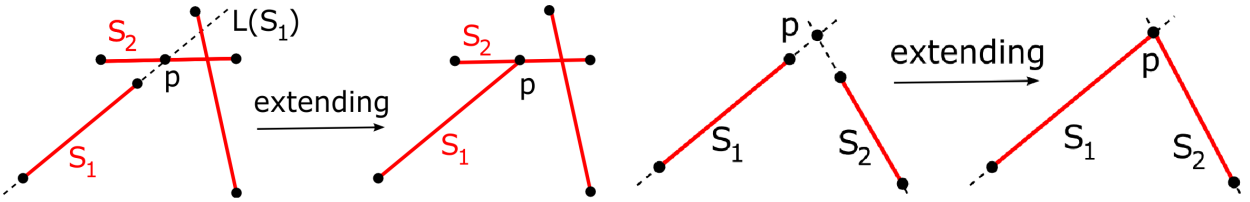

Fig 6 Left: extend S_1 to the point $p = L(S_1) \cap S_2$. **Right**: extend segments S_1 and S_2 to the point $p = L(S_1) \cap L(S_2)$.

Step (4.1c) If $d(p, S_1) > \text{Min\_Distance}$, we stop considering p. Otherwise we find the acute angle θ between $L(S_1)$ and S_2. If $\theta \geq \varphi$, we extend S_1 to the new vertex p, which splits S_2 into two new segments, see the left hand side picture of Fig. 6. If any of these new segments within S_2 is shorter than Min_Distance, we remove this segment together with its endpoint different from p.

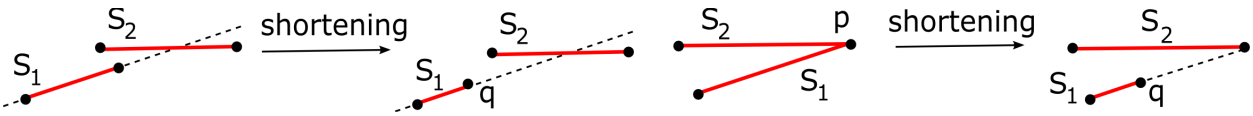

Fig 7 Left: shorten a segment S_1 to the new endpoint q at the distance $d(q, S_2) = \text{Min\_Distance}$. **Right**: shorten a segment S_1 by pulling its endpoint away from S_2 to avoid a small angle $\theta < \varphi$ between the segments S_1, S_2.

Step (4.1d) If $\theta < \varphi$, we find a point $q \in S_1$ with $d(q, S_2) = \text{Min\_Distance}$. If q exists, we shorten S_1 to the new endpoint q, see Fig. 7. Otherwise we remove the whole segment S_1, because S_1 is covered by the Min_Distance-offset of S_2.

4.2 Extending Segments to Line-Line Intersections

If two segments don't intersect as in subsection 4.1, but have very close endpoints, we again avoid tiny triangles later inserting a proper intersection. Similarly to Step (4.1), we find intersection

points within $\mathrm{Min\_Distance}$. Now we take the intersection of the infinite lines $L(S_1)$ and $L(S_2)$ outside S_1, S_2. These steps are described in Algorithm 2 below.

Algorithm 2 Extending segments to line-line intersections in Steps (4.2a)–(4.2c)

Input: Line segments $S_i \in S$
Output: Refined line segments S_R

for all $S_1 \in S$ **do**
  Take the infinite line $L(S_1)$ and $L(S_2)$ for all other segments $S_2 \in S$
  Take all S_2 such that either of the rays $L(S_1) - S_1$ meets one of the rays $L(S_2) - S_2$
  for all Intersections of $L(S_1) - S_1$ and $L(S_2) - S_2$ choose p closest to an endpoint of S_1 **do**
    if $d(p, S_1) < \mathrm{Min\_Distance}$ and $d(p, S_2) < \mathrm{Min\_Distance}$ **then**
      Find the acute angle θ between $L(S_1)$ and $L(S_2)$.
      if $\theta \geq \varphi$ **then**
        Extend S_1 and S_2 to the new vertex p
      end if
      if $\theta < \varphi$ **then**
        Move endpoint of the weaker segment to a point q such that $d(q, S_2) = \mathrm{Min\_Distance}$
      end if
    end if
  end for
end for
$S_R \leftarrow S$

Step (4.2a) For a segment S_1, we consider the infinite line $L(S_1)$ through S_1, and the lines $L(S_2)$ through all the other segments. Then we find all segments S_2 such that either of the rays $L(S_1) - S_1$ meets one of the rays $L(S_2) - S_2$.

Step (4.2b) Among all intersections of $L(S_1) - S_1$, $L(S_2) - S_2$, we choose a point p closest to an endpoint of S_1, then do Step (4.2c) for both endpoints of S_1.

Step (4.2c) If $d(p, S_1) < \mathrm{Min\_Distance}$ and $d(p, S_2) < \mathrm{Min\_Distance}$, we find the acute angle θ between $L(S_1)$ and $L(S_2)$. If the angle $\theta \geq \varphi$, we extend S_1 and S_2 to the new vertex p, see Fig. 6. If the angle $\theta < \varphi$, we move the endpoint of the weaker segment to a point q such that $d(q, S_2) = \mathrm{Min\_Distance}$, see Fig. 7.

4.3 Splitting Line Segments at Intersection Points

Many segments in the LSD output may actually intersect, so the steps below insert this intersection point to get a planar graph without self-intersections, see Algorithm 3.

Algorithm 3 Splitting line segments at intersection points in Steps (4.3a)–(4.3d)

Input: Line segments $S_i \in S$
Output: Refined line segments S_R

```
for all S_1, S_2 ∈ S do
    if S_1, S_2 intersect at a point p internal to both S_1,S_2 then
        Split the segments into four smaller segments
        Remove new segments shorter than Min_Distance
        Calculate θ, the smallest angle between remaining segments
        if θ < φ then
            Shorten the weaker segment S_1 such that d(S_1, S_2) = Min_Distance
        end if
    end if
end for
S_R ← S
```

Step (4.3a) For each pair of segments S_1, S_2, we check if S_1, S_2 intersect at a point p that is internal to both S_1, S_2. If a new segment is shorter than $\mathrm{Min\_Distance}$, we remove it together with its endpoint different from p, see Fig. 8.

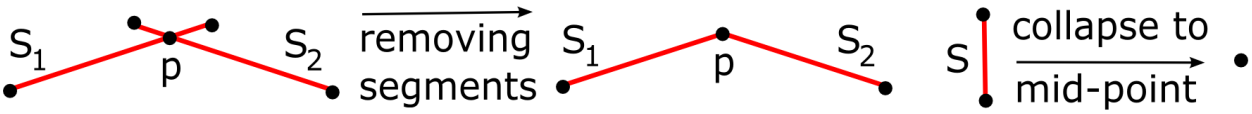

Fig 8 Removing new segments and collapsing segments shorter than $\mathrm{Min\_Distance}$.

Step (4.3b) Let θ be the smallest angle between the remaining segments (also denoted by S_1, S_2) with the common endpoint p. If $\theta \geq \varphi$, we stop considering the point p. The steps below similarly work for the symmetric angle θ at p.

Step (4.3c) If $\theta < \varphi$, we shorten the weaker segment S_1 to make the distance $d(S_1, S_2) = \mathrm{Min\_Distance}$ as in (4.1d), see the right hand side picture of Fig. 7.

Step (4.3d) We collapse any isolated edge shorter than $\mathrm{Min\_Distance}$ to its mid-point and remove all non-isolated edges shorter than $\mathrm{Min\_Distance}$, see Fig. 8.

4.4 Approximation Guarantees for the LSD refinement

We further justify all the steps in subsections 4.1–4.3 by the following result which shows that the graph produced has no small angles between its edges and is close to the original line segments.

Theorem 5. *Let line segments $S_1, \ldots, S_k$ have m intersections. The LSD refinement described in subsections 4.1–4.3 outputs a planar straight line graph G with $O(k+m)$ edges in time $O((k+m)\log k)$ such that*

(5a) any angle in the graph G between adjacent edges is not smaller than φ;

(5b) the union $\cup_i S_i$ of segments is covered by the $\mathrm{Min\_Distance}$*-offset of G.*

Proof. Due to subsection 4.3, all final segments meet only at their endpoints. A line segment may intersect any other segment only once. Any new intersection may generate two extra segments, so G has at most $O(k+m)$ edges. There are $m = O(k^2)$ intersections of k segments in the worst case. In practice, any segment S detected by LSD meets at most two segments, only one near each endpoint of S, so $m = O(k)$. The output-sensitive swipe line algorithm [19, chapter 2] finds all intersections between segments in time $O((k+m)\log k)$ and can be extended to line-segment intersections in Step (4.1a). Steps (4.1d), (4.3c) guarantee that all angles are not smaller than the given lower bound φ. A segment S_1 can become longer by at most $\mathrm{Min\_Distance}$ in Step (4.1c) and shorter in Steps (4.1d), (4.2c), (4.3c), (4.3d). The removed part of S_1 is in the $\mathrm{Min\_Distance}$-offset of a stronger segment S_2, which can't be shortened later. □

5 A Convex Constrained Mesh Without Small Angles

5.1 Fast Shewchuk Triangulations Without Small Angles

Any planar straight line graph $G \subset \mathbb{R}^2$ can be the input for Shewchuk's Triangle code.[17] The output is a Steiner constrained Delaunay Triangulation $T(G)$ without small angles.

Theorem 6. *[17, Theorem 12] For a planar straight line graph G having n vertices and no angles smaller than $\varphi \leq 60°$, in time $O(n \log n)$ one can build a triangulation $T(G)$ without angles smaller than* $\text{Min\_Angle} = \arcsin\left(\frac{1}{\sqrt{2}} \sin\frac{\varphi}{2}\right)$.

If $\varphi = 60°$, then $\text{Min\_Angle} = \arcsin\left(\frac{1}{\sqrt{2}} \sin\frac{\varphi}{2}\right) \approx 20.7°$. If a graph G has angles smaller than Min_Angle, they are preserved in a Shewchuk triangulation. So the LSD refinement in section 4 is needed to prove the main results in Theorem 8 later in subsection 5.4.

We use OpenMesh to store $T(G)$ and merge triangles into convex faces as described below.

5.2 Simple and Advanced Merge Operations to Get Convex Faces

Here we process unconstrained edges in decreasing order of length. The steps below are motivated by the aim to simplify the mesh and get a smaller number of superpixels keeping them convex.

Hereafter, we will use different colors to characterize the edges: red, blue and black. These colors are used to highlight constrained, unconstrained and unknown type edges, respectively.

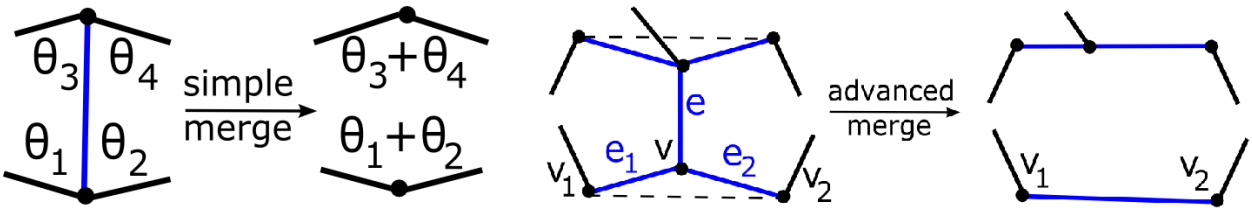

Fig 9 Left: a simple merge by removing an unconstrained edge e between two faces if the new larger face is convex. **Right**: an advanced merge by removing an unconstrained edge e between two faces if each of its endpoints can be resolved by Step (5.2b) or (5.2c) so that the new larger face is convex.

Algorithm 4 Simple and advance merge operations in Steps (5.2a)-(5.2d)

Input: Mesh M with edges $e \in M$
Output: Simplified mesh M_S

```
for all Unconstrained edges e ∈ M do
    Sort by decreasing order of length
    Find the four internal angles θ1, θ2, θ3, θ4 at the endpoints
    if θ1 + θ2 ≤ 180° AND θ3 + θ4 ≤ 180° then
        Remove edge e
    end if
    if θ1 + θ2 > 180° then
        if The edges adjacent to the two angles are unconstrained AND no small angles would
be created then
            Remove these edges and replace with a single edge
        end if
    end if
    if θ3 + θ4 > 180° then
        if The edges adjacent to the two angles are unconstrained AND no small angles would
be created then
            Remove these edges and replace with a single edge
        end if
    end if
end for
M_S ← M
```

Step (5.2a) For each unconstrained edge e, we find 4 angles $\theta_1, \theta_2, \theta_3, \theta_4$ at its endpoints in Fig. 9.

Step (5.2b) If $\theta_1 + \theta_2 \leq 180^\circ$ and $\theta_3 + \theta_4 \leq 180^\circ$, the convexity is preserved at both endpoints of e, so we remove e and the new larger face is convex.

If one of the angles $\theta_1 + \theta_2, \theta_3 + \theta_4$ in Step (5.2b) is greater than 180°, the simple merge operation along the edge e leads to a non-convex face. Then we try to make the *triangular cut* in Step (5.2c) without disturbing constrained edges.

Step (5.2c) Assume that $\theta_1 + \theta_2 > 180^\circ$ at the vertex v in Fig. 9, and both edges e_1, e_2 are unconstrained. Then we try replacing $e_1 \cup e_2$ by the new unconstrained edge connecting v_1, v_2 in the last picture of Fig. 9. If no angle becomes smaller than $\mathrm{Min\_Angle}$ or larger than 180°, then this *triangular cut* is successful.

Step (5.2d) If all angles are in $[\mathrm{Min\_Angle}, 180^\circ]$ after removing the edge e in Fig. 9, we finish Step (5.2c), else we reverse Step (5.2c) to avoid small angles.

After Step (5.2d) we check if any new edges can be removed by simple merge operations, which further simplifies the mesh. Finally, at any degree 2 vertex with a 180° angle, we replace two adjacent edges by one longer straight edge.

5.3 Collapsing unconstrained edges for a further simplification

We process unconstrained edges of the mesh in the increasing order of length. Both endpoints of any constrained edge are called *fixed* vertices. We will perturb only *non-fixed* vertices whose incident edges are all unconstrained. The steps are set out in Algorithm 5.

Algorithm 5 Collapse unconstrained edges in Steps (5.3a)–(5.3b) below

Input: Mesh M with edges $e \in M$
Output: Simplified mesh M_S

```
for all Unconstrained edges e ∈ M do
    Sort by decreasing order of length
    if e has an endpoint v which is not fixed then
        Collapse the edge e towards the opposite endpoint w
        if A non-convex face around w OR an angle θ < Min_Angle has been created then
            Cancel the collapse, consider the collapse in the opposite direction from w to v
        end if
    end if
end for
M_S ← M
```

Step (5.3a) If an endpoint v of an unconstrained edge e is not fixed, we try to collapse the edge e from the endpoint v towards the opposite endpoint w.

Step (5.3b) If any of the faces around the vertex w is non-convex or has an angle smaller than $\mathrm{Min\_Angle}$, we cancel this collapse and consider the opposite edge directed from w to v, or the next edge from the ordered list above.

If the edge e belongs to a triangle, this triangle also collapses, which reduces the number of faces without affecting constrained edges. The average decrease of the number of faces due to collapses above is 10% across BSD500 images.

5.4 Theoretical Guarantees for a Convex Constrained Mesh (CCM)

The following result justifies all the steps in subsections 5.2–5.3.

Lemma 7. *For any planar straight line graph G with n vertices and without angles smaller than $\varphi \le 60°$, the algorithm in subsections 5.2–5.3 build in time $O(n \log n)$ a Convex Constrained Mesh $CCM(G)$ such that*

(7a) $\mathrm{CCM}(G)$ *has no angles* $\theta < \mathrm{Min\_Angle} = \arcsin\left(\frac{1}{\sqrt{2}} \sin\frac{\varphi}{2}\right)$;

(7b) the graph G is fully covered by the edges of $\mathrm{CCM}(G)$.

Proof. Theorem 6 guarantees no small angles in $T(G)$ built in time $O(n \log n)$. All steps in section 5 check that $\mathrm{CCM}(G)$ has no angles $\theta < \mathrm{Min\_Angle}$. All edges of G are kept by the merge operations and can become longer only in Step (5.3c), so the edges of CCM cover G. □

Theorem 8. *Let line segments $S_1, \dots, S_k$ have m intersections. Then the algorithms in sections 4 and 5 build a Convex Constrained Mesh in time $O((k+m)\log(k+m))$ such that*

(8a) any internal angle in any CCM *polygon is not smaller than* Min_Angle;

(8b) the union $\cup_i S_i$ is covered by the Min_Distance*-offset of the* CCM*'s edges.*

Proof. Theorem 5 in time $O((k+m)\log k)$ builds a planar straight line graph G with $O(m)$ vertices and angles not smaller than $\varphi = 2\arcsin(\sqrt{2}\sin \mathrm{Min\_Angle})$. Lemma 7 in time $O(n \log n)$ for $n = O(k+m)$ builds $\mathrm{CCM}(G)$ without angles smaller than $\arcsin\left(\frac{1}{\sqrt{2}} \sin\frac{\varphi}{2}\right) = \mathrm{Min\_Angle}$. Finally, conditions (5b) and (5b) of earlier Theorem 5 imply condition (8b). □

6 Experiments on 500 BSD Images, Conclusions and Further Problems

Subsections 6.1 and 6.2 compare the sizes and reconstruction errors of the CCM and Voronoi superpixels. Subsection 6.3 also includes two more superpixel algorithms SLIC[9] and SEEDS[13] for the benchmarks based on human drawings in the Berkeley Segmentation Database[6] (BSD).

6.1 Sizes of CCMs, Shewchuk's Triangulations and Voronoi meshes

The first picture in Fig. 10 is the original LSD output. The second picture shows the LSD refinement in Section 4. The refined LSD output has more edges than the original LSD, because we include boundary edges of images and also intersection points, which become vertices of graphs.

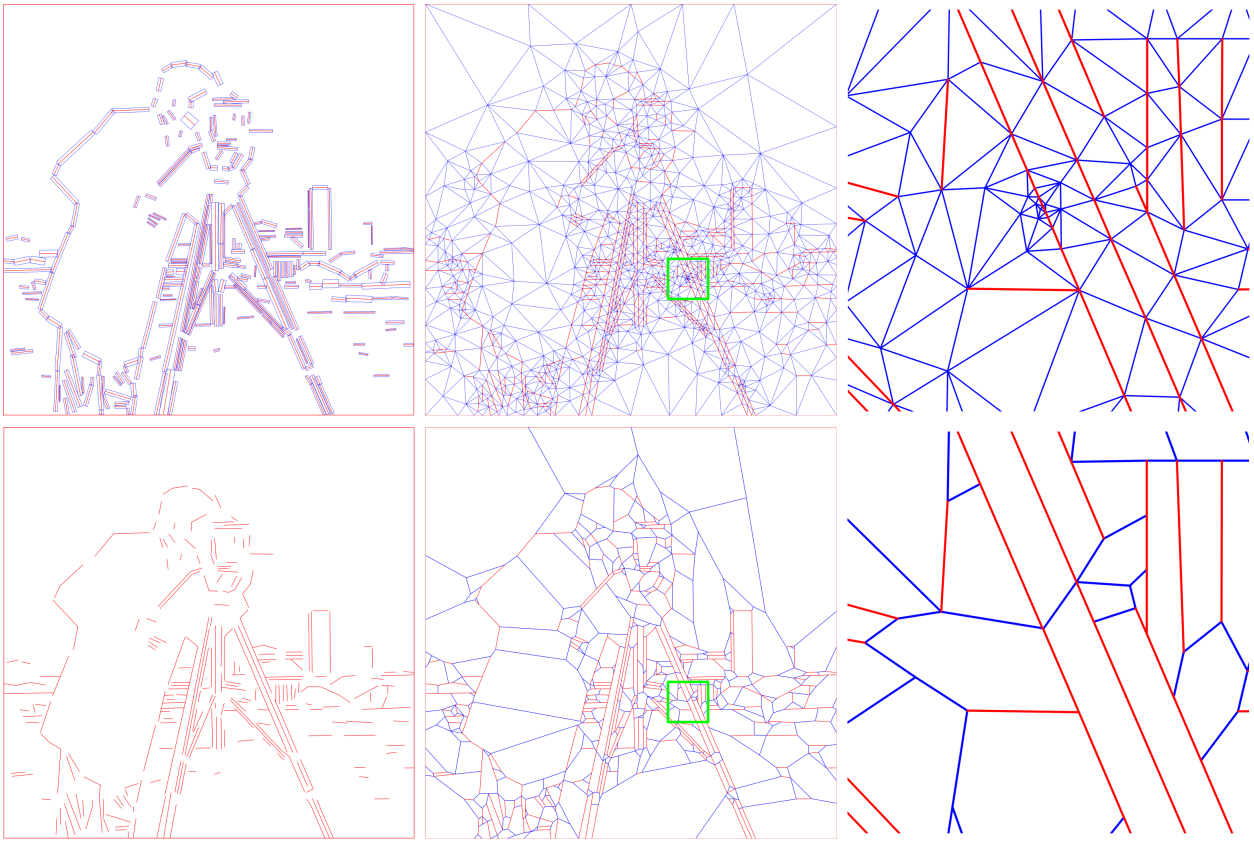

Fig 10 **Top left**: 259 LSD red middle segments in blue rectangles before the refinement. **Bottom left**: the refined LSD output (a graph G) with 294 edges. **Top middle**: Shewchuk triangulation $T(G)$ with 2260 triangles. **Bottom middle**: the Convex Constrained Mesh $\mathrm{CCM}(G)$ with 416 superpixels. **Top right**: zoomed in green box with tiny triangles. **Bottom right**: zoomed in green box, all tiny triangles are merged into convex polygonal superpixels.

Table 1 Ratios of superpixels for CCM and Voronoi meshes on the same LSD edges, averaged across 500 BSD images.[6] Eps_Radius (approximate size of Voronoi superpixels) of the Duan and Lafarge algorithm[5] is in pixels.

Eps_Radius of Voronoi superpixels	3	4	5	6	7	8	9	10	11	12
Mean $\frac{\text{number of Voronoi superpixels}^5}{\text{number of CCM superpixels}}$	14.09	8.91	6.21	4.86	4.03	3.96	3.43	3.27	3.27	3.26

We use $\varphi = 30°$ for the LSD refinement, which leads to $\text{Min\_Angle} \approx 10.5°$ in Shewchuk's Triangle.[17] We compare Shewchuk triangulations on the original LSD output and CCM on the refined LSD output in Fig. 10, where the 3rd picture shows a zoomed-in green box with many tiny triangles. The final picture in Fig. 10 contains relatively fewer faces after the merging operations in Section 5. The ratio of Shewchuk triangles to the number of CCM polygons across BSD is 7.6.

The first step for Voronoi superpixels[5] is to post-process the LSD output when close and near parallel lines are removed, because the target application was satellite images of urban scenes with many straight edges of buildings. Then long thin structures such as legs of a camera tripod in Fig. 10 are represented only by one edge and may not be recognized in any further processing.

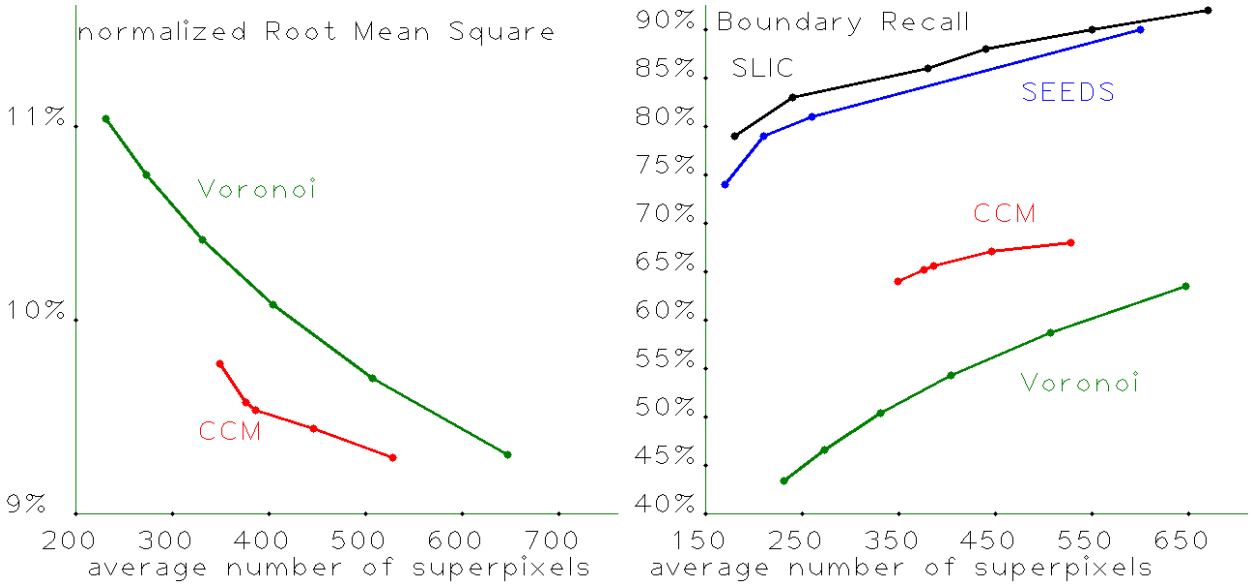

Fig 11 Left: normalized Root Mean Square in percents for Voronoi and CCM superpixels. **Right**: Boundary Recall.

That is why the LSD refinement in section 4 follows another approach leading to Theorem 8.

Table 1 displays the average ratios of numbers of superpixels over BSD images. Even when the parameter $\mathrm{Eps\_Radius}$ is increased to 12, these ratios converge to a factor of about 3.25.

6.2 Approximation Quality and Affine Invariance of the CCM and Past Superpixels

The key advantage of polygonal superpixels at subpixel resolution is the possibility to render such a polygonal mesh at any higher resolution, because vertices have real coordinates and straight edges are not restricted to a given pixel grid. Definition 9 introduces optimal colors for this rendering.

Definition 9. *Let an image I consist of n pixels and let each pixel be the 1×1 square B_p with Intensity$(p) \in [0, 255]$. Let I split into superpixels F_j (convex polygons) for some unknown constant Color$(F_j) \in [0, 255]$, $j = 1, \dots, s$.*

$$\textit{The } \text{Reconstruction Error } \textit{is } RE = \min \sum_{\textit{pixels } p} \Big(\textit{Intensity}(p) - \sum_{j=1}^{s} \textit{Area}(B_p \cap F_j) \textit{Color}(F_j) \Big)^2, \tag{9a}$$

where the minimum is over all Color(F_j), $j = 1, \dots, s$. The internal sum in RE is small, because each square B_p non-trivially intersects only few superpixels F_j, so the intersection Area$(B_p \cap F_j)$ is almost always 0 (when B_p is outside F_j) or 1 (when F_j covers B_p). For a fixed splitting $I = \cup_{j=1}^{s} F_j$, the function RE quadratically depends on Color(F_j), which are found from a linear system. The reconstructed image *is the superpixel mesh with optimal Color(F_j) minimizing $nRMS$ below. This colored mesh can be rendered at any resolution, see Fig. 4.*

$$\textit{The } \text{normalized Root Mean Square } \textit{of the approximation by superpixels is } nRMS = \sqrt{\frac{RE}{n}} \cdot \frac{100\%}{255}. \tag{9b}$$

The Reconstruction Error in (9a) makes sense not only for images consisting of square pixels B_p, but for any colored meshes. Indeed, the regions B_p can be any polygons, e.g. obtained from square pixels by an affine transformation. Neubert and Protzel[20] proposed to study stability of superpixels with respect to affine transformations, which motivated us to prove Theorem 10 justifying the Reconstruction Error as the objective quality measure for polygonal superpixels.

Theorem 10. *Let $M(I)$ be a polygonal mesh (globally) minimizing the Reconstruction Error in (9a) for an image I. Let T be an affine transformation so that the new image is $T(I) \subset I$. Assuming that the complement $I - T(I)$ is black, the optimal mesh minimizing the Reconstruction Error for $T(I)$ is $T(M(I))$ obtained from the optimal mesh $M(I)$ by the same affine transformation T.*

Proof. The Reconstruction Error for the transformed image $T(I)$ is defined by formula (9a), where the sum is over the transformed pixels $T(B_p)$ instead of original square pixels B_p. Under any affine transformation T, the area of a polygon is multiplied by $|\det(T)|$. The coefficients $Area(B_p \cap F_j)$ of the unknown variables $Color(F_j)$ are multiplied by the same constant factor $|\det(T)|$.

The condition that the complement $I - T(I)$ has only zero colors guarantees that any polygon outside $T(I)$ in a mesh minimizing the Reconstruction Error should also have zero colors. Since $Intensity(p)$ is interpreted as the original color for a pixel p and its transformation $T(B_p)$, the sum in (9a) is (globally) minimized by $Color(F_j)/|\det(T)|$ for transformed mesh $T(M(I))$. □

Since the aim of superpixels is to approximate a large image by a reconstructed image based on a smaller superpixel mesh, the standard statistical error $nRMS$ is the important quality measure. Definition 9 makes sense for RGB images if we replace the pixel intensity by a vector of colors. We consider only grayscale images only because the LSD algorithm works for grayscale images.

Fig. 11 shows that the colored CCM meshes better approximate original images than the colored Voronoi meshes. Some convex polygons of CCMs are much larger than Voronoi superpixels, simply because the corresponding regions in images indeed have almost the same intensity, e.g. the sky. Hence taking the best constant color over each superpixel is reasonable.

Despite CCMs being obtained from only LSD edges without using colors, the reconstructions have smaller errors in comparison with Voronoi meshes that take colors into account, see Fig. 12.

Fig 12 **Top**: 791 Voronoi superpixels (boundary mesh and reconstructed image) with $nRMS \approx 8.45\%$. **Bottom**: 791 CCM superpixels (boundary mesh and reconstructed image) with $nRMS \approx 7.22\%$.

Fig. 11 confirms smaller approximation errors of CCM superpixels across all BSD500 images, where we used the same LSD parameters for CCM and Voronoi superpixels. We computed all optimal colors minimizing $nRMS$, which is measured in percents percentage as in Definition 9.

Each BSD experiment outputs 500 pairs (number of superpixels, $nRMS$). We average each coordinate of these pairs and output a single dot per experiment. The first red dot at $(377.1, 9.626\%)$

in Fig. 11 means that CCMs have 377 superpixels and the average $nRMS \approx 9.6\%$. For a fixed image, the LSD algorithm outputs roughly the same number of edges for all its parameters.

So smaller CCMs seem impossible, because all LSD edges are hard constraints, while all superpixels should be convex. To get larger CCMs, we stop merging faces in Section 5 after getting a certain number of convex faces. The five experiments on Voronoi superpixels with $\mathrm{Eps\_Radius} = 7, 8, 9, 10, 11$ produced 5 dots along a decreasing curve. Fig. 11 implies that Voronoi meshes require more superpixels (507.3 on average) to achieve the similar $nRMS \approx 9.696\%$.

6.3 Standard BSD Benchmarks for Pixel-based and Polygonal Superpixels

The *Berkeley Segmentation Database* BSD500[6] contains 500 images widely used for evaluating segmentation algorithms due to human-sketched ground truth boundaries. For an image I, let $I = \cup G_j$ be a segmentation into ground truth regions and $I = \cup_{i=1}^{k} S_i$ be an oversegmentation into superpixels produced by an algorithm. Each quality measure below compares a set of superpixels $S_1, \ldots, S_k$ with the best suitable ground truth for every image from BSD500.

Let $G(I) = \cup G_j$ be the union of ground truth boundary pixels and $B(I)$ be the set of boundary pixels produced by a superpixel algorithm. The Boundary Recall BR is the ratio of ground truth boundary pixels $p \in G(I)$ that are within 2 pixels from the superpixel boundary $B(I)$.

The Undersegmentation Error $UE = \dfrac{1}{n} \sum_j \sum_{S_i \cap G_j \neq \emptyset} |S_i - G_j|$ was often used in the past, where $|S_i - G_j|$ is the number of pixels that are in S_i, but not in G_j. However a superpixel is fully penalized when $S_i \cap G_j$ is 1 pixel. Corrections of UE required ad hoc tweaks, e.g. the 5% threshold $|S_i - G_j| \geq 0.05|S_i|$ by Achanta et al.[9] or ignoring boundary pixels of S_i by Liu et al.[8]

Let $G_{max}(S_i)$ be the ground truth region having the largest overlap with S_i. Van den Bergh et

al.[13] suggested the Corrected Undersegmentation Error $CUE = \frac{1}{n}\sum_i |S_i - G_{max}(S_i)|$.

The Undersegmentation Symmetric Error[20] is $USE = \frac{1}{n}\sum_j \sum_{S_i \cap G_j \neq \emptyset} \min\{in(S_i), out(S_i)\}$, where $in(S_i)$ is the area of S_i inside G_j, while $out(S_i)$ is the area of S_i outside G_j.

The BSD benchmarks BR, CUE and USE are designed for pixel-based superpixels, not for polygonal superpixels. That is why we had to discretize CCM and Voronoi superpixels by drawing polygonal meshes in OpenCV to get only boundary pixels. We put all square pixels into the same discretized superpixel if their centers belong to the original polygonal superpixel.

CCM superpixels achieve smaller undersegmentation errors than SEEDS and SLIC and most importantly beat Voronoi superpixels on the objective measure $nRMS$ as well as on BR.

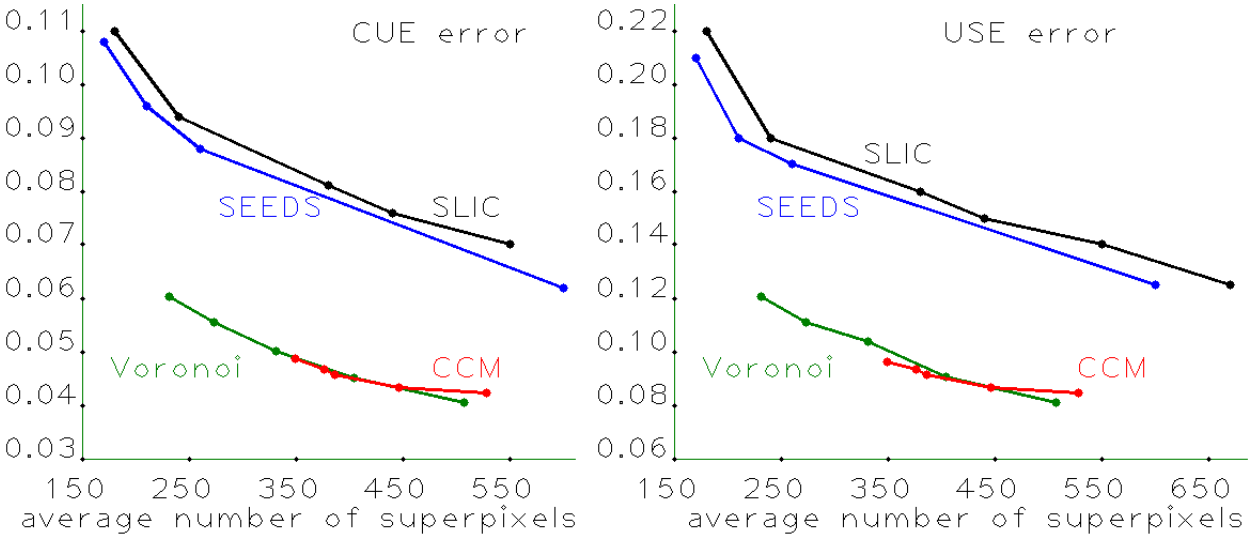

Fig 13 Left: Corrected Undersegmentation Error (CUE). **Right**: Undersegmentation Symmetric Error (USE).

Pixel-based superpixels SLIC and SEEDS achieve better results on the Boundary Recall in Fig. 11, because their superpixels can have zigzag boundaries of horizontal and vertical edges. However, humans are likely to sketch straight edges than boundaries consisting of short zigzags.

Traditional pixel-based superpixels are often split by straight ground truth boundaries. Polygonal superpixels at subpixel resolution can have straight edges of any direction and better approx-

imate human drawings as confirmed by the undersegmentation errors in Fig. 13.

The simplest compactness measure for superpixels from[21] is based on the isoperimetric quotient $Q(S) = \dfrac{4\pi \text{ area}(S)}{\text{perimeter}^2(S)}$, which has the maximum value 1 for a round disk S. The *Compactness* is the average $Comp = \sum_{\text{superpixels S}} \dfrac{Q(S)}{\#\text{superpixels}}$.

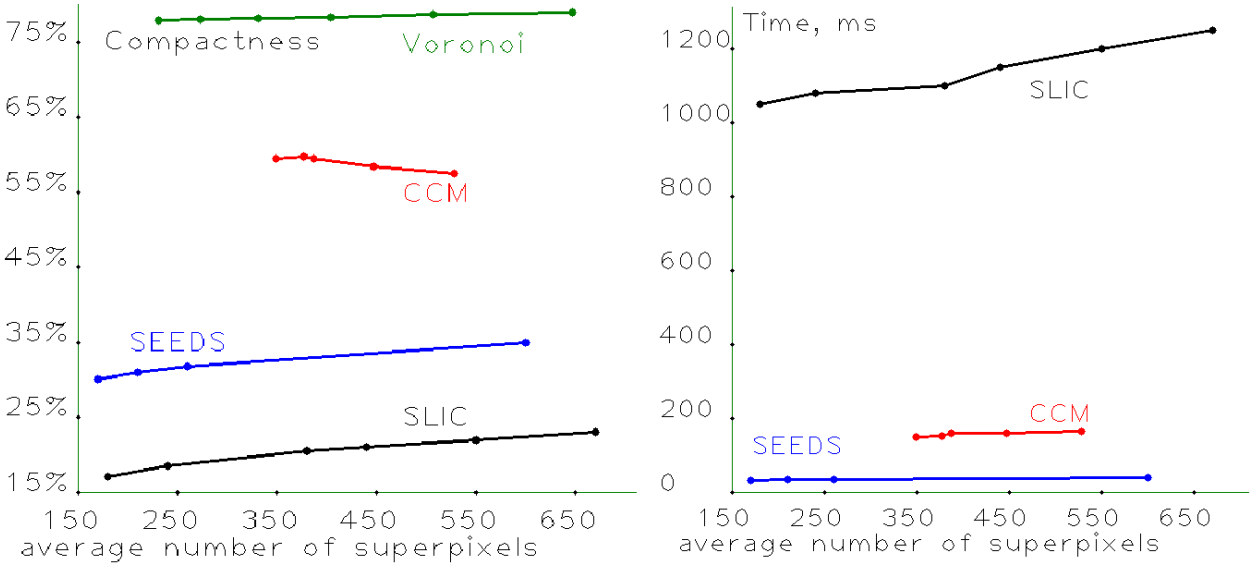

Fig 14 Left: Compactness. **Right**: Times for BSD images on a laptop with 8G RAM 2.6 GHz Intel Core i5.

Since only a Windows demo is available for Voronoi superpixels[5] and we worked on a different platform, we couldn't directly compare the times of polygonal superpixels. The times are similar for $\text{Min\_Angle} \in \{10, 12, 14, 16, 18, 20\}$ and Min_Distance around the default value of 3 pixels.

6.4 Conclusions and Open Problems for Superpixels at Subpixel Resolution

The key contribution is the new concept of a Convex Constrained Mesh (CCM), which extends any given straight line segments (possibly with intersections) to a full mesh of convex polygons without small angles. This paper proves theoretical guarantees of the LSD refinement and CCM approximations in Theorems 5 and 8, which were announced in the shorter conference version.[1]

• Theorem 8 guarantees the approximation quality and no small angles in CCMs, which also have smaller sizes in comparison with Shewchuk triangulations 17 and Voronoi superpixels 5.

• The Reconstruction Error is introduced in Definition 9 as the objective quality measure for polygonal superpixels and is justified by Theorem 10 on affine invariance for optimal superpixels.

• The CCM outperforms the only past algorithm[5] for polygonal superpixels on BR and $nRMS$ in Fig. 11, and even outperforms pixel-based superpixels on the CUE and USE in Fig. 13.

The C++ code can be available by request. The next step in improving superpixel meshes can be a replacement LSD by a stable under noise straight line skeleton obtained form a noisy cloud of edgels, e.g. by extending a Homologically Persistent Skeleton for a noisy cloud of edgels.[22–24] Usually this skeletonization problem is solved by using manual thresholds.[25] However the method of persistent homology can offer scale-free solutions that are provably stable under noise.[26]

Acknowledgments. The first author was supported by the Austrian Science Fund (FWF) project P24600-N23 at TU Wien. We thank all the reviewers for their helpful suggestions.

References

1 J. Forsythe, V. Kurlin, and A. Fitzgibbon, "Resolution-independent superpixels based on convex constrained meshes without small angles," in *Proc. of ISVC*, 223–233 (2016).

2 F. Viola, A. Fitzgibbon, and R. Cipolla, "A unifying resolution-independent formulation for early vision," in *Proceedings of Computer Vision and Pattern Recognition*, 494–501 (2012).

3 J. Philbin, O. Chum, M. Isard, *et al.*, "Object retrieval with large vocabularies and fast spatial matching," in *Proceedings of Computer Vision and Pattern Recognition*, (2007).

4 R. Grompone von Gioi, J. Jakubowicz, J.-M. Morel, *et al.*, "Lsd: a line segment detector," *Image Processing On Line* **2**, 35–55 (2012).

5 L. Duan and F. Lafarge, "Image partitioning into convex polygons," in *Proceedings of CVPR (Computer Vision and Pattern Recognition)*, 3119–3127 (2015).

6 P. Arbelaez, M. Maire, C. Fowlkes, *et al.*, "Contour detection and hierarchical image segmentation," *Transactions PAMI* **33**, 898–916 (2011).

7 J. Shi and J. Malik, "Normalized cuts and image segmentation," *T-PAMI* **22**, 888–905 (2000).

8 M.-Y. Liu, O. Tuzel, S. Ramalingam, *et al.*, "Entropy rate superpixel segmentation," in *Proceedings of CVPR*, 2097 – 2104 (2011).

9 R. Achanta, A. Shaji, K. Smith, *et al.*, "Slic superpixels compared to state-of-the-art superpixel methods," *Transactions PAMI* **34**, 2274–2282 (2012).

10 Z. Li and J. Chen, "Superpixel segmentation using linear spectral clustering," in *Proceedings of CVPR (Computer Vision and Pattern Recognition)*, 1356–1363 (2015).

11 R. Giraud, V.-T. Ta, and N. Papadakis, "Scalp: Superpixels with contour adherence using linear path," in *International Conference on Pattern Recognition (ICPR)*, 2374–2379 (2016).

12 P. Buyssens, M. Toutain, A. Elmoataz, *et al.*, "Eikonal-based vertices growing and iterative seeding for efficient graph-based segmentation," in *Proc. of ICIP*, 4368–4372 (2014).

13 M. Van de Bergh, X. Boix, G. Roig, *et al.*, "Seeds: superpixels extracted via energy-driven sampling," *Int J Computer Vision* **111**, 298–314 (2015).

14 J. Yao, M. Boben, S. Fidler, *et al.*, "Real-time coarse-to-fine topologically preserving segmentation," in *Proceedings of Computer Vision and Pattern Recognition*, 2947–2955 (2015).

15 L. A. Royer, D. L. Richmond, C. Rother, *et al.*, "Convexity shape constraints for image segmentation," in *Computer Vision and Pattern Recognition (CVPR)*, 402–410, IEEE (2016).

16 V. Machairas, M. Faessel, D. Cárdenas-Peña, *et al.*, "Waterpixels," *IEEE Transactions on Image Processing* **24**(11), 3707–3716 (2015).

17 J. R. Shewchuk, "Delaunay refinement algorithms for triangular mesh generation," *Computational Geometry: Theory and Applications* **22**, 21–74 (2002).

18 S.-W. Cheng, T. K. Dey, and J. R. Shewchuk, *Delaunay Mesh Generation*, CRC Press (2012).

19 M. de Berg, O. Cheong, M. van Kreveld, *et al.*, *Computational Geometry : Algorithms and Applications*, Springer (2010).

20 P. Neubert and P. Protzel, "Superpixel benchmark and comparison," in *Proc. Forum Bildverarbeitung*, 1–12 (2012).

21 P. Neubert, "Superpixels and their application for visual place recognition in changing environments (phd thesis at tu chemnitz)," (2015).

22 V. Kurlin, "Auto-completion of contours in sketches, maps and sparse 2d images based on topological persistence," in *Proceedings of CTIC*, 594–601 (2014).

23 V. Kurlin, "A one-dimensional homologically persistent skeleton of a point cloud in any metric space," *Computer Graphics Forum* **34**, 253–262 (2015).

24 V. Kurlin, "A fast persistence-based segmentation of noisy 2d clouds with provable guarantees," *Pattern Recognition Letters* **83**, 3–12 (2016).

25 A. Chernov and V. Kurlin, "Reconstructing persistent graph structures from noisy images," *Image-A* **3**, 19–22 (2013).

26 V. Kurlin, "A fast and robust algorithm to count topologically persistent holes in noisy clouds," in *Proceedings of CVPR*, 1458–1463 (2014).