TurboPixels: Fast Superpixels Using Geometric Flows

Alex Levinshtein, Adrian Stere, Kiriakos N. Kutulakos, David J. Fleet, Sven J. Dickinson
University of Toronto
Toronto, Canada
babalex,adrianst,kyros,fleet,sven@cs.toronto.edu

Kaleem Siddiqi
McGill University
Montreal, Canada
siddiqi@cim.mcgill.ca

***Abstract*— We describe a geometric-flow based algorithm for computing a dense over-segmentation of an image, often referred to as superpixels. It produces segments that on one hand respect local image boundaries, while on the other hand limit under-segmentation through a compactness constraint. It is very fast, with complexity that is approximately linear in image size, and can be applied to megapixel sized images with high superpixel densities in a matter of minutes. We show qualitative demonstrations of high quality results on several complex images. The Berkeley database is used to quantitatively compare its performance to a number of over-segmentation algorithms, showing that it yields less under-segmentation than algorithms that lack a compactness constraint, while offering a significant speed-up over N-cuts, which does enforce compactness.**

***Index Terms*— superpixels, image segmentation, image labeling, perceptual grouping.**

I. Introduction

Superpixels [18] represent a restricted form of region segmentation, balancing the conflicting goals of reducing image complexity through pixel grouping while avoiding under-segmentation. They have been adopted primarily by those attempting to segment, classify or label images from labelled training data [8], [9], [10], [16], [18]. The computational cost of the underlying grouping processes, whether probabilistic or combinatorial, is greatly reduced by contracting the pixel graph to a superpixel graph. For many such problems, it is far easier to merge superpixels than to split them, implying that superpixels should aim to *over-segment* the image. Region segmentation algorithms which lack some form of compactness constraint, e.g., local variation [6], mean-shift [3], or watershed [7], can lead to *under-segmentation* in the absence of boundary cues in the image. This can occur, for example, when there is poor contrast or shadows. Algorithms that do encode a compactness constraint, including N-Cuts[27] and *TurboPixels* (the framework we propose), offer an important mechanism for coping with under-segmentation. Figure 1 shows the over-segmentations obtained using these five algorithms; the effect of a compactness constraint in limiting under-segmentation can be clearly observed in the results produced by *TurboPixels* and *N-Cuts*.

The superpixel algorithm of Ren and Malik [18] is a restricted graph cut algorithm, constrained to yield a large number of small, compact, quasi-uniform regions. Graph cut segmentation algorithms operate on graphs whose nodes are pixel values and whose edges represent affinities between pixel pairs. They seek a set of recursive bi-partitions that globally minimize a cost function based on the nodes in a segment and/or the edges between segments. Wu and Leahy [26] were the first to segment images using graph cuts, minimizing the sum of the edge weights across cut boundaries. However, their algorithm is biased toward short boundaries, leading to the creation of small regions. To mitigate this bias, the graph cut cost can be normalized using the edge weights being cut and/or properties of the resulting regions. Although many cost functions have been proposed (e.g., [5], [11], [19], [25]), the most popular normalized cut formulation, referred to widely as N-Cuts, is due to Shi and Malik [21], and was the basis for the original superpixel algorithm of [18].

The cost of finding globally optimal solutions is high. Since the normalized cut problem is NP-hard for non-planar graphs, Shi and Malik proposed a spectral approximation method with (approximate) complexity $\mathcal{O}(N^{3/2})$, where N is the number of pixels. Space and run-time complexity also depend on the number of segments, and become prohibitive with large numbers of segments. In [20] a further reduction in complexity by a factor of $\sqrt{N}$ is achieved, based on a recursive coarsening of the segmentation problem. However, the number of superpixels is no longer directly controlled, nor is the algorithm designed to ensure the quasi-uniformity of segment size and shape. Cour et al. [4] also proposed a linear time algorithm by solving a constrained multi-scale N-Cuts problem, but this complexity does not take the number of superpixels into account. In practice, this method remains computationally expensive and thus unsuitable for large images with many superpixels.

There are fast segmentation algorithms with indirect control over the number of segments. Three examples include the *local variation* graph-based algorithm of Felzenszwalb and Huttenlocher [6], the *mean-shift* algorithm of Comaniciu and Meer [3], and Vincent and Soille's watershed segmentation [7]. However, as mentioned earlier, since they lack a compactness constraint, such algorithms typically produce regions of irregular shapes and sizes.

The *TurboPixel* algorithm introduced in this paper segments an image into a lattice-like structure of compact regions (superpixels)

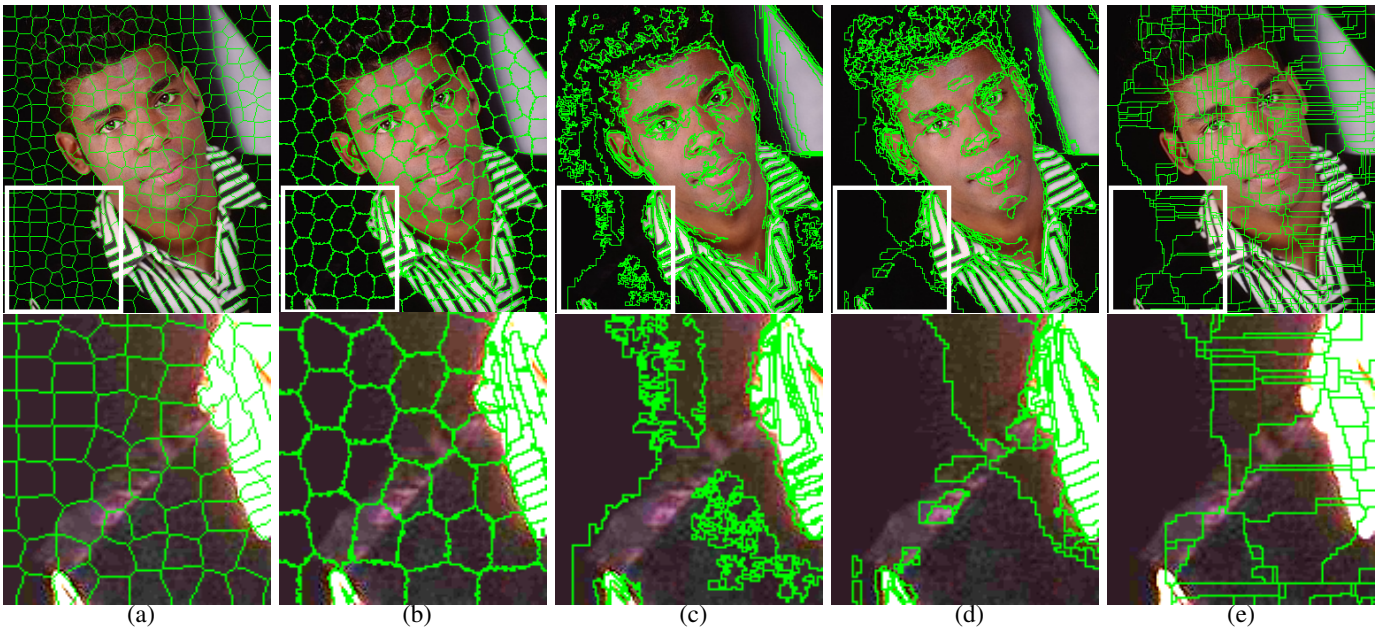


Fig. 1. Over-segmentations obtained with five algorithms: (a) TurboPixels (b) N-Cuts[27] (c) Local variation [6] (d) Mean-shift [3] (e) Watershed [7]. Each segmentation has (approximately) the same number of segments. The second row zooms in on the regions of interest defined by the white boxes.

by dilating seeds so as to adapt to local image structure. Computationally, the approach is rooted in the early curve evolution techniques in computer vision (e.g., [1], [13], [14]). In an approach that is similar in philosophy to the one we develop in this paper, in [28] properties of the medial axis are used to modify the evolution of two simultaneously evolving contours, in application to carpal bone segmentation. In the reaction-diffusion space of [13], a constant motion (reaction) term was played off against a curvature term (diffusion) for shape analysis. This flow was subsequently adapted to the problem of segmentation in [14] and [1], via the inclusion of a multiplicative image gradient stopping term. These methods led to active contour models that could handle changes in topology in a natural way. Formal theoretical justification, as the gradient flows associated with particular weighted length or area functionals, followed [2], [12], [23]. A reaction-diffusion space of *bubbles* was further developed in [24], where instead of a single contour, multiple bubbles were simultaneously placed and grown from homogeneous regions of the image.

While there are many variations on the theme of dilating seeds using geometric flows (e.g., this idea has been used for segmenting vasculature in medical imaging [29]), none of these methods have been applied thus far to superpixel segmentation. Below we develop such a technique by combining a curve evolution model for dilation with a skeletonization process on the background region to prevent the expanding seeds from merging. We demonstrate that this technique advances the state of the art in compact superpixel computation by 1) being applicable to megapixel size images, with very high superpixel densities, and 2) providing comparable accuracy to N-Cuts, but with significantly lower run times.

II. Superpixels from Geometric Flows

The key idea in our approach is to reduce superpixel computation to an efficiently-solvable geometric flow problem. Our approach is guided by five basic principles:

- **Uniform size and coverage:** Superpixel segmentation should partition an image into regions that are approximately uniform in size and shape (compactness), minimizing region under-segmentation, provided that superpixel size is comparable to the size of the smallest target region. We achieve this by designing a geometric flow that dilates an initial set of uniformly-distributed seeds, where each seed corresponds to one superpixel. The seeds behave initially like reaction-diffusion bubbles [24].
- **Connectivity:** Each superpixel should represent a simply-connected set of pixels. Our dilation-based flow combined with its level-set implementation, ensures that this constraint is always satisfied.
- **Compactness:** In the absence of local edge information, superpixels should remain compact. Our flow begins from circular seeds and assumes no prior bias on the location of superpixel boundaries. To maximize compactness, we include a term that produces constant motion in the direction of the outward normal in regions of uniform intensity. This term maximizes the rate of area growth, while retaining the minimum possible isoperimetric ratio, which is 4π for a circular region.
- **Smooth, edge-preserving flow:** When growth stops, superpixel boundaries should coincide with image edges. This requires a geometric flow formulation with three properties: (1) it should slow down boundary growth in the vicinity of edges; (2) it should be attracted to edges; and (3) it should produce smooth boundaries. To do this, we borrow ideas from work on geometric active contours [1], [2], [12], [13], [14]. Such formulations provide an easy way to incorporate image-based controls on boundary growth, and include both a "doublet" term for attraction and a curvature term for shape regularization.
- **No superpixel overlap:** A superpixel segmentation should assign every pixel to a single superpixel. Therefore, boundary evolution should stop when two distinct dilating seeds are about to collide. To achieve this we incorporate a simple skeleton-based mechanism for collision detection in the background.

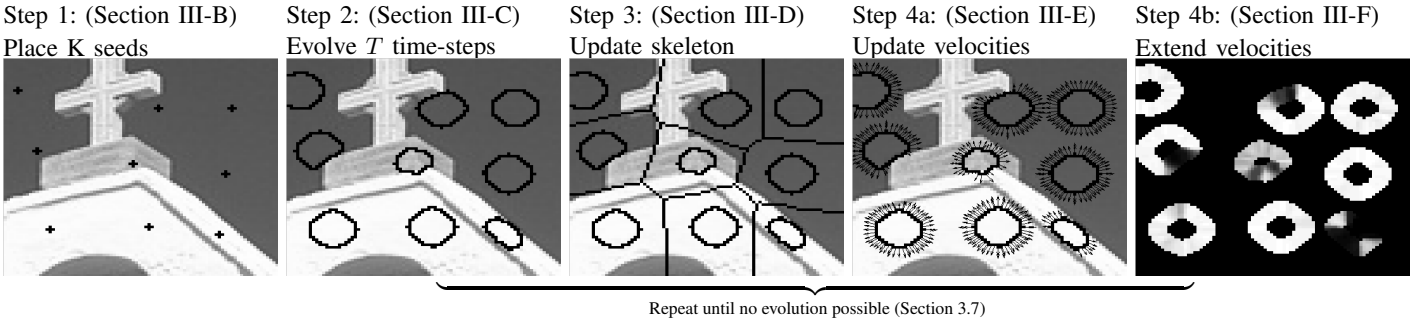


Fig. 2. Steps of the TurboPixel algorithm. In Step 4a the vectors depict the current velocities at seed boundaries. Where edges have been reached the velocities are small. In Step 4b the magnitude of velocities within the narrow band is proportional to brightness.

These considerations lead to a geometric flow-based algorithm, that we call *TurboPixels*, whose goal is to maintain and evolve the boundary between the *assigned region*, which contains all pixels that are already inside some superpixel, and the *unassigned region*, which contains all other pixels. At a conceptual level, the algorithm consists of the following steps, as illustrated in Fig. 2:

1) place initial seeds
2) iterate over the following basic steps until no further evolution is possible, i.e., when the speed at all boundary pixels is close to zero.
   a) evolve this boundary for T time steps;
   b) estimate the skeleton of the unassigned region;
   c) update the speed of each pixel on the boundary and of unassigned pixels in the boundary's immediate vicinity.

See Algorithm 1 for a pseudocode summary of these steps, each of which is discussed in detail below.

III. The TurboPixel Algorithm

A. Level-Set Boundary Representation

Geometric flows of the type associated with the TurboPixel algorithm are commonly implemented using level-set methods [17]. The basic idea is to devise a flow by which curves evolve to obtain superpixel boundaries. Let $\mathbf{C}$ be a vector of curve coordinates parameterized by p, a parameter which runs along the curve, and t, a parameter to denote evolution in time. Let $\mathbf{N}$ represent its outward normal and let each point move with speed S by a curve evolution equation $\frac{\partial \mathbf{C}}{\partial t} = S\mathbf{N}$. This curve evolution equation is implemented by first embedding $\mathbf{C}$ as a level set of a smooth and continuous function $\Psi : \mathbf{R}^2 \times [0, \tau) \rightarrow \mathbf{R}^2$ and then evolving this embedding function according to:

$$\Psi_t = -S \, ||\nabla \Psi|| \ . \tag{1}$$

In practice we define Ψ over the image plane as the signed Euclidean distance of each image pixel to the closest point on the boundary between assigned and unassigned (background) regions. A pixel's distance is positive if it is in the unassigned region and negative if it is not, with the boundary represented implicitly as the zero level set of Ψ. Since we are only interested in the zero level set of Ψ, we maintain an accurate representation of Ψ only in a narrow band around its current zero level set (typically 4 pixels wide on each side of the boundary). This narrow band is computed using the Fast Marching implementation in LMSLIB[1]. The superpixel boundaries can be computed at subpixel resolution by interpolation.

B. Initial Seed Placement

One of our key objectives is to compute superpixels that are evenly distributed over the image plane. Given a user-specified value K of superpixels, we place K circular seeds in a lattice formation so that distances between lattice neighbors are all approximately equal to $\sqrt{\frac{N}{K}}$, where N is the total number of pixels in the image. This distance completely determines the seed lattice, since it can be readily converted into a distance across lattice rows and columns. In our implementation the initial seed radius is 1 pixel.

The above strategy ensures that superpixels in a uniform-intensity image will satisfy the uniform distribution objective exactly. In practice, of course, images are not uniform and this deterministic placement may cause some seeds to accidentally fall on or close to a strong edge, inhibiting their early growth. To avoid this we perturb the position of each seed by moving it in the direction of the image gradient as a function of the gradient magnitude (see Sec. III-E), with the maximum perturbation determined by the seed density.

C. Numerical Level Set Evolution

We use the following first-order discretization in time of Eq. (1):

$$\Psi^{n+1} = \Psi^n - \ S_I S_B \ \| \nabla \Psi^n \| \ \Delta t \ \ . \tag{2}$$

Each application of Eq. (2) corresponds to one "time step" Δt in the evolution of the boundary. We apply this equation until any point on the evolving boundary reaches the edge of the narrow band. The key term controlling the evolution is the product of two speeds $S_I S_B$, which lie at the heart of our TurboPixel algorithm. The first term (S_I) depends on local image structure and superpixel geometry at each boundary point, and the second (S_B) depends on the boundary point's proximity to other superpixels. We detail the computation of these velocities in Sections III-E and III-D, respectively.

In theory, the velocities in Eq. (2) are defined at every point on the zero level set. In practice, we compute this term for a small band of pixels in the vicinity of the zero level set at iteration n. We discuss this process in Sec. III-F. For notational simplicity, we omit the parameter n from Ψ^n in the following sections, except where it is explicitly needed.

[1] $LSMLIB$ is a library of level set routines written by K. Chu (`http://www.princeton.edu/~ktchu/software/lsmlib/`).

Algorithm 1: TurboPixel Algorithm

Input: Image I, number of seeds K
Output: Superpixel boundaries B

Place K seeds on a rectangular grid in image I;
Perturb the seed positions away from high gradient regions;
Set all seed pixels to "assigned";
Set Ψ^0 to be the signed Euclidean distance from the "assigned" regions;
assigned_pixels $\leftarrow \sum_{x,y} [\Psi^0(x,y) >= 0]$;
Compute the pixel affinity $\phi(x,y)$;
$n \leftarrow 0$;
while *Change in assigned_pixels is large* **do**
    Compute the image velocity S_I;
    Compute the boundary velocity S_B;
    $S \leftarrow S_I S_B$;
    Extend the speed S in a narrow band near the zero level-set of Ψ^n;
    Compute Ψ^{n+1} by evolving Ψ^n within the narrow band;
    $n \leftarrow n + 1$;
    assigned_pixels $\leftarrow \sum_{x,y} [\Psi^n(x,y) >= 0]$;
$B \leftarrow$ homotopic skeleton of Ψ^n;
return B

D. Proximity-Based Boundary Velocity

The proximity-based velocity term ensures that the boundaries of nearby superpixels never cross each other. To do this, we use a binary stopping term that is equal to 0 on the 2D homotopic skeleton of the unassigned region and is equal to 1 everywhere else, i.e., $S_B(x,y) = 0$ if and only if (x,y) is on the skeleton. This formulation allows the boundary of each superpixel to be guided entirely by the underlying image, until it gets very close to another superpixel boundary.

Since the regions between evolving curves change at each iteration of our algorithm, the skeleton must be updated as well. We do this efficiently by marking all pixels in these unassigned regions (i.e., those with $\Psi(x,y) > 0$) and then applying a homotopy preserving thinning algorithm [22] on them to compute the skeleton. The thinning algorithm removes pixels ordered by their distance to the boundary of the region with the constraint that all digital points that can be removed without altering the topology are removed.

E. Image-Based Boundary Velocity

Our image-based speed term combines the reaction-diffusion based shape segmentation model of [1], [14], [24] with an additional "doublet" term provided by the geodesic active contour [2], [12] to attract the flow to edges:

$$S_I(x,y) = \underbrace{[\,1 - \alpha\,\kappa(x,y)\,]\,\phi(x,y)}_{\text{reaction-diffusion term}} - \underbrace{\beta\,[\,\mathbf{N}(x,y)\cdot\nabla\phi(x,y)\,]}_{\text{"doublet" term}} . \tag{3}$$

The reaction-diffusion term ensures that the boundary's evolution slows down when it gets close to a high-gradient region in the image. It is controlled by three quantities: (1) a "local affinity" function $\phi(x,y)$, computed for every pixel on the image plane, that is low near edges and high elsewhere; (2) a curvature function $\kappa(x,y)$ that expresses the curvature of the boundary at point (x,y) and smoothes the evolving boundary; and (3) a "balancing" parameter α that weighs the contribution of the curvature term.

Intuitively, the doublet term ensures that the boundary is attracted to image edges, i.e., pixels where the affinity is low. Specifically, when a point (x,y) on the boundary evolves toward a region of decreasing affinity (an image edge), its normal $\mathbf{N}(x,y)$ will coincide with the negative gradient direction of ϕ, and the term acts as an attractive force. If the boundary crosses over an edge these two vectors will point in the same direction and cause a reversal in the boundary's direction of motion.

Local affinity function Our algorithm does not depend on a specific definition of the function ϕ, as long as it is low on edges and is high elsewhere. For almost all the experiments in this paper, we used a simple affinity measure based on the grayscale intensity gradient:

$$\phi(x,y) = e^{-E(x,y)/\nu},\ E(x,y) = \frac{\|\nabla I\|}{G_\sigma * \|\nabla I\| + \gamma} . \tag{4}$$

Our affinity function ϕ produces high velocities in areas with low gradients, with an upper bound of 1. Dividing the gradient magnitude in $E(x,y)$ by a local weighted sum of gradient magnitudes provides a simple form of contrast normalization. The support width of the normalization, controlled by σ, is proportional to the expected initial distance between seeds. This normalization allows weak but isolated edges to have a significant effect on speed, while suppressing edge strength in dense texture. The constant γ ensures that the effect of insignificant signal gradients remains small. We note that whereas our $E(x,y)$ is a simple measure of grayscale image gradient, the implementation of N-Cuts we use for comparison ([27]) in our experiments in Section IV employs a more complex measure of intervening contours computed using a texture-based edge map.

Normal and curvature functions The outward normal of the zero level set of Ψ at a point (x,y) is given by the derivatives of Ψ, i.e., $\mathbf{N} = \nabla\Psi / \|\nabla\Psi\|$. The curvature of the zero level set, at a point (x,y), is given by [17]:

$$\kappa = \frac{\Psi_{xx}\Psi_y^2 - 2\Psi_x\Psi_y\Psi_{xy} + \Psi_{yy}\Psi_x^2}{(\Psi_x^2 + \Psi_y^2)^{\frac{3}{2}}} . \tag{5}$$

As is standard for diffusive terms, the derivatives of Ψ used for κ are computed using central difference approximations. Central difference approximations are also used for all other calculations with the exception of $\|\nabla\Psi^n\|$ in the level set form for the reaction term ($\phi(x,y)$) in Eq. 3, for which upwind derivatives [17] must be used since it is a hyperbolic term.

Balancing parameters The balancing parameters α and β in Eq. 3 control the relative contributions of the reaction-diffusion and doublet terms. Higher values of α prevent "leakage" through narrow edge gaps, but also prevent sharp superpixel boundaries that may be sometimes desirable. High values of β cause better stopping behavior of seeds on weak edges, but also slow down the evolution of seeds elsewhere.[2]

F. Speed Extension

The velocity terms S_I and S_B have meaning only on the current superpixel boundaries, i.e., the zero level set of Ψ. This leads

[2]Based on empirical observation, the values $\alpha = 0.3$ and $\beta = 1$ were chosen. These values limit the amount of leakage during seed evolution, without slowing down the evolution in other regions. In the future, we intend to learn the optimal values for these parameters automatically by evaluating the performance of the algorithm on a training set of images.

to two technical difficulties. First, the zero level set is defined implicitly and, hence, it lies "in between" the discrete image pixels. Second, each time we invoke a level set update iteration (Eq. 2), the boundary must move by a finite amount (i.e., at least a sizeable fraction of a pixel).

Speed extension gives a way to solve both problems and is common in existing curve evolution implementations [14]. Here, we extend ϕ and $\nabla\phi$, the only image-dependent terms, in the same narrow band we use to maintain an accurate estimate of Ψ (see Sec. III-C). To each pixel (x, y) in this narrow band, we simply assign the ϕ and $\nabla\phi$ values of its closest pixel on the boundary[3].

G. Termination Conditions & Final Segmentation

The algorithm terminates when the boundaries stop evolving. Since in theory the boundaries can evolve indefinitely with ever-decreasing velocities, the algorithm terminates when the relative increase of the total area covered by superpixels falls below a threshold. We used a relative area threshold of 10^{-4} in all our experiments.

After termination, the evolution results are post-processed so that the superpixel boundaries are exactly one pixel in width. This is done in three steps. First, any remaining large unassigned connected regions are treated as superpixels. Next, very small superpixels are removed, making their corresponding pixels unassigned. Finally, these unassigned regions are thinned, as in Sec. III-D, according to the algorithm in [22]. The thinning is ordered by a combination of Euclidean distance to the boundary and a ϕ-based term, in order to obtain smooth superpixel contours that are close to edges.

H. Algorithm Complexity

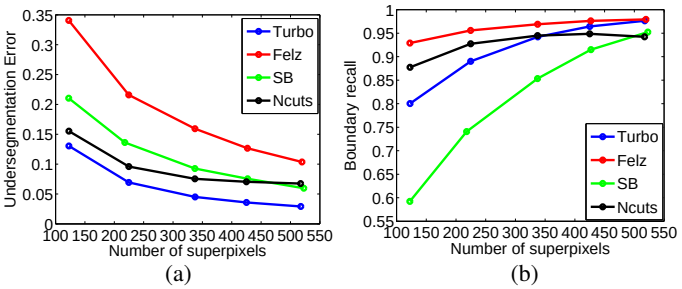
The complexity of our algorithm is roughly linear in the total number of image pixels N for a fixed superpixel density. At each time step, all elements of the distance function Ψ are updated (see Eq. 2). Each update requires the computation of the partial derivatives of Ψ and evaluation of $S_I S_B$. Thus each update takes $\mathcal{O}(N)$ operations.

The speed extension and homotopic skeleton computations are not linear in image size. Both actions are $\mathcal{O}(N \log N)$ but can be made faster in practice. If b is the number of pixels in the narrow band (which is linear in the number of pixels that lie on zero-crossings), then the complexity of speed extension is $\mathcal{O}(N) + \mathcal{O}(b \log b)$. While b can approach N in theory, it is usually much smaller in practice.

The homotopic skeleton computation is $\mathcal{O}(N)+\mathcal{O}(k \log k)$ [22], where k is the number of unassigned pixels. In practice, $k \ll N$, especially toward the end of the evolution when few unassigned pixels remain.

It now remains to take into account the number of iterations of the algorithm. Under ideal conditions, all the curves evolve with maximal speed until they meet or reach an edge. Since the expected distance between seeds is D_n initially (see Sec. III-B), it will take $\mathcal{O}(\sqrt{\frac{N}{K}})$ iterations for the algorithm to converge. Hence, the algorithm converges more slowly for larger images, and more quickly as the superpixel density increases. Thus for a fixed superpixel density (keeping D_n constant), the number of iterations will be constant, making the overall complexity roughly $\mathcal{O}(N)$.

[3]The algorithm that efficiently computes Ψ for all pixels within the narrow band provides their closest boundary pixel as a byproduct, so no extra computations are necessary.



(a) (b)

Fig. 3. Under-segmentation error (a) and accuracy (boundary recall) (b) as a function of superpixel density.

IV. Experimental Results

We evaluate the performance of the TurboPixel algorithm by comparing its accuracy and running time to three other algorithms: Normalized Cuts (Ncuts) and square blocks (Sb), both of which encode a compactness constraint, and Felzenszwalb and Huttenlocher (Felz), which does not. The TurboPixel algorithm was implemented in Matlab with several C extensions[4]. For Ncuts, we use the 2004 Ncut implementation based on [27][5], while for Sb, we simply divide the image into even rectangular blocks, providing a naive but efficient benchmark for accuracy (other algorithms are expected to do better). All experiments were performed on a quad-core Xeon 3.6 Ghz computer. We use the Berkeley database, which contains 300 (481×321 or 321×481) images. In our experiments, the image size is defined as the fraction of the area of the full image size of 154401 pixels. In all experiments, performance/accuracy is averaged over at least 25 images and in most cases over a larger number.[6] Finally, the gradient-based affinity function of a grayscale image (Eq. 4) was used for the TurboPixel algorithm, a difference in image intensity was used as affinity in Felz, and a more elaborate (intervening contours) affinity was used for Ncuts.

A. Under-segmentation Error

As stated in Section I, algorithms that do not enforce a compactness constraint risk a greater degree of under-segmentation. Given a ground-truth segmentation into segments $g_1, \ldots, g_K$ and a superpixel segmentation into superpixels $s_1, \ldots, s_L$, we quantify the under-segmentation error for segment g_i with the fraction

$$\frac{\left[\sum_{\{s_j \mid s_j \cap g_i \neq \emptyset\}} Area(s_j)\right] - Area(g_i)}{Area(g_i)}. \quad (6)$$

Intuitively, this fraction measures the total amount of "bleeding" caused by superpixels that overlap a given ground-truth segment, normalized by the segment's area.

To evaluate the under-segmentation performance of a given algorithm, we simply average the above fraction across all ground-truth segments and all images. Figure 3(a) compares

[4]A beta-version of our code is available at `http://www.cs.toronto.edu/~babalex/turbopixels_supplementary.tar.gz`; the default parameter values are the same as those used for the experiments in this article.

[5]We use Version 7 from Jianbo Shi's website `http://www.cis.upenn.edu/~jshi/software/files/NcutImage_7_1.zip`

[6]Due to the long running time and large memory requirements of Ncuts, using the entire database was prohibitively expensive.

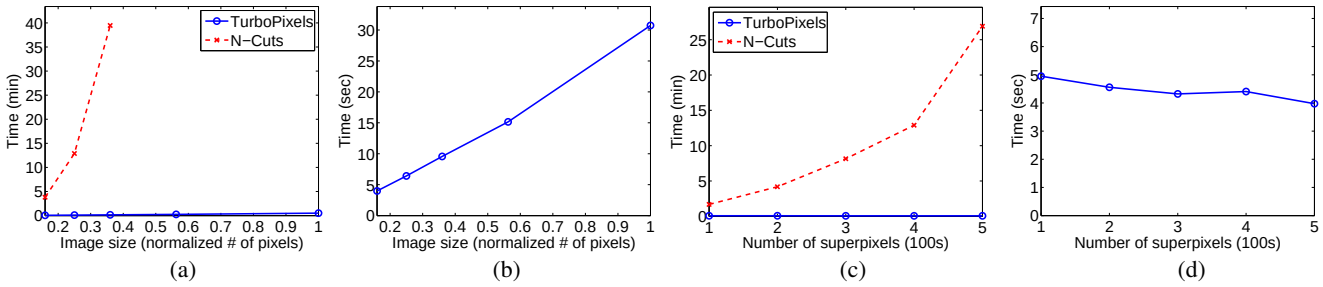


Fig. 4. Timing evaluation. (a) Running time vs. image size. (b) An expanded version (a) to show the behavior of the TurboPixel algorithm. (c) Running time vs. superpixel density. (d) An expanded version of (c) showing the behavior of the TurboPixel algorithm.

the four algorithms using this metric, with under-segmentation error plotted as a function of superpixel density. The inability of Felz to stem the bleeding is reflected in the significantly higher under-segmentation error over all three algorithms that encode a compactness constraint. Of these three, the TurboPixel algorithm achieves the least under-segmentation error.

B. Boundary Recall

Since precise boundary shape might be necessary for some applications, we adopt a standard measure of boundary recall (what fraction of the ground truth edges fall within a small distance threshold (2 pixels in this experiment) from at least 1 superpixel boundary. As shown in Fig. 3(b), Felz offers better recall at lower superpixel densities, while at higher superpixel densities, Felz and TurboPixel are comparable, with both outperforming Ncuts and Sb. The fact that Felz does not constrain its superpixels to be compact means that it can better capture the boundaries of thin, non-compact regions at lower superpixel densities.

C. Timing Evaluation

With the exception of the naive and clearly inferior Sb algorithm, the cost of enforcing a compactness constraint (Ncuts, TurboPixel) is significant; for example, Felz is, on average, 10 times faster than TurboPixel. For our timing analysis, we therefore restrict our comparison to TurboPixel and Ncuts, the two primary competitors in the class of algorithms with a compactness constraint. For any superpixel algorithm, it is appropriate to increase the number of superpixels as the image size increases, so that the expected area (in pixels) of each superpixel remains constant. Fig. 4 (a and b) shows the running time of the two algorithms as a function of increased image size. The expected size of a superpixel is kept fixed at about 10×10 pixels.

The TurboPixel algorithm is several orders of magnitude faster. It is almost linear in image size compared to Ncuts, whose running time increases non-linearly. Due to "out of memory" errors, we were unable to run Ncuts for all of the parameter settings used for the TurboPixel results. Fig. 4 (c and d) show running time as a function of superpixel density, with the image size fixed at 240×160 (one quarter of the original size). The running time of Ncuts increases in a non-linear fashion whereas the running time of the TurboPixel algorithm decreases as the density of the superpixels increases. This is due to the fact that the seeds evolve over a smaller spatial extent on average and thus converge faster.

D. Qualitative Results

Fig. 5 gives a qualitative feel for the superpixels obtained by the TurboPixel algorithm for a variety of images from the Berkeley database. Observe that the superpixel boundaries respect the salient edges in each image, while remaining compact and uniform in size.[7] Fig. 6 provides a qualitative comparison against the results obtained using Ncuts. The TurboPixel algorithm obtains superpixels that are more regularly shaped and uniform in size than those of Ncuts.

The TurboPixel algorithm is of course not restricted to work with affinity functions that are based strictly on image gradient, as discussed in Section III-E, and hence more refined measures can be used for superpixel boundary velocity. Fig. 7 shows the performance of the algorithm when the boundary velocity incorporates the Pb edge detector [15]. Note how the edge between the leopard and the background is captured much better when a Pb-based affinity is used. Moreover, the shapes of the superpixels inside the leopard are more regular for the latter case.

V. Conclusions

The task of efficiently computing a highly regular over-segmentation of an image can be effectively formulated as a set of locally interacting region growing problems, and as such avoids the high cost of computing globally optimal over-segmentations (or their approximations), such as N-Cuts. Combining the power of a data-driven curve evolution process with a set of skeletal-based external constraints represents a novel, highly efficient framework for superpixel segmentation. The results clearly indicate that while superpixel quality is comparable to the benchmark algorithm, our algorithm is several orders of magnitude faster, allowing it to be applied to large megapixel images with very large superpixel densities.

The framework is general and, like any region segmentation algorithm, is based on a user-defined measure of affinity between pixels. While our experiments have demonstrated the use of intensity gradient-based and Pb-based affinities, other more complex affinity measures, perhaps incorporating information from multiple scales, are possible. Selecting the appropriate affinity measure is entirely task dependent. We offer no prescription, but rather offer a general framework into which a domain-dependent affinity measure can be incorporated.

It is also important to note that we have intentionally skirted several important domain-dependent problems. One global issue is the fact that our framework allows the user to control the superpixel shape and density. On the issue of density, our approach is very generic, and one could imagine that with domain

[7] Supplementary material (`http://www.cs.toronto.edu/~babalex/turbopixels_supplementary.tar.gz`) contains additional results of the TurboPixel algorithm on megapixel sized images with superpixel densities in the thousands. Obtaining superpixels under such conditions using Ncuts is prohibitively expensive.

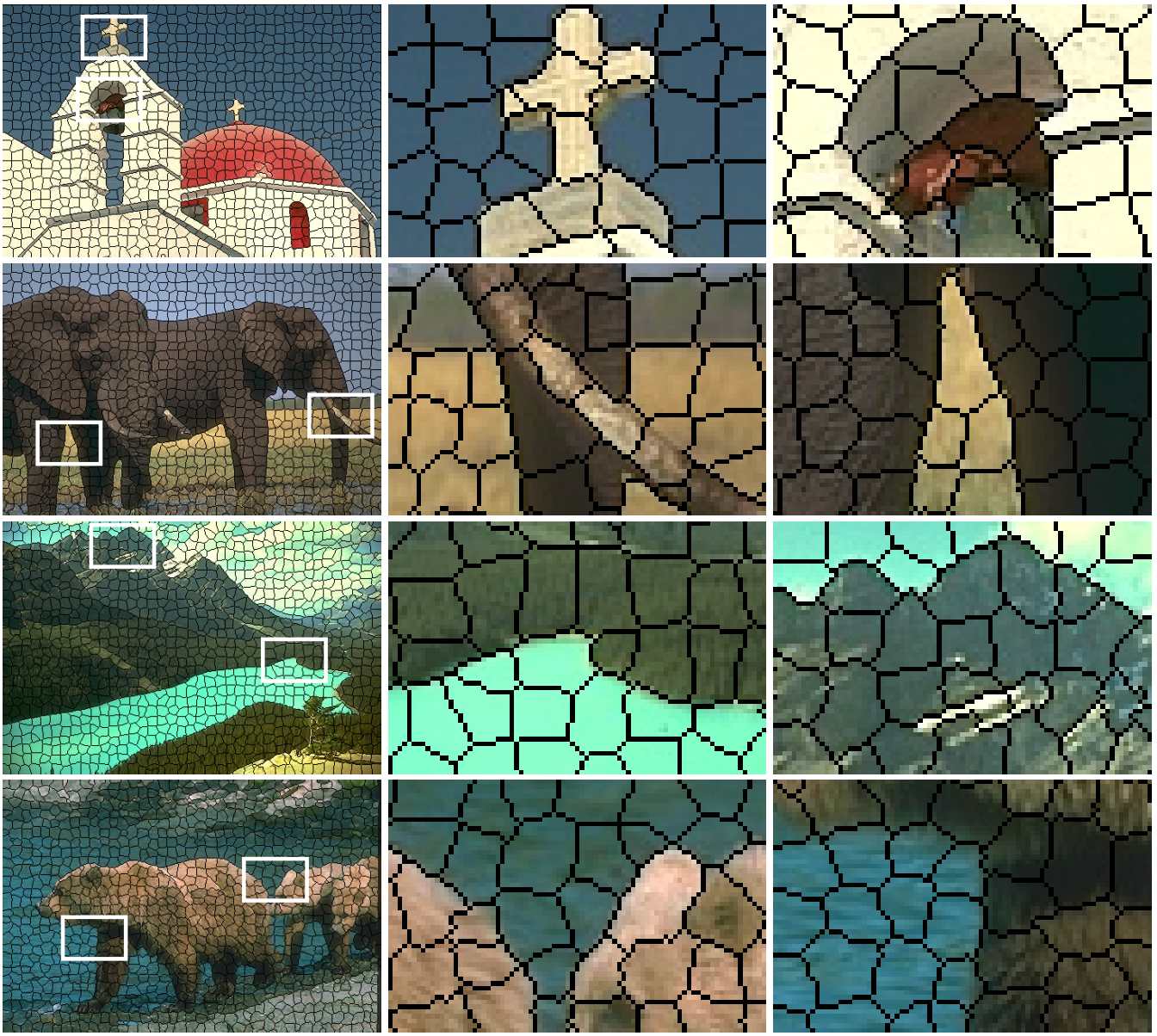


Fig. 5. TurboPixel results on a variety of images from the Berkeley database, with a zoom-in on selected regions in the middle and right columns.

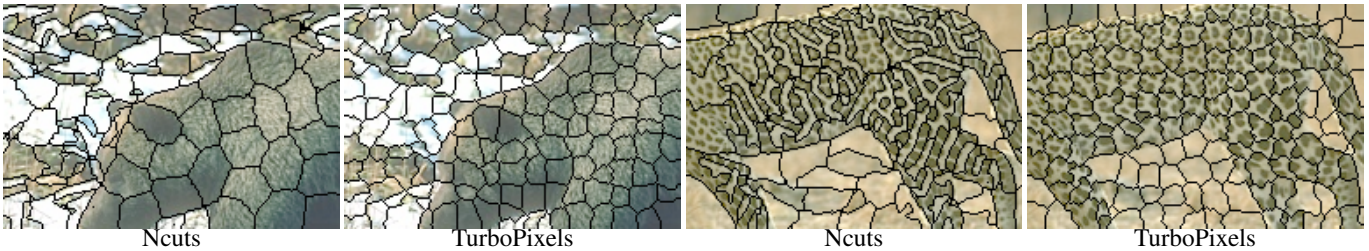


Fig. 6. A qualitative comparison of TurboPixel results with gray-level gradient-based affinity compared to results with Ncuts.

knowledge, seeds could be placed much more judiciously. And depending on the task, seeds could be placed with varying density at the cost of lower superpixel uniformity. In some domains, varying seed density may be more desirable. In textured images, for example, seeds could be placed to capture the individual texture elements better (like the spots of the leopard in Figure 6). Moreover, our framework allows us to guide superpixels to have a certain shape. Currently, in the absence of edges, the superpixels would grow in a circular manner. However, one could imagine growing superpixels to be elliptical instead. This could be more useful for extracting superpixels in narrow structures. Still, as shown in the experiments, the use of a compactness constraint clearly minimizes under-segmentation at a significantly higher computational cost. If both under-segmentation and irregularly shaped superpixel boundaries can be tolerated, the Felz algorithm is clearly the better choice, offering a tenfold speed-up as well as improved boundary recall at lower superpixel densities.

Perhaps the most important issue is what to do with the

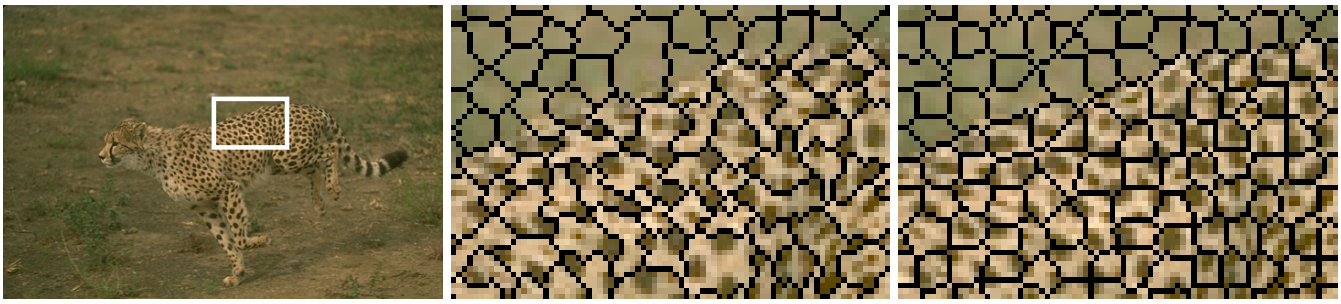


Fig. 7. Qualitative results of the TurboPixel algorithm using gradient-based (middle) and Pb-based (right) affinity functions.

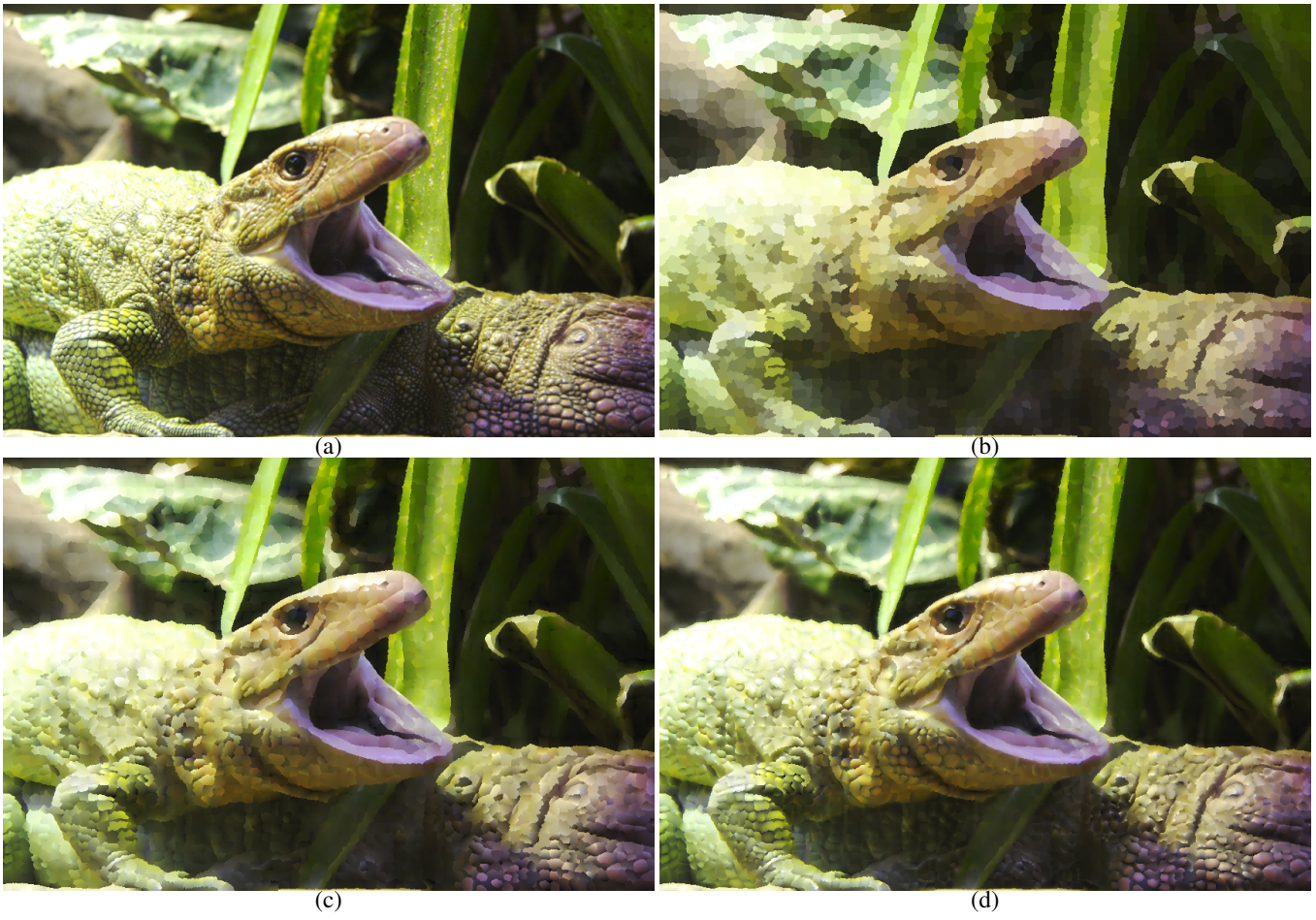


Fig. 8. Image representation using superpixels. Each superpixel from the original image (a) is colored with: (b) The average color of the original pixels in it. (c) The best linear fit to the color of the original pixels in it. (d) The best quadratic fit to the color of the original pixels in it.

resulting superpixels. The application possibilities are numerous, ranging from image compression to perceptual grouping to figure-ground segmentation. Currently, superpixels are mainly used for image labeling problems to avoid the complexity of having to label many more pixels. In the same manner, superpixels can be used as the basis for image segmentation. In the graph cuts segmentation algorithm, the affinity can be defined over superpixels instead of over pixels, resulting in a much smaller graph. Superpixels can also be considered as a compact image representation. To illustrate this idea, in Figure 8 each superpixel's color is approximated by three polynomials (one per channel). Note that whereas the mean and the linear approximations seem poor, the quadratic approximation approaches the quality of the original image.

ACKNOWLEDGMENTS

We thank Timothee Cour and Jianbo Shi for making their N-Cut package available, and Kevin Chu for his level set method library (LSMLIB), used in our TurboPixel implementation.

REFERENCES

[1] V. Caselles, F. Catte, T. Coll, and F. Dibos. A geometric model for active contours in image processing. *Num. Mathematik*, 66:1–31, 1993.
[2] V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. *IEEE ICCV*, pp. 694–699, 1995.
[3] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. PAMI*, 24(5):603–619, 2002.
[4] T. Cour, F. Benezit, and J. Shi. Spectral segmentation with multiscale graph decomposition. *IEEE CVPR*, vol. 2, pp. 1124–1131, 2005.
[5] I. Cox, S. Rao, and Y. Zhong. 'ratio regions': A technique for image segmentation. *ICPR*, pp. B:557–564, 1996.

[6] P. Felzenszwalb and D. Huttenlocher. Efficient graph-based image segmentation. *IJCV*, 59(2):167–181, 2004.

[7] Luc Vincent and Pierre Soille. Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations. *PAMI*, 13(6):583–598, 1991.

[8] X. He, R. Zemel, and D. Ray. Learning and incorporating top-down cues in image segmentation. *ECCV*, vol. 1, pp. 338–351, 2006.

[9] D. Hoiem, A. Efros, and M. Hebert. Automatic photo pop-up. *ACM Trans. Graph.*, 24(3):577–584, 2005.

[10] D. Hoiem, A. Efros, and M. Hebert. Geometric context from a single image. *IEEE ICCV*, pp. 654–661, 2005.

[11] I. Jermyn and H. Ishikawa. Globally optimal regions and boundaries as minimum ratio weight cycles. *IEEE Trans. PAMI*, 23(10):1075–1088, 2001.

[12] S. Kichenassamy, A. Kumar, P. Olver, A. Tannenbaum, and A. Yezzi. Gradient flows and geometric active contour models. *IEEE ICCV*, pp. 810–815, 1995.

[13] B. Kimia, A. Tannenbaum, and S. Zucker. Toward a computational theory of shape: An overview. *Lecture Notes in Computer Science*, 427:402–407, 1990.

[14] R. Malladi, J. Sethian, and B. Vemuri. Shape modeling with front propagation: A level set approach. *IEEE Trans. PAMI*, 17(2):158–175, 1995.

[15] D. Martin, C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Trans PAMI* 26(5):530–549, 2004.

[16] G. Mori, X. Ren, A. Efros, and J. Malik. Recovering human body configurations: Combining segmentation and recognition. *IEEE CVPR*, vol. 2, pp. 326–333, 2004.

[17] S. Osher and J. Sethian. Fronts propagation with curvature dependent speed: Algorithms based on hamilton-jacobi formulations. *J. Comp. Physics*, 79:12–49, 1988.

[18] X. Ren and J. Malik. Learning a classification model for segmentation. *IEEE ICCV*, pp. 10–17, 2003.

[19] S. Sarkar and P. Soundararajan. Supervised learning of large perceptual organization: Graph spectral partitioning and learning automata. *IEEE PAMI*, 22:504–525, 2000.

[20] E. Sharon, A. Brandt, and R. Basri. Fast multiscale image segmentation. *IEEE CVPR*, vol. 1, pp. 70–77 2000.

[21] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. PAMI*, 22(8):888–905, 2000.

[22] K. Siddiqi, S. Bouix, A. Tannenbaum, and S. Zucker. Hamilton-jacobi skeletons. *IJCV*, 48(3):215–231, 2002.

[23] K. Siddiqi, Y. Lauzière, A. Tannenbaum, and S. Zucker. Area and length minimizing flows for shape segmentation. *IEEE Trans. IP*, 7(3):433–443, 1998.

[24] H. Tek and B. Kimia. Image Segmentation by Reaction-Diffusion Bubbles. *IEEE ICCV*, pp. 156–162, 1995.

[25] S. Wang and M. Siskind. Image segmentation with ratio cut - supplemental material. *IEEE Trans. PAMI*, 25(6):675–690, 2003.

[26] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: Theory and application to image segmentation. *IEEE Trans. PAMI*, 15(11):1101–1113, 1993.

[27] S. Yu and J. Shi. Multiclass spectral clustering. *IEEE ICCV*, vol. 1, pp. 313–319, 2003.

[28] Thomas B. Sebastian, Hüseyin Tek, Joseph J. Crisco, Scott W. Wolfe, and Benjamin B. Kimia. Segmentation of Carpal Bones from 3d CT Images Using Skeletally Coupled Deformable Models. *MICCAI*, pp. 1184–1194, 1998.

[29] L. Lorigo, O. Faugeras, W. Grimson, R. Keriven, R. Kikinis, A. Nabavi, and C. Westin. Codimension-Two Geodesic Active Contours for the Segmentation of Tubular Structures. *CVPR*, pp. 444–451, 2000.