

Chapter 8

A General Framework for Contour Extraction

In the previous chapter, we described an efficient search framework for convex contour extraction. The framework is based on a simple measure of affinity between line segments, and relies on local affinity normalization to increase robustness. We described a search control technique that allows the algorithm to avoid searching through a potentially large number of variations of the same contour, and presented a robust procedure for evaluating contour quality. We then showed that the algorithm can efficiently extract convex shapes in complex imagery; in particular, it achieves a reduction of several orders of magnitude in the amount of search that has to be carried out to extract salient contours when compared to a search framework based on a boundary coverage threshold.

In this chapter, we will extend the constrained search framework to handle non-convex contours. We will describe alternate constraints that can be used in place of convexity to keep the search for contours under control, even in the presence of texture and significant amounts of clutter. We will show experimental results that indicate that the extended search framework is capable of extracting salient non-convex contours efficiently. Finally, we will propose extensions that allow for the incorporation of additional cues such as colour and texture, providing

increased robustness, and a more comprehensive evaluation of contour quality.

8.1 General Constraints for Contour Search

In the previous chapter we successfully used convexity to constrain the formation of contours. Convexity is advantageous because it removes from consideration many possible search paths at each step of contour growth, and because it biases contour growth toward closure. Without the convexity constraint, the search framework we described in the previous chapter is rendered impractical for images with moderate clutter and texture.

Observation of the behavior of such an unconstrained grouping method reveals that there are two potential combinatorial holes that the search can get drawn into. First, an unconstrained search procedure will find exponentially many possible paths to try in regions with texture or clutter, most of these paths lead to open chains, and a large number of the remaining ones describe shapes with irregular boundaries that wind their way through textured regions, and do not correspond to salient image contours. Second, the lack of a bias for closure causes the algorithm to explore paths that are on average much longer than those explored when convexity was used. Even a well constrained search will become impractical if it has to visit nodes deep down the search tree too often.

To restore search efficiency, we must achieve a large reduction in search complexity using only general constraints. We must reduce the total number of combinations of edges that need to be explored at each step. Affinity normalization goes a long way in this direction, but for non-convex contours this reduction alone is not sufficient to avoid the combinatorial sink-hole. We must also find a way to bias the search toward closure, thereby avoiding the large number of arbitrary open chains that can be generated on a typical line-set, and reducing the average depth the algorithm has to explore in the search tree.

Recall that our original affinity measure is based on proximity, and on the geometric con-

figuration of a pair of line segments:

$$T_affinity(l1, l2) = G_affinity(l1, l2) + \kappa, \quad (8.1)$$

where the geometric affinity term $G_affinity(l1, l2)$ depends on the length of the gaps or tails between each segment and the intersection point, and incorporates a measure of confidence about the location of the intersection; and κ is a suitable constant.

We expand this affinity measure so that it includes a term based on the angle difference between the segments

$$AF_{l1,l2} = (\cos(\theta_{l1 \rightarrow l2}) + 1)/2 \quad (8.2)$$

where $\theta_{l1 \rightarrow l2}$ is the angle between the segments when traveling in the direction $l1 \rightarrow l2$. This term effectively encourages grouping into smooth boundaries whenever the choice exists, and is useful in keeping the contour search algorithm from wandering into textured objects. The angle term multiplies the geometric affinity term in (8.1)

$$T_affinity(l1, l2) = (G_affinity(l1, l2)AF_{l1,l2}) + \kappa. \quad (8.3)$$

The spatial distribution of affinities given by the above formula is similar in shape to the stochastic completion field of Williams and Jacobs [113], and the tensor voting field of Guy and Medioni [40].

We use the same affinity normalization procedure described in the previous chapter to generate a set \mathcal{K} that contains the best groups that can be formed with every segment of the line-set, sorted in decreasing order of normalized affinity. Figure 8.1 shows a line-set, and the corresponding histogram of normalized affinities. The histogram shows that the shape of the distribution of normalized affinities is similar to the one for the original affinity measure, this means that the properties of normalized affinities discussed in the previous chapter still hold for the new affinity measure, and thus we can expect the same benefits of using a normalized affinity threshold to constrain the search for contours.

The second change we make to our previous search framework is that instead of convexity, we use compactness as a grouping constraint. Compactness is defined as the ratio of a contour's

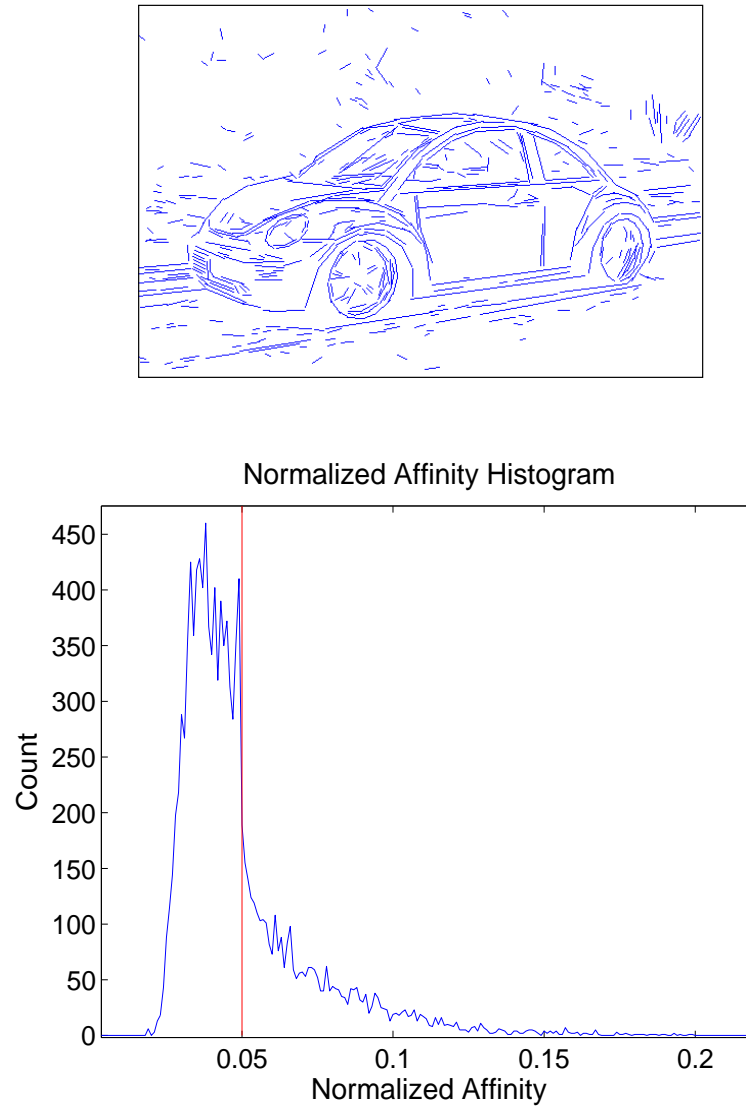


Figure 8.1: Distribution of normalized affinities for the extended affinity measure that includes the smooth continuation term. The red line corresponds to the value of $1/k$ with $k = 20$. Notice that the distribution of normalized affinities is similar to the distributions we obtained in the previous chapter, using the original affinity measure.

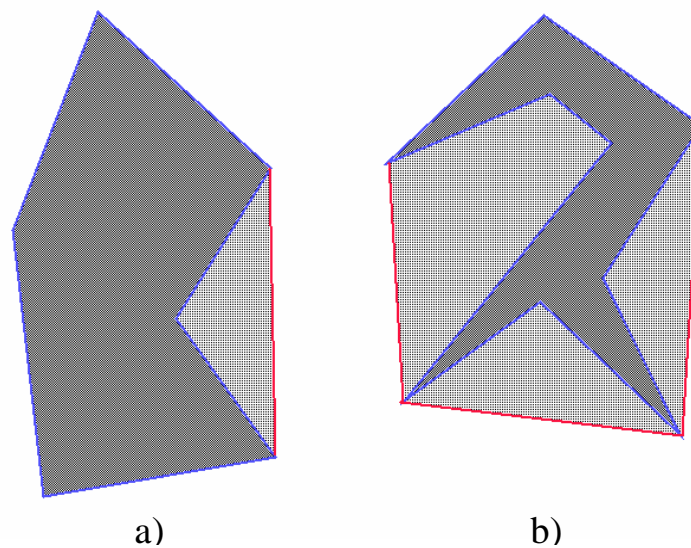


Figure 8.2: The Gestalt principle of compactness of shape. a) Compact shape, the area bounded by the contour (dark gray) is a large fraction of the area of the convex hull (dark gray plus light gray). b) Non-compact shape, in this case the area of the convex hull is much greater than the area bounded by the contour.

area to the area of the convex hull of the contour, this is illustrated in Figure 8.2. Saund [93] has previously demonstrated the use of compactness in the context of contour grouping, here we follow the same formulation, and impose a minimum threshold τ_c on compactness that every chain of edges must satisfy to avoid being pruned. For partial chains, we consider the area of the closed contour formed by joining the first and last vertices of the chain with a straight line. Compactness is an attractive constraint for grouping because it keeps contours from forming complicated, twisting shapes, and because it encourages closure (though not as strongly as convexity).

We have found that with the above modifications, our framework is capable of keeping the search for closed contours under control, even for images with moderate clutter and texture. The updated search algorithm is the same as the method described for convex groups, but with compactness instead of convexity, and using the modified affinity measure:

- 1 - For each segment i in the line-set, generate an initial group containing i , set $\tau = \tau_0$.
- 2 - Find the set \mathcal{K} with the best k junctions for the last segment in the group (sorted by decreasing normalized affinity).
- 3 - For each segment j in \mathcal{K}
 - If $N_affinity(i, j) < \tau$ end loop, otherwise add j to the group.
 - Check whether j is covered by a previously found group, if so, try the next j .
 - Test for compactness, if compactness $< \tau_c$ try the next j .
 - Test for simplicity (no self-intersections, no spiral shapes), if test fails try the next j .
 - Check for closure, if the group is closed, compute its saliency and report it.
 - Increase $\tau = \tau + \epsilon_\tau$.
 - Go to step 2.
- 4 - Report the extracted polygons sorted in order of decreasing saliency.

Contour saliency is estimated using Qualitative Probabilities with the same procedure used for the convex contour case. However, the task of ranking the extracted contours becomes more difficult given that there is a substantially larger number of shapes that can be generated on any line-set, and many of these correspond to variations of a few salient shapes. With convex contours, we reduced the number of such shapes by testing each contour for similarity against previously extracted shapes. Two polygons were considered similar if the endpoints of one contour were closer than some small distance from an edge in the other, and vice versa. This simple test was sufficient to establish similarity for convex shapes, but it can fail for general contours in cases such as the one illustrated in Figure 8.3.

To remove variations of the same shape in the new framework, we test for overlap between the areas enclosed by the contours. If the overlapping area is greater than some threshold

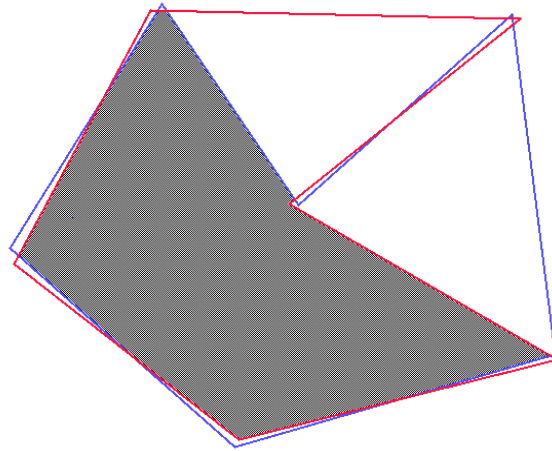


Figure 8.3: Our old equivalence test is not sufficient to detect similar shapes anymore. The image above shows two distinct contours (blue and red) that pass the old similarity test (all the vertices of each polygon are within a small distance of the other contour). We determine that the contours are different by computing the ratio of the overlapping area (gray) to the area of each polygon, we say the contours are equivalent if both ratios are above .9.

(relative to the area of each shape), we say the contours are equivalent, and keep only the one with the highest QP rank. We can check for overlap efficiently by first computing the overlap between the bounding boxes of the contours, and only performing a complete overlap test if the bounding boxes have more than 75% overlap. For shapes that satisfy this condition, a complete overlap computation is performed using a fast polygon scan-conversion routine. This procedure successfully eliminates many of the repeated contours found by the algorithm.

8.2 Experimental Results

Here we show contour extraction results on several images, we use a threshold on compactness of .75, an overlap threshold (for equivalence testing) of .9, and the same values for the parameters of the affinity function that we used for convex polygons, namely: $\sigma_{gap} = 20$, $\sigma_{tail} = \sigma_{gap}/2 = 10$, $\sigma_u = 15$, $\kappa = .25$. Once more, we set $k = 20$ to be the number of groups to consider for normalization for each segment, and set the normalized affinity threshold to $1.2/k$. These single set of parameters results in efficient contour extraction on a variety of images, which agrees with the observation made for convex groups that the use of normalized affinities provides excellent robustness to variation in the input line-set.

Figures 8.4-8.9 show some of the contours extracted from several line-sets. There are several observations to be made here, first, that many of the contours found by the algorithm correspond to salient image structure, though ranking the contours is now more complicated, which means that salient contours now appear further down the list of extracted shapes. Secondly, though we introduced a bias for smooth continuation, the algorithm is capable of finding contours with sharp bends, this is a result of the affinity normalization procedure. Whenever a sharp turn offers the best possible choice for grouping within the local neighborhood of a segment, the normalized affinity for the group will be large, regardless of the fact that the absolute affinity value may be low because of the angle component.

For all our tests, the number of nodes searched, the number of contours extracted, and

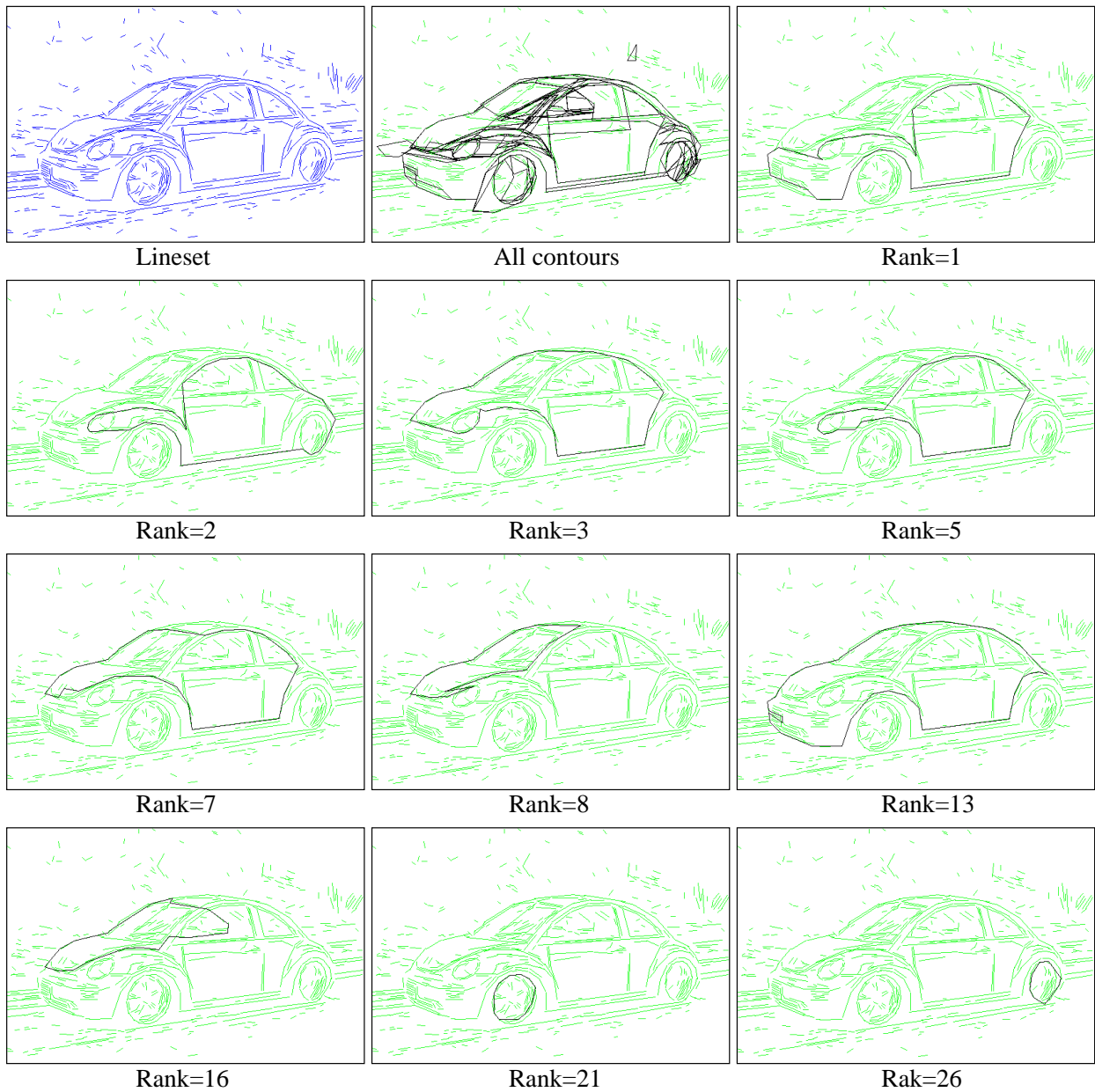


Figure 8.4: Beetle line-set with 591 segments, and several of the extracted contours (the QP rank is noted below each image). The algorithm explored 4.2×10^5 nodes, found 87 distinct contours, and took 28 seconds to complete.

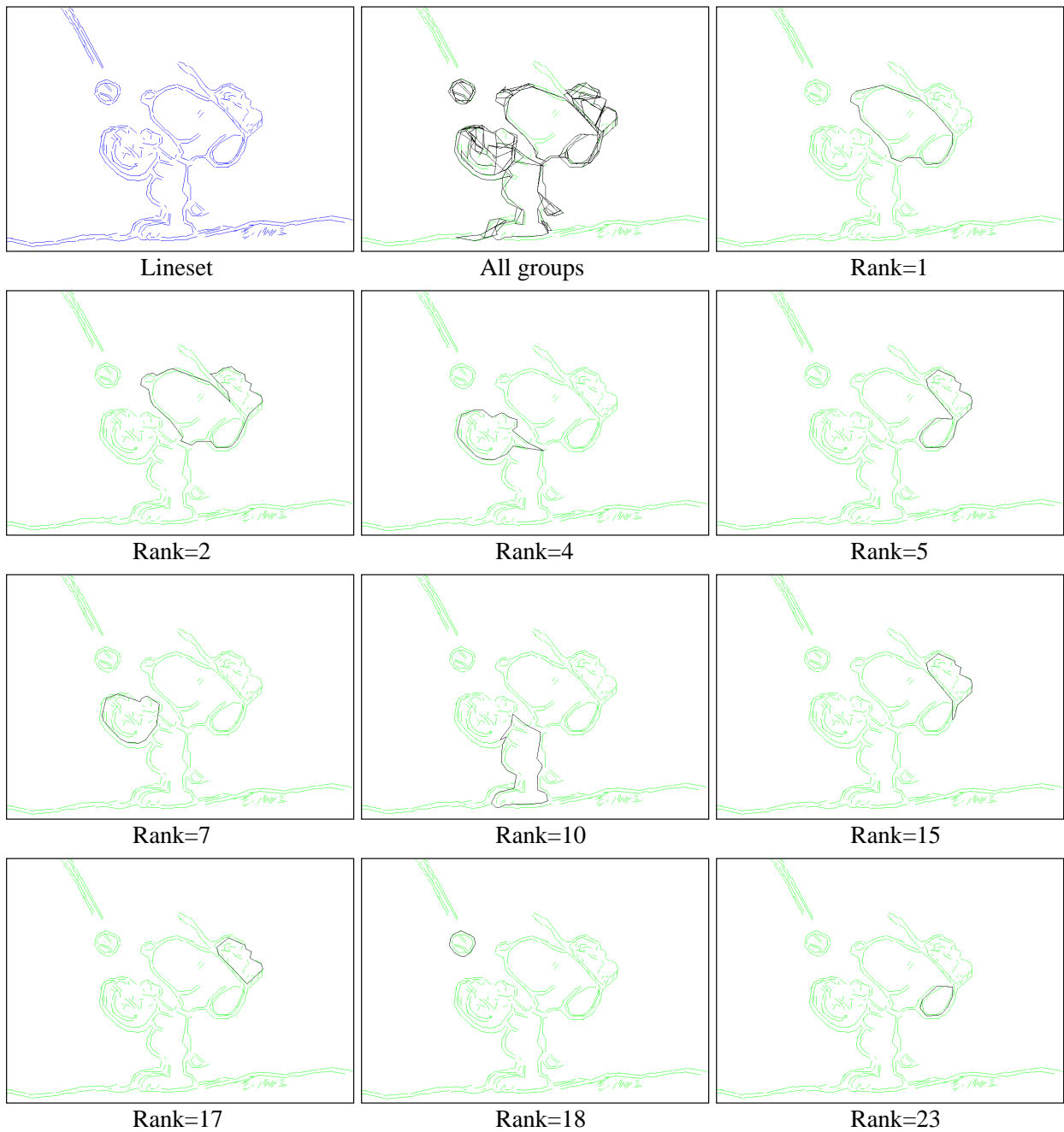


Figure 8.5: Snoopy4 line-set with 432 segments, and several of the extracted contours. The algorithm explored 5.0×10^5 nodes, found 94 distinct contours, and took 58 seconds to complete.

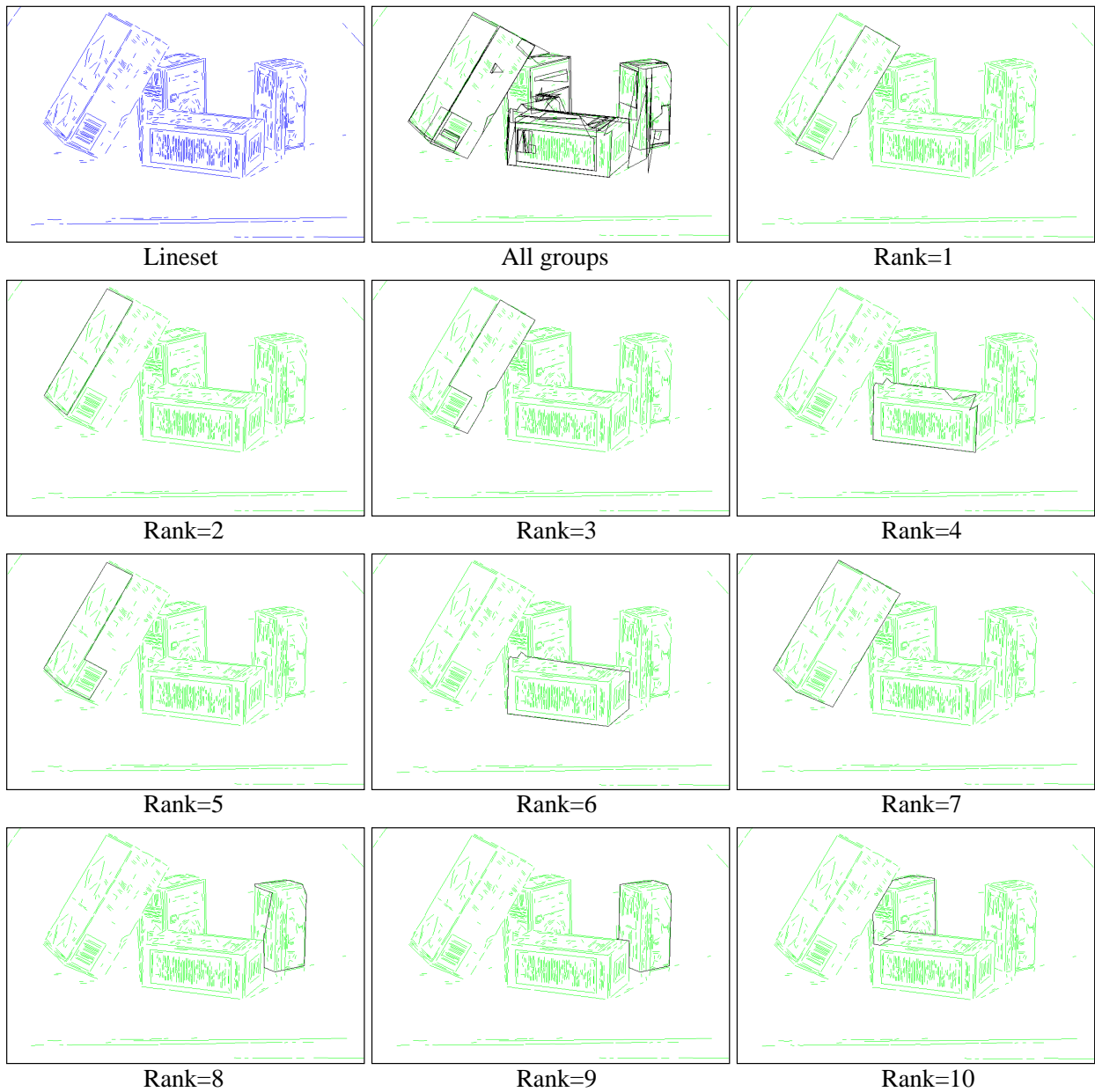


Figure 8.6: Boxes2 line-set with 829 segments, and several of the extracted contours. The algorithm explored 8.9×10^5 nodes, found 92 distinct contours, and took 35 seconds to complete.

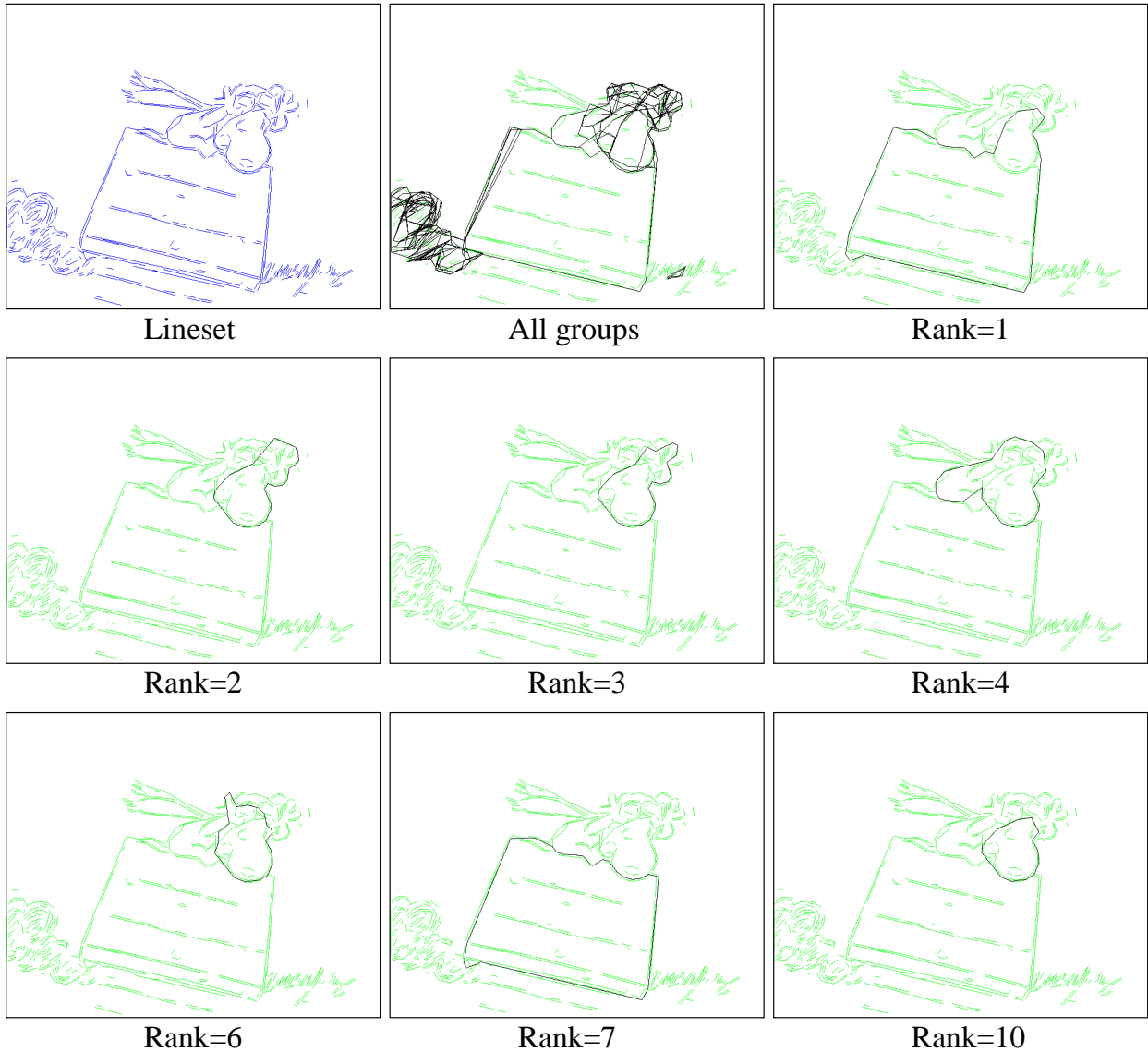


Figure 8.7: Snoopy7 line-set with 506 segments, and several of the extracted contours. The algorithm explored 5.2×10^5 nodes, found 141 distinct contours, and took 45 seconds to complete.

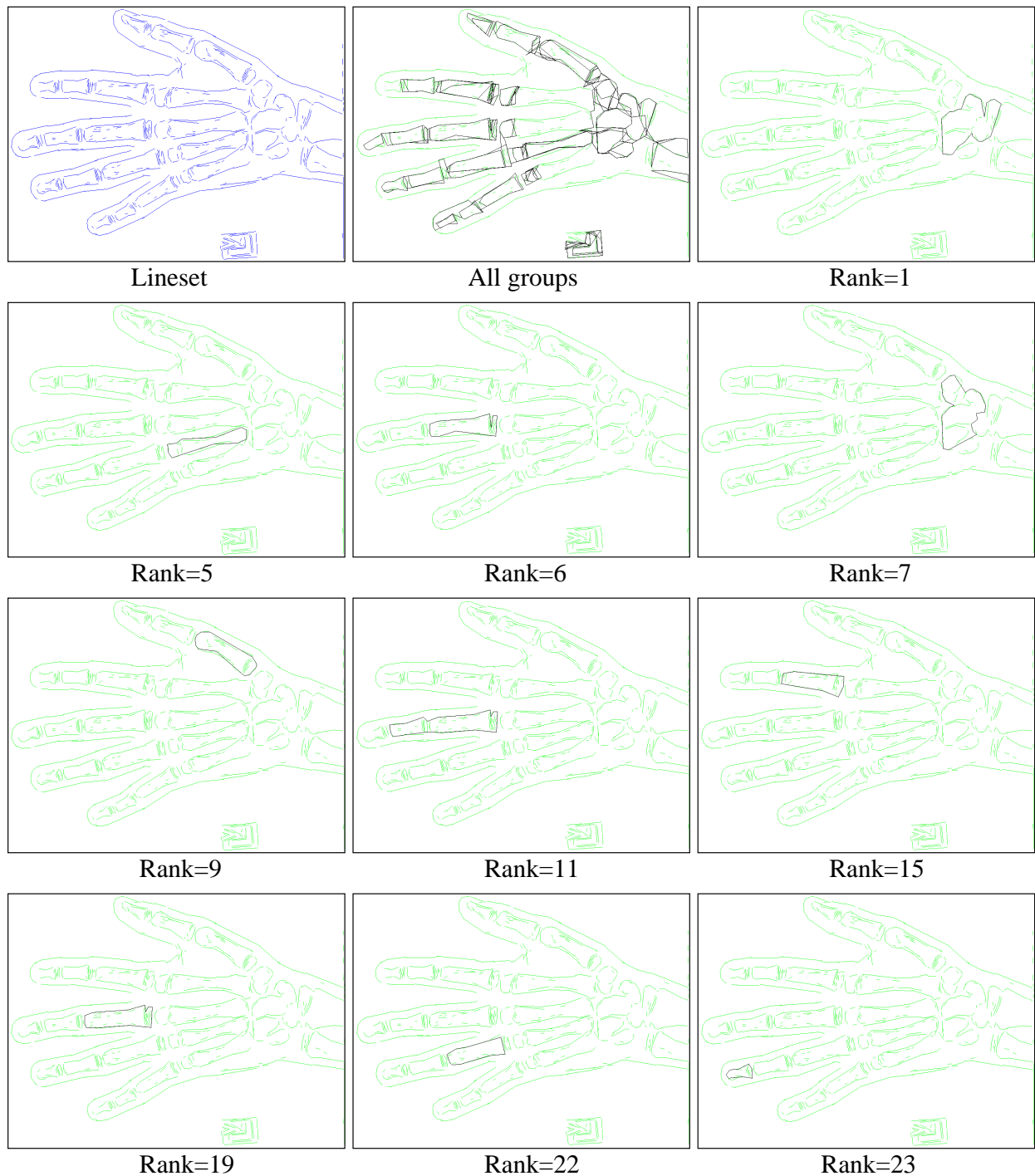


Figure 8.8: X-ray line-set with 726 segments, and several of the extracted contours. The algorithm explored 3.7×10^5 nodes, found 136 distinct contours, and took 30 seconds to complete.

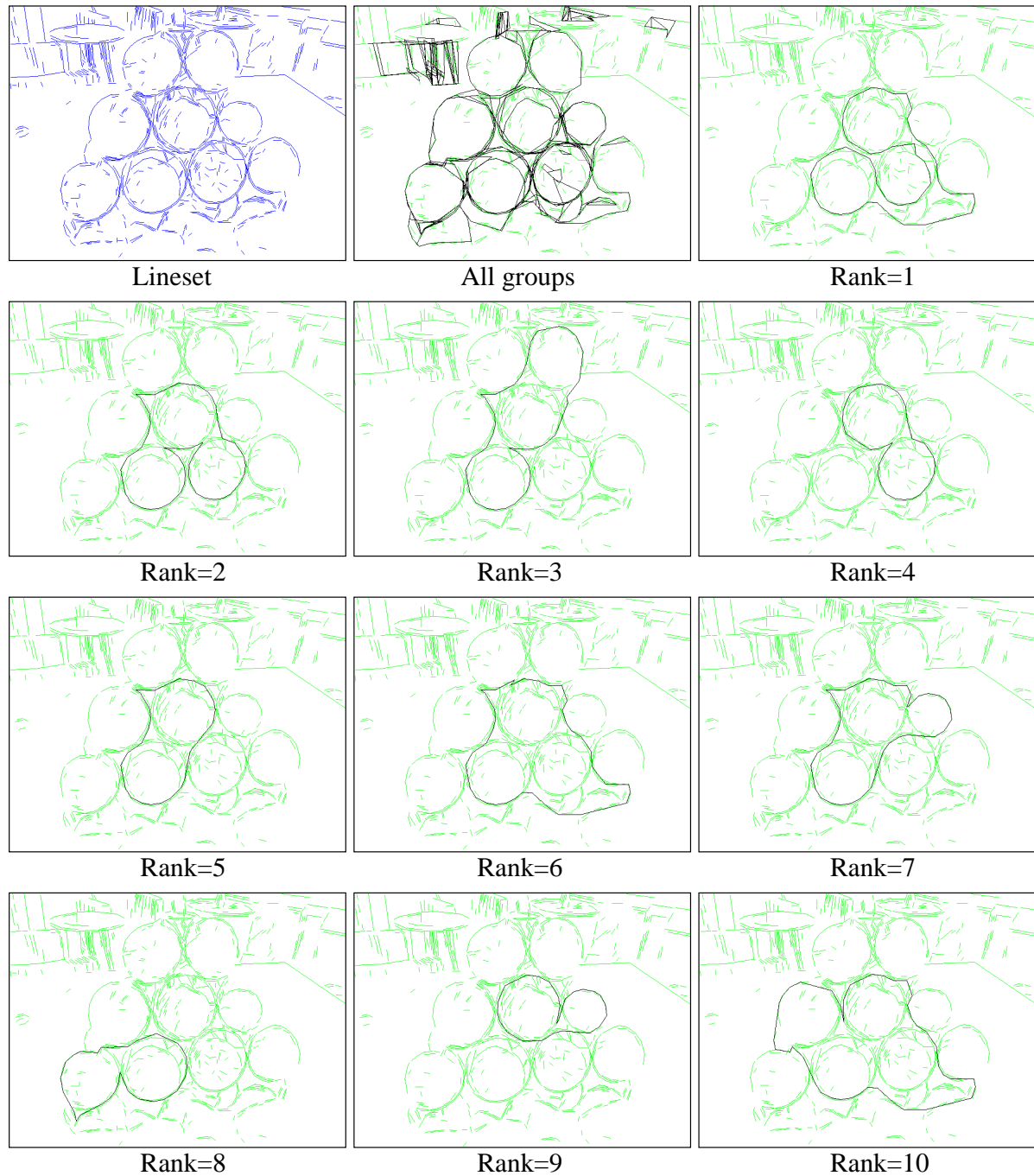


Figure 8.9: Fruits line-set with 797 segments, and several of the extracted contours. The algorithm explored 8.1×10^5 nodes, found 102 distinct contours, and took 77 seconds to complete.

the run-times are comparable to those obtained for convex groups (run-times were measured on the same P4, 1.9GHz machine used for testing the convex group extraction algorithm). This is particularly important because it indicates that the use of compactness and smooth continuation within our search framework reduces search complexity to the point of making the general contour extraction problem comparable to the significantly more constrained problem of detecting convex groups.

Though the algorithm described above can successfully extract salient contours for objects that are not heavily textured, without additional information the contour grouping algorithm may wander into textured regions and generate a multitude of groups that do not represent object boundaries, Figure 8.10 illustrates this situation. On such line-sets, the information provided by the segments is ambiguous in that there are too many paths through the textured regions of the image that lead to closed contours. Though the search itself can be carried out efficiently, the extracted contours are unlikely to represent salient image structure. In fact, determining contour saliency in such cases becomes problematic. The Qualitative Probabilities framework shows a preference for contours that wind their way through texture, because such contours can account for more of the observed segments in the line-set.

However, we will show that it is possible to bring additional information into the search framework to guide contour growth, this information is used to favour the grouping of contours that are more likely to correspond to object boundaries. In particular, we will show that colour information can be used to bias contour growth, and that the search framework allows for a dynamic bias that may depend on accumulated statistics that change as the contour grows.

8.3 Cue Integration for Enhanced Contour Extraction

Up to this point, we have only used gestalt grouping principles to decide which groups look better. However, our search framework can easily be extended to incorporate other sources of information. In this section, we will extend our algorithm to use colour information from

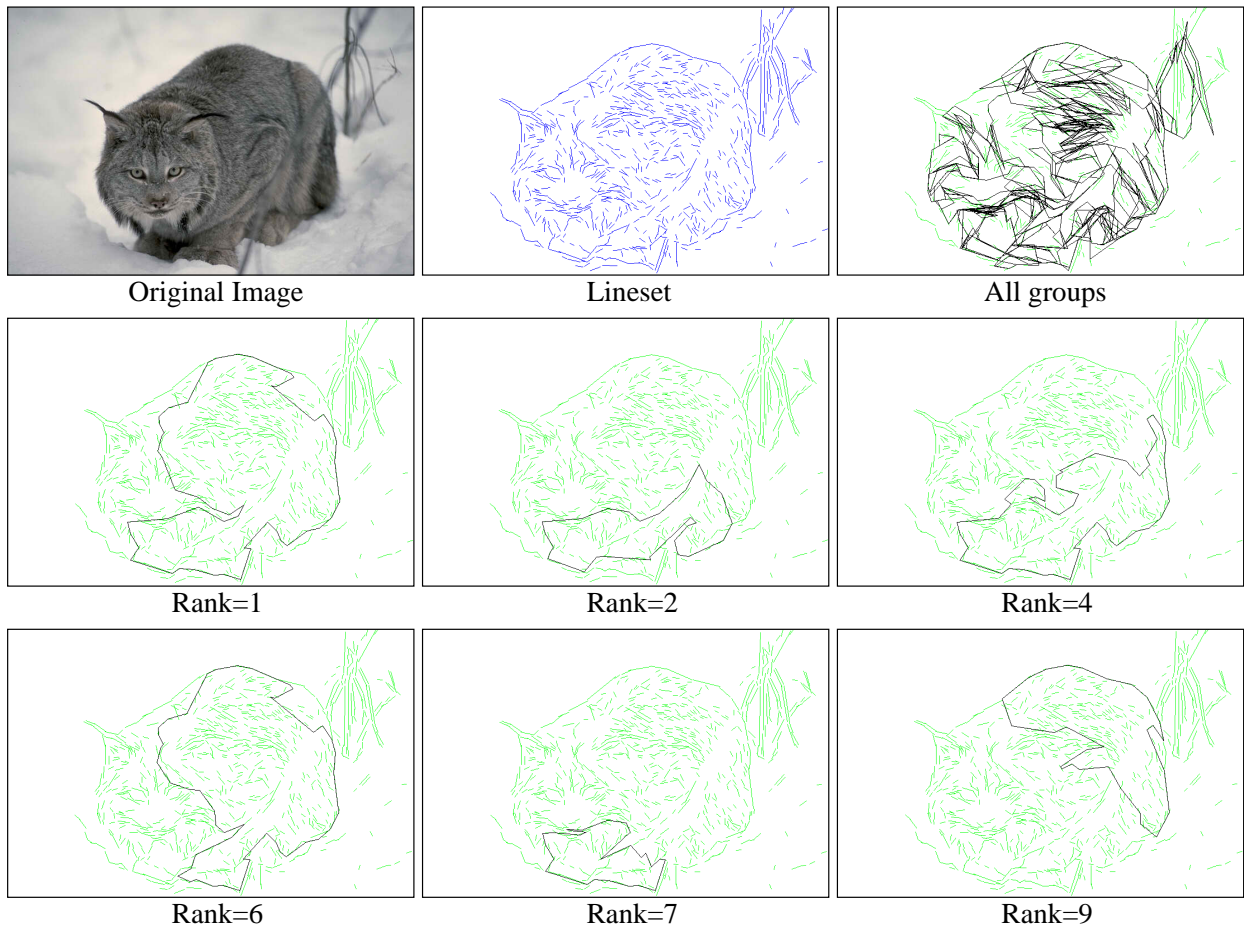


Figure 8.10: Beliris image and line-set, 784 segments, and several of the extracted contours. On this image, the algorithm generates many contours that wind through the textured region. These contours receive a high QP score because they account for many of the observed line-segments. The algorithm explored 8.9×10^5 nodes, found 232 distinct contours, and took 66 seconds to complete.

the image to guide group formation, and also to evaluate the saliency of output contours. We will not explore the difficult problem of defining a suitable colour similarity measure, instead, we use a simple metric based on colour-histograms, and focus on the problem of using the additional information provided by this metric to bias the search for contours.

There are two ways in which our framework can incorporate additional image cues. We could add another term to the affinity function just like we did in the previous section to bias the contours toward smooth continuation. The colour metric we propose here could be incorporated in this fashion, however, we want to illustrate the fact that a search-based algorithm offers the possibility of modifying the affinity values at search-time.

This is important because it means the search can be biased using non-local cues and accumulated statistics that depend on the shape of a partially completed contour. We will incorporate colour information in this fashion as a proof of principle, and leave the exploration of other useful cues for future work.

8.3.1 Local Colour Histograms

For our purposes, we will represent colour information in the form of local colour histograms, these histograms are computed over neighborhoods of 25×25 pixels centered at a particular image location (x, y) . Each histogram contains 20 bins, and three layers corresponding to three colour components. The colour values at each pixel within the neighborhood are accumulated onto the histogram, with contributions weighted by a 2-D Gaussian with $\sigma_x = \sigma_y = 8$ centered at (x, y) . The choice of σ is determined by the window size, so that at least 1.5σ fit within the window on either side. The window size and the number of bins were set experimentally, and were found to yield good results. We will therefore use these values in all our tests. Determining the optimal values for these parameters on an arbitrary image is a difficult problem we will not explore here.

Since the 2-D Gaussian used to weight the contribution of each pixel to the histogram has unit area, each layer of the resulting histograms will also have unit area. However, we know that

the RGB values in the image are highly correlated, so instead of computing RGB histograms, we use the opponent color model [106]

$$L = R + G + B, \quad YB = R + G - (2 * B), \quad RG = R - G, \quad (8.4)$$

where L is the luminance channel, YB represents yellow minus blue, and RG represents red minus green. These colour components are less correlated, so their histograms are more informative. In practice, we normalize the values on each channel to be within $[0, 1]$, that is

$$L = \frac{R + G + B}{3}, \quad YB = \frac{R + G - (2 * B) + 2}{4}, \quad RG = \frac{R - G + 1}{2}. \quad (8.5)$$

Figure 8.11 shows a small image region, a 25×25 window selected for analysis, the Gaussian mask that determines the weight of each pixel's contribution to the histograms, and the RGB and $L - YB - RG$ histograms for the window.

Before the search phase begins, we compute and store local colour histograms over the complete image using a uniform grid with 3 pixels between neighboring histograms in the vertical and horizontal directions. Given the size of the histogram windows, we could use a sparser grid, but we find the denser sampling useful at search time.

8.3.2 Biasing the Search Using Histogram Similarity

Given the local colour histograms, we can bias the growth of a contour by favouring groups that bound regions of the image with similar colour distributions. More formally, we say that each partial contour separates the image into two regions: an inside region, which lies to the right of the contour when traveling in the direction of contour growth, and an outside region that lies to the left of the contour. We maintain, and update colour histograms for the inside and outside colour distributions along the partial contour.

We associate each segment along the contour with two local histograms, one for the inside and one for the outside (depending on the direction of contour growth). To select the appropriate local histograms, we look in the direction perpendicular to the segment starting from

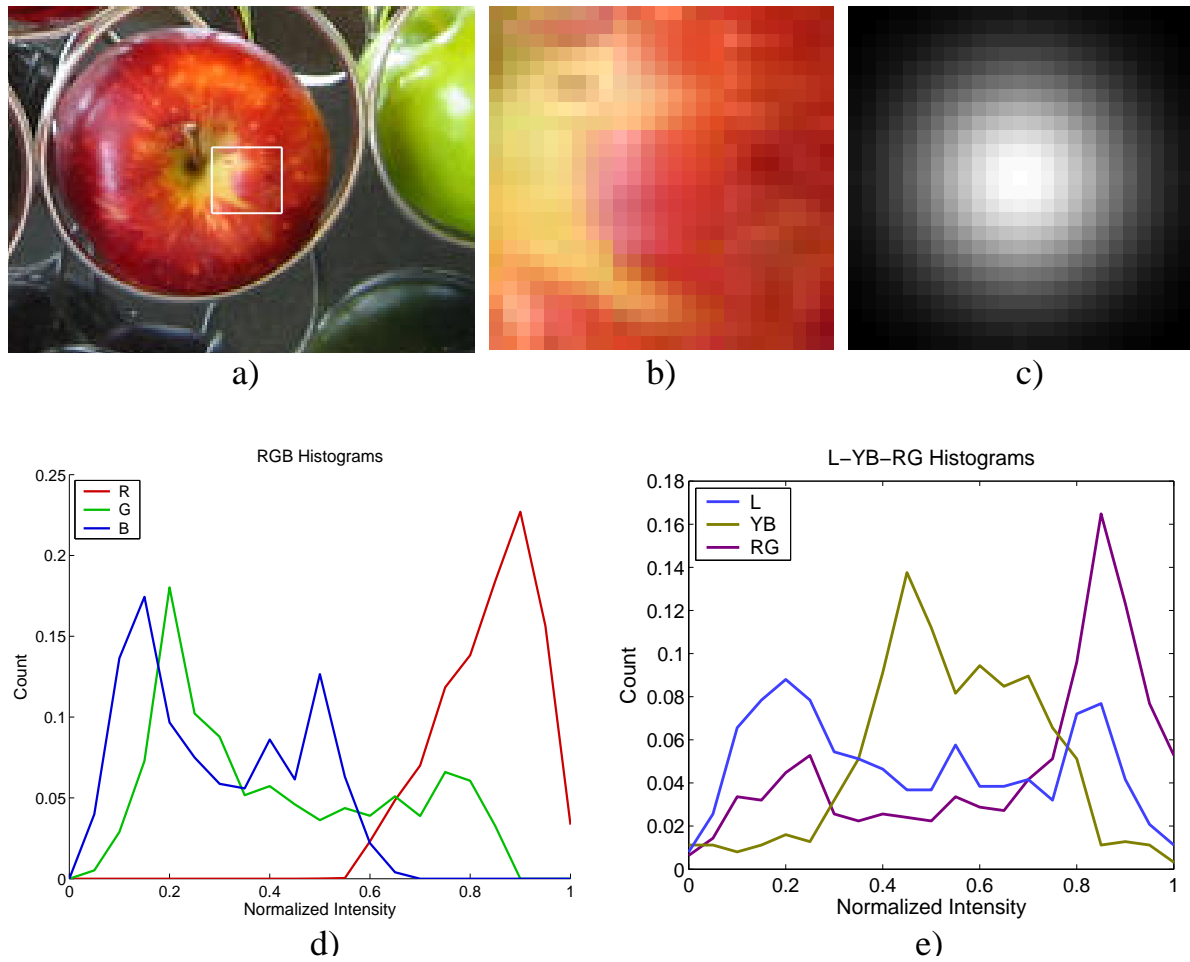


Figure 8.11: Colour histogram computation. a) Input image with a 25×25 region of interested marked by a white square. b) The selected image window. c) Gaussian weights that determine each pixel's contribution to the histograms. d) RGB histograms. e) Y-LB-RG histograms.

whose area has been normalized to 1, this measure yields a value within $[0, 1]$, with 1 indicating that the histograms are identical.

To simplify notation, let us define a colour histogram $H^{L,YB,RG}$ that is formed by concatenating the three colour components of the local histograms defined above. We normalize this concatenated histogram to have unit area. Given the contour's inside histograms $H_{\text{contour_in}}^{L,YB,RG}$, its outside histograms $H_{\text{contour_out}}^{L,YB,RG}$, and a segment's inside and outside histograms $H_{\text{segment_in}}^{L,YB,RG}$ and $H_{\text{segment_out}}^{L,YB,RG}$. We compute the histogram intersections between the two inside histograms, and between both combinations of inside-outside histograms (inside of segment vs. outside of contour, and vice versa), we are not interested in the similarity between the outside histograms since it is assumed that the background may change arbitrarily along the contour.

We use these intersections to estimate the similarity between the colour distributions on the inside, and the dissimilarities between the colour distributions on opposite sides of the boundary:

$$\begin{aligned} S_{in} &= \cap(H_{\text{contour_in}}^{L,YB,RG}, H_{\text{segment_in}}^{L,YB,RG}), \\ D_{i \rightarrow o} &= 1 - \cap(H_{\text{contour_out}}^{L,YB,RG}, H_{\text{segment_in}}^{L,YB,RG}), \\ D_{o \rightarrow i} &= 1 - \cap(H_{\text{contour_in}}^{L,YB,RG}, H_{\text{segment_out}}^{L,YB,RG}), \end{aligned}$$

We define the histogram based colour affinity between a segment l and a partial contour C as

$$Col_{\text{affinity}}(l, C) = (S_{in} \cdot D_{i \rightarrow o} \cdot D_{o \rightarrow i}) + \kappa_c. \quad (8.7)$$

The first term in the colour affinity is just the similarity between the inside colour distributions of the segment and the contour. The second term is the dissimilarity between the colour distribution on the outside of the new segment, and the inside of the contour; it discourages grouping with segments that are inside an object (in which case, the inside similarity is high, but the outside-inside dissimilarity will be low). The last term is the dissimilarity between the inside of the contour and the outside of the new segment; it is useful when no candidate segment can be found whose inside colour distribution matches the inside of the contour. In

this case, we favour the segment whose outside is most dissimilar to the inside of the contour. Notice that once again we add a constant term $\kappa_c = .1$, this constant puts a lower bound on the value of the colour affinity, and ensures that the affinity measure behaves appropriately after normalization.

Incorporating the colour affinity into the search framework is straightforward. Given the set \mathcal{K} that contains the best k grouping choices for the last segment in the contour, we compute the colour affinity of each of these k segments with regard to the current contour's inside and outside colour distributions, and multiply the segment's normalized affinity by the colour affinity value. We then re-normalized the affinities of the k segments to add up to 1.

Once a segment has been chosen for grouping, it is added to the contour, and its inside and outside histograms are accumulated onto the contour's inside and outside histograms, which are then re-normalized to have unit area. The process is repeated with the new set \mathcal{K} of the latest segment added to the contour. In general, this procedure will change the order in which segments are tried out for grouping. Some grouping choices that would have been explored before may not be tried once colour is taken into consideration, similarly, segments that might not have been considered before will be given a chance to form groups if their colour distributions are favorable. These effects balance out after re-normalization, so the addition of colour information should not increase the amount of search performed. In the extreme cases when the colour affinity is uninformative (either because it is equally high, or equally low for all segments), the original search ordering provided by the normalized line-segment affinities will persist. Our extended algorithm for contour extraction is summarized below.

- 1 - For each segment i in the line-set, generate an initial group containing i , set $\tau = \tau_0$. Initialize the inside/outside histograms for the contour to be local histograms of i (this will depend on the direction toward the next segment).
- 2 - Find the set \mathcal{K} with the best k junctions for the last segment in the group (sorted by decreasing normalized affinity).

- 3 - Compute the colour affinity for each segment in the set \mathcal{K} , and multiply it with the segment's normalized affinity.
- 4 - Re-normalize the affinities of the k segments to sum to 1.
- 5 - For each segment j in \mathcal{K}
 - If $N_affinity(i, j) < \tau$ end loop, otherwise add j to the group.
 - Check whether j is covered by a previously found group, if so, try the next j .
 - Test for compactness, if compactness $< \tau_c$ try next j .
 - Test for simplicity, if test fails try the next j .
 - Check for closure, if the group is closed, compute its saliency and report it.
 - Accumulate j 's inside/outside histograms onto the contour's inside/outside histograms, and re-normalize the contour's histograms to have unit area.
 - Increase $\tau = \tau + \epsilon_\tau$.
 - Go to step 2.
- 5 - Report the extracted polygons sorted in order of decreasing saliency.

8.4 Using Colour to Determine Saliency

We discussed above that our original ranking procedure using Qualitative Probabilities tends to favour contours that wind through texture because such contours account for a larger number of the observed segments. Here we define a measure of saliency that depends on how well the complete image is modeled by the colour histograms for the inside and outside of a closed contour. This metric is based on the assumption that a good contour should separate two reasonably distinct colour distributions.

Given the colour histograms $H_R^{L,YB,RG}$ for image region R , we compute the cost of encoding a pixel $\vec{p} = (p_L, p_{YB}, p_{RG}) \in R$, as the probability of drawing the pixel's colour

components from the colour distributions specified by the region's histograms. Assuming independence between colour components, we have

$$\text{cost}(\vec{p}|H_R^{L,YB,RG}) = H_{R,\text{bin}(p_L)}^L \cdot H_{R,\text{bin}(p_{YB})}^{YB} \cdot H_{R,\text{bin}(p_{RG})}^{RG}. \quad (8.8)$$

If we treat each pixel as an independent sample from R 's colour distribution, we can compute the encoding cost for all pixels within the region as

$$\text{Region_cost}(R) = \prod_{\vec{p} \in R} \text{cost}(\vec{p}|H_R^{L,YB,RG}). \quad (8.9)$$

Finally, we take the negative log of (8.9) to obtain the region's encoding cost,

$$\text{Encoding_cost}(R) = - \sum_{\vec{p} \in R} \log(\text{cost}(\vec{p}|H_R^{L,YB,RG})), \quad (8.10)$$

which will be small for regions with homogeneous colour distributions.

For each contour C , we compute colour histograms $H_{in}^{L,YB,RG}$ and $H_{out}^{L,YB,RG}$ for the inside and outside of the region R bounded by the contour (notice that these are different from the accumulated inside and outside histograms used during search, which only include data from the local histograms near the segments that form the contour). From these histograms we compute the encoding cost for the image

$$\text{Encoding_cost}(I|C) = - \sum_{\vec{p} \in R} \log(\text{cost}(\vec{p}|H_{in}^{L,YB,RG})) - \sum_{\vec{p} \notin R} \log(\text{cost}(\vec{p}|H_{out}^{L,YB,RG})). \quad (8.11)$$

We expect the encoding cost to be small when the inside region is homogeneous in appearance, and distinct from parts of the image that lie outside the contour. This encoding cost favours larger regions as long as the inside histogram for such regions does not become too flat. To account for this effect, and to further enhance the saliency of contours that bound homogeneous image regions, we compute a contour saliency score as

$$\text{saliency} = \alpha \frac{\sum_{l \in C} -\log(\text{Col}_{\text{affinity}}(l, C))}{N} + (1 - \alpha) \text{Encoding_cost}(I|C) \quad (8.12)$$

where l is a segment that forms part of the contour, N is the number of segments in the contour, and the first term represents the average log-colour affinity along the contour. The parameter α

controls how much weight we give to each of the terms in the saliency equation, and we set it to $\alpha = .1$ for all our tests.

The saliency value defined above is computed for every polygon in the final contour list, and the extracted shapes are sorted in decreasing order of saliency. To speed up saliency computation, we pre-compute a colour histogram for the complete image. The inside histogram for a region is computed for each contour, but we generate the outside histogram by subtracting the inside histogram from the pre-computed full image histogram.

8.5 Experimental Results Using Colour Information

Figure 8.13 shows contour extraction results on the same image used in Figure 8.10, notice that this time the algorithm manages to recover a more reasonable boundary, and that the top groups are less irregular than before. Figures 8.14 and 8.15 show contour extraction results on two other images. We used the same normalized affinity threshold and algorithm parameters described before. For these figures we used a normalized affinity threshold of $1.4/k$, and the same parameters for all other algorithm parameters. We can use a larger threshold on normalized affinity because the bias introduced by the colour affinity increases the normalized affinity of good boundary groups.

In general, the use of colour information leads to contours that better capture salient image structure. The ranking of the contours is also improved, with polygons that capture interesting image structure closer to the top of the list. The increased overhead of computing colour affinities, re-normalizing and re-ordering the list of grouping candidates, and computing contour saliency increases the run-time of the algorithm, but the increase is not significant given the large computational savings achieved during the search phase.

More importantly, the above images show the feasibility of integrating other image cues into the search framework. Additional cues can be integrated to the affinity measure computed before the search begins, or they can be incorporated at search-time, by modifying and re-

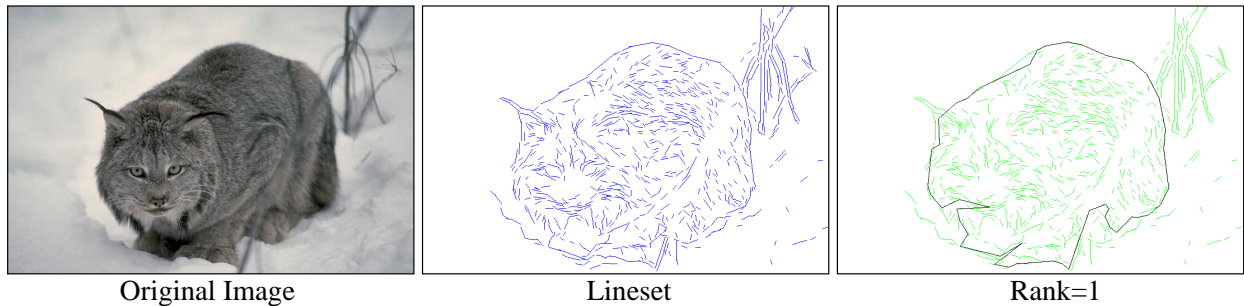


Figure 8.13: Contour extraction using colour information. Original image, line-set with 784 segments, and top-ranked contour (all the remaining contours describe small, closed polygons within the textured region). The algorithm explored 3.3×10^5 nodes, found 104 distinct contours, and took 42 sec. to complete. Compare these results with the contours shown in Fig. 8.10.

normalizing the affinity values according to accumulated statistics or non-local cues.

It is illustrative to consider which paths within the line-set are being explored by the algorithm, Figure 8.16 shows four images, the corresponding line-sets, and traces of the search paths explored by the algorithm on each line-set. The paths that were traced most often agree well with salient image contours, which indicates that the majority of the search effort is taking place along interesting image boundaries. This provides further evidence that the search framework is capable of efficiently and robustly selecting search paths that are likely to lead to salient contours.

8.6 Discussion

In the previous chapters, we have presented a framework for contour extraction based on the use of normalized, pairwise affinities between line segments. We have demonstrated the algorithm first on convex contours, for which we showed that the algorithm is robust, and provides a reduction in search complexity of several orders of magnitude, compared to a competing convex group detection method based on the prior state-of-the-art. We then showed that the

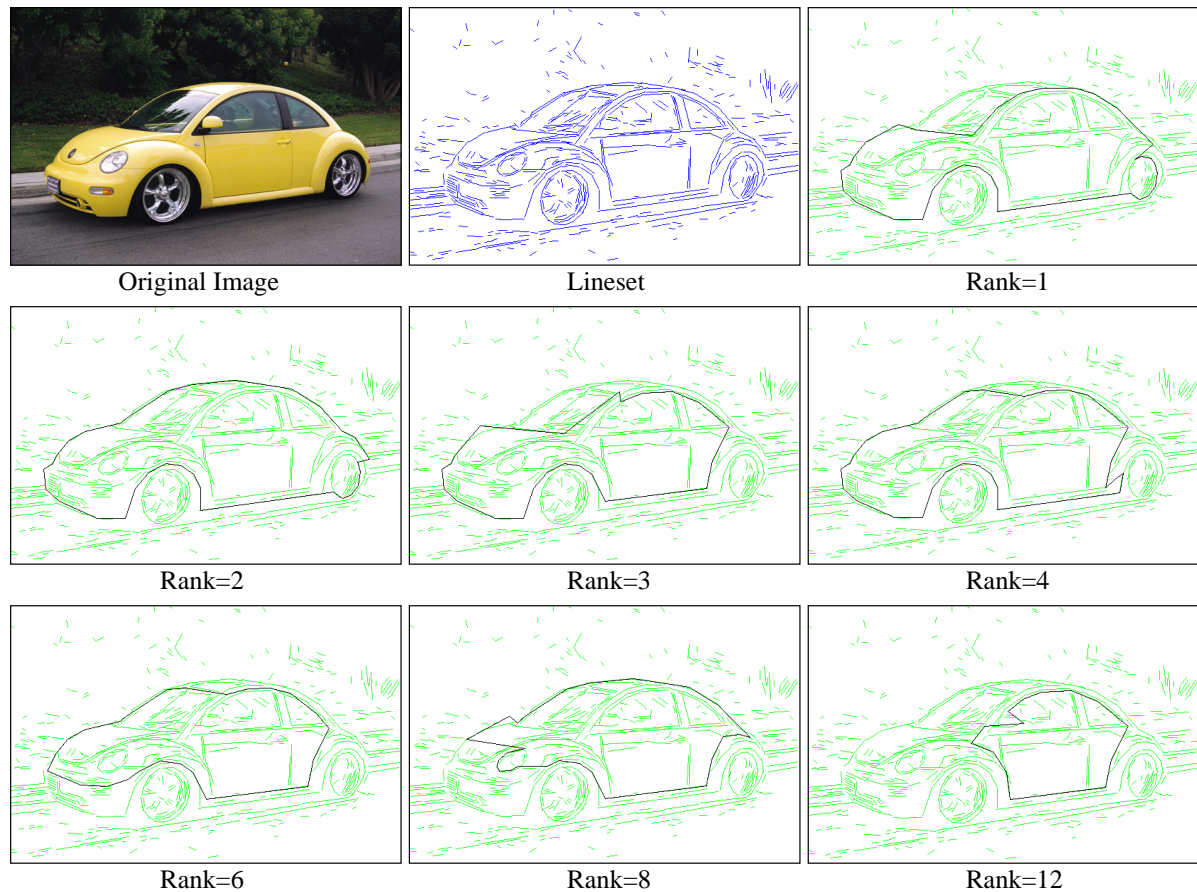


Figure 8.14: Original beetle image, line-set with 591 segments, and several of the extracted contours (ranking based on the colour saliency metric described in the text). The algorithm explored 2.9×10^5 nodes, found 41 distinct contours, and took 47 seconds to complete.

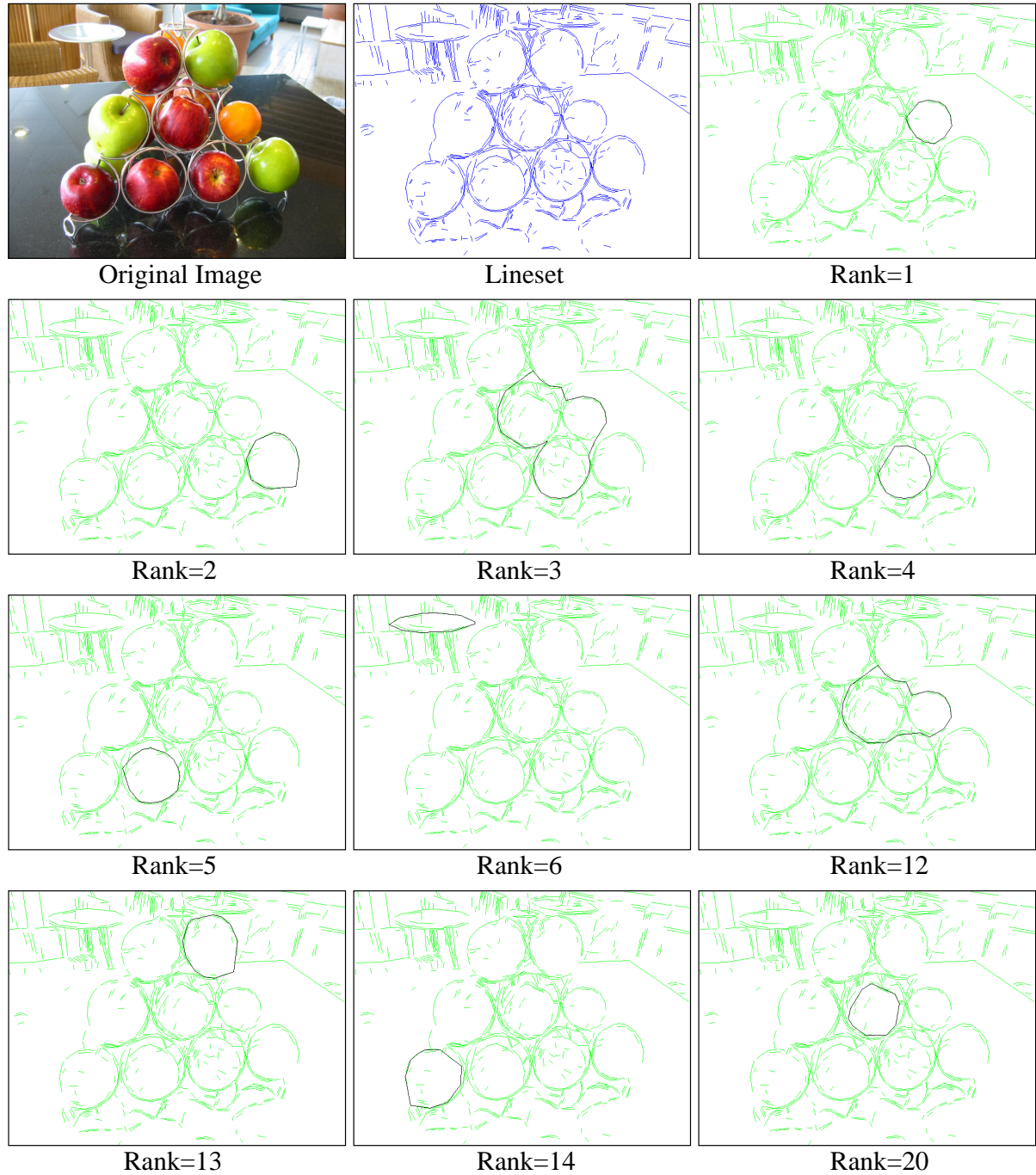


Figure 8.15: Original fruits image, line-set with 797 segments, and several of the extracted contours. The algorithm explored 4.9×10^5 nodes, found 53 distinct contours, and took 78 seconds to complete. Image courtesy of Mr. Peter N. Lewis.

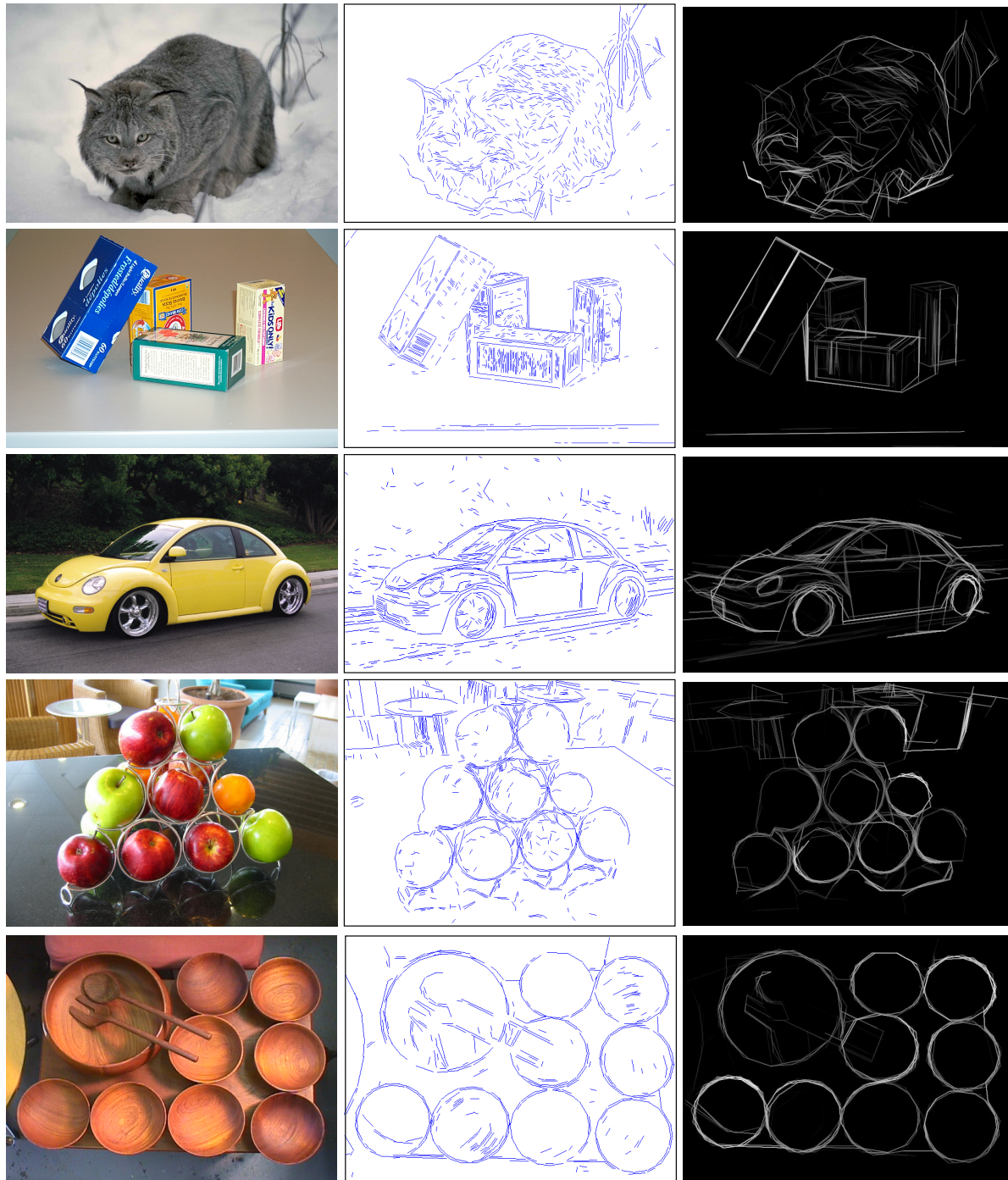


Figure 8.16: Traces of the search paths followed by the algorithm. From left to right: original images, input line-set, and traces of the search paths explored by the algorithm on the corresponding line-set. Brighter contours correspond to partial paths that were explored more often.

framework can be extended to deal with non-convex contours without a significant increase in search complexity, making it possible to find such contours efficiently even in the presence of texture and clutter. Finally, we demonstrated that the framework can be easily extended to incorporate other image cues, which can be simple local measurements, or more complex cues based on non-local properties of partial contours. Along the way, we explored the closely related issue of determining the saliency of a contour, and proposed two techniques for ranking the polygons output by the algorithm. One of these is based on the Qualitative Probabilities framework [48], and the second is based on accumulated colour statistics of the inside and outside of a particular contour.

The proposed framework was successful in detecting salient contours on a variety of line-sets showing significant variation in size, complexity, and amount of clutter and texture. From our experimental results, we conclude that the search framework is robust, and accomplishes the goal of performing contour extraction efficiently. It should be noted that while all the experimental results shown in the previous two chapters were produced with a standard set of parameters, the algorithm may be tuned to extract contours that are better suited to particular tasks or image domains. For example, the influence of each of the components of the affinity measure (length of gaps and tails, smooth continuation term, and colour affinity) can be tuned to obtain contours with particular characteristics. Additionally, the ranking stage can be easily extended to give higher weight to the desired contour characteristics, or to include domain specific information.

This concludes the part of this thesis that explores the problem of closed contour extraction, the main contributions of our work in perceptual grouping are:

- A general contour extraction procedure that is efficient and robust in the presence of clutter and texture.
- A search control technique based on normalized, pairwise affinities between lines.
- Showed that the framework can easily be extended to incorporate different image cues.

- Showed that the search can be biased using non-local cues, or accumulated statistics that depend on partial contours.
- Demonstrated the use of Qualitative Probabilities for contour saliency estimation.
- Proposed an alternative saliency measure based on the colour distributions on the inside and outside of a contour.

In the final chapter of this thesis, we will briefly describe opportunities for future work, including possible means to incorporate the information provided by the segmentation and grouping algorithms to achieve better figure-ground segmentation. We will then conclude this work with a summary of our contributions to the fields of image segmentation and perceptual grouping.