Anytimeness Avoids Parameters in Detecting Closed Convex Polygons

Michael Zillich Markus Vincze
Institute of Automation and Control
Vienna University of Technology
{zillich, vincze}@acin.tuwien.ac.at

Abstract

Many perceptual grouping algorithms depend on parameters one way or another. It is always difficult to set these parameters appropriately for a wide range of input images, and parameters tend to be tuned to a small set of test cases. Especially certain thresholds often seem unavoidable to limit search spaces in order to obtain reasonable runtime complexity. Furthermore early pruning of less salient hypotheses is often necessary to avoid exponential growth of the number of hypotheses. In the presented work we show how the adoption of a simple anytime algorithm, i. e. an algorithm which returns the best answer possible when interrupted and may improve on the answer if allowed to run longer, for finding closed convex polygons eliminates the need for parameter tuning. Furthermore it quite naturally allows the incorporation of attentional mechanisms into the grouping process.

1. Introduction

Detection of closed convex contours is a well-studied problem in computer vision and perceptual organisation and there is a vast amount of literature on the topic.

Huttenlocher and Wayner [5] present a system for finding convex line groupings which avoids thresholds in defining neighbourhood relations. They perform a Delaunay triangulation of the line image in time $O(n \log n)$, where n is the number of lines. Visible lines form part of the triangulation and lines added by the triangulation define scale-invariant endpoint neighbourhoods, which are weighted with costs depending on closeness and straightness. Convexity is enforced by assigning an infinite value for counter-clockwise turning angles. The local convexity graph constructed from visible lines and least cost neighbourhoods is planar and allows search for all convex polygonal chains in time $O(n)$. However selecting only the least cost neighbour constitutes an early, purely local pruning of alternative hypotheses. I.e. clutter can easily "steal" the neighbourhood from the actual (desired) neighbour, that would eventually lead to a globally better grouping.

Jacobs [6] presents a system for robust and efficient detection of convex groups of lines. It can deal with moderate amounts of clutter and occlusion. It depends however on appropriate settings of various thresholds in order to cope with the $O(n^2 \log n + mn)$ runtime complexity, where n is the number of lines and m the number of groups to detect. Early work on perceptual grouping by Lowe [9] already addressed the problem of quadratic run-time complexity (in the number of features) using a grid overlaid on the image indexed by line endpoints. A typical problem of indexing is the appropriate choice of bin size. Sarkar and Boyer [12] used further curve parameters to construct index spaces of higher-parametric models and also addressed the problem of bin size and indices close to bin boundaries.

Ackermann et al. [1] use areas of perceptual attentiveness in the earlier stages of their MRF-based grouping approach, which they learn from hand-labeled training sets. These areas however still serve as hard thresholds, no grouping candidates outside an attentive area are considered, which still leaves the system brittle. A small increase in the gap size between two line endpoints will prevent the formation of a grouping hypothesis. Guy and Medioni [4] avoid thresholds and use an infinitely large extension field. Edgels and segment endpoints vote with directional vectors weighted by the length of segments and decreasing with distance. Runtime complexity however is $O(k^2)$, where k is the number of edgels.

The system of Saund [14] finds mostly convex closed paths in sketches and line drawings. Bidirectional best-first search with backtracking uses local preferences based on turning angles as well as global figural saliency measures such as compactness and closedness. Preferences are given to either maximally turning or maximally smooth paths, reflecting the types of closed contours typically occurring in sketches. Several parameters and thresholds have to be set manually. The system is able to deal with even complex sketches but is not designed for the fuzzyness and clutter of real world images.

Zhu et al. [17] present a method for segmenting salient

closed as well as open 1D contours from the otherwise 2D image clutter. Any 1D topological structure can be put into a specific ordering, whereas such an ordering is impossible for a 2D structure, where violations of that ordering lead to entanglements. Finding the segmentation with an optimal ordering of edgels (i. e. with minimal entanglement) is a hard combinatorial problem and the authors use circular embedding (mapping the edge graph to a circle) to arrive at a tractable solution which is based on finding the top complex eigenvectors of the random walk matrix. Good results are shown on several benchmarks.

In [16] Wang et al. extend their previous work on grouping edges into smooth salient curves (RC - ratio contour algorithm) with a pruning step which removes any non-convex edges and edge links to enforce convexity (CRC - convex ratio contour). Edge links are represented as Bezier curves, which is well suited for detecting smooth boundaries. They find globally optimal convex contours in polynomial time, in the worst case roughly $O(n^3)$ with n edges, though the convexity constraint tends to hold n small.

Estrada and Jepson [3] present an algorithm for grouping line segments into salient closed contours. A measure of affinity between pairs of lines is based on the quality of intersections and the uncertainty of endpoints and guides a depth-first search. For convex contours this search is very effective. For non-convex contours compactness, defined as the ratio between the area of a contour and its convex hull, and an additional smoothness term limit the combinatorial explosion of search paths. Detected contours are ranked using the Qualitative Probabilities framework of Jepson and Mann [8] to prefer contours with a high number of segments along their boundary, which split few lines and are confirmed by edges terminating at the boundary. The algorithm can deal successfully with cluttered and textured regions. However many parameters and thresholds have to be selected ad-hoc or determined experimentally.

Sarkar and Soundararajan [13] address the problem of parameter settings by proposing a supervised reinforcement learning framework. Figure-ground segmentation from edge images is formulated as spectral graph partitioning of a scene structure graph. The relative importance of relations (such as parallelity, photographic similarity) as well as edge segmentation and spectral partitioning parameters are learned by an N-player learning automata framework.

Even if runtime complexity is addressed successfully, most systems process the input image in one pass and limit the number of generated hypotheses by setting some saliency threshold.

An early system by Sha'ashua and Ullman [15] finds image curves that will optimise a cost function based on total curvature and number and size of gaps in a curve using an iterative dynamic programming approach in a network of kn^2 nodes, where k is the number of edge orientations and $n \times n$ the image size. Their approach has a nice anytime quality: processing can be stopped after any number N of steps (having performed $O(Nkn^2)$ computations) to find the smoothest curves of length N. Each pixel at each orientation then is assigned the saliency value of the length N curve starting from it. The system does not require parameters apart from the choice of discretisation k of angles. Although the optimisation is global (for the curve) decisions are rather local. In order to find e.g. the most salient curve in the image, one starts with the most salient node and follows along its most salient neighbour and so on. The system does not handle multiple hypotheses in cases where it might not be clear which neighbour to follow, but always (locally) chooses the best. And, depending on the structures in the scene, it is not always the smoothest edges which corresponds to an object contour.

Jacot-Descombes and Pun [7] order contour primitives based on saliency prior to grouping and then sequentially process primitives as they come in. They achieve runtime linear in the number of image contours. Acceptance of junctions is based on thresholds however, introducing again the danger of brittleness. Casadei and Mitter [2] present an elaborate system for contour estimation and introduce emission points at curvature maxima and endpoints. Search lines for finding good contour continuations are "emitted" from these points. Their system however too depends on proper threshold settings. Zillich and Matas [18] form convex groups of circular arcs rather than lines using a simple indexing scheme in the image space, where tangential search lines emanating from arc endpoints are drawn in the image. They achieve runtime linear in the number of arcs. The length of search lines is however fixed, again introducing problematic thresholds.

Mahamud et al. [10] also present an anytime system for segmenting salient smooth closed contours. Contrary to most other approaches their saliency measure explicitly takes into account the global property of closure of contours. Proximity and good continuation are modelled as a distribution of smooth curves traced by particles emanating from edge endpoints and moving with constant speed undergoing Brownian motion. The transition probability between edge i and j is the sum of the probabilities of all paths that a particle can take between the two edges, and is denoted as P_{ij}. The saliency measure then is based on the largest positive real eigenvalue of the transition matrix P. The search for closed contours finally corresponds to finding strongly connected components in the edge graph. Contours are segmented successively in order of decreasing saliency. The more salient a contour and the smaller the number of edges it contains, the faster it can be segmented and segmentation can be stopped after any number of detected contours. The elegance of this global saliency measure however comes at a cost, as it requires finding the

largest positive real eigenvector of a $2N \times 2N$ matrix, with N the number of edges. The authors propose a special technique exploiting the sparseness and symmetry properties of the transition matrix, which significantly speeds up computation. The whole approach however is still computationally intensive and generally does not seem to scale well to bigger problems. Furthermore the method is specifically designed for smooth contours (such as fruits, stones, coins) and does not work for contours containing sharp turns.

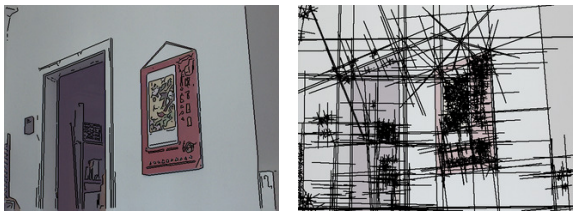
In this paper we address the issues of parameter tuning and runtime complexity in detecting closed convex contours from a given set of line segments. We formulate the problem as first connecting neighbouring lines to form a graph $G = (V, E)$, with lines as vertices (nodes) V and junctions between lines as edges E and then finding cycles in G, while making sure these cycles constitute convex polygons. We detect junctions in runtime linear in the number of lines. We avoid tuning of parameters, especially nearness thresholds for junctions, by using an anytime algorithm, which also allows incorporation of attentional mechanisms.

The remainder of this paper is organised as follows. In section 2 we review the image based indexing scheme for junction detection of [18] and look at its runtime behaviour. We then extend this scheme to incrementally build the graph G in section 3. We explain detection of closed convex contours as cycles of G in section 4. Finally we look at the incorporation of attentional mechanisms in section 5 and conclude with section 6.

2. Image Space Indexing

Checking all pairs of n lines for possible junctions has a runtime complexity of $O(n^2)$. This quadratic runtime complexity quickly becomes prohibitive for a larger number of lines. Looking at the lines extracted from a typical indoor image such as Figure 1(a) (we use Canny edge detection with self-adjusting hysteresis thresholds followed by non-parametric recursive segmentation into lines using an algorithm by Rosin and West [11]) it becomes evident that there is of course no need to check all pairs of lines for possible junctions, but only those being "close" to each other. A well known technique to overcome combinatorial explosions in perceptual grouping is indexing. An indexing space is constructed such that primitives sharing the grouping relationship (e. g. closeness) index into the same bin. Thus one only has to perform the indexing operation n times to find all good candidates for pairs. As indexing is only a heuristic, all primitives of one bin are then further analysed whether they actually fulfill the given relation. If the indexing space is well constructed the number of incorrect matches is low and the runtime complexity is reduced to $O(n)$. As in [18] we use the image itself as index space, i. e. single pixels serve as bins.

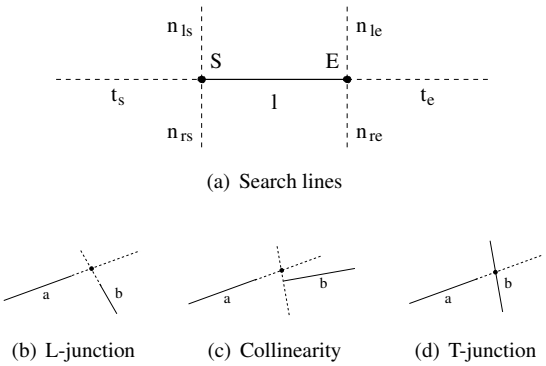
Each line endpoint defines a set of *search lines* consisting of one *tangential* and two *normal* search lines. Together with the line l itself this results in a total of seven search lines: start and end tangent (t_s, t_e), left and right normal, again for start and end $(n_{ls}, n_{rs}, n_{le}, n_{re})$ (see Figure 2(a)). These search lines are drawn into the index image using the Bresenham line drawing algorithm, with a slight modification to draw "dense" lines in order not to miss any intersections. This constitutes the indexing step. Figure 1(b) shows the search lines drawn for the office wall scene, where search line length was chosen to be $s = l$, i. e. equal to the length of the originating lines.



(a) Extracted lines (b) Search lines

Figure 1. Office wall scene.

Whenever two lines a and b index into the same bin, i. e. their search lines intersect, we create a new junction between lines a and b. Depending on the types of search lines intersecting we form an L-junction (Figure 2(b)), a collinearity (Figure 2(c)) or a T-junction (Figure 2(d)) between the respective originating lines. If more than two lines intersect, the according number of pairwise junctions are created.

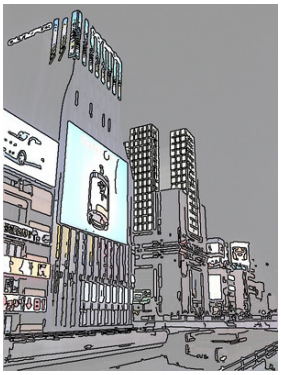


(a) Search lines

(b) L-junction (c) Collinearity (d) T-junction

Figure 2. Search lines and junctions.

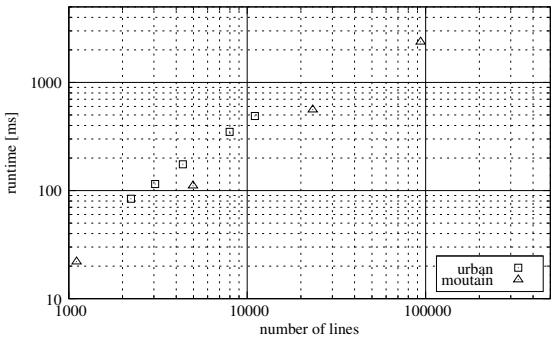
Runtime Complexity Figures 3(a) and 3(b) show an urban scene with rather long, straight lines and a landscape scene with many very short lines. Figure 4 shows results for detection of junctions for these two scenes. Search line length was chosen to be $s = l$, i. e. equal to the length of the originating lines. Detection was run on different resolutions



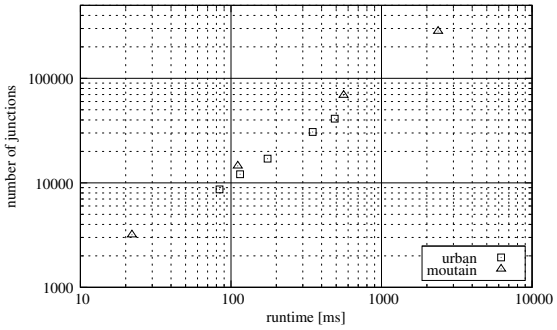
(a) Urban scene

(b) Landscape scene

Figure 3. Two scenes with overlaid edges.



(a) Runtime



(b) Created junctions

Figure 4. Runtime complexity for fixed search line length.

of these images ranging from 384×256 to 3072×2048 pixels on a 2.16 MHz PC. The times shown are runtimes in milliseconds for junction detection only, i. e. excluding time for edge and line segmentation. The number of lines grows from a few hundred in the low resolution images up to around 100000 for the highest resolution. The linearity of runtime versus number of lines is plainly visible in the double-logarithmic plot of Figure 4(a) as a slope of one per decade. It reflects the linear growth of the number of junctions over time (Figure 4(b)).

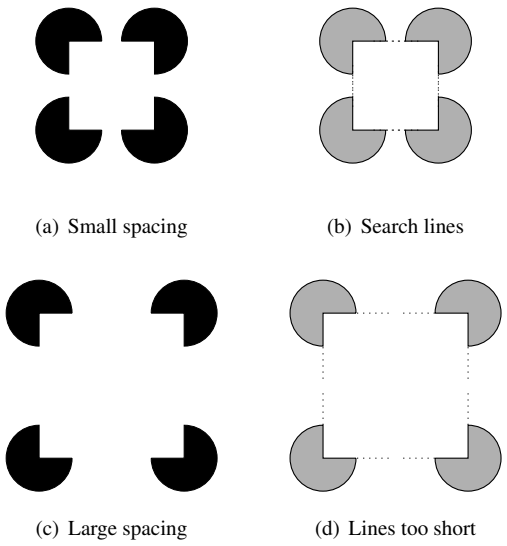
3. Incremental Processing and Anytimeness

In the above examples we used a search line length equal to the length l of the respective originating line. As we remarked in the introduction such hard thresholds make a system brittle and lead to problems even in typical, simple viewing conditions. Figure 5(a) shows the well known Kanizsa square, an often cited example of amodal completion. The human observer can easily perceive a white square that seems to float above four black disks. If we move the disks further apart (Figure 5(c)) we can still perceive a white square, albeit slightly less prominent. Using a fixed search line length of l in Figure 5(b) enables us to find collinearities and thus eventually the completed closed contour, i. e. the square. If we continually increase the spacing of the disks we realise that at some point suddenly the search lines are too short (see Figure 5(d)). Suddenly, because the increase of one pixel in spacing is enough to let the pairwise grouping and subsequent search for closures fail.

Therefore we do not draw the search lines to a fixed length, find all junctions and then perform search for cycles on the resulting graph G. Instead we incrementally grow search lines, continuously checking for junctions, thus gradually building G, and perform cycle searches as the graph grows. We therefore interleave creation of junctions and cycle search.

What we have achieved now is to get rid of the troublesome parameter *search line length*. Actually we have replaced it with another parameter: *run time*. If indexing is only given a limited amount of time, e. g. some fixed frame rate, it will only find the closest junctions. The longer we let the process of image space indexing operate, the more junctions and thus closed convex contours we will find.

Generally small gaps will be traversed quickly. So the most prominent groupings will pop out early, even in cluttered scenes. Bridging large gaps (e. g. due to partial occlusion) in cluttered scenes is obviously more difficult amongst all the clutter, which uses up processing time as the system



(a) Small spacing (b) Search lines

(c) Large spacing (d) Lines too short

Figure 5. Kanizsa square problem.

tries in vain to hallucinate groupings into the clutter. Eventually however even the largest gap will be bridged. It may take longer, but it will not be missed. *Incremental* image space indexing gives us the possibility to evaluate the best possible hypotheses given a limited amount of time, or put the other way round, allows us to interrupt processing at *any time* while making sure that the best hypotheses have been evaluated.

Directing Search Not all lines are equally salient. Long lines are more likely (though not guaranteed!) to be part of an interesting scene structure, while shorter lines tend to be clutter. We want to concentrate processing on salient lines. So we first rank lines according to length and then randomly select lines for growth using an exponential distribution over the rank, thus ensuring that longer lines get selected more often for growth. Furthermore we prefer tangential search lines over normal ones as we typically expect good grouping candidates rather in line direction. So we introduce another random selection step, giving higher probability to tangential search lines. (We can think of search lines as a degenerate and dynamic version of extension fields [4]. Preferring tangential search lines amounts to defining the shape of these fields.) Once a line has formed a junction at one end, we deweight that end to direct processing to open ends. So for each line we prefer that end with fewer formed junctions. Probabilities of selecting the front or back end are inversely proportional to the number of junctions at the respective end: $Pr(front)/Pr(back) = (1 + n_{back})/(1 + n_{front})$, where n_{front} and n_{back} are the total numbers of collinearities and L-junctions at the front and back end respectively. (The constant 1 refers to the open line end itself and avoids divisions by zero.) After randomly selecting an end we decide further if the pixel to be drawn should be awarded the line itself or, if there are junctions at this end, to one if its neighbours. Each neighbour and the line itself are chosen with equal probability. If the line itself is chosen, one of its search lines at this end is extended (see above). Otherwise the pixel "moves" to the selected neighbouring line and the whole procedure is repeated. I. e. we again decide randomly whether the pixel should be awarded this new line or move along further junctions. This scheme of distributing voting pixels ensures that search for further grouping candidates is concentrated at the open ends of partially formed contours.

4. Finding Convex Groups of Lines

Creating junctions between lines builds up the graph G. We now want to find cycles in this graph, which correspond to closed convex polygons. This can be expressed as finding another path between two nodes which are already connected. Note that this formulation is well suited to the anytimeness of our approach. Edges appear one at a time and each new edge is possibly the missing edge of a cycle. So given a new edge we perform shortest path search between its nodes (excluding obviously the newly created edge). We use Dijkstras algorithm for finding shortest paths with modifications to enforce convexity.

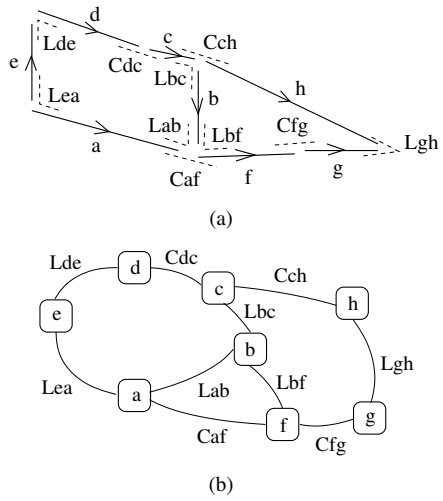


Figure 6. Lines and junctions (a) and the corresponding graph (b).

Graph Construction Figure 6(a) shows a simple example of several lines (the arrows indicating direction and thus start and end point) and junctions and Figure 6(b) shows the corresponding graph constructed from them. Nodes are formed by lines, edges are formed by L-junctions and

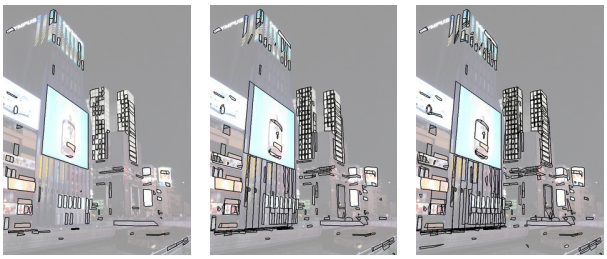
collinearities. T-junctions do not take part in this closure search, because a closure can not have a T-junction with itself. Similar to Huttenlocher and Wayner [5], edges leaving a node are distinguished into several groups. They can be formed by an L-junction (e.g. Lab) or a collinearity (Caf). Furthermore L-junctions can bend to the LEFT or RIGHT, as seen from the end of a line (Lab bends to the LEFT, Lea bends to the RIGHT as seen from line a). Finally junctions connect the START or END of a line to another line (Lea and Lab respectively). This results in $2 \times 3 = 6$ groups of edges per node: START or END and one of LEFT, COLL, RIGHT.

Graph Search In order to find a closed path we pick two lines which are joined at one end and search the shortest path between the two remaining ends. Remember that the graph is built incrementally. Initially we just have n nodes and 0 edges. As search lines grow and junctions between lines are created, we gradually add new edges to the graph. Whenever a new L-junction Lst or collinearity Cst is created, a new edge $s-t$ is added. This new edge could lead to the closure of a path, so we perform a shortest path search between nodes s and t. We use Dijkstras algorithm, where edge costs represent the saliency of junctions and are defined in a scale-invariant manner similar to Lowe [9]. To avoid unnecessary searches, we do not start a search if a junction is isolated, i. e. one of its lines has an unconnected end. During search we only follow edges which maintain convexity of the path so far. To this end we maintain the current *bend* of the path. The first L-junction encountered on the path sets the bend to LEFT or RIGHT. Subsequent extensions of the path must be compatible with the current bend. We also enforce that a path entering via a START edge leaves via an END edge and vice versa. Finally we check whether a closed path would lead to a self-intersecting polygon, in which case we reject it.

So for each new edge as seed we perform a search returning the shortest, i. e. lowest cost cycle. In Figure 6(b) starting with Lea as seed we might find the cycle $abcde$, which happens to be a minimal cycle or the cycle $afghcde$, depending on whether e. g. edges Lab and/or Caf were already created at that time and if so, depending on the total costs of paths $abcde$ vs. $afghcde$. Or we might find $abcde$ first with Lab as seed and later $afghcde$ with seed Caf. It depends on the order in which graph edges are created, which is related to the saliency of the corresponding junction (smaller gaps are traversed quicker).

Note that we do not find only strictly convex polygons. Collinearities detected in the previous section are never perfectly straight. So a collinearity will sometimes actually bend somewhat "inwards" as seen from the current convex path (such as Cdc in Figure 6(a)). Many "obviously" convex surfaces, such as book covers, often appear slightly concave (e.g. a paper-back with a slightly bent cover). We would not want to rule these out. This tolerance in the notion of convexity is well suited for real world situations.

Runtime Complexity Search of one shortest path on a graph with n nodes and m edges using Dijkstras algorithm has a runtime complexity of $O((m+n)\log n)$. This is worse than the $O(n)$ complexity for finding all minimal cycles in the planar graph of Huttenlocher and Wayner [5] or Jacot-Descombes and Pun [7]. The difference between their approach and ours is that they perform search once the graph is completed after defining all nodes and edges, while we build the graph incrementally and search for closures as we build the graph. This helps to reduce the runtime of shortest path search. We start with a completely unconnected graph ($m = 0$) and gradually increase the number of edges. Path searches for earlier groupings are therefore very fast and only as the graph gets more densely connected the $m \log n$ complexity starts to dominate. Note however that the constraint on path convexity quickly limits search depth and actual runtimes typically stay well below the theoretical value.



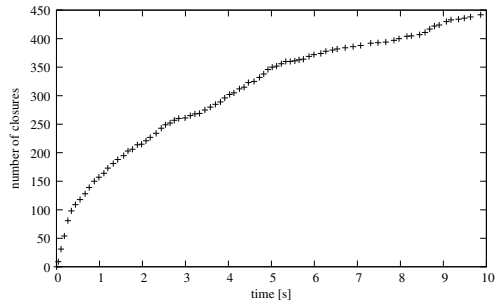
(a) Closures after 1 s (b) ... 5 s (c) ... 10 s

Figure 7. Gradually detecting closures.

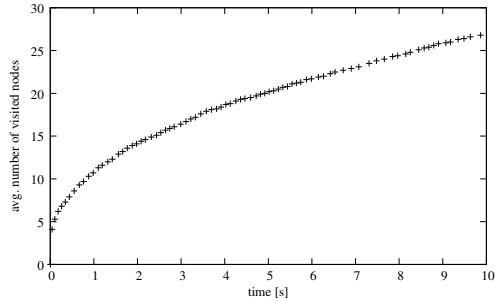
Figures 7 and 8 show the runtime behaviour of closure search for the urban scene of Figure 4. Figure 8(a) shows the number of detected closures over time. We see that the rate of finding new closures drops as time progresses. This is a consequence of the increasing average number of nodes visited during a search (Figure 8(b)), and therefore the increasing path lengths. These in turn mean more expensive checks for non-selfintersecting paths, which are of $O(k^2)$, with k the number of nodes in the path.

5. Including Attention

Up to now we based concentration of processing on line saliency and some general considerations. Sometimes however we might have external clues. The user of our system might provide us with regions of interest (ROIs), perhaps deduced from other vision modules, e. g. change detection. Also higher level knowledge might be available such as the gaze direction of a human or the position of a grasping hand.



(a) Created closures

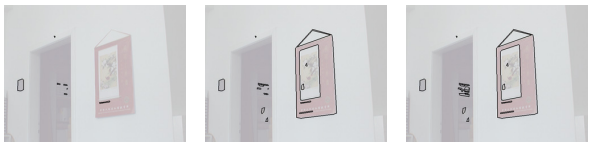


(b) Average number of visited nodes

Figure 8. Runtime behaviour for closure search.

Regions of Interest It is quite straight-forward to include such regions of interest into the above selection scheme. Rather than ranking lines according to their length, we rank them according to their nearness to the provided region(s) of interest.

(a) 15 ms (b) 50 ms (c) 80 ms



(d) 15 ms (e) 50 ms (f) 80 ms

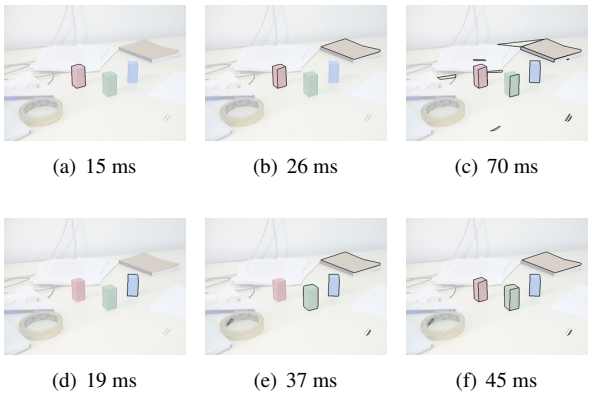
Figure 9. Normal grouping (top) and grouping biased towards door as region of interest (bottom).

Figure 9 shows in the top row three images with detected closures in the office wall scene after 15, 50 and 80 ms, using the above incremental indexing scheme. We can see that the big red poster to the right of the door is quickly picked up and smaller closed contours coming in later. In the bottom row we use a ROI in the form of a Gaussian PDF. Its center is placed inside the open doorway and the one-σ area roughly covers the doorway. We then calculate the weight of a line as the sum of its weighted pixels and rank the lines accordingly. We can see that early on we start picking up the small boxes on the shelf in the next room, then larger contours further away. After some time both methods will have found the same contours. The ROI allows us to specify where to look first and more "carefully". Note that we do not *exclude* other regions of the image, as would be the case if we simply cut out the ROI and then work on the sub-image. If there is some really prominent structure outside the ROI (such as the big red poster) we certainly still want to be able to detect it. Cutting out sub-images would basically constitute yet another thresholding and slightly mis-placing a cut-out ROI might just threshold away the interesting bits, requiring cumbersome tuning of ROI sizes.

Colours of Interest In the scene depicted in Figure 10 we might be asked to "find the red block". Performing colour segmentation would be the equivalent to a cut-out ROI in colour space, with similar problems. Getting the colour model slightly wrong (e. g. due to unknown lighting conditions) will affect the segmentation and lead to uncorrectable errors. And again it is not necessarily *only* the red block which we would like our system to perceive.

Instead we use a very simple colour model: "red" is equivalent to RGB value $(255, 0, 0)$. We then weight each line with the inverse of its Euclidean distance to this colour and again use these weights for ranking. (Note that there are certainly more sophisticated colour spaces and distance measures, but that is not our concern here.) The top row of Figure 10 shows the results of detecting closed contours for consecutive time steps. The bright red block (leftmost) is detected first, followed by the brownish-red book in the background to the right and then the other blocks, which also constitute prominent contours. Compare that to the bottom row of Figure 10. Here we set the colour of interest to blue $(0, 0, 255)$. This time the blue block (rightmost) is found first, followed again by the other prominent structures.

Influencing the ranking of lines, and thus the amount of processing spent on each line, by various weights is a flexible way to provide the system with a notion of attention. Further sources of attention are conceivable. Certainly motion is known to provide attention for many sighted animals, and we could use the magnitude of an optical flow field as weight. And provided with a rough notion of depth we could use nearness, therefore concentrating on the foreground.



(a) 15 ms (b) 26 ms (c) 70 ms

(d) 19 ms (e) 37 ms (f) 45 ms

Figure 10. Looking for red (top) or blue (bottom).

6. Conclusion

We presented a simple extension (incremental processing) to a simple idea (image space indexing) to address issues in the detection of closed convex contours. Generally there are two ways to limit combinatorial explosion inherent in such grouping tasks: indexing and early thresholding of hypotheses. We showed how the adoption of an incremental indexing scheme removes the need for thresholds. We do not rely on any threshold or tuning parameter in the above method, but just rank grouping primitives according to saliency or external weights based on attention, and use these ranks to control incremental processing. Note that we actually do not *avoid* combinatorial explosion but rather *control* it, always being able to interrupt the process if things get too complicated. After interruption we have as a result the best that was achievable up to that point. Note also that we do not prune less salient hypotheses. The longer processing runs the more awkward hypotheses will be "hallucinated". We therefore conclude with a famous quote by Max Clowes that *vision is controlled hallucination*.

References

[1] F. Ackermann, A. Maßmann, S. Posch, G. Sagerer, and D. Schlüter. Perceptual grouping of contour segments using markov random fields. *IEEE Transactions on Pattern Recognition and Image Analysis*, 7(1):11–17, 1997.

[2] S. Casadei and S. Mitter. A perceptual organization approach to contour estimation via composition, compression and pruning of contour hypotheses. Technical Report LIDS-P2415, Lab. for Information and Decision Systems, MIT, April 1998.

[3] F. J. Estrada and A. D. Jepson. Perceptual Grouping for Contour Extraction. In *International Conference on Pattern Recognition*, pages 32–35, 2004.

[4] G. Guy and G. Medioni. Inferring global perceptual contours from local features. *International Journal of Computer Vision, Special issue on computer vision research at the University of Southern California*, 20(1-2):113–133, 1996.

[5] D. Huttenlocher and P. Wayner. Finding convex edge groupings in an image. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 406–412, June 1991.

[6] D. W. Jacobs. Robust and efficient detection of convex groups. *IEEE Transactions on Pattern Analysis and Machine Vision*, 18(1):23–37, 1996.

[7] A. Jacot-Descombes and T. Pun. Asynchronous Perceptual Grouping: From Contours to Relevant 2-D Structures. *Computer Vision and Image Understanding*, 66(1):1–24, Apr. 1997.

[8] A. D. Jepson and R. Mann. Qualitative Probabilities for Image Interpretation. In *IEEE International Conference on Computer Vision*, pages 1123–1130, 1999.

[9] D. G. Lowe. Three-dimensional object recognition from single two-dimensional images. *Artificial Intelligence*, 31(3):355–395, 1987.

[10] S. Mahamud, L. R. Williams, K. K. Thornber, and K. Xu. Segmentation of Multiple Salient Closed Contours from Real Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(4), Apr. 2003.

[11] P. L. Rosin and G. West. Non-parametric segmentation of curves into variuos representations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(12):1140–1153, 1995.

[12] S. Sarkar and K. L. Boyer. A computational structure for preattentive perceptual organization: Graphical enumeration and voting methods. *IEEE Transactions on System, Man and Cybernetics*, 24(2):246–266, February 1994.

[13] S. Sarkar and P. Soundararajan. Supervised Learning of Large Perceptual Organization: Graph Spectral Partitioning and Learning Automata. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(5):504–525, May 2000.

[14] E. Saund. Finding Perceptually Closed Paths in Sketches and Drawings. *IEEE Transactions on Pattern Analysis and Machine Vision*, 25(4):475–491, Apr. 2003.

[15] A. Sha'ashua and S. Ullman. Structural Saliency: The Detection of Globally Salient Structures Using a Locally Connected Network. In *IEEE International Conference on Computer Vision*, pages 321–327, 1988.

[16] S. Wang, J. S. Stahl, A. Bailey, and M. Dropps. Global detection of salient convex boundaries. *International Journal of Computer Vision*, 71(3):337–359, 2007.

[17] Q. Zhu, G. Song, and J. Shi. Untangling Cycles for Contour Grouping. In *IEEE International Conference on Computer Vision (ICCV)*, 2007.

[18] M. Zillich and J. Matas. Ellipse Detection Using Efficient Grouping of Arc Segments. In *Proceedings of the 27th Workshop of the Austrian Association for Pattern Recognition (OAGM/AAPR), Laxenburg*, pages 143–148, 2003.