Combinatorial Shape Decomposition

Ralf Juengling, Melanie Mitchell

Department of Computer Science
P.O. Box 751
Portland State University
Portland, Oregon 97207-0751

To appear in *Proceedings of the Third International Symposium on Visual Computing (ISVC07).* Springer (Lecture Notes in Computer Science).

Abstract. We formulate decomposition of two-dimensional shapes as a combinatorial optimization problem and present a dynamic programming algorithm that solves it.

1 Introduction

Identifying a shape's components can be essential for object recognition, object completion, and shape matching, among other computer vision tasks [1]. In this paper we present a novel shape-decomposition algorithm, aimed at capturing some of the heuristics used by humans when parsing shapes.

In 1984, Hoffman and Richards [2] proposed the *minima rule*, a simple heuristic for making straight-line cuts that decompose a given shape (or silhouette): Given a silhouette such as the one in Fig. 1(a), the end-points of cuts should be negative minima of curvature of its bounding contour (Fig. 1(b)). Note that this rule does not specify which pairs of these points should be connected to make cuts. Fig. 1(c) gives one possible set of cuts connecting negative minima.

Later, Singh, Syranian, and Hoffman [3] proposed an additional simple heuristic, supported by results of psychophysics experiments on human subjects, called the *short-cut rule*: If there are several competing cuts, select the one with shortest length. For example, in Fig. 1(c), most people would prefer the cuts shown as compared with a cut between the two topmost black dots, which would be significantly longer.

Singh et al. make clear that the minima and short-cut rules are not the only necessary heuristics for shape decomposition; other possible heuristics could involve local symmetries or good continuation, or rely on prior knowledge about the shape's category. However, the two simple heuristics seem to explain many of the experimental results on people.

In this paper we propose an efficient algorithm for shape decomposition that results in these two heuristics being approximately satisfied, without having to compute boundary curvature. Given a polygonal description of a silhouette, our algorithm computes the constrained Delaunay triangulation of the shape, and chooses among the interior edges of this triangulation an optimum set of cuts by solving a corresponding combinatorial optimization problem.

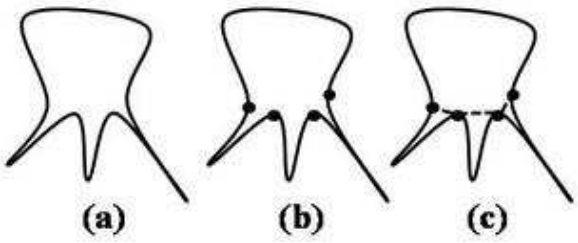

Fig. 1. (a) Silhouette. (b) Black dots mark points at which curvature has a negative minimum. (c) Three possible cuts based on these points.

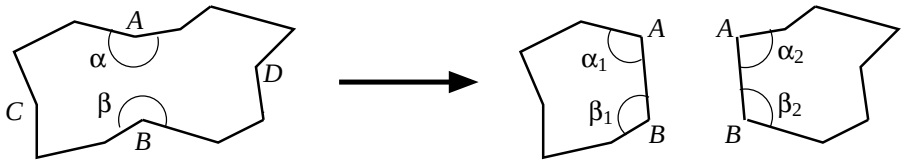

Fig. 2. Making a cut means breaking a polygon into two polygons. Here the cut is made between two concave vertices A and B. Because $\alpha = \alpha_1 + \alpha_2$ and $\alpha < 2\pi$ the number of concave vertices never increases through a cut.

2 Finding Cuts by Combinatorial Optimization

We assume a polygonal description of the shape and require that a cut (1) connects two polygon vertices, and (2) does not cross a polygon edge. It follows that there is only a finite number of possible cuts for any given polygon. Our strategy is to define an objective function over the set of possible cuts, $\mathcal{C}$, and to select the subset of cuts that minimizes the objective function. This is a combinatorial optimization problem with a number of possible solutions exponential in $|\mathcal{C}|$. We introduce a third constraint on the set of possible cuts in Section 2.1 to make the problem amenable to a solution by dynamic programming.

Curvature is not available in our polygonal framework and we need to adapt the minima rule. As in Latecki and Lakaemper [4], vertices with a concave angle play the role of boundary points of negative curvature (we measure the inside angle and call vertices with an angle greater than π *concave*; see Fig. 2).

As Fig. 2 illustrates, placing a cut amounts to breaking a polygon in two. Our objective function favors a decomposition into convex parts by penalizing concave vertices. It is is basically of the form $\sum_k f(\theta_k)$, where θ_k ranges over all angles of a given partition or cut set. For the example in Fig. 2 the difference of the objective function values between the empty cut set (left) and the set $\{AB\}$ (right) is therefore

$$f(\alpha) - f(\alpha_1) - f(\alpha_2) + f(\beta) - f(\beta_1) - f(\beta_1) \tag{1}$$

Thus f should be such that this sum is negative when the cut AB is considered desirable. We will resume discussing the objective function below in Section 2.2.

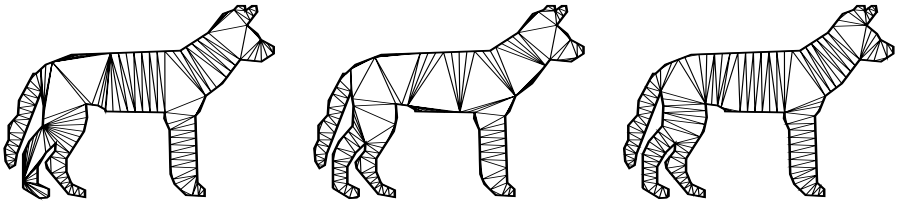

Fig. 3. Different triangulations of a dog-shaped polygon: A minmax length triangulation (left) minimizes the maximum edge length, the minimum weight triangulation (middle) minimizes total edge length, and the constrained Delaunay triangulation (right) minimizes the maximum triangle circumcircle.

2.1 The Set of Possible Cuts

If a set of cuts for the polygon in Fig. 2 includes the cut AB, then it cannot simultaneously include the cut CD because AB and CD cross. On the other hand, if it were understood that possible cuts never cross, then it is enough to know all other cuts ending in either A or B to decide whether AB should be included to improve a tentative cut set. This insight is the key to our dynamic programming optimization algorithm (Section 2.4). We therefore pose as a third requirement on $\mathcal{C}$, the set of possible cuts or *chords*, that no two elements in $\mathcal{C}$ cross.

This also means that we are excluding a number of possible cuts outright when choosing $\mathcal{C}$ for a given shape. For the shape in Fig. 2, for example, we have to decide between AB and CD, among others.

Any maximum set of chords obeying our third requirement corresponds to a *triangulation* of the shape polygon, a well-studied subject in computational geometry [5]. For $\mathcal{C}$ we need to choose a triangulation that contains most of the desired cuts. Since by the short-cut rule we prefer shorter cuts over longer ones, the *minmax edge length* or the *minimum weight* triangulation [6] ought to be good candidates (cf. Fig. 3). In addition we consider the constrained Delaunay triangulation (CDT, Fig. 3 right).

The CDT optimizes several criteria (e.g., it maximizes the minimum angle and minimizes the maximum triangle circumcircle). While it tends to yield short chords as well, it is in general not optimal with respect to length criteria [6]. However, we find that chords of the CDT match our intuitive notion of "possible cut" best. This has to do with the defining property of the CDT, that every circumcircle is an *empty circle* [5]: If a sequence of polygons converges to a silhouette then the empty circles of the respective CDTs converge to maximum inscribed circles of the silhouette, and hence, in the limit, the chords of the CDT connect boundary points in local symmetry [7]. This observation corresponds to a third rule stated by Singh et al. [3], that a cut ought to cross an axis of local symmetry.

i_{AB}	i_{AC}	$F_A(i_{AB}, i_{AC})$
0	0	$f(\alpha_1 + \alpha_2 + \alpha_3)$
0	1	$\frac{l_{AC}}{l_A} f(\alpha_1) + \frac{l_{AC}}{l_A} f(\alpha_2 + \alpha_3)$
1	0	$\frac{l_{AB}}{l_A} f(\alpha_1 + \alpha_2) + \frac{l_{AB}}{l_A} f(\alpha_3)$
1	1	$\frac{l_{AC}}{l_A} f(\alpha_1) + \frac{l_{AB} l_{AC}}{l_A l_A} f(\alpha_2) + \frac{l_{AB}}{l_A} f(\alpha_3)$

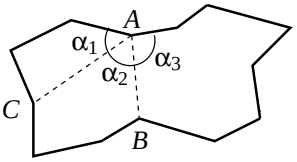

Fig. 4. Left: Term F_A; l_{AB} denotes the length of chord AB, l_A is the length of the shortest chord incident to A. **Right:** Chords (dashed) and corner angles incident to A.

2.2 The Objective Function

We now define a function E that determines whether one set of cuts is "better" than another. To that end we introduce a binary indicator variable i_c for every chord $c \in \mathcal{C}$ and use the notation $E(i_c | \mathcal{C})$ to indicate that E is a function of the $|\mathcal{C}|$ variables i_c, $c \in \mathcal{C}$. The assignment $i_c = 1$ means that chord c is a cut, $i_c = 0$ means c is not a cut. A set of assignments to all i_c is called a *configuration*.

$$E(i_c | \mathcal{C}) = \sum_{v \in V} F_v(i_c | \mathcal{C}_v) \tag{2}$$

Function E is the sum of $|V|$ terms, V being the set of polygon vertices. For every $v \in V$ we write $\mathcal{C}_v$ for the set of chords incident to v (every chord is incident to two vertices). Each term F_v in Equation (2) is itself a sum of the form $\sum_k w_k f(\alpha_k)$, where $\{\alpha_k\}$ are angles of part corners incident to v and $\{w_k\}$ are weights, which we will discuss shortly. The number of angles depends on the configuration and ranges between 1 and $|\mathcal{C}_v| + 1$.

For example, assume there are two chords, AB and AC, incident to vertex A (Fig. 4 Right). Then there are four possible configurations of $\mathcal{C}_A$ (Fig. 4 Left). With configuration $i_{AB} = i_{AC} = 0$ (no cuts incident to A) the value of F_A depends on the interior angle of the polygon at A only. With $i_{AB} \neq i_{AC}$ (one cut) it depends on the angles of the two corners separated by the cut and of the relative length of the cut, and with $i_{AB} = i_{AC} = 1$ it depends on three angles and two relative cut lengths.

Thus the f-terms in a sum F_v are weighted by the relative lengths of the cuts involved (the lengths are normalized by the length of the shortest chord incident to v). This weighting scheme is again motivated by the short-cut rule.

We finally turn to the function f, which has to be defined on the interval $(0, 2\pi)$. We derive its qualitative form from three principles:

1. Cuts should remove concave angles, except minor ones.
2. A convex polygon should never be partitioned.
3. Cuts that create an angle π or close to π are preferable.

From the second principle it follows that f should be non-increasing and have non-negative curvature in the range $(0, \pi]$. From the third principle it follows that f should have a minimum at π. We are free to choose $f(\pi) = 0$ as adding a constant to E does not affect the ranking of the configurations.

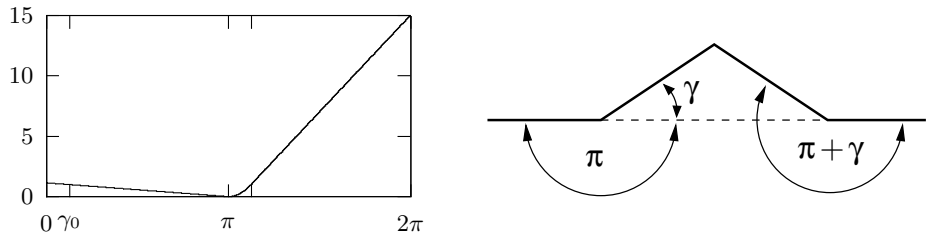

Fig. 5. Left: Plot of $f(\alpha)$ over α with $\gamma_0 = \frac{\pi}{8}$. **Right:** The chord (dashed) should become a cut only if $\gamma > \gamma_0$.

From the first principle it follows that $f(\alpha_1+\alpha_2) > f(\alpha_1)+f(\alpha_2)$ when $\alpha_1+\alpha_2 > \pi+\gamma_0$, where γ_0 is some small angle by which we realize the tolerance for minor concavities. To derive a constraint on f related to this tolerance parameter, we consider the situation depicted in Fig. 5 Right. The protrusion should be separated by a cut if and only if $\gamma > \gamma_0$. With $f(\pi) = 0$ it follows that $f(\pi+\gamma) > f(\gamma)$ when $\gamma > \gamma_0$ and $f(\pi+\gamma) < f(\gamma)$ when $\gamma < \gamma_0$.

The following simple function meets all the stated constraints. It is plotted in Fig. 5 Left.

$$f(\alpha) = \begin{cases} \frac{\alpha-\pi}{\gamma_0-\pi} & ,\alpha < \pi \\ \left(\frac{\alpha-\pi}{\gamma_0}\right)^2 & ,\pi \le \alpha < \pi+\gamma_0 \\ \frac{2}{\gamma_0}(\alpha-\pi)-1 & ,\pi+\gamma_0 \le \alpha \end{cases} \tag{3}$$

2.3 Robustness to Similarity Transforms

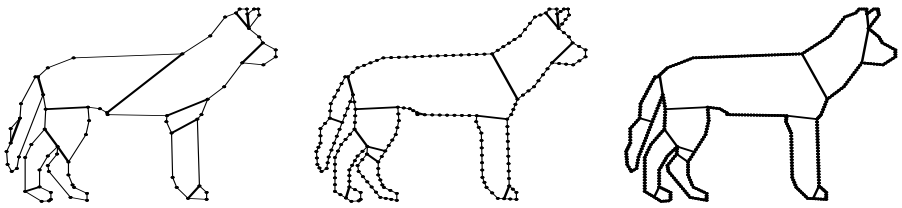

Fig. 6. Simplified polygon obtained with Lowe's algorithm (left). Polygon obtained from first by regular resampling with parameter $r = 8$ (middle) and $r = 3$ (right), respectively. Polygon vertices are indicated by points; in the right polygon, points overlap and appear as a continuous line. Each polygon is shown with best cut set.

The objective function Eq. (2) is invariant under rotation, translation and scaling as it depends only on angles between edges and chords, and on ratios of chord lengths. However, this invariance is irrelevant if the process by which

the shape contour is obtained is not also invariant or at least robust to these transforms.

We therefore take the output of a contour tracing algorithm and simplify it with Lowe's algorithm [8] to obtain a polygonal description robust to the named transformations. We next add polygon vertices so that the Euclidean distance between two adjacent vertices is bounded from above by some value r ($r = 8$ is used for all following results). This step is to ensure that the set of chords is dense in the sense that there are chords close to the cuts that would be obtained in the continuous limit $r \to 0$ (Fig. 6).

2.4 Minimizing the Objective Function

We briefly discuss two algorithms for minimizing the objective function, a dynamic programming algorithm which yields an optimal solution, and a greedy algorithm which finds a good but not always optimal solution.

The Dynamic Programming Algorithm: First observe that the number of arguments of each term F_v in the objective function Eq. (2) is much smaller than the number of arguments of E, $|\,\mathcal{C}_v| << |\,\mathcal{C}|$. Second, each variable i_c is an argument of only two terms. Whether $i_c = 0$ or $i_c = 1$ in the optimal configuration is conditional on the optimal arguments of these two terms only. For example, the optimal configuration for the variables in $\{i_c | c \in \mathcal{C}_A \cup \mathcal{C}_B - \{AB\}\}$ determines the optimal value for i_{AB}.

The dynamic programming or "variable elimination" approach consists of reducing the number of variables in the objective function one by one. We write E_n for the objective function obtained after n elimination steps. The functions E_n are related in that the optimal configuration for E is also optimal for all E_n. A variable i_c that is an argument of E_n is eliminated by replacing the two terms of E_n that i_c appears in by a single new term. The new term is defined as the minimum over the values of the eliminated variable of the sum of the replaced terms. For example, assume i_{AB} appears in the two terms $F(i_c|\,\mathcal{C}_F)$ and $G(i_c|\,\mathcal{C}_G)$ of E_n. We eliminate i_{AB} by replacing F and G by

$$H(i_c|\,\mathcal{C}_F \cup \mathcal{C}_G - \{AB\}) = \min_{i_{AB}} F(i_c|\,\mathcal{C}_F) + G(i_c|\,\mathcal{C}_G) \tag{4}$$

in E_n to obtain E_{n+1}.

The process of iteratively eliminating variables eventually yields a function E_m of only one variable. Call the variables eliminated in the successive steps i_1, i_2, ..., i_{m-1}, and the remaining variable i_m. The value o_m that minimizes E_m is the optimal assignment to i_m. The value that minimizes E_{m-1} with $i_m = o_m$ is the optimal assignment to i_{m-1}. Optimal assignments to variables i_{m-2}, i_{m-3},...i_1 are obtained in the same way.

The Greedy Algorithm: The greedy algorithm works as follows. Starting with the empty set as current cut set it repeatedly finds a single cut that, when added

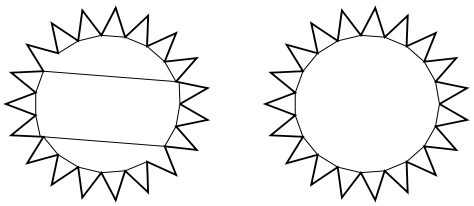

Fig. 7. An example in which the greedy algorithm gives an unsatisfactory result (left; optimal cut set on the right).

to the current cut set, reduces the objective function value by the largest amount. The algorithm terminates when no such cut can be found.

This algorithm often yields the optimal or a close-to-optimal solution, and it has the advantage of being faster and simpler than our original algorithm. However, it sometimes makes inappropriate cuts, as illustrated in Fig. 7.

2.5 Computational Complexity

To summarize, our algorithm first ensures the sampling of the silhouette is dense enough, and the number of vertices n is proportional to silhouette length. Second, $\mathcal{C}$, the set of possible cuts, is computed in $O(n \log n)$ time [6], and $|\mathcal{C}|$ is linear in n. Third, the objective function with $|\mathcal{C}|$ variables is dynamically created. Fourth, a configuration is found which minimizes the objective function.

The objective function is represented by a set of tables—one table for each term in Eq. (2). The complexity of the third step is governed by the number of tables (n), and by the size of the tables, which is $|\mathcal{C}_v| 2^{|\mathcal{C}_v|}$. Thus, if we use d to denote the maximum degree of the CDT computed in the second step, then the third step is $O(n(d-2)2^{d-2}) = O(nd2^d)$ in time and space.

Evaluation of the objective function may be implemented to cost $O(n)$ if the argument is a configuration, or $O(c)$ if the argument is a list of the c variables with non-zero assignment. We use the second variant.

With the greedy algorithm we first evaluate the objective function for all singleton cut sets (all but one variable values are zero) and store the corresponding variables in a heap with objective function value as key (takes $O(n \log n)$ time). We then take variables from the heap until the current configuration cannot be improved upon (m times, say), updating the heap in each iteration ($O(md + d \log n)$, as at most $2(d-2)$ variables are affected). Thus, the greedy algorithm takes $O(n \log n + nd2^d + md \log n)$time, which is $O(n \log n)$ if d is bounded.

For the optimal algorithm the situation is more complicated because the computational cost of computing the tables corresponding to new terms as in Eq. (4) hinges on the number $|\mathcal{C}_F \cup \mathcal{C}_G|$, respectively. How big these numbers get depends crucially on the order in which the variables are eliminated, and finding a variable ordering that minimizes the total table size is known as the "secondary

optimization problem" in nonserial dynamic programming [9]. We lack the space to discuss it here, but point out that, when the shape is simple (it has no holes), the structure of objective function (2) is described by an "interaction graph" [9] that is *chordal*. In this case an optimal elimination ordering can be found in $O(n+|\mathcal{C}|) = O(n)$ time [10]. The computational complexity of the optimal algorithm then is $O(n \log n + nD2^D)$, where D is the maximum number of variables of a single term that occurs in the course of the variable elimination process.

3 Results

Fig. 8 presents results of the global optimization and the greedy algorithm, respectively. For most shapes both cut sets are very similar, due to the fact that cut sets tend to be sparse subsets of $\mathcal{C}$ (every cut that does not share a vertex with another cut is selected by both algorithms). However, the results show the more complicated optimal algorithm might yield more appropriate results. For instance, all teeth of the cockscomb were separated by the optimal algorithm because cutting all teeth results in a relatively smooth part corresponding to the cock's head.

Many of the resulting parts seem too small (e.g., the parts of the lizard shape). This is because "part significance" is not reflected in the objective function. We believe that the best approach would be to determine part significance in subsequent processing and prune the cut set instead of incorporating part significance directly into the objective function. Occasionally neither of our algorithms selects a desired cut, as the elephant example shows: the leftmost leg is not separated from the torso. This is because the leg's left and right boundary curve both are too smooth and do not feature a "concave enough" vertex.

4 Related work

We briefly discuss the relation to other algorithms that assume a polygonal shape description as input. The algorithm by Latecki and Lakaemper [4] also chooses cuts that connect vertices of the input polygon. These vertices are found as the most stable vertices in a process of "discrete curve evolution". While Latecki and Lakaemper's method selects good vertices it does not pair according to proximity and sometimes yields implausibly long cuts.

Rosin's approach is perhaps closest in spirit to ours [11]. It also defines cuts as interior edges connecting polygon vertices and optimizes an objective function that favors a decomposition into convex parts. Unlike our algorithm, which chooses from $\mathcal{C}$, Rosin's algorithm considers all possible cuts and needs to identify crossing cuts during optimization. The computational cost depends exponentially on the number of cuts; searching for an optimal decomposition is only feasible if the number of cuts is very small (most decompositions presented in [11] contain only one or two cuts).

Prasad recently proposed selecting cuts from the chords of the CDT of a shape [12]. He defines a function on the set of chords that evaluates the degree

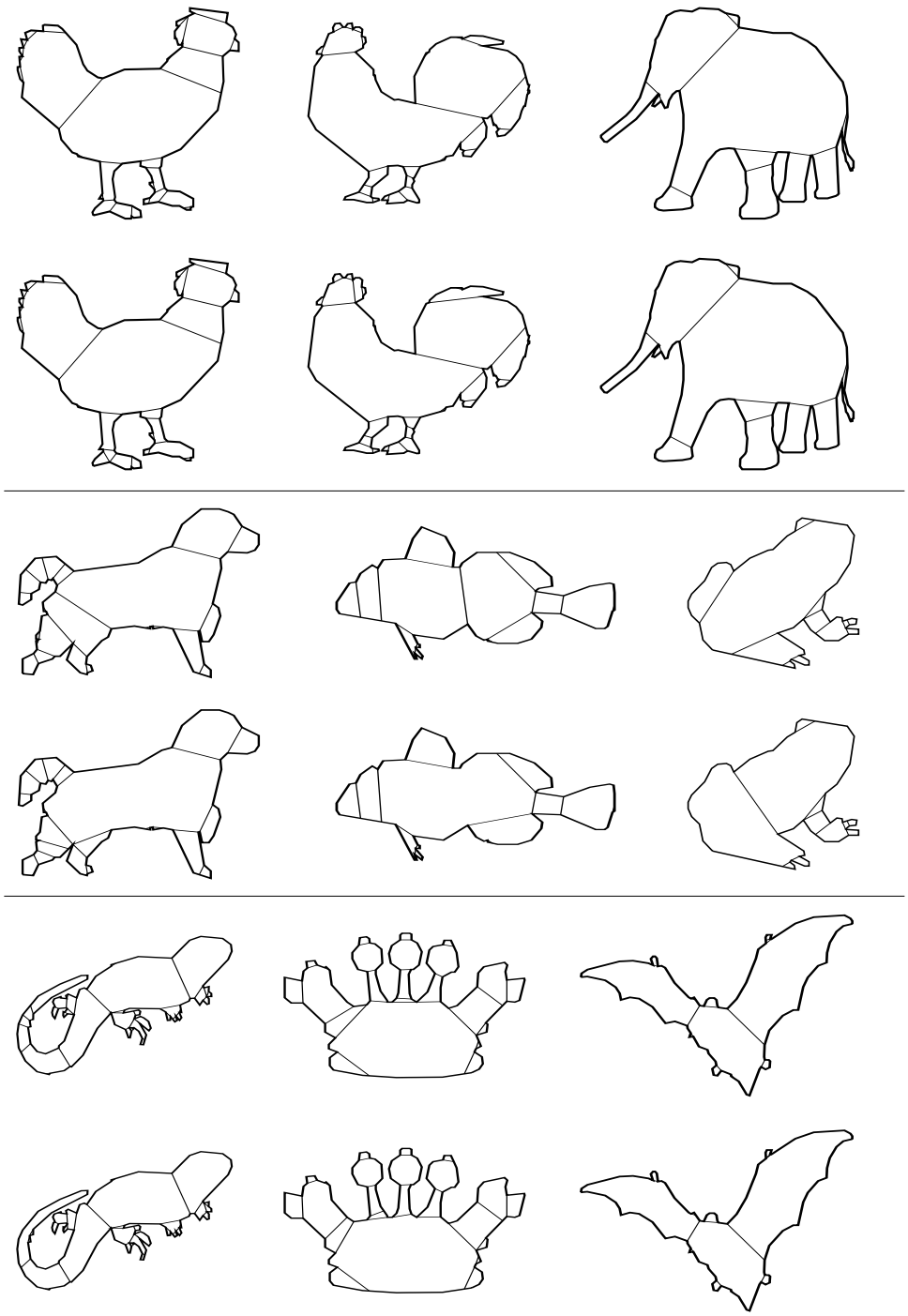

Fig. 8. Example shapes with optimal cut sets (top figure in each row) juxtaposed with the corresponding result of the greedy algorithm (bottom figure in each row).

of overlap of adjacent circumcircles ("approximate co-circularity"), and selects chords with low co-circularity score, resulting in good correspondence to the minima and short-cut rules. However, the connection to his co-circularity measure is unclear.

A quantitative evaluation and comparison with the cited other approaches is very desirable, but to our knowledge no established benchmark for shape decomposition currently exists. We plan to propose evaluation measures and to conduct a comparative study in future work.

5 Acknowledgements

This work was supported by a grant to MM from the J. S. McDonnell Foundation. We used J. R. Shewchuk's Triangle code [13], Y. LeCun and L. Bottou's Lush programming environment, and obtained the shape data for our experiments from R. Lakaemper.

References

1. Zhu, S.C., Yuille, A.L.: Forms: A flexible object recognition and modelling system. International Journal of Computer Vision **20** (1996) 187–212
2. Hoffman, D.D., Richards, W.A.: Parts of recognition. Cognition **18** (1985) 65–96
3. Singh, M., Seyranian, G., Hoffman, D.: Parsing silhouettes: The short-cut rule. Perception and Psychophysics **61** (1999) 636–660
4. Latecki, L.J., Lakaemper, R.: Convexity rule for shape decomposition based on discrete contour evolution. Computer Vision and Image Understanding **73** (1999) 441–454
5. Goodman, J.E., O'Rourke, J., eds.: Handbook of Discrete and Computational Geometry. CRC Press (1997)
6. Bern, M., Eppstein, D.: Mesh Generation and Optimal Triangulation. In Hwang, F.K., Du, D.Z., eds.: Computing in Euclidean Geometry. World Scientific (1992)
7. Brady, M., Asada, H.: Smoothed local symmetries and their implementation. International Journal of Robotics Research **3** (1984) 36–61
8. Lowe, D.G.: Three-dimensional object recognition from single two-dimensional images. Artificial Intelligence **31** (1987) 355–395
9. Bertele, U., Brioschi, F., eds.: Nonserial Dynamic Programming. Volume 91 of Mathematics in Sciece and Engineering. Academic Press (1972)
10. D. J. Rose, R.E.T., Lueker, G.S.: Algorithmic aspects of vertex elimination on graphs. SIAM Journal on Computing **5** (1976) 266–283
11. Rosin, P.L.: Shape partitioning by convexity. IEEE Transactions on Systems, Man, and Cybernetics, Part A **30** (2000) 202–210
12. Prasad, L.: Rectification of the chordal axis transform and a new criterion for shape decomposition. In E. Andres, G.D., Lienhardt, P., eds.: Discrete Geometry for Computer Imagery, 12th International Conference, Poitiers (France), April 2005, Proceedings. Volume 3429 of LNCS., Springer (2005) 263–275
13. Shewchuk, J.R.: Triangle: Engineering a 2D quality mesh generator and delaunay triangulator. In Lin, M.C., Manocha, D., eds.: Applied Computational Geometry, Towards Geometric Engineering, FCRC'96 Workshop, WACG'96, Philadelphia, PA, May 27-28, 1996, Selected Papers. Volume 1148 of LNCS., Springer (1996) 203–222