Interpolating an Unorganized 2D Point Cloud with a Single Closed Shape

Stefan Ohrhallinger[1], Sudhir P. Mudur[2]

Abstract

Given an unorganized two-dimensional point cloud, we address the problem of efficiently constructing a single aesthetically pleasing closed interpolating shape, without requiring dense or uniform spacing. Using Gestalt's laws of *proximity*, *closure* and *good continuity* as guidance for visual aesthetics, we require that our constructed shape be minimal perimeter, non-self intersecting and manifold. We find that this yields visually pleasing results. Our algorithm is distinct from earlier shape reconstruction approaches, in that it exploits the overlap between the desired shape and a related minimal graph, the Euclidean Minimum Spanning Tree ($EMST$). Our algorithm segments the $EMST$ to retain as much of it as required and then locally partitions and solves the problem efficiently. Comparison with some of the best currently known solutions shows that our algorithm yields better results.

Keywords:
computational geometry, reconstruction, construction, shape, curve, boundary, point cloud, point set, EMST

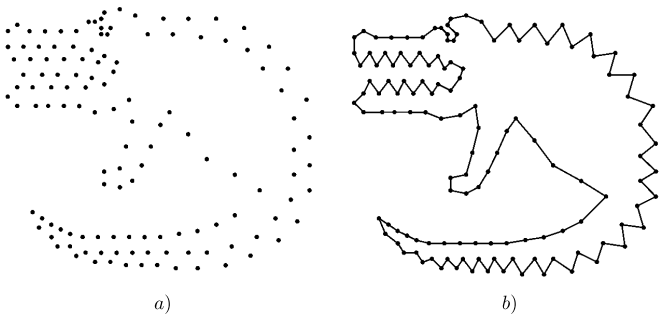

Figure 1: a) A sparsely sampled unorganized point set. b) Its closed manifold interpolating boundary with minimum perimeter.

1. Introduction

The goal is to identify a single aesthetically pleasing shape connecting points in a 2D unorganized point set, without requiring dense or uniform spacing. To fulfil the aesthetic requirement, we use for guidance Gestalt's laws of *proximity* (human tendency to connect close dots), *closure* and *good continuity* (smoothness) [27] to obtain measurable and objective criteria. Accordingly we require that the shape be a closed, non-selfintersecting manifold (law of *closure*) which interpolates all the points with minimum length (law of *proximity*), henceforth denoted by S_{min}. For the law of *good continuity* we will present a further constraint below. If we exclude extreme point distributions from our problem domain, the algorithm presented in this paper provides an efficient solution (see Figure 1).

The task of 2D shape reconstruction from boundary sampled points plays an especially important role in a number of engineering fields: reverse engineering of geometric models, outline reconstruction from feature points in medical image analysis, etc. The closed boundary is essential for calculating various shape moments, a characteristic property with many applications.

2. Related Work

Polygons interpolating a point set are also a topic in computational geometry, but the focus there is mostly on investigating lower and upper bounds on the total number of interpolating polygons based on the size of the given point set, say, as in [10].

If we consider all the methods in the literature for 2D shape reconstruction, they can be classified according to two major approaches which are discussed further below.

2.1. Reconstruction with a Local Sampling Condition

Early methods worked only on smooth and uniformly sampled point sets, such as α-shapes [8, 16], Figueiredo and Gomes [17], β-skeleton [24], γ-neighborhood graph [30] and r-regular shapes [6]. For example, α-shapes requires user-specification of a global constant which depends on sampling. It does not work for non-uniformly sampled point sets. It also cannot guarantee a manifold the way our algorithm does.

Amenta et al. [3] with their *Crust* algorithm introduced the concept of local feature size which allows reconstruction from non-uniformly sampled point sets. The stated sampling requirements of the *Crust* method and its successors [11, 12] are however quite restrictive in theory and difficult to ensure in practice. Not only is it difficult to check if a given point cloud satisfies the sampling requirement, but it is even more difficult to construct a sampling satisfying the requirement. It should be noted though that the presented algorithms often show reconstruction of less restricted point sets but with no guarantees. *DISCUR* [31] uses the two properties of proximity and smoothness but still requires rather dense sampling in sharp corners. Some improvements on these aspects have been made in *VICUR* [28],

but it relies very much on user-tuned parameters and regresses for other point sets. The *Gathan* algorithm from Dey et al. [13] also handles sharp corners, but again without guarantees. *GathanG* [14] is an extension which, like our work, is targeted at closed shapes and gives guarantees exclusively for certain conditions. It still does not work for many cases we tested. In spite of this, it provides in our opinion the best solution to date for this 2D shape reconstruction problem.

All of the above-mentioned algorithms reconstruct a boundary using edges in the Delaunay Graph (*DG*) and results have shown that this is a very reasonable choice. The *DG* has the property of maximizing its angles and minimizing its edge lengths, which conform to the Gestalt laws of *good continuity* and *proximity*. A minimum boundary which is not constrained to *DG* may trade in longer edges and sharper angles instead.

A fundamental advantage of our method versus using a local criterion is that we can achieve far superior results for reconstruction of the single closed manifold shape which we require, for the particular subclass of point sets which represent such a shape. Instead the output of the previous methods only partially reconstructs such a shape as one or more open curves or as a number of ambiguous shapes (Figure 13 shows a number of such cases).

2.2. Construction as Global Minimization of a Criterion

Finding the minimum perimeter closed boundary actually requires a global search of the solution space.

A first attempt on global construction presented in [20] finds spanning Voronoi trees and selects the one with minimal length by integer programming, with O($n^2 log$ n) complexity. It does not work well for sharp angles and non-uniform sampling; obviously it prunes good solutions too early.

Giesen shows in [19] that the exact solution to the travelling salesman problem (TSP) can reconstruct the shape for sufficiently dense sampling. Althaus et al. extend this work in [1] to non-uniform sampling with some conditions, and in [2] they compare it with both the *Crust*-type family of algorithms and TSP-approximations. They note that the latter two methods fail for certain curves with sparser sampling which the exact TSP method handles well. They also mention that the exponential complexity of the TSP decreases with denser sampling. With the exception of [19], these methods do not require user-specified parameters. Unfortunately, finding the exact solution using the TSP approach takes unreasonable time O(2^n) even for small *P*. The *concorde* exact TSP solver [4] scales sub-exponentially and can take hundreds of CPU-years for medium-sized point sets. A detailed discussion on its complexity is available in [23].

TSP approximations show more reasonable complexity but are not linearithmic, i.e. $O(n^{2.2})$ [22] or $O(n(\log n)^{O(c)})$ for a $(1 + 1/c)$ approximation of the optimal tour of an Euclidean TSP [5], and $O(n^{O(1/\epsilon)})$ for $(1 + \epsilon)$ times the solution for a planar graph TSP [21]. More importantly, they fail to guarantee the minimum solution and even a single wrongly connected edge may have a significant impact on aesthetic quality of the reconstruction. Hence approximation schemes for TSP cannot guarantee the desired interpolating and manifold shape.

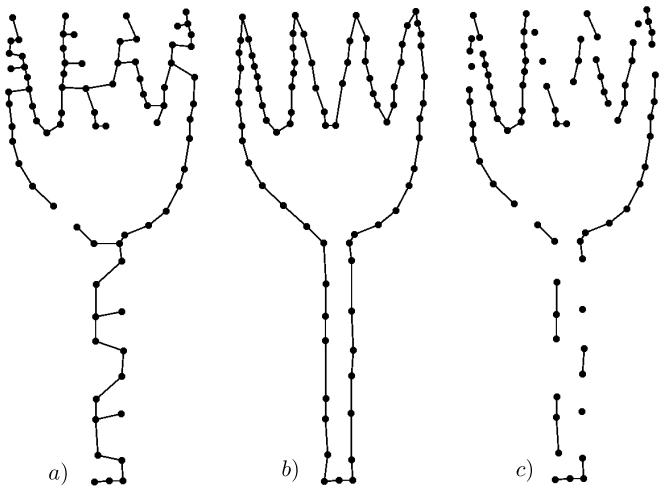

Figure 2: $EMST$ and S_{min} have considerable overlap even for sparsely sampled point sets like the tulip: a) $EMST$ has 78 edges. b) S_{min} has 79 edges. c) 57 of their edges are shared.

While we too impose the minimum perimeter requirement, by restricting the sub-domain of that problem to edges in *DG*, we exploit the relationship between the Euclidean minimum Spanning Tree ($EMST$) and S_{min} (constrained to *DG*). Our algorithm segments the $EMST$ and classifies the segments so as to retain as many of them as possible in the reconstruction of S_{min}. This way we partition the problem and provide an efficient solution. S_{min} differs from $EMST$ only by restricting its vertices to be manifold, which in turn increases its length (see Figure 2).

This relationship between the minimum spanning tree and the shape has been mentioned in Figueiredo and Gomes [17]. However, they only prove reconstruction for very densely sampled point sets: an $EMST$ without branches. They do suggest some parameter-based heuristics for more sparsely sampled point sets, but do not really exploit this relationship in the unique way we do in our algorithm. We show that our algorithm can quickly find the desired solution and scales well to handle very large point sets. And if we exclude extremely sparse and highly non-uniformly sampled point sets, our algorithm's complexity is just O(n *log* n). While we do note that a constrained TSP solution restricted to the planar graph *DG* would yield the same result, i.e., S_{min}, we are not aware of any TSP solution with this performance.

2.3. Intuitive Overview of our Method

Our method starts from a Delaunay graph (*DG*) and the Euclidean minimum spanning tree ($EMST$) of the point set (see Figures 3a and 3b). $EMST$ is a subset of *DG* (Attene and Spagnuolo [7]) and can be constructed in $O(n \log n)$ (Kruskal [25]).

It can have a number of non-manifold vertices with degree not equal to 2 (leaf or fork vertices). To such vertices we apply edge exchange operations like those used in the degree-constrained spanning tree problem [29]. Determining this set of operations so that $EMST$ is transformed into S_{min} is NP-hard. Our main contribution in this paper is an innovative way

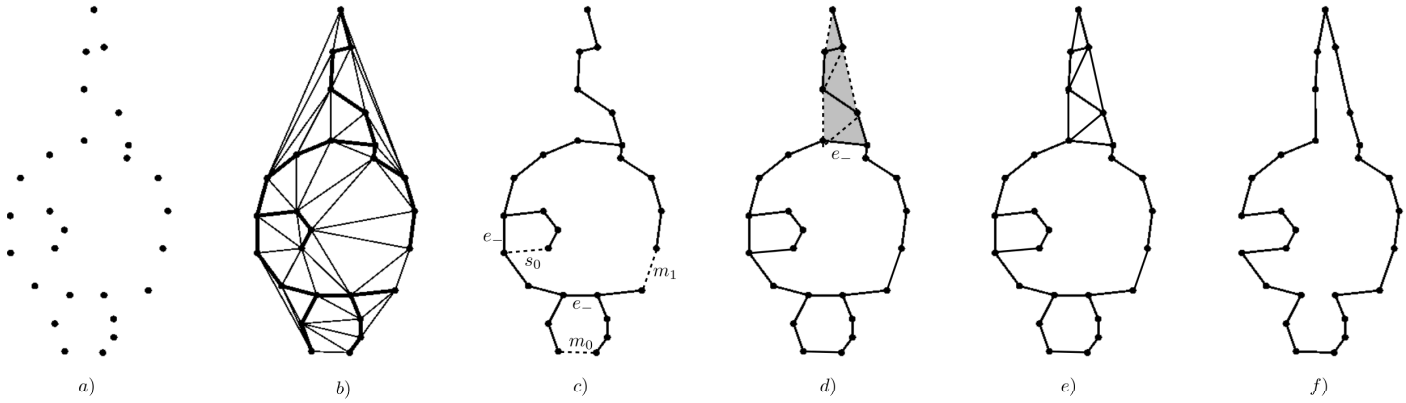

Figure 3: a) Example point set P. b) DG with edges in $EMST$ emphasized. c) Edges incident to leaf vertices (m_0, m_1 and s_0) shown with dotted lines. d) $EMST$ with edges added and envelope of *inflatable* branch shaded with its edges not in $EMST$ shown with dotted lines. e) Envelope edges added. f) Removal of cut edges yields S_{min}.

of efficiently performing these edge exchange operations.

The other steps in our algorithm are the following:

- Segment the $EMST$ at fork vertices and retain as many segments as possible, since such segments are already of minimum length (Figure 3c).
- Adding an edge in DG to the $EMST$ graph creates a loop. If two loops share a single edge called as *cut edge*, then deleting just the cut edge results in a single loop interpolating points in both loops. On the other hand if two loops share a single vertex or an edge-chain, then removal of these shared vertices will result in either a split graph or vertices which are not interpolated. Hence, we only add edges which lead to loops sharing a cut edge. The next step is therefore to select and add the $DG \setminus EMST$ edges, incident to leaf vertices, needed to make S_{min}.
- The resulting graph will have the following configuration: (i) a single loop or multiple loops connected by pairs of loops sharing cut edges, and (ii) segments (like strands) of $EMST$ connected to a loop at one end and open or connected to another loop at the other end (Figure 3d).
- In the next step the strand-like segments are converted (inflated) into loops and then all cut edges are removed to yield a manifold interpolating shape (Figure 3d-f).

In the following sections we present these steps in detail and further identify the point configurations for which our algorithm is guaranteed to work.

3. Definitions

Closed shape S is a single manifold polygon interpolating all $v_i \in$ point set P and consisting of edges $e_i \in$ Delaunay graph DG of P. $\{S_i\}$ denotes the set of all such closed shapes in P and S_{min} denotes the one with minimum perimeter.

Hamiltonian graph is a graph $G = (V, E)$ with at least one closed shape S (Dillencourt [15]). Genoud [18] shows that DG for any P is rarely non-Hamiltonian.

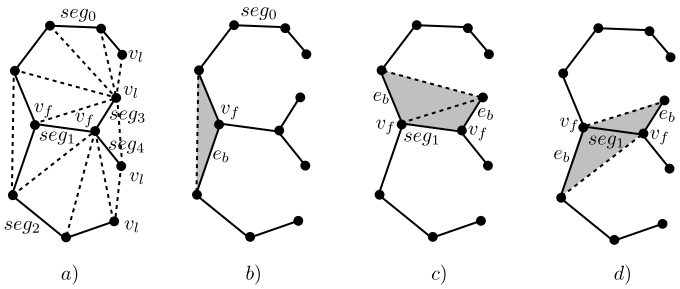

Figure 4: $EMST$ with other edges in DG shown dotted: a) Segments labeled as seg_i, leaf vertices as v_l, fork vertices as v_f. seg_1 is a trunk segment, all other segments are branches. b) One of two non-manifold envelope boundaries of seg_0 (shaded grey) with base edge e_b. seg_0 is therefore a *retained* segment. c) First manifold envelope boundary of seg_1 with two fork vertices and a base edge each. seg_1 is therefore a *non-retained* segment. d) Second manifold envelope boundary of seg_1; the other two envelope boundaries are non-manifold.

Loop is a cyclic sequence of edges.

Segment s is a sequence of manifoldly connected edges terminated by non-manifold vertices (with degree $\neq 2$), either *leaf vertices* v_l (degree 1) or *fork vertices* v_f (degree >2). A segment with at least one leaf vertex is called a *branch* b. All other segments are *trunk segments* (see Figure 4 for examples).

Cut is the set of vertices of a connected graph G whose removal renders G disconnected. A *cut edge* has two cut vertices and a *cut edge-chain* has more than two.

Uniformity of sampling $u = d_i/d_j$ where d_i, d_j are the Euclidean distances between a point $p \in P$ and its neighbors in S_{min}, sorted such that $d_i > d_j$. u_{max} is then the largest u for any $p \in P$. The larger u_{max} is, the less uniform is P.

Sharp angled features: In [3] the notion of *Local Feature Size* was introduced, primarily for smooth shapes (no sharp corners), which depends on local curvature and proximity. They state: *For an r-sampled curve in the plane,* $r < 1$*, the angle spanned by three adjacent samples is at least* $\pi - 4arcsin(r/2)$. This condition does not evaluate for $r >= 1$, therefore it cannot support angles <= 60°. Since our method can handle much sharper angles, we use instead α_{min}, the minimum angle between any three adjacent points in the desired closed shape as a

measure of sharp features.

4. Defining Operations

4.1. Segment Classification

We classify $EMST$ segments into those which are part of S_{min} and those which are not. This classification is based on the observation that for *retained* segments, any other edge sequence connecting the segment's interior vertices (all except the end vertices) will either result in an increase in length or will not be manifold.

Base edge e_b for a segment s is defined as follows. Let the edge $e_i \in s$ be incident to a fork vertex $v_f \in s$. Then two base edges e_b are the immediately adjacent edges in *cw* and *ccw* sense, incident to v_f . It may be noted that a trunk segment has four base edges, a branch with one leaf vertex has two and a branch with both ends as leaf vertices has none. The vertex of e_b opposite to v_f is called a *base vertex* v_b.

Envelope $env(s)$ for a segment s is the set of Delaunay triangles for which the vertices consist of the vertex set $\in s$ and one base vertex per fork vertex. The envelope boundary may or may not be *manifold*. A branch has two envelopes and a trunk has four.

Retained segment is a segment for which none of its envelope boundaries is manifold. For such segments, there is no alternative way of manifoldly interpolating the segment vertices without increasing their length. All other segments are *non-retained* (see Figure 4 for both cases).

4.2. Inflate and Select Minimum Loop Operation

A non-retained segment for which a manifold envelope boundary exists is a strand which may require to be modified to become part of the desired closed shape. A manifold envelope boundary can always be modified to form a loop which interpolates all the vertices in that segment. There may be a number of different choices for forming these interpolating loops. We select the minimum length one. We call this as the *inflate* operation associated with the segment. Since all loops of a segment share its base edges with the remainder of S_i, this operation corresponds to solving the reconstruction problem locally, i.e., segment-wise.

4.3. Edge Displacement Operations

S_{min} for a point set with n points has n edges, while its $EMST$ has one edge less since it is a tree.

Let $E_+ = S_{min} \setminus EMST$ denote the set of edges that we need to add to $EMST$ when transforming it into S_{min}. Note that the same number of edges minus 1 must be removed from $EMST$ to maintain the edge count.

A subset of potential edges in E_+ is easily identifiable. In this subset edges are incident to leaf vertices. Each of them forms a loop when added to $EMST$. We classify the operations of adding edges into two types of edge displacement operations (see Figure 5 for examples):

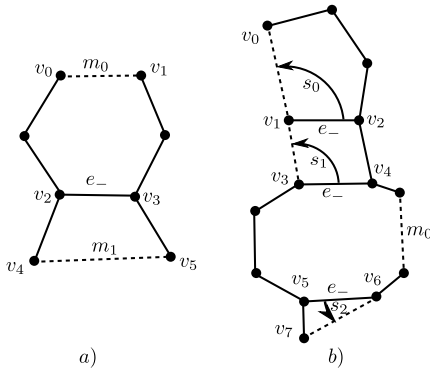

Figure 5: a) Example of $EMST$ with *moves*: $m_0(v_0, v_1)$ and $m_1(v_4, v_5)$ both associated with the trunk segment $e_-(v_2, v_3)$. b). Another example of $EMST$ with *move* and *snaps*: $s_0(v_1, v_0)$ with $e_-(v_1, v_2)$ has two leaf vertices, $s_1(v_3, v_1)$ with $e_-(v_3, v_4)$ and $s_2(v_6, v_7)$ with $e_-(v_6, v_5)$. For s_0: $v_- = v_0$, $v_= = v_1$, $v_+ = v_2$.

- *move* edge e_+: between two leaf vertices. For satisfying the manifold condition, there has to exist a corresponding edge $e_- \in EMST$ incident to a fork vertex in the *move*'s loop. We say that e_- *moves* to e_+ since the two edges do not share any vertices. Let us recall that $EMST$ has one edge less than S_{min}. Therefore there will exist one *move* without e_-.

- *snap* edge e_+: incident to a leaf vertex v_-. Its other vertex $v_=$ can be a leaf or a manifold vertex. $e_-(v_=, v_+)$ is the edge incident to a fork vertex v_+ in the *snap*'s loop. We say that e_- *snaps* from v_+ about $v_=$ to v_-, to become e_+.

We want to underline that only a subset of the *move* and *snap* operations identified this way will actually need to be applied. Therefore we will use the term *candidate* for them in the context where they are just potential operations, as opposed to when they have been definitely applied as operations.

The e_- edge of a *snap* candidate can be locally identified. The e_- edge of a *move* could be anywhere in its loop, and in principle entails a global search. However, as mentioned earlier, we ingeniously avoid this global search, by clustering together all the add edge operations (e_+) and then removing all the corresponding e_- edges at once, as they are all identifiable as cut edges.

4.4. Associations of Candidates to Segments

In order to decide which of the candidate edges should be added to make S_{min}, we create a segment - candidate association table and then evaluate the candidate's applicability based on three conditions, interpolation, manifold and minimum length. This table enables the evaluation of all potential solutions. One column indicates the type (*retained*/*non-retained*) and another distinct column lists all the candidates associated with each segment in $EMST$.

Every segment entirely contained in the loop created by the addition of a candidate edge is said to be *associated* with it and vice versa. The only exception is the segment containing edge e_- in the loop formed by a *snap* edge e_+, since this e_- is

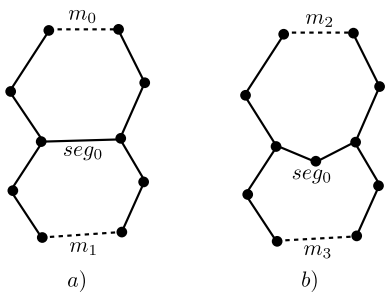

Figure 6: a) Single-edge trunk segment seg_0 can be contained in the loops of two candidates m_0, m_1 since it forms a *cut edge*. b) Candidates m_2, m_3 can not both be permitted to be applied for multiple-edge trunk segment seg_0: since its removal will disconnect v_0.

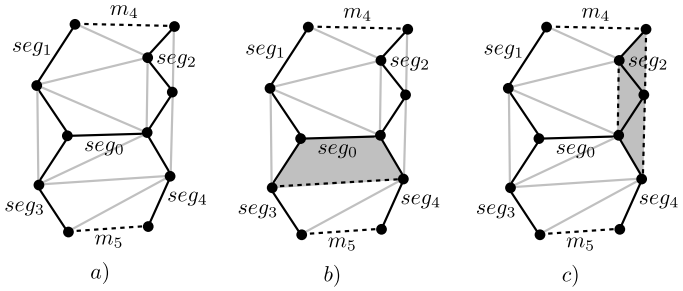

Figure 7: a) $EMST$ of a point set with DG edges shaded grey and segments and candidates marked. b) manifold envelope of seg_0 shaded. c) manifold envelope of seg_2 shaded.

removed in a subsequent step. Let us note that this segment-candidate association is many-to-many, see for example Figure 6b, in which seg_0 is contained in the loops of two *move* candidates m_0 and m_1. Of course, we can only apply one of them while respecting the condition of a manifold boundary.

4.5. The Solution Space Tree

A set of candidates associated with a segment is said to be *multi-choice* if not all of them can be applied simultaneously. Table 1 shows this for the example point set shown in Figure 7. In this example, trunk #0 and branch #2 are the multi-choice segments. Using such a table, we can explore all potential solutions by viewing the solution space as a tree. Each candidate associated with a multi-choice segment represents a branching point. Each terminal node contains a potential solution with a permissible subset of candidates per segment. The size of the solution space tree is the product of its multi-choice candidate set sizes. So this example contains $3 * 2 = 6$ potential solutions.

4.6. Pruning of the Solution Space Tree

It is easy to see that the solution space tree can grow quickly. For efficient searching, our algorithm prunes parts of this tree as early as possible by eliminating any candidates leading to a non-manifold solution. In fact it dynamically constructs the solution tree, doing the branching required for exploring new solutions only where it is unavoidable. If a candidate is the only one associated with a segment, then it is applied. This in turn could lead to reducing the number of candidates in other

Segment	Segment type	Candidate Ops
trunk #0	*non-retained*	m_4 m_5 $inflate$
branch #1	*retained*	m_4
branch #2	*non-retained*	m_4 $inflate$
branch #3	*retained*	m_5
branch #4	*retained*	m_5

Table 1: Segment-candidate table for point set of Figure 7.

multi-choice segments, which may result in more such single candidates. Thus evaluation of multiple solutions is only necessary when we are left with nothing but multi-choice segments in the table.

We shall see in the examples later that even in problems with very large solution spaces, this procedure of eliminating candidates is very effective and typically results in construction of only a small part of the solution tree. This is what makes this algorithm efficient. More details of the algorithm follow, giving all the states when candidates can be applied, eliminated and when multiple solutions have to be evaluated.

5. Algorithm

1. For a given point set P, create DG and $EMST$, segment $EMST$ and classify segments as *retained* or *non-retained*.
2. Identify candidate operations and create segment-candidate association table.
3. Initialize $E_{curr} = EMST$.
4. Apply an applicable *move* or *snap* candidate from the segment-candidate association table. A candidate is not applicable if it results in a non-manifold condition, namely, a cut vertex, a cut edge-chain or its e_- causes a loop to become open.
5. Prune the segment-candidate association table by *eliminating* all associations of invalidated candidates as follows. As a result of the previous add edge operation, some segments will already be part of a loop; their associated candidates are no longer needed and are removed from the table. Some of the leaf vertices will become manifold; candidates incident on such vertices are also removed from the table. Lastly, any candidate which if applied will lead to a non-manifold boundary is also removed.
6. Repeat steps 5 and 6 (*Apply* and *Prune*) until there are no more *move* or *snap* operations left.
7. Carry out *inflate* operations remaining in the table.
8. Remove cut edges in E_{curr}

The above are the major steps in our algorithm. In the implementation there is detailed case analysis based on the type of operation *move* or *snap* for detecting the non-manifold condition.

Once the association table contains only multi-choice segments, we select the segment with least number of candidates, and explore all the solutions. The order in which candidates are applied (and thus the order in which the solution space is traversed) does not matter. This is because our algorithm evaluates all potential solutions.

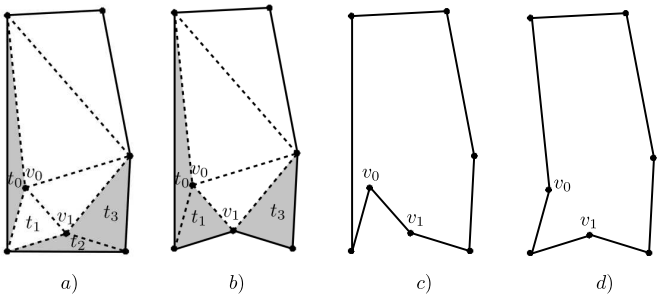

Figure 8: A manifold envelope boundary env_0 (shown using solid lines) with two interior vertices: a) shows triangles t_0, t_2 and t_3 for removal from env_0. b) $env_1 = env_0 \setminus t_2$: new removal triangles are t_0, t_1 and t_3. c) $env_2 = env_0 \setminus \{t_2, t_1\}$: one potential local loop since no interior vertices remain. d) Final minimum length local loop for the example: $env_3 = env_0 \setminus \{t_0, t_2\}$

5.1. Inflate Operation

As already mentioned, we will need to apply the *inflate* operation to all the connected sets of remaining segments (*inflatable sub-trees*) which do not form part of any loop. This is described next. Triangles with a single edge on the envelope's boundary and one vertex in the interior are placed in a list and removed until no interior vertices are left. This will yield a loop interpolating all the vertices in the segment. The loop causing minimum increase in perimeter of S_{min} is chosen (see Figure 8).

5.2. Remove Cut Edges

After all the inflate operations are carried out, there are no more operations in the segment-candidate association table to apply. E_{curr} will have a number of loops connected to each other through cut edges, which have to be detected and deleted to yield an interpolating manifold boundary. A generic algorithm to detect cut edges in a graph is of higher than linear (worst case) complexity. But we can do this with linear time worst case complexity by exploiting the knowledge we have about the applied candidates as follows:

We know that the cut edges of *snaps* are their e_- and the ones of the *inflates* are inside their chosen modified envelope. The remaining cut edges are associated with *moves*. These are not known but do not overlap among each other.

Therefore we just have to first remove all cut edges resulting from application of *snaps* and *inflates* and then remove all edges between vertices of degree > 2.

5.3. Examples with multi-choice segments

The segment-candidate shown in Table 2 for our first example (a figure-eight), has all multi-choice segments, except branch #2 (Figure 9. The *Apply-Prune* steps leading to the solution are also shown.

Let us briefly follow the progress of our algorithm for this example. m_2, the only operation associated with branch #2 is first applied. Consequently s_1, m_3 and s_6 get eliminated. m_5, the only operation associated with branch #4 is applied next. All remaining operations get eliminated, resulting in the desired shape.

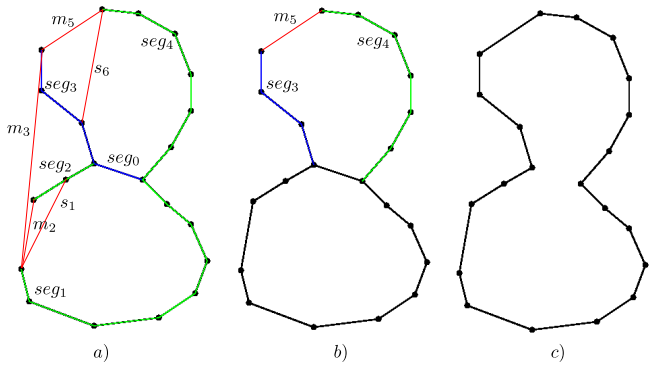

Figure 9: a) $EMST$ with segments (blue: *non-retained*, green: *retained*) and candidates (red) marked. b) Applying m_2 creates a loop in the lower half and leaves only one candidate m_5. c) Applying m_5 and removing the cut edge yields a single S_i, which is the desired solution.

Segment	Type	Candidate Ops	m_2 applied	m_5 applied
trunk #0	*non-retained*	s_1 s_6 m_2 m_3 m_5 *inflate*	$\mathbf{m_2}$ m_5 *inflate*	$\mathbf{m_2}$ $\mathbf{m_5}$
branch #1	*retained*	s_1, m_2, m_3	$\mathbf{m_2}$	$\mathbf{m_2}$
branch #2	*retained*	$m_2 \leftarrow$	$\mathbf{m_2}$	$\mathbf{m_2}$
branch #3	*non-retained*	m_3 m_5 *inflate*	m_5 *inflate*	$\mathbf{m_5}$
branch #4	*retained*	s_6 m_5	$m_5 \leftarrow$	$\mathbf{m_5}$

Table 2: Progression in segment-candidate table (right-most column is solution). Candidate to apply next is marked by arrow.

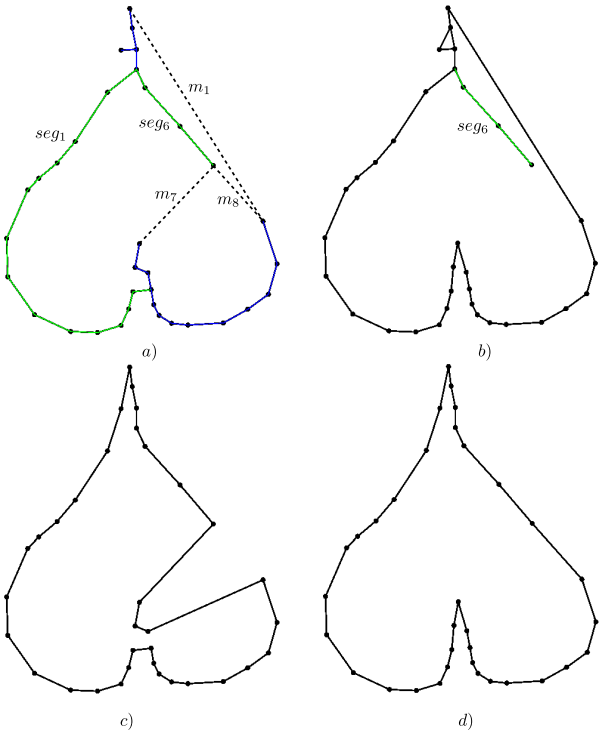

Figure 10: a) $EMST$ for point set from [31] with segments marked: green = *retained*, blue = *non-retained*. The multiple choice of candidates at seg_1 creates three potential solutions PS_i: b) PS_0 applies m_1: in the end seg_6 remains without candidates therefore PS_0 becomes invalid. c) PS_1 applies m_7 and produces S_0. d) PS_2 applies m_8 and produces S_1 which is S_{min}.

Segment	Type	Candidate Ops
trunk #0	*non-retained*	m_0 m_1 s_4 *loop*
trunk #1	*retained*	m_1 m_7 m_8
branch #2	*non-retained*	m_0 m_1 s_2
branch #3	*non-retained*	s_2 s_3 s_4
branch #4	*non-retained*	m_6 m_7 *loop*
branch #5	*non-retained*	m_1 m_6 m_8 s_9
branch #6	*retained*	m_0 m_7 m_8

Table 3: Initial segment-candidate table.

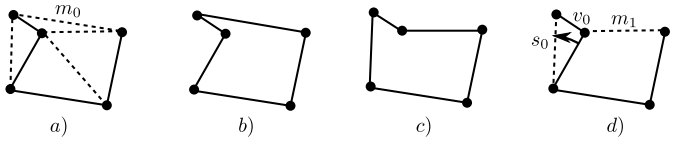

Figure 11: a) $EMST$ of non-conforming point set, with single operation m_0. b) Result is S_0 which is not minimal. c) S_{min}. d) Therefore m_1 and s_0 are the correct operations to obtain S_{min} as their vertex degree changes cancel out at v_0 but they are not detectable.

5.4. Examples with several potential solutions

For the point set shown in Figure 10 all are multi-choice segments (see Table 3). The complete solution tree would have $4 * 3 * 3 * 3 * 3 * 4 * 3 = 3888$ terminal nodes (potential solutions). However, using our algorithm, very large parts of this solution tree get pruned and only a small number of potential solutions have to be actually evaluated to get S_{min}.

The algorithm chooses the first segment with the minimum number of (all applicable) associated candidates: seg_1. It branches into three potential solutions as shown in Figure 10. All other parts of the solution tree get pruned, so no further solutions need to be explored. Thus the total size of the search remains at 3.

5.5. Conforming Point Set Configurations

Like most other algorithms, dense, uniformly sampled point sets are rather easily handled by our algorithm; this is also evident from the proof given in [17]. For such dense point configurations, the $EMST$ already shares most edges with the desired boundary and the solution space tree remains small. As example we show a fairly large point set derived from a silhouette in Figure 12e. Noisy data is also interpolated as long as shape features are not significantly affected. If the noise is such that the point spacing gets unreasonably non-uniform, then the algorithm will terminate with an incomplete or incorrect result. For such noisy data an approximation algorithm like the one presented in [26] should be preferred. If one looks closely at the statistics in Table 4, those point sets which are densely sampled (Figure 12e) are relatively less complex to reconstruct than sparsely sampled sets.

Below we shall define the class of point configurations for which our algorithm can guarantee a minimum length boundary shape and give the proof for this. In a subsequent section we derive the computational complexity of our algorithm.

Our algorithm can guarantee the result for point set configurations in which edges in all required *moves* and *snaps* operations are connected to leaf vertices in $EMST$. Hence our algorithm requires that the input point set satisfy the following condition:

move and *snap* operations must not overlap such that a e_+ and a e_- are incident to the same vertex. While such an overlap does not violate the manifold condition, the operations themselves are individually non-detectable (see an example in Figure 11).

We denote point sets satisfying the above condition as the *conforming class*.

Theorem 1. *Our algorithm always terminates for point sets in the conforming class and produces their S_{min}.*

Proof 1. *All edges in* retained *segments of $EMST$, excepting the edges at each end, are guaranteed to be in S_{min}. Since the vertices of these edges are manifold, they do not permit any add edge operation without creating a non-manifold result. This proves that these edges do belong to S_{min}. For inflatable segments all interpolating loops through their vertices are evaluated and the minimal one selected. Finally, by the above condition, all remaining edge exchange combinations are detected and evaluated, and the combination yielding the manifold closed shape with minimal length is chosen.*

In our experience the conforming class includes most sampled point sets encountered in practice, as they are usually dense and uniform. It also includes point configurations that are considerably more non-uniform and sparse. Further, in practice, for many point sets outside the conforming class, the algorithm will terminate and produce an interpolating shape, which is also aesthetic. However the above guarantee does not apply. We suggest later an extension to the algorithm to include an expanded class of point sets. With this extension, except in places where points are highly non-uniformly spaced or extremely sparse, our algorithm will reconstruct an aesthetically pleasing shape.

6. Results

Point set	n	l	m	i	Comb.	s_g	s_l	α_{min}	u_{max}
Tulip	79	15	18	2	$> 10^{14}$	1	12	20°	4.2
Goose	74	14	10	1	$> 10^{8}$	1	4	61°	3.6
Octopus	166	37	20	3	$> 10^{35}$	4	40	72°	2.7
Crocodile	129	8	7	1	46080	2	5	34°	2.5
Elephant	75	15	9	0	$> 10^{14}$	5	2	13°	5.5
Inverted heart	34	5	6	0	3888	3	1	19°	3.3
Close curves	13	4	10	2	128	3	25	49°	2.1
Random10	10	4	0	0	270	3	1	40°	6.6
Three loops	18	5	8	0	540	1	1	76°	2.0
Rail joint	30	3	10	1	9	2	3	122°	16.5
Family	9990	33	5	1	$> 10^{14}$	16	2	45°	1.4

Table 4: Complexity table: n=points, l=leaf vertices, m=vertices in largest inflatable sub-tree, i=largest number of interior vertices in such a sub-tree. *Combinations* gives the number of the terminal nodes (solutions) in the hypothetical complete solution tree. s_g is the number of solutions evaluated globally. s_l is the maximum number of solutions evaluated locally for any *inflatable* sub-tree. α_{min} is the minimum angle. u_{max} is the largest local non-uniformity factor.

We have implemented this algorithm and tested it with a very large number of sample point data sets. Specifically, we have tested the performance of our algorithm on many of what are considered as problematic point sets, point data for which the

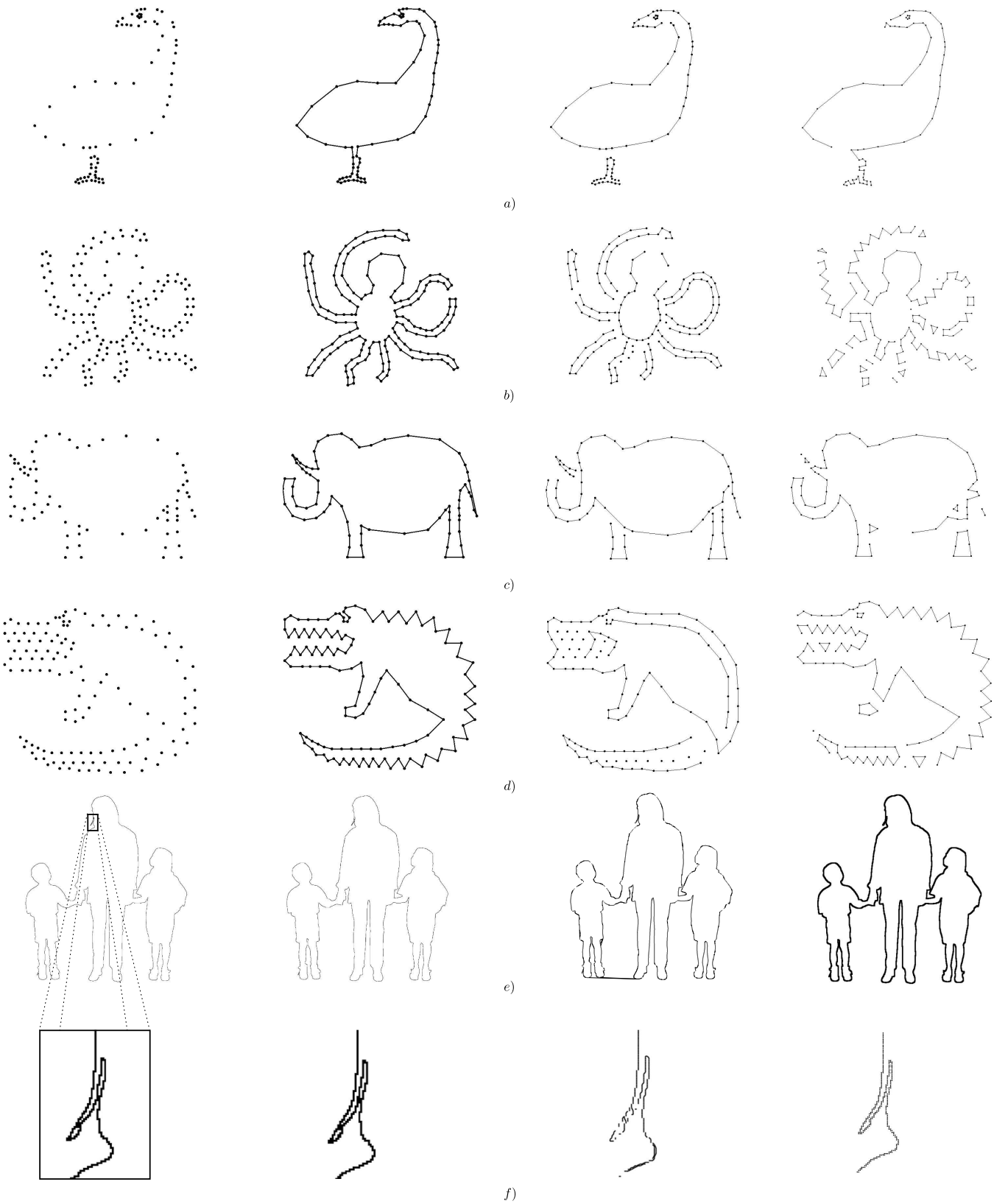

Figure 12: *Columns*: 1) Point set. 2) Our reconstruction. 3) *GathanG* with default parameters *minAngle* = 10 and *maxIter* = 4. 4) *DISCUR*. *Rows*: a) Goose (Amenta et al. [3]). b) Octopus: Close curves with sparse sampling. c) Crocodile: Sharp features. d) Elephant: non-uniform sampling and very sparsely sampled at corners. e) 10k points sampled on silhouette image: only our method produces a manifold. f) Detail of e.

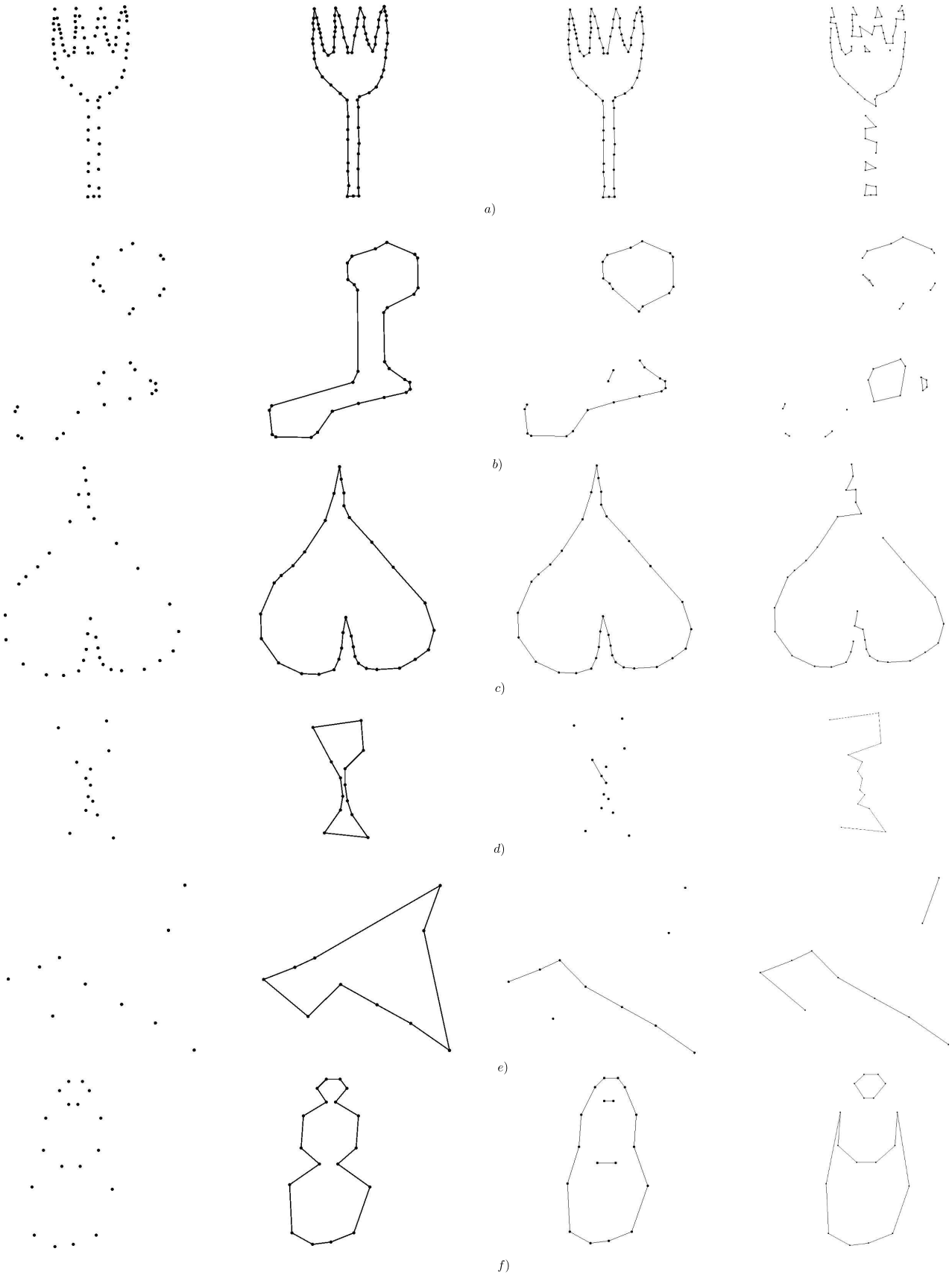

Figure 13: *Columns*: 1) Point set. 2) Our reconstruction. 3) *GathanG* with default parameters *minAngle* = 10 and *maxIter* = 4. 4) *DISCUR*. *Rows*: a) Tulip (Althaus and Mehlhorn [1]). b) Rail-joint (engineering part). c) Inverted heart (Zeng et al. [31]). d) Close curves. e) 10 random points. f) Three loops.

currently best known algorithms fail to produce desirable results. We focus on critical details as typical point sets are likely dense and thus not useful to properly illustrate the advantages of using our method. We compare our results with the previous best reconstruction method, *GathanG* from [14] (see Figure 12 and Figure 13) and also with *DISCUR* [31]. The latter have already compared their results with many of the other methods mentioned earlier. We have also compared with other approaches such as α-shapes or greedy algorithms (i.e. Boissonnat's sculpturing technique applied to $2D$ [9]), but we find that these methods settle rather too quickly into local minima and yield poor results for the non-uniform and sparsely sampled cases that we are illustrating in our comparative studies.

In our figures we have excluded the normal well sampled cases for which most methods including ours yield good results. We have mainly included point sets which sampling-oriented reconstruction algorithms have not been able to handle correctly and efficiently. This is clear from the examples. Further the results demonstrate that closely spaced shape segments, sharp corners, non-uniform and non-dense sampling are all handled very well by our algorithm. What is a bit surprising to us is the fact that the image silhouette data which is actually dense and uniform in most places could not be handled correctly by the other algorithms. Since we only had access to the executables of other algorithms, we can only show their results using screen shots of the output. Hence problems in connectivity are not always visible for highly dense point sets. We have noted that the total run times for all the algorithms are dominated by Delaunay graph computation time, and hence are all nearly the same. We do not feel that other comparisons such as actual run-times are illustrative. For example, for *DISCUR* only their binary is available to us. It strictly works with integer coordinates and the implementation is probably not optimized as it becomes very slow even for medium-sized point sets.

It could be argued that our requirement of a single closed shape (guided by the Gestalt law of *closure*) limits our algorithm's applicability. Where as *Crust*, *Gathan*, *DISCUR* and others are not. And even *GathanG*, although it is mainly targeted towards closed curves, handles open curves as well. However, the closed or open result from these algorithms has to be user specified or based on requiring the point configuration to satisfy a specified geometric condition, such as limit on point separation distance, abrupt curvature change, etc. In our algorithm, we could always apply the same conditions in a post-processing operation to remove offending edges and yield an open curve.

Also, if it is known that an open curve is to be generated, Steiner points can be appropriately introduced as user input, although deciding on the location for these Steiner points puts an additional burden on the user. On the other hand, it is important to restate that the imposition of Gestalt's law of *closure* lets our method realize far better results for sparsely sampled point sets.

Actually, we conjecture that our requirement of closed shape, restriction to edges in *DG*, classification and retention of *retained* segments and imposition of the manifold condition are what helps us significantly prune the solution space which otherwise has to be explored in full by TSP algorithms.

6.1. Complexity

As can be seen in the results section above, actual run time is nowhere near the worst case. However for theoretical completeness, we derive the worst case performance of this algorithm. The worst case is of course for data sets which have completely random distribution of points in 2D space. We first provide definitions of a few parameters needed in the complexity formulation.

- Global solutions s_g: denotes the number of calls to *apply and eliminate* procedure (see start of Section 5).
- Local solutions s_l: denotes the maximal number of solutions evaluated in any call to *inflate and select minimum loop* procedure (see Section 5.1).

Based on a point set with n vertices with l of them being leaf vertices, S_{min} can be reconstructed in:

$$O(max(s_g(n\log n), s_l)) \tag{1}$$

Below we derive the worst case complexity for the individual steps in the algorithm:

- Create *DG*, *EMST*, segment and classify: O(n *log* n).
- Create segment-candidate association table: In the worst case O(nl) for the traversal of all n edges of the tree for at most dl candidates ($d = 6$ is the average of incident edges for a vertex in *DG*). In practice for non-randomly distributed point sets the complexity is lower.
- *apply and prune*: O(n *log* n). It can be called $O(c^r)$ times, where c are the columns and r the rows of the segment-candidate table.
- *inflate and select minimum loop*: O((m *log* m)i!), where i is the maximum number of interior vertices in an inflatable segment.
- *remove cut edges*: O(n).

Table 4 shows the actual values of these parameters for the various point sets used in Figures 12 and 13.

As can be derived from the global complexity equation above, run-time increase is linearithmic with the number of points, provided that the global factor s_g is small w.r.t. n and local s_l is small w.r.t. $nlogn$. This is the case for all figures in Table 4, even for ones which are large, sparse and non-uniform at the same time. These factors become large only when the point set configuration is extreme in sparseness or non-uniformity of point spacing.

7. Conclusion and Future Work

We have presented a powerful and efficient algorithm which is capable of reconstructing aesthetically pleasing single closed interpolating 2D shape for an unorganised point set without requiring highly dense or uniform sampling. The results are better

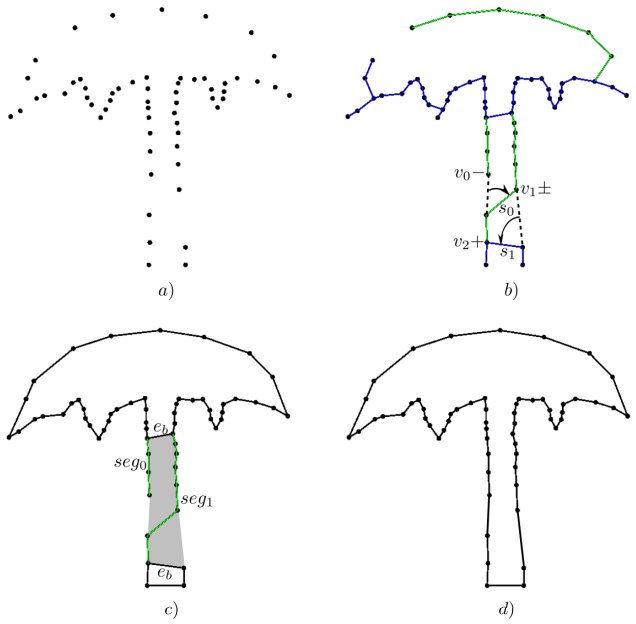

Figure 14: a) The *non-conforming* mushroom point set (Dey and Wenger [13]). b) $EMST$ with segments (blue: *non-retained*, green: *retained*) marked. The two *snaps* s_0, s_1 share an impact at v_1 but their e_+ are not contained in one segment, contrary to the required condition. c) The consequence is incomplete reconstruction at those *snaps*' affected segments: the two *retained* segments seg_0, seg_1 remain without candidates. We can extend our algorithm to consider the envelope of their combined point set, including the base edges e_b, shaded grey, and then carry out the *inflate* operation to yield the desired shape. d) S_{min}.

than those of all other known solutions for this 2D reconstruction problem. The actual run-time complexity statistics demonstrate that it is linearithmic for most practical cases.

Our algorithm does not employ any user-specified parameters and it does not require a sampling criterion. It does have a limitation for the point configuration which is mainly a safeguard to avoid very badly spaced points.

We also note that the number of leaf vertices in the $EMST$ of a point set correlates well with the running time required for the reconstruction of its S_{min}.

For another class of point sets which fall in the non-conforming category as they fail our required condition, i.e. as in the case seen in Figure 14 we show a potential extension.

Further, we believe that the primary methodology of starting with a skeleton shape and then transforming it into the final interpolating shape can be extended to 3D. We are presently making progress on the formulation of such an extension for 3D shape reconstruction, a much more difficult problem.

Acknowledgements

We are very grateful to the anonymous referees of the earlier versions for their insightful comments, which have helped immensely to improve the presentation in this version. This research was supported through an NSERC Discovery Grant, NCE Grand research grant, Engineering and Computer Science Faculty Research Grants and Canada Foundation for Innovation which supported equipment acquisition. We are grateful to Prof. T. K. Dey and Prof. Y. Zeng for providing us the code for their algorithms which we have used in our comparison study.

References

[1] Althaus, E., Mehlhorn, K., 2000. Tsp-based curve reconstruction in polynomial time. In: Proc. 11th ACM-SIAM SODA'00. pp. 686–695.

[2] Althaus, E., Mehlhorn, K., Schirra, S., 2000. Experiments on curve reconstruction. In: Proc. 2nd Workshop Algorithm Eng. Exper. pp. 103–114.

[3] Amenta, N., Bern, M., Eppstein, D., 1998. The crust and the β-skeleton: Combinatorial curve reconstruction. Graph. Models and Im. Proc 60 (2), 125–135.

[4] Applegate, D., Bixby, R., Chvtal, V., Cook, W., 2011 (accessed Mar 8, 2011). Concorde TSP Solver.
URL `http://www.tsp.gatech.edu/concorde.html`

[5] Arora, S., 1996. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. In: Journal of the ACM. pp. 2–11.

[6] Attali, D., 1997. r -regular shape reconstruction from unorganized points. In: Symposium on Computational Geometry. pp. 248–253.

[7] Attene, M., Spagnuolo, M., 2000. Automatic surface reconstruction from point sets in space. Computer Graphics Forum 19 (3), 457–465.

[8] Bernardini, F., Bajaj, C. L., 1997. Sampling and reconstructing manifolds using alpha-shapes. In: In Proc. 9th Canad. Conf. Comput. Geom. p. 0.

[9] Boissonnat, J.-D., 1984. Geometric structures for three-dimensional shape representation. ACM Trans. Graph. 3 (4), 266–286.

[10] Buchin, K., Knauer, C., Kriegel, K., Schulz, A., Seidel, R., 2007. On the Number of Cycles in Planar Graphs. Computing and Combinatorics, 97–107.

[11] Dey, T. K., Kumar, P., 1999. A simple provable algorithm for curve reconstruction. In: Proc. 10th ACM-SIAM SODA '99. pp. 893–894.

[12] Dey, T. K., Mehlhorn, K., Ramos, E. A., 1999. Curve reconstruction: Connecting dots with good reason. In Proc. 15th ACM Symp. Comp. Geom 15, 229–244.

[13] Dey, T. K., Wenger, R., 2001. Reconstructing curves with sharp corners. Computational Geometry 19 (2-3), 89 – 99.

[14] Dey, T. K., Wenger, R., 2002. Fast reconstruction of curves with sharp corners. Int. J. Comput. Geometry Appl. 12 (5), 353 – 400.

[15] Dillencourt, M. B., 1987. A non-hamiltonian, nondegenerate delaunay triangulation. Inf. Process. Lett. 25 (3), 149–151.

[16] Edelsbrunner, H., Kirkpatrick, D. G., Seidel, R., 1983. On the shape of a set of points in the plane. IEEE Trans. Inf. Theor. IT-29 (4), 551–559.

[17] Figueiredo, L. H. d., Mirandas Gomes, J. d., 1994. Computational morphology of curves. The Visual computer 11 (2), 105–112.

[18] Genoud, T., 1990. Etude du caractère hamiltonien de delaunays aléatoires, travail de semestre.

[19] Giesen, J., 1999. Curve reconstruction in arbitrary dimension and the traveling salesman problem. In: Proc. 8th DCGI '99. pp. 164–176.

[20] Glanvill, M., Broughan, K., 1997. Curve and surface reconstruction in r2 and r3. In: HPC-ASIA '97: Proceedings of the High-Performance Computing on the Information Superhighway, HPC-Asia '97. IEEE Computer Society, Washington, DC, USA, p. 395.

[21] Grigni, M., Koutsoupias, E., Papadimitriou, C., 1995. An approximation scheme for planar graph tsp. In: Proceedings of the 36th Annual Symposium on Foundations of Computer Science. FOCS '95. IEEE Computer Society, Washington, DC, USA, pp. 640–.

[22] Helsgaun, K., 2009. General k-opt submoves for the linkernighan tsp heuristic. Mathematical Programming Computation 1 (2-3), 119 – 163.

[23] Hoos, H. H., 2009. On the empirical scaling of run-time for finding optimal solutions to the traveling salesman problem. Technical Report 17, University of British Columbia, Department of Computer Science.

[24] Kirkpatrick, D. G., Radke, J. D., 1985. A framework for computational morphology. Computational Geometry, 217–248.

[25] Kruskal, Joseph B., J., 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. of the American Mathematical Society 7 (1), 48–50.

[26] Lee, I.-K., 1999. Curve reconstruction from unorganized points. Computer Aided Geometric Design 17 (2), 161 – 177.
URL `http://www.sciencedirect.com/science/article/pii/S0167839699`

[27] Mather, G., 2008. Foundations of perception. In: CAD. pp. 210–222.

[28] Nguyen, T. A., Zeng, Y., 2008. Vicur: A human-vision-based algorithm for curve reconstruction. Robotics and Computer-Integrated Manufacturing 24 (6), 824 – 834, fAIM 2007, 17th International Conference on Flexible Automation and Intelligent Manufacturing.

[29] Savelsbergh, M., Volgenant, T., 1985. Edge exchanges in the degree-constrained minimum spanning tree problem. Computers & Operations Research 12 (4), 341 – 348.

[30] Veltkamp, R. C., 1993. 3D computational morphology. Computer Graphics Forum (Eurographics '93) 12 (3), 115–127.

[31] Zeng, Y., Nguyen, T. A., Yan, B., Li, S., 2008. A distance-based parameter free algorithm for curve reconstruction. CAD 40 (2), 210–222.