The Intrinsic Shape of Point Clouds

Stefan Ohrhallinger

A Thesis
In the Department
of
Computer Science & Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy (Computer Science) at
Concordia University
Montréal, Québec, Canada

July 2012

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Stefan Ohrhallinger

Entitled: The Intrinsic Shape of Point Clouds

and submitted in partial fulfillment of the requirements for the degree of

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Y. Zeng	Chair
Dr. F. Samavati	External Examiner
Dr. A. Agarwal	External to Program
Dr. T. Bui	Examiner
Dr. P. Grogono	Examiner
Dr. S. Mudur	Thesis Supervisor

Approved by

Chair of Department or Graduate Program Director

Dean of Faculty

Abstract

The Intrinsic Shape of Point Clouds

Stefan Ohrhallinger, PhD
Concordia University, 2012

Given a point cloud, in the form of unorganized points, the problem of automatically *connecting the dots* to obtain an aesthetically pleasing and piecewise-linear closed interpolating boundary shape has been extensively researched for over three decades. In $\mathbb{R}^3$, it is even more complicated to find an aesthetic closed oriented surface. Most previous methods for shape reconstruction exclusively from coordinates work well only when the point spacing on the shape boundary is dense and locally uniform. The problem of shape construction from non-dense and locally non-uniformly spaced point sets is in our opinion not yet satisfactorily solved. Various extensions to earlier methods do not work that well and do not provide any performance guarantees either.

Our main thesis in this research is that a point set, even with non-dense and locally non-uniform spacing, has an intrinsic shape which optimizes in some way the Gestalt principles of form perception. This shape can be formally defined as the minimum of an energy function over all possible closed linear piece-wise interpolations of this point set. Further, while finding this optimal shape is NP-hard, it is possible to heuristically search for an acceptable approximation within reasonable time.

Our minimization objective is guided by Gestalt's laws of *Proximity*, *Good Continuity* and *Closure*. Minimizing curvature tends to satisfy proximity and good continuity. For computational simplification, we globally minimize the longest-edge-in-simplex, since it is intrinsic to a single facet and also a factor in mean curvature. And we require a closed shape.

Using such an intrinsic criterion permits the extraction of an approximate shape with a linearithmic algorithm as a simplicial complex, which we have named the *Minimum Boundary Complex*. Experiments show that it seems to be a very close approximation to the desired boundary shape and that it retains its genus. Further it can be constructed locally and can also handle sensor data with significant noise. Its quick construction is due to not being restricted by the manifold property, required in the boundary shape. Therefore it has many applications where a manifold shape is not necessary, e.g. visualization, shape retrieval, shadow mapping, and topological data analysis in higher dimensions. The definition of the Minimum Boundary Complex is our first major contribution.

Our next two contributions include new methods for constructing boundary shapes by transforming the boundary complex into a close approximation of the minimum boundary shape. These algorithms vary a topological constraint to first *inflate* the boundary complex to recover a manifold hull and then *sculpture* it to extract a *Minimum Boundary* approximation, which interpolates all the points. In the $\mathbb{R}^3$ method, we show how local minima can be avoided by covering holes in the hull. Finally, we apply a mesh fairing step to optimize mean curvature directly. We present results for shape construction in $\mathbb{R}^2$ and $\mathbb{R}^3$, which clearly demonstrate that our methods work better than the best performing earlier methods for non-dense and locally non-uniformly spaced point sets, while maintaining competitive linearithmic complexity.

Acknowledgements

This thesis would not have been possible without the contributions of many people. First, I sincerely thank my supervisor, Prof. Sudhir P. Mudur, for his excellent guidance and relentless support in all aspects of the creation of my thesis. I am grateful to the members of my supervisory committee, Prof. Anjali Agarwal, Prof. T.D. Bui and Prof. Peter Grogono, for insightful comments. This research was supported through an NSERC Discovery Grant, Engineering and Computer Science Faculty Research Grants, Grand NCE and Canada Foundation for Innovation which supported equipment acquisition. It was a pleasure to share the lab with my colleagues of the graphics group. Also I am grateful to Prof. Michael Wimmer who provided me in the final stage of writing with a space in his group in the Vienna University of Technology. A heart-felt thanks goes out to my wife, my parents, and my sisters, who have supported me in these years in various ways. Finally, it is due to the persistence of people like Richard Stallman that I was able to pursue this work using entirely open source software.

Contents

List of Figures

List of Tables

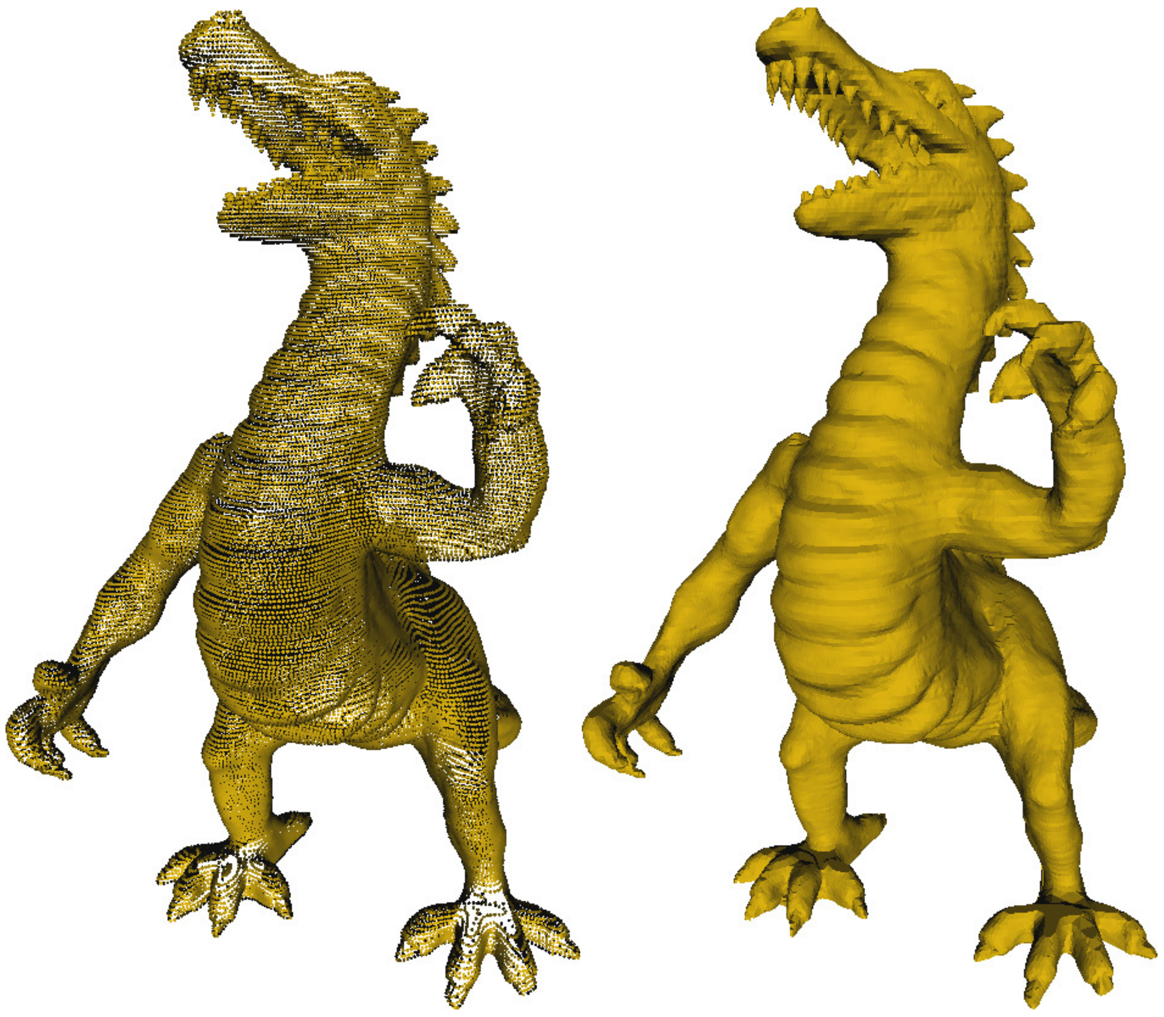
Chapter 1

Introduction

Determining the surface topology of an aesthetically pleasing shape from a large set of data points given just their coordinates finds many applications. In this chapter, we review the motivation for researching this problem, along with its challenges, and introduce the concept of intrinsic shape. The progress of the development of this idea and the solutions it has led to so far are documented in this thesis.

1.1 Motivation

Defining the piece-wise linear boundary shape for a solid object in $\mathbb{R}^2$ or $\mathbb{R}^3$, and possibly higher dimensions, for which only the surface point coordinates but nothing of its topology is known, is a difficult problem (see Figure 1.1). It is recognized as such, quite some time back, by Boissonnat [17] and understandably, has been the subject of a lot of continuous research over the last 3 decades. Specifically, in these days, given



(a) Unorganized point set (b) Reconstructed intrinsic shape

Figure 1.1: Searching for the intrinsic shape of a point set yields far superior results than what sampling-based methods can deliver.

the advances in sensor technology and computing capabilities, unorganized point sets on the boundary surfaces of objects are increasingly encountered. These result from simulations of objects with temporally incoherent topology or from sensing devices, like 3D scanners. Scanning techniques may supply the topology partly though not reliably, say in the form of estimated normals, contours, or overlaps from different perspectives. However, the need for solving the problem as a generic one is argued very well in Hoppe et al. [52].

In $\mathbb{R}^2$, shape construction is useful for reverse engineering of geometric models, outline reconstruction from feature points in medical image analysis, pattern recognition, etc. The closed boundary is essential for calculating various moments of the shape, a characteristic property which finds many applications.

In $\mathbb{R}^3$ it is even more complex to find a manifold shape in the form of an interpolating oriented surface, bounding a volume. Automating this shape construction is, in our opinion, a problem that has not yet been satisfactorily solved.

Fast construction of a close approximate shape with not necessarily a manifold boundary surface also has many applications. This could be useful where the overall shape is more important than the exact interpolating boundary surface, such as visualization, shadow mapping, extracting a smoothed surface, checking 3D scan quality, generating shape descriptors, topological data analysis etc.

1.2 Problem Statement

The problem researched in this thesis can now be stated as follows:

Given a set of points with just their coordinate data and no other shape information, efficiently derive the connectivity graph among all these points for defining a piecewise linear, interpolating, closed and oriented surface, satisfying the Gestalt laws of *proximity*, i.e., connectivity to nearby points on the surface, *good continuity*, i.e., avoidance of abrupt changes in curvature and *closure*, i.e., surface bounds a solid region, without assuming dense and locally uniform spacing of the points on the surface.

The point distribution in any given point set cannot be completely arbitrary. Point sets which are excluded as part of our problem domain are described next.

Visualization of the constructed shape is an important requirement in most applications. Therefore determining the connectivity which corresponds to the *good* shape in the given point set is most important, assuming the point distribution does possess this *good* shape. On the other side of that spectrum there exist sparse point distributions, which do not fulfill the Shannon-Nyquist theorem for sampling on the boundary of a desired shape. These require prior knowledge to construct and would appear more or less random otherwise. The topology of such a shape is also not robust with respect to minor point displacements. Therefore we exclude from our problem domain such extreme point distributions, which contain largely sparse sampling.

Presence of noise is an important theme in shape reconstruction and has led to the popularity of methods approximating the points. While we acknowledge its significance, we believe that in the context of shape reconstruction from sample points, noise filtering requires either the existence of a noise model, or an assumed surface. Since in our work, we do not assume any prior knowledge of the resulting shape other than that its boundary is closed, we treat de-noising as either a pre- or post-processing task. Therefore our focus in this research is on what we see as the basic research task, determining connectivity for the given point data as they are, i.e. interpolation, while nevertheless demonstrating that this works also for data sets with significant noise.

1.3 Challenges

Previous methods for reconstructing a surface exclusively from point coordinates often rely on a sampling criterion for the spacing of points on the surface. A sampling criterion for orientable surfaces was first formulated in Amenta et al. [6]. Such criteria give a guarantee of homeomorphy, meaning that a point set is interpolated locally by a unique surface. However, these criteria require that the resulting surface curvature be severely restricted so as to permit only a unique fit. In practice, sampling-based algorithms try to reconstruct surfaces for point sets outside these theoretical limits. They operate therefore without these guarantees and the resulting surfaces could have undesirable artifacts, say, holes, or do not interpolate many of the given points, because real-world point sets are not locally uniformly and densely spaced, and are often contaminated with noise.

Consequently, these techniques often fail to produce a manifold surface, or a close approximation to the shape, even with added corrective operations like hole-filling. Hence the first challenge is to devise a method for this surface construction which does not require dense and locally uniform sampling for the given point set, can deal with a reasonable amount of noise, and also is able to do that efficiently in terms of computational resources.

Another major challenge is to be able to construct point sets which are so large that they can only be stored out-of-core. This requires not only that the surface can be constructed locally (local as in sub-sets which can be contained in-memory), but that this construction is also deterministic, so that the locally constructed shapes always match at a global level to yield the same closed and oriented surface. It is clear that for point sets of such size their boundary can only be constructed by an algorithm of near-linear computational complexity.

Some kinds of data sets, i.e. those acquired from sensing devices, contain inherent noise. Hence another challenge is to be able to construct shapes even in the presence of noise in the coordinate data. Ideally, a shape construction method which interpolates the given points should be able to handle a certain amount of noise. It must be noted that the higher the noise level present in the data, the less it makes sense to interpolate the noisy samples. Methods for approximate fitting of a smoothed surface would be the better solution.

1.4 Thesis

Our main thesis in this research is that a point set, even with non-dense and locally non-uniform spacing, has an intrinsic shape which optimizes in some way the Gestalt principles of form perception. This shape can be formally defined as the minimum of an energy function over all possible closed linear piece-wise interpolations of this point set. Further, while finding this optimal shape is NP-hard, it is possible to heuristically search for an acceptable approximation within reasonable time.

1.5 Contributions

As mentioned above, finding an oriented boundary for an unorganized set of points in $\mathbb{R}^3$ is a difficult problem. Our research methodology was to approach this by first addressing the problem in $\mathbb{R}^2$ (less complex to handle) with the goal to extend our solution into higher dimensions.

1.5.1 Finding the Interpolating Boundary in $\mathbb{R}^2$ as Minimizing Problem

From all previously published work in this area, we know that shapes made up from edges in the Delaunay Graph (*DG*) of the given points yield good results due to the intrinsic properties of that graph, namely, maximizing angles and minimizing edge lengths. Our desired boundary shape must satisfy Gestalt laws of shape perception to the extent possible. We therefore derive a formal definition for the desired interpolating shape as a non-self intersecting and manifold closed boundary (*Closure*) made up from *DG* edges with minimal perimeter (*Proximity* and *Good Continuity*). Basically, this optimal shape is the minimum perimeter interpolating loop of *DG* edges, the *Minimum Boundary* (B_{min}). This minimization problem is NP-hard and is very closely related to the highly researched Traveling Salesman Problem (TSP). Its difference arises from our requirement of an aesthetic interpolating shape (not necessarily the exact minimum) and efficient construction. Hence we have restricted the solutions to edges in DG. We are also not interested in completely arbitrary point distributions, but only interested in that sub-domain of point sets in which the *good* shape exists. For this we have developed our own heuristic solution. We noticed that the Euclidean Minimum Spanning Tree ($EMST$) by its definition is very similar to B_{min}, with the single difference of relaxing a topological constraint. The number of edges incident at a vertex in B_{min} is exactly 2, while it is greater than or equal to 1 for the $EMST$. We exploit this similarity to locally partition the problem and define a transformation from $EMST$ to B_{min}, which guarantees finding the minimum boundary for a large class of point sets. The results show quality excelling currently known solutions.

This was initial work which led to the main findings described in more detail in this thesis. Appendix A includes more details of this initial work in the form of a paper published as Stefan Ohrhallinger and Sudhir Mudur: *Interpolating Unorganized 2D Point Clouds with Closed Shapes*, in Computer-Aided Design Journal, 2011.

1.5.2 Finding a Methodology which extends to higher Dimensions

This algorithm, while it is of linearithmic complexity ($O(n \log n)$) for a certain class of point distributions, does not guarantee the same performance for a number of other point distributions encountered in practice. Also it does not extend into $\mathbb{R}^3$. However, there are three major findings from this work which prompted most of the further research reported in this thesis. These findings are:

- Most given point sets have an intrinsic interpolating shape which can be defined as the one optimizing an objective function satisfying certain Gestalt laws of shape perception and is also aesthetically pleasing. This shape or connectivity graph is referred throughout this thesis as B_{min}.

- Computation of B_{min} is NP-hard, since finding it would require a search over all possible boundaries interpolating the given point set. There are related graphs which minimize the same objective, but may be faster to compute because of the slightly different topological constraint. An example in $\mathbb{R}^2$ is the Euclidean Minimum Spanning Tree ($EMST$). If suitable heuristics can be developed to transform the related graph into B_{min} or a close approximation, then we would have an efficient solution to our problem.

- It is important to develop an approach that extends to higher dimensions, particularly to $\mathbb{R}^3$.

Based on the above findings, and noting the similarity between both $EMST$ and B_{min} as minimizing graphs, we define the *Minimum Boundary Complex* (BC_{min}). It is another minimizing graph, varying from the two former graphs only in its topological constraint, the vertex degree. While computation of BC_{min} is also NP-hard, we found that a close approximation can be constructed quickly using a greedy algorithm, since we have relaxed its manifold constraint. We noticed that the graph computed by this greedy algorithm is actually also a close approximation to B_{min}, since the vertex degree varies only slightly. Next, we developed our heuristic method for transforming it into B_{min} or a close approximation as follows. We adapt a technique from $\mathbb{R}^3$, *sculpturing* by Boissonnat [17] and also introduce a dual to it, which we call *inflating*, and use them together to transform BC_{min} into B_{min} or an acceptable approximation. The entire algorithm is straight-forward, is of linearithmic time complexity and is also extensible into higher dimensions. Its results are of competitive quality with our previous algorithm, and it works well, especially for shapes with sharp corners, yielding results superior to those from currently known solutions for this problem. Source code is available on-line [65].

1.5.3 Constructing the Intrinsic Shape in $\mathbb{R}^3$

The ideas developed in the previous work extend well into 3D.

We first needed to develop the criterion for minimization of a boundary in higher dimensions. Minimizing curvature globally seemed to be a good objective considering that surfaces in nature also do that, for example, the surface of merged bubbles. Given that we are considering piece-wise linear surfaces, curvature at vertices (or edges) would depend on the many entities incident at the vertices (edges). For simplicity in developing a search strategy, we prefer that the criterion be computable independently for each entity in the surface. Based on experimentation, we decided to use the longest-edge-in-simplex (i.e. longest-edge-in-triangle for $\mathbb{R}^3$). Edge length is one of the two factors in the calculation of mean curvature for a piecewise linear surface. Our experiments confirm that this criterion works especially well in $\mathbb{R}^3$. Further, using such a criterion, which is intrinsic to a facet in the boundary, enables us to extend this approach into higher dimensions, if needed.

The next step is to define the relaxation of the topological constraint in $\mathbb{R}^3$. For this we define u-valence as the number of umbrellas incident at a vertex. In $\mathbb{R}^2$, an umbrella at a vertex is formed by its pair of incident edges. An umbrella at a vertex v in $\mathbb{R}^3$ is defined as the set of triangles incident at v such that all edges incident to v have exactly two incident triangles. Clearly, B_{min} being manifold, it has u-valence $= 1$ at every vertex. We relax this constraint and permit u-valence ≥ 1, to construct BC_{min}. A greedy algorithm computes an approximation which, as we see again from extensive experiments, is a close approximation to the boundary shape.

Our final step is to extend the transformation algorithm to $\mathbb{R}^3$. We note that from our previously introduced algorithm, the two main steps of inflating and sculpturing extend very well directly. But since the topology of the boundary complex in $\mathbb{R}^3$ is more complex, we need to enhance our method. After the inflating step we find that the boundary triangles form a thin-thick triangulation, which we call as the hull. We adapt sculpturing to work from both inside and outside of the thin-thick hull. Directly inflating the boundary complex, which results from the greedy algorithm, however tends to terminate quite often at local minima. This manifests in the form of holes in the hull. Hence, prior to the inflate step, we need to detect and cover these hull-holes. We describe the heuristic method we have developed for that, which is based on careful topological analysis of the boundary complex. Lastly, we apply a mesh fairing step, to directly minimize global mean curvature. The results from our method are far superior to those from previous solutions, especially for non-dense and locally non-uniform point sets. Our method is also efficient as it completes in expected $O(n \log n)$ time.

A concise overview was presented as a poster and published in the Poster Proceedings of Eurographics 2012 in Cagliari, Italy. It won the Best Poster Award.

1.5.4 Other Applications of the Boundary Complex

The boundary complex is a very interesting constrained simplicial complex and using it to extract an orientable interpolating boundary is just one application. Its ease of construction makes it a much more powerful shape characteristic. We analyze its properties, note its tolerance to noise and capability for local construction and present quick visualization as another application, since it retains all the important features, including the genus. Outside the field of visualization we believe that it can contribute to shape descriptor construction of point sets, collision detection, topological data analysis in higher dimensions and combinatorial optimization.

1.6 Thesis Organization

First we give relevant background in Chapter 2 consisting of related work in Section 2.1 for $\mathbb{R}^2$ and in Section 2.2 for $\mathbb{R}^3$. The terminology used in the rest of this thesis is listed in Section 2.3.

In Chapter 3, the development of the Minimum Boundary Complex in $\mathbb{R}^2$ is presented in Section 3.1. Then, in Section 3.2, we extend it to $\mathbb{R}^3$. Finally, we present relevant properties of the Minimum Boundary Complex in Section 3.3.

Its main application in boundary construction of the intrinsic shape is discussed in detail in Chapter 4 for $\mathbb{R}^2$ and in Chapter 5 for $\mathbb{R}^3$.

We present various experimental results from our implementation of these methods in Chapter 6, both for $\mathbb{R}^2$ (Section 6.1) and for $\mathbb{R}^3$ (Section 6.2).

In Chapter 7 we present important extensions and in Chapter 8 we give our conclusions and potential for future work.

Chapter 2

Background and Related Work

In this chapter we present work related to our thesis, structured in two-dimensional and three-dimensional approaches, give an overview of our approach and introduce basic terminology used throughout this thesis.

Shape construction has been well researched in both $\mathbb{R}^2$ and $\mathbb{R}^3$. Many approaches, especially those based on the Delaunay graph, extend well between those two spaces.

2.1 Related Work in $\mathbb{R}^2$

In the literature we find two major approaches. One is to cast this problem as $\mathbb{R}^2$ shape reconstruction by considering the points as samples on a known $\mathbb{R}^2$ object. This then makes it possible for algorithms to work for point sets satisfying specified sampling criteria. Usually these criteria impose quite strict conditions with regard to point spacing properties, requiring high density, uniformity and smoothness. The second approach is to view this problem as a global search through all possible solutions. Below, we provide a comprehensive review of previous work using these two approaches.

2.1.1 Local Sampling Condition Approach

Algorithms using this approach and discussed further below, connect the points 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.

α-shapes, introduced by Edelsbrunner et al. [36] and extended by Bernardini and Bajaj [15], minimum spanning tree-based methods by Figueiredo and Gomes [39], the β-skeleton by Kirkpatrick and Radke [57], the γ-neighborhood graph by Veltkamp [73] and r-regular shapes from Attali [12] are among the early methods which worked only on smooth and uniformly sampled point sets. 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.

Amenta et al. [8] 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 by Dey and Kumar [30] and Dey et al. [31] are however quite restrictive in theory and difficult to ensure in practice. The *Gathan* algorithm from Dey et al. [32] which is based on their work handles sharp corners, but without performance guarantees, and does not take aesthetic aspects into consideration. In spite of this, it provides in our opinion the best sampling-oriented solution for this $\mathbb{R}^2$ shape reconstruction problem.

Zeng et al. use in [75] the two properties of proximity and smoothness derived from Gestalt laws but still require rather dense sampling in sharp corners. Some improvements on these aspects have been made in Nguyen and Zeng [64], but they rely very much on several user-tuned parameters.

A fundamental disadvantage of using a local criterion is that one cannot guarantee reconstruction of a closed and manifold shape, the way our method does. In fact, our observation from the many experiments we have conducted is that enforcing the Gestalt law of closure actually yields more pleasing shapes. This can be seen later e.g. in Figure 6.4 in Chapter 6 which shows a number of such cases. And if one indeed desires to get an open shape, then an openness condition, such as large distance between points, very sharp turns and other such conditions, can be applied to the resulting closed curve to make it open.

2.1.2 Global Search Approach

A first attempt using a global search approach is the one presented by Glanvill and Broughan [44]. They construct spanning Voronoi trees and select 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 [42] that the solution for the Euclidean traveling salesman problem ($ETSP$), called a tour, can reconstruct the shape for sufficiently dense sampling. Later Althaus and Mehlhorn show in [4] that such a tour reconstructs aesthetic shapes also for non-uniform sampling with a specified density. They show that this NP-hard problem can be solved in polynomial time if the point set is restricted to a certain sampling criterion. For unrestricted sets Arora [11] gives a $(1 + 1/c)$-approximation to the optimal ETSP tour in $O(n(\log n)^{O(c)})$ time complexity. But these approximations fail to guarantee an aesthetically pleasing solution as per our requirement. Our experiments showed that non-optimal solutions include polygons with crossed edges, violating our requirement of non-intersection.

In the work by Althaus et al. [5], the exact TSP based solution is compared with $Crust$-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 the method in Giesen [42], these methods do not require user-specified parameters. Unfortunately, finding the exact solution using a naive TSP solver takes unreasonable time $\mathrm{O}(2^n)$ even for small P. The *concorde* exact TSP solver [10] scales sub-exponentially and can take hundreds of CPU-years for medium-sized point sets, its complexity is discussed in detail here [51].

While, in principle, a TSP solution constrained to DG would yield B_{min}, we consider the TSP as too generic to be applicable for the problem we have stated. Our focus is on an algorithm for quick construction of an interpolating and aesthetic closed shape, and it only needs to work on point sets that are reasonably distributed and contain humanly recognizable shape boundaries.

In the algorithm included in Appendix A [66] it is shown that for a certain class of point sets, there exists a relation between minimum perimeter polygon in DG and the Euclidean minimum spanning tree ($EMST$) of P. This relation is characterized by well-defined edge exchange operations. While this algorithm gives very good results for sharp corners, it cannot guarantee linearithmic complexity since in some cases a global search of the solution space may be required. The very important contribution there is in the approach to formulate curve reconstruction as a minimization problem, by relating to properties of the Gestalt laws for aesthetic shape.

All of the previously discussed work assumes points sampled on a noise-free curve. Cheng et al. [23] discuss curve reconstruction in the presence of a noise model, which they define artificially, and give reconstruction guarantees in terms of probability. Mehra et al. [61] adapt the point-based visibility method proposed in Katz et al. [55] for reconstruction of noisy samples, but their method is quite ad-hoc. It does not fulfill its aim of constructing closed shapes and also their removal of outliers seems somewhat arbitrary.

2.2 Related Work in $\mathbb{R}^3$

Surface reconstruction methods can be classified into two distinct groups by their guarantees with respect to closeness of the given points to the constructed surface. *Interpolating* methods try to fit a surface, e.g. a piece-wise linear one, through the points and may filter outliers to achieve that goal well. *Approximating* methods fit an implicit surface within a threshold distance to the points by creating a signed distance function and then extract a polygonal mesh. Consequently their resulting boundaries will not interpolate the input points, but this enables them to deal better with noisy data. Some of the methods give further topological guarantees such as homeomorphy, genus and water-tightness for the constructed surface.

2.2.1 Interpolating a Point Set

α-shapes

Based on the three-dimensional Delaunay triangulation of the points, the concept of α-*shapes* was extended into $\mathbb{R}^3$ by Edelsbrunner and Mücke [38]. This formulation requires a globally uniform parameter, which leads to a tradeoff between loss of detail and hole filling. Edelsbrunner's work is further extended in Veltkamp [74] with a γ-neighborhood graph that adapts locally to variable point density, however the results are not convincing. Still, the fact that many point sets are determined by mostly (although rarely exclusively) locally uniform sampling has motivated the extraction of such a guarantee from point sets and to apply local reconstruction to avoid the more expensive construction of the global Delaunay graph.

Advancing-front Algorithms

Bernardini et al. [16] introduced an *advancing-front* algorithm based on α-shapes, which for the reasons listed above fails for non-uniformly spaced point sets. Cohen-Steiner and Da [25] extended it to locally non-uniform sampling, but it still does not interpolate all points. Since such algorithms depend on a seed-triangle, their results are also not deterministic.

Local Tangent Plane Estimation

Boissonnat [17] estimates at each given point a *local tangent plane* using nearby points and then determines the local neighborhood by projecting those points on that plane. He assigns points as neighbors based on an angle criterion. Gopi et al. [45] derive natural neighbors from the Delaunay graph projected on such a plane. For both methods, a plane is fit using the k-nearest neighbors. Using a single value for k globally has the disadvantage that for many points this value will either be too small or too large to give suitable local support. Where these neighbors are distributed an-isotropically, the resulting normal will not be representative. The more recent method of Dumitriu et al. [34] based on theoretical guarantees of Funke and Ramos [40] for uniformly sampled point sets suffers from the same problem. They require prior extraction of a dense uniformly sampled point set whose quality depends again on estimation of the underlying surface.

Umbrella Matching

Adamy et al. [1] create *umbrellas* locally at the vertices from the set of Gabriel triangles. They use topological post-processing to match these umbrellas and fill holes by solving a system of integer linear inequalities, but this becomes very slow for larger sparsely spaced sub-sets of points. The surface is guaranteed to be watertight, but no aspect of the surface is optimized and these inequalities lead to undesired disconnection of some of the surface components. Kós [58] creates umbrellas for a selected sub-set of points and then re-inserts unprocessed points. Contrary to our requirement of the water-tightness guarantee, his work is targeted to include unorientable surfaces.

Shape from Sculpturing

Boissonnat [17] introduced, in a second approach in that paper, the technique of *sculpturing*. He mentions a proof that any polyhedron of genus 0 can be extracted from the convex hull by removing tetrahedra in the Delaunay graph, based on certain rules. Since this is also of combinatorial complexity, he proposed a greedy algorithm which removes such tetrahedra from the outside of the convex hull of the point set, sorted by an intrinsic criterion. However, this process ends up quickly in local minima and may therefore miss interpolating many of the given points. Further, the resulting shape is restricted to a genus of 0. Attene and Spagnuolo [13] constrain sculpturing such that Gabriel triangles are not removed from the boundary. They detect and create holes in the object (genus > 0) where edges in the Euclidean Minimum Spanning Tree span non-neighbor vertices on its boundary, but the object can become hollowed out at under-sampled regions. Chaine [22] uses surface tension as criterion for sculpturing, but does not give any guarantees for the resulting surface. Allègre et al. [2] present an out-of-core extension of that algorithm by first decimating the point set and then sub-sampling to a criterion. Given the way the sculpturing operation is defined, all the above methods are global and hence do not scale well to very large point sets. Being able to limit the sculpturing operation and more generally the surface construction operation to a local subset of points is therefore very important.

Homeomorphic Guarantees

Amenta and Bern [7] were the first to prove homeomorphic surface reconstruction (their *Crust* algorithm), given a sampling criterion, although the resulting surface may contain many slivers and is therefore not manifold. They however do guarantee such a surface for an ϵ-dense sampling in proportion to medial axis distance, where ϵ=0.06. This sampling criterion is extremely stringent, permitting only very blunt dihedral angles (an averaged $\approx 166°$) at the edges, in order to be able to fit a surface uniquely. Except in some parts of the surface, these criteria are not met by point sets usually encountered in practice. The actual surface triangles are extracted by locally filtering the Delaunay graph. Following this work, Amenta et al. [9] present the *Power Crust* algorithm which reconstructs under-sampled regions better, but introduces many additional points. Amenta et al. [6] simplified the original *Crust* to the *Cocone* algorithm. Later, Dey and Goswami [28] extended it to their *TightCocone* algorithm which fills holes, provided the under-sampling is local. In a follow up paper [29], they propose some filtering of the restricted Delaunay graph to remove noisy points, which are not clearly oriented outside or inside its envelope. A recent extension by Dey et al. [27] focused on localizing reconstruction while maintaining its theoretical guarantees, which enables parallel and out-of-core handling of large point sets.

Other Delaunay-Based Methods

Both Giesen and John [43] and Edelsbrunner [35] used *flow*, based on critical points of a distance function, to restrict the Delaunay complex. However they give no guarantee other than that the resulting surface will be water-tight. Guibas and Oudot [46] used the witness complex to extract an interpolating surface from noisy point sets.

Optimization Approaches

In one of the first attempts to use an *optimization* approach Petitjean and Boyer [67] use an initial set of Gabriel triangles and then select triangles by minimal circumradius to extract a manifold. Labatut et al. [59] formulate the problem as a graph cut by weighting approximate visibility and minimizing the longest-to-shortest ratio of edges in triangles, but they require carefully optimized parameters for each point set. Hiyoshi [50] proposes a global algorithm which minimizes a criterion for the surface and analyzes different heuristics. He extends the edge length criterion from $\mathbb{R}^2$ to correspond to area or circumradius of triangles in $\mathbb{R}^3$. But his heuristic of inserting triangles into a set to fulfill the constraint of ≤ 2 triangles per edge, as well as its proposed dual, get very easily stuck in local minima, yielding sub-optimal results.

Concluding remarks

While interpolating algorithms, which are based on a global structure (Delaunay graph), yield in practice the best results, they are slower and not easy to implement in parallel. Fast advancing-front and umbrella-matching algorithms fail in turn for locally non-uniform point sets, since closedness cannot be guaranteed from local analysis alone (see the Gestalt law of *Closure*). Based on our experiments we judge the above-mentioned TightCocone [28] algorithm to be a good solution for interpolating point sets which include non-dense, locally non-uniformly spaced points. There are no recent significant extensions of this work and the more lately proposed minimization approaches are not competitive in our view.

2.2.2 Fitting an Approximate Surface to a Point Set

Much more recent work has been done on approximation of surfaces, given that sensor data is often noisy and contains outliers. We find however that it deals with a practical aspect of the shape construction problem and that the basic research problem still remains one of interpolation. Approximating methods either require oriented normals, which are unreliable if coming from sensing devices, or they try to estimate them, with variable success, since this depends on the choice of neighborhood.

Methods Assuming Normals

Methods assuming *existing normals* were for example proposed by Boissonnat and Cazals [18]. They use *natural neighbors* interpolation, but the resulting surface is often not manifold. Carr et al. [21] describe a method in which they apply radial basis functions (RBF). Their method yields good results but is rather slow. Mederos et al. [60] use curvature-variant vertex *clustering* to create a representative point set and then do an advancing-front triangulation. Kazhdan et al. [56] introduced *Poisson* surface reconstruction, which handles noise well. Bolitho et al. extend it first to a streaming [19] and then to a parallel approach [20].

Methods Estimating Normals

In yet another seminal contribution in this area, Hoppe et al. [52] *estimate the normals* and propagate their orientation so that they become globally consistent. Alliez et al. [3] search k-nearest neighbors, until an *anisotropy threshold* is reached or k reaches a global maximum, to fit the implicit surface to a tensor field. Samozino et al. [69] extended the work of [21] by *clustering* points in the Voronoi diagram to reduce the complexity of RBF. Hornung and Kobbelt [53] *dilate* the surface crust by volumetric expansion and create a confidence-weighted graph for which a minimal cut yields a watertight surface, however its resolution is restricted by that grid. Mullen et al. [63] eliminated this restriction by defining an ϵ-*band* enveloping the surface with ϵ globally estimated from the Delaunay graph. Shalom et al. [71] construct visibility cones in order to use global visibility information to improve fitting holes with RBF. Avron et al. [14] utilize sparsity, using the theory of compressed sensing given in Mishali and Eldar [62], to implement a l_1-based method which avoids the over-smoothing of the l_2-norm and therefore can reconstruct surfaces with sharp features.

Concluding remarks

Approximating a surface is difficult to do well, locally. Also, unreliable normals and the fact that the surface is not required to pass through the given points might work better in the presence of noise, but generally does not yield as good a shape as interpolating methods do.

2.3 Terminology and Notations

2.3.1 Definitions

Let P in d-dimensional Euclidean space ($\mathbb{R}^d$), with $d = 2$ or $d = 3$, denote the given unorganized *Point set* for which an aesthetic closed, non-intersecting and manifold interpolating piece-wise linear boundary B has to be constructed. P is assumed to belong to the boundary surface of some closed object in $\mathbb{R}^d$.

Let B_{min} refer to the $B_i \in \{B\}$ for a given P such that it minimizes an intrinsic criterion of B_i.

Let $DG(P)$ denote the *Delaunay graph* of the point set P. $DG(P)$ is by definition a simplicial complex, denoted generically as $\mathbb{C}$. For the various types of elements in $\mathbb{C}$, we will use the following naming convention: v for vertices, e for edges, t for triangles, q for tetrahedra, f for facets, s for d-simplices in $\mathbb{R}^d$ and x for generic entities in that list.

Euclidean minimum spanning tree $EMST$ is the tree spanning all points in P in $\mathbb{R}^2$ such that the sum of its edge lengths is the minimum. $EMST \subset DG$ (Jaromczyk and Toussaint [54]).

A facet is a boundary primitive, therefore in $\mathbb{R}^2$ an edge and in $\mathbb{R}^3$ a triangle. We define a facet f in $\mathbb{C}$ as a *boundary facet* if f has at most 1 incident d-simplex s in $\mathbb{C}$.

We name the sub-set of boundary facets for a connected set of facets F as its *hull* $H(F)$. This implies that the facets in the remaining sub-set $F \setminus H(F)$ are all *interior*

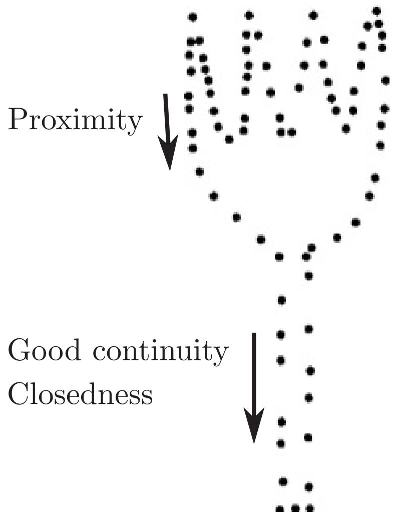


Figure 2.1: Three *Gestalt* laws illustrated: *Proximity* connects close points. *Good continuity* minimizes curvature. *Closedness* keeps the shape water-tight.

to the hull. We call a vertex v_i in $H(F)$ as *manifold* if v_i is only visited once when traversing the hull. We further call a hull H as *manifold* if all vertices v in H are manifold.

$\|n\|$ denotes the Euclidean norm in $\mathbb{R}^d$ for a vector n.

2.4 Our Proposed Approach

Despite the wide range of methods developed so far, we believe that the fundamental problem of determining the shape of unorganized point sets has not been addressed that well. This is confirmed by the unwanted artifacts and the lack of surface construction guarantees in these methods. We believe that most point sets have an intrinsic shape which optimizes in some way the Gestalt principles of form perception. This shape is well defined and can be formulated using an optimization metric based on

Gestalt laws of visual perception (see Figure 2.1). Further, our various experiments with point sets in 2D and 3D have confirmed this very strongly. Most previous surface reconstruction and rendering algorithms do an estimation of the local surface and can give some theoretical guarantees, but only under extremely stringent conditions required of the point set configuration.

In our approach, we propose that there is a *good* surface, well-defined by minimizing an energy functional, which is based on aesthetics. We propose further that while we can not guarantee its exact construction always, we can approximate it well, depending on a time-budget. This *good* surface exists in a much larger sub-class of point sets with not too extreme spacing, basically point sets bounding aesthetically pleasing shapes. By assuming such a good surface, every given point set has a *shape* in the form of the topology of that surface. We can then search for that shape or an acceptable approximation, instead of estimating something that we cannot give guarantees on.

Chapter 3

The Minimum Boundary Complex

In this chapter we present our first major contribution, the Minimum Boundary Complex (BC_{min}), a subgraph of the Delaunay graph of the given point set. BC_{min} closely approximates the minimum boundary shape (B_{min}) and its approximation can be constructed fast using a greedy algorithm. It is also robust to noise. We first define it in $\mathbb{R}^2$ and $\mathbb{R}^3$, and are able to generalize it to $\mathbb{R}^d$. Further, we show that it can also be constructed locally.

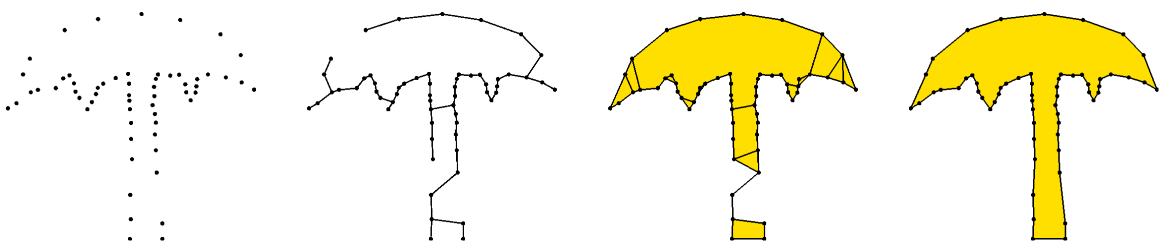


Figure 3.1: Comparison of spanning graphs with upper constraints on vertex degree c: a) Point set from Dey and Wenger [32]. b) $EMST$ ($c \geq 1$). c) BC_{min} ($c \geq 2$) with the interior of its manifold hull shaded. d) B_{min} ($c = 2$) with interior shaded.

This chapter is organized as follows. In Section 3.1 we describe how we obtain a formalized definition for an aesthetic shape boundary based on Gestalt laws. We provide this definition first for $\mathbb{R}^2$. Then we give the inspiration for developing the Minimum Boundary Complex from this concept. By finding a suitable surface criterion to minimize, in Section 3.2 we extend these definitions of both the Minimum Boundary and the Minimum Boundary Complex into $\mathbb{R}^3$. Finally, we generalize their definitions into $\mathbb{R}^d$ and describe some properties of the Minimum Boundary Complex in Section 3.3.

3.1 Shape Boundaries in $\mathbb{R}^2$

Althaus and Mehlhorn showed in [5] that the travelling salesman problem (TSP) solves the curve reconstruction problem for non-uniformly sampled smooth curves under the assumption of a sampling condition similar to Amenta et al. [8]. Our own experiments in [66] have shown that minimal length polygonizations of point sets yields very good results and the shapes produced are aesthetically pleasing to human viewers. This led us to the idea that every given point set, unless it is random or extremely non-uniform, has an intrinsic shape which minimizes an energy function over all possible closed linear piece-wise interpolations of the point set. And it this idea that is pursued throughout the research reported in this thesis.

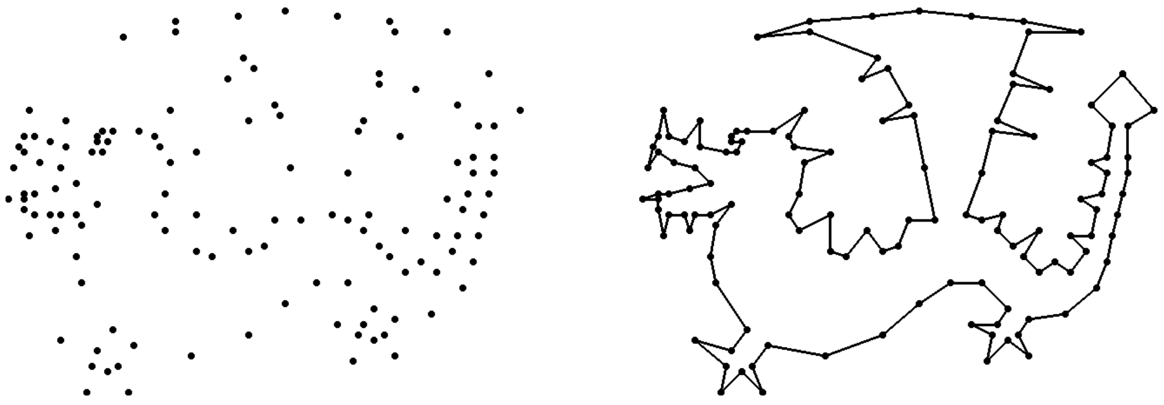


Figure 3.2: a) Unorganized point set P. b) A pleasing shape connecting P, which is a closed, non-intersecting and interpolating curve with minimal boundary length, B_{min}.

3.1.1 The Minimum Boundary (B_{min})

In $\mathbb{R}^2$, we select edge length $\lambda(e) = \|e\|$ as the criterion for the Minimum Boundary (B_{min}). This criterion is used for creating an aesthetic boundary shape of a point set, by minimizing its total over B. $\lambda(e)$ relates directly to the Gestalt law of *Proximity*. *Closure* is fulfilled as well by the constraint of minimum two incident edges for every interpolated point.

Good continuity is not always strictly followed, since it can conflict with closure (for an example see Figure 3.2 where the uppermost point of the tail and the rightmost point of the wing are not connected as one would expect following good continuity). We therefore restrict the edges in B_{min} to those in the Delaunay graph (DG). Then the law of *Good continuity* is fulfilled implicitly, since restricting the edges to DG maximizes angles between edges and selects small edges, which in turn correlates well to low curvature.

Finding B_{min} by minimizing some criterion in its geometric primitives is a non-polynomial (NP)-hard problem.

3.1.2 The Minimum Boundary Complex (BC_{min})

We observed that for point sets, except in those which are random or extremely non-uniformly spaced, the $EMST$ graph characterizes the above-mentioned intrinsic boundary shape rather well. Contrary to B_{min}, it can be constructed in $O(n \log n)$ time. However, there are leaf vertices in $EMST$ and there are no leaf vertices in the interpolating, closed manifold curve B_{min}.

Based on this observation we formulate an extension to the $EMST$ by requiring that each vertex must have at least two incident edges. The manifold hull of the resulting graph approximates the shape boundary much better (see Figure 3.1c). It shares a large sub-set of edges with B_{min}, because the only change in definition is a topological constraint, its vertex degree, which is slightly different. Since it is not a tree, but a simplicial complex consisting of edges and triangles, we have chosen to name it the *minimum boundary complex* (BC_{min}).

Definition 1 *The boundary complex $BC \subseteq DG$ in $\mathbb{R}^2$ is defined as a graph $G = (V, E)$ spanning P such that each vertex v_i in BC has ≥ 2 incident edges in BC. Note that DG is a BC.*

For any given set of points, BC_{min} is the BC satisfying the following objective:

$$BC_{min} = \sum_{e_i}^{E} \lambda(e_i) \rightarrow \min \tag{3.1}$$

An approximation of BC_{min} can be constructed using a greedy algorithm in $O(n \log n)$ time, which we denote as BC_0 (see Algorithm 1).

Extensive experiments using this algorithm show that the manifold hull of BC_0 closely resembles the boundary shape, i.e., B_{min} or a close approximation, since many edges of these two graphs overlap (see Figure 3.1). We will look more closely at the properties of BC_0, after we have presented its extension in $\mathbb{R}^3$.

```
Input: $P, DG$
Output: $BC_0$
$BC_0 = \{\}$;
$PQ$ := priority-queue of $e_i$ in $DG$, sorted by $\lambda(e_i)$;
while ($BC_0 \neq$ connected component) $\vee$ ($\exists v_i$ in $P$ with $< 2$ incident edges in $BC_0$)
do
    Remove first $e_i$ from $PQ$;
    if ($e_i$ connects components in $BC_0$) $\vee$ ($e_i$ contains leaf vertex in $BC_0$) then
        Insert $e_i$ into $BC_0$;
    end
end
```

Algorithm 1: Construction of the BC_0-complex in $\mathbb{R}^2$

Lemma 1 *Given a point set P with n points and its Delaunay graph $DG(P)$, Algorithm 1 constructs BC_0 in $O(n \log n)$ time.*

Proof 1 *Creating PQ inserts at most the $O(n)$ edges of DG, with each insert operation being $O(n \log n)$. The while loop is executed $O(n)$ times. Testing for and keeping track of connectedness is done via a disjoint set. Its operations are an amortized*

$O(\alpha(n))$, with $\alpha(n)$ as inverse of the Ackermann function. Total complexity of the algorithm is therefore $O(n \log n)$.

Note that Algorithm 1 may not have a unique result if the DG contains edges of equal length. However, this is not a problem, as based on perception either result will be equally valid.

3.2 Shape Boundaries in $\mathbb{R}^3$

3.2.1 Choosing a Suitable Criterion for B_{min}

As shown above, in $\mathbb{R}^2$ we use edge length as criterion to minimize a piece-wise linear curve, as it corresponds well to Gestalt laws. This needs to be extended for a triangulated boundary in $\mathbb{R}^3$. We prefer that this criterion be contained in a single primitive (point, edge or triangle) to avoid combinatorial dependencies in the design of any optimal search algorithm.

Different triangle measures in $\mathbb{R}^3$ could be considered as extensions of this $\mathbb{R}^2$ minimization criterion. These include area, circumradius, inradius, longest side, perimeter, aspect ratio, and possibly others.

We have evaluated these measures by plugging them into Algorithm 2, the $\mathbb{R}^3$ version of Algorithm 1, which is presented later in Sub-section 3.2.3 (see Figure 3.3). Using area or inradius produces many long, thin triangles. The circumradius gives in general good results, but may also include long, thin triangles because it avoids small

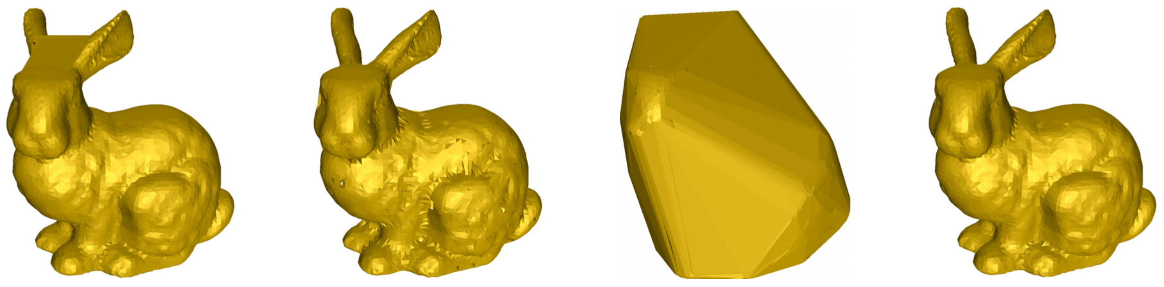

(a) Circumradius (b) Area (c) Aspect ratio (d) Longest edge $\lambda(t)$

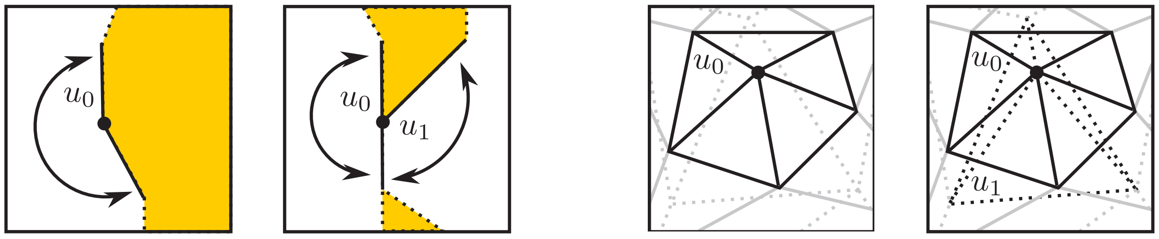
Figure 3.3: Comparing surface minimization criteria.

triangles with near coplanar vertices. Minimizing the *longest edge* in a triangle seems to work the best (see Figure 3.3d). We relate this to the fact that short edges also tend to minimize curvature, as discussed next.

3.2.2 Minimizing Curvature gives Good Shape

Hildebrandt and Polthier [49] define *mean curvature* for edges in polyhedral surfaces as $H_e = 2\|e\| cos\frac{\theta_e}{2}$, where θ_e is the dihedral angle at edge e. Intuitively, this corresponds to the amount of work for bending a metal sheet, with the two factors being largeness of the sheet and the angle to bend. Subsequently, we define B_{min} as the B for which $\sum_{e_i}^{B} H_e \to min$. However, H_e can not be evaluated independently for a single triangle.

Edge length is one of the two factors in the definition of H_e. Hence selecting triangles with short edges still yields small $\sum H_e$ for the entire triangulation since it also seems to increase the obtuseness of dihedral angles. This relates well to both of the Gestalt laws of *Proximity* and *Good Continuity*, (*Closure* is satisfied by requiring a closed surface). Accordingly we define our measurable criterion for an individual



(a) u=1 (b) u=2 (c) u=1 (d) u=2

Figure 3.4: The u-valence is the number of *umbrellas* at a vertex: a, b) In 2D the interior of boundary is shaded yellow. c, d): In 3D.

triangle t as:

$$\lambda(t) = max(\|e_i\|int) \tag{3.2}$$

Like in $\mathbb{R}^2$, we restrict boundary triangles to the Delaunay graph, as it maximizes their minimal angle and therefore tends to yield both short edges and obtuse dihedral angles between triangles.

3.2.3 The Minimum Boundary Complex (BC_{min}) in $\mathbb{R}^3$

In $\mathbb{R}^3$, an *umbrella* $U(v)$ for a vertex v is any set of triangles incident to v such that each edge in $U(v)$ incident to v is contained by exactly two triangles in $U(v)$. We shall use the term *u-valence* to denote the number of umbrellas incident at a vertex (see Figure 3.4). Different umbrellas at a vertex can overlap partially.

We have shown above in Subsection 3.1.2 that in $\mathbb{R}^2$, the two graphs BC_{min} and B_{min} minimize the same criterion and differ only in a single topological condition, their vertex degree. Let us note that for $\mathbb{R}^2$, an umbrella corresponds to an incident pair of edges in the boundary (see Figure 3.4). So we shall use *u-valence* for consistent values between $\mathbb{R}^2$ and $\mathbb{R}^3$.

We name a vertex v in a set of triangles T as *manifold* if it has exactly one umbrella in T. We further say that T is *manifold* if it is bounded by a single closed edge chain L (which may be an empty set) and all its vertices not in L, which we call *interior*, are also manifold.

Each vertex in B_{min} has exactly one umbrella in B_{min}, and therefore the u-valence $= 1$. Allowing ≥ 2 triangles per edge corresponds to relaxing the u-valence to be ≥ 1. This way we can define the *minimum boundary complex* (BC_{min}) in $\mathbb{R}^3$ formally as:

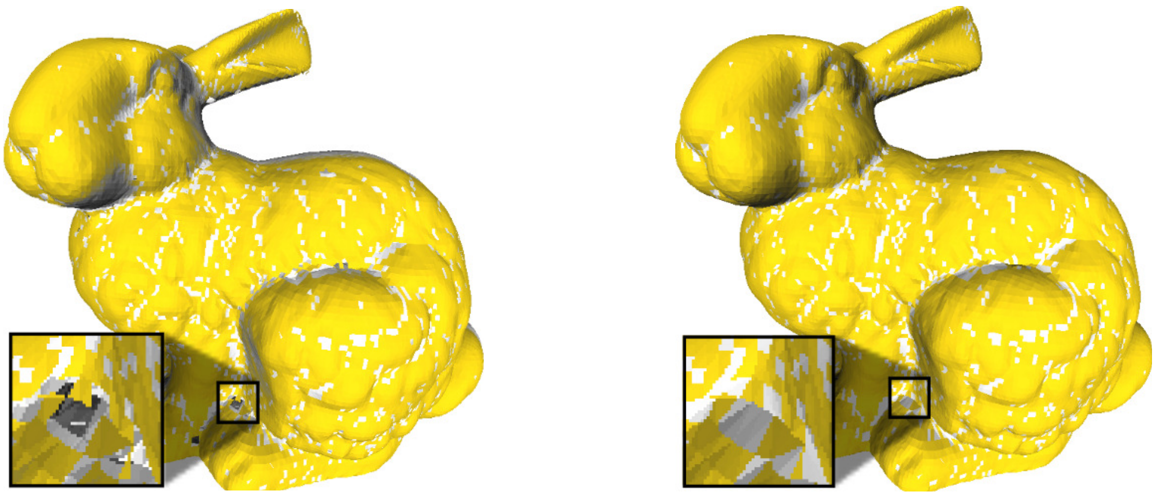
Definition 2 *The boundary complex $BC \subseteq DG$ in $\mathbb{R}^3$ is defined as a connected set of triangles spanning P such that each edge e_i in BC has ≥ 2 incident triangles in BC. Note that DG is a BC.*

For any given set of points, BC_{min} is the BC satisfying the following objective:

$$BC_{min} = \sum_{t_i}^{T} \lambda(t_i) \rightarrow \min$$

BC_{min} has the following important properties:

- Since an edge can have two or more incident triangles, BC_{min} is a single connected set but in general, not manifold.
- The relaxed topology constraint enables us to construct a close approximation using the greedy Algorithm 2 in $O(n \log n)$ time (Lemma 2). Since the output of this greedy construction algorithm is not guaranteed to be minimal, we call



(a) BC_0 (b) Desired surface B

Figure 3.5: Yellow triangles overlap in the boundary complex and in the manifold interpolating oriented surface: a) BC_0. b) B_{min} approximation.

it BC_0. Note that BC_0 construction is not an advancing-front algorithm, since it adds triangles sorted by a criterion independent of locality.

- BC_0 is a very good shape approximation because its triangles overlap largely with those in the desired boundary shape (see Figure 3.5). This is due to the fact that the only difference in its definition, compared to B_{min}, is the slightly relaxed topological constraint.

- This sub-set of BC_0 triangles overlapping with B_{min} can be easily identified by querying BC_0 (see conjecture next).

Our conjecture is that vertices, which in BC_0 are *uniquely interpolated* by a single umbrella, are very likely to be interpolated by that same umbrella in B_{min}, since the triangles in BC_0 are selected by the same minimization criterion.

In Algorithm 2 a *boundary edge* in a given triangle set T is any edge which has just a single incident triangle in T.

```
Input: $P, DG$
Output: $BC_0$
$BC_0 := \{\}$;
$PQ :=$ priority-queue of $t_i$ in $DG$, sorted by $\lambda(t_i)$;
while ($BC_0$ is not a connected component) $\vee$ ($\exists$ (($v_i$ in $DG$) not in $BC_0$)) $\vee$ ($\exists$
boundary edge $e_i$ in $BC_0$) do
    Remove first $t_i$ from $PQ$;
    if ($t_i$ connects unconnected triangles in $BC_0$) $\vee$ ($t_i$ contains a boundary edge in
    $BC_0$) then
        Insert $t_i$ into $BC_0$;
        foreach boundary edge $e_j$ in $t_i$ do
            Insert all $t_j$ not in $BC_0$ containing $e_j$ into $PQ$, together with their $\lambda(t_j)$;
        end
    end
end
```

Algorithm 2: Construction of the BC_0-complex

Lemma 2 *Given a point set P with n points and its Delaunay graph DG, Algorithm 2 constructs BC_0 in $O(n \log n)$ time.*

Proof 2 *Creating PQ inserts at most the $O(n)$ triangles of DG, with each insert operation being $O(\log n)$. The while loop is executed $O(n)$ times. Testing for and keeping track of connectedness is done via a disjoint set. Its operations are an amortized $O(\alpha(n))$, with $\alpha(n)$ as inverse of the Ackermann function. The inner loop is executed at most 3 times. Total complexity of the algorithm is therefore $O(n \log n)$.*

3.3 Properties of the BC

3.3.1 Topological Properties of the BC in $\mathbb{R}^3$

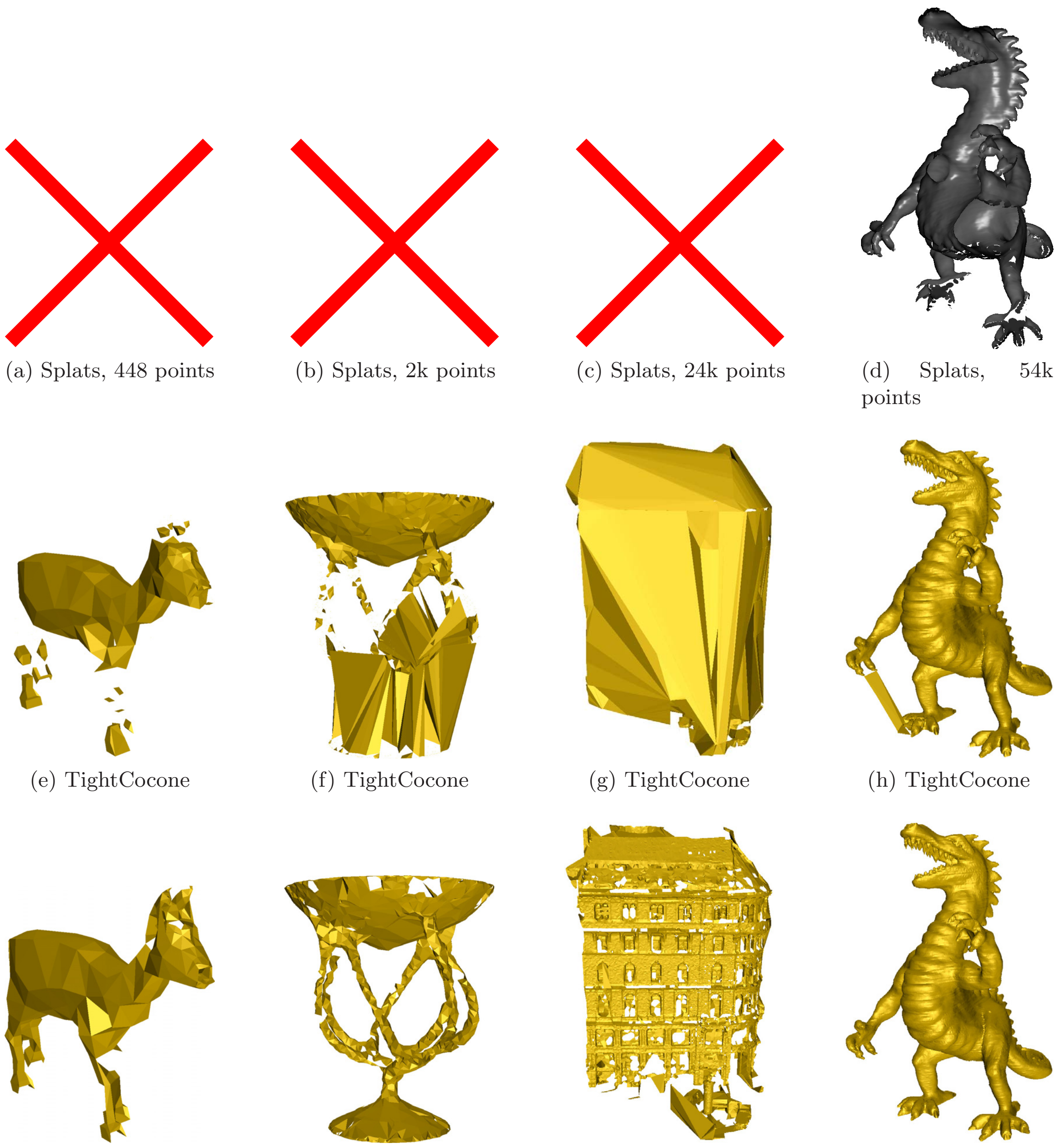
As we have mentioned before, the boundary complex is a very close approximation to the desired boundary shape, as can be seen here in comparison with the surface reconstruction method of Dey and Wenger [28] (Figure 3.6). It can therefore be used to do a quick and dirty visualization of point sets, especially non-dense and locally non-uniformly spaced, a class of point sets which are not handled that well by currently available point rendering methods.

For obtaining a closed interpolating oriented surface we would have to eliminate artifacts, which are due to the greedy BC_0 construction algorithm terminating in a local minimum, and enforce the manifold constraint. To determine these artifacts we need to analyze the topological properties of the entities in the boundary complex.

For readability, we present those by an example in $\mathbb{R}^3$, with illustrative figures in $\mathbb{R}^2$. However, since they are based on the properties of the Delaunay graph, they extend into $\mathbb{R}^d$, $d \geq 2$.

The BC in $\mathbb{R}^3$ is also a simplicial complex $\mathbb{C} \subseteq DG$.

Let P_{inf} denote the point set P enhanced by an infinite *Steiner* vertex p_{inf}, and let $DG_{inf} = DG(P_{inf})$. Using DG_{inf} instead of DG ensures that triangles in the convex hull of P also have two incident tetrahedra (one of them is infinite) and thus we can deal with all triangles in DG in a generic way.

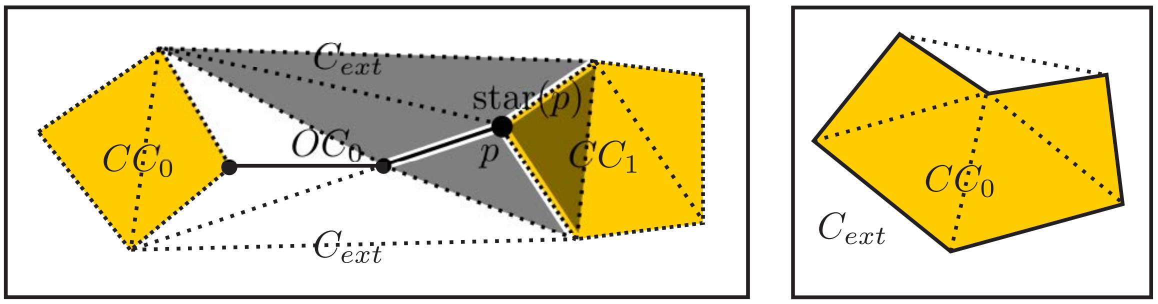


(a) Splats, 448 points (b) Splats, 2k points (c) Splats, 24k points (d) Splats, 54k points

(e) TightCocone (f) TightCocone (g) TightCocone (h) TightCocone

(i) BC_0 (j) BC_0 (k) BC_0 (l) BC_0

Figure 3.6: Comparing visualizing point sets as (row 1) splats (MeshLab, with default parameters), with (row 2) BC_0 and (row 3) with complete reconstruction TightCocone [28] for varied point sets. Note the poor results (TightCocone) and complete failures (splats) where point spacing is non-dense or locally non-uniform. The small holes in BC_0 are artifacts of local minima.



(a) Deflated entities in 2D (b) Polyhedral hull

Figure 3.7: Deflated entities illustrated by 2D example with dotted lines showing the Delaunay graph: a) Deflated vertices and edges shown as continuous lines, with the inside of closed components shaded yellow, remaining space is $\mathbb{C}_{ext}$. The *star* of the deflated vertex p, shaded grey, consists of two connected components in $\mathbb{C}_{ext}$. b) Polyhedral-hull components, shaded yellow, have no deflated entities on their boundary.

Let $\mathbb{C}_{ext}$ be the connected $\mathbb{C} \in (DG_{inf} \setminus BC)$ which is incident to p_{inf}. We call $\mathbb{C}_{ext}$ as the *external space* of BC.

We refer to a connected set of triangles T as a *polyhedral-hull component* if its hull $H(T)$ is manifold. Clearly, DG is one such polyhedral-hull component.

A *deflated* entity $x \in BC$ is a d-simplex with $d \leq 2$ (vertices, edges and triangles) such that its *star* in $\mathbb{C}_{ext}$ is not connected (see Figure 3.7 for an example in 2D).

A *deflated* component denoted by OC, $OC \subseteq BC$, is any connected set of deflated entities x such that no other deflated entity in BC is incident to that OC. Intuitively, this term comes to mind by looking at the BC_{min} as a not entirely inflated air mattress. Its hull is still deflated in some places, causing depressed pockets in the surface of the mattress.

It is easy to see that if a BC has no deflated entities, then $H(BC)$ is polyhedral.

BC minus the set of all its deflated triangles (deflated edges and vertices are not considered, since if they are not part of deflated triangles, they belong to non-deflated triangles) consists of a set of (possibly vertex- or edge-connected) triangle sets, which we name as closed components. By this definition a closed component, denoted by CC, has no deflated entities and hence it is a polyhedral-hull component, i.e. $H(CC)$ is a polyhedron for every CC.

Separating these closed and deflated components, as will be shown later, is the key to identify artifacts due to the local minima and to transform BC_0 into a manifold boundary.

3.3.2 Local Construction of BC_0

Effectively visualizing large out-of-core data sets requires construction of the shape in parts by just considering local sub-sets of points (see Figure 3.10a), which however must be deterministic for the global point set P with n points. We prove that BC_0 construction has this property. The proof is based on a well-known property of its underlying Delaunay graph DG. The set of *natural neighbors* $Z(p)$ for a $p \in P$ is its 1-neighborhood in $DG(P)$ and $\rho(p)$ is the radius of its circumsphere (ρ-circumsphere) containing $Z(p)$. We name the i-nearest neighbor of $p \in P$ as $NN_i(p)$. The Delaunay property for a set of tetrahedra Q incident to $p \in P$ is fulfilled if the circumsphere of each $q \in Q$ does not include any other point of P. We define the shadow space of a point p w.r.t. a tetrahedron q as the collection of all half rays originating at a point p and intersecting q. The Delaunay graph $DG(p)$ at p containing the non-empty set

Q is complete if all tetrahedra $q \in Q$ fulfill the Delaunay property and the ambient space at p which is not shadowed w.r.t. any $q \in Q$, does not contain any points in P. We base Algorithm 3 on incremental Delaunay graph construction, for which Edelsbrunner and Shah [37] have given an expected complexity of $O(n \log n + n^{[d/2]})$, although they use randomized insertion order.

```
Input: $P$, $p$
Output: $DG(p)$
$DG(p) := q(p, NN_1(p), NN_2(p), NN_3(p))$;
$i := 4$;
while $DG(p)$ is not the complete Delaunay graph for $p$ do
    Insert $NN_i(p)$ into $DG(p)$;
    $i := i + 1$;
end
```

Algorithm 3: Find DG locally for p

Lemma 3 *For $P_{sub} \subset P$, $DG(Z(P_{sub}))$ can be found by querying just the space inside and on the ρ-circumsphere of P_{sub} in P, provided P contains no 4 co-planar points.*

Proof 3 *Assuming that P is non-degenerate (no 4 points are co-planar), $DG(P)$ has the property that no tetrahedron in $DG(P)$ contains in its interior any $p \in P$. For finding the local $DG(p)$, no p_i outside its circumsphere with radius ρ needs to be queried, since its farthest natural neighbor has distance ρ to p. If p_i were to be contained inside the circumsphere of any tetrahedron q in $DG(p)$, then $DG(p)$ would have to be modified to contain p_i, and it would also become one of its natural neighbors.*

By employing a suitable searching strategy which, for already constructed tetrahedra containing p, does not query points in their sector outside their circumsphere,

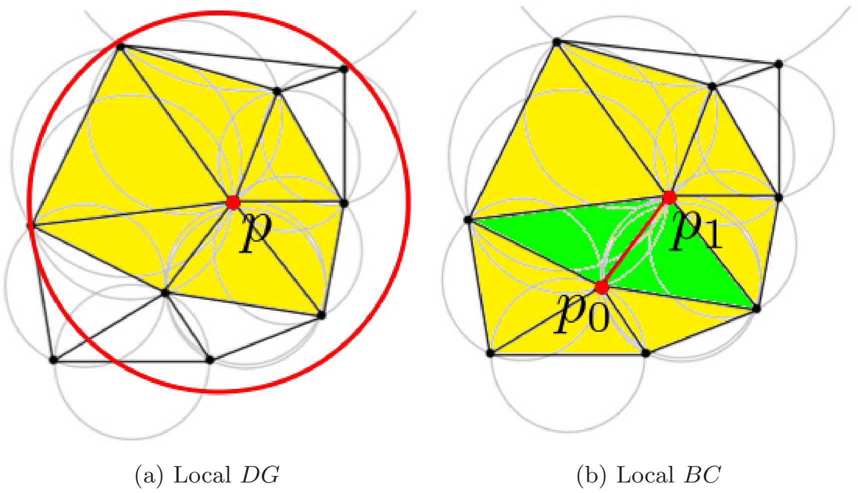


Figure 3.8: Illustration in $\mathbb{R}^2$: Delaunay graph with the empty circumcircles of its triangles shown as grey circles. a) Local construction of Delaunay graph, shaded yellow. The natural neighbors of p (linked by an edge) are bound by the red circle with radius ρ. b) Local construction of boundary complex, for both p_0 and p_1, shaded yellow. The triangles incident to their shared edge, shaded green, are identical for both points.

the number of queried points could be further limited to the set of natural neighbor points $Z(p)$.

Now we can prove that the *Boundary Complex* can be constructed locally from a sub-set of P such that it is deterministic for P (see also Figure 7.2).

Theorem 1 *$BC_0(P_{sub}) \subseteq BC_0(P)$ for $P_{sub} \subseteq P$.*

Proof 4 *Lemma 3 states that for any $p_i \in P$, $DG(p_i) \subset DG(P)$, the set of incident Delaunay tetrahedra, can be constructed by querying only a local sub-set of P. It follows that for an edge e in $DG(p_i)$ incident to p_i all triangles incident to e are also in $DG(p_i)$. Then the condition of ≥ 2 triangles per edge and their sorting order are deterministic at each end-point of e. Induction extends it from p_i to $P_{sub} \subset P$.*

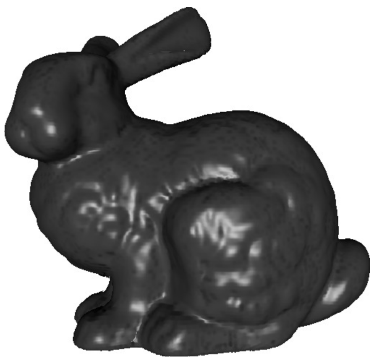
3.3.3 Point Spacing Density

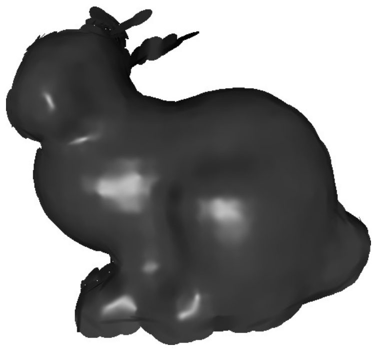
The boundary complex is very robust with respect to density of point spacing and is hardly affected by down-sampling (see Figure 3.9, with up to 99.7% points removed). Let us note that down-sampling a point set not only increases sparsity, but could also affect uniformity, depending on the method used.

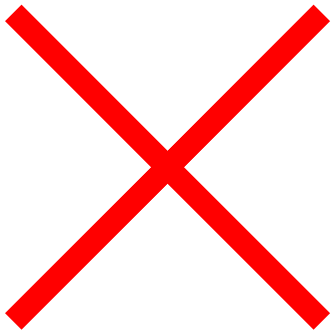
3.3.4 Noise Tolerance

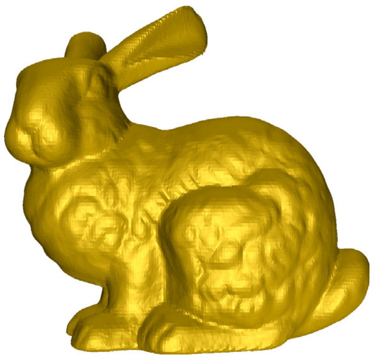
The presence of significant noise can cause major problems for visualization methods which require construction of an interpolating surface, especially where the noise level not only exceeds feature size but leads to regions with very sparse point spacing. The boundary complex relaxes the requirement of a *manifold interpolating* surface to simply *interpolating* the points. Even extremely noisy points are simply incorporated into a *thick crust* surface, with some points becoming interior to the visible boundary (see Figure 3.10b- 3.10d).

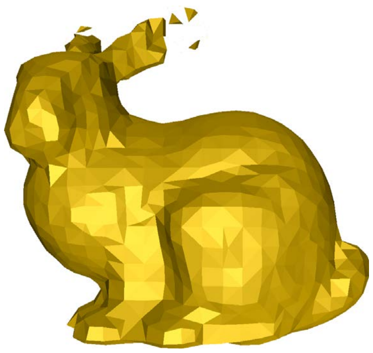
The reason for this robustness to noise is that the requirement of separating noise from features is dropped altogether when we relax the manifold constraint. That separation requires either an assumed surface, which is manifold or with restricted curvature, or else the availability of a noise model. Since our algorithm is agnostic to both, it treats noise exceeding feature size just as features of different extent and in the extreme case, noisy points are seen as a sparsely sampled signal representation.

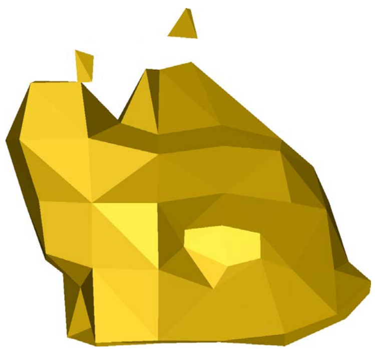

(a) Splats, 100%

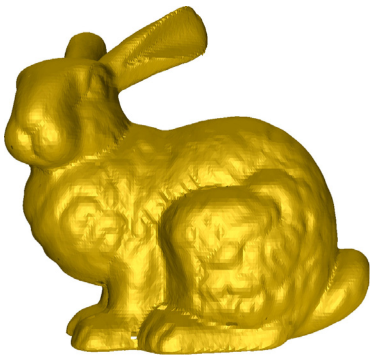

(b) Splats, 4%

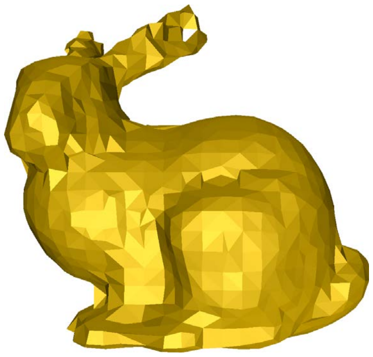

(c) Splats, 0.3%, failed


(d) TightCocone, 100%


(e) TightCocone, 4%


(f) TightCocone, 0.3%


(g) BC_0, 100%


(h) BC_0, 4%

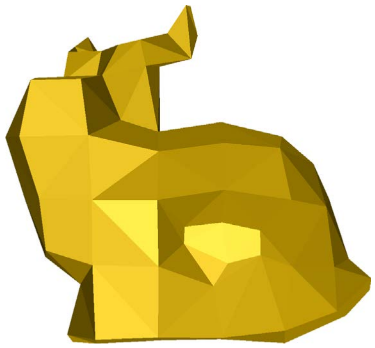

(i) BC_0, 0.3%

Figure 3.9: Comparing results of heavily down-sampled point sets (vertices are clustered and decimated on a uniform grid, with MeshLab). Left: Original Stanford bunny, 36k points. Center: Bunny, down-sampled to 1270 points (4%). Right: Bunny, down-sampled to 107 points (0.3%). Row 1: Rendered as uniformly-sized splats, note the errors at silhouettes and (complete) failure at the non-dense or locally non-uniformly spaced points at the ears. Row 2: TightCocone [28] reconstruction results for the challenging regions in disconnected components. Row 3: Our boundary complex handles the challenging regions gracefully.

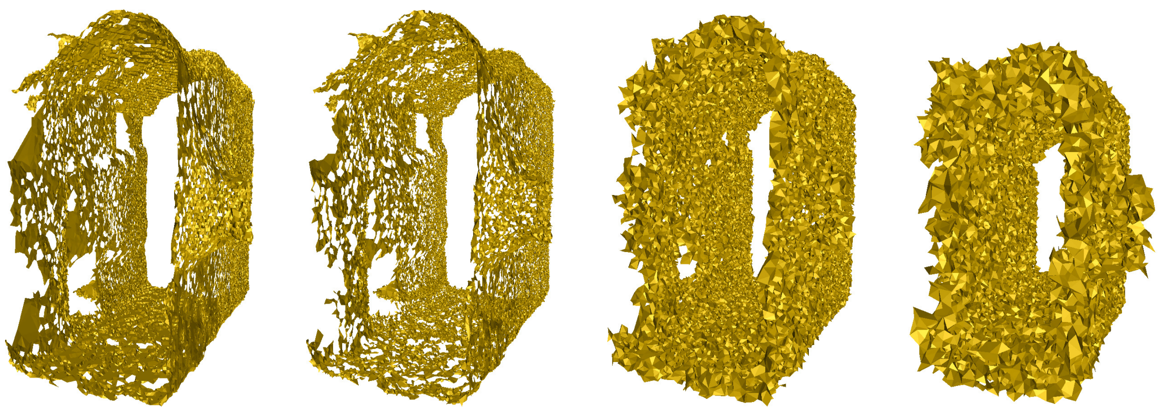

(a) BC_0, 174k points (b) 0.4% perturbed (c) 4% perturbed (d) 10% perturbed

Figure 3.10: a) Noisy range-scan data (catacomb corridor section), 174k points. b-d) additionally perturbed by given percentage of z-extent of model. Note that neither MeshLab can construct splats, nor TightCocone outputs any boundary for all of these models, including the original data on the left. Figure *d* has been slightly turned to show the opening.

3.4 Concluding Remarks on Minimum Boundary Complex

As can be seen from the above, the Minimum Boundary Complex is a very interesting simplicial complex with potentially many applications in dealing with point sets. These applications include quick and dirty visualization, generating shape descriptors for use in point set retrieval, topological analysis, providing a good starting structure for boundary shape construction, etc. Constructing an aesthetically pleasing shape for an unorganized set of points, given just the coordinate data is a complex problem and especially if the spacing of points on the desired shape is not dense or is locally non-uniform. In subsequent chapters we describe in detail the new algorithms we have developed for shape construction based on the minimum boundary complex. The results from these algorithms are superior to current solutions for this problem.

Chapter 4

Boundary Construction in $\mathbb{R}^2$

In this chapter we present an application of the minimum boundary complex, a new method for constructing the boundary shape of a given unorganized point set in $\mathbb{R}^2$. This method significantly improves quality of results compared to previous methods, especially for non-dense and locally non-uniform point sets. Its complexity is competitive with an expected $O(n \log n)$.

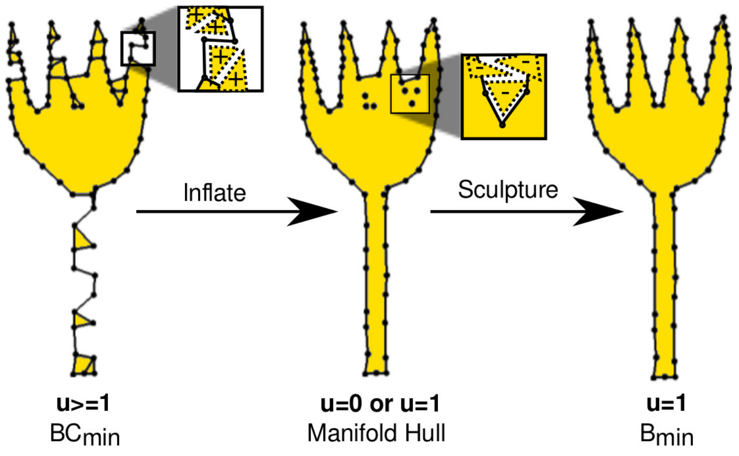


Figure 4.1: Comparing the minimizing graphs in $\mathbb{R}^2$ with their u-valence.

4.1 Overview

We have mentioned above in Subsection 3.1.2 that the boundary complex BC is close to B_{min} because the only difference is a slightly varying topological constraint, namely vertex degree, or more generally, umbrella count per vertex.

The $EMST$ has vertices with u-valence ≥ 0 and B_{min} has u-valence $= 1$. Based on this, we have defined BC_{min} as a minimizing graph with u-valence ≥ 1. We further show that its close approximation, which we name BC_0, can be efficiently constructed by a greedy algorithm.

Figure 4.1 illustrates how BC_0 can be transformed by two steps into the desired interpolating boundary shape B_{min} or its close approximation. It also demonstrates how the u-valence varies during this transformation (a u-valence of 0 signifies vertices inside of the boundary).

Once BC_0 has been constructed based on the DG, the next stage of our method is to transform BC_0 so that each vertex in BC_0 is contained in exactly 1 umbrella each to yield B_{min}. For this, we start with $H(BC_0)$, the manifold hull of BC_0. Let us note that $H(BC_0)$ has both manifold vertices ($= 1$ umbrella) and non-manifold vertices (> 1 umbrellas). To better understand the steps in this transformation process, we employ for $H(BC_0)$ the metaphor of a yet partially inflated air mattress. This mattress can be fully inflated by adding triangles from DG to the triangulation inside its $H(BC_0)$ until all its non-manifold vertices have either become manifold or become inside of the resulting new $H(BC)'$. We call this first step in transforming BC_0 to

B_{min} or at least a close approximation, as the *inflating* operation and the result as inflated boundary $H(BC')$. BC' is then a closed component CC.

The vertices in CC therefore have as their degree in $H(CC)$ either 0 if *interior* to $H(CC)$, or 1 if on $H(CC)$. For the second step, we use the dual of the *inflating* operation, namely, *sculpturing*, to remove triangles from the triangulation inside $H(CC)$ until all interior vertices get exposed on the boundary.

In Table 4.1 we compare the properties of the graphs described so far (see also Figure 3.1).

This *sculpturing* operation results in an interpolating boundary B. Boissonnat [17] first introduced this term in 3D where he defined it as removing boundary tetrahedra from the convex hull in order to expose all interior vertices on that boundary. A criterion such as tetrahedron circumsphere radius is used to determine the order of removal. His algorithm is guaranteed to expose all vertices in a combinatorial search. However, with the convex hull as start set and using only heuristic sorting, it ends up quickly in local minima.

Our contribution to sculpturing is to appropriately choose the sorting criterion for shape characteristic minimization, and to apply it starting from a close approximation of the desired shape, namely $H(CC)$. This way we can avoid getting stuck in a local minimum frequently.

Given P and $DG(P)$, the entire algorithm consists of the following steps (see Figure 4.2):

Graph	Vertex Degree	Umbrellas
$EMST$	$c \geq 1$	$u \geq 0$
BC	$c \geq 2$	$u \geq 1$
B_{infl}	$c = 0$, or $c = 2$	$u = 0$, or $u = 1$
B_{min}	$c = 2$	$u = 1$

Table 4.1: A comparison of the described graphs by their constraints of vertex degree c and corresponding umbrella count u.

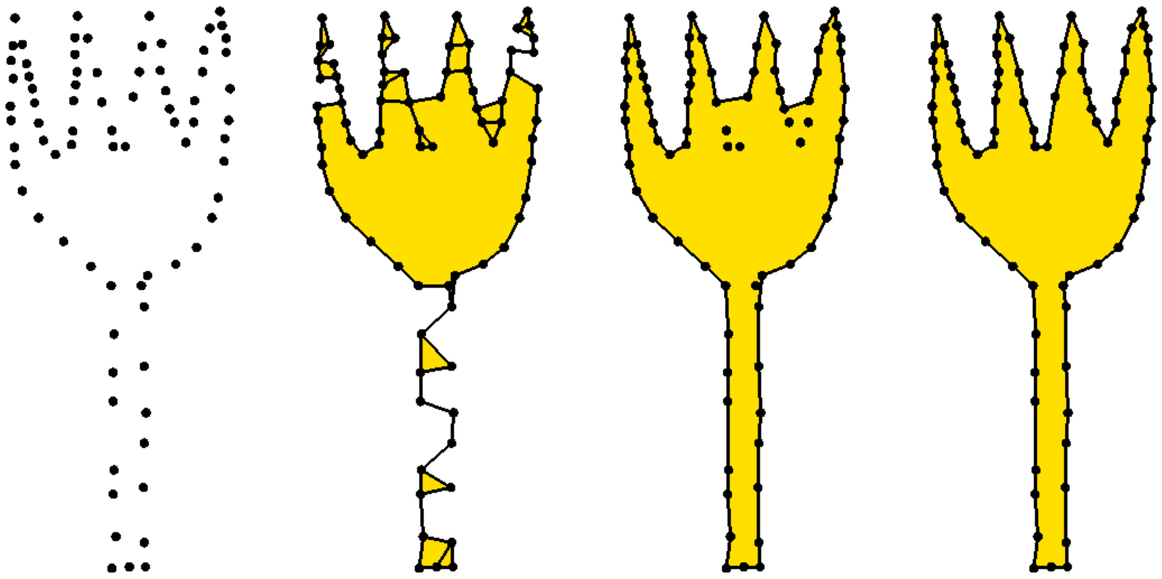


Figure 4.2: Area inside hull always shaded: a) Point set. b) BC_0. c) After *inflating*: Manifold boundary $H(CC)$. d) After *sculpturing*: Interpolating boundary B_{min}.

- Construct BC_0 from DG.
- Identify edges in BC_0 which make up $H(BC_0)$.
- Apply inflating operation to $H(BC_0)$ to transform it into a manifold boundary $H(CC)$.
- Apply sculpturing operation to $H(CC)$ to determine interpolating boundary B_{min} or its near minimum.

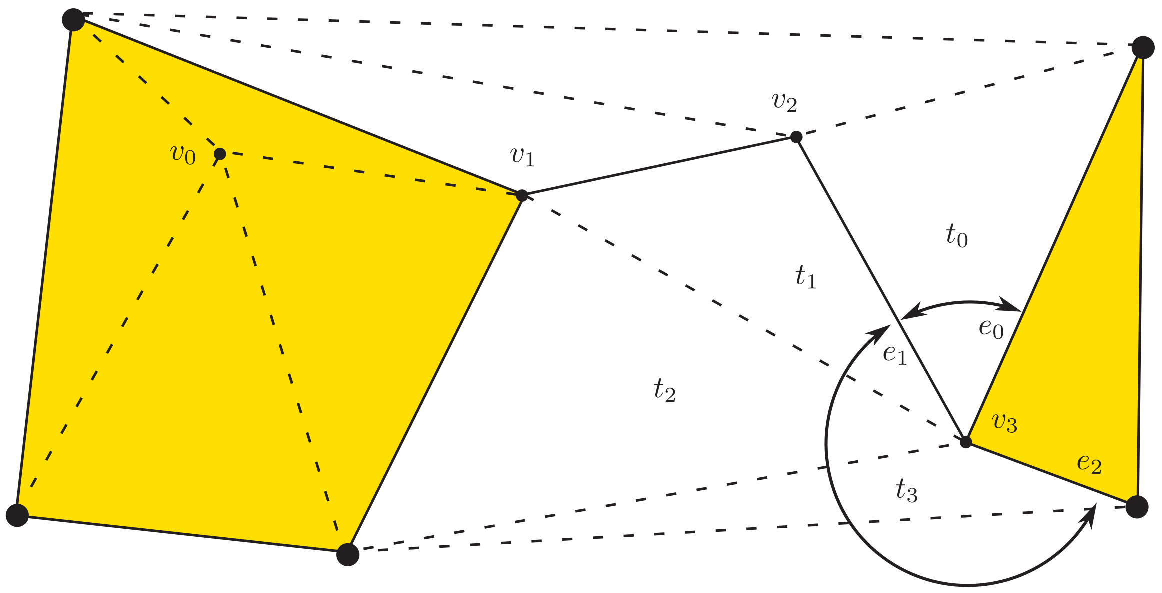


Figure 4.3: Vertex classification. Dotted edges are in DG, solid edges in BC_0, here equal to its $H(BC_0)$. Large dots are *manifold* (v_0 is *interior*, v_1, v_2, v_3 are *deflated*). v_3 has two umbrellas on the manifold hull (delimited by the arcs): e_0-e_1 bounds t_0, e_1-e_2 bounds the triangle fan $\{t_1, t_2, t_3\}$.

For the exact minimum boundary, the number of combinations of triangles which have to be considered in the process of inflating and sculpturing can be very large, and this is what makes our problem NP-hard. In our method, we use the heuristic of sorting triangle candidates based on their $\Delta\|B\|$ impact on the boundary, so that the total boundary length is minimized. This process results in B_{min} or a close approximation for a large class of point sets which we define more precisely later.

4.2 Find the Manifold Hull

We say an edge in BC is reachable from the convex hull of the point set if:

1. it is an edge of the convex hull, or

2. it is an edge of a triangle (in DG) with at least one edge on the convex hull, or
3. starting from a triangle with an edge on the convex hull, it can be reached by traversing connected triangle edges, without traversing any other BC edge.

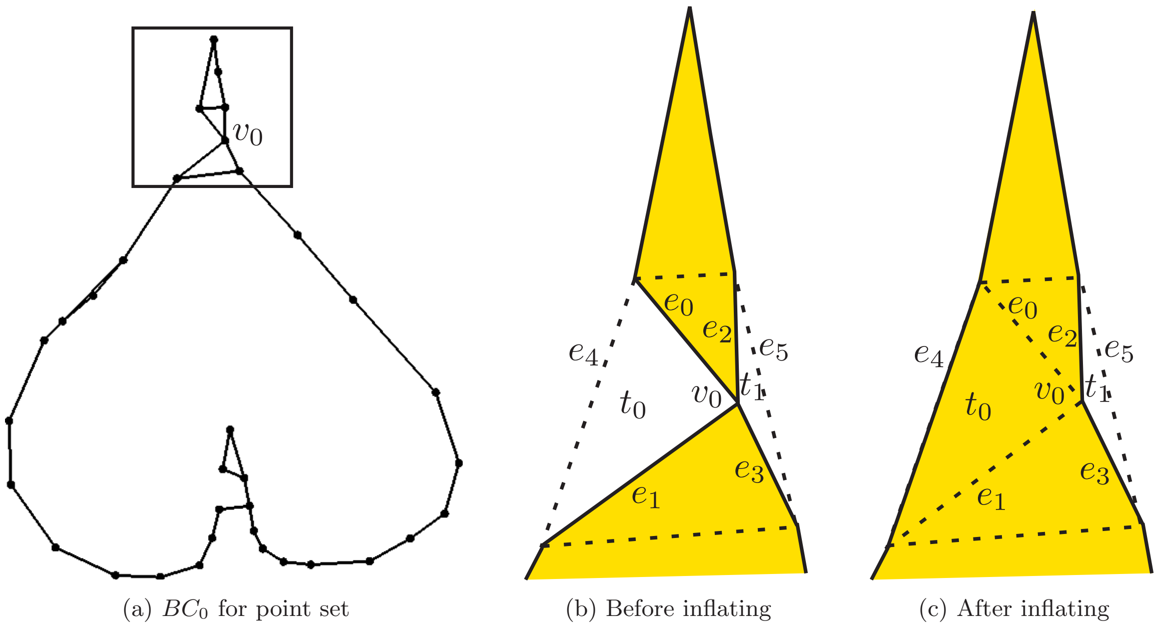
All reachable edges of BC make up the manifold hull $H(BC)$ (see Algorithm 4). We define $T(B)$ as all triangles inside of that boundary B.

```
Input: $BC$
Output: $H(BC)$
$T$ = set of triangles in $DG(P)$;
while ($e_i$ in $H(T)$) not in $BC$ do
    $t_i$ is triangle incident to $e_i$ inside of $H(T)$;
    $T := T \setminus t_i$
end
$H(BC) = H(T)$
```

Algorithm 4: Determine manifold hull $H(BC)$ for BC

Lemma 4 *The manifold hull $H(BC)$ for BC in DG contains all vertices in P either in $H(BC)$ or in its interior. Further, Algorithm 4 computes $H(BC)$ in $O(n \log n)$ time.*

Proof 5 *The convex hull of P, denoted as the initial $H(T)$, includes all $p_i \in P$ on it or in its interior. Removing a triangle from T never moves an edge e_i in BC to the outside of $H(T)$. Since BC interpolates all vertices in P, it follows that no $p_i \in P$ can become exterior to $H(T)$, which eventually becomes $H(BC)$. Since each loop removes a triangle in DG from T, it is executed at most $O(n)$ times. As local operation and set operation its complexity is $O(\log n)$ and therefore the overall complexity of the algorithm is $O(n \log n)$.*



(a) BC_0 for point set (b) Before inflating (c) After inflating

Figure 4.4: a) BC with single *non-manifold* vertex v_0 on its $H(BC)$. Details inside the frame, with DG as dotted and $H(BC)$ as solid lines (inside shaded): b) t_0 and t_1 are both candidates at v_0. Since $\Delta\|t_0\|$ is minimal, t_0 is selected to add to $T(BC)$. c) v_0 is now manifold in $H(BC')$, therefore t_1 is no longer a candidate.

4.3 Inflating

Any vertex $v_i \in P$ can be classified by the number of umbrellas u its incident edges form in $H(BC)$ (see Figure 4.3) as follows: v_i is *interior* to $H(BC)$ if $u = 0$, *manifold* on $H(BC)$ if $u = 1$ and *non-manifold* otherwise.

We define an *inflating-candidate* triangle for $H(BC)$ as a triangle t_i on its outside which is incident to a non-manifold vertex v_i in $H(BC)$.

Let $T(BC)$ denote all triangles which are inside of $H(BC)$. The operation "Add a triangle t_i to $T(BC)$" combines their space inside the new enclosing boundary $H(BC')$ which is formed by XORing the edges of t_i in $H(BC)$. $\Delta\|t_i\|$ provides a measure of the impact of changes to $H(BC)$. It is the value calculated by adding $\|e_j\|$ for all its edges $e_j \notin H(BC)$ and removing it for its edges e_j in $H(BC)$ (see Figure 4.4).

To *inflate* $H(BC)$ we select and add a triangle from the current set of inflating-candidate triangles. This reduces the number of its non-manifold vertices. We repeat this triangle addition process until $H(BC)$ becomes manifold. The selection criterion used is the smallest $\Delta\|t_i\|$ (value is always negative). It gives priority to adding the largest triangles and such ones having the most acute angles at the non-manifold vertices. In turn, this helps minimize the length of $H(BC)$.

Every vertex v_i in $H(BC)$ which is non-manifold has candidate triangles. This is so because of the following: if all its incident triangles were in $H(BC)$, it would have to be either interior or on the convex hull.

```
Input: $BC$
Output: $CC$
$PQ$:=priority-queue of candidate $t_i$, sorted by $\Delta\|t_i\|$;
while $PQ \neq \{\}$ do
    Remove first triangle $t_i$ from $PQ$;
    $BC$:=$BC \cup t_i$;
    foreach $v_j \in V(t_i)$ do
        Update state for $v_j$
    end
    foreach $t_j$ in $(DG \setminus BC)$ and sharing a vertex with $t_i$ do
        Determine if $t_j$ is a candidate and update $PQ$ with it
    end
end
$CC$:=$BC$
```

Algorithm 5: Inflating the manifold hull to a manifold boundary

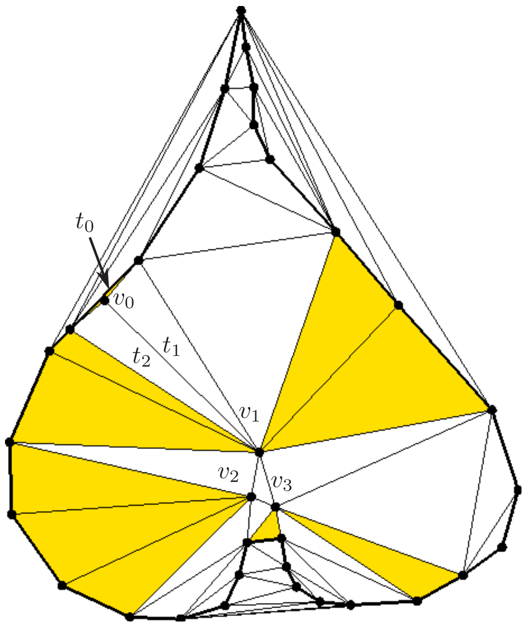
Lemma 5 *Given a manifold hull $H(BC)$ in DG, Algorithm 5* inflates *it in $O(n \log n)$ time to construct $H(CC)$, which is always a manifold.*

Proof 6 *We first prove that the while loop is guaranteed to terminate. Any triangle* t_i *in* DG *which is on the outside of* $H(BC)$ *is a candidate to add to* $T(BC)$*, if it is incident to a non-manifold vertex in* B_e*. All triangles outside* $H(BC)$ *can at most be added once, since there is no removal operation. The while loop terminates if all vertices in* $H(CC)$ *are manifold or in the limit all candidate triangles are added. In the latter case,* $H(CC)$ *becomes identical to the convex hull, which is a manifold boundary. Determining the* manifold *state of a vertex, if a triangle* t_i *is a candidate, and calculating its* $\Delta\|t_i\|$ *are all of* $O(1)$ *complexity, since the computation is only dependent on the* k *incident edges in* DG*. The* $O(n)$ *triangles in* DG *are at most inserted a constant* $3k$ *times into* PQ*. Both inner loops execute a number of local operations, only the second loop contains operations on sorted lists or sets, which are* $O(\log n)$*. The algorithm is therefore of complexity* $O(n \log n)$*.*

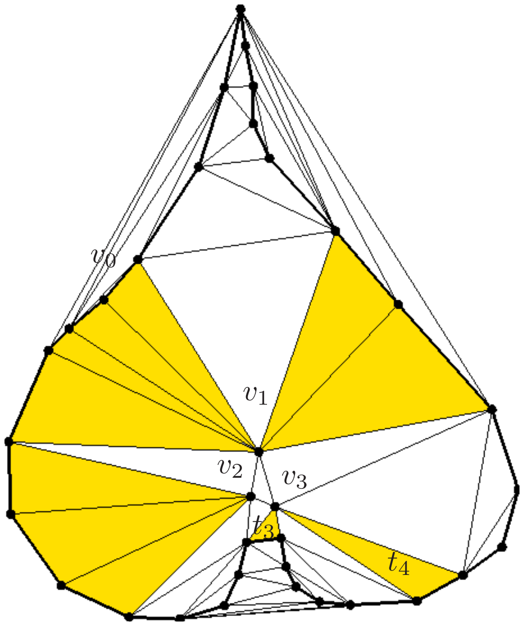
4.4 Sculpturing

$H(CC)$ is manifold, but may contain some points of P as interior vertices. In [17] Boissonnat defines some well-defined rules in 3D for removing tetrahedra from the convex hull of a point set. According to these rules, all interior vertices can be exposed onto the boundary and any polyhedron with genus 0 can be reached.

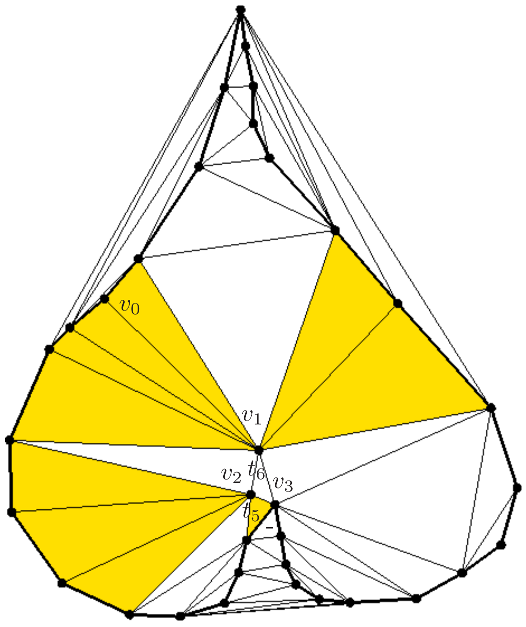
It follows that in $\mathbb{R}^2$ any triangle (in DG) can be removed from the convex hull $H(DG)$, if it has one edge on $H(DG)$ and its opposing vertex is interior to H. With a series of such removals, an interpolating polygon is reached. It is easy to see that this



(a) Before removing t_0



(b) Before removing t_3



(c) After removing t_3

Figure 4.5: Point set with manifold hull of closed component $H(CC)$ shown using thick lines, other edges in DG shown with thin lines, interior vertices marked and triangle candidates for sculpturing shown shaded. a) Of the 10 candidates, $\Delta\|t_0\|$ is minimal (t_0 is small and very thin). b) By removing t_0 from $T(B)$, interior v_0 becomes interpolated by $H(CC)$. For v_1, two new candidates are added (t_1, t_2) as they now share edges with $H(CC)$. t_3 is selected next to remove. c) t_4 is no longer a candidate, since it is not incident to an interior vertex. t_5 and t_6 will be removed subsequently to interpolate v_2 and v_1 leading to B_{min}.

holds if we replace the convex hull with any manifold hull $H(CC)$ in DG. Removal of the triangle exposes the interior vertex onto $H(CC)$ since its incident edges become part of it, and this permits us to obtain any contained polygon in DG.

Of course, the DG must contain a Hamiltonian cycle. However we have not encountered any point sets with non-Hamiltonian DG. Those have been observed to be extremely rare by Genoud [41] (see Dillencourt [33] for a contrived example), it is therefore not a real concern in practice.

We define a *sculpturing-candidate* triangle for a $H(CC)$ as a triangle t_i on its inside with one edge in $H(CC)$ and its opposite vertex as interior in $H(CC)$.

Algorithm 8 exposes vertices efficiently to get an interpolating boundary (see Figure 4.5).

```
Input: $H(CC)$
Output: $B$
$B = H(CC)$;
$PQ$:=priority-queue of candidate $t_i$, sorted by $\Delta\|t_i\|$;
while $PQ \neq \{\}$ do
    Remove first triangle $t_i$ from $PQ$;
    $v_i$ in $t_i \notin B$;
    $T(B)$:=$T(B) \cap t_i$;
    foreach $t_j \in T(B)$ and sharing a vertex with $t_i$ do
        Determine if $t_j$ is a candidate and update $PQ$ with it
    end
end
```

Algorithm 6: Sculpture to an interpolating boundary

For unreasonably non-uniform point spacing, sculpturing may not expose all points on the boundary. So the resulting boundary will still be manifold, but may not interpolate the entire point set. That effect is limited to the local neighborhood of these points. We shall denote such points as *dominantly interior*.

Lemma 6 *Sculpturing triangles from a boundary B in DG with Algorithm 6 is of $O(n \log n)$ complexity and produces a manifold B interpolating all but dominantly interior vertices.*

Proof 7 *Since $H(CC)$ is manifold and each sculpturing operations preserves this property, the resulting B is also guaranteed to be manifold. Points which are not dominantly interior, are at some point contained in a triangle with an edge on the current $H(CC)$ and can therefore be exposed onto it. Determining if a vertex is* interior *is $O(1)$ complexity and so are calculating $\Delta\|t_i\|$ for a triangle t_i and determining if it is a candidate. The outer loop is executed at most for the $O(n)$ triangles in DG. The inner loop contains operations on sorted lists or sets, which are $O(\log n)$. The algorithm is therefore of complexity $O(n \log n)$.*

4.5 Complexity

Theorem 2 *Using the main algorithm in Section 4.1, a minimum, closed, non-intersecting and manifold shape interpolating all but the dominantly interior points, can be found in expected $O(n \log n)$ time, provided DG contains a Hamiltonian cycle.*

Proof 8 *The Delaunay triangulation step is of $O(n \log n)$ expected complexity as shown in Guibas and Stolfi [47]. All the following steps are also of $O(n \log n)$ complexity as proved in the respective lemmas: Construction of the boundary complex*

(Lemma 1), locating its manifold hull (Lemma 4), inflating it to a manifold boundary (Lemma 5) and including its non-dominantly interior vertices on that boundary while maintaining its manifold property (Lemma 6). The total complexity is therefore linearithmic.

We have observed that in practice the performance is determined to a large factor by the time for DG construction.

4.6 Concluding Remarks on Boundary Construction in $\mathbb{R}^2$

In this chapter we have presented a new method for constructing the boundary shape of a given set of unorganized points in $\mathbb{R}^2$ by pursuing the idea of minimizing a shape characteristic of the edges making up the shape. Our method is distinct from all previous methods. It adapts the idea of sculpturing (originally defined for $\mathbb{R}^3$) [17]) by removing triangles. Further, it starts not from the convex hull, but from a close approximation of the desired boundary shape. For deriving this close approximation, it first uses a greedy algorithm to construct an approximation of the minimum boundary complex. Then it applied the new inflating operation, a dual of sculpturing, which actually adds triangles to obtain a transformed boundary complex whose hull is a closer approximation of the desired shape. An implementation of this method and the results for different example point sets are presented in Chapter 6 clearly showing that the results are superior to previous methods, especially for non-dense and locally non-uniform point sets. A very important aspect of this method is that it extends well into $\mathbb{R}^3$), even though the boundary shape construction problem in $\mathbb{R}^3$) is very much more complex. This is discussed in the following chapter.

Chapter 5

Boundary Construction in $\mathbb{R}^3$

In this chapter we present the algorithms making up our new method which constructs the boundary shape for a given set of unorganized points in $\mathbb{R}^3$, extending the previously described $\mathbb{R}^2$ method. It shows as well significant improvements over previous methods, especially for non-dense and locally non-uniform point sets, and competitive complexity of expected $O(n \log n)$.

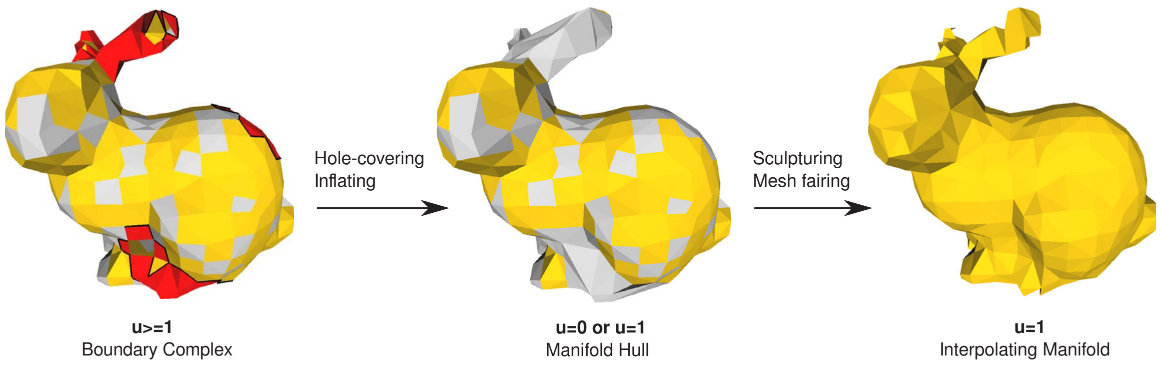


Figure 5.1: Our method in a nutshell. Red triangles surround hull-holes, the triangulation is ”thick” (gray) or ”thin” (yellow). Left: Initial Boundary Complex. Center: Manifold hull. Right: Interpolating manifold with minimized curvature.

5.1 Overview

Sculpturing: From a given simplicial complex $\mathbb{C}$, with $B(\mathbb{C})$ denoting its triangulated boundary surface, the sculpturing operation eliminates tetrahedra, ordered so as to minimize curvature in the resulting surface. The goal of sculpturing is to expose vertices which are interior to $B(\mathbb{C})$ onto it. It guarantees that manifoldness and genus of $B(\mathbb{C})$ remain unchanged.

In its original proposal, sculpturing starts from the convex hull of a given point set. Since this can quickly end up into local minima, our proposal is to start from the boundary complex BC_0, actually its hull, $H(BC_0)$. The main idea in this heuristic is that, BC_0 being a much closer shape approximation to the desired surface, the sculpturing process would terminate closer to it. However since the hull $H(BC_0)$ is not guaranteed to be manifold, we will first need to transform BC_0 into a BC' so that $H(BC')$ is manifold, while remaining close to the optimal shape, B_{min}.

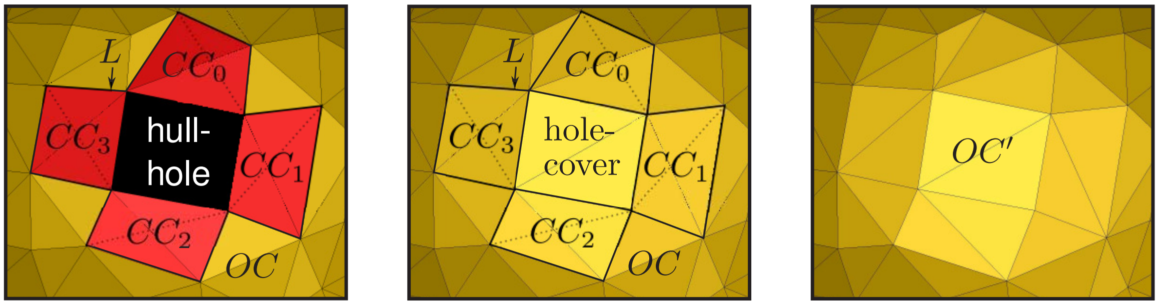
For this, we introduce an operation called *Inflating* as the *dual* of sculpturing. While sculpturing removes tetrahedra, inflating adds tetrahedra to BC_0, sorted by the same criterion, until the resulting BC' contains no more deflated components. Its hull $H(BC')$ is then manifold, although it may contain some points of P in its interior. Ideally, $H(BC')$ should minimize the same objective as BC_{min} and B_{min}, while also having the least number of points in its interior.

These two operations of inflating and sculpturing correspond to modifying the topological constraints of the simplicial complexes in $\mathbb{R}^3$ as shown in Figure 5.1. They are also directly parallel to the corresponding operations in $\mathbb{R}^2$ shown in Figure 4.1.

Inflating has, as does sculpturing, problems of getting stuck at local minima resulting in a BC' with a hull that could be far from the desired surface. This is because of unwanted artifacts which show up in BC_0 due to the greedy construction algorithm used, particularly in regions where the density of point spacing decreases non-uniformly. In such regions, the relaxed condition of ≥ 2 triangles per edge leads the algorithm to fold back the surface onto itself (see Figure 5.2), resulting in pockets (in the hull) with a closed edge chain L. That is, triangles incident to L with smaller $\lambda(t)$ get added to BC_0, creating closed components everywhere at L. This manifests as a hole in the polyhedral hull for BC_0. Hence we shall refer to this as a *hull-hole*. The hull-hole is such that a triangulated disk bounded by L would close it. Since any $H(CC)$ is polyhedral, a hull-hole can only exist in the presence of a deflated component. It can be detected by inspecting its surrounding polyhedral-hull components.

We introduce a *hole-covering* operation for closing hull-holes, which is applied prior to inflating, to reduce the possibility of the inflating operation from quickly falling into a local minimum. The main heuristic here is to get a deflated component to become part of a larger closed component, while still keeping the operation local to the components associated with this hull-hole.

In order to get to the main algorithm quickly, we shall defer a detailed description of the operations involved in hole-covering till until later in this chapter and for now



(a) Hull-hole (b) Covered hull-hole (c) Sculptured

Figure 5.2: Local minimum in BC_0: a) A hull-hole (black) corresponding to a triangulated disk with 4 vertices. Closed components (here, 4 tetrahedra) are shaded red, the deflated component shaded yellow. The closed edge chain L shown by thick black edges encloses the hull-hole, separating closed and deflated components. b) *Hole-covering* has added the triangulated hole-cover. c) *Sculpturing* has removed redundant triangles from the tetrahedra of the closed components.

only give a quick overview. The first step is to detect hull-holes in a given BC. For this, we segment the BC into deflated (OC) and polyhedral-hull components (CC) and determine their shared boundaries. Then we inspect CCs associated with these shared boundaries to see if they surround a hull-hole which needs to be covered. Lastly, a set of triangles is found which contains a triangulated disk for covering a detected hull-hole.

All L containing hull-holes must be in the form of closed edge chains. For this, we require the BC to *conform* to three properties, which we also list later. In practice, non-conforming entities are rare. And further, it is easy to make a given BC to conform to these properties.

Once the operations of hole-covering, inflating and sculpturing are completed, we apply a simple *mesh fairing* algorithm to directly minimize curvature where not already achieved by the λ-criterion. For an overview of these steps in sequence, please see Figure 5.3.

5.2 Inflating

We shall use BC to generically denote the simplicial complex which results from successively performing the various operations starting from BC_0. The inflating operation (Algorithm 7) converts a given BC into a CC.

Let V_o denote the set of deflated vertices in BC. A tetrahedron q in $(DG \setminus BC)$ is a candidate for adding if it contains ≥ 1 vertices in V_o and ≥ 1 triangles in $H(BC)$.

```
Input: $BC$, $V_o$
Output: $CC_f$
$PQ$ := priority-queue of candidate $q_i$, sorted by $\sum \lambda(t_i$ in $q_i$ in $BC)$;
while $PQ \neq \{\}$ do
    Remove first tetrahedron $q_i$ from $PQ$;
    $BC$:=$BC \cup q_i$;
    foreach $q_j$ in $(DG \setminus BC)$ and sharing a vertex with $q_i$ do
        Determine if $q_j$ is a candidate and update $PQ$ with it;
    end
end
$CC_f$ := $BC$;
```

Algorithm 7: Inflating

Lemma 7 *Inflating BC converts it into a single polyhedral-hull component, denoted as CC_f, in $O(n \log n)$ time.*

Proof 9 *All finite tetrahedra in $\mathbb{C}_{ext}$ can at most be added once to BC, since no tetrahedra are ever removed from it. The while loop terminates if BC does not contain any more deflated vertices. In the limit all n tetrahedra in $(DG \setminus BC)$ would have*

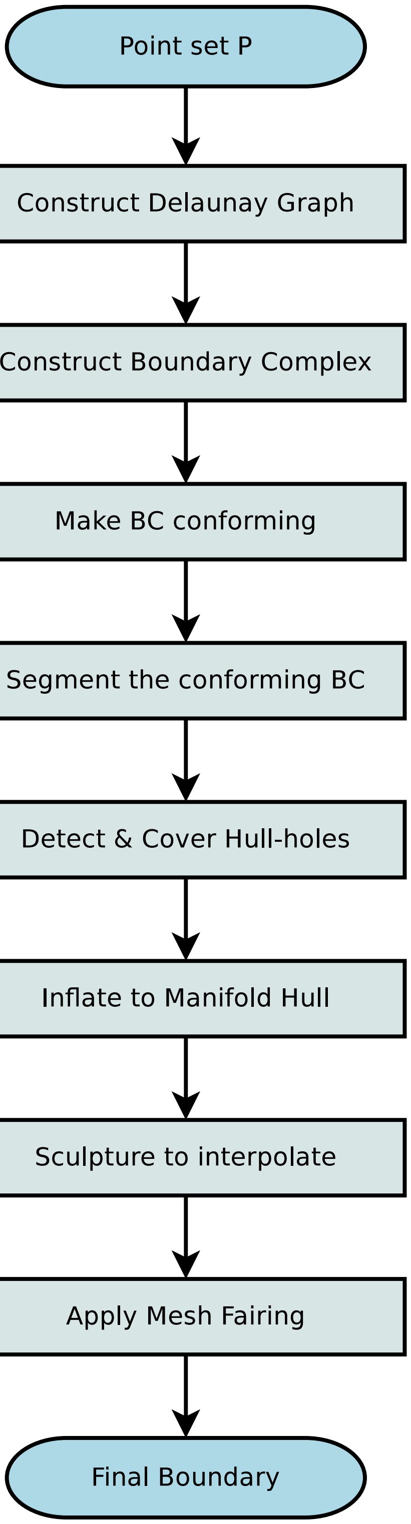
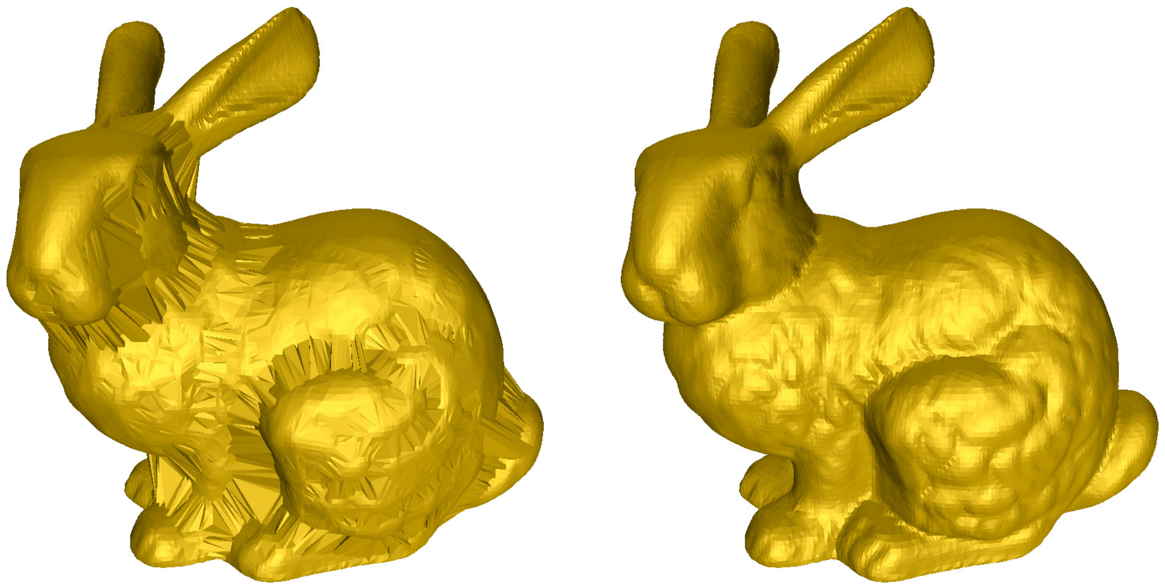


Figure 5.3: The steps of our algorithm in order.



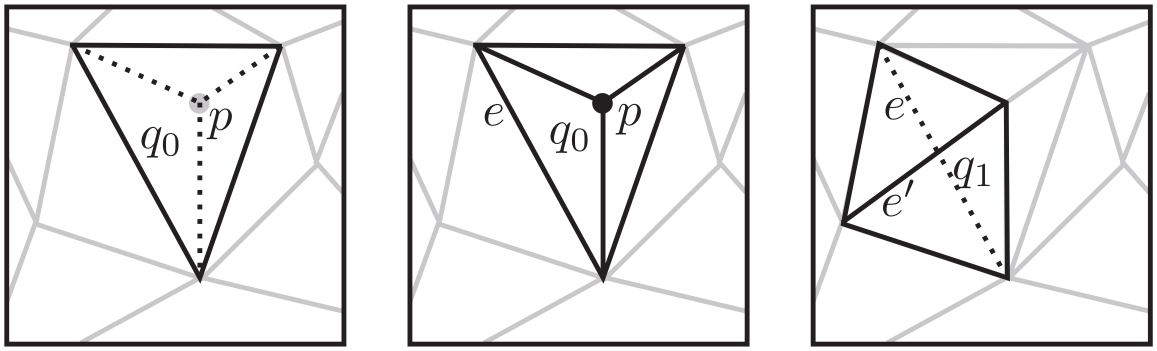
(a) From $H(DG)$ (b) From $H(CC_f)$

Figure 5.4: Sculpturing with $\lambda(t)$: a) Directly from the convex hull boundary - a large hole in the bottom leads to a hollowing-out of the object. b) From the inflated boundary complex, both from inside and outside.

been added to BC, resulting in $H(BC)$ becoming the convex hull. The convex hull is a manifold surface. The operation of updating the local neighborhood of a tetrahedron in the priority-queue is $O(\log n)$. The complexity of the algorithm is therefore $O(n \log n)$.

5.3 Sculpturing

$H(CC_f)$ is a manifold surface interpolating many of the given points but it may still contain some of the given points in its interior. For obtaining a manifold interpolating boundary B, we need to expose the interior vertices onto the boundary. We do this by adapting the *sculpturing* method described by Boissonnat [17] in such a way as to address its weaknesses, namely, quickly falling into local minima, restriction to



(a) Interior vertex (b) Exposed vertex (c) Flipped edge

Figure 5.5: a) Tetrahedron q_0 with single boundary triangle. b) Removal of q_0 exposes its interior vertex p. c) Removal of tetrahedron q_1 flips edge e to e'.

genus 0 and the bias caused by removing from the outside only, as described later. Figure 5.4 illustrates well the difference in resulting quality.

5.3.1 Sculpturing Operation as defined by Boissonnat

Boissonnat states that starting from the convex hull $H(DG)$, a tetrahedron q can be removed from a polyhedral hull H if it satisfies either of the following conditions (see Figure 5.5):

- q has exactly 3 vertices, 3 edges and 1 triangle in H: this will add the single interior vertex onto it.
- q has exactly 4 vertices, 5 edges and 2 triangles in H: the equivalent of an edge-flip.

Boissonnat has further noted that the following can be proved: by combinatorially removing tetrahedra in this fashion from the convex hull of the point set, $H(DG)$, any polyhedron of genus 0 in DG can be obtained. We generalize this statement to state that any boundary surface B' in DG can be sculptured from another boundary surface B in DG as long as B' is contained entirely on or interior to B. Its genus is retained.

5.3.2 Sculpturing From A Close Approximation

We therefore start sculpturing from $H(CC_f)$. Since it is already a much better approximation to B_{min} than $H(DG)$ in terms of shape and genus, it is much less likely to fall often into an unacceptable local minima. We sort tetrahedra for removal by largest $\lambda(t)$ in their triangles t in $H(CC_f)$. However, we do not want to remove all removable tetrahedra (i.e. all possible edge-flips), since the bias towards removal from outside is likely to hollow out the surface. Hence we propose that sculpturing be done simultaneously from spaces both outside and inside of $H(CC_f)$ as described next. This process will try to expose all points interior to $H(CC_f)$ on the boundary. We do not consider tetrahedra as removal candidates if their triangle with smallest circumradius is in the boundary, as we have seen through our experimentations that it leads to increased curvature. Therefore there could still be tetrahedra in the space between the inside and the outside of $H(CC_f)$. These are like thin membranes inside the boundary. Hence, our sculpturing process includes a membrane removal operation to correct this.

5.3.3 Sculpturing From Both Inside and Outside

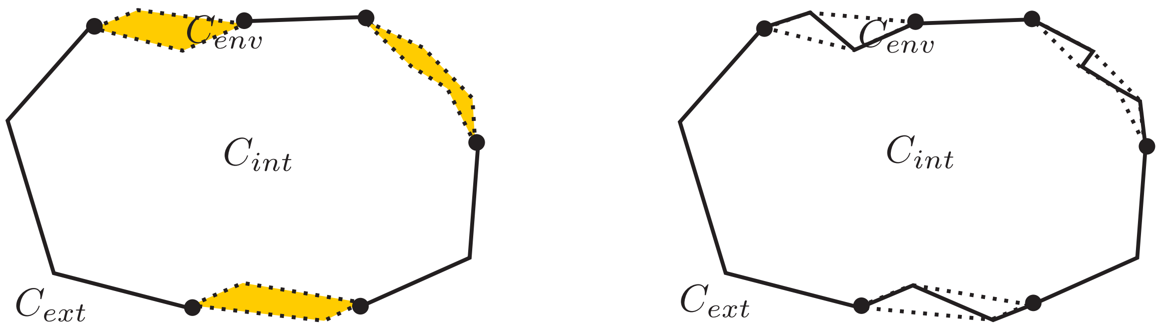
We decompose $DG_{inf} \setminus T(CC_f)$ into simplicial complexes, which are connected sets of face-connected tetrahedra:

- $\mathbb{C}_{ext}$
- a usually large and single $\mathbb{C}_{int}$ interior to $H(CC_f)$
- and a set of smaller ones, $\{\mathbb{C}_{env}\}$, which represent the thick parts of CC_f. We name $\mathbb{C} \in \{\mathbb{C}_{env}\}$ as *bubbles*, as in bubble-wrap.

The CC_f we obtained after inflation can be thought of as a *crust* triangulation, which is *thick* where there are bubbles, offering ambiguous local interpolation for its vertices due to multiple umbrellas in CC_f, and *thin* elsewhere (single umbrella). By merging all tetrahedra inside bubbles either with $\mathbb{C}_{ext}$ or $\mathbb{C}_{int}$, we can transform CC_f into the manifold interpolating boundary B. This corresponds to sculpturing simultaneously from inside and outside.

$\{\mathbb{C}_{int}\}$ contains all connected simplicial complexes in $DG_{inf} \setminus CC_f$ which are finite and incident to manifold vertices v_i in CC_f. These manifold v_i imply their unique interpolation in $H(CC_f)$. $\{\mathbb{C}_{int}\}$ may be empty for some contrived point sets, such as a regular octahedron consisting of eight tetrahedra, which share its single interior vertex. Our sculpturing algorithm handles such cases also well.

Let $\{\mathbb{C}_{amb}\} = \mathbb{C}_{ext} \cup \{\mathbb{C}_{int}\}$. Let T_M denote the set of triangles which will not be changed by sculpturing. T_M consists of all triangles with both incident tetrahedra in



(a) Boundary with bubbles (b) Manifold boundary

Figure 5.6: 2D illustration: a) Bubbles shaded yellow, T_M shown as continuous lines, T_B as dotted lines. b) Sculpturing the tetrahedra from these bubbles makes the boundary thin and manifold.

different $\mathbb{C}_i \in \{\mathbb{C}_{amb}\}$. Let T_B denote the set of triangles with one incident tetrahedron in $\{\mathbb{C}_{amb}\}$. Each triangle $t_i \in T_B$ has therefore its other incident tetrahedron in a $\mathbb{C}_i \in \{\mathbb{C}_{env}\}$. T_B as the boundary of bubbles is therefore also the boundary for sculpturing in Algorithm 8 (see Figure 5.6).

```
**Input**: $\{\mathbb{C}_{amb}\}$, $\{\mathbb{C}_{env}\}$, $T_B$
**Output**: $\{\mathbb{C}_{amb}\}$, $\{\mathbb{C}_{env}\}$, $T_B$
$PQ :=$ priority-queue of candidate pairs $(q_i, \mathbb{C}_{amb})$, sorted by inverse $max(\lambda(t_i$ in $q_i$ in $\mathbb{C}_{amb}))$;
while $PQ \neq \{\}$ do
    Remove first $(q_i, \mathbb{C}_{amb})$ from $PQ$;
    $\mathbb{C}_{amb} := \mathbb{C}_{amb} \cup q_i$;
    $\mathbb{C}_{env}(q_i) := \mathbb{C}_{env}(q_i) \setminus q_i$;
    foreach $q_j$ in $\{\mathbb{C}_{env}\}$ sharing a vertex with $q_i$ do
        foreach $\mathbb{C}_i \in \{\mathbb{C}_{amb}\}$ do
            if $(q_j, \mathbb{C}_i)$ is candidate then
                $PQ := PQ \cup (q_j, \mathbb{C}_i)$;
            end
        end
    end
end
```

Algorithm 8: ExposeInteriorPoints

Lemma 8 *The total computation in Algorithm 8 is* $O(n \log n)$ *.*

Proof 10 *The above-mentioned sculpturing rules re-label a sub-set of tetrahedra in* $\{\mathbb{C}_{env}\}$ *as being in* $\mathbb{C}_{amb}$*. At most all of those tetrahedra are re-labeled once. The operations in the inner loops are local and therefore of constant time. Since the re-labeled tetrahedra are stored in a priority-queue, its total complexity is* $O(n \log n)$*.*

5.3.4 Membrane Removal

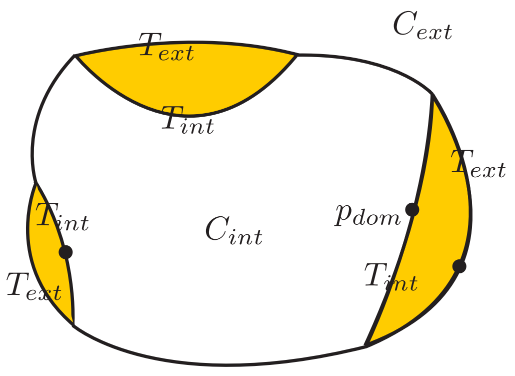
After the above algorithm is terminated, bubbles may still exist. This is so because of the condition of not sculpturing tetrahedra which have their triangle with smallest circumradius in the boundary. Therefore we add a step in which we test if contiguous sets of triangles can be removed from CC_f such that the entire set of tetrahedra per remaining bubble merges with $\{\mathbb{C}_{amb}\}$.

We call the two sets of triangles which a bubble shares with $\mathbb{C}_{ext}$ and $\mathbb{C}_{int}$ as their *membranes* T_{ext} respective T_{int}.

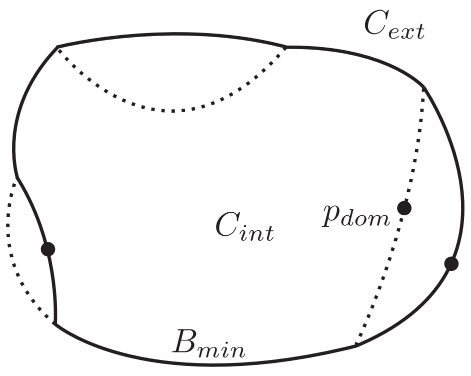
A membrane can be removed from CC_f if it does not contain any interior manifold vertices v_i in T such that v_i has a single umbrella in CC_f. Otherwise, this would disconnect vertices.

T_{ext} can only be removed if T_{int} is edge-connected, in order to retain CC_f as a connected component.

A membrane cannot be removed if it would add another umbrella in CC_f to a vertex already having an umbrella in it.



(a) Boundary with membranes



(b) Manifold boundary

Figure 5.7: 2D illustration of membrane removal: a) Thick CC_f with membranes (T_{int}, T_{ext}) for the bubbles. b) Removal of membranes yields a thin boundary. Handling of the membrane-pairs: On the left, T_{ext} is removed, since T_{int} contains an interior vertex. At the top, T_{int} is removed to minimize curvature. On the right, both membranes have interior vertices, therefore T_{int} is removed, leaving the dominantly interior vertex p_{dom}.

If both membranes of a bubble are removable, we remove the one with higher mean curvature summed over its edges.

If none is removable, we label the interior manifold vertices of T_{int} as *dominantly interior*, since they will not be interpolated in B due to their large distance from the surface (see Figure 5.7). For the same reason we do not require that the DG contains a Hamiltonian cycle, though in the case of $\mathbb{R}^2$, as already mentioned in Section 4.4, it is required, at least in theory. Membrane removal is done by Algorithm 9.

The boundary surface B (unsmoothed) for the given point set P is defined as $H(T_M \cup T_B)$.

Lemma 9 *Algorithm 9 converts $T_M \cup \{\mathbb{C}_{env}\}$ into a connected manifold triangulation, resulting in a $H(CC_f)$ which interpolates all but the dominantly interior vertices in P and the computational complexity is $O(n)$.*

```
Input: $\{\mathbb{C}_{amb}\}$, $\{\mathbb{C}_{env}\}$, $T_B$
Output: $T_B$
foreach $\mathbb{C}_i \in \{\mathbb{C}_{env}\}$ do
    $T_{ext} := T(\mathbb{C}_i) \cap T(\mathbb{C}_{ext})$;
    $T_{int} := T(\mathbb{C}_i) \cap T(\{\mathbb{C}_{int}\})$;
    if ($T_{ext}$ is removable) $\wedge$ ($T_{int}$ is removable) then
        if $H_e(T_{int} \cup T_M) < H_e(T_{ext} \cup T_M)$ then
            $T_B := T_B \setminus T_{ext}$;
        end
        else
            $T_B := T_B \setminus T_{int}$;
        end
    end
    else if ($T_{ext}$ is removable) then
        $T_B := T_B \setminus T_{ext}$;
    end
    else if ($T_{int}$ is removable) then
        $T_B := T_B \setminus T_{int}$;
    end
end
```

Algorithm 9: Membrane Removal

Proof 11 *Each $\mathbb{C}_i \in \{\mathbb{C}_{env}\}$ is merged to a $\mathbb{C}_j \in \{\mathbb{C}_{amb}\}$. This is done by removing their shared set of triangles, the membrane. A membrane is never removed if this operation would split CC_f and therefore CC_f remains a single connected component. A membrane is also never removed if it contains interior vertices in its triangulation. Therefore only dominantly interior vertices can become disconnected from $H(CC_f)$, contained in membranes remaining interior to it. It is also manifold because then $\{\mathbb{C}_{env}\}$ for $H(CC_f)$ does not contain any tetrahedra and the merging operations have not introduced additional umbrellas at its vertices. For both the operations of classifying and removing membranes, each triangle in CC_f is queried at most once. Therefore the complexity of this algorithm is $O(n)$.*

5.4 Surface Smoothing

Our choice for the minimization criterion can not minimize H_e exactly. As already stated, if we were to use curvature as the criterion, then it would require querying adjacent triangles of an edge. This interdependency among triangles not only makes the search for the optimal triangle set very complex, but also from our experiments, we have seen that applying the H_e criterion directly tends more often to very quickly fall into local minima. Using λ gets us a close approximation of $\sum H_e$ for the B.

To get even closer to the optimal triangle set, we apply a simple mesh fairing operation, which minimizes H_e locally. We have found that this substantially reduces $\sum H_e(B)$, taking the computed surface even closer towards $\sum H_e(B_{min})$. Algorithm 10 iterates over all edges $e_i \in B$ and flips e_i where that operation reduces H_e locally in B.

An edge e_i in the closed manifold triangulation B has the incident pair of triangles $T(e_i)$ in B, which are contained by a tetrahedron q in DG. If there exist both an edge e_k and a triangle pair $T(e_k)$ in q which are not in B, we name the operation of replacing them in B with $e_i, T(e_i)$ to create B' as *edge-flipping* and call e_i as *flippable*, since it guarantees that B' remains an interpolating manifold. We only flip those edges which lead to further minimization of curvature for B.

We define the difference in mean curvature for flipping an edge e_i in B as $\Delta H_e(e_i) = H_e(B') - H_e(B)$, which requires evaluating H_e for all the edges in the tetrahedron q.

Note that to the best of our knowledge there is no known upper bound for the

```
Input: $B$
Output: $B$
$E$:= set of all edges in $B$;
while $E \neq \{\}$ do
    Remove first $e_i$ from $E$;
    if ($e_i$ is flippable) $\wedge(\Delta H_e(e_i) < 0)$ then
        Edge-flip $e_i$ in $B$;
        foreach $e_t$ in $T(e_i)$ do
            $E := E \cup e_t$;
        end
    end
end
```

Algorithm 10: Mesh Fairing

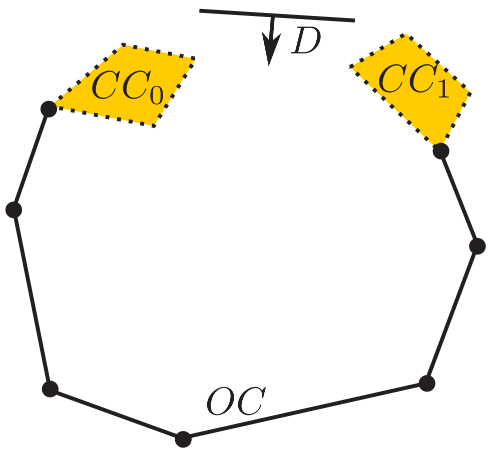
complexity of flipping edges in a surface mesh until such a non-intrinsic global value is minimized, unless points are densely spaced with minimum angle, as described in Cheng and Jin [24]. Although based on our experimentations (see also Table 6.2) it seems to be linear and fast for practical cases.

5.5 Segmenting Deflated from Closed Components

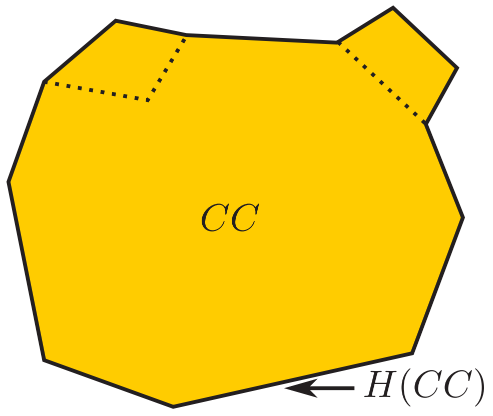
5.5.1 Hull-Holes

We describe next the details of procedures for detecting and covering hull-holes in a BC, which as mentioned earlier is done prior to applying the inflating operation. A hull-hole exists in a BC at the places where a deflated component OC and one or more closed components CC are connected together. We therefore have to segment these types of components in the BC.

Adding a hole-cover D_i transforms this subset of deflated and closed components



(a) Hull-hole



(b) Covered hole

Figure 5.8: a) Inside of closed components (CC) is shaded yellow, OC is shown using continuous edges, hole-cover is D. b) Adding D transforms the BC into a single CC with the manifold hull $H(CC)$.

into a single closed component (see Figure 5.8). Let us note here that D_i is made up from entities in DG as a whole, and not just from BC. We can visualize the deflated OC as being surrounded by the CCs connected to its edges. Hence, we shall denote the side with CCs incident to triangles in $D_i \setminus DG$ as the outside of the L enclosing the hull-hole and the other side, where both OC and other CC can be incident, as its inside. We traverse the edges of L in an orientation consistent with this.

BC has no boundary triangles, since each edge in BC has ≥ 2 incident triangles, which are also in BC. Any triangle in the hole-cover must therefore be such that it is incident to an $H(CC)$ but not to any OC. Otherwise in places where the hole-cover touches the deflated component, the hull would remain deflated.

5.5.2 Properties of a Conforming BC

Let us recall the following. The L enclosing a hull-hole is made up of edges shared by an OC with one or more CCs. The OC has to be on the inside of L. Lastly, covering of hull-holes *must* not add further deflated entities. In order to ensure the above, we will need entities in BC to conform to the following properties:

1. The shared entities between an OC and associated CCs must not include any isolated deflated vertex.

2. The inside OC and all CCs associated with the hull-hole must be uniquely identifiable. This requires that each edge in L is contained in exactly two components (made up of either CCs incident to L or the inside OC with incident deflated triangles).

3. An OC must satisfy the following: Any vertex v in the OC must be *manifold* with respect to OC, that is, there exists a single umbrella in DG which contains all its incident deflated entities. The latter are contained in the OC. Since some of the umbrella triangles may be in DG but not in BC, it should be noted that v need not have an umbrella in BC.

We call vertices or edges not satisfying the above properties as *non-conforming* entities. As previously mentioned, we have observed in the many point sets we have experimented with, that cases not conforming to these properties are indeed quite rare. Where found, they are mostly contained in sparsely sampled regions. Later in

Sub-section 5.7, we specify a simple procedure which converts non-conforming entities into *conforming* entities.

5.5.3 Topology of Hull-Holes

Assuming a conforming BC, let an OC boundary (OCB) denote a closed edge chain in an OC such that each edge in the OCB is shared with a CC. Let $\{CC\}$ denote this set of CCs associated with this OCB. Further, edges in OCB are traversed so that OC is on the inside. If we cannot find a manifold triangulation T on or interior to $H(\{CC\})$ with the boundary of T exactly coinciding with the OCB, then this OCB is likely to be an L enclosing a hull-hole. Hence, we say that this is a *potential-hole-OCB*.

5.5.4 Locating the OCBs

We use the following procedure for locating the OCBs in a conforming BC.

Let SE denote the set of boundary edges of all OC in BC. Since $H(BC)$ is conforming, its last two properties require that the edges $e \in SE$ incident at any vertex in $H(BC)$ can be mapped without self-intersections onto a plane. Because of that local property it follows that the loops and trees formed in SE also do not self-intersect if mapped onto a plane.

First we traverse subsequent edges in SE in consistent direction in order to locate OCB loops in it, such that at one side, which we denote as the outside of the OCB, no other $e \in SE$ are contained.

Then, we traverse again edges in SE to find new loops as OCBs, but now require that they contain at least one edge in $SE' = SE \setminus OCB$ and permit edges $e \in SE'$ to be contained at their outside, as long as they are not contained in any loop in SE (see Figure 5.9).

Let $\{OCB\}$ denote the set all of OCBs located using the above algorithm.

Lemma 10 *Edges of SE in BC can be assigned to $OCBs$ in $O(n)$ time such that only trees remain in $SE \setminus \{OCB\}$.*

Proof 12 *The above-mentioned algorithm maps each edge $e \in SE$ to an OCB which is contained in a loop in SE. Therefore $SE \setminus \{OCB\}$ consists entirely of trees in SE. Since any $e \in SE$ can only be contained in at most two loops, it traverses each $e \in SE$ at most twice. Therefore the complexity of this operation is $O(n)$.*

5.6 Detecting and Covering Hull-Holes

5.6.1 Classifying Potential-Hole-OCBs

- If the OCB has more than one associated CC (as is the case in Figure 5.2 or 5.9), it is a potential-hole-OCB, since it will not have a covering triangulation T in $H(\{CC\})$. Otherwise, this would edge-connect the CCs by deflated edges, and then the OCB would not be a boundary of the OC.

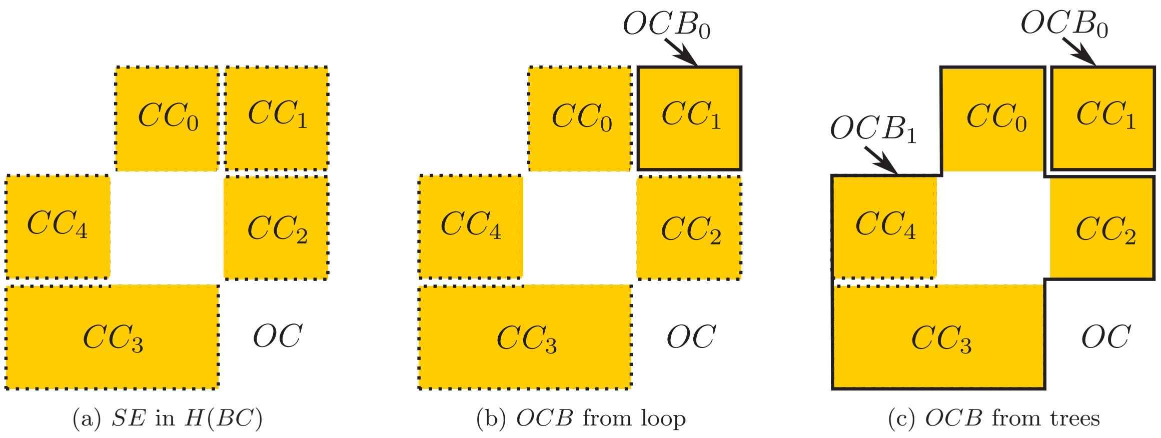


Figure 5.9: a) Hull-hole (in center) with multiple CC outside (shaded yellow) and another, CC_1, with genus$=0$. Edges in SE are dotted. b) The single loop is labeled as OCB_0. c) Multiple trees are assembled to OCB_1.

- If the OCB has just one single associated CC with genus 0, there will always exists a covering triangulation in $H(CC)$ for this OCB.
- If the OCB has just one single associated CC, but with genus > 0, there may or may not exist a covering triangulation in $H(CC)$, making this a potential-hole-OCB.

For the last case we use the following test to determine whether its genus is > 0. If there exists an orientable triangle strip $T \in H(CC)$ such that two triangles $t_i, t_k \in T$ contain the same edge $e \in OCB$, then this is a potential-hole-OCB, i.e. not covered.

The linearithmic Algorithm 11 detects the existence of such a triangle strip. It requires the following definition:

Each edge in OCB has a pair of incident triangles in $H(CC)$. These triangles can be classified as belonging to the *top-side* or the *bottom-side*, by assigning OCB edges a consistent orientation.

```
**Input**: $OCB$
**Output**: IsOpen
$E_{curr}$:=$OCB$;
IsOpen := false;
while $E_{curr} \neq \{\}$ do
    Remove an $e_i$ from $E_{curr}$;
    $t_i$:=top-side triangle in $H(CC)$ incident to $e_i$ and outside $E_{curr}$ ;
    foreach $e_j$ in $t_i$ do
        $E_{curr} := E_{curr} \oplus e_j$;
        if $(e_j \in E_{curr}) \wedge (e_j \in OCB)$ then
            IsOpen=true;
        end
    end
end
```

Algorithm 11: Test if OCB with single CC encloses a potential hull-hole.

5.6.2 Determining Hole-OCBs

Inflating adds tetrahedra incident to deflated entities to make them non-deflated. Many of the hull-holes in potential-hole-OCBs get covered by this inflating operation. However, since this operation adds tetrahedra by order of edge length, it is not able to cover certain hull-holes in a manner that minimizes our criterion. This happens when an OC has edges which are longer than all edges in any hole cover D for that hole. It is only to such hull-holes that we would add a hole cover prior to inflating. On the other hand, we prefer to handle all other potential hull-holes by inflating, since for those due to their configuration a hole cover may add many more triangles and result in a local minimum.

We detect the desired hole-OCBs using the following heuristic:

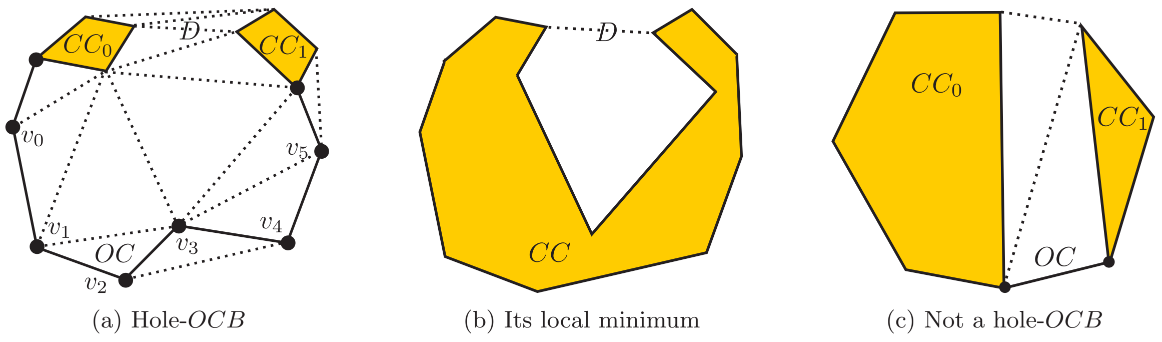


Figure 5.10: 2D example illustrating the process of determining hole-OCBs, CC are shaded yellow, deflated entities are shown as continuous lines and large dots, DG as dotted lines, hole-cover is D: a) Is a hole-OCB because the OC contains interior vertices ($v_{0..5}$). b) Inflating of the OCB in (a) leads to a local minimum, since interior edges in DG are longer than D. c) Not a hole-OCB since the OC contains no interior vertices. It can be inflated.

A potential-hole-OCB is designated as hole-OCB if there exists a vertex v_1 on $H(BC)$ not contained in any other open OCB, and the vertex is connected by an edge in $H(BC)$ to a vertex v_2 in that OCB. v_1 can be either in an OC or in the hull of a CC with genus $= 0$. If such a vertex does not exist, it means that there can not exist any deflated vertices in the OC inside the OCB and the inflation operation would be sufficient to cover it (see Figure 5.10).

5.6.3 Covering the Hull-Holes

To find a hole-cover for a hole-OCB, we determine a covering triangulation based on the following lemma.

We define the hull $H(V)$, for a set of vertices V in DG, as the set of triangles T in DG such that $V(T) \subseteq V$ and each edge e in T has ≥ 2 incident triangles in T. Vertices in $V \setminus V(T)$ are *interior* to $H(V)$.

Lemma 11 *Let $H(V)$ be the hull for V, the set of all vertices in $\{CC\}$ associated with a hole-OCB. Let T be the set of triangles on and interior to $H(V)$. Then T contains all disks of triangles D_i in DG, which have that hole-OCB as their boundary and for which $V(D_i) \subseteq V$. $H(V)$ can be constructed for all hole-OCBs in BC in $O(n \log n)$ time.*

Proof 13 *The set of disks $\{D\}$ bounded by the hole-OCB can only contain triangles which have no external edges in $BC \cup (D_i \in \{D\})$ and whose vertices are entirely contained in V. Since $H(V)$ contains all triangles with these conditions, it follows that all such disks are in $H(V)$. Based on the definition of a hull for a set of vertices, $H(V)$ can be constructed by first evaluating the k triangles in DG incident to V. Then all its triangles having a single-triangle edge are removed to yield $H(V)$. This computation is of order $O(k \log k)$ for k triangles in $H(V)$. Since the triangles in $H(V)$ are contained in the $\{CC\}$ associated with the hole-OCB, and those do not overlap for different hole-OCBs, $k \leq n$ and total computation time therefore $O(n \log n)$.*

As part of this *hole-covering* operation we have added new entities to BC. Could this introduce new non-conforming entities? In the point sets we have investigated so far, this has not happened. We feel that this is so, because little of BC is changed by the hole-covering operation. Further, if at all some non-conforming entities are added, then we can easily handle these subsequently using the same procedure described later in Sub-section 5.5.2.

From above Lemma 11 it is clear that this hole-covering operation does not work if among the covering triangulations in DG there is none with vertices entirely in V. However this is the case only for thin and very sparsely spaced point sets for which the surface forms a saddle, resulting in a twisted hole-OCB[1] (e.g. in Figure 6.12e).

5.7 Handling Non-conforming Entities

Prior to performing the segmentation of BC_0, we have to determine and correct non-conforming entities. The linearithmic Algorithm 12 below guarantees this by adding a number of tetrahedra in DG to BC. Of course, we would like to add as few tetrahedra as possible.

For a non-conforming entity (vertex or edge) x_i in $H(BC_0)$ we name $T(x_i)$ as the set of its incident triangles in $H(BC_0)$. $V(T(x_i))$ is then the *1-neighborhood* for x_i on $H(BC_0)$.

We define a finite tetrahedron q_j in $\mathbb{C}_{ext}$ as *addable* to an entity x_i if q_j contains a triangle $t_k \in H(BC_0)$ which in turn contains x_i.

Let X_{nc} be the set of non-conforming vertices and edges in $H(BC_0)$.

We give the following intuitive explanation for the two successive main loops in this algorithm:

First we try to add only the hull for the 1-neighborhood of non-conforming entities

[1]It is however possible to close even such a *twisted hole* by letting the covering triangulation connect partially to vertices in $P \setminus V$. It can be determined by firstly relating the top and bottom sides of the OCB to its OC, then determining at these sides tetrahedra which are oriented clearly to one side and lastly walking the orientation along face-connected tetrahedra. This results in a simplicial complex similar to the above hull, containing the covering triangulation.

```
**Input**: $BC$, $X_{nc}$
**Output**: $BC$
while $X_{nc} \neq \{\}$ do
    $X'_{nc} = \{\}$;
    repeat
        $X'_{nc} = X_{nc}$;
        foreach $x_i \in X_{nc}$ do
            Add hull $H(V(T(x_i)))$ to $BC$;
            if $x_i$ is conforming then
                $X_{nc} := X_{nc} \setminus x_i$;
            end
            foreach non-conforming $x_k$ in $h$ do
                $X_{nc} := X_{nc} \cup x_k$;
            end
        end
    until $X_{nc} = X'_{nc}$ ;
    foreach $x_i \in X_{nc}$ do
        $T := \{\}$;
        $PQ :=$ priority-queue of $q_i$ addable to $x_i$, sorted by $\sum \lambda(t_i$ in $q_i$ not in $BC)$;
        while $x_i$ is non-conforming do
            Remove first $q_j$ from $PQ$;
            $T_j := t_j$ in $q_j$ not in $BC$;
            $T := T \cup T_j$;
            $BC := BC \cup T_j$;
        end
        foreach non-conforming $x_i$ in $T$ do
            $X_{nc} := X_{nc} \cup x_i$;
        end
    end
end
```

Algorithm 12: Correct nonconforming entities

to the BC_0. This adds relatively few triangles, makes most of these entities conforming. For the point sets we have experimented with, it creates only very rarely, new non-conforming entities.

For the non-conforming entities remaining after that step, we add all their incident tetrahedra to BC_0 such that the entities become interior, then resume with the first loop.

Lemma 12 *All non-conforming entities in BC_0 can be transformed into conforming ones in $O(n \log n)$ time.*

Proof 14 *The algorithm adds triangles incident to non-conforming vertices to BC, until all vertices in BC have become conforming. In the limit, at most all triangles in $DG \setminus BC$ would have been added to BC, yielding a CC such that $H(CC)$ is the convex hull. Determining for each added triangles, which of its vertices have changed their conforming state, is a local operation. Since at most n triangles are added, using priority queue operations, the total complexity of the algorithm is $O(n \log n)$.*

5.8 Complexity and Quality Analysis

Theorem 3 *Using the main algorithm in section 5.1, a closed, non-intersecting and manifold boundary B interpolating all but the dominantly interior points can be found in expected $O(n \log n)$ time, provided BC_0 contains no twisted holes.*

Proof 15 *The construction of the Delaunay Graph DG is of $O(n \log n)$ expected complexity as shown in Guibas and Stolfi [47]. Based on the DG we can construct the Boundary Complex BC_0 (Lemma 2) and make it conforming (Lemma 12) in order to detect hull-holes (Lemma 10). We then cover all non-twisted holes in that BC (Lemma 11) and inflate the BC to a CC with a manifold hull (Lemma 7). Then we sculpture that CC (Lemma 8) and remove membranes (Lemma 9) to extract a B which interpolates all but dominantly interior points, with total $O(n \log n)$ complexity.*

5.9 Concluding remarks

Pursuing the idea of the intrinsic shape in a point set, the method developed for boundary shape construction in $\mathbb{R}^2$ extends well into $\mathbb{R}^3$, although requiring a number of enhancements like hull-hole covering. Implementation details of this new method, and results of experiments on various unorganized point sets are presented and analyzed in the next chapter. As we shall see, the results from our new method are superior to the results of previous algorithms, especially for non-dense and locally non-uniform point spacing.

Chapter 6

Results

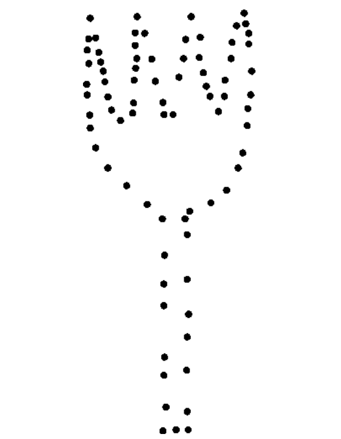
In this chapter we present and analyze the results from our implementations of boundary shape construction methods, for both $\mathbb{R}^2$ and $\mathbb{R}^3$.

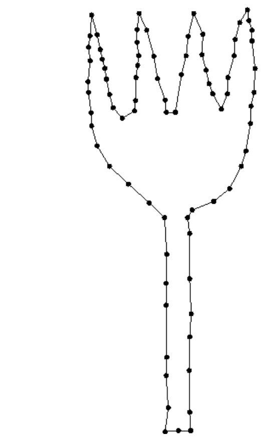
6.1 Results for $\mathbb{R}^2$

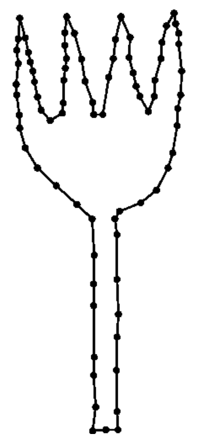
The complete source-code for our method in $\mathbb{R}^2$ is available online [65]. We use existing implementations for Delaunay triangulation from CGAL [48] and disjoint sets from Stefanov [72].

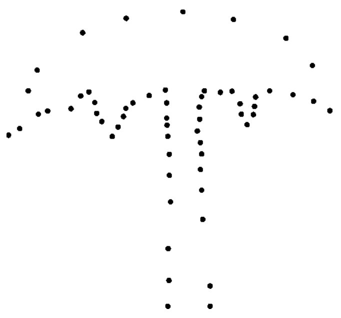
6.1.1 Comparison

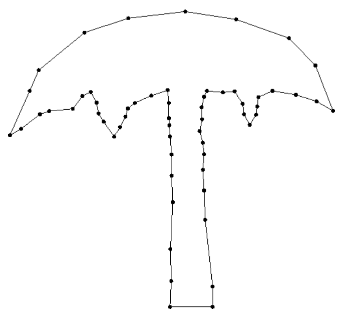
We have tested our method with a very large number of point data sets (some examples are shown in Figures 6.1, 6.2, 6.3, 6.4). Since many other algorithms also

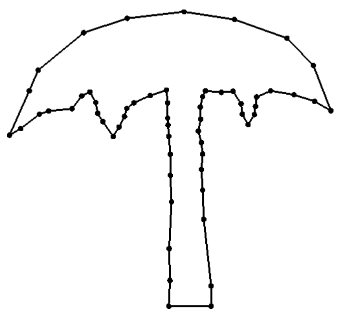

(a) Point set [4]

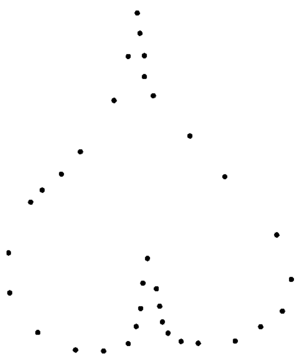

(b) GathanG

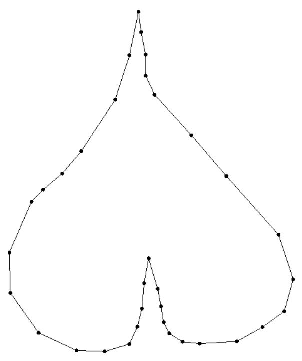

(c) Our method


(d) Point set [26]


(e) GathanG


(f) Our method


(g) Point set [75]


(h) GathanG

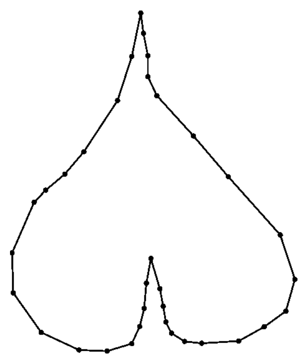
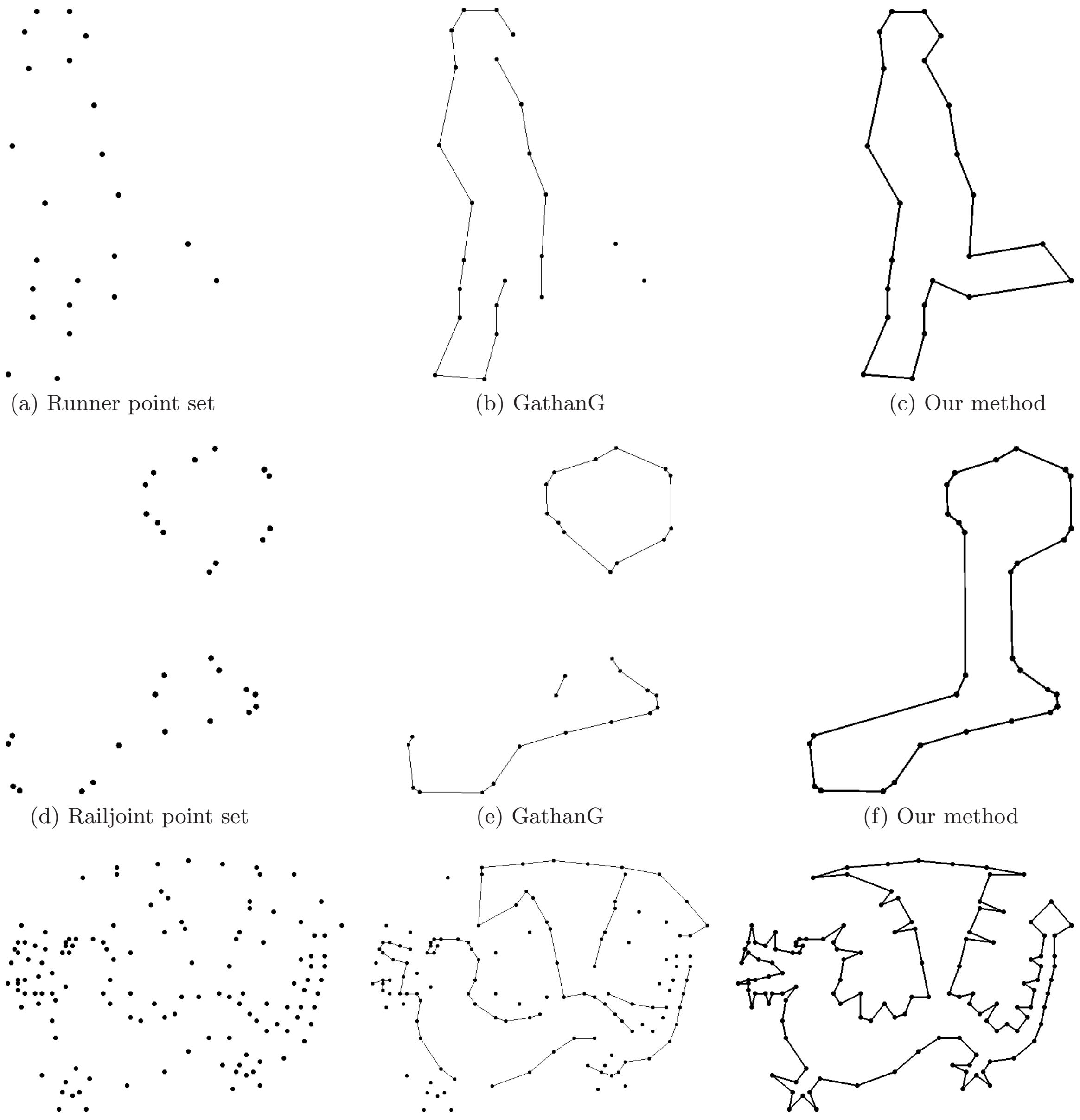

(i) Our method

Figure 6.1: Boundary construction of nicely sampled point sets: Left column: Point set. Center column: *GathanG* with default parameters [26]. Right column: Our manifold result.

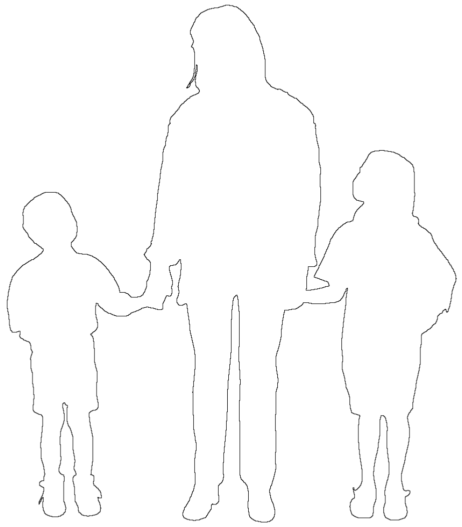


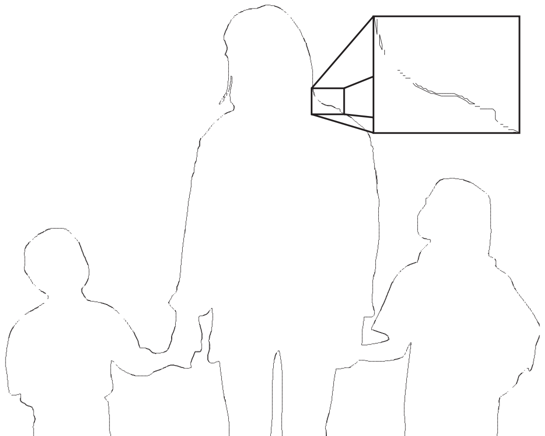
(a) Runner point set (b) GathanG (c) Our method

(d) Railjoint point set (e) GathanG (f) Our method

(g) Dragon point set (h) GathanG (i) Our method

Figure 6.2: Boundary construction of challenging point sets: Left column: Point set. Center column: *GathanG* with default parameters [26]. Right column: Our manifold result. a) Point set sub-sampled from a silhouette video image. d) Rail-joint, an engineering part [66]. g) Dragon point set, with many sharp corners.


(a) 10k points


(b) GathanG

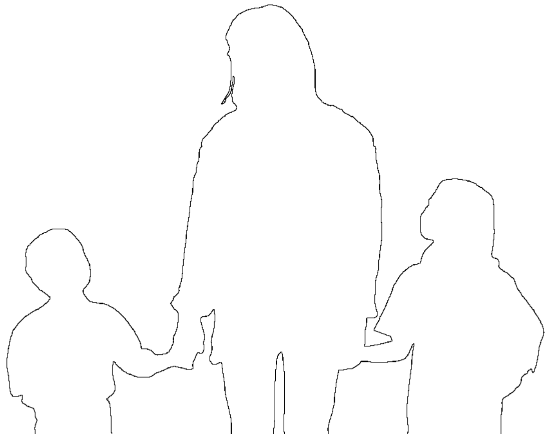
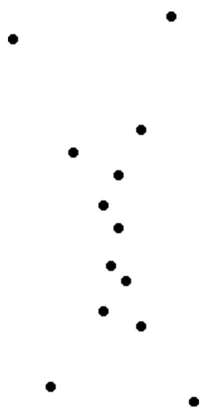
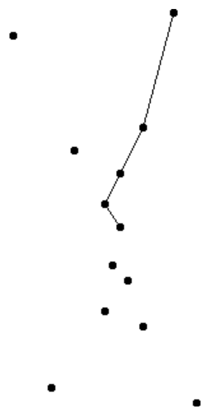
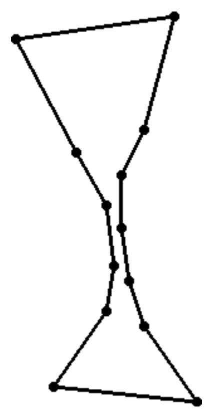
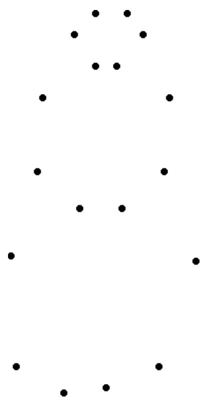

(c) Our method

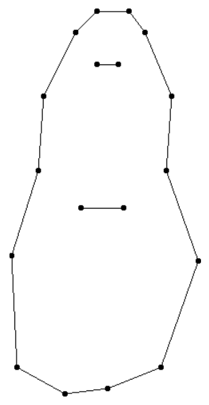
Figure 6.3: 10000 points sampled from silhouette image. a) Point set. b) *GathanG* with default parameters [26]. Note the false connections, disconnections and doubled boundaries. c) Our method constructs a closed manifold, even for the extremely close boundaries.

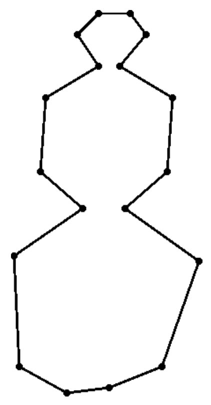

(a) Close curves

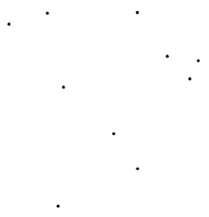

(b) GathanG

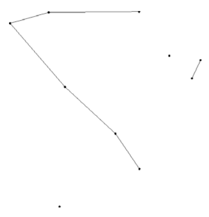

(c) Our method

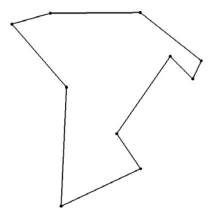

(d) 3 circles

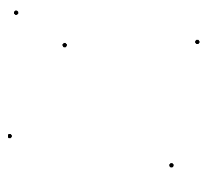

(e) GathanG

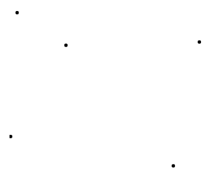

(f) Our method


(g) Random, 10


(h) GathanG


(i) Our method


(j) Concave, 5


(k) GathanG

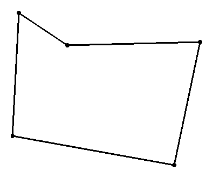

(l) Our method

Figure 6.4: Left column: Point set. Center column: *GathanG* with default parameters [26]. Right column: Our manifold result. a) Shape with extremely narrow portion. d) 10 random points. g) Three loops [66]. j) Concave polygon.

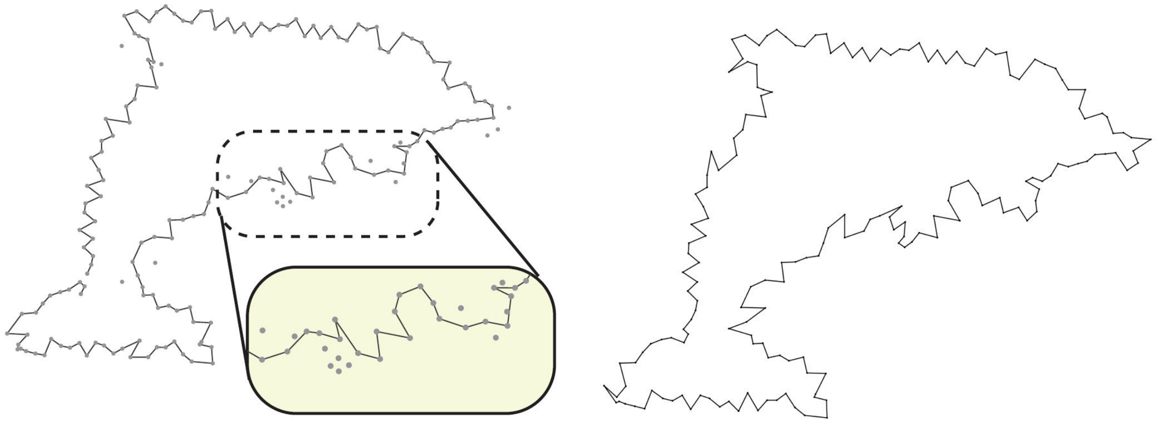

(a) Mehra et al. (b) Our method

Figure 6.5: Constructing the shape boundary from a noisy set of points. a) Figure presented in Mehra et al. [61]: It eliminates outliers rather arbitrarily, and does not fulfill its aim of producing a closed shape. b) Result of our method: It interpolates all the points and approximates very well B_{min}.

work for typical large point sets such as Figure 6.3a which are usually uniformly and densely spaced, we focus on showing critical details of point sets which sampling-oriented reconstruction algorithms have not been able to handle correctly and efficiently (see Figure 6.4). We compare our results for a number of difficult point sets with $GathanG$ from Dey and Wenger [26], using default parameters of $minAngle = 10$ and $maxIter = 4$ (see center column in the above-mentioned figures). Zeng et al. [75] has already compared their results with many of the other methods mentioned earlier. The results demonstrate that closely spaced boundary parts and sharp corners as artifacts of sparsely and non-uniformly spaced points are all handled very well by our algorithm.

Our method handles also well, shape boundaries which have been perturbed by a significant amount of noise (see Figure 6.5). Once the topology of the boundary

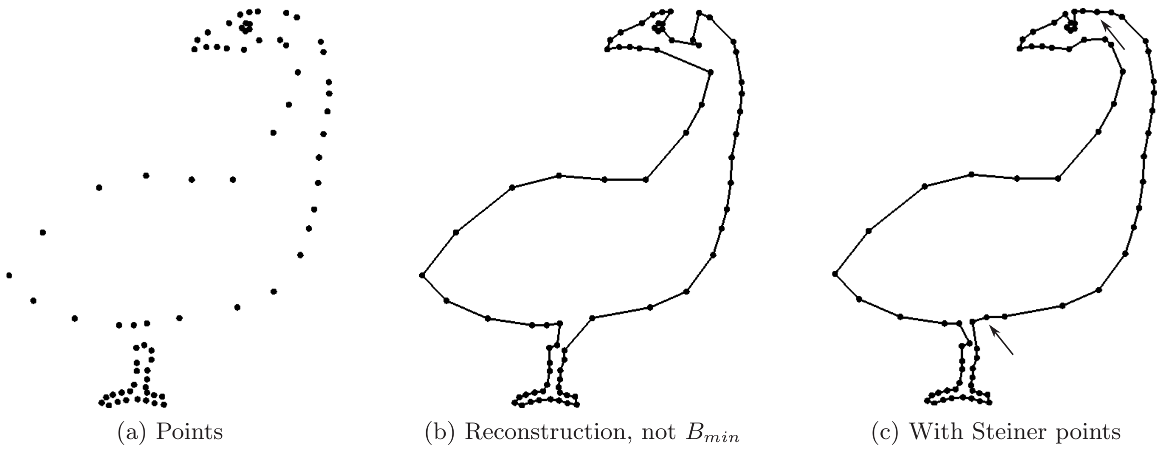


Figure 6.6: a) Goose point set from Amenta et al. [8]. It does not represent a solid. b) B construction by our method is not B_{min}. c) After two Steiner points have been added to the point set (indicated by arrows), B_{min} is constructed correctly.

shape is created, outliers could more effectively be detected and eliminated by a post-processing step.

We demonstrate the limits of our method by showing two point sets for which the constructed shape is not the desired one. In the first case (see Figure 6.6) this is due to poor point spacing in certain places of the desired boundary. It is easy to see this, because after insertion of just two Steiner points, in the places where points are poorly spaced, our algorithm produces the desired result. In the second case (see Figure 6.7) our algorithm yields a local minimum. Both examples can be constructed correctly by using an exhaustive search algorithm such as in Ohrhallinger and Mudur [66] (see Appendix A), but with much higher complexity than $O(n \log n)$.

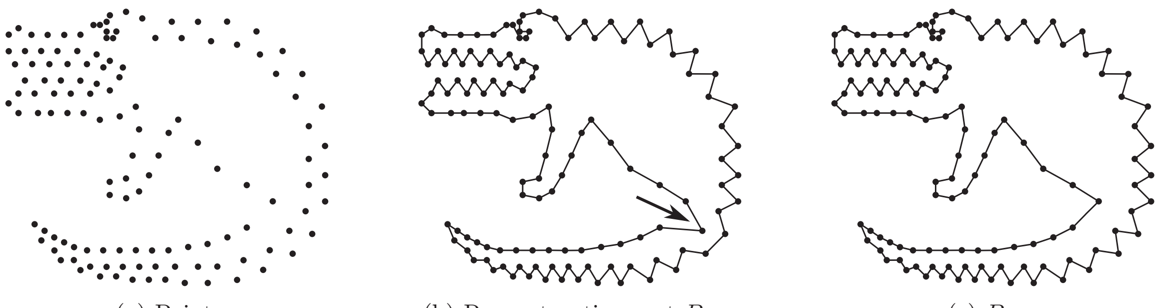


Figure 6.7: a) Crocodile point set from Ohrhallinger and Mudur [66]. b) B construction by our method has a local minimum (see arrow). c) Exhaustive non-linearithmic search as in Ohrhallinger and Mudur [66] yields B_{min}.

6.1.2 Class of Well-Spaced Point Sets

Our method reconstructs B_{min} faithfully and in linearithmic time for a class of point sets which we shall call X. Since the condition of closedness is global, it is not possible to give a point spacing criterion. The latter operates on a local neighborhood and thus can not consider the configuration of points outside that neighborhood. Formulating a criterion to precisely specify X is difficult.

We, however, give below a criterion which is somewhat restrictive (i.e. permitting only a sub-class of point sets in X):

Definition 3 *Let $B_{min}(P)$, a manifold closed boundary interpolating the point set P, denote the desired boundary. A vertex v in $B_{min}(P)$ is said to be* strictly well-spaced *if its two neighbor vertices in $B_{min}(P)$ are also its nearest neighbors in P by Euclidean distance. A vertex v in $B_{min}(P)$ is said to be* well-spaced *if either itself or its two neighbor vertices in $B_{min}(P)$ are strictly well-spaced. A point set P is said to* *be well-spaced if each $p \in P$ is well-spaced in $B_{min}(P)$.*

Note that edges between strictly well-spaced points are also contained in the nearest neighbor graph (NNG) for P.

Conjecture 1 *For a well-spaced P its $B_{min}(P)$ can be found in linearithmic time.*

We base our conjecture on the following observation: Non-neighbor vertices in $B_{min}(P)$ are only connected by single edges, and its cycle has only holes of single edges in BC, as opposed to edge-chains.

Actually, we would require a proof that inflating and sculpturing are deterministic such that their results do not depend on their heuristic ordering. We have not been able to construct any counter-example and are investigating further on developing a proof for this conjecture.

Figure 6.8 shows an example of a well-spaced point set. Note that each point which is manifold in BC is also necessarily strictly well-spaced, while the reverse is not true.

In practice our method works for many more point sets than those which satisfy above condition. We give some intuitive reasoning why this is the case: the above condition guarantees that $B_{min} \subseteq BC$ and the enclosing boundary of BC already equals B_{min}. If that condition is not fulfilled, then there exist edges in B_{min} which are not in BC. In such cases the two steps of *Inflating* and subsequent *Sculpturing* provide a good heuristic to locate those.

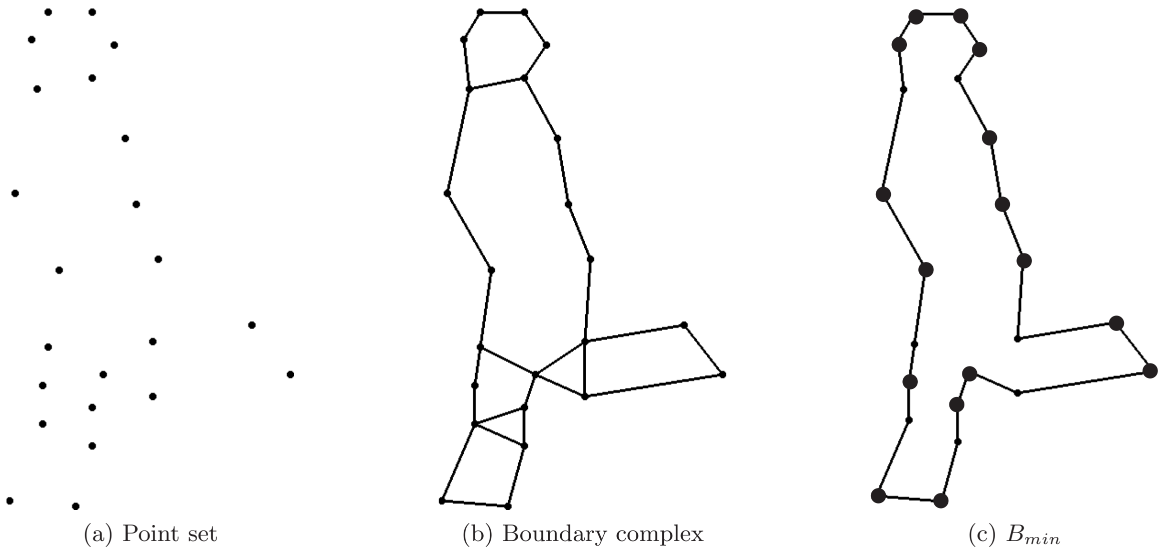


Figure 6.8: a) Point set P. b) Its BC. c) $B_{min}(P)$ with strictly well-spaced points enlarged. Each non-strictly well-spaced point has as neighbours on $B_{min}(P)$ strictly well-spaced points since P is a well-spaced point set.

This works also where points are sparsely placed on B_{min} but are reasonably uniformly spaced. However, in cases where points are too non-uniformly spaced, such that unrelated boundaries are close, these would be wrongly connected (see Figure 6.6), while only affecting the reconstruction locally.

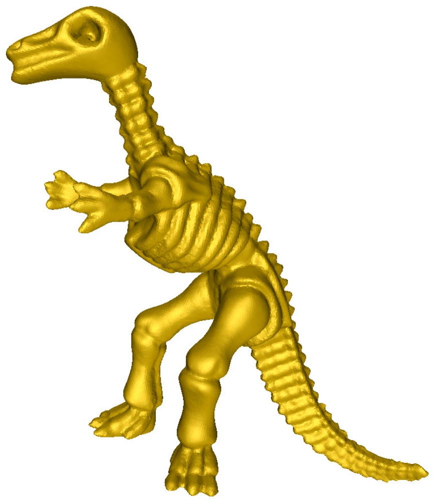
6.2 Results for $\mathbb{R}^3$

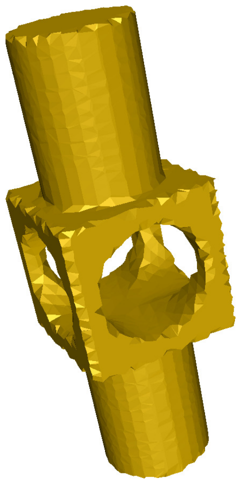
We have implemented all the algorithms in our method for $\mathbb{R}^3$ in C++, again using the CGAL library [48] for Delaunay graph construction and the Disjoint set library of Emil Stefanov.

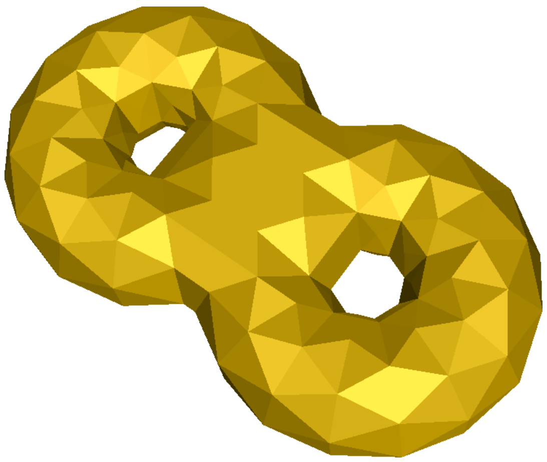
In Figure 6.9 we show examples of boundary shapes constructed with our algorithm. We compare our method with [28], taking a few examples of somewhat more

(a) Mannequin, 13k

(b) Armadillo, 172k


(c) Dinosaur, 65k


(d) Mechpart, 4k


(e) Torus, 0.2k

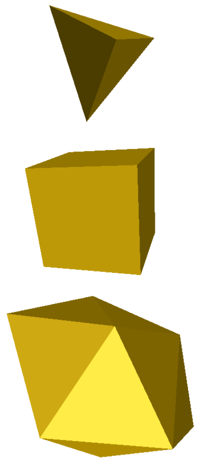
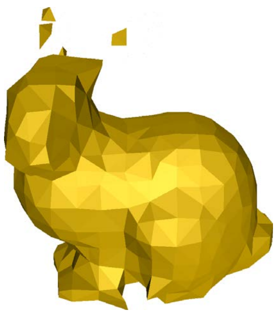
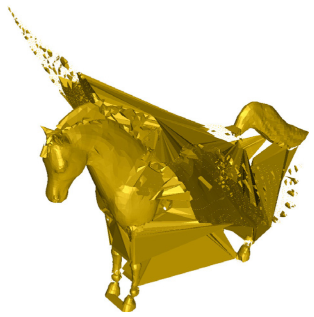
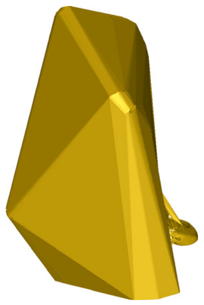
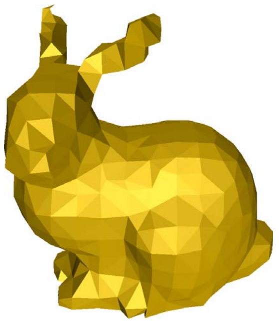

(f) Basic

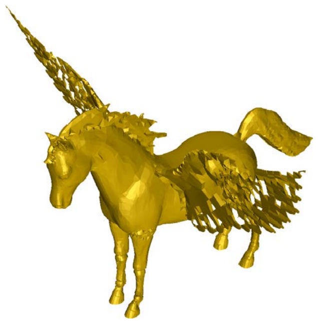
Figure 6.9: a-e): Example results of our reconstruction methods along with point set sizes. f): The three point sets, regular tetrahedron, cube and polyhedron with 16 faces cannot be reconstructed by TightCocone.

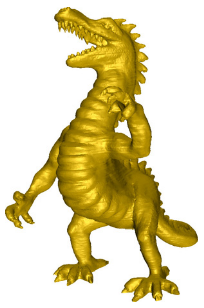

(a) TightCocone

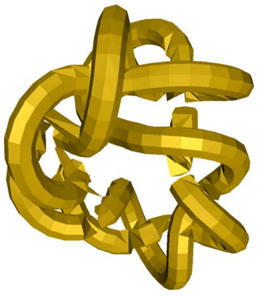

(b) TightCocone

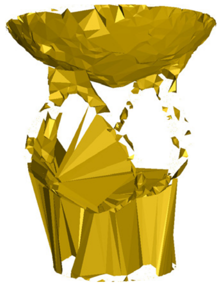

(c) TightCocone

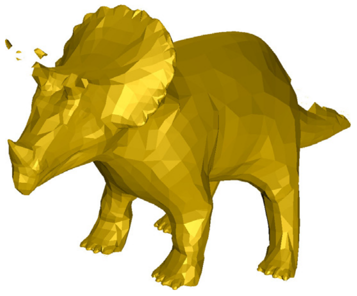

(d) Ours, 0.5k

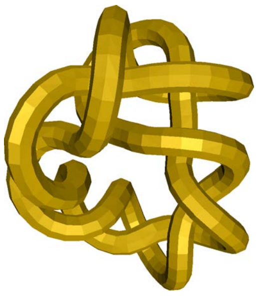

(e) Ours, 14k

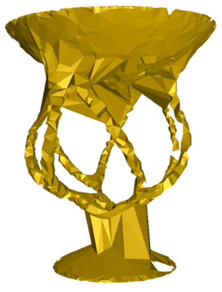

(f) Ours, 54k


(g) TightCocone


(h) TightCocone


(i) TightCocone


(j) Ours, 1k


(k) Ours, 3k

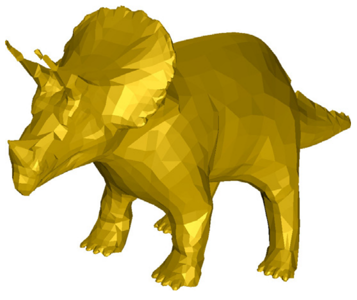
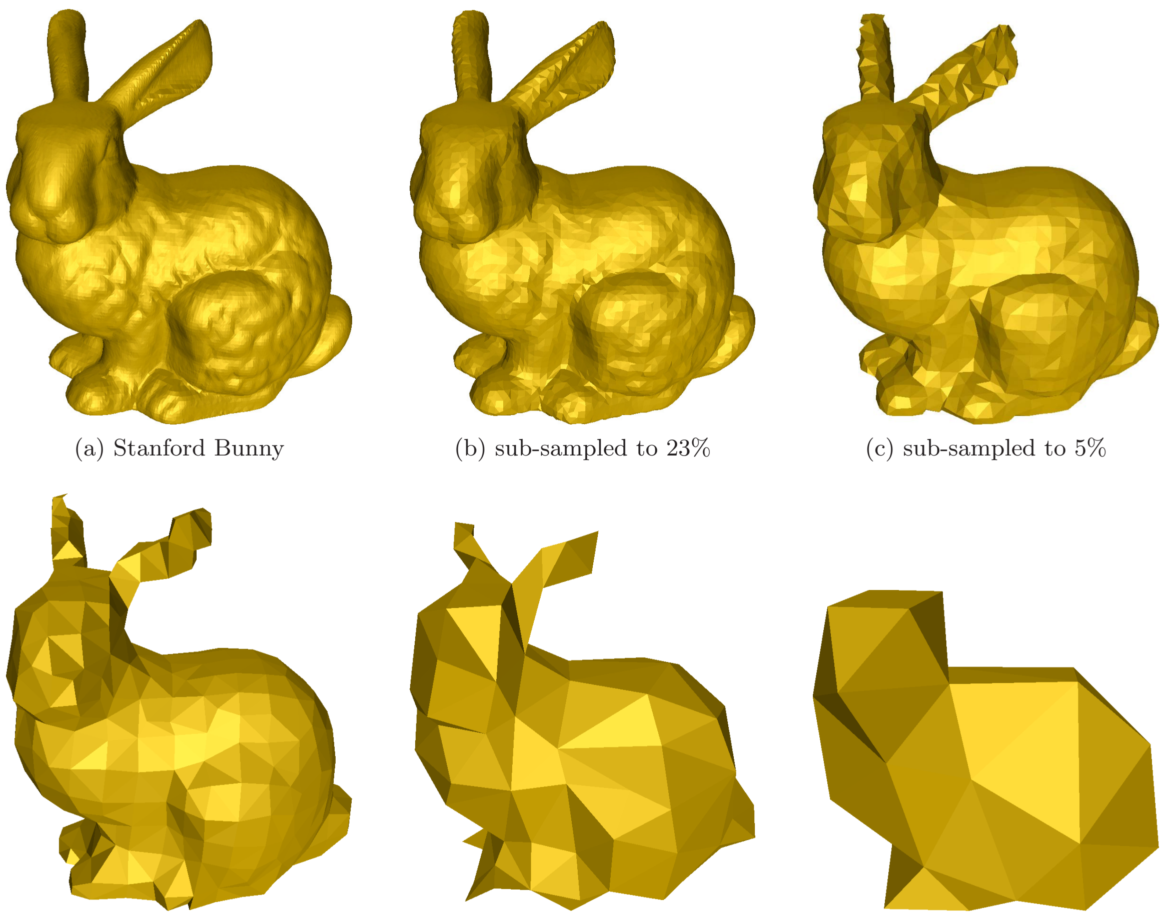

(l) Ours, 3k

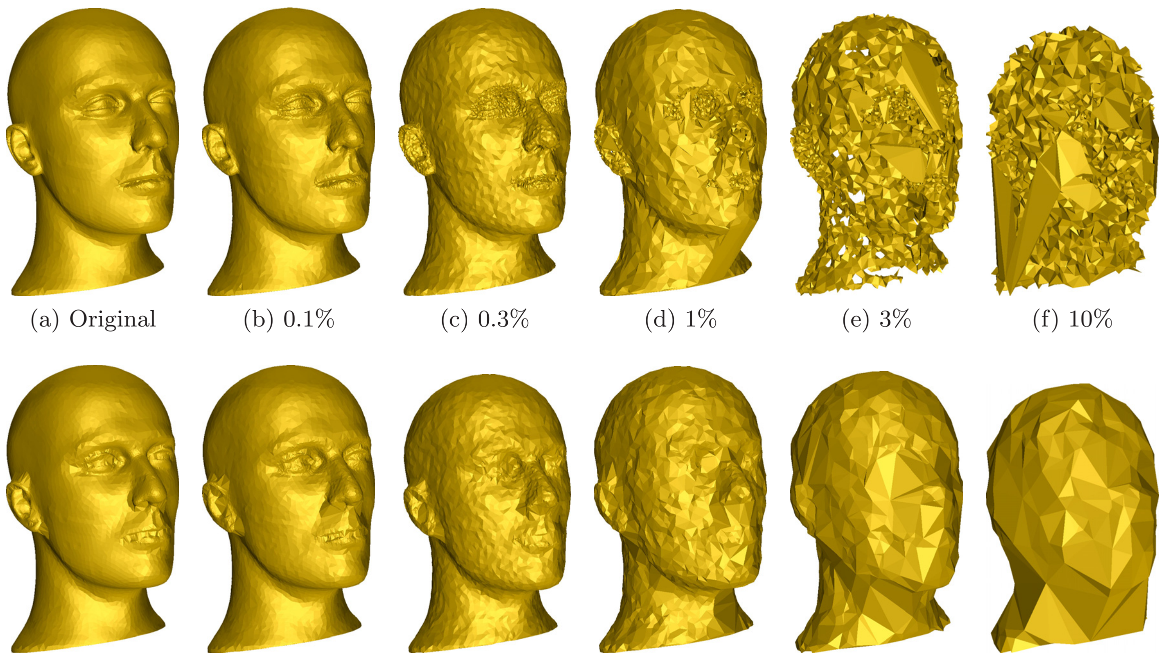
Figure 6.10: Comparing (row 1, 3) our method with TightCocone (row 2, 4), for varied point sets with extremely non-dense or non-uniform sub-sets; note the poor results for TightCocone in those cases. Numbers of points in models are given.



(a) Stanford Bunny (b) sub-sampled to 23% (c) sub-sampled to 5%

(d) sub-sampled to 1% (e) sub-sampled to 0.3% (f) sub-sampled to 0.1%

Figure 6.11: Robustness for non-dense point sets is demonstrated by the Stanford bunny keeping its shape well, when down-sampling from the original 36k to just 33 points.



(a) Original (b) 0.1% (c) 0.3% (d) 1% (e) 3% (f) 10%

(g) -14% (h) -19% (i) -38% (j) -70% (k) -90% (l) -96%

Figure 6.12: Row 1 demonstrates robustness to noise (and local non-uniformity) of our method by perturbing the Mannequin model with noise of up to 10% of its z-extent. c) the noise level exceeds point distances in the fine features such as the eyes. Row 2 reconstructs the surface with RobustCocone [29]: Note that it does so by dropping many points (percentages given in the figure), and thus removes not only outliers but also features, already in the original very dense point set (mouth, eyes, behind the ear).

Name	Vertices	*DG*	Total	Dey	unint.
Torus	0.2k	0.02s	0.7s	0.04s	0
Bunny	0.5k	0.02s	0.14s	0.08s	0
Knot	1k	0.06s	0.27s	0.14s	0
Triceratops	3k	0.14s	1.07s	0.74s	0
Bowl	3k	0.17s	2.67s	0.62s	3
Mechpart	4k	0.18s	1.35s	1.1s	0
Mannequin	13k	0.45s	3.75s	3.36s	0
Pegasus	14k	1.71s	9.25s	6.08s	564
Dragon	54k	3.58s	19.29s	22.36s	14
Dinosaur	65k	3.88s	17.56s	22.64s	0
Armadillo	172k	10.45s	60.86s	71.2s	0

Table 6.1: Runtime for our entire surface reconstruction algorithm (non-optimized implementation), with proportion of Delaunay Graph construction, and compared to TightCocone, on a single 2.67Ghz 64bit AMD CPU. Actual complexity for our method seems to decrease with model size compared to TightCocone. The number of un-interpolated vertices in B are also given. TightCocone fails to interpolate far more points than our algorithm does.

Name	*DG*	BC_0	Conf	Seg	Hole	Infl	Scpt	Fair
Mannequin	12%	15%	37%	4%	15%	1%	15%	2%
Pegasus	19%	6%	21%	6%	10%	6%	30%	1%
Dragon	19%	15%	33%	3%	16%	1%	13%	2%
Dinosaur	22%	13%	35%	2%	15%	0%	10%	2%
Armadillo	17%	16%	33%	4%	15%	0%	12%	2%

Table 6.2: Proportional timings for the steps of our algorithm, for the larger models, in order: Delaunay graph construction, boundary complex, making entities conforming, segmentation, hole-covering, inflating, sculpturing and mesh fairing.

challenging non-uniform point sets in Figure 6.10. That shapes for non-dense point sets are robustly constructed by our method is well illustrated in Figure 6.11. It shows results for point sets obtained by down-sampling the Stanford bunny up to the factor 1000. Finally, the tolerance of our method for construction from noisy point sets is demonstrated in Figure 6.12 by perturbing the points to up to 100 times the feature size (average point distance). We compare it with RobustCocone [29]. Their algorithm removes many outliers and over-smoothes already very densely sampled point sets, therefore losing features which we still can preserve in our method.

We show in Table 6.1 that our algorithm (non-optimized implementation) has nevertheless competitive run-time, compared with TightCocone [28], and that it is linearithmic and proportional to that of the Delaunay graph construction. The implementations of detecting non-conforming entities and making them conforming, as well as hole-covering and sculpturing still have a large potential for optimization. This can be seen in Table 6.2. Since the steps of our algorithm only need to operate on sub-sets of the entire Delaunay graph, we expect the run-time of an optimized implementation to be smaller than the Delaunay graph construction.

6.3 Concluding Remarks

Efficiently constructing the interpolating boundary surface for a set of unorganized points, given just the coordinate data and no other surface or topology information, is known to be a difficult problem. It gets even more difficult when the spacing in the given point sets is non-dense or locally non-uniform. The solution we have proposed in this thesis is based on a few clever heuristics which have been developed over a number of years of careful study of point sets with different spacing and the successes respective failures of previous solutions. The experimental results presented and discussed in this chapter clearly show that our solution yields better results than previous solutions for this problem. There is plenty of scope for further extensions, more theoretical analysis of the limits of our solutions, etc. which form the content of the next chapter.

Chapter 7

Extensions

In this chapter we present potential extensions of our work, namely, generalizing the definition of boundary complex in R^d and defining the conditions for a locally constructible minimum boundary complex. Lastly, we extend the Euclidean Minimum Spanning Tree ($EMST$) in R^d as the Minimum Spanning Surface (MSS), which is a simplicial complex.

7.1 Extension of the Minimum Boundary Complex into $\mathbb{R}^d$

The boundary complex definition can be easily extended into $\mathbb{R}^d$ with $d >= 2$ since both its base concepts, the criterion of longest-edge-in-simplex and its property of umbrella-count-per-vertex in the hull of the boundary are dimension-agnostic. We can therefore give this generalized definition based on the following generalized umbrella definition:

An umbrella $U(v)$ for a vertex v in R^d is a connected set of facets containing v

such that $U(v)$ can be mapped without self-intersections to R^{d-1}.

Definition 4 *The boundary complex* $BC \subseteq DG$ *in* $\mathbb{R}^d$, $d \geq 2$, *is defined as a connected set of facets spanning* P *such that each vertex* v_i *in* BC *has* ≥ 1 *umbrellas in* BC. *Note that* DG *is a* BC.

For any given set of points, BC_{min} *is the* BC *satisfying the following objective:*

$$BC_{min} = \sum_{f_i}^{F} \lambda(f_i) \rightarrow \min$$

The properties mentioned before in Subsection 3.2.3 extend as well.

7.2 The Boundary Operator

7.2.1 The Boundary Operator in $\mathbb{R}^2$

The boundary complex needs to be constructed globally because of its closedness condition. By dropping this condition we can construct another simplicial complex by merging umbrellas locally computed at its vertices. We shall call this as the *Boundary Operator*.

For a given point set P, we define the *boundary operator* as the *minimal umbrella* $U_{min}(p)$ of a point $p \in P$, formed by the edges connecting it to its two nearest neighbors in P. The simplicial complex $\mathbb{B}$ for an unorganized point set P is defined as the union of all $U_{min}(p_i)$ for each point $p_i \in P$ (see Figure 7.1). Note that $\mathbb{B}$ is also a graph, but only in $\mathbb{R}^2$, since in higher dimensions it contains higher-order simplices, such as triangles.

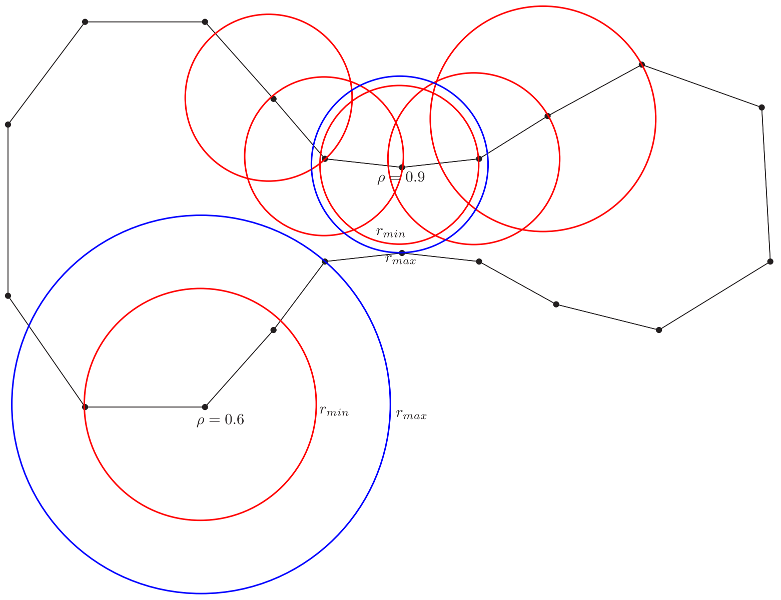


Figure 7.1: Illustration of ρ-sampling for a densely sampled C, which can be reconstructed exactly as its merged umbrellas: ρ is given for two sample points in C. *Neighbors circles* (red) include neighbor samples in C, *empty circles* (blue) exclude non-neighbor points. In a densely sampled C *empty circles* are always larger than *neighbor circles*.

Our aim is to prove that a piece-wise linear curve C interpolating P can be reconstructed just from its point coordinates, under the condition that the neighbors of any point $p \in P$ on C are identical to its two nearest neighbors in Euclidean distance. First we give some definitions to define a measure for such a local density condition.

We call the *minimum* circle which contains *both* the two neighbor sample points for a point p in C as its *neighbor circle*, with its radius as r_n. The *maximum empty* circle which contains *no* non-neighbor sample points is called as its *empty circle*, with its radius as r_e. Then, $\rho(p) = r_n/r_e$.

If $\rho(p) < 1$, then the two nearest neighbor sample points of p are its neighbors in C. If $\rho < 1$ for every point $p \in P$, we say P is *densely sampled* with respect to C.

Theorem 4 *For a given point set P, the minimum length piece-wise linear curve C interpolating all $p \in P$ can be reconstructed by the boundary operator, provided P is densely sampled with respect to C.*

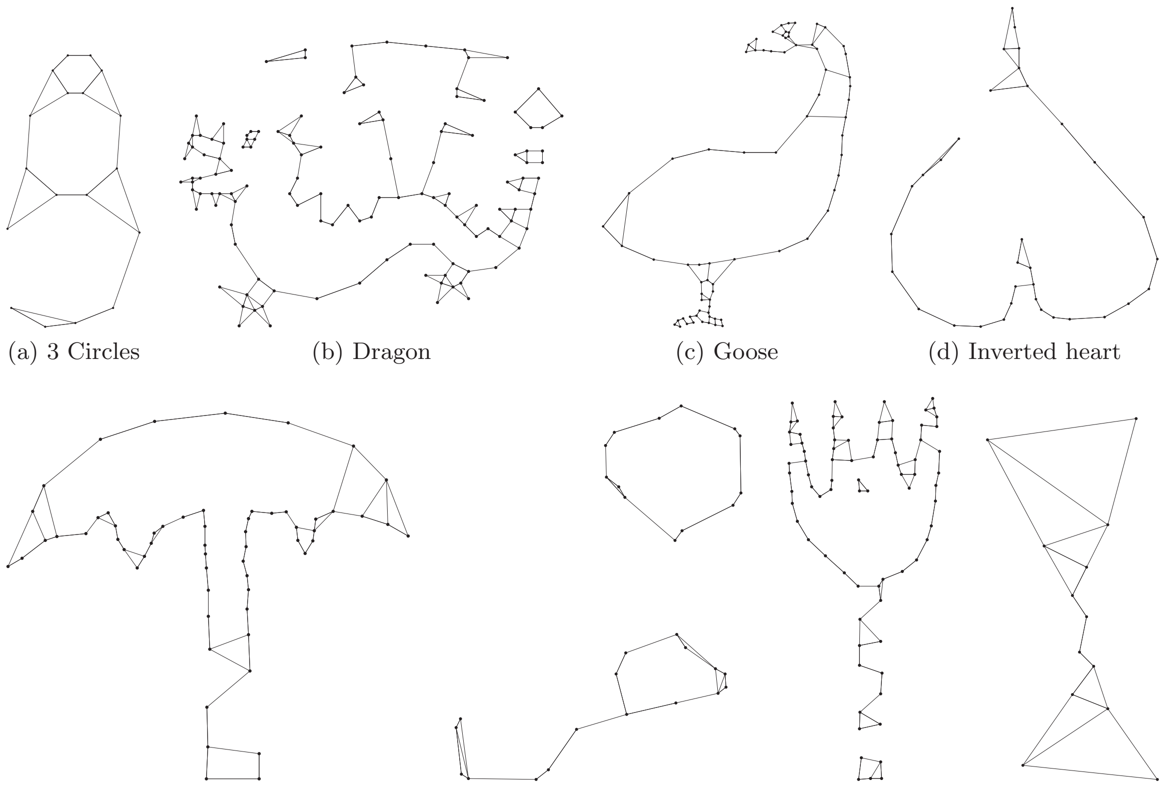
Proof 16 *Proof by counter-example: If a non-boundary neighbor q is nearer to a point p in C than the two neighbors of p in C, then $\rho \geq 1$.*

Corollary 1 *A densely sampled piece-wise linear curve C is locally reconstructible.*

Corollary 2 *For a point set P which is not densely sampled with respect to C, then for all such $p \in P$ with $\rho(p) < 1$, the boundary operator correctly constructs those parts of C.*

Following Corollary 2, we name a vertex v in $\mathbb{B}$ as *well-configured* if it is contained by a single umbrella in $\mathbb{B}$, because it is locally uniquely interpolated by the boundary operator.

$\mathbb{B}$ for non-densely sampled curves shares the property of no leaf vertices with the boundary complex, making the two graphs appear quite similar. But unlike the BC, $\mathbb{B}$ is not necessarily connected, since the minimal umbrellas created by the boundary operator are local (see Figure 7.2). A global post-processing step such as hole-covering, similar to the one described in the $\mathbb{R}^3$ surface reconstruction method, could connect them. This can then be followed by inflating and sculpturing as already described in the $\mathbb{R}^2$ method in Chapter 4.



(a) 3 Circles (b) Dragon (c) Goose (d) Inverted heart

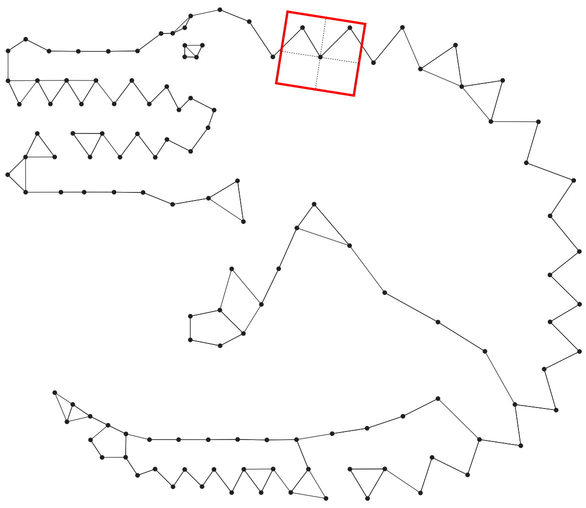
(e) Mushroom (f) Rail joint (g) Tulip (h) Close curves

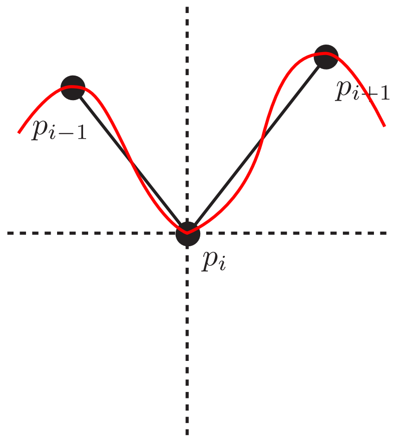
Figure 7.2: $\mathbb{B}$ (union of umbrellas from the boundary operator), for sparsely sampled C. Well-configured vertices have a single umbrella. Note that in (b, f, g) $\mathbb{B}$ does not form a connected set.

7.2.2 Relating the Sampling Theorem to Boundaries

In the literature, to the best of our knowledge, relating boundaries to the sampling theorem has only been addressed by Poliannikov and Krim [68] as follows: transposing a closed curve in a polar coordinate form to create a function space.

As we can see in the back of the crocodile (Figure 7.3), ρ close to 1 approximates an angle of 120° between the umbrella edges. Intuitively, this corresponds to the angle of an equilateral triangle, if both sides are of equal length. We use the conjecture below


(a) $\mathbb{B}$



(b) Detail

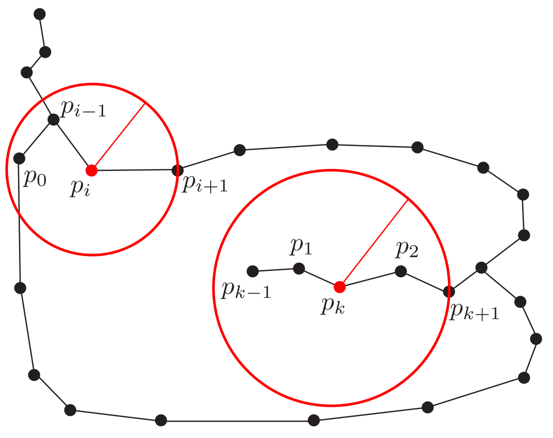
Figure 7.3: a) $\mathbb{B}$ (union of umbrellas from the boundary operator) for a partially densely sampled point set. ρ oscillates around 1 at the back and the tail. b) Detail mapped to function space with p_i at the origin of the local coordinate system.

to relate the sampling theorem, which is defined in function space, to a boundary in space.

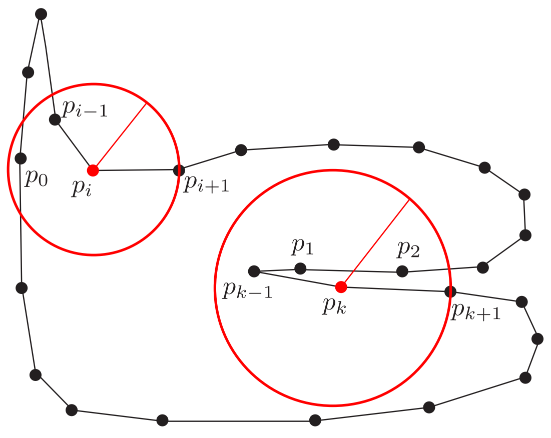
Theorem 5 *A densely sampled piece-wise linear curve C fulfills locally the Nyquist-Shannon theorem.*

Proof 17 *For C with with $\rho < 1$, at any sample point p in C, its boundary neighbors are uniquely defined by the boundary operator (see Theorem 4). Therefore it is possible to find a rigid transformation such that p maps to the origin and its two neighbors locally onto a function space $x(t)$, such that $x(t)$ fulfills the Nyquist-Shannon theorem.*

Theorem 5 further permits us to state that the Gestalt law of Proximity relates rather well to the sampling theorem.



(a) $\mathbb{B}$



(b) Reconstruction

Figure 7.4: Reconstruction for sparse point spacing. Red circles are the neighbor circles centered at p_i, p_k which contain their neighbors on the boundary. p_0 is contained in another segment in the minimum circle of p_i. p_1, p_2 are contained in another segment for p_k. Our inflating operation reconstructs the boundary correctly in these cases (maximally two contiguous and isolatedly densely sampled segments inside the neighbor circle).

7.2.3 Reconstructing Sub-Nyquist Point Spacing

With a priori assumptions, a sparse (= non-dense) function can be reconstructed beyond the limits of the Nyquist-Shannon theorem, e.g. as is done in compressed sensing, for which the theory is described in Mishali and Eldar [62]. Avron et al. [14] implement this using a solver for reconstructing surfaces with sharp features.

Our shape boundary construction method (presented in Chapters 4, 5) manages to reconstruct the boundary from a sparse point sampling, because we assume the inherent property of an object's boundary shape, namely that it is always closed, which is also Gestalt law of Closedness. We describe the conditions for the permitted sparsity of point spacing in Theorem 6 (see Figure 7.4 for an illustration).

Theorem 6 *A piece-wise linear curve C can be reconstructed by the boundary operator at a sparsely sampled point ($\rho \geq 1$), if its neighbor circle contains at most two contiguous segments s in C whose points are densely sampled if not considering the points interior to the other segment.*

Proof 18 *If sample points of more than one contiguous segment are contained in the minimum radius circle centered at a sample point p in B, these will be joined as a non-manifold boundary, containing deflated vertices. By applying the inflating operation, these deflated vertices are transformed into non-deflated ones which permits the construction of at most two contiguous segments inside that circle.*

7.2.4 The Boundary Operator in $\mathbb{R}^3$

The above definitions extend very well into $\mathbb{R}^3$, as follows.

Let B denote the piece-wise linear oriented surface interpolating P in $\mathbb{R}^3$.

The following algorithm constructs the *minimal umbrella* at a vertex p: Add the triangles in DG incident to p sorted by minimal longest edge in ascending order, but only those, whose adding keeps this resulting triangle fan manifold. Triangles are added until they form an umbrella at p.

This algorithm results in a deterministically unique umbrella for each p, for which no simplex in that umbrella is traversed more than once in any single orientation. We call this the *boundary operator* and $\mathbb{B}_u$ as the union of all umbrellas of $p \in P$. Since DG can be constructed locally, this property makes the boundary operator local as well.

We call the *minimum radius* sphere which contains all the sample points of the umbrella $U(p)$ for a point $p \in P$ as its *neighbor sphere*, with its radius as r_n. The *maximum radius* sphere which contains just the sample points in its minimal umbrella and no other sample points is called as its *empty sphere*, with its radius as r_e.

Based on this, we can extend all the previously stated definitions and results into $\mathbb{R}^3$, namely Theorem 4, Corollary 1, Corollary 2, Theorem 5 and Theorem 6.

Unlike the boundary complex, $\mathbb{B}_u$ can contain triangles with boundary edges (edge with only one incident triangle). Such triangles with boundary edges, which only belong to a single umbrella can be removed. Then only triangles with boundary edges which are contained by two or more umbrellas, are retained, so that $\mathbb{B}_u$ forms a bounded manifold. Its boundaries are then manifold and the bounded holes can be triangulated using existing methods.

The greedy construction of boundary complex suffers from artifacts created by local minima which manifest as tetrahedra and hull holes, sometimes even in densely sampled regions of the point set. We give the following conjecture (without attempting a proof here) that our shape boundary construction in $\mathbb{R}^3$ could be improved by using $\mathbb{B}_u$ instead of the boundary complex.

Conjecture 2 $\mathbb{B}_u$ *does not contain the artifacts due to local minima, which exist in the boundary complex. Hull holes with local non-uniformity of factor* ≥ 2 *in the point spacing are perceived as holes by human viewers.*

We conjecture that the boundary operator can be generalized in $\mathbb{R}^d$ with $d \geq 2$, based on the easily extensible definitions of umbrella and ρ.

7.3 The Minimum Spanning Surface (MSS)

As we noted before in Section 4.1, in $\mathbb{R}^2$ the $EMST$, BC_{min} and B_{min} are all graphs which minimize the same criterion of edge length, λ, and differ only in their vertex degree, alternatively umbrella count denoted as u-valence. Based on the criterion λ, both BC_{min} and B_{min} can be extended into higher dimensions, while their respective umbrella count stays the same. There exists no equivalent for the $EMST$, whose u-valence is ≥ 0. By applying λ and this u-valence, we can define its extension into $\mathbb{R}^d$.

Definition 5 *The spanning surface $SS \subseteq DG$ in $\mathbb{R}^d$ is defined as a connected set of facets spanning P such that each vertex v_i in SS has ≥ 1 incident facets in SS.*

For any given set of points, MSS is the SS satisfying the following objective:

$$MSS = \sum_{f_i}^{F} \lambda(f_i) \rightarrow \min$$

The MSS in R^2 is just the $EMST$.

Similar to the boundary complex, a close approximation MSS_0 can be constructed efficiently by extending the topological entities to higher dimensions, as we show in Algorithm 13.

Theorem 7 *Given a point set P with n points and its Delaunay graph $DG(P)$, the*

```
Input: $P, DG$ in $R^d$
Output: $MSS_0$
$F = \{\}$;
$PQ$:=priority-queue of $f_i$ in $DG$, sorted by $\lambda(f_i)$;
while $F$ is not a connected and interpolating set do
    Remove first facet $f_i$ from $PQ$;
    if ($f_i$ contains $v_i$ not in $F$) $\vee$ ($f_i$ connects sets of facets in $F$) then
        $F$:=$F \cup f_i$;
    end
end
$MSS_0 = F$
```

Algorithm 13: Greedy construction of MSS_0 in R^d

MSS can be constructed in $O(n \log n)$ time.

Proof 19 *Creating PQ inserts at most the $O(n)$ facets of DG, an operation of $O(\log n)$ complexity. The while loop which follows is executed at most $O(n)$ times. Testing for and keeping track of connectedness is done via a disjoint set. Its operations are an amortized $O(\alpha(n))$, where $\alpha(n)$ is the inverse of the Ackermann function. Insertion into set F is $O(\log n)$. Total complexity of the algorithm is therefore $O(n \log n)$.*

7.3.1 Properties of the MSS

- Connectedness: By its definition, it is vertex connected and interpolates all of P.
- Sub-set of DG: Again by definition, its facets are contained in DG and therefore the MSS is also a simplicial complex, consisting of simplices up to dimension $d - 1$.

- Not closed: This is because the vertex connected requirement can be fulfilled by just a single facet incident to a vertex.
- Not manifold: This follows partly from above property of the surface not being closed - a vertex may have insufficient incident facets to form an umbrella which is required for it to be manifold and partly from the minimization criterion - a vertex may have more incident facets than can be contained in a manifold.

The MSS generalizes well the properties of $EMST$ into $\mathbb{R}^d$. Contrary to the minimum boundary complex, its surface is not closed. The edges in its simplicial complex reflect well the neighborhood connectivity, which is far more dense than the $EMST$ in dimensions greater than two. We believe that the MSS is an interesting dimension-agnostic spanning structure which needs to be investigated further both for its mathematical properties and applications, such as in clustering of high-dimensional data points.

Chapter 8

Conclusions and Future Work

In this chapter we first present our main conclusions of the research work reported in this thesis. Then we introduce some avenues for future work, as well as implications to other areas of research.

8.1 Conclusion

8.1.1 The Intrinsic Shape of Unorganized Point Sets

All the research reported in this thesis has been the pursuit of one principal idea, that, there exists an intrinsic shape in any given set of unorganized points, assuming that the given point set configuration is not random and that such a shape when rendered should be aesthetically pleasing to a human viewer. We therefore posed this as a problem of constructing the boundary shape interpolating the given points. The desired shape should be based on Gestalt principles of visual perception. Our solution to this problem forms the main contributions of this research. We formulate the problem as finding the global minimum, then define the derivation of a structure which has close resemblance to the desired boundary, and yet can be computed efficiently. This closely resembling shape has many applications including visualization and transformation into the desired shape, the methods developed for that form a good part of this thesis. Results from these methods show superior results when compared to the previous best solutions for the same problem, especially when dealing with points whose spacing is non-dense or locally non-uniform.

8.1.2 Boundary Shape Derivation using Gestalt Laws

From the Gestalt laws, we derived that determining the boundary is a minimization problem, which is NP-hard. Concretely, the laws of *Proximity* and *Good Continuity* translate to minimizing the total mean curvature (in $\mathbb{R}^3$) of the boundary. In $\mathbb{R}^2$ this corresponds to length minimization, similar to the Traveling Salesman Problem. Additionally, we require a closed boundary, from the law of *Closure*. In practice, based on our extensive experiments, the results we get get seem to support these criteria extremely well.

8.1.3 Complexity can be Minimized with a Facet-Based Criterion

In order to find a solution for an aesthetic boundary, approximating this minimum in reasonable time, we have proposed a criterion intrinsic to facets of the boundary, namely longest-edge-in-simplex, which corresponds to edge length in $\mathbb{R}^2$ and longest-edge-in-triangle in $\mathbb{R}^3$.

8.1.4 Minimum Boundary Complex for Shape Representation

By relaxing the manifold constraint of the boundary shape that we need to construct, we present an interesting and powerful restrained simplicial complex, which we have named the minimum boundary complex. A simple heuristic constructs its close approximation in linearithmic time. This close approximation of the minimum boundary complex may not be a manifold, but our experiments show that when it is rendered it seems to yield results closely resembling the desired boundary shape. This serves as a ”quick and dirty” visualization of point clouds and in addition we show several nice properties of the minimum boundary complex, such as its capability to represent shape well even if points are non-densely or non-uniformly spaced, high noise tolerance and construction from any local sub-set of the point set. It can also be efficiently transformed into the desired boundary shape.

8.1.5 Application: Boundary Shape Construction

We have developed clever heuristics to transform the boundary complex into a closed manifold boundary shape interpolating all the points. This corresponds to re-imposing the previously relaxed manifold constraint. Our innovation here is the definition of the inflating operation, which is the dual of the sculpturing operation, and is applied first to get a good starting shape for sculpturing. Our entire method completes in linearithmic time, which is of competitive complexity compared with previous methods, but with far superior results, especially for non-dense and locally non-uniform point sets. The goal of getting closer to the global minimum defined for an aesthetic boundary shape permits us to overcome the major challenges in handling non-uniformly spaced point sets. Furthermore, enforcing the law of closure as an *a priori* assumption, which is inherent to a shape boundary, enables our method to construct even extremely sparsely sampled point sets. Unlike earlier methods, we do not rely on any sampling criterion. For the $\mathbb{R}^3$ case, which is of greater topological complexity, we have introduced additional steps to handle the problem of the operations terminating in local minima. For the $\mathbb{R}^2$ case, we show also how for a large sub-class of point sets, the minimum can be guaranteed to be extracted, although with higher computational complexity.

8.1.6 Well-Suited Heuristics

For the kind of point sets (those with an intrinsic shape) we are interested in, our shape boundary construction algorithm seems to solve or approximate this NP-hard problem quite well in linearithmic time. It clearly prunes the solution space rather efficiently. We explain this with the fact that any reasonable input point set is well-distributed in space (along a lower-dimensional boundary) to accommodate the implicit pruning of our algorithm steps. We think it makes little sense to search for an aesthetic shape in the ill-defined situation of randomly distributed point sets, as it is not robust with respect to small perturbations.

8.2 Future Work

8.2.1 Topological Terms and Definitions

In the algorithm for boundary construction in $\mathbb{R}^3$, we have defined some new terms for topological properties of a well-behaved sub-class of simplicial complexes and operations on those, which extend significantly from their equivalents already defined in $\mathbb{R}^2$. We believe that these terms will very likely find use in other topics such as mesh repairing, resolving of self-intersections, topological post-processing and hole-filling.

8.2.2 Extensions of Shape Boundary Construction

Search for Optimality

If desired, the steps of the algorithm can be altered to permit searching more exhaustively for an even closer approximation towards B_{min} within a given time budget. It would be very interesting to investigate theoretical bounds for this optimal search problem in 3D, as is extensively done in the domain of Linear Programming for 2D by Schrijver [70].

Minimizing other Aspects than Curvature

Minimizing curvature does not support construction of sharp edges in a surface. This is a local feature and does not affect the topology of the surface. We believe that B_{min}, as minimizing curvature, is a member of an isotopic family $\{B_f\}$, in which there exist members B_i minimizing other aesthetic criteria (area, sharp edges, volume, etc.) either locally or globally. Further, that there exists a simplicial complex containing $\{B_f\}$ which proves that other members or their close approximations can be determined in reasonable time from any B_i by edge-flipping its triangulation. Thus, for example, a shape with sharp edges can be constructed from B_{min} by applying a local operator on the 1-neighborhood of vertices as a post-processing step.

Efficient Construction for Uniform Point Spacing

For shape construction, finding the exact minimal BC is not essential. So, it is the required prior DG construction operation which determines to a larger extent the performance of a method to compute minimal BC approximation. Point sets which are partially uniformly distributed, as is often the case for laser-range data, may be binned locally to efficiently determine their approximate nearest neighbors and thus locally the relevant parts of the minimal BC approximation. Then only a constrained DG between non-uniform points has to be determined to complete the global BC. This would lead to a parallel and closer to linear computation.

Local Construction

In Section 7.2 we have shown how the boundary complex can be adapted to design the *boundary operator*, which approximates the shape boundary locally, both in $\mathbb{R}^2$ and in $\mathbb{R}^3$. By extending it with the subsequent steps for manifold shape boundary construction, we believe that it can yield an even better approximation, because more local minima are avoided by the local computation of the operator. A local algorithm cannot construct a closed shape as such. But by considering a greater local context, where the point spacing is not well-configured, it can become equivalent to global construction and therefore construct a closed shape. Extremely sparse sampling can be addressed in the same way. We further note that local construction can be exploited very well on parallel architectures such as today's GPUs.

Out-of-core Construction

As BC_0 can be constructed locally, our boundary construction method is local within the extent of its topological features and can thus easily be adapted to out-of-core processing.

Compression

Triangulated models may be compressed and streamed by suitably ordering and transmitting the points along with just the topology differing from the deterministic shape constructed by our boundary construction algorithm.

Smoothing for $\mathbb{R}^2$

For shape boundaries of noisy point sets in $\mathbb{R}^2$ (e.g. Figure 6.5) we can, not unlike to mesh fairing in $\mathbb{R}^3$, where edges are swapped, exchange sub-tours (edge-chains between vertices) with shorter ones, up to a certain size corresponding to a time budget, to reduce the length of B and attain a better approximation of B_{min}.

Deformable Point Sets

The boundary shape of deformable point sets can in principle be determined anew after each deformation. However, changes with locally limited impact could be detected by properties of the deformation, or changes to the Delaunay graph at vertices. Then the manifold triangulation of the boundary can be kept and repaired locally.

8.2.3 Applications of the Boundary Complex

Shape Characteristic

In *shape retrieval*, the boundary complex can serve as an effective base for the generation of many kinds of shape descriptors, since it is not affected by noise and can be constructed on arbitrary sub-sampled sets of the points. There are other applications, concerning dynamic point sets or sub-sampled geometry, where finding the exact manifold is not essential, e.g. shadow mapping, collision detection and possibly others.

Approximating with an Implicit Surface

The alternatively thin and thick crust triangulation of the boundary complex permits the construction of a signed distance function, to approximate an implicit surface inheriting the same properties. Its potential applications are quite a few, *de-noising*, *silhouette rendering*, etc.

De-noising

The boundary complex in $\mathbb{R}^3$ is not manifold but still represents close neighbors as the connectivity graph of its edges. Laplace smoothing could be applied to the points of a noisy point set, based on their connectivity, as in Mehra et al. [61], as well as out-lier detection, before attempting to construct the shape boundary.

Visibility Culling for Point Clouds

Using the minimum boundary complex approximation, a level-of-detail surface could be dynamically constructed when rendering from an out-of-core multi-resolution data structure. Construction is then just needed for the visible set of points, filtered by the view frustum and with representation proportional to camera distance. This could be an effective way to do visibility culling. Also, huge point clouds could be handled with entirely output-sensitive complexity. This would permit fly-throughs for simulations of such data with only having to operate on the view-dependent in-memory detail. The GPU architecture is ideally suited to handle the massively parallel needs of e.g. physics-based calculations, or global illumination, on such a low-resolution representation.

Segmentation

It can also be used to *segment* point sets containing distinct objects, by both relaxing its condition of a single connected set and specifying a local non-uniformity threshold factor for $\lambda(t)$.

8.3 Implications for Other Areas

In many areas related to computer graphics, such as the simulation of solid or deformable bodies, continuous topology changes are necessary. Examples for this are virtual surgeries and simulations of collapsing buildings or avalanches. However, algorithms and data structures for such basic mesh modification operations are complex, and difficult to implement on massively parallel architectures. Therefore the capability of rebuilding the surface, even if only an approximate one, just from point coordinates, is of great importance for many areas of scientific research. We believe that the novel concept of a point set possessing an intrinsic shape will permeate related scientific domains and sparkle new interest in researching point-based graphics, especially with the prospect of handling enormous point sets from sensing devices on the horizon.

Appendix A

Interpolating Unorganized 2D Point Clouds with Closed Shapes

The paper which follows was the initial attempt to cast the desired aesthetic shape as a minimization problem and to give a solution for unorganized points in $\mathbb{R}^2$. Given a large unorganized two-dimensional point cloud, this work addressed the problem of efficiently reconstructing an aesthetically pleasing closed interpolating shape. Using Gestalt's laws of proximity, closure and good continuity as guidance for visual aesthetics, it was required that the reconstructed shape be minimal perimeter, non-self intersecting and manifold. This yielded visually pleasing results. The algorithm exploits a related minimal graph, the $EMST$, to locally partition and solve the problem efficiently. While this solution required an exhaustive search for a number of practical cases and it did not extend to $\mathbb{R}^3$, it was the basis for the principal idea pursued in this research, namely the existence of an intrinsic shape in unorganized point sets.

Computer-Aided Design 43 (2011) 1629–1638

Contents lists available at SciVerse ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

Interpolating an unorganized 2D point cloud with a single closed shape

Stefan Ohrhallinger [*], Sudhir P. Mudur

Department of Computer Science and Software Engineering, Concordia University, 1455 De Maisonneuve Blvd. West, Montreal, Quebec, Canada H3G 1M8

ARTICLE INFO

Article history:
Received 7 December 2010
Accepted 18 September 2011

Keywords:
Computational geometry
Reconstruction
Construction
Shape
Curve
Boundary
Point cloud
Point set
EMST

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 a minimal perimeter, non-self intersecting 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.

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) [1] 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 Fig. 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 [2].

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 [3,4], Figueiredo and Gomes [5], β-skeleton [6], γ-neighbourhood graph [7] and r-regular shapes [8]. For example, α-shapes require 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. [9] 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 [10,11] 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* [12] 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* [13], but it relies very much on user-tuned parameters and regresses for other point sets. The *Gathan* algorithm from Dey and Wenger [14] also handles sharp corners, but again without guarantees. *GathanG* [15] is an

[*] Corresponding author. Tel.: +1 43 681 20170567.
E-mail addresses: s_ohrhal@cse.concordia.ca, stefango@gmail.com (S. Ohrhallinger), mudur@cse.concordia.ca (S.P. Mudur).

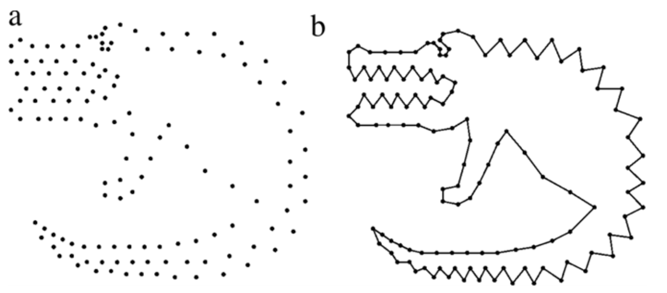


Fig. 1. (a) A sparsely sampled unorganized point set. (b) Its closed manifold interpolating boundary with minimum perimeter.

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 the 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 (Fig. 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 [16] 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 [17] 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 [18] to non-uniform sampling with some conditions, and in [19], 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 [17], 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 [20] scales sub-exponentially and can take hundreds of CPU-years for medium-sized point sets. A detailed discussion on its complexity is available in [21].

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 [23], and $O(n^{O(1/\epsilon)})$ for $(1 + \epsilon)$ times the solution for a planar graph TSP [24]. 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.

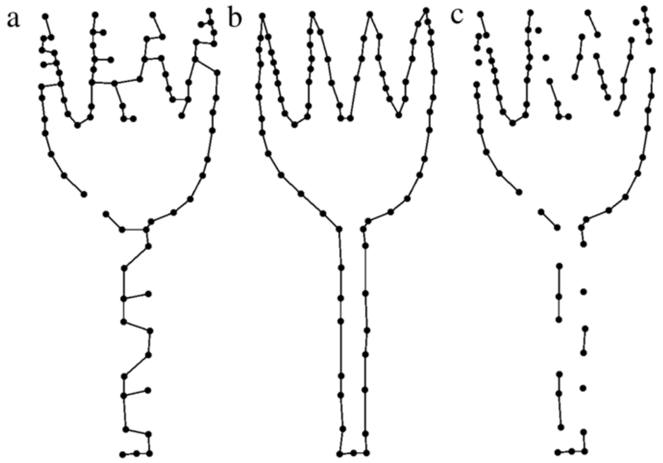


Fig. 2. *EMST* and $S_{\min}$ have considerable overlap even for sparsely sampled point sets like the tulip: (a) *EMST* has 78 edges; (b) $S_{\min}$ has 79 edges; (c) 57 of their edges are shared.

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 Fig. 2).

This relationship between the minimum spanning tree and the shape has been mentioned by Figueiredo and Gomes [5]. 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 Fig. 3(a) and (b)). *EMST* is a subset of *DG* (Attene and Spagnuolo [25]) and can be constructed in $O(n \log n)$ (Kruskal [26]).

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 [27]. 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 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 (Fig. 3(c)).
- Adding an edge in *DG* to the *EMST* graph creates a loop. If two loops share a single edge called a *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.

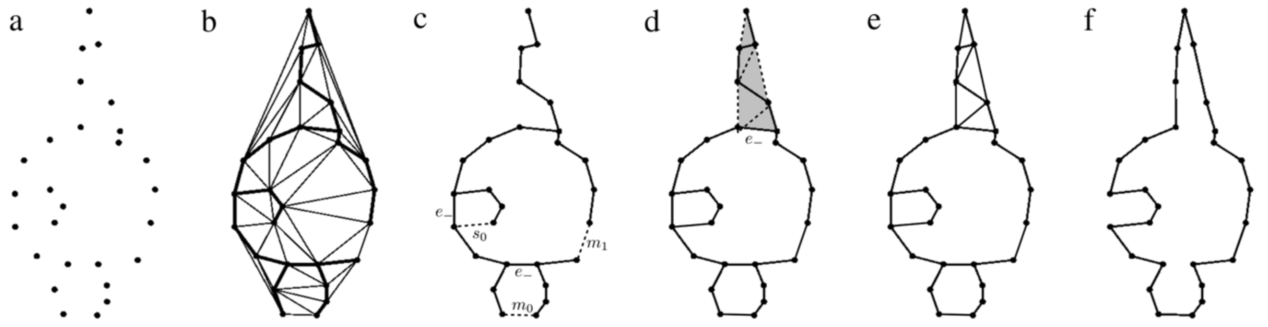


Fig. 3. (a) Example point set P. (b) *DG* with edges in *EMST* emphasized. (c) Edges incident to leaf vertices (m_0, m_1 and s_0) shown with dotted lines. (d) *EMST* with edges added and envelope of *inflatable* branch shaded with its edges not in *EMST* shown with dotted lines. (e) Envelope edges added.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

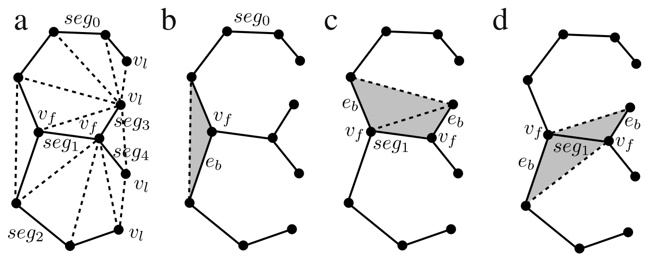


Fig. 4. *EMST* with other edges in *DG* is shown with dotted lines: (a) Segments labelled as seg_i, leaf vertices as v_l and fork vertices as v_f. seg_1 is a trunk segment and all other segments are branches. (b) One of the two non-manifold envelope boundaries of seg_0 (shaded grey) with base edge e_b. 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.

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 (Fig. 3(d)).
- 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 (Fig. 3(d)–(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 [28]). Genoud [29] shows that DG for any P is rarely non-Hamiltonian.

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 Fig. 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 neighbours 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 P is.

Sharp angled features: in [9], 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 $\pi - 4\arcsin(r/2)$*. 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 $\alpha_{\min}$, 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 Fig. 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 one with minimum length. 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.

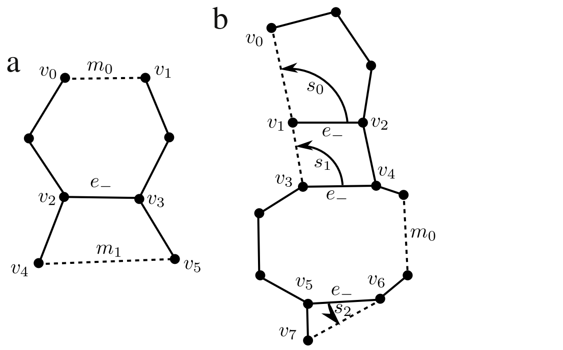


Fig. 5. (a) Example of *EMST* with *moves*: $m_0(v_0, v_1)$ and $m_1(v_4, v_5)$ both associated with the trunk segment $e_-(v_2, v_3)$. (b). Another example of *EMST* with *move* and *snaps*: $s_0(v_1, v_0)$ with $e_-(v_1, v_2)$ has two leaf vertices, $s_1(v_3, v_1)$ with $e_-(v_3, v_4)$ and $s_2(v_6,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

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 Fig. 5 for examples):

- *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_-

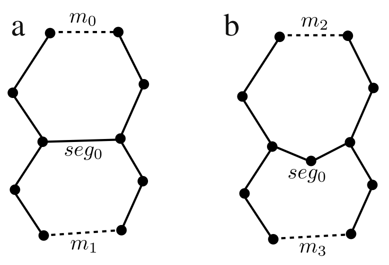


Fig. 6. (a) Single-edge trunk segment seg_0 can be contained in the loops of two candidates m_0, m_1 since it forms a *cut edge*. (b) Candidates m_2, m_3 cannot both be permitted to be applied for multiple-edge trunk segment seg_0: since its removal will disconnect v_0.

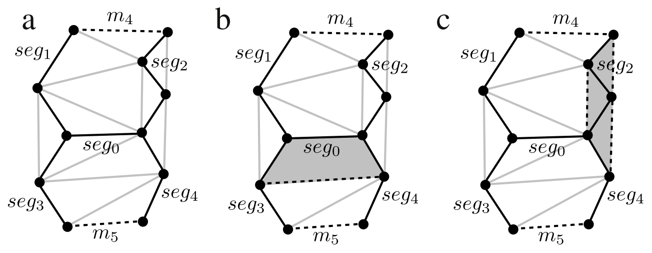


Fig. 7. (a) *EMST* of a point set with *DG* edges shaded grey and segments and candidates marked. (b) Manifold envelope of seg_0 shaded. (c) Manifold envelope of seg_2 shaded.

in the loop formed by a *snap* edge e_+, since this e_- is removed in a subsequent step. Let us note that this segment–candidate association is many-to-many, see for example Fig. 6(b), 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 Fig. 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 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.

Table 1
Segment–candidate table for point set of Fig. 7.

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

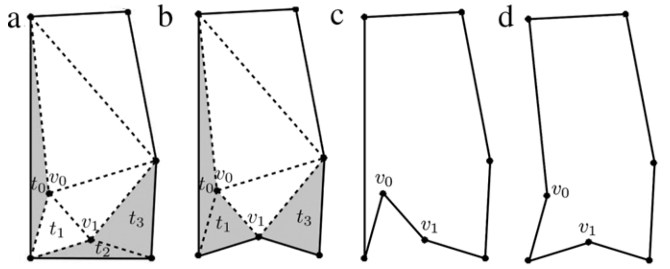


Fig. 8. A manifold envelope boundary env_0 (shown using solid lines) with two interior vertices: (a) shows triangles t_0, t_2 and t_3 for removal from env_0. (b) $env_1 = env_0 \setminus t_2$: new removal triangles are t_0, t_1 and t_3. (c) $env_2 = env_0 \setminus \{t_2, t_1\}$: one potential local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

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. Finally, 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 a 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.

5.1. Inflate operation

As already mentioned, we will need to apply the *inflate* operation to all the connected sets of remaining segments

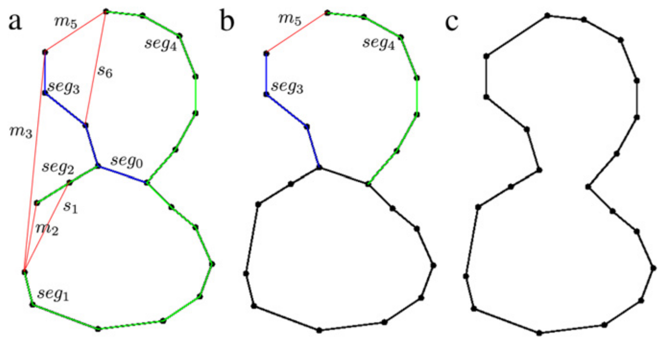


Fig. 9. (a) *EMST* with segments (blue: *non-retained*, green: *retained*) and candidates (red) marked. (b) Applying m_2 creates a loop in the lower half and leaves only one candidate m_5. (c) Applying m_5 and removing the cut edge yields a single S_i, which is the desired solution. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

(*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 the perimeter of $S_{\min}$ is chosen (see Fig. 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 Fig. 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.

5.4. Examples with several potential solutions

For the point set shown in Fig. 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

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

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*	$s_6 m_5$	$m_5 \leftarrow$	$\mathbf{m_5}$

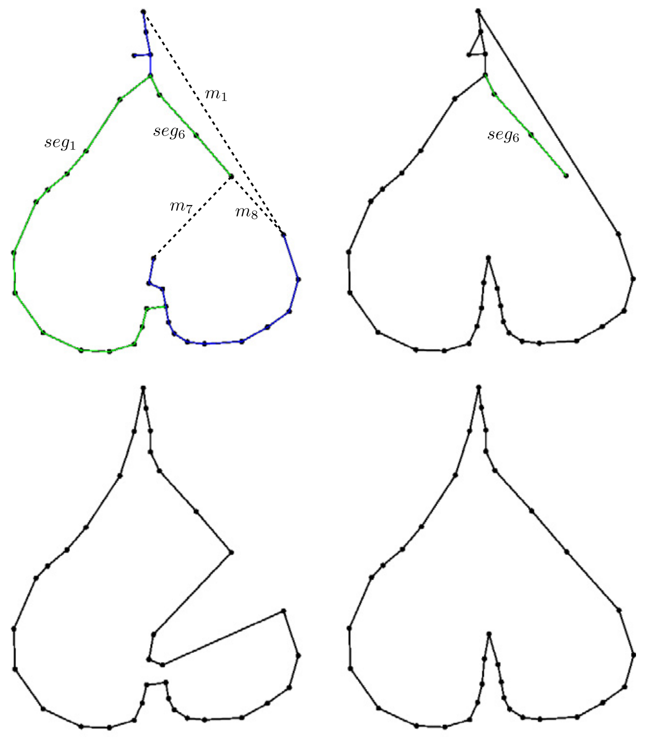


Fig. 10. (a) *EMST* for point set from [12] with segments marked: green = *retained*, blue = *non-retained*. The multiple choice of candidates at seg_1 creates three potential solutions PS_i. (b) PS_0 applies m_1: in the end seg_6 remains without candidates therefore PS_0 becomes invalid. (c) PS_1 applies m_7 and produces S_0. (d) PS_2 applies m_8 and produces S_1 which is $S_{\min}$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 3
Initial segment–candidate table.

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

into three potential solutions as shown in Fig. 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 [5]. 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

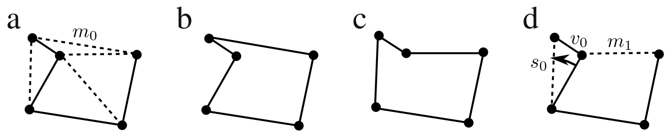


Fig. 11. (a) *EMST* of non-conforming point set, with single operation m_0. (b) Result is S_0 which is not minimal. (c) $S_{\min}$. (d) Therefore m_1 and s_0 are the correct operations to obtain $S_{\min}$ as their vertex degree changes cancel out at v_0 but they are not detectable.

a fairly large point set derived from a silhouette in Fig. 12(e). 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 [30] should be preferred. If one looks closely at the statistics in Table 4, those point sets which are densely sampled (Fig. 12(e)) 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 Fig. 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. 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.

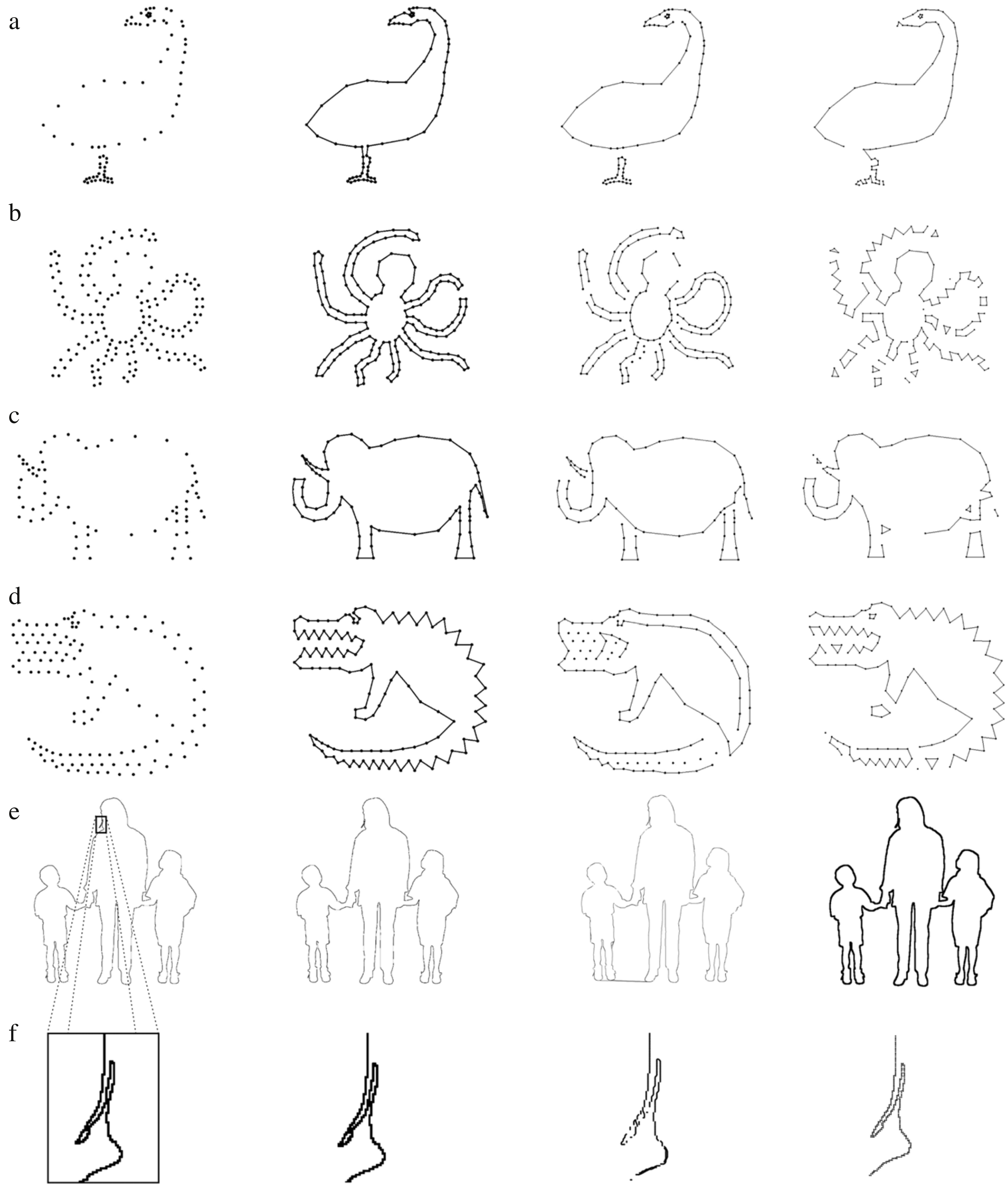


Fig. 12. *Columns*: (1) Point set. (2) Our reconstruction. (3) *GathanG* with default parameters *minAngle* = 10 and *maxIter* = 4. (4) *DISCUR*. *Rows*: (a) Goose (Amenta et al. [9]). (b) Octopus: close curves with sparse sampling. (c) Crocodile: sharp features. (d) Elephant: non-uniform sampling and very sparsely sampled at corners. (e) 10k points sampled on silhouette image: only our method produces a manifold. (f) Detail of e.

6. Results

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 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 [15] (see Figs. 12 and 13) and also with *DISCUR* [12]. 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 2*D* [31]), but we find that these methods settle rather too quickly

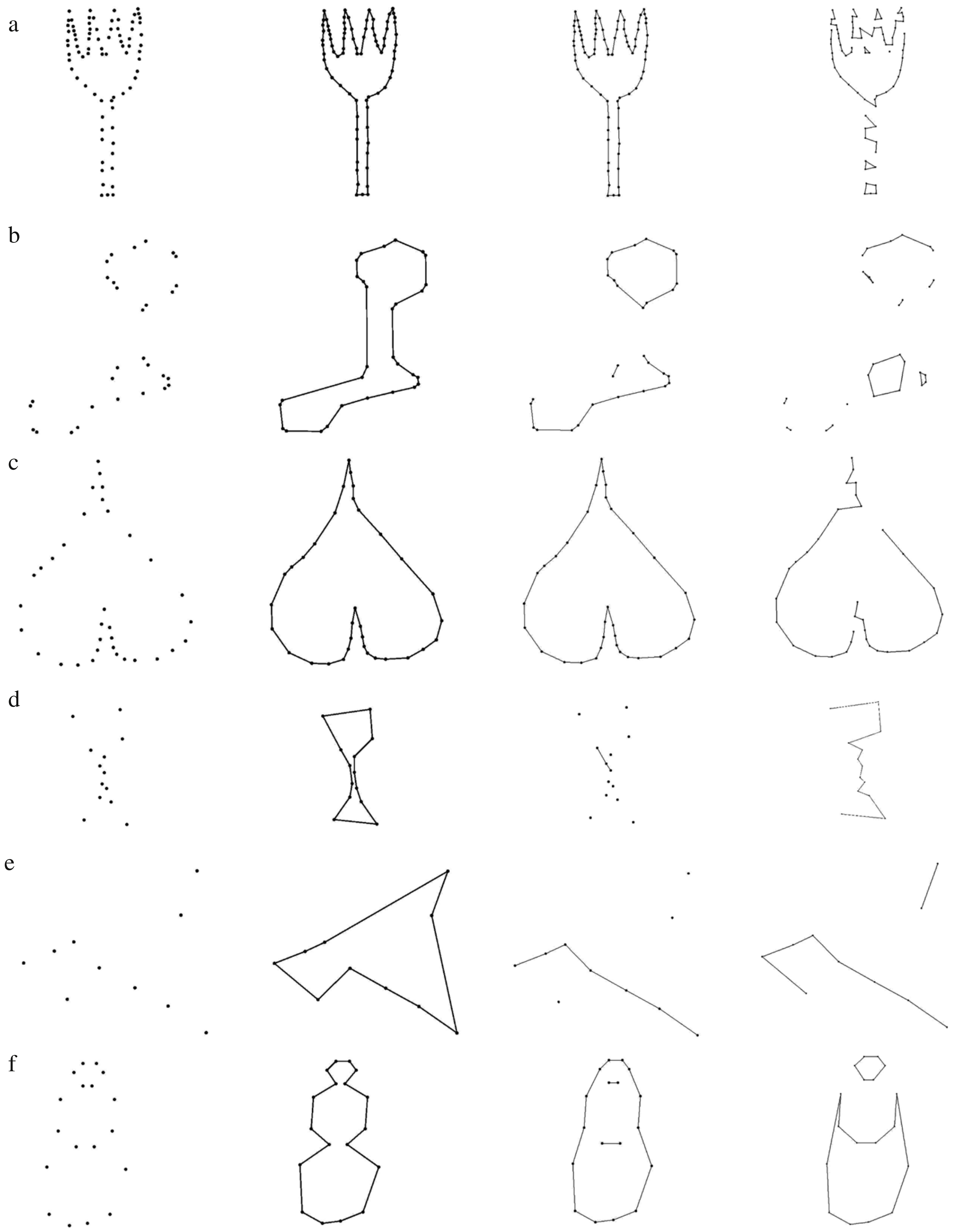


Fig. 13. *Columns*: (1) Point set. (2) Our reconstruction. (3) *GathanG* with default parameters *minAngle* = 10 and *maxIter* = 4. (4) *DISCUR*. *Rows*: (a) Tulip (Althaus and Mehlhorn [18]). (b) Rail-joint (engineering part). (c) Inverted heart (Zeng et al. [12]). (d) Close curves. (e) 10 random points. (f) Three loops.

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

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* give 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. $\alpha_{\min}$ is the minimum angle. $u_{\max}$ is the largest local non-uniformity factor.

Point set	n	l	m	i	Comb.	s_g	s_l	$\alpha_{\min}$	$u_{\max}$
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

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:

$$O(\max(s_g(n\log n), s_l)). \quad (1)$$

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 Figs. 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. $n\log n$. 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 an aesthetically pleasing single closed interpolating 2D shape for an unorganized point set without requiring highly dense or uniform sampling. The results are better 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 $S_{\min}$.

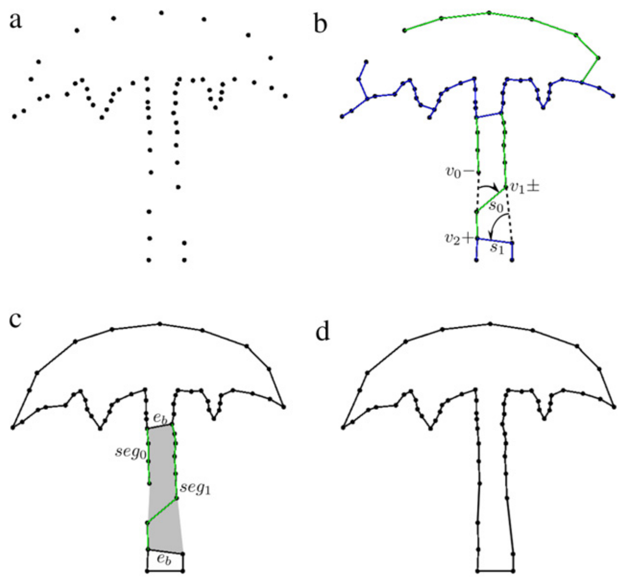


Fig. 14. (a) The *non-conforming* mushroom point set (Dey and Wenger [14]). (b) *EMST* with segments (blue: *non-retained*, green: *retained*) marked. The two *snaps* s_0, s_1 share an impact at v_1 but their e_+ are not contained in one segment, contrary to the required condition. (c) The consequence 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) $S_{\min}$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

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 Fig. 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] Mather G. Foundations of perception. CAD 2008;210–22.
[2] Buchin K, Knauer C, Kriegel K, Schulz A, Seidel R. On the number of cycles in planar graphs. Computing and Combinatorics 2007;97–107.
[3] Bernardini F, Bajaj CL. Sampling and reconstructing manifolds using alpha-shapes. In: Proc. 9th Canad. conf. comput. geom. 1997. p. 0.
[4] Edelsbrunner H, Kirkpatrick DG, Seidel R. On the shape of a set of points in the plane. IEEE Transactions on Information Theory 1983;IT-29(4):551–9.
[5] Figueiredo LHd, Mirandas Gomes Jd. Computational morphology of curves. The Visual Computer 1994;11(2):105–12.
[6] Kirkpatrick DG, Radke JD. A framework for computational morphology. Computational Geometry 1985;217–48.
[7] Veltkamp RC. 3D computational morphology. Computer Graphics Forum (Eurographics '93) 1993;12(3):115–27.
[8] Attali D. r-regular shape reconstruction from unorganized points. In: Symposium on computational geometry. 1997. p. 248–53.
[9] Amenta N, Bern M, Eppstein D. The crust and the β-skeleton: combinatorial curve reconstruction. Graph. Models and Im. Proc 1998;60(2): 125–135.
[10] Dey TK, Kumar P. A simple provable algorithm for curve reconstruction. In: Proc. 10th ACM-SIAM SODA '99. 1999. p. 893–4.
[11] Dey TK, Mehlhorn K, Ramos EA. Curve reconstruction: Connecting dots with good reason. In: Proc. 15th ACM symp. comp. geom 15. 1999. p. 229–44.
[12] Zeng Y, Nguyen TA, Yan B, Li S. A distance-based parameter free algorithm for curve reconstruction. CAD 2008;40(2):210–22.
[13] Nguyen TA, Zeng Y. Vicur: a human-vision-based algorithm for curve reconstruction. Robotics and Computer-Integrated Manufacturing 2008;24(6): 824–34. fAIM 2007, 17th International Conference on Flexible Automation and Intelligent Manufacturing.
[14] Dey TK, Wenger R. Reconstructing curves with sharp corners. Computational Geometry 2001;19(2–3):89–99.
[15] Dey TK, Wenger R. Fast reconstruction of curves with sharp corners. Int. J. Comput. Geometry Appl. 2002;12(5):353–400.
[16] Glanvill M, Broughan K. 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. Washington, DC, USA: IEEE Computer Society; 1997. p. 395.
[17] Giesen J. Curve reconstruction in arbitrary dimension and the traveling salesman problem. In: Proc. 8th DCGI '99. 1999. p. 164–76.
[18] Althaus E, Mehlhorn K. Tsp-based curve reconstruction in polynomial time. In: Proc. 11th ACM-SIAM SODA'00. 2000. p. 686–95.
[19] Althaus E, Mehlhorn K, Schirra S. Experiments on curve reconstruction. In: Proc. 2nd workshop algorithm eng. exper. 2000. p. 103–14.
[20] Applegate D, Bixby R, Chvtal V, Cook W. (accessed March 8, 2011). Concorde TSP solver. 2011. URL: http://www.tsp.gatech.edu/concorde.html.
[21] Hoos HH. On the empirical scaling of run-time for finding optimal solutions to the traveling salesman problem. Technical Report 17. 2009. University of British Columbia, Department of Computer Science.
[22] Helsgaun K. General k-opt submoves for the linkernighan tsp heuristic. Mathematical Programming Computation 2009;1(2–3):119–63.
[23] Arora S. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. Journal of the ACM 1996; 2–11.
[24] Grigni M, Koutsoupias E, Papadimitriou C. An approximation scheme for planar graph tsp. In: Proceedings of the 36th annual symposium on foundations of computer science. FOCS '95. Washington, DC, USA: IEEE Computer Society; 1995. p. 640.
[25] Attene M, Spagnuolo M. Automatic surface reconstruction from point sets in space. Computer Graphics Forum 2000;19(3):457–65.
[26] Kruskal JB. On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society 1956; 7(1):48–50.
[27] Savelsbergh M, Volgenant T. Edge exchanges in the degree-constrained minimum spanning tree problem. Computers & Operations Research 1985; 12(4):341–8.
[28] Dillencourt MB. A non-hamiltonian, nondegenerate delaunay triangulation. Inf. Process. Lett. 1987;25(3):149–51.
[29] Genoud T. Etude du caractère hamiltonien de delaunays aléatoires, travail de semestre. 1990.
[30] Lee I-K. Curve reconstruction from unorganized points. Computer Aided Geometric Design 1999;17(2):161–77. http://www.sciencedirect.com/science/article/pii/S0167839699000448.
[31] Boissonnat J-D. Geometric stuctures for three-dimensional shape representation. ACM Trans. Graph. 1984;3(4):266–86.

Bibliography

[1] Udo Adamy, Joachim Giesen, and Matthias John. Surface reconstruction using umbrella filters. *Computational Geometry: Theory and Applications*, 21(1):63–86, 2002.

[2] Rémi Allègre, Raphalle Chaine, and Samir Akkouche. A flexible framework for surface reconstruction from large point sets. *Computers & Graphics*, 31(2):190 – 204, 2007.

[3] P. Alliez, D. Cohen-Steiner, Y. Tong, and M. Desbrun. Voronoi-based variational reconstruction of unoriented point sets. In *Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 39–48, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[4] Ernst Althaus and Kurt Mehlhorn. Tsp-based curve reconstruction in polynomial time. In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 686–695, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[5] Ernst Althaus, Kurt Mehlhorn, and Stefan Schirra. Experiments on curve reconstruction. In *Proc. 2nd Workshop Algorithm Eng. Exper*, pages 103–114, 2000.

[6] N. Amenta, S. Choi, T. K. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. In *SCG '00: Proceedings of the sixteenth annual symposium on Computational geometry*, pages 213–222, New York, NY, USA, 2000. ACM.

[7] Nina Amenta and Marshall Bern. Surface reconstruction by voronoi filtering. *Discrete and Computational Geometry*, 22:481–504, 1998.

[8] Nina Amenta, Marshall Bern, and David Eppstein. The crust and the β-skeleton: Combinatorial curve reconstruction. *Graphical Models and Image Processing*, 60(2):125–135, 1998.

[9] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust, unions of balls, and the medial axis transform. *Computational Geometry*, 19(2-3):127 – 153, 2001.

[10] David Applegate, Robert Bixby, Vašek Chvátal, and William Cook. Concorde tsp solver. `http://www.tsp.gatech.edu/concorde.html`, 2011 (accessed Mar 8, 2011).

[11] Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. In *Journal of the ACM*, pages 2–11, 1996.

[12] Dominique Attali. r -regular shape reconstruction from unorganized points. In *Symposium on Computational Geometry*, pages 248–253, 1997.

[13] Marco Attene and Michela Spagnuolo. Automatic surface reconstruction from point sets in space. *Computer Graphics Forum*, 19(3):457–465, 2000.

[14] Haim Avron, Andrei Sharf, Chen Greif, and Daniel Cohen-Or. ℓ1-sparse reconstruction of sharp point set surfaces. *ACM Transactions on Graphics*, 29(5):135:1–135:12, November 2010.

[15] Fausto Bernardini and Chandrajit L. Bajaj. Sampling and reconstructing manifolds using alpha-shapes. In *Proceedings of the Ninth Canadian Conference on Computational Geometry*, pages 193–198, 1997.

[16] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Cláudio Silva, and Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999.

[17] Jean-Daniel Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics*, 3(4):266–286, 1984.

[18] Jean-Daniel Boissonnat and Frédéric Cazals. Smooth surface reconstruction via natural neighbour interpolation of distance functions. In *SCG '00: Proceedings of the sixteenth annual symposium on Computational geometry*, pages 223–232, New York, NY, USA, 2000. ACM.

[19] Matthew Bolitho, Michael Kazhdan, Randal Burns, and Hugues Hoppe. Multilevel streaming for out-of-core surface reconstruction. In *Proceedings of the fifth Eurographics symposium on Geometry processing*, SGP '07, pages 69–78, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[20] Matthew Bolitho, Michael Kazhdan, Randal Burns, and Hugues Hoppe. Parallel poisson surface reconstruction. In *Proceedings of the 5th International Symposium on Advances in Visual Computing: Part I*, ISVC '09, pages 678–689, Berlin, Heidelberg, 2009. Springer-Verlag.

[21] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 67–76, New York, NY, USA, 2001. ACM.

[22] Raphaëlle Chaine. A geometric convection approach of 3-d reconstruction. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 218–229, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[23] Siu-Wing Cheng, Stefan Funke, Mordecai Golin, Piyush Kumar, Sheung-Hung Poon, and Edgar Ramos. Curve reconstruction from noisy samples. *Computational Geometry*, 31(12):63 – 100, 2005.

[24] Siu-Wing Cheng and Jiongxin Jin. Edge flips and deforming surface meshes. In *Proceedings of the 27th annual ACM symposium on Computational geometry*, SoCG '11, pages 331–340, New York, NY, USA, 2011. ACM.

[25] David Cohen-Steiner and Frank Da. A greedy delaunay-based surface reconstruction algorithm. *The Visual Computer*, 20(1):4–16, 2004.

[26] T. K. Dey and R. Wenger. Fast reconstruction of curves with sharp corners. *International Journal of Computational Geometry and Applications*, 12(5):353 – 400, 2002.

[27] Tamal K. Dey, Ramsay Dyer, and Lei Wang. Localized cocone surface reconstruction. *Computers & Graphics*, 35(3):483 – 491, 2011. Shape Modeling International (SMI) Conference 2011.

[28] Tamal K. Dey and Samrat Goswami. Tight cocone: a water-tight surface reconstructor. In *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 127–134, New York, NY, USA, 2003. ACM.

[29] Tamal K. Dey and Samrat Goswami. Provable surface reconstruction from noisy samples. *Computational Geometry: Theory and Applications*, 35(1):124–141, August 2006.

[30] Tamal K. Dey and Piyush Kumar. A simple provable algorithm for curve reconstruction. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete*

algorithms, SODA '99, pages 893–894, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.

[31] Tamal K. Dey, Kurt Mehlhorn, and Edgar A. Ramos. Curve reconstruction: Connecting dots with good reason. *In Proceedings of the 15th ACM Symposium on Computational Geometry*, 15:229–244, 1999.

[32] Tamal K. Dey and Rephael Wenger. Reconstructing curves with sharp corners. *Computational Geometry*, 19(2-3):89 – 99, 2001.

[33] Michael B. Dillencourt. A non-hamiltonian, nondegenerate delaunay triangulation. *Information Processing Letters*, 25(3):149–151, 1987.

[34] Daniel Dumitriu, Stefan Funke, Martin Kutz, and Nikola Milosavljevic. How much Geometry it takes to Reconstruct a 2-Manifold in r^3. In *Proceedings of the 10th Workshop on Algorithm Engineering and Experiments, ALENEX 2008*, pages 65–74, San Francisco, USA, 2008. ACM-SIAM, SIAM.

[35] H. Edelsbrunner. Surface Reconstruction by Wrapping Finite Sets in Space. *Discrete and Computational Geometry - The Goodman-Pollack Festschrift*, 25(1):379–404, 2003.

[36] H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, IT-29(4):551–559, 1983.

[37] H. Edelsbrunner and N. R. Shah. Incremental topological flipping works for regular triangulations. *Algorithmica*, 15:223–241, 1996.

[38] Herbert Edelsbrunner and Ernst P. Mücke. Three-dimensional alpha shapes. In *VVS '92: Proceedings of the 1992 workshop on Volume visualization*, pages 75–82, New York, NY, USA, 1992. ACM.

[39] L. H. de Figueiredo and J. de Mirandas Gomes. Computational morphology of curves. *The Visual computer*, 11(2):105–112, 1994.

[40] Stefan Funke and Edgar A. Ramos. Smooth-surface reconstruction in near-linear time. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 781–790, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[41] T. Genoud. Etude du caractère hamiltonien de delaunays aléatoires. Travail de semestre, 1990.

[42] Joachim Giesen. Curve reconstruction in arbitrary dimension and the traveling salesman problem. In *DCGI '99 Proceedings of the 8th International Conference on Discrete Geometry for Computer Imagery*, pages 164–176, London, UK, 1999. Springer-Verlag.

[43] Joachim Giesen and Matthias John. The flow complex: a data structure for geometric modeling. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '03, pages 285–294, Philadelphia, PA, USA,

2003. Society for Industrial and Applied Mathematics.

[44] Martin Glanvill and Kevin Broughan. 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*, pages 395–, Washington, DC, USA, 1997. IEEE Computer Society.

[45] M. Gopi, S. Krishnan, and C.T. Silva. Surface reconstruction based on lower dimensional localized delaunay triangulation. *Computer Graphics Forum*, 19(3):467–478, 2000.

[46] Leonidas Guibas and Steve Oudot. Reconstruction using witness complexes. *Discrete & Computational Geometry*, 40:325–356, 2008.

[47] Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi. *ACM Transactions on Graphics*, 4:74–123, April 1985.

[48] Susan Hert and Michael Seel. dD convex hulls and Delaunay triangulations. In *CGAL User and Reference Manual*. CGAL Editorial Board, 3.8 edition, 2011. `http://www.cgal.org/Manual/3.8/doc_html/cgal_manual/packages.html#Pkg:ConvexHullD`.

[49] Klaus Hildebrandt and Konrad Polthier. Anisotropic filtering of non-linear surface features. *Computer Graphics Forum*, 23:391–400, 2004.

[50] Hisamoto Hiyoshi. Optimization-based approach for curve and surface reconstruction. *Computer-Aided Design*, 41:366–374, May 2009.

[51] Holger H. Hoos. 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, 2009.

[52] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics*, 26(2):71–78, 1992.

[53] Alexander Hornung and Leif Kobbelt. Robust reconstruction of watertight 3d models from non-uniformly sampled point clouds without normal information. In *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 41–50, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[54] J. Jaromczyk and G. Toussaint. Relative neighborhood graphs and their relatives. In *Proceedings of IEEE*, volume 80, pages 1502–1517, 1992.

[55] Sagi Katz, Ayellet Tal, and Ronen Basri. Direct visibility of point sets. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[56] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry*

processing, SGP '06, pages 61–70, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

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

[58] Géza Kós. An algorithm to triangulate surfaces in 3d using unorganised point clouds. In *Geometric Modelling*, pages 219–232, London, UK, 2001. Springer-Verlag.

[59] P. Labatut, J.-P. Pons, and R. Keriven. Robust and efficient surface reconstruction from range data. *Computer Graphics Forum*, 28(8):2275–2290, 2009.

[60] B. Mederos, L. Velho, and L.H. De Figueiredo. Moving least squares multiresolution surface approximation. In *Computer Graphics and Image Processing, 2003. SIBGRAPI 2003. XVI Brazilian Symposium on*, pages 19 – 26, oct. 2003.

[61] Ravish Mehra, Pushkar Tripathi, Alla Sheffer, and Niloy J. Mitra. Visibility of noisy point cloud data. *Computers & Graphics*, 34(3):219 – 230, 2010. Shape Modelling International (SMI) Conference 2010.

[62] M. Mishali and Y.C. Eldar. Blind multiband signal reconstruction: Compressed sensing for analog signals. *Signal Processing, IEEE Transactions on*, 57(3):993 –1009, march 2009.

[63] Patrick Mullen, Fernando De Goes, Mathieu Desbrun, David Cohen-Steiner, and Pierre Alliez. Signing the unsigned: Robust surface reconstruction from raw pointsets. *Computer Graphics Forum*, 29(5):1733–1741, 2010.

[64] Thanh An Nguyen and Yong Zeng. Vicur: A human-vision-based algorithm for curve reconstruction. *Robotics and Computer-Integrated Manufacturing*, 24(6):824 – 834, 2008. FAIM 2007, 17th International Conference on Flexible Automation and Intelligent Manufacturing.

[65] Stefan Ohrhallinger. Source-code for shape extraction algorithm. `https://sourceforge.net/p/connect2dlib/home/`, May 2011.

[66] Stefan Ohrhallinger and Sudhir P. Mudur. Interpolating an unorganized 2d point cloud with a single closed shape. *Computer-Aided Design*, 43(12):1629–1638, 2011.

[67] Sylvain Petitjean and Edmond Boyer. Regular and non-regular point sets: Properties and reconstruction. *Computational Geometry*, 19:101–126, 2001.

[68] Oleg V. Poliannikov and Hamid Krim. On sampling closed planar curves and surfaces. *The Journal of Sampling Theory in Image and Signal Processing*, 2(1):53–82, 2003.

[69] M. Samozino, M. Alexa, P. Alliez, and M. Yvinec. Reconstruction with voronoi centered radial basis functions. In *Proceedings of the 4th Eurographics symposium*

on Geometry processing, SGP '06, pages 51–60, Aire-la-Ville, Switzerland, 2006. Eurographics Association.

[70] Alexander Schrijver. *Theory of Linear and Integer Programming*. Wiley, New York, NY, USA, June 1998.

[71] Shy Shalom, Ariel Shamir, Hao Zhang, and Daniel Cohen-Or. Cone carving for surface reconstruction. *ACM Transactions on Graphics*, 29(6):150:1–150:10, December 2010.

[72] Emil Stefanov. Disjoint sets data structure implementation, 2011 (accessed May 13, 2011).

[73] Remco C. Veltkamp. 3D computational morphology. *Computer Graphics Forum (Eurographics '93)*, 12(3):115–127, 1993.

[74] Remco C. Veltkamp. Boundaries through scattered points of unknown density. *Graphical Models and Image Processing*, 57(6):441–452, 1995.

[75] Yong Zeng, Thanh An Nguyen, Baiquan Yan, and Shuren Li. A distance-based parameter free algorithm for curve reconstruction. *Computer-Aided Design*, 40(2):210–222, 2008.

Curriculum Vitae

Contact Information

Name:	Stefan Ohrhallinger
Email:	stefango@gmail.com
Telephone:	+43-(0)681-20170567
Address:	Taborstrasse 18/124, 1020 Wien, Austria

Personal details

Date of birth:	March 14th, 1974
Place of birth:	Braunau/Inn, Austria
Nationality:	Austrian
Languages:	German (native), English (fluent), French (fluent)

Education

Sep. 1980 – Jun. 1984	Primary school (*Volksschule*) in Altheim, Austria.
Sep. 1984 – Jun. 1992	Secondary school (*Realgymnasium*) in Braunau/Inn, Austria.
Okt. 1992 – Nov. 1998	University Linz, Austria
	Graduated Master in business economics (*Magister der Wirtschaftsinformatik*). Thesis in computer graphics (*Inst. f. Graphische und Parallele Programmierung*). Advisors: Dr. Alfred Spalt, Prof. Jens Volkert.
Sep. 2006 – present	Concordia University, Montréal, Canada
	Ph.D. in computer science, in the 3D Graphics Group at the Dept. of Computer Science & Software Engineering. Completing thesis: "Intrinsic Shape of Point Clouds". Advisor: Dr. Sudhir P. Mudur.

Employment

Nov 1997 – Jan 1999	Beko Engineering, Vienna
	Software Developer: Work-flow management in an insurance company.
Jan 1999 – Apr 2001	Beko Consulting, Vienna
	Consultant: evaluations of middleware in-house and at VÖEST, Linz. **Team lead** (10 developers) for a project in the ministry of interior (Zentrales Melderegister).
May 2001 – Aug 2006	Free-lance Consultant, Vienna
	Team lead (10 developers) for a project in the ministry of interior (Zentrales Melderegister).
Sep 2006 – present	Concordia University, Montréal, Canada
	Research Assistant: Research and teaching at the Dept. of Computer Science & Software Engineering.
Oct 2006 – Feb 2008	Xtranormal, Montréal, Canada
	Software Developer: Location of facial features in photos.