How much Geometry it takes to Reconstruct a 2-Manifold in $\mathbb{R}^3$

Daniel Dumitriu[†] Stefan Funke[‡] Martin Kutz[†] Nikola Milosavljevic[§]

Abstract

Known algorithms for reconstructing a 2-manifold from a point sample in $\mathbb{R}^3$ are naturally based on decisions/predicates that take the geometry of the point sample into account. Facing the always present problem of round-off errors that easily compromise the exactness of those predicate decisions, an exact and robust implementation of these algorithms is far from being trivial and typically requires the employment of advanced datatypes for exact arithmetic as provided by libraries like CORE, LEDA or GMP. In this paper we present a new reconstruction algorithm, one of whose main novelties is to throw away geometry information early on in the reconstruction process and to mainly operate combinatorially on a graph structure. As such it is less susceptible to robustness problems due to round-off errors and also benefits from not requiring expensive exact arithmetic by faster running times. A more theoretical view on our algorithm including correctness proofs under suitable sampling conditions can be found in a companion paper [**?**].

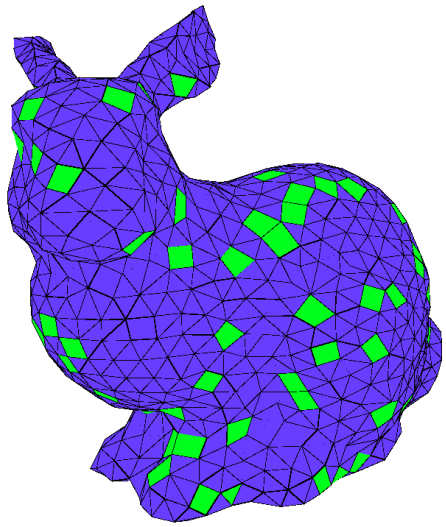

Figure 1: Output of our algorithm for the Stanford Bunny. Due to the conservative adjacency creation, some faces (light) are non-triangular, which can easily be triangulated later on, though.

1 Introduction

Robust Geometric Computation Geometric algorithms use geometric predicates in their conditionals (e.g. does a point r lie to the left or right of an oriented line through p and q). A common strategy for the exact implementation of geometric algorithms is to evaluate all geometric predicates exactly. While floating-point filters have proved to be quite efficient both in theory and practice to speed-up the exact evaluation of predicates, they tend to become less efficient for more complicated (i.e. higher degree) predicates like the *insphere predicate*[1] which is the basis for all algorithms based on the Delaunay tetrahedralization in $\mathbb{R}^3$.

The evaluation of a geometric predicate amounts to the computation of the sign of an arithmetic expression. Round-off errors during floating-point computation might easily lead to reporting a wrong sign of such an arithmetic expression, which most of the time has disastrous consequences ([**?**]). A floating-point filter evaluates the expression using floating-point arithmetic and also computes an error bound to determine whether the floating point computation is reliable. If the error bound does not suffice to prove reliability, the expression is re-evaluated using exact arithmetic. The quality of the error bounds typically deteriorates, though, with the complexity of the predicate expression and hence, more predicate decisions require the fallback of expensive exact arithmetic computation.

A different approach to the robustness problem is to design the algorithm to be able to cope with round-off errors right from the beginning. The difficulty of designing robust algorithms is illustrated in [**?**, **?**, **?**]. In [**?**] Sugihara et al. develop an algorithm for computing the Voronoi diagram of a set of points whose termination is guaranteed by certain *purely combinatorial* graph properties; geometric predicate evaluations are only used to steer the execution of the algorithm in a certain direction. In particular, their algorithm would also terminate if the outcome of all geometric predicates was completely randomized.

In this paper we present an algorithm for recon-

[†]Max-Planck-Institut für Informatik, Saarbrücken, Germany, dumitriu@mpi-inf.mpg.de

[‡]Ernst-Moritz-Arndt-Universität Greifswald, Germany, stefan.funke@uni-greifswald.de

[§]Stanford University, Stanford, U.S.A, nikolam@stanford.edu

[1]The *insphere predicate* for points p, q, r, s, t decides whether point t lies inside, on or outside the sphere defined by p, q, r, s.

structing a 2-manifold in $\mathbb{R}^3$ in the spirit of the work by Sugihara et al. The main idea is that only at the beginning of the algorithm we require the exact evaluation of a few *simple* (i.e. low-degree) predicates and then essentially work combinatorially without evaluating any geometric predicates. Nevertheless we show (both experimentally as well as theoretically in a companion paper [**?**]) that the output of our algorithm faithfully resembles the original 2-manifold. Most importantly, our algorithm produces this output *without* any evaluation of a complicated geometric test like the insphere predicate in $\mathbb{R}^3$, which is the basis of all related Voronoi-based reconstruction algorithms.

Related Work The problem of reconstructing a surface Γ in $\mathbb{R}^3$ from a finite point sample V has attracted a lot of attention both in the computer graphics community as well as in the computational geometry community. While in the former the emphasis is mostly on algorithms that work 'well in practice', the latter has focused on algorithms that come with a theoretical guarantee: if the point sample V satisfies a certain *sampling condition*, the output of the respective algorithm is guaranteed to be 'close' to the original surface.

In [**?**], Amenta and Bern proposed a framework for rigorously analyzing algorithms reconstructing smooth closed surfaces. They define for every point $p \in \Gamma$ on the surface the *local feature size* $\mathrm{lfs}(p)$ as the distance of p to the *medial axis*[2] of Γ. A set of points $V \subset \Gamma$ is called a ε-sample of Γ if $\forall p \in \Gamma\ \exists s \in V : |sp| \leq \varepsilon \cdot \mathrm{lfs}(p)$. Many algorithms have been proposed that determine a collection of Delaunay triangles which form a piecewise linear surface that is topologically equivalent to the original surface and converges to the latter both point-wise as well as in terms of the surface normals as the sampling density goes to infinity ($\varepsilon \to 0$). Common to almost all those algorithms is the fact that they require the computation of a Voronoi diagram/Delaunay triangulation of a point set in $\mathbb{R}^3$ which incurs both an inherent $\Omega(n^2)$ running time as well as the need for the exact evaluation of the *insphere predicate*.

In [**?**] Funke and Milosavljevic present an algorithm for computing *virtual coordinates* for the nodes of a wireless sensor network which are themselves unaware of their location. Their approach crucially depends on a subroutine to identify a provably planar subgraph of a communication graph that is a quasi-Unit-Disk graph. A similar subroutine will also be used in our surface reconstruction algorithm presented in this paper.

[2]The medial axis of Γ is defined as the set of points which have at least two closest points on Γ.

Our Contribution We propose a new graph-based algorithm for reconstructing a 2-manifold in $\mathbb{R}^3$. Our algorithm fundamentally differs from previous approaches in two respects: first, it mainly operates combinatorially on a graph structure, which is derived from the original geometry; secondly, the created adjacencies/edges are "conservative" in a sense that two samples are only connected if in all reasonable reconstructions those two samples are adjacent. Interestingly we can show in the theoretical companion paper [**?**], though, that conservative edge creation only leads to small, constant-size faces in the respective reconstruction, hence the topology of the original surface is faithfully captured. While the theoretical analysis requires an absurdly high sampling density – like most of the above mentioned algorithms do – this paper shows that our algorithm is actually very practical for real-world datasets. Due to the local nature of computation in our algorithm there is also potential for use e.g. in parallel computing or external memory scenarios.

2 Our Algorithm

The main idea of our algorithm is first to derive a graph $G(V)$ which captures mutual proximity information of the samples in V and then decide on adjacencies between (some of the) samples of V only based on the connectivity structure of $G(V)$. To keep the presentation simple, we assume that V is a *uniform*[3] ε-sample from a closed smooth 2-manifold Γ in $\mathbb{R}^3$. In practice, sample sets are indeed often close to uniform and even small non-uniformities in the sample set never prevented our algorithm to work in practice. In theory, a preprocessing stage can enforce *local uniformity* which is sufficient to prove correctness of our algorithm. See [**?**] for more details. The high-level view of our algorithm is as follows:

1. Construct a local neighborhood graph $G(V)$ by creating an edge from every point v to all other points v' with $|vv'| \leq O(\varepsilon)$.
2. Compute a subsample S of V as a maximal k-hop stable set in $G(V)$
3. Construct the *graph Voronoi diagram* for V with respect to the subsample S
4. Identify adjacencies between elements in S by inspection of the graph Voronoi diagram
5. Output faces of the reconstruction as minimal cycles in the graph induced by the adjacencies between elements in S

In step one we create a unit-disk graph of the point set connecting two samples iff their distance is less

[3]V is a uniform ε-sample if for any $p \in \Gamma\ \exists s \in V$ with $|ps| \leq \varepsilon$.

than some constant times ε. In step two we compute a maximal subsample $S \subset V$ with the property that there are no two $s_1, s_2 \in S$ closer than k hops in the graph $G(V)$. Such a maximal (not maximum!) k-hop stable set can easily computed in a greedy fashion. In step three, we essentially compute a discrete analogue of the geometric Voronoi diagram but using $G(V)$ as a discrete approximation of the space between the subsample vertices in S. That is, we assign each node $v \in V$ its closest (in terms of hop-distance) node in S (breaking ties according to node IDs). This partitions V into tiles consisting of nodes which have the same closest $s \in S$ which we also call *landmark* or *site*. Two elements $s_1, s_2 \in S$ are then declared adjacent in step four, iff the respective tiles are touching each other by a sufficient amount, i.e. the number of nodes in one of the tiles with direct neighbors in the other tile is above some threshold. Finally, in step five, we collect the adjacencies to actually create faces of the reconstruction of our algorithm. Observe that only Step 1. involves the geometry of the sample set V, all other steps can be implemented fully combinatorially. Also, the only geometric predicate required in step one is the comparison of coordinates and distances between input points – very low degree predicates that can be evaluated exactly very efficiently using known floating-point filter techniques. We want to emphasize two things:

1. Our algorithm does not compute a reconstruction of V with respect to the original surface Γ (involving all $v \in V$) but only of a subsample $S \subset V$. For sufficiently dense sample sets, this reconstruction of S with respect to Γ still captures the topology of Γ. There is also a generic (and rather simple) way of incorporating the remaining points of V into the reconstruction (requiring some higher-degree geometric predicates, though), see [**?**]. In practice, with the presence of scanning devices producing millions or even billions of point samples, the fact that we only compute a reconstruction of a subsample is not a real concern.

2. The reconstruction computed by our algorithm does not only contain triangular faces, but also larger faces (though of size less than 5 in practice and of constant size in theory). If required, these non-triangular faces can be triangulated easily (requiring only low-degree geometric predicates).

In [**?**] we provide a theoretical justification for the correctness of our algorithm under the ε-sampling condition proposed in [**?**]. The core components of the correctness proof are:

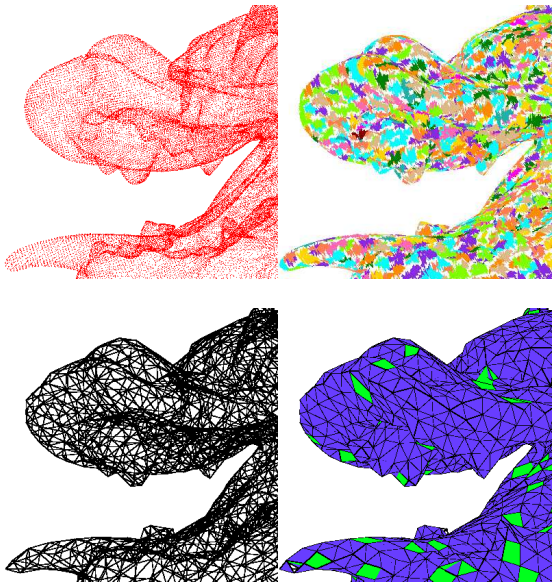

Figure 2: The Dragon, head close-up: point cloud, graph Voronoi diagram, identified adjacencies between subsamples and discovered faces (left to right, top to bottom)

- We show that the local neighborhood graph corresponds locally to a quasi-unit-disk graph for a set of points in the plane.
- The identified adjacencies locally form a planar graph.
- The faces of this graph have bounded size.

What does this mean? The graph that we constructed on the subsample of points S is a *mesh* that is locally planar and covers the whole 2-manifold. The mesh has the nice property that all its *cells* (aka faces) have constant size (number of bounding vertices). Therefore its connectivity structure faithfully reflects the topology of the underlying 2-manifold.

THEOREM 2.1. *([**?**]) The adjacencies created between samples in S form a graph which faithfully reflects the topology of the underlying 2-manifold.*

As can be read from Table 1, the running times are heavily dominated by the construction of the local neighborhood graph. We expect considerable improvements by an order of magnitude by performing batched nearest neighbor queries (and not asking for the κ nearest neighbors separately for each sample).

3 Implementation

We have implemented our algorithm in `C++`, using `Qt4` and `OpenGL` for rendering and the graphical user

Model	Points	Open	Read	Octr	Ngb	MIS	Vor	CDM	Cyc	Total	CoCone
Bunny	35947	0.83	0.39	0.14	3.42	0.45	0.29	0.22	0.05	5.79	29
Bone	177907	0.83	1.62	0.97	20.45	2.77	1.70	1.29	0.62	30.25	137
Hand	327323	0.83	3.08	2.13	36.87	4.77	2.81	2.31	0.86	53.66	1216
Dragon	435545	0.83	4.79	2.97	49.03	6.44	4.01	3.59	1.33	72.99	761
Buddha	543524	0.83	6.10	4.10	63.18	8.30	5.11	4.20	1.89	93.71	876
Blade	882954	0.83	6.95	6.23	104.32	14.31	8.45	7.09	4.74	152.92	>1800

Table 1: Time spent in different stages (seconds)

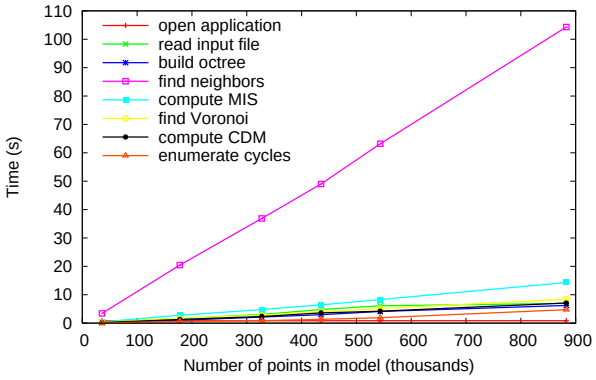

Figure 3: Plot of the time spent in different stages of the algorithm for different input sizes

interface.

For our implementation we make the simplifying assumption that the set of input data points V is a *locally uniform* sampling, which is typically true for sample data acquired by laser scanning (this means in particular that apart from a lower bound on the density of the sample there is also an upper bound, see the companion paper for a more precise definition). In that way we can determine the local neighborhood graph $G(V)$ (first step of our algorithm) by connecting a sample to its κ nearest neighbors ($\kappa = 15$ worked well in our experiments).

Also making use of the assumption that V is a locally uniform sample, we can declare two samples $s_1, s_2 \in S$ adjacent as follows: s_1 and s_2 are adjacent if the number of edges linking a sample v_1 (belonging to the graph Voronoi cell of s_1) to a sample v_2 (belonging to s_2's cell) is above a certain threshold a ($a = 7$ worked well in our experiments). Figure 2 illustrates the main steps of our algorithm using a close-up of the *Dragon* model.

3.1 Implementation Details We provide here supplementary information about our design choices.

Input. The input represents the points in a point cloud P and comes in a simplified `PLY` format, a simple object description designed as a convenient format for working with polygonal models. In our case there is no special header, since we are only interested in the points themselves. That is why each line in the file corresponds to a sample point and is a simple list of its three coordinates. We simply report but skip any line not conforming to this format.

Spatial representation. In order to allow for efficient positional queries, the points in V are stored in an octree. An octree is a structure suited for storing spatial data, which takes into consideration the relative distribution of points, backed up by an 8-ary tree. Each node in the tree represents a cube. The tree root corresponds to the bounding box of the point cloud and stores all the points. This cube is split in two along each axis, yielding eight smaller cubes.

A threshold o (called octree *granularity*) is imposed on the number of points that can lie within any node/cube. When the threshold is exceeded, the cube is split into eight smaller cubes of equal volume. They provide a refinement of the larger cube and this is recorded by making them the children of the tree node corresponding to the big cube. The process of splitting continues recursively and stops when all the tree leaves contain less than o points.

κ-nearest neighbors. The octree structure is particularly useful when looking for the k-nearest neighbors of a given point.

First, we perform a traversal of the tree starting at the root and aiming at ending in the leaf that contains the query point. The decisions about which direction to follow at each step in this traversal are made based on the geometric coordinates of the query point; three simple comparisons with the middle coordinates of the current node's bounding box along each axis are enough to decide which child to move to. As we go down in the traversal, we insert all siblings of nodes on this path into a priority queue based on the Euclidean distance from the query point to the circumsphere of the current node's corresponding cube. The priority queue's first element will contain the node with the *smallest* distance.

After finishing the traversal, a *top-κ*-like algorithm

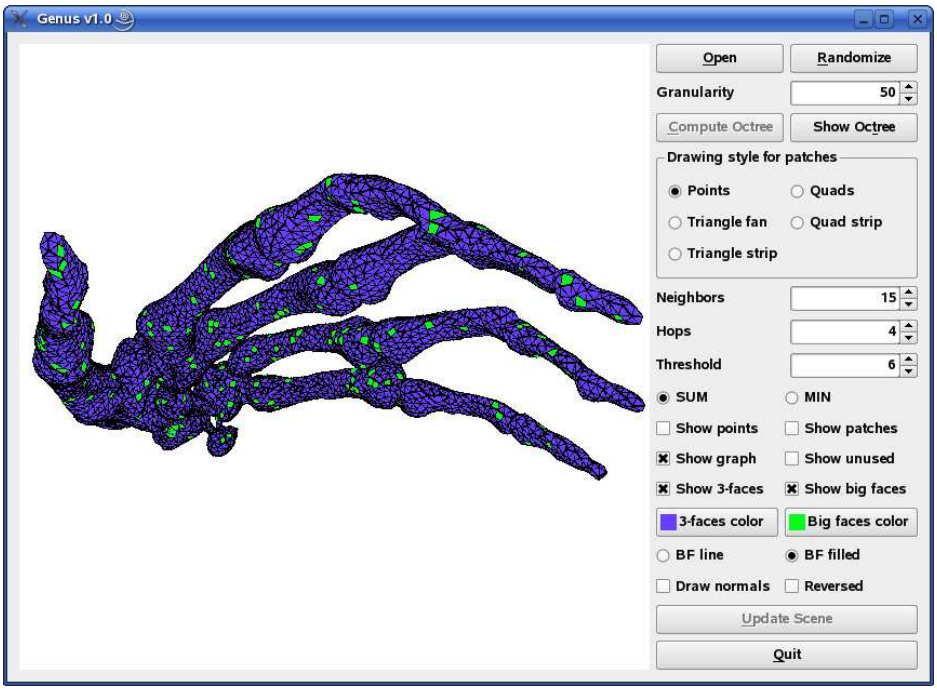

Figure 4: Main window of our application

is used. The first node in the priority queue is extracted and if it is a leaf, a test is performed for the points stored in the node to check whether they are eligible for the set of κ-nearest neighbors. For internal nodes, their eight children are inserted back into the queue. The algorithm ends when the distance d_{max} to the furthest nearest-neighbor cannot be improved, i.e. when the distance to the circumsphere of the node currently popped out of the queue is larger than d_{max}.

Neighborhood graph. We compute an (undirected) neighborhood graph $G_N = (V_N, E_N)$ by taking as vertices all the initial sample points V. For each of them we compute its κ-nearest neighbors as shown above, and we create an edge between points and each of its nearest neighbors discovered in this way.

Independent set. In the next step, a maximal independent set S in G_N is computed, such that any two vertices in S are at least k hops away.

To this end, we use a straightforward greedy algorithm: in the beginning, all vertices of G_N are marked as eligible. We choose an eligible vertex v and perform a breadth-first search to discover all vertices reachable in less than k hops from v, which are then marked as ineligible. This process continues until there are no more eligible vertices left.

Graph Voronoi diagram. The vertices in S are used as landmark nodes (sites) for a graph Voronoi diagram in the neighborhood graph G_N. Since the distance function for this diagram is simply the number of hops in the graph (i.e. all edges have unit weight), we use a parallel breadth-first search.

The processing queue is initialized with all the landmarks, then when a new vertex v is first seen, it is assigned to the vertex u from which it was reached. If u had already been assigned to a landmark ℓ (i.e. it is not a landmark itself), we assign v also to ℓ. In this way in the end we know for each vertex its "dominating" landmark, and we can easily determine the graph Voronoi cells.

Combinatorial Delaunay map. We create cell adjacencies in a conservative manner, leading to a so called combinatorial Delaunay map $CDM(S) = (V_{CDM(S)}, E_{CDM(S)})$. Let S_1 and S_2 be two graph Voronoi cells; the number $b_{S_1S_2}$ of points in S_1 with at least one (1-hop) neighbor in S_2 is calculated. If $b_{S_1S_2} + b_{S_2S_1}$ is greater than some threshold a, the points s_1 and s_2, corresponding to Voronoi cells S_1 and S_2 respectively, are added to $V_{CDM(S)}$, and the edge (s_1, s_2) is added to $E_{CDM(S)}$.

Face enumeration. The adjacencies on S are further used for face detection. Faces correspond to elementary cycles in this graph, with the supplementary constraint that any edge is allowed to appear in at most two cycles. Since we are interested in finding faces aka small cycles, we need an algorithm that enumerates the graph cycles in increasing order of cycle length. To this end, we implemented a simple enumeration algorithm

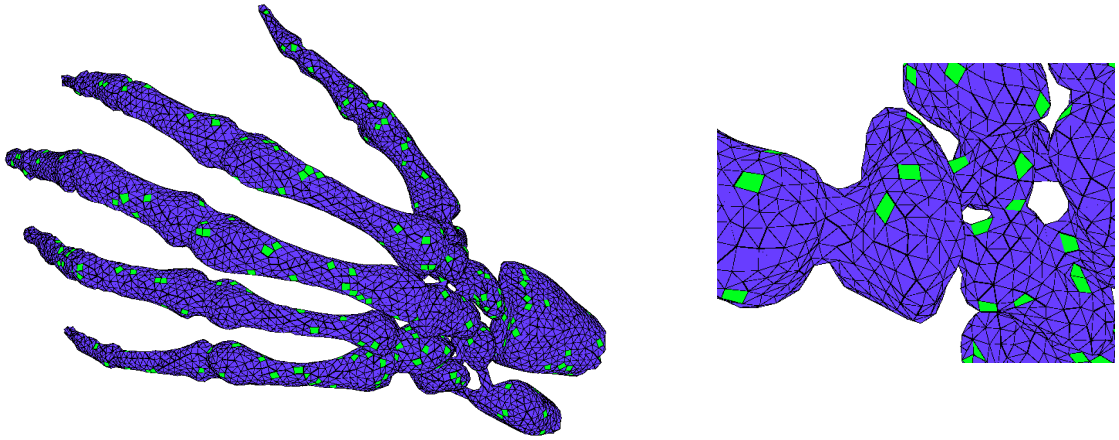

Figure 5: Skeleton hand: full view (left), metacarpi (right)

which first outputs cycles of length 3, then 4, and so forth, always only considering those edges which have not already been used in two cycles before.

4 Experimental Evaluation

4.1 Benchmarking on standard Data Sets Our reconstruction application was compiled with `g++ 3.3.5`, optimization `-O3`, and evaluation was made on a machine with a Pentium 4 processor at 3 GHz, 1 GB of RAM, running Debian Linux kernel version `2.6.13`. A screenshot of the main window of our implementation is given in Figure 4.

We benchmarked our implementation[4] on publicly available data sets obtained from laser scans of physical models, most of them from the *Large Geometric Models Archive* at Georgia Institute of Technology and the *3D Scanning Repository* at Stanford University.

In Table 1 you can see a detailed account of the running times spent in different parts of our implementation (*Octr* refers to the octree construction, *Ngb* to the construction of the local neighborhood graph; MIS, *Vor*, CDM denote respectively the three phases to construct S (the subsample), the tile partitioning and $CDM(S)$ (the adjacencies between the elements of the subsample), *Cyc* the routine to identify face cycles). We also ran[5] the CoCone algorithm [**?**] on the same datasets to give a rough comparison with other algorithms. We want to emphasize, though, that the Cocone algorithm does more than ours as it incorporates *all* sample points into the reconstruction (which would be done in a post-processing step of our algorithm which we expect to take only a fraction of the overall time). Figure 3 shows a plot of the same data. We never ran into any robustness problems with our algorithm, even when only employing floating-point arithmetic (the only real predicate that we require is the comparison of distances between points). The CoCone algorithm – as far as the authors reported to us – though, requires exact arithmetic for reliable operation due to its far more complex geometric tests like the insphere predicate.

In Figure 1 you can see the output of our algorithm for the dataset *Bunny*; the light color denotes non-triangular faces. Figure 6 shows the largest model tested (*Blade*), with almost 900,000 sample points. On the left, a full view of the model; at its right edge, we can observe the ragged structure, and on the lower part the reconstructed curved areas. Top right, the long sharp edge has been correctly discovered; we can also clearly recognize the concavity of the blade. Bottom right: close-up on the lower side of the model, where a screw hole pierces through the blade from one side to the other. In Figure 5 we see the model *Hand*. In the full view we observe that the phalanges are correctly individualized, as well as the wrist. On the right: close-up on the metacarpi: the holes in-between are visible; the algorithm detects the narrow spaces between the bones. Figure 7 shows one of the most difficult models (*Happy Buddha*) because of its curved features: the face, the hand-supported object, the necklace, and especially the folded outfit embellished with religious symbols. This causes sharp transitions, holes and concave regions. On top right, close-up on discovered openings in the cape. On bottom right, details of the bottom of the small round three-legged seat on which Buddha stands; its rims were correctly emphasized, as well as the decorative embossed rings just above the seat legs.

[4] The source code and some models are available online at `http://www.mpi-inf.mpg.de/~dumitriu/work/genus`

[5] We thank Tamal K. Dey for providing us with an executable of the CoCone code.

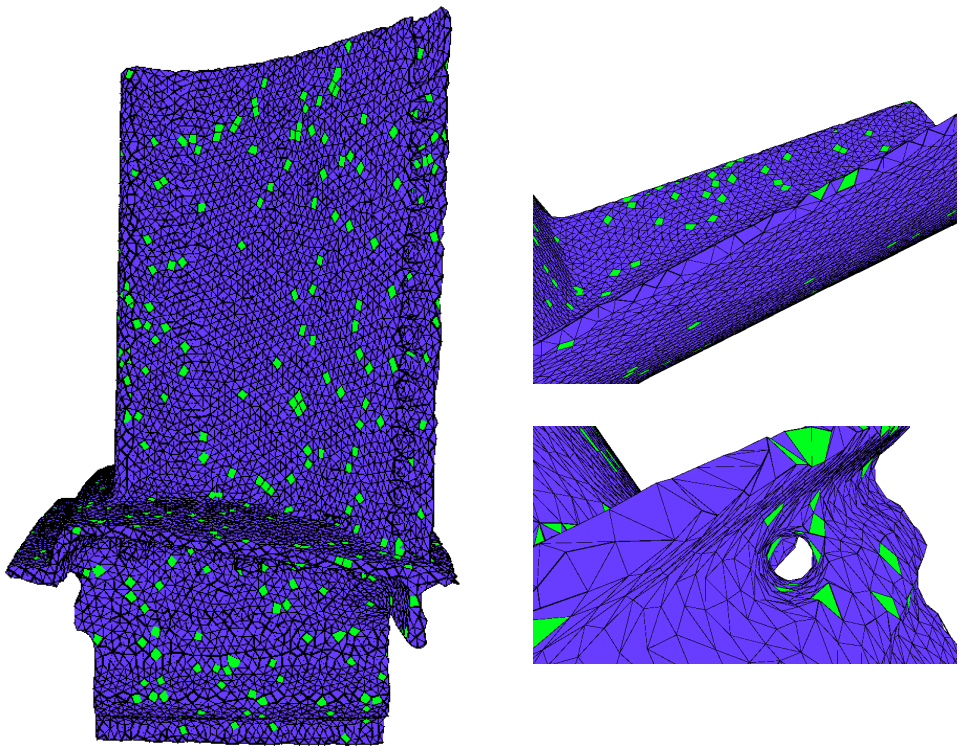

Figure 6: Turbine blade: full view (left), cutting edge and screw hole (right)

4.2 Parameter Variation We examine more closely the effect of varying parameters k and a on the experimental results obtained. In the following we use the model *Stanford Bunny*, fixing the other parameters at $o = 50$ (the octree is built with at most fifty original points stored in any leaf), and $\kappa = 15$ (for each input point, we find its fifteen nearest neighbors).

As a supplementary measure of the reconstruction quality, we also computed a value corresponding to the *genus* of the reconstruction if it forms a 2-manifold. The *genus* is a topologically invariant property, defined as the largest number of non-intersecting simple closed curves that can be drawn on the surface without separating it. Roughly speaking, it is the number of handles of an orientable 2-manifold.

The genus of a surface, also called the *geometric genus*, is related to the Euler's formula. In fact, Euler's formula is a particular version (for simply connected polyhedra, i.e. of genus 0) of the general relation

$$n - e + f = 2 - 2g \tag{4.1}$$

where n = number of vertices, e = number of edges, f = number of faces, and g = geometric genus value.

Solving out Equation 4.1 for g, we get

$$g = \frac{2 - n + e - f}{2} \tag{4.2}$$

We computed the genus value of our Bunny reconstruction according to the formula in Equation 4.2. A genus different than zero shows that something went wrong in the reconstruction (either the reconstruction does not form a 2-manifold or it does but exhibits handles). The results are shown in Table 2 and Figure 8.

Dependence on k To determine the dependence on k, we fixed the threshold a at 7 and varied k. For very small values of k, there is not enough "room" to identify reasonable adjacencies, hence the reconstruction does not patch up to a 2-manifold, leading to bad genus values. As we increase the parameter, things improve quickly (with a strange anomaly at $k = 7$ which we cannot explain at this time), of course at the cost of a much coarser subsampling.

Dependence on a This time we fixed k at 5 and varied a. A small value yields a denser combinatorial Delaunay map, with a higher percentage of triangles, but the genus number is extremely large (since there are too many adjacencies and the reconstruction does

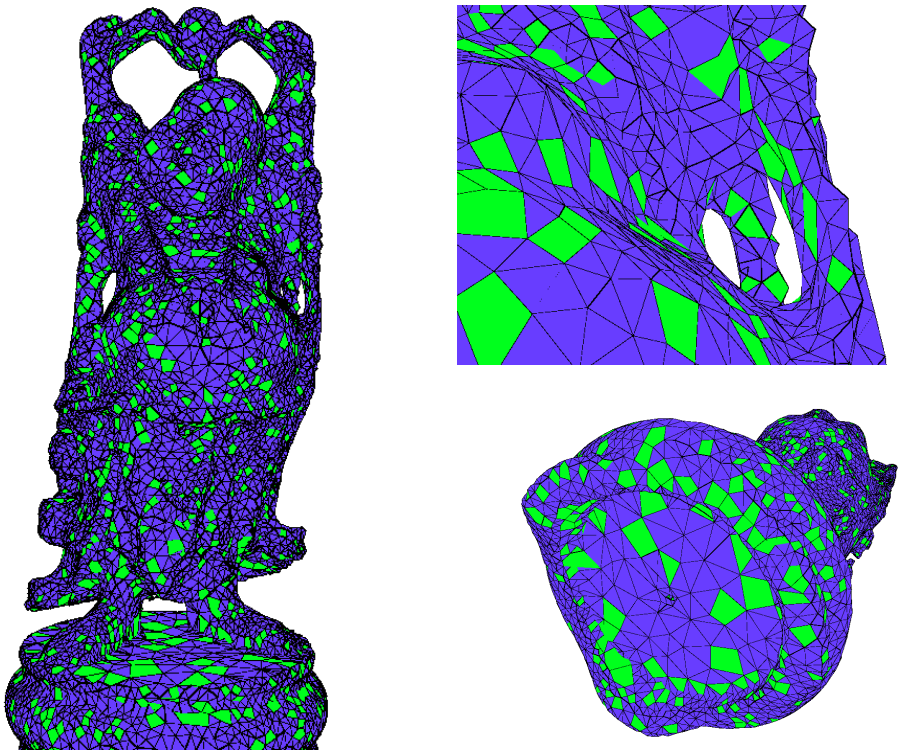

Figure 7: Happy Buddha: full view (left), side holes and bottom ridge (right)

k	faces	genus value
2	6415	62.5
3	3199	5
4	1620	1
5	976	0
7	457	1.5
14	94	0
15	80	0

a	faces	triangles	genus value
2	1024	99.4%	43
3	1023	99.5%	24
6	998	97.3%	1.5
7	976	95.3%	0
11	898	87.0%	0
15	778	73.3%	3.5
20	543	48.4%	18.5

Table 2: Dependence on k (left, for $a = 7$) and a (right, for $k = 5$)

not form a 2-manifold). Increasing the value causes a drastic drop in adjacencies and finally leads to a 2-manifold with correct genus. With larger values of a, of course, there are more and more non-triangular faces. Too large values for a lead to too few adjacencies and hence the reconstruction does not form a 2-manifold.

Overall, we consider k the most crucial parameter for our algorithm, which essentially determines the 'coarseness' of the subsample S. According to the theory in [**?**], k must be chosen very large to guarantee sufficient density of $CDM(S)$; of course, this comes at the cost of a very coarse subsample S (see Figure 8, top right). A too large k might 'smooth out' important details of the model. A too small choice of k, on the other hand, even in practice allows only for very few certified adjacencies (see Figure 8, top left). This results in larger faces. In our experiments, a value of $k = 5$ has proven to perform very well in practice, see Figure 8, top center.

4.3 Further Examples In Figure 9 we can see the Dragon model. The spurs on front and back legs are visible, as well as the front claw clenched on a ball. On bottom, a close-up of the scale ridge on the back; we can notice that the individual scales are clearly delimited.

The output of our algorithm on the Stanford Bunny can be seen in Figure 10. The main features are correctly detected, including paws, snout, and ears. On bottom, a close-up on the carved ears, whose concavity can be easily noticed (dots represent the original sample

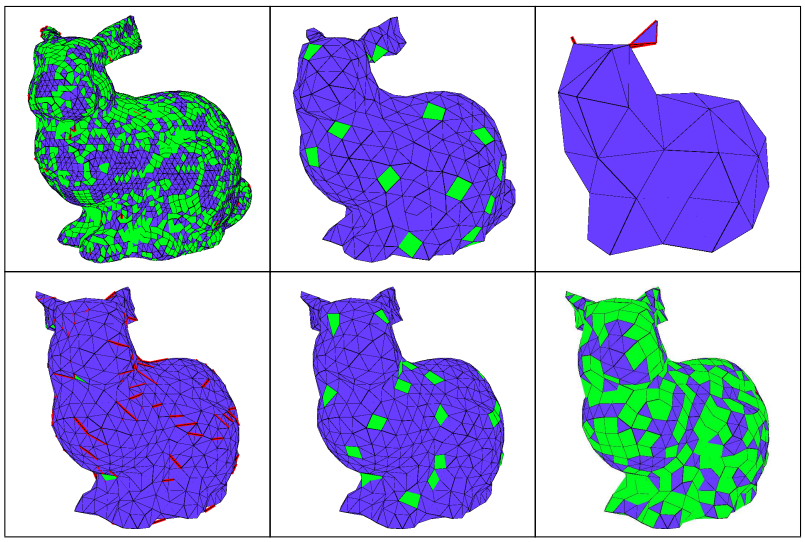

Figure 8: Effect of varying parameters k (top, for values $2, 5, 14$) and a (bottom, for values $2, 7, 15$); the thick red edges are not part of any face

points).

5 Outlook

Theoretically, our approach has the potential to work for reconstructing 2-manifolds even in higher dimensions; it does not extend to non-2-manifolds, though, as the "planarity property" of a graph that our algorithm crucially depends on (see [**?**] for more details) does not have an equivalent for non-2-manifolds.

On the practical side, we intend to complete our implementation and incorporate the pruned sample points into the reconstruction via weighted Delaunay triangulations. In the future, it might also be interesting to apply our algorithm to massive datasets; the inherently local computation and decision making exhibits nice locality properties which might be of use both in a parallel computing as well as an external memory scenario. Maybe the most interesting aspect of this paper is the fact that we were able to produce reasonable reconstructions of scanned 3D objects in a robust manner using only very simple geometric predicates (comparison of distances between points).

References

[1] N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. In *Proc. 14th ACM SoCG*, 1998.

[2] N. Amenta, S. Choi, T. K. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. In *Proc. 16th SoCG*, 2000.

[3] D. Dumitriu, S. Funke, M. Kutz, and N. Milosavljevic. On the locality of reconstructing a 2-manifold in $\mathbb{R}^3$. manuscript. http://www.mpi-inf.mpg.de/∼funke/Papers/LocalRecon.pdf.

[4] S. Fortune and V. Milenkovic. Numerical stability of algorithms for line arrangements. In *Symposium on Computational Geometry*, pages 334–341, 1991.

[5] S. Funke and N. Milosavljevic. Network Sketching or: "How Much Geometry Hides in Connectivity? – Part II". In *Proc. ACM-SIAM SODA*, pages 958–967, 2007.

[6] S. Funke and E. Ramos. Smooth-surface reconstruction in near-linear time. In *Proc. ACM-SIAM SODA*, 2002.

[7] L. Kettner, K. Mehlhorn, S. Pion, S. Schirra, and C. Yap. Classroom examples of robustness problems in geometric computations. In *12th Annual European Symposium on Algorithms*, volume 3221 of *LNCS*, pages 702–713, Bergen, Norway, 2004. Springer.

[8] V.J. Milenkovic. Verifiable implementation of geometric algorithms using finite precision arithmetic. *Artif. Intell.*, 37(1-3):377–401, 1988.

[9] K. Sugihara, Y. Ooishi, and T. Imai. Topology-oriented approach to robustness and its applications to several Voronoi-diagram algorithms. In *Proc. 2nd Canad. Conf. Comput. Geom.*, pages 36–39, 1990.

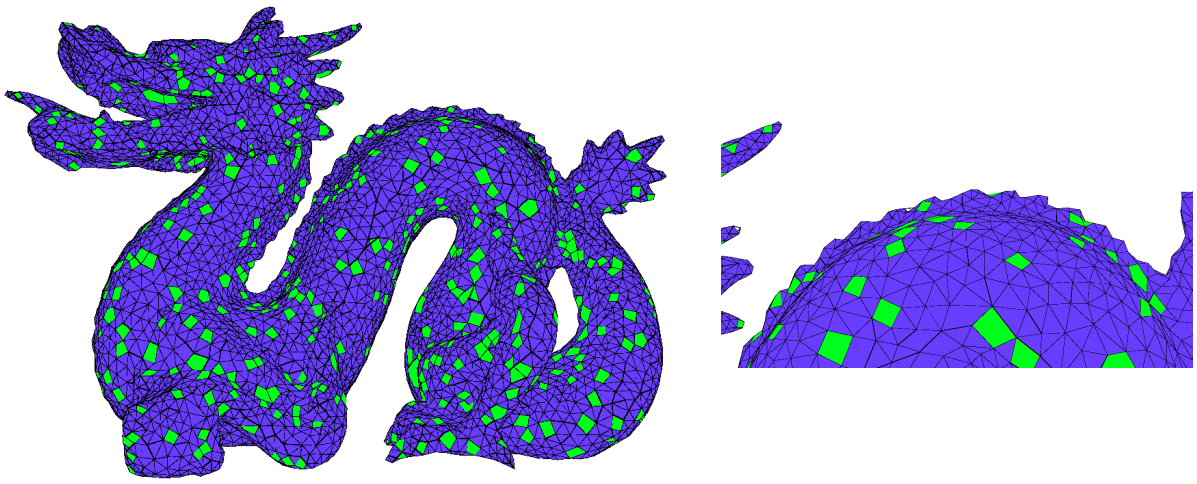

Figure 9: Dragon: full view (left), back scales (right)

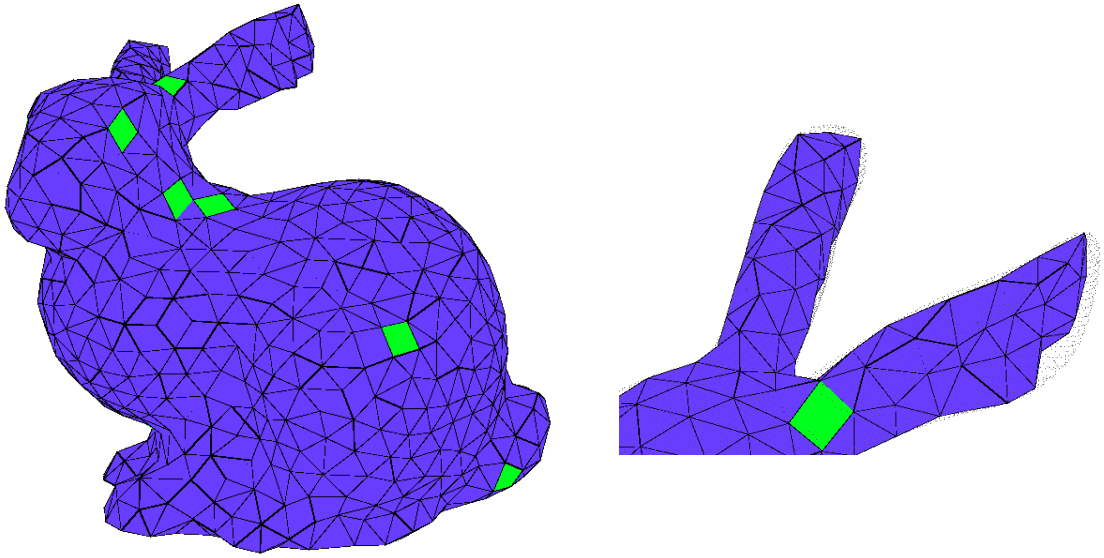

Figure 10: Stanford bunny: full view (left), carved ears (right)