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

DANIEL DUMITRIU and MARTIN KUTZ
Max-Planck-Institut für Informatik, Saarbrücken, Germany
STEFAN FUNKE
Ernst-Moritz-Arndt-Universität, Greifswald, Germany
and
NIKOLA MILOSAVLJEVIĆ
Stanford University, Stanford, USA

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 employment of advanced datatypes for exact arithmetic, as provided by libraries like CORE [Karamcheti et al. 1999], LEDA [Mehlhorn et al. 1997] or GMP [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. More precisely, our algorithm only requires *distances* between the sample points and not the actual embedding in $\mathbb{R}^3$. 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 [Dumitriu et al. 2008].

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*surface representations*; *geometric algorithms*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*geometrical problems and computations*; *computations on discrete structures*; G.2.3 [**Discrete Mathematics**]: Applications

General Terms: Algorithms, Design, Experimentation, Theory

Additional Key Words and Phrases: Combinatorial surface reconstruction, conservative adjacencies creation

---

Authors' addresses: Daniel Dumitriu, Max-Planck-Institut für Informatik, Campus E 1.4, 66123 Saarbrücken, Germany; e-mail: `dumitriu@mpi-inf.mpg.de`. Stefan Funke, Institut für Mathematik und Informatik, Jahnstr. 15a, 17487 Greifswald, Germany; e-mail: `stefan.funke@uni-greifswald.de`. Nikola Milosavljević, Clark Center, Room S257, 318 Campus Drive, Stanford, CA 94305, USA; e-mail: `nikolam@stanford.edu`. Part of this work has been done while the third author was at the Max-Planck-Insitut für Informatik.

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*[2] 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 [Kettner et al. 2004]. 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 falling back to 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 [Fortune and Milenkovic 1991; Milenkovic 1988; Sugihara et al. 1990]. Sugihara et al. [1990] 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 also terminates if the outcome of all geometric predicates are completely random.

In this paper we present an algorithm for reconstructing a 2-manifold in $\mathbb{R}^3$ in the spirit of the work by Sugihara et al. The main idea is that we require exact evaluation of a few *simple* (i.e., low-degree) predicates only at the beginning of the algorithm. From that point, our algorithm is mostly combinatorial, and does not evaluate any geometric predicates. Nevertheless, we show (both experimentally as well as theoretically in a companion paper [Dumitriu et al. 2008]) that the output of our algorithm is a good approximation to the original 2-manifold. Most importantly, our algorithm produces this output *without* any evaluation of a complicated geometric test, such as 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

[2]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.

a certain *sampling condition*, the output of the respective algorithm is guaranteed to be 'close' to the original surface.

Amenta and Bern [1998] 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*[3] of Γ. A set of points $V \subset \Gamma$ is called an 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 computing the Voronoi diagram/Delaunay triangulation of a point set in $\mathbb{R}^3$, which incurs both an inherent $\Omega(n^2)$ running time and the need for exact evaluation of the *insphere predicate*.

Funke and Milosavljević [2007] 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.

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 solely from the distances between the sample points, i.e., with no real geometry involved; secondly, created adjacencies/edges are 'conservative', in a sense that two samples are only connected if they are adjacent in all 'reasonable' reconstructions. Interestingly, we can show (in the theoretical companion paper [Dumitriu et al. 2008]) 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 unrealistically high sampling density – like most algorithms that come with a theoretical guarantee, such as CRUST [Amenta and Bern 1998] or CoCone [Amenta et al. 2000] – this paper shows that our algorithm is actually quite practical for real-world datasets. Due to the local nature of computation in our algorithm, there is also potential for use 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 of $G(V)$. To keep the presentation simple, we assume that V is a *uniform*[4] ε-sample from a closed smooth 2-manifold Γ in $\mathbb{R}^3$. In practice, sample sets are indeed often close to

[3] The medial axis of Γ is defined as the set of points which have at least two closest points on Γ.
[4] V is a uniform ε-sample if for any $p \in \Gamma\ \exists s \in V$ with $|ps| \leq \varepsilon$.

uniform, and even small non-uniformities in the sample set never prevented our algorithm from working in practice. In theory, a preprocessing stage can enforce *local uniformity*, which is sufficient to prove correctness of our algorithm (see [Dumitriu et al. 2008] 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 its κ nearest neighbors
(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 1 we approximate the creation of a unit-disk graph of the point set, where two samples are connected if and only if their distances are less than some constant times ε. To be independent of the sampling density parameter ε, we chose to create adjacencies to the κ nearest neighbors each. In practice, for values of κ around 15, we obtained very good results, independent of the scale of the scanned objects, that is, without the algorithm knowing the actual value of ε.

In Step 2 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 be computed easily in a greedy fashion.

In Step 3, 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 *graph Voronoi cells* (also called *tiles*) consisting of nodes which have the same closest $s \in S$, called *landmark* or *site*.

Two elements $s_1, s_2 \in S$ are then declared adjacent, in Step 4, if and only if 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, more precisely, if there is a path between s_1 and s_2 all of whose nodes are at least two hops away from any tile other than those of s_1 and s_2.

Finally, in Step 5, we collect the adjacencies to actually create faces of the reconstruction of our algorithm. Observe that only the first step involves some geometry of the sample set V. In fact, it does not really require geometry, but only *distances* between the sample points. If the geometry is given, the respective distance comparisons involve only very low-degree predicates that can be evaluated exactly and efficiently using known floating-point filter techniques. All other steps can be implemented fully combinatorially.

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 [Funke and Ramos 2002]. 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 less of a 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 [Dumitriu et al. 2008] we provide a theoretical justification for the correctness of (a slight modification of) our algorithm under the ε-sampling condition proposed by Amenta and Bern [1998]. The core components of the correctness proof are:

—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.

This implies that 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* (i.e., faces) have constant size (number of bounding vertices). Therefore it faithfully reflects the topology of the underlying 2-manifold.

3. IMPLEMENTATION

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

As mentioned before, 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). This actually makes the local neighborhood graph $G(V)$ (determined by connecting a sample to its κ nearest neighbors) look quite similar to a unit-disk graph, where each sample is connected to all its nearby sample points within distance $O(\varepsilon)$. The value of $\kappa = 15$ worked well in our experiments.

Also using 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). Note that this criterion is somewhat different from the theoretically prescribed Step 4 of the algorithm, although the two are quite similar in spirit, i.e., both require that the two tiles have 'significant common boundary'. We opt for the threshold-based rule because we find it easier to implement, and expect it to be faster in practice because its computation involves only two graph Voronoi cells (per adjacency check) and is therefore more 'local'.

Figure 1 illustrates the main steps of our algorithm using a close-up of the Dragon model.

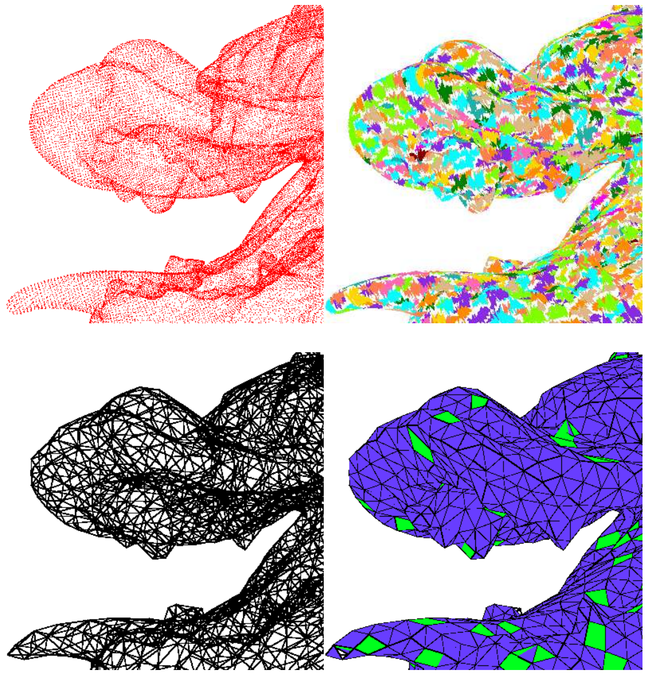

Fig. 1. The Dragon model, head close-up: point cloud, graph Voronoi diagram, identified adjacencies between subsamples and discovered faces (left to right, top to bottom)

3.1 Implementation Details

We provide here some additional 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 relative distribution of points. Each node in this 8-ary tree represents a cube in $\mathbb{R}^3$. The root of the tree corresponds to a bounding cube of the points, 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 for computing the κ 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 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, we compute a maximal set S in G_N 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 the 'dominating' landmark for each vertex, and we can easily determine the graph Voronoi cells.

Landmark adjacencies. We create landmark adjacencies in a conservative manner, i.e., only if their graph Voronoi cells have significant 'common boundary'. Let S_1 and S_2 be two graph Voronoi cells corresponding to landmarks s_1 and s_2 respectively; we calculate the number $b_{S_1 S_2}$ of points in S_1 with at least one (1-hop) neighbor in S_2. If $b_{S_1 S_2} + b_{S_2 S_1}$ is greater than some threshold a, we make s_1 and s_2 adjacent.

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, i.e., small cycles, we need an algorithm that enumerates

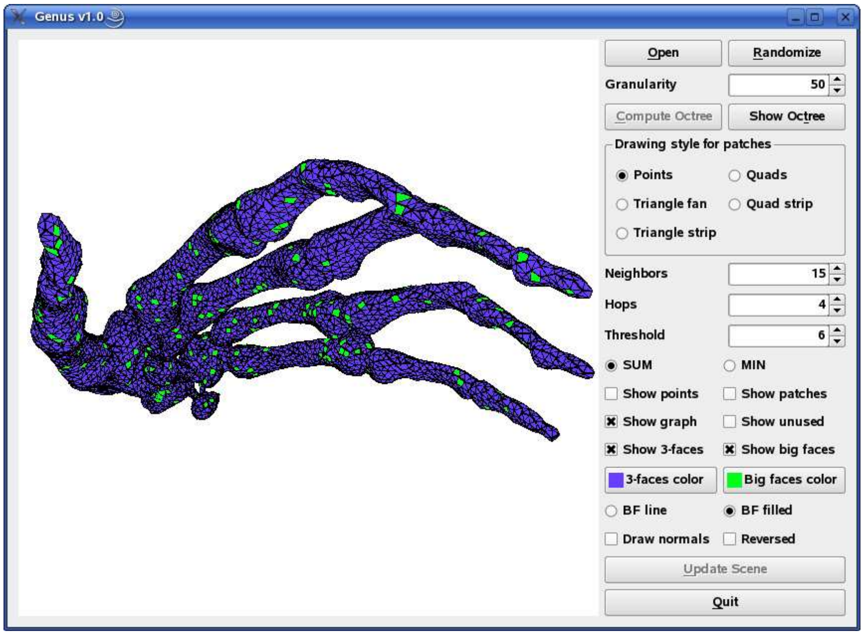

Fig. 2. Main window of our application

the graph cycles in increasing order of cycle length. To this end, we implemented a simple enumeration algorithm 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

Our reconstruction application was compiled with `g++ 3.3.5`, optimization level `-O3`, and evaluation was performed 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 2.

4.1 Benchmarking on Standard Data Sets

We benchmarked our implementation[5] 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.

Table I shows a detailed account of the running times spent in different parts of our implementation. We also ran[6] the CoCone algorithm [Amenta et al. 2000] 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 in-

[5]The source code of the application and some models used in our experiments are available online at `http://www.mpi-inf.mpg.de/~dumitriu/work/genus`

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

Table I. Time spent in different stages (in seconds). Octr: octree, Ngb: local neighborhood graph, MIS: landmark set, Vor: graph Voronoi diagram, Adj: landmark adjacencies, Cyc: face cycles.

Model	Points	Open	Read	Octr	Ngb	MIS	Vor	Adj	Cyc	Total	CoCone
Bunny	35947	0.8	0.3	0.1	3.4	0.4	0.2	0.2	0.1	5.7	29
Bone	177907	0.8	1.6	0.9	20.4	2.7	1.7	1.2	0.6	30.2	137
Hand	327323	0.8	3.0	2.1	36.8	4.7	2.8	2.3	0.8	53.6	1216
Dragon	435545	0.8	4.7	2.9	49.0	6.4	4.0	3.5	1.3	72.9	761
Buddha	543524	0.8	6.1	4.1	63.1	8.3	5.1	4.2	1.8	93.7	876
Blade	882954	0.8	6.9	6.2	104.3	14.3	8.4	7.0	4.7	152.9	>1800

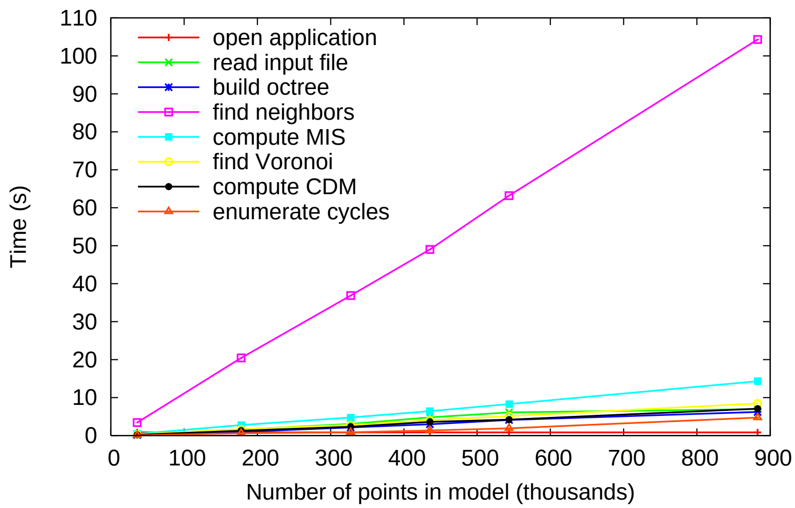

Fig. 3. Plot of the time spent in different stages of the algorithm for different input sizes

corporates *all* sample points into the reconstruction (which would be done in a postprocessing 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.

As can be read from Table I, the running times are heavily dominated by the construction of the local neighborhood graph. 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) might be possible.

The output of our algorithm on the Bunny model can be seen in Figure 5; light-colored faces are non-triangular. The main features are correctly detected, including paws, snout, and ears. On the right we give a close-up on the carved ears, whose concavity can be easily noticed (dots represent the original sample points).

Figure 4 shows the largest model tested (Blade), with almost 900,000 sample points. We show 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; on top right, the long sharp edge has been correctly discovered; we can also clearly recognize the concavity of the blade. On the bottom right we give close-up on the lower side of the model, where a screw hole pierces through the blade from one side to the other.

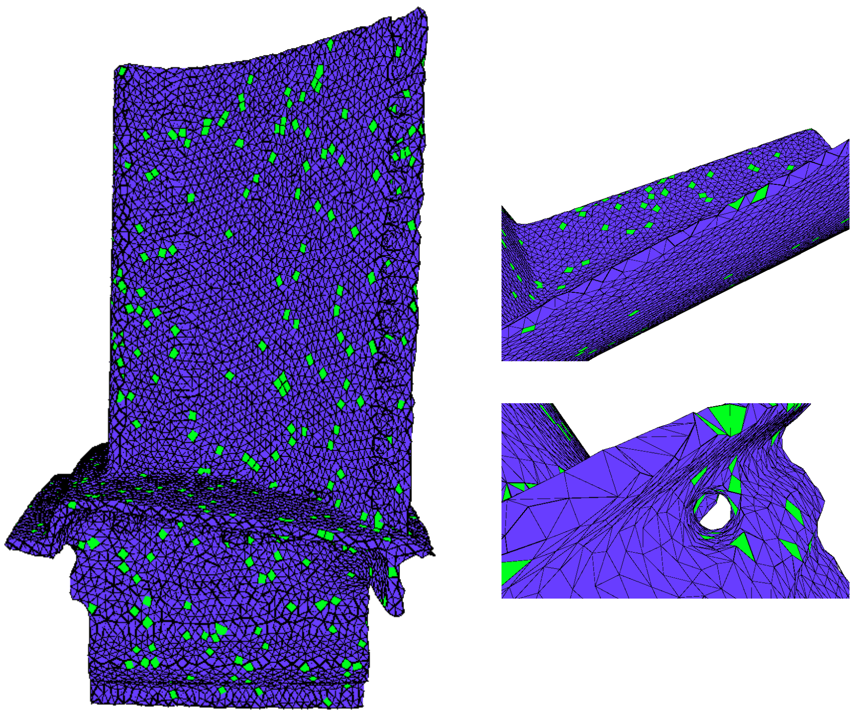

Fig. 4. Turbine blade: full view (left), cutting edge and screw hole (right)

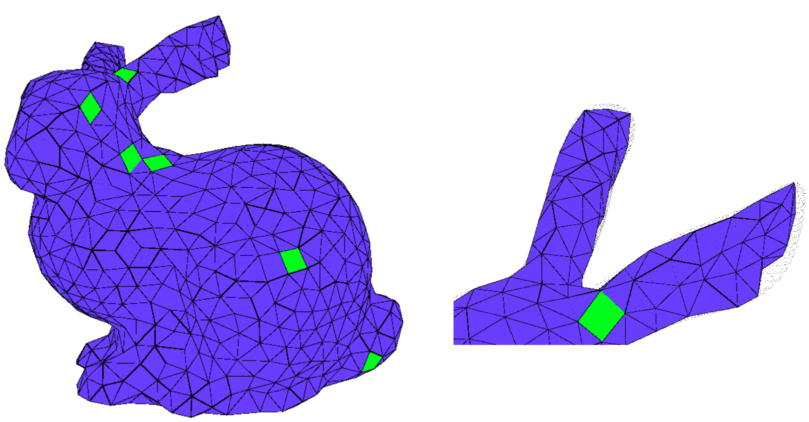

Fig. 5. Bunny: full view (left), carved ears (right). Due to the conservative adjacency creation, some faces (light color) are non-triangular, but can easily be triangulated later on.

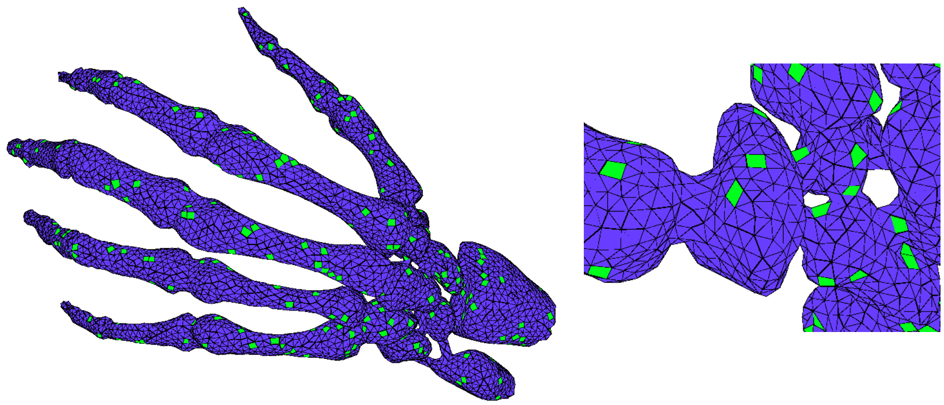

Fig. 6. Hand: full view (left), metacarpi (right)

In Figure 6 we see the Hand model. In the full view we observe that the phalanges are correctly separated, 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. We notice on top right, close-up on discovered openings in the cape, and 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.

Figure 8 shows the Dragon model. The spurs on front and back legs are visible, as well as the front claw clenched on a ball. On the right we show a close-up of the scale ridge on the back; notice that the individual scales are clearly delimited.

4.2 Robustness and Accuracy

Both the CoCone algorithm and our algorithm never exhibited any robustness issues. The reason in case of CoCone is the fact that it is based on an exact geometry kernel provided by CGAL [cga], i.e., all geometric predicates (in particular the insphere test) are guaranteed to return the correct result. This, of course, comes at the cost of an increased running time compared to a pure (and typically incorrect) floating-point implementation where geometric predicates might err. Our algorithm, on the other hand, is robust because its flow of computation is hardly determined by the evaluation of geometric predicates. We were told by the authors of CoCone that their algorithm actually requires an underlying exact geometry kernel, otherwise its output is flawed or the program crashes even for simple problem instances.

It would certainly be interesting to compare with a floating-point-only CoCone version and see what datasets can be robustly reconstructed with our algorithm, but not with the floating-point CoCone implementation. Unfortunately, we do not have access to the CoCone source code, and hence cannot replace the exact geometry

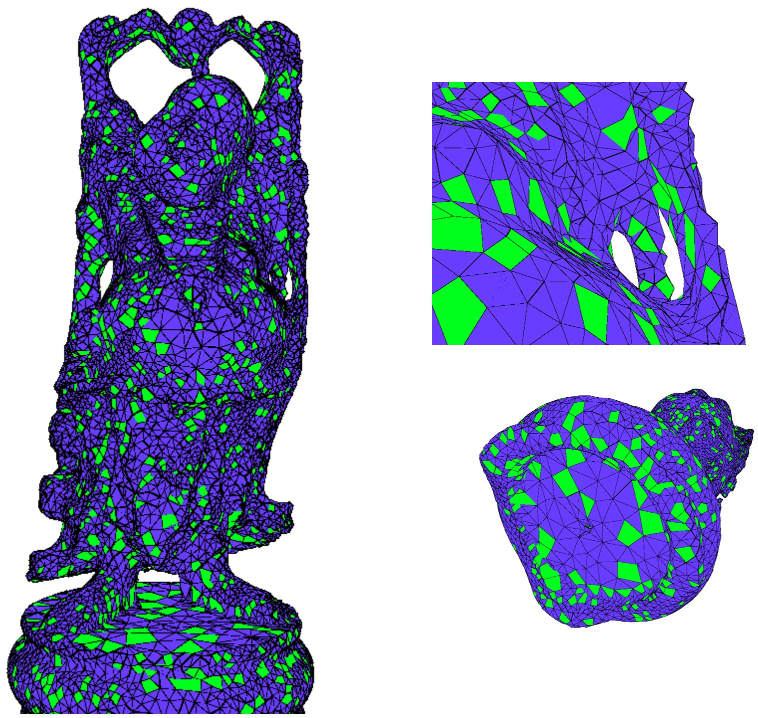

Fig. 7. Happy Buddha: full view (left), side holes and bottom ridge (right)

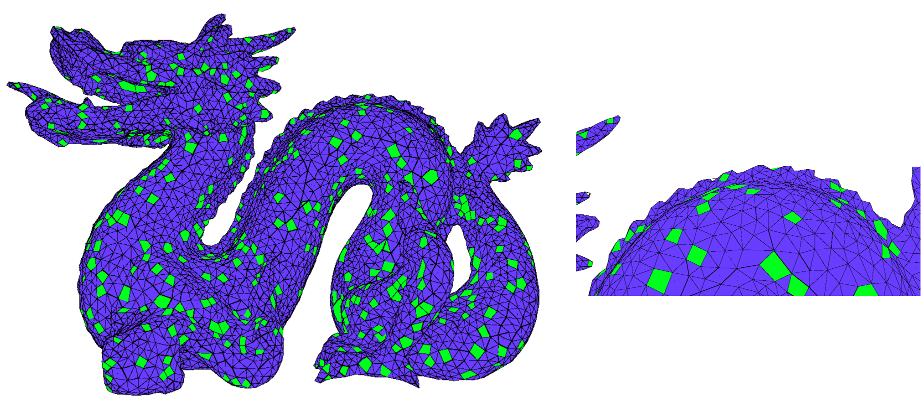

Fig. 8. Dragon: full view (left), back scales (right)

kernel by a float kernel. Equally interesting is an analysis of the amount of time CoCone spends for the exact predicate evaluations. The common way to (lower) bound the time spent in the arithmetic parts is to evaluate all predicates twice (this only yields a lower bound since evaluating twice typically requires less than twice the time due to cache effects and other issues related to the implementation of exact arithmetic) and use the timings to infer the time spent on arithmetic. Again,

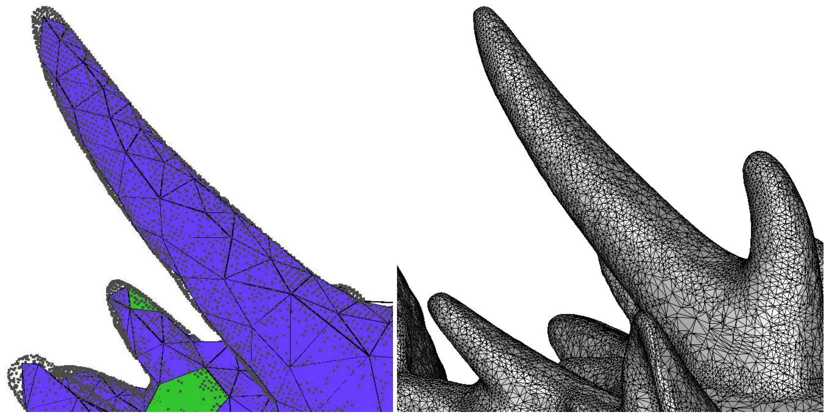

Fig. 9. Detailed view on subsampled reconstruction of our algorithm (left) vs. reconstruction computed by CoCone (right).

without the source code of CoCone it is hard to break down the execution time of Cocone into arithmetic and non-arithmetic parts.

Regarding the issue of accuracy: a typical measure of quality of a reconstruction is the distance from the 'real surface', here one would consider both the CoCone output as well as the output from our algorithm. Unfortunately, we do not have the 'real surface' at hand. In Figure 9 we show two close-ups of the same region for the Dragon model – one from CoCone, the other from our algorithm. This should give an idea of what the difference in the output is (though certainly not a quantitative comparison). With our algorithm using a subsample, it is certainly not as close to the 'real surface' as CoCone's output, its topology is correctly determined, though. How many samples are removed during the execution of our algorithm can be seen in Table II. Using the method described in [Funke and Ramos 2002], one could reinsert the removed samples to obtain a reconstruction with respect to all samples. This will require some higher-degree predicates, though.

4.3 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 Bunny model, 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

Table II. Size of the selected subsample for some models ($\kappa = 15$, $k = 5$)

Model	Points	Subsample size
Bunny	35947	868
Bone	177907	3649
Hand	327323	7822
Dragon	435545	9979
Buddha	543524	12245
Blade	882954	20973

Table III. Dependence on k (left, for $a = 7$) and a (right, for $k = 5$) (the Bunny model)

k	faces	genus
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
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

a value corresponding to the *genus* of the reconstruction if it forms a 2-manifold. The *genus* is a topological invariant, defined as the largest number of mutually non-intersecting simple closed curves that can be drawn on the surface such that the surface stays connected after it has been cut along the curves. 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., manifolds of genus 0) of the general relation

$$n - e + f = 2 - 2g \tag{1}$$

where $n =$ number of vertices, $e =$ number of edges, $f =$ number of faces, and $g =$ geometric genus. Solving Equation 1 for g, we get

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

We computed the genus of our Bunny reconstruction according to the formula in Equation 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 III and Figure 10.

Dependence on $\boldsymbol{k}$. To determine the dependence on k, we fixed the threshold a at 7 and varied k. For 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. 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 $\boldsymbol{a}$. 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 is extremely large (since there are too many adjacencies and the reconstruction does 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

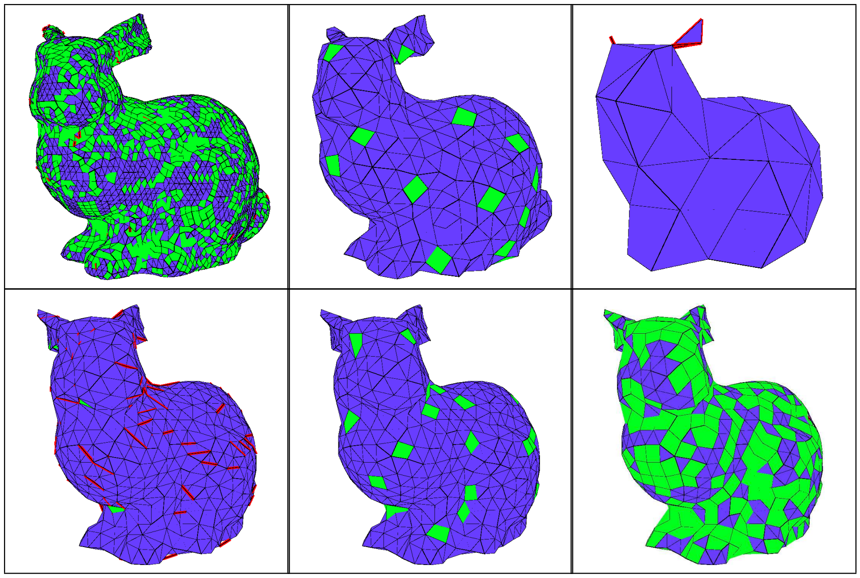

Fig. 10. Effect of varying parameters k (top, for values $2, 5, 14$) and a (bottom, for values $2, 7, 15$); the thick red edges are not part of any face

in [Dumitriu et al. 2008], k must be chosen extremely large to guarantee sufficient density of the landmark adjacency graph; of course, this comes at the cost of a coarse subsample S (see Figure 10, top right). A too large value of 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 few certified adjacencies (see Figure 10, top left). This results in larger faces. In our experiments, a value of $k = 5$ has proven to perform well in practice also for the other models, see Figure 10 (center top).

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 [Dumitriu et al. 2008] for more details) does not have an equivalent for non-2-manifolds.

On the practical side, one might look into incorporating the pruned sample points into the reconstruction via weighted Delaunay triangulations; this would lead to a more 'geometric' algorithm with more complicated, higher-degree predicates, though. A probably more interesting field of research would be 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.

In summary, we consider the most interesting aspect of this paper the fact that we were able to produce reasonable reconstructions of scanned 3D objects in a very robust manner using only simple geometric predicates (comparison of distances between points).

ACKNOWLEDGMENT

In memory of our dear friend and colleague Martin Kutz.

REFERENCES

GNU Multiple Precision Arithmetic Library. http://gmplib.org.

Cgal, Computational Geometry Algorithms Library. http://www.cgal.org.

Amenta, N. and Bern, M. 1998. Surface reconstruction by Voronoi filtering. In *SCG '98: Proceedings of the fourteenth annual symposium on Computational geometry*. ACM, New York, NY, USA, 39–48.

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

Dumitriu, D., Funke, S., Kutz, M., and Milosavljević, N. 2008. On the Locality of Extracting a 2-Manifold in $\mathbb{R}^3$. In *11th Scandinavian Workshop on Algorithm Theory (SWAT)*, J. Gudmundsson, Ed. Lecture Notes in Computer Science, vol. 5124. Springer, 270–281.

Fortune, S. and Milenkovic, V. 1991. Numerical stability of algorithms for line arrangements. In *SCG '91: Proceedings of the seventh annual symposium on Computational geometry*. ACM, New York, NY, USA, 334–341.

Funke, S. and Milosavljević, N. 2007. Network sketching or: "How Much Geometry Hides in Connectivity?–Part II". In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 958–967.

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

Karamcheti, V., Li, C., Pechtchanski, I., and Yap, C. 1999. A core library for robust numeric and geometric computation. In *15th ACM Symp. on Computational Geometry, 1999*. 351–359.

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

Mehlhorn, K., Näher, S., and Uhrig, C. 1997. The LEDA Platform of Combinatorial and Geometric Computing. In *ICALP '97: Proceedings of the 24th International Colloquium on Automata, Languages and Programming*. Springer-Verlag, London, UK, 7–16.

Milenkovic, V. 1988. Verifiable implementation of geometric algorithms using finite precision arithmetic. *Artif. Intell. 37,* 1-3, 377–401.

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

Received Month Year; revised Month Year; accepted Month Year