Simple and Scalable Surface Reconstruction

Dobrina Boltcheva, Bruno Lévy

Simple and Scalable Surface Reconstruction

Dobrina Boltcheva [1,2] and Bruno Lévy [2]

[1] Univerity of Lorraine - LORIA [2] INRIA Nancy

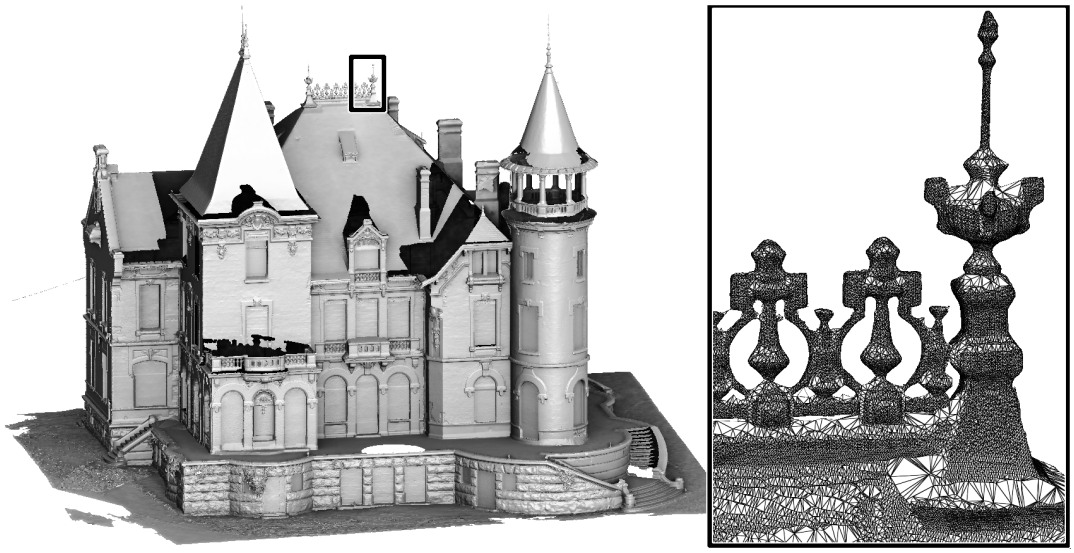


Figure 1: *Our method applied to a pointset with 33 million points. The total computation took 360 seconds (49 sec. for 2 iterations of smoothing and normal estimation, 78 sec. reconstruction, 61 sec. surface extraction, 114 sec. post-processing and 58 sec. loading and saving)*

Abstract

We present a practical reconstruction algorithm that generates a surface triangulation from an input pointset. In the result, the input points appear as vertices of the generated triangulation. The algorithm has several desirable properties: it is very simple to implement, it is time and memory efficient, and it is trivially parallelized. On a standard hardware (core i7, 16Gb) it takes less than 10 seconds to reconstruct a surface from 1 million points, and scales-up to 36 million points (then it takes 350 seconds). On a telephone (ARMV7 Neon, quad core), it takes 55 seconds to reconstruct a surface from 900K points. The algorithm computes the Delaunay triangulation of the input pointset restricted to a "thickening" of the pointset (similarly to several existing methods, like alpha-shapes, crust and co-cone). By considering the problem from the Voronoi point of view (rather than Delaunay), we use a simple observation (radius of security) that makes the problem simpler. The Delaunay triangulation data structure and associated algorithms are replaced by simpler ones (KD-Tree and convex clipping) while the same set of triangles is provably obtained. The restricted Delaunay triangulation can thus be computed by an algorithm that is not longer than 200 lines of code, memory efficient and parallel. The so-computed restricted Delaunay triangulation is finally post-processed to remove the non-manifold triangles that appear in regions where the sampling was not regular/dense enough.

Sensitivity to outliers and noise is not addressed here. Noisy inputs need to be pre-processed with a pointset filtering method. In the presented experimental results, we are using two iterations of projection onto the best approximating plane of the 30 nearest neighbours (more sophisticated ones may be used if the input pointset has many outliers).

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Boundary representations & Curve, surface, solid, and object representations

1. Introduction

Over the past two decades there has been a significant amount of effort dedicated to the problem of surface reconstruction. The problem of surface reconstruction may be formulated as follows: given a sampling of points from a surface, recover the original surface from which those points were sampled. This general problem is motivated by numerous applications in reverse-engineering, prototyping, visualisation, or computer vision since a growing variety of scanning devices nowadays provides measurements of objects in the form of point sets.

In the surface reconstruction problem we are given only a finite sampling $P \subset \mathbb{R}^3$ of an unknown surface S. The goal is to build a model of the surface S from the sampling P which is referred to as a reconstruction of S from P. It is generally represented as a triangulated surface mesh that can be directly used by computer programs from further processing. The reconstruction should match the original surface in terms of geometric and topological properties. In general, surface reconstruction is highly under-constrained and there are multiple triangulated surfaces that might fulfil these criteria. The difficulty of meeting geometric and topological criteria depends on properties of the sampling and on properties of the sampled surface. In particular, sparsity, redundancy, occlusion, noise, non-smoothness and boundaries make surface reconstruction a challenging problem.

Surface mesh reconstruction methods can be divided into methods that approximate the points by an implicit function and methods that connect the points to form a surface mesh. The first approach is widely spread because it generates smooth and closed meshes, usually by extracting the zero level of some potential field. The second type of approach is useful whenever it is desired to keep the input points, for instance when they represent measurements. In geological applications, the points are usually obtained by complex drilling simulations and they are expected to be contained in the final mesh for further processing. In this paper, we are interested in the second kind of approach, which aims at building a mesh by connecting the input points.

We present a new reconstruction algorithm that generates a surface triangulation from an input point set. Our algorithm is designed to process a clean and accurately registered point set, and does not attempt to average out noise, outliers or residual registration errors. The algorithm has several interesting properties: it is very simple to implement and it is time and memory efficient.It is "embarrassing parallel" and outperforms the concurrent algorithms (by two order of magnitude in some cases). Our algorithm uses only one data structure (a kd-tree), and accommodates evenly sampled irregular points sets, as demonstrated in the results. It handles large pointsets - up to 36 million points in minutes on an off-the-shelf PC.

2. Related work

There are two classes of approaches which differ as to whether they approximate or interpolate the input points.

The approximation methods aim to fit the best surface approximating the sampling and they usually ignore the sampling noise in the input data. A recent state-of-the-art and a benchmark of this family of algorithms can be found in [BLN*13, BTS*14]. These methods usually fit the points using the zero set of an implicit function, such as a signed distance function [HDD*92, CL96, BC02], a sum of radial bases [CBC*01], piecewise polynomial functions [ABCO*03, OBA05, NOS09] or an indicator function [KBH06, KH13, MPS08, ACSTD07]. In particular, the Poisson surface reconstruction method solves for an approximate indicator function of the inferred solid, whose gradient best matches the input normals. The output scalar function, represented in an adaptive octree, is then iso-contoured using an adaptive marching cubes [LC87]. Recently, a screened Poisson algorithm [KH13] has been introduced which resolves the over-smoothing problem and offers the ability to reconstruct meshes with boundaries.

Most of these methods estimate a signed-distance function. If the input points are unoriented, an important step is to correctly infer the sign of the resulting distance field [ACSTD07, MdGD*10]. These approaches are well-equipped to handle various imperfections in the data. However, normal estimation in the presence of imperfect data remains an open problem [MN03, DLS05].

The interpolation methods elaborate upon Voronoi-Delaunay concepts and come with theoretical guarantees under sampling conditions. Surveys of these algorithms can be found in [CG06, Dey06]. Among these methods there are *Tangent plane methods*: [Boi84, GK02, CSD04, KY05, BG14], *Inside/Outside labelling methods*: Power Crust [ACK01], Natural neighbors [BC02], Wrap [Ede03, RS07], Convection [ACA07], Delaunay refinement [BO05] and *Restricted Delaunay based methods*.

More precisely, our algorithm belongs to the category of the *Restricted Delaunay based surface reconstruction methods*. The main idea behind these methods is to filter out a subset of the Delaunay triangulation $Del(P)$ of the sampling P, by restricting it to some subset of $\mathbb{R}^3$ (*thickening* of the point set) which is a good approximation of the unknown surface S and can be efficiently computed from P.

This idea is motivated by the fact that, if the input meets prescribed sampling assumptions on its quality (ε-sampling), it has been proven that the output mesh is homeomorphic to S (see the theorems in [ES94, AB98]). The general ε-sampling framework proposed by Amenta Bern [AB98] defines a real-valued function f, called the *local feature size* on a smooth surface S (as the distance to the medial axis of S) and, essentially, requires that the sample set P leaves no ball centered at a surface point p and of radius $\varepsilon \times f(p)$ unsampled.

Alpha shapes are one of the first attempt to use this idea for surface reconstruction, [EM94]. These methods define the dilation of the pointset as the union of balls of radius α: $\overline{P} = \cup B_\alpha(p_i)$. Then they build the α-complex which is the restriction of the Delaunay triangulation of P to the union of α-balls : $Del(P)|\cup B_\alpha(p_i)$.

The Ball Pivoting algorithms [BMR*99, DADSG11] are optimized variants of Alpha Shapes where the computation of the 3D Delaunay triangulation is avoided. They generate the surface incrementally, starting from a single triangle and adding new triangles to the surface by pivoting an α-ball. Since all the computations are local, these algorithms exhibit linear complexity and are efficiently

parallerized to handle large data sets [Dig14]. The method introduced by Digne et al. [DMSL11] improves the noise-resilience of the BPA by applying it to a scale-space version of the point set.

The Cocone algorithms, fist designed by Amenta and Bern [ACDL00], build on the same idea. Here, the *thickening* of the pointset is defined as the union of *co-cones* placed at each point (double cones with opening angle of $\pi/8$ with respect to the estimated normal). For the Cocone algorithm the theoretical guarantees come with a sampling density at least $0.18 \times lfs(x)$ and for $\theta = \pi/8$. Intuitively, the Cocone angle θ absorbs the deviation between the estimated normal vector and the true normal of the surface.

One of the main limitations of the Cocone algorithm is its computation cost. For large datasets, the data structures employed exceed the system memory and processing speed becomes very slow. Even if efficient and robust implementations of the 3D Delaunay triangulation are used [CGAL, GEOGRAM], these algorithms do not scale up well and cannot accommodate large data sets comprising dozens of million of points within a reasonable space and time. Moreover, even on well sampled smooth surfaces, the worst-case complexity of the 3D Delauany triangulation can be quadric $O(n^2 \log n)$, where $n = \|P\|$ is the size of the sampling [Eri01].

Local Cocone variants were developed to mitigate this problem. These algorithms are local in sense that the restricted Voronoi cells of each point are computed using a small neighbourhood around the point. Our method belongs to this category and in the following we detail the state of the art of these algorithms.

Funke and Ramos [FR02] demonstrated that the Cocone algorithm can be modified so that no global computation of the 3D Delaunay triangulation of the entire pointset is required. They introduced an additional *local uniformity* assumption on the sampling which requires a minimum separation between the sample points (there is some $\delta \in \{0,1\}$ such that any $p,q \in S$ must be separated y a distance of at least $\delta\varepsilon lfs(p)$). This involves decimating the input pointset so that the working point set is a locally uniform subset of the original input. Within this theoretical framework they showed that the cocone triangles can be computed locally. Indeed, the resulting restricted cocone of each point p is a local object in a sense that it is completely determined by nearby samples, when the sampling conditions are met. When the algorithm expands the calculation of the 3D Voronoi cell for each point p, it stops when the distance between p and the new samples exceeds $3 \times d_{max}$, where the maximum distance of a point on a Voronoi edge of V_p within C_p. Thus, the algorithm makes sure that the cocone region of each 3D Voronoi cell (with respect to the estimated normal) is computed exactly since further nearest neighbours are added until it is sure that the cocone region is not affected any more. The complexity of the resulting theoretical algorithm is almost linear $O(n \log n)$, but it has not been implemented.

Dumitriu et al. [DFKM08] used this locally uniform ε-sampling framework to design a surface reconstruction algorithm with correctness guarantees. Although the algorithm is quite complicated, Dumitriu et al. have produced an implementation [DFKM10]. Amenta and Kil [KA08] employed similar techniques to design a very simple and efficient reconstruction algorithm which processes the sample points in parallel on the GPU. However, all these algorithms require a noise free *locally uniform* ε-sampling which is not reasonable to expect from raw pointsets and the problem of building such sampling from an arbitrary input pointset is still open.

Alternatively, Dey et al. [DDW11] developed an octree-based version of the Cocone algorithm such that the 3D Delaunay triangulation is only computed on small clusters of the total point set. They relaxed the local uniformity criterion and adopted the assumption that the local sampling density is bounded by a constant, [ABL03]. Under this assumption, the Localized Cocone maintains the theoretical guarantees of the original Cocone algorithm.

The Localized Cocone has been implemented using the CGAL library, to facilitate the computation of the 3D Delaunay triangulation on each octree leaf. The reported results show that it reconstructs inputs of 5 million points in more than 1 hour and uses more than 2000MB of memory. Our algorithm described below takes less than a minute on comparable point sets. In addition, in the localized cocone algorithm, when the input pointset does not meet the stringent sampling requirements, the manifold extraction step is not particularly robust. If the sampling density is insufficient to ensure a full umbrella of cocone triangles around a vertex, then in the subsequent trimming stage of the algorithm, *all* cocone triangles are removed. In practice, it is necessary to introduce heuristics to prevent catastrophic failure to accommodate real sample sets. But in this case, there is no longer a guarantee that the output will be topologically correct.

3. The SSSR algorithm

Our algorithm is strongly related to the theoretical algorithm of Funke and Ramos [FR02]. In particular, it uses similar ideas as in the Localized Cocone algorithm [DDW11] and the Vorpaline algorithm [LB12] but it differs from previous Cocone algorithms in two main respects:

- Our algorithm computes the Delaunay triangulation of the pointset restricted to a set of disks. Instead of computing the Delaunay triangulation *then* restricting it, we *directly* compute the restricted Voronoi cells (intersections between the Voronoi cells and the disks), and deduce the restricted Delaunay triangles from the combinatorial information of the restricted Voronoi vertices (see Section 3.4, Figure 3, Figure 5 below), this results in an algorithm that is simple and "embarassingly parallel".
- We also present a new manifold surface extraction method which avoids the trimming step (the most critical because it can end up with an empty mesh). Starting from a subset of candidate triangles which forms a clean and orientable manifold with boundaries, we add the remaining triangles, one by one, only if they do not break the topological properties of the initial mesh. The algorithm is guaranteed to terminate with a non-empty manifold mesh.

The input is an unorganized 3D point set, possibly with normal attributes (unoriented or oriented). The output of the algorithm is a triangular mesh connecting the input points which approximates the surface. Specifically, the output triangulation is a compact 2-pseudo-manifold (see Section 3.7), it is orientable (no Moebius

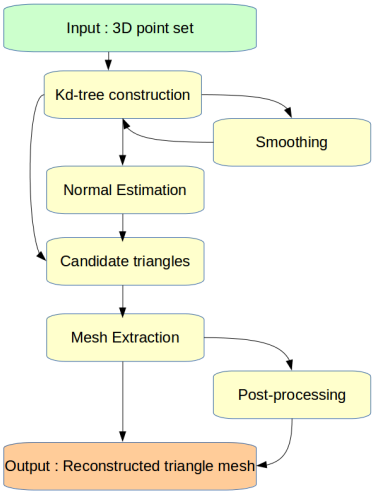


Figure 2: *Overview of the algorithm.*

strip) and may contain several connected components and boundaries.

The algorithm uses two pre-processing steps which allows us to accommodate raw data sets. First, it uses a smoothing algorithm to reduce the noise in the input data. Second, if input data does not come with normals, it uses a normal estimation algorithm. Note, however, that this algorithm does not need to consistently orient the normals.

The general workflow of our algorithm in summarised in Figure 2. We give details of each of these steps in the following sections.

3.1. Kd-tree construction

During this initial step the data is loaded and prepared for the remaining operations. The vertices P are organized in a geometric search data structure, such that for any p_i we can efficiently compute the list of nearest neighbours p_j sorted by increasing distance to p_i, [Sam05].

To do this step, one can use the *Approximate Nearest Neighbours (ANN)* library [MA97] which with ε set to 0 computes the exact nearest neighbours for each point. We have developed our own version of the same algorithm that only uses contiguous arrays in memory. To save one integer per vertex, we compute a *balanced* binary tree. As a consequence, the links are completely implicit in the tree.

3.2. Smoothing

Since our algorithm interpolates the input points, it requires a clean and accurately registered point set. Thus, if the input set is noisy such as raw scanning data, it has to be smoothed before the reconstruction.

Our current implementation uses projection onto the best approximating plane of the k-nearest neighbours, [HDD*92]. Specifically, for a point p and its k nearest neighbours $\{p_i\}_{i=1}^{k}$, we find the fitting plane $n^T x = c$ for p by minimizing the error term $e(n,c) = \sum_{i=1}^{k}(n^T p_i - c)^2$ under the constraint $n^T n = 1$. Then, the point p is projected onto this plane. We compute the best fit plane for each point in parallel. Once all the points are projected, we update the kd-tree.

Parameters: This step uses two parameters : the number of nearest neighbours *nb_neighbors* (by default 30) and the number of smoothing iterations *smooth_iter* (by default 0). We have used the default values for the small data sets shown in the results, while, for the large ones, we have done two iterations

3.3. Normal estimation

If the input points come without normals, our algorithm estimates them locally with the best fitting plane following exactly the same formulation as in the smoothing step, [HDD*92].

Notice that the algorithm dose not need to orient the normals since we only need an approximation of the (unoriented) normal direction at each point which is used to create the tangent disk, in the next step of the algorithm.

Parameters: This step uses one parameter: the number of nearest neighbours *nb_neighbors* set by default to 30.

3.4. Candidate triangles

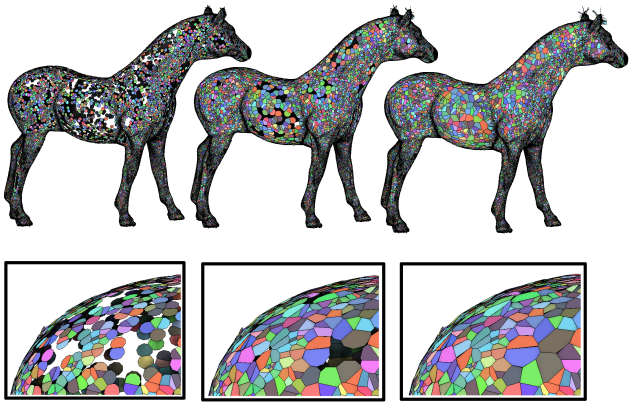


Figure 3: *We compute the Voronoi diagram of the input pointset restricted to a union of disks centered on the points and orthogonal to the estimated normals. From left to right: the radius of the disks is increased from 0.2% to 4% of the bounding box diagonal. All the restricted Voronoi cells (colored polygons) are computed in parallel. Whenever three restricted Voronoi cells meet, the three corresponding input points will be connected with a triangle.*

Given a pointset $\{p_i\}_{i=1}^{n}$ with estimated normals $\{n_i\}_{i=1}^{n}$ and a radius r, our algorithm computes the intersection between the Voronoi cells of the points $Vor(p_i)$ and the disks D_i^r of radius r centered on the points and orthogonal to the normals.

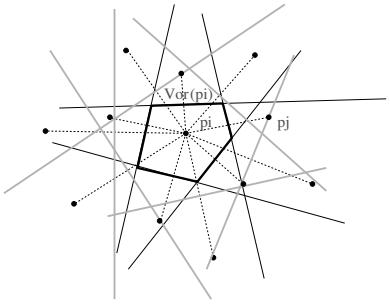


Figure 4: *The Voronoi cell $Vor(p_i)$ is the result of clipping the entire space by all the bisectors (continuous lines) of the segments $[p_i, p_j]$ (dashed lines). Among the bisectors, some are contributing (black) and some are non-contributing (gray).*

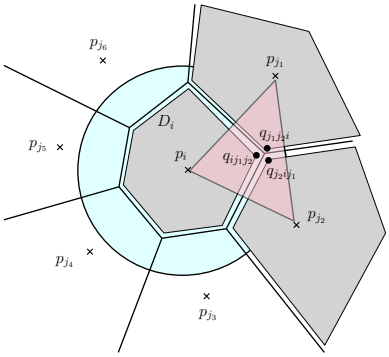


Figure 5: *Clipping a disk D_i^r (in blue) by the bisectors of the neighboring points p_{j_i}. The so-defined restricted Voronoi vertex $q_{ij_1j_2}$ and the associated restricted Delaunay triangle (p_i, p_{j_1}, p_{j_2}) are highlighted.*

During this step, we fist build a disk D_i^r at each point $p_i \in P$ orthogonal to n_i and with user-given radius r . In our implementation, we use a polygon (with 10 vertices) that approximates the disk. In contrast with previous algorithms, we *explicitly* build the *restriction* of the Voronoi cell $Vor(p_i)$ to the tangent disk D_i^r at p, $Vor(p_i) \cap D_i^r$, as shown on Fig.3 and Fig.5. Thus we collect a set of *restricted* Voronoi vertices.

The set T of *candidate triangles* is then defined by the triangles which are dual to the restricted Voronoi vertices (more on this below).

The basic operation in this step of the algorithm consists of clipping the disk D_i^r with the Voronoi cell $Vor(p_i)$ of each point p_i. Here we use ideas that are similar to those presented in the Localized Cocone algorithm [DDW11] which allows us to build locally the candidates triangles by using only the k-nearest neighbours. More precisely, we use the clipping algorithm and the local characterization of the restricted Voronoi cells introduced in [LB12] which we recall in the following in order to make the explanation self-contained.

3.5. Clipping a disk

Let us recall that a Voronoi cell is a convex polytope, that can be defined as the intersection of halfspaces : $Vor(p_i) = \bigcap_{j \neq i} \Pi^+(i,j)$, where $\Pi^+(i,j)$ denotes the half-space bounded by the bisector of (p_i, p_j) that contains p_i. Note that the bisectors between p_i and all the other points are involved in the definition above, whereas only a small subset of them corresponds to actual Voronoi edges (see Figure 4). As a consequence, we can classify the bisectors $\Pi^+(i,j)$ into two sets : *non-contributing* if $Vor(p_i) \subset \Pi^+(i,j)$ and *contributing* otherwise. In other words, clipping a Voronoi cell by a non-contributing bisector does not change the result. Therefore, the Voronoi cell can be computed by intersecting a superset of the contributing bisectors.

Let $p_{j_1}, p_{j_2}, \ldots p_{j_{n-1}}$ denote the vertices sorted by increasing distance from p_i. Let V_k denote the intersection of the k first halfspaces and let R_k denote its p_i-centered radius :

$$\begin{aligned} V_k(p_i) &= \bigcap_{l=1}^{k} \Pi^+(i, j_l) \\ R_k &= \max\{d(p_i, x) | x \in V_k(p_i)\}. \end{aligned}$$

We can then make the following simple observation: For all j such that $d(p_i, p_j) > 2R_k$, the bisector $\Pi^+(i,j)$ is non-contributing, i.e. $V_k(p_i) \subset \Pi^+(i,j)$.

Consider $p \in V_k(p_i)$ and p_j such that $d(p_i, p_j) > 2R_k$. By definition of R_k, $d(p, p_i) < R_k$. We have $d(p_i, p) + d(p, p_j) \geq (p_i, p_j)$ (triangular inequality) and $d(p_i, p_j) > 2R_k$, therefore $d(p, p_j) > R_k > d(p, p_i)$ and $p \in \Pi^+(i,j)$ ■

As a direct consequence of the previous observation: $d(p_i, p_{j_{k+1}}) > 2R_k \Rightarrow V_k = Vor(p_i)$.

We call *radius of security* the first value of R_k that satisfies this condition. Note that this observation does not have a practical value in the case of the (unrestricted) Voronoi diagram, since some cells are unbounded and have infinite R_k therefore, for an infinite cell, all the bisectors are considered, leading to prohibitive computation time. However, the observation can be clearly used to compute the restricted Voronoi cells $Vor(p_i) \cap D_i^r$, since they are finite (they are contained by D_i^r).

We use Algorithm 1 to build the restricted Voronoi cell for each point, which is the intersection between the tangent disk and the Voronoi cell of the point.

Note that each restricted Voronoi vertex q is the intersection between the disk D_i^r and the bisectors Π_{i,j_1} and Π_{i,j_2}. When we build the clipped disk, we memorize the indices j_1 and j_2 which generate every restricted Voronoi vertex. This allows us to easily output the corresponding dual triangle i, j_1, j_2, see Fig. 5.

3.6. Set of candidate triangles

Once we have computed the restricted Voronoi cell $D_i^r \cap Vor(p_i)$ for each point (p_i), we are able to find all the triangles i, j_1, j_2 using the stored indices. We call T the set of *candidate triangles*, that are dual

Algorithm 1 *Computes a Restricted Voronoi Cell, i.e. the intersection between a disk and the Voronoi cell of a point.*

Data: the index i of the point x_i, the disk D_r centred on it, the number of neighbours n and the set of points P
Result: Restricted Voronoi Facet at p_i: $Vor(p_i) \cap D_i^r$

$V \leftarrow D_i^r$
$R_k \leftarrow \max\{d(p,p_i) | p \in V\}$
$k \leftarrow 1$
while $d(x_i,x_{j_k}) < 2R_k f$ *and* $k < n$ **do**
  $V \leftarrow V \cap \Pi^+(i,j_k)$
  $R_k \leftarrow \max\{d(p,p_i) | p \in V\}$
  $k \leftarrow k+1$
end

to every restricted vertex. From this set T of candidate triangles, that may contain a non-manifold configuration, we extract a clean surface, in the next steps of the algorithm. Let us further analyse the structure of T:

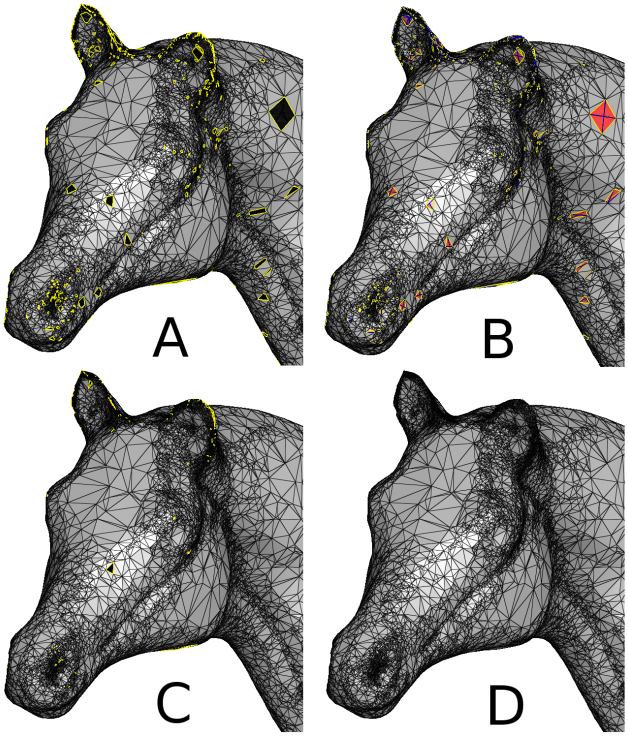


Figure 6: *A: the set of triangles seen from three restricted Voronoi cells (T_3), with some holes (yellow). B: the set of triangles seen from one or two Voronoi cells ($T_{1,2}$, in red) fills some holes, but have non-manifold configurations (e.g., the "sliver" formed by the four red triangles in the right). C: result after carefully inserting the $T_{1,2}$ triangles one by one in a way that preserve manifold topology. D: the remaining holes are suppressed by a post-processing step.*

The set T of candidate triangles is composed of the restricted Delaunay triangles T_3 plus some other triangles, as explained below.

The set T_3 of restricted Delaunay triangles is defined as:

$$T_3 = \{(i,j,k) \mid (Vor(p_i) \cap D_i^r) \cap (Vor(p_j) \cap D_j^r) \cap (Vor(p_k) \cap D_k^r) \neq \emptyset\}. \quad (1)$$

In other words, they correspond to triples of restricted Voronoi cells (i,j,k) that are mutually in contact (see Figures 5 and 3). These triangles can be easily found, by generating all the index triples that correspond to the restricted Voronoi vertices, sorting them (using for instance the lexicographic order) and keeping the triples that appear three times in the sorted sequence. The so-computed T_3 triangles are displayed in Figure 6-A. Note that in the sorted sequence, there are also triples that appear once or twice. The corresponding set of triangles is referred to as $T_{1,2}$ and is defined by:

$$T_{12} = \{(i,j,k) \mid D_i^r \cap Vor(j) \cap Vor(k) \neq \emptyset\} - T_3. \quad (2)$$

In other words, a triangle (i,j,k) is in T_{12} whenever the restricted Voronoi cell of p_i "sees" p_j and p_k but not both conversely. A 2d example is shown in Figure 7(a): in the Voronoi cell of p_j, D_j^r (symbolized as a thick line) touches a side that is adjacent to a cell $Vor(p_k)$ that is different from $Vor(p_i)$. Other examples are shown in Figure 7(b), that correspond to nearly co-spherical configurations. The so-defined T_{12} triangles are displayed in red in Figure 6-B. As expected, they are not as "reliable" as the T_3 triangles, in the sense that they are likely to generate non-manifold configurations (see for instance the four interconnected red triangles that form a "sliver" in the top right part of the neck in Figure 6-B). However, one can observe that they contain interesting information, and could be used to fill the holes in the T_3 triangles (Figure 6-A).

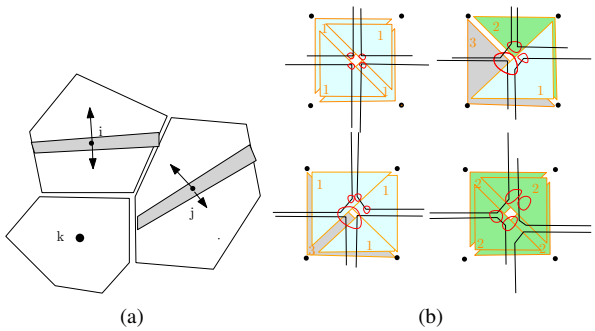


Figure 7: *(a) Configuration resulting into a T_{12} triangle . (b) Configurations of 4 co-circular where T_{12} triangles appear.*

Thus, the idea is to start from the T_3 triangles, then carefully insert the triangles from the T_{12} set, one triangle at a time, making sure that the reconstructed mesh remains manifold with the additional triangle inserted. If it is not the case, the triangle is rejected. The algorithm that implements this idea is detailed in the next section.

Parameters: This step uses one parameter : the disk *radius* (in % of the bounding box diagonal). Note that this is not a "real" parameter as it is the case for any Ball Pivoting algorithm, for example.

Indeed, when the radius r is sufficiently large (i.e. larger than the width of the Voronoi cells), the disks D^r touch all the sides of their Voronoi cells (like on the bottom-right of Figure 3), and increasing the radius no longer changes the result. Thus, the radius can always be set to some large value. In our experimental results, we used r=5% of the bounding box diagonal and we decrease the size in order to accelerate the clipping for the very large data sets (r=0.5% bbox diagonal when number of vertices > 10M).

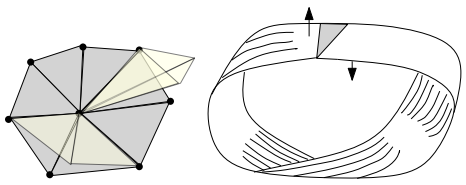
3.7. Manifold mesh extraction

The original "prunning and walking" approach [ACDL00] consists of first deleting all triangles incident to sharps edges and then extracting the outer boundary of the set of remaining triangles by a depth-first walk along each of its connected components. As mentioned earlier, this algorithm is not particularly robust and uses many heuristics for the practical implementation and may end up with an empty mesh.

Here we take a different strategy which allows us to avoid this problem. We first build a clean orientable 2-manifold mesh with a subset of the candidate triangles. Then we fill the holes iteratively, with the remaining triangles, if they do not jeopardize the topological properties of the output.

More precisely, the mesh structure that we build incrementally is a manifold mesh where each edge can only be adjacent to two facets, edges with only one incident triangle are allowed, as well as triangles adjacent only by one vertex (this is a non-manifold vertex whose neighbourhood is not a topological 2-sphere). The output mesh (also known as a 2-pseudo-manifold [Mun84]) is orientable and can have several connected components.

We initialize the output mesh with all T_3 triangles. Then we remove the triangles which exhibit non-manifold edges and *non-manifold vertices by excess*. Recall that a non-manifold edge is incident to more than 2 triangles, while a 'by-excess' non-manifold vertex has a closed loop of triangles (clean umbrella) in its neighbours plus additional triangles (see Fig. 8(a)) We also check that there are no Moebius strips within the initial mesh (see Fig. 8(b)). Note, that the initial mesh can however exhibit triangles incident to each other only by a vertex.



(a) (b)

Figure 8: *(a) Combinatorial test III. A non-manifold vertex 'by excess'. (b) Combinatorial test IV. Moebius band.*

We then fill the holes of the initial mesh with the remaining triangles from T_{12}. The T_{12} triangles are tested against the following criteria, from the simplest to the most time consuming. Whenever a criterion is not satisfied, the considered triangle is rejected. Only the triangles that pass all the test are inserted in the final surface.

- **Geometric test**: This criterion ensures that the normals of the neighbouring triangles agree, which means that they do not make a sharp edge (less than $\pi/4$)
- **Combinatorial test I**: This connectivity test ensures that the triangle is properly connected to the current mesh. Every new triangle should be either incident to two edges of existing triangles, or to one edge of an existing triangles and one isolated point.
- **Combinatorial test II**: This test checks for non-manifold edges and tests whether the three candidate edges are manifold.
- **Combinatorial test III**: This criterion checks that inserting the new triangle do not generate 'by-excess' non-manifold vertices.
- **Combinatorial test IV**: This global orientability test checks if in the neighbourhood of the triangle the same connected component appears with two opposite orientations, then connecting the triangle would create a Moebius strip. The triangle is rejected if it is incident to the same connected component with two different orientations.

In the following, we give the overview of the manifold extraction algorithm :

Algorithm 2 *Extracting the manifold mesh.*

Data: A set of candidate triangles T
Result: An orientable 2-manifold mesh built from T

(1) Compute the subsets $T3$ and $T12$ of triangles
(2) Initialize the mesh M with the subset $T3$
(3) Remove from M :
- triangles incident to non-manifold edges (Test II)
- triangles which are incident to *non-manifold vertices by excess* (Test III)
- triangles inducing orientability problems (Moebius strip) (Test IV)

(4) Tentatively add into M each triangle t from $T12$ if it successfully passes all following tests:
- Geom. Test : the normals to t and its neighbour should not point to opposite directions
- Test I : t is incident to at least two edges of existing triangles, - Test II : t does not create non-manifold edges - Test III : t does not create *non-manifold vertex by excess*
- Test IV : t maintains the global orientability of the mesh

The output of this algorithm is an orientable triangular mesh which has no non-manifold edges but can have non-manifold vertices, such as a pinched torus. It possibly has several connected components and boundaries.

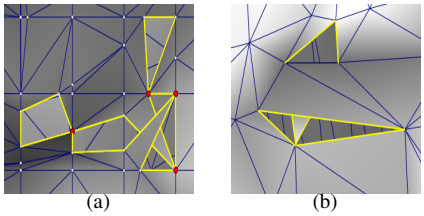
Parameters: This step uses one parameter which is the maximum deviation angle between the triangles normals *max_N_angle*. We always use the default value set to 60 degrees.

3.8. Post-processing

At this point, after the manifold extraction test, the obtained mesh may still contain a number of isolated holes, caused mainly by insufficiently sampled sharp features (see e.g. the top of the ears of the horse in Figure 12 C). To eliminate these, we add an extra post-processing step, that fills some of the holes.

The algorithm takes an user-defined *maximum hole size* parameter, in terms of number of edges. Then it looks for *simply connected holes* and fills them using a classical loop-split algorithm [Lie03]. This algorithm splits the hole recursively until it gets holes composed of exactly 3 edges and fills them with the corresponding triangle.

In order to ensure that the holes are simple loops, we first search and remove the configurations where the holes contain "bridges"(see Fig.9). We call bridge triangles those which are visited more than one time when we walk around the border edges of a hole. Note that this definition dose not capture bridges wider than two triangles but we did not encounter any, in practice.



(a) (b)

Figure 9: *(a) Simple bridge. (b) Double bridge.*

Parameters: This step uses two of the following four parameters which give the size of the holes to be filled and the connected components to be removed. In our experiments, we have always used the default values:

- *max_hole_area* (=5%) : Fill holes smaller than (in % total area)
- *max_hole_edges* (=500) : Fill holes with a smaller number of edges
- *min_comp_area* (=0.01%) : Remove small components (in % total area)
- *min_comp_facets* (=10) : Remove small components (in facet number)

4. Experimental results and Discussion

Our implementation is available in C++, in the supplemental material, together with datasets and comparison data for ScaleSpace reconstruction [Dig15], screened Poisson [KH13] and SuperCocone [DDW11]. The implementation is straightforward. A naive implementation fits within 200 lines of code (100 for the Kd-Tree, 50 for the polygon clipping routine and 50 for putting everything together). Our parallel implementation uses 2000 lines (the additional lines are for manifold extraction, book-keeping, handling several options, and for the parallel implementation).

We tested our algorithm on a database of examples available from the world-wide-web [AIM, Sta, Far, EE] and compared the results with the three state-of-the-art methods cited above. Our database of examples is decomposed into two subsets SMALL (up to 1 Million of points) and LARGE (from 1M up to 36M).

Table 1 lists the models and reports the timings for the four algorithms. We used implementations written by the authors of the cited methods, and the same computer for all the tests [Processor: Intel Core i7-4900MQ CPU @ 2.80GHz, RAM:16Gb, Disk: SSD LITEONIT LCS-256M6S 2.5 7mm 256GB, OS:Linux 4.4.0-1-amd64 - Debian distribution]. Our algorithm takes 5 minutes to reconstruct a surface from 36.2 million points.

Model	Size #M vert.	SSSR sec.	SCocone sec.	ScSpace sec.	ScrPoisson sec.
hotdog	0.001	0.07	0.22	10.86	1.78
sphere	0.03	0.24	2.16	1.54	7.77
horse	0.1	1.17	13.96	KO (3004)	18.82
daratech	0.24	2.07	34.79	37.43	28.23
anchor	0.26	2.23	34.57	crash	48.73
dame	0.31	2.88	44.91	39.98	46.00
hand	0.32	2.59	52.01	KO (922)	59.50
lord quas	0.35	3.42	54.31	45.99	32.14
bunny	0.36	4.86	50.23	29.31	48.95
dc	0.46	4.23	71.52	32.41	49.14
MagaliHand	0.7	8	109.7	KO (591)	48.18
eagle	0.79	10.72	113.4	127	118.8
Nasa	0.88	9.89	117.5	105.01	105.27
Cube	1.01	14.51	139.78	161.81	134.04
PerfumeBottle	1.25	13.27	172.18	187.74	122.39
Galaad	1.45	17.19	210.20	368.75	79.72
ToyTurtle	1.47	16.34	207.62	303.23	100.19
SophieHand	1.58	19.29	217.27	334.43	107.54
chineseDragon	1.76	24.63	250.39	KO (899)	149.65
PooranHand	2.10	21.76	312.98	480.45	94.42
PierreHand	2.63	27.40	379.66	643.41	115.94
TrungHand	2.83	42.19	408.76	699.65	132.13
HeleneHand	3.65	38.60	543.62	736.48	119.91
cutanubis	5	45.50	crash	KO (2522)	84.01
anubis	9.99	88.50	1806.62	2153.13	170.18
night	11.05	110.19	1805.97	crash	567.25
lucy	14.02	131.76	2307.91	1483.31	267.42
tanagra	16.38	173.27	2673.32	2301.94	274.55
facadeValGrace	29.46	340.66	KO (4730)	crash	856.70
chateauRives	32.75	360.56	crash	crash	KO (543)
rosetta	36.2	352.88	crash	crash	675.81

Table 1: *Comparison of processing times for the 4 methods. Timings are in seconds and model sizes in million vertices. We used implementations written by the authors of the cited methods, and the same computer for all the tests [Intel Core i7-4900MQ CPU @ 2.80GHz, 16GB]. For the three biggest models, ScaleSpace crashed on all of them, SuperCocone ran out of memory on some models, and Poisson failed on some models (probably due to incoherent normals). KO means that the result has too many holes or is mostly unrelated with the data or the computation time explodes as compared to data of similar size.*

The implementation is very simple, with no dependencies, and thus can be easily ported to any architecture. We ported it to an Android phone [HTC M7, Processeur: ARMv7 NeonFPU 4 cores, RAM: 2Gb, Android 4.4 (kitkat)]. The algorithm exploits the 4 cores of the phone. It is approximately 10 times slower than on the PC. It took 1.5 seconds to reconstruct 25K points, 7.5 sec. for 70K points, 14 sec. for 200K points and 55 sec. for 900K points.

The timing breakdown of the different phases of our algorithm, computed on the entire database are summarized in the charts in Figure 10 and size-time curves for all methods and datasets are shown in Figure 11.

We also show some images with example results from our dataset. To be able to analyse and compare the behaviour of the algorithm, we deactivated the post-processing step of our algorithm, so that what we compare with the state of the art in the images is the output of the manifold extraction of our algorithm. Figure 12 shows a horse sampled with 100000 points with a varying sampling density. ScaleSpace has difficulties finding the correct scale

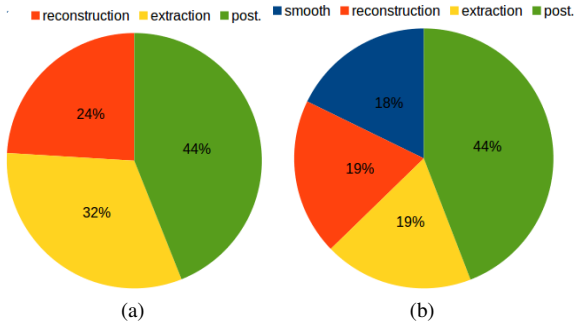


Figure 10: *Timing breakdown of the different steps of our algorithm. (a) Small data. (b) Large data with two iterations of smoothing.*

Figure 11: *Performance of the tested algorithms. Up, timings obtained on the Small data. Bottom, timing obtained on the Large data.*

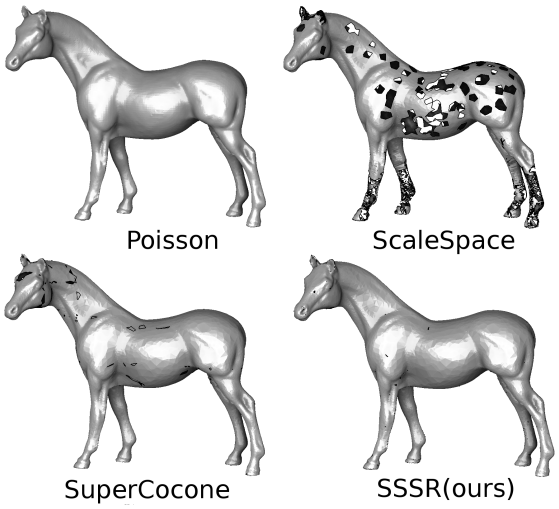


Figure 12: *Comparison of the algorithms with a small dataset (100K vertices) that has high variations of sampling density.*

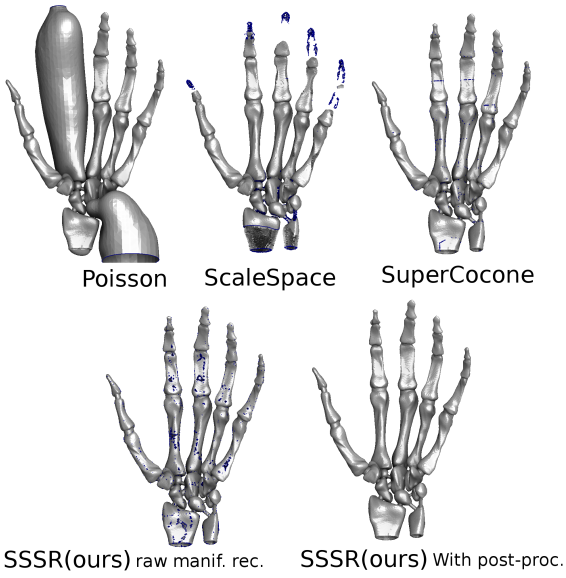


Figure 13: *Poisson needs consistent normal orientations. ScaleSpace had difficulties estimating the scale, and lost features (finger tips).*

in this dataset, and takes more than 3000 seconds. The result has holes when the triangles are larger than the estimated scale. SuperCocone divides the input points into several subsets using an octree. It sometimes fails to recover triangles that cross octree cells boundaries. This behaviour of ScaleSpace and SuperCocone is also observed on the hand dataset in Figure 13. In this figure, one can observe that Poisson depends on a correct orientation of the normals.

In Figure 14, we show the behaviour of the algorithms for a noisy pointset(original registered pointset data for the Stanford bunny). For this data, Poisson both cancels the noise and recovers fine details. ScaleSpace manages to extract a manifold mesh that exactly follows the details of the data, even when it is very bumpy. Our algorithm outputs a result that is very similar to that of SuperCocone. The same dataset with one iteration of smoothing is shown in Figure 15. All the algorithms give a similar result. Note the aligned missing triangles in SuperCocone at boundaries between octree cells.

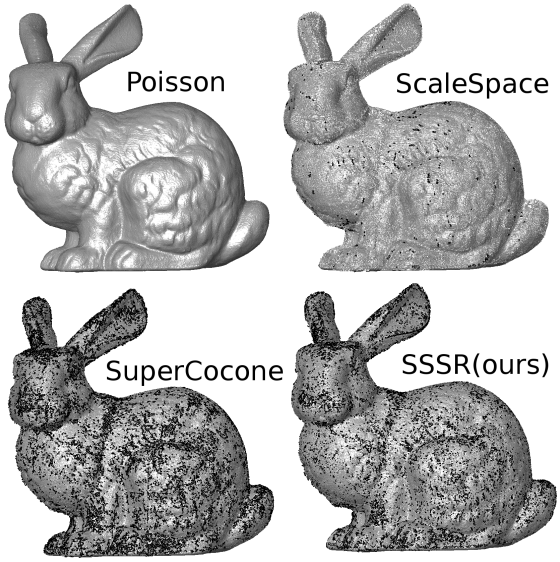


Figure 14: *A noisy dataset (original scanner data of the Stanford Bunny). Poisson both filters-out the noise and recovers the fine features. ScaleSpace successfully extracts a continuous manifold mesh. SuperCocone and our algorithm recover a smaller number of triangles.*

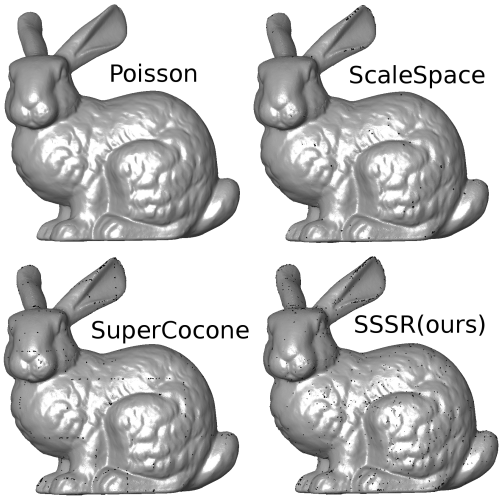


Figure 15: *The same dataset as in Figure 14 with one iteration of smoothing applied. The methods give similar results.*

Figure 16 shows the results obtained from another scanner dataset. Similar behaviour is observed. On this dataset too, ScaleSpace had difficulties estimating the right scale, resulting in some missing triangles (but processing time, 300s, remains reasonable as compared to the irregularly horse dataset). Our algorithm computes the result in 27 seconds.

Figure 17 compares the result on a clean pointset that has bumpy features (the feathers of the eagle). Poisson faithfully reconstructs

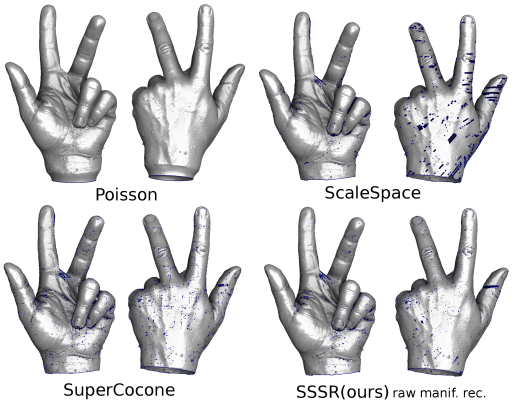


Figure 16: *Experimental results with a scanned pointset of intermediary size (2.6 M vertices). Poisson performs well on this data (it takes 45 seconds) ScaleSpace encounters some difficulties, probably due to the structured patterns and varying sampling, and misses some triangles (647 seconds). SuperCocone (379 s.) and our method (27 s.) give a similar result. Note the aligned missing triangles in SuperCocone result, at the junction between octree cells.*

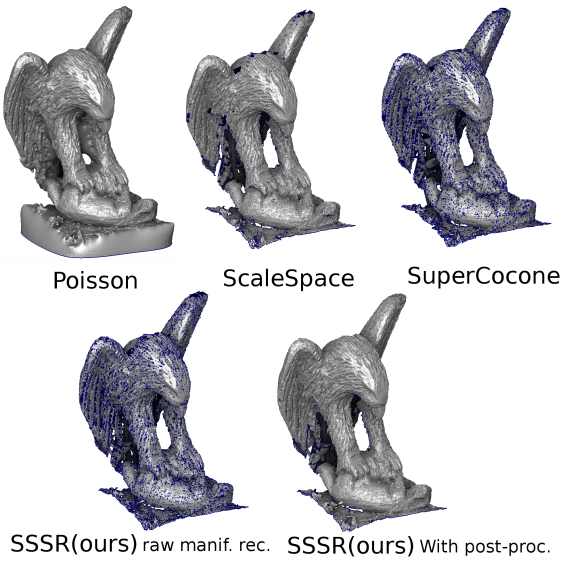


Figure 17: *This dataset is clean (no much noise) but has small-scale bumpy features. The behaviour of the methods is similar to Figure 14: Poisson creates a detailed isosurface, and ScaleSpace manages to extract a manifold mesh. SuperCocone and our method create a smaller number of triangles. However, in our result, missing triangles form small holes, easily filled by our post-processing.*

the features. ScaleSpace reconstructs a continuous manifold mesh that also follows the bumpy features. SuperCocone and our algorithm give a similar result, with more missing triangles than ScaleSpace. However, the missing triangles are mostly small holes that can be removed by our post-processing step.

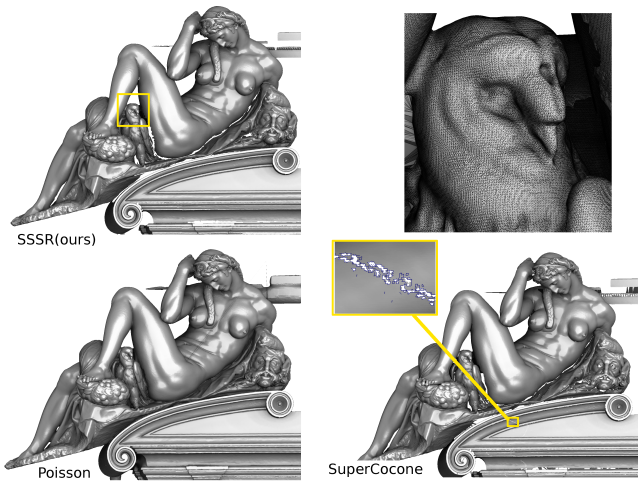


Figure 18: *A dataset with 11 million vertices, extracted from a reconstruction (courtesy of the Digital Michelangelo project). Since it is not raw scanned data, the only difficulty is the number of points. Poisson creates a nice isosurface, that both respects the details and fills-in the holes (time: 230 seconds). SuperCocone gives a good result. It misses some triangles, probably in junctions between octree cells (time: 1806 seconds). Our algorithm gives a result that is similar to the output of SuperCocone in 110 seconds. ScaleSpace crashed on this dataset.*

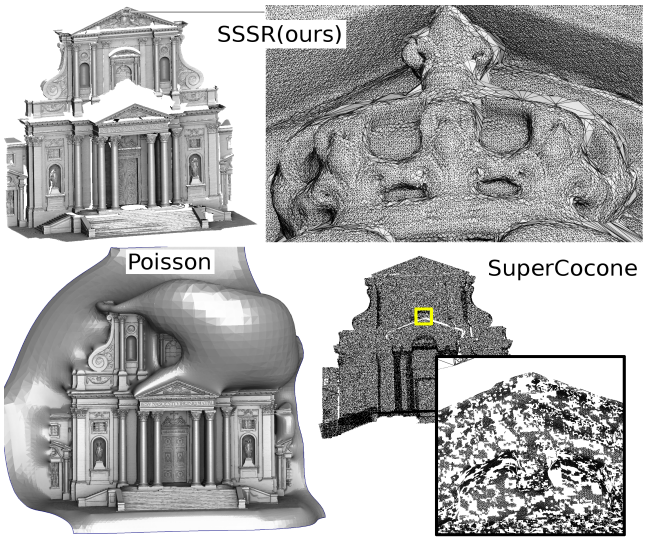


Figure 19: *A scanned pointset with 30 million vertices. On this dataset, some normals were not coherent, causing some bubbles in Poisson reconstruction (time: 856 seconds). SuperCocone gave a result in 5011 seconds, but has many missing triangles (maybe it starts reaching memory limits). Our algorithm took 340 seconds.*

The 'night' dataset in Figure 18 is a set of vertices from an already reconstructed surface, therefore it has no noise, and the only challenge is its size. Poisson has no problem scaling-up, since its representation (implicit function on an octree) is decoupled from the data. Moreover, it very nicely fills all the holes in the input data. SuperCocone scales up, but still misses some triangles at junctions between octree cells. Scale Space crashed on this dataset.

Finally, we compare and analyse the result of our method for the three largest datasets, with up to 36 million points. For such datasets, existing Delaunay-based method (ScaleSpace and SuperCocone) start encountering memory limitations. ScaleSpace crashed on all examples. SuperCocone gave an incorrect result on one of them ('facade-val-de-grace') since many triangles were missing and crashed on the others ('chateau-rives' and 'rosetta'). The results are displayed in Figures 19 and Figure 1. A rendering of 'rosetta' is available in the supplemental material.

Results of the 'facade-val-de-grace' are shown in Figure 19. For this dataset, Poisson had some problems with some incoherent normals, causing bubbles to appear. It may not be too difficult to fix, but the need for having coherent normals is a difficulty for using Poisson. SuperCocone missed many triangles, we suspect this comes form their implementation that starts to encounter memory problems.

We also show in Figure 1 on the first page, the 'chateau-rives' example with 36 million points. For this dataset, only our algorithm outputs a valid result.

5. Conclusion

Based on a simple observation about the restricted Voronoi cell, our algorithm has a trivial implementation, is embarassingly parallel and economic in terms of memory resources, therefore it scales up quite well to datasets with tens of millions vertices while keeping computation times smaller than 5 minutes. Its simplicity makes it possible to embed it into devices such as smartphones and cameras with a CPU (see Android ARMV7 timings in Section 4).

Compared to the state of the art, it is not as good as Poisson for filling the holes, and when ScaleSpace can estimate the correct scale, ScaleSpace has a better set of candidate triangles (see e.g. the eagle dataset). Regarding timings, our algorithm is the fastest, by two order of magnitudes in some cases. However, one needs to take into account that the implementation of SuperCocone that we obtained from its authors is sequential, and could probably be easily parallelized. We think that our algorithm will still be faster since it does not need to compute any Delaunay triangulation (it directly computes the Restricted Delaunay Triangulation).

Observing the relative timings of the different steps of our algorithm, sadly, manifold extraction and post-processing clearly dominate the "core" of the algorithm. We think there is room for improvement. Our implementation of hole filling is not well optimized, not parallel, and we think it could probably be made two or three times faster.

We designed this algorithm mainly with practical considerations in mind. A theoretical study would be interesting. For now, the only property that we can guarantee is that what we compute is exactly the Delaunay triangulation of the input points restricted to the union

of the D_i^r disks (see the observation in Section 3.5). In the results, we observed that the behaviour of our algorithm is very similar to the Cocone algorithm. Therefore, it would be interesting to characterize the configurations where the outputs of both algorithms match. We conjecture that it will be some variant of ε-sampling conditions, that we plan to study in future work.

References

[AB98] AMENTA N., BERN M.: Surface reconstruction by voronoi filtering. In *Proc. of the 14th annual symposium on Computational geometry* (NY, USA, 1998), SCG, ACM, pp. 39–48. 2

[ABCO*03] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., T. SILVA C.: Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics 9*, 1 (Jan. 2003), 3–15. 2

[ABL03] ATTALI D., BOISSONNAT J.-D., LIEUTIER A.: Complexity of the delaunay triangulation of points on surfaces: the smooth case. In *Proc. of the 19th annual symposium on Computational geometry* (USA, 2003), SCG, ACM, pp. 201–210. 3

[ACA07] ALLEGRE R., CHAINE R., AKKOUCHE S.: A flexible framework for surface reconstruction from large point sets. *Computers & Graphics 31*, 2 (2007), 190 – 204. 2

[ACDL00] AMENTA N., CHOI S., DEY T. K., LEEKHA N.: A simple algorithm for homeomorphic surface reconstruction. In *Proc. of the 16th SCG* (2000), pp. 213–222. 3, 7

[ACK01] AMENTA N., CHOI S., KOLLURI R. K.: The power crust. In *Proc. of the 6th SMA* (2001), pp. 249–266. 2

[ACSTD07] ALLIEZ P., COHEN-STEINER D., TONG Y., DESBRUN M.: Voronoi-based variational reconstruction of unoriented point sets. In *Proc. of the 5th SGP* (2007), pp. 39–48. 2

[AIM] Aim @ shapes repository. URL: http://visionair.ge.imati.cnr.it/ontologies/shapes/. 8

[BC02] BOISSONNAT J.-D., CAZALS F.: Smooth surface reconstruction via natural neighbour interpolation of distance functions. *Computational Geometry 22*, 1 - 3 (2002), 185 – 203. 2

[BG14] BOISSONNAT J.-D., GHOSH A.: Manifold reconstruction using tangential Delaunay complexes. *Discrete and Computational Geometry 51*, 1 (2014), 221–267. 3

[BLN*13] BERGER M., LEVINE J. A., NONATO L. G., TAUBIN G., SILVA C. T.: A benchmark for surface reconstruction. *ACM Trans. Graph. 32*, 2 (Apr. 2013), 20:1–20:17. 2

[BMR*99] BERNARDINI F., MITTLEMAN J., RUSHMEIER H., SILVA C., TAUBIN G.: The ball-pivoting algorithm for surface reconstruction. *IEEE Trans. on Vis. and Comp. Graphics 5* (1999), 349–359. 2

[BO05] BOISSONNAT J.-D., OUDOT S.: Provably good sampling and meshing of surfaces. *Graph. Models 67*, 5 (Sept. 2005), 405–451. 2

[Boi84] BOISSONNAT J.-D.: Geometric structures for three-dimensional shape representation. *ACM Trans. Graph. 3*, 4 (Oct. 1984), 266–286. 2

[BTS*14] BERGER M., TAGLIASACCHI A., SEVERSKY L., ALLIEZ P., LEVINE J., SHARF A., SILVA C.: State of the Art in Surface Reconstruction from Point Clouds. In *Eurographics 2014* (France, 2014), vol. 1, pp. 161–185. 2

[CBC*01] CARR J. C., BEATSON R. K., CHERRIE J. B., MITCHELL T. J., FRIGHT W. R., MCCALLUM B. C., EVANS T. R.: Reconstruction and representation of 3d objects with radial basis functions. SIGGRAPH '01, pp. 67–76. 2

[CG06] CAZALS F., GIESEN J.: Delaunay triangulation based surface reconstruction: Ideas and algorithms. In *Effective computational geometry for curves and surfaces* (2006), Springer, pp. 231–273. 2

[CL96] CURLESS B., LEVOY M.: A volumetric method for building complex models from range images. SIGGRAPH '96, ACM, pp. 303–312. 2

[CSD04] COHEN-STEINER D., DA F.: A greedy delaunay-based surface reconstruction algorithm. *The Visual Computer 20*, 1 (2004), 4–16. 2

[DADSG11] DI ANGELO L., DI STEFANO P., GIACCARI L.: A new mesh-growing algorithm for fast surface reconstruction. *Comput. Aided Des. 43*, 6 (June 2011), 639–650. 2

[DDW11] DEY T. K., DYER R., WANG L.: Localized cocone surface reconstruction. *Computers and Graphics 35*, 3 (2011), 483–491. SMI Conference 2011. 3, 5, 8

[Dey06] DEY T. K.: *Curve and Surface Reconstruction: Algorithms with Mathematical Analysis*. Cambridge University Press, NY, USA, 2006. 2

[DFKM08] DUMITRIU D., FUNKE S., KUTZ M., MILOSAVLJEVIĆ N.: On the locality of extracting a 2-manifold in r3. vol. 5124 of *Lecture Notes in Computer Science*, Springer, pp. 270–281. 3

[DFKM10] DUMITRIU D., FUNKE S., KUTZ M., MILOSAVLJEVIĆ N.: How much geometry it takes to reconstruct a 2-manifold in r3. *J. Exp. Algorithmics 14* (Jan. 2010), 2:2.2–2:2.17. 3

[Dig14] DIGNE J.: An Analysis and Implementation of a Parallel Ball Pivoting Algorithm. *Image Processing On Line 4* (2014), 149–168. 3

[Dig15] DIGNE J. J.: An Implementation and Parallelization of the Scale-Space Meshing Algorithm. *Image Processing On Line 5* (Nov. 2015), 282–295. 8

[DLS05] DEY T. K., LI G., SUN J.: Normal estimation for point clouds: A comparison study for a voronoi based method. In *Proc. of the Conference on Point-Based Graphics* (2005), pp. 39–46. 2

[DMSL11] DIGNE J., MOREL J.-M., SOUZANI C.-M., LARTIGUE C.: Scale space meshing of raw data point sets. *Computer Graphics Forum 30*, 6 (2011), 1630–1642. 3

[Ede03] EDELSBRUNNER H.: *Discrete and Computational Geometry: The Goodman-Pollack Festschrift*. Springer, Berlin, 2003, ch. Surface Reconstruction by Wrapping Finite Sets in Space, pp. 379–404. 2

[EE] Statue model repository. EPFL Computer Graphics and Geometry Laboratory. URL: http://lgg.epfl.ch/statues_dataset.php. 8

[EM94] EDELSBRUNNER H., MÜCKE E. P.: Three-dimensional alpha shapes. *ACM Trans. Graph. 13*, 1 (Jan. 1994), 43–72. 2

[Eri01] ERICKSON J.: Nice point sets can have nasty delaunay triangulations. In *Proc. of the 7th Annual Symposium on Computational Geometry* (NY, USA, 2001), SCG '01, ACM, pp. 96–105. 3

[ES94] EDELSBRUNNER H., SHAH N. R.: Triangulating topological spaces. In *Proc. of the tenth annual symposium on Computational geometry* (NY, USA, 1994), SCG '94, ACM, pp. 285–292. 2

[Far] Farman institute 3d point sets. URL: http://www.ipol.im/pub/art/2011/dalmm_ps/. 8

[FR02] FUNKE S., RAMOS E. A.: Smooth-surface reconstruction in near-linear time. In *Proceedings of the 13th annual ACM-SIAM symposium on Discrete algorithms* (PA, USA, 2002), SODA '02, Society for Industrial and Applied Mathematics, pp. 781–790. 3

[GK02] GOPI M., KRISHNAN S.: A fast and efficient projection-based approach for surface reconstruction. In *High Performance Computer Graphics, Multimedia and Visu.* (2002), pp. 1–12. 2

[HDD*92] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Surface reconstruction from unorganized points. *SIGGRAPH 26*, 2 (July 1992), 71–78. 2, 4

[KA08] KIL Y. J., AMENTA N.: Gpu-assisted surface reconstruction on locally-uniform samples. In *IMR* (2008), pp. 369–385. 3

[KBH06] KAZHDAN M., BOLITHO M., HOPPE H.: Poisson surface reconstruction. In *Proceedings of the 4th SGP* (2006), pp. 61–70. 2

[KH13] KAZHDAN M., HOPPE H.: Screened poisson surface reconstruction. *ACM Trans. Graph. 32*, 3 (2013), 29:1–29:13. 2, 8

[KY05] KUO C.-C., YAU H.-T.: A delaunay-based region-growing approach to surface reconstruction from unorganized points. *Computer-Aided Design 37*, 8 (2005), 825 – 835. 2

[LB12] LEVY B., BONNEEL N.: Variational Anisotropic Surface Meshing with Voronoi Parallel Linear Enumeration. In *Proc. of the 21st International Meshing Roundtable* (2012). 3, 5

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph. 21*, 4 (Aug. 1987), 163–169. 2

[Lie03] LIEPA P.: Filling holes in meshes. In *Proc. of the 2003 Symposium on Geometry Processing* (Switzerland, 2003), SGP '03, Eurographics Association, pp. 200–205. 8

[MA97] MOUNT D. M., ARYA S.: ANN: A library for approximate nearest neighbor searching. In *CGC Workshop on Computational Geometry* (1997), pp. 33–40. 4

[MdGD*10] MULLEN P., DE GOES F., DESBRUN M., COHEN-STEINER D., ALLIEZ P.: Signing the unsigned: Robust surface reconstruction from raw pointsets. *Comput. Graph. Forum 29*, 5 (2010), 1733–1741. 2

[MN03] MITRA N. J., NGUYEN A.: Estimating surface normals in noisy point cloud data. In *Proc. of the SCG* (2003), pp. 322–328. 2

[MPS08] MANSON J., PETROVA G., SCHAEFER S.: Streaming surface reconstruction using wavelets. In *Proc.of the SGP* (2008), pp. 1411–1420. 2

[Mun84] MUNKRES J. R.: *Elements of algebraic topology*. Addison-Wesley, 1984. 7

[NOS09] NAGAI Y., OHTAKE Y., SUZUKI H.: Smoothing of partition of unity implicit surfaces for noise robust surface reconstruction. In *Proc. of the SGP '09* (2009), pp. 1339–1348. 2

[OBA05] OHTAKE Y., BELYAEV A., ALEXA M.: Sparse low-degree implicit surfaces with applications to high quality rendering, feature extraction, and smoothing. In *Proc. of the SGP '05* (2005). 2

[RS07] RAMOS E. A., SADRI B.: Geometric and topological guarantees for the wrap reconstruction algorithm. In *Proc. of SODA '07* (2007), pp. 1086–1095. 2

[Sam05] SAMET H.: *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., CA, USA, 2005. 4

[Sta] Stanford scanning repository. Stanford Computer Graphics Laboratory. URL: http://graphics.stanford.edu/data/3Dscanrep. 8