Tight Cocone : A Water-tight Surface Reconstructor *

Tamal K. Dey
Dept. of CIS
The Ohio State University
Columbus, OH 43210, USA
tamaldey@cis.ohio-state.edu

Samrat Goswami
Dept. of CIS
The Ohio State University
Columbus, OH 43210, USA
goswami@cis.ohio-state.edu

ABSTRACT

Surface reconstruction from unorganized sample points is an important problem in computer graphics, computer aided design, medical imaging and solid modeling. Recently a few algorithms have been developed that have theoretical guarantee of computing a topologically correct and geometrically close surface under certain condition on sampling density. Unfortunately, this sampling condition is not always met in practice due to noise, non-smoothness or simply due to inadequate sampling. This leads to undesired holes and other artifacts in the output surface. Certain CAD applications such as creating a prototype from a model boundary require a water-tight surface, i.e., no hole should be allowed in the surface. In this paper we describe a simple algorithm called Tight Cocone that works on an initial mesh generated by a popular surface reconstruction algorithm and fills up all holes to output a water-tight surface. In doing so, it does not introduce any extra points and produces a triangulated surface interpolating the input sample points. In support of our method we present experimental results with a number of difficult data sets.

Categories and Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling; I.4 [**Image Processing and Computer Vision**]: Reconstruction,Image Representation

General Terms

Algorithms, Experimentation, Theory, Design

Keywords

Point cloud, Surface reconstruction, Voronoi diagram, Delaunay triangulation

*This work is partially supported by NIMA under grant NMA201-01-1-2012, NSF under grant CCR-9988216 and an U.S. Army research grant DAAD19-02-1-0347.

SM'03, June 16-20, 2003, Seattle, Washington, USA.

1. INTRODUCTION

The problem of approximating a surface in three dimensions from its point samples appears in many applications of science and engineering. Various formulations of the problem are possible with different requirements on the input and output. Here we focus on the problem of computing a piecewise linear surface that approximates the original surface from which only a set of discrete sample points are given as input.

Because of its widespread application, this problem has been studied intensely in recent years. A very early paper on the problem was by Boissonnat [6] who proposed a sculpting of the Delaunay triangulation for reconstruction. A more refined sculpting strategy was designed by Edelsbrunner and Mücke [14] in their α-shape algorithm. Bajaj, Bernardini and Xu [4] used α-shapes for reconstructing scalar fields and three dimensional CAD models. The α-shapes require a uniform sampling and an appropriate choice of α according to the sampling density. Hoppe et. al [18] reconstructed the surface by using the zero level set of a distance function defined by the input point samples. Curless and Levoy [11] extended this distance function approach to multiple range scans. The reconstruction quality was further improved by Turk and Levoy [19]. This approach produces nice results though the reconstructed surface may not interpolate the input data points and it requires the normal information at the sample points in the input. Bernardini et al. [5] designed the ball-pivoting algorithm that also requires the normals in the input. Gopi, Krishnan and Silva [17] proposed to project the sample points with their neighbors on a plane and then lift the local two dimensional Delaunay triangulations to reconstruct the surface.

The first algorithm that has been proposed with theoretical guarantees in reconstruction is due to Amenta, Bern and Kamvysselis [1]. This algorithm called CRUST exploits the structures of the Voronoi diagram of the input point set to reconstruct the surface. Amenta, Choi, Dey and Leekha [2] introduced the COCONE algorithm which improved CRUST both in theory and practice. Boissonnat and Cazals [7] designed another algorithm based on natural neighbors following the development of CRUST. Funke and Ramos improved the theoretical complexity of the COCONE algorithm [15]. Recently Cohen-Steiner and Da have designed another Delaunay based surface reconstruction algorithm [10]. All these algorithms work nicely on dense data sets, but they face difficulty if the data contains undersampling. In most cases these algorithms produce holes or other artifacts in the vicinity of the undersampling. Many applications such as CAD need water-tight surfaces so that the surface bounds a solid. Amenta, Choi and Kolluri [3] devised POWER CRUST to meet this goal. However, this algorithm introduces many extra points in the output and also does not produce a triangulated surface. Edelsbrunner [13] proposed a Morse theo-

retic approach to reconstruct surfaces which also produces water-tight surfaces. However, the algorithm may not recover some small features of the surface even if it is densely sampled. Along the same line Giesen and John [16] designed another algorithm based on the flow complex. This algorithm may not interpolate the sample points and introduces extra points in the output.

The purpose of this paper is to announce a simple algorithm called TIGHT COCONE that outputs water-tight surfaces. The algorithm first computes a preliminary surface using COCONE which almost completes the reconstruction except at the vicinity of the undersamplings. A subsequent marking and peeling phase completes the reconstruction by filling all holes. This phase computes the output surface as the boundary of a union of Delaunay tetrahedra. This guarantees that the surface cannot have any holes. TIGHT COCONE relies on the locality of undersampling. When this property is violated, the output surface, though water-tight may be improper including being non-manifold or empty. In achieving water-tightness TIGHT COCONE does not introduce any extra points and finds triangles from the Delaunay triangulation to fill up the gaps produced by COCONE.

2. PRELIMINARY RECONSTRUCTION

The input to our algorithm is a point set in three dimensions. We assume that this point set has been sampled from a compact surface $S \subset \mathbb{R}^3$ that has no boundary. We use a Voronoi based approach to obtain an initial approximation to S from its point sample. Amenta and Bern [1] showed that the Voronoi cells are long and thin along the direction of the normals at each sample point if the sample is sufficiently dense. Based on this important observation they pioneered the surface reconstruction algorithm called CRUST. In a subsequent work, Amenta, Choi, Dey and Leekha proposed the COCONE algorithm that improved CRUST both in theory and practice. Further improvements were carried out by Dey and Giesen [12] which include a detection of undersampling in the input data. We implemented this modified COCONE algorithm which we use to obtain a preliminary surface out of the input sample. We review this algorithm briefly below; a complete description can be found in [12].

2.1 Definitions

Voronoi diagram and its dual Delaunay triangulation constitute the main data structure in our algorithm. Let P be a finite set of points in $\mathbb{R}^3$. The *Voronoi cell* of $p \in P$ is given as $V_p = \{x \in \mathbb{R}^3 : \forall q \in P - \{p\}, \|x-p\| \leq \|x-q\|\}$. The sets V_p are convex polyhedra. Closed faces shared by two and three Voronoi cells are called *Voronoi facets* and *Voronoi edges* respectively. The points shared by four or more Voronoi cells are called *Voronoi vertices*. The *Voronoi diagram* V_P of P is the collection of all Voronoi cells, faces, edges and vertices. It defines a cell decomposition of $\mathbb{R}^3$.

The *Delaunay triangulation* D_P of a set of points P is dual to the Voronoi diagram of P. The convex hull of four or more points in P defines a *Delaunay cell* if the intersection of the corresponding Voronoi cells is not empty and there exists no superset of points in P with the same property. Analogously, the convex hull of $1 \leq k \leq 3$ points defines a $(k-1)$-dimensional *Delaunay face* if the intersection of their corresponding Voronoi cells is not empty. A 0-,1-, and 2-dimensional Delaunay face is also called a Delaunay *vertex*, Delaunay *edge*, and Delaunay *triangle* respectively. The collection of Delaunay cells and their faces defines a decomposition of the convex hull of all points in P. This decomposition is a triangulation where the Delaunay cells are tetrahedra if the points are in general position.

In our case P is the input point sample of a surface $S \subset \mathbb{R}^3$. It turns out that the Voronoi cells are elongated along the normals to the surface S if P is sufficiently dense for S. This fact is used by Amenta, Bern and Kamvysselis to estimate the normals at the sample points with poles [1].

Poles: The farthest Voronoi vertex p^+ in V_p is called the positive *pole* of p. The negative pole of p is the farthest point $p^- \in V_p$ from p such that the two vectors from p to p^+ and p^- make an angle more than $\frac{\pi}{2}$. We call $\mathbf{v_p} = p^+ - p$, the *pole vector* for p. If V_p is unbounded, p^+ is taken at infinity, and the direction of $\mathbf{v_p}$ is taken as the average of all directions given by unbounded Voronoi edges.

The pole vector $\mathbf{v}_p$ estimates $\mathbf{n}_p$, the normal to S at p. Therefore, the plane passing through p with the normal as $\mathbf{v}_p$ approximates the tangent plane at p. A conservative estimate of this plane is given by thickening it around p. This motivates the following definition of *cocone* which turns out to be useful for the surface reconstruction [2, 12].

Cocone: The set $C_p = \{y \in V_p : \angle((y-p), \mathbf{v}_p) \geq \frac{3\pi}{8}\}$ is called the cocone of p where $\angle((y-p), \mathbf{v}_p)$ denotes the acute angle between the supporting lines of the vectors $y - p$ and $\mathbf{v}_p$. In words, C_p is the complement of a double cone (clipped within V_p) centered at p with an opening angle $\frac{3\pi}{8}$ around the axis aligned with $\mathbf{v}_p$. See Figure 1 for an example of a cocone.

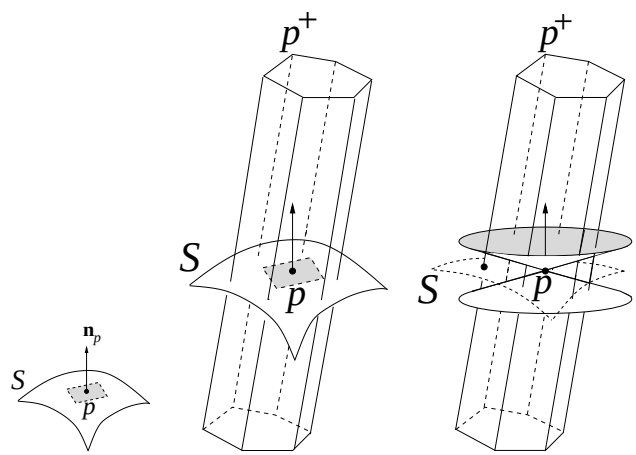


Figure 1: A Voronoi cell V_p is elongated along the normal $\mathbf{n}_p$. The pole vector $p^+ - p$ approximates $\mathbf{n}_p$. The cocone C_p is the region in V_p between the two cones at p (right).

2.2 COCONE algorithm

The COCONE algorithm proceeds as follows. Each sample chooses a set of triangles from the Delaunay triangulation of the sample P whose dual Voronoi edges are intersected by the cocones defined at the sample. All such chosen triangles over all samples are called the *candidate triangles*. If the sampling density is sufficiently high, these candidate triangles lie close to the original surface S. Further they have normals oriented nearly in the same direction as those at its three sample vertices [2]. A subsequent manifold extraction step extracts a manifold surface out of this set of candidate triangles. This manifold is homeomorphic and geometrically close to S. All these claims are theoretically proved [2].

This algorithm works nicely when the input point set P samples S densely. However, in practice the input often undersamples the surface. This undersampling may be caused by non-smoothness, inadequate sampling or noise. It is our experience that the algorithm computes many undesirable triangles near undersampled regions. The appearance of the undesirable triangles can be attributed

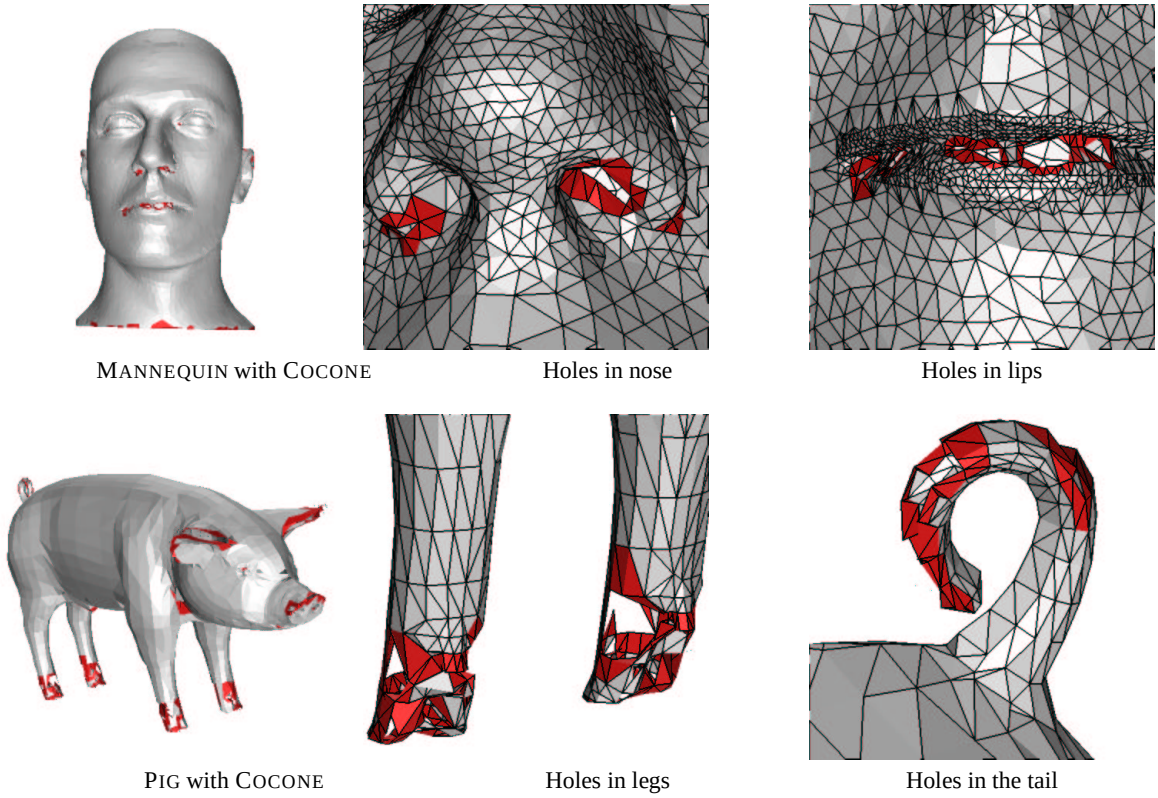


Figure 2: Preliminary surface with COCONE may have holes. The triangles bordering the holes are shaded darker.

to the fact that the samples in the undersampled regions do not have the normals reliably estimated and as a result they choose 'garbage' triangles as candidates.

In a subsequent work Dey and Giesen [12] proposed a method to detect the undersampled regions from the input point set. The algorithm which is called BOUNDARY detects the sample points that lie in the undersampled regions. The Voronoi cells of these sample points are not long and thin along the normals to the surface. This is detected by two conditions called *ratio* and *normal* conditions respectively. The ratio condition tests the 'skinniness' of the Voronoi cells while the normal condition tests if its elongation matches with those of its cocone neighbors. We refer to [12] for details.

The COCONE algorithm is modified to take advantage of the detection of sample points in the undersampled regions. It calls BOUNDARY and then lets only those sample points to choose their candidate triangles which are not marked by BOUNDARY. This modification removes most of the unwanted triangles near the undersampled regions creating holes in the surface. In Figure 2 we show reconstructions with this COCONE algorithm.

3. WATER-TIGHT RECONSTRUCTION

The modified COCONE algorithm detects undersampled regions and leaves holes in the surface near the vicinity of undersampling. Although this may be desirable for reconstructing surfaces with boundaries, many applications such as CAD designs require that the output surface be *water-tight*, i.e. a surface that bounds a solid. Formally, we define:

> *Water-tight surface*: A 2-complex embedded in $\mathbb{R}^3$ whose underlying space is same as the boundary of the closure of a 3-manifold in $\mathbb{R}^3$.

We design a very simple but elegant algorithm TIGHT COCONE to compute water-tight surfaces from an input point sample.

The overall idea of TIGHT COCONE is to label the Delaunay tetrahedra computed from the input sample as *in* or *out* according to an initial approximation of the surface and then peeling off all *out* tetrahedra. This leaves the *in* tetrahedra, the boundary of whose union is output as the water-tight surface. The output of COCONE is taken as the initial approximated surface possibly with holes and other artifacts.

By being the boundary of the union of a set of tetrahedra, the output surface has to be water-tight. However, it is not only the water-tightness, but also the geometric proximity to the original surface that are desired for the output. For this we make some decisions in the algorithm based on the following principle.

> *Principle of locality*: The undersampling is local.

This means that COCONE computes most of the intended surface except with holes that are locally repairable. Of course, if this principle is not obeyed, the output surface, though water-tight may not be close to the original one including being empty.

3.1 Marking

The COCONE algorithm as described in the previous section computes a preliminary surface possibly with holes and other artifacts at the undersampled regions. The sample points in the rest of the surface have their neighborhoods well approximated. Specifically, the set of surface triangles incident to these points form a *topological disk*. We call the points whose incident surface triangles form a topological disk *good*. The rest of the points are called *poor*.

DEFINITION 1. *The union of surface triangles incident to a good point p is called its* Umbrella *denoted as U_p.*

The marking of tetrahedra walks through the Delaunay triangulation in a depth first manner using the vertex and triangle adjacencies. It maintains a stack of pairs (p, σ) where p is a good point and σ is a tetrahedron incident to p marked *out*. Suppose the pair (p, σ) is currently popped out from the stack. The umbrella U_p locally separates the tetrahedra incident to p into two clusters, one on each side; see Figure 4. The cluster that contains σ is marked *out* since σ is already marked *out*. The other cluster gets the marking *in*. This is done by initiating a local walk from σ that traverses all tetrahedra through triangle adjacency without ever crossing a triangle in U_p and marking each tetrahedron as *out*. The rest of the tetrahedra that are not encountered in this walk get the *in* marking. During this local walk in the *out* cluster, when a vertex q of U_p is reached through a tetrahedron σ', the pair (q, σ') is put into the stack if q is good and is not explored yet, see Figure 4.

Now we face the question of initiating the stack. For this we assume that D_P is augmented with 'infinite' tetrahedra that are incident to a triangle on the convex hull of P and a point at infinity. The stack is initiated with a good point on the convex hull paired with an incident infinite tetrahedron. A pseudo-code for the marking step is given in Figure 3.

```
MARK(D_P)
  push an infinite tetrahedron with its incident
  convex hull triangle to stack S;
  Let Σ_p denote the set of tetrahedra in D_P;
  while S ≠ φ do
    let (p, σ) := pop S
    C := {σ};
    loop
      if ∃σ ∈ C and σ' ∈ Σ_p \ C
        and σ ∩ σ' is a triangle ∉ U_p
          C := C ∪ {σ'};
          if σ' ∉ S and a vertex q of σ' is good
            where q ∈ U_p
            push (q, σ') to S;
      else exit;
    forever
    mark each σ ∈ C out;
    mark each σ ∈ Σ_p \ C in;
  endwhile
```

Figure 3: Pseudo-code for marking.

For most of the data in practice the surface computed by COCONE is well connected, i.e. all triangles incident to good points can be reached from any other good point via a series of triangle adjacencies. Assuming this connectivity of the preliminary surface computed by COCONE, the above procedure marks all tetrahedra that are incident to at least one good sample point. However, the tetrahedra all of whose vertices are poor points are not marked by this step. We call them *poor tetrahedra*.

The poor tetrahedra whose vertices lie in a single undersampled region tend to be small due to the principle of locality. We choose to mark them *in* and the peeling process later is not allowed to peel them away. This allows the surface to get repaired in the undersampled region. See Figure 5 for an illustration.

Other poor tetrahedra that connect vertices from different undersampled regions tend to be big. If such a big poor tetrahedron lies outside the intended surface, we need to take it out. So, it should

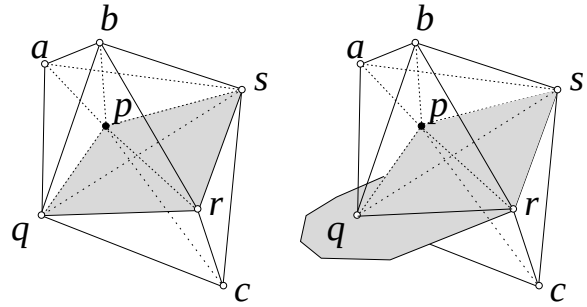


Figure 4: The umbrella of p has three triangles pqr, prs and pqs. This umbrella separates the the tetrahedra incident to p into two clusters, the upper cluster $\{absp, asqp, abpq, bqrp, brsp\}$ and the lower cluster $\{cqrp, csrp, cqsp\}$. Suppose the walk entered p with the pair $(p, bprs)$. The right picture shows that the unexplored point q has an umbrella. Therefore, the pair $(q, bprq)$ is entered into the stack since q is a good and unexplored point.

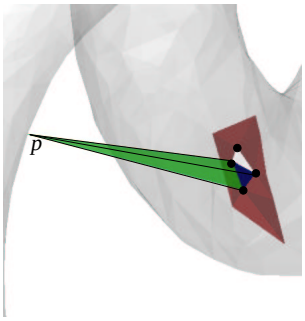


Figure 5: The four vertices marked with dark circles border a hole. The poor tetrahedron with this four vertices is marked *in*. The sharp tip p of the long tetrahedron is a good point which marks it *out*. When this tetrahedron is peeled, the triangle opposite to p fills the hole partially. The other triangle of the hole also gets into the output by a similar peeling.

be marked *out*. On the other hand if this big poor tetrahedron lies inside the intended surface, we need to mark it as *in*. Otherwise, a large void/tunnel in the surface is created by taking out this tetrahedron. We eliminate this dilemma using the principle of locality. Call a triangle in a tetrahedron *small* if its circumradius is the least among all triangles in the tetrahedron. If a poor tetrahedron has a triangle with vertices from the same undersampled region, then that triangle is small. The poor tetrahedra lying inside the intended surface have to be reached by the peeling process that peels away all *out* marked tetrahedra. This means that the inner poor tetrahedra have to be reached through the small triangle. We take this observation into account during peeling while dealing with the poor tetrahedra and defer designating them during the marking step.

3.2 Peeling

After the marking step, a walk is initiated to peel off tetrahedra that are marked *out* and some others. The boundary of the union of the remaining tetrahedra form the water-tight surface. This is also the surface of the union of the peeled tetrahedra, see Figure 6.

The walk maintains a stack of surface triangles that form the boundary of the union of the tetrahedra peeled so far. It is initiated with all convex hull triangles. At any generic step, a triangle, say

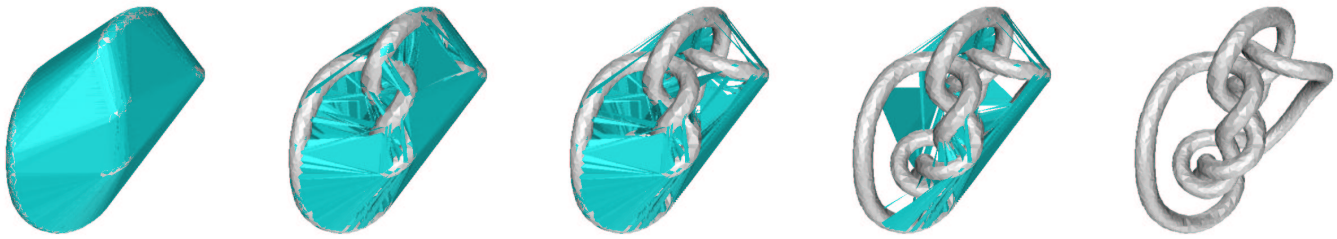


Figure 6: The boundary of the union of peeled tetrahedra as peeling process progresses.

t, is popped out from the stack. One of the tetrahedra incident to t is already peeled. If the other incident tetrahedron, say σ, is also already peeled the triangle t seprates two *out* tetrahedra and is not put in the output. Otherwise, there are two possibilities. If σ is not poor and marked *in*, we put t in the output list. In the other case either σ is marked *out* or σ is poor. When σ is marked *out* the walk should move into σ through t, which is done by replacing t with the other three triangles of σ into the stack. If σ is a poor tetrahedron, the walk is also allowed to move into σ through t only if t is not the small triangle in σ. This is done to protect peeling of the inner poor tetrahedra as we discussed before. Notice that if σ is a poor tetrahedron outside the intended surface, it will be eventually reached by the peeling process at triangles other than the small one. But, if σ is a poor tetrahedron inside, it can only be reached from outside through its small triangle due to the principle of locality.

The walk terminates with the surface triangles in the output list when there are no more triangles to process from the stack. A pseudo-code of the peeling process is given in Figure 7.

```
PEEL(D_P)
  push all convex hull triangles in D_P to stack S;
  mark all infinite tetrahedra peeled; T := φ;
  while S ≠ φ do
    t :=pop S;
    if ∃ σ ∈ D_P incident to t and not marked peeled
      if (σ is not poor) and (marked in)
        T := T ∪ t;
      else
        if (σ is marked out) or (σ is poor and t is
            not the smallest triangle in σ)
              mark σ peeled;
              push all triangles of σ other than t to S;
        endif
      endif
    endif
  endwhile
  output T
```

Figure 7: Pseudo-code for peeling.

4. EXPERIMENTAL RESULTS

4.1 Examples

In Figure 8 we show the results of our implementation of TIGHT COCONE on some difficult data sets. In MANNEQUIN there are undersamplings in eyes, lips and ears which produce holes. TIGHT COCONE closes all these holes. In particular, in the ear there is a relatively large hole since points cannot be sampled for occlusion. This hole is nicely filled. The PIG data has severe undersampling in the hoofs, ears and nose. They are mostly due to the fact that these thin and highly curved regions should have more sample points to capture the features properly. TIGHT COCONE fills all holes and produces a water-tight surface for this difficult data set. The machine part CONNECTOR has undersampling mainly due to non-smoothness. Sharp edges and corners cause undersampling which cannot be avoided by any finite sampling. All anomalies caused by this non-smooth regions are repaired to produce a water-tight surface. The undersampling in the highly curved regions such as the neck in DINOSAUR is also gracefully handled by TIGHT COCONE. The code for TIGHT COCONE has been released for academic use. It can be downloaded from [9].

In Figure 9 we show how TIGHT COCONE handles noise. Recall that TIGHT COCONE works on the principle of local undersampling. Noise can cause to violate this principle and as a result TIGHT COCONE may fail to output a surface. In the FOOT example of Figure 9, TIGHT COCONE reconstructs the surface when there is undersampling but no noise. We add noise by perturbing the points randomly within a small ball around each point. The principle of locality starts getting violated as the amount of perturbation increased. Figure 9 illustrates how TIGHT COCONE tolerates some amount of noise and fails when the noise is increased further.

4.2 Timings

The time and space complexities of TIGHT COCONE are dominated by those of the three dimensional Voronoi diagram computation. Thus, for an input sample of n points, the algorithm runs in $O(n^2)$ time and space in the worst case. However, in practice we do not observe this quadratic behavior. In Figure 10 we show the breakup of the timings of TIGHT COCONE for a number of data sets on a PC with 733 Mhz Pentium III CPU and 512 MB memory. The code was compiled with g++ compiler and at the `O1` level of optimization. CGAL 2.3 is used for all computations [8]. The timing is broken into the three major steps of TIGHT COCONE, the Delaunay triangulation, the COCONE surface generation and the final postprocessing to make the surface water-tight. It is clear from the table that the Delaunay triangulation is the most time consuming step. All computations are done with filtered floating point arithmetic for numerical robustness. If one uses floating point arithmetic, usually a gain of a factor of two in computing time is observed, but the correctness of the output may be compromised due to numerical errors.

5. DISCUSSIONS

We designed a simple and elegant algorithm to produce water-tight surfaces out of point samples. The algorithm guarantees that the output surface is water-tight. It does not introduce any extra points to do so. The output is always a subcomplex of the Delaunay triangulation. Thus, one can use the Delaunay tetrahedra inside

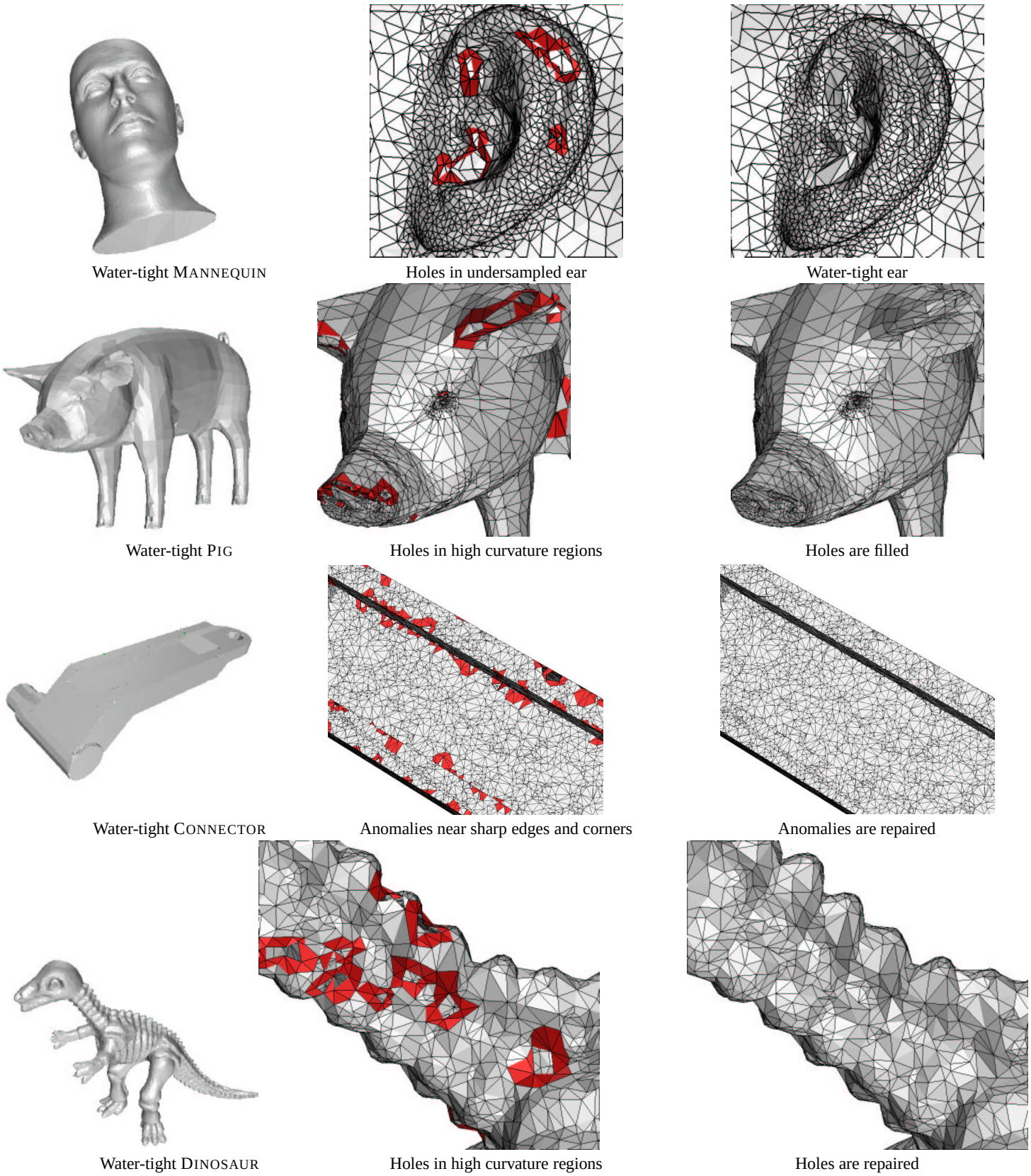
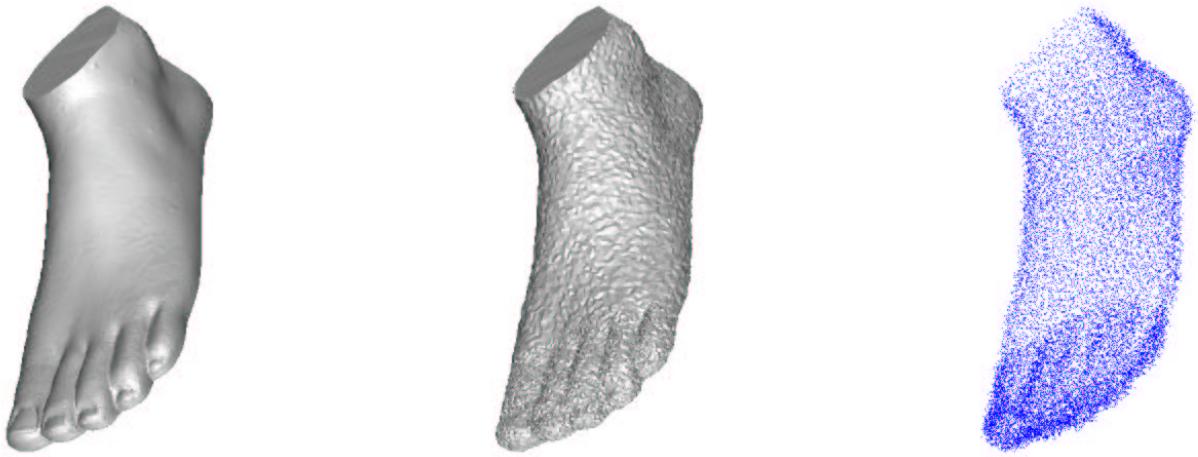


Figure 8: Results of TIGHT COCONE**. Second column shows the holes in the preliminary surface while the third column shows how they are filled. Triangles bordering the holes are shaded darker.**



FOOT without noise FOOT with little noise FOOT with more noise

Figure 9: Performance of TIGHT COCONE with increasing noise. Surface is reconstructed when points are perturbed slightly (middle). No surface is constructed when the perturbation is increased further.

object	# points	Delaunay (sec.)	Cocone (sec.)	Mark/peel (sec.)
CACTUS	3337	6.5	2.7	1.3
PIG	3511	2.8	2.7	1.4
MECH	4102	3.6	3.2	1.7
EARCAN	8459	16.5	7.1	3.2
CAT	10000	8.5	8.2	4.0
KNOT	10000	10.0	11.1	5.2
MANNEQ	12772	9.3	10.0	4.8
DINOSAUR	14050	12.6	12.1	5.9
FANDISK	16475	16.1	12.4	6.3
CLUB	16864	23.1	13.4	6.3
CONNECT	26793	41.5	18.5	9.6
FEMALE	30432	38.5	27.6	13.2
OILPUMP	30936	28.6	23.9	11.8
HEART	37912	39.9	30.3	14.5
HORSE	48485	90.2	42.0	19.4
BREVI	56152	54.3	41.8	20.2
HAND	74315	123	64.2	29.8

Figure 10: Time data.

the water-tight surface to produce a three dimensional triangulation of the model. Our experiments show the effectiveness and the efficiency of the algorithm on many example data sets.

As we have indicated, TIGHT COCONE relies on the locality of undersampling. Thus, when this condition is not satisfied, TIGHT COCONE faces trouble in computing a surface. Noisy samples are examples of such data sets as Figure 9 illustrates. Although it handles some amount of noise gracefully, it may not output any surface if the noise is beyond its tolerance limit.

The TIGHT COCONE as presented in this paper cannot reconstruct internal voids as it peels tetrahedra starting from the convex hull boundary. One needs to modify the marking and peeling routines to handle internal voids. After one phase of marking, the tetrahedra constituting the internal voids remain unmarked. Therefore, new marking phases can be initiated from an unmarked tetrahedron in order to mark all tetrahedra in the voids as *out*. Subsequently, in the peeling step, new peeling phases can be initiated as long as there are *out* tetrahedra not visited yet.

In some applications the output surface needs to produce certain intended boundaries. The COCONE algorithm/software is designed for that purpose [9]. Although it detects the intended boundaries, it also produces holes where undersampling happens unintentionally. Is it possible to recognize the intended boundaries while filling up other holes? The problem is ill-posed, but is there a proper formulation of the problem which may lead to an algorithm with certain reasonable assumptions on input? We plan to pursue this line of research.

6. REFERENCES

[1] N. Amenta, M. Bern and M. Kamvysselis. A new Voronoi-based surface reconstruction algorithm. *SIGGRAPH 98*, (1998), 415-421.

[2] N. Amenta, S. Choi, T. K. Dey and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. *Internat. J. Comput. Geom. & Applications*, **12**, (2002), 125–141.

[3] N. Amenta, S. Choi and R. K. Kolluri. The power crust. *Proc. 6th Annu. Sympos. Solid Modeling Applications*, (2001), 249–260.

[4] C. Bajaj, F. Bernardini and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3D scans. *SIGGRAPH 95*, (1995), 109–118.

[5] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Trans. Vis. Comput. Graphics*, **5**, 349–359.

[6] J. D. Boissonnat. Geometric structures for three dimensional shape representation, ACM Transact. on Graphics 3(4), (1984) 266–286.

[7] J. D. Boissonnat and F. Cazals. Smooth surface reconstruction via natural neighbor interpolation of distance functions. *Proc. 16th. Annu. Sympos. Comput. Geom.*, (2000), 223–232.

[8] CGAL: `http://www.cgal.org`

[9] COCONE: http://www.cis.ohio-state.edu/∼tamaldey/cocone.html

[10] D. Cohen-Steiner and F. Da. A greedy Delaunay based surface reconstruction algorithm. Rapport de recherche 4564, INRIA, 2002.

[11] B. Curless and M. Levoy. A volumetric method for building complex models from range images. *SIGGRAPH 96*, (1996), 303-312.

[12] T. K. Dey and J. Giesen. Detecting undersampling in surface reconstruction. *Proc. 17th Annu. Sympos. Comput. Geom.* (2001), 257–263.

[13] H. Edelsbrunner. Surface reconstruction by wrapping finite point set in space. *Ricky Pollack and Eli Goodman Festschrift*, ed. B. Aronov, S. Basu, J. Pach and M. Sharir, Springer-Verlag, to appear.

[14] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Trans. Graphics*, **13**, (1994), 43–72.

[15] S. Funke and E. A. Ramos. Smooth-surface reconstruction in near-linear time. *Proc. 13th ACM-SIAM Annu. Sympos. Discrete Algorithms*, (2002), 781–790.

[16] J. Giesen and M. John. The flow complex: A data structure for geometric modeling. *Proc. 14th. Annu. ACM-SIAM Sympos. Discrete Algorithms*, (2003), 285–294.

[17] M. Gopi, S. Krishnan and C. T. Silva. Surface reconstruction based on lower dimensional localized delaunay triangulation. *Computer Graphics Forum*, **19**(3), August 2002.

[18] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald and W. Stuetzle. Surface reconstruction from unorganized points. *SIGGRAPH 92*, (1992), 71-78.

[19] G. Turk and M. Levoy. Zippered polygon meshes from range images. *Proc. SIGGRAPH 94*, (1994), 311-318.