Robustness Issues in Surface Reconstruction

Tamal K. Dey, Joachim Giesen, and Wulue Zhao*

Abstract. The piecewise linear reconstruction of a surface from a sample is a well studied problem in computer graphics and computational geometry. A popular class of reconstruction algorithms filter a subset of triangles of the three dimensional Delaunay triangulation of the sample and subsequently extract a manifold from the filtered triangles. Here we report on robustness issues that turned out to be crucial in implementations.

1 Introduction

While implementing geometric algorithms one often has to face the problem of numerical instabilities. That is also the case for Delaunay based surface reconstruction algorithms that filter a subset of Delaunay triangles for reconstruction. But careful examination shows that the only step that inherently requires instable numerical decisions is the construction of the Delaunay triangulation itself. All other steps can be implemented relying either on numerically stable or purely combinatorial decisions. Here we want to emphasize the following design principle for geometric implementations: *Avoid numerical decisions whenever possible.* As our experience shows this pays off as well in robustness as in running time.

2 Filter based reconstruction algorithms

Filter based algorithms consider a subset of triangles of the three dimensional Delaunay triangulation of a sample $P \subset \mathbb{R}^3$ for reconstruction. All these algorithms contain three generic steps:

(1) FilterTriangles. A set of *candidate* triangles is extracted from the Delaunay triangulation of the the sample. In general the underlying space of these triangles is not a manifold, but a manifold with boundary can be extracted.
(2) Pruning. We want to extract a manifold from the set of candidate triangles by walking either on the inside or outside of this set. During the walk we may encounter the problem of entering a triangle with a bare edge, i.e. an edge with only one incident triangle. The purpose of this step is to get rid of such triangles.
(3) Walk. We walk on the in- or outside of the set of triangles that remained after Pruning and report the triangles walked over.

Different filter based reconstruction algorithms distinguish themselves in the FilterTriangles step. In the following we shortly explain two different filter strategies which both come with theoretical guarantees. But there are also other algorithms that fit in the general scheme presented above.

* Department of CIS, Ohio State University, Columbus, OH 43210, {tamaldey,giesen,zhaow}@cis.ohio-state.edu

2.1 Crust

The Crust algorithm of [1] first computes the Voronoi diagram of the sample P, i.e. the dual of the Delaunay triangulation. A subset of the Voronoi vertices called *poles* is used to filter Delaunay triangles.

Poles: Let V_p be the Voronoi cell of a sample point $p \in P$. The Voronoi vertex p^+ in the Voronoi cell V_p farthest from p is called the positive pole of p. The negative pole of p is the point $p^- \in V_p$ farthest from p such that the two vectors $(p^+ - p)$ and $(p^- - p)$ make an angle more than $\frac{\pi}{2}$. We call $\mathbf{v_p} = p^+ - p$ the pole vector of the sample p. See Figure 1. If V_p is unbounded special care has to be taken.

The Crust algorithm computes the Delaunay triangulation of the union of the sample P with the set of poles. All triangles in this Delaunay triangulation that are incident to three samples from the original sample P are candidate triangles for the reconstruction.

2.2 Cocone

The Cocone algorithm of [2] avoids the second Delaunay computation. This algorithm is using a set called *cocone* for every sample point $p \in P$ to filter Delaunay triangles.

Cocone: The set $C_p(\theta) = \{y \in V_p \ : \ \angle((y-p), \mathbf{v_p}) \geq \frac{\pi}{2} - \theta\}$ is called the cocone of p. In words, $C_p(\theta)$ is the complement of a double cone centered at p (clipped within V_p) with opening angle $\frac{\pi}{2} - \theta$ around the axis aligned with $\mathbf{v_p}$. See Figure 1.

The Cocone algorithm filters a triangle t from the Delaunay triangulation of the sample P if all cocones of the three sample points incident to t intersect the Voronoi edge dual to t.

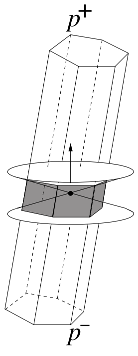

Fig. 1. A Voronoi cell together with the normalized pole vector and the cocone.

3 Robustness

In this section we are going to discuss the robustness of the four steps (including the computation of the Delaunay triangulation / Voronoi diagram) of the generic filter based algorithm.

3.1 Delaunay triangulation

Delaunay triangulation algorithms are usually designed for the real RAM, a random access machine that can handle real numbers at unit cost. Most of these algorithms assume that two geometric predicates, the *sidedness* test and the *incircle* test, can be evaluated accurately. The sidedness test decides whether a point lies left of, right of or on an oriented hyperplane. The incircle test decides whether a point lies outside of, inside of or on a sphere. Both predicates amount to the computation of the sign of a determinant. Implementing these tests using floating point arithmetic can result in completely unreliable output or even infinite loops depending on the chosen algorithm.

The naive way to circumvent these problems is to compute the value of the determinants using exact arithmetic and to read of the sign from the value. A more efficient technique is the use of floating point filters. A floating point filter computes an approximate value of an expression and a bound for the maximal deviation from the true value. If the error bound is smaller than the absolute value of the approximation, approximation and exact value have the same sign. In this case we can use the sign of the approximation to decide the predicate.

In our implementations we used the floating point filters provided by the computational geometry algorithms library CGAL [3]. Our experience shows that the running time is no more than twice the running time of a pure floating point implementation. See Figure 2(a) for an example how the use of floating point arithmetic can affect the reconstruction algorithms (after FILTERTRIANGLES).

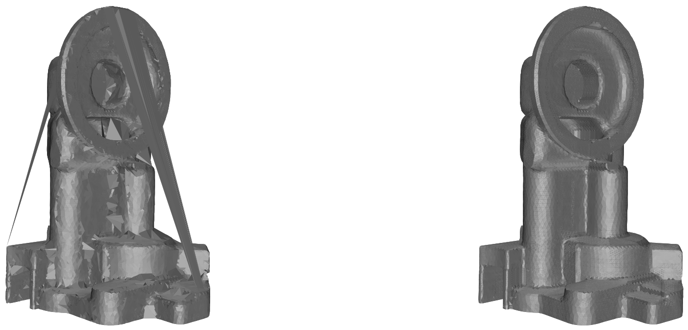

Fig. 2. Candidate Triangles computed by the COCONE algorithm from a Delaunay triangulation computed with floating point arithmetic (left) and filtered exact arithmetic (right).

3.2 Filter triangles and pruning

The step FILTERTRIANGLES is purely combinatorial in the CRUST algorithm and hence robust. In the COCONE algorithm this step involves the numerical decision if a Voronoi edge intersects a cocone. But it turns out that the exact size of the opening angle of the cocone is not important. Thus the decision if a Voronoi edge intersects a cocone need not be really accurate.

The PRUNING step is purely combinatorial. It involves only the decision if an edge is bare, i.e. if it has exactly one incident triangle. Hence this step is also robust.

3.3 Walk

A pseudo code for the implementation of the walk is given below.

```
WALK (C, (t, e))
 1  S := {t}
 2  Pending := ∅
 3  push (t, e) on Pending.
 4  while Pending ≠ ∅
 5    pop (t, e) from Pending
 6    if e is not marked processed
 7      mark e processed.
 8      t' := SURFACENEIGHBOR (C, t, e)
 9      S := S ∪ {t'}
10      if e' ≠ −e incident to t' induces the same orientation on t' as −e
11        push (t', e') on Pending.
12  return S
```

The WALK takes two parameters, a complex C containing the candidate triangles and an oriented triangle t. The orientation of t is given by an oriented edge $\boldsymbol{e}$ incident to t. First, the surface S is initialized with the triangle t (line 1). Next a stack *Pending* is initialized with the oriented triangle t (lines 2 and 3). As long as the stack *Pending* is not empty, we pop its top element $(t, \boldsymbol{e})$. If the edge $\boldsymbol{e}$ is not already processed we call the function SURFACENEIGHBOR to compute the surface neighbor of the oriented triangle t, i.e. the triangle t' that 'best fits' t (line 8). Then t' is inserted in S and two new oriented triangles are pushed on the stack *pending* (lines 9 to 11). Finally we return S (line 12).

The question is how to implement the function SURFACENEIGHBOR which has to circle around edge $\boldsymbol{e}$ according to the orientation of $\boldsymbol{e}$ until it first encounters another candidate triangle. This is the triangle we are looking for. Let t' always denote a candidate triangle incident to t via $\boldsymbol{e}$.

A naive implementation could compute first the value $(\boldsymbol{n}_{t'} \times \boldsymbol{n}_t) \cdot \boldsymbol{e}$ for every triangle t'. Here $\boldsymbol{n}_t$ and $\boldsymbol{n}_{t'}$ denote the normalized normals of t and t' both oriented according to the orientation of t. From the sign of this value one can decide if t' and t lie on the same side of the hyperplane h_1 spanned by the vectors $\boldsymbol{n}_t$ and $\boldsymbol{e}$. Next the value $\lambda_{t'} = (\boldsymbol{v}_{t'} \cdot \boldsymbol{n}_t)$ is computed. Here $v_{t'}$ denotes the normalized vector from the head of $\boldsymbol{e}$ to the vertex opposite of $\boldsymbol{e}$ in t'. See Figure 3(a). Using the sign of $\lambda_{t'}$ one can decide if t' lies above or below the oriented hyperplane h_2 defined by t. In case that there exists a triangle t' which lies above h_2 and on the same side of h_1 as t the function SURFACENEIGHBOR returns the triangle which has the smallest value $\lambda_{t'}$ among all such triangles. Otherwise it returns the triangle which has the largest value $\lambda_{t'}$ among all triangles t' that do not lie on the same side of h_1 as t. If such a triangle does not exist the function just returns the triangle t' which has the smallest value $\lambda_{t'}$. The WALK with this implementation of SURFACENEIGHBOR can produce holes due to numerical inaccuracy in the computation of $\lambda_{t'}$ when walking over *slivers*, i.e. flat tetrahedra which frequently appear in the Delaunay triangulation of surface samples. See Figure 3(b).

Fig. 3. The instable way to compute the surface neighbor (left) and a zoom on a reconstruction after the WALK (right).

A robust and faster implementation of the function SURFACENEIGHBOR avoids numerical computations by exploiting the combinatorial structure of the Delaunay triangulation to choose the next triangle. Every triangle in the Delaunay triangulation has two incident tetrahedra. We fix a global orientation. For the triangle t we choose the tetrahedron that is oriented according to the orientation of $(t, \boldsymbol{e})$ and the global orientation. In Figure 3(b) this is tetrahedron T_1. Then we go to neighboring tetrahedra $T_2, T_3, \ldots$ also incident to $\boldsymbol{e}$ until we find the triangle t'. See Figure 4(a). The WALK with this implementation of SURFACENEIGHBOR is robust since no numerical decisions are involved. The latter is also the reason why it is fast provided the Delaunay triangulation is given in a form which allows to answer queries for neighboring tetrahedra quickly. In our implementations which use CGAL such queries are supported. We observe that the time spend for the WALK is only a tiny fraction of the time needed to compute the Delaunay triangulation.

Fig. 4. The stable way to compute the surface neighbor (left) and a zoom on a reconstruction after the WALK (right).

References

1. N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. *Discr. Comput. Geom.*, **22**, (1999), 481–504.
2. N. Amenta, S. Choi, T. K. Dey and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. *Proc. 16th. ACM Sympos. Comput. Geom.*, (2000), 213–222.
3. `http://www.cgal.org`