Tight Cocone and CGAL[*]

Tamal K. Dey Samrat Goswami[†]

April 29, 2002

Abstract

The purpose of this article is to announce the successful completion of the TIGHT COCONE software developed under the COCONE project by the JYAMITI group at the Ohio State University, and the roles CGAL played in the project over the past three years. We plan to elaborate on these two issues in the workshop.

1 Introduction

During the past three years, the JYAMITI group at the Ohio State University was engaged in developing a robust and efficient software for reconstructing surfaces from point clouds [3]. The algorithm described by Dey and Giesen [1] was implemented in the COCONE software for this purpose. CGAL libraries [4] were selected for a robust Delaunay triangulation that was needed for the COCONE software. A number of people from academia and research laboratories have downloaded the software since its release.

Being capable of detecting boundaries in the surface, the released software produced holes in the reconstructed surfaces where undersamplings occur. Some applications such as GIS or scientific visualizations require that the reconstructor have the capability of detecting boundaries in the surface. The COCONE software was built to have this property. Since intended boundaries in the surface are indistinguishable from the ones created by undersampling, this software also created "holes" where sample density is poor. In some applications such as CAD designs and reverse engineering, a water tight surface is needed so that the digital form of the surface can be translated into a real prototype. To this end we have designed an algorithm which takes the output of COCONE as input and makes it water tight using the Delaunay triangulation produced by CGAL.

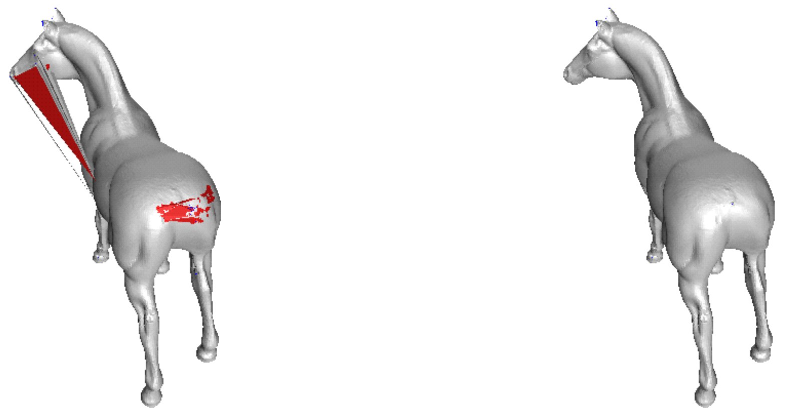
The purpose of this article is to announce this recent success in surface reconstruction from point clouds and the roles CGAL played in achieving it. In Figure 2 we show examples where the new version of COCONE, which we call TIGHT COCONE, repairs the surface in the vicinity of nonsmoothness, noise or inadequate sample points. The output surface is guaranteed to be water tight though it may contain nonmanifold properties where undersampling is extreme. In achieving water tightness TIGHT COCONE does not introduce any extra point and finds triangles from the Delaunay triangulation to fill up the gaps produced by COCONE.

The COCONE project has benefited tremendously from using CGAL libraries. At the beginning when we were searching for publicly available robust and efficient codes for Delaunay triangulations and other basic geometric computations such as computing the center and radius of a sphere

[*]This research has been supported by NSF under grant CCR-9988216.

[†]Department of Computer and Information Sciences, Ohio State U., Columbus, OH 43210, USA. e-mail: `{tamaldey,goswami}@cis.ohio-state.edu`

circumscribing four points, we settled on CGAL. Never did we change that platform during the last three years of the COCONE project. In section 2 we describe some of our experiences with CGAL which we believe will serve as a good feedback to the CGAL developers and its users.



HORSE with Floating Point computation HORSE with Filtered Floating Point computation

Figure 1: Effect of the mode of computation on the output of Tight COCONE

2 Experience with CGAL

The CGAL libraries have been extremely useful for the COCONE project. The data structures and functionalities provided by CGAL make the code development easy. In particular, CGAL lets the user to add new classes on top its own classes with different number types. This makes the code robust and modular. While implementing Tight Cocone using CGAL we tumbled upon some anomalies and aspects of CGAL that we feel worth reporting.

Robustness. Numerical robustness is an important aspect of any surface reconstruction software. See Figure 1 for an example where floating point computations give incorrect results. On the other hand, exact computations are slow. To trade-off between accuracy and speed, we use `Filtered_exact()` for number types that uses floating point filters. Even then we found that the following two computations run into problem with some data sets.

- Circumcenter computation:
  Even though CGAL does not generate completely flat tetrahedra in the Delaunay Triangulation, it does generate "almost" flat tetrahedra occassionally. In case of such tetrahedra, circumcenter computation using `CGAL::Filtered_exact(double, CGAL::MP_Float)` numbertype is error-prone. This may be problematic for our software since datasets having these kind of tetrahedra in their Delaunay Triangulation are not rare. So we use exact computation to circumvent the problem with the `CGAL::Quotient<CGAL::MP_Float>` number type. It is reliable but at the same time it is quite slow compared to `Filtered_exact`. To optimize the trade-off between time and accuracy and also to exploit the fact that the occurrence of flat tetrahedra are not very frequent, we used the wrapper around the exact number type, called `Lazy_exact_nt`, in our circumcenter computation. This number type speeds up the computation considerably as it uses exact computation in an adaptive fashion.

- Comparing distances:
  The function `has_smaller_dist_to_point()` to find out if a point p is closer to q than another point r is not stable and it gives "segmentation fault" on certain datasets. We avoid the problem by first computing the distances separately and then comparing them explicitly.

Insertion order dependency. A degenerate Delaunay polytope (more than four vertices co-spherical) can be triangulated in many ways. We believe that CGAL makes this choice depending upon the insertion order of input points. This creates two problems.

First, CGAL produces combinatorially different triangulations on different platforms when the datasets have degeneracies. For example, we get different triangulations for the data set CONNECTOR on a PC, on a SGI machine and on a SUN machine. Although this is not so much of a problem for TIGHT COCONE, it is annoying.

Secondly, sometimes CGAL produces different triangulations on the same dataset when fed in different order to the same machine. This created a problem for another version of COCONE called SUPERCOCONE which handles large data. It would be nice if this annoying property can be eliminated.

Timings. Computing time is always an issue with the surface reconstruction software. In Table 3 we show the breakup of the timings of TIGHT COCONE on a PC with 733 Mhz Pentium III CPU and 512 MB memory. The code was compiled with g++ compiler and at the 01 level of optimization. CGAL 2.3 is used for all computations. 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. If we are willing to give up the numerical robustness, we can gain in computing time by a factor of 2. Table 4 summarizes the timings with only floating point computations and with the filtered floating point computations. As expected, the floating point computation produces wrong results as shown in Figure 1 and sometimes causes the program to crash as indicated in Table 4. This experiment suggests that exact numerical computations are absolutely necessary for the surface reconstruction software.

Acknowledgements. We thank Monique Teillaud and Sylvain Pion for their help with CGAL number types.

References

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

[2] T. K. Dey and S. Goswami. Tight cocone: A watertight surface reconstructor. *Manuscript*, 2002.

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

[4] http://www.cgal.org

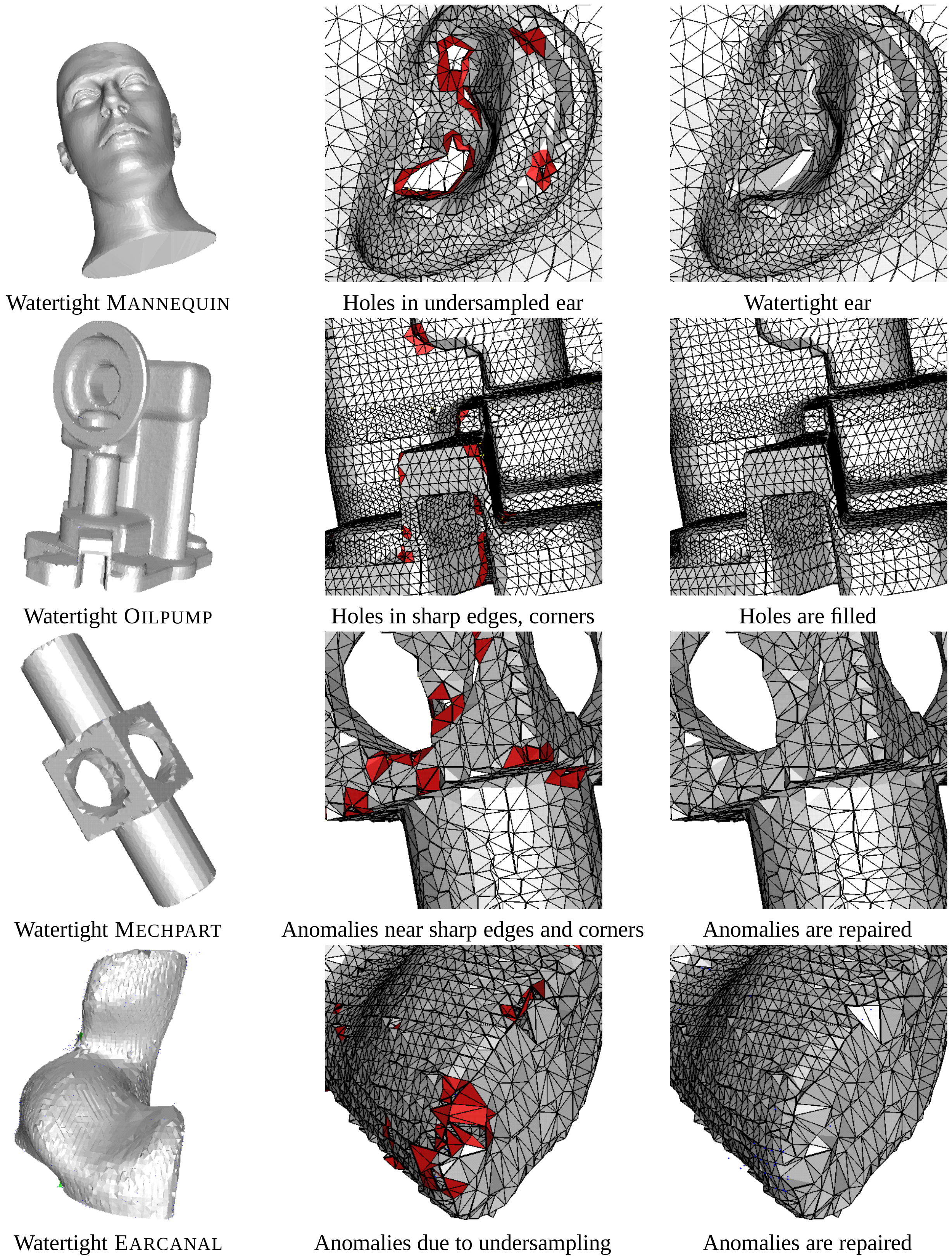


Figure 2: Tight cocone produces water tight surfaces.

object	# points	Delaunay time(sec.)	Cocone time(sec.)	Postprocess time(sec.)
CACTUS	3337	6.56	2.72	1.38
MECHPART	4102	3.68	3.27	1.73
EARCANAL	8459	16.54	7.16	3.22
CAT	10000	8.51	8.22	4
KNOT	10000	10.01	11.18	5.29
MANNEQUIN	12772	9.31	10.08	4.89
DINOSAUR-25	14050	12.63	12.16	5.91
FANDISK	16475	16.16	12.44	6.3
CLUB	16864	23.12	13.47	6.34
HAND	25626	62	22.32	10.39
CONNECTOR	26793	41.58	18.5	9.68
FEMALE	30432	38.56	27.61	13.28
OILPUMP	30936	28.68	23.92	11.88
HEART	37912	39.99	30.36	14.5
HORSE	48485	90.24	42.07	19.45
BREVI	56152	54.39	41.82	20.28
DRAGON HAND	74315	123	64.21	29.83

Figure 3: Time data.

object	# points	Floating Point Comput. time(sec.)	Filtered Floating Point Comput. time(sec.)
CACTUS	3337	- Crash -	10.66
MECHPART	4102	4.69	8.68
EARCANAL	8459	- Crash -	26.92
CAT	10000	11.22	20.73
KNOT	10000	14.75	26.48
MANNEQUIN	12772	13.31	24.28
DINOSAUR-25	14050	16.96	30.7
FANDISK	16475	17.96	34.9
CLUB	16864	18.93	42.93
HAND	25626	- Crash -	94.71
CONNECTOR	26793	- Infinite Loop -	69.76
FEMALE	30432	40.59	79.45
OILPUMP	30936	- Crash -	64.48
HEART	37912	41.09	84.85
HORSE	48485	61.62	151.76
BREVI	56152	- Crash -	116.49
DRAGON HAND	74315	- Crash -	217.04

Figure 4: Time Comparison.