Advanced Algorithms	Fall 2015

Program Assignment 1

Prof. Jyh-Ming Lien

152cpg04

Yunjoo Park

yunjoopark12@gmail.com

git clone https://github.com/yunjoopark/programmingAssignment1.git

In addition to the implementation of the algorithms, you also need to turn in a report in LaTeX. Your report should include an overview of your implementation intuitions (e.g., how the circumsphere is computed, how the poles are found from qhull), all the example outputs, known bugs, and known limitations.

1 Part 1: 3D Delaunay Triangulation

1.1 Problem definition and Analysis

Let $P := \{p_1, p_2, ..., p_n\}$ be a set of points in the plane. In that, a *triangulation* of P is defined a maximal planar subdivision whose vertex set is P and it denoted by $DT(P)$. When there are three points $p_i, p_j, p_k \in P$ are vertices in the same face of the $DT(P)$, the circle through p_i, p_j, p_k contains no other points. This circle is called the circumcircle of the triangle defined by (p_i, p_j, p_k). We can relatively easily compute the *Delaunay Triangulate* by using convex hull("qhull").

1.2 Platforms

Languages: C

Platform: Microsoft visual Studio 2013

OS: Windows

1.3 Examples of Inputs

1. There are 11 test data.
2. Each data file has different values.
3. Each line has three value, x, y, z.

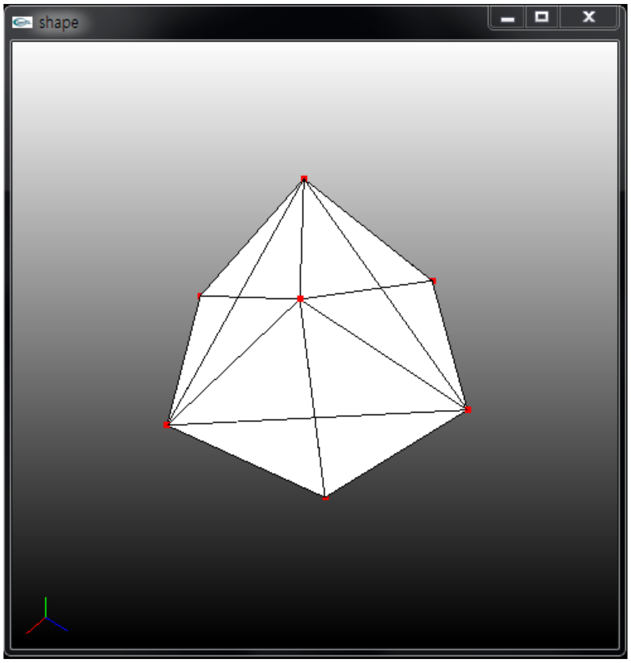

Figure 1: An example of tetrahedra *i.cube*

1.4 Outputs

2 Part 2: 3D α-shape (30 pts)

For a given value of α, the α-shape includes all the tetrahedra in the Delaunay triangulation which have an empty circumsphere with *squared radius* equal or smaller than α. Algorithm 2.1 sketches this idea. Examples of α-shape is illustrated in Figure 3. More detailed information about 3-d α-shape can be found in this CGAL page and in the paper by Edelsbrunner and Mücke [2].

Algorithm 2.1: α SHAPE($P[1 \cdots n], \alpha$)

comment: P is a set n points

D = Delaunay triangulation of P
for each tetrahedron $t \in D$
  do { **if** the circumsphere of t has squared radius larger than α
    do Remove t

2.1 What should you do?

Your goal is to use *qhull* to implement Algorithm 2.1. The output of your program should remain the same as it is now, therefore the provided OpenGL interface can understand your output.

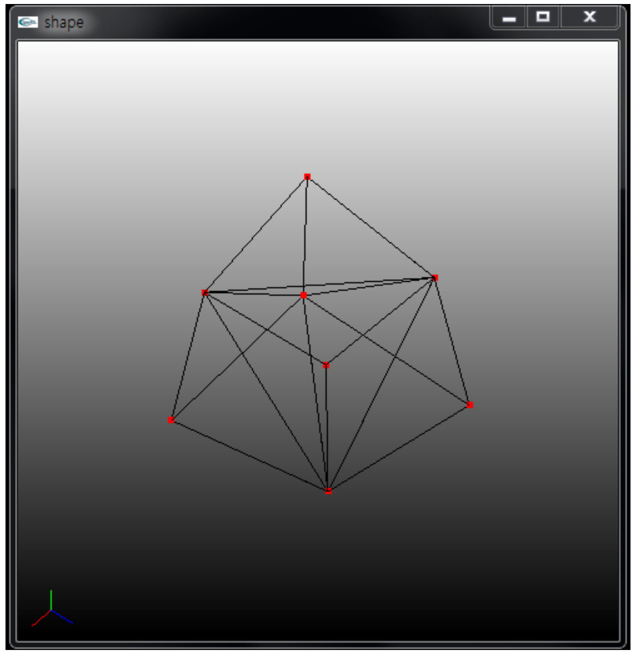

Figure 2: An example of edges *i.cube*

3 Part 3: Crust Algorithm (40 pts)

This part is due on Oct 22, 11:59pm, 2015.

The idea of Crust for surface reconstruction is proposed by Nina Amenta et al. [1] in 1998. Briefly, the crust of a set of points is a set of edges (in 2d) or triangles (in 3d) whose circumcircles is empty of (1) input points and (2) the medial axis. A complete crust algorithm is shown in Fig. 4.

The main challenge of this part of assignment is to find the poles for each site. The poles of site s are the furthest vertices of s's Voronoi cell; one on each side of the surface. An example of the poles is shown in Fig. 5.

3.1 What should you do?

Your goal is to use *qhull* to implement the crust algorithm. The output of your program should remain the same as it is now, therefore the provided OpenGL interface can understand your output.

References

[1] N. Amenta, M.Bern, and M. Kamvysselis. A new voronoi-based reconstruction algorithm. In *SIGGRAPH, Orlando, FL*, 1998. 3

[2] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Trans. Graph.*, 13(1):43–72, January 1994. 2

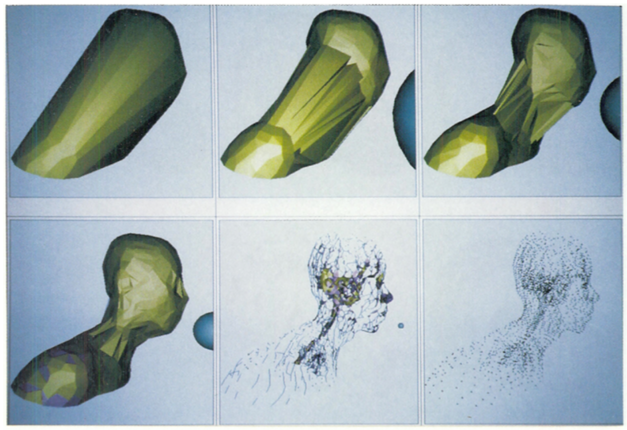

Figure 3: An example of alpha shape with different values

1. *Compute the Voronoi diagram of the sample points S*
2. *For each sample point s do:*
   (a) *If s does not lie on the convex hull, let p^+ be the farthest Voronoi vertex of V_s from s. Let n^+ be the vector $sp+$.*
   (b) *If s lies on the convex hull, let n^+ be the average of the outer normals of the adjacent triangles.*
   (c) *Let p^- be the Voronoi vertex of V_s with negative projection on n^+ that is farthest from s.*
3. *Let P be the set of all poles p^+ and p^-. Compute the Delaunay triangulation of $S \cup P$.*
4. *Keep only those triangles for which all three vertices are sample points in S.*

Figure 4: 3D crust algorithm

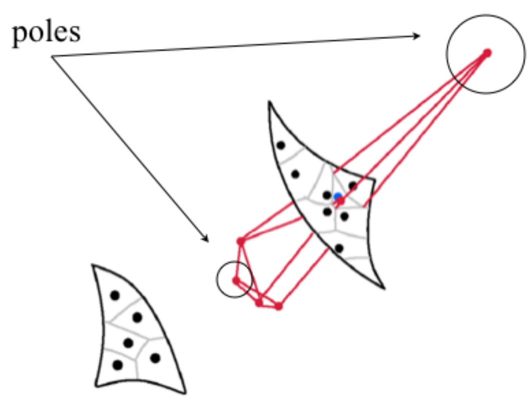

Figure 5: Poles