3D Convex Hulls & Lifting Maps

Ankush Gupta
gupta@berkeley.edu

In the dead-week and after finals, I had some free time and hence I thought that it would be fun to implement the convex-hull algorithm for higher dimensions, as it is one of the most important alogrithms. I implemented the algorithm for three-dimensions.

I followed the description in the Dutch book and online references. I am representing the subdivision using a DCEL structure[1]. I am also using the conflict graph to get expected $\mathcal{O}(n \log n)$ time.

The algorithm runs pretty fast (but not quite as fast as my delaunay implementation). The following are its run-times on different sizes of randomly generated points; it was ran on my laptop running which has `Intel Core i3 CPU M 350 @ 2.27GHz` processor and 4GB RAM.

10k	100k	1000k
0.44 s	6.28 s	82.97s

Lifting Maps

I also ran the algorithm on parabolically lifted 2d-points. The convex-hulls indeed look similar to delaunay triangulations! Below are some screenshots. The 3d convex hull was visualized using OpenSceneGraph,

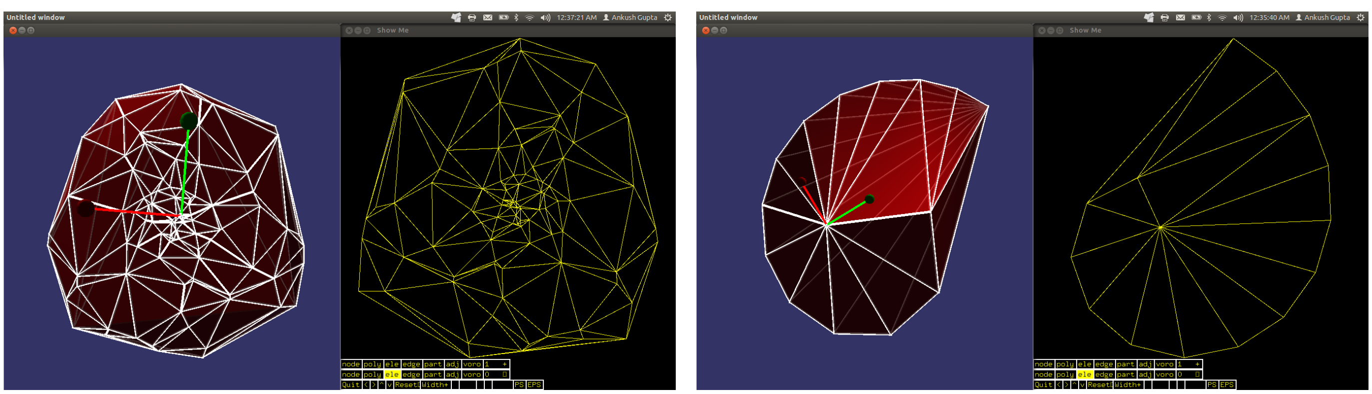

Figure 1: Note: The convex-hull (red-colored) has the horizontal axis mirrored, because we are looking from below.

[1] I tried using quad-edges but always ran into some bug; I guess I need to think about quad-edges more deeply.

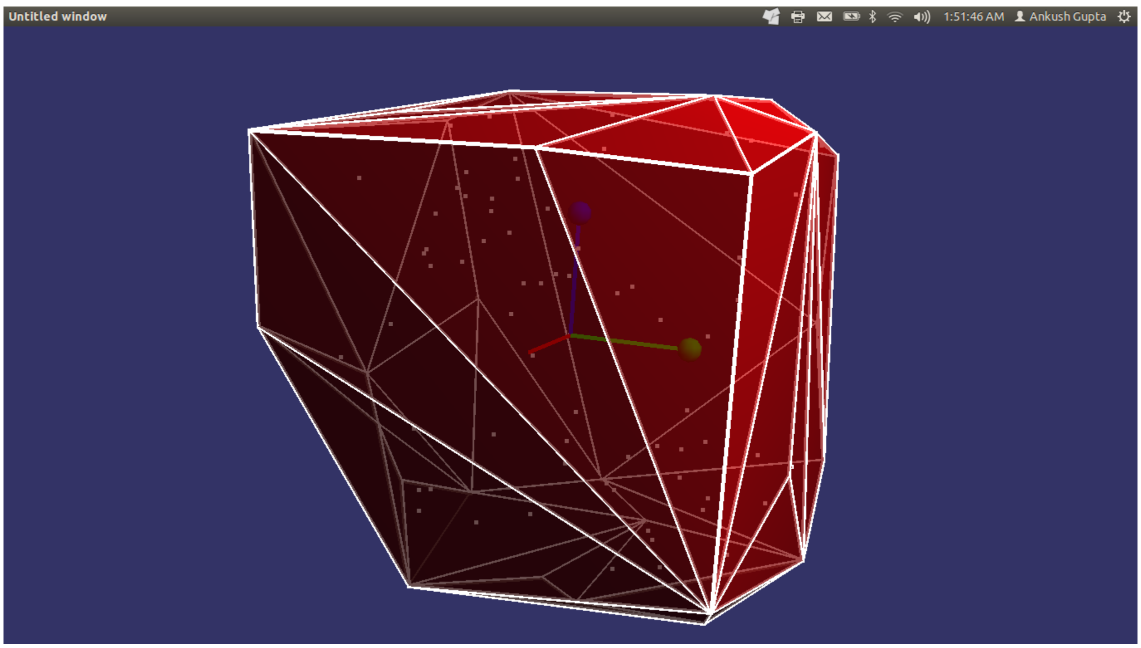

Figure 2: Convex hull of some random points.