Class work: Convex hulls

Summary: Convex hull in 3D

- Structure: Consists of vertices, edges and faces and is a convex polyhedron, also called a *polytope*.
- Properties: A face is on the hull if and only of it is *extreme* (all points in P are on or on one side of it). All angles between faces on the hull are convex (< 180).
- Size: Polytopes in 3D satify Euler Formula $V - E + F = 2$. This implies that the hull of n points in 3D is such that $E = O(n), F = O(n)$.
- Algorithms: Graham scan does *not* extend to 3D. Gift wrapping, quickhull, incremental and divide-and-conquer approaches all extend nicely to 3D. CH via divide-and-conquer runs in $O(n \lg n)$ time, which is the same as in 2D.

Higher dimensions

A point in 3 dimensions is specified by 3 coordinates (x, y, z). A point in 4 dimensions is specified by 4 coordinates (x, y, z, t). We can think of the 4th coordinate as the time, but not necessarily; for e.g. a person can be viewed as a point in a 4-dimensional space $(height, age, salary, numberOfFriends)$.

- Structure: Consists of vertices (0-dimensional), edges (1-dimensional facets), triangular (2-dimensional) facets, 3-dimensional tetrahedral facets, ...
- Size: It was shown that the size of the hull of n points in 4D can have $\Omega(n^2)$ faces ($\Omega(n^{\lfloor d/2 \rfloor})$ in d-dimensions).
- Algorithms: Can be computed in $O(n \lg n + n^{\lfloor d/2 \rfloor})$. Also in $O(n \cdot d \cdot F)$ to produce F faces.

Problems

1. **Hypercube:** A zero-dimensional cube is a point. A one-dimensional cube is a segment, and can be viewed as a point swept in the first dimension. A two-dimensional cube is a square, and can be viewed as a segment swept in the 2nd dimension. A three-dimensional cube is a normal cube, and can be viewed as a square swept in the third dimension.

   A 4-dimensional cube (a *hypercube*) can be viewed as consisting of two copies of cubes in 3 dimensions: start with one 3d-cube and sweep it in the fourth dimension, dragging new edges between the original's cube vertices and the final cube.

   Write the number of vertices and edges in cubes, for $d = 0, 1, 2, 3, 4$.

2. (assume 2D) The *diameter* of a set of points $p_0, p_1, ...$ is defined to be the largest distance between any two points in the set: $max_{i,j}|p_i - p_j|$. Prove that the diameter of a set is achieved by two hull vertices.

3. (assume 2D) Onion peeling: Start with a finite set of points $S = S_0$ in the plane, and consider the following iterative process: Let S_1 be the set $S_0 \setminus \delta H(S_0)$: S_0 minus the boundary of the convex hull of S_0. Similarly, define $S_{i+1} = S_i \setminus \delta H(S_i)$. The process continues until the empty set is reached. The hulls $H_i = \delta H(S_i)$ are called the *layers* of the set, and the process is called *onion peeling* for obvious reasons. Any point on H_i is said to have *onion depth*, or just *depth*, i. Thus the points on the hull of the original set have depth 0.

   - Consider a set of points and draw its layers.
   - What is the maximum/minimum number of layers of a set of n points? Draw examples.
   - How would you go about computing the layers of a set of points and how long?

Towards implementing Gift wrapping in 3D

Write down pseudocode for implementing GiftWrapping in 3D (at the moment it looks like this will be your next assignment). Assume that you start with an array of points that store the coordinates of the points. Express the pseudo-code for gift wrapping in terms of generic functions, such as:

- a queue/stack that stores all the edges that are on the frontier of the search (in other words: all edges that have found only one adjacent face so far);
- a generic function that takes an edge as an argument and returns TRUE if the edge is on the frontier: edgeOnFrontier(edge e): return TRUE if precisely one face adjacent to edge e has been discovered so far.
- ...

Refine the algorithm to include details on the representation of the CH. You'll want to keep track of the vertices, edges and faces that are on the convex hull, and their adjacencies. Basically a 3d hull consists of:

- An array/list of the vertices on the hull (ideally pointers to points in P so that coordinates are not duplicated)
- An array/list of edges on the hull (edges may be inferred from the faces so perhaps they dont need to be stored explicitly)
- An array/list of faces on the hull

```
struct _face3d {
   point3d *p1, *p2, *p3;  //the vertices on this face (in ccw as looking from outside)
   face3d *f12; //face to the left of p1p2
   face3d *f23; //face to the left of p2p3
   face3d *f31; //face to the left of p3p1
} face3d;
```