Class work: Convex hulls

Convex hull in 3D: Summary

- 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.

Convex hulls in 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.

Class work: Problems

1. **Cubes and Hypercubes:** 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 $d = 0, 1, 2, 3, 4$. (Hint: think recursively in terms of the number of faces, edges and vertices in lower dimensions).

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: $max_{i,j}|p_i - p_j|$. Argue 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?

Next assignment will be to implement finding the 3hull using: (1) the naive algorithm; and (b) either gift wrapping or incremental algorithms.

Assume that you start with an array of points that stores the coordinates of the points.

4. Write detailed pseudocode for implementing the naive 3d hull algorithm.

   For this assignment, for simplicity, you do not need to store the topology of the hull, that is, the neighboring information for its faces, edges and vertices. Instead you will store the hull as an array/vector of faces.

   You do need to consider degeneracies, and how to handle them. In 2d, points that are collinear need to be handled separately and are considered degenerate cases. What are the degenerate cases in 3d? Come up with a simple way to handle them.

5. Write pseudo-code for implementing Gift wrapping/Incremental in 3D.

   A big question will be: what sort of structure do you need to store the hull? At the very least the hull will need to store its faces (an array/vector of faces on the hull). Ideally the faces use pointers to points in P, rather than duplicating the points and their coordinates, see below.

   For example:

```
typedef struct _face3d {
   point3d *p1, *p2, *p3;  //the vertices on this face (in ccw as looking from outside)
} face3d;
```

   In addition to the faces, you may need to keep track of what vertices are on the hull, what edges are on the hull, and whether an edge has found both its adjacent faces or only one of them. Come up with structures to kep track of these. Consider using hash tables and maps and trees and such, and avoid storing a topological structure for the hull (to avoid the complexity or programming).

   Start by writing high-level pseudo-code. For example you may want to keep 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).