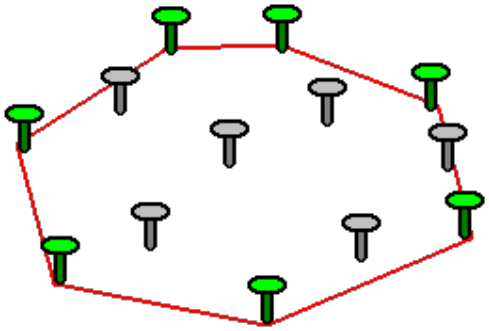
Recall

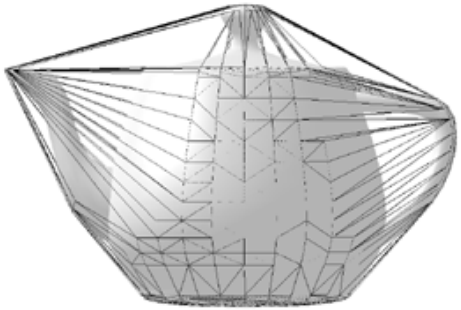
Given points in d-dimensional space, find a good "container" = convex polytope.
Many applications, e.g. collision detection, pattern recognition, motion planning . . .

In 2D, imagine putting a rubber band around the points



https://brilliant.org/wiki/convex-hull/

In 3D, wrap with shrink-wrap



Newton Collision Convex Hull

More formally:

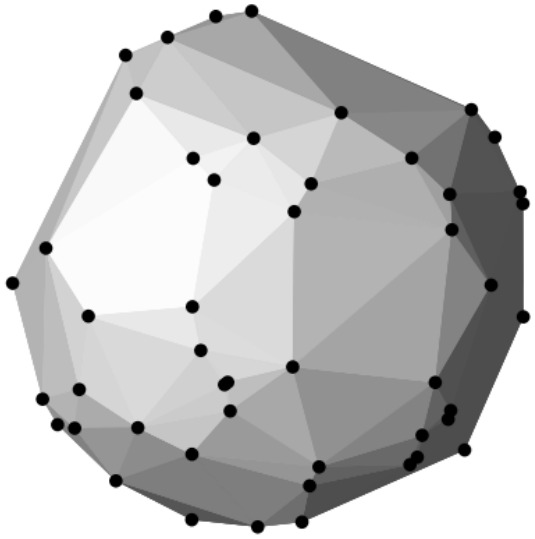
A set is ***convex*** if for every two points *p*, *q* in the set, all points on the line segment *pq* are also in the set.

The ***convex hull*** of set *S* is the intersection of all convex sets that contain *S*.

Note that the convex hull of S is convex.
The fact that the convex hull of a set of points S is a convex polytope whose vertices are points of S requires a proof, which we will do later.

Plan for the next two lectures

- definitions of convex hull in any dimension, and more about convex polyhedra
- divide and conquer for convex hull in 3D
- randomized algorithm for convex hull in any dimension



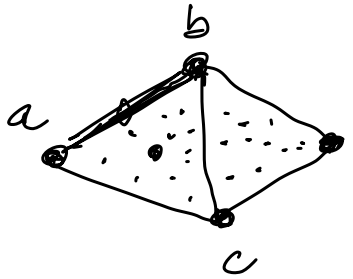
Equivalent definitions of ***Convex Hull*** of a set of points S

1. intersection of all *convex sets* containing S
2. intersection of all half-spaces containing S
3. all *convex combinations* of points in S
4. A *convex polyhedron* P (polygon in 2D) with vertices from S and such that all points of S are inside P

Recall: A set C is ***convex*** if for all points p, q in C, the line segment pq is in C.

A ***convex combination*** of points p1, p2, . . , pn is

$$\sum_{i=1}^{n} \lambda_i p_i \text{ for } \sum_{i=1}^{n} \lambda_i = 1, \ \lambda_i \geq 0$$

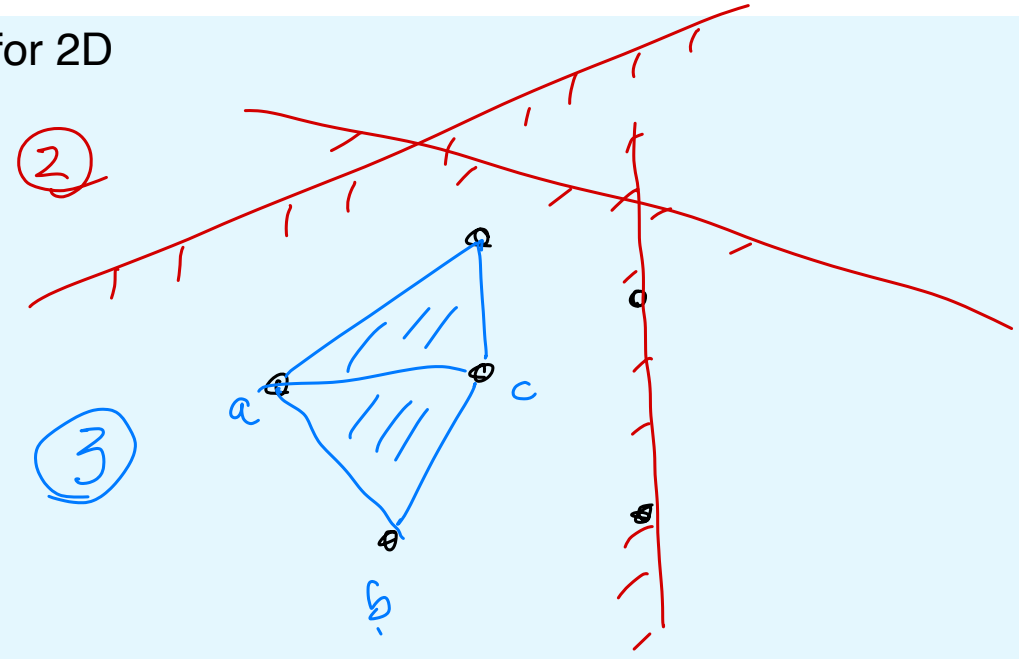


For 2D in the last lecture, we used 1 and 4.

Equivalent definitions of ***Convex Hull*** of a set of points S

1. intersection of all *convex sets* containing S
2. intersection of all half-spaces containing S
3. all *convex combinations* of points in S
4. A *convex polyhedron* P (polygon in 2D) with vertices from S and such that all points of S are inside P

Illustration for 2D



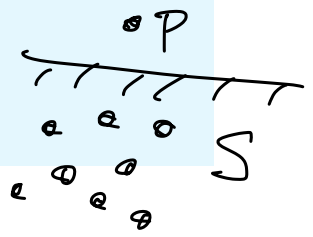
Equivalent definitions of ***Convex Hull*** of a set of points S

1. intersection of all *convex sets* containing S
2. intersection of all half-spaces containing S
3. all *convex combinations* of points in S

How are these related?

We can think of (2) and (3) as positive and negative characterization. NP and co-NP

- to show that p is IN the convex hull
  show how p is a convex combination of pts in S
- to show p is OUT of the convex
  show a half-space separating p from S



Equivalence of 2 and 3 is proved using some version of Farkas's Lemma

either p is a convex combination of points of S
OR
there is a plane separating p from S
AND NOT BOTH

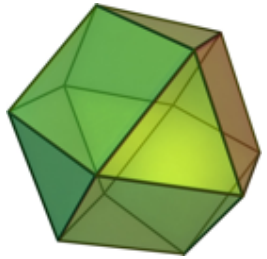
This is a fundamental result for Linear Programming. https://en.wikipedia.org/wiki/Farkas%27_lemma

Equivalent definitions of ***Convex Hull*** of a set of points S

1. intersection of all *convex sets* containing S
2. intersection of all half-spaces containing S
3. all *convex combinations* of points in S
4. A *convex polyhedron* P (polygon in 2D) with vertices from S and such that all points of S are inside P

A ***convex polyhedron*** (in dimension *d*) is a bounded intersection of half-spaces

$$\{x \in \mathbb{R}^d : Ax \leq b\}, \quad A = m \times d \text{ matrix}, b = m \text{ vector}$$



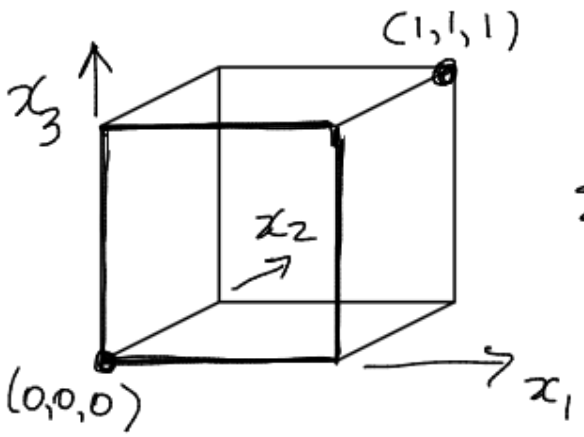
Caution: the term "polyhedron" means different things in different areas (convex/non-convex, bounded/unbounded)

A polyhedron has vertices, edges, faces, . . .

2D

3D

Face of a polyhedron



A cube in 3D is the intersection of 6 half-spaces

$$\{(x_1, x_2, x_3) : 0 \le x_1 \le 1,\ 0 \le x_2 \le 1,\ 0 \le x_3 \le 1\}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \le \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

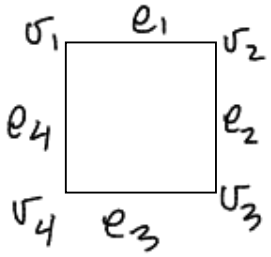
A face is $\{x \in \mathbb{R}^d : Ax \le b\}$ and some inequalities are changed to equalities.

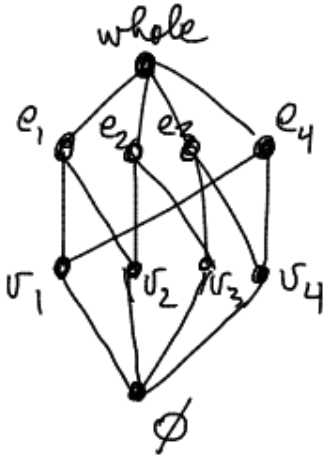
e.g. front face of cube has $x_2 = 0$.

back face $x_2 = 1$

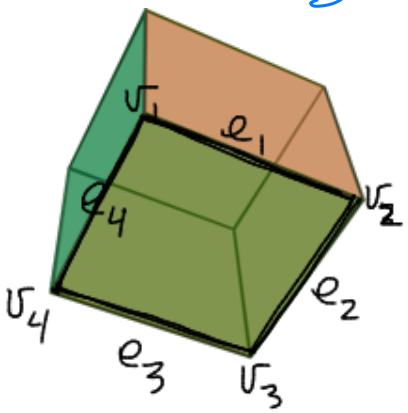
The face lattice of a convex polyhedron

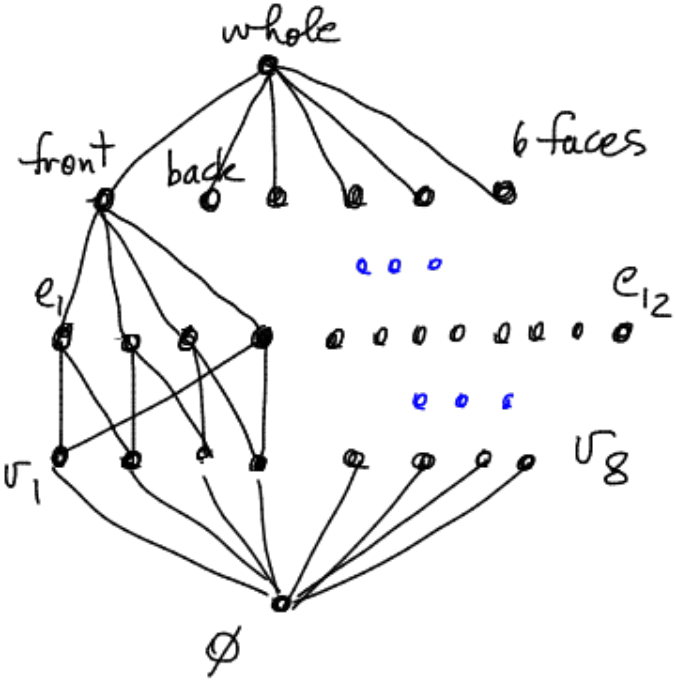
d=2



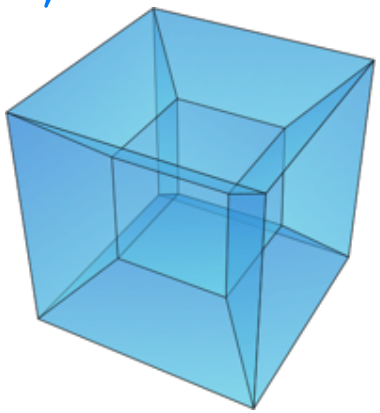


d=3

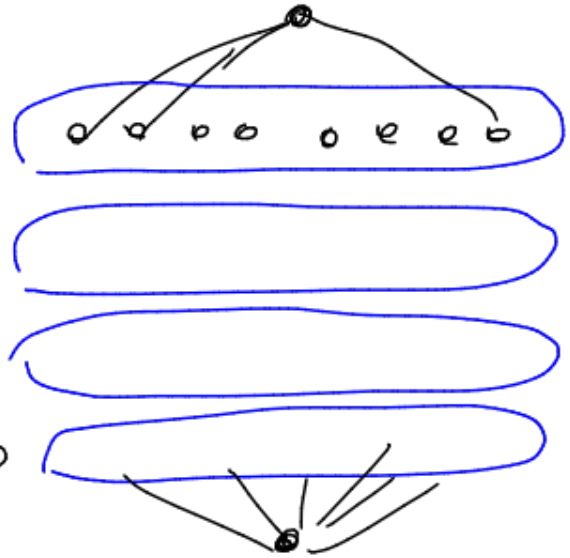




d=4



8 cubes
24 squares
32 edges
16 vertices



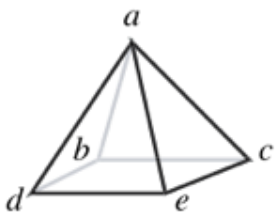
Ex. Write down formula.

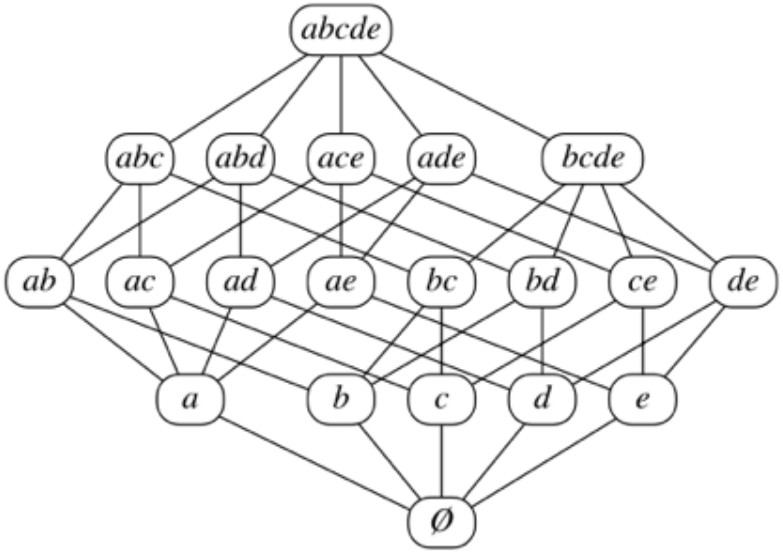
(x_1, x_2, x_3, x_4)

x_i: 0 or 1

2^4

The face lattice of a convex polyhedron





i-face = i dimensional face

0-face = vertex

1-face = edge

e

o

(d-1)-face = facet

d-face = the whole polyhedron.

Equivalent definitions of ***Convex Hull*** of a set of points S

1. intersection of all *convex sets* containing S
2. intersection of all half-spaces containing S
3. all *convex combinations* of points in S
4. A *convex polyhedron* P (polygon in 2D) with vertices from S and such that all points of S are inside P

Equivalence of these definitions proved using:

Theorem [Minkowski, Weyl] The set of all convex combinations of p1, . . . , pn is a bounded convex polyhedron whose vertices are a subset of p1, . . . , pn

The convex hull problem

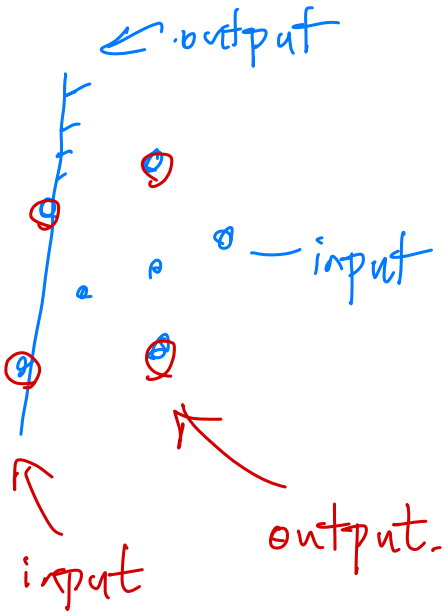
Given a set of n points in d-dimensions, find their convex hull (as an intersection of half-spaces)

or sometimes we ask for the whole face lattice

Dual problem:

Given a set of m halfspaces in d-dimensions, find their intersection (as the set of vertices of the polyhedron)

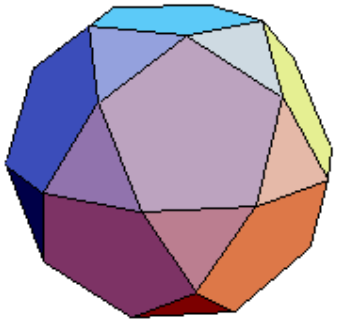
or sometimes we ask for the whole face lattice



In fact, these two problems are the same by a ***duality*** map (more later)

3D convex hull. Given n points in 3D, find their convex hull, i.e. find the vertices, edges and faces of the convex hull.

What is the size of the convex hull?



skeleton

The graph of edges and vertices is planar (*)

Use Euler's formula

(**) $v - e + f = 2$ v = # vertices, e = # edges, f = # faces.

$v \le n$

$2e \ge 3f$ so $f \le \frac{2}{3}e$

plug into (**) $2 = v - e + f \le v - e + \frac{2}{3}e = v - \frac{1}{3}e$

$e \le 3(n-2)$ $f \le 2(n-2)$

All of v, e, f are $O(n)$

So whole face lattice has size $O(n)$.

2D 3D 4D 5D
$O(n)$ $O(n)$ $O(n^2)$??

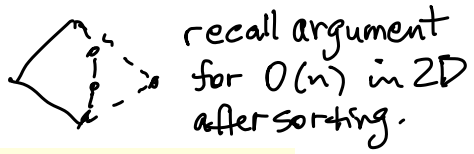
what about 4D?

(*) converse: any 3-connected planar graph is skeleton of convex polygon. Steinitz.

https://www.ics.uci.edu/~eppstein/junkyard/euler/

3D Convex Hull Algorithms

Some of the 2D algorithms extend to 3D.



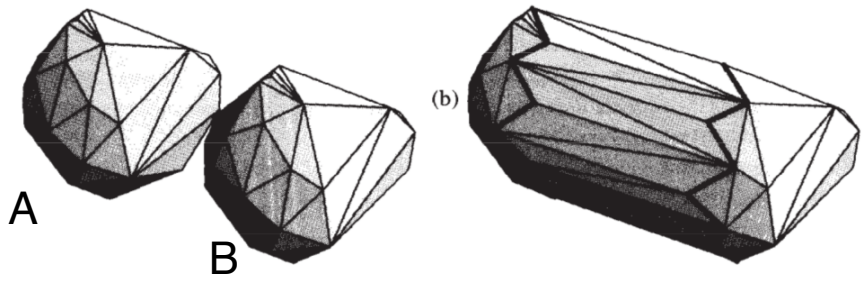
Exercise: Does the incremental algorithm extend? Is it O(n log n)?

Divide and Conquer.

Basically the ONLY known O(n log n) 3D convex hull algorithm.
Preparata and Hong 1977.

- sort points by x coordinate
- divide by orthogonal plane at median x coordinate into two sets of size n/2
- recurse on each side to find convex hulls A and B
- combine A and B into one convex hull

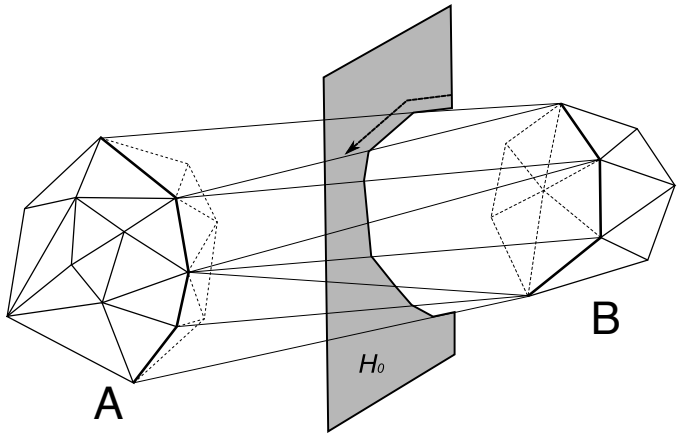
If we combine in O(n)
we get T(n) = 2T(n/2) + O(n)
which yields T(n) = O(n log n)



O'Rourke, Comp. Geom. in C

How to combine two disjoint convex hulls in O(n)

we must find the "band" of faces
that cross our dividing plane
and then discard "hidden" faces

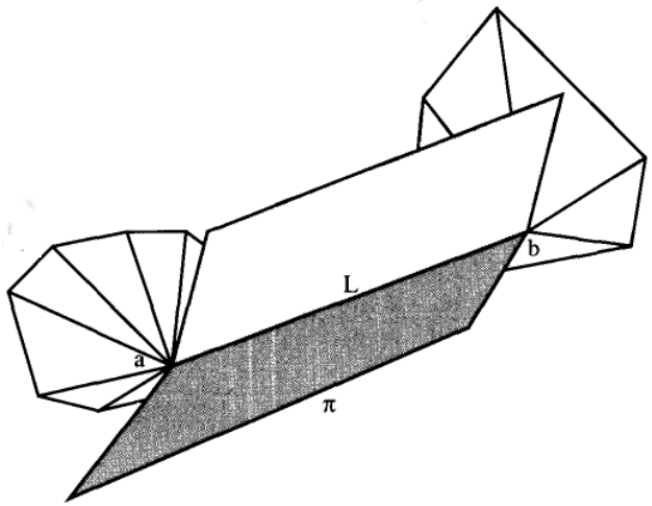


1. find an edge ab of the convex hull, a in A, b in B and a plane through ab with A and B to one side.

2. pivot the plane through ab to find a face of the convex hull band

3. repeat until we wrap back to ab

4. remove hidden faces



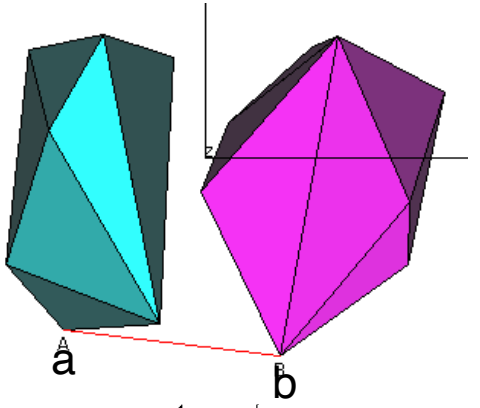
O'Rourke, Comp. Geom. in C

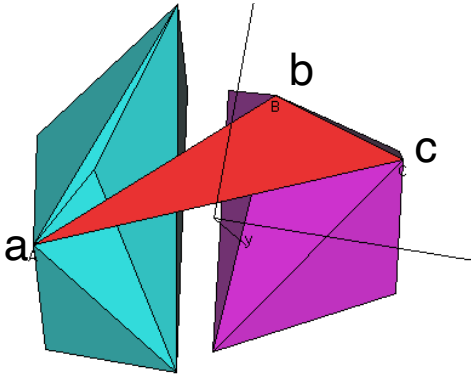
1. find an edge ab of the convex hull, a in A, b in B and a plane through ab with A and B to one side.

2. pivot the plane through ab to find a face of the convex hull band

3. repeat until we wrap back to ab

4. remove hidden faces





Details and Timing

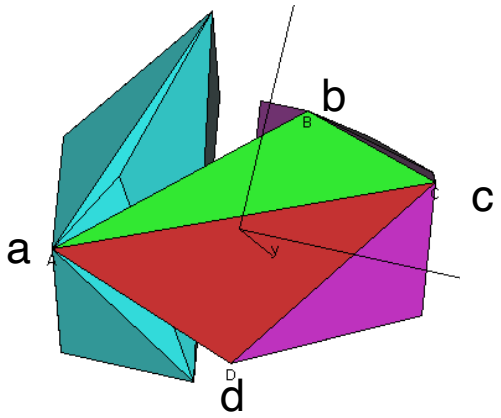
Step 1. Project to 2D and find lower bridge

Step 2.
Lemma. The next point, c, is a neighbour of a or b.

So, find "best" neighbour of a and "best" neighbour of b.

Lemma. If the next point, c, is a neighbour of b, then the next "best" neighbour of a remains the same.

Total time: Sum of all vertex degrees. This is O(n) because we have a planar graph.

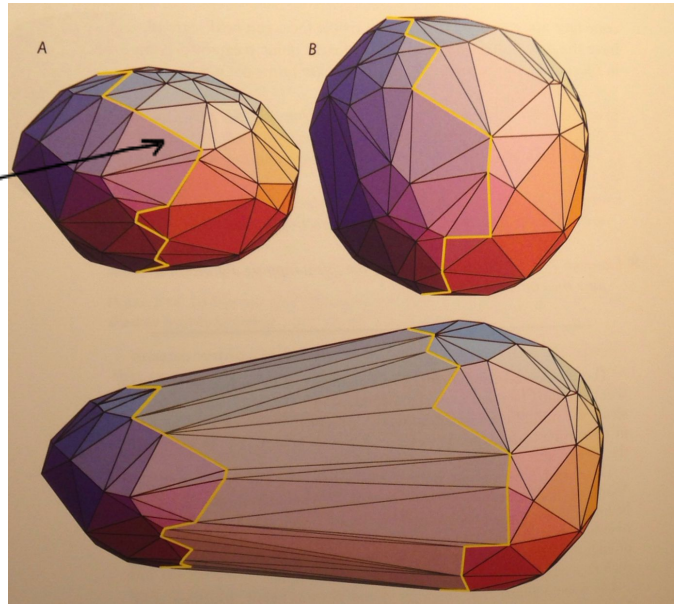


Roger Hernando

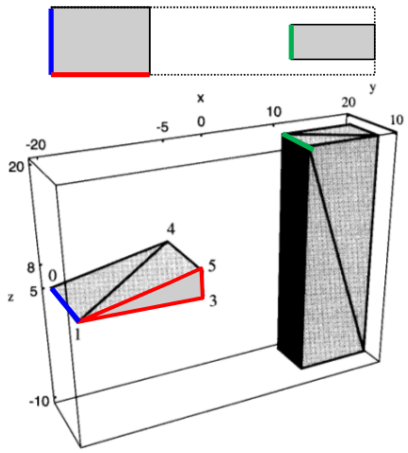
Removing hidden faces can be done in O(n) too.

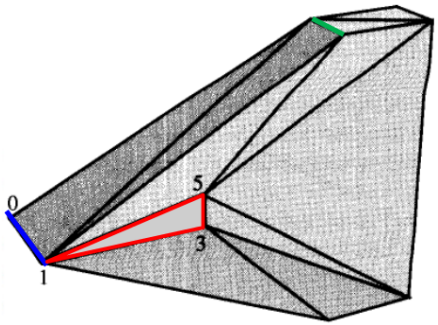
Note that the cycle of "horizon" edges need not be simple

this "horizon" is a simple cycle



O'Rourke and Devadoss





O'Rourke, Comp. Geom. in C

Here the horizon is not a simple cycle

topology of the horizon

Conventional wisdom was that the divide and conquer algorithm is hard to implement
e.g. see O'Rourke's book, "Computational Geometry in C", 1998.

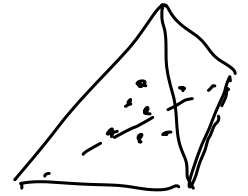
However, there are now good implementations
e.g. "A Minimalist's Implementation of the 3-d Divide-and-Conquer Convex Hull Algorithm", Timothy Chan, 2003. https://cs.uwaterloo.ca/~tmchan/ch3d/ch3d.pdf

Appendix: Complete Code

```
// Timothy Chan    "ch3d.cc"    12/02    3-d lower hull (in C++)

// a simple implementation of the O(n log n) divide-and-conquer algorithm

//   input: coordinates of points
//     n x_0 y_0 z_0 ... x_{n-1} y_{n-1} z_{n-1}

//   output: indices of facets
//     i_1 j_1 k_1 i_2 j_2 k_2 ...

//   warning: ignores degeneracies and robustness
//   space: uses 6n pointers


#include <stream.h>

struct Point {
  double x, y, z;
  Point *prev, *next;
  void act() {
    if (prev->next != this) prev->next = next->prev = this;  // insert
    else { prev->next = next; next->prev = prev; }  // delete
  }
};

const double INF = 1e99;
static Point nil = {INF, INF, INF, 0, 0};
Point *NIL = &nil;

inline double turn(Point *p, Point *q, Point *r) {  // <0 iff cw
  if (p == NIL || q == NIL || r == NIL) return 1.0;
  return (q->x-p->x)*(r->y-p->y) - (r->x-p->x)*(q->y-p->y);
}

inline double time(Point *p, Point *q, Point *r) {  // when turn changes
  if (p == NIL || q == NIL || r == NIL) return INF;
  return ((q->x-p->x)*(r->z-p->z) - (r->x-p->x)*(q->z-p->z)) / turn(p,q,r);
}

Point *sort(Point P[], int n) {  // mergesort

  Point *a, *b, *c, head;

  if (n == 1) { P[0].next = NIL; return P; }
  a = sort(P, n/2);
  b = sort(P+n/2, n-n/2);
  c = &head;
  do
    if (a->x < b->x) { c = c->next = a; a = a->next; }
    else { c = c->next = b; b = b->next; }
  while (c != NIL);
  return head.next;
}

void hull(Point *list, int n, Point **A, Point **B) {  // the algorithm

  Point *u, *v, *mid;
  double t[6], oldt, newt;
  int i, j, k, l, minl;

  if (n == 1) { A[0] = list->prev = list->next = NIL; return; }

  for (u = list, i = 0; i < n/2-1; u = u->next, i++) ;
  mid = v = u->next;
  hull(list, n/2, B, A);  // recurse on left and right sides
  hull(mid, n-n/2, B+n/2*2, A+n/2*2);

  for ( ; ; )  // find initial bridge
    if (turn(u, v, v->next) < 0) v = v->next;
    else if (turn(u->prev, u, v) < 0) u = u->prev;
    else break;

  // merge by tracking bridge uv over time
  for (i = k = 0, j = n/2*2, oldt = -INF; ; oldt = newt) {
    t[0] = time(B[i]->prev, B[i], B[i]->next);
    t[1] = time(B[j]->prev, B[j], B[j]->next);
    t[2] = time(u, u->next, v);
    t[3] = time(u->prev, u, v);
    t[4] = time(u, v->prev, v);
    t[5] = time(u, v, v->next);
    for (newt = INF, l = 0; l < 6; l++)
      if (t[l] > oldt && t[l] < newt) { minl = l; newt = t[l]; }
    if (newt == INF) break;
    switch (minl) {
    case 0:  if (B[i]->x < u->x) A[k++] = B[i];  B[i++]->act();  break;
    case 1:  if (B[j]->x > v->x) A[k++] = B[j];  B[j++]->act();  break;
    case 2:  A[k++] = u = u->next;  break;
    case 3:  A[k++] = u;  u = u->prev;  break;
    case 4:  A[k++] = v = v->prev;  break;
    case 5:  A[k++] = v;  v = v->next;  break;
    }
  }
  A[k] = NIL;

  u->next = v;  v->prev = u;  // now go back in time to update pointers
  for (k--; k >= 0; k--)
    if (A[k]->x <= u->x || A[k]->x >= v->x) {
      A[k]->act();
      if (A[k] == u) u = u->prev; else if (A[k] == v) v = v->next;
    }
    else {
      u->next = A[k]; A[k]->prev = u; v->prev = A[k]; A[k]->next = v;
      if (A[k]->x < mid->x) u = A[k]; else v = A[k];
    }
}

main() {

  int n, i;
  cin >> n;

  Point *P = new Point[n];  // input
  for (i = 0; i < n; i++) { cin >> P[i].x; cin >> P[i].y; cin >> P[i].z; }

  Point *list = sort(P, n);
  Point **A = new Point *[2*n], **B = new Point *[2*n];
  hull(list, n, A, B);

  for (i = 0; A[i] != NIL; A[i++]->act())  // output
    cout << A[i]->prev-P << " " << A[i]-P << " " << A[i]->next-P << "\n";
  delete A;  delete B;  delete P;
}
```

The Gift-wrapping algorithm extends to 3D, O(nh), h = # faces of the convex hull.

Use the same kind of "wrapping" we just saw for divide and conquer.

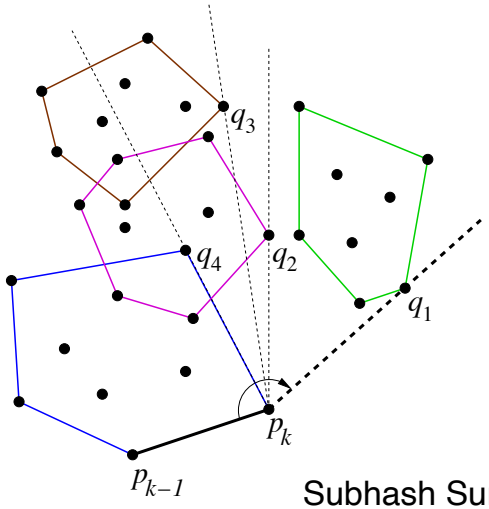


h can be << n

Timothy Chan's O(n log h) algorithm extends to 3D.

Recall it needs an O(n log n) algorithm (the divide and conquer algorithm) plus an O(nh) algorithm (the gift-wrapping algorithm).

The step of finding the "extreme" point in each of the smaller convex hulls needs more detail.



Subhash Suri

Summary

- mathematical notions of convex hull
- algorithms for Convex Hull in 3D

References

- [CGAA] Chapter 11
- [O'Rourke] Chapter 4

Next: randomized algorithm for convex hull in higher dimensions.