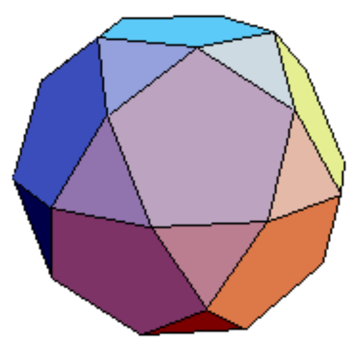
Next: Algorithms for convex hull in 3D and higher dimensions.

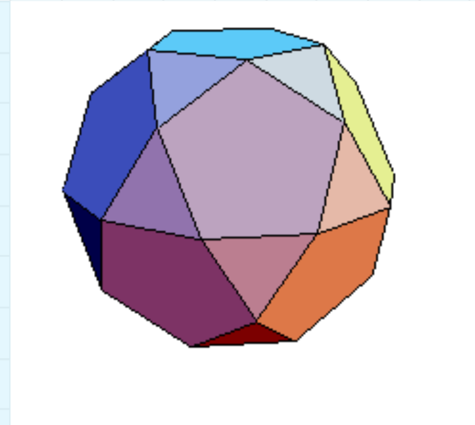
Given n points in 3D, find their convex hull, i.e. find the vertices, edges and faces of the convex hull.

What is the size of the convex hull?

In 3D, n vertices, how many edges and faces?



3D convex hull. n vertices, how many edges and faces?



use Euler's formula
the graph is planar.*

$v - e + f = 2$, $v \le n$

vertices edges faces

count # edge-face incidences

$2e \ge 3f$ — use that the graph comes from polyhedron

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

$2 = v - e + f \le v - e + \frac{2}{3}e = v - \frac{1}{3}e$

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

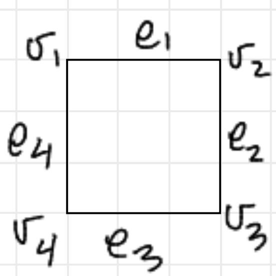
all of e, f are $O(n)$

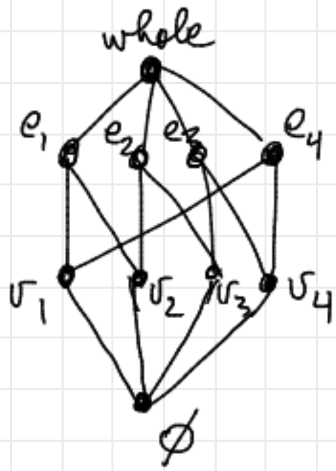
d	2	3	4	5	. . .
# face	$O(n)$	$O(n)$	$O(n^2)$	$O(n^2)$	. . . .

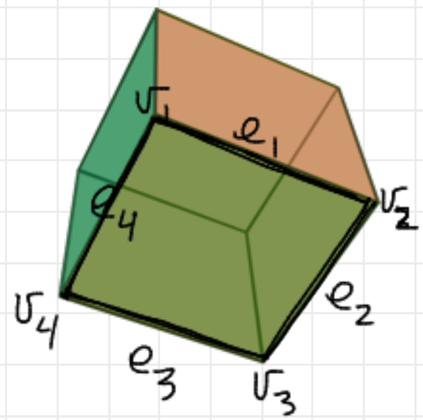
* converse: Steinitz's thm: any 3-connected planar graph is the graph of a convex polyhedron.

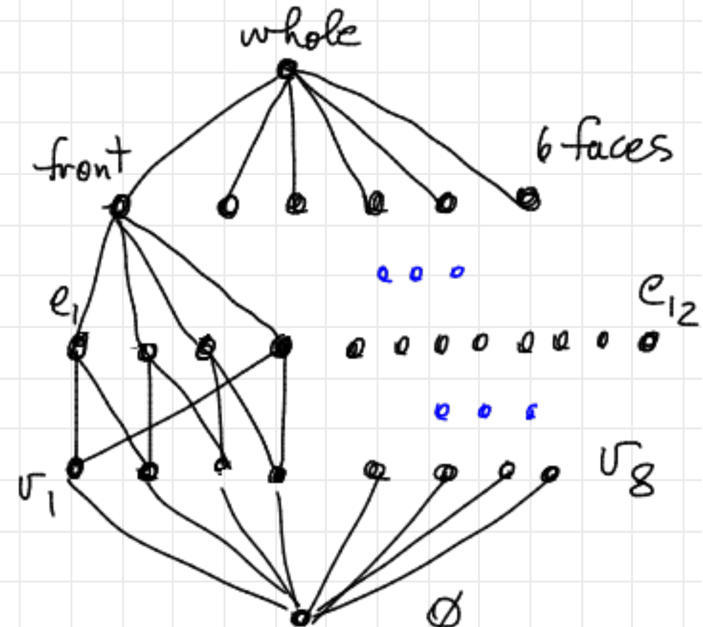
Recall

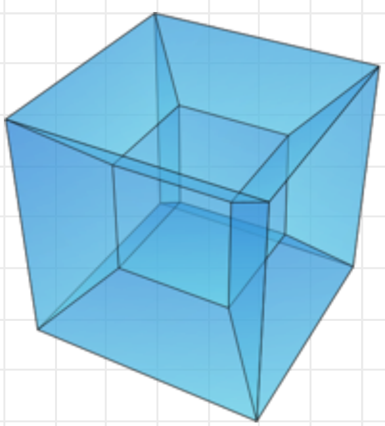
The face lattice of a convex polyhedron









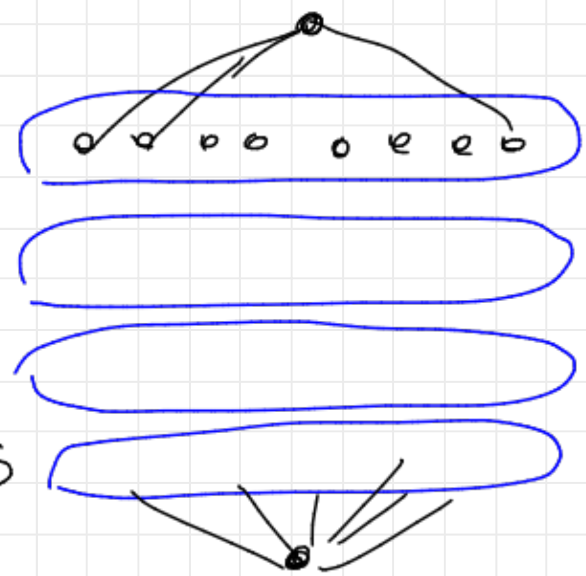


8 cubes

24 squares

32 edges

16 vertices



Size of convex hull of n points in d-dimensions

McMullen's Upper bound Theorem
For a convex polyhedron in *d* dimensions (*d* fixed) with *n* vertices the worst case number of faces is

$$\Theta(n^{\lfloor d/2 \rfloor})$$

The number of facets has the same bound (we get a 2^d constant appearing).

In fact, McMullen gave more exact bounds — the above asymptotic bound is easier to show https://graphics.stanford.edu/courses/cs268-11-spring/notes/upper_bound_theorem.pdf

For $d = 2,3$ bound is $\Theta(n)$.
For $d = 4,5$ bound is $\Theta(n^2)$

$d = 4$ matters! One application of 4D convex hull is to find 3D Delaunay triangulations.

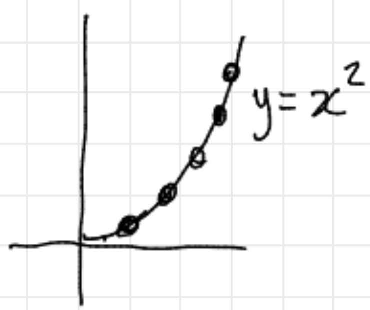
The bound of $\Theta(n^{\lfloor d/2 \rfloor})$ is realized by a ***cyclic polytope*** — the convex hull of *n* points on the ***moment curve***

moment curve = $\{(t, t^2, \ldots, t^d) : t \in \mathbb{R}\}$

Place *n* points on the moment curve. $t_1 \leq t_2 \leq \cdots \leq t_n$

Claim. The number of facets of their convex hull is $\Theta(n^{\lfloor d/2 \rfloor})$

in 2D



in 4D

Can prove:

- every pair $t_i t_j$ gives an edge of the CH, so #edges is $\Theta(n^2)$

- every 4-tuple $t_i t_{i+1} t_j t_{j+1}$ gives a facet of the CH, so #facets is $\Theta(n^2)$

incremental in 2D

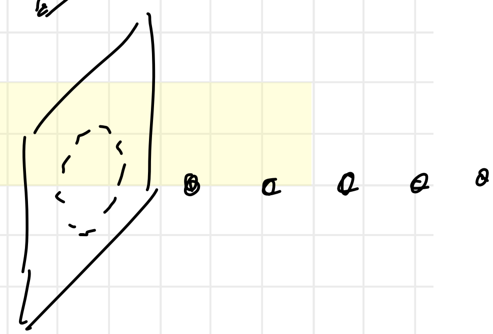
3D Convex Hull Algorithms

Some of the 2D algorithms extend to 3D.

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

sort by x.

No, it is $\Theta(n^2)$ worst case,

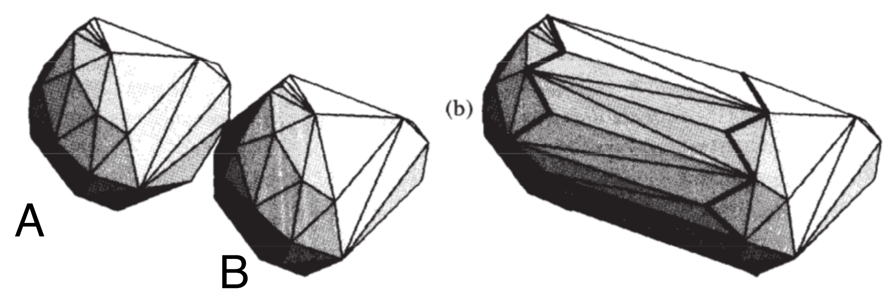


Divide and Conquer.

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

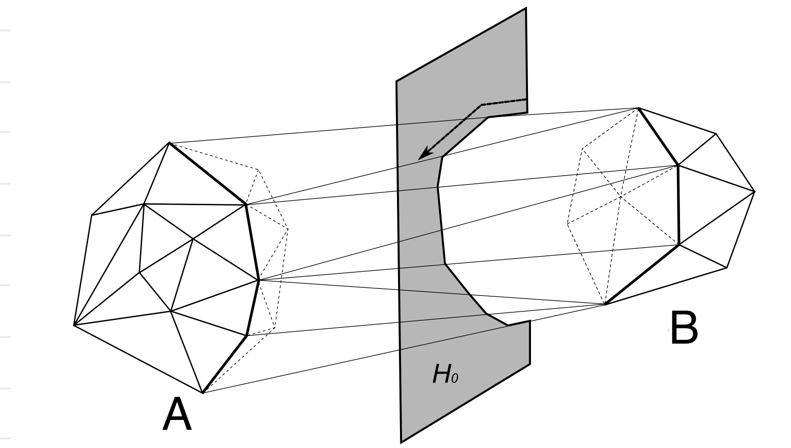
If we combine in O(n)
we get $T(n) = 2T(n/2) + O(n)$
which yields $T(n) = O(n \log n)$



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

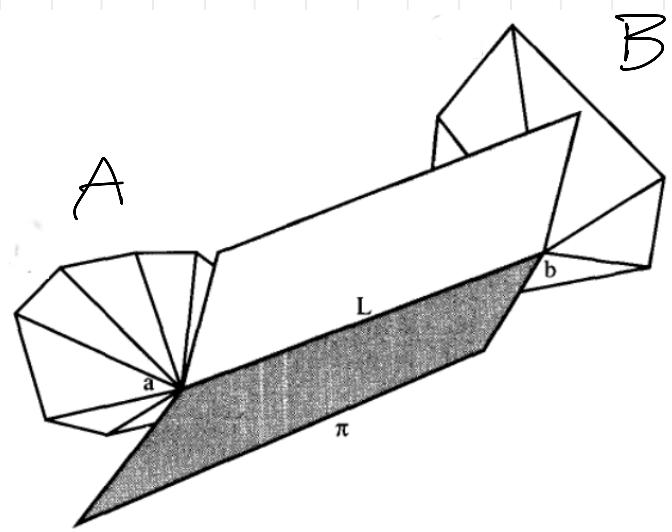


1. find an edge ab of the convex hull, a in A, b in B and a plane through ab with A and B to one side.

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

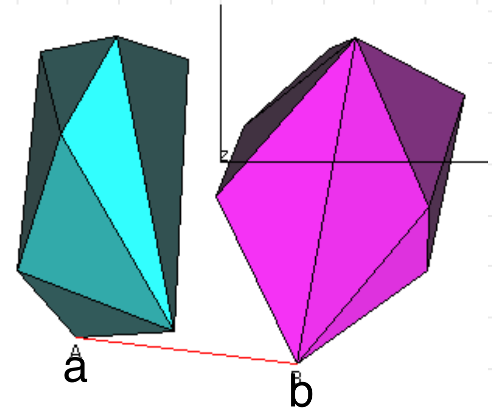
maybe assume no 4 points lie on plane.

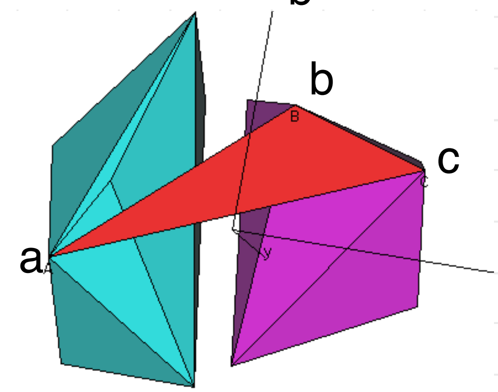
1. find an edge ab of the convex hull, a in A, b in B and a plane through ab with A and B to one side.

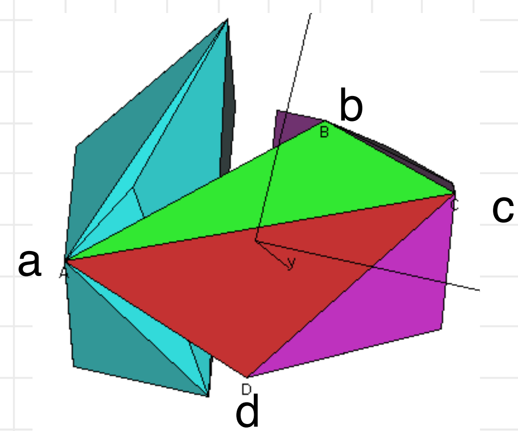
2. pivot the plane through ab to find a face of the convex hull band

3. repeat until we wrap back to ab

4. remove hidden faces







Roger Hernando

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.
by angle = sidedness test

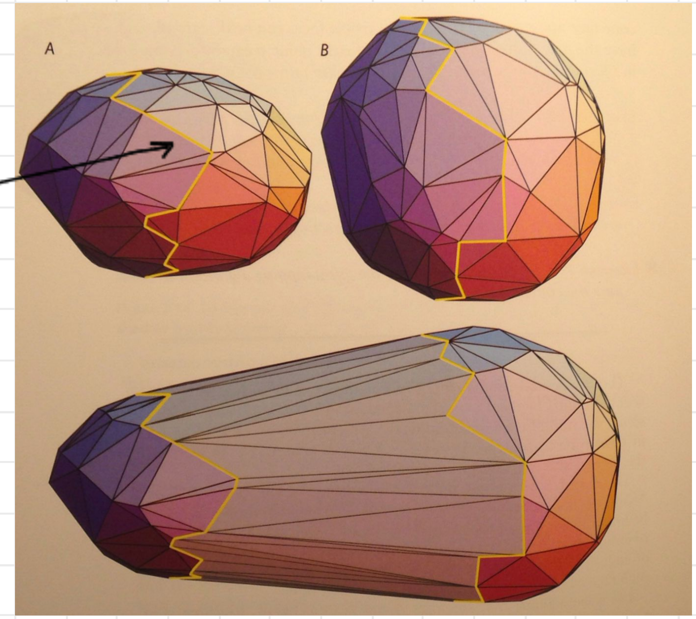
Lemma. If the next point, c, is a neighbour of b, then the next "best" neighbour of a remains the same.

Total time: Sum of all vertex degrees. This is O(n) because we have a planar graph.
= 2e

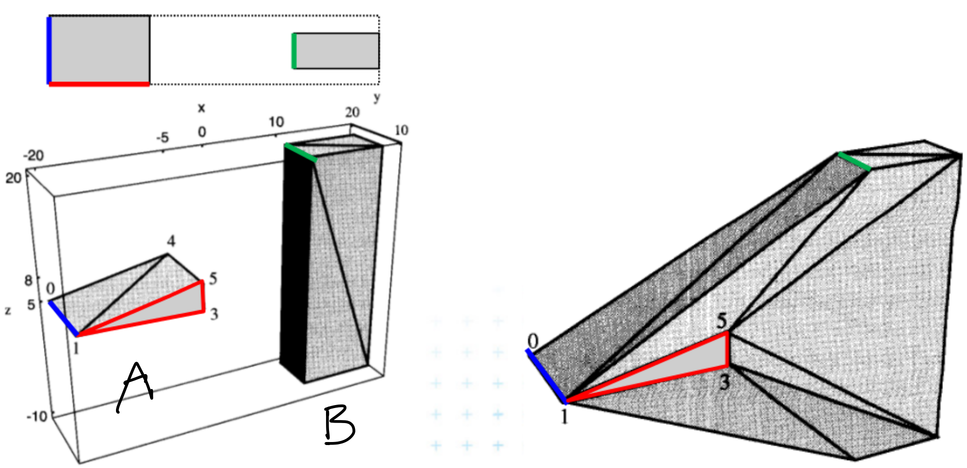
Removing hidden faces can be done in O(n) too.

Note that the cycle of “horizon” edges need not be simple

this "horizon" is a simple cycle

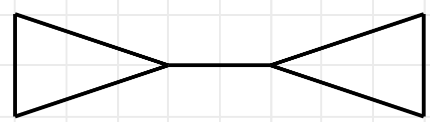


O’Rourke and Devadoss



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;

```

11

```
  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;
}
```

12

Note: CGAL has implementations (see course web page).

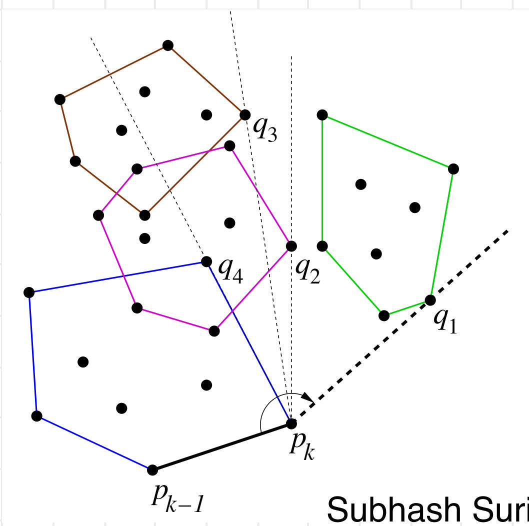
The Gift-wrapping algorithm extends to 3D, O(nh), h = # faces of the convex hull.

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

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.



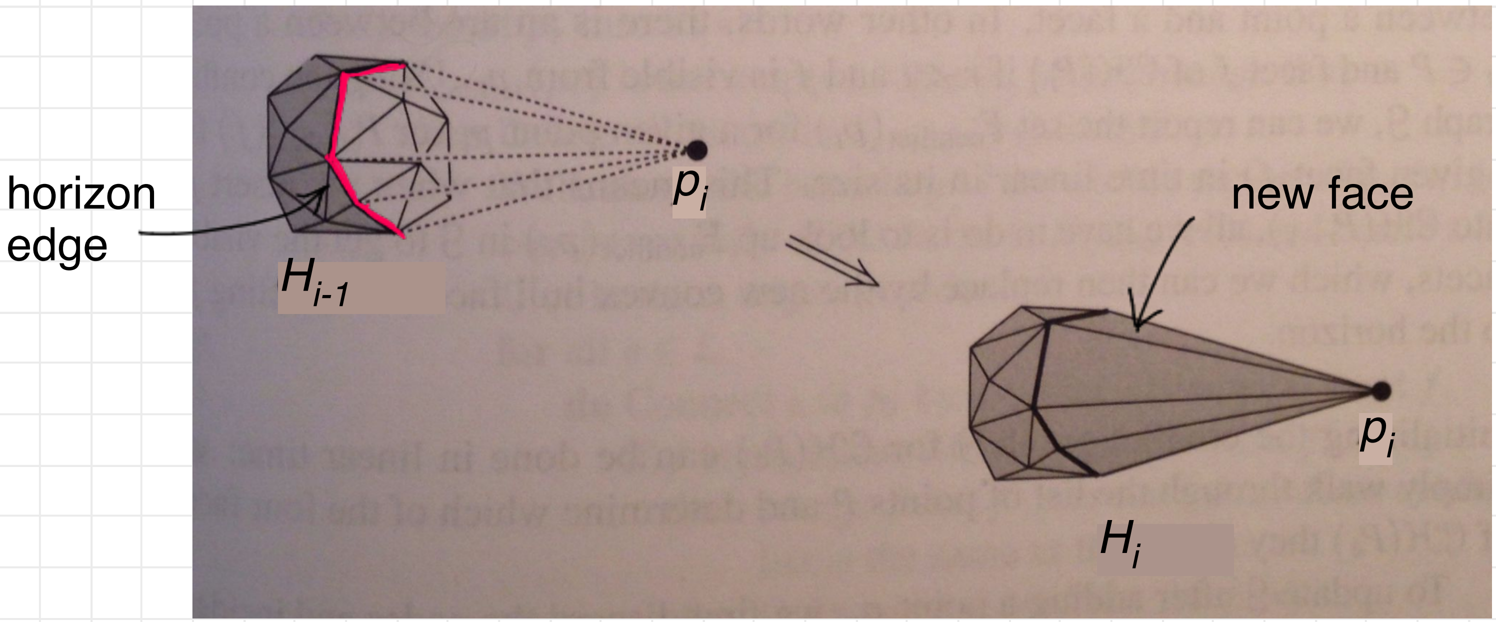
Subhash Suri

[Randomized] Incremental Convex Hull Algorithm

We will describe the algorithm for 3D though it does extend to general dimensions.

Assume no 4 points lie on a plane (this means that all faces will be triangles).
See [CGAA] book for details on more general case.

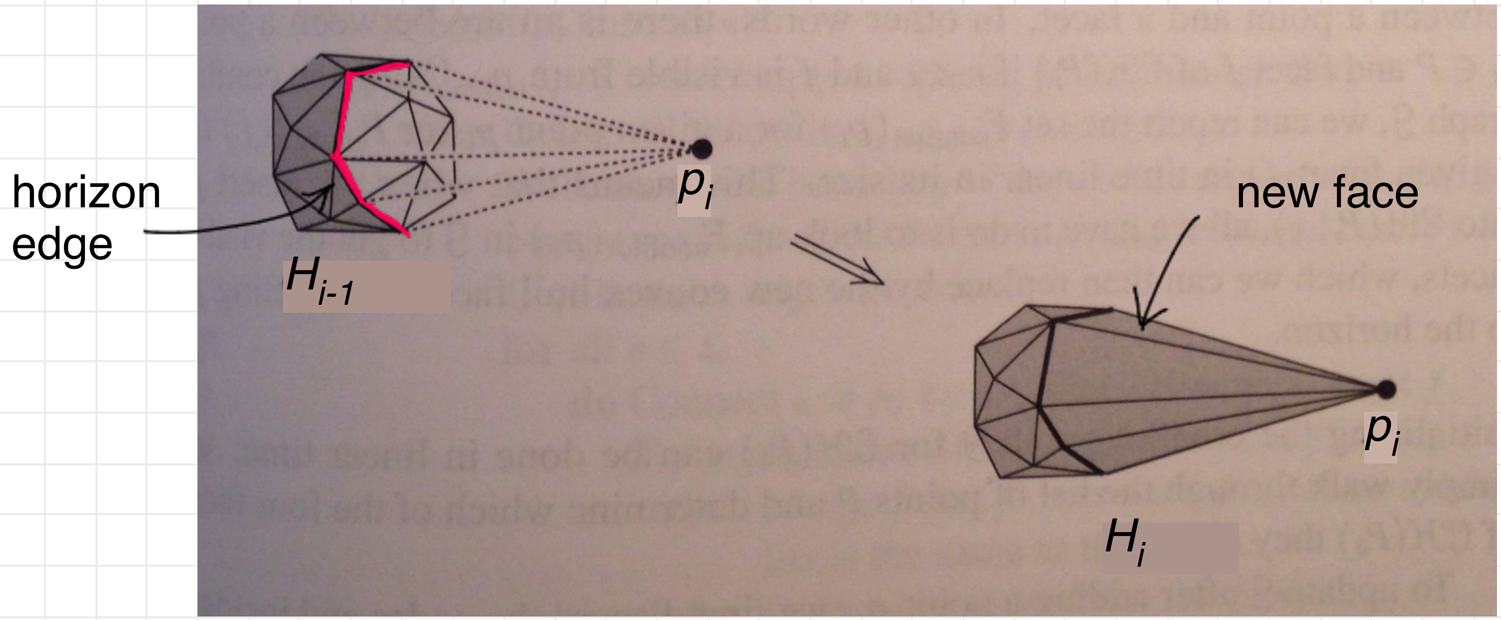
Idea: Add the points one by one in random order.



[CGAA]

Goal: Give details and prove expected runtime is O(n log n).

Idea: Add the points one by one in random order.



[CGAA]

Imagine the new point p_i as a light source.

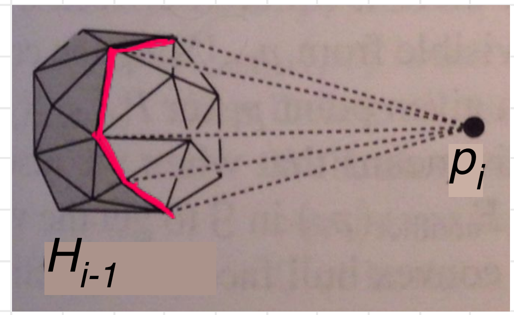
Some faces are ***lit*** — we remove those.
Some faces are ***dark*** — we keep those.

A ***horizon edge*** is incident to one lit face and one dark face.
New faces join p_i to horizon edges.

Note that p_i is not in the plane of a face of H_{i-1} by our general position assumption.

Algorithm

1. take a random order of the points $p_1 \ldots p_n$
2. form a tetrahedron H_4 on $p_1\ p_2\ p_3\ p_4$
3. for $i = 5 \ldots n$
4. if p_i is inside H_{i-1} do nothing else
5. locate a face F lit by p_i
6. find and delete all faces lit by p_i
7. for each horizon edge, make a new face to p_i



Straight-forward implementation and analysis:

worst case lines 4-7 take $O(n)$

total is $O(n^2)$, and we saw example that gives $\Theta(n^2)$ in worst case.

Ex: Find an example (with bad ordering of points) where the algorithm takes $\Theta(n^2)$

Theorem. For points in random order the expected run time is $O(n \log n)$.

3. for $i = 5 \ldots n$
4. if p_i is inside H_{i-1} do nothing else
5. locate a face F lit by p_i
6. find and delete all faces lit by p_i
7. for each horizon edge, make a new face to p_i

Deal separately with lines 6 - 7 and lines 4 - 5.

Lines 6 - 7

6 find all lit faces using DFS (depth first search) starting from F

time: O(# lit faces)

- a face is deleted at most once, so work spent on finding & deleting lit faces can be charged to cost of adding new faces

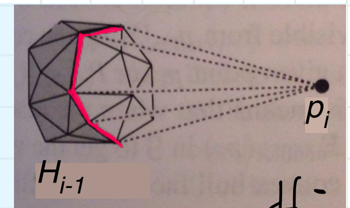
A_i = # faces added in iteration i

total work (steps 6-7) is $\sum A_i$

Lemma. [Clarkson, Shor 1989] In 3D, if points are added in random order then the expected value of A_i is O(1). A_i = # faces added in iteration i.

So expected value of $\sum A_i = O(n)$

Proof.



H_i

Backwards Analysis.

Consider $H_i = CH(\{p_1 \cdots p_i\})$

faces A_i = degree of p_i

* p_i is equally likely to be any vertex of H_i

So we need average degree of vertices of H_i

H_i has planar graph (vertices & edges)

Claim. Average vertex degree in planar graph is < 6

pf. (in case you don't know)

sum of degrees is $2e$

$e = 3(v-2)$ from previous slide Euler + fact that faces are triangles

avg. degree $< \frac{6v}{v} = 6$

Theorem. For points in random order the expected run time is O(n log n).

3. for $i = 5 \, . \, . \, n$
4. if p_i is inside H_{i-1} do nothing else
5. locate a face F lit by p_i
6. find and delete all faces lit by p_i
7. for each horizon edge, make a new face to p_i

Deal separately with lines 6 - 7 and lines 4 - 5.

Lines 6 - 7: expected run time (over whole algorithm) is O(n)

Lines 4 - 5: We must maintain some kind of search structure to get O(n log n) total.

Various approaches:

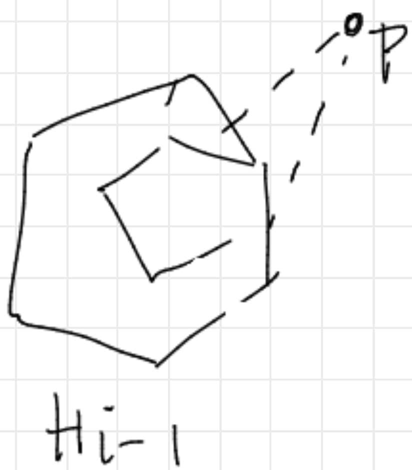
- Clarkson, Shor 1989 — ***conflict graph***

- Seidel 1991 — it's linear programming and for $d > 3$, this gives a good solution but for $d = 3$ we get back to O(n^2)

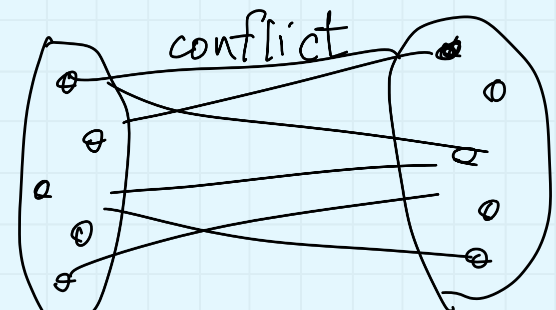
Test if point p_i is inside/outside H_{i-1} (and if outside, find a lit face F)

Maintain ***conflict graph***

conflict = a face f of H_{i-1} is lit by an unprocessed point p



Store conflicts as a bipartite graph



Test if point p_i is inside/outside H_{i-1} (and if outside, find a lit face F)

$O(1)$ {
p_i inside = p_i has no edges in conflict graph
p_i outside – conflict edges incident to p_i gives us the lit face F.

What's left:
- how to update the conflict graph
- expected case analysis

} skip this

this is where the work happens.

Randomized Incremental Convex Hull Algorithm

- expected run time O(n log n) in 3D

- expected run time for d >= 4 is $O(n^{\lfloor d/2 \rfloor})$

  and we can use linear programming instead of the conflict graph

Recall: size of convex hull (facets or whole face lattice) is $\Theta(n^{\lfloor d/2 \rfloor})$

Combining lower bound for d = 2 and lower bound due to output size, we get lower bound of

$$\Omega(n \log n + n^{\lfloor d/2 \rfloor}) \qquad \text{for } d \text{ constant}$$

So randomized incremental algorithm is optimal (it achieves the lower worst case bound).

Is there a deterministic (non-randomized) algorithm?

Yes. Chazelle '93 by derandomizing the above algorithm (choose an order of points that has the good properties of a random order). complicated.

Output sensitive? Lower bound is Θ(h + n log h).
Chan got O(n log h) for d = 2,3. Seidel '86 achieved O(n^2 + h log h) for fixed d.

Summary

- divide and conquer 3D convex hull algorithm

- randomized incremental convex hull algorithm
  $O(n \log n)$ in 3D; optimal $O(n^{\lfloor d/2 \rfloor})$ in dimension d

References

- [CGAA] Chapter 11

- [O'Rourke] Chapter 4