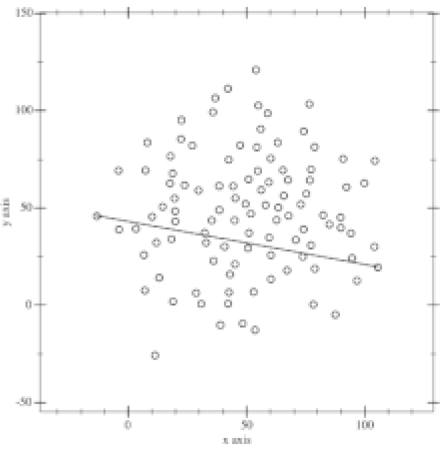
WIKIPEDIA

Quickhull

Quickhull is a method of computing the convex hull of a finite set of points in n-dimensional space. It uses a divide and conquer approach similar to that of quicksort, from which its name derives. Its worst case complexity for 2-dimensional and 3-dimensional space is considered to be $O(n\log(r))$, where n is the number of input points and r is the number of processed points.[1] However, unlike quicksort, there is no obvious way to convert quickhull into a randomized algorithm. Nevertheless, there exist works from Smoothed Analysis which tell us that the 2-dimensional Quick hull algorithm has expected runtime $O(n\log(n))$. Indeed,[2] and related works show that the number of points on the convex hull of any randomly perturbed pointset with Gaussian noise is $O(\sqrt{\log n})$ from which it follows that Quick hull (and many other algorithms) can only takes time $O(n\sqrt{\log n})$ on any set of perturbed points.

N-dimensional Quickhull was invented in 1996 by C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa.[1] It was an extension of Jonathan Scott Greenfield's 1990 planar Quickhull algorithm, although the 1996 authors did not know of his methods.[3] Instead, Barber et al describes it as a deterministic variant of Clarkson and Shor's 1989 algorithm.[1]

Contents

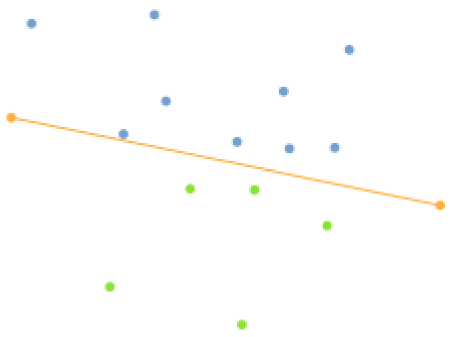


This animation depicts the quickhull algorithm.

Algorithm

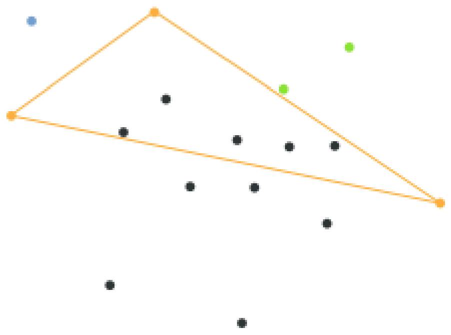
Under average circumstances the algorithm works quite well, but processing usually becomes slow in cases of high symmetry or points lying on the circumference of a circle. The algorithm can be broken down to the following steps:[3]

1. Find the points with minimum and maximum x coordinates, as these will always be part of the convex hull. If many points with the same minimum/maximum x exist, use ones with minimum/maximum y correspondingly.
2. Use the line formed by the two points to divide the set in two subsets of points, which will be processed recursively.
3. Determine the point, on one side of the line, with the maximum distance from the line. This point forms a triangle with those of the line.
4. The points lying inside of that triangle cannot be part of the convex hull and can therefore be ignored in the next steps.
5. Repeat the previous two steps on the two lines formed by the triangle (not the initial line).

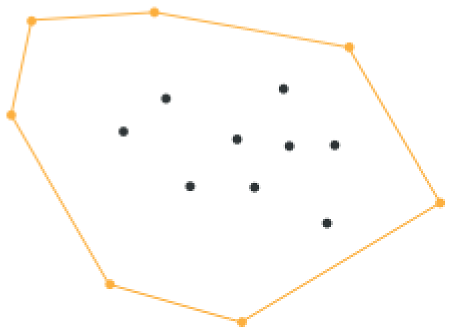

Steps 1-2: Divide points in two subsets

6. Keep on doing so on until no more points are left, the recursion has come to an end and the points selected constitute the convex hull.

The problem is more complex in the higher-dimensional case, as the hull is built from many facets; the data structure needs to account for that and record the line/plane/hyperplane (ridge) shared by neighboring facets too. For *d* dimensions:[1]


Steps 3-5: Find maximal distance point, ignore points inside triangle and repeat it

1. Pick *d* + 1 points from the set that do not share a plane or a hyperplane. This forms an initial hull with facets *Fs[]*.
2. For each *F* in *Fs[]*, find all unassigned points that are "above" it, i.e. pointing away from the center of the hull, and add it to an "outside" set *F.O* associated with *F*.
3. For each *F* with a non-empty *F.O*:
    1. Find the point *p* with the maximum distance from *F*. We will add it to the hull.
    2. Create a visible set *V* and initialize it to *F*. Extend *V* in all directions for neighboring facets *Fv* until no further facets are visible from *p*. *Fv* being visible from *p* means that *p* is above *Fv*
    3. The boundary of *V* then forms the set of horizon ridges *H*.
    4. Let *Fnew[]* be the set of facets created from *p* and all ridges in *H*.
    5. For each new facet in *Fnew[]*, perform step (2) and initialize its own outside sets. This time look only from points that are outside of a facet in *V* using their outside sets *V[i].O*, since we have only expanded in that direction.
    6. Delete the now-internal facets in *V* from *Fs[]*. Add the new facets in *Fnew[]* to *Fs[]* and continue the iteration.


Step 6: Recurse until no more points are left

Pseudocode for 2D set of points

```
Input = a set S of n points
Assume that there are at least 2 points in the input set S of points

function QuickHull(S) is
    // Find convex hull from the set S of n points
    Convex Hull := {}
    Find left and right most points, say A & B, and add A & B to convex hull
    Segment AB divides the remaining (n − 2) points into 2 groups S1 and S2
        where S1 are points in S that are on the right side of the oriented line from A to B,
        and S2 are points in S that are on the right side of the oriented line from B to A
    FindHull(S1, A, B)
    FindHull(S2, B, A)
    Output := Convex Hull
end function

function FindHull(Sk, P, Q) is
    // Find points on convex hull from the set Sk of points
    // that are on the right side of the oriented line from P to Q
    if Sk has no point then
        return
    From the given set of points in Sk, find farthest point, say C, from segment PQ
    Add point C to convex hull at the location between P and Q
    Three points P, Q, and C partition the remaining points of Sk into 3 subsets: S0, S1, and S2
        where S0 are points inside triangle PCQ, S1 are points on the right side of the oriented
        line from P to C, and S2 are points on the right side of the oriented line from C to Q.
    FindHull(S1, P, C)
    FindHull(S2, C, Q)
end function
```

A pseudocode specialized for the 3D case is available from Jordan Smith. It includes a similar "maximum point" strategy for choosing the starting hull. If these maximum points are degenerate, the whole point cloud is as well.[4]

See also

- Convex hull algorithms

References

1. Barber, C. Bradford; Dobkin, David P.; Huhdanpaa, Hannu (1 December 1996). "The quickhull algorithm for convex hulls" (http://www.cs.princeton.edu/~dpd/Papers/BarberDobkinHuhdanpaa.pdf) (PDF). *ACM Transactions on Mathematical Software*. **22** (4): 469–483. doi:10.1145/235815.235821 (https://doi.org/10.1145%2F235815.235821).
2. Devillers, Olivier; Glisse, Xavier Goaoc; Thomasse, Remy (2015). *On the Smoothed Complexity of Convex Hulls* (https://drops.dagstuhl.de/opus/volltexte/2015/5145/). 31st International Symposium on Computational Geometry. Schloss Dagstuhl--Leibniz-Zentrum fuer Informatik.
3. Greenfield, Jonathan S. (1 April 1990). "A Proof for a QuickHull Algorithm" (https://surface.syr.edu/eecs_techreports/65). *Electrical Engineering and Computer Science - Technical Reports*.
4. Smith, Jordan. "QuickHull 3D" (http://algolist.ru/maths/geom/convhull/qhull3d.php). *algolist.ru*. Retrieved 22 October 2019.

- Dave Mount. "Lecture 3: More Convex Hull Algorithms" (http://www.cs.wustl.edu/~pless/506/l3.html).
- Pseudocode, "http://www.cse.yorku.ca/~aaw/Hang/quick_hull/Algorithm.html".

External links

- Implementing QuickHull (GDC 2014) (http://media.steampowered.com/apps/valve/2014/DirkGregorius_ImplementingQuickHull.pdf) – Algorithm presentation with 3D implementation details.