Basic Geometry for competitive programing

Duy Huynh - 2016

Contents

Chapter 1

Computational geometry

For the journal, see Computational Geometry (journal).

Computational geometry is a branch of computer science devoted to the study of algorithms which can be stated in terms of geometry. Some purely geometrical problems arise out of the study of computational geometric algorithms, and such problems are also considered to be part of computational geometry. While modern computational geometry is a recent development, it is one of the oldest fields of computing with history stretching back to antiquity.

Computational complexity is central to computational geometry, with great practical significance if algorithms are used on very large datasets containing tens or hundreds of millions of points. For such sets, the difference between $O(n^2)$ and $O(n \log n)$ may be the difference between days and seconds of computation.

The main impetus for the development of computational geometry as a discipline was progress in computer graphics and computer-aided design and manufacturing (CAD/CAM), but many problems in computational geometry are classical in nature, and may come from mathematical visualization.

Other important applications of computational geometry include robotics (motion planning and visibility problems), geographic information systems (GIS) (geometrical location and search, route planning), integrated circuit design (IC geometry design and verification), computer-aided engineering (CAE) (mesh generation), computer vision (3D reconstruction).

The main branches of computational geometry are:

- *Combinatorial computational geometry*, also called *algorithmic geometry*, which deals with geometric objects as discrete entities. A groundlaying book in the subject by Preparata and Shamos dates the first use of the term "computational geometry" in this sense by 1975.[1]

- *Numerical computational geometry*, also called *machine geometry*, *computer-aided geometric design* (CAGD), or *geometric modeling*, which deals primarily with representing real-world objects in forms suitable for computer computations in CAD/CAM systems. This branch may be seen as a further development of descriptive geometry and is often considered a branch of computer graphics or CAD. The term "computational geometry" in this meaning has been in use since 1971.[2]

1.1 Combinatorial computational geometry

The primary goal of research in combinatorial computational geometry is to develop efficient algorithms and data structures for solving problems stated in terms of basic geometrical objects: points, line segments, polygons, polyhedra, etc.

Some of these problems seem so simple that they were not regarded as problems at all until the advent of computers. Consider, for example, the *Closest pair problem*:

- Given n points in the plane, find the two with the smallest distance from each other.

One could compute the distances between all the pairs of points, of which there are *n(n-1)/2*, then pick the pair with the smallest distance. This brute-force algorithm takes O(n^2) time; i.e. its execution time is proportional to the square of the number of points. A classic result in computational geometry was the formulation of an algorithm that takes O(n log n). Randomized algorithms that take O(n) expected time,[3] as well as a deterministic algorithm that takes O(n log log n) time,[4] have also been discovered.

1.1.1 Problem classes

The core problems in computational geometry may be classified in different ways, according to various criteria. The following general classes may be distinguished.

Static problems

In the problems of this category, some input is given and the corresponding output needs to be constructed or found. Some fundamental problems of this type are:

- Convex hull: Given a set of points, find the smallest convex polyhedron/polygon containing all the points.
- Line segment intersection: Find the intersections between a given set of line segments.
- Delaunay triangulation
- Voronoi diagram: Given a set of points, partition the space according to which points are closest to the given points.
- Linear programming
- Closest pair of points: Given a set of points, find the two with the smallest distance from each other.
- Euclidean shortest path: Connect two points in a Euclidean space (with polyhedral obstacles) by a shortest path.
- Polygon triangulation: Given a polygon, partition its interior into triangles
- Mesh generation
- Boolean operations on polygons

The computational complexity for this class of problems is estimated by the time and space (computer memory) required to solve a given problem instance.

Geometric query problems

In geometric query problems, commonly known as geometric search problems, the input consists of two parts: the search space part and the query part, which varies over the problem instances. The search space typically needs to be preprocessed, in a way that multiple queries can be answered efficiently.

Some fundamental geometric query problems are:

- Range searching: Preprocess a set of points, in order to efficiently count the number of points inside a query region.
- Point location: Given a partitioning of the space into cells, produce a data structure that efficiently tells in which cell a query point is located.
- Nearest neighbor: Preprocess a set of points, in order to efficiently find which point is closest to a query point.
- Ray tracing: Given a set of objects in space, produce a data structure that efficiently tells which object a query ray intersects first.

If the search space is fixed, the computational complexity for this class of problems is usually estimated by:

- the time and space required to construct the data structure to be searched in
- the time (and sometimes an extra space) to answer queries.

For the case when the search space is allowed to vary, see "Dynamic problems".

Dynamic problems

Yet another major class is the dynamic problems, in which the goal is to find an efficient algorithm for finding a solution repeatedly after each incremental modification of the input data (addition or deletion input geometric elements). Algorithms for problems of this type typically involve dynamic data structures. Any of the computational geometric problems may be converted into a dynamic one, at the cost of increased processing time. For example, the range searching problem may be converted into the dynamic range searching problem by providing for addition and/or deletion of the points. The dynamic convex hull problem is to keep track of the convex hull, e.g., for the dynamically changing set of points, i.e., while the input points are inserted or deleted.

The computational complexity for this class of problems is estimated by:

- the time and space required to construct the data structure to be searched in
- the time and space to modify the searched data structure after an incremental change in the search space
- the time (and sometimes an extra space) to answer a query.

Variations

Some problems may be treated as belonging to either of the categories, depending on the context. For example, consider the following problem.

- Point in polygon: Decide whether a point is inside or outside a given polygon.

In many applications this problem is treated as a single-shot one, i.e., belonging to the first class. For example, in many applications of computer graphics a common problem is to find which area on the screen is clicked by a pointer. However, in some applications the polygon in question is invariant, while the point represents a query. For example, the input polygon may represent a border of a country and a point is a position of an aircraft, and the problem is to determine whether the aircraft violated the border. Finally, in the previously mentioned example of computer graphics, in CAD applications the changing input data are often stored in dynamic data structures, which may be exploited to speed-up the point-in-polygon queries.

In some contexts of query problems there are reasonable expectations on the sequence of the queries, which may be exploited either for efficient data structures or for tighter computational complexity estimates. For example, in some cases it is important to know the worst case for the total time for the whole sequence of N queries, rather than for a single query. See also "amortized analysis".

1.2 Numerical computational geometry

Main article: computer-aided geometric design

This branch is also known as **geometric modelling** and **computer-aided geometric design** (CAGD).

Core problems are curve and surface modelling and representation.

The most important instruments here are parametric curves and parametric surfaces, such as Bézier curves, spline curves and surfaces. An important non-parametric approach is the level set method.

Application areas include shipbuilding, aircraft, and automotive industries. The modern ubiquity and power of computers means that even perfume bottles and shampoo dispensers are designed using techniques unheard of by shipbuilders of the 1960s.

1.3 See also

- List of combinatorial computational geometry topics
- List of numerical computational geometry topics
- CAD/CAM/CAE
- Robotics
- Solid modeling
- Computational topology
- Digital geometry
- Discrete geometry (combinatorial geometry)
- Space partitioning
- Tricomplex number
- Wikiversity:Topic:Computational geometry
- Wikiversity:Computer-aided geometric design

1.4 References

[1] Franco P. Preparata and Michael Ian Shamos (1985). *Computational Geometry - An Introduction*. Springer-Verlag. 1st edition: ISBN 0-387-96131-3; 2nd printing, corrected and expanded, 1988: ISBN 3-540-96131-3.

[2] A.R. Forrest, “Computational geometry”, *Proc. Royal Society London*, 321, series 4, 187-195 (1971)

[3] S. Khuller and Y. Matias. A simple randomized sieve algorithm for the closest-pair problem. Inf. Comput., 118(1):34—37,1995

[4] S. Fortune and J.E. Hopcroft. “A note on Rabin’s nearest-neighbor algorithm.” Information Processing Letters, 8(1), pp. 20—23, 1979

1.5 Further reading

- List of books in computational geometry

1.5.1 Journals

Combinatorial/algorithmic computational geometry

Below is the list of the major journals that have been publishing research in geometric algorithms. Please notice with the appearance of journals specifically dedicated to computational geometry, the share of geometric publications in general-purpose computer science and computer graphics journals decreased.

- *ACM Computing Surveys*
- *ACM Transactions on Graphics*
- *Acta Informatica*
- *Advances in Geometry*
- *Algorithmica*
- *Ars Combinatoria*

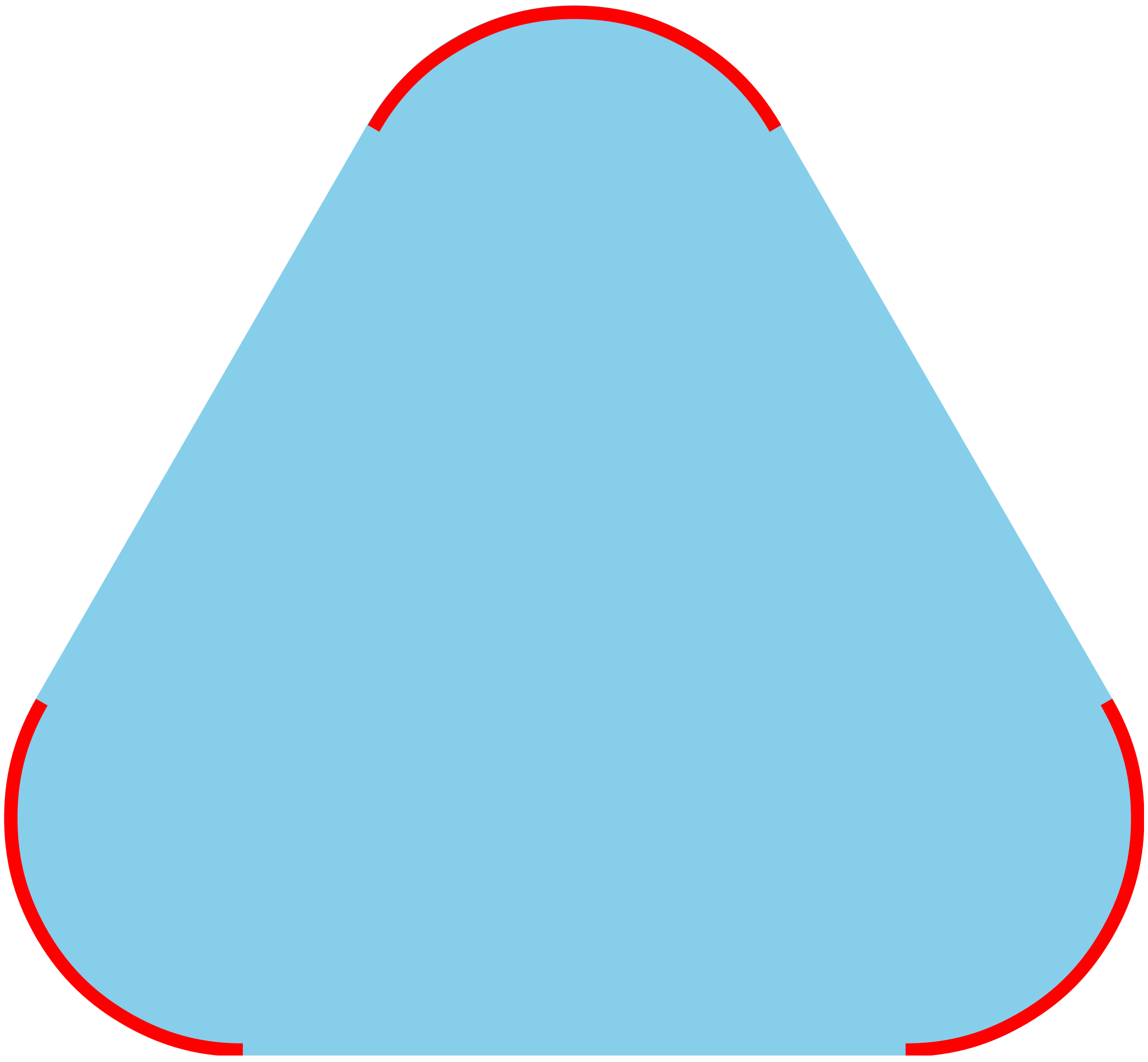
- *Computational Geometry: Theory and Applications*
- *Communications of the ACM*
- *Computer Aided Geometric Design*
- *Computer Graphics and Applications*
- *Computer Graphics World*
- *Discrete & Computational Geometry*
- *Geombinatorics*
- *Geometriae Dedicata*
- *IEEE Transactions on Graphics*
- *IEEE Transactions on Computers*
- *IEEE Transactions on Pattern Analysis and Machine Intelligence*
- *Information Processing Letters*
- *International Journal of Computational Geometry and Applications*
- *International Journal of Differential Geometry*
- *Journal of Combinatorial Theory, series B*
- *Journal of Computational Geometry*
- *Journal of the ACM*
- *Journal of Algorithms*
- *Journal of Computer and System Sciences*
- *Management Science*
- *Pattern Recognition*
- *Pattern Recognition Letters*
- *SIAM Journal on Computing*
- *SIGACT News*; featured the "Computational Geometry Column" by Joseph O'Rourke
- *Theoretical Computer Science*
- *The Visual Computer*

1.6 External links

- Computational Geometry
- Computational Geometry Pages
- Geometry In Action
- "Strategic Directions in Computational Geometry—Working Group Report" (1996)
- Journal of Computational Geometry
- (Annual) Winter School on Computational Geometry

Chapter 2

Convex hull



The convex hull of the red set is the blue and red convex set.

In mathematics, the **convex hull** or **convex envelope** of a set X of points in the Euclidean plane or Euclidean space is the smallest convex set that contains X. For instance, when X is a bounded subset of the plane, the convex hull may be visualized as the shape enclosed by a rubber band stretched around X.[1]

Formally, the convex hull may be defined as the intersection of all convex sets containing X or as the set of all convex combinations of points in X. With the latter definition, convex hulls may be extended from Euclidean spaces to arbitrary real vector spaces; they may also be generalized further, to oriented matroids.[2]

The algorithmic problem of finding the convex hull of a finite set of points in the plane or other low-dimensional Euclidean spaces is one of the fundamental problems of computational geometry.

2.1 Definitions

A set of points is defined to be convex if it contains the line segments connecting each pair of its points. The convex hull of a given set X may be defined as

1. The (unique) minimal convex set containing X
2. The intersection of all convex sets containing X
3. The set of all convex combinations of points in X.
4. The union of all simplices with vertices in X.

It is not obvious that the first definition makes sense: why should there exist a unique minimal convex set containing X, for every X? However, the second definition, the intersection of all convex sets containing X is well-defined, and it is a subset of every other convex set Y that contains X, because Y is included among the sets being intersected. Thus, it is exactly the unique minimal convex set containing X. Each convex set containing X must (by the assumption that it is convex) contain all convex combinations of points in X, so the set of all convex combinations is contained in the intersection of all convex sets containing X. Conversely, the set of all convex combinations is itself a convex set containing X, so it also contains the intersection of all convex sets containing X, and therefore the sets given by these two definitions must be equal. In fact, according to Carathéodory's theorem, if X is a subset of an N-dimensional vector space, convex combinations of at most $N + 1$ points are sufficient in the definition above. Therefore, the convex hull of a set X of three or more points in the plane is the union of all the triangles determined by triples of points from X, and more generally in N-dimensional space the convex hull is the union of the simplices determined by at most $N + 1$ vertices from X.

If the convex hull of X is a closed set (as happens, for instance, if X is a finite set or more generally a compact set), then it is the intersection of all closed half-spaces containing X. The hyperplane separation theorem proves that in this case, each point not in the convex hull can be separated from the convex hull by a half-space. However, there exist convex sets, and convex hulls of sets, that cannot be represented in this way. One example is an open halfspace together with a single point on its boundary.

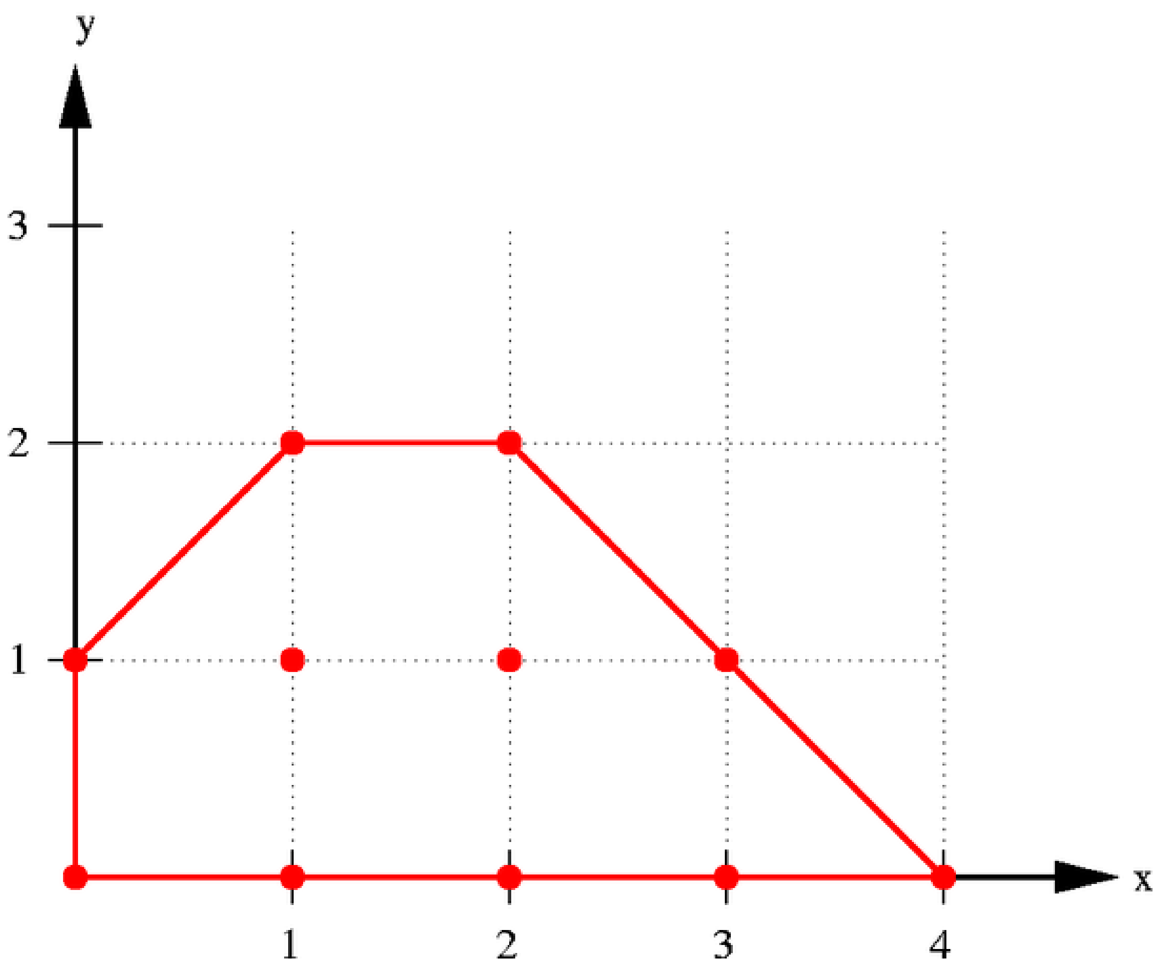
More abstractly, the convex-hull operator Conv() has the characteristic properties of a closure operator:

- It is *extensive*, meaning that the convex hull of every set X is a superset of X.
- It is *non-decreasing*, meaning that, for every two sets X and Y with $X \subseteq Y$, the convex hull of X is a subset of the convex hull of Y.
- It is *idempotent*, meaning that for every X, the convex hull of the convex hull of X is the same as the convex hull of X.

2.2 Convex hull of a finite point set

The convex hull of a finite point set S is the set of all convex combinations of its points. In a convex combination, each point x_i in S is assigned a weight or coefficient α_i in such a way that the coefficients are all non-negative and sum to one, and these weights are used to compute a weighted average of the points. For each choice of coefficients, the resulting convex combination is a point in the convex hull, and the whole convex hull can be formed by choosing coefficients in all possible ways. Expressing this as a single formula, the convex hull is the set:

$$\mathrm{Conv}(S) = \left\{ \sum_{i=1}^{|S|} \alpha_i x_i \,\middle|\, (\forall i : \alpha_i \geq 0) \wedge \sum_{i=1}^{|S|} \alpha_i = 1 \right\}.$$



Convex hull of some points in the plane

The convex hull of a finite point set $S \subsetneq \mathbb{R}^n$ forms a convex polygon when *n* = 2, or more generally a convex polytope in $\mathbb{R}^n$. Each point x_i in S that is not in the convex hull of the other points (that is, such that $x_i \notin \mathrm{Conv}(S \setminus \{x_i\})$) is called a vertex of $\mathrm{Conv}(S)$. In fact, every convex polytope in $\mathbb{R}^n$ is the convex hull of its vertices.

If the points of S are all on a line, the convex hull is the line segment joining the outermost two points. When the set S is a nonempty finite subset of the plane (that is, two-dimensional), we may imagine stretching a rubber band so that it surrounds the entire set S and then releasing it, allowing it to contract; when it becomes taut, it encloses the convex hull of S .[1]

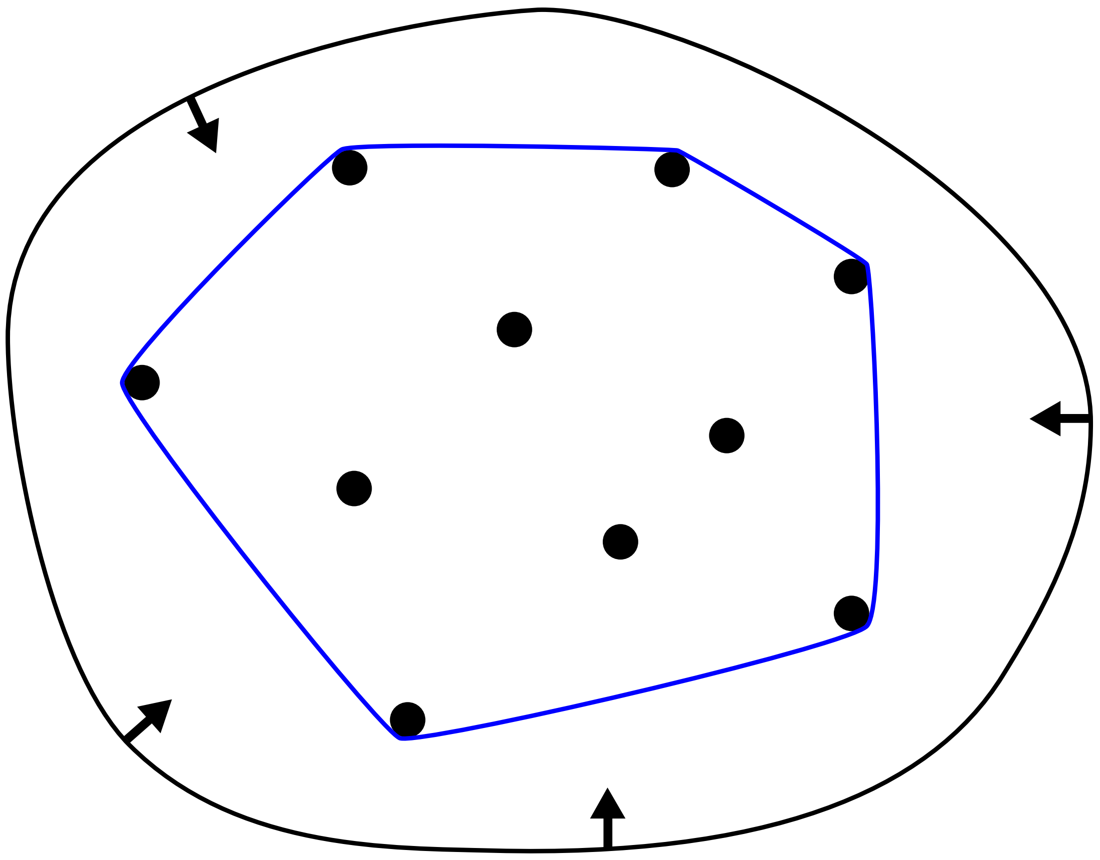
In two dimensions, the convex hull is sometimes partitioned into two polygonal chains, the upper hull and the lower hull, stretching between the leftmost and rightmost points of the hull. More generally, for points in any dimension in general position, each facet of the convex hull is either oriented upwards (separating the hull from points directly above it) or downwards; the union of the upward-facing facets forms a topological disk, the upper hull, and similarly the union of the downward-facing facets forms the lower hull.[3]

2.3 Computation of convex hulls

Main article: Convex hull algorithms

In computational geometry, a number of algorithms are known for computing the convex hull for a finite set of points and for other geometric objects.

Computing the convex hull means constructing an unambiguous, efficient representation of the required convex shape. The complexity of the corresponding algorithms is usually estimated in terms of ***n***, the number of input points, and ***h***, the number of points on the convex hull.



Convex hull of a finite set: elastic-band analogy

For points in two and three dimensions, output-sensitive algorithms are known that compute the convex hull in time O(n log h). For dimensions d higher than 3, the time for computing the convex hull is $O(n^{\lfloor d/2 \rfloor})$, matching the worst-case output complexity of the problem.[4]

2.4 Minkowski addition and convex hulls

See also: Minkowski addition and Shapley–Folkman lemma
The operation of taking convex hulls behaves well with respect to the Minkowski addition of sets.

- In a real vector-space, the *Minkowski sum* of two (non-empty) sets S_1 and S_2 is defined to be the set $S_1 + S_2$ formed by the addition of vectors element-wise from the summand-sets

  $S_1 + S_2 = \{ x_1 + x_2 : x_1 \in S_1 \text{ and } x_2 \in S_2 \}$.

More generally, the *Minkowski sum* of a finite family of (non-empty) sets S_n is the set formed by element-wise addition of vectors

$\sum S_n = \{ \sum xn : xn \in S_n \}$.

- For all subsets S_1 and S_2 of a real vector-space, the convex hull of their Minkowski sum is the Minkowski sum of their convex hulls

  $\mathrm{Conv}(S_1 + S_2) = \mathrm{Conv}(S_1) + \mathrm{Conv}(S_2)$.

This result holds more generally for each finite collection of non-empty sets

$$\mathrm{Conv}(\sum S_n) = \sum \mathrm{Conv}(S_n).$$

In other words, the operations of Minkowski summation and of forming convex hulls are commuting operations.[5][6]

These results show that *Minkowski addition* differs from the *union* operation of set theory; indeed, the union of two convex sets need *not* be convex: The inclusion Conv(S) ∪ Conv(T) ⊆ Conv(S ∪ T) is generally strict. The convex-hull operation is needed for the set of convex sets to form a lattice, in which the "*join*" operation is the convex hull of the union of two convex sets

$$\mathrm{Conv}(S) \vee \mathrm{Conv}(T) = \mathrm{Conv}(S \cup T) = \mathrm{Conv}(\mathrm{Conv}(S) \cup \mathrm{Conv}(T)).$$

2.5 Relations to other structures

The Delaunay triangulation of a point set and its dual, the Voronoi diagram, are mathematically related to convex hulls: the Delaunay triangulation of a point set in $\mathbf{R}^n$ can be viewed as the projection of a convex hull in $\mathbf{R}^{n+1}$.[7]

Topologically, the convex hull of an open set is always itself open, and the convex hull of a compact set is always itself compact; however, there exist closed sets for which the convex hull is not closed.[8] For instance, the closed set

$$\left\{(x,y) \mid y \geq \frac{1}{1+x^2}\right\}$$

has the open upper half-plane as its convex hull.

2.6 Applications

The problem of finding convex hulls finds its practical applications in pattern recognition, image processing, statistics, geographic information system, game theory, construction of phase diagrams, and static code analysis by abstract interpretation. It also serves as a tool, a building block for a number of other computational-geometric algorithms such as the rotating calipers method for computing the width and diameter of a point set.

The convex hull is commonly known as the minimum convex polygon (MCP) in ethology, where it is a classic, though perhaps simplistic, approach in estimating an animal's home range based on points where the animal has been observed.[9] Outliers can make the MCP excessively large, which has motivated relaxed approaches that contain only a subset of the observations (e.g., find an MCP that contains at least 95% of the points).[10]

2.7 See also

- Affine hull
- Alpha shape
- Choquet theory
- Concave set
- Helly's theorem
- Holomorphically convex hull
- Krein–Milman theorem
- Linear hull
- Oloid
- Orthogonal convex hull

2.8 Notes

[1] de Berg et al. (2000), p. 3.

[2] Knuth (1992).

[3] de Berg et al. (2000), p. 6. The idea of partitioning the hull into two chains comes from an efficient variant of Graham scan by Andrew (1979).

[4] Chazelle (1993).

[5] Krein & Šmulian (1940), Theorem 3, pages 562–563.

[6] For the commutativity of Minkowski addition and convexification, see Theorem 1.1.2 (pages 2–3) in Schneider (1993); this reference discusses much of the literature on the convex hulls of Minkowski sumsets in its "Chapter 3 Minkowski addition" (pages 126–196).

[7] Brown (1979).

[8] Grünbaum (2003), p. 16.

[9] Kernohan, Gitzen & Millspaugh (2001), p. 137–140

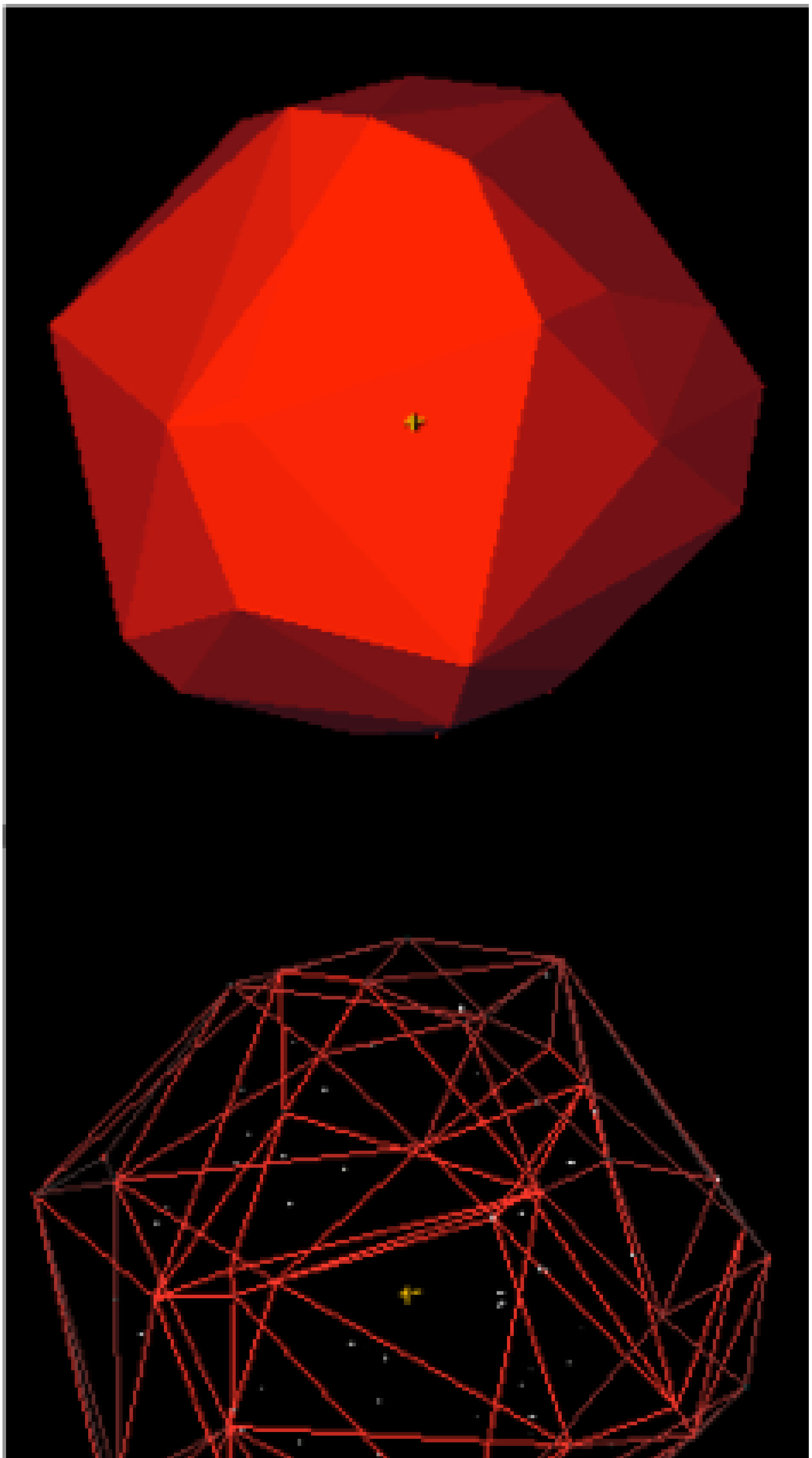
[10] Examples: The v.adehabitat.mcp GRASS module and adehabitatHR R package with percentage parameters for MCP calculation.

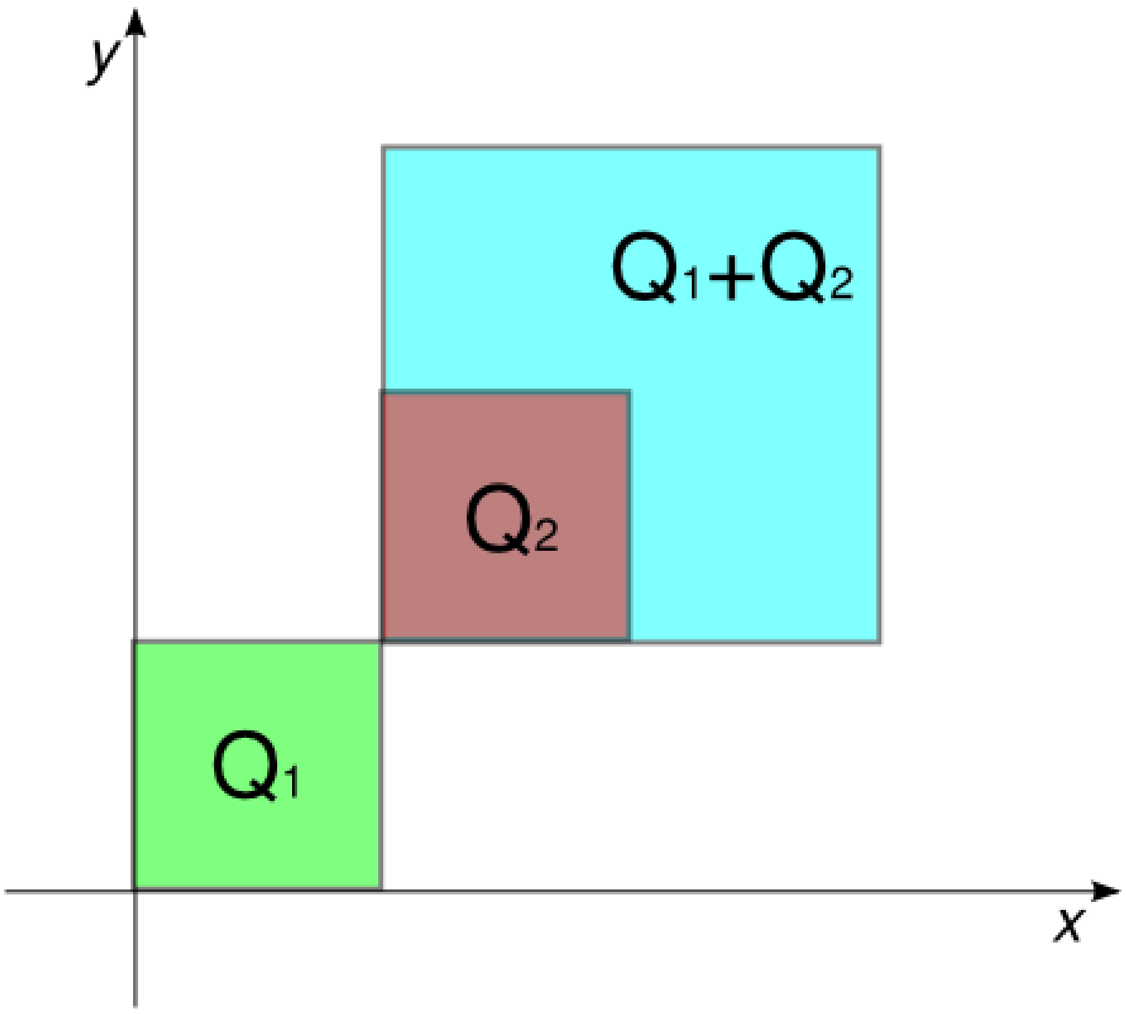
2.9 References

- Andrew, A. M. (1979), "Another efficient algorithm for convex hulls in two dimensions", *Information Processing Letters*, **9** (5): 216–219, doi:10.1016/0020-0190(79)90072-3.
- Brown, K. Q. (1979), "Voronoi diagrams from convex hulls", *Information Processing Letters*, **9** (5): 223–228, doi:10.1016/0020-0190(79)90074-7.
- de Berg, M.; van Kreveld, M.; Overmars, Mark; Schwarzkopf, O. (2000), *Computational Geometry: Algorithms and Applications*, Springer, pp. 2–8.
- Chazelle, Bernard (1993), "An optimal convex hull algorithm in any fixed dimension" (PDF), *Discrete and Computational Geometry*, **10** (1): 377–409, doi:10.1007/BF02573985.
- Grünbaum, Branko (2003), *Convex Polytopes*, Graduate Texts in Mathematics (2nd ed.), Springer, ISBN 9780387004242.
- Kernohan, Brian J.; Gitzen, Robert A.; Millspaugh, Joshua J. (2001), Joshua Millspaugh, John M. Marzluff, eds., "Ch. 5: Analysis of Animal Space Use and Movements", *Radio Tracking and Animal Populations*, Academic Press, ISBN 9780080540221.
- Knuth, Donald E. (1992), *Axioms and hulls*, Lecture Notes in Computer Science, **606**, Heidelberg: Springer-Verlag, p. ix+109, doi:10.1007/3-540-55611-7, ISBN 3-540-55611-7, MR 1226891.
- Krein, M.; Šmulian, V. (1940), "On regularly convex sets in the space conjugate to a Banach space", *Annals of Mathematics*, 2nd ser., **41**: 556–583, doi:10.2307/1968735, JSTOR 1968735, MR 2009.
- Schneider, Rolf (1993), *Convex bodies: The Brunn–Minkowski theory*, Encyclopedia of Mathematics and its Applications, **44**, Cambridge: Cambridge University Press, ISBN 0-521-35220-7, MR 1216521.

2.10 External links

- Weisstein, Eric W. "Convex Hull". *MathWorld*.
- "Convex Hull" by Eric W. Weisstein, Wolfram Demonstrations Project, 2007.





Minkowski addition of sets. The sum of the squares Q_1=[0,1]^2 and Q_2=[1,2]^2 is the square Q_1+Q_2=[1,3]^2.

Chapter 3

Graham scan

Graham's scan is a method of finding the convex hull of a finite set of points in the plane with time complexity O(*n* log *n*). It is named after Ronald Graham, who published the original algorithm in 1972.[1] The algorithm finds all vertices of the convex hull ordered along its boundary.

3.1 Algorithm

The first step in this algorithm is to find the point with the lowest y-coordinate. If the lowest y-coordinate exists in more than one point in the set, the point with the lowest x-coordinate out of the candidates should be chosen. Call this point *P*. This step takes O(*n*), where *n* is the number of points in question.

Next, the set of points must be sorted in increasing order of the angle they and the point *P* make with the x-axis. Any general-purpose sorting algorithm is appropriate for this, for example heapsort (which is O(*n* log *n*)).

Sorting in order of angle does not require computing the angle. It is possible to use any function of the angle which is monotonic in the interval $[0, \pi)$. The cosine is easily computed using the dot product, or the slope of the line may be used. If numeric precision is at a stake, the comparison function used by the sorting algorithm can use the sign of the cross product to determine relative angles.

The algorithm proceeds by considering each of the points in the sorted array in sequence. For each point, it is first determined whether traveling from the two points immediately preceding this point constitutes making a left turn or a right turn. If a right turn, the second-to-last point is not part of the convex hull, and lies 'inside' it. The same determination is then made for the set of the latest point and the two points that immediately precede the point found to have been inside the hull, and is repeated until a "left turn" set is encountered, at which point the algorithm moves on to the next point in the set of points in the sorted array minus any points that were found to be inside the hull; there is no need to consider these points again. (If at any stage the three points are collinear, one may opt either to discard or to report it, since in some applications it is required to find all points on the boundary of the convex hull.)

Again, determining whether three points constitute a "left turn" or a "right turn" does not require computing the actual angle between the two line segments, and can actually be achieved with simple arithmetic only. For three points $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$ and $P_3 = (x_3, y_3)$, simply compute the *z*-coordinate of the cross product of the two vectors $\overrightarrow{P_1P_2}$ and $\overrightarrow{P_1P_3}$, which is given by the expression $(x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)$. If the result is 0, the points are collinear; if it is positive, the three points constitute a "left turn" or counter-clockwise orientation, otherwise a "right turn" or clockwise orientation (for counter-clockwise numbered points).

This process will eventually return to the point at which it started, at which point the algorithm is completed and the stack now contains the points on the convex hull in counterclockwise order.

3.2 Time complexity

Sorting the points has time complexity O(*n* log *n*). While it may seem that the time complexity of the loop is O(n^2), because for each point it goes back to check if any of the previous points make a "right turn", it is actually O(*n*), because each point is considered at most twice in some sense. Each point can appear only once as a point (x_2, y_2) in a "left turn" (because the algorithm advances to the next point (x_3, y_3) after that), and as a point (x_2, y_2) in a "right turn" (because the point (x_2, y_2) is removed). The overall time complexity is therefore O(n log n), since the time to sort dominates the time to actually compute the convex hull.

3.3 Pseudocode

First, define

# *Three points are a counter-clockwise turn if ccw > 0, clockwise if # ccw < 0, and collinear if ccw = 0 because ccw is a determinant that # gives twice the signed area of the triangle formed by p1, p2 and p3.* **function** ccw(p1, p2, p3): **return** (p2.x - p1.x)*(p3.y - p1.y) - (p2.y - p1.y)*(p3.x - p1.x)

Then let the result be stored in the array points.

let N = number of points **let** points[N+1] = the array of points **swap** points[1] with the point with the lowest y-coordinate **sort** points by polar angle with points[1] # *We want points[0] to be a sentinel point that will stop the loop.* **let** points[0] = points[N] # *M will denote the number of points on the convex hull.* **let** M = 1 **for** i = 2 **to** N: # *Find next valid point on convex hull.* **while ccw**(points[M-1], points[M], points[i]) <= 0: **if** M > 1: M -= 1 # *All points are collinear* **else if** i == N: **break else** i += 1 # *Update M and swap points[i] to the correct place.* M += 1 **swap** points[M] with points[i]

This pseudocode is adapted from Sedgewick and Wayne's *Algorithms, 4th edition*.

The check inside the while statement is necessary to avoid the case when all points in the set are collinear.

3.4 Notes

The same basic idea works also if the input is sorted on x-coordinate instead of angle, and the hull is computed in two steps producing the upper and the lower parts of the hull respectively. This modification was devised by A. M. Andrew[2] and is known as Andrew's Monotone Chain Algorithm. It has the same basic properties as Graham's scan.[3]

The stack technique used in Graham's scan is very similar to that for the all nearest smaller values problem, and parallel algorithms for all nearest smaller values may also be used (like Graham's scan) to compute convex hulls of sorted sequences of points efficiently.[4]

3.5 References

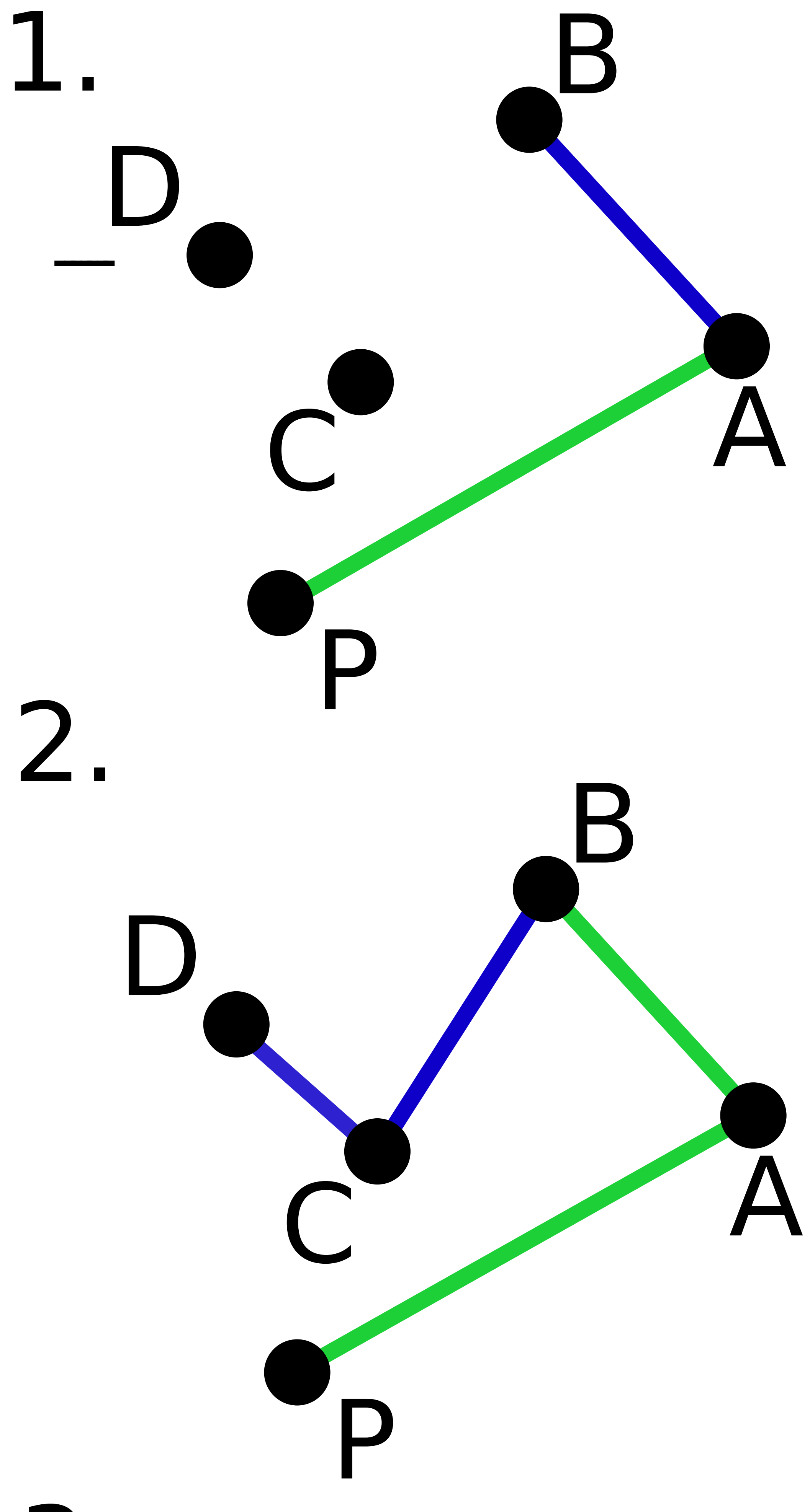
[1] Graham, R.L. (1972). An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. Information Processing Letters 1, 132-133

[2] Andrew, A. M. (1979). "Another efficient algorithm for convex hulls in two dimensions". *Information Processing Letters*. **9** (5): 216–219. doi:10.1016/0020-0190(79)90072-3.

[3] De Berg, Mark; Cheong, Otfried; Van Kreveld, Marc; Overmars, Mark (2008). *Computational Geometry Algorithms and Applications*. Berlin: Springer. pp. 2–14. doi:10.1007/978-3-540-77974-2. ISBN 978-3-540-77973-5.

[4] Berkman, Omer; Schieber, Baruch; Vishkin, Uzi (1993). "Optimal double logarithmic parallel algorithms based on finding all nearest smaller values". *Journal of Algorithms*. **14** (3): 344–370. doi:10.1006/jagm.1993.1018..

- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001) [1990]. "33.3: Finding the convex hull". *Introduction to Algorithms* (2nd ed.). MIT Press and McGraw-Hill. pp. 949–955. ISBN 0-262-03293-7.


1.
B
D
A
C
P
2.
B
D
C
A
P

Chapter 4

Quickhull

Main article: Convex hull algorithms

Quickhull is a method of computing the convex hull of a finite set of points in the plane. It uses a divide and conquer approach similar to that of quicksort, which its name derives from. Its average case complexity is considered to be O(n * log(n)), whereas in the worst case it takes O(n^2) (quadratic).

4.1 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:

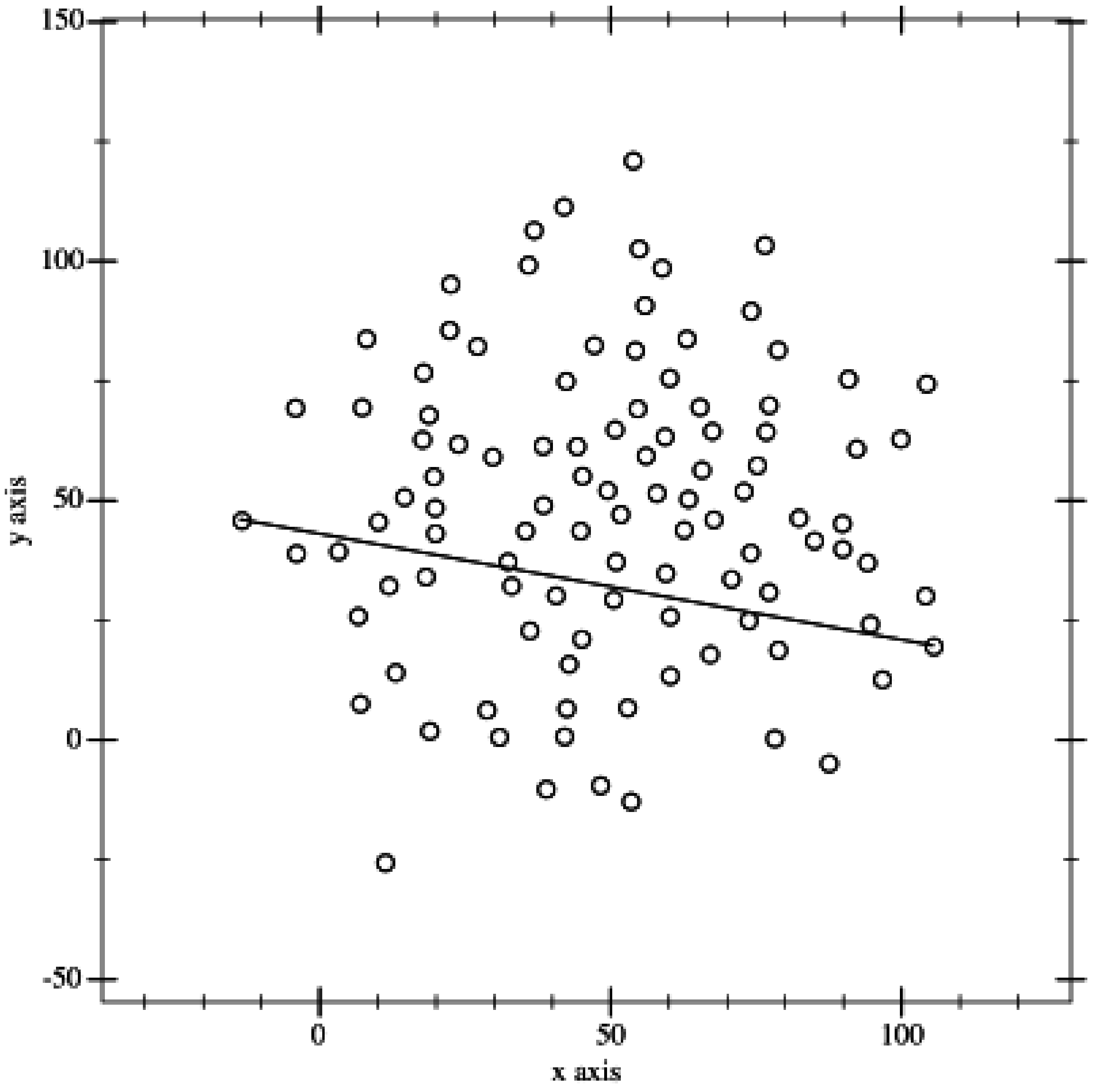
1. Find the points with minimum and maximum x coordinates, those are bound to be part of the convex hull.
2. Use the line formed by the two points to divide the set in two subsets of points, which will be processed recursively.
3. Determine the point, on one side of the line, with the maximum distance from the line. The two points found before along with this one form a triangle.
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).
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.

4.2 Alternative algorithm

This is a minor change towards the initial steps of the algorithm which might help save some computation time. One can note that the points with minimum and maximum y coordinates are also bound to be part of the convex hull, therefore, there are four points (or less depending on whether these points are the same or different) which are bound to be part of the convex hull. It is possible to discard directly all points lying inside the quadrilateral found within these four points (or less). Also, a first check needs to be made to check the number of points at start, if there are only three points, then the algorithm can be solved in O(1) since the three points are part of the convex hull.

The algorithm can be broken down to the following steps:

1. If the set of point is of size 3, then the three points are part of the convex hull and the algorithm can be ended.



This animation depicts the quickhull algorithm.

2. Find the points with minimum and maximum x coordinates and with minimum and maximum y coordinates; those are bound to be part of the convex hull.

3. The points lying inside of the quadrilateral formed by the previous extrema cannot be part of the convex hull and can therefore be ignored in the next steps.

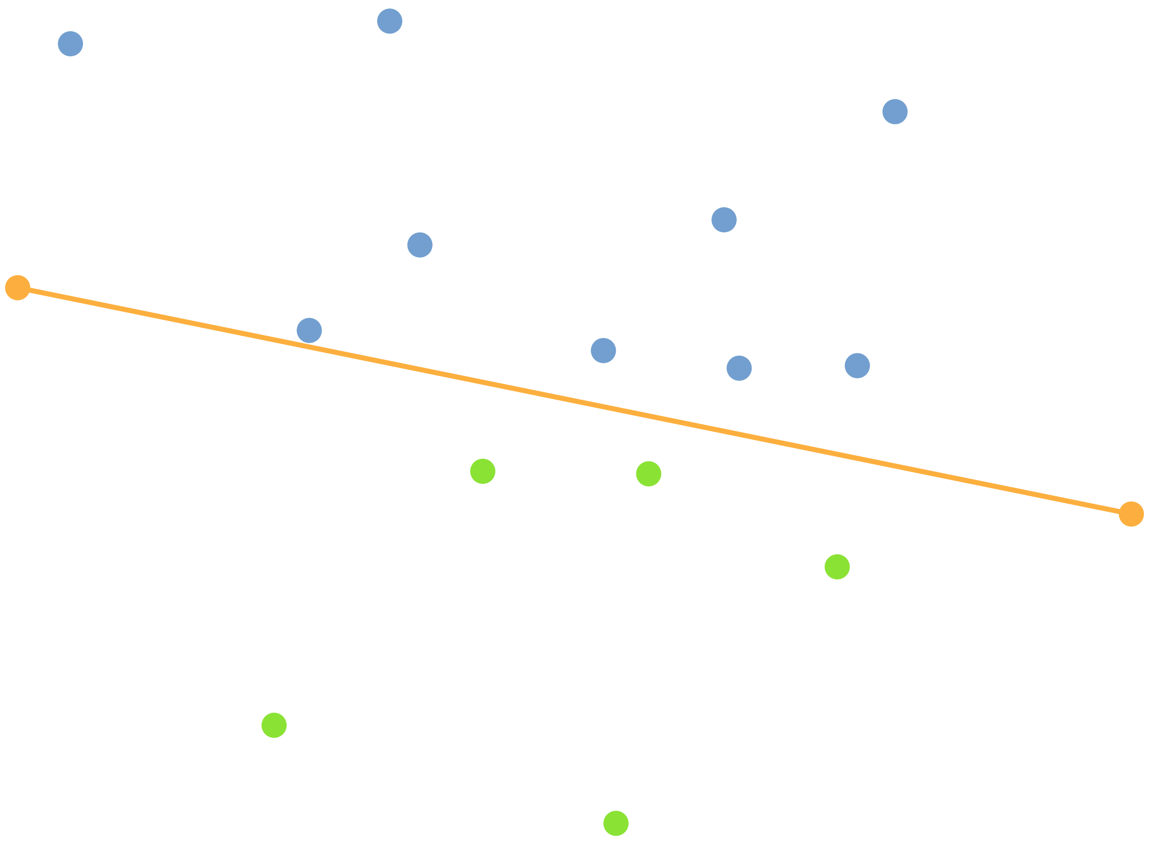
4. Use the line formed by a pair of extrema from step two to divide the remaining set in two subsets of points, which will be processed recursively.

5. Determine the point, on one side of the line, with the maximum distance from the line. The two points found before along with this one form a triangle.

6. The points lying inside of that triangle cannot be part of the convex hull and can therefore be ignored in the next steps.

7. Repeat the previous two steps on the two lines formed by the triangle (not the initial line).

8. 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.



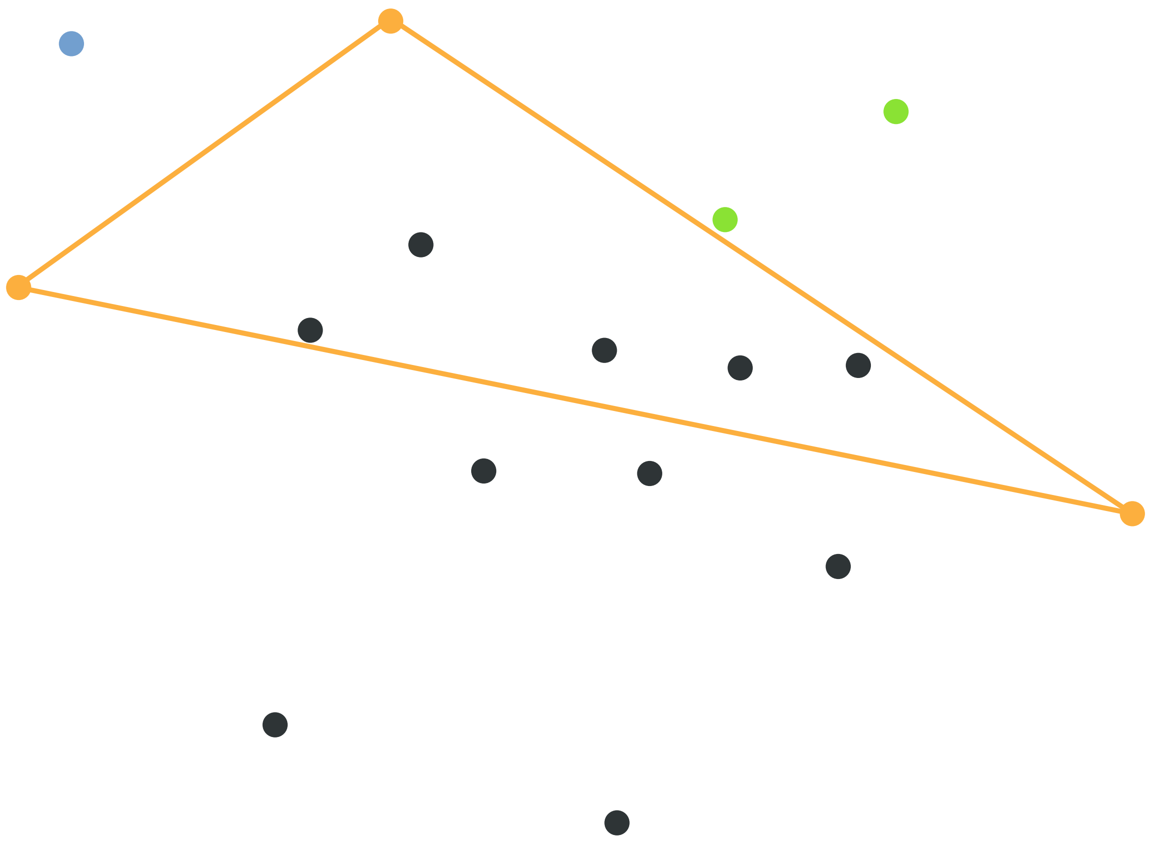
Steps 1-2: Divide points in two subsets

4.3 Pseudo Code

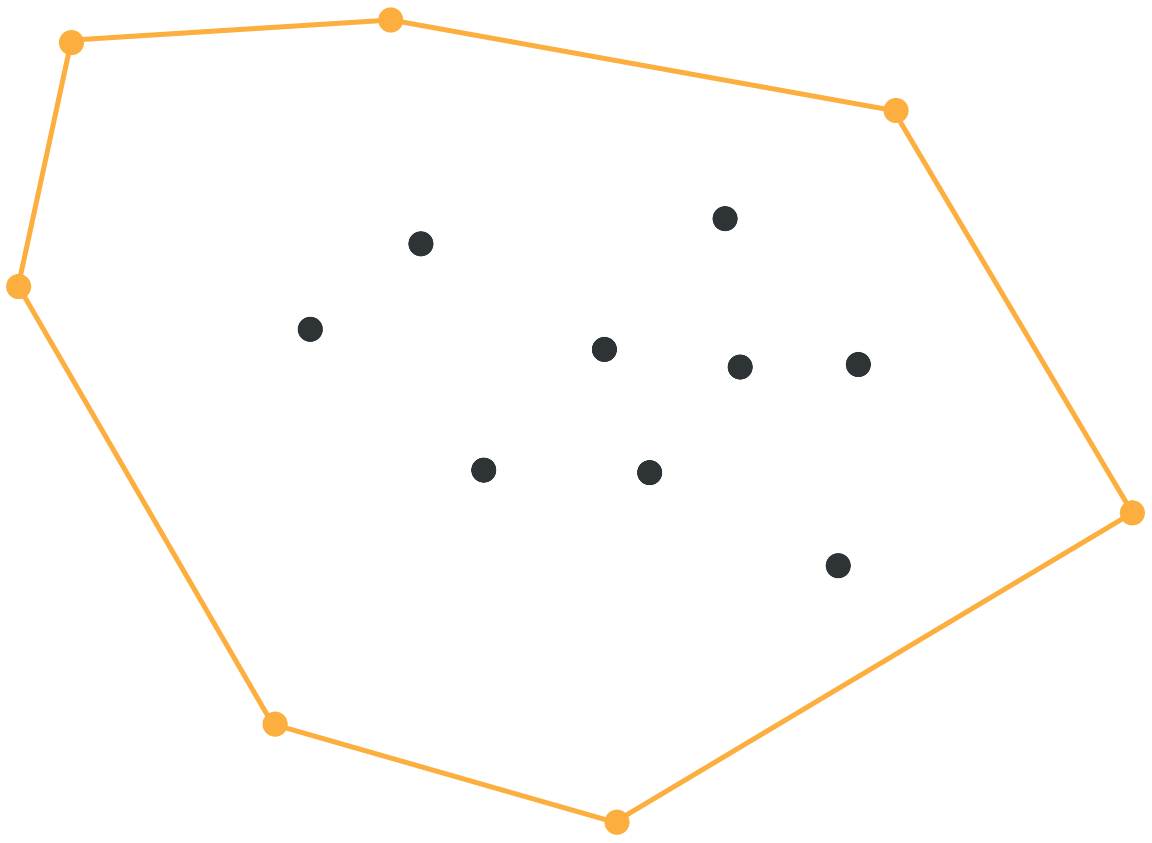
Input = a set S of n points Assume that there are at least 2 points in the input set S of points QuickHull (S) { // Find convex hull from the set S of n points Convex Hull := {} Find left and right most points, say A & B, and add A & B to convex hull Segment AB divides the remaining (n-2) points into 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) } FindHull (Sk, P, Q) { // 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) } Output = convex hull

4.4 References

- Barber, C. Bradford; Dobkin, David P.; Huhdanpaa, Hannu (1 December 1996). "The quickhull algorithm for convex hulls" (PDF). *ACM Transactions on Mathematical Software*. **22** (4): 469–483. doi:10.1145/235815.235821.
- Dave Mount. "Lecture 3: More Convex Hull Algorithms".
- Pseudocode, "http://www.cse.yorku.ca/~{}aaw/Hang/quick_hull/Algorithm.html".



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



Step 6: Recurse until no more points are left

Chapter 5

Pick’s theorem

For the theorem in complex analysis, see Schwarz lemma § Schwarz–Pick theorem.

Given a simple polygon constructed on a grid of equal-distanced points (i.e., points with integer coordinates) such that all the polygon’s vertices are grid points, **Pick’s theorem** provides a simple formula for calculating the area *A* of this polygon in terms of the number *i* of *lattice points in the interior* located in the polygon and the number *b* of *lattice points on the boundary* placed on the polygon’s perimeter:[1]

$$A = i + \frac{b}{2} - 1.$$

In the example shown, we have $i = 7$ interior points and $b = 8$ boundary points, so the area is $A = 7 + 8/2 - 1 = 7 + 4 - 1 = 10$ (square units)

Note that the theorem as stated above is only valid for *simple* polygons, i.e., ones that consist of a single piece and do not contain “holes”. For a polygon that has *h* holes, with a boundary in the form of $h + 1$ simple closed curves, the slightly more complicated formula $i + b/2 + h - 1$ gives the area.

The result was first described by Georg Alexander Pick in 1899.[2] The Reeve tetrahedron shows that there is no analogue of Pick’s theorem in three dimensions that expresses the volume of a polytope by counting its interior and boundary points. However, there is a generalization in higher dimensions via Ehrhart polynomials. The formula also generalizes to surfaces of polyhedra.

5.1 Proof

Consider a polygon *P* and a triangle *T*, with one edge in common with *P*. Assume Pick’s theorem is true for both *P* and *T* separately; we want to show that it is also true for the polygon *PT* obtained by adding *T* to *P*. Since *P* and *T* share an edge, all the boundary points along the edge in common are merged to interior points, except for the two endpoints of the edge, which are merged to boundary points. So, calling the number of boundary points in common *c*, we have[3]

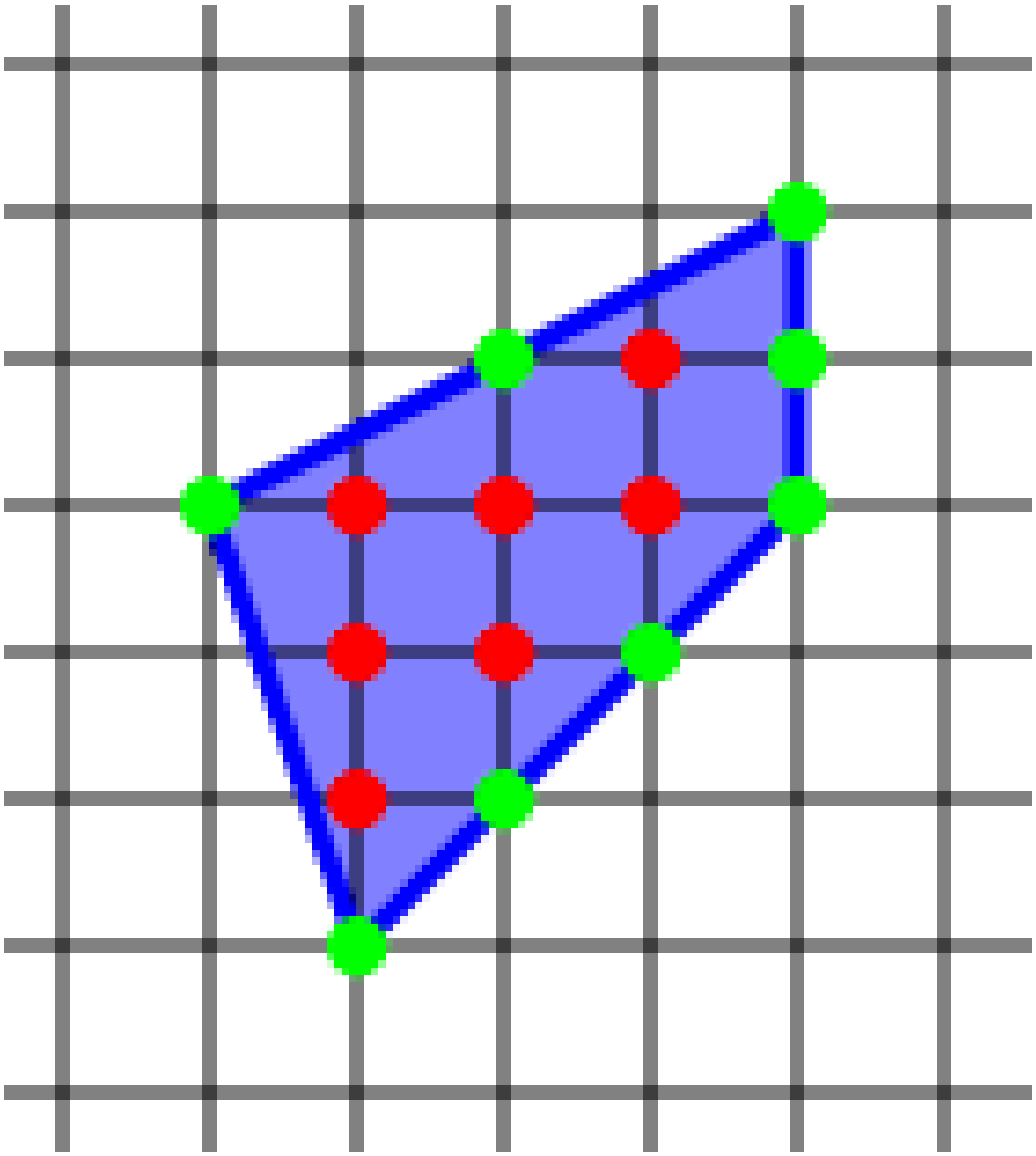
$$i_{PT} = (i_P + i_T) + (c - 2)$$

and

$$b_{PT} = (b_P + b_T) - 2(c - 2) - 2.$$

From the above follows

$$(i_P + i_T) = i_{PT} - (c - 2)$$



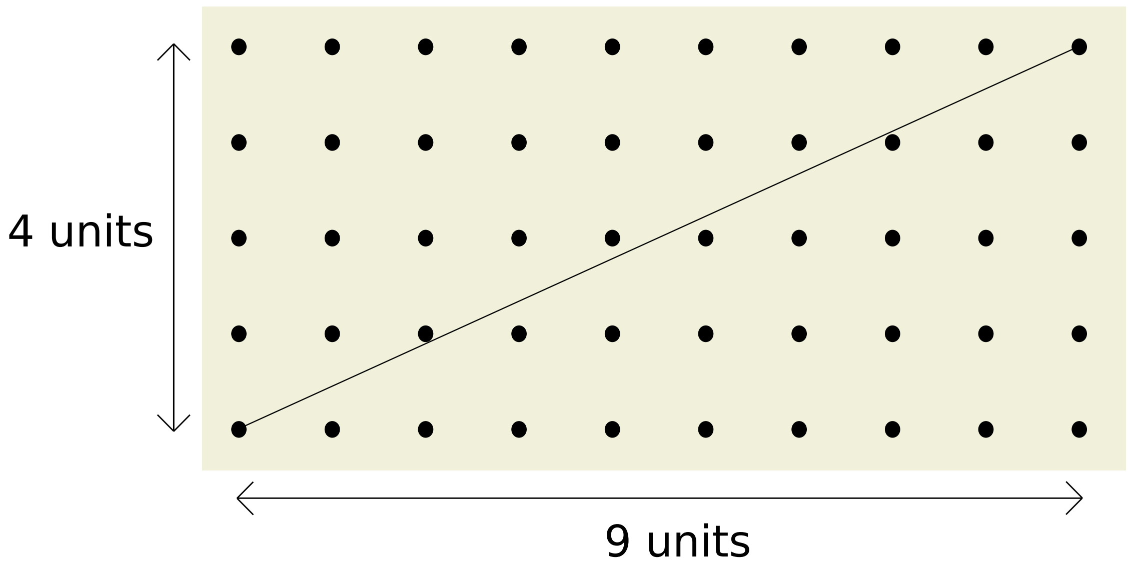
i = 7, b = 8,
A = i + b/2 − 1 = 10

and

$$(b_P + b_T) = b_{PT} + 2(c-2) + 2.$$

Since we are assuming the theorem for *P* and for *T* separately,

$$\begin{aligned} A_{PT} &= A_P + A_T \\ &= (i_P + b_P/2 - 1) + (i_T + b_T/2 - 1) \\ &= (i_P + i_T) + (b_P + b_T)/2 - 2 \\ &= i_{PT} - (c-2) + (b_{PT} + 2(c-2) + 2)/2 - 2 \\ &= i_{PT} + b_{PT}/2 - 1. \end{aligned}$$



The triangle with vertices at the lower left, lower right, and upper right points has i = *12 and* b = *14, giving by Pick's theorem A = i + b/2 − 1 = 18; this is confirmed by the triangle area formula* $^1/_2$ *× base × height =* $^1/_2$ *× 9 × 4 = 18.*

Therefore, if the theorem is true for polygons constructed from n triangles, the theorem is also true for polygons constructed from $n + 1$ triangles. For general polytopes, it is well known that they can always be triangulated. That this is true in dimension 2 is an easy fact. To finish the proof by mathematical induction, it remains to show that the theorem is true for triangles. The verification for this case can be done in these short steps:

- observe that the formula holds for any unit square (with vertices having integer coordinates);
- deduce from this that the formula is correct for any rectangle with sides parallel to the axes;
- deduce it, now, for right-angled triangles obtained by cutting such rectangles along a diagonal;
- now any triangle can be turned into a rectangle by attaching such right triangles; since the formula is correct for the right triangles and for the rectangle, it also follows for the original triangle.

The last step uses the fact that if the theorem is true for the polygon PT and for the triangle T, then it's also true for P; this can be seen by a calculation very much similar to the one shown above.

5.2 See also

- Integer points in convex polyhedra

5.3 References

[1] Trainin, J. (November 2007). "An elementary proof of Pick's theorem". *Mathematical Gazette*. **91** (522): 536–540.

[2] Pick, Georg (1899). "Geometrisches zur Zahlenlehre". *Sitzungsberichte des deutschen naturwissenschaftlich-medicinischen Vereines für Böhmen "Lotos" in Prag*. (Neue Folge). **19**: 311–319. JFM 33.0216.01. CiteBank:47270

[3] Beck, Matthias; Robins, Sinai (2007), *Computing the Continuous Discretely, Integer-point enumeration in polyhedra*, Undergraduate Texts in Mathematics, New York: Springer-Verlag, ISBN 978-0-387-29139-0, MR 2271992: chapter 2.

5.4 External links

- Pick's Theorem (Java) at cut-the-knot
- Pick's Theorem
- Pick's Theorem proof by Tom Davis
- Pick's Theorem by Ed Pegg, Jr., the Wolfram Demonstrations Project.
- Weisstein, Eric W. "Pick's Theorem". *MathWorld*.

5.5 Text and image sources, contributors, and licenses

5.5.1 Text

5.5.2 Images

5.5.3 Content license