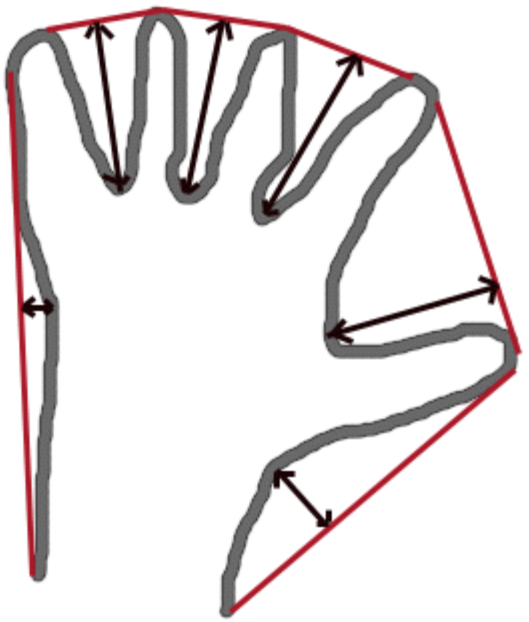
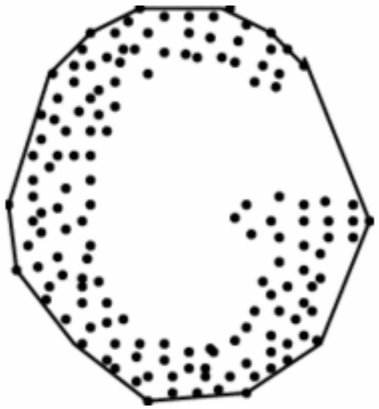
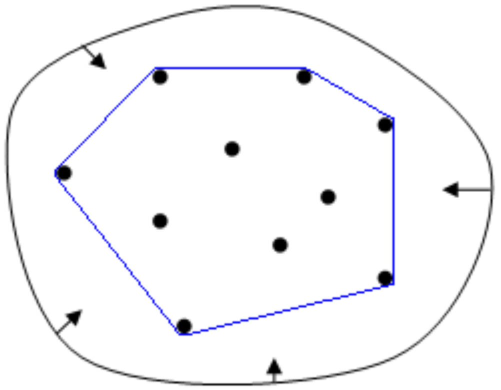
Flavor of Computational Geometry

Convex Hull in 2D

Shireen Y. Elhabian
Aly A. Farag
University of Louisville

February 2010

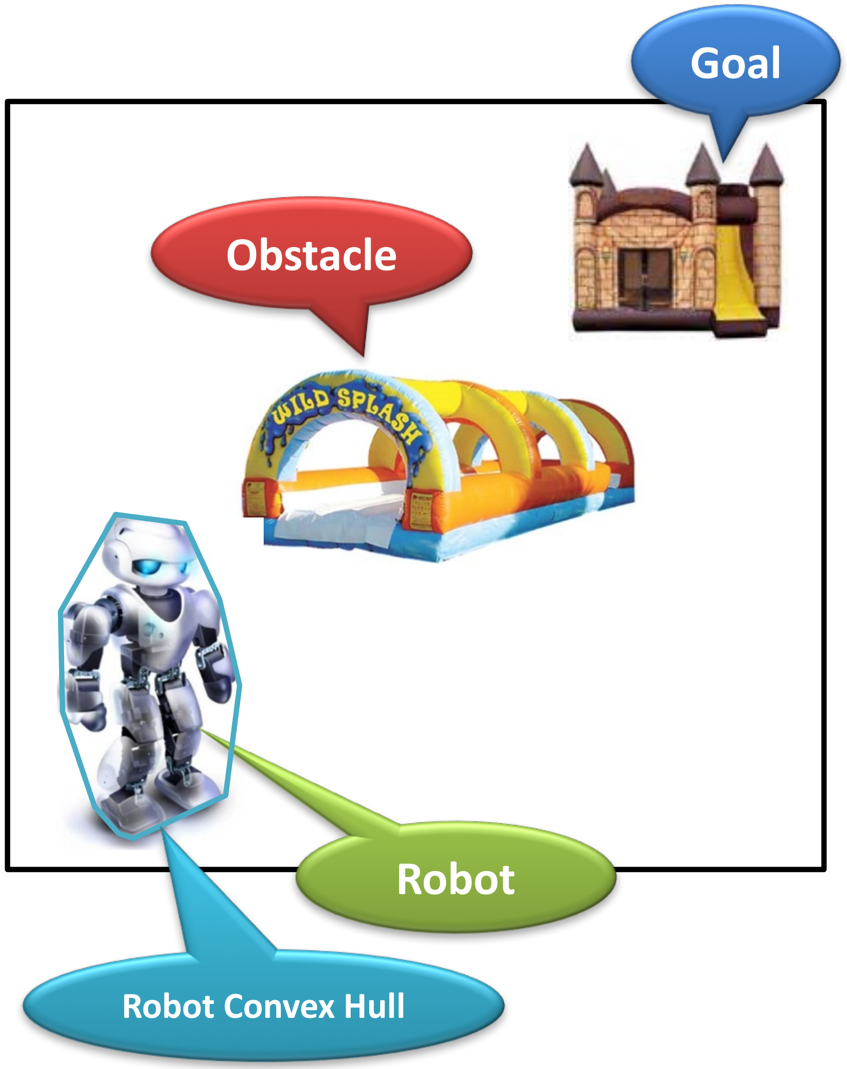




Agenda

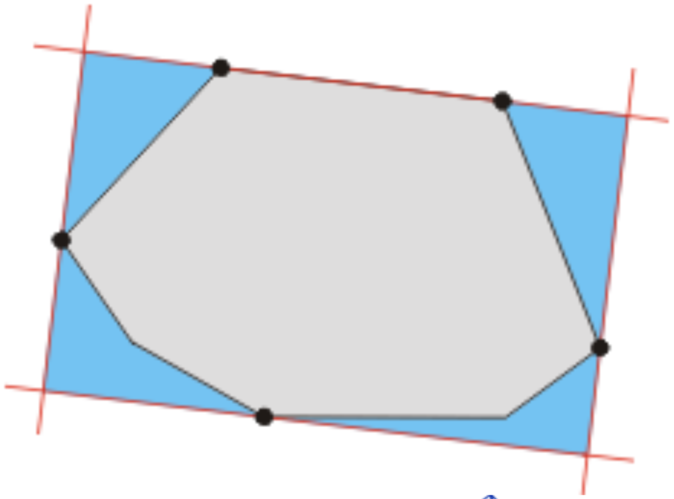
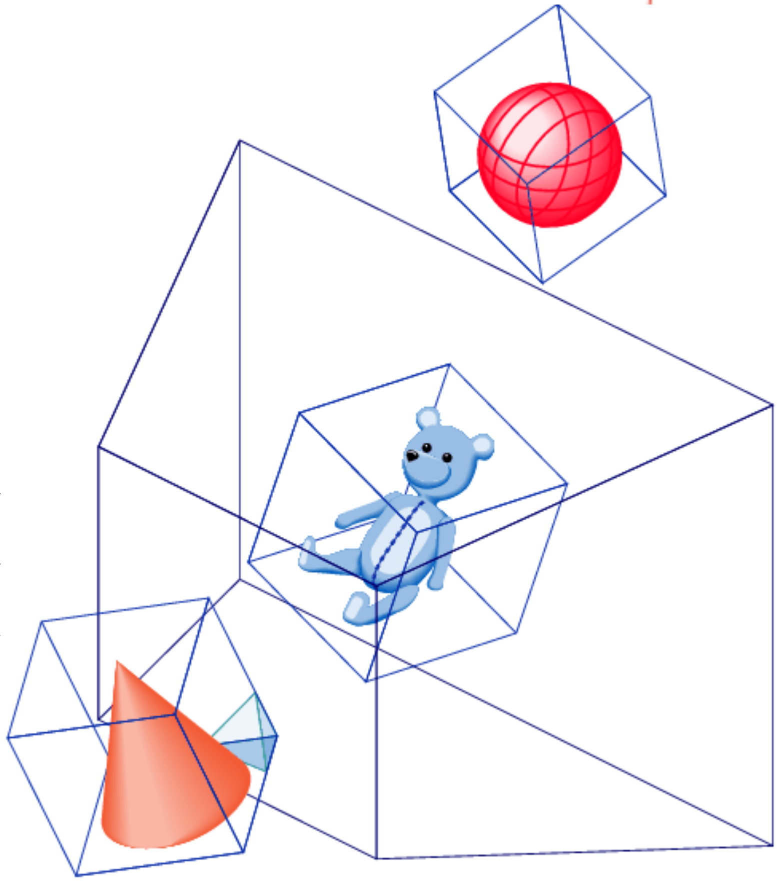
- Introduction
- Definitions of Convexity and Convex Hulls
- Naïve Algorithms for Extreme Points
- Gift Wrapping
- QuickHull
- Graham’s Algorithm
- Incremental Algorithm

Introduction

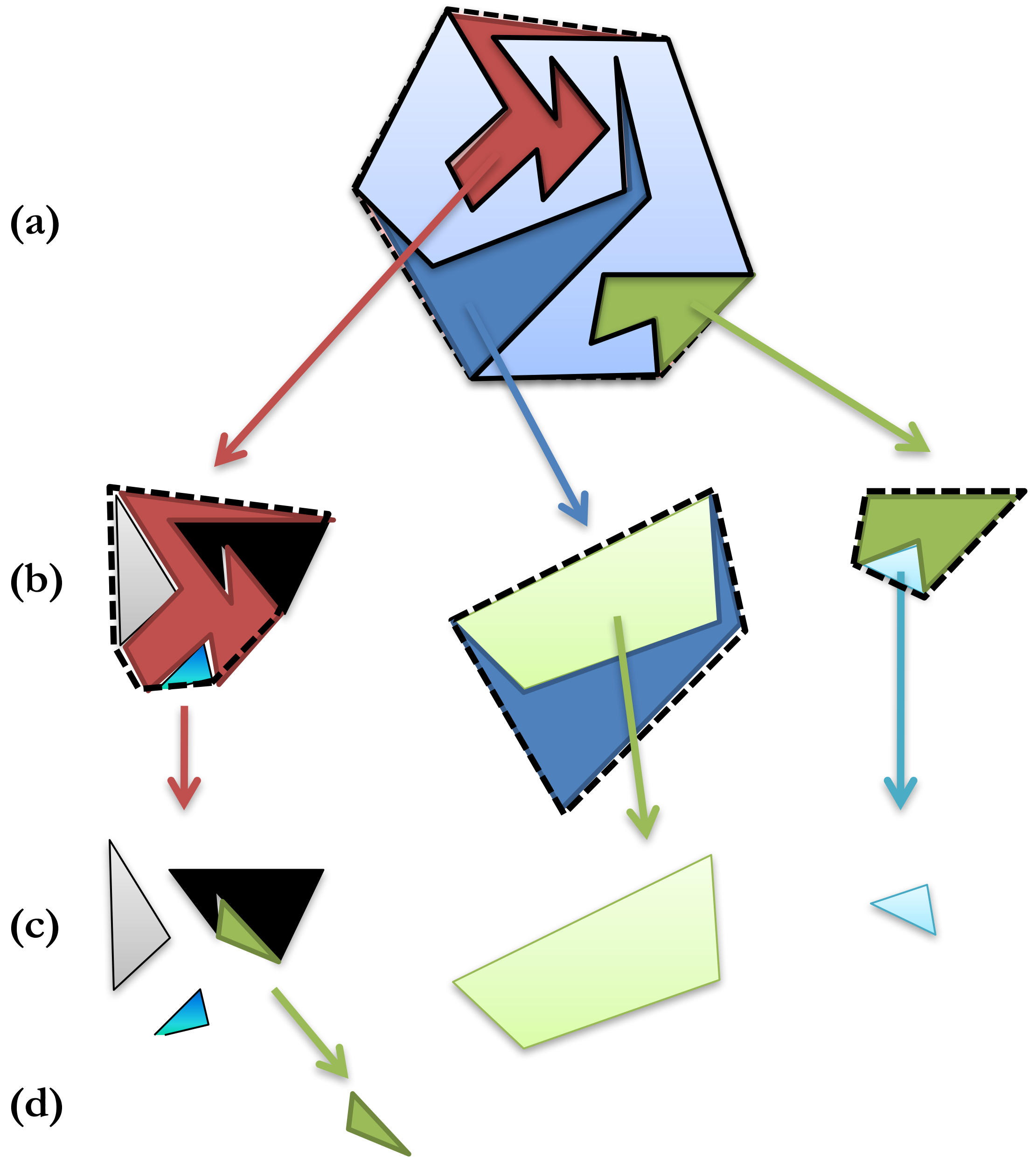
- Convex hull (or *the* hull) is perhaps the most commonly used structure in computational geometry.
- It can be used in various applications, let's mention a few here;
- Having a robot which tries to avoid obstacles as it moves, if its convex hull succeeded to avoid such obstacles, the robot itself won't hit any of them.
- Hence computing paths for robots which avoid collision becomes much easier if we are dealing with a convex robot rather than a non-convex one.



- In some applications, we need to find the rectangle with the smallest area which encloses a certain polygon.
- It was found that such rectangle has one of its sides coincides with the convex hull of the polygon.
- Hence computing the hull can be used as the first step of minimum rectangle algorithms.
- Likewise, finding the smallest 3-dimensional box surrounding certain object in space will definitely depend on computing the convex hull of such object.

- In the field of shape analysis, shapes can be classified for matching purposes by their *convex deficiency trees*, which are structures that mainly depend on finding convex hulls.



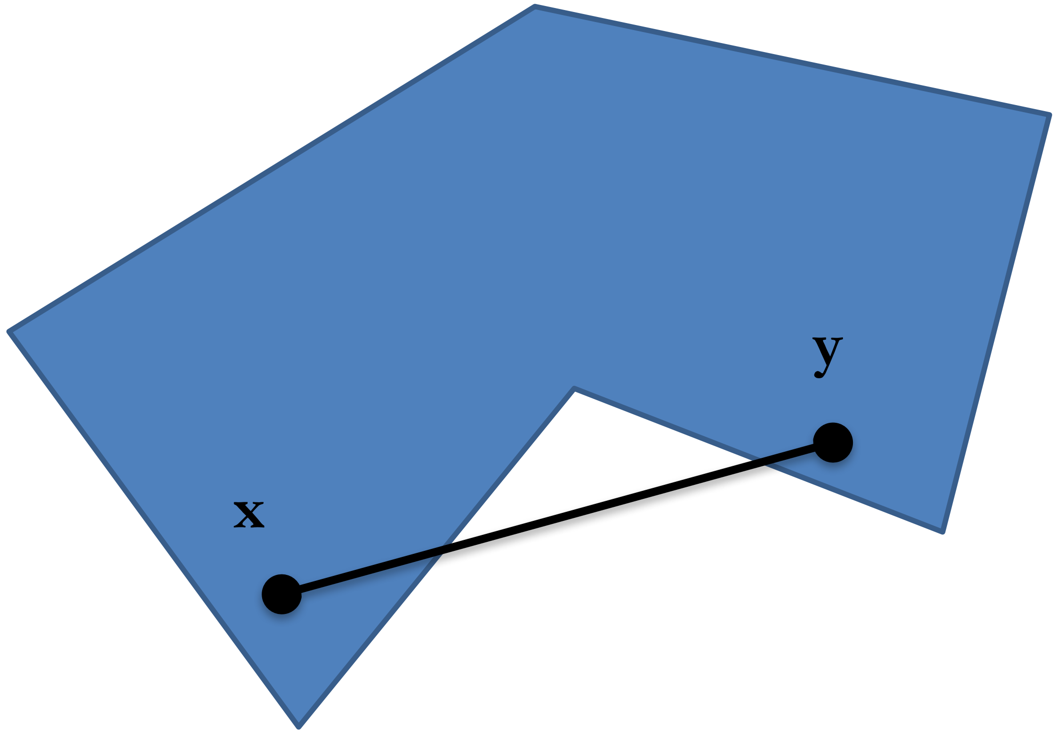
Example of convex deficiency tree (a) the polygon and its convex hull (b) deficiencies and their hulls (c) deficiencies of polygons from (b), (d) deficiency of the one non-convex piece from (c).

- Informally,
  - The convex hull of a set of points in the plane is the shape taken by a rubber band stretched around nails pounded into the plane at each point.
  - The boundary of the convex hull of points in three dimensions is the shape taken by plastic wrap stretched tightly around the points.
- Now let's move for more formal definitions for convexity and convex hulls.

Definitions of Convexity and Convex Hulls

Without being restricted to any particular dimension of the points, whether a set is connected, bounded or unbounded, closed or open, a primary definition of convexity can be given as follows;

Definition 12: *A set S is* **convex** *if $x \in S$ and $y \in S$ implies that the segment $xy \subseteq S$.*



It should be clear that for a given polygon, any region with a *dent* is not convex, since two points spanning the dent can be found such that the segment they determine contains points exterior to the polygon, hence any polygon with a reflex vertex is not convex.

Definition 13: *Let* x, y be two points in $\mathbb{R}^d$, a **segment** xy *joining these two points is the set of all points of the form* $\alpha x + \beta y$ *where* $\alpha \geq 0, \beta \geq 0$ *and* $\alpha + \beta = 1$.

For example, the midpoint of the segment is given by $\frac{1}{2}(x + y)$ with $\alpha = \beta = \frac{1}{2}$, where the end points are achieved with one weight set to zero and the other one set to one.

Definition 14: *A* **convex combination** *of points* $x_1, x_2, \ldots, x_n$ *is a sum of the form* $a_1x_1 + a_2x_2 + \cdots + a_nx_n$ *with* $a_i \geq 0$, *for* $i = 1,2, \ldots, n$ *and* $a_1 + a_2 + \ldots + a_n = 1$.

As a result of the Definitions 13 and 14, a line segment consists of all convex combinations of its endpoints, and a triangle consists of all convex combinations of its three comers. In three dimensions, a tetrahedron is the convex combinations of its four comers. Convex combinations lead to the concept of "barycentric coordinates".

There are various definitions for convex hull in literature;

Definition 15: *The* **convex hull** *, denoted by $\mathcal{H}(S)$, of a set of points S in d dimension:*

(1) is the set of all convex combinations of points of S. It is intuitively clear that the convex hull defined in this way cannot have a dent.

(2) is the set of all convex combinations of $\mathrm{d}+1$ (or fewer points) of S. Thus the hull of a 2-dimensional set is the convex combinations of its subsets of three points, each of which determine a triangle.

(3) Is the intersection of all convex sets that contain S.

(4) Is the intersection of all halfspaces that contain S, where a halfspace in two dimensions is a halfplane which is the set of points on or to one side of a line, hence a halfspace is the set of points on or to one side of a plane.

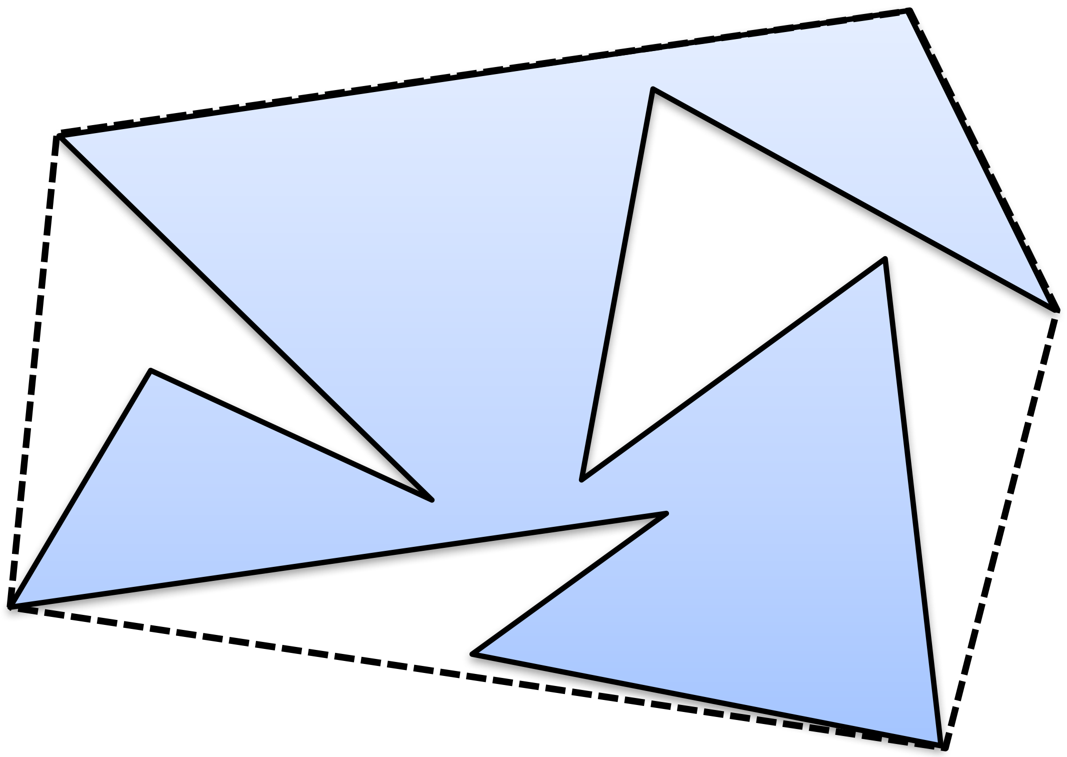
(5) In case of two-dimensions, it is the smallest convex polygon $\mathcal{P}$ that encloses S, smallest in the sense that there is no other polygon $\mathcal{P}'$ such that $S \subseteq \mathcal{P}' \subset \mathcal{P}$, that is it has the smallest area and perimeter.

(6) In case of two-dimensions, it is the union of all the triangles determined by points in S.

Note that the convex hull of a set is a closed "solid" region, including all the points inside.

Often the term is used more loosely in computational geometry to mean the boundary of this region, since it is the boundary we compute, and that implies the region.

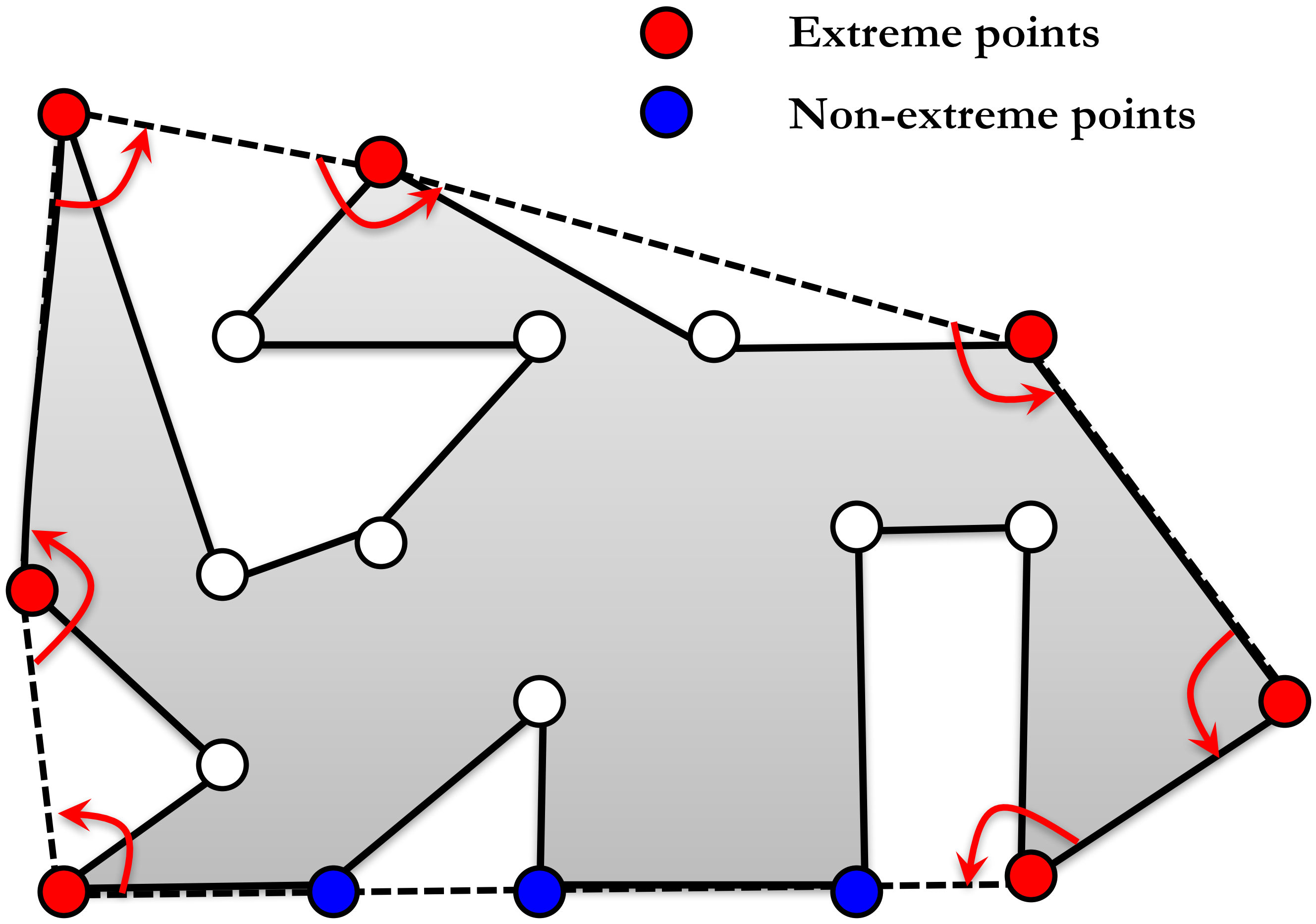
We will use the phrase "on the hull" to mean "on the boundary of the convex hull."



An example of a convex hull (dashed line) surrounding a given polygon

- We will be concentrating on algorithms for constructing the boundary of the convex hull of a finite set of points in two dimensions, starting from inefficient algorithms, gradually working towards an optimal algorithm.
- However, before starting talking about algorithms, we should first know what we expect from such algorithms, in particular what do we mean by constructing the boundary? Four outputs can be distinguished:
  - All points on the hull in arbitrary order
  - The extreme points in arbitrary order
  - All the points on the hull in boundary traversal order
  - The extreme points in boundary traversal order

Definition 16: *The* **extreme points** *of a set* S *of points in the plane are* <u>*the vertices of the convex hull*</u> *at which the interior angle is strictly convex, that is less than* π. *Hence points in the interior of a segment of the hull are not considered to be extreme. A more mathematical definition is that, a point* x *in* S *is extreme if there is no non-degenerate line segment in* S *that contains* x *in its relative interior.*

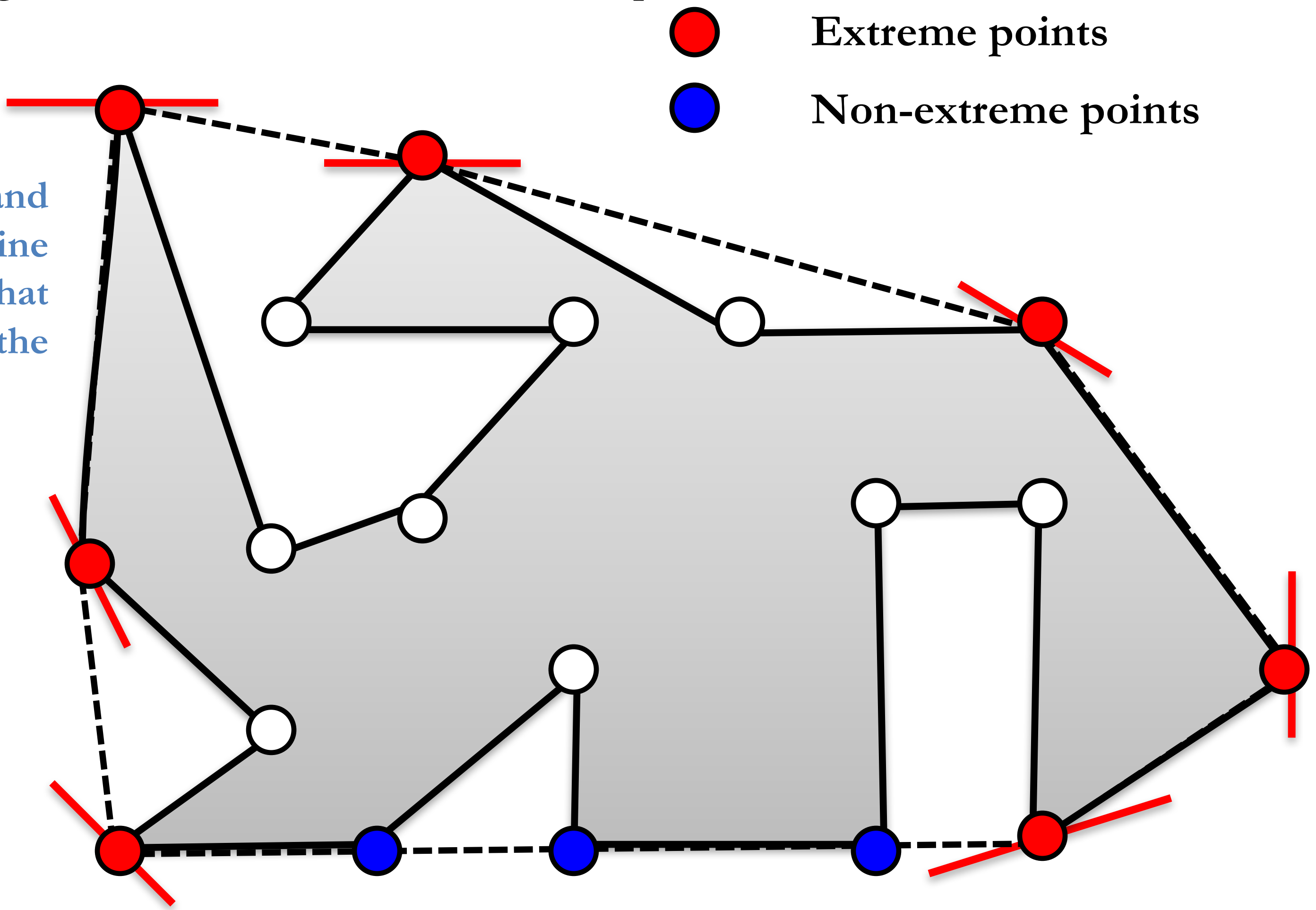


Extreme points are the vertices of the convex hull at which the interior angle is strictly convex, i.e. less than

- Let us concentrate on identifying the extreme points. Note that the highest point of S, the one with the largest y coordinate, is extreme if it is unique, or even if there are exactly two equally highest vertices (both can then be extreme). The same is of course true of the lowest points, the rightmost points, and the leftmost points.

- It should be clear that a point is extreme if and only if there exists a line through that point that otherwise does not touch the convex hull. Such "there exists" formulations, however, do not immediately suggest a method of find such extreme points.

A point is extreme if and only if there exists a line through that point that otherwise does not touch the convex hull.



Naïve Algorithms for Extreme Points

Let us therefore look at the other side of the coin, the non-extreme points.

Non-extreme Points

Logically, if we identify the non-extreme points, this would be enough to identify the extreme points, hence let's define non-extreme point

Lemma 8: *A point is* **non-extreme** *if and only if it is inside some (closed) triangle whose vertices are points of the set and is not itself a corner of that triangle.*

Proof:

Let's check all the possibilities; It is clear that if a point is interior to a triangle, it is non-extreme, in the mean time the corners of a triangle might be extreme. A point that lies on the boundary of a triangle but not at a corner is not extreme. Hence a point which lies inside a closed triangle and is not itself a corner to that triangle is indeed non-extreme.

□

Let $S = \{p_0, p_1, \dots, p_{n-1}\}$ be a set of distinct points in $\mathbb{R}^2$. Based on the preceding lemma, we can introduce the following algorithm to obtain the interior points of S, where the in-triangle test can be implemented by checking whether a point is on the left or collinear with another two points.

Algorithm: Interior Points

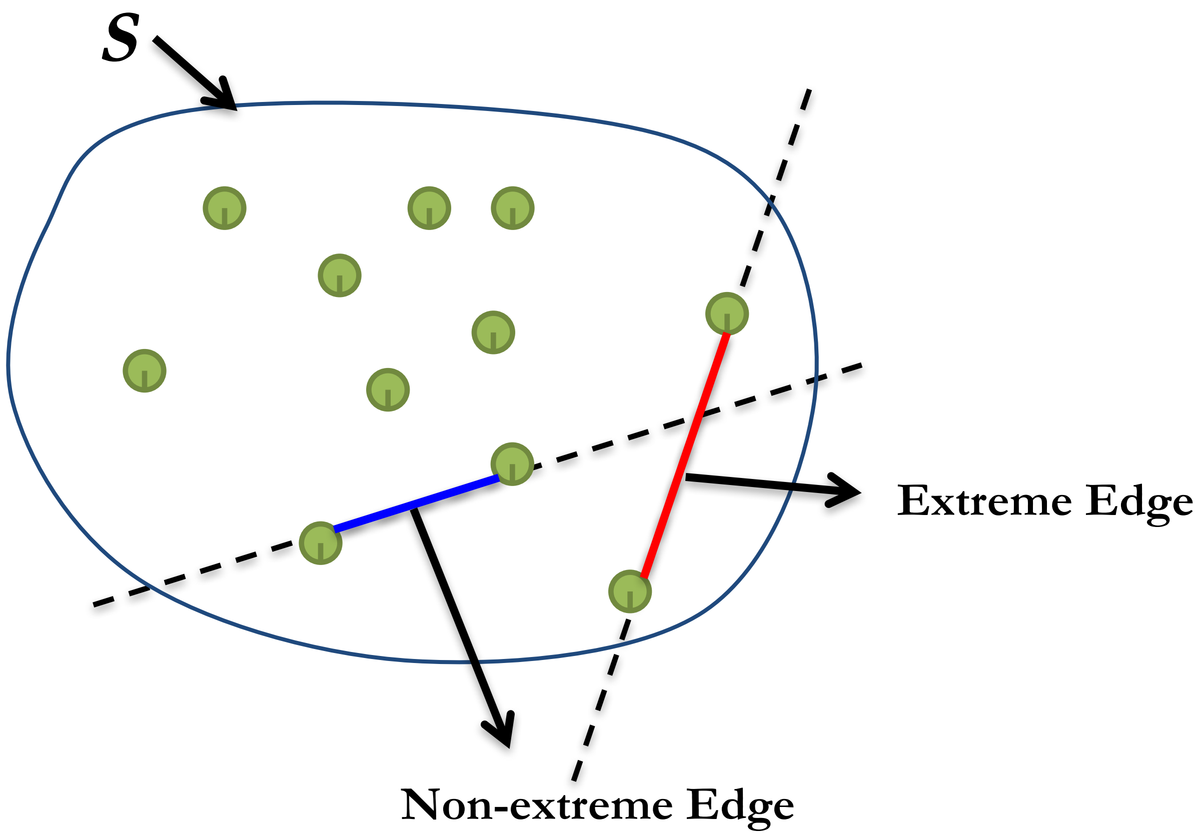
```
for each $i$ do
    for each $j \neq i$ do
        for each $k \neq i \neq j$ do
            for each $l \neq i \neq j \neq k$ do
                if $p_l \in \Delta(p_i, p_j, p_k)$ , then $p_l$ is non-extreme (interior) point
```

This algorithm clearly runs in *O(n^4)* time because there are four nested loops, each *O(n)*: For each of the n^3 triangles, the test for extremeness costs *n*. It would be a challenge to find a slower algorithm!

Extreme Edges

However, it is somewhat easier to identify extreme edges rather than non-extreme (interior) points, where extreme edges are edges of the convex hull.

Lemma 9: *An* **edge** *is* **extreme** *if every point of S is on or to one side of the line determined by the edge. Phrased negatively, a directed edge is* **not extreme** *if there is some point in S that does not lie to its left or on it.*



Let S be a set of distinct points, the red line is an extreme edge since every point of S is on or to one side of the line determined by that edge, on the other hand, the blue line is not an extreme edge since there are some points in S which lie on the right of the line determined by such edge.

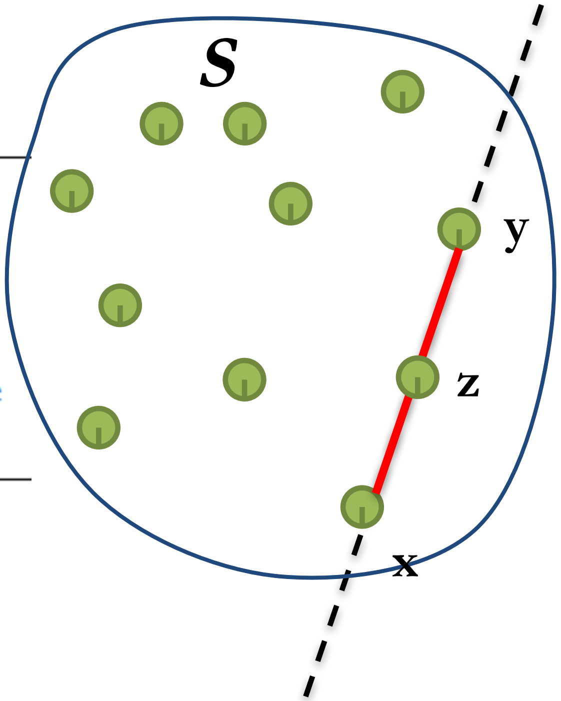
Unfortunately this algorithm computes points on the hull in arbitrary order (output (1)) rather than the extreme points in output (2).

Suppose that xy is an extreme edge, and z lies on the interior of the segment xy, i.e. $z \in xy$. Then xz and zy will both have the property that there is no point strictly to their rights – that is there is no point that is not left of or on xy.

Now x, y and z are considered points on the hull, however to determine whether they are extreme points or not, it would make sense to say that neither of these counts as an extreme edge and to demand that both endpoints of an extreme edge, be extreme vertices.

Algorithm: Extreme Edges

```
for each $i$ do
    for each $j \neq i$ do
        for each $k \neq i \neq j$ do
            if $p_k$ is not left or on $p_i p_j$ then $p_i p_j$ is not extreme
```



This algorithm runs in $O(n^3)$ time because there are three nested loops, each $O(n)$: For each of the n^2 pairs of points, the test for extremeness costs n. Which vertices are extreme can be found easily now (under the general position assumption, where there is no three points being collinear), since an extreme point is an endpoint of two extreme edges.

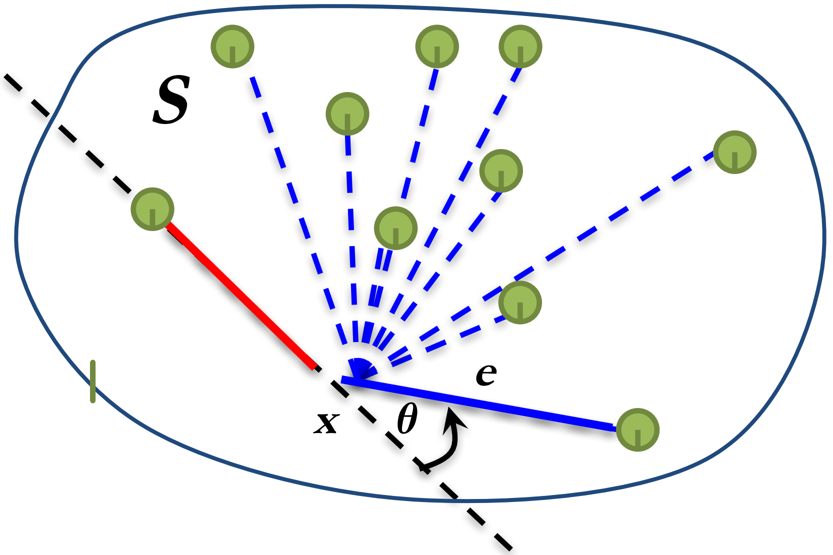
Gift Wrapping

Let’s now move to a more realistic hull algorithm. With a minor variation on the Extreme Edge algorithm, we can both accelerate it by a factor of *n* and at the same time output the points in the order in which they occur around the hull boundary.

Algorithm: Gift Wrapping

```
Find the lowest point (smallest $y$ coordinate)
Let $i_0$ be its index and set $i \leftarrow i_0$
Repeat
        for each $j \neq i$ do
                Compute counterclockwise angle $\theta$ from the previous hull edge
        Let $k$ be the index of the point with the smallest $\theta$.
        Output $p_i p_k$ as a hull edge
        set $i \leftarrow k$
until $i = i_0$
```

For many years such algorithm was the primary algorithm especially for high dimensions, however it was found that it is "output-size sensitive," in the sense that it runs faster when the hull is small: Its complexity is $O(nh)$ if the hull has h edges.



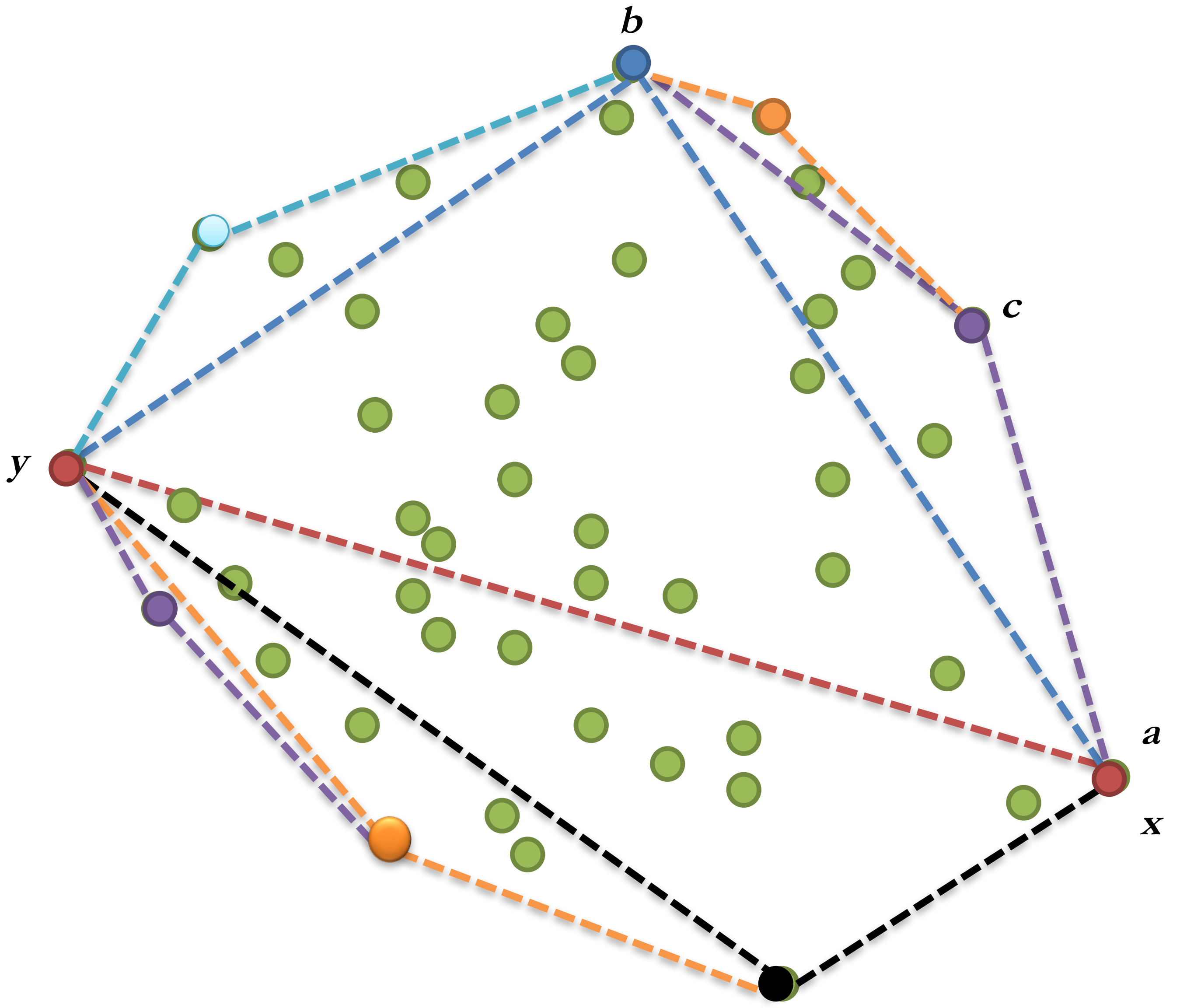
Let the red line be a detected extreme edge with x be the unlinked endpoint, we need to search for the next extreme edge e which will share the unlinked endpoint x. The next edge e makes the smallest angle θ with respect to the previous edge.

QuickHull

The basic idea is very simple; Intuitively, we can discard many points which are interior to the hull and then concentrate on those points which are closer to the hull boundary.

The first step of the QuickHull algorithm is to find two distinct extreme points; we will use the rightmost lowest and leftmost highest points x and y, which are guaranteed extreme and distinct (since every polygon must have at least one strictly convex vertex).

The full hull is composed of an "upper hull" above xy and a "lower hull" below xy. QuickHull finds these through a procedure that starts with extreme points (a, b), finds a third extreme point c strictly right of ab, discards all points Δabc, and operates recursively on $A = (a, c)$ and $B = (c, b)$.



QuickHull: two distinct extreme points are firstly found, we use the rightmost lowest (x) and the leftmost highest (y). Points are divided into those above xy from which the upper hull is obtained, and those below xy from which the lower hull is obtained. The upper and lower hulls are obtained in the following manner; start with extreme points (a, b) where $a = x$ and b belongs to the upper hull. Find a third extreme point c which is strictly to the right of ab, discards all points Δabc, and operates recursively on $A = (a, c)$ and $B = (c, b)$.

Graham's Algorithm

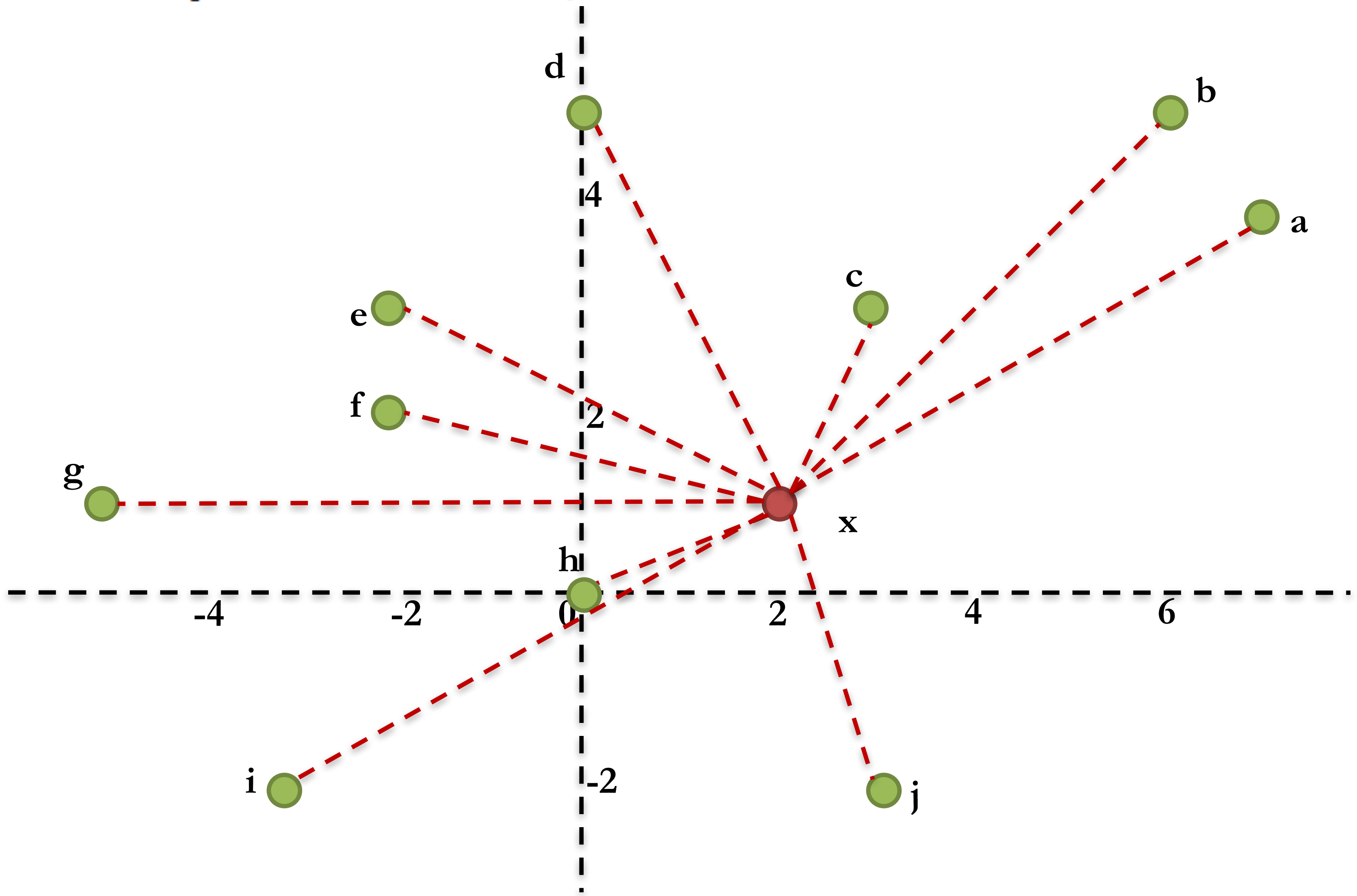
Now, it is time for the most efficient algorithm to compute the convex hull of a given set of points in plane which operates in $O(n\log n)$ time

Top Level Description

The basic idea of Graham's algorithm is quite simple. We will first explain it with an example, making several assumptions that will be removed later. Let’s assume the general position assumption where there is no three points in the given set are collinear.

Assume we are given a point x interior to the hull.

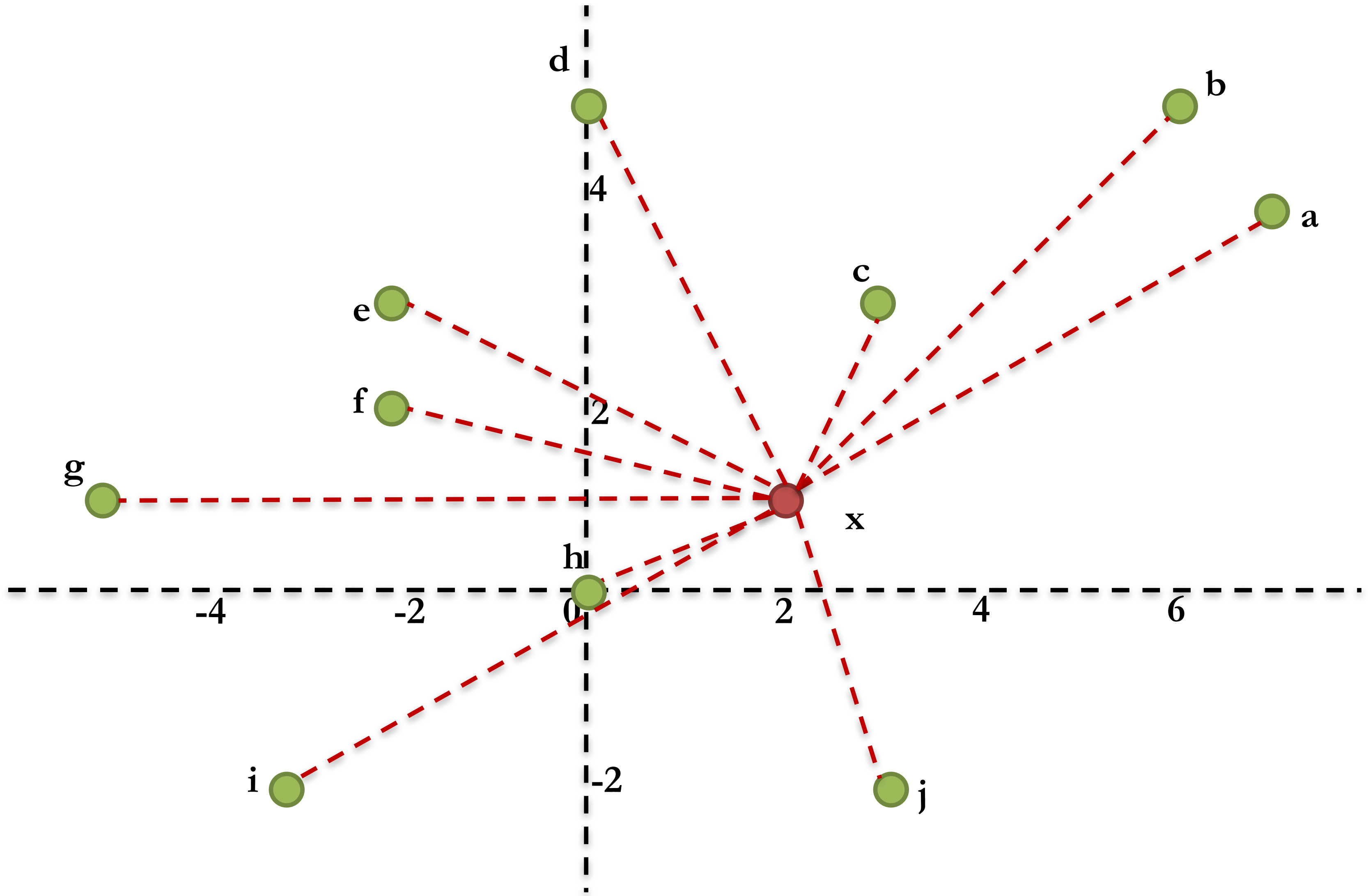
Now sort the points by angle, counterclockwise about x. For the example shown in the following figure, the sorted points are labeled $a, b, \ldots, j$.



Example for Graham's algorithms, x = (2,1); S= {(7,4),(6,5),(3,3),(0,5),(-2,3),(-2,2),(-5,1),(0,0),(-3,-2),(3,-2)}.

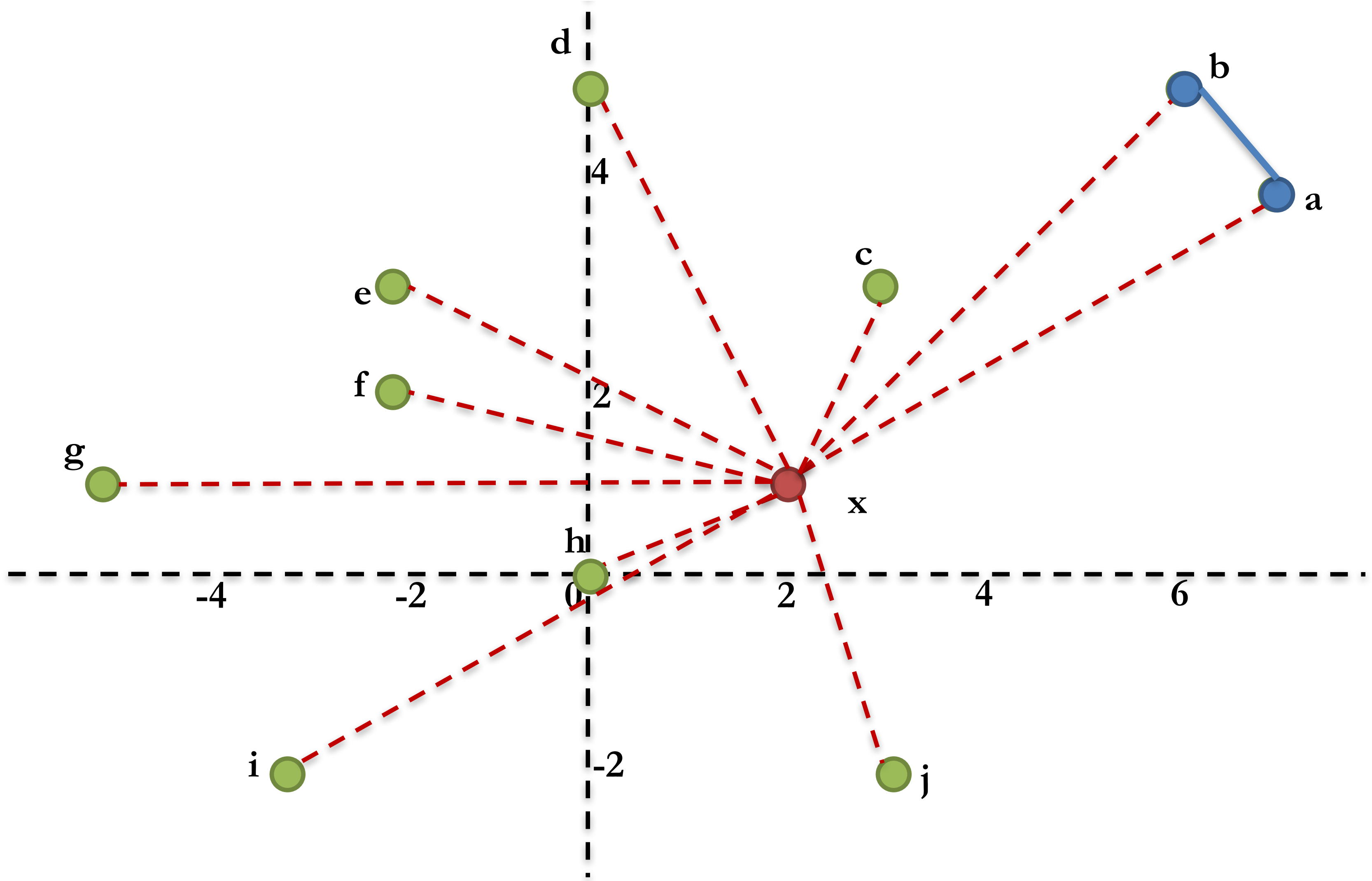
The points are now processed in their sorted order, and the hull grown incrementally around the set.

At any step, the hull will be correct for the points examined so far, but of course points encountered later will cause earlier decisions to be reevaluated.



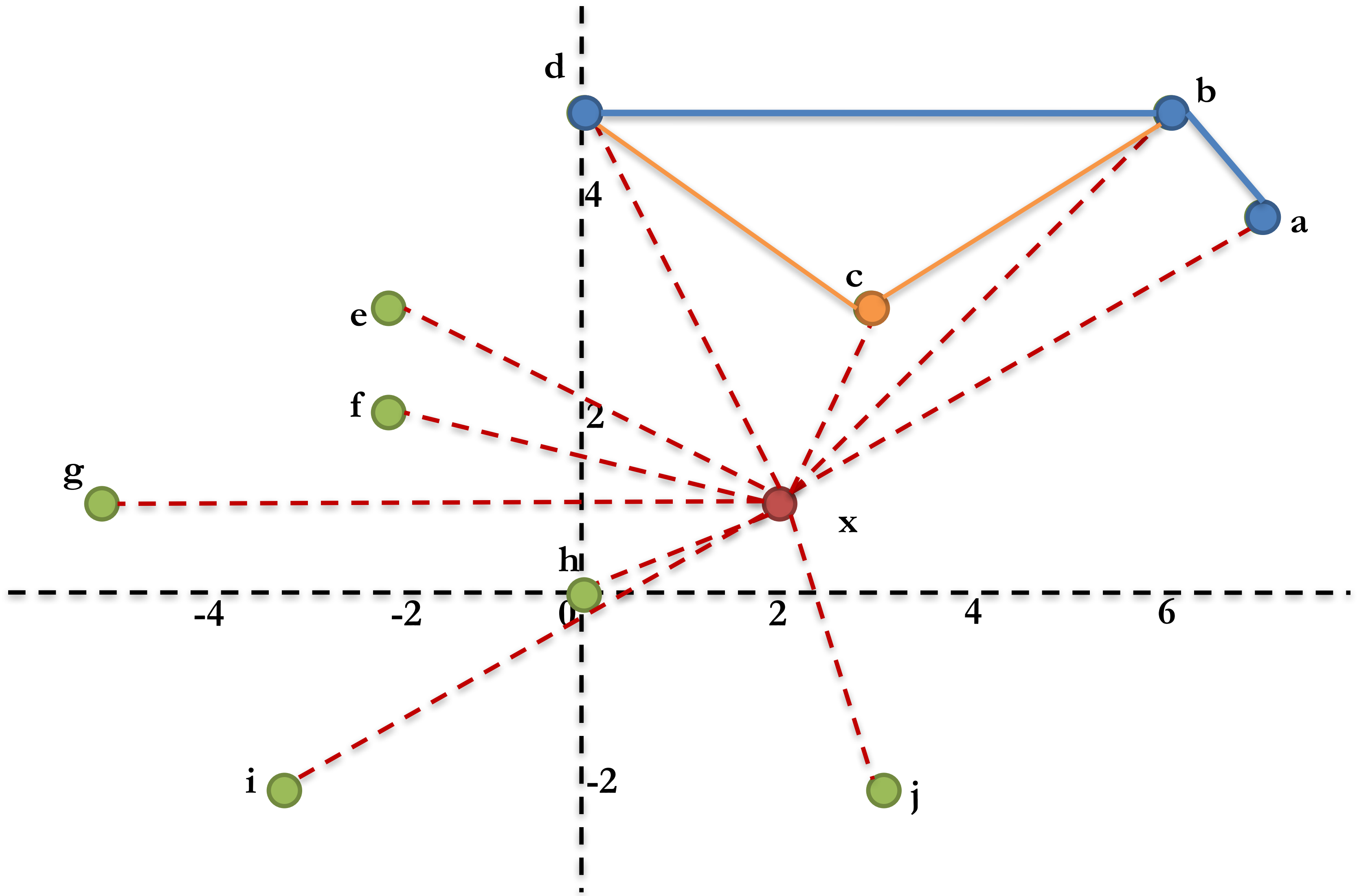
Example for Graham's algorithms, x = (2,1); S= {(7,4),(6,5),(3,3),(0,5),(-2,3),(-2,2),(-5,1),(0,0),(-3,-2),(3,-2)}.

The hull-so-far is maintained in a stack S of points. Initially the stack contains the first two points, $S = (b, a)$ in our example, with b on top. We will use the convention of listing the stack top to bottom, left to right. Point c is added because (a, b, c) forms a left turn at b, the previous stack top. Note that $S = (c, b, a)$ is a convex chain, a condition that will be maintained throughout.



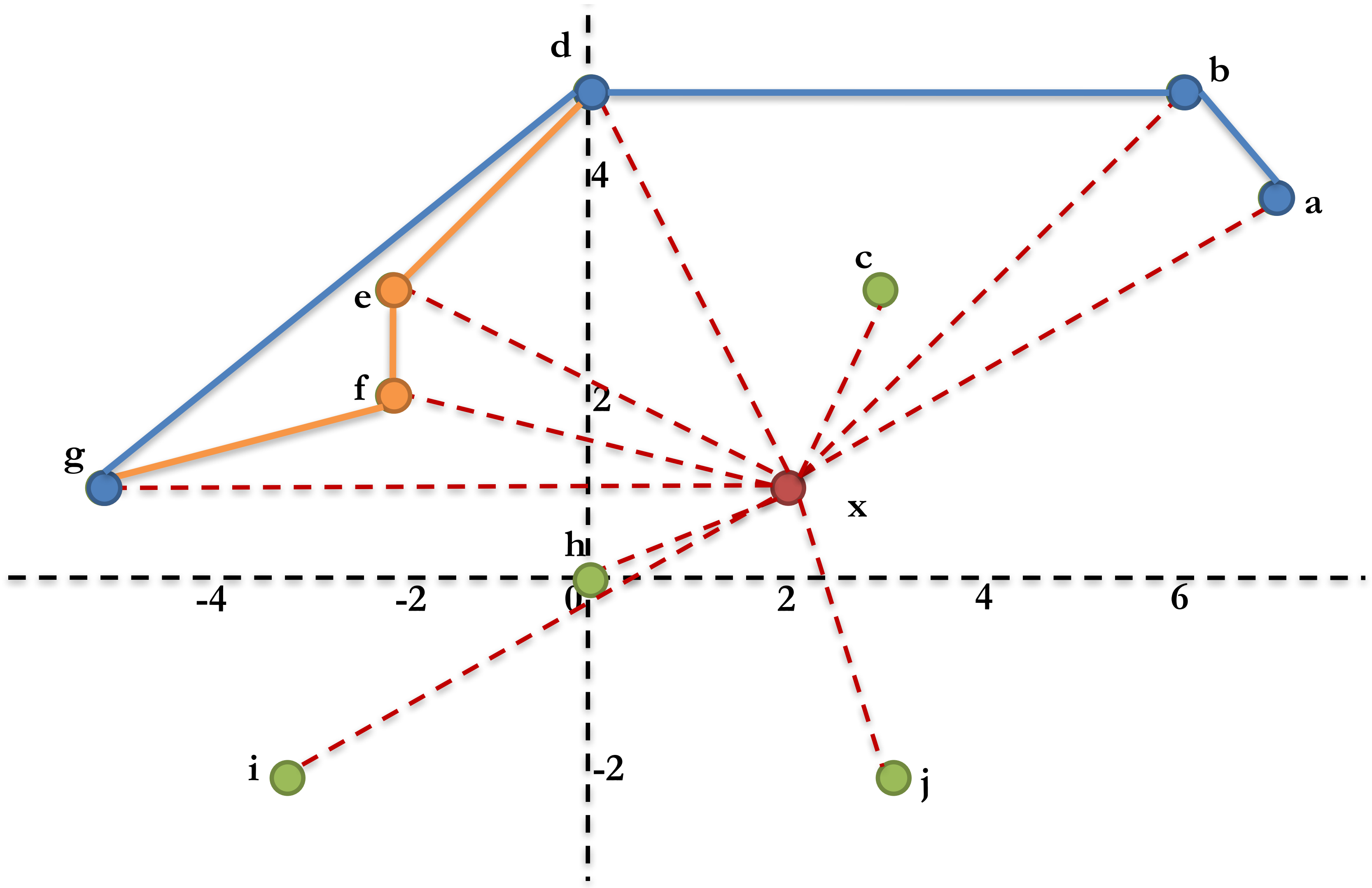
Example for Graham's algorithms, x = (2,1); S= {(7,4),(6,5),(3,3),(0,5),(-2,3),(-2,2),(-5,1),(0,0),(-3,-2),(3,-2)}.

Next point d is considered, but since (b, c, d) forms a right turn at the stack top c, the chain is not extended, but rather the last decision, to add c, is revoked by poping e from the stack, which then becomes $S = (b, a)$ again. Now d is added, because (a, b, d) forms a left turn at b.



Example for Graham's algorithms, x = (2,1); S= {(7,4),(6,5),(3,3),(0,5),(-2,3),(-2,2),(-5,1),(0,0),(-3,-2),(3,-2)}.

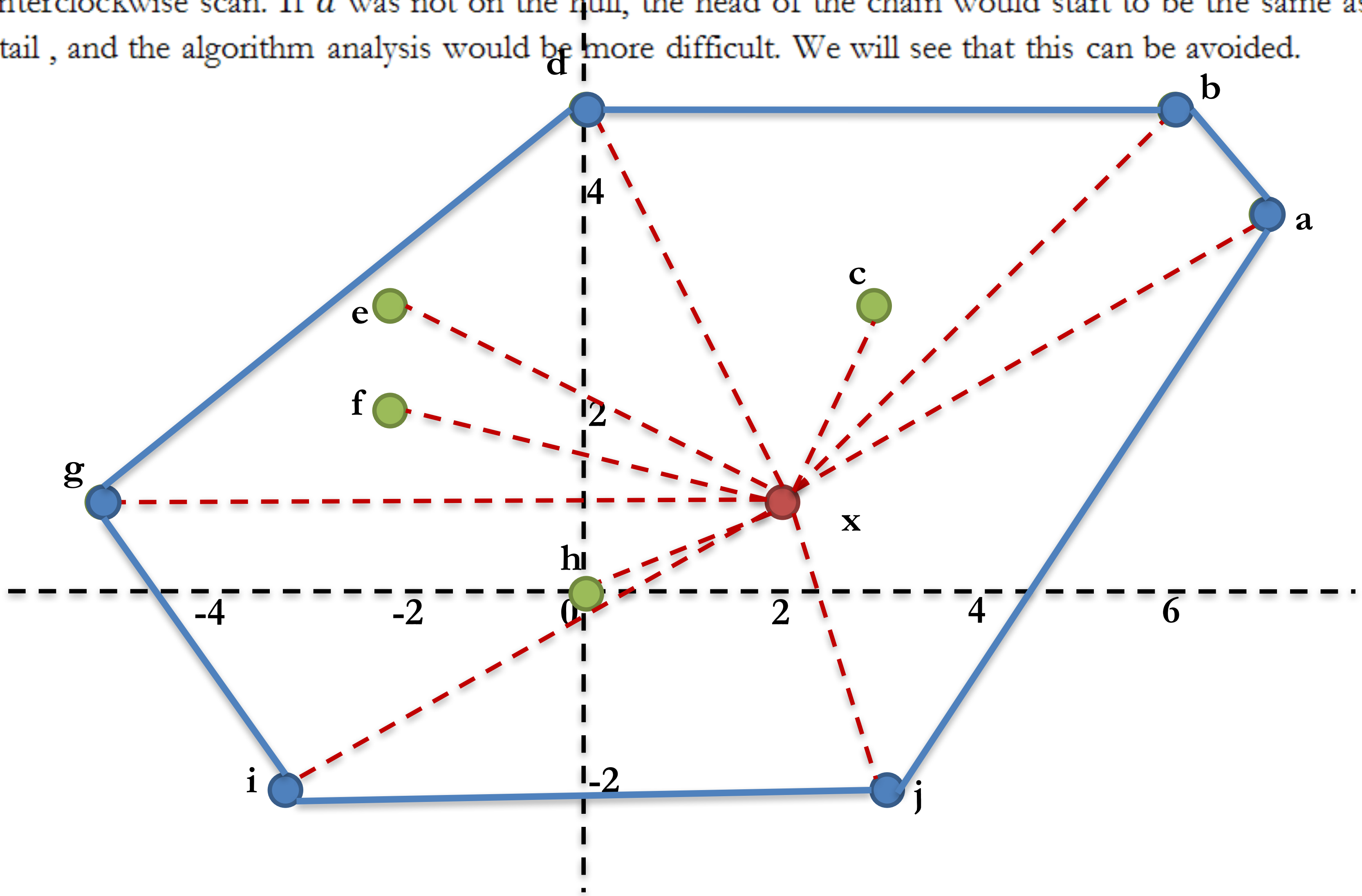
Continuing in this manner, e and f are added, after which the stack is $S = (f, e, d, b, a)$. Point g causes f and then e to be deleted, since both (e, f, g) and (d, e, g) are right turns. Then g can be added, and the stack is $S = (g, d, b, a)$. And so on.



Example for Graham's algorithms, x = (2,1); S= {(7,4),(6,5),(3,3),(0,5),(-2,3),(-2,2),(-5,1),(0,0),(-3,-2),(3,-2)}.

If we are as fortunate as in the considered example and our first point a is on the hull, the convex chain will close naturally, resulting in the final hull $S = (j, i, g, d, b, a)$.

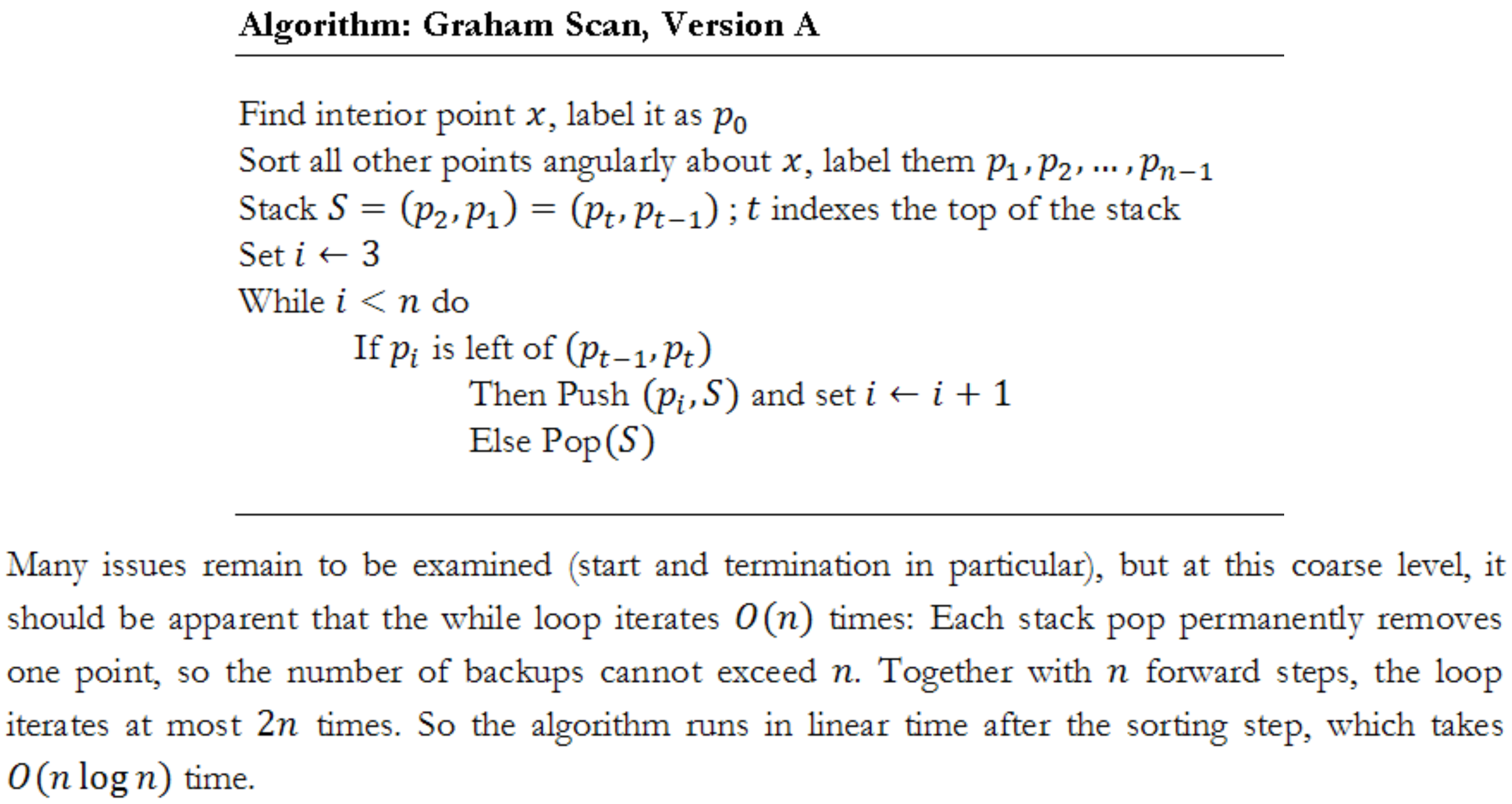
Note that from stack top to bottom represents a clockwise traversal, as we built it up via counterclockwise scan. If a was not on the hull, the head of the chain would start to be the same as the tail , and the algorithm analysis would be more difficult. We will see that this can be avoided.



Example for Graham's algorithms, x = (2,1); S= {(7,4),(6,5),(3,3),(0,5),(-2,3),(-2,2),(-5,1),(0,0),(-3,-2),(3,-2)}.

Pseudo code, Version A

- Let's now summarize the rough algorithm in the following pseudo code, where we assume stack primitives *Push(p, S)* and *Pop(S)*, which push *p* onto the top of the stack *S*, and pop the top off, respectively. We use *t* to index the stack top and *i* for the angularly sorted points.



Algorithm: Graham Scan, Version A

```
Find interior point $x$, label it as $p_0$
Sort all other points angularly about $x$, label them $p_1, p_2, \dots, p_{n-1}$
Stack $S = (p_2, p_1) = (p_t, p_{t-1})$ ; $t$ indexes the top of the stack
Set $i \leftarrow 3$
While $i < n$ do
        If $p_i$ is left of $(p_{t-1}, p_t)$
                Then Push $(p_i, S)$ and set $i \leftarrow i + 1$
                Else Pop$(S)$
```

Many issues remain to be examined (start and termination in particular), but at this coarse level, it should be apparent that the while loop iterates $O(n)$ times: Each stack pop permanently removes one point, so the number of backups cannot exceed n. Together with n forward steps, the loop iterates at most $2n$ times. So the algorithm runs in linear time after the sorting step, which takes $O(n \log n)$ time.

Details: Boundary Conditions

Now it is time to discuss details related to various boundary conditions which have been ignored so far.

Start and Stop of Loop

The algorithm so far presented might have trouble at either the starting of the loop or its end. We already mentioned the termination difficulties that would arise if a, the stack bottom, was not on the hull. Startup difficulties occur when b, the second point pushed on the stack, is not on the hull too.

Suppose that (a, b, c) is a right turn. Then b would be popped from the stack, and the stack reduced to $S = (a)$. But at least two points are needed to determine if a third forms a left turn with the stack top.

Clearly both startup and stopping problems are avoided if both a and b are on the hull. How this can be arranged will be shown in the next subsection.

Sorting Origin

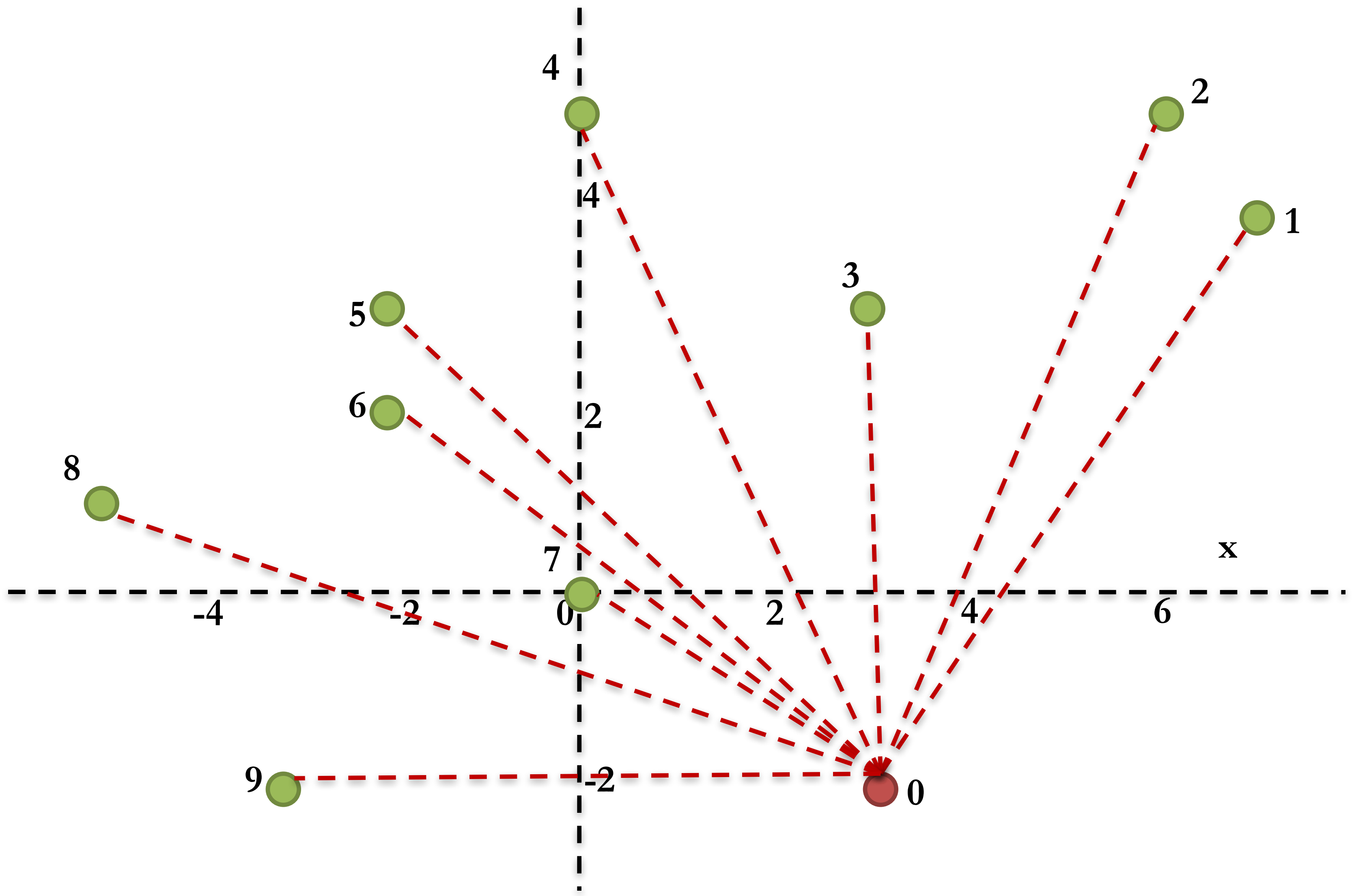
We assumed that the point x, about which all others are sorted, is interior to the hull.

However computing such an interior point is unnecessary, it also may force the use of floating-point numbers even when the input coordinates are all integers. We would like to avoid all floating-point calculations to guarantee a correct answer on integer input.

A simplification is to sort with respect to a point of the set, and in particular, with respect to a point on the hull. We will use the lowest point, which is clearly on the hull. In case there are several with the same minimum *y* coordinate, we will use the rightmost of the lowest as the sorting origin.

Now we are prepared to solve the startup and termination problems discussed above. If we sort points with respect to their counterclockwise angle from the horizontal ray emanating from our sorting origin p_0, then p_1 must be on the hull, as it forms an extreme angle with p_0. However, it may not be an extreme point.

If we initialize the stack to $S = (p_0, p_1)$, the stack will always contain at least two points, avoiding startup difficulties, and will never be consumed when the chain wraps around to p_0 again, avoiding termination difficulties.



New sorting origin for the points in the previous example, where the lowest rightmost point is considered to be our origin around which other points are angularly sorted.

Collinearities

The final "boundary condition" we consider is the possibility that three or more points are collinear. This issue affects several aspects of the algorithm. First we focus on defining precisely what we seek as output.

Hull Collinearities.

We insist here on the most useful output (4): the extreme vertices only, ordered around the hull. Thus if the input consists of the corners of a square, together with points in-between such corners along the square's boundary, the output should consist of just the four corners of the square. Avoiding non-extreme hull points is easily achieved by requiring a *strict* left turn (p_{t-1}, p_t, p_i) to push p_i onto the stack, where p_t and p_{t-1} are the top two points on the stack. Then if p_t is collinear with p_{t-1} and p_i it will be deleted.

Sorting Collinearities.

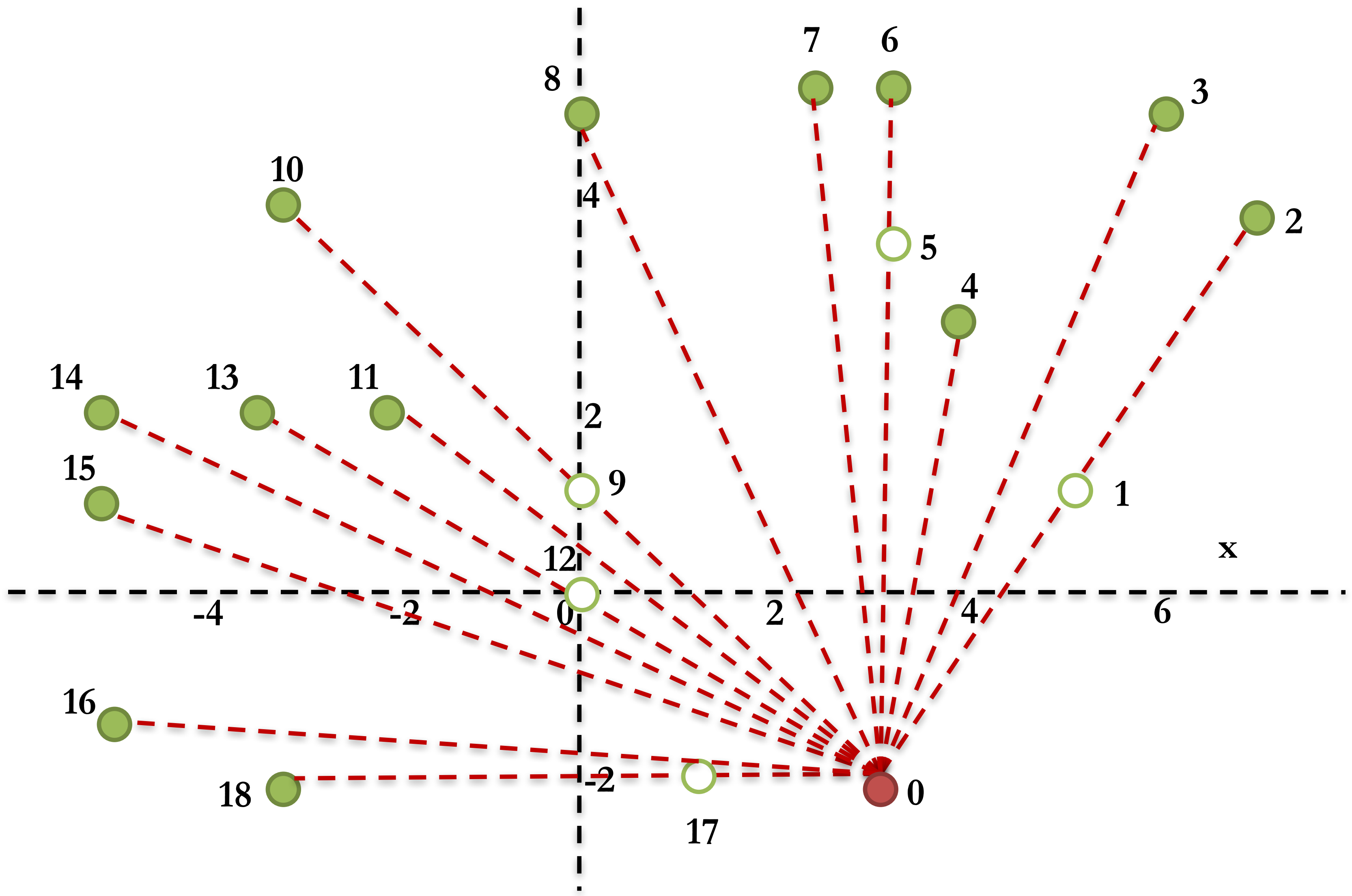
Collinearities raise another issue: How should we break ties in the angular sorting if both points a and b form the same angle with p_0? There are at least two options.

First, use a consistent sorting rule, and then ensure start and stop and hull collinearities are managed appropriately. A reasonable rule is that if angle(a) = angle(b), then define $a < b$ if $|a - p_0| < |b - p_0|$. Closer points are treated as earlier in the sorting sequence.

We choose here a second option. It is based on this simple observation: If angle(a) = angle(b) and $a < b$ according to the above sorting rule, then a is not an extreme point of the hull and may therefore be deleted.

Coincident Points.

Often code that works on distinct points crashes for sets that may include multiple copies of the same point. We will see that we can treat this issue as a special case of a sorting collinearity, deleting all but one copy of each point.



Sorting points with collinearities, indices indicate sorting rank, points to be deleted are shown as open circles.

Pseudo code, Version B

Algorithm: Graham Scan, Version B

```
Find rightmost lowest point, label it as $p_0$
Sort all other points angularly about $p_0$,
        In case of tie, delete the point closer to $p_0$ (or all but one copy for multiple points)
Stack $S = (p_1, p_0) = (p_t, p_{t-1})$ ; $t$ indexes the top of the stack
Set $i \leftarrow 2$
While $i < n$ do
        If $p_i$ is strictly left of $(p_{t-1}, p_t)$
                Then Push $(p_i, S)$ and set $i \leftarrow i + 1$
                Else Pop$(S)$
```

Incremental Algorithm

The difficulty is that Graham's algorithm has no obvious extension to three dimensions: It depends crucially on angular sorting, which has no direct counterpart in three dimensions. So we now proceed to describe one further algorithm in two dimensions which can be extended to three dimensions.

The algorithm is very straightforward: The incremental algorithm.

Its basic plan is simple: Add the points one at a time; at each step, construct the hull of the first k points and use that hull to incorporate the next point. It turns out that "factoring" the problem this way simplifies it greatly, in that we only have to deal with one very special case: adding a single point to an existing hull.

Let $P = \{p_0, p_1, \dots, p_{n-1}\}$ be our set of points, and assume for simplicity that the points are in general position, that is no three of them are collinear. The following outline highlights the high-level structure of this algorithm.

Algorithm: Incremental Algorithm

```
Let $\mathcal{H}_2 = chull\{p_0, p_1, p_2\}$
For $k = 3$ to $n - 1$ do
    $\mathcal{H}_k = chull\{\mathcal{H}_{k-1} \cup p_k\}$
```

The first hull is the triangle $chull\{p_0, p_1, p_2\}$. Let $Q = \mathcal{H}_{k-1}$ and $p = p_k$. The problem of computing $chull\{Q \cup p\}$ falls into two cases, depending on whether $p \in Q$ or $p \notin Q$. (we could ensure that $p \notin Q$ by presorting the $p_i's$ by x-coordinate for example).

Case 1: $p \in Q$

Once p is determined to be in Q, it can be discarded.

Note that we can discard p even if it is on the boundary of Q if we assume that we only want extreme points.

Although there are several ways to decide if $p \in Q$, perhaps the most robust way is to use LeftOn, i.e. $p \in Q$ if and only if p is left of or on every directed edge in Q.

Apparently this test takes time linear in the number of vertices of Q. Note that this method only works for convex polygon Q which is all we need here, however more general point-in-polygon algorithms can be used.

Case 2: $p \notin Q$

If any LeftOn test returns false, then $p \notin Q$ and we have to compute $chull\{Q \cup p\}$. What makes this task relatively easy is that we need only find the two lines of tangency from p to Q and modify the hull accordingly.

Our general position assumption assures that each line of tangency between p and Q touches Q at just one point. Suppose p_i is one such point of tangency, how can we find it?

The following figure shows that we can use the results of the LeftOn tests to determine tangency.

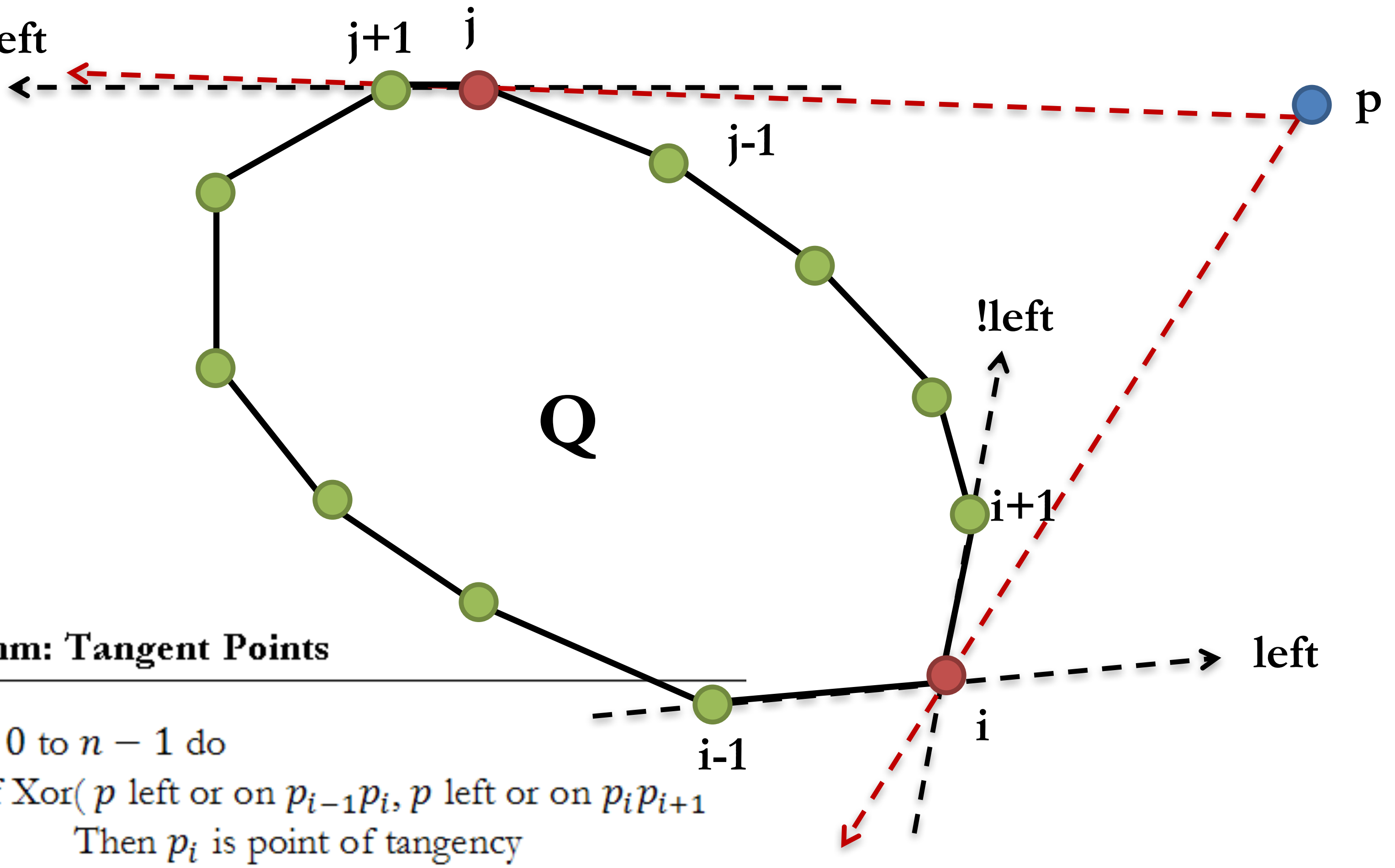
For the lower point of tangency p_i, p is left of $p_{i-1}p_i$ but right of $p_i p_{i+1}$.

For the upper point of tangency p_j, the sense is reversed, i.e. p is right of $p_{j-1}p_j$ but left of $p_j p_{j+1}$.

Both cases can be captured with the exclusive-Or function: that is p_i is a point of tangency if two successive edges yield different LeftOn results. Thus the two points of tangency can be identified via the same series of LeftOn tests used to decide if $p \in Q$.

It only remains to form the new hull, the new hull is $\{p_0, p_1, \ldots, p_{i-1}, p_i, p, p_j, p_{j+1}, \ldots, p_{n-1}\}$. If the hulls are represented by linked lists, this update can be accomplished by a simple sequence of insertion and deletions.

Tangent lines from p to Q, 'left' means that p is left of the indicated directed line, and '!left' mean 'not left'



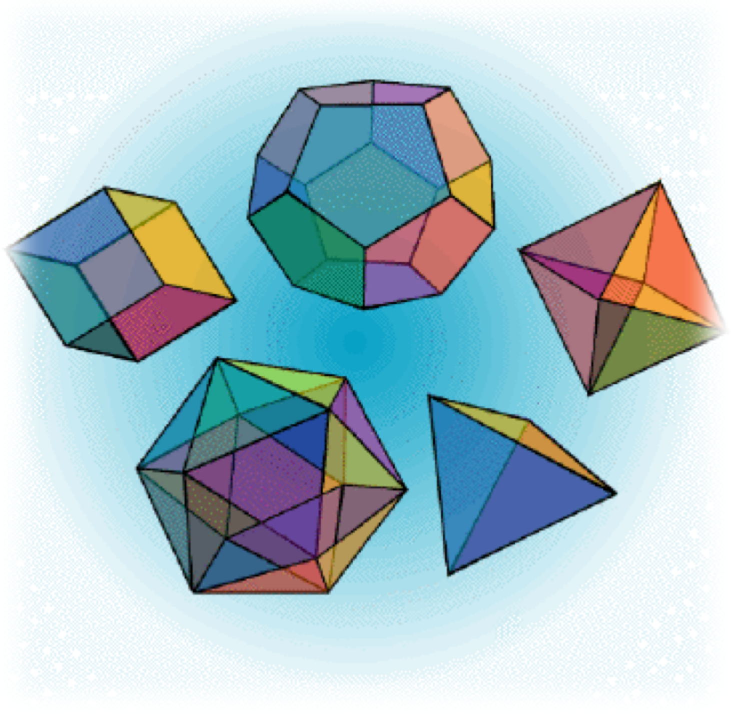
Algorithm: Tangent Points

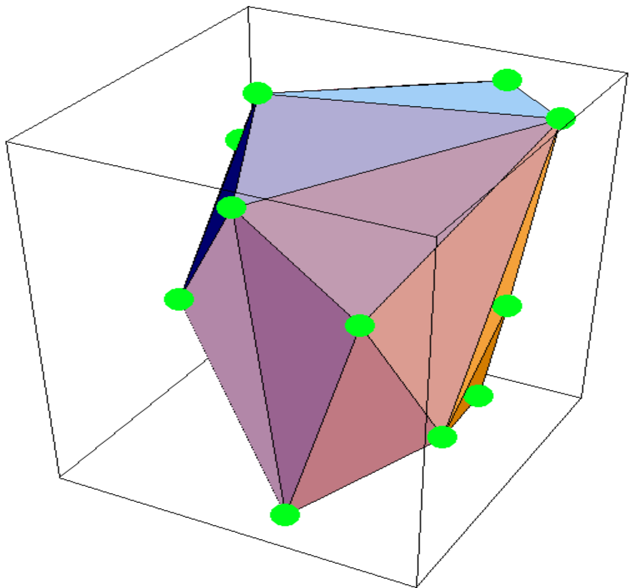
```
For $i = 0$ to $n - 1$ do
        If Xor( $p$ left or on $p_{i-1}p_i$, $p$ left or on $p_ip_{i+1}$
                Then $p_i$ is point of tangency
```

The complexity analysis of this algorithm is simple: The work at each step is O*(n);* more precisely, it is proportional to the number of vertices of the *k*th hull. In the worst case we would have : $p \notin Q$ at each step, resulting in total work proportional to 3 +4 +... +*n* = $O(n^2)$. It turns out that with only a little more effort, the time complexity can be reduced to $O(n \log n)$.

Flavor of Computational Geometry

Convex Hull in 3D





Shireen Y. Elhabian

Aly A. Farag

University of Louisville

February 2010

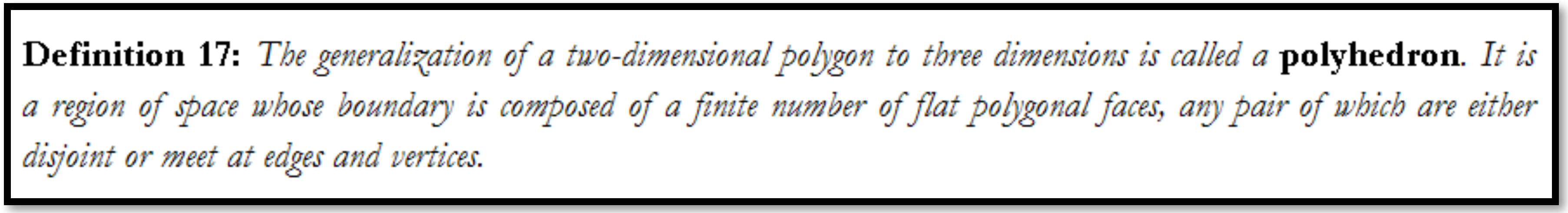
Agenda

- Polyhedra
  - Introduction
  - Regular Polytopes
  - Euler’s Formula
- Incremental Algorithm in 3D

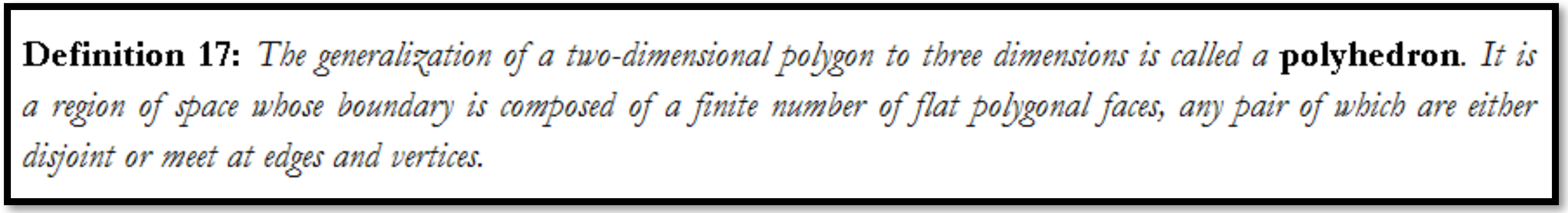
Polyhedra

The focus of this section is discussing algorithm for constructing the convex hull of a set of points in three dimensions. We will also touch basis on some properties of polyhedra and how to represent polyhedra.

Introduction


Definition 17: *The generalization of a two-dimensional polygon to three dimensions is called a* **polyhedron**. *It is a region of space whose boundary is composed of a finite number of flat polygonal faces, any pair of which are either disjoint or meet at edges and vertices.*

- In this section we are mainly concerned with <u>convex polyhedra</u>, which are simpler than general polyhedra.
- The boundary or surface of a polyhedron is composed of three types of geometric objects:
  - zero-dimensional vertices (0-cells, i.e. points),
  - one-dimensional edges (1-cells, i.e. segments), and
  - two-dimensional faces (2-cells, i.e. polygons).
- Faces can be convex polygons, which are defined to be bounded, without losing generality, since non-convex polygons/faces can be partitioned into convex ones, however we must then allow adjacent faces to share the same plane, i.e. coplanar.

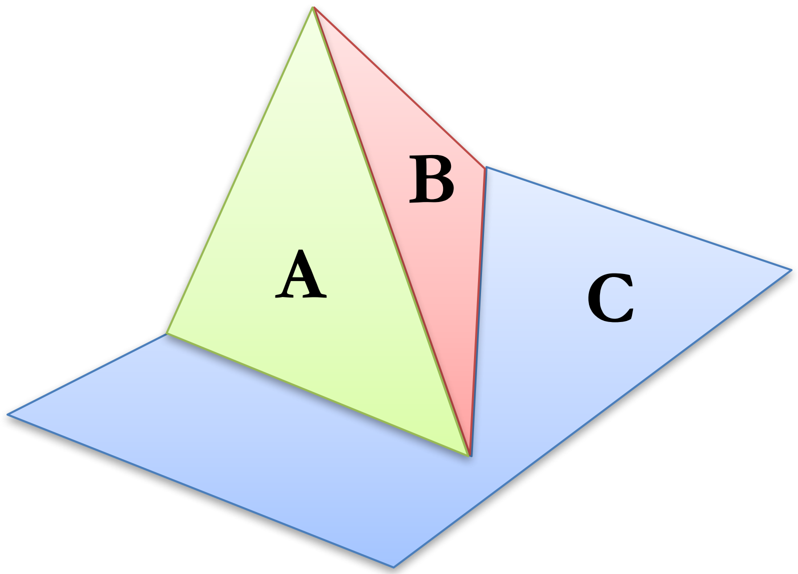


Definition 17: *The generalization of a two-dimensional polygon to three dimensions is called a* **polyhedron**. *It is a region of space whose boundary is composed of a finite number of flat polygonal faces, any pair of which are either disjoint or meet at edges and vertices.*

- There are certain conditions which should be satisfied to construct a valid polyhedral surface, such conditions defines how polyhedral surface components are related to each other. We have three types of conditions:
  - The components intersect "properly,"
  - the local topology is "proper," and
  - the global topology is "proper."
- Now let’s discuss these conditions in details

1. Components intersect "properly."

- For each pair of faces, we require that either
  - they are disjoint, or
  - they have a single vertex in common, or
  - they have two vertices, and the edge joining them, in common.
- Having convex faces simplifies these conditions.
- Improper intersections include not only penetrating faces, but also faces touching in the "wrong" way.



Faces A and B meet face C improperly even though they do not penetrate C

- There is no need to specify conditions on the intersection of edges and vertices, as the condition on faces covers them also. Thus an improper intersection of a pair of edges implies an improper intersection of faces.

2. Local topology is "proper."

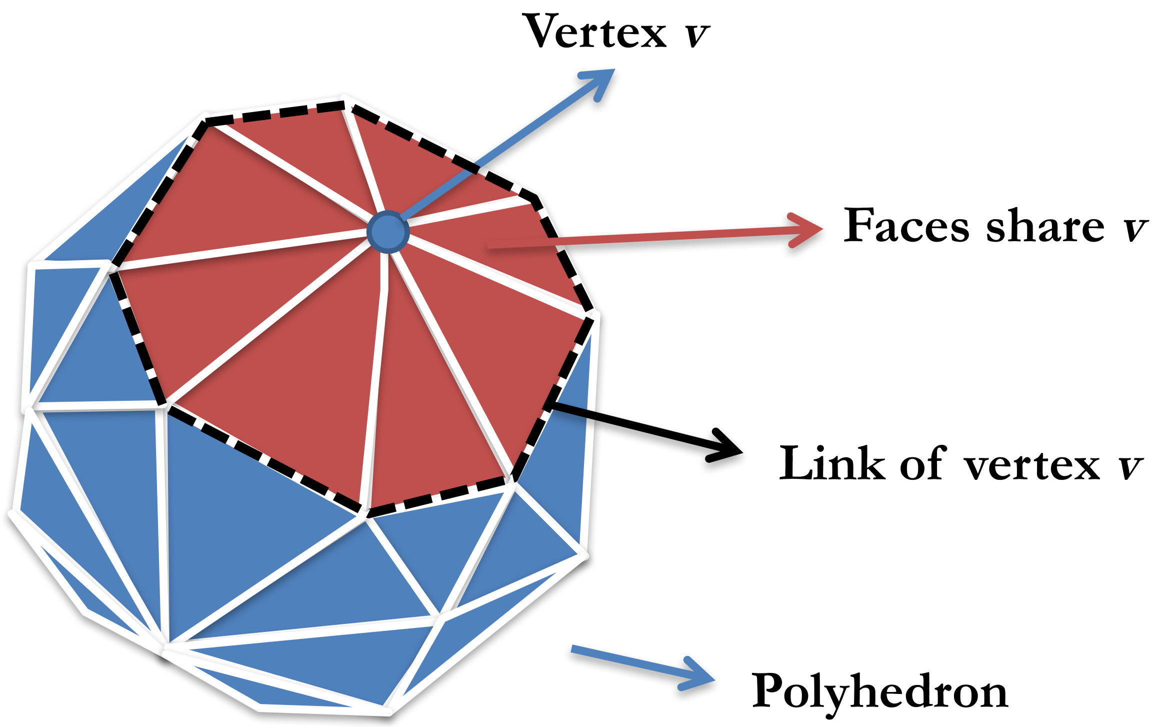
- Geometrically, the local topology is what the surface looks like in the vicinity of a point.
- This notion can be made precise via the notion of *neighborhoods* which are arbitrarily small portions (open regions) of the surface surrounding a point.
- The points should have neighborhoods that are topological two-dimensional disks, i.e. neighborhoods of every point on the surface is "homeomorphic" to a disk, where a *homeomorphism* between two regions permits stretching and bending, but no tearing.
- A bug walking on the surface would find the neighborhood of every point to be topologically like a disk.
- A surface for which every point have a neighborhood homeomorphic to an open disc is called a *2-manifold,* which is a class more general than the boundaries of polyhedra.

2. Local topology is "proper."

- Moving to a combinatorial description to this condition, suppose we triangulate the polygonal faces, then every vertex is the apex of a number of triangles. Let's define the *link* of a vertex as follows;

Definition 18: *The* **link** *of a vertex v is the collection of edges opposite to v in all the triangles incident to v. Thus the link is in a sense the combinatorial neighborhood of v. For a legal triangulated polyhedron, we require that the link of every vertex be a simple, closed polygonal path (polygonal chain).*

- One consequence of this condition is that every edge is shared by exactly two faces.

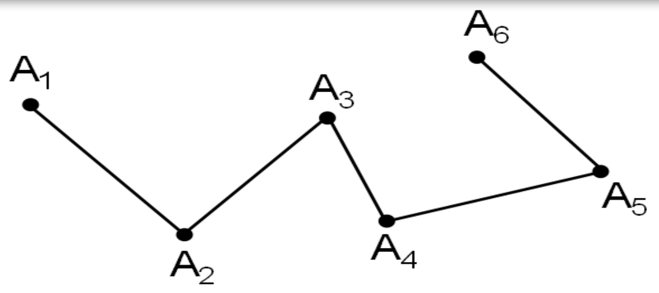


An example of a valid polyhedral surface, where the link of a vertex *v* is defined to be the collection of edges opposite to *v* in all the triangles incident to *v*. Thus the link is in a sense the combinatorial neighborhood of *v*. For a legal triangulated polyhedron, we require that the link of every vertex be a simple, closed polygonal path.

Definition 19: *A* **polygonal chain**, **polygonal curve**, **polygonal path**, *or* **piecewise linear curve**, *is a connected series of line segments. More formally, a polygonal chain P is a curve specified by a sequence of points* $(A_1, A_2, \dots, A_n)$ *called its* **vertices** *so that the curve consists of the line segments connecting the consecutive vertices.*

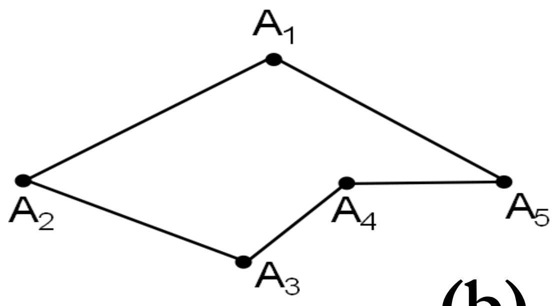
Definition 20: *A* **simple polygonal chain** *is one in which only consecutive (or the first and the last) segments intersect and only at their endpoints.*

Definition 21: *A* **closed polygonal chain** *is one in which the first vertex coincides with the last one, or, alternatively, the first and the last vertices are also connected by a line segment. A simple closed polygonal chain in the plane is the boundary of a simple polygon. Often the term "polygon" is used in the meaning of "closed polygonal chain".*

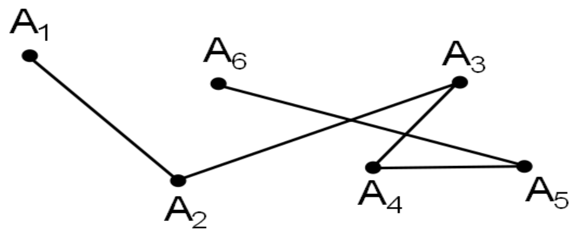


(a)

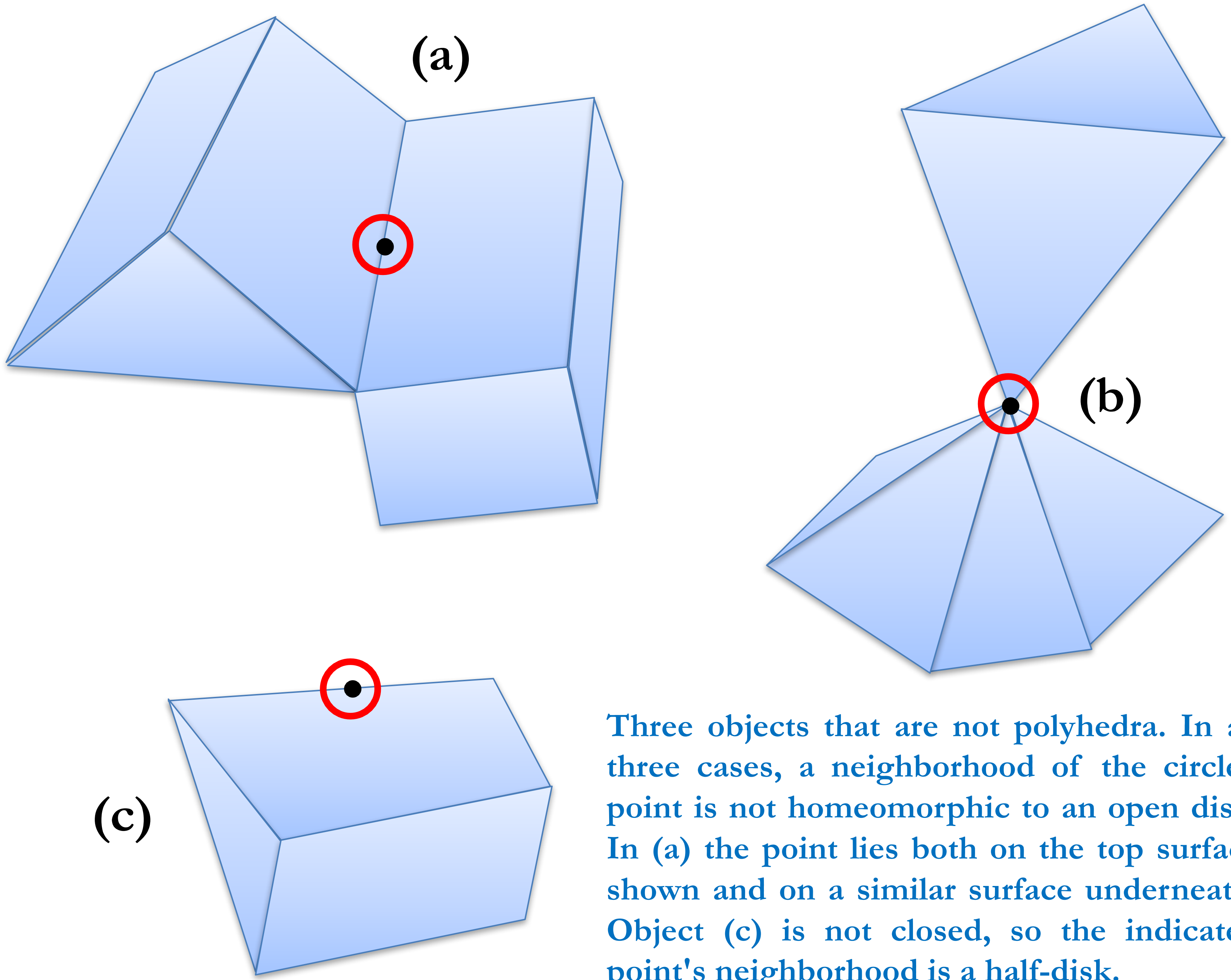
Example of (a) a simple polygonal chain (b) a self-intersecting polygonal chain and (c) a closed polygonal chain



(b)



(c)



Three objects that are not polyhedra. In all three cases, a neighborhood of the circled point is not homeomorphic to an open disk. In (a) the point lies both on the top surface shown and on a similar surface underneath. Object (c) is not closed, so the indicated point's neighborhood is a half-disk.

3. Global topology is "proper."

- We would like the surface to be connected, closed, and bounded. So we require that the surface be connected in the sense that from any point, one may walk to any other point on the surface.
- This can be stated combinatorially by requiring that the *1-skeleton,* the graph of edges and vertices, be connected.
- Such condition with emphasizing having a finite number of faces, our previous conditions already imply closed and bounded surfaces.
- One might be inclined to rule out "holes" in the definition of polyhedron, holes in the sense of "channels" from one side of the surface to the other that do not disconnect the exterior (unlike cavities).
- Should a torus (a shape like a doughnut) be a polyhedron? We adopt the usual terminology and permit polyhedra to have an arbitrary number of such holes. The number of holes is called the *genus* of the surface.
- **Normally we will only consider polyhedra with genus zero: those topologically equivalent to the surface of a sphere.**

In Summary …

- The boundary of a polyhedron is a finite collection of planar, bounded convex polygonal faces such that:
  - the faces intersect properly;
  - the neighborhood of every point is topologically an open disk, or (equivalently) the link of every vertex is a simple polygonal chain; and
  - the surface is connected, or (equivalently) the 1-skeleton is connected.
- The boundary is closed and encloses a bounded region of space. Every edge is shared by exactly two faces; these faces are called *adjacent*.

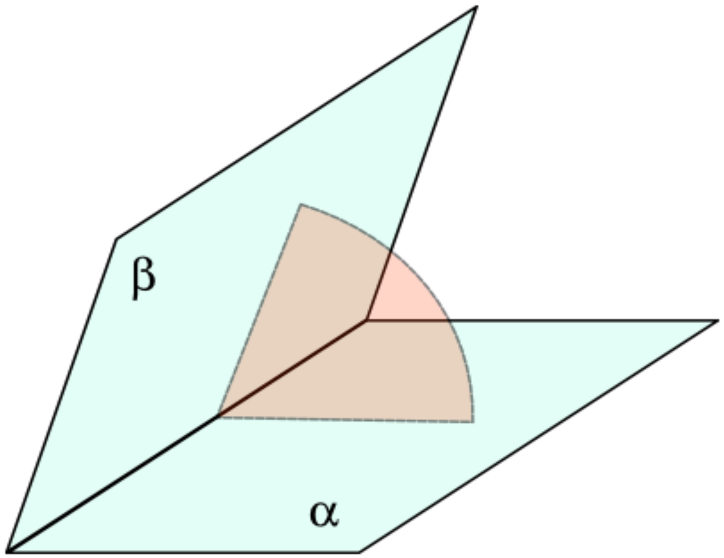
Definition 22: *A* **polytope**, *or sometimes 3-polytopes to emphasize their three-dimensionality, is a polyhedron that is convex in that the segment connecting any two of its points is inside.*

Just as convex polygons can be characterized by the local requirement that each vertex be convex, polytopes can be specified locally by requiring that all *dihedral* angles be convex ($\leq \pi$).

Definition 23: Dihedral angles *are the internal angles in space at an edge between the planes containing its two incident faces.*

For any polytope, the sum of the face angles around each vertex is at most 2π, but this condition does not alone imply convexity.

It is important for building intuition and testing out ideas to become intimately familiar with a few polyhedra. We therefore take time to discuss the five Platonic solids.



The dihedral angle (pale red) is the part of the space between two half-planes (pale blue).

Regular Polytopes

Definition 24: *A* **regular polygon** *is one with equal sides and equal angles: equilateral triangle, square, regular pentagon, regular hexagon, and so on. Hence there are an infinite variety of regular polygons, one for each n (number of vertices).*

Definition 25: Regular polyhedra *are convex polyhedra, they are often called regular polytopes in the sense that all faces are congruent regular polygons, and the number of faces incident to each vertex is the same for all vertices. This implies equal dihedral angles.*

- The surprising implication of these regularity conditions is that there are only five distinct types of regular polytopes. These are known as the *Platonic solids.*
- We now prove that there are exactly five regular polytopes.
- The proof is very elementary. The intuition is that the internal angles of a regular polygon grow large with the number of vertices of the polygon.

Let p be the number of vertices per face; so each face is a regular p-gon.

The sum of the faces angles for one p-gon is $\pi(p - 2)$, so each face angle is $\frac{1}{p-th}$ of this, i.e. $\pi(1 - \frac{2}{p})$.

Let v be the number of faces meeting at a vertex.

The key constraint is that the sum of the face angles meeting at a vertex is less than 2π, in order for the polyhedron to be convex. (we only consider real vertices at which the face angles sum to strictly less than 2π).

This can be seen intuitively by noticing that if the polyhedron surface is flat in the vicinity of a vertex, the sum of the angles is exactly 2π and the sum of angles at a needle-sharp vertex is quite small.

So the angle sum is in the range $(0,2\pi)$ Thus we have v angles, each $\pi(1 - \frac{2}{p})$, which must sum to less than 2π.

We transform this inequality with a series of algebraic manipulations to reach a particularly convenient form:

$$v\pi\left(1-\frac{2}{p}\right)<2\pi$$

$$1-\frac{2}{p}<\frac{2}{v}$$

$$pv<2v+2p$$

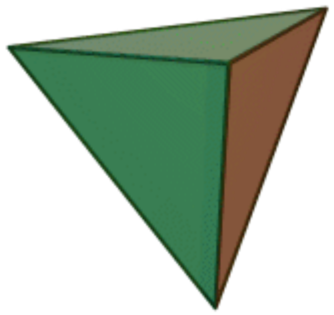
$$pv-2v-2p+4<4$$

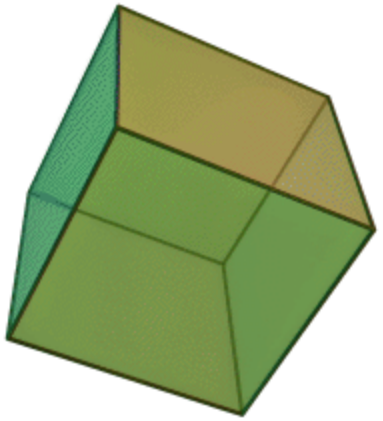
$$(p-2)(v-2)<4$$

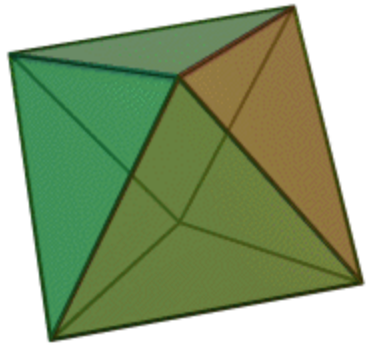
Where both *p* and *v* are integers. Because a polygon must have at least three sides, i.e. $p \geq 3$. It is perhaps less obvious that $v > 3$: i.e. at least three faces must meet at each vertex, since no "solid angle" could be formed at *v* with only two faces. These constraints suffice to limit the possibilities to those listed in Table 1. For example, $p = 4$ and $v = 4$ leads to $(p - 2)(v - 2) = 4$, violating the inequality. And indeed if four squares are pasted at a vertex, they must be coplanar, and this case cannot lead to a polyhedron.

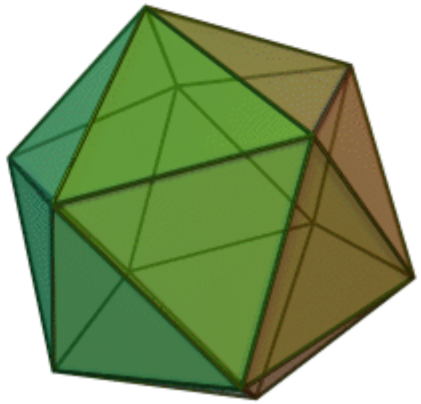
Table 1 – Legal *p/v* values, The Greek prefixes in the names refer to the number of faces: tetra = 4, acta = 8, dodeca = 12, icosa = 20. Sometimes a cube is called a "hexahedron". V is the number of vertices, E is the number of edges and F is the number of faces.

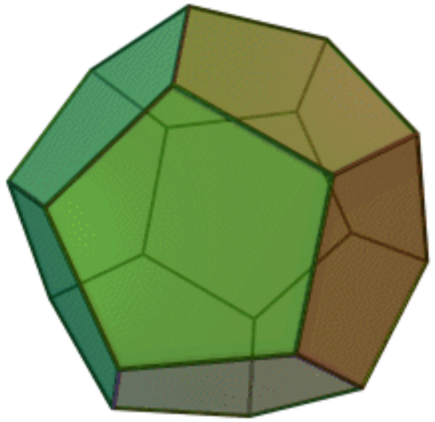
p	*v*	*(p-2)(v-2)*	*Name*	*Description*	*V*	*E*	*F*
3	3	1	Tetrahedron	3 triangles at each vertex	4	6	4
4	3	2	Cube	3 squares at each vertex	8	12	6
3	4	2	Octahedron	4 triangles at each vertex	6	12	8
5	3	3	Dodecahedron	3 pentagons at each vertex	20	30	12
3	5	3	Icosahedron	5 triangles at each vertex	12	30	20


Tetrahedron


Cube


Octahedron

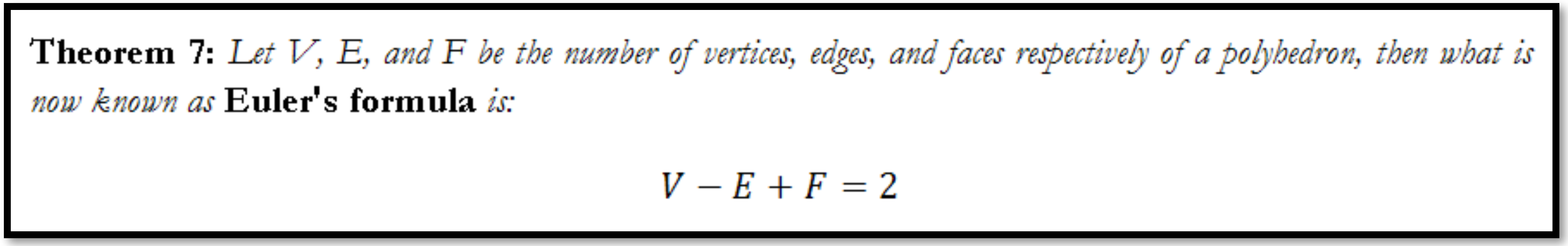

Dodecahedron


Icosahedron

Euler’s Formula

Leonhard Paul Euler (15 April 1707 – 18 September 1783) was a pioneering Swiss mathematician and physicist who spent most of his life in Russia and Germany.
Euler made important discoveries in fields as diverse as calculus and graph theory. He also introduced much of the modern mathematical terminology and notation, particularly for mathematical analysis, such as the notion of a mathematical function.

- In 1758 Leonard Euler noticed a remarkable regularity in the numbers of vertices, edges, and faces of a polyhedron of genus zero:
  - The number of vertices and faces together is always two more than the number of edges; and this is true for *all* polyhedra.
- So a cube has 8 vertices and 6 faces, and 8 + 6 = 14 is two more than its 12 edges. And the remaining regular polytopes can be seen to satisfy the same relationship.



Theorem 7: *Let V, E, and F be the number of vertices, edges, and faces respectively of a polyhedron, then what is now known as* **Euler's formula** *is:*

$$V - E + F = 2$$

Incremental Algorithm in 3D

The overall structure of the three-dimensional incremental algorithm is identical to that of the two-dimensional version:

At the *i*-th iteration, compute $\mathcal{H}_i = chull\{\mathcal{H}_{i-1} \cup p_i\}$. And again the problem of computing the new hull naturally divides into two cases.

Let $p \;=\; p_i$ and $Q = \mathcal{H}_{i-1}$.

Decide if $p \in Q$. If so, discard $p;$ if not, compute the *cone tangent* to Q whose apex is $p,$ and construct the new hull.

The test $p \in Q$ can be made in the same fashion as in two dimensions: p is inside Q if an only if p is to the positive side of every plane determined by a face of Q.

The left-of-triangle test is based on the volume of the determined tetrahedron, just as the left-of-segment test is based on the area of the triangle.

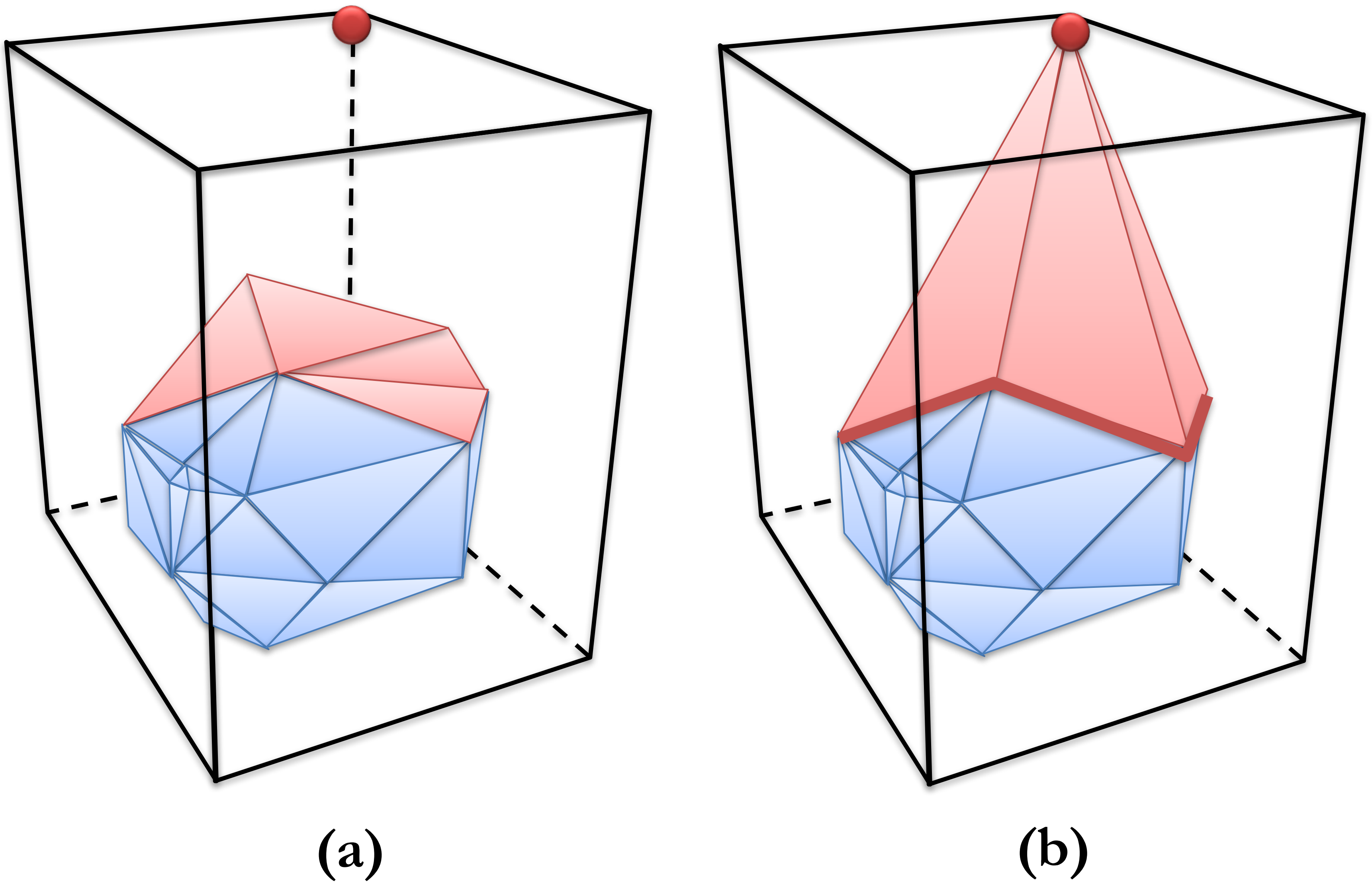
If all faces are oriented consistently, the volumes must all have the same sign (positive under our conventions). This test clearly can be accomplished in time proportional to the number of faces of Q. which as we saw previously, is $O(n)$.

When p is outside Q, the problem becomes more difficult, as the hull will be altered.

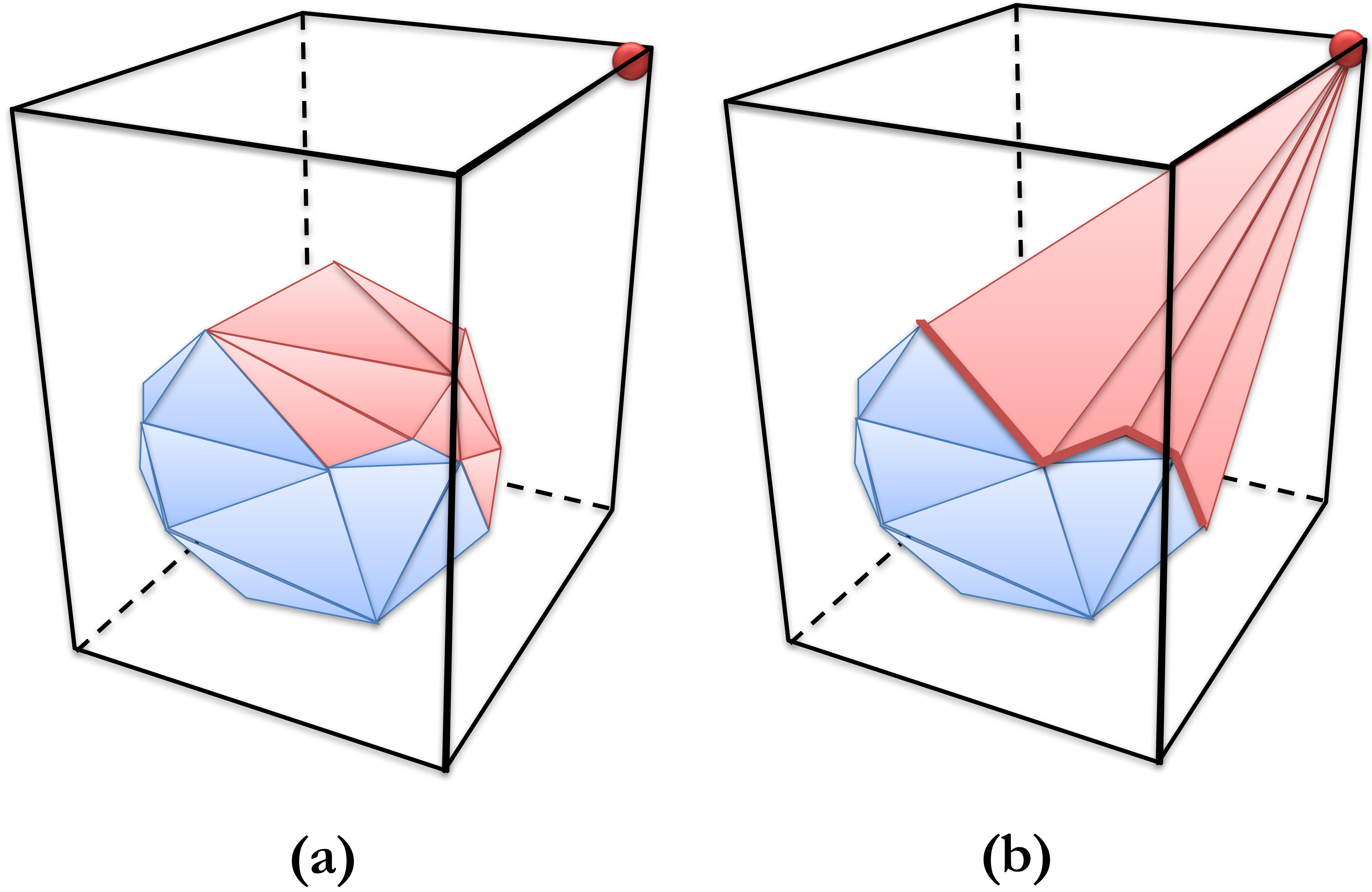
Recall that in the two-dimensional incremental algorithm, the alteration required finding two tangents from p to Q.

In three dimensions, there are tangent planes rather than tangent lines. These planes bound a *cone* of triangle faces, each of whose apex is p, and whose base is an edge e of Q.

An example is shown in the following figures. The first figure shows $\mathcal{H}_{i-1}$ and $\mathcal{H}_i$; from one point of view, and the second one shows the same example from a different viewpoint.



-Viewpoint one: (a) $\mathcal{H}_{i-1}$ before adding point in corner, (b) after: $\mathcal{H}_i$



· Viewpoint two: (a) $\mathcal{H}_{i-1}$ before adding point in corner, (b) after: $\mathcal{H}_i$

We now discuss how these cone faces can be constructed.

Imagine standing at p and looking toward Q.

Assuming for the moment that no faces are viewed edge-on; the interior of each face of Q is either visible or not visible from p.

It should be clear that the visible faces are precisely those that are to be discarded in moving from $Q = \mathcal{H}_{i-1}$ to $\mathcal{H}_i$.

Moreover, the edges on the border of the visible region are precisely those that become the bases of cone faces apexed at p.

Since suppose e is an edge of Q such that the plane determined by e and p is tangent to Q. Edge e is adjacent to two faces, one of which is visible from p, and one of which is not.

Therefore, e is on the border of the visible region.

An equivalent way to view this is to think of a light source placed at p. Then the visible region is that portion of Q illuminated, and the border edges are those between the light and dark regions.

From this discussion, it is evident that if we can determine which faces of Q are visible from p and which are not, then we will know enough to find the border edges and therefore construct the cone, and we will know which faces to discard. We now need a precise definition of visibility.

Definition 26: *A face said to be* **visible** *from* p *if and only if some point* x *interior to* f *is visible from* p, *that is,* px *does not intersect* Q *except at* x: $px \cap Q = \{x\}$

Note that under this definition, seeing only an edge of a face does not render the face visible, and faces seen edge-on are also considered invisible. Whether a triangle face (a, b, c) is visible from p can be determined from the signed volume of the tetrahedron (a, b, c, p). It is visible if and only if the volume is strictly negative.

We can now outline the algorithm based on the visibility calculation. Of course many details remain to be explained, but the basics of the algorithm should be clear.

Algorithm: 3D Incremental Algorithm

```
Initialize $\mathcal{H}_3$ to tetrahedron $(p_0, p_1, p_2, p_3)$.
    For $i = 4, \dots, n-1$ do
      For each face $f$ of $\mathcal{H}_{i-1}$ do
        Compute the volume of tetrahedron determined by $f$ and $p_i$
        Mark $f$ visible if and only if volume $< 0$
      If no faces are visible
        Then Discard $p_i$ (it is inside $\mathcal{H}_{i-1}$)
        Else
          For each border edge $e$ of $\mathcal{H}_{i-1}$ do
            Construct cone face determined by $e$ and $p_i$
         For each visible face $f$ do
           Delete $f$
        Update $\mathcal{H}_i$
```

Thanks