Convex Hulls and Ortho-convex Hulls

Partha Bhowmick

Associate Professor

http://cse.iitkgp.ac.in/~pb/

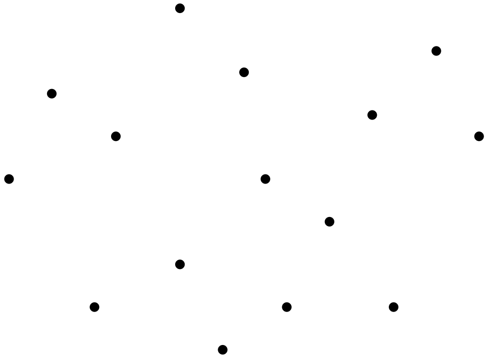
Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

India

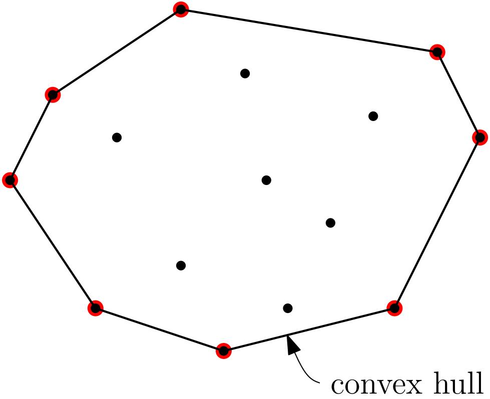
Kalyani Univ. **21-Mar-2014**

Convex hull



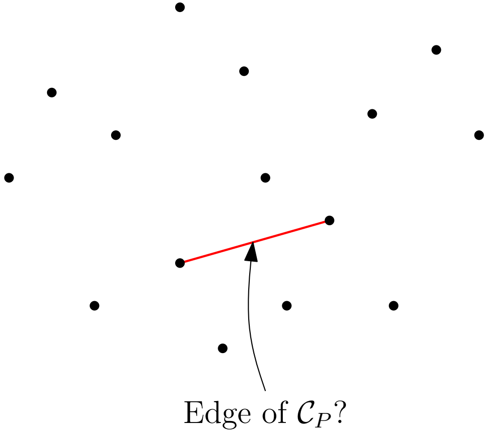
Input: Point set P on xy-plane.

Convex hull

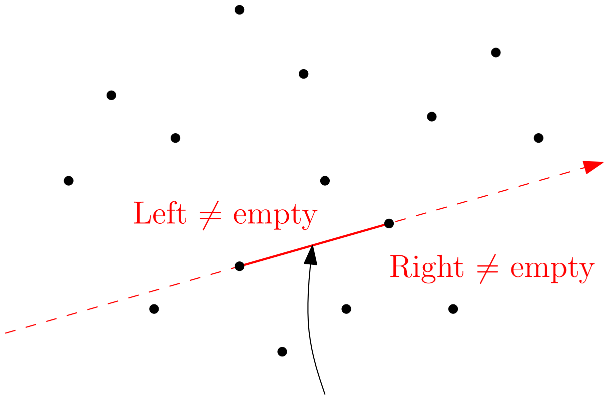


Output: Convex hull, $\mathcal{C}_P$ = a sequence of vertices/edges.

Convex hull

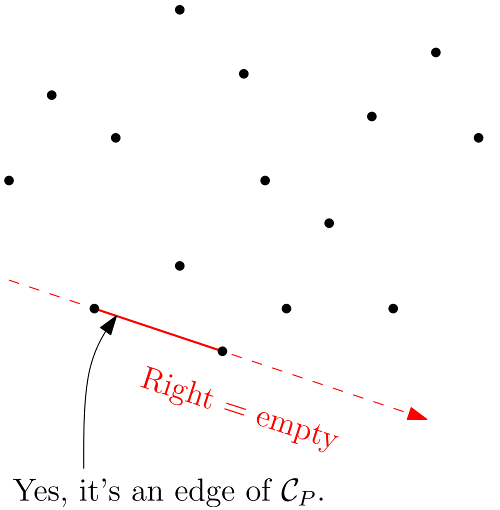


Convex hull

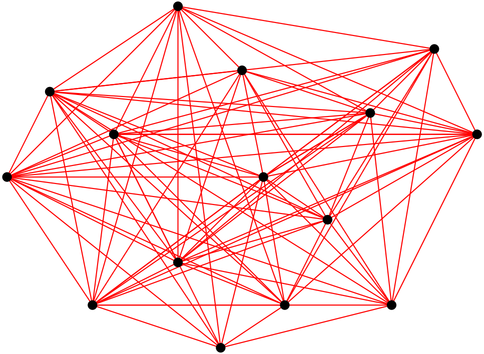


No, it's not an edge of $\mathcal{C}_P$.

Convex hull

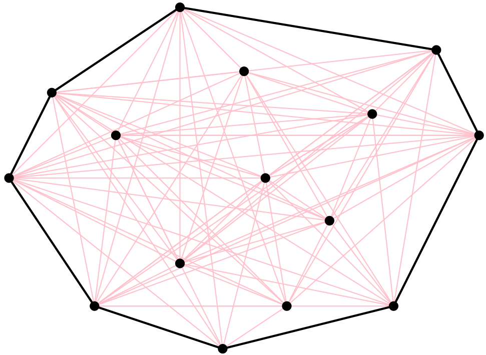


Convex hull



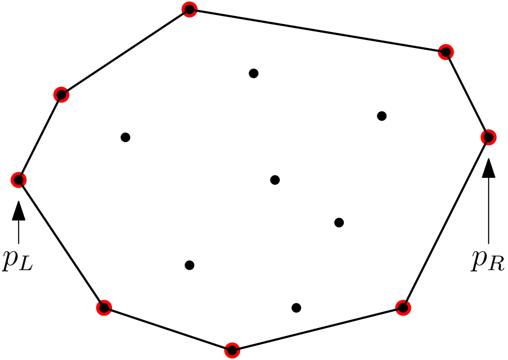
$$|P| = n \Rightarrow O(n^2) \text{ pairs } \Rightarrow O(n^3) \text{ time}$$

Convex hull



$|\mathcal{C}_P| = O(n)$: $O(n^3)$ time is quite high!

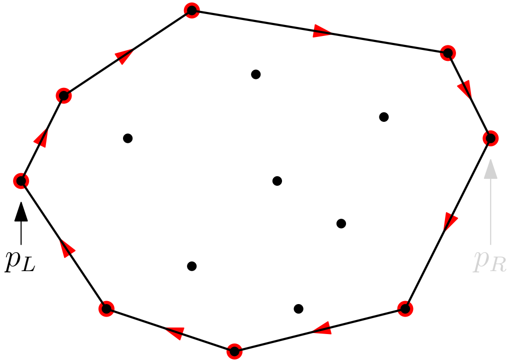
Better observations



Obs 1

The leftmost point p_L and the rightmost point p_R of P form the leftmost and the rightmost vertices of $\mathcal{C}_P$.

Better observations



Obs 2

Clockwise traversal along the boundary of $\mathcal{C}_P$ always yields a right turn at each vertex of $\mathcal{C}_P$.

Better observations

The clue

Use turn type to decide whether a triplet of points forms a pair of consecutive edges of $\mathcal{C}_P$.

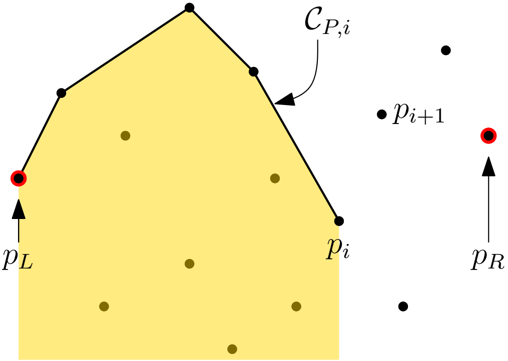
But how?
We have $O(n^3)$ triplets of points!
We can avoid checking so many triplets if we use incremental approach.

Better observations

A question

Let $\mathcal{C}_{P,i}$ = vertices of upper hull up to p_i.

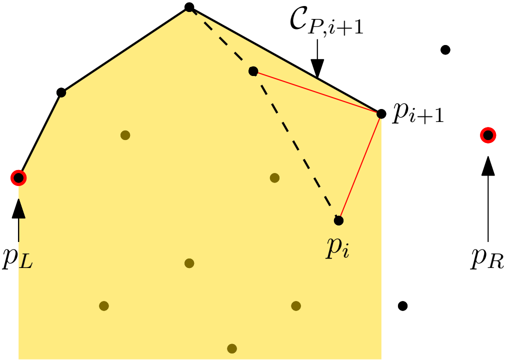
Then what's the relation between $\mathcal{C}_{P,i+1}$ and $\mathcal{C}_{P,i}$?



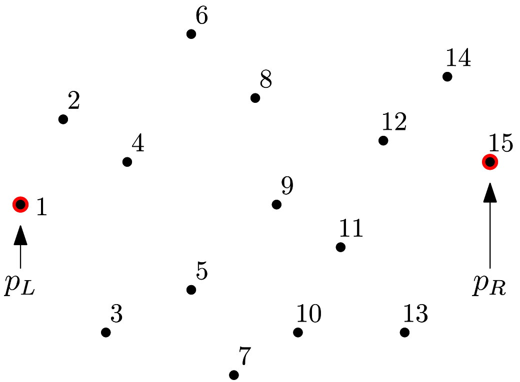
The answer

$\mathcal{C}_{P,i+1} \subseteq \mathcal{C}_{P,i} \cup \{p_{i+1}\}$.

It's a strong observation $\Rightarrow$ *Incremental algorithm!*

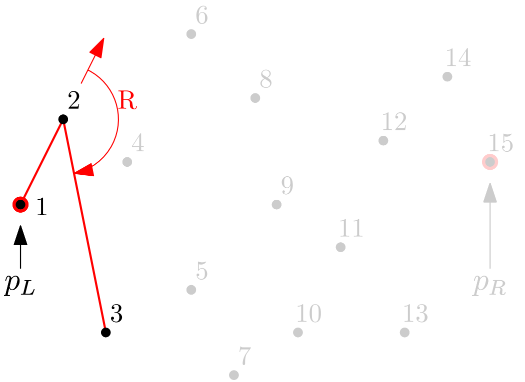


Incremental algorithm: *Graham scan*

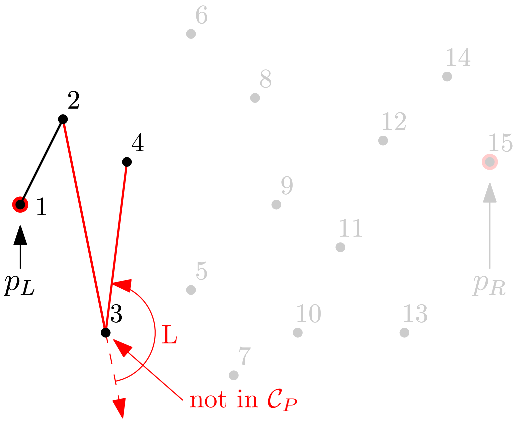


After lexicographic sorting
(x = primary key, y = secondary key)

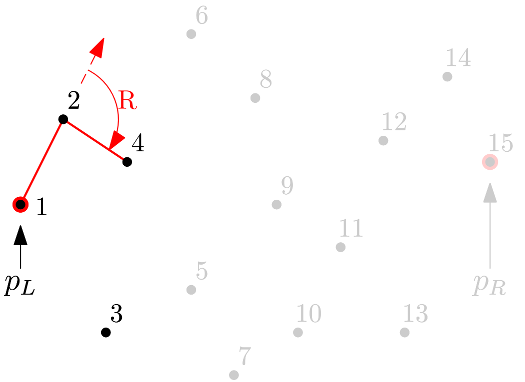
Incremental algorithm: *Graham scan*



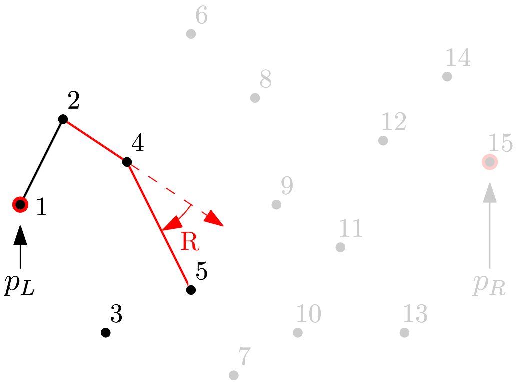
Incremental algorithm: *Graham scan*



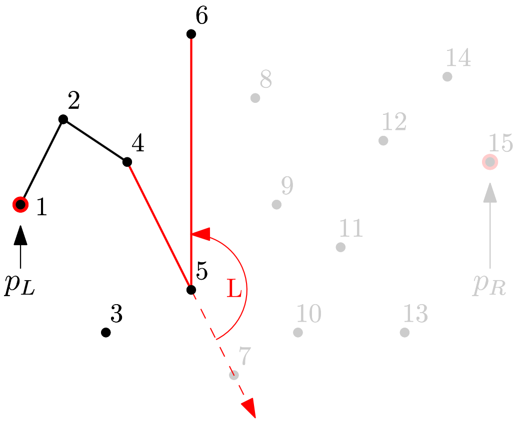
Incremental algorithm: *Graham scan*



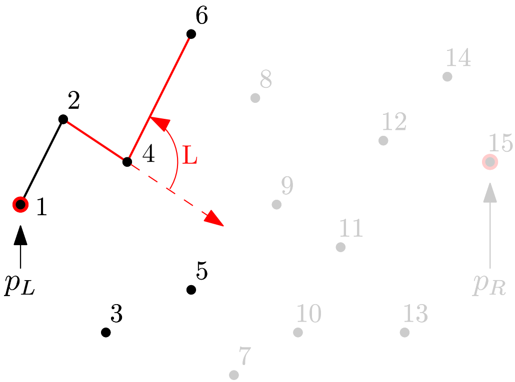
Incremental algorithm: *Graham scan*



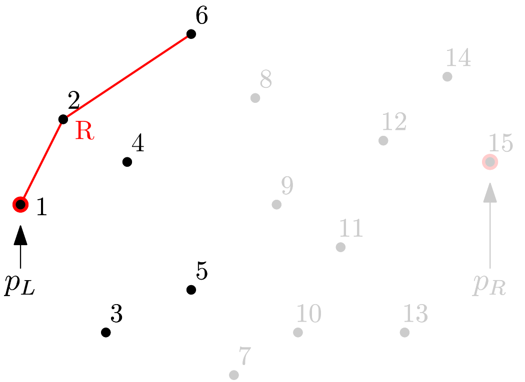
Incremental algorithm: *Graham scan*



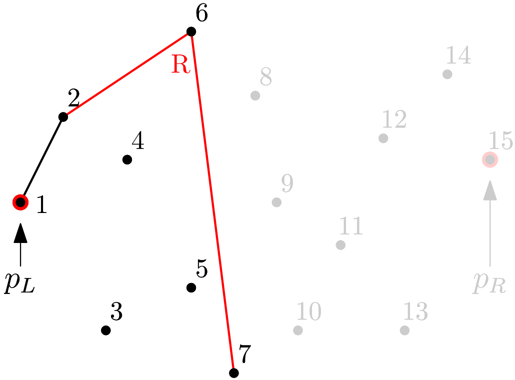
Incremental algorithm: *Graham scan*



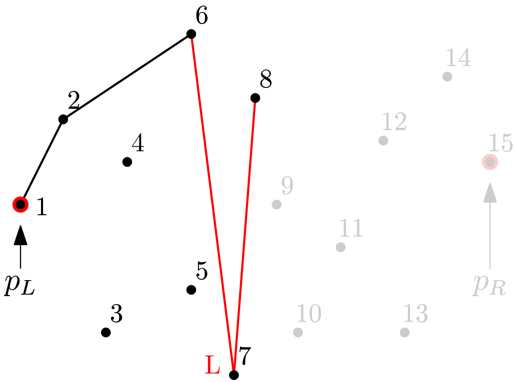
Incremental algorithm: *Graham scan*



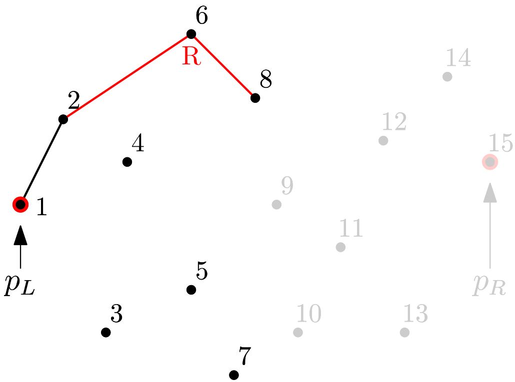
Incremental algorithm: *Graham scan*



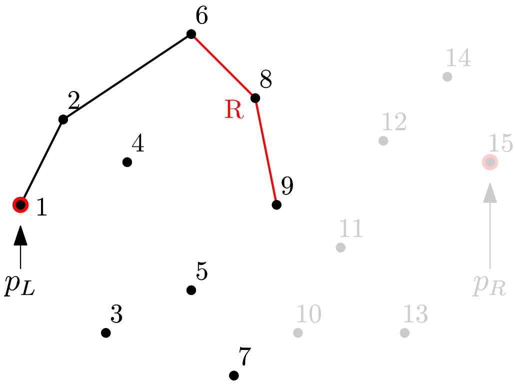
Incremental algorithm: *Graham scan*



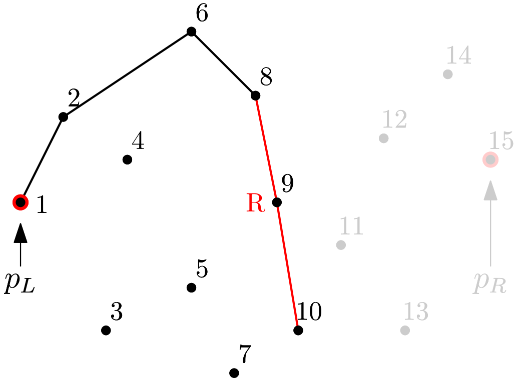
Incremental algorithm: *Graham scan*



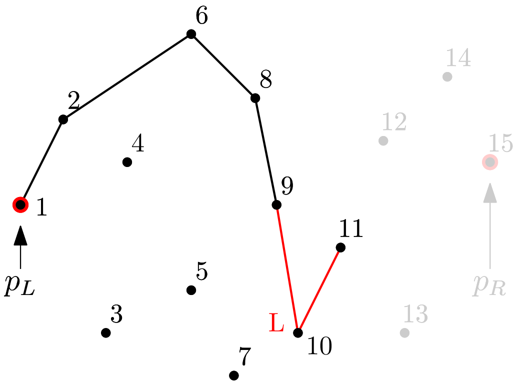
Incremental algorithm: *Graham scan*



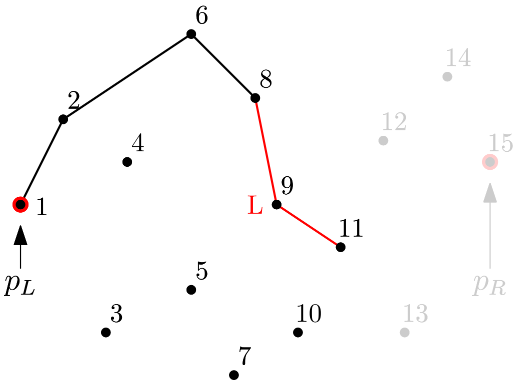
Incremental algorithm: *Graham scan*



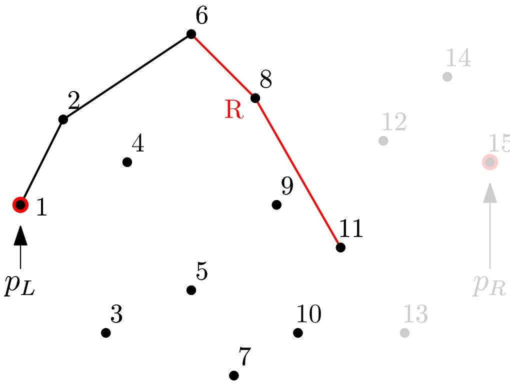
Incremental algorithm: *Graham scan*



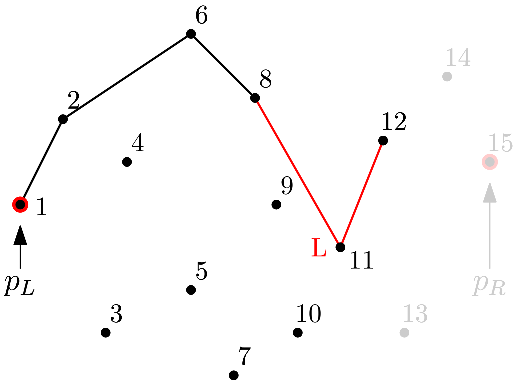
Incremental algorithm: *Graham scan*



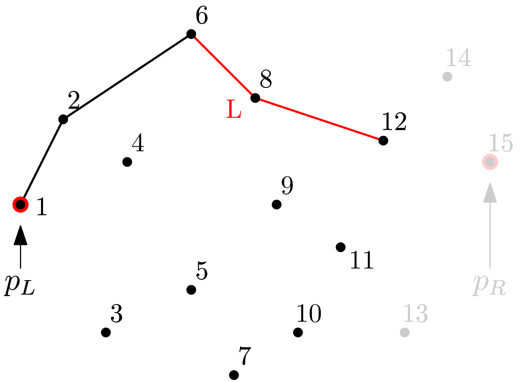
Incremental algorithm: *Graham scan*



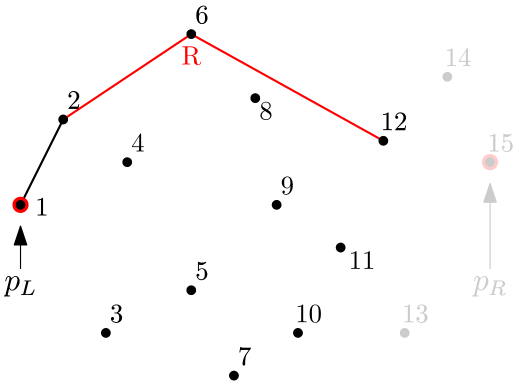
Incremental algorithm: *Graham scan*



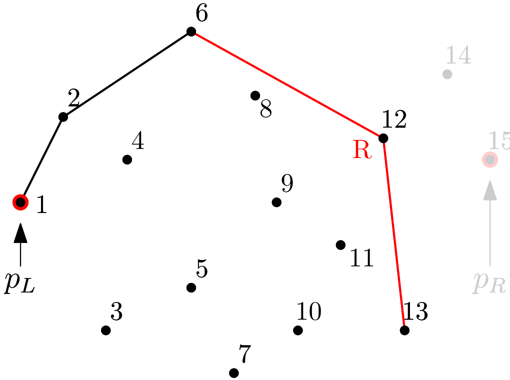
Incremental algorithm: *Graham scan*



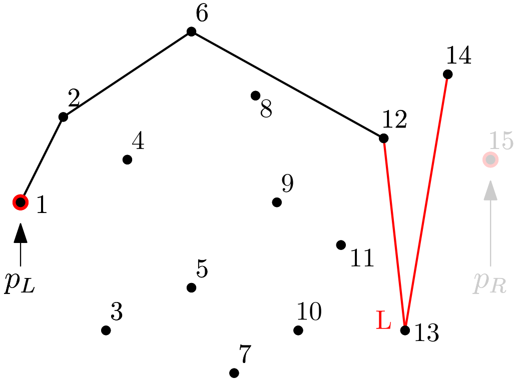
Incremental algorithm: *Graham scan*



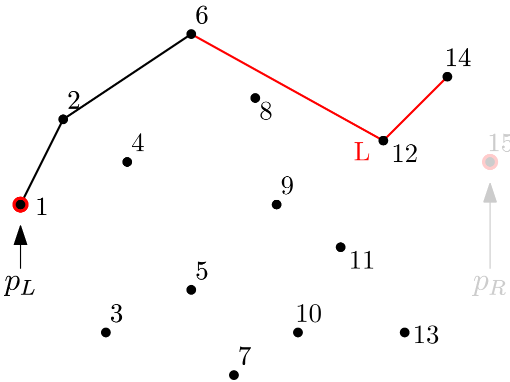
Incremental algorithm: *Graham scan*



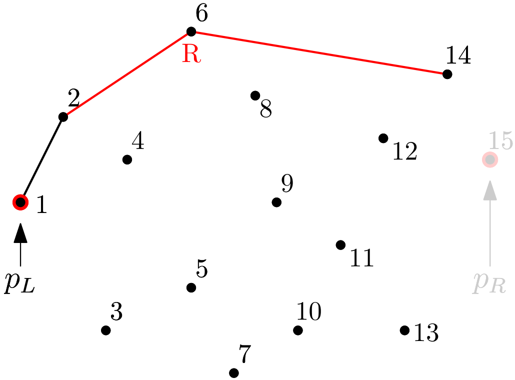
Incremental algorithm: *Graham scan*



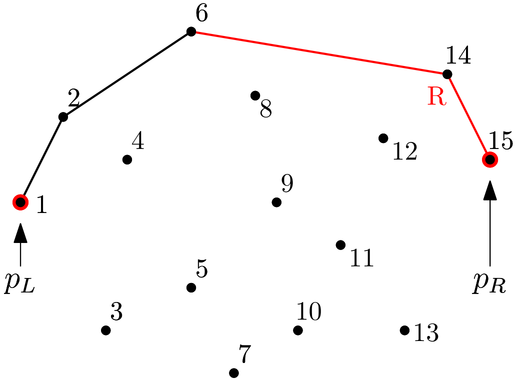
Incremental algorithm: *Graham scan*



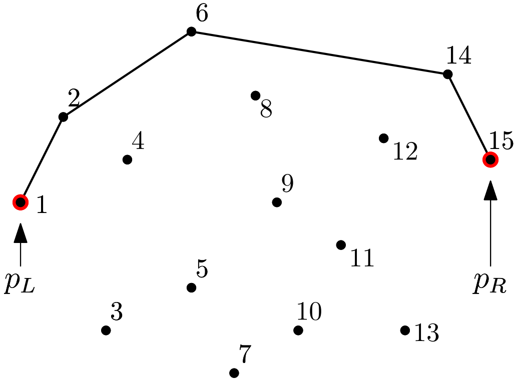
Incremental algorithm: *Graham scan*



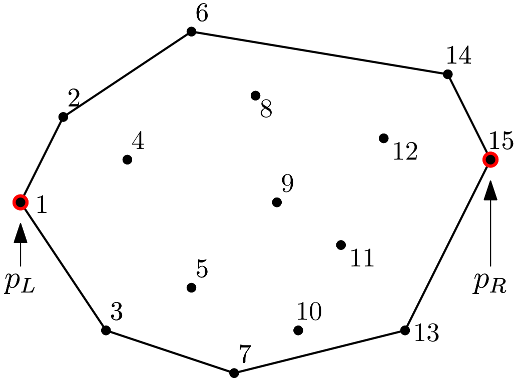
Incremental algorithm: *Graham scan*



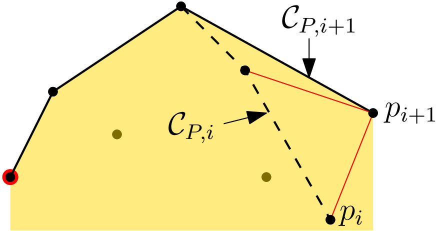
Incremental algorithm: *Graham scan*



Incremental algorithm: *Graham scan*



Time complexity

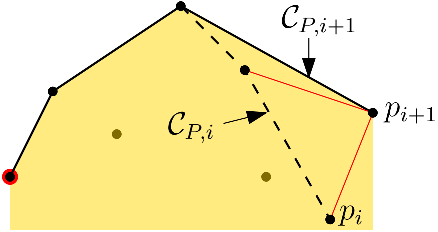


Let $p_j \in \mathcal{C}_{P,i}$.

If $p_j \notin \mathcal{C}_{P,i+1}$, then $p_j \notin \mathcal{C}_{P,i+2}, p_j \notin \mathcal{C}_{P,i+3}, \ldots, p_j \notin \mathcal{C}_{P,n}$, since $\mathcal{C}_{P,i+1} \subseteq \mathcal{C}_{P,i} \cup \{p_{i+1}\}$.

So, once p_j is removed from the upper hull, it's never reconsidered.

Time complexity

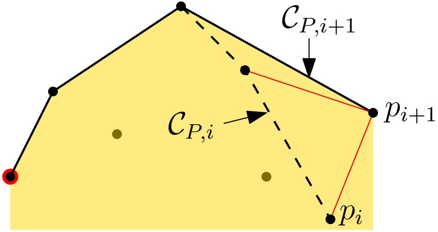


Let $p_j \in \mathcal{C}_{P,i}$.

If $p_j \notin \mathcal{C}_{P,i+1}$, then $p_j \notin \mathcal{C}_{P,i+2}, p_j \notin \mathcal{C}_{P,i+3}, \ldots, p_j \notin \mathcal{C}_{P,n}$, since $\mathcal{C}_{P,i+1} \subseteq \mathcal{C}_{P,i} \cup \{p_{i+1}\}$.

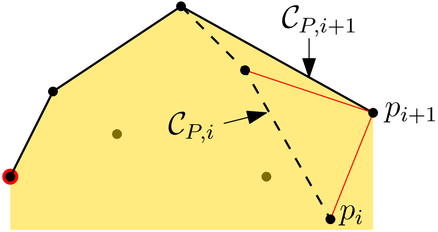
So, once p_j is removed from the upper hull, it's never reconsidered.

Time complexity



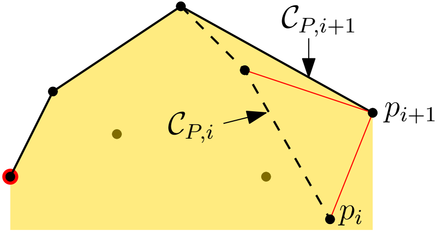
Let $p_j \in \mathcal{C}_{P,i}$.
If $p_j \notin \mathcal{C}_{P,i+1}$, then $p_j \notin \mathcal{C}_{P,i+2}, p_j \notin \mathcal{C}_{P,i+3}, \ldots, p_j \notin \mathcal{C}_{P,n}$, since $\mathcal{C}_{P,i+1} \subseteq \mathcal{C}_{P,i} \cup \{p_{i+1}\}$.
So, once p_j is removed from the upper hull, it's never reconsidered.

Time complexity



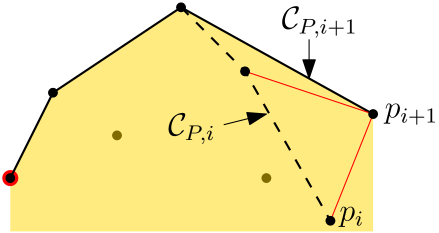
Let $p_j \in \mathcal{C}_{P,i}$.
If $p_j \notin \mathcal{C}_{P,i+1}$, then $p_j \notin \mathcal{C}_{P,i+2}, p_j \notin \mathcal{C}_{P,i+3}, \ldots, p_j \notin \mathcal{C}_{P,n}$, since $\mathcal{C}_{P,i+1} \subseteq \mathcal{C}_{P,i} \cup \{p_{i+1}\}$.
So, once p_j is removed from the upper hull, it's never reconsidered.

Time complexity



Let $p_j \in \mathcal{C}_{P,i}$.
If $p_j \notin \mathcal{C}_{P,i+1}$, then $p_j \notin \mathcal{C}_{P,i+2}, p_j \notin \mathcal{C}_{P,i+3}, \ldots, p_j \notin \mathcal{C}_{P,n}$, since $\mathcal{C}_{P,i+1} \subseteq \mathcal{C}_{P,i} \cup \{p_{i+1}\}$.
So, once p_j is removed from the upper hull, it's never reconsidered.

Time complexity



Data structure: Stack, whose $top = p_i$.

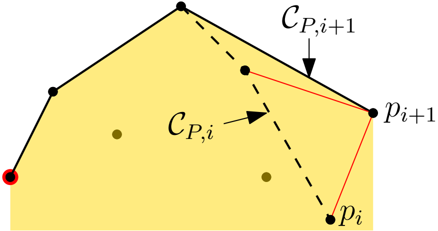
If top two vertices in stack and p_{i+1} do not form a right turn at p_i, then p_i is popped out for ever!

$\Rightarrow$ #pushes $= n$ and #pops $< n$

$\Rightarrow T(n) = O(n) \leftarrow$ no best, average, or worst case!

For lexicographic sorting, it takes $O(n \log n)$ time.

Time complexity



Data structure: Stack, whose *top* $= p_i$.

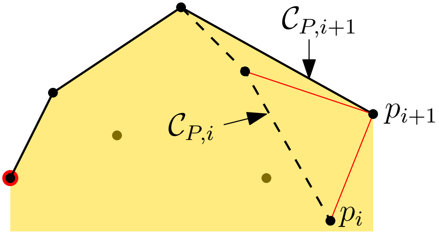
If top two vertices in stack and p_{i+1} do not form a right turn at p_i, then p_i is popped out for ever!

$\Rightarrow$ #pushes $= n$ and #pops $< n$

$\Rightarrow T(n) = O(n) \leftarrow$ no best, average, or worst case!

For lexicographic sorting, it takes $O(n \log n)$ time.

Time complexity



Data structure: Stack, whose *top* $= p_i$.

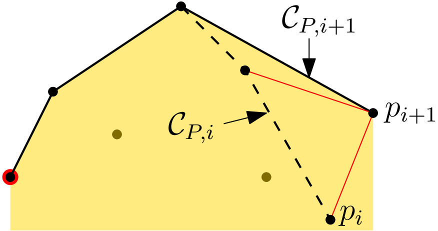
If top two vertices in stack and p_{i+1} do not form a right turn at p_i, then p_i is popped out for ever!

$\Rightarrow$ #pushes $= n$ and #pops $< n$

$\Rightarrow T(n) = O(n) \leftarrow$ no best, average, or worst case!

For lexicographic sorting, it takes $O(n \log n)$ time.

Time complexity



Data structure: Stack, whose $top = p_i$.

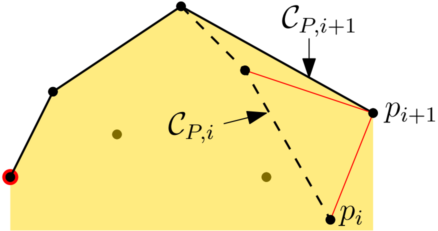
If top two vertices in stack and p_{i+1} do not form a right turn at p_i, then p_i is popped out for ever!

$\Rightarrow$ #pushes $= n$ and #pops $< n$

$\Rightarrow T(n) = O(n) \leftarrow$ no best, average, or worst case!

For lexicographic sorting, it takes $O(n \log n)$ time.

Time complexity



Data structure: Stack, whose *top* $= p_i$.

If top two vertices in stack and p_{i+1} do not form a right turn at p_i, then p_i is popped out for ever!

$\Rightarrow$ #pushes $= n$ and #pops $< n$

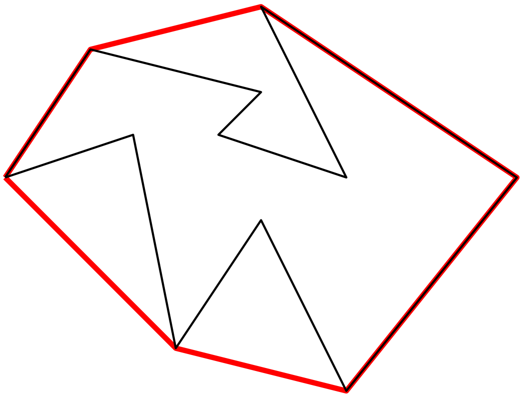
$\Rightarrow T(n) = O(n) \leftarrow$ no best, average, or worst case!

For lexicographic sorting, it takes $O(n \log n)$ time.

Reference of Algorithms

1. **Incremental** — $O(n \log n)$ ▷ $n = \#$points
R. Graham, An Efficient Algorithm for Determining the Convex Hull of a Finite Point Set, *Info. Proc. Letters*, **1**, pp. 132–133, 1972.
2. **Gift wrapping** — $O(nh)$ ▷ $h = \#$hull vertices
R. A. Jarvis, On the Identification of the Convex Hull of a Finite Set of Points in the Plane, *Info. Proc. Letters*, **2**, pp. 18–21, 1973.
3. **Divide and Conquer** — $O(n \log n)$
F. P. Preparata and S. J. Hong, Convex Hulls of Finite Sets of Points in Two and Three Dimensions, *Commun. ACM*, **20**, pp. 87–93, 1977.
4. **Marriage before Conquest** — $O(n \log h)$
D. G. Kirkpatrick and R. Seidel, The Ultimate Planar Convex Hull Algorithm?, *SIAM J. Comput.*, **15**, pp. 287–299, 1986.
5. **Simpler optimal output-sensitive** — $O(n \log h)$
T. M. Chan, Optimal Output-Sensitive Convex Hull Algorithms in Two and Three Dimensions, *Discrete & Computational Geometry*, **16**, pp. 361–368, 1996.

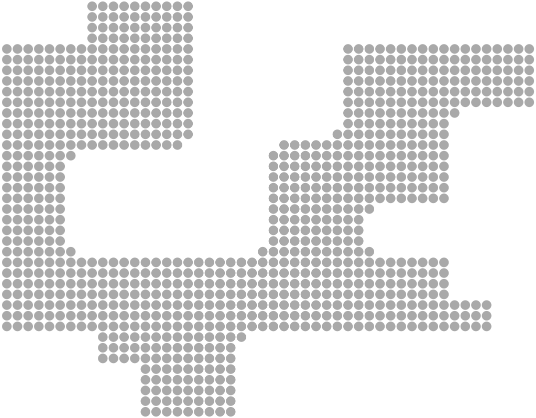
Convex hull of a polygon



Linear-time algorithms

1. 1979 McCallum-Avis, IPL
2. 1983 Lee, Intl. J. Computers & Info. Sc.
3. 1983 Graham-Yao, J. Algorithms
4. 1983 ElGindy-Avis-Toussaint, Computing
5. 1984 Bhattacharya-ElGindy, IEEE Trans. Info. Thy.
6. 1985 Preparata-Shamos, Computational Geometry, Ch. 4
7. 1985 Orlowski, Pattern Rec.
8. 1986 Shin-Woo, Pattern Rec.
9. 1987 Melkman, IPL

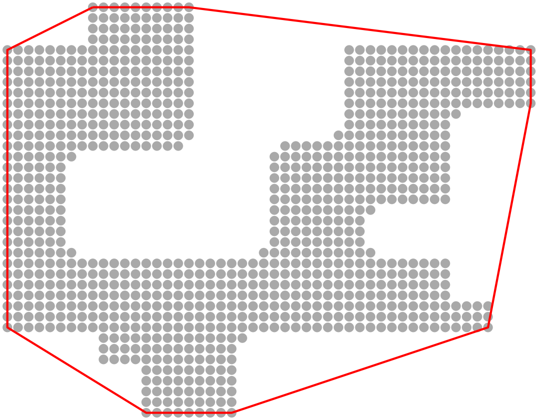
Convex Hull versus Orthogonal Hull



Digital object

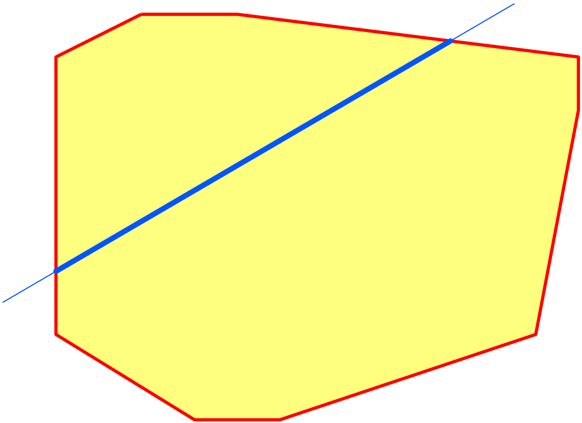
(A = set/connected component of *integer points*)

Convex Hull versus Orthogonal Hull



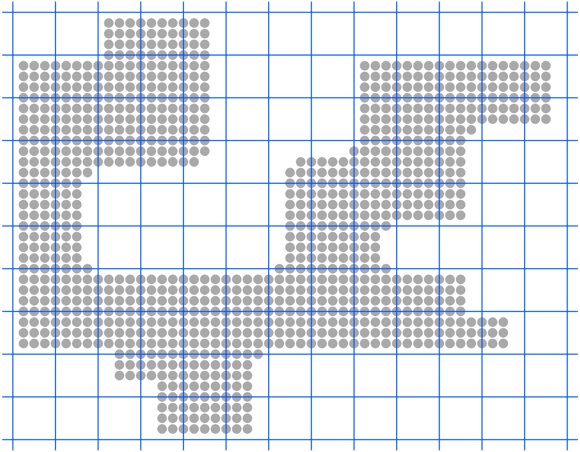
Convex hull $\mathcal{C}_A$

Convex Hull versus Orthogonal Hull



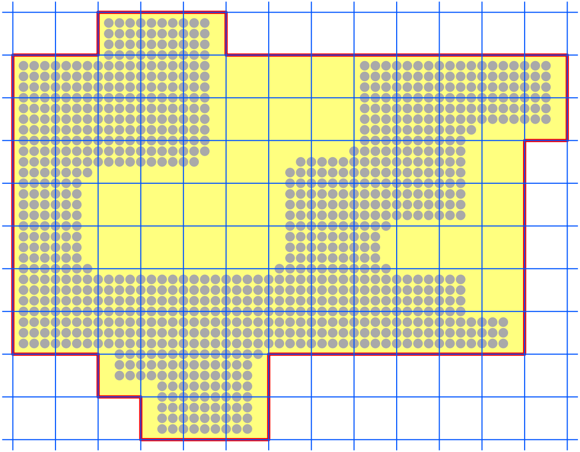
Any straight line has at most one segment of intersection
(a necessary property)

Convex Hull versus Orthogonal Hull



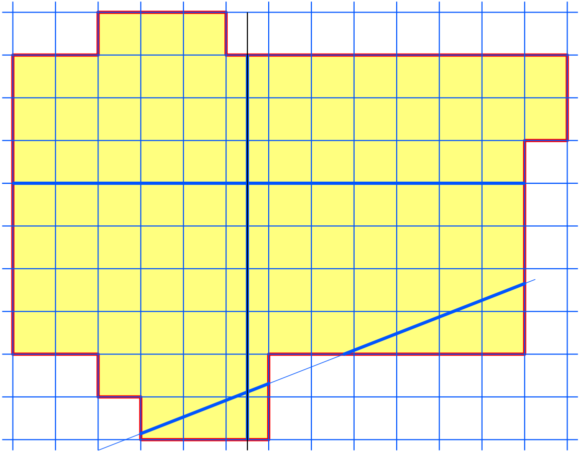
Object A imposed on a grid G of size $g = 4$

Convex Hull versus Orthogonal Hull



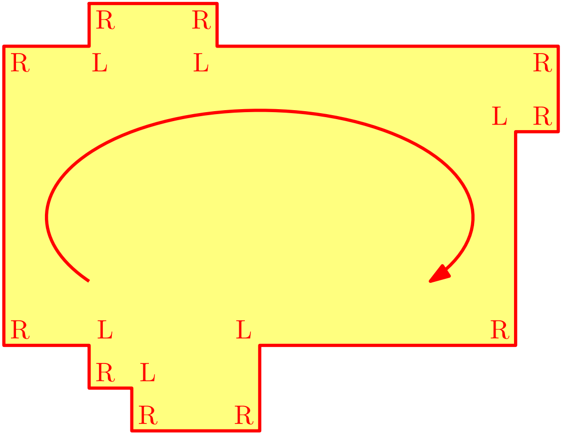
Orthogonal hull $\mathbb{C}_A$

Convex Hull versus Orthogonal Hull



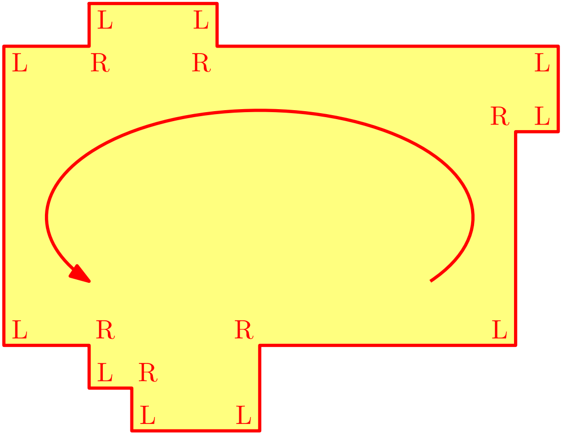
Any horizontal or vertical line has at most one segment of intersection (a necessary property)

Observations



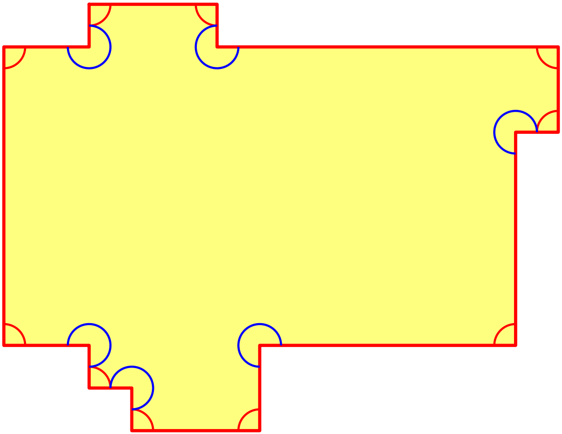
There are both left and right turns!
(clockwise)

Observations



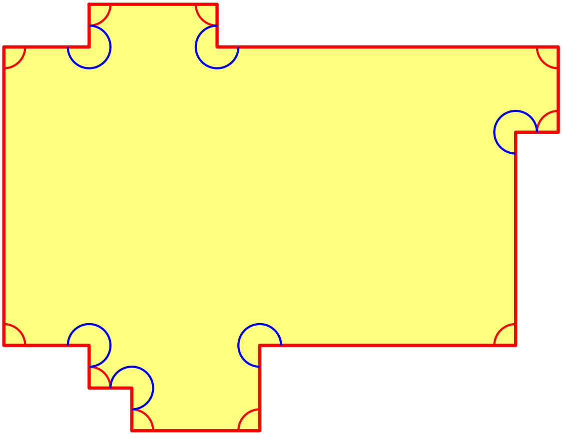
or, both right and left turns (anticlockwise)

Observations



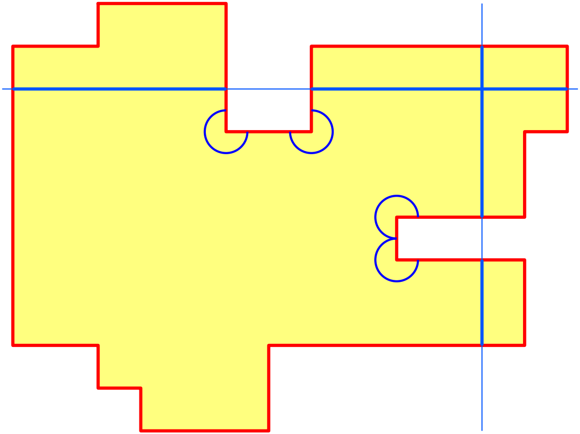
or, 90^0 (Type 1) and 270^0 (Type 3) vertices

Observations



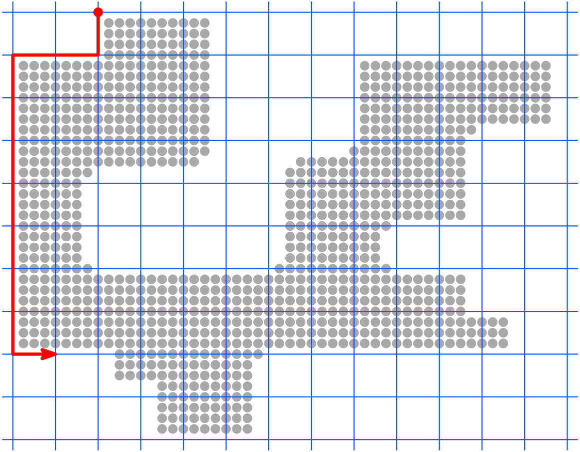
But no two consecutive Type 3 vertices

Observations



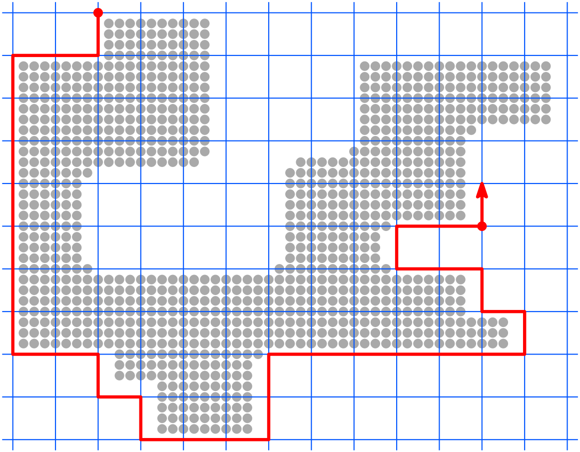
Two consecutive Type 3 vertices defy the necessary property of line intersection

Algorithm



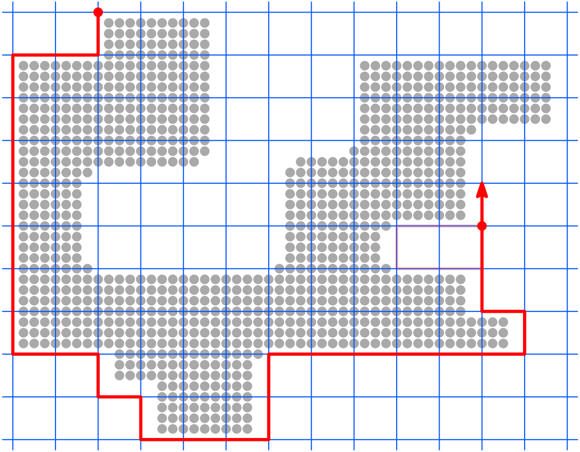
Step 1: Traverse the border of isothetic cover of A

Algorithm



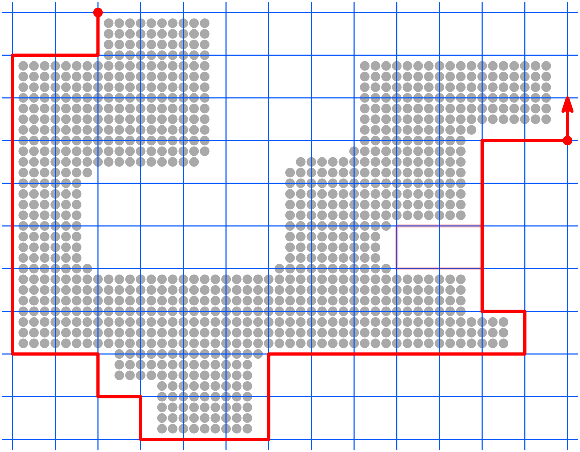
Step 2: If 33, then process to remove the concavity.

Algorithm



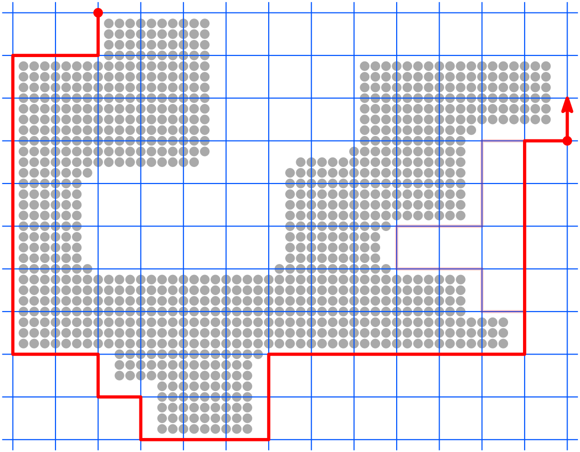
Step 2: If 33, then process to remove the concavity.

Algorithm



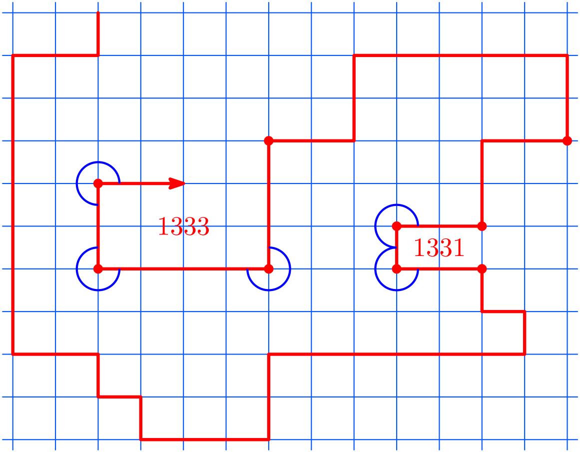
Step 2: If 33, then process to remove the concavity.

Algorithm



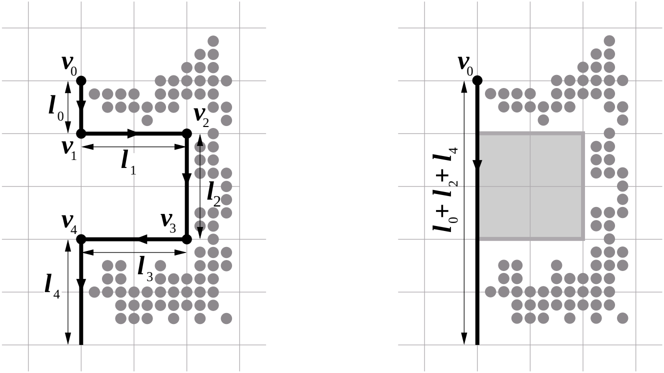
Step 2: If 33, then process to remove the concavity.

Combinatorial cases



Just 1331 and 1333 ($= 1333^+$)

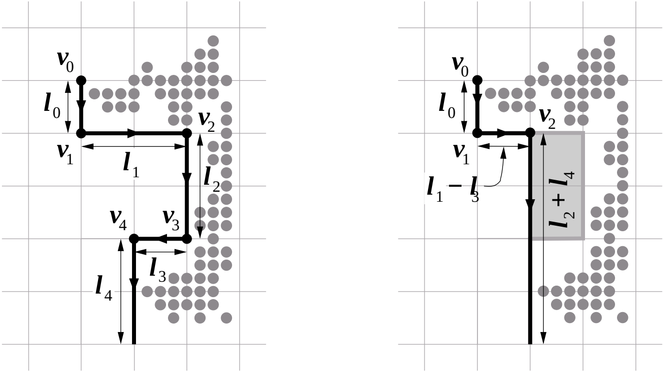
Pattern **1331**



Rule R11 ($l_1 = l_3$):

$$\langle v_0(\mathbf{t_0}, l_0), v_1(\mathbf{1}, l_1), v_2(\mathbf{3}, l_2), v_3(\mathbf{3}, l_3), v_4(\mathbf{1}, l_4)\rangle \rightarrow \langle v_0(\mathbf{t_0}, l_0 + l_2 + l_4)\rangle$$

Pattern **1331**

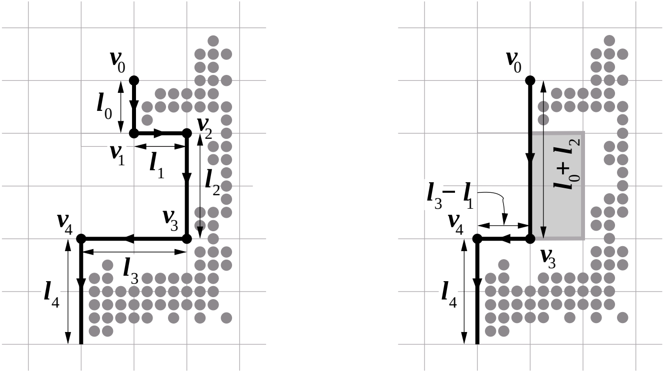


Rule R12 $(l_1 > l_3)$:

$\langle v_0(\mathbf{t_0}, l_0), v_1(\mathbf{1}, l_1), v_2(\mathbf{3}, l_2), v_3(\mathbf{3}, l_3), v_4(\mathbf{1}, l_4)\rangle \rightarrow$

$\langle v_0(\mathbf{t_0}, l_0), v_1(\mathbf{1}, l_1 - l_3), v_2(\mathbf{3}, l_2 + l_4)\rangle$

Pattern **1331**

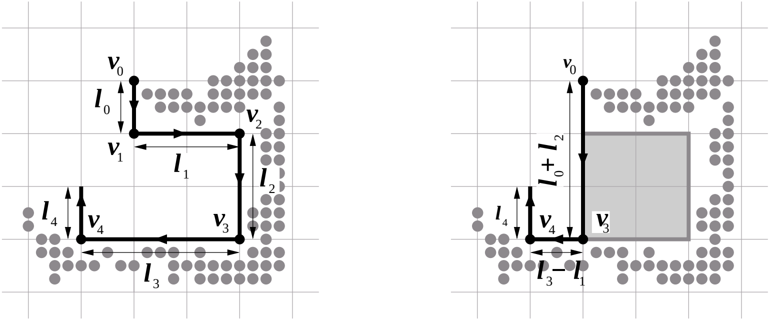


Rule R13 $(l_1 < l_3)$:

$\langle v_0(\mathbf{t_0}, l_0), v_1(\mathbf{1}, l_1), v_2(\mathbf{3}, l_2), v_3(\mathbf{3}, l_3), v_4(\mathbf{1}, l_4)\rangle \rightarrow$

$\langle v_0(\mathbf{t_0}, l_0 + l_2), v_3(\mathbf{3}, l_3 - l_1), v_4(\mathbf{1}, l_4)\rangle$

Pattern **1333** (1)

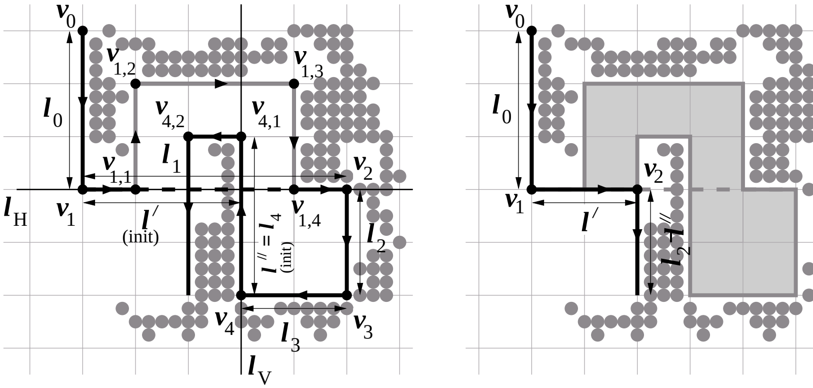


Rule R21 ($l_1 < l_3$):

$\langle v_0(\mathbf{t_0}, l_0),\ v_1(\mathbf{1}, l_1),\ v_2(\mathbf{3}, l_2),\ v_3(\mathbf{3}, l_3),\ v_4(\mathbf{3}, l_4)\rangle \rightarrow$

$\langle v_0(\mathbf{t_0}, l_0 + l_2), v_3(\mathbf{3}, l_3 - l_1), v_4(\mathbf{3}, l_4)\rangle$

Pattern **1333**



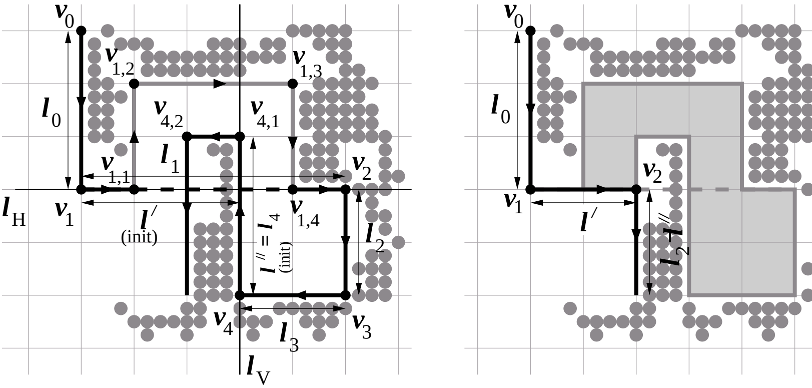
Let v = current vertex (under traversal).

l_H = horizontal line thru' v_2, l_V = vertical line thru' v_4.

$l_H^- \cap l_V^-$ = region lying below l_H and left of l_V.

if $v \in l_H^- \cap l_V^-$, **then** apply **R22**; **else** traverse ahead to get v.

Pattern **1333** (3)



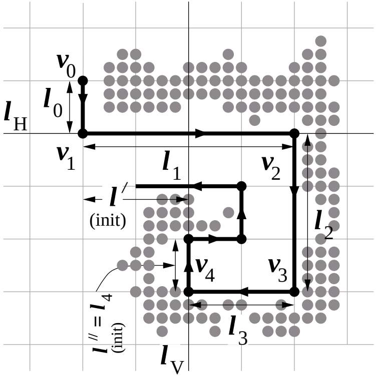
Rule R22 ($l_1 \geqslant l_3$ and $d = d_2$):

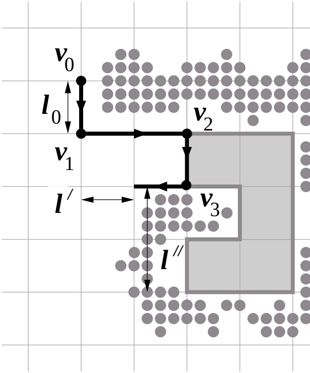
$\langle v_0(\mathbf{t_0}, l_0),\ v_1(\mathbf{1}, l_1),\ v_2(\mathbf{3}, l_2),\ v_3(\mathbf{3}, l_3),\ v_4(\mathbf{3}, l_4)\rangle \rightarrow$

$\langle v_0(\mathbf{t_0}, l_0),\ v_1(\mathbf{1}, l'),\ v_2(\mathbf{3}, l_2 - l'')\rangle$

$d =$ direction from v, $d_2 =$ direction from v_2.

Pattern **1333** (4)





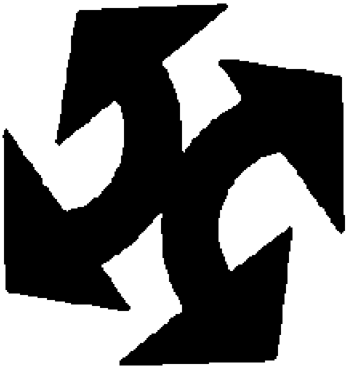
if $v \in l_H^- \cap l_V^-$, **then** apply **R23**; **else** traverse ahead to get v.

Rule R23 ($l_1 \geqslant l_3$ and $d = d_3$):

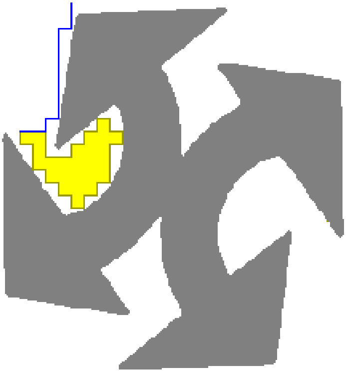
$\langle v_0(\mathbf{t_0}, l_0),\ v_1(\mathbf{1}, l_1),\ v_2(\mathbf{3}, l_2),\ v_3(\mathbf{3}, l_3),\ v_4(\mathbf{3}, l_4)\rangle \rightarrow$

$\langle v_0(\mathbf{t_0}, l_0), v_1(\mathbf{1}, l_1 - l_3), v_2(\mathbf{3}, (l_2 - l'')), v_3(\mathbf{3}, (l_1 - l_3 - l'))\rangle$

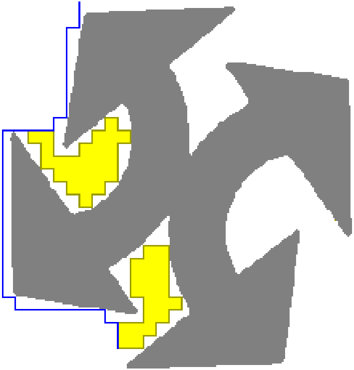
Demo



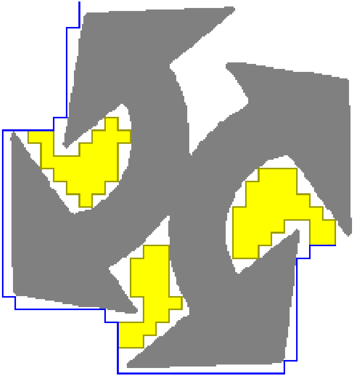
Demo



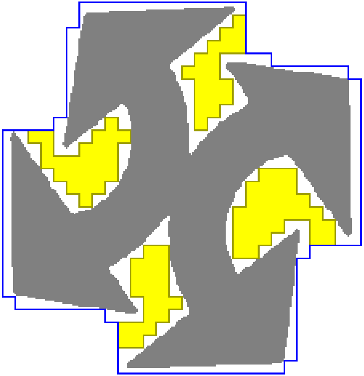
Demo



Demo



Demo



Time Complexity

Let n = #points on object border, g = grid size.

1. Checking object containment in a cell: $O(g)$ time.
2. #grid points visited: $O(n/g)$
   $\Rightarrow$ Visiting all vertices: $O(n/g) \cdot O(g) = O(n)$ time.
3. Removal of a concavity (applying Rule): $O(1)$ time.
4. Maximum #reductions: $O(n/g) - 4$.
   $\Rightarrow$ Total #operations: $(O(n/g) - 4) \cdot O(1) = O(n/g)$.
5. Total time complexity: $O(n) + O(n/g) = O(n)$.

Time Complexity

Let n = #points on object border, g = grid size.

1. Checking object containment in a cell: $O(g)$ time.
2. #grid points visited: $O(n/g)$
   $\Rightarrow$ Visiting all vertices: $O(n/g) \cdot O(g) = O(n)$ time.
3. Removal of a concavity (applying Rule): $O(1)$ time.
4. Maximum #reductions: $O(n/g) - 4$.
   $\Rightarrow$ Total #operations: $(O(n/g) - 4) \cdot O(1) = O(n/g)$.
5. Total time complexity: $O(n) + O(n/g) = O(n)$.

Time Complexity

Let n = #points on object border, g = grid size.

1. Checking object containment in a cell: $O(g)$ time.
2. #grid points visited: $O(n/g)$
   $\Rightarrow$ Visiting all vertices: $O(n/g) \cdot O(g) = O(n)$ time.
3. Removal of a concavity (applying Rule): $O(1)$ time.
4. Maximum #reductions: $O(n/g) - 4$.
   $\Rightarrow$ Total #operations: $(O(n/g) - 4) \cdot O(1) = O(n/g)$.
5. Total time complexity: $O(n) + O(n/g) = O(n)$.

Time Complexity

Let n = #points on object border, g = grid size.

1. Checking object containment in a cell: $O(g)$ time.
2. #grid points visited: $O(n/g)$
   $\Rightarrow$ Visiting all vertices: $O(n/g) \cdot O(g) = O(n)$ time.
3. Removal of a concavity (applying Rule): $O(1)$ time.
4. Maximum #reductions: $O(n/g) - 4$.
   $\Rightarrow$ Total #operations: $(O(n/g) - 4) \cdot O(1) = O(n/g)$.
5. Total time complexity: $O(n) + O(n/g) = O(n)$.

Time Complexity

Let n = #points on object border, g = grid size.

1. Checking object containment in a cell: $O(g)$ time.
2. #grid points visited: $O(n/g)$
   $\Rightarrow$ Visiting all vertices: $O(n/g) \cdot O(g) = O(n)$ time.
3. Removal of a concavity (applying Rule): $O(1)$ time.
4. Maximum #reductions: $O(n/g) - 4$.
   $\Rightarrow$ Total #operations: $(O(n/g) - 4) \cdot O(1) = O(n/g)$.
5. Total time complexity: $O(n) + O(n/g) = O(n)$.

Time Complexity

Let n = #points on object border, g = grid size.

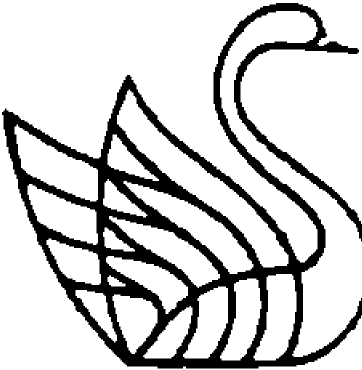
1. Checking object containment in a cell: $O(g)$ time.
2. #grid points visited: $O(n/g)$
   $\Rightarrow$ Visiting all vertices: $O(n/g) \cdot O(g) = O(n)$ time.
3. Removal of a concavity (applying Rule): $O(1)$ time.
4. Maximum #reductions: $O(n/g) - 4$.
   $\Rightarrow$ Total #operations: $(O(n/g) - 4) \cdot O(1) = O(n/g)$.
5. Total time complexity: $O(n) + O(n/g) = O(n)$.

Time Complexity

Let $n = \#$points on object border, $g =$ grid size.

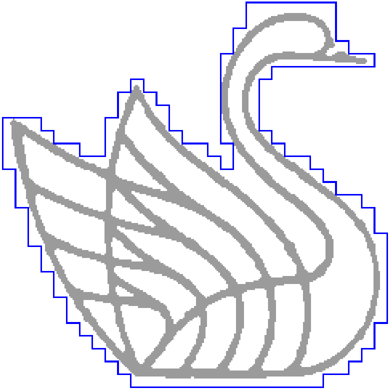
1. Checking object containment in a cell: $O(g)$ time.
2. #grid points visited: $O(n/g)$
   $\Rightarrow$ Visiting all vertices: $O(n/g) \cdot O(g) = O(n)$ time.
3. Removal of a concavity (applying Rule): $O(1)$ time.
4. Maximum #reductions: $O(n/g) - 4$.
   $\Rightarrow$ Total #operations: $(O(n/g) - 4) \cdot O(1) = O(n/g)$.
5. Total time complexity: $O(n) + O(n/g) = O(n)$.

Result



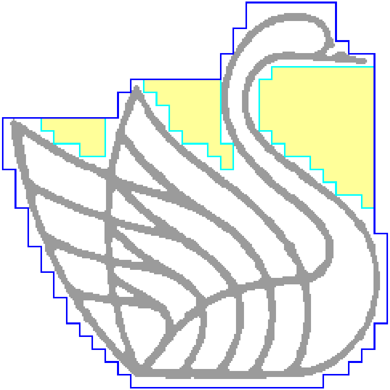
digital object = 10541 points

Result



Isothetic cover

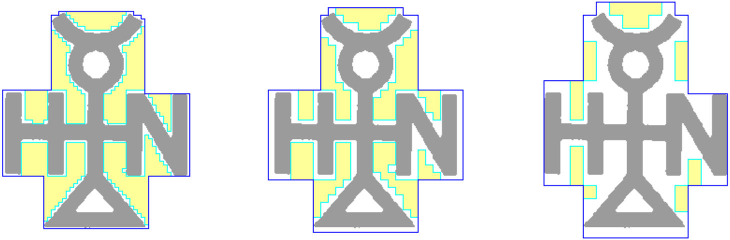
Result



Orthogonal hull

Result

$$g = 4, 8, 14$$



$$\#\text{vertices} = 18, 16, 16$$

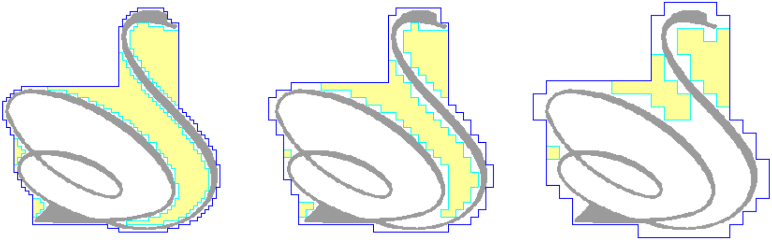
Result

$$g = 4, 8, 14$$

$$\#\text{vertices} = 120, 60, 32$$

Result

$g = 4, 8, 14$



$\#\text{vertices} = 88, 44, 32$

Feature analysis

- Concavity strength and concavity relation
- Narrow mouthed concavity
- Concavity complexity

Hulls

P Bhowmick

References

- A. Biswas, P. Bhowmick, M. Sarkar, B. B. Bhattacharya, A linear-time combinatorial algorithm to find the orthogonal hull of an object on the digital plane, *Information Sciences*, **216**, pp. 176–195, 2012.
- A. Biswas, P. Bhowmick, B. B. Bhattacharya, Construction of isothetic covers of a digital object: A combinatorial approach, *Journal of Visual Communication and Image Representation*, **21**, pp. 295–310, 2010.

Thank you