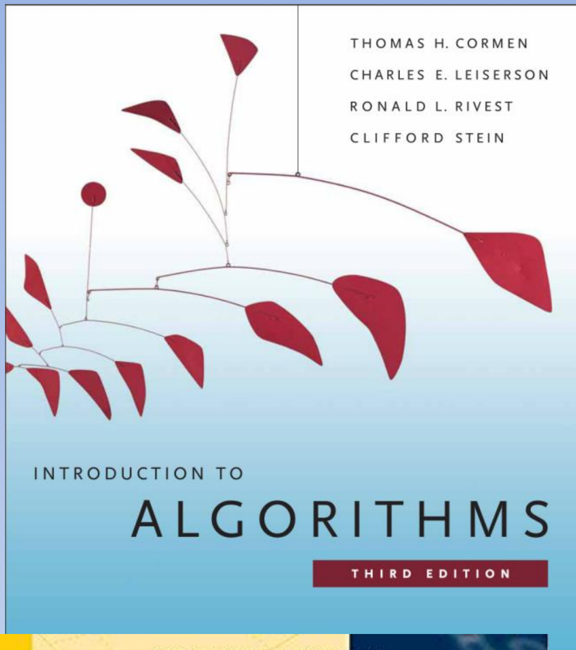
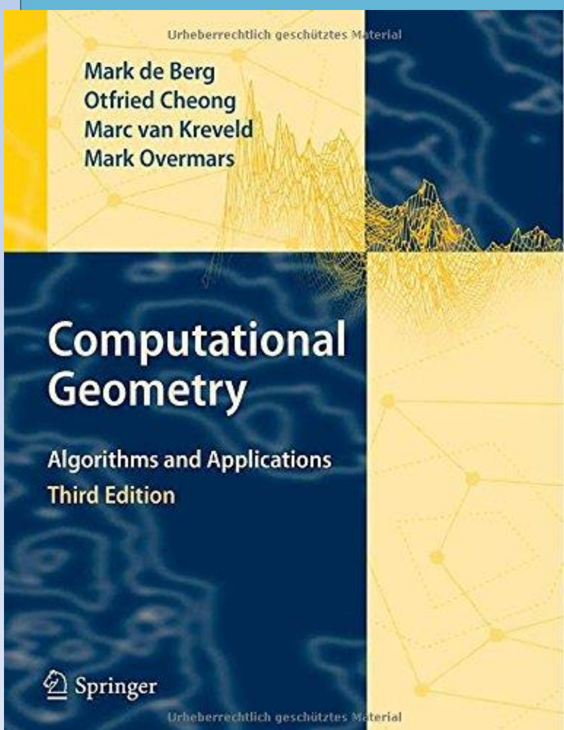
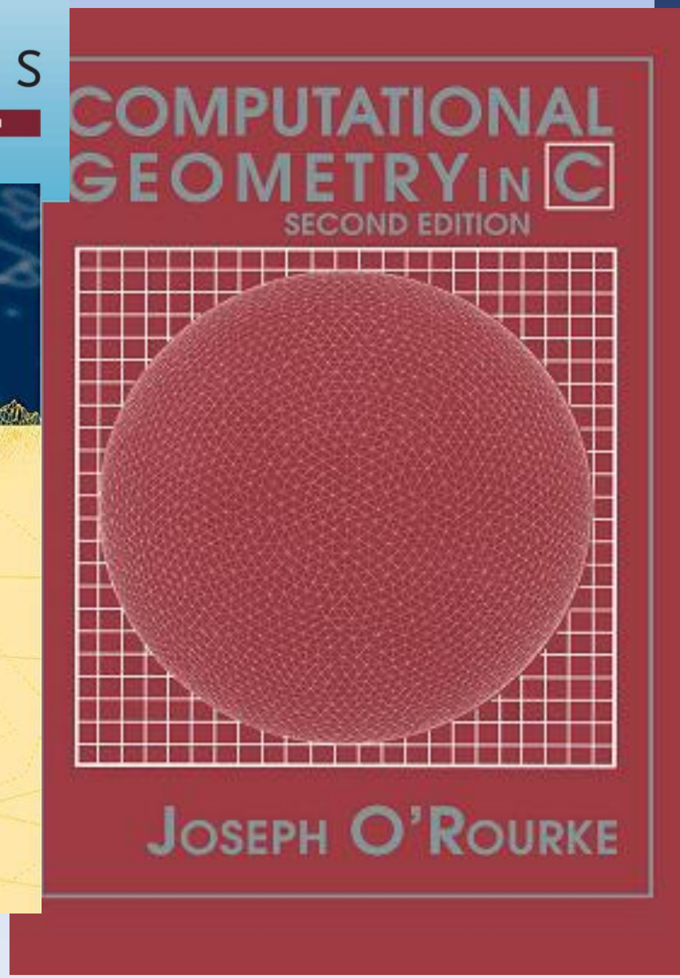
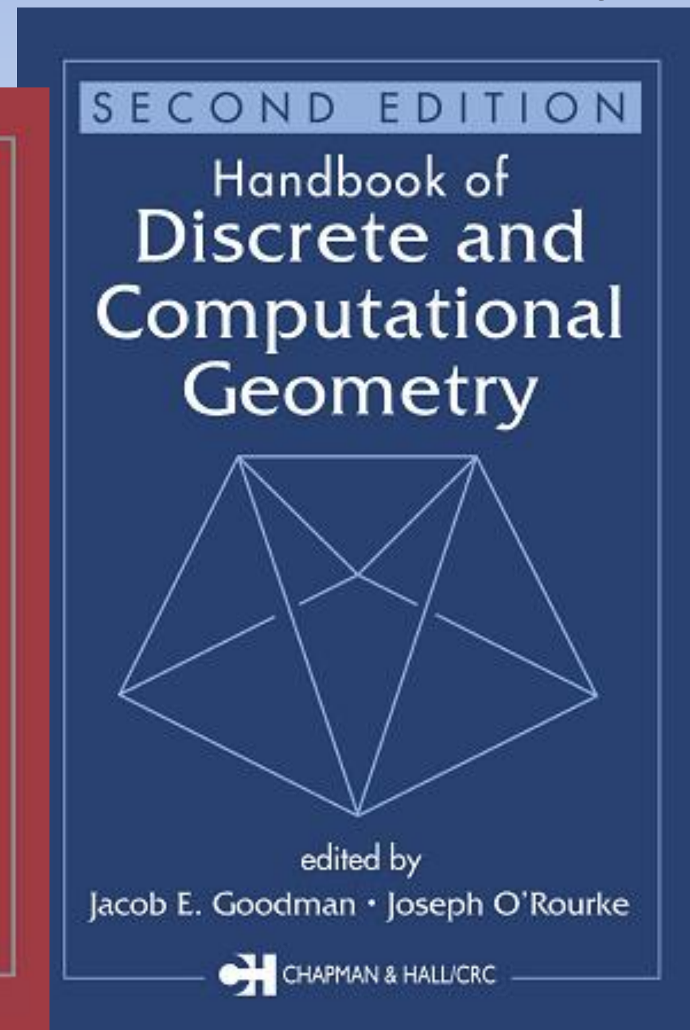
Design and Analysis of Algorithms : Lecture 4

Topic: Computational Geometry

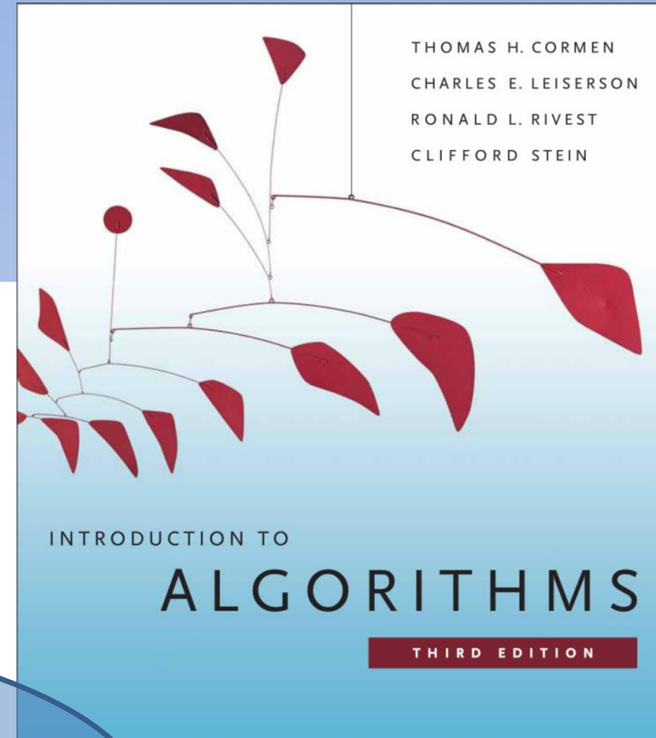








Chapter 33
Computation Geometry

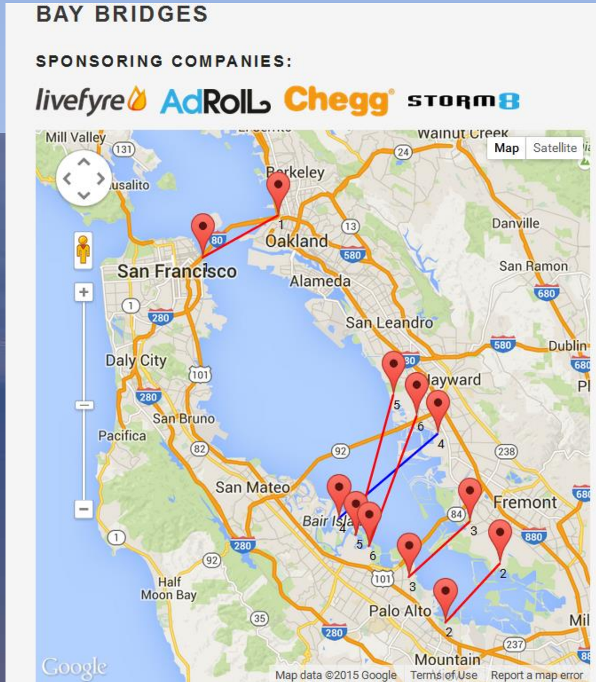


Computational Geometry

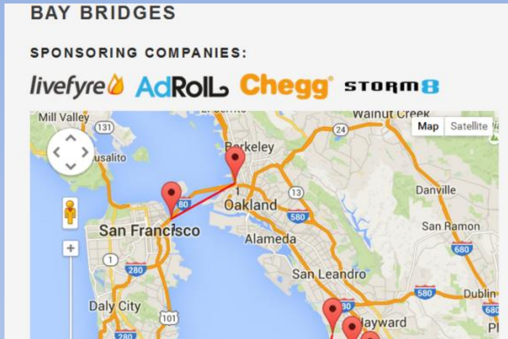
- Branch of Computer Science that studies ALGORITHMS for solving geometric problems.
- Applications to:
  1) Computer Graphics
  2) Robotics
  3) VLSI design
  4) Computer-Aided Design
  5) Molecular modeling
  6) Metallurgy
  7) Manufacturing
  8) Textile layout
  9) Forestry
  10) Statistics

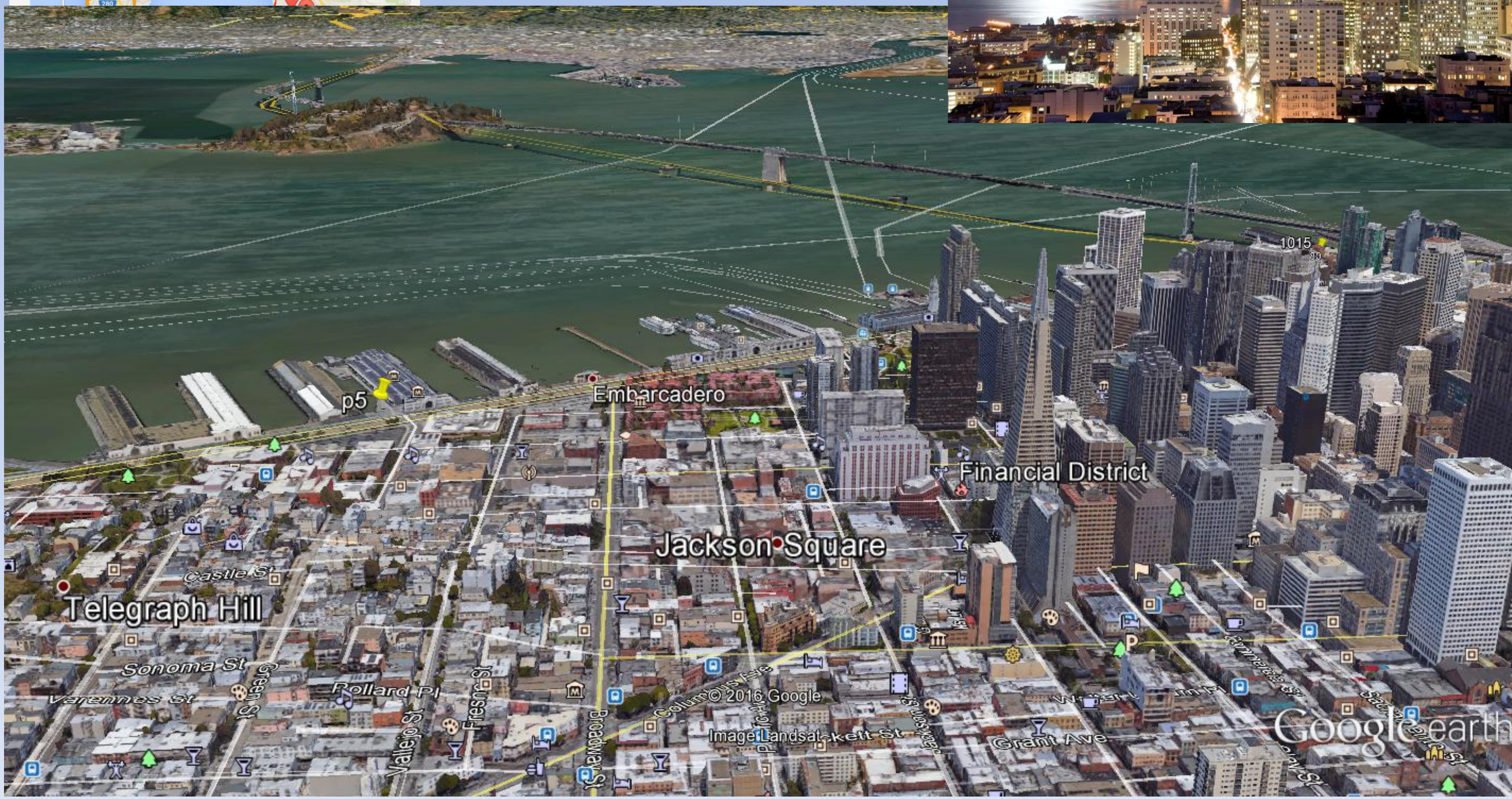
  Many More..

Bay Bridges Challenge



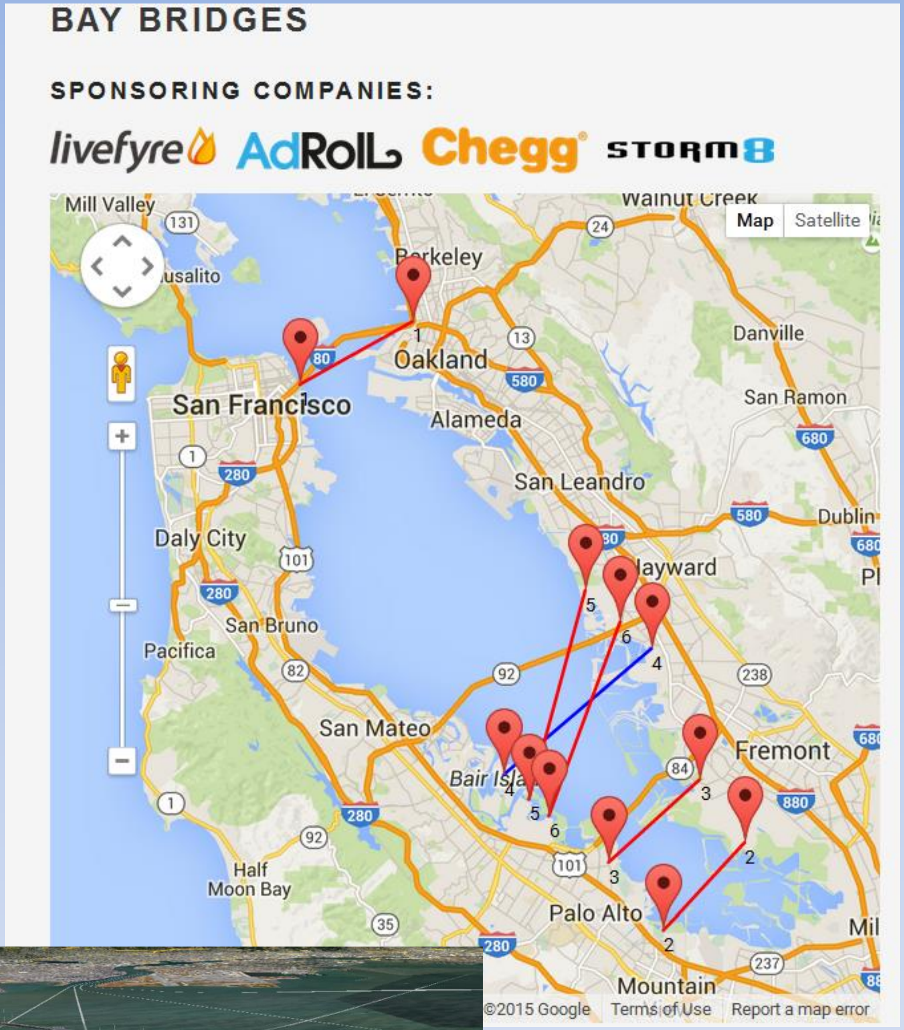
Bay Bridges Challenge

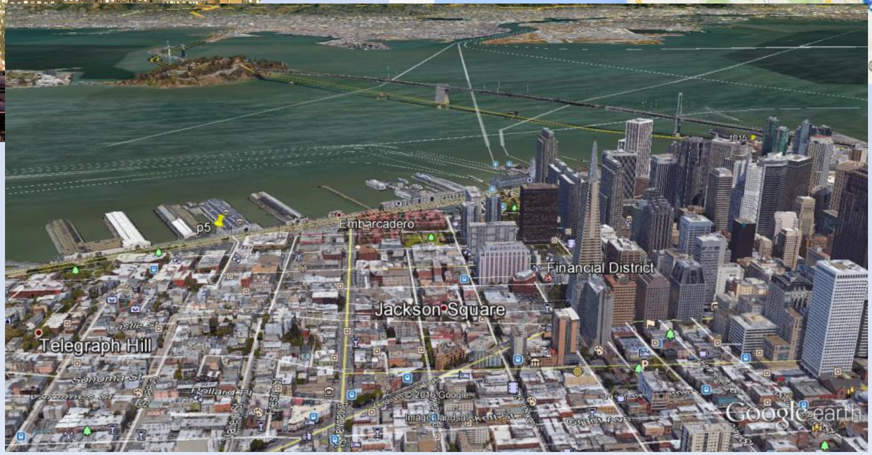




Bay Bridges Challenge

- Build Non-Overlapping Bridges across the Bay!





Computation Geometry: Problem Representations

- Input often a set of geometric objects
  - Set of points
  - Set of line segments
  - Vertices of a polygon in counterclockwise order
- Output often Query w/ Objects
  - Lines Intersect?
- Output also New Object
  - Convex Hull of Set of Points
    - Smallest Enclosing Convex Polygon

Chapter 33:
Topics

- Computational-Geometry Algorithms in Two Dimensions
- Input objects are set of points
- Of Course: higher dimensions also possible
- But: Good Sample with Two Dimensions

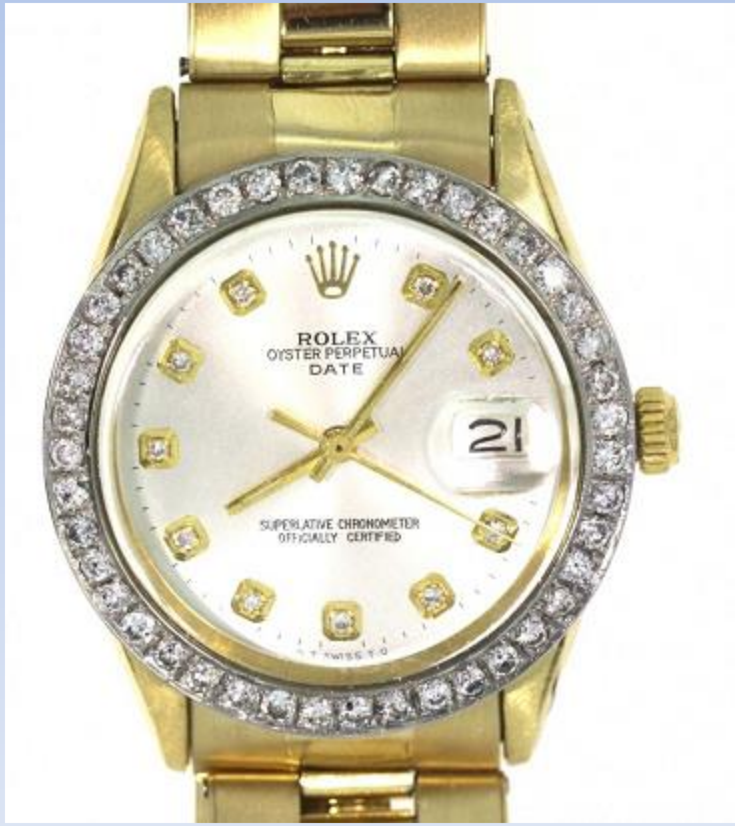
Section 33.1
Queries w/ Line Segments

- Efficiently & Accurately answer queries w/ Line Segments

QUERY:

Is Segment Clockwise or Counterclockwise from another that shares an endpoint?

- Big Hand Clockwise of Little Hand?



QUERY:

Is Segment Clockwise or Counterclockwise from another that shares an endpoint?

- Big Hand Clockwise of Little Hand?

Assume Hands End @ center

QUERY:

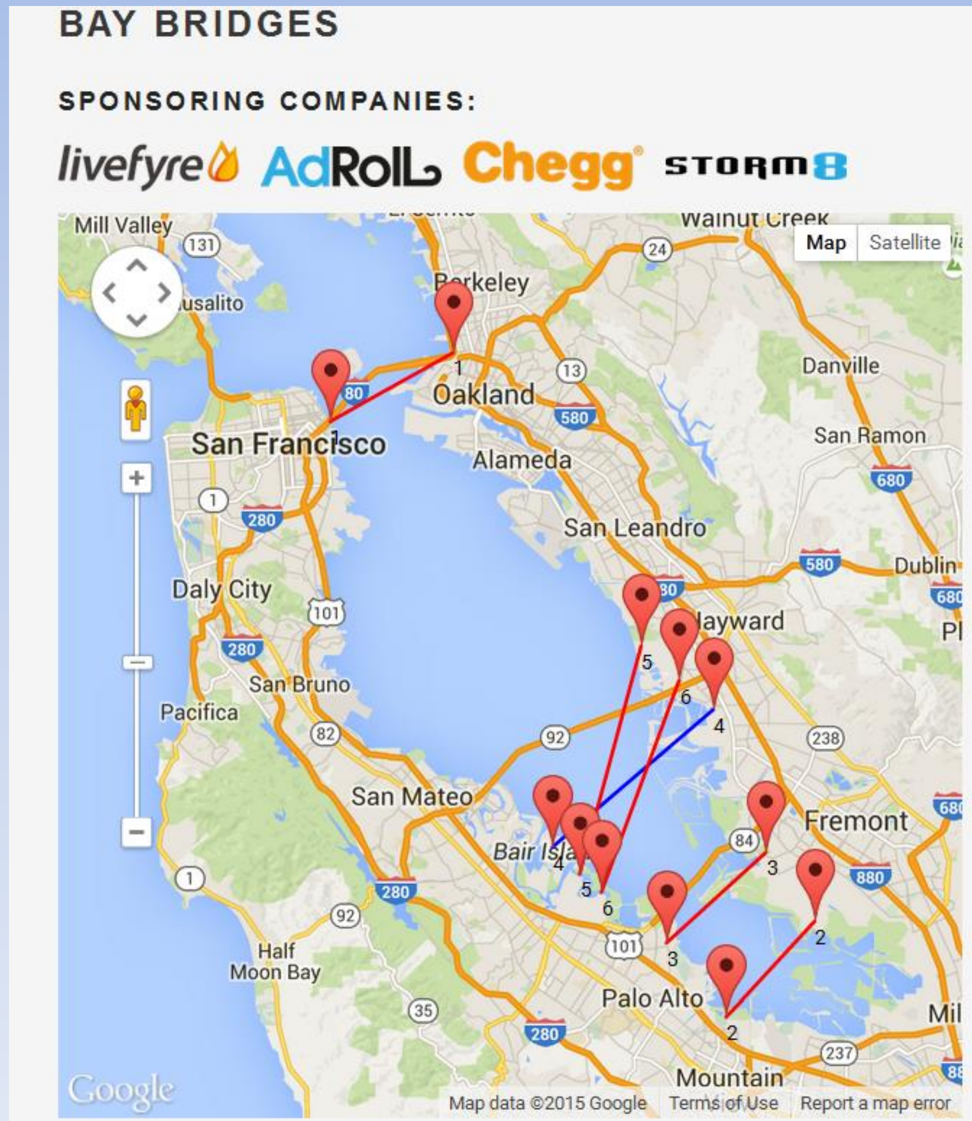
Is Segment Clockwise or Counterclockwise from another that shares an endpoint?

- Big Hand Clockwise of Little Hand?

Assume Hands End @ center

- If we traverse from Little Hand to Big hand do we make a left turn @ center?

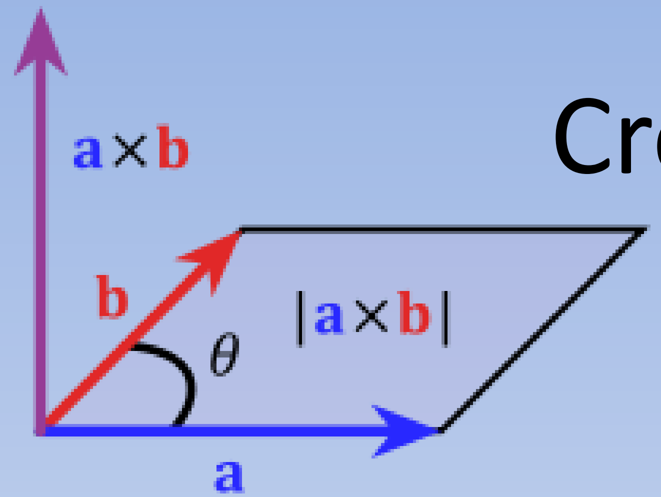
Query:
Line Segments Intersection?

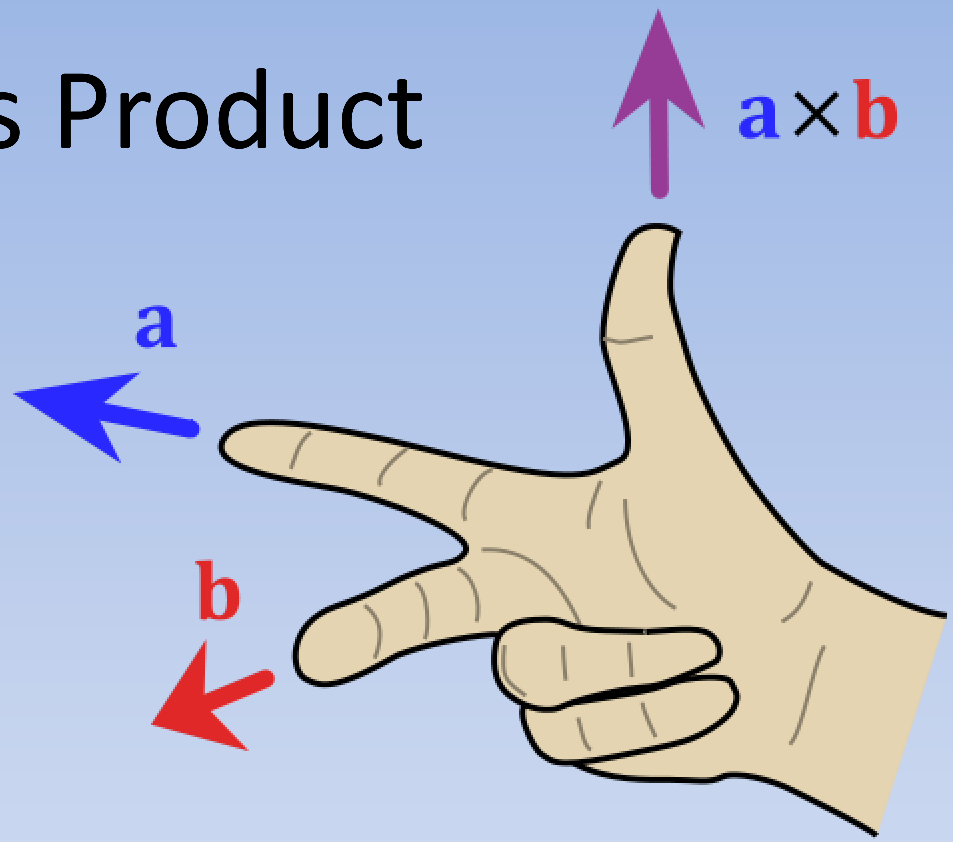


Line Segment Properties

- Line Segment Properties:
  - Given two directed segments $\overrightarrow{p_0p_1}$ and $\overrightarrow{p_0p_2}$, is $\overrightarrow{p_0p_1}$ clockwise from $\overrightarrow{p_0p_2}$ with respect to their common endpoint p_0 ?
  - Given two line segments $\overrightarrow{p_0p_1}$ and $\overrightarrow{p_0p_2}$, if we traverse $\overrightarrow{p_0p_1}$ and then $\overrightarrow{p_1p_2}$, do we make a left turn at p_1 ?
  - Do line segments $\overrightarrow{p_1p_2}$ and $\overrightarrow{p_3p_4}$ intersect?
- O(1) time (constant) required for these queries!
- MOREOVER: our methods use only additions, subtractions, multiplications, and comparisons:
  - Divisions and trigonometric functions expensive and prone to problems with round-off errors.

Cross Product



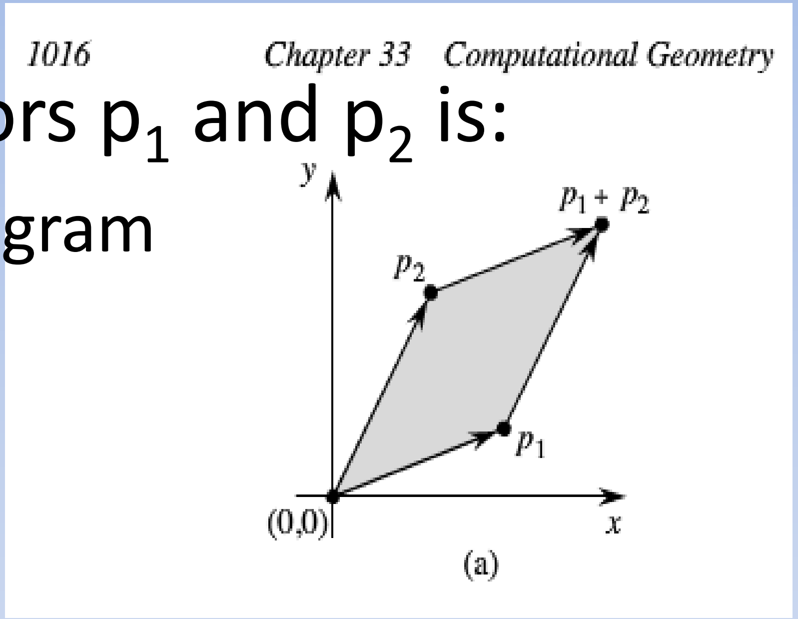


- Cross Product of two vectors p_1 and p_2 is:
  - Vector perpendicular to p_1 and p_2 according to the "right-hand rule"
  - Magnitude of vector is $|x_1 y_2 - x_2 y_1|$

Wikipedia: http://en.wikipedia.org/wiki/Cross_product#CITEREFWilson1901

Cross Product

- Cross Product of two vectors p_1 and p_2 is:
  - Signed area of the parallelogram

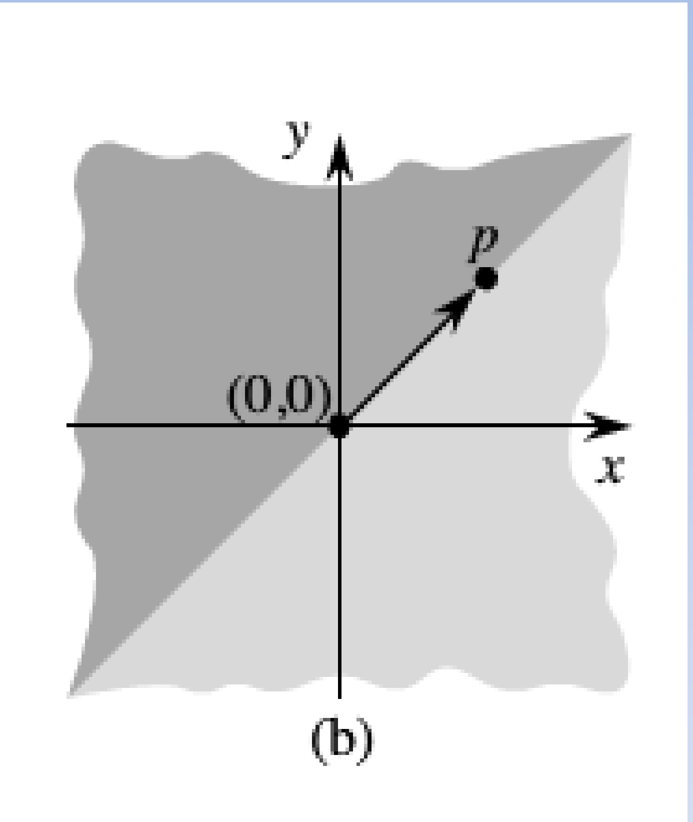


Cross Product as Determinant

- $p_1 \times p_2 = \det\begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix}$
$$= x_1y_2 - x_2y_1$$
$$= \ -p_2 \times \mathrm{p}_1$$
- For our Purposes: treat cross product as value
$$x_1y_2 - x_2y_1$$

Sign of the Crossproduct: Clockwise versus Counterclockwise

- Sign of the Crossproduct p_1 X p_2 is positive, then p_1 is clockwise from p_2.
- Sign of the Crossproduct p_1 X p_2 is 0, then p_1 and p_2 are colinear.
- Dark Region contains vectors that are CounterClockwise with respect to P
  - Sign of the Crossproduct Negative

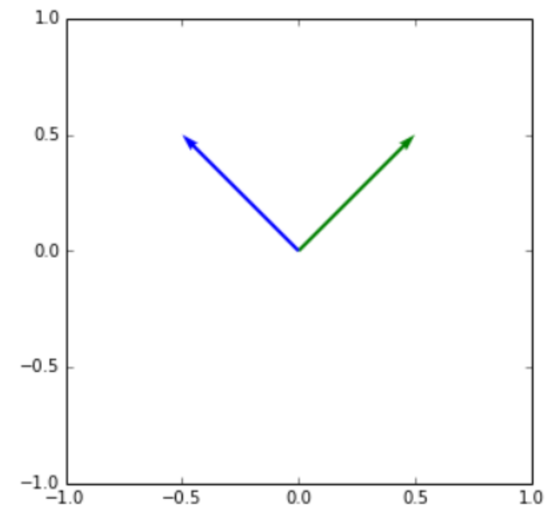


Clockwise Versus Counterclockwise

```python
def CrossProduct(p1, p2):
    (x1, y1) = p1
    (x2, y2) = p2
    return x1*y2 - x2*y1
```

Example 1

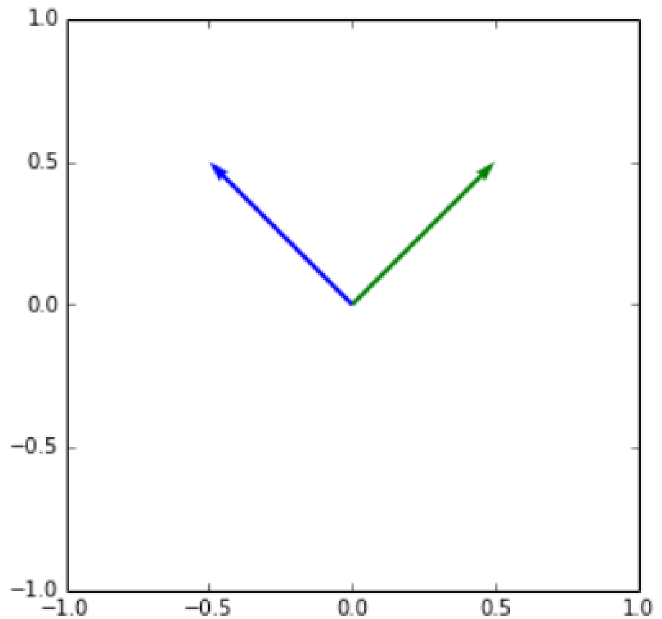
```python
p1 = (0.5, 0.5)
p2 = (-0.5, 0.5)
PlotVectorP1P2(p1, p2)
print "P1(G) X P2(B): ", CrossProduct(p1, p2)
```




```
P1(G) X P2(B):  0.5
```

- Determine:
  - With respect to a common end point p_0
  - is $\overrightarrow{p_0 p_1}$ is closer to $\overrightarrow{p_0 p_2}$ in a clockwise or counterclockwise direction

```python
def CrossProduct(p1, p2):
    (x1, y1) = p1
    (x2, y2) = p2
    return x1*y2 - x2*y1
```




```
P1(G) X P2(B):  0.5
```

Example 1

```python
p1 = (0.5, 0.5)
p2 = (-0.5, 0.5)
PlotVectorP1P2(p1, p2)
print "P(G)1 X P2(B): ", CrossProduct(p1, p2)
```

- Points are tuples: (x, y)
  - (0, 0), (1, 2)
- CrossProduct is really just:
  - Given: (x1, y1), (x2, y2)
  - Return: x1*y2 – x2*y1

Clockwise Versus Counterclockwise

- Translate to use p_0 as origin:
  - $p_1 - p_0$ denotes vector $p'_1 = (x_1', y_1')$
  - $x_1' = x_1 - x_0$
  - $y_1' = y_1 - y_0$
  - $p_2 - p_0$ defined similarly
- Compute Cross Product:
  - $(p_1 - p_0)$ X $(p_2 - p_0)$
  - = $(x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)$ X $(y_1 - y_0)$
- If Cross Product :
  - Positive: $\overrightarrow{p_0 p_1}$ is clockwise from $\overrightarrow{p_0 p_2}$
  - Negative: $\overrightarrow{p_0 p_1}$ is *counter*clockwise from $\overrightarrow{p_0 p_2}$

Cross Product: Left Turn or Right Turn

33.1 Line-segment properties *1017*

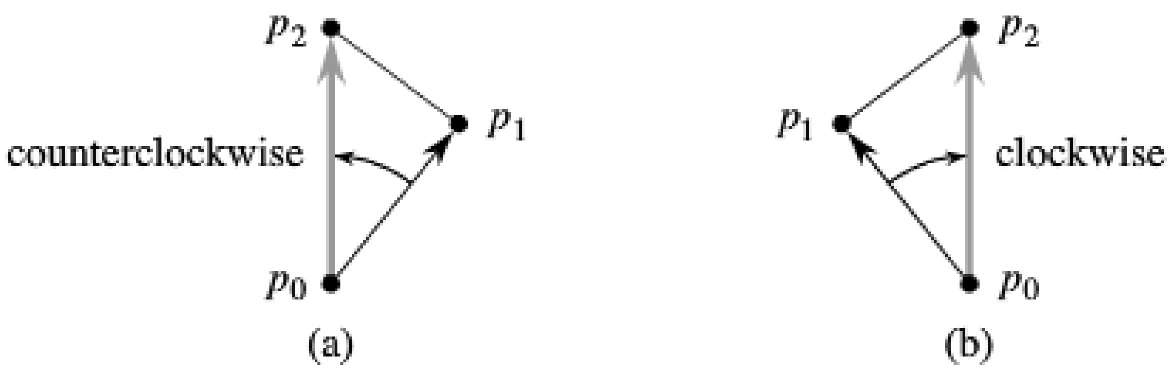


Figure 33.2 Using the cross product to determine how consecutive line segments $\overline{p_0 p_1}$ and $\overline{p_1 p_2}$ turn at point p_1. We check whether the directed segment $\overrightarrow{p_0 p_2}$ is clockwise or counterclockwise relative to the directed segment $\overrightarrow{p_0 p_1}$. **(a)** If counterclockwise, the points make a left turn. **(b)** If clockwise, they make a right turn.

- Cross products answer query without angles!

Example 1

- P0 = (4, 1)
- P1 = (1, 3)
- P2 = (7, 7)

Intersection Checking

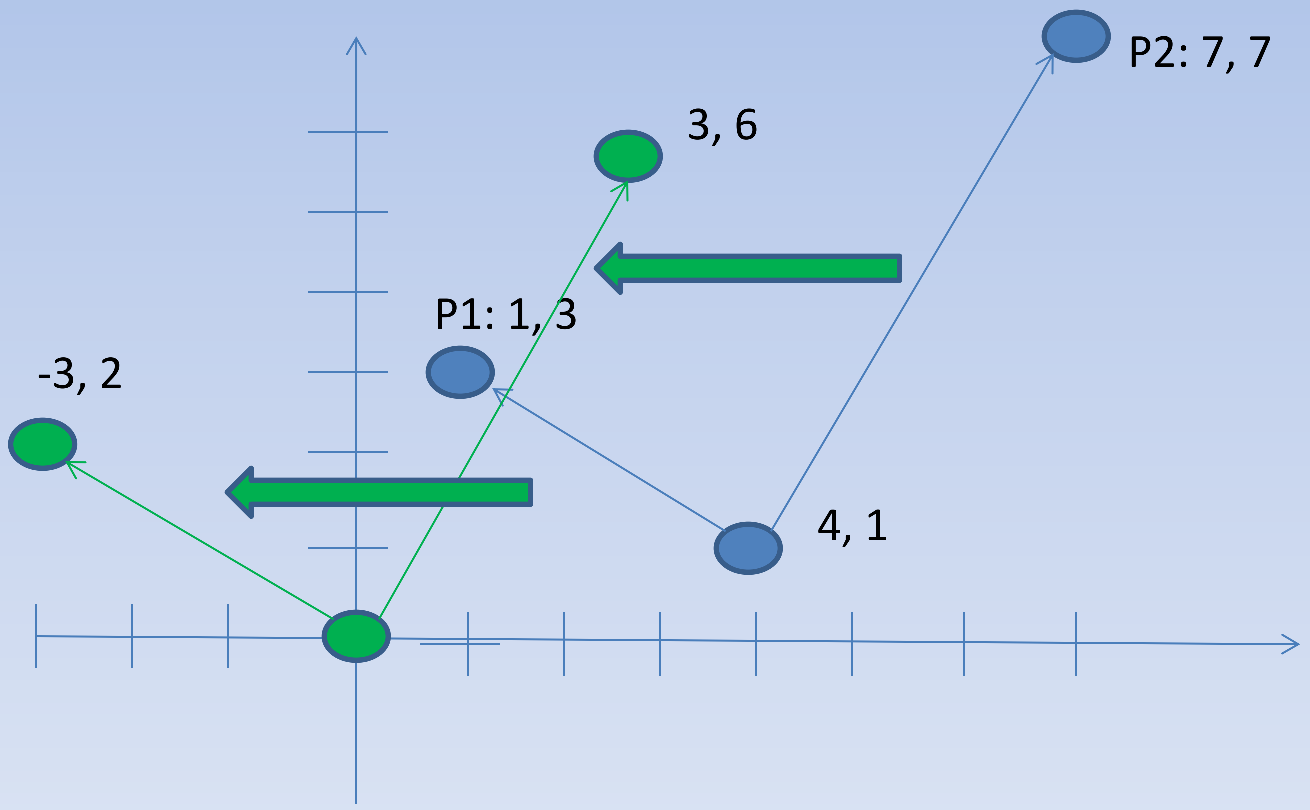
Code checking intersecting line segments.

```
def Direction(p0In, p1In, p2In):
    p1 = p2In[0]-p0In[0], p2In[1]-p0In[1]
    p2 = p1In[0]-p0In[0], p1In[1]-p0In[1]
    return CrossProduct(p1, p2)

def PrintDirection(p0, p1, p2):
    cp = Direction(p0, p1, p2)
    if cp < 0:
        print "CounterClockwise: Left Turn"
    elif cp > 0:
        print "Clockwise: Right Turn"
    else:
        print "CoIncident"
```

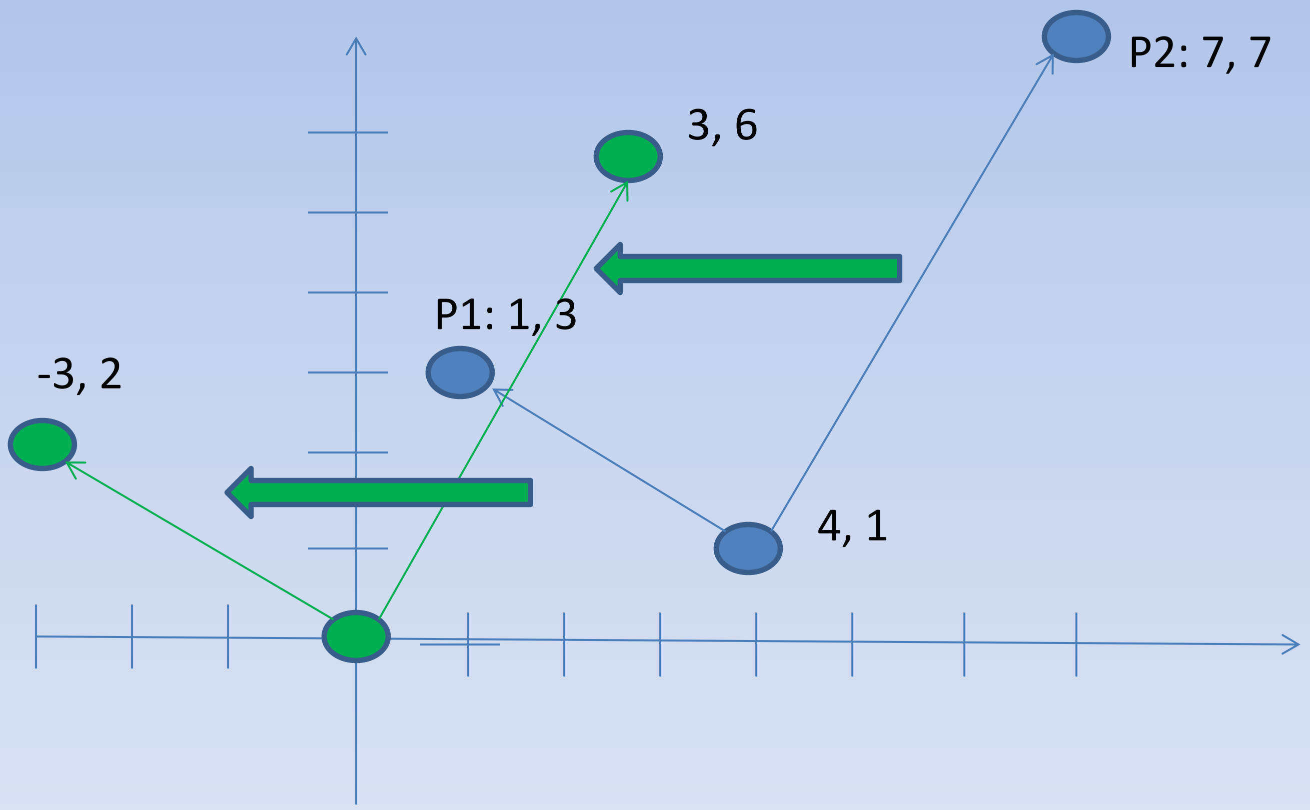
Start by moving vectors to the origin!

- Go from Vector p0->p1
- To Vector (0,0)->(p1-p0)

Move Vectors to Origin

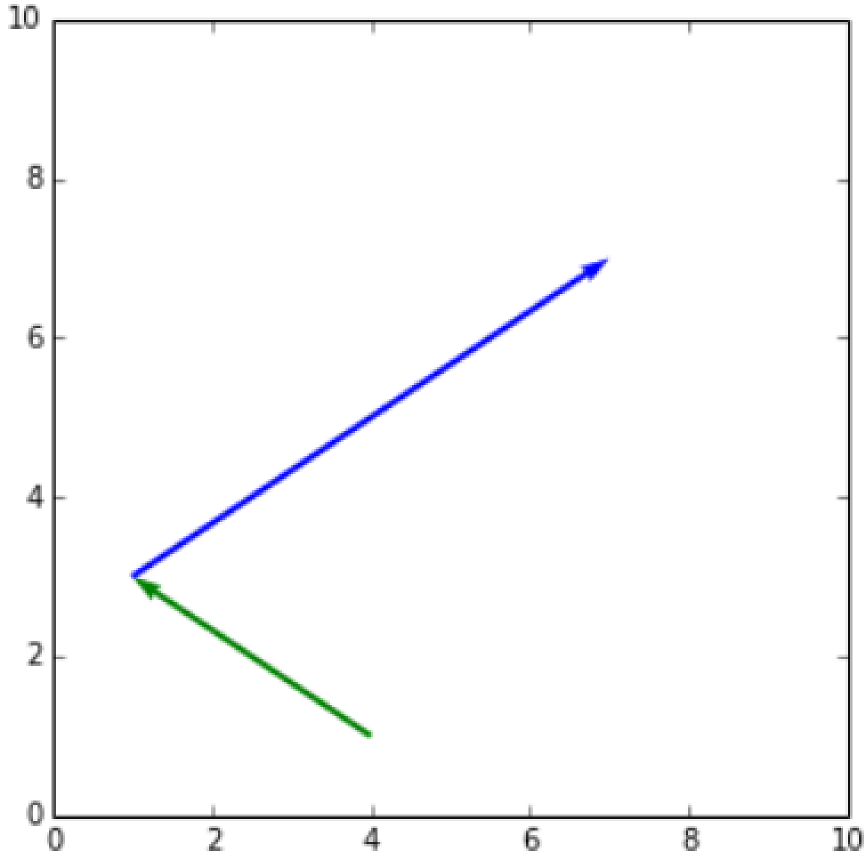


Sign of the Crossproduct Calculated from Origin



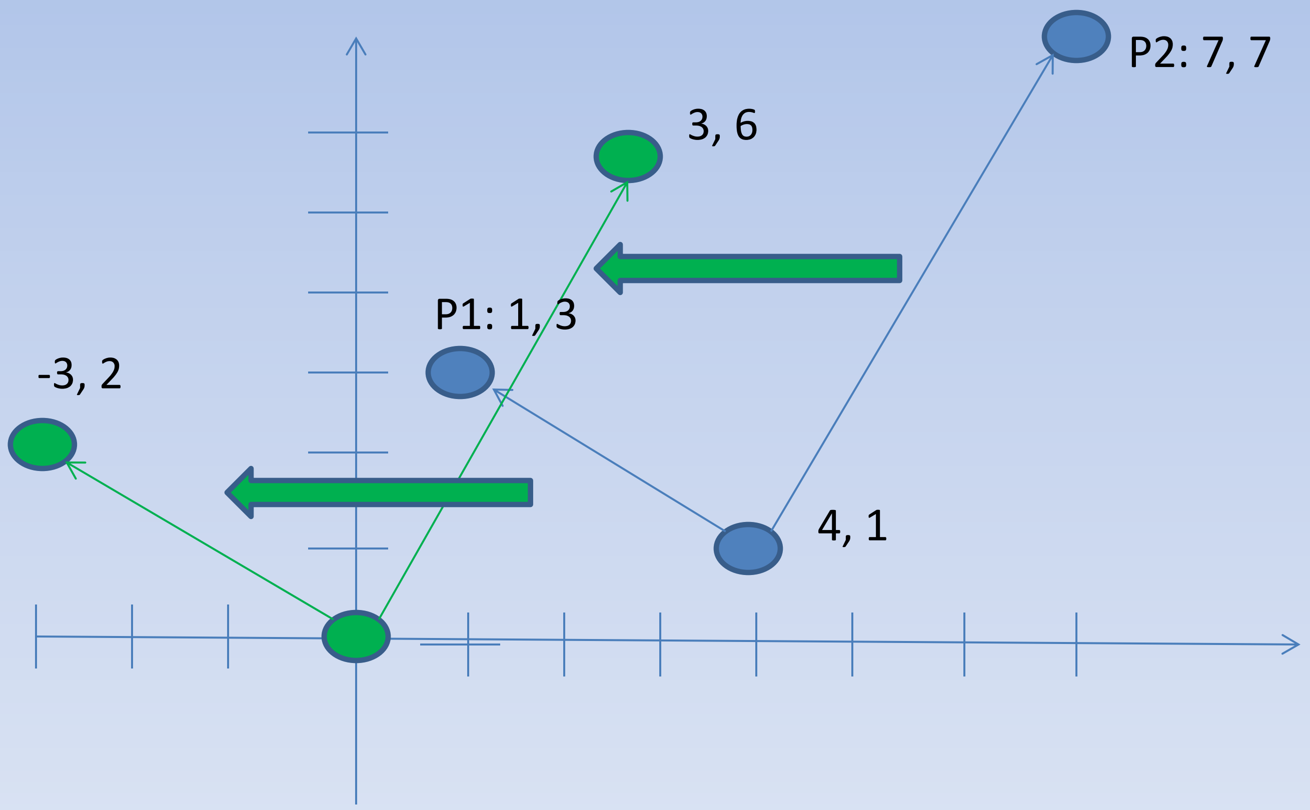
return (-3,2) X (3,6) =(-3*6) – (2*3) = -18 – 6 = -24

```
p0 = (4, 1)
p1 = (1, 3)
p2 = (7, 7)
PlotVectorP0P1P2(p0, p1, p2)
print "Direction(p0, p1, p2) ", Direction(p0, p1, p2)
PrintDirection(p0, p1, p2)
```




```
Direction(p0, p1, p2)  24
Clockwise: Right Turn
```

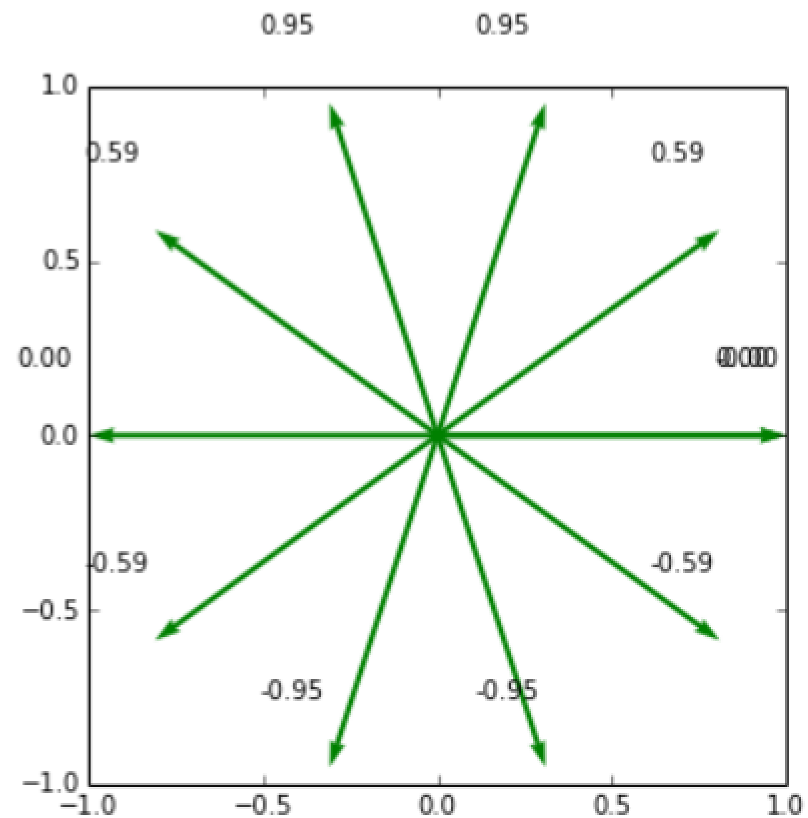
Sign of the Crossproduct Calculated from Origin



return (3,6) X (-3,2) =(3*2) – (6*(-3)) = 6 – (-18) = 24

Sign of CrossProduct as a Measure of Angle

```
pointsRads = [i*(2*math.pi)/10 for i in range(11)]
points=[(math.cos(a), math.sin(a)) for a in pointsRads]
c = ["{0:.2f}".format(CrossProduct((1,0), p)) for p in points]
PlotVectors(points,c)
```



Next Steps

- Now we want to use this clockwise/counterclockwise info to determine line segment intersection!
- Determining if Line Segs Intersect define **STRADLE.**

Straddling & Intersections

- Determining if Line Segs Intersect define **STRADLE.**
- STRADLE: A segment $\overline{p_1 p_2}$ **straddles** a line if :
  - Point p_1 lies on one side of the line
  - Point p_2 lies on the other side of the line
- Two line segments INTERSECT if and only if either (or both) of the following conditions holds :
  1. Each segment straddles the line containing the other.
  2. An endpoint of one segment lies on the other segment.

Straddling

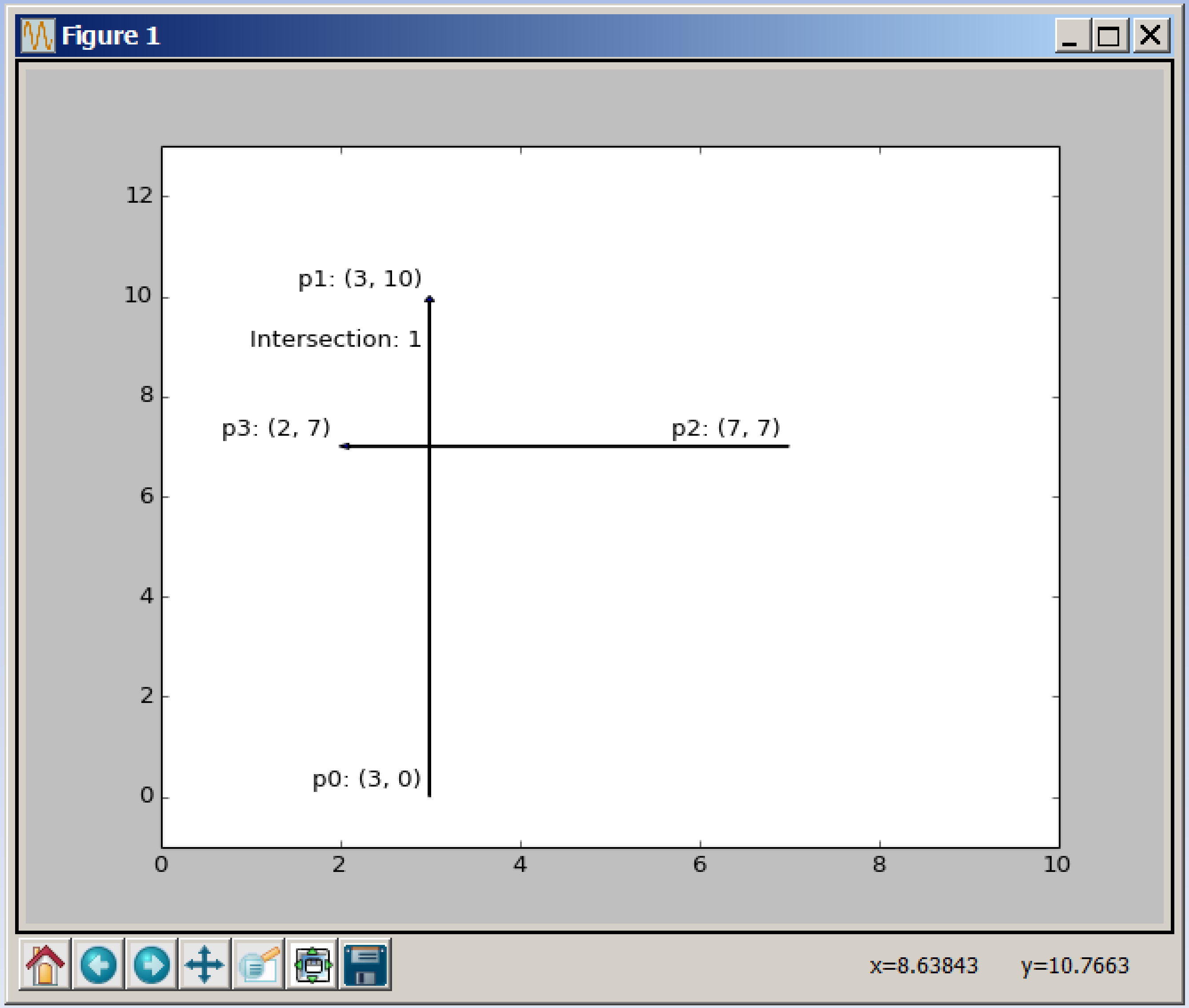
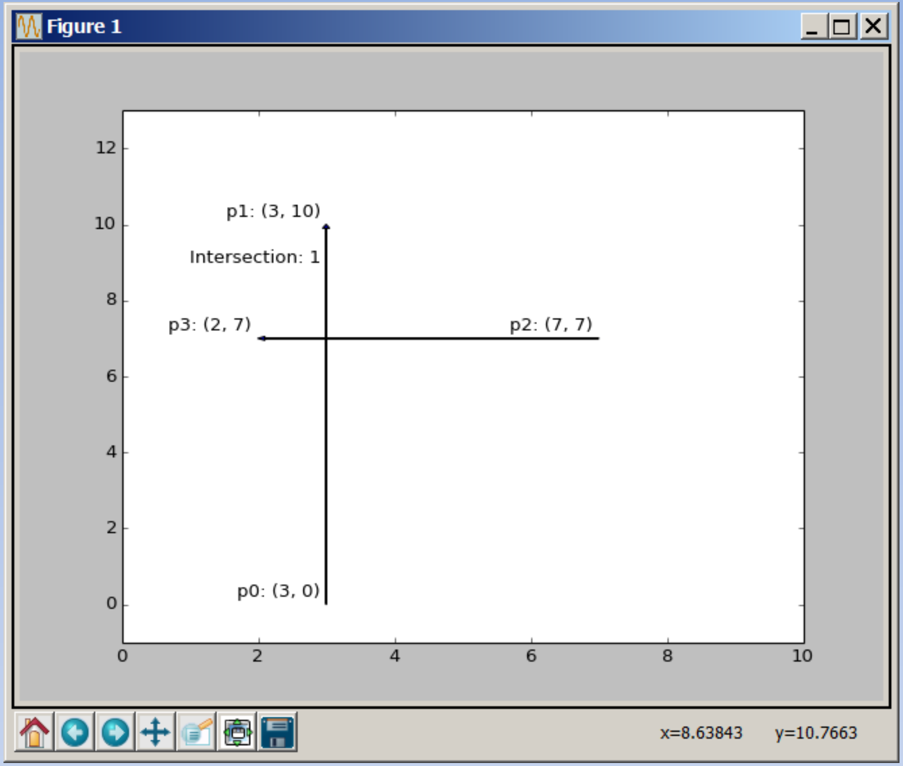


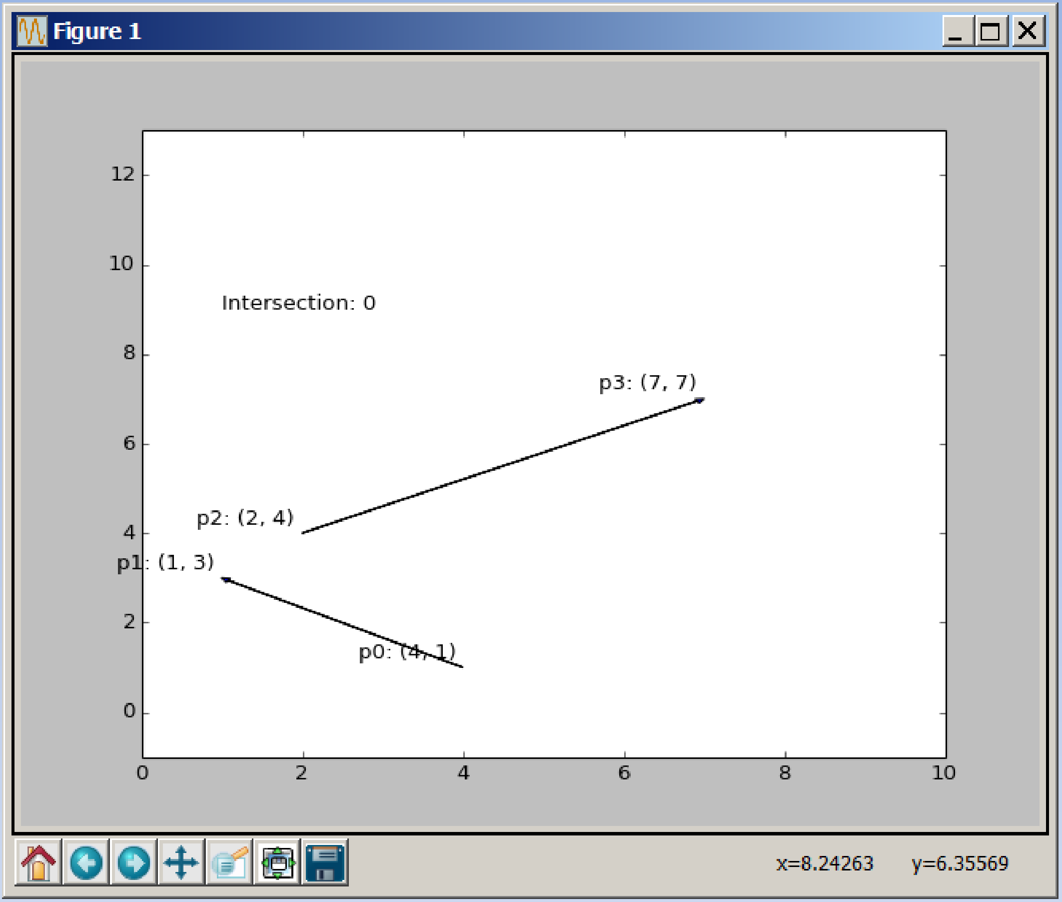
Figure 1
p1: (3, 10)
Intersection: 1
p3: (2, 7)
p2: (7, 7)
p0: (3, 0)
12
10
8
6
4
2
0
0
2
4
6
8
10
x=8.63843
y=10.7663

Straddling



- Line 1: p3: (2, 7), p2: (7, 7)
- Line 2: p0: (3,0), p1:(3, 10)

Not Straddling



- Line 1: p2:(2,4), p3:(7,7)
- Line 2: p0:(4,1), p1:(1,3)
  - Points p2 & p3 are both clockwise of line segment p0,p1!

Algorithm

SEGMENTS-INTERSECT(p_1, p_2, p_3, p_4)

```
SEGMENTS-INTERSECT(p1, p2, p3, p4)
    d1 = DIRECTION(p3, p4, p1)
    d2 = DIRECTION(p3, p4, p2)
    d3 = DIRECTION(p1, p2, p3)
    d4 = DIRECTION(p1, p2, p4)
    if ((d1 > 0 and d2 < 0) or (d1 < 0 and d2 > 0)) and
            ((d3 > 0 and d4 < 0) or (d3 < 0 and d4 > 0))
        return TRUE
    elseif d1 == 0 and ON-SEGMENT(p3, p4, p1)
        return TRUE
    elseif d2 == 0 and ON-SEGMENT(p3, p4, p2)
        return TRUE
    elseif d3 == 0 and ON-SEGMENT(p1, p2, p3)
        return TRUE
    elseif d4 == 0 and ON-SEGMENT(p1, p2, p4)
        return TRUE
    else return FALSE
```

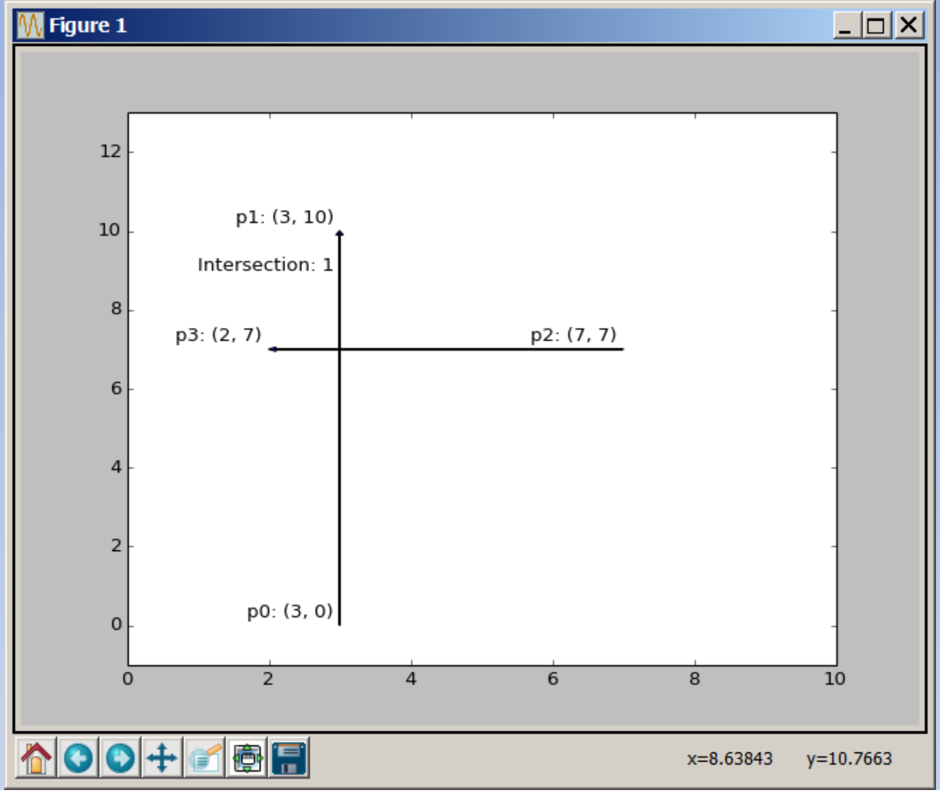
Algorithm

SEGMENTS-INTERSECT(p_1, p_2, p_3, p_4)

```
DIRECTION(p_i, p_j, p_k)
    return (p_k − p_i) × (p_j − p_i)

ON-SEGMENT(p_i, p_j, p_k)
    if min(x_i, x_j) ≤ x_k ≤ max(x_i, x_j) and min(y_i, y_j) ≤ y_k ≤ max(y_i, y_j)
        return TRUE
    else return FALSE
```

```
    elseif d_1 == 0 and ON-SEGMENT(p_3, p_4, p_1)
        return TRUE
    elseif d_2 == 0 and ON-SEGMENT(p_3, p_4, p_2)
        return TRUE
    elseif d_3 == 0 and ON-SEGMENT(p_1, p_2, p_3)
        return TRUE
    elseif d_4 == 0 and ON-SEGMENT(p_1, p_2, p_4)
        return TRUE
    else return FALSE
```

- Line 1: p0, p1
  - p_i, p_j
- Need to evalute p2 & p3 relative to Line 1.



DIRECTION(p_i, p_j, p_k)

return $(p_k - p_i) \times (p_j - p_i)$

ON-SEGMENT(p_i, p_j, p_k)

if $\min(x_i, x_j) \leq x_k \leq \max(x_i, x_j)$ and $\min(y_i, y_j) \leq y_k \leq \max(y_i, y_j)$

  return TRUE

else return FALSE

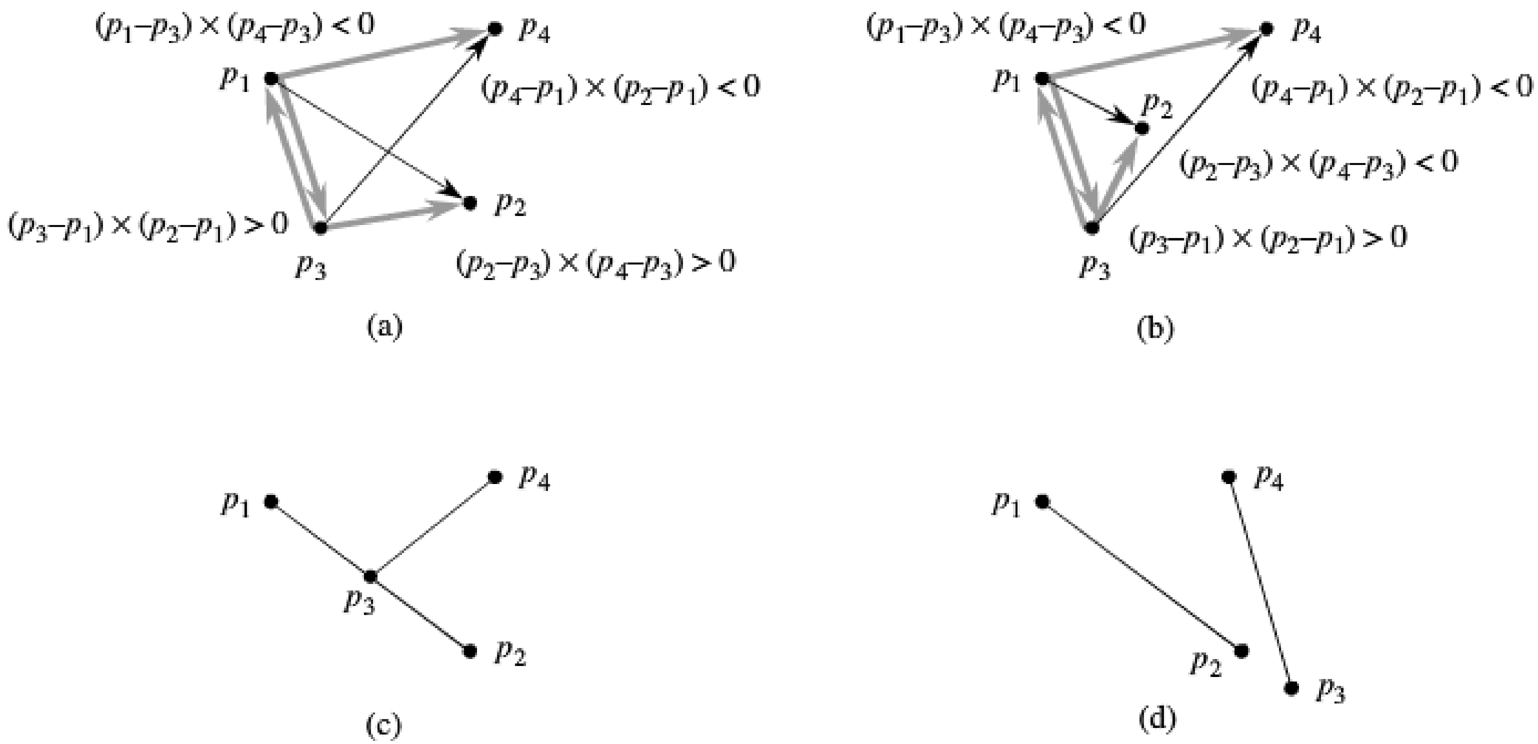
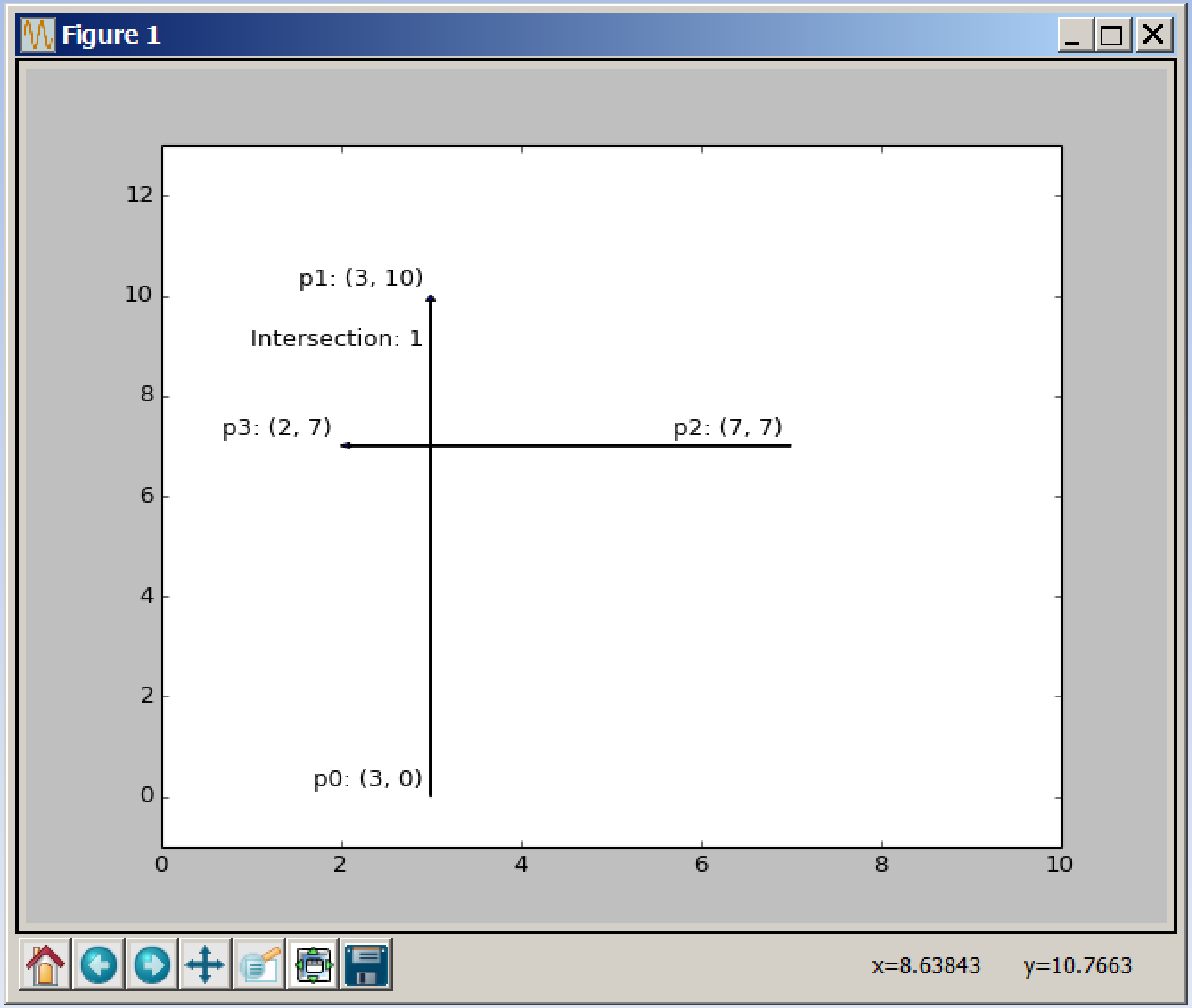


Figure 33.3 Cases in the procedure SEGMENTS-INTERSECT. **(a)** The segments $\overline{p_1 p_2}$ and $\overline{p_3 p_4}$ straddle each other's lines. Because $\overline{p_3 p_4}$ straddles the line containing $\overline{p_1 p_2}$, the signs of the cross products $(p_3 - p_1) \times (p_2 - p_1)$ and $(p_4 - p_1) \times (p_2 - p_1)$ differ. Because $\overline{p_1 p_2}$ straddles the line containing $\overline{p_3 p_4}$, the signs of the cross products $(p_1 - p_3) \times (p_4 - p_3)$ and $(p_2 - p_3) \times (p_4 - p_3)$ differ. **(b)** Segment $\overline{p_3 p_4}$ straddles the line containing $\overline{p_1 p_2}$, but $\overline{p_1 p_2}$ does not straddle the line containing $\overline{p_3 p_4}$. The signs of the cross products $(p_1 - p_3) \times (p_4 - p_3)$ and $(p_2 - p_3) \times (p_4 - p_3)$ are the same. **(c)** Point p_3 is colinear with $\overline{p_1 p_2}$ and is between p_1 and p_2. **(d)** Point p_3 is colinear with $\overline{p_1 p_2}$, but it is not between p_1 and p_2. The segments do not intersect.

Intersection



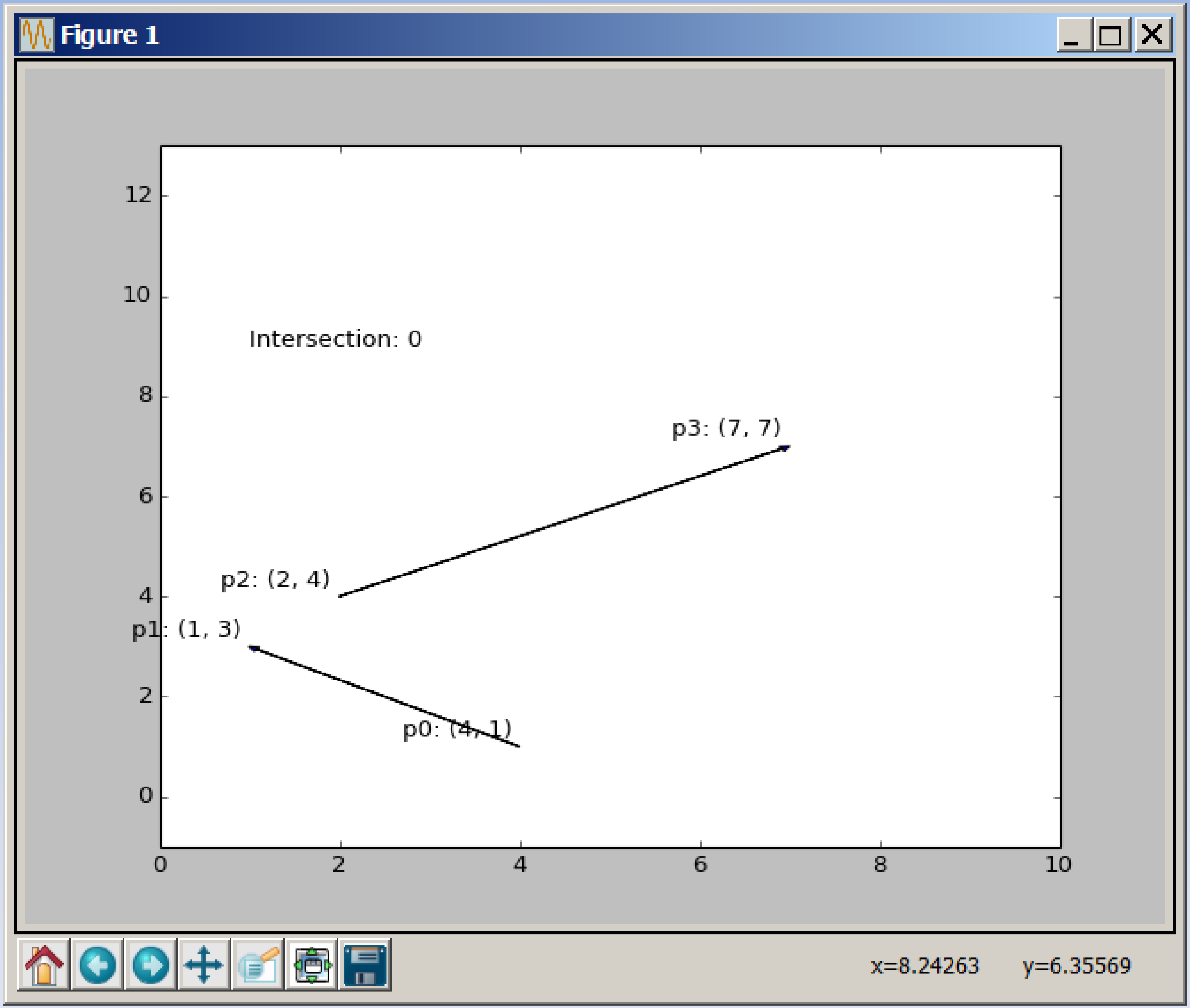
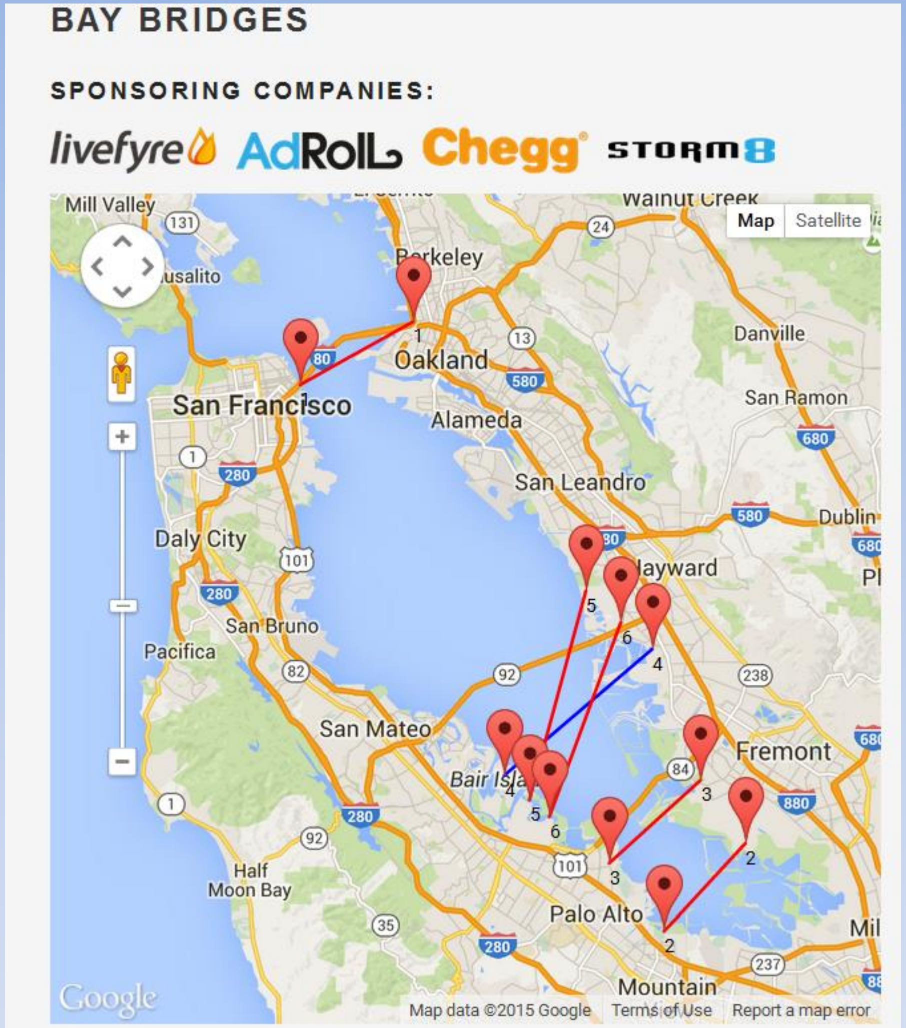


Figure 1
Intersection: 0
p3: (7, 7)
p2: (2, 4)
p1: (1, 3)
p0: (4, 1)
0
2
4
6
8
10
12
x=8.24263
y=6.35569

Bay Bridges Challenge

- Build Non-Overlapping Bridges across the Bay!



Bay Bridges

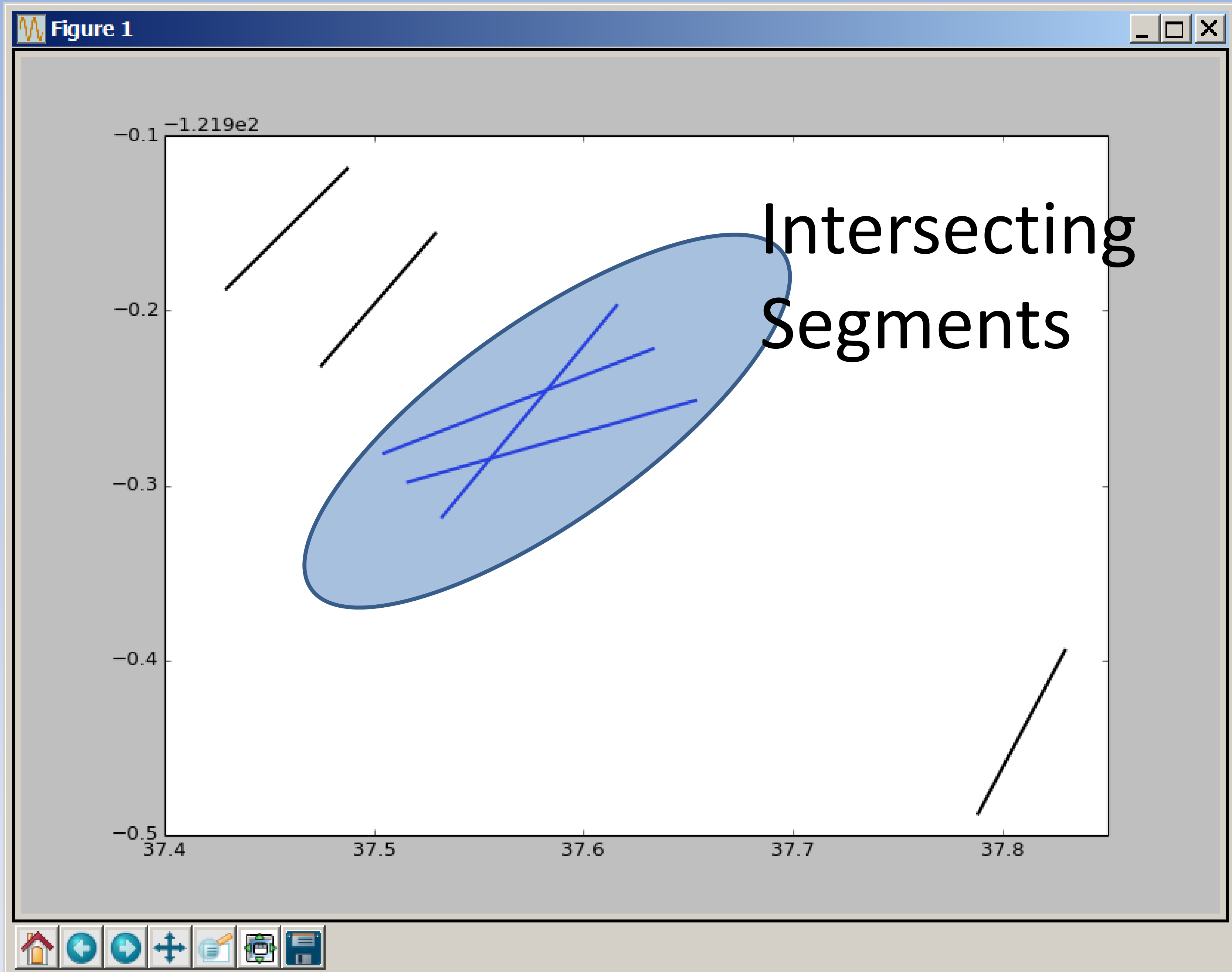
INPUT SAMPLE:

Your program should accept as its first argument a path to a filename. Input example is the following

```
1: ([37.788353, -122.387695], [37.829853, -122.294312])
2: ([37.429615, -122.087631], [37.487391, -122.018967])
3: ([37.474858, -122.131577], [37.529332, -122.056046])
4: ([37.532599,-122.218094], [37.615863,-122.097244])
5: ([37.516262,-122.198181], [37.653383,-122.151489])
6: ([37.504824,-122.181702], [37.633266,-122.121964])
```

Each input line represents a pair of coordinates for each possible bridge.

Intersecting Segments

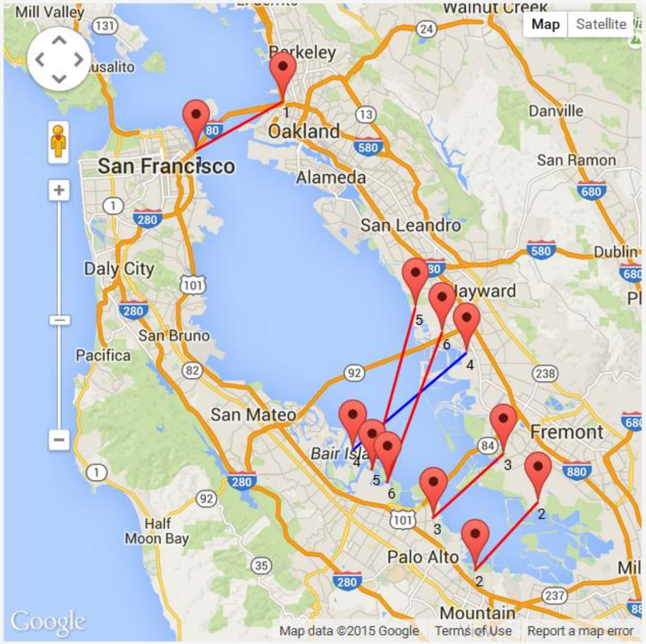


Ouput for Codeeval

BAY BRIDGES

SPONSORING COMPANIES:

livefyre AdRoll Chegg STORM8



OUTPUT SAMPLE:

You should output bridges in ascending order.

```
1
2
3
5
6
```

(Check lines on the map)

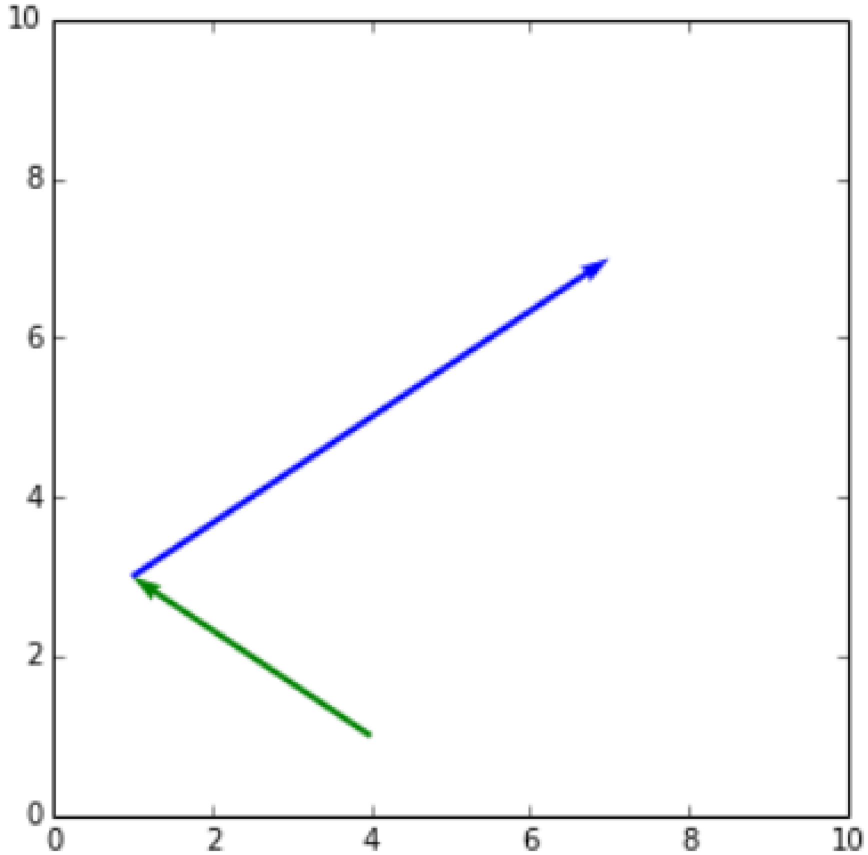
INPUT SAMPLE:

Your program should accept as its first argument a path to a filename. Input example is the following

```
1: ([37.788353, -122.387695], [37.829853, -122.294312])
2: ([37.429615, -122.087631], [37.487391, -122.018967])
3: ([37.474858, -122.131577], [37.529332, -122.056046])
4: ([37.532599,-122.218094], [37.615863,-122.097244])
5: ([37.516262,-122.198181], [37.653383,-122.151489])
6: ([37.504824,-122.181702], [37.633266,-122.121964])
```

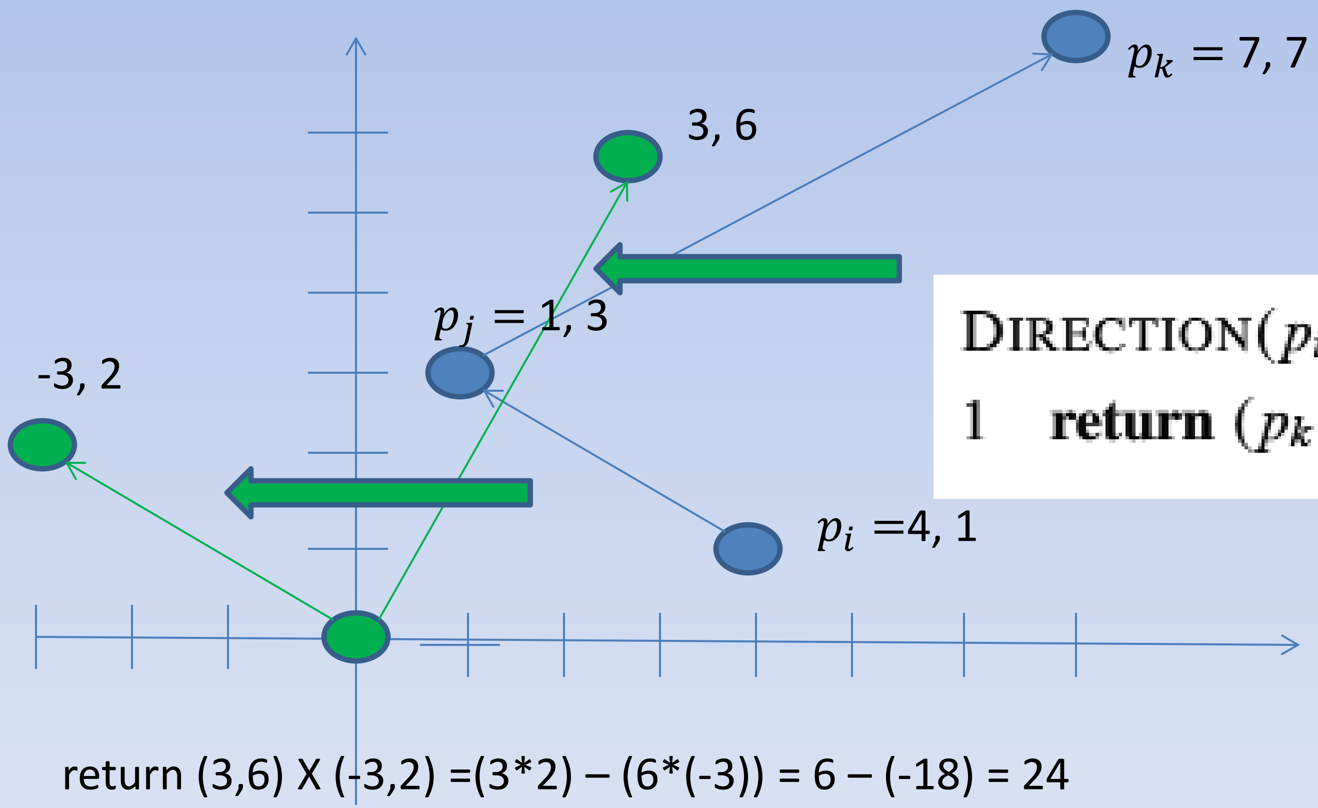
Each input line represents a pair of coordinates for each possible bridge.

```
p0 = (4, 1)
p1 = (1, 3)
p2 = (7, 7)
PlotVectorP0P1P2(p0, p1, p2)
print "Direction(p0, p1, p2) ", Direction(p0, p1, p2)
PrintDirection(p0, p1, p2)
```




```
Direction(p0, p1, p2)  24
Clockwise: Right Turn
```

Sign of the Crossproduct Calculated from Origin



DIRECTION(p_i, p_j, p_k)

1 **return** $(p_k - p_i) \times (p_j - p_i)$

return (3,6) X (-3,2) =(3*2) – (6*(-3)) = 6 – (-18) = 24

p_k is CLOCKWISE from p_j

SEGMENTS-INTERSECT(p_1, p_2, p_3, p_4)

ON-SEGMENT(p_i, p_j, p_k)

if $\min(x_i, x_j) \le x_k \le \max(x_i, x_j)$ **and** $\min(y_i, y_j) \le y_k \le \max(y_i, y_j)$
    return TRUE
else return FALSE

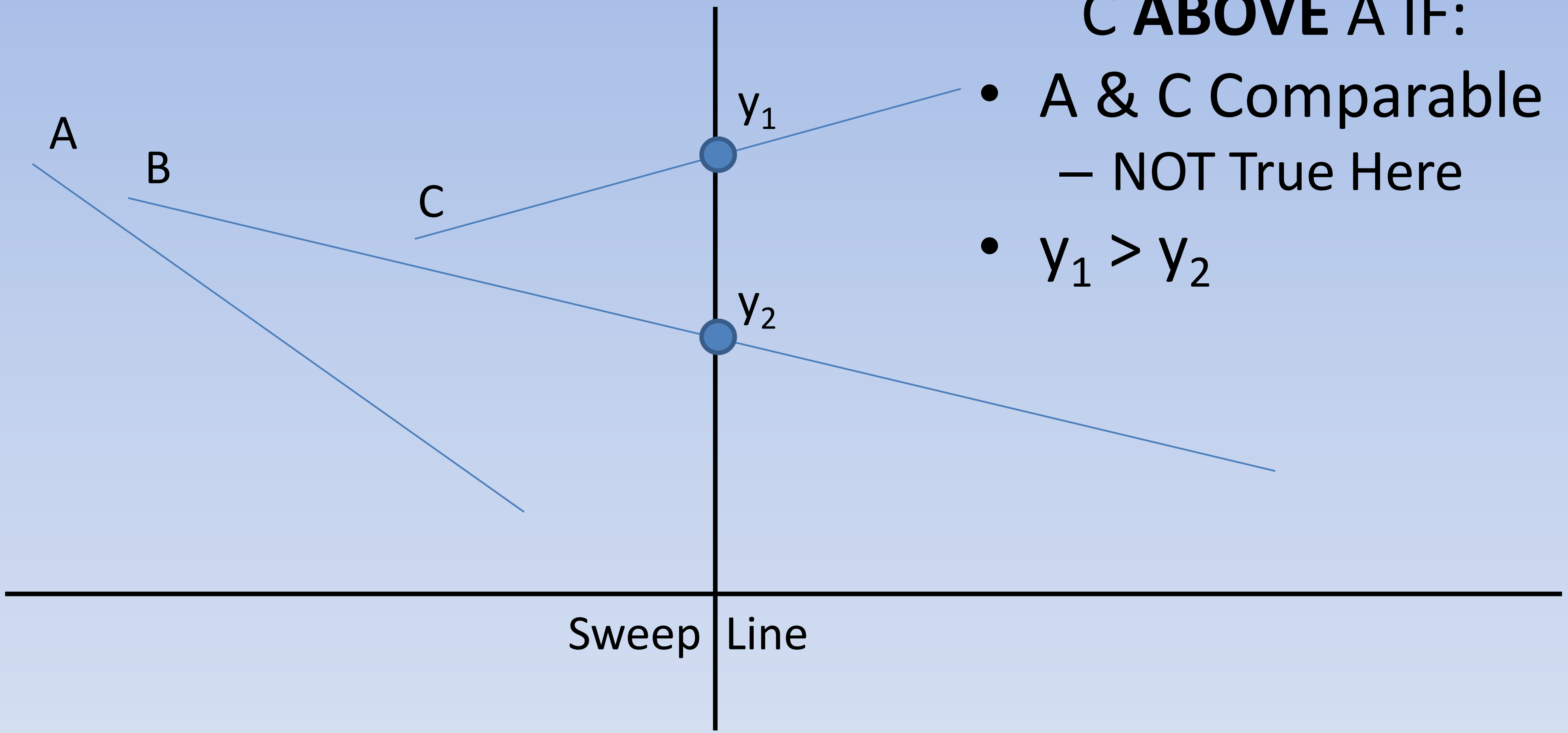
SEGMENTS-I

$d_1 =$ DIRECTION(p_3, p_4, p_1)
$d_2 =$ DIRECTION(p_3, p_4, p_2)
$d_3 =$ DIRECTION(p_1, p_2, p_3)
$d_4 =$ DIRECTION(p_1, p_2, p_4)
if $((d_1 > 0$ **and** $d_2 < 0)$ **or** $(d_1 < 0$ **and** $d_2 > 0))$ **and**
    $((d_3 > 0$ **and** $d_4 < 0)$ **or** $(d_3 < 0$ **and** $d_4 > 0))$
    return TRUE
elseif $d_1 == 0$ **and** ON-SEGMENT(p_3, p_4, p_1)
    return TRUE
elseif $d_2 == 0$ **and** ON-SEGMENT(p_3, p_4, p_2)
    return TRUE
elseif $d_3 == 0$ **and** ON-SEGMENT(p_1, p_2, p_3)
    return TRUE
elseif $d_4 == 0$ **and** ON-SEGMENT(p_1, p_2, p_4)
    return TRUE
else return FALSE

Query Any pair of segments Intersects w/ **Sweeping**

- Determines If ANY Intersecting line segments in O(NlgN) time.
  - N is the number of segments.
- **Sweeping:**
  - Imaginary vertical **Sweep Line** passes through objects.
  - Sweep dimension treated as time dimension
- Sweeping allows ordering the geometric objects.

Ordering Segments



C **ABOVE** A IF:

- A & C Comparable
  - NOT True Here
- $y_1 > y_2$

- No Vertical Lines
- Line Segments **Comparable** if both intersect Sweep Line.

Sweep Line

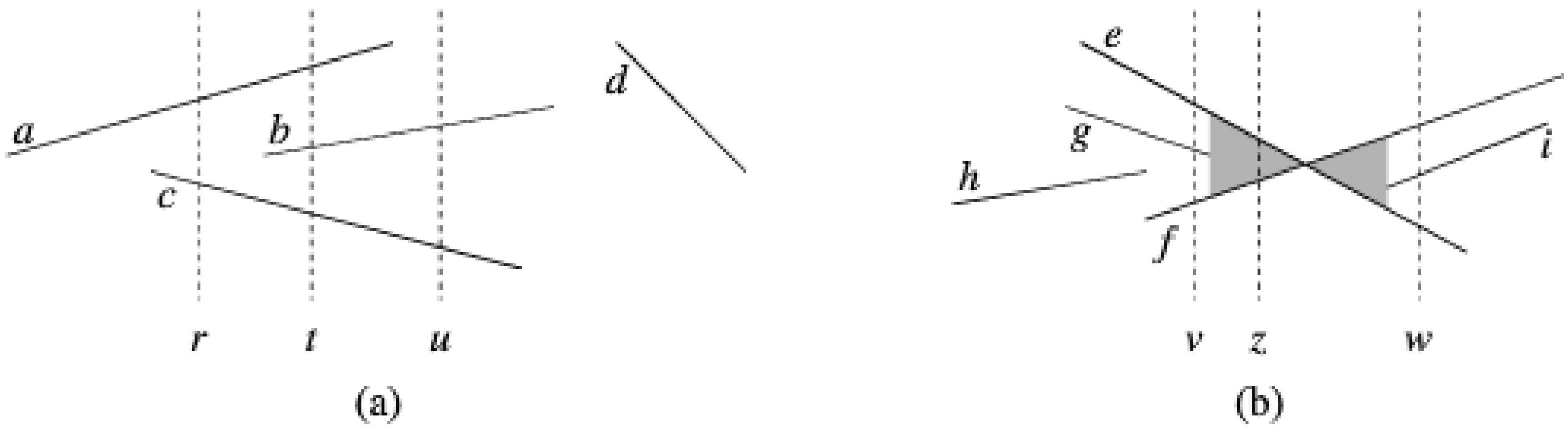


Figure 33.4 The ordering among line segments at various vertical sweep lines. **(a)** We have $a \succeq_r c$, $a \succeq_t b$, $b \succeq_t c$, $a \succeq_t c$, and $b \succeq_u c$. Segment d is comparable with no other segment shown. **(b)** When segments e and f intersect, they reverse their orders: we have $e \succeq_v f$ but $f \succeq_w e$. Any sweep line (such as z) that passes through the shaded region has e and f consecutive in the ordering given by the relation $\succeq_z$.

Moving a Sweep Line

- Manage Two Sets of Data:
  - **Sweep-Line Status**: Ordering induced among objects intersecting sweep line.
  - **Event-Point Schedule**: This sequence of **event-points** are ordered left to right according to the x-coordinates, and mark the points where sweeping halts and processing takes place.

Maintaining Sweep-Line Status

- Sweep-Line Status Data Structure maintains a complete preorder of a set of line segments.
- Sweep-Line Status Data Structure Operations
  - Insert(T, s): inserts segment **s** into **T** .
  - Delete(T, s): delete segment **s** from **T** .
  - Above(T, s): returns the segment immediately above segment **s** in **T** .
  - Below(T, s): return the segment immediately below segment **s** in **T** .
- A balanced binary tree (avl, red-black) can implements ops in O(lnN).

```
ANY-SEGMENTS-INTERSECT(S)
T = Ø
sort the endpoints of the segments in S from left to right,
        breaking ties by putting left endpoints before right endpoints
        and breaking further ties by putting points with lower
        y-coordinates first
for each point p in the sorted list of endpoints
    if p is the left endpoint of a segment s
        INSERT(T, s)
        if (ABOVE(T, s) exists and intersects s)
            or (BELOW(T, s) exists and intersects s)
            return TRUE
    if p is the right endpoint of a segment s
        if both ABOVE(T, s) and BELOW(T, s) exist
            and ABOVE(T, s) intersects BELOW(T, s)
            return TRUE
        DELETE(T, s)
return FALSE
```

Sort on X-Coords

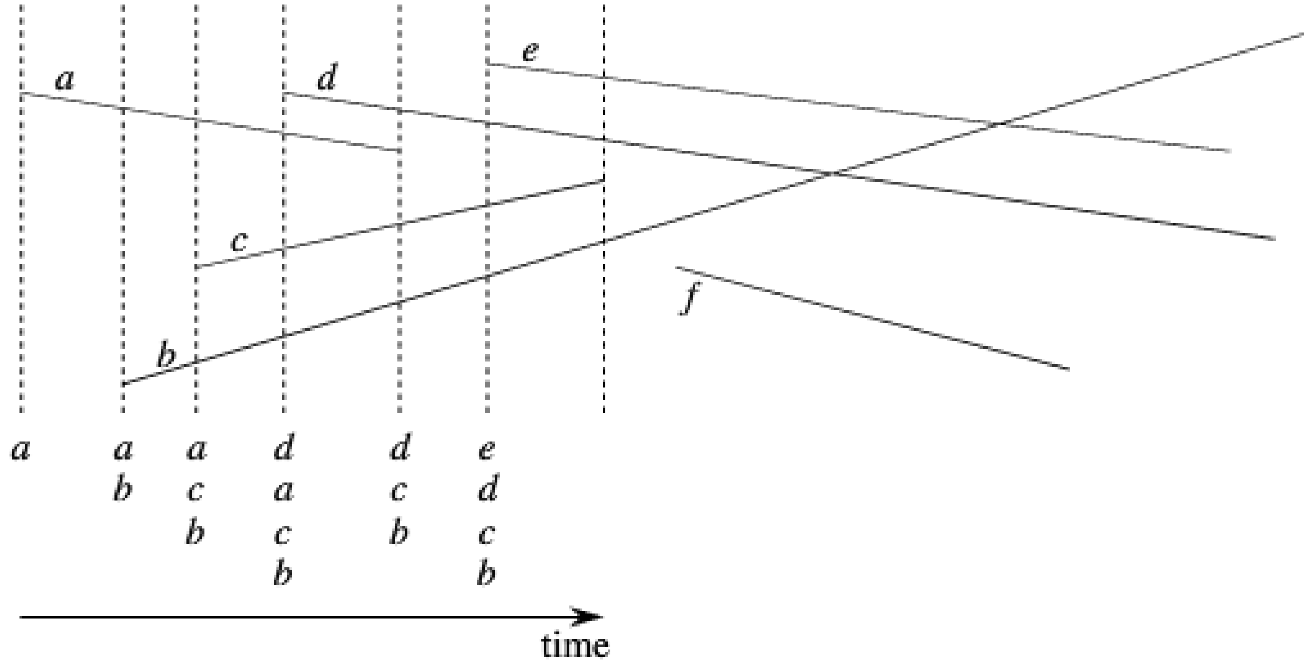
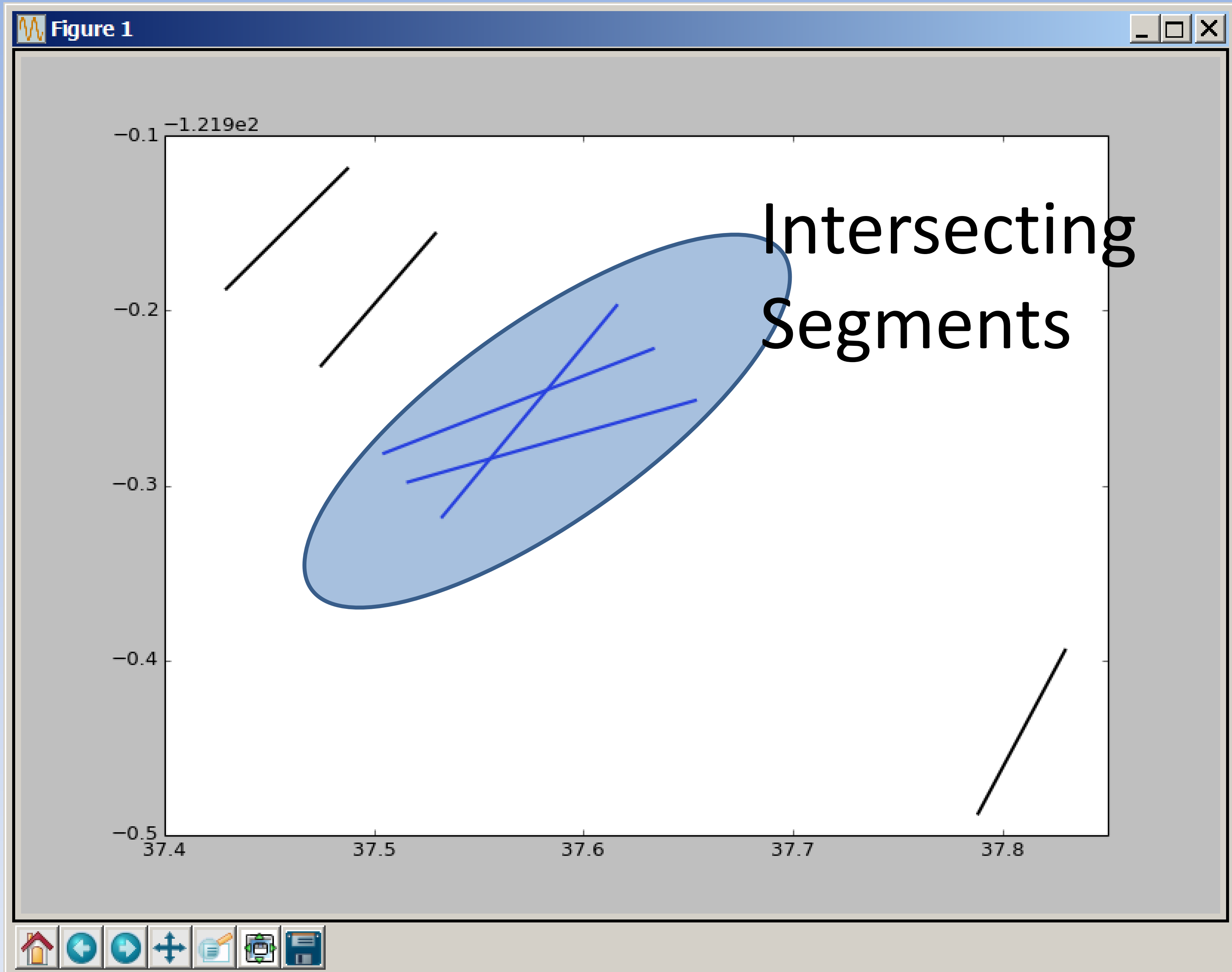


Figure 33.5 The execution of ANY-SEGMENTS-INTERSECT. Each dashed line is the sweep line at an event point. Except for the rightmost sweep line, the ordering of segment names below each sweep line corresponds to the total preorder T at the end of the **for** loop processing the corresponding event point. The rightmost sweep line occurs when processing the right endpoint of segment c; because segments d and b surround c and intersect each other, the procedure returns TRUE.

Intersecting Segments



Proving Correctness (1)

- Need to prove we don't miss any intersections and mistakenly return False!
  - Assume missed intersection is point p of segments a and b
  - Assume all was well up to intersection point p.
  - a and b become consecutive at some event-point z.
    - z is to the left or goes through p.

Proving Correctness: Key Text

We also need to show the converse: that if there is an intersection, then ANY-SEGMENTS-INTERSECT returns TRUE. Let us suppose that there is at least one intersection. Let p be the leftmost intersection point, breaking ties by choosing the point with the lowest y-coordinate, and let a and b be the segments that intersect at p. Since no intersections occur to the left of p, the order given by T is correct at all points to the left of p. Because no three segments intersect at the same point, a and b become consecutive in the total preorder at some sweep line z.[2] Moreover, z is to the left of p or goes through p. Some segment endpoint q on sweep line z

---

[2]If we allow three segments to intersect at the same point, there may be an intervening segment c that intersects both a and b at point p. That is, we may have $a \succcurlyeq_w c$ and $c \succcurlyeq_w b$ for all sweep lines w to the left of p for which $a \succcurlyeq_w b$. Exercise 33.2-8 asks you to show that ANY-SEGMENTS-INTERSECT is correct even if three segments do intersect at the same point.

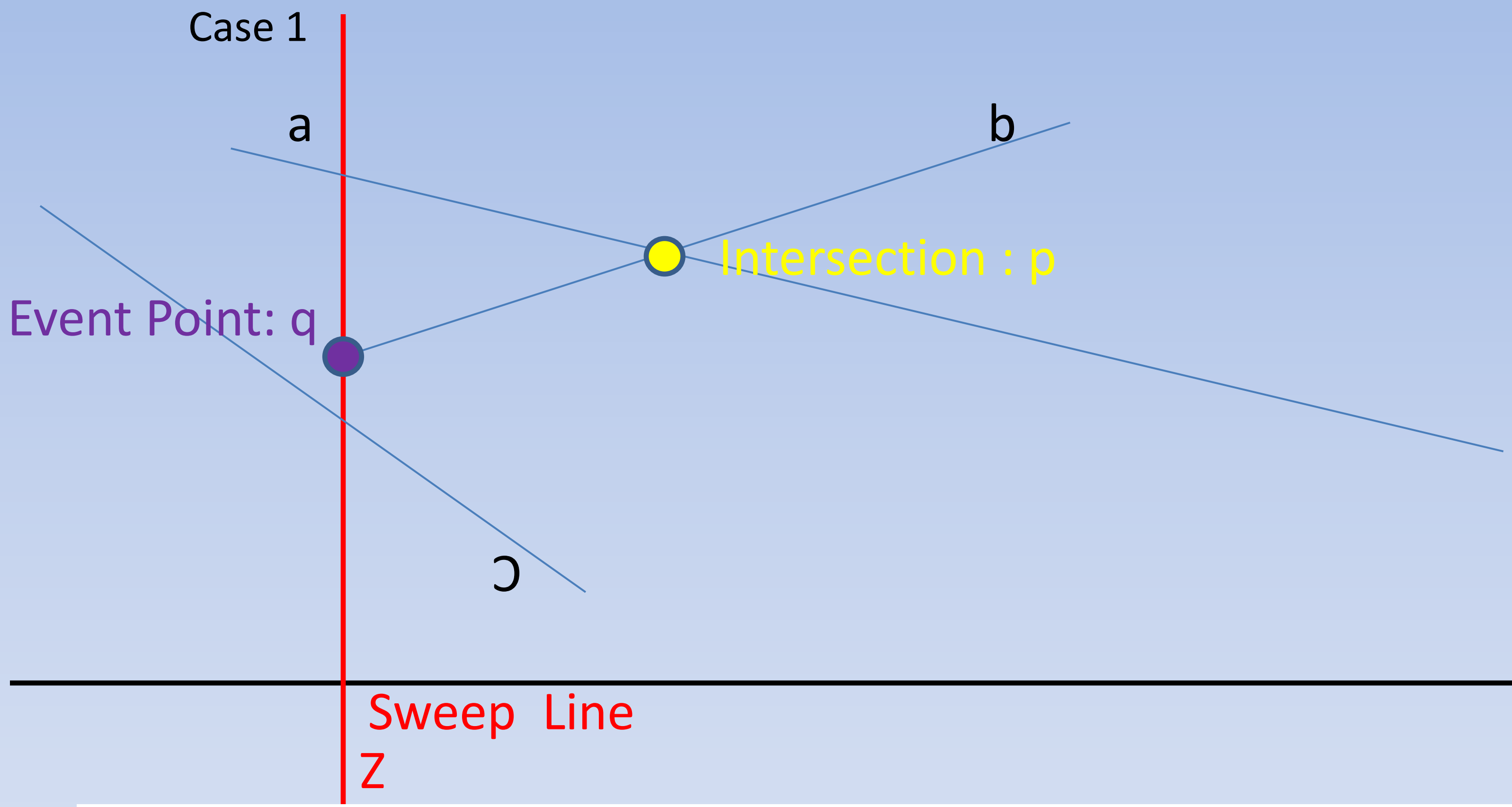
is the event point at which a and b become consecutive in the total preorder. If p is on sweep line z, then $q = p$. If p is not on sweep line z, then q is to the left of p. In either case, the order given by T is correct just before encountering q. (Here is where we use the lexicographic order in which the algorithm processes event points. Because p is the lowest of the leftmost intersection points, even if p is on sweep line z and some other intersection point p' is on z, event point $q = p$ is processed before the other intersection p' can interfere with the total preorder T. Moreover, even if p is the left endpoint of one segment, say a, and the right endpoint of the other segment, say b, because left endpoint events occur before right endpoint events, segment b is in T upon first encountering segment a.) Either event point q is processed by ANY-SEGMENTS-INTERSECT or it is not processed.

If q is processed by ANY-SEGMENTS-INTERSECT, only two possible actions may occur:

1. Either a or b is inserted into T, and the other segment is above or below it in the total preorder. Lines 4–7 detect this case.

2. Segments a and b are already in T, and a segment between them in the total preorder is deleted, making a and b become consecutive. Lines 8–11 detect this case.

Correctness: Case 1

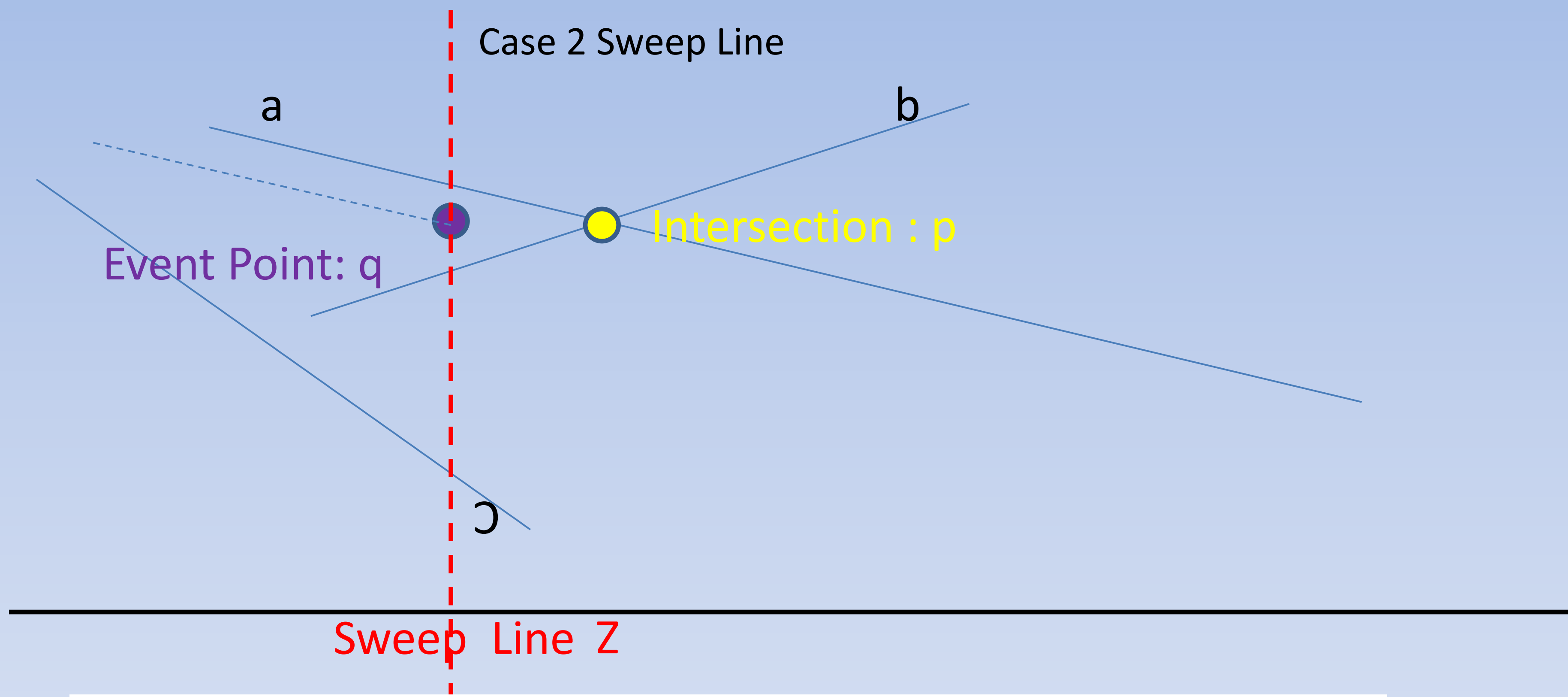


1. Either a or b is inserted into T, and the other segment is above or below it in the total preorder. Lines 4–7 detect this case.
2. Segments a and b are already in T, and a segment between them in the total preorder is deleted, making a and b become consecutive. Lines 8–11 detect this case.

Proving Correctness (2)

- Sweep line point z must be the end-point of some segment q.
- If the intersection point p is on the sweep-line, then p=q.
- T order is correct up to point q.
- Using lexicographic order, p is the lowest of the leftmost intersection points.
  - Therefore either q is processed and intersection found.
  - Or q is not processed because an intersection has already been found.

Correctness: Case 2



1. Either a or b is inserted into T, and the other segment is above or below it in the total preorder. Lines 4–7 detect this case.
2. Segments a and b are already in T, and a segment between them in the total preorder is deleted, making a and b become consecutive. Lines 8–11 detect this case.

ANY-SEGMENTS-INTERSECT(S)

```
T = Ø
sort the endpoints of the segments in S from left to right,
        breaking ties by putting left endpoints before right endpoints
        and breaking further ties by putting points with lower
        y-coordinates first
for each point p in the sorted list of endpoints
    if p is the left endpoint of a segment s
        INSERT(T, s)
        if (ABOVE(T, s) exists and intersects s)
                or (BELOW(T, s) exists and intersects s)
            return TRUE
    if p is the right endpoint of a segment s
        if both ABOVE(T, s) and BELOW(T, s) exist
                and ABOVE(T, s) intersects BELOW(T, s)
            return TRUE
        DELETE(T, s)
return FALSE
```

Case 2:
Above w/ Below

Running Time

- Given n segments running time = O(n Lg n)
  - Line 2 sort takes O(n Lg n)
- For loop iterates 2n times (each end point)
- Each iteration takes = lg n
- Intersection tests takes O(1)

Convex Hull

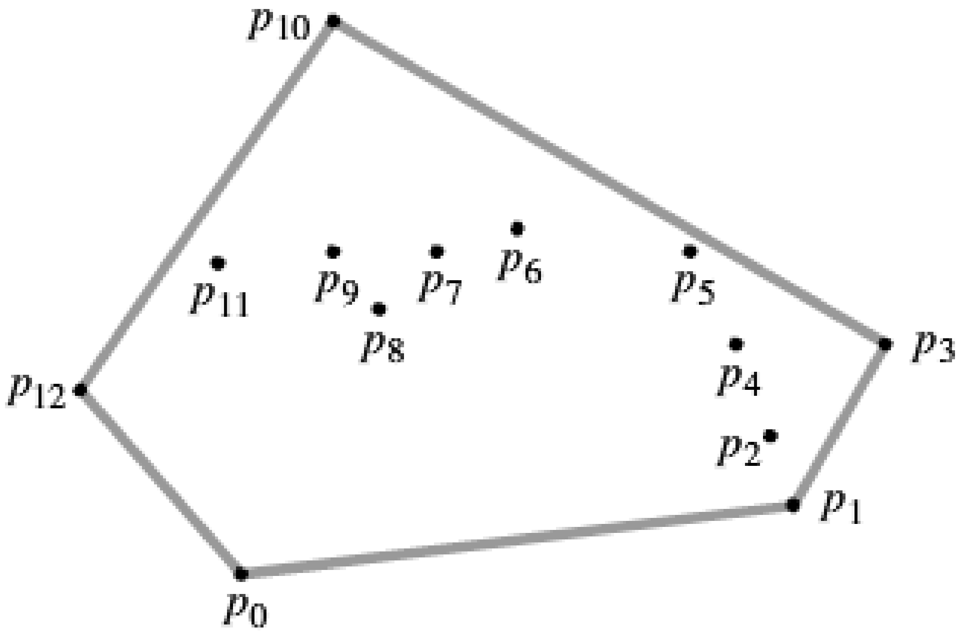


Figure 33.6 A set of points $Q = \{p_0, p_1, \ldots, p_{12}\}$ with its convex hull $\mathrm{CH}(Q)$ in gray.

Graham's Scan

- Maintains a stack of candidate points.
- When program completes the stack contains exactly the points of the convex hull.

Graham-Scan

- Input: Q a set of points (>3)
- Functions:
  - Top(S): returns top of stack S without changing S.
  - Next-To-To(S): returns the point one entry below the top of stack S without changing S.
  - Push(p, S)/Pop(S)

Graham-Scan

GRAHAM-SCAN(Q)

let p_0 be the point in Q with the minimum y-coordinate,
    or the leftmost such point in case of a tie
let $\langle p_1, p_2, \ldots, p_m \rangle$ be the remaining points in Q,
    sorted by polar angle in counterclockwise order around p_0
    (if more than one point has the same angle, remove all but
    the one that is farthest from p_0)
if $m < 2$
    return "convex hull is empty"
else let S be an empty stack
    PUSH(p_0, S)
    PUSH(p_1, S)
    PUSH(p_2, S)
    for $i = 3$ **to** m
        while the angle formed by points NEXT-TO-TOP(S), TOP(S),
                and p_i makes a nonleft turn
            POP(S)
        PUSH(p_i, S)
    return S

Graham-Scan

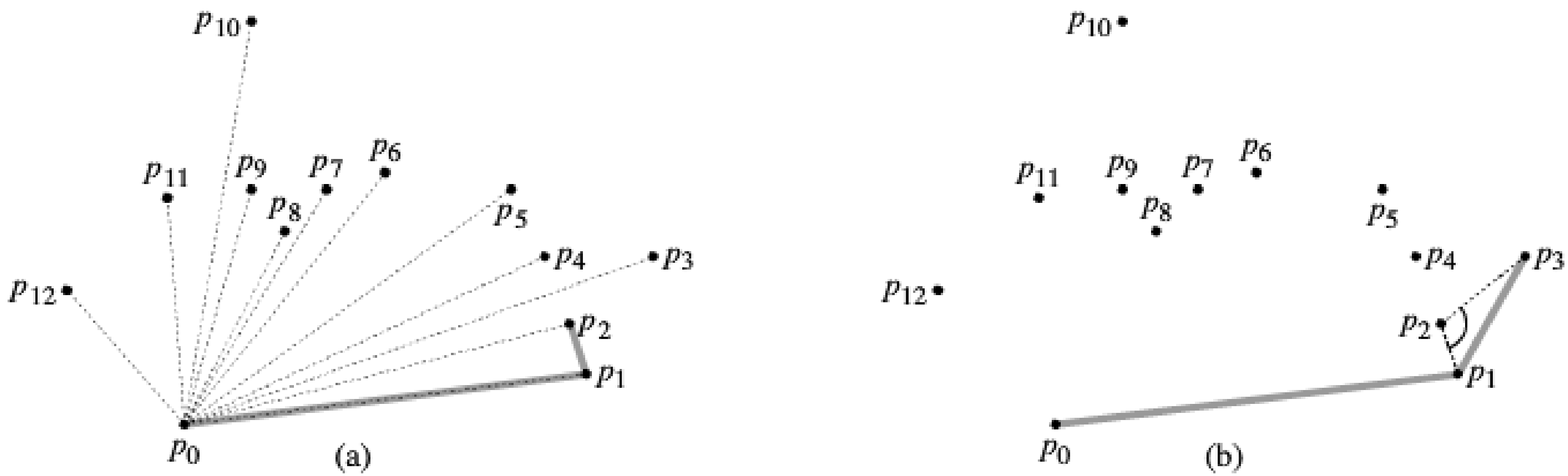


Figure 33.7 The execution of GRAHAM-SCAN on the set Q of Figure 33.6. The current convex hull contained in stack S is shown in gray at each step. **(a)** The sequence $\langle p_1, p_2, \ldots, p_{12} \rangle$ of points numbered in order of increasing polar angle relative to p_0, and the initial stack S containing p_0, p_1, and p_2. **(b)–(k)** Stack S after each iteration of the **for** loop of lines 9–12. Dashed lines show nonleft turns, which cause points to be popped from the stack. In part (h), for example, the right turn at angle $\angle p_7 p_8 p_9$ causes p_8 to be popped, and then the right turn at angle $\angle p_6 p_7 p_9$ causes p_7 to be popped.

Graham-Scan

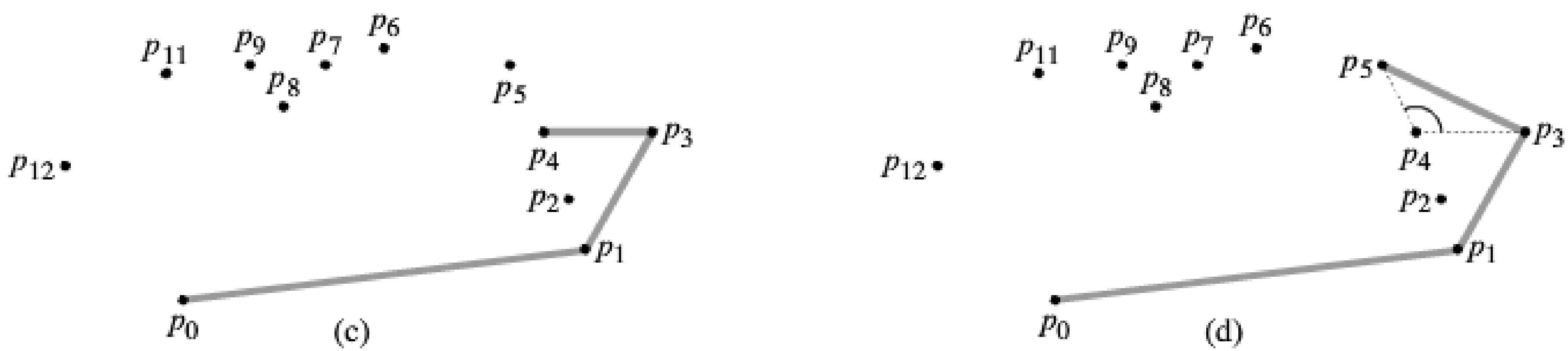


Figure 33.7 The execution of GRAHAM-SCAN on the set Q of Figure 33.6. The current convex hull contained in stack S is shown in gray at each step. **(a)** The sequence $\langle p_1, p_2, \ldots, p_{12} \rangle$ of points numbered in order of increasing polar angle relative to p_0, and the initial stack S containing p_0, p_1, and p_2. **(b)–(k)** Stack S after each iteration of the **for** loop of lines 9–12. Dashed lines show nonleft turns, which cause points to be popped from the stack. In part (h), for example, the right turn at angle $\angle p_7 p_8 p_9$ causes p_8 to be popped, and then the right turn at angle $\angle p_6 p_7 p_9$ causes p_7 to be popped.

Graham-Scan

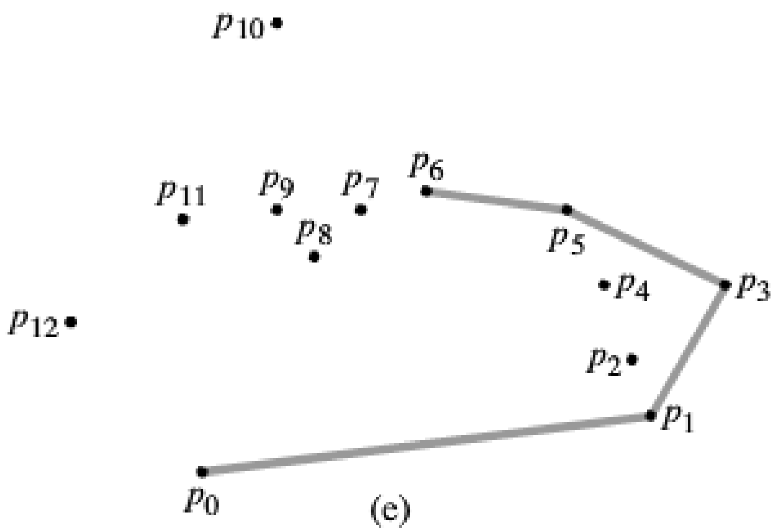
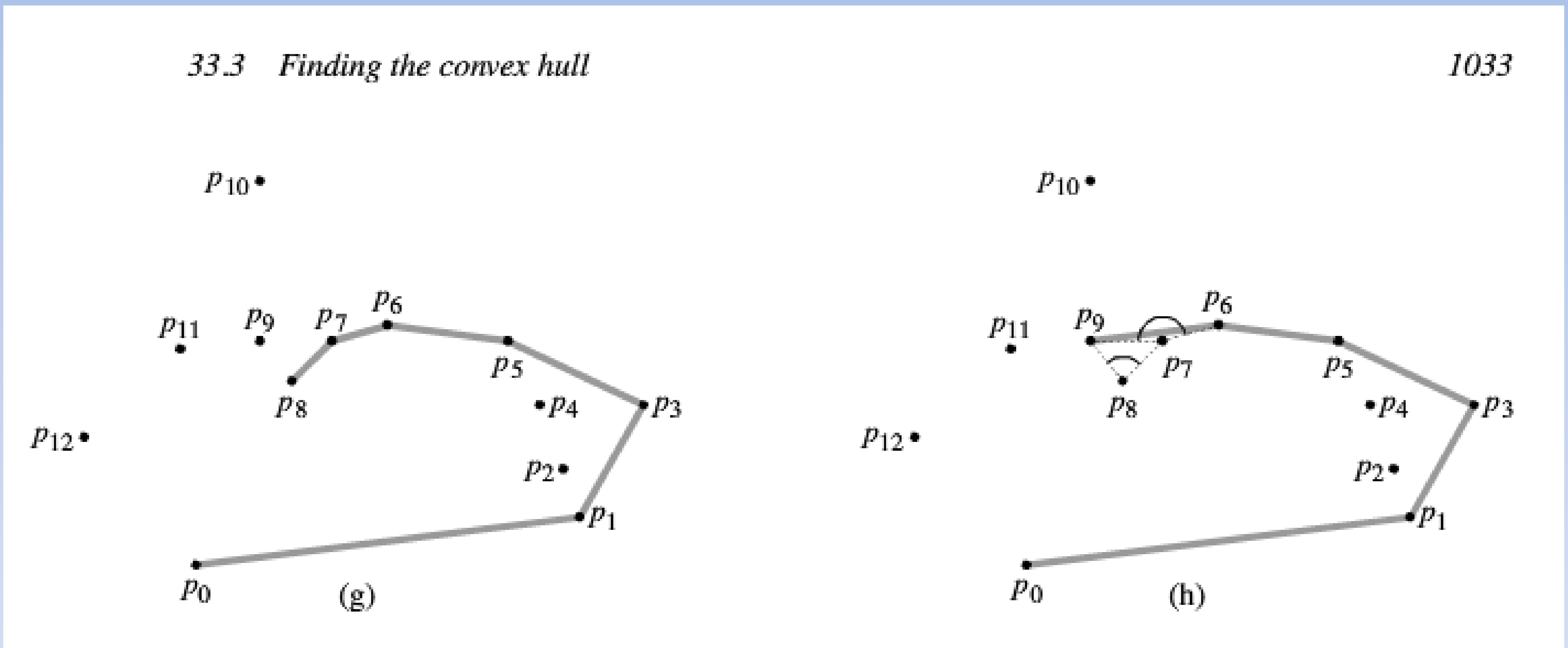
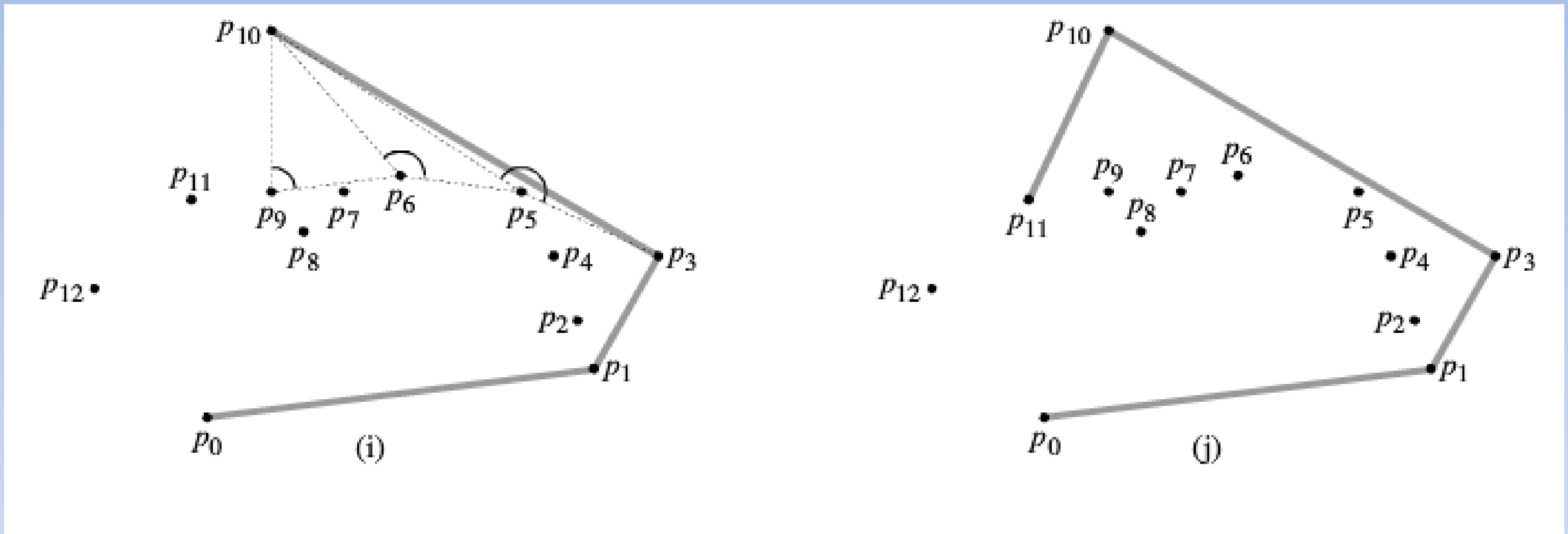


Figure 33.7 The execution of GRAHAM-SCAN on the set Q of Figure 33.6. The current convex hull contained in stack S is shown in gray at each step. **(a)** The sequence $\langle p_1, p_2, \ldots, p_{12} \rangle$ of points numbered in order of increasing polar angle relative to p_0, and the initial stack S containing p_0, p_1, and p_2. **(b)–(k)** Stack S after each iteration of the **for** loop of lines 9–12. Dashed lines show nonleft turns, which cause points to be popped from the stack. In part (h), for example, the right turn at angle $\angle p_7 p_8 p_9$ causes p_8 to be popped, and then the right turn at angle $\angle p_6 p_7 p_9$ causes p_7 to be popped.

Graham-Scan

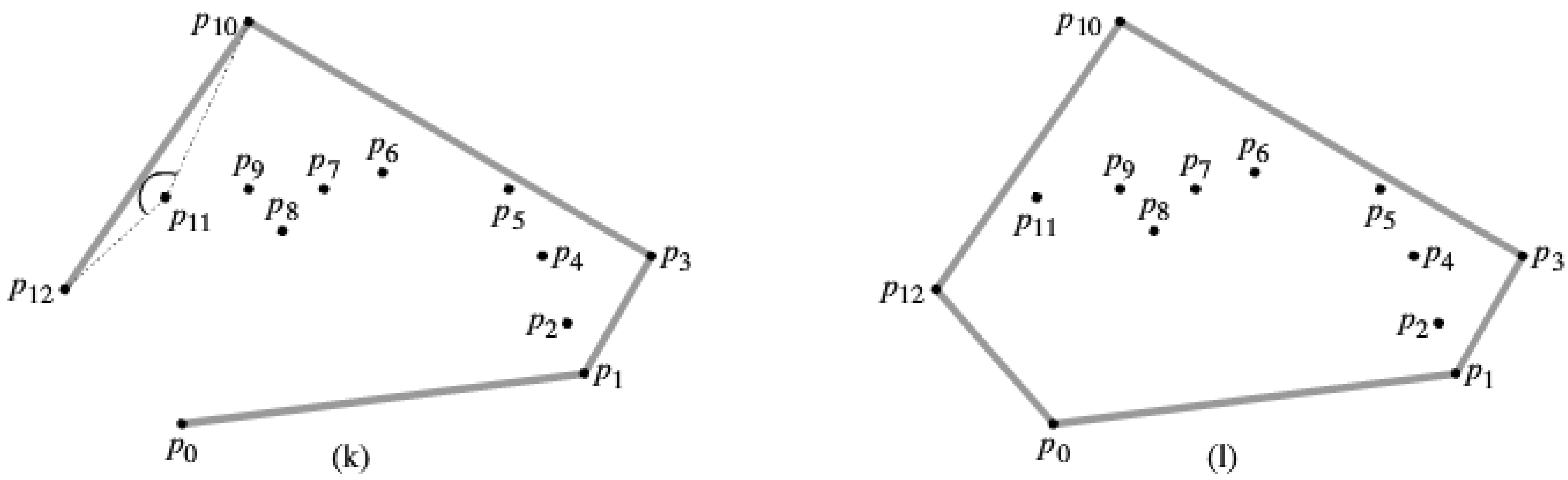

33.3 Finding the convex hull
1033
p10
p11
p9
p7
p6
p5
p8
p4
p3
p12
p2
p1
p0
(g)
p10
p11
p9
p6
p7
p5
p8
p4
p3
p12
p2
p1
p0
(h)

Graham-Scan



p_{10}
p_{11}
p_9
p_7
p_6
p_5
p_8
p_4
p_3
p_{12}
p_2
p_1
p_0
(i)
p_{10}
p_9
p_7
p_6
p_8
p_{11}
p_5
p_4
p_3
p_{12}
p_2
p_1
p_0
(j)

Graham-Scan



p_{10}
p_9
p_7
p_6
p_{11}
p_8
p_5
p_4
p_3
p_{12}
p_2
p_1
p_0
(k)
p_{10}
p_9
p_7
p_6
p_{11}
p_8
p_5
p_4
p_3
p_{12}
p_2
p_1
p_0
(l)

Graham Scan w/ Python

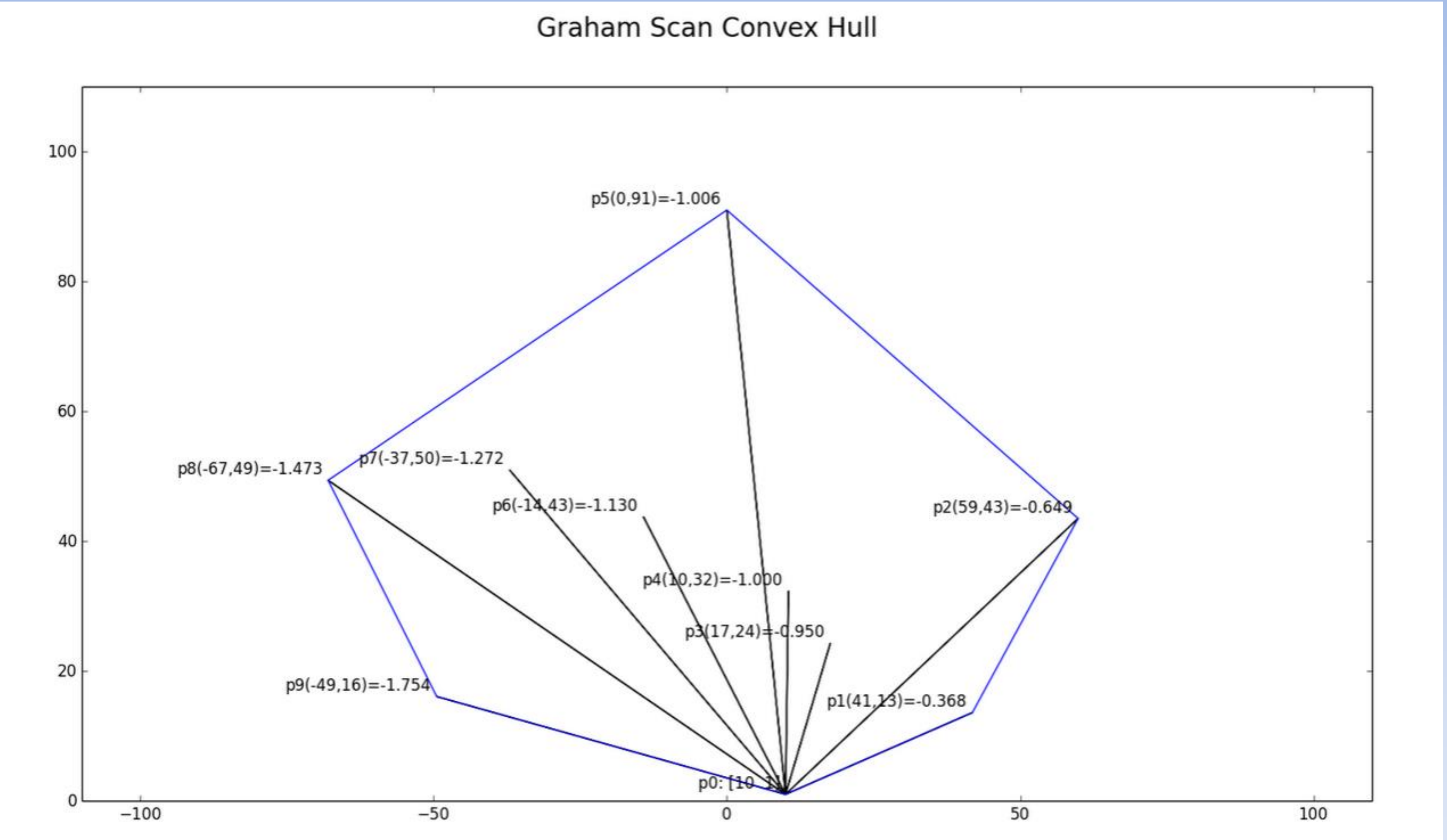
97 lines (83 sloc) | 2.8 kB

Raw | Blame | History

```python
import math
import numpy as np
import matplotlib.pyplot as plt
import random

def CrossProduct(p1, p2):
    (x1, y1) = p1
    (x2, y2) = p2
    return x1*y2 - x2*y1

def Direction(p0, p1, p2):
#    x = np.cross(p2-p0, p1-p0)
    y = CrossProduct( (p2[0]-p0[0], p2[1]-p0[1]), (p1[0]-p0[0], p1[1]-p0[1]) )
    return y

def CalculateRelativePolarAngle(p0, p1):
    p0p1N = p1-p0 # Create vector p0->p1

    # calculate cross-product with x-axis vector (1,0)
    p0p1 = CrossProduct(p0p1N, (1.0,0.0))/math.sqrt(p0p1N[0]**2+p0p1N[1]**2)
    if Direction(p0, (p0[0],p1[1]), p1) < 0 :
        #add 90 degrees when crossing y axis
        p0p1 = -1 - (p0p1 + 1)
    return p0p1
```

Graham Scan w/ Python (Assumes Points Sorted)

```python
def GrahamScan(points):
    p0 = points[0]
    p1 = points[1]
    p2 = points[2]
    plist = points[3:]
    ch = [p0, p1, p2]
    for p2 in plist:
        p0 = ch[-2]
        p1 = ch[-1]
        print "1: ", p0, p1, p2, Direction(p0,p1,p2)
        while Direction(p0, p1, p2) > 0:
            ch.pop()
            p0 = ch[-2]
            p1 = ch[-1]
        ch.append(p2)
    return ch
```


Graham Scan Convex Hull
p5(0,91)=-1.006
p8(-67,49)=-1.473
p7(-37,50)=-1.272
p6(-14,43)=-1.130
p2(59,43)=-0.649
p4(10,32)=-1.000
p3(17,24)=-0.950
p9(-49,16)=-1.754
p1(41,13)=-0.368
p0: [10
100
80
60
40
20
0
−100
−50
0
50
100

Algorithms Notes

everestso / **ics1293**

Unwatch 2

Information & Computer Science -- Notes — Edit

206 commits | **1** branch | **0** releases | **1** contributor

Branch: **master** ics1293 / +

Big Data

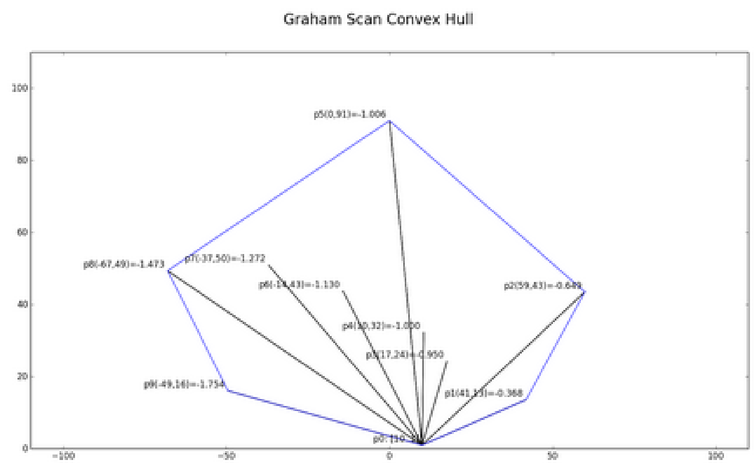
everestso authored 11 days ago latest commit 2566546cf4

Code	reviews	14 days ago
Img	code illustration link	2 months ago
Slides	satisfiable?	3 months ago
Topics	Big Data	11 days ago
LICENSE	clean-up	3 months ago
README.md	clean-up	3 months ago

- Euler's Phi Function: φ(n)
- Chinese Remainder Theorem
- Euclid's Algorithm for GCD
- Additional References:
    - Avi Kak (Academic Descendant of : Oswald Veblen & E.H. Moore)

33: Computational Geometry

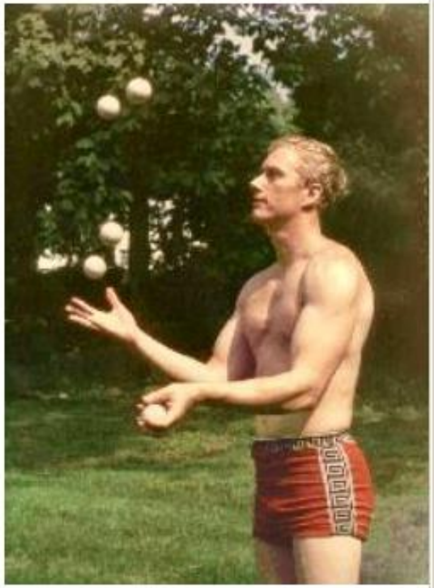
- Graham's scan
    - Ron Graham: http://www.math.ucsd.edu/~fan/ron/
    - Code Illustration:



34: NP-Completeness

- Problems verifiable in O(n**k)

About Ron Graham



- **The Mathemagician**
  Several mathematical areas were started by Ron's work, such as worst case analysis in scheduling theory, on-line algorithms and amortized analysis in the Graham's scan in Computational Geometry, and of course, his favorite topics on Ramsey Theory, and the recent work on quasi-randomness. Ron's mathematics was highlighted in the nomination article written by Gian-Carlo Rota for the first contested election of AMS President. (He won). He received the Steele Prize for Lifetime Achievement in 2003. The book "*Magical Mathematics*", coauthored with Persi Diaconis, was published October 2011 and here are reviews in NY Times and WSJ.
- **Chief Scientist** at California Institute for Telecommunication and Information Technology, Cal-IT^2, of UC San Diego.
  Irwin and Joan Jacobs Professor at Department of Computer Science and Engineering of UCSD.
- **Internet Visionary**
  "Here is a picture that Ron and Tom talked about putting routers all over the globe---way before Akamai was built." said David Johnson at Ron's party.
- **Ex President of the International Jugglers Association**
  Ron was involved in creating Mill's mess and numerous new juggling tricks for site swaps (see "Drops and descents" with Joe Buhler and several other math papers). Ron has many juggling students including Steve Mills. There is a 20-minute video of Ron teaching the connection between juggling and mathematics to junior high school students. Also, there is a link to a video on "The secret to juggling". Here is another video on teaching juggling in a segment of Live from Bell Labs.

- **Ex Chief-Tech-Honcho at AT&T**
  "When Ron first went to Bell Labs, some friends said that it could be the end of his research. Well, he made this place the center of focus in research. He held a series of titles, some were first or one of a kind, 'Adjunct Director', 'Assistant Vice President', to 'Chief Scientist' 'Emeritus Chief Scientist'. He had a ball at the labs. Indeed, there was a great party when he finally left there in 1999."
- **Guinness Book of World Records**
  "The highest number ever used in a mathematical proof is a bounding value published in 1977 and known as Graham's number." The biggest number in the universe. Here is a nice video about Graham's number by Catalist and also in Wait But Why.. More.
- **The monster proof**
  "This is perhaps the most complicated set of recurrences that will ever be solved", said D. E. Knuth.