ACM Training
Computational Geometry

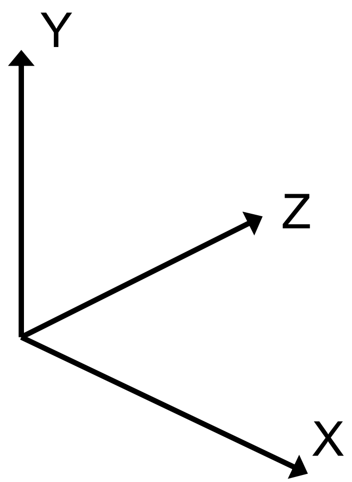
Basics

- Coordinates & Vector
- Revision on Line Operations
- Calculating Distances
- Finding Point Position
- Finding Vector Direction
- Line Intersection
- Area of Triangle
- Exercises

Coordinates & Vector (1)

- 2D/3D Point Coordinates
  - e.g.

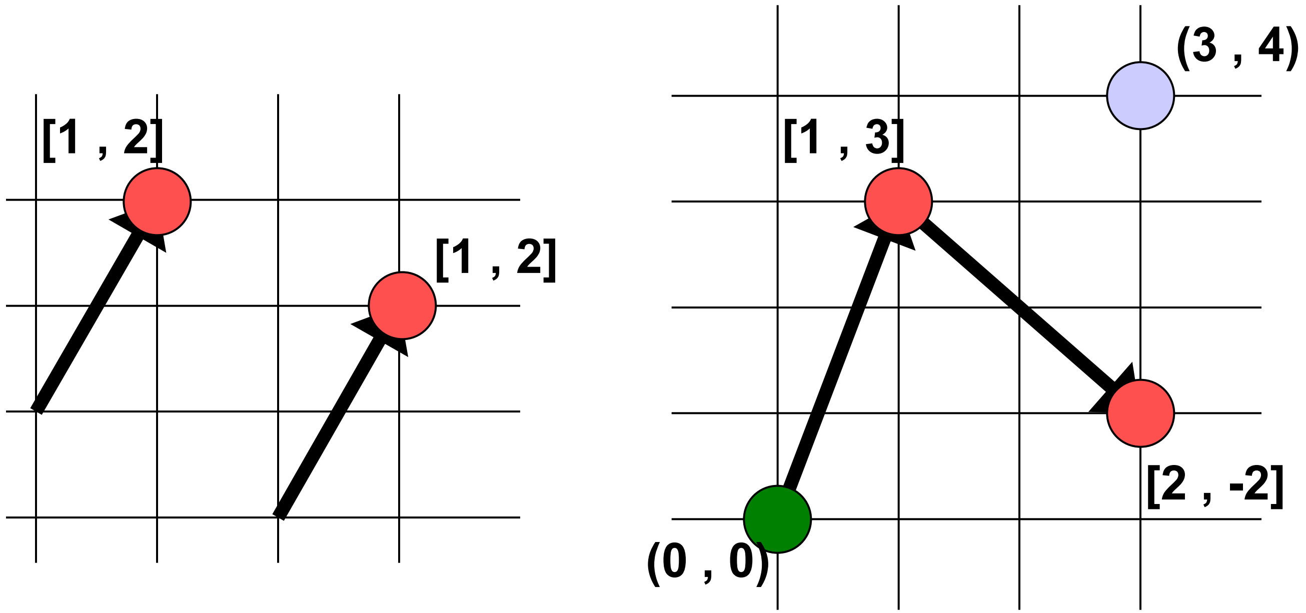
```
float Point3D_A[3];
Point3D_A[0] =  12.3f;
Point3D_A[1] =    .23f;
Point3D_A[2] = -20.00f;

struct {
  float x, y, z;
} Point3D_B;
Point3D_B.x  =  32.f;
Point3D_B.y  =  -1.2e4;
Point3D_B.z  =   0.23f;
```



  - 2 Structures are Logically Equivalent
  - Sharing Matrix Operations

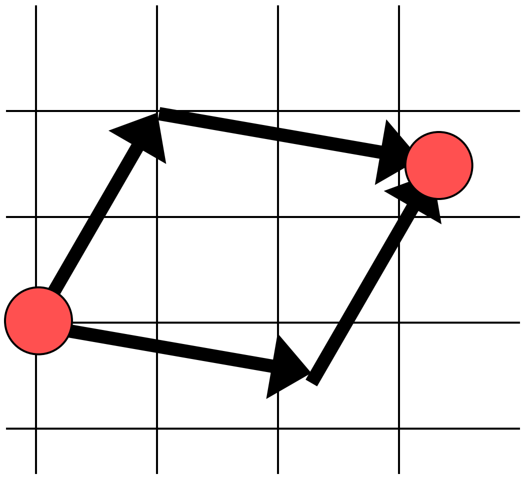
Coordinates & Vector (2)

- Vector
  - Same data structure as the point
  - Relative to the previous coordinates
  - Two Vectors are equal regardless of their initial point



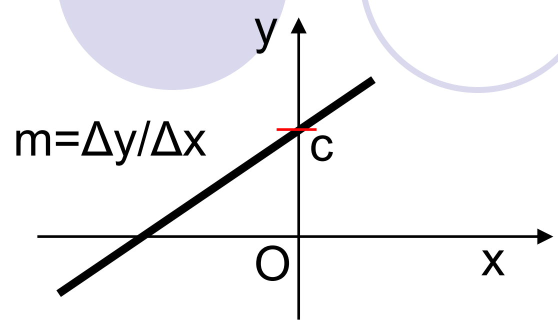
Coordinates & Vector (3)

- Vector Rules (A, B, C are vector & m, n are scalars)
  - Commutative:
    - A + B = B + A
  - Associative:
    - A + (B + C) = (A + B) + C
    - (m * n) * A = m * (n * A)
  - Additive inverses:
    - A + (-A) = 0
  - Scalar distribution over Vector addition:
    - m * (A + B) = m A + m B
  - Vector distribution over Scalar addition:
    - (m + n) * A = m * A + n * A
  - Others
    - A + 0 = A
    - A * 1 = A

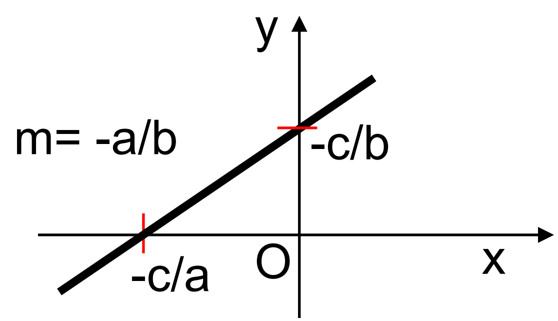


Line Representation

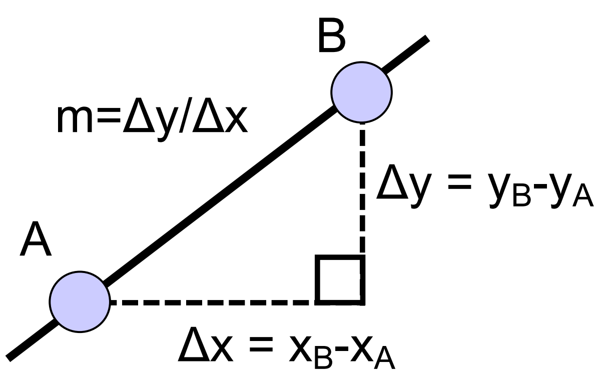
- Point Slope Form:
  - y = m * x + c
  - m = Slope = Δy/Δx
  - c = y-intercept



- General Form
  - a * x + b * y + c = 0



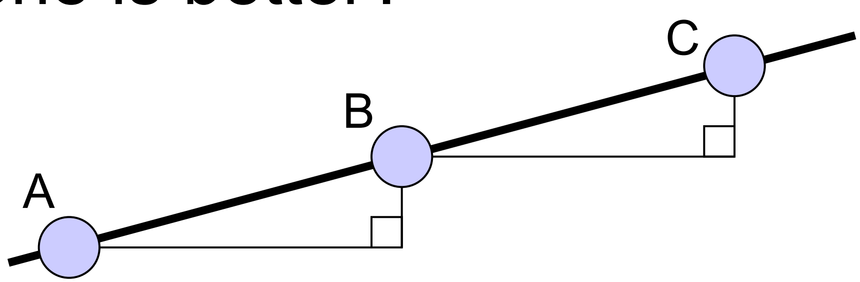
- Two-Point Form
  - $(y - y_B) / (x - x_B) = (y_B - y_A) / (x_B - x_A)$



- All 3 forms are referring to an infinite line but not a ***line segment***

Three points on the same line

- Collinear Points/Lines
  - Given 3 points A, B, C
  - Ensure $(x_C - x_B) <> 0$ and $(x_B - x_A) <> 0$
  - Slopes are the same for line (A, B) and (B, C)
    - $\mathbf{(y_C - y_B) / (x_C - x_B) = (y_B - y_A) / (x_B - x_A)}$
  - Another form
    - $\mathbf{(y_C - y_B) * (x_B - x_A) = (y_B - y_A) * (x_C - x_B)}$
  - Which one is better?



Calculating Distances

- Point-Point Distance - 2D

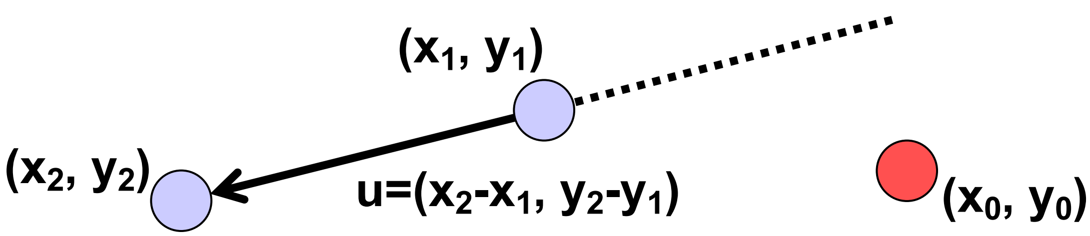
$$\|u\| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- Point-Point Distance - 3D

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

- Point-Line Distance - 2D

$$\frac{|(x_2 - x_1)(y_1 - y_0) - (x_1 - x_0)(y_2 - y_1)|}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}$$



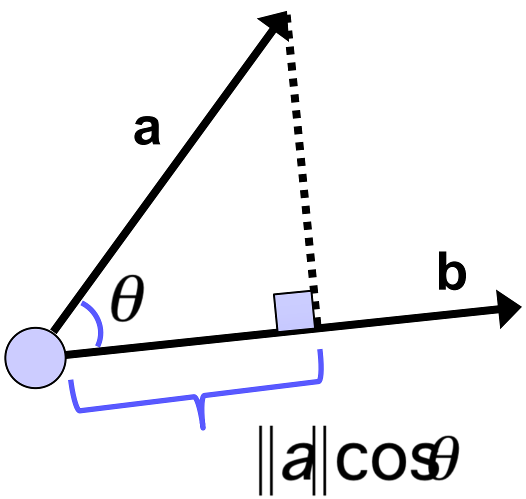
Dot Product

- Dot product: A way to test similar directions

$$a \bullet b = x_a x_b + y_a y_b$$

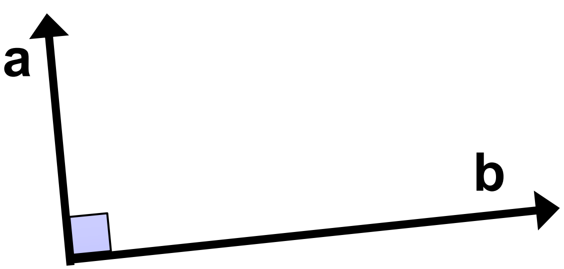
$$a \bullet b = \|a\| \|b\| \cos\theta$$

$$\|a\| \cos\theta = \frac{a \bullet b}{\|b\|}$$

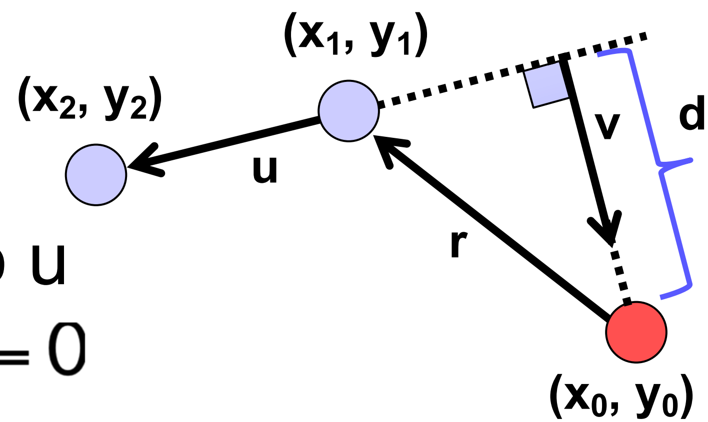


- Dot product is zero when vectors are perpendicular

$$a \bullet b = 0$$



Point-Line Distance - 2D

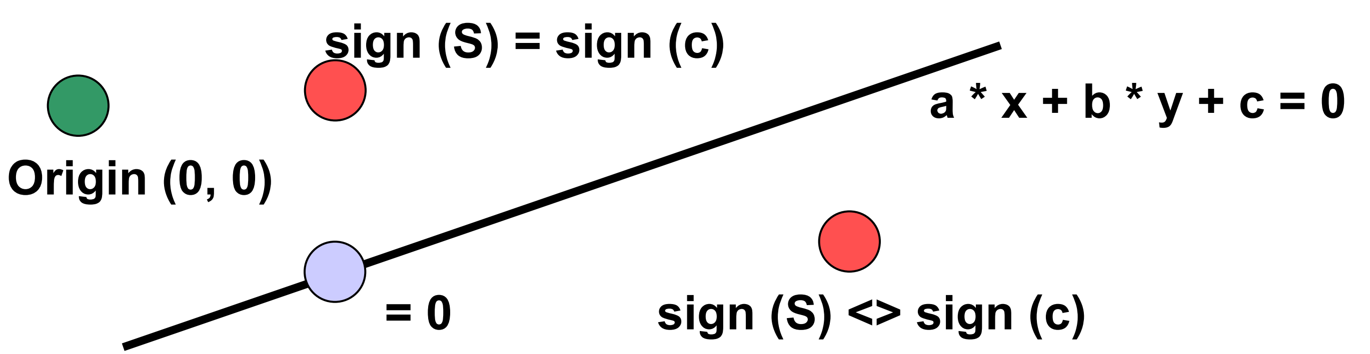


- **u** is the vector from P_2 to P_1
  - $u = (x_2-x_1, y_2-y_1)$
- **v** is the vector perpendicular to u
  - $v = (-(y_2-y_1), x_2-x_1)$ Note: $u \bullet v = 0$
- **r** is the vector from P_0 to P_1
  - $r = (x_1-x_0, y_1-y_0)$
- distance *d* is the projection of r into the direction of v

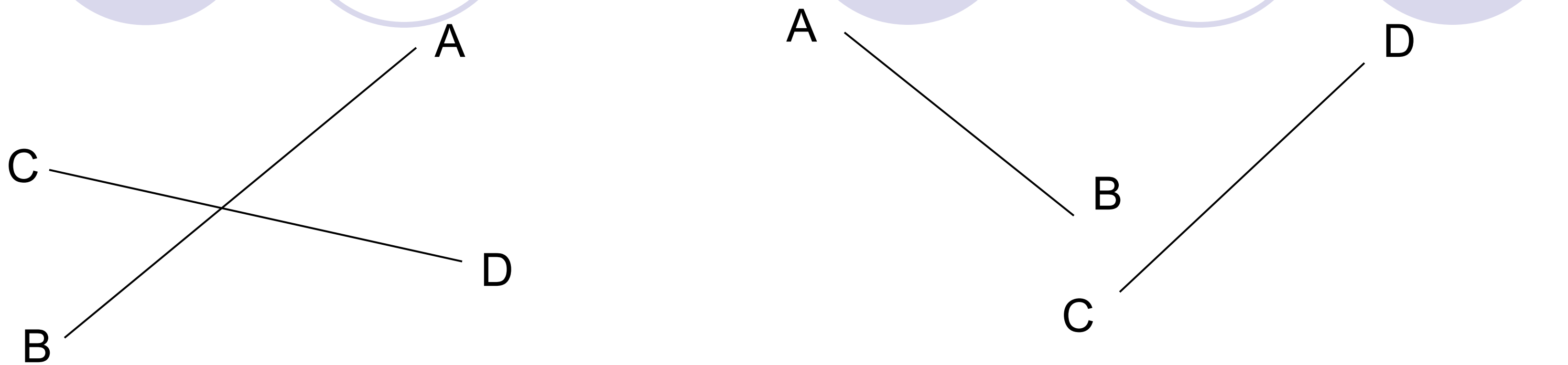
$$d = \frac{|v \bullet r|}{\|v\|} = \frac{|(x_2 - x_1)(y_1 - y_0) - (x_1 - x_0)(y_2 - y_1)|}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}$$

Finding Point Position

- Which Side of the Line
  - Given a point $P=(x_0,y_0)$ and substitute into
    - **$a * x + b * y + c = 0$**
    - **$S= a * x_0 + b * y_0 + c$**
  - 3 possible results of S
    - Equal to **Zero:** On the line
    - Same sign as **c**: Same side as (0, 0)
    - Different sign as **c**: Different side as (0, 0)



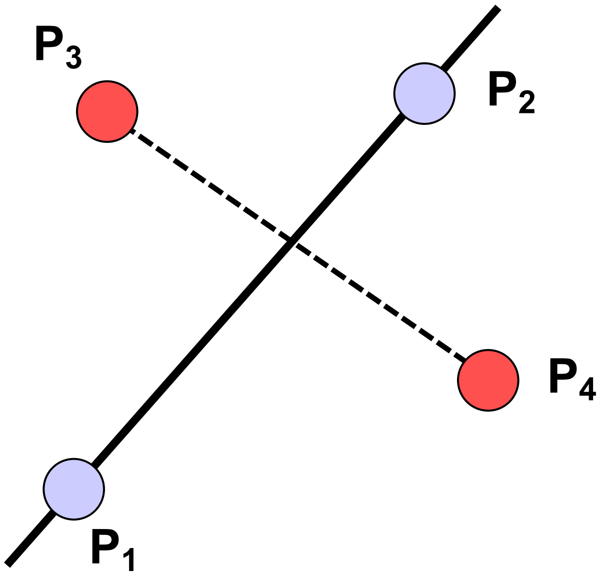
Line Intersection

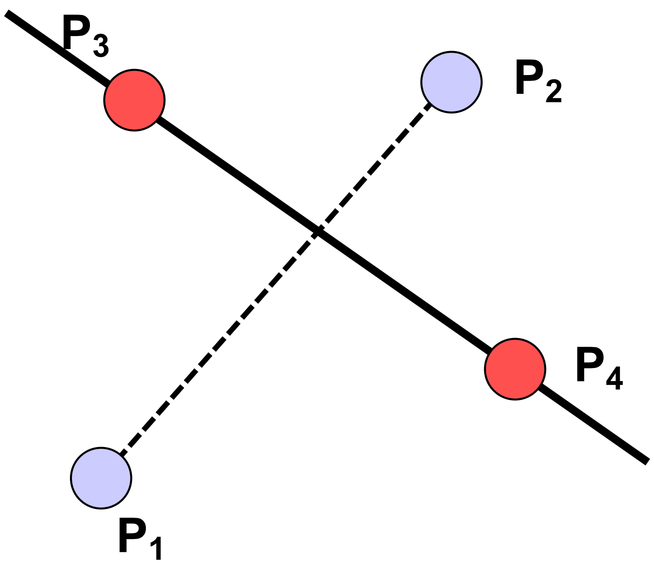


- Methods
  - line representation of AB and CD
    - Point slope form? No
    - Two point form? No
    - General form?
  - Calculate the common solution
    - Yes – intersect
    - No – parallel
  - If you get the unique intersection point of two lines
    - Test whether it is in the range of two segments
    - Float point division error make this not accurate

Line Intersection

- 2 Ends of the 2nd Line should be in different sides respect to the 1st Line
- Vice Versa

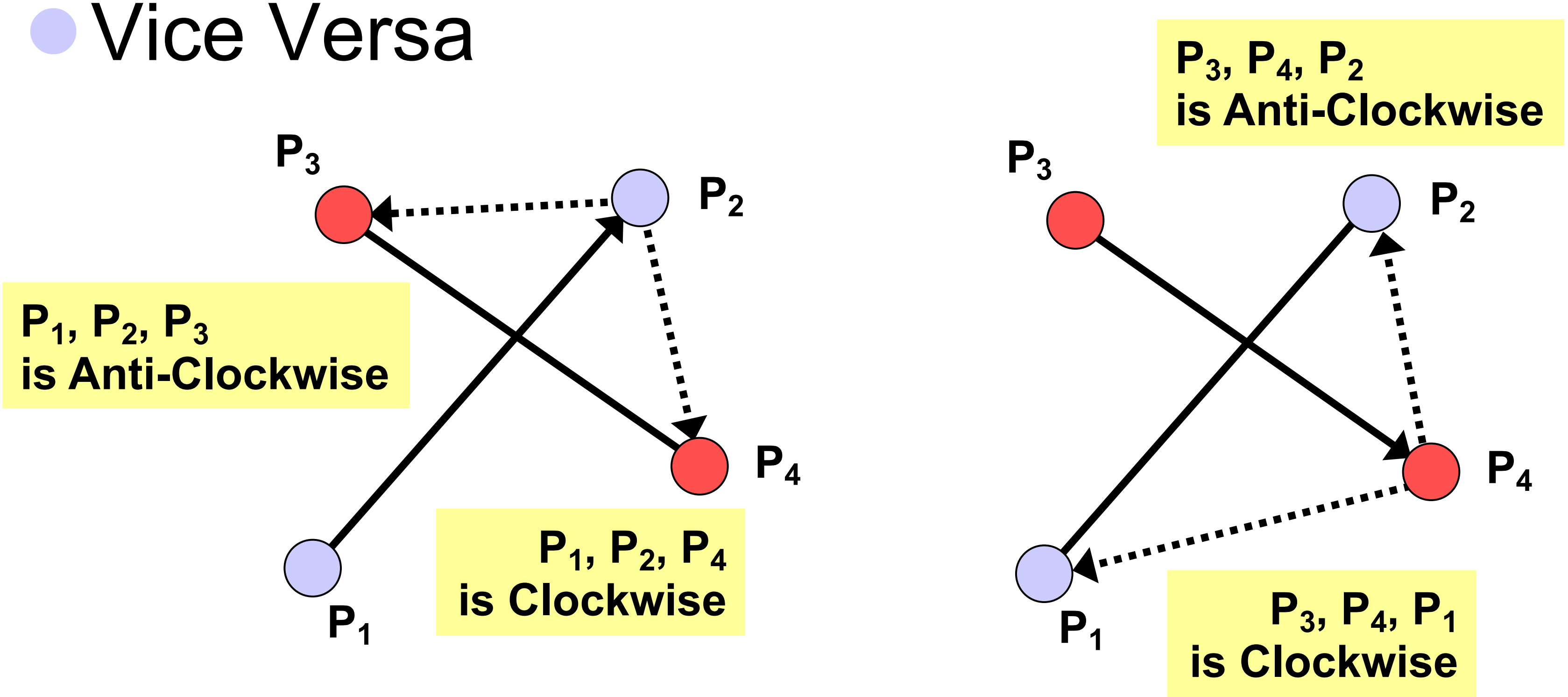




- Why need to test twice?

Line Intersection (improvement)

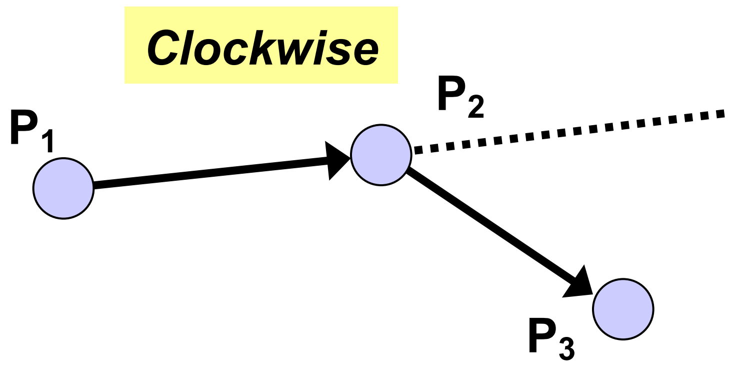
- 2 Ends of the 2nd Line should be in different sides respect to the 1st Line
- Vice Versa

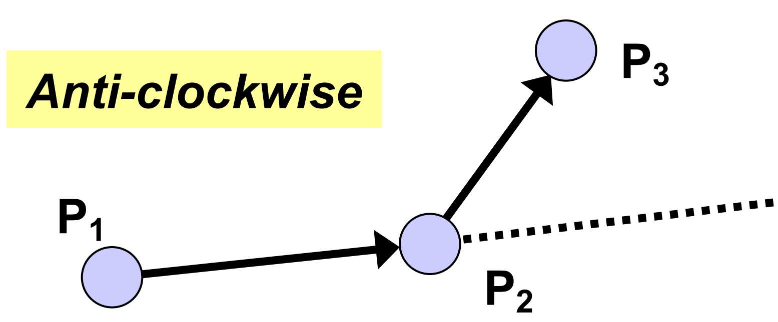


- Look at direction of vector turns.

Finding Vector Direction

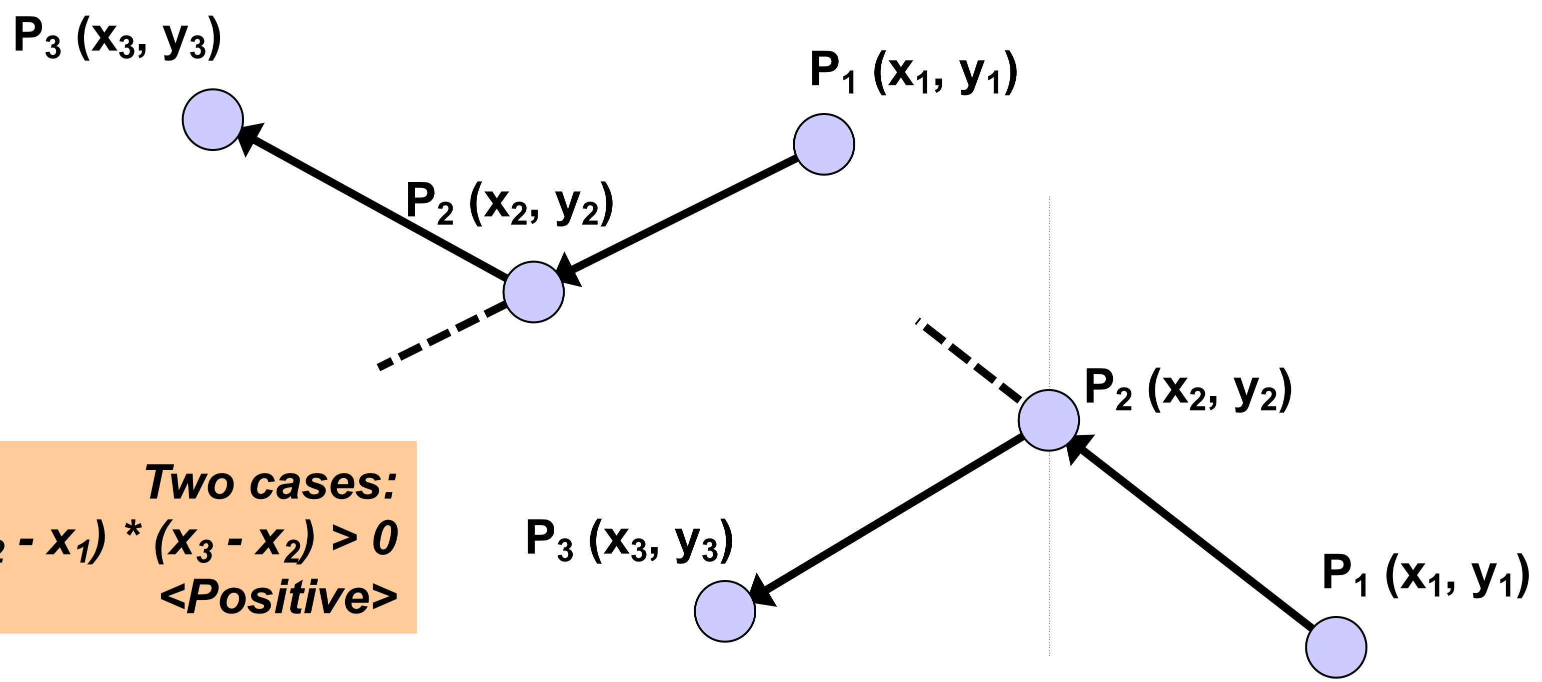
- In 2D Coordinate System, judging which side the vector turns
  - Given 3 points
  - We cannot determine the direction simply by their slopes





Finding Vector Direction

- Determining Direction (Case 1)
  - When $(x_3 - x_2)$ and $(x_2 - x_1)$ are the same sign



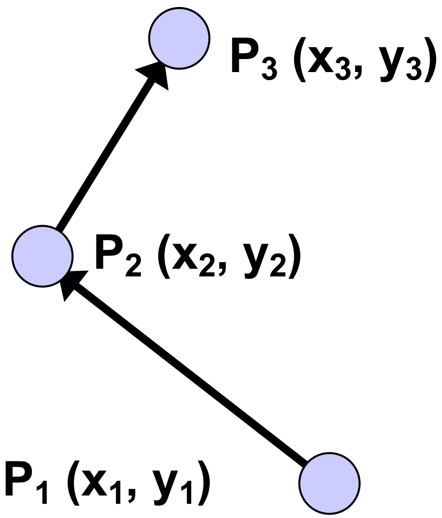
Two cases:
$(x_2 - x_1) * (x_3 - x_2) > 0$
<Positive>

  - It is Anti-Clockwise, if and only if

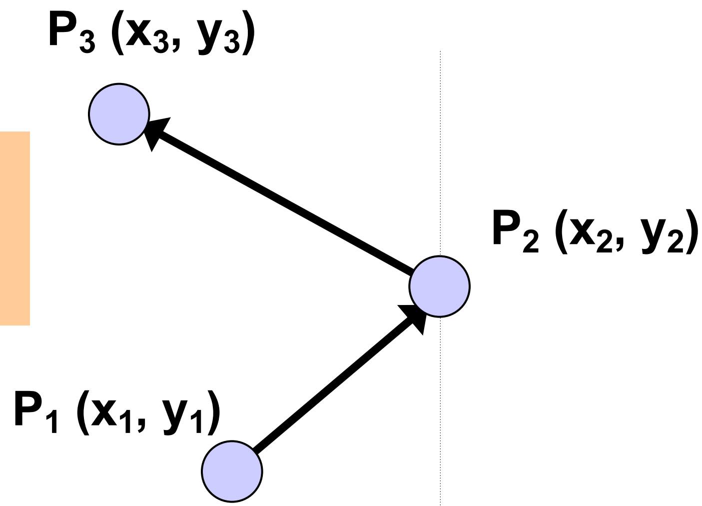
$(y_2 - y_1) / (x_2 - x_1) < (y_3 - y_2) / (x_3 - x_2)$

Finding Vector Direction

- Determining Direction (Case 2)
  - When $(x_3 - x_2)$ and $(x_2 - x_1)$ are different signs



Two cases:
$(x_2 - x_1) * (x_3 - x_2) < 0$
<Negative>



  - It is Anti-Clockwise, if and only if

    $(y_2 - y_1) / (x_2 - x_1)$ **>** $(y_3 - y_2) / (x_3 - x_2)$

Finding Vector Direction

- Determining Anti-Clockwise (Case 1)
  - $(y_2 - y_1) / (x_2 - x_1)$ **<** $(y_3 - y_2) / (x_3 - x_2)$
  - Multiply 2 sides with $(x_2 - x_1) * (x_3 - x_2) > 0$
  - Then, $\boldsymbol{(x_3 - x_2)(y_2 - y_1) < (y_3 - y_2)(x_2 - x_1)}$
- Determining Anti-Clockwise (Case 2)
  - $(y_2 - y_1) / (x_2 - x_1)$ **>** $(y_3 - y_2) / (x_3 - x_2)$
  - Multiply 2 sides with $(x_2 - x_1) * (x_3 - x_2) < 0$
  - Then, $\boldsymbol{(x_3 - x_2)(y_2 - y_1) < (y_3 - y_2)(x_2 - x_1)}$
- Determining Anti-Clockwise (Both Cases)
  - $\mathbf{(y_3 - y_2)(x_2 - x_1) - (x_3 - x_2)(y_2 - y_1) > 0}$
  - $\mathbf{(x_1y_2 - x_2y_1) + (x_2y_3 - x_3y_2) + (x_3y_1 - x_1y_3) > 0}$

Finding Vector Direction

- General Form of Determinant

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

$$\begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix} = a_1 \begin{vmatrix} b_2 & b_3 \\ c_2 & c_3 \end{vmatrix} - a_2 \begin{vmatrix} b_1 & b_3 \\ c_1 & c_3 \end{vmatrix} + a_3 \begin{vmatrix} b_1 & b_2 \\ c_1 & c_2 \end{vmatrix}$$

$$= a_1b_2c_3 - a_1b_3c_2 - a_2b_1c_3 + a_2b_3c_1 + a_3b_1c_2 - a_3b_2c_1$$

- Simplified Form of Determinant

$$\begin{vmatrix} 1 & a_1 & a_2 \\ 1 & b_1 & b_2 \\ 1 & c_1 & c_2 \end{vmatrix} = (a_1b_2 - a_2b_1) + (b_1c_2 - b_2c_1) + (c_1a_2 - c_2a_1)$$

$$= \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} + \begin{vmatrix} b_1 & b_2 \\ c_1 & c_2 \end{vmatrix} + \begin{vmatrix} c_1 & c_2 \\ a_1 & a_2 \end{vmatrix}$$

Finding Vector Direction

- Determining Anti-Clockwise (Generalize)

$$(x_1 y_2 - x_2 y_1) + (x_2 y_3 - x_3 y_2) + (x_3 y_1 - x_1 y_3)$$

$$= \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & y_2 \\ x_3 & y_3 \end{vmatrix} + \begin{vmatrix} x_3 & y_3 \\ x_1 & y_1 \end{vmatrix} = \begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} > 0$$

- Determining Directions (Generalize)

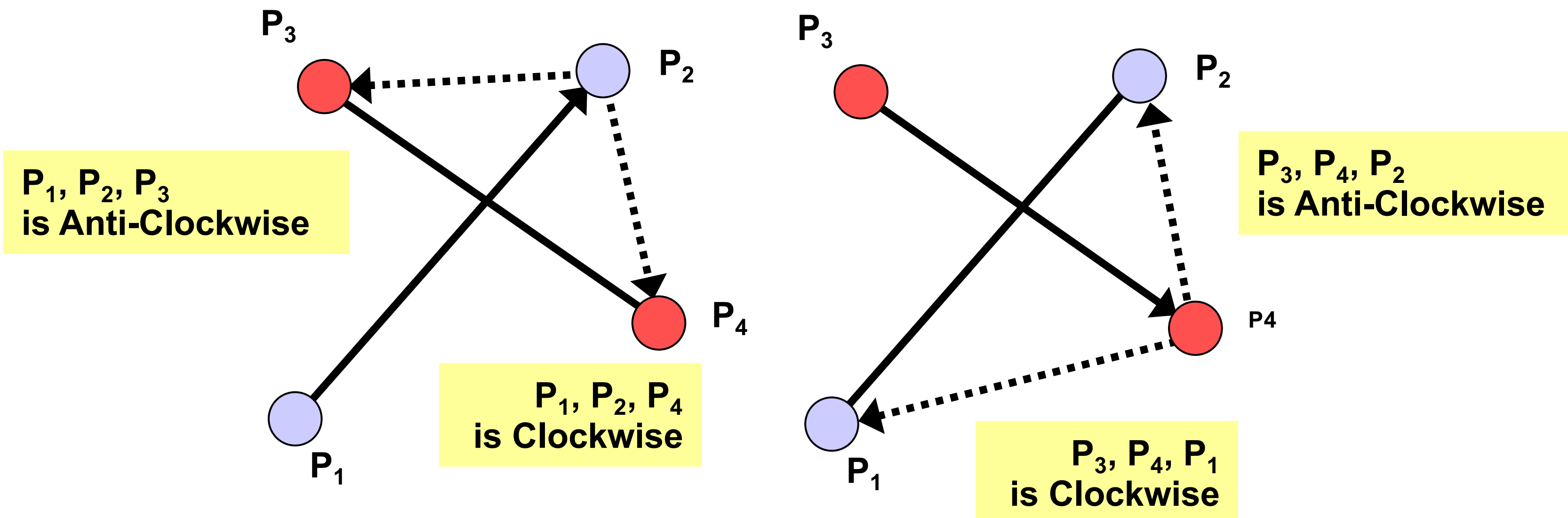
If > 0, then Anti-Clockwise

If < 0, then Clockwise

If = 0, then Collinear

Line Intersection

- Let $\boldsymbol{cw}\ (P_1, P_2, P_3) = \begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix}$
- Given 2 Lines (P_1, P_2) and (P_3, P_4)

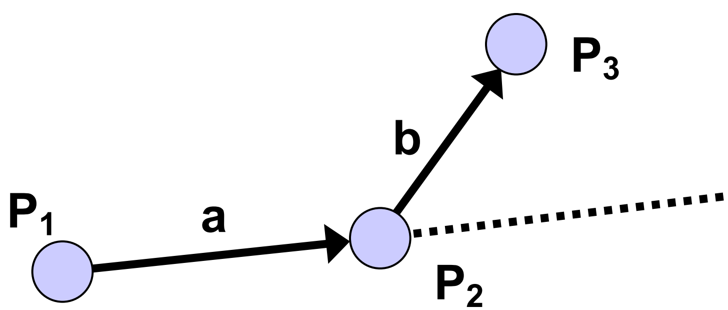


- They are intersected, if and only if
  - ***cw*** (P_1, P_2, P_3) * ***cw*** $(P_1, P_2, P_4) < 0$
  - ***cw*** (P_3, P_4, P_1) * ***cw*** $(P_3, P_4, P_2) < 0$
- Be careful on those marginal cases!

Vector Direction - Another Explanation

- If we consider two vectors instead of three points
  - $(y_3 - y_2)(x_2 - x_1) - (x_3 - x_2)(y_2 - y_1) > 0$
  - $x_b=x_3-x_2$; $y_b=y_3-y_2$; $x_a=x_2-x_1$; $y_a=y_2-y_1$
  - $y_b x_a - x_b y_a > 0$
- This is called the "cross product of two vectors"

$$a \times b = x_a y_b - x_b y_a = \begin{vmatrix} x_a & y_a \\ x_b & y_b \end{vmatrix}$$



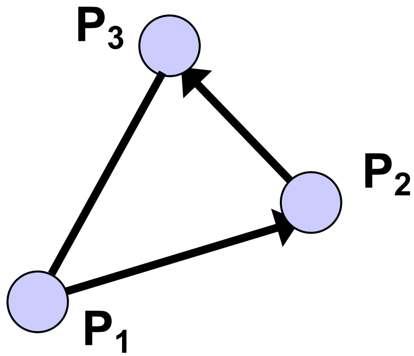
Anti-clockwise

$$a \times b > 0$$

Area of Triangle

- Given 3 Points

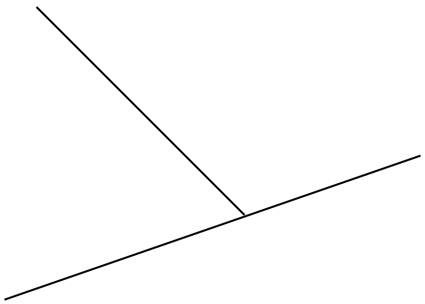
$$\textbf{Area} = \frac{1}{2} * \begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix}$$



- Also, notice those marginal cases
  - Determinant is negative (points are clockwise)
  - 3 points are collinear
  - In C/C++, testing a float number is Zero may not always get the expected result
    - You may define a constant called Epsilon = 0.0001, which specified the floating point accuracy
    - If the absolute value of a float number N is less than or equal to the Epsilon, it is considered as zero

Line Intersection

- How to find out boundary cases?



Exercises

- #184 - Laser Lines (**Try this first**)
  http://online-judge.uva.es/p/v1/184.html
- #191 - Intersection
  http://online-judge.uva.es/p/v1/191.html
- #248 - Cutting Corners (Challenging)
  http://online-judge.uva.es/p/v2/248.html
- #460 - Overlapping Rectangles
  http://online-judge.uva.es/p/v4/460.html
- #866 - Intersecting line segments (**So Easy**)
  http://online-judge.uva.es/p/v8/866.html

Polygons

- Revision on Intersection
- Area of Polygons
- Convex Polygon and Non-Convex Polygon
- Point Inside Polygon
- Exercises
- References

Revision on Intersection

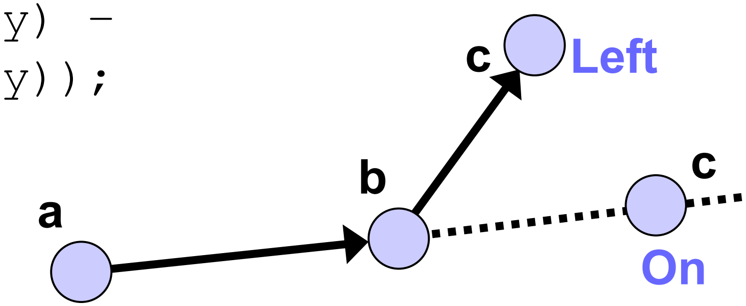
- Basic Knowledge
  - Distinguish between a Line and a Line Segment
  - Which side is a Point located according to a Directed Line (Vector)
    - Anti-clockwise & Clockwise
  - The general case can be illustrated by the determinant
  - The determinant is the area of the triangle times 2
    - *area (a, b, c) * 2 = determinant (a, b, c)*

Revision on Intersection (2)

- 2 times of the triangle area

```
int areaX2(Point2D &a, Point2D &b, Point2D &c)
{
  return ((b.x - a.x) * (c.y - a.y) -
          (c.x - a.x) * (b.y - a.y));
}
```

- Point Positions

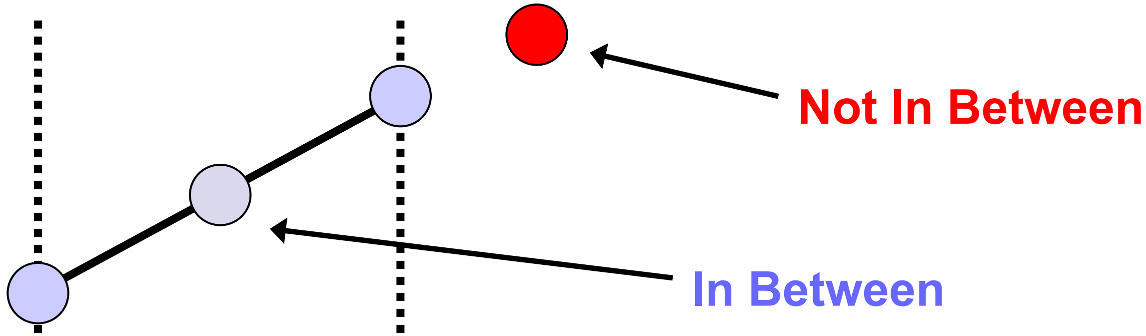



```
int leftSide(Point2D &a, Point2D &b, Point2D &c)
{  return (areaX2(a, b, c) > 0);  }

int leftSideOn(Point2D &a, Point2D &b, Point2D &c)
{  return (areaX2(a, b, c) >= 0);  }

int collinear(Point2D &a, Point2D &b, Point2D &c)
{  return (areaX2(a, b, c) == 0);  }
```

Revision on Intersection

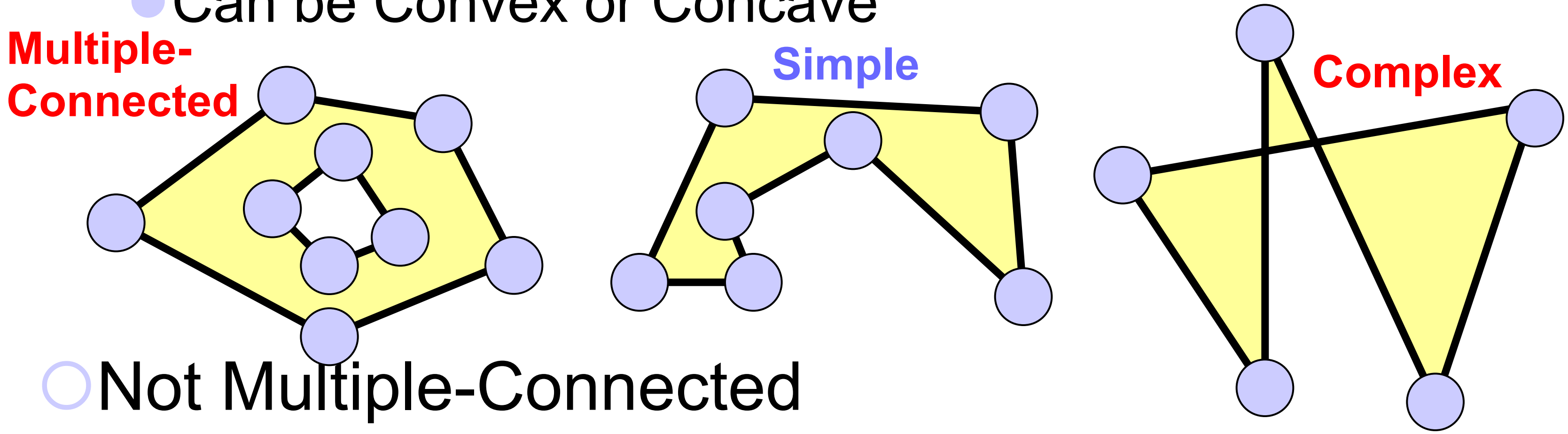
Point C Between Point A & B

```
bool between(Point2D &a, Point2D &b, Point2D &c)
{
  Point2D  ba, ca;
  if (!collinear(a,b,c)) return false;

  if (a.x != b.x) // Line A-B is not Vertical
    return (((a.x <= c.x) && (c.x <= b.x)) ||
            ((a.x >= c.x) && (c.x >= b.x)));
  else
    return (((a.y <= c.y) && (c.y <= b.y)) ||
            ((a.y >= c.y) && (c.y >= b.y)));
}
```



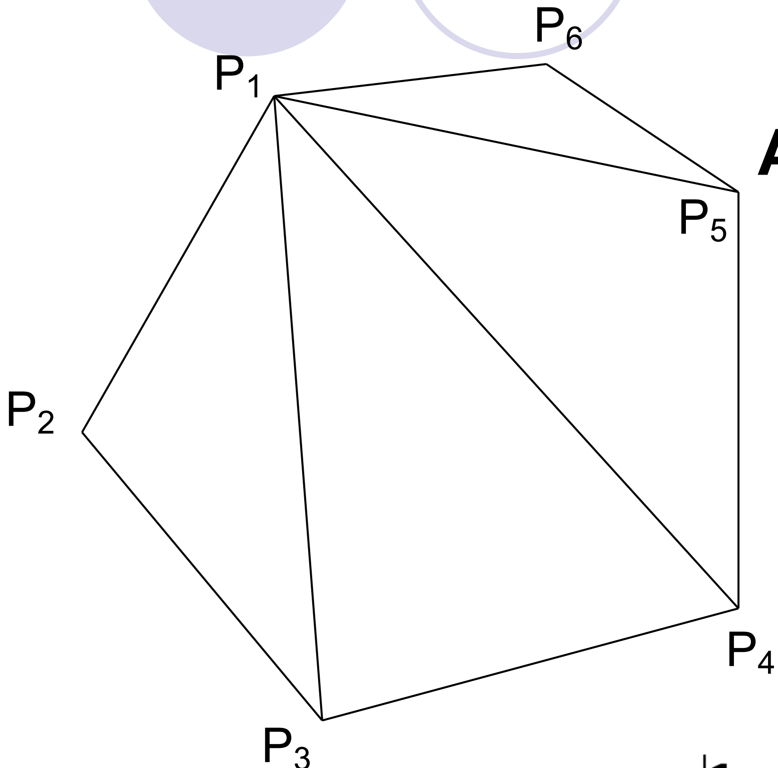
Polygon Triangulation

- Given a 2D Polygon
  - A set of directed points/vectors
  - Simple Polygon
    - Boundary not intersect itself anywhere
    - Can be Convex or Concave



  - Not Multiple-Connected
  - Divide the Simple Polygon into triangles
  - No need to introduce new point

Triangulation of Convex Polygon



$$A(P_1,P_2,P_3) = \frac{1}{2} * \begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix}$$

$$= \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & y_2 \\ x_3 & y_3 \end{vmatrix} + \begin{vmatrix} x_3 & y_3 \\ x_1 & y_1 \end{vmatrix}$$

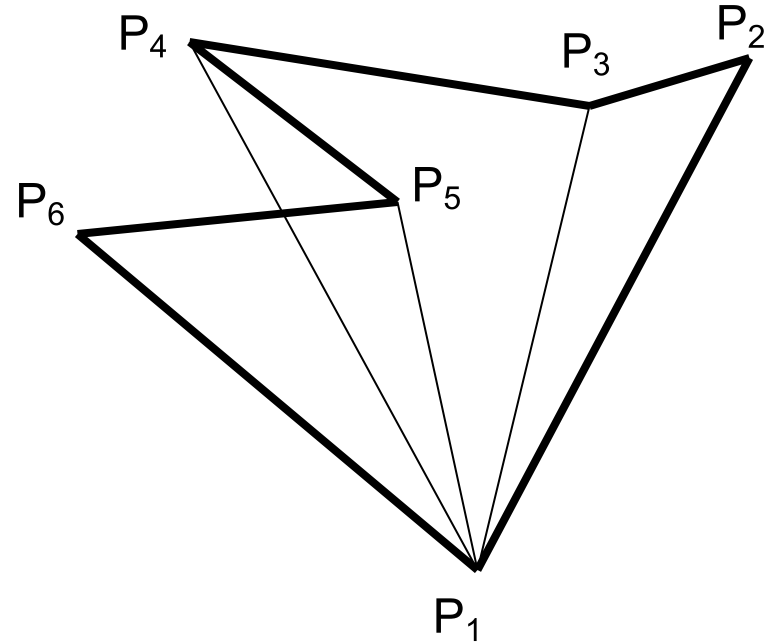
$$A = \sum_{i=1}^{N-2} A(P_1, P_{i+1}, P_{i+2}) = \frac{1}{2} \sum_{i=1}^{N-2} \begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_{i+1} & y_{i+1} \\ 1 & x_{i+2} & y_{i+2} \end{vmatrix} = \frac{1}{2} \sum_{i=1}^{N} \begin{vmatrix} x_i & y_i \\ x_{i+1} & y_{i+1} \end{vmatrix}$$

Note: $P_{N+1} = P_1$

Partition of General Polygons

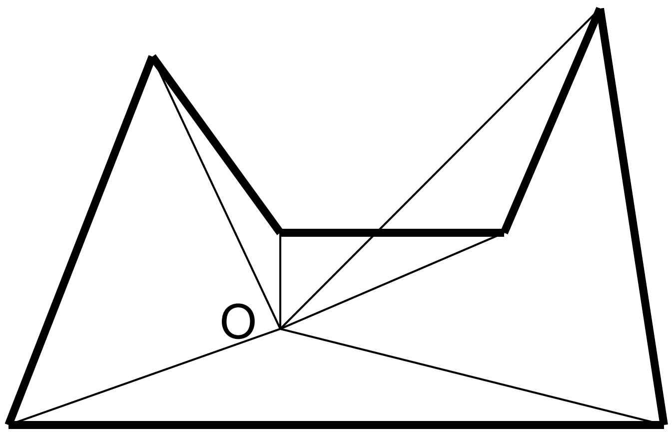
- Directed Area
  - Right hand order (clockwise): negative
  - Left hand order (anti-clockwise): positive

$$A = \sum_{i=1}^{N-2} A(P_1, P_{i+1}, P_{i+2}) = \frac{1}{2}\sum_{i=1}^{N} \begin{vmatrix} x_i & y_i \\ x_{i+1} & y_{i+1} \end{vmatrix}$$



A Direct Way to get the formula

- Use an arbitrary point to partition the polygon into N triangles
- Use the origin!

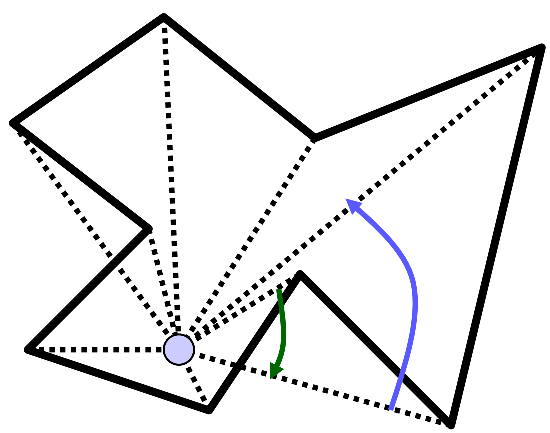


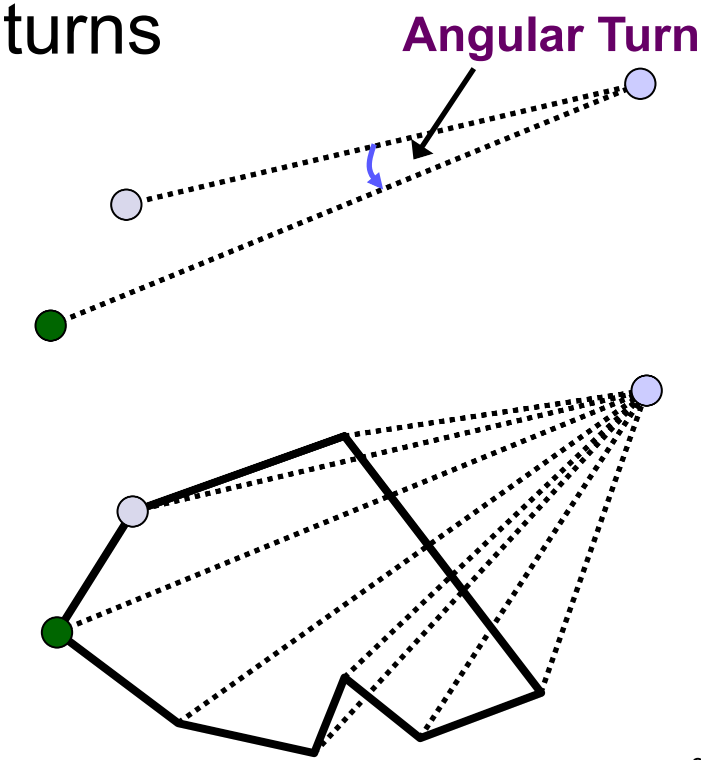
Point Inside Polygon

- Given a Simple Polygon & a Point
  - The point will be either
    - *Inside the polygon* (Strictly Interior)
    - *Outside the polygon* (Strictly Exterior)
    - *On the **edge*** (but not an Endpoint)
    - *Same as the polygon **vertex/point***
  - Winding Number Algorithm
    - Simple to implement
    - Hard to detect on **Edge** and **Vertex** cases
  - Ray Crossings Algorithm
    - Complicated
    - More Precise to identify all 4 cases

Point Inside Polygon

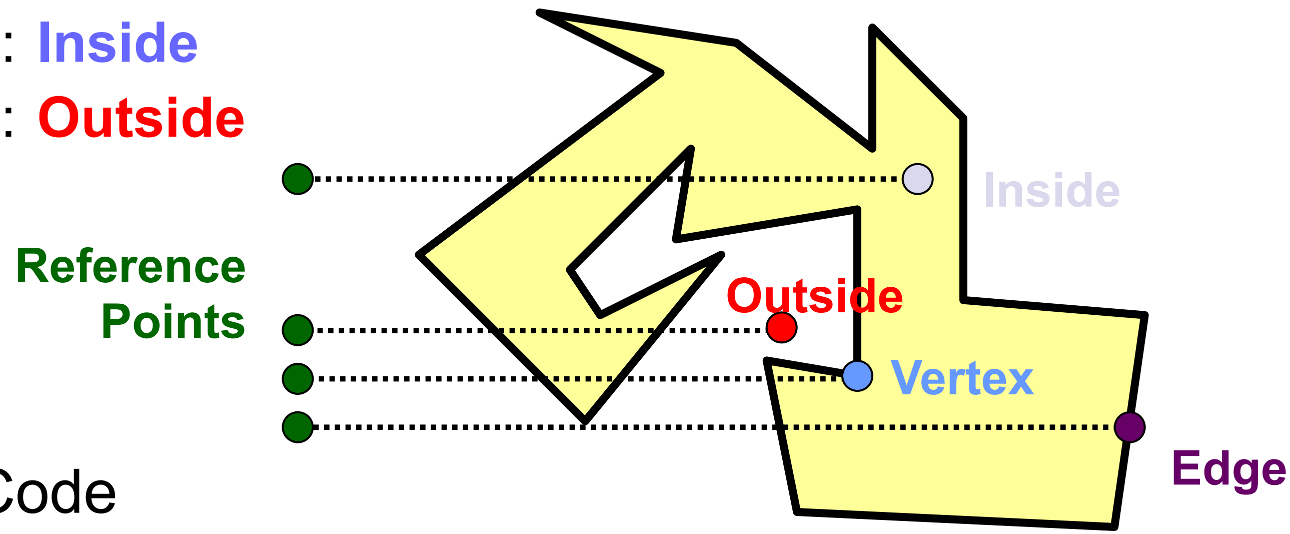
- Winding Number Algorithm
  - Calculate all the Angular Turn of 2 consecutive edge points
  - Sum all the angles of the turns
    - 0º : **Outside**
    - 360º : **Inside**





Point Inside Polygon

- Ray Crossings Algorithm
  - Using a reference point
    - Far enough to be outside the polygon
    - Form a **Horizontal Line** with Testing Point
  - Count the number of Intersections for the **Horizontal Line** and **All Edges**
    - Odd : **Inside**
    - Even : **Outside**



  - Source Code
    http://maven.smith.edu/~orourke/books/compgeom.html

Exercises

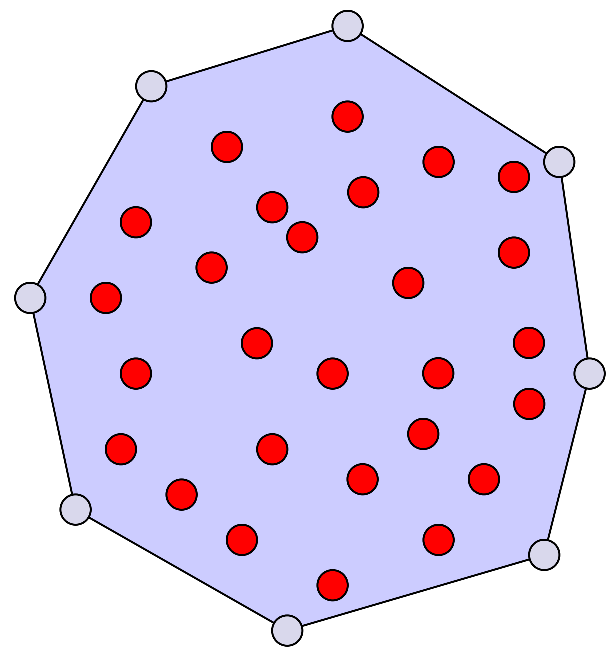
- #137 - Polygons
  http://online-judge.uva.es/p/v1/137.html
- #132 - Bumpy Objects
  http://online-judge.uva.es/p/v1/132.html
- #588 - Video Surveillance
  http://online-judge.uva.es/p/v5/588.html
- #881 - Points, Polygons & Containers
  http://online-judge.uva.es/p/v8/881.html
- #10321 - Polygon Intersection
  http://online-judge.uva.es/p/v103/10321.html
- #634 - Polygon
  http://online-judge.uva.es/p/v6/634.html
- ACM ICPC 2002 World Final
  - Problem F, Toil for Oil
    http://icpc.baylor.edu/past/icpc2002/Finals/problems.pdf

Convex Hull

- What is Convex Hull?
- Finding the Extremes
- Gift Wrapping Algorithm
- More Convex Hull Property
- Quick Hull Algorithm
- Merge Hull Algorithm
- Graham Scan
- Exercises
- References

What is Convex Hull?

- Given a finite number of points as **S**
- The smallest convex set containing **S**
- 2-D Convex Hull
  - All points are on a plane
  - Like using a stretched rubber band to surround all the points

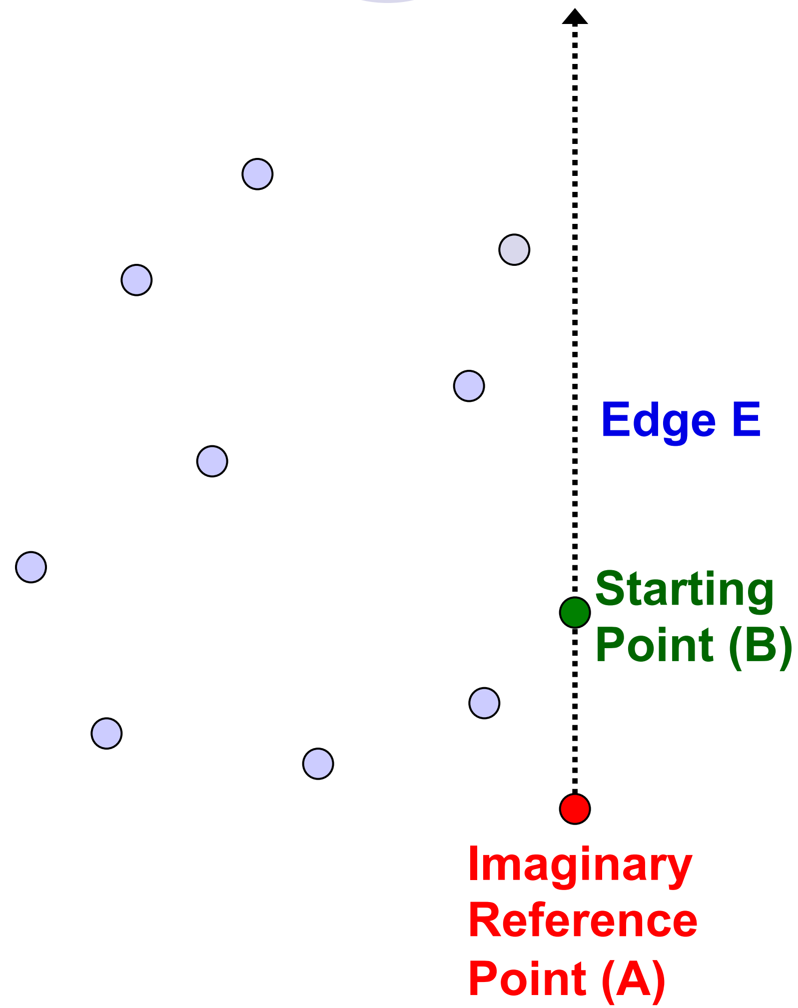


What is Convex Hull?

- What are the Extremes?
  - Extreme Point:
    - Point on the Convex Hull Polygon
  - Extreme Edge:
    - Line Segment on the Convex Hull Polygon
- Possible Solutions
  - All Extreme Points in boundary traversal order
  - All Extreme Edges in boundary traversal order

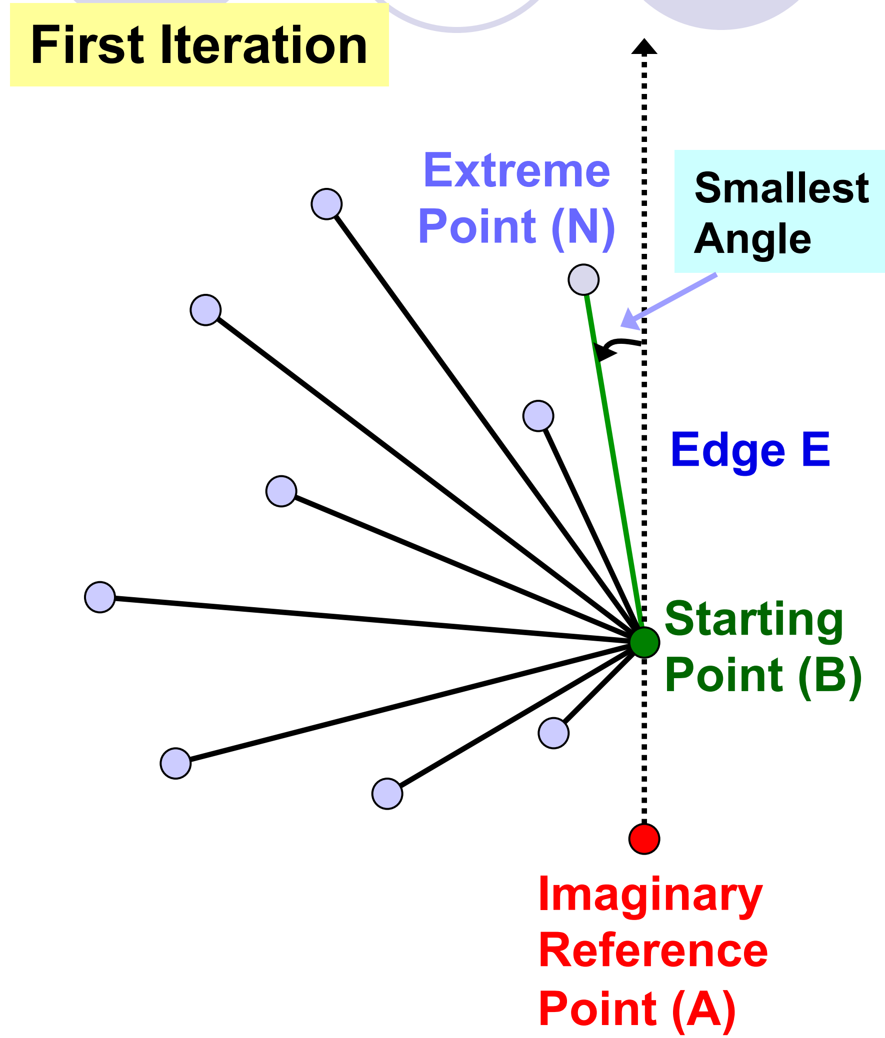
Gift Wrapping Algorithm

- Start from one of the **Extreme Point** on the *Convex Hull Polygon*
  - Usually the **Rightmost-Lowest** point
  - Which is guaranteed to be an Extreme Point
- Extend the **Extreme Point** to a Vertical Line as the starting edge **E**
- All the other points are on the left side of this starting edge **E**



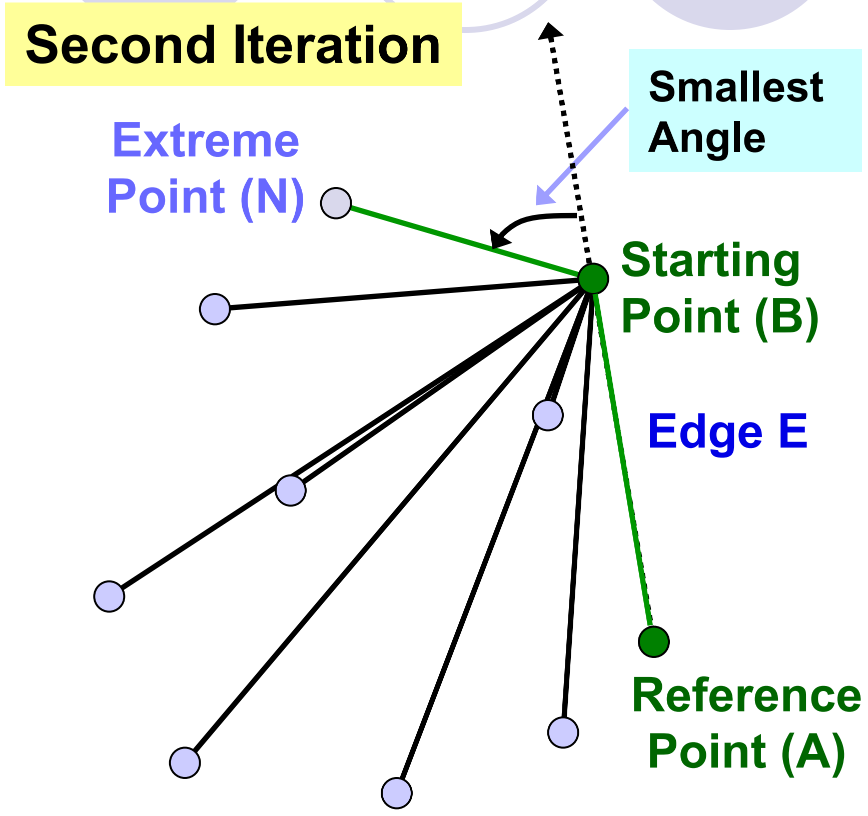
Gift Wrapping Algorithm

- The point with the smallest left turning angle with respect to edge E is the next **Extreme Point**
- **Turning Angles** are important



Gift Wrapping Algorithm

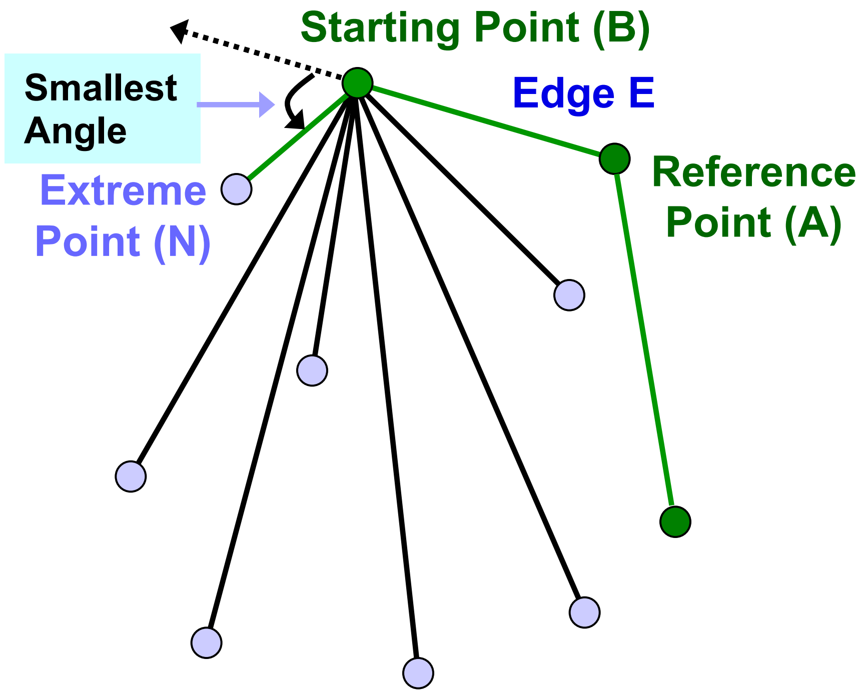
- Form new Edge E from previous two extreme points.
- Find next extreme point
- Iterate…



Gift Wrapping Algorithm

- Complexity is O(N*H), where H is the number of Extreme Edges on the Convex Hull Polygon
- Worst case is that all points are lying on a circle, where H = N

Third Iteration



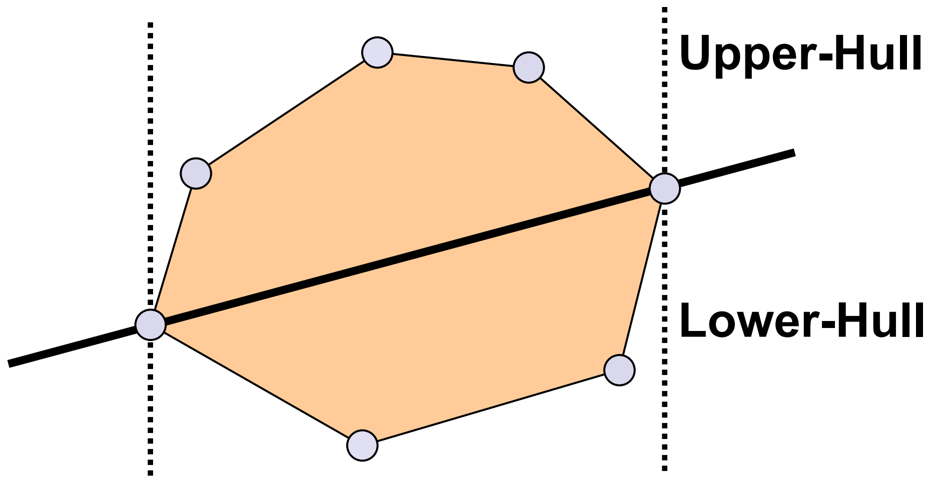
Gift Wrapping Algorithm

```
OrderedPtSet GiftWrap (PointSet S) {
  OrderedPtSet H = EMPTY;
  Point B = RightmostLowest(S); // Starting Point
  Point A = Point(B.x, 0);      // Reference Point
  Point P, N;
  float minAngle = 180.0f, curAngle;
  do {
    for (P=S.begin(); P<=S.end(); P++) {
      curAngle = Angle(A, B, P);// Left-Turn Angle
      if (minAngle > curAngle) {
          minAngle = curAngle;  // Largest Angle
          N = P;                // Rightmost Point
      }
    }
    H.append(B);  // Append the Extreme Point
    A = B;        // Shift the Reference Point
    B = N;        // Shift the Starting Point
  } while (B != H[0]);
  return H;
}
```

Gift Wrapping (Web Sample)

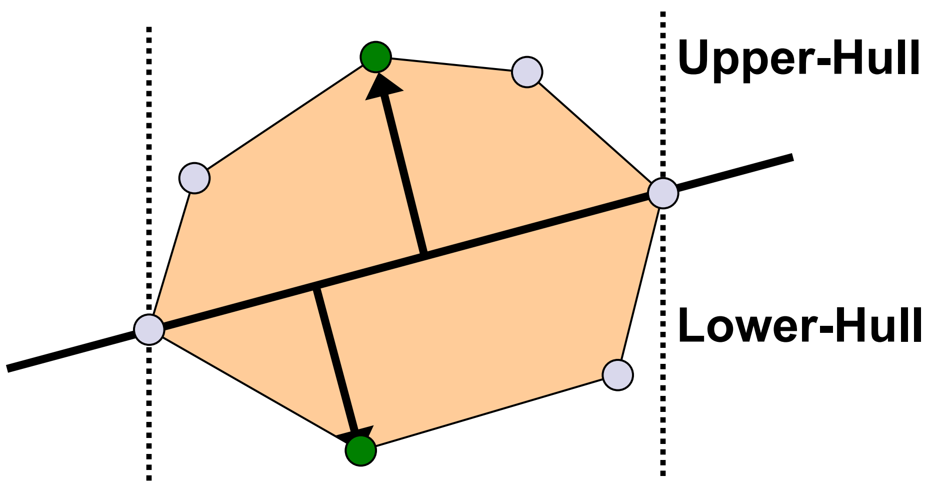
More Convex Hull Property

- For each edge (Extreme Edge) of the convex hull polygon, all points lying on one side only
- Both the **Leftmost** and **Rightmost** points are definitely in the Convex Hull Polygon
- Connecting these 2 points will separate the points into 2 subsets



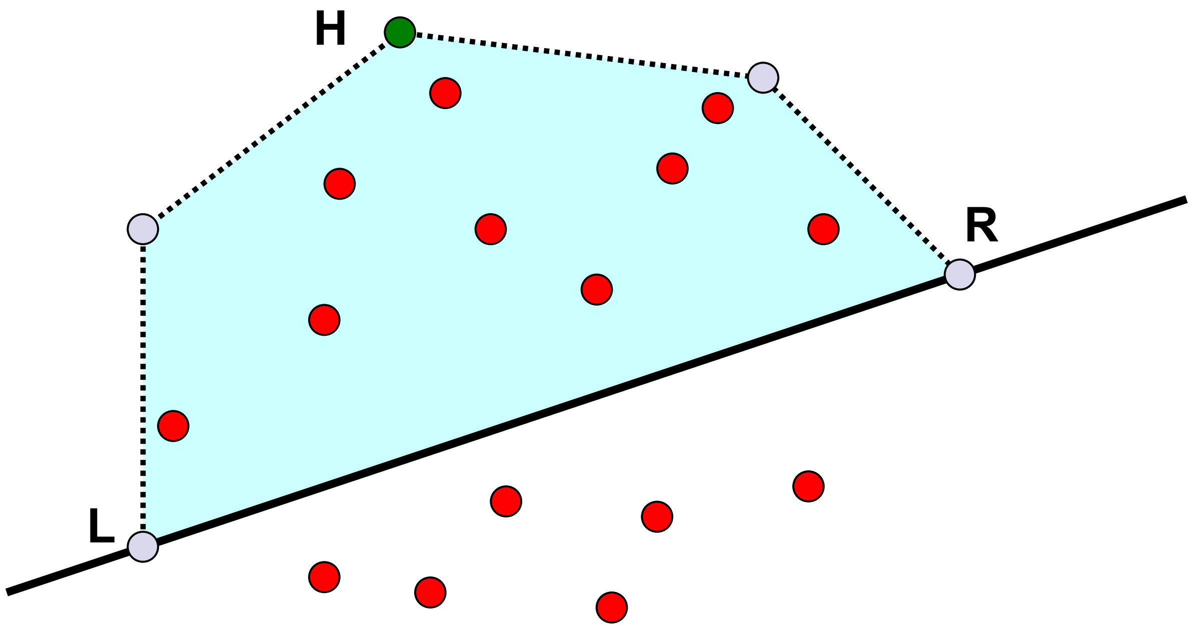
More Convex Hull Property

- Union the Convex Hulls of the points in Upper-Hull and points in Lower-Hull will construct the full-size Convex Hull
- Respect to the separating line, the **farthest** points are the Extreme Points



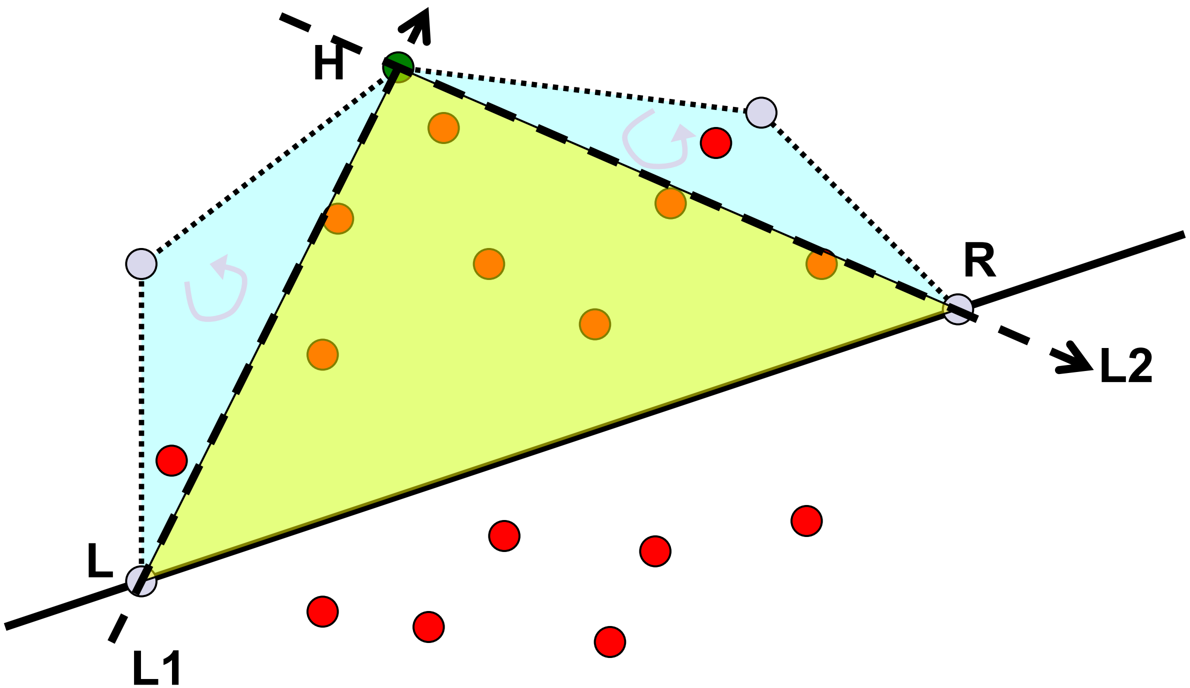
Quick Hull Algorithm

- Initially separate the point set into the *Upper Hull* and the *Lower Hull*
- Find the farthest point **H** with respect to Line **LR**



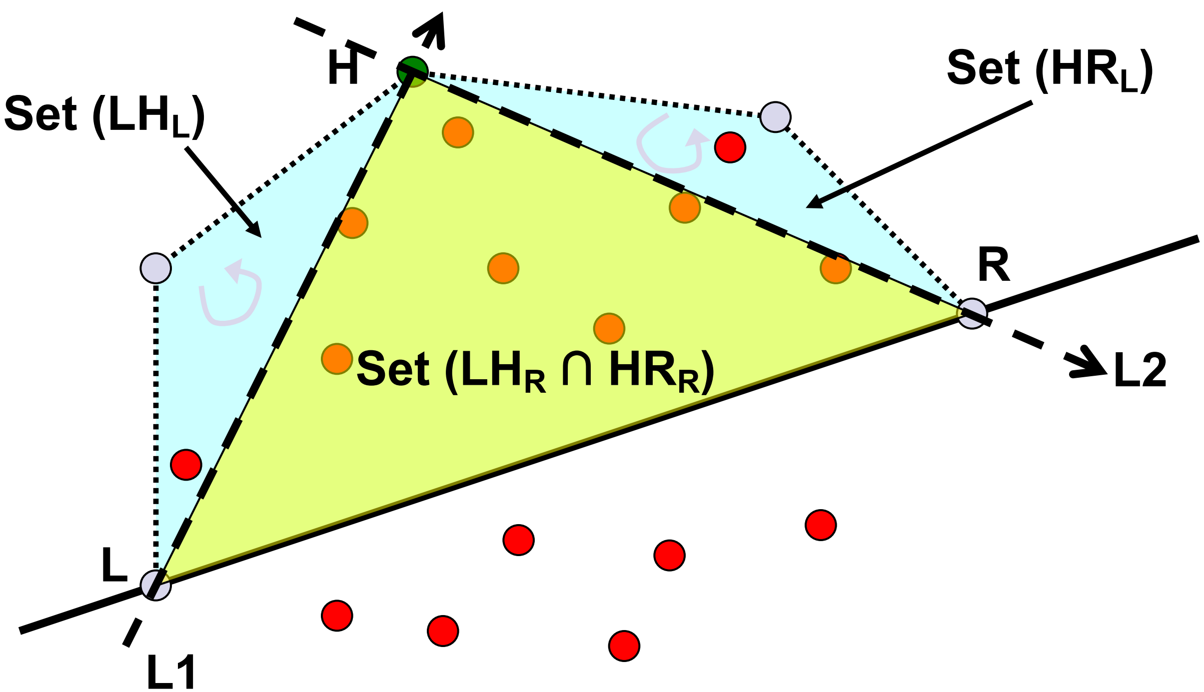
Quick Hull Algorithm

- Construct 2 lines
  - **L1**, directed from **L** to **H**
  - **L2**, directed from **H** to **R**
- Test where points are lying with respect to both **L1** and **L2:** 3 possibilities…



Quick Hull Algorithm

1) Points on Right side of both L1 and L2 (**LH_R** and **HR_R**)
   - not on the Convex Hull Polygon.
2) Points on Left side of L1 (**LH_L**)
   - Find the farthest point with respect to **L1**
   - Recursively perform the test
3) Points on Left side of L2 (**HR_L**) – same procedure as (2)



Quick Hull Algorithm

- Close analogy to the ***Quick-Sort***
  - Find a Pivot to partition the elements into **Left set** and **Right set**
- Difference: all points in set $\mathbf{LH_R HR_R}$ are eliminated in each level of the recursion
- Due to the elimination, it generally run at O (N log N) like the ***Quick-Sort***
- Suffer the same disability as the ***Quick-Sort*** with O (N^2) in the worst-case

Quick Hull Algorithm

```
OrderedPtSet QuickHull(Point L, Point R, PointSet S)
{
  if (S.empty())
    return EMPTY;  // No Point means No Hull
  Point H;  // Farthest Point to Line LR.
  float length = 0, distance;
  for (Point P=S.begin(); P <= S.end(); P++) {
    if (length < (distance = Distance(L, R, P))) {
        length = distance; // Farthest Distance
        H = P;             // Farthest Point
    }
  }
  PointSet A, B;  // Upper and Lower Hull Set
  for (Point P=S.begin(); P <= S.end(); P++) {
    if      (Left(L, H, P)) A.insert(P);
    else if (Left(H, R, P)) B.insert(P);
  }
  return QuickHull(L,H,A) | H | QuickHull(H,R,B);
}
```

Quick Hull (Web Sample)

Quick Hull Algorithm

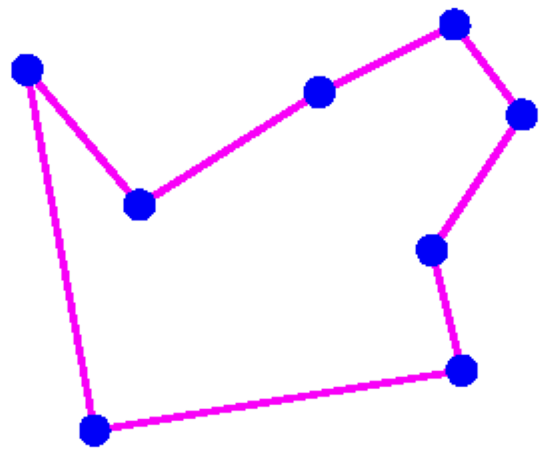
- Pay Attention to
  - Handling the Collinear Points on the Convex Hull Polygon
  - The order/direction of the output points are essential to construct the Convex Hull Polygon
  - What if there are more than 1 Leftmost and Rightmost points?
    - If collinear points are included in the Convex Hull, all the Leftmost and Rightmost points are definitely in the Convex Hull Polygon

Merge Hull Algorithm

- Divide-and-Conquer Algorithm
  - Divide the points in 2 separated sets
  - Until there are <= 3 points in a set
    - Convex Hull of 3 points is a triangle
  - Conquer the solution by merging 2 hulls
- Similar to the Merge-Sort which executes in O (N log N)
- Able to achieve the O (N log N) even in 3-D environment
- Relatively Complicated

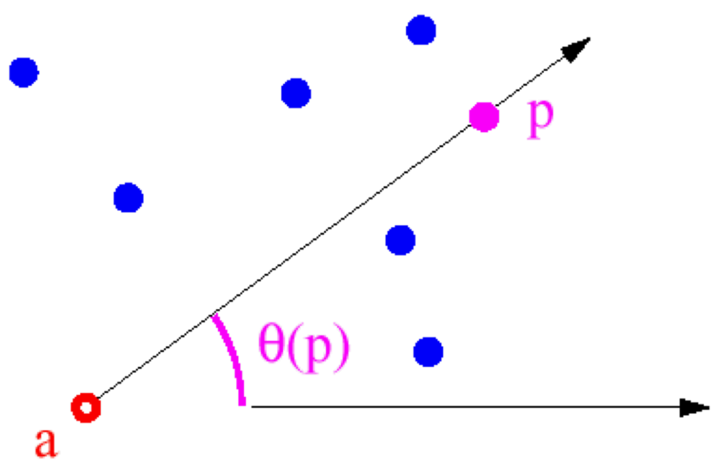
Graham Scan for Convex Hull

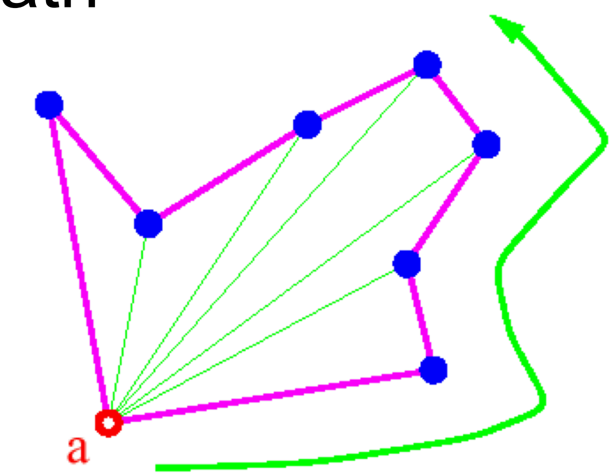
- ***Graham Scan** algorithm.*
  - *Phase 1:* Solve the problem of finding the non-crossing closed path visiting all points



Finding Non-crossing Path

- *How do we find such a non-crossing path:*
  - Pick the bottommost point *a* as the anchor point
  - For each point p, compute the angle θ(p) of the segment (a,p) with respect to the *x*-axis.
  - Traversing the points by **increasing angle** yields a simple closed path

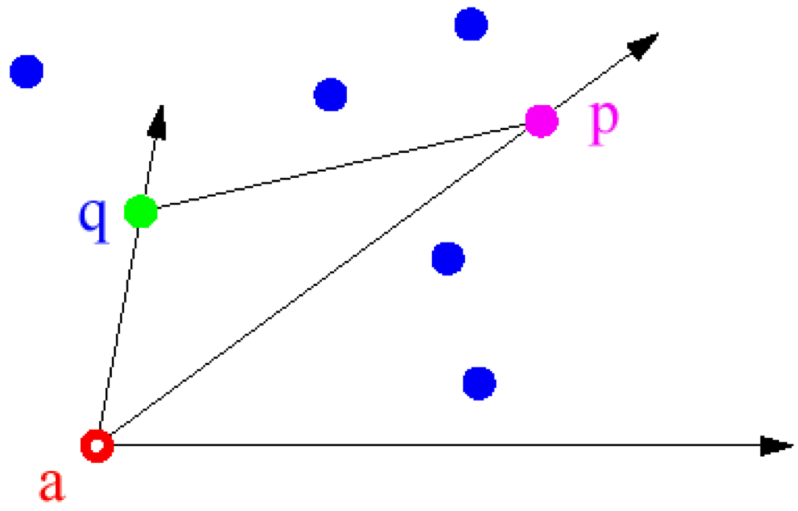




Sorting by Angle

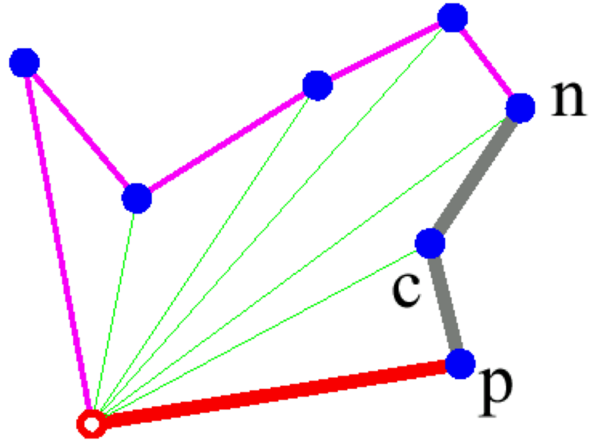
- *How do we sort by increasing angle*?
  - *Observation*: We do not need to compute the actual angle!
  - We just need to compare them for sorting

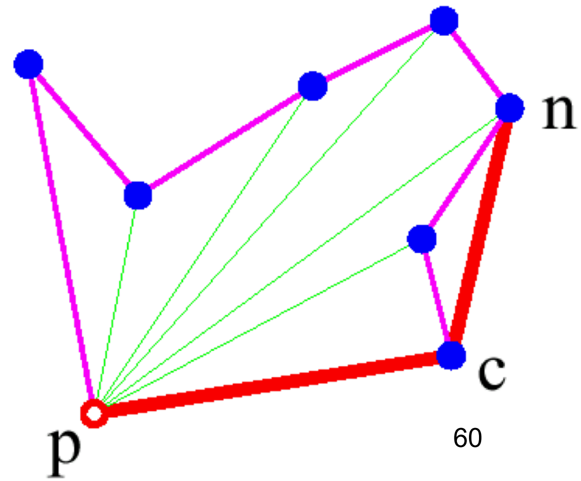
$$\theta(p) < \theta(q) \Leftrightarrow \text{orientation(a,p,q) is anti-clockwise}$$



Rotational Sweeping

- *Phase 2 of Graham Scan:* ***Rotational sweeping***
  - The anchor point and the first point in the polar-angle order have to be in the hull
  - Traverse points in the sorted order:
    - Before including the next point *n* check if the new added segment makes a left turn
    - If not, keep discarding the previous point (*c*) until the left turn is made





Implementation and Analysis

- *Implementation:*
  - *Stack to store vertices of the convex hull*
- *Analysis*:
  - Phase 1: O(n log n)
    - points are sorted by angle around the anchor
  - Phase 2: O(n)
    - each point is pushed into the stack once
    - each point is removed from the stack at most once
  - Total time complexity O(n log n)

Exercises

- #596 - The Incredible Hull

  http://online-judge.uva.es/p/v5/596.html

- #681 - Convex Hull Finding

  http://online-judge.uva.es/p/v6/681.html

References

- Books
  - **Computational Geometry in C**
    by Joseph O'Rourke
  - **Computational Geometry: An Introduction**
    by Franco P. Preparata, Michael Ian Shamos
  - **Computational Geometry: Algorithms & Applications**
    by de Berg, Schwarzkopf, van Kreveld, Overmars
- Web
  - **Geometry: An Introduction (Terms)**
    http://math.about.com/library/weekly/aa031503a.htm
  - **Department of Mathematics (CityU)**
    http://personal.cityu.edu.hk/~ma4527/
  - **MathWorld (Geometry)**
    http://mathworld.wolfram.com/topics/Geometry.html
  - **Mathtools.net (Computational Geometry)**
    http://www.mathtools.net/C_C__/Computational_geometry/index.html
  - **Triangulation (MathWorld)**
    http://mathworld.wolfram.com/Triangulation.html
  - **Mathtools.net (Computational Geometry)**
    http://www.mathtools.net/C_C__/Computational_geometry/index.html
  - **Computational Geometry in C (With Source Code)**
    http://maven.smith.edu/~orourke/books/compgeom.html
  - **Convex Hull Demo (Brute-force & Quick-Hull)**
    http://www.piler.com/convexhull/
  - **Convex Hull 2D/3D Demo (Merge-Hull & Quick Hull)**
    http://www.cse.unsw.edu.au/~lambert/java/3d/hull.html
  - **QHull code for Convex Hull (and other features)**
    http://www.qhull.org/