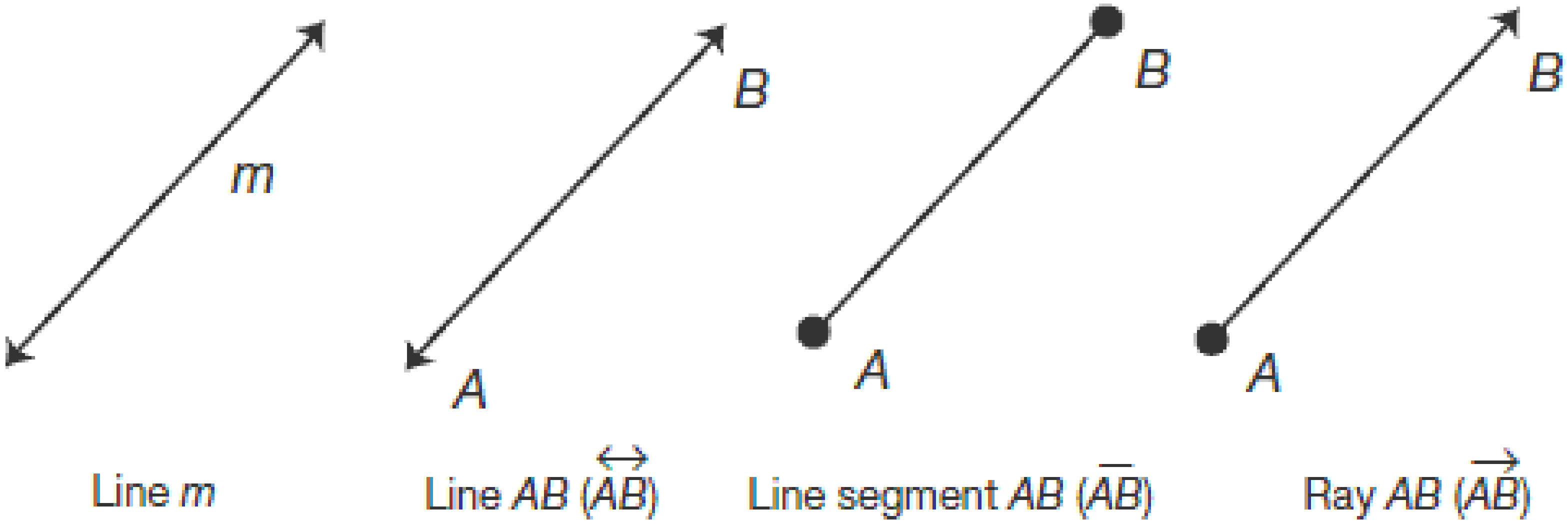
Recall: Double comparison notes

- Double comparison is tricky
- We need to compare against Epsilon value
  - Value might be problem dependent

```
int dcmp(double a, double b) {
    return fabs(a-b) <= EPS ? 0 : a < b ? -1 : 1;
}
```

Line - Ray - Segment



Src: https://sheridanmath.wikispaces.com/file/view/GEOMETRY_04.GIF/177066721/GEOMETRY_04.GIF

Line Equations

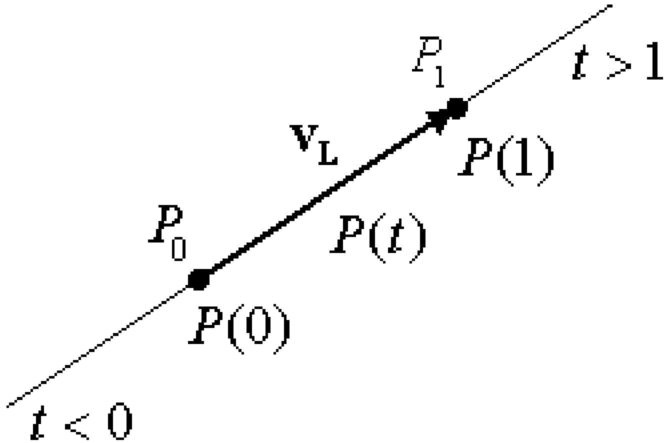
Type	Equation	Usage
Explicit 2D	$y = \mathrm{f}(x) = mx + b$	a non-vertical 2D line
Implicit 2D	$\mathrm{f}(x, y) = ax + by + c = 0$	any 2D line
Parametric	$P(t) = P_0 + t\,\mathbf{v_L}$	any line in any dimension

- Typically we avoid Explicit 2D one
- Even implicit one is not that frequent
    - Note: (a, b) is a vector that **perpendicular** to the line
- Parametric one is typically perfect representation

Src: http://geomalgorithms.com/a02-_lines.html

Parametric equation

$$
\begin{aligned}
P(t) &= P_0 + t\,\mathbf{v_L} \\
&= P_0 + t\left(P_1 - P_0\right) \\
&= \left(1-t\right)P_0 + t\,P_1
\end{aligned}
$$



- t = {0-1} : where some point on segment
  - e.g. t = 0.5, then the target point in the middle of segment
- if t < 0 then P(t) is outside the segment on the P0 side
- if t > 1 then P(t) is outside on the P1 side.
- f
- E.g. P0 (2, 0) and p1(10, 0) ⇒ then v = (8, 0)
- Then, P(t) = (2, 0) + t(8, 0)
- E.g. P(0.5) = (2, 0) + 0.5 * (8, 0) = (6, 0)

Line Equations: Conversions

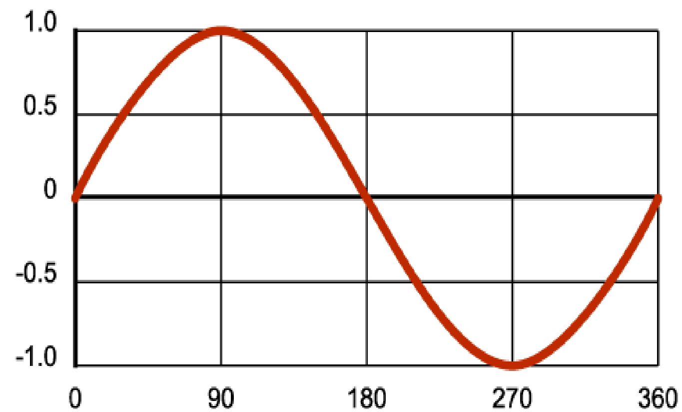
- Given $P_0 = (x_0, y_0)$ $P_1 = (x_1, y_1)$
  - Then $(y_0 - y_1)x + (x_1 - x_0)y + (x_0 y_1 - x_1 y_0) = 0$
  - One can convert that later format to easily to mx+b
  - also remember m = (y0-y1)/(x0-x1) so easy to convert
- Line L makes an **angle theta** with the x-axis
  - Then $-\sin(\theta)x + \cos(\theta)y + (\sin(\theta)x_0 - \cos(\theta)y_0) = 0$
  - And $P(t) = (x_0 + t\cos\theta, y_0 + t\sin\theta)$
- Little more readings
  - Link1 Link2 Link3

Is point C on line A - B

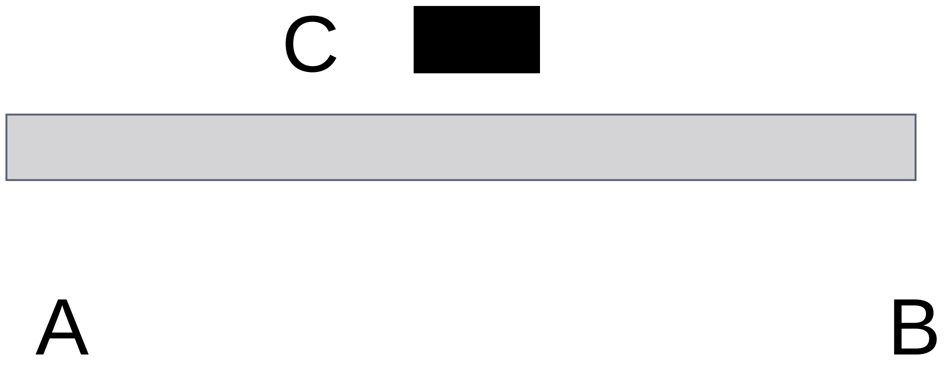
A C B

- Think about vector from A->C and from A->B, these vectors has **zero angle** between them
- Recall: **Cross product** between 2 vectors is ZERO if angle is ZERO = sin (0) = sin(180)
- Note: If 3 points are on a line, they are called collinear points

```
bool isCollinear(point a, point b, point c) {
    return fabs( cp(b-a, c-a) ) < EPS;
}
```

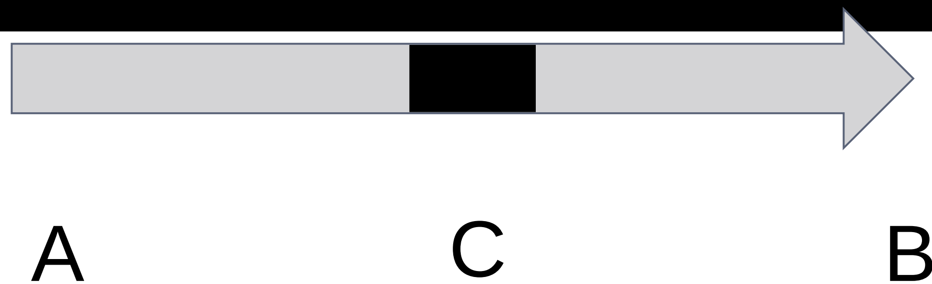


Is point C above line A - B



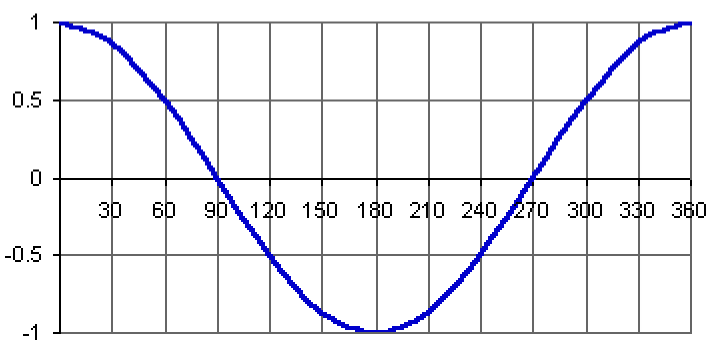
- Again just use the **Cross product** between 2 vectors
- If 180 > angle > 0 ⇒ sign(angle) => positive
- So just see if cp > EPS
- One can also use the **slope equation** to make check (y = mx+c)
- Or if have equation (ax+by+c = 0), convert to slope first, then check it

Is point C on ray A - B

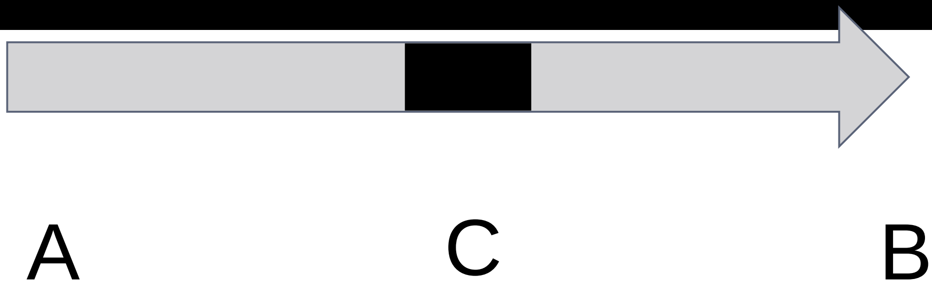


- Point on ray IFF it is on line
- If it is on the line: is in A->B direction or B->A direction ? Check if angle is **acute**
- Recall **dot product** between v1 and v2 > 0 IF angle is acute

```
bool isPointOnRay(point a, point b, point c) {
    if(!isCollinear(a, b, c))
        return false;
    return dcmp(dp(b-a, c-a), 0) == 1;
}
```



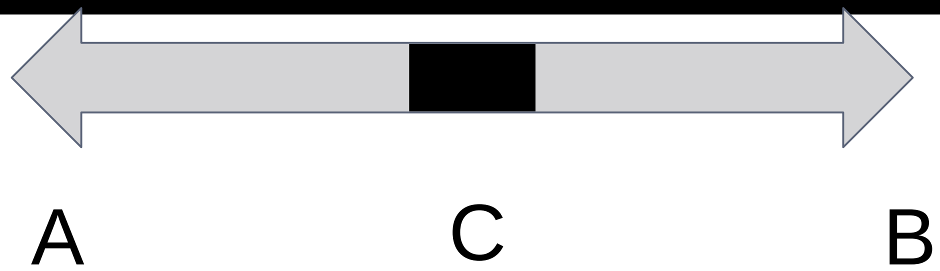
Is point C on ray A - B



- Can we do it in another way?
- Yes, normalize vector A->B and vector A->C
  - E.g. normalizing (0, 10) => (0, 1)
- If 2 normalized vectors are same = on ray

```
bool isPointOnRay(point a, point b, point c) {
    if(length(c-a) < EPS)
        return true;    // 3 points are same (x, y)
    return same( normalize(b-a), normalize(c-a) );
}
```

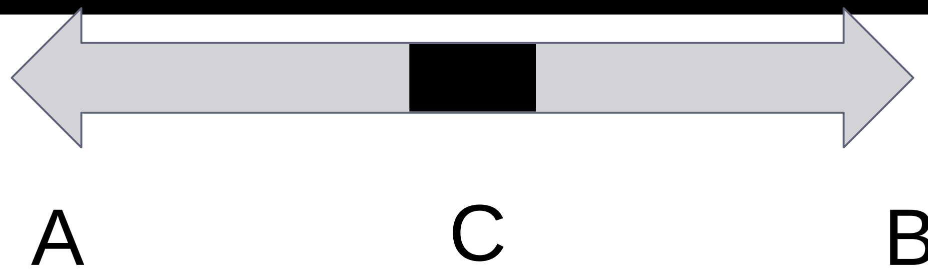
Is point C on segment A - B



- Point on segment if it is on 2 rays: A->B and B->A
- Below code can be written in more efficient way

```
bool isPointOnSegment(point a, point b, point c) {
    return isPointOnRay(a, b, c) && isPointOnRay(b, a, c);
}
```

Is point C on segment A - B

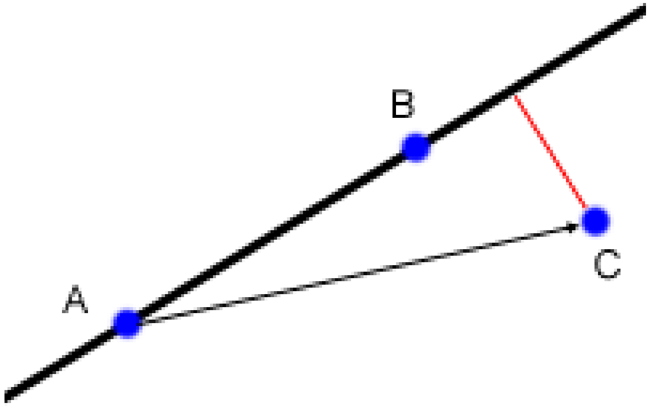


- Can we do it in another way? Yes several other ways
- E.g.: Distance from A to C + Distance from C to B = All Distance

```
bool isPointOnSegment(point a, point b, point c) {
    double acb = length(a-b), ac = length(a-c), cb = length(b-c);
    return dcmp(acb-(ac+cb), 0) == 0;
}
```

Point C distance to Line A-B

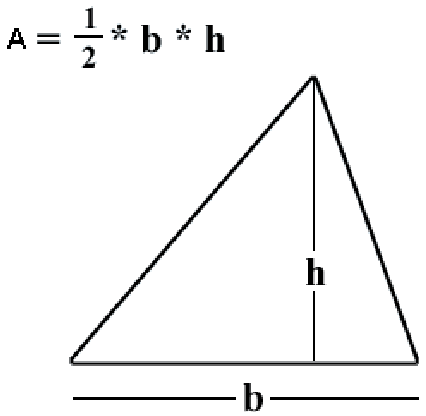
- How to compute the Line-Point Distance?
  - Notice C can be above A-B or much far left or right
  - Remeber, line extends infinitely

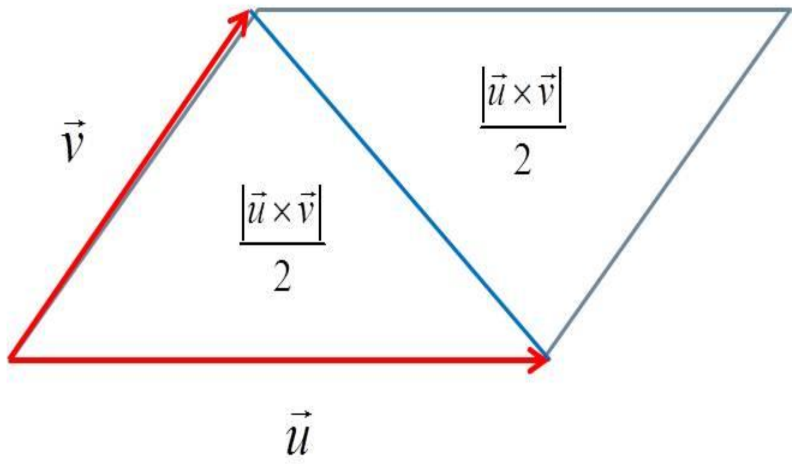


Src: http://community.topcoder.com/i/education/geometry04.png

Point C distance to Line A-B

- We need to recall 2 things first:





Src: http://blogs.jccc.edu/rgrondahl/files/2012/02/trianglefrompara.jpg http://www.stepbystep.com/wp-content/uploads/2013/02/How-to-Find-the-Height-of-a-Triangle-with-Base-and-Area.gif

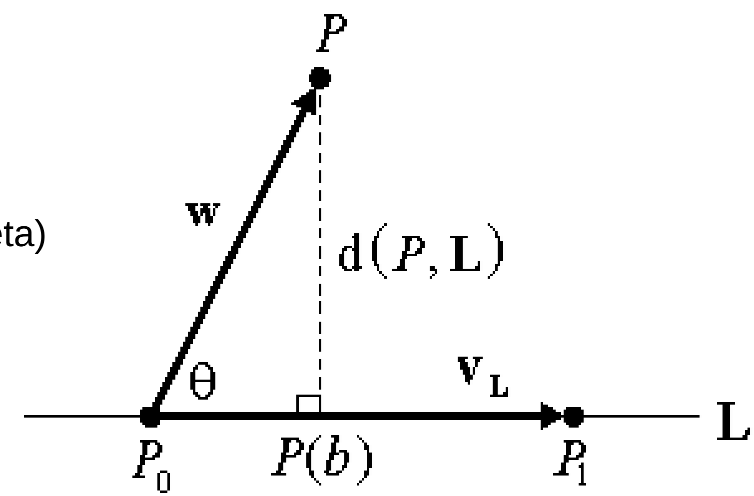
Point C distance to Line A-B

- Assume line A-B and Point C
  - Distance = Perpendicular Height from C to A-B
  - Then A-B is the base
- Area = ½ base x height
  - Height = 2 x Rectangle Area / base
  - 2 x Rectangle Area = Cross Product of (A->B and A->C)

```
double distToLine(point a, point b, point c) {
    double dist = cp(b-a, c-a) / length(a-b);
    return fabs(dist);
}
```

Point C distance to Line A-B

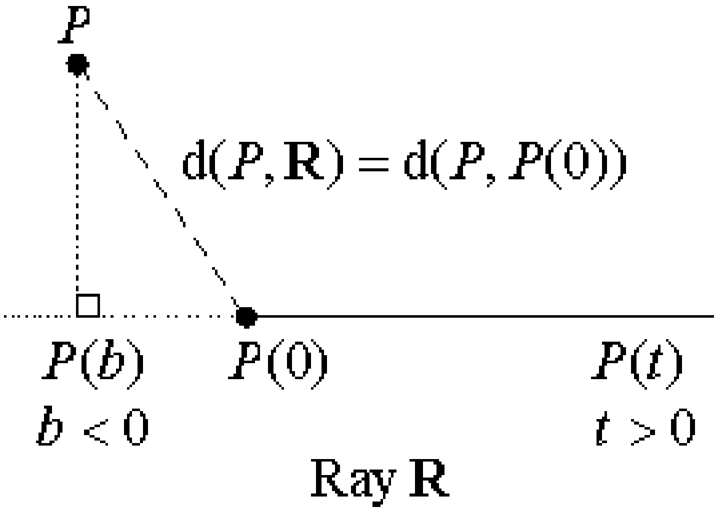
- Another way using **parametric** method
- If we know the projection point n L
- We can compute p0 - p(b) = distance
- b = partial distance / total distance
- From triangle equ: d(p0, p(b)) = w cos(theta)
- This value is actually the dot product
- However, previous way is shorter / easier
- *But we will use b calculation later*

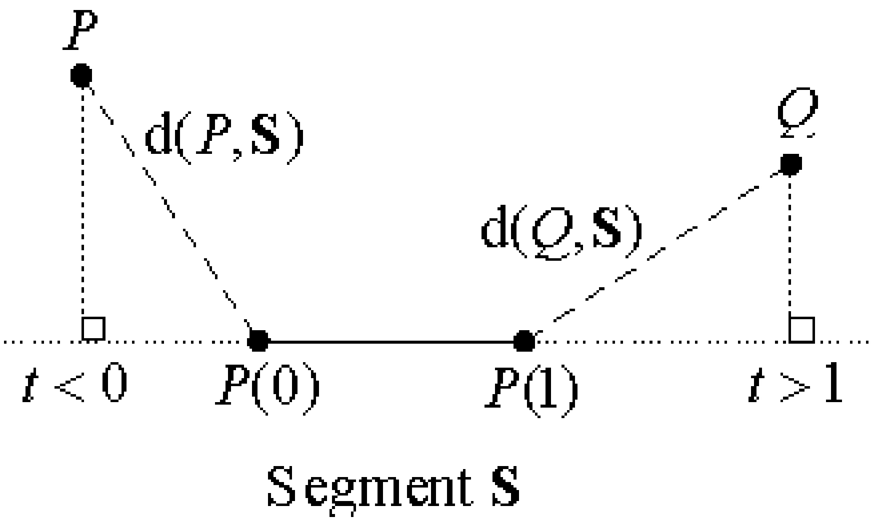


So, with $\mathbf{v}_{\mathbf{L}} = (P_1 - P_0)$ and $\mathbf{w} = (P - P_0)$, we get that:

$$b = \frac{\mathrm{d}\left(P_0, P(b)\right)}{\mathrm{d}\left(P_0, P_1\right)} = \frac{|\mathbf{w}|\cos\theta}{|\mathbf{v}_{\mathbf{L}}|} = \frac{\mathbf{w} \cdot \mathbf{v}_{\mathbf{L}}}{|\mathbf{v}_{\mathbf{L}}|^2} = \frac{\mathbf{w} \cdot \mathbf{v}_{\mathbf{L}}}{\mathbf{v}_{\mathbf{L}} \cdot \mathbf{v}_{\mathbf{L}}}$$

Point C distance to Segment A-B





Src: http://geomalgorithms.com/a02-_lines.html

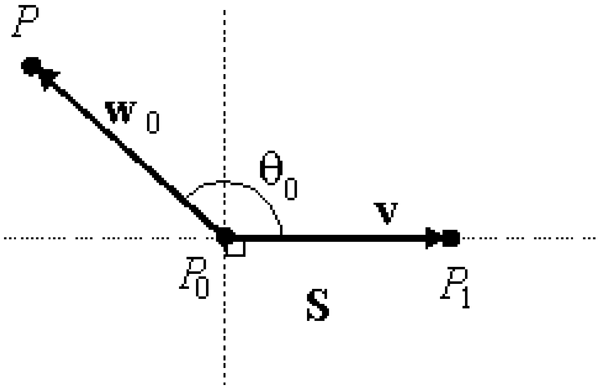
Point C distance to Segment A-B

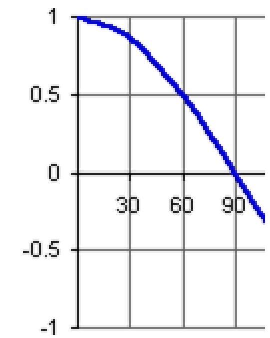
$$\mathbf{w}_0 = P - P_0 \text{ and } \theta_0 \in \left[-180^\circ, 180^\circ\right]$$

$$\mathbf{w}_0 \bullet \mathbf{v} \le 0$$

$$\Leftrightarrow \left|\theta_0\right| \ge 90^\circ$$

$$\Leftrightarrow \mathrm{d}\left(P, \mathbf{S}\right) = \mathrm{d}\left(P, P_0\right)$$



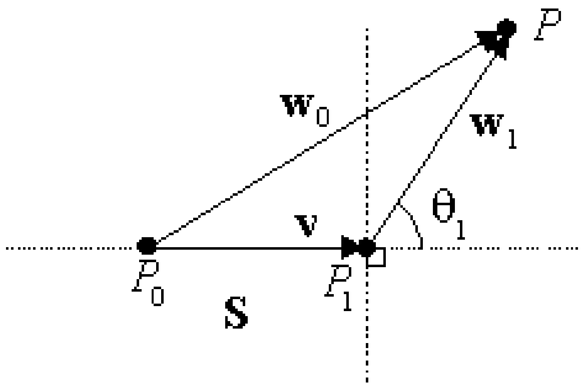


$$\mathbf{w}_1 = P - P_1 \text{ and } \theta_1 \in \left[-180^\circ, 180^\circ\right]$$

$$\mathbf{w}_1 \bullet \mathbf{v} \ge 0 \Leftrightarrow \mathbf{w}_0 \bullet \mathbf{v} \ge \mathbf{v} \bullet \mathbf{v}$$

$$\Leftrightarrow \left|\theta_1\right| \le 90^\circ$$

$$\Leftrightarrow \mathrm{d}\left(P, \mathbf{S}\right) = \mathrm{d}\left(P, P_1\right)$$



Note: $\mathbf{w}_0 = \mathbf{v} + \mathbf{w}_1$

Src: http://geomalgorithms.com/a02-_lines.html

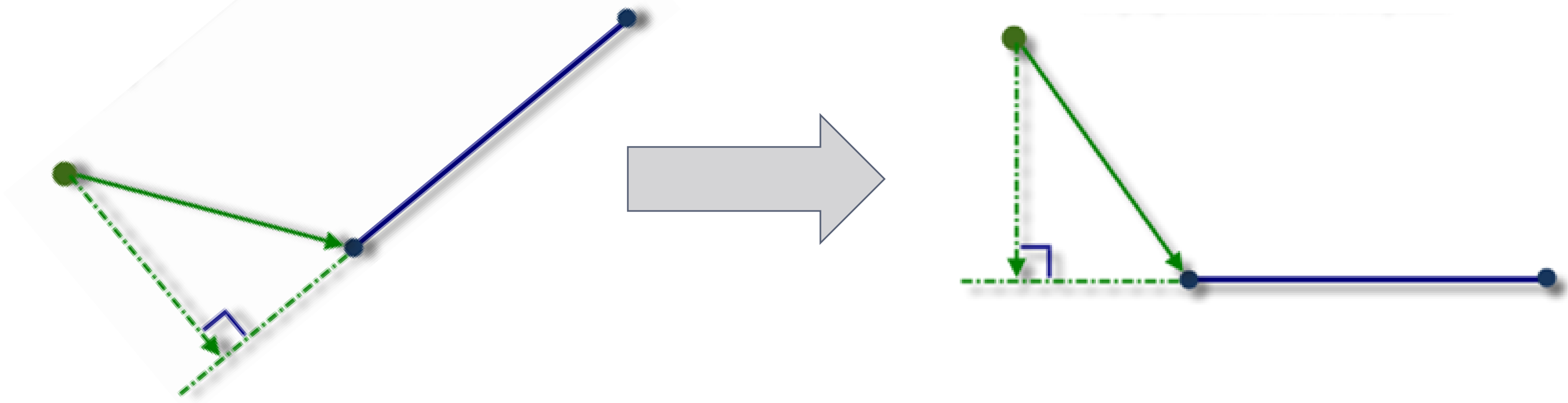
Point C distance to Segment A-B

```
//distance from point p2 to segment p0-p1
double distToSegment( point p0, point p1, point p2 ) {
    double d1, d2;
    point v1 = p1-p0, v2 = p2-p0;

    if( (d1 = dp(v1, v2) ) <= 0)
        return length(p2-p0);
    if( (d2 = dp(v1, v1) ) <= d1)
        return length(p2-p1);

    double t = d1/d2;   // the on segment
    return length(p2 - (p0 + v1*t) );
}
```

$$b = \frac{d\left(P_0, P(b)\right)}{d\left(P_0, P_1\right)} = \frac{|\mathbf{w}|\cos\theta}{|\mathbf{v_L}|} = \frac{\mathbf{w} \cdot \mathbf{v_L}}{|\mathbf{v_L}|^2}$$

Point C distance to Segment A-B

- Assume you are in the contest...forgot calculations of such distance
- Remember, sometimes, there are many ways to compute same things in geometry
- Let’s do it using just rotation and length functions!
- Let’s rotate the segment so that it be on x-axis
- Now our target point is just above x-axis, so we can use target.X to know its location



Src: http://desktop.arcgis.com/en/arcmap/10.3/tools/analysis-toolbox/GUID-859A69DD-F5D8-40D5-B7F6-89571F7F17C0-web.png

Point C distance to Segment A-B

```
int main() {
    //point p0(50, 50), p1(100, 100), p2(500, 17);  // right
    //point p0(50, 50), p1(100, 100), p2(-70, 17);  // left
    point p0(50, 50), p1(100, 100), p2(70, 100);    // above

    p1 -= p0, p2 -= p0, p0 = 0; // shift to origin, so cancel p0

    double ang = angle(p1);
    p1 = rotateO(p1, -ang);     // p0-p1 now is on the x axis
    p2 = rotateO(p2, -ang);     // using p2.x, you can know distance trivially

    if (p2.X <= 0)              cout<<length(p2)    <<" left of segment";
    else if (p2.X >= p1.X)      cout<<length(p2-p1) <<" right of segment";
    else                        cout<<fabs(p2.Y)    <<" above segment\n";

    return 0;
}
```