Physics Programming

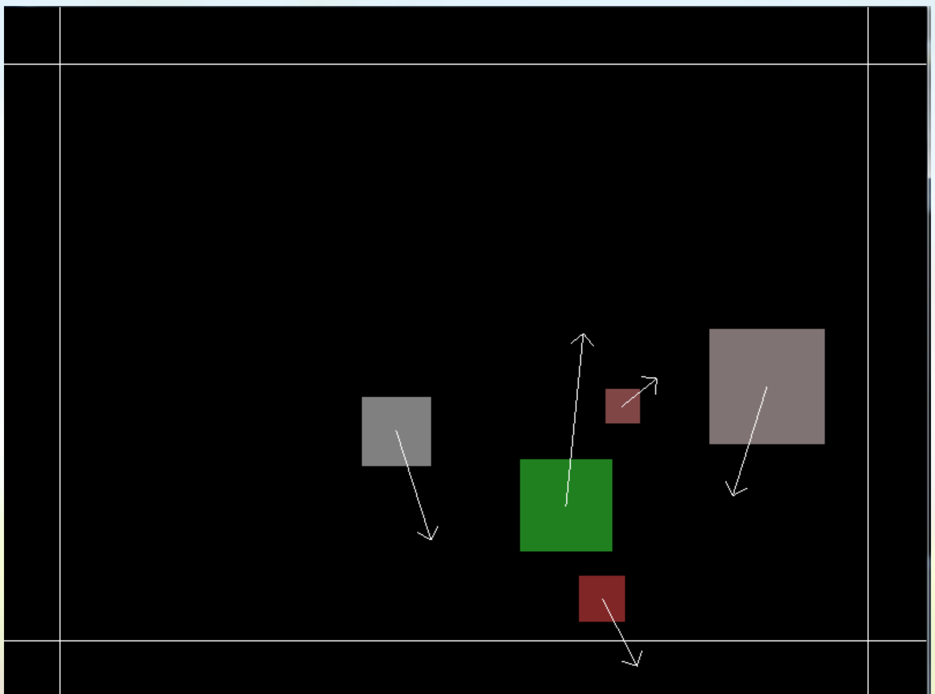
The Dot Product and Angled Lines

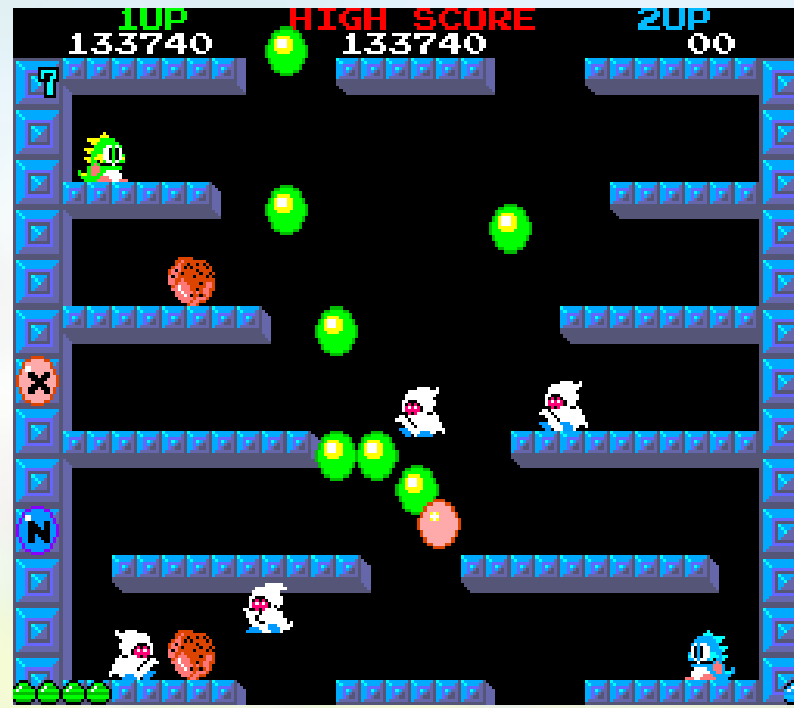
Slides & lesson materials by Hans Wichman & Paul Bonsma

Image: http://communities.ptc.com/people/DanMarotta

Previous Lecture

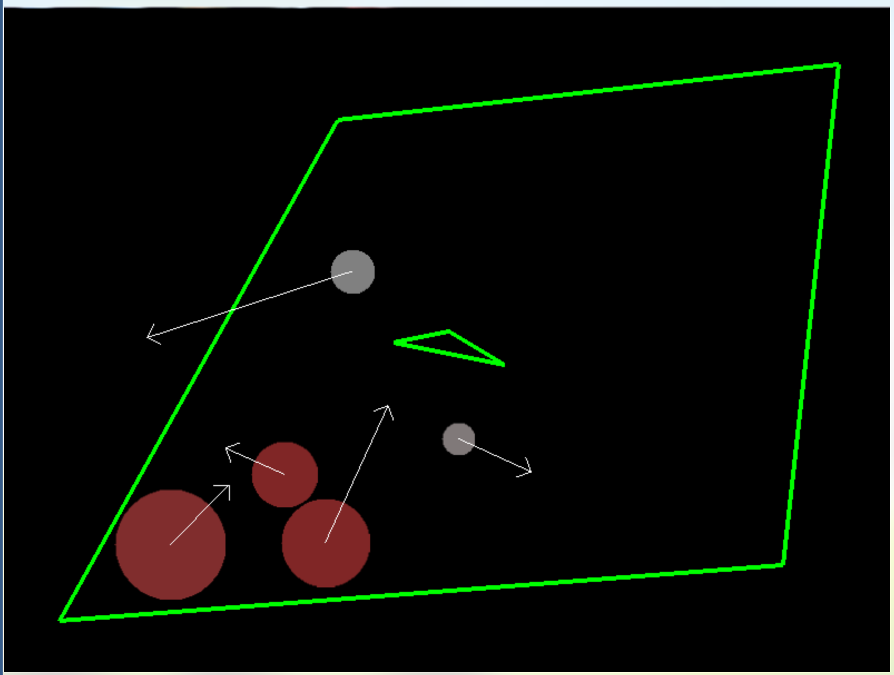
- Last week everything was horizontal or vertical





This Lecture

- This week: collisions between a ball and angled lines



Grading Criteria

	Insufficient	Sufficient	Good	Excellent
Vectors and Unit Testing (20%)	**4 pts** The Vec2 struct is not consistently used, basic functionality missing, or not all of its methods are unit tested.	**12 pts** The Vec2 struct is used for most vector operations, all the basic functionality is present (see the weekly assignments), and functional unit tests are included for most methods.	**16 pts** S + the Vec2 struct is used consistently for almost all vector operations, methods are implemented with code reuse and efficiency in mind, good unit tests are chosen and cover all Vec2 methods.	**20 pts** G + methods are all implemented efficiently in terms of code reuse or computation time. Useful extra functionality is added to the Vec2 struct.
Aiming and shooting (20%)	**4 pts** Aiming is not implemented correctly or the sprite rotation does not match the movement direction.	**12 pts** Aiming (+ shooting or moving) in the current direction, or aiming to a target is implemented, using a rotated sprite (without using the GXP Engine's methods such as Move)	**16 pts** S + Aiming in the current direction and aiming to a target are both implemented.	**20 pts** G + Advanced aiming functionality has been added. (Examples: leading a moving target, correctly aiming a gravity-influenced or bouncing projectile, timing fixed angle shots to hit a moving target.)
Collisions (30%)	**6 pts** Collision detection + resolve contains bugs, or is not included for angled lines, or no bouncing is included, or is not explained properly.	**18 pts** Correct collision detection + resolve (incl. bouncing / velocity reflection) on angled lines (without using the GXP Engine's methods such as MoveUntilCollision). The student can explain dot product applications in detail.	**24 pts** S + Correct point of impact calculation, correct collision with line segments (without using the GXP Engine's methods such as MoveUntilCollision).	**30 pts** G + Robust handling of advanced collisions (Examples: multiple moving objects following Newton's laws, combining gravity with sliding or rolling, kinematic objects such as moving platforms, or collision friction)

Today: final 3 *required* methods

Basics: last week

Today

Next week

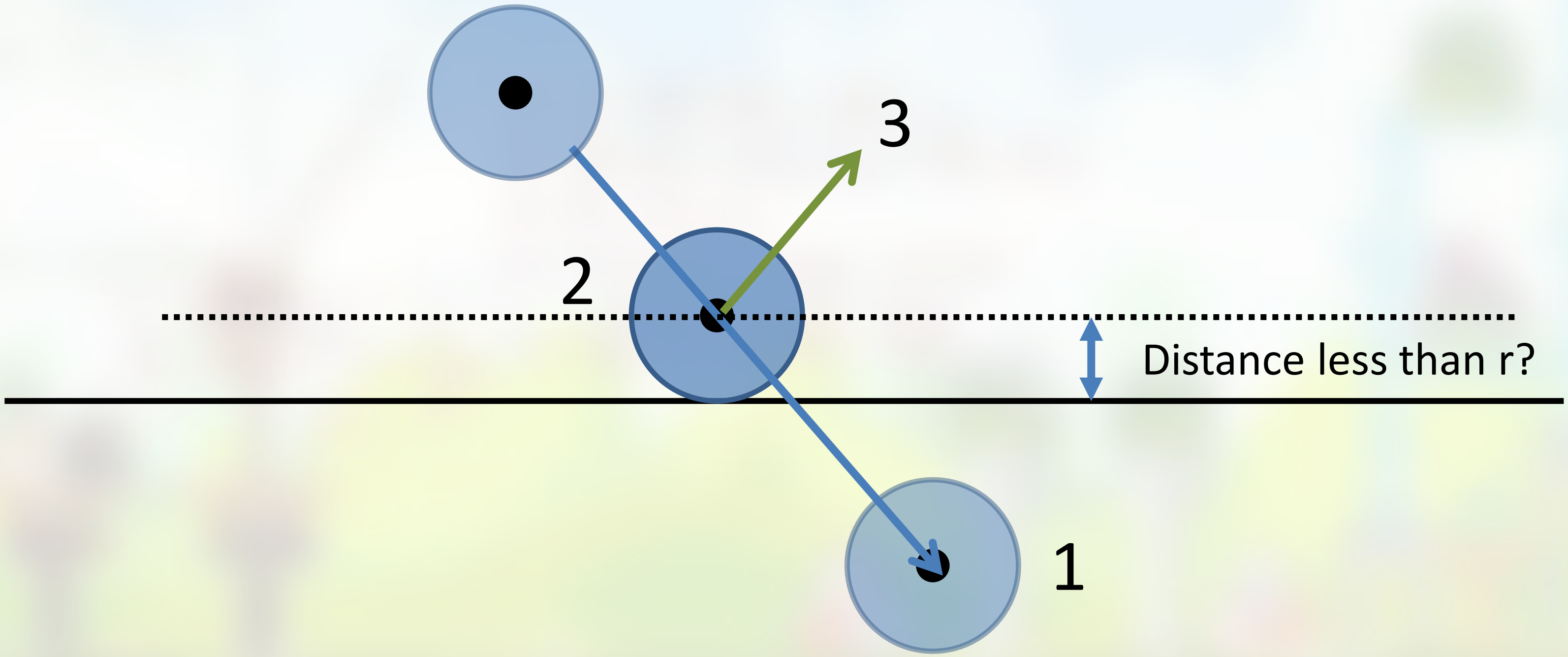
Lecture overview

- Bouncing off angled lines
  - Normals
  - Scalar & Vector projection
  - Dot product
  - Law of reflection
- Assignment 4

Bouncing off a line recap

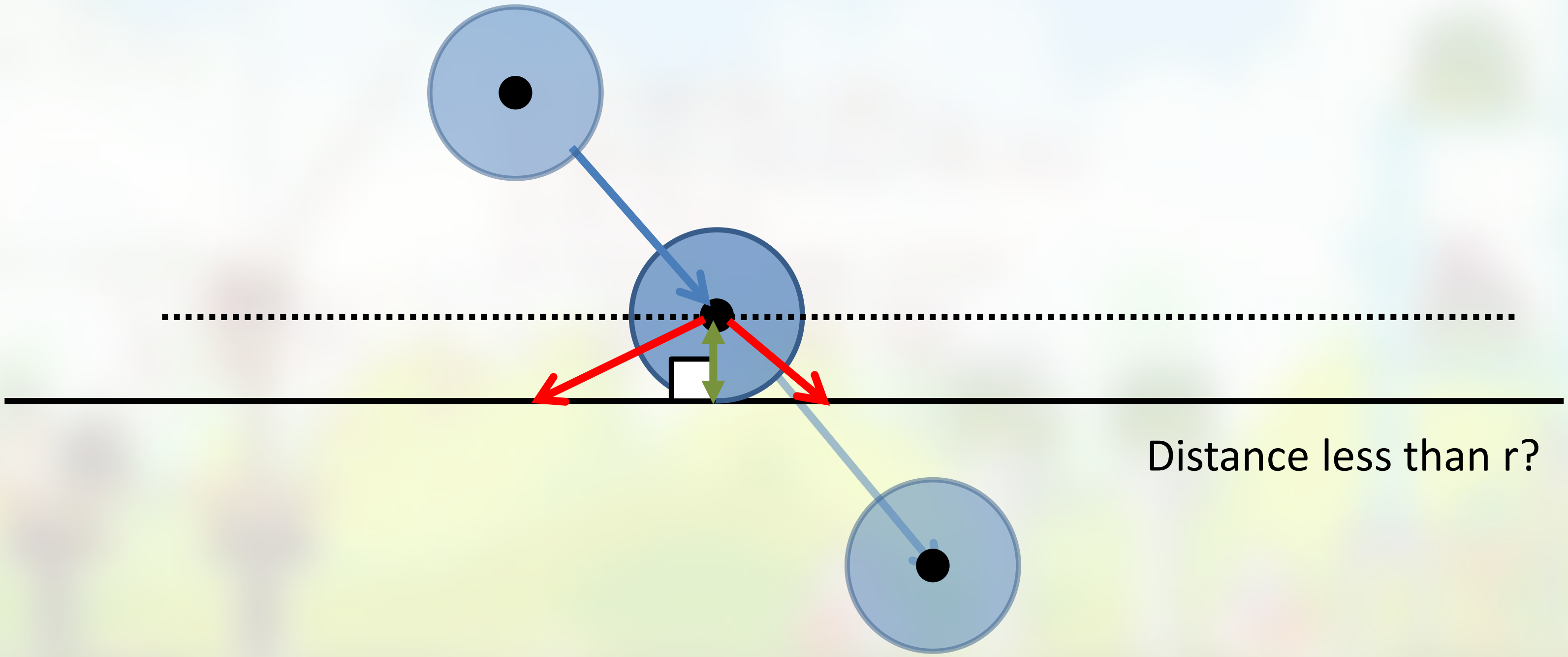
Bouncing recap

- Bouncing off a line:
  1. **Detect a hit**
  2. Move ball back onto the line
  3. Reflect velocity



Not just any distance: shortest distance

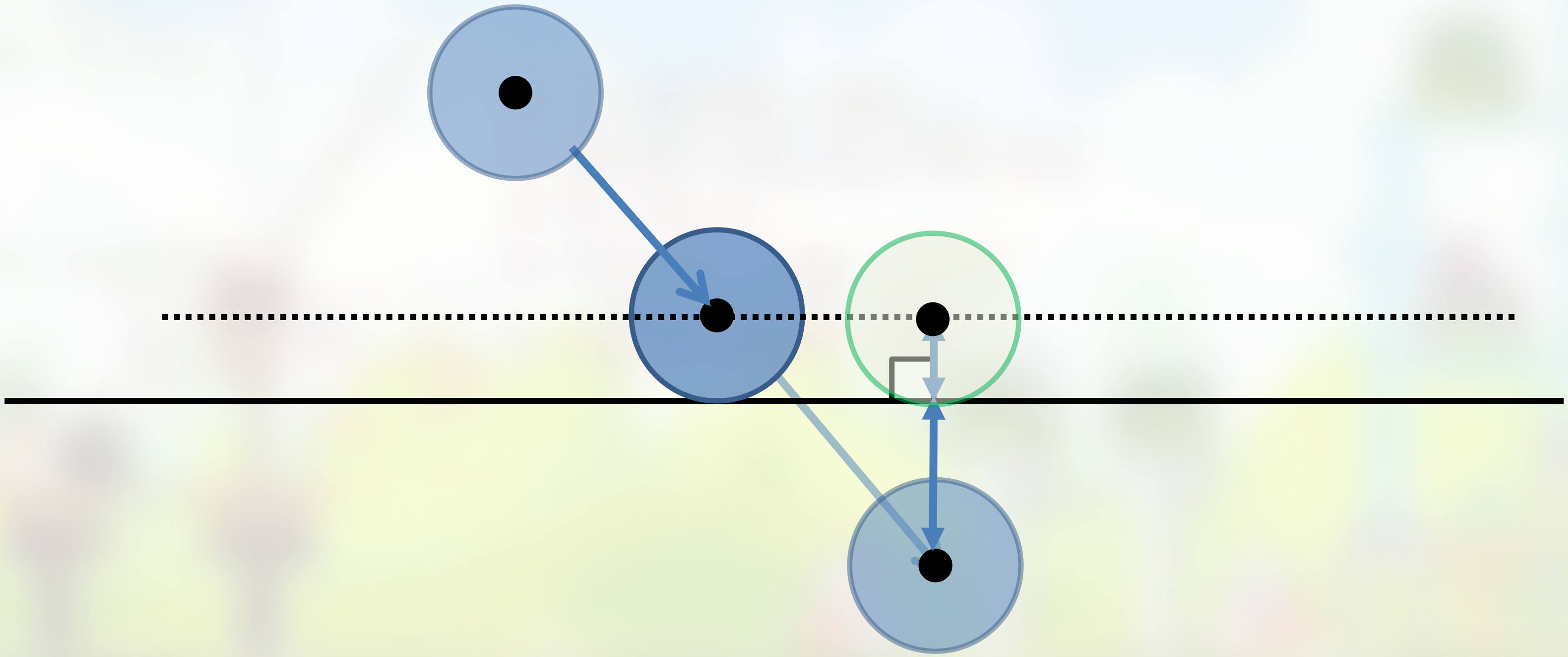
- Shortest distance: the distance measured on a line perpendicular (green) to the original line
- if (**shortest distance from ball to line** < ball radius) then there is a collision



Distance less than r?

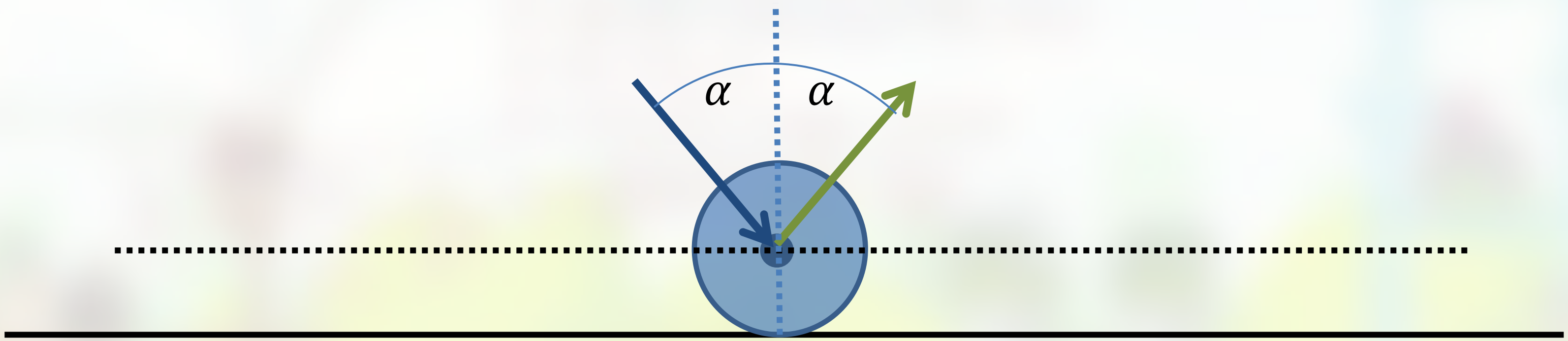
Move ball back onto the line

- Bouncing off a straight line:
  1. Detect a hit
  2. **Move ball back onto the line** (simple way: only up)
  3. Reflect velocity



Bouncing recap

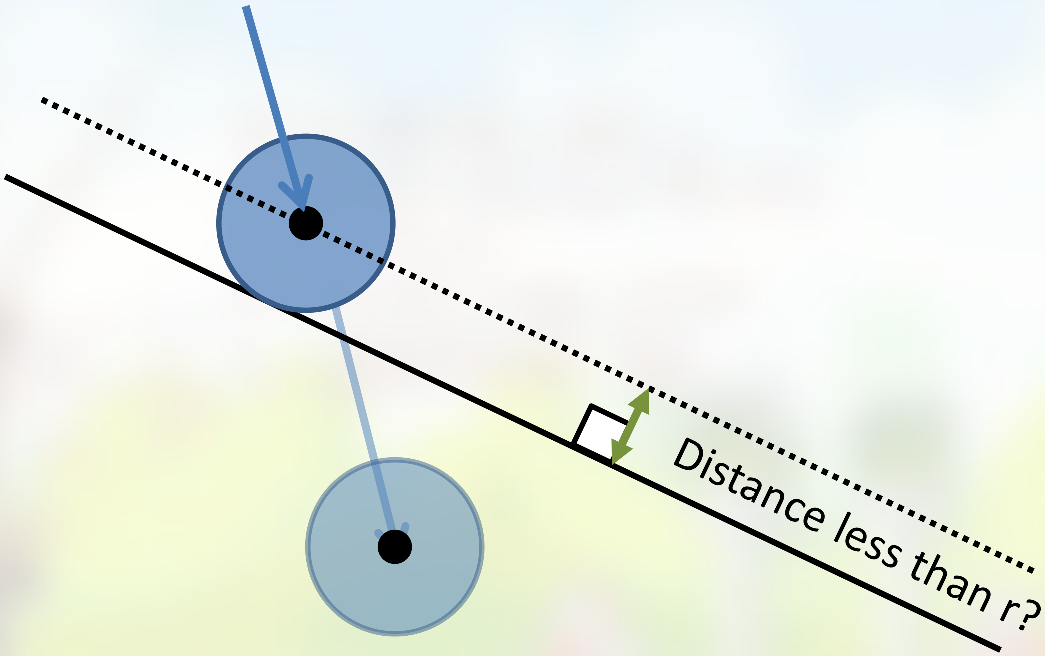
- Bouncing off a straight line:
  1. Detect a hit
  2. Move ball back onto the line
  3. **Reflect velocity**



Bouncing off angled lines

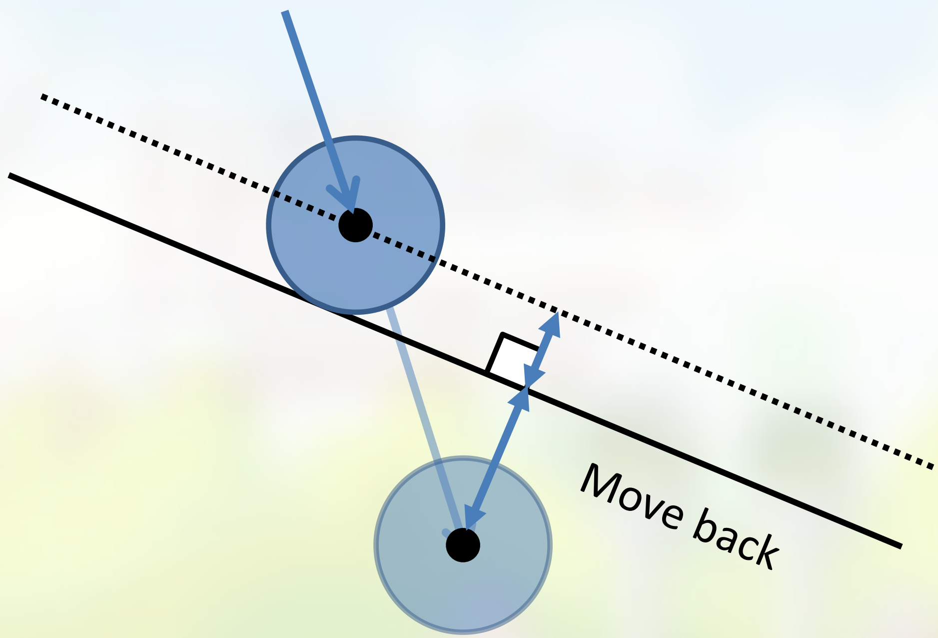
Bouncing off angled lines

- Bouncing off an angled line:
  1. **Detect a hit**
  2. Move ball back onto the line
  3. Reflect velocity



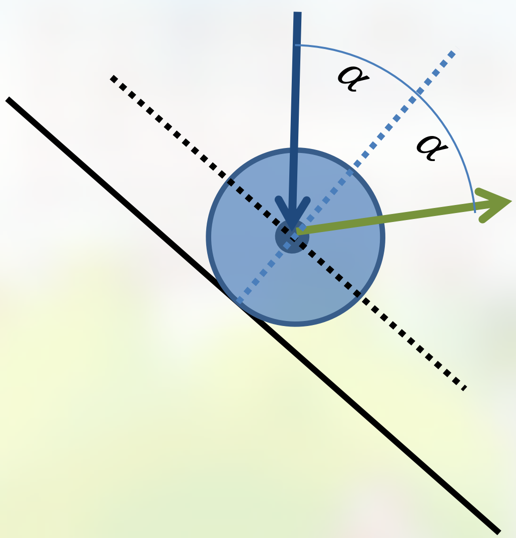
Bouncing off angled lines

- Bouncing off an angled line:
  1. Detect a hit
  2. **Move ball back onto the line** (the simple way)
  3. Reflect velocity



Bouncing off angled lines

- Bouncing off an angled line:
  1. Detect a hit
  2. Move ball back onto the line
  3. **Reflect velocity**



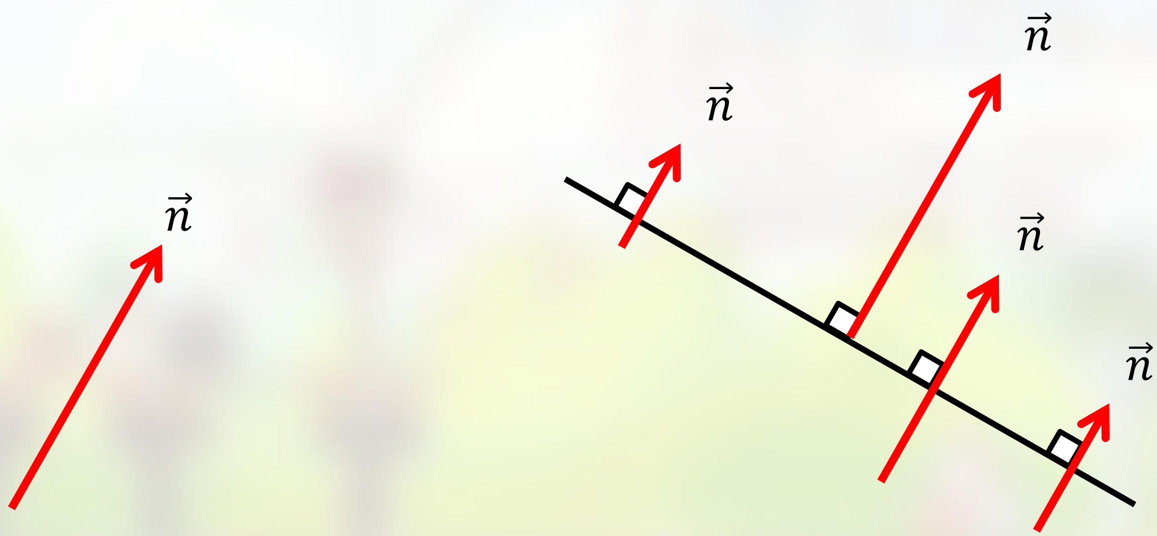
Conclusion

- Previously (easy case):
  - Just check / modify x or y coordinate
- But the general case requires us to:
  - Detect a hit using the distance along the *perpendicular* (the shortest distance)
  - Move ball back onto the line along the *perpendicular*
  - Reflect velocity using the *perpendicular*

Normal vectors / Vector normals

Normal vector

- The perpendicular of a line is called its **normal** vector
- Angle between a vector and its normal is **90**°
- Denoted with $\vec{n}$
- Important to understand:
  - Normal has no position, it only defines a direction
  - It is based on a line's slope not its position



Normal vs normalized

- A normalized vector != a normal vector
  - A **normalized** vector is a vector with **length 1**
  - A **normal** vector is a vector **perpendicular** to some other vector
- A normal vector can be normalized however:
  - A normalized normal vector is called a *unit normal vector*, denoted as $\hat{n}$, pointing in the direction of $\vec{n}$ with length 1
  - Often when talking about a normal, we assume it is normalized!

“Calculating” the normal

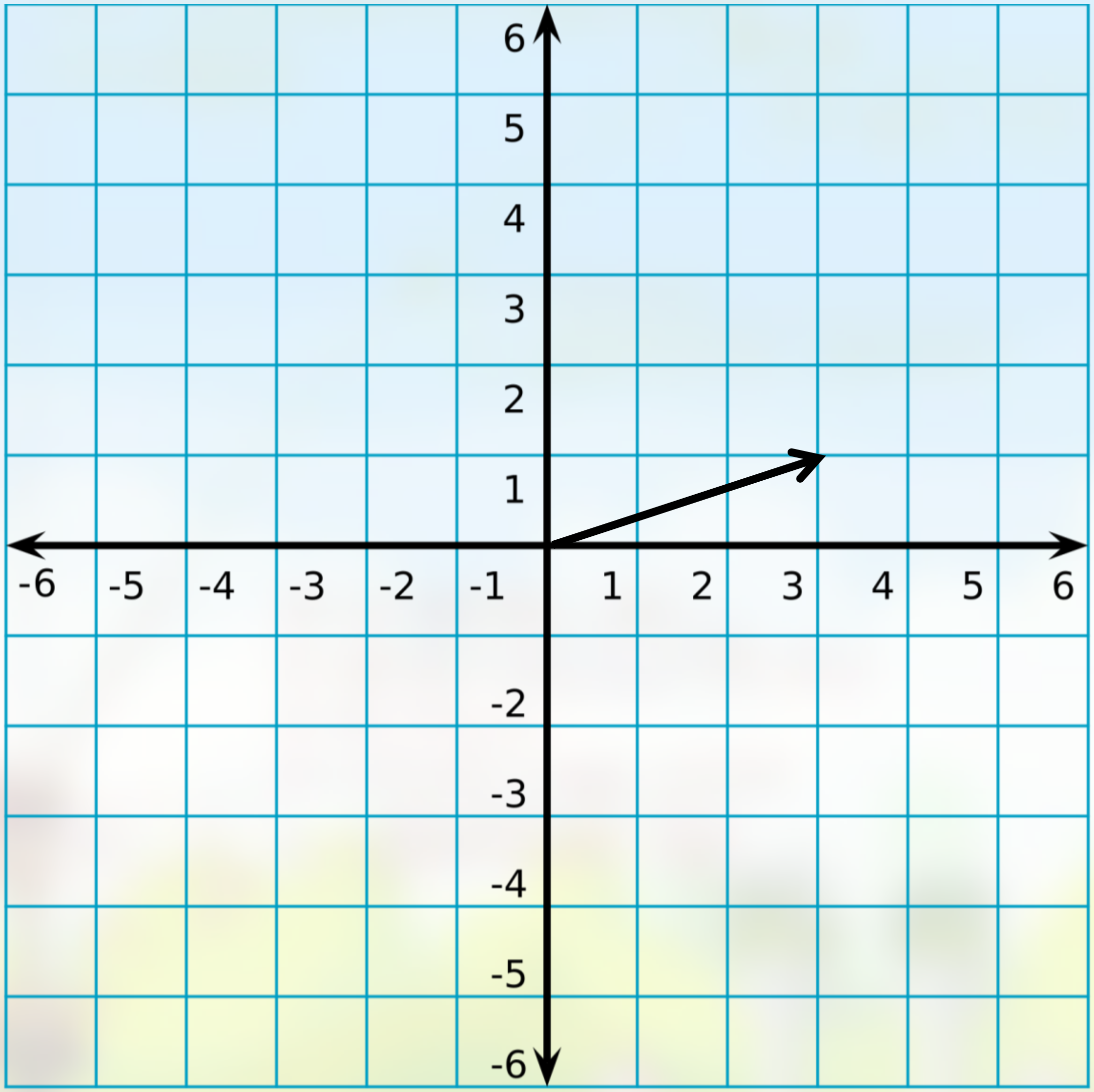
- Let $\vec{v}$ =(x,y) denote the “direction vector” of a line segment L (=end point – start point).
- Normal of L is then equal to $\vec{v}$ rotated by 90°.
- The formula for 2d rotation was:
  - $x' = x * \cos\alpha - y * \sin\alpha$
  - $y' = x * \sin\alpha + y * \cos\alpha$
- What is the normal of $\vec{v}$ = (x,y)?

  Cos (90°) = 0, Sin (90°) = 1, so:

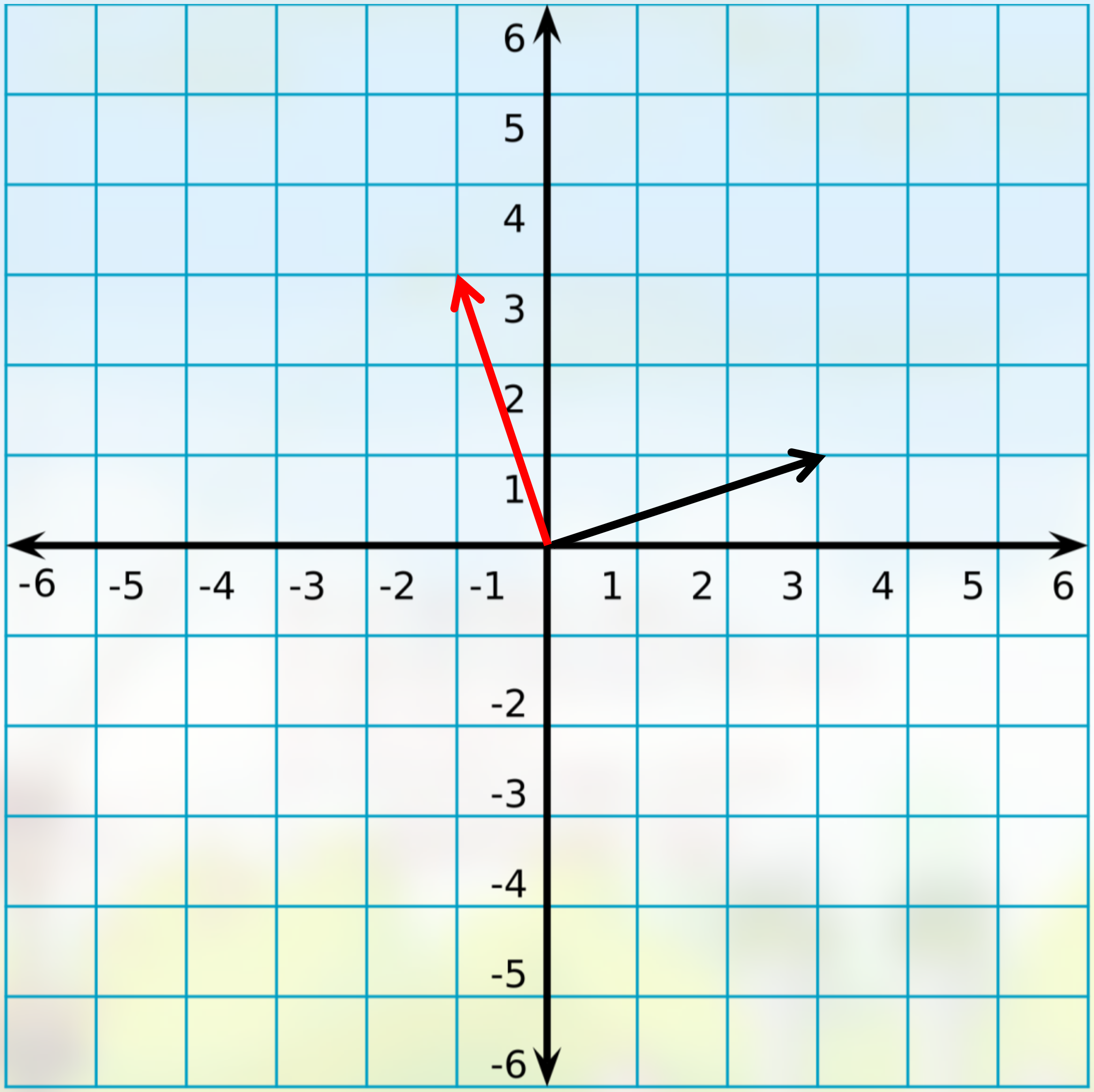
  $x' = x * 0 - y * 1 = -y$

  $y' = x * 1 + y * 0 = x$
- *So the normal of $\vec{v}$ =(x , y) is $\vec{n}$ = (-y , x)*

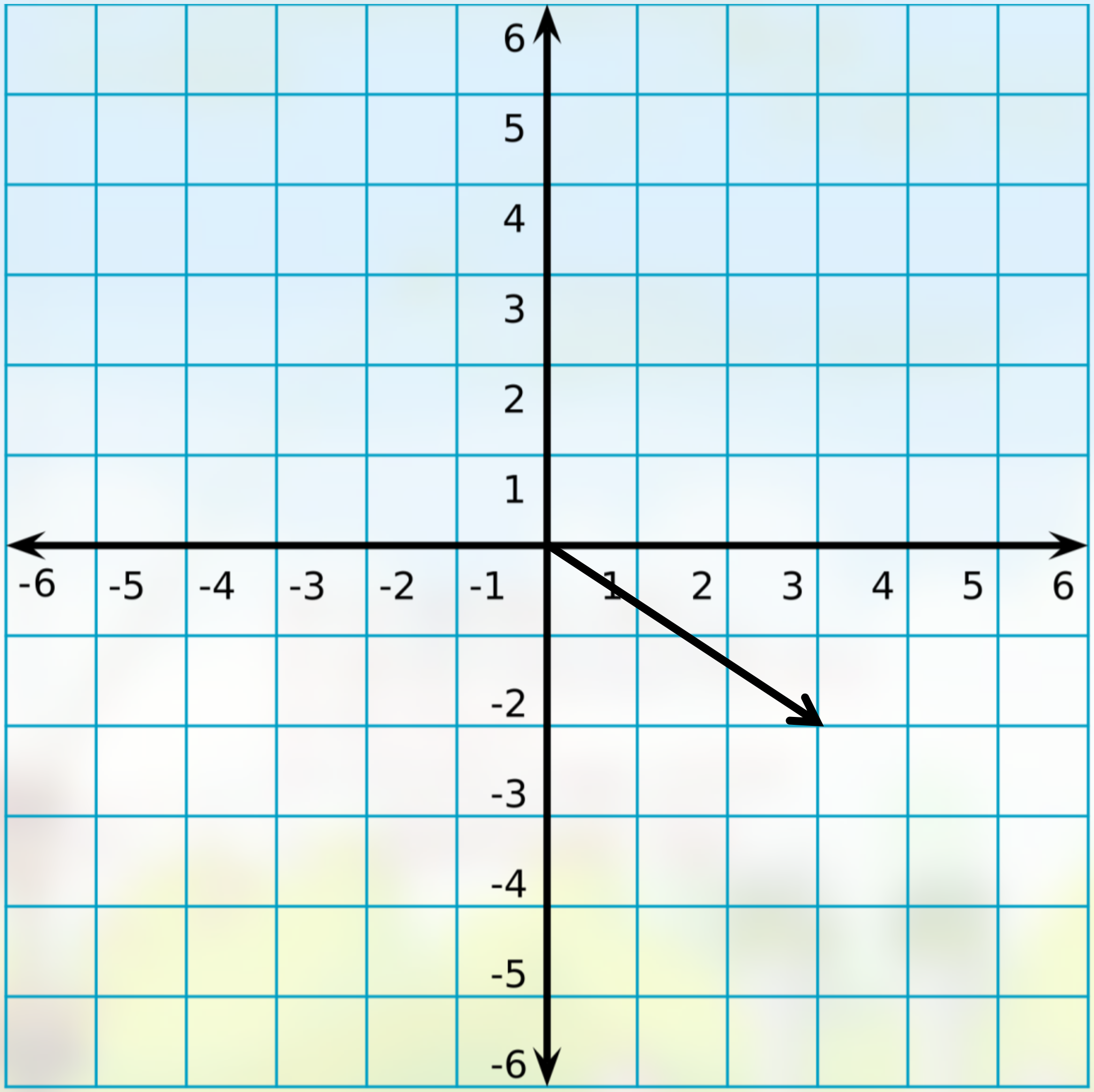
Example $\vec{v} = (3,1)$ ➔ $\vec{n} = (?,?)$



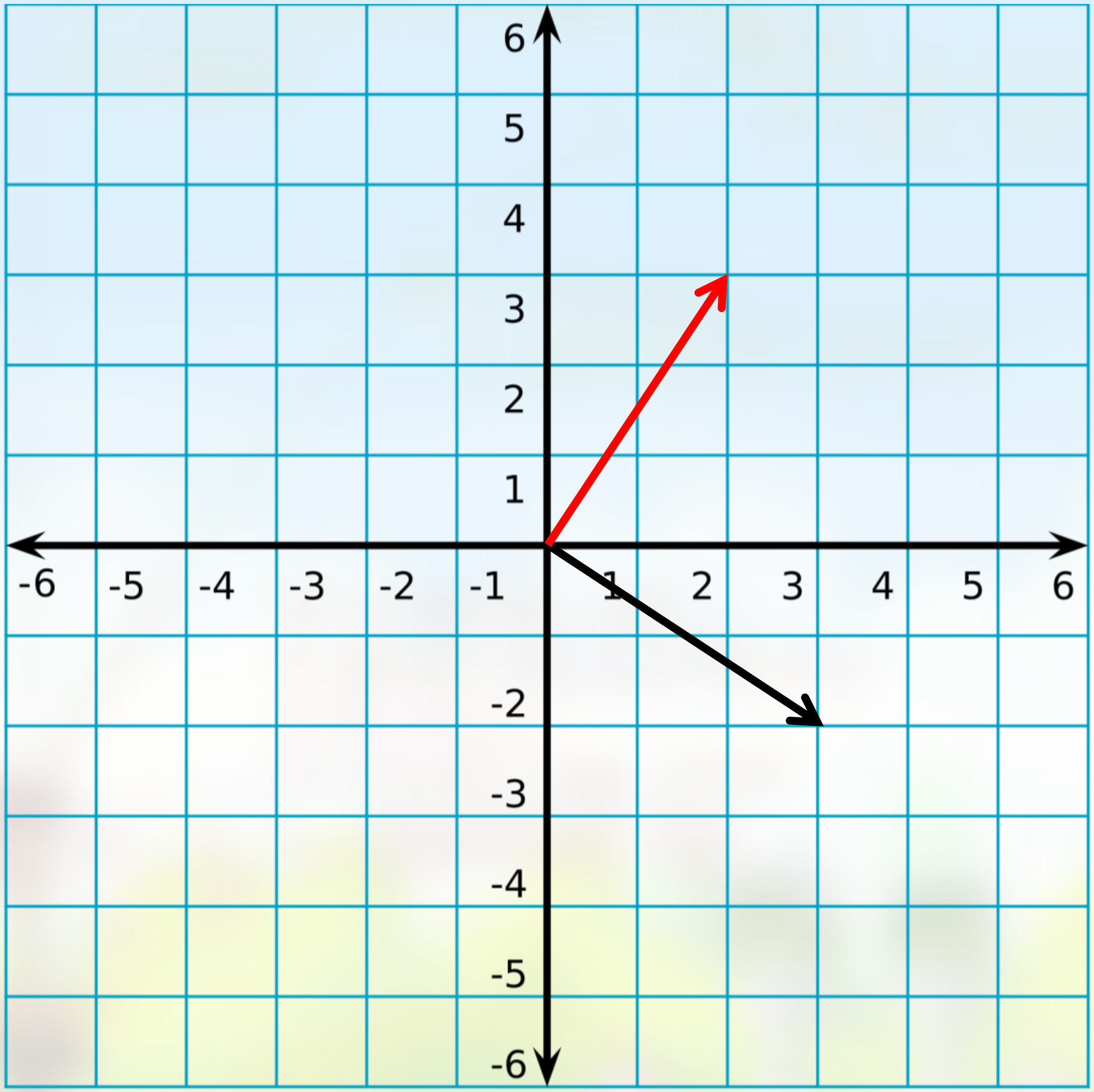
Example $\vec{v}$ = (3,1) ➔ $\vec{n}$ = (-1,3)



Example $\vec{v} = (3,-2)$ ➔ $\vec{n}$ = (?,?)



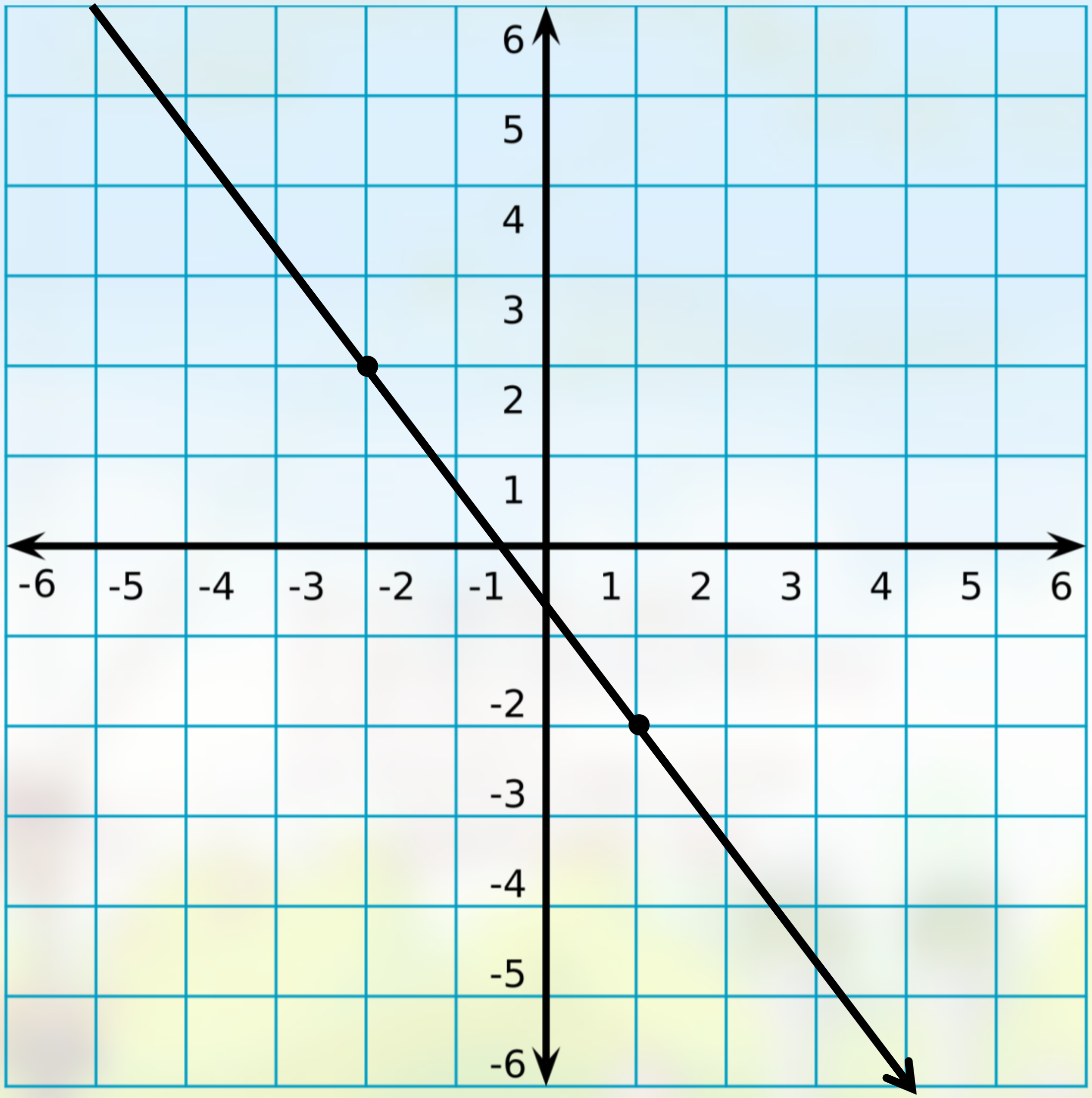
Example $\vec{v} = (3,\text{-}2) \rightarrow \vec{n} = (2,3)$



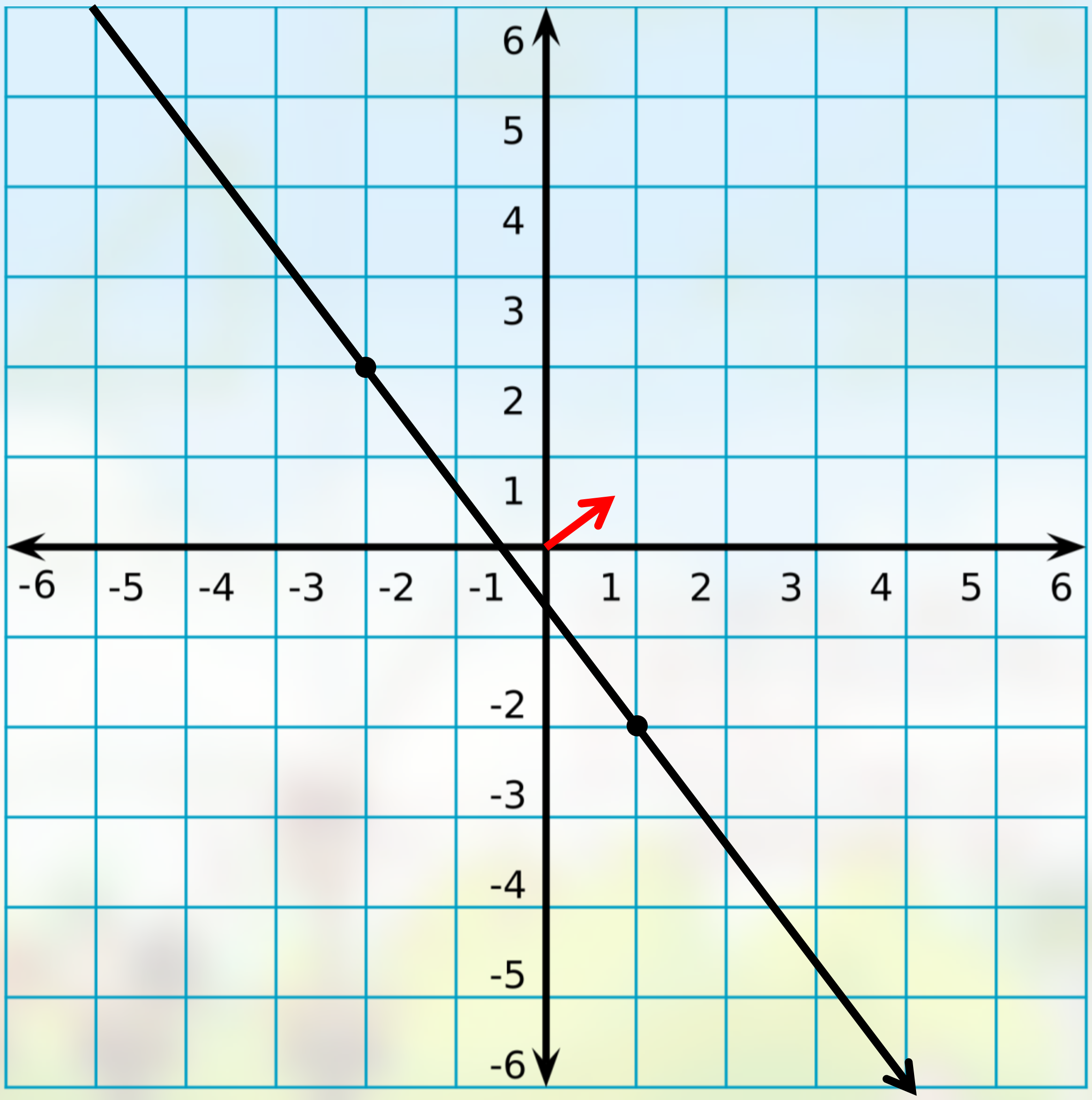
The "opposite" normal

- Is there an opposite normal?
  - same vector just negated: (y,-x)
  - By convention, the normal is +90° and not -90°
- In GXPEngine +90° is clockwise
- In standard Cartesian space +90° is counterclockwise

Exercise: calculate $\hat{n}$



Exercise: calculate $\hat{n}$



- $\vec{v} = (3, -4)$
- $\vec{n} = (4, 3)$
- $|\vec{n}|=5$
- $\hat{n} = \frac{\vec{n}}{|\vec{n}|} = \frac{(4,3)}{5}$
- $\hat{n} = (\frac{4}{5}, \frac{3}{5})$

Implementing Normal()

Implementing Normal()

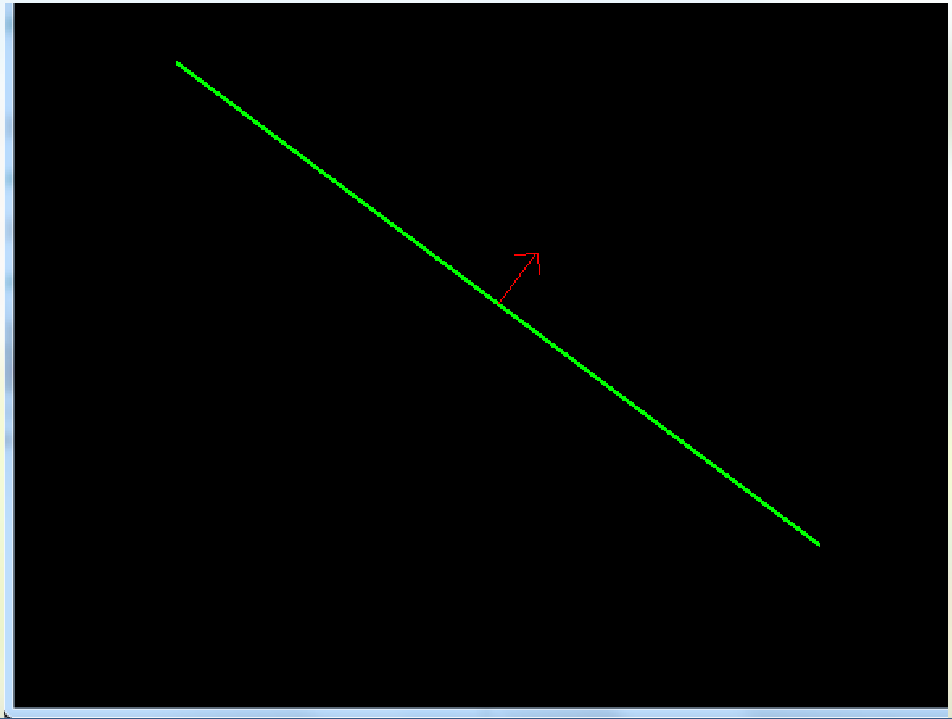
- Vec2.Normal() returns the *unit normal* of the current vector as a **new** vector:

```
public Vec2 Normal() {
        return ???;
}
```

- Tip: it's too small to draw without scaling!

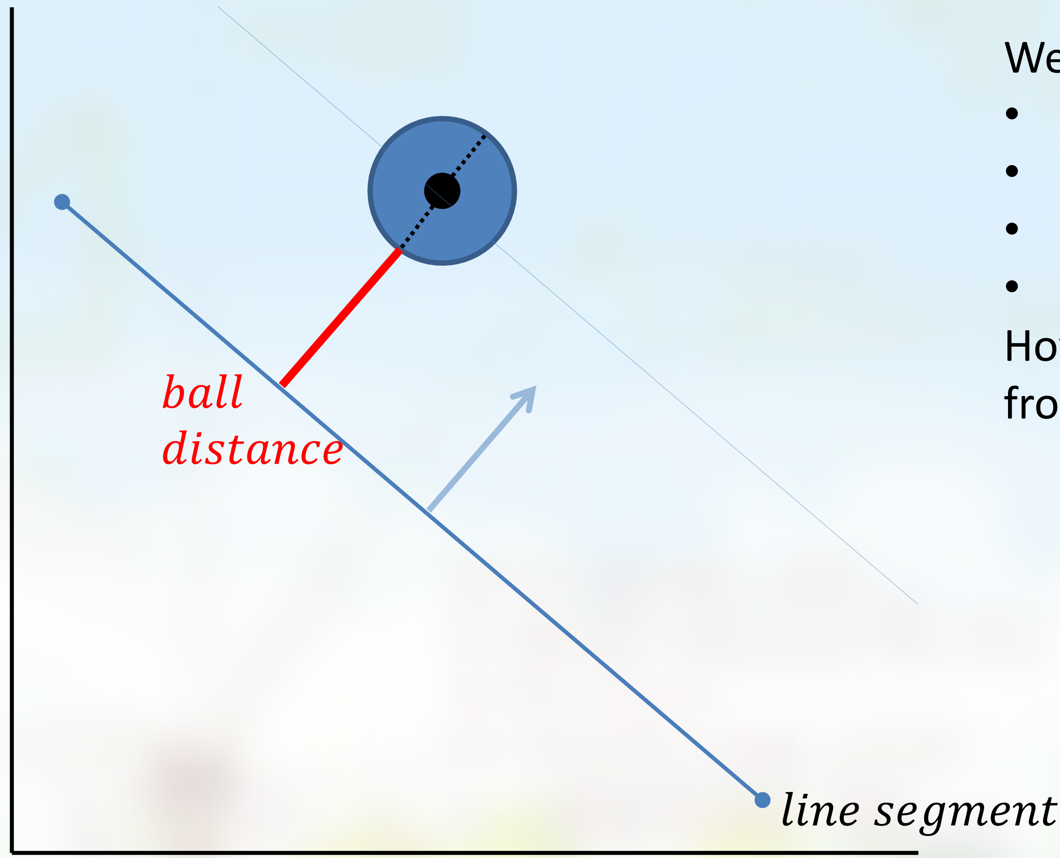
Implementing Normal()

- Test code provided:
  - +002_line_collision_detection
  - Uses NLineSegment: a Line with Normal displayed
- When Normal() is correctly implemented it should look like:



Ball/Line Distance

Ball/Line distance computation

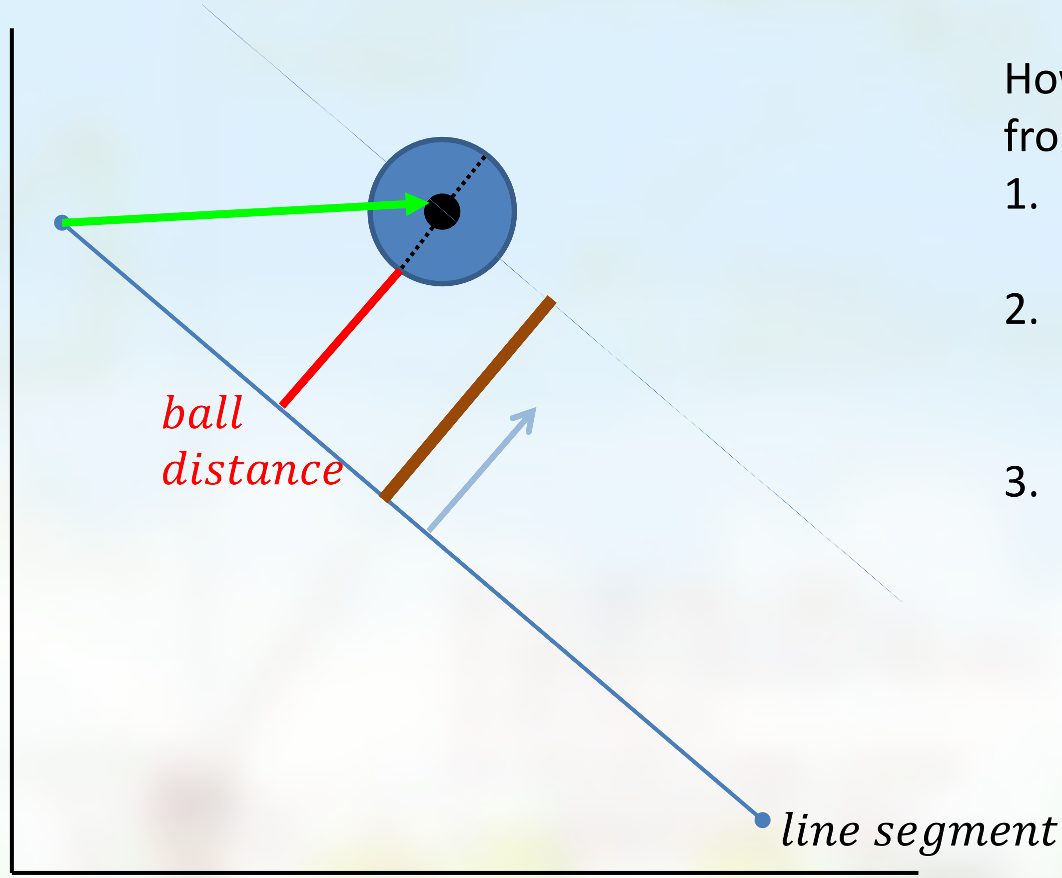


We know:

- Ball position
- Ball radius
- Two points on a line
- Line normal

How to compute the distance from ball to line?

Ball/Line distance computation



How to compute the distance from ball to line?

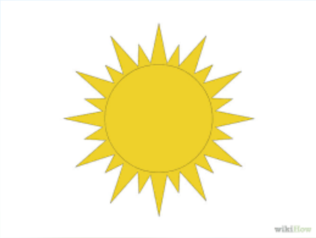
1. Compute “difference vector” (green)
2. Use “scalar projection” to project onto normal → gives length of brown line
3. Subtract radius

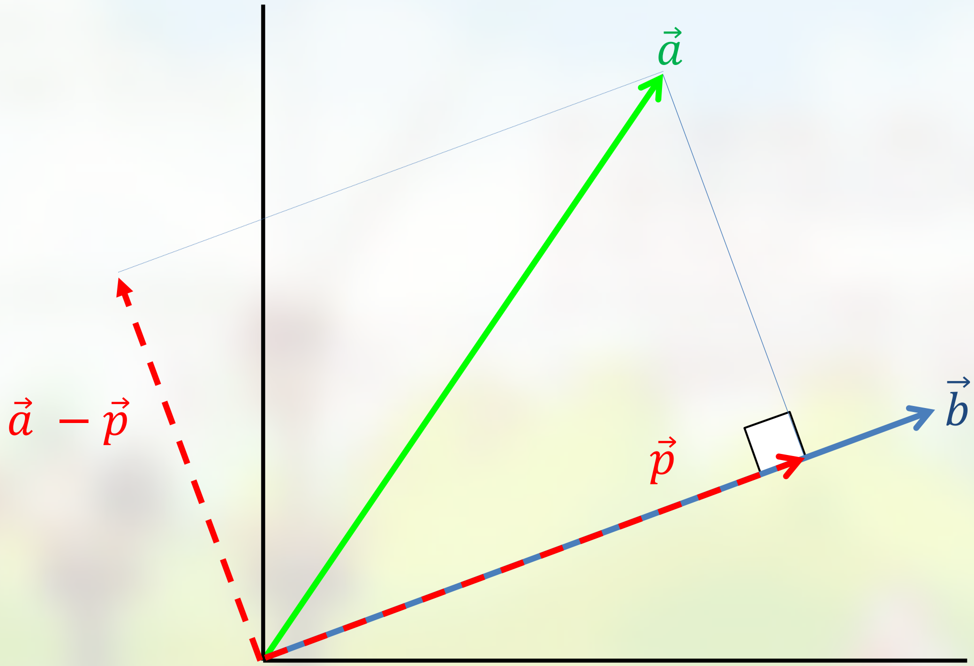
Scalar projection

Vector projection

The vector $\vec{p}$ is the projection of $\vec{a}$ onto $\vec{b}$ if

1. $\vec{p}$ is parallel to $\vec{b}$ and
2. $\vec{a} - \vec{p}$ is perpendicular to $\vec{b}$.



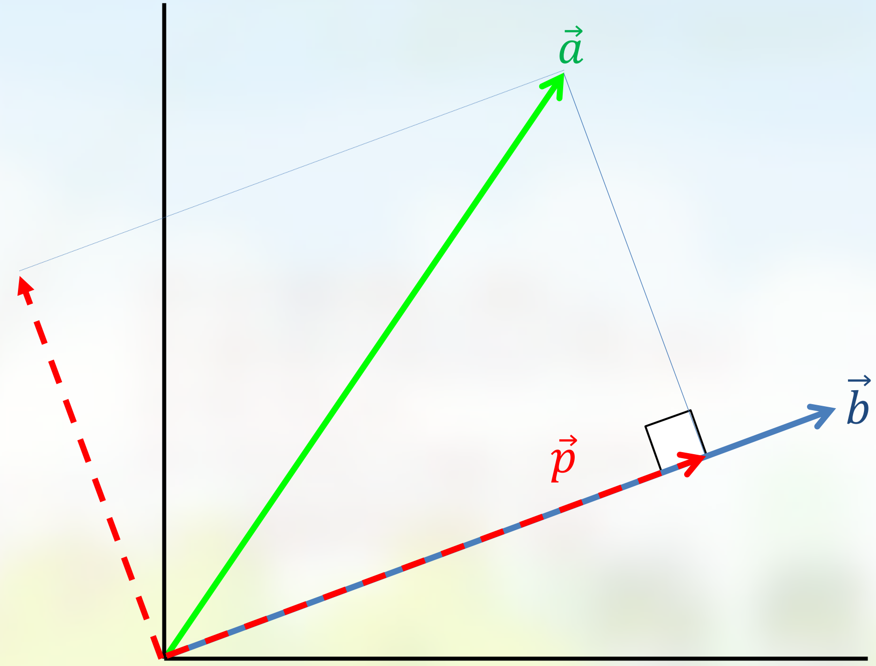


Informally: $\vec{p}$ and $\vec{a} - \vec{p}$ are the “shadows” cast by $\vec{a}$ when shining a light perpendicular resp. parallel to $\vec{b}$

Scalar projection

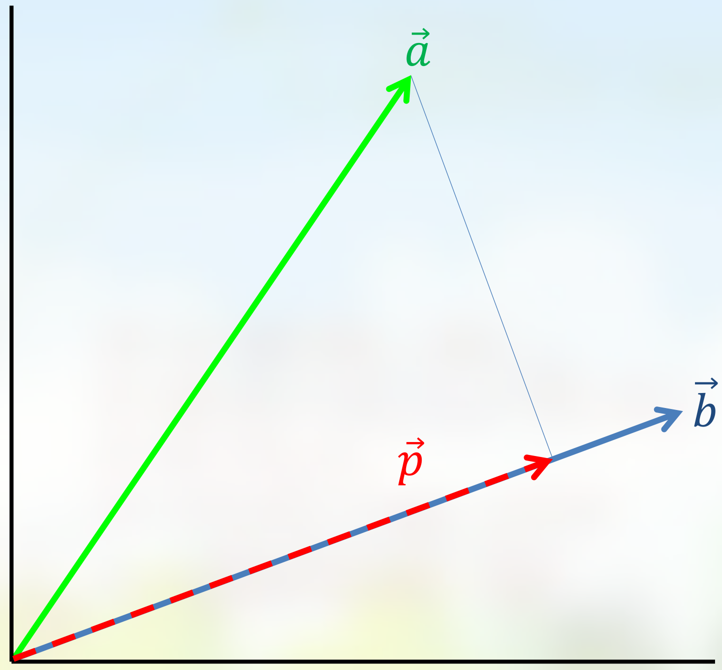
The scalar projection of $\vec{a}$ onto $\vec{b}$ is the "length"* of the vector projection

*Note: if the angle between $\vec{a}$ and $\vec{b}$ is >90, this number is *negative.*



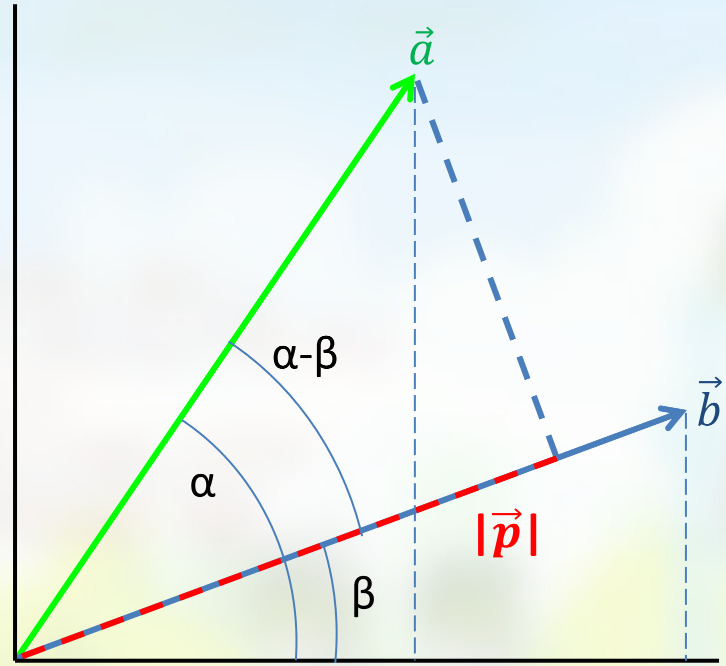
Exercise

Calculate $|\vec{p}|$ using trigonometry:



“Primitive” solution

- In theory we could calculate α, β, α-β and then $|\vec{p}|$ using a combination of atan2, Pythagoras, cos:
  - slow
  - cumbersome
  - error prone
  - repetitive
  - hurts your brain



Quicker way: Dot product

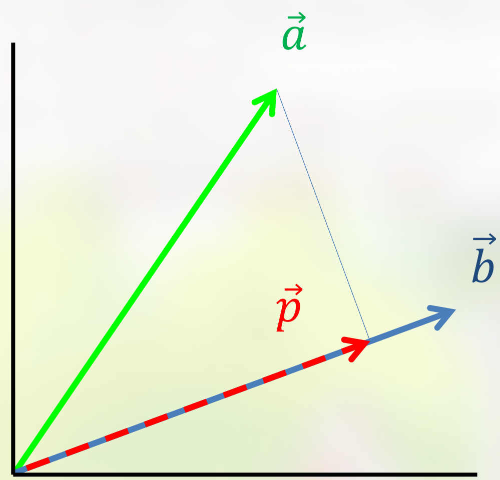
Definition of the • (dot) product:

$$\vec{a} \bullet \vec{b} = \vec{a}.x * \vec{b}.x + \vec{a}.y * \vec{b}.y$$

As it turns out:

$$\vec{a} \bullet \hat{b} = \vec{a}.x * \hat{b}.x + \vec{a}.y * \hat{b}.y = |\vec{p}|$$

Note: must use the normalized version of *b!*



Dot Product = Scalar Projection

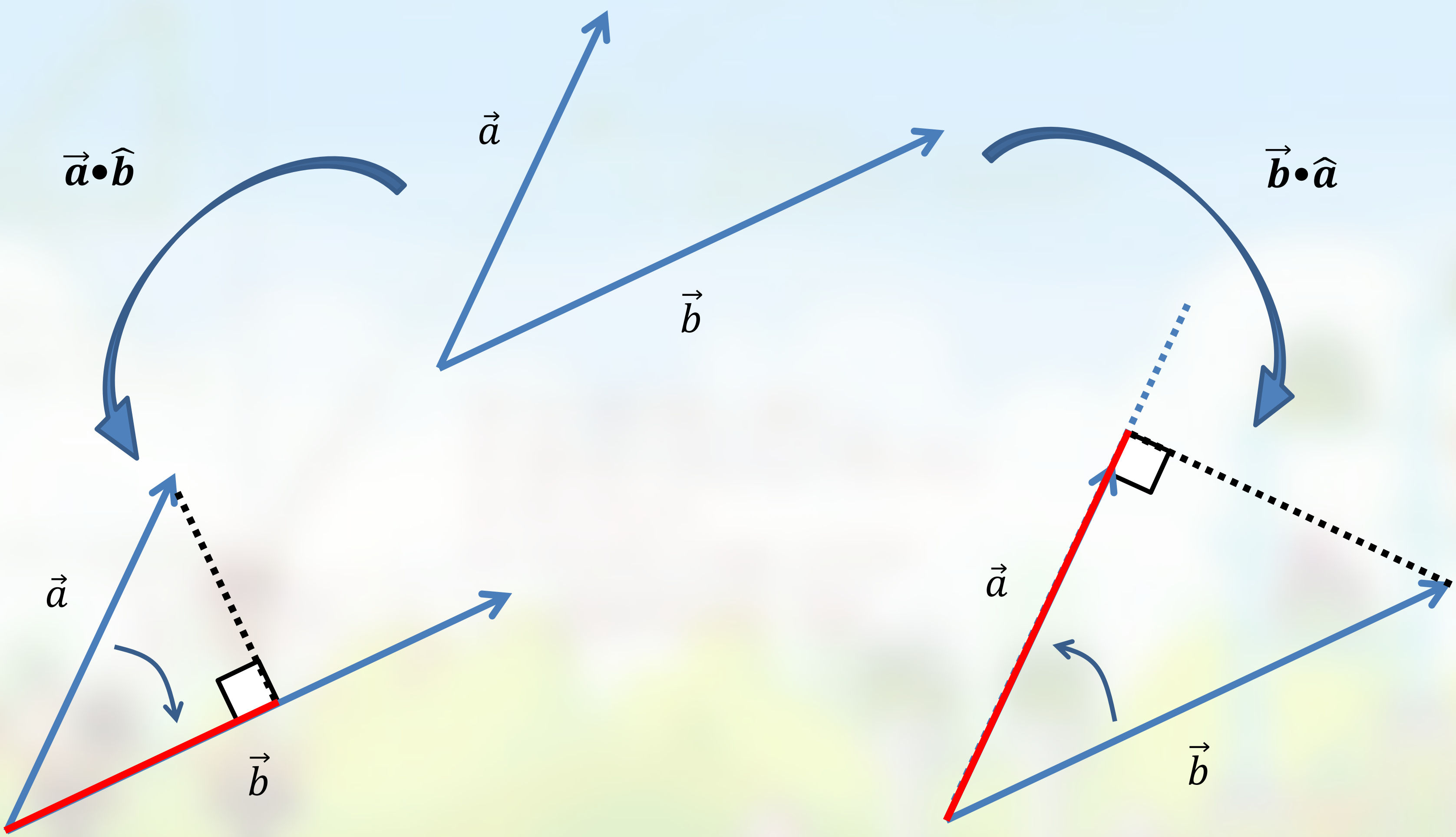
There are different ways to convince yourself of this claim:

$$\vec{a} \bullet \hat{b} = \vec{a}.x * \hat{b}.x + \vec{a}.y * \hat{b}.y = |\vec{p}|$$

1. Believe the teacher…?
2. Implement it in code sample 001, and see for yourself (see Assignment 4.1)
3. …See also the next <demo>
4. Check out the *proof* at the end of these slides.

$\vec{a} \bullet \hat{b}$ vs $\vec{b} \bullet \hat{a}$

Order matters!

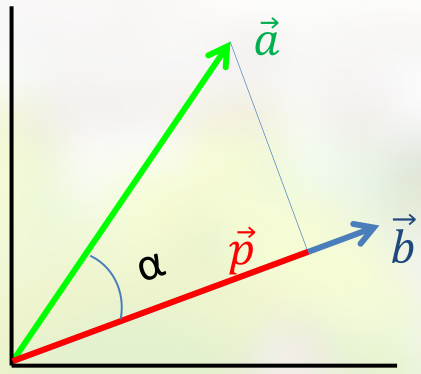


The official dot product formula

$$|\vec{p}| = \vec{a} \bullet \hat{b} = \vec{a} \bullet \frac{\vec{b}}{|\vec{b}|} = \frac{\vec{a} \bullet \vec{b}}{|\vec{b}|} \xrightarrow{\text{therefore}}$$

$|\vec{p}||\vec{b}| = \vec{a} \bullet \vec{b}$, but $|\vec{p}|$ is also $|\vec{a}|\cos(\alpha) \xrightarrow{\text{therefore}}$

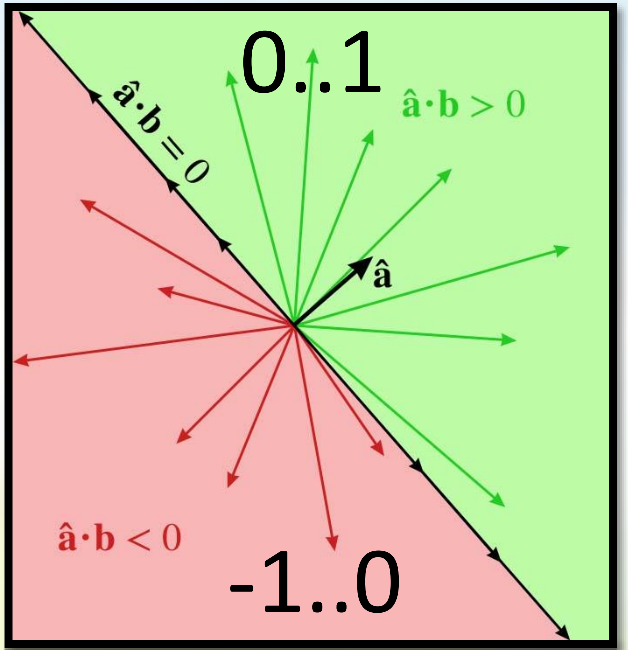
$$\boxed{\vec{\boldsymbol{a}} \bullet \vec{\boldsymbol{b}} = |\vec{a}||\vec{b}|\, \boldsymbol{\cos(\alpha)}}$$



The dot product

$\vec{a} \bullet \vec{b}$ = $\|\vec{a}\| \|\vec{b}\| \, cos(\alpha)$ implies

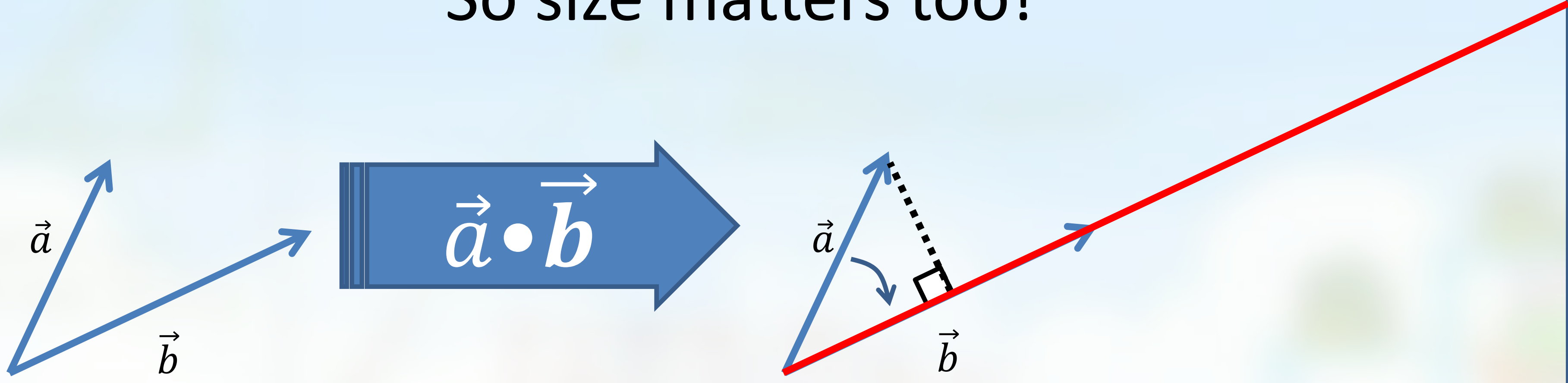
$\hat{a} \bullet \hat{b} = cos(\alpha)$ = -1..1



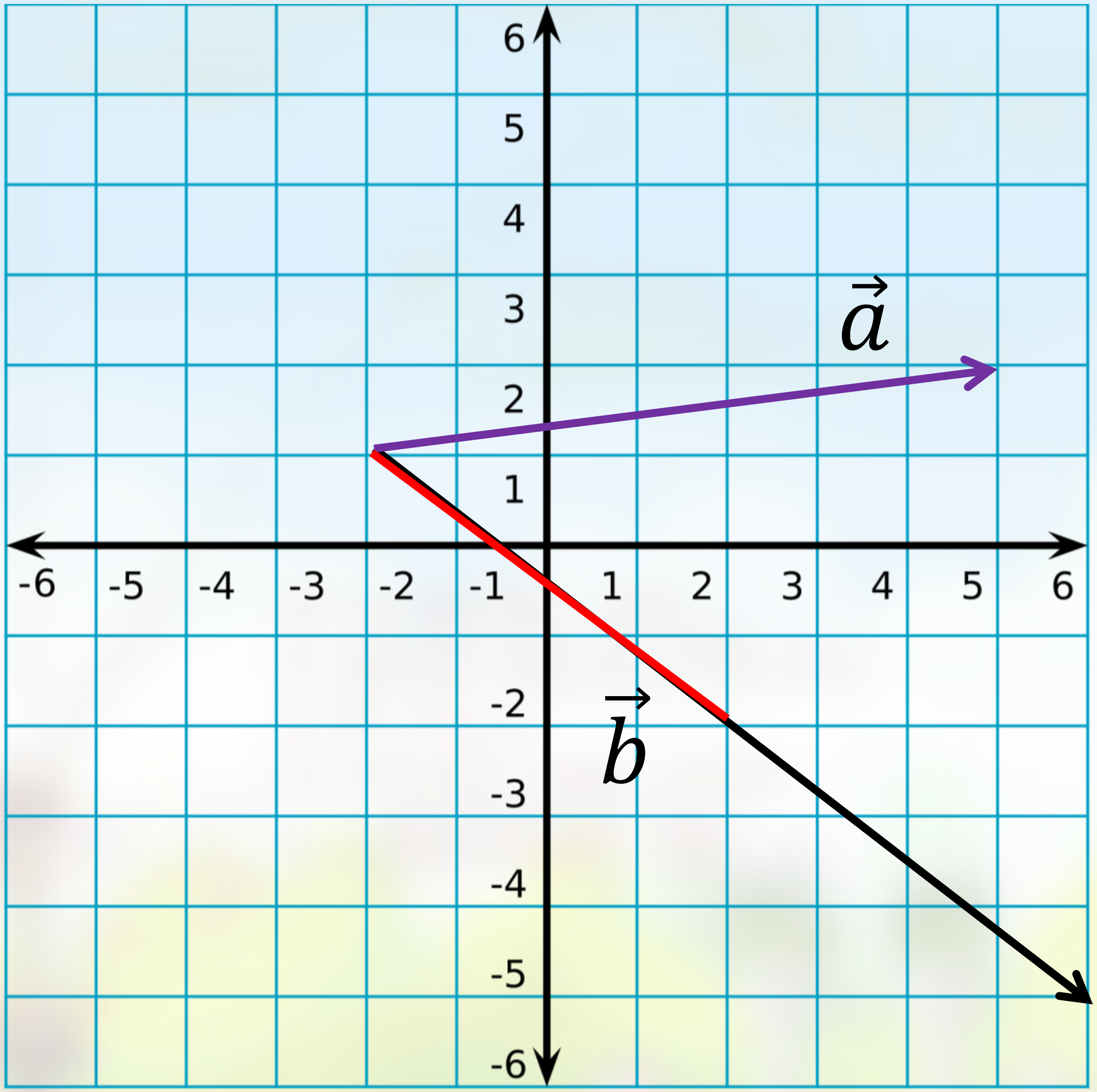
Don't forget to normalize $\vec{\boldsymbol{b}}$!!

Otherwise the projection is $|\vec{\boldsymbol{b}}|$ times too big !

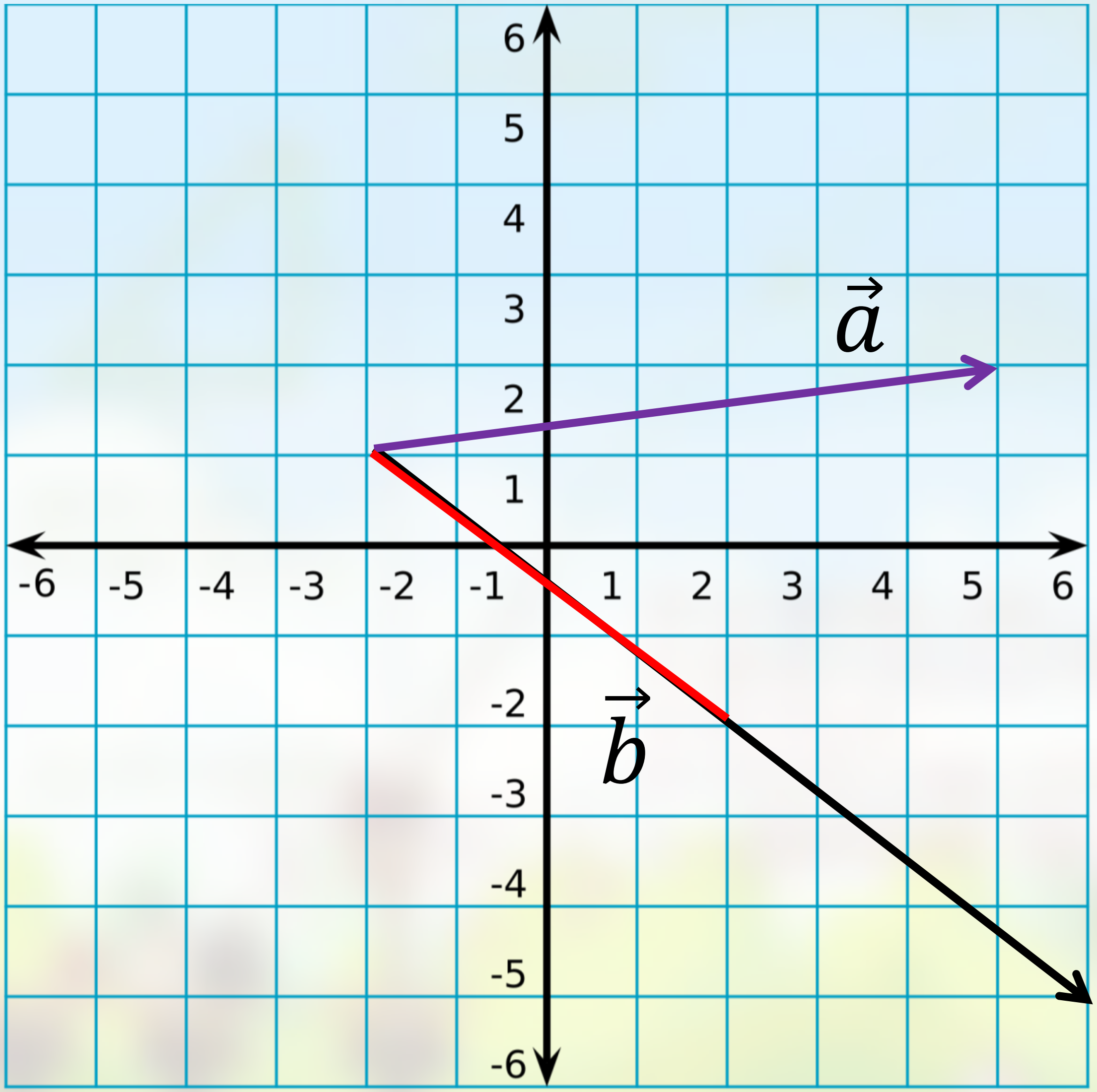
So size matters too!



Exercise: calculate $|\vec{p}|$ using the dot product



Exercise: calculate $|\vec{p}|$ using the dot product



$\vec{a} = (7, 1)$

$\vec{b} = (8, -6)$

$|\vec{b}| = \sqrt{8^2 + (-6)^2} = \sqrt{100} = 10$

$\hat{b} = \frac{\vec{b}}{|\vec{b}|} = (\frac{4}{5}, -\frac{3}{5})$

$|\vec{p}| = \vec{a} \bullet \hat{b} = \frac{7 \cdot 4 - 1 \cdot 3}{5} = 25/5 = 5$

Implementing Dot(…)

Implementing Dot()

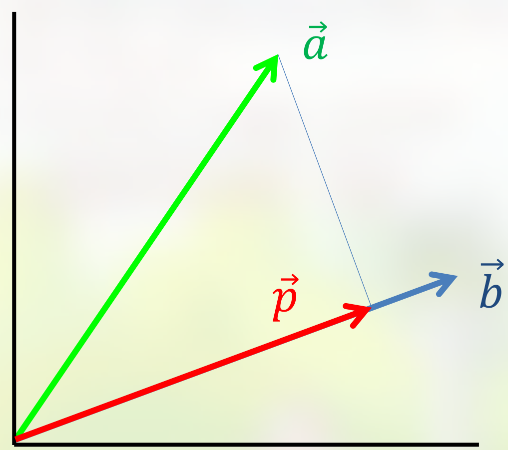
- Implement Vec2.Dot(???):
  - Return the dot product of this and the given vector:
  - $\vec{a} \bullet \vec{b} = \vec{a}.x * \vec{b}.x + \vec{a}.y * \vec{b}.y$

```
public ??? Dot(???) {
        return ???;
}
```

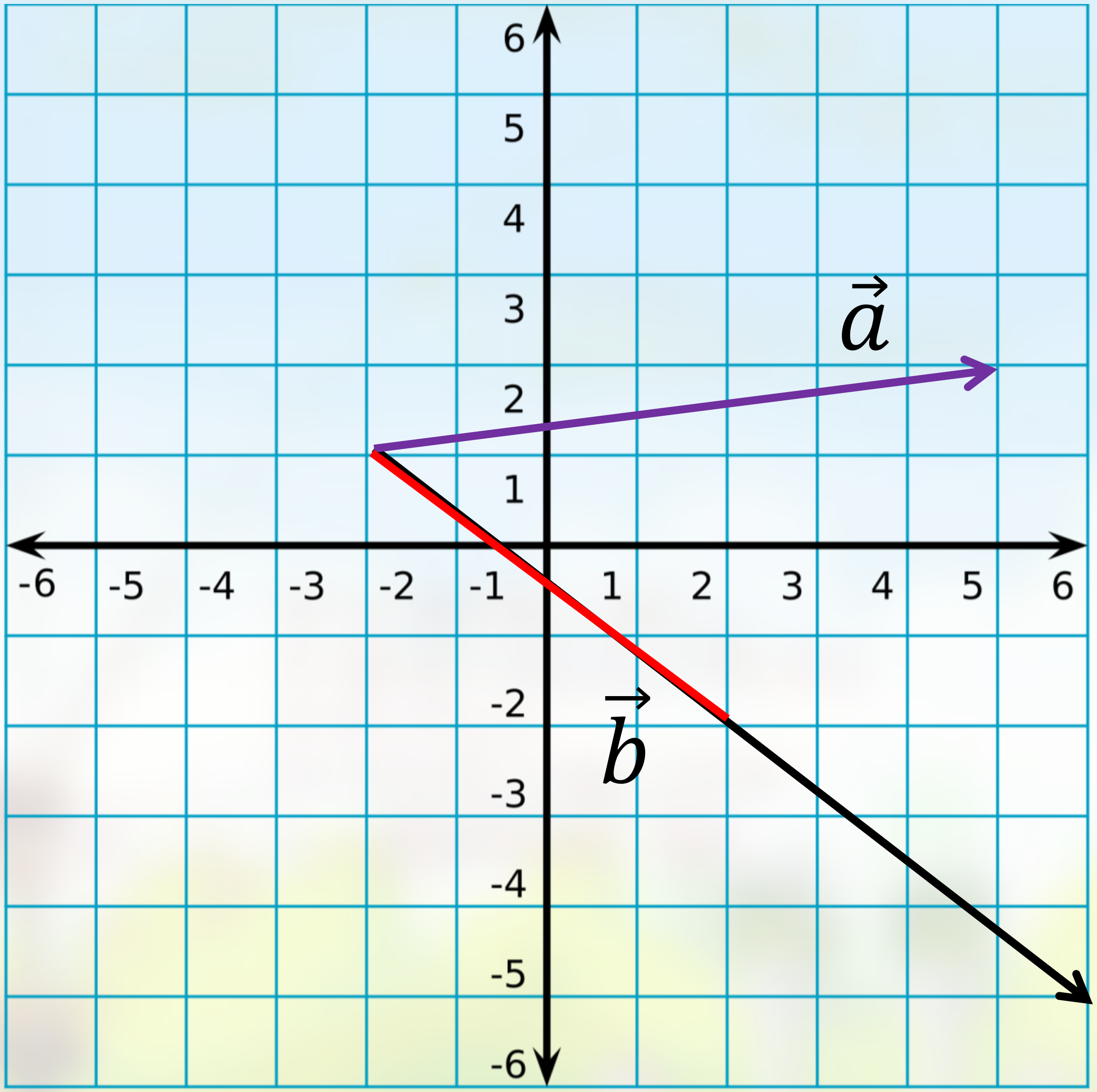
How can we test the dot product?

- If we draw $\vec{a}$ and $\vec{b}$ we could calculate:
  - $|\vec{p}|$ using $\vec{a} \bullet \hat{b}$
  - $\vec{p} = |\vec{p}| * \hat{b} = \vec{a} \bullet \hat{b} * \hat{b}$

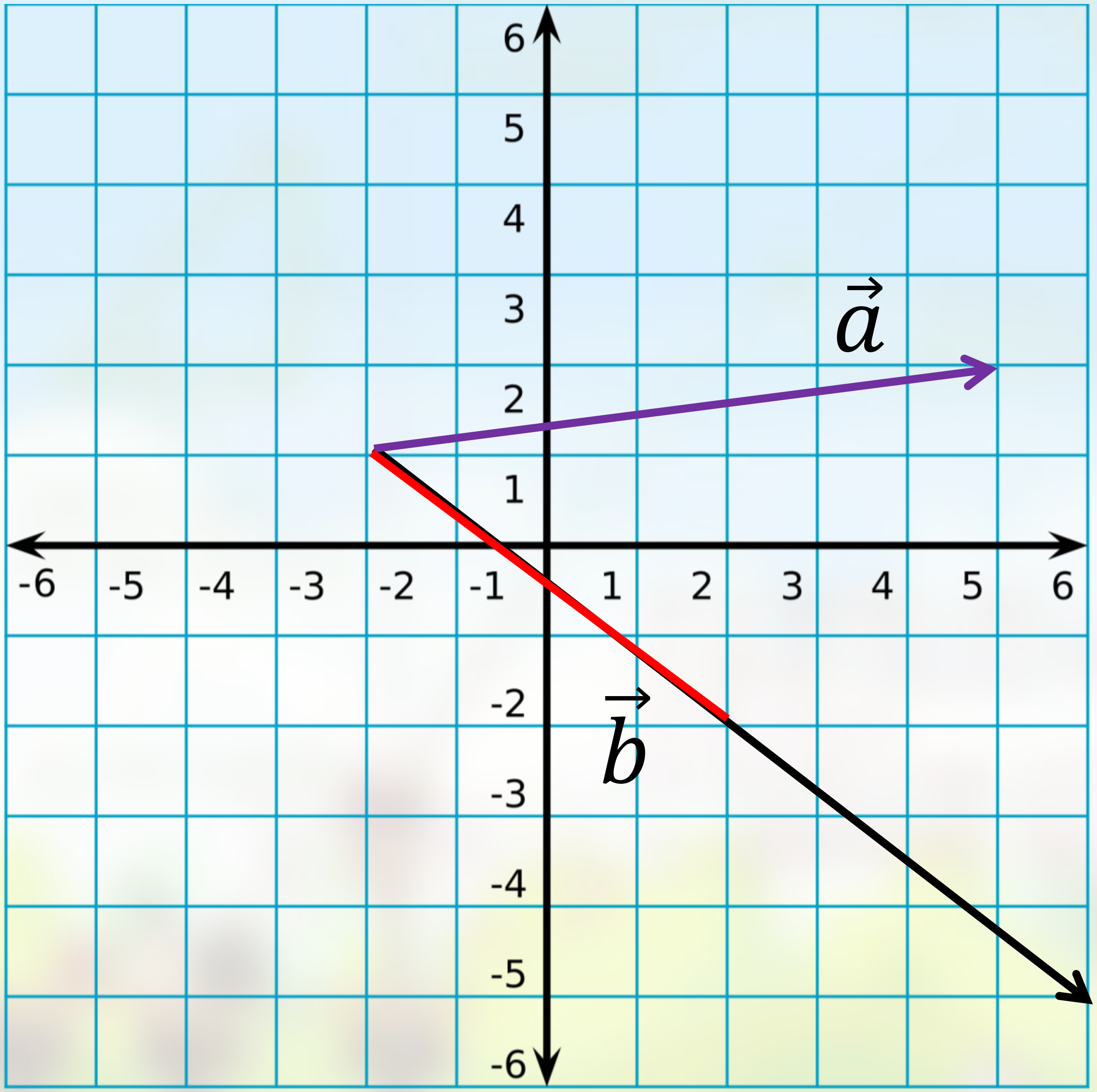
- This is *vector projection* (vs scalar):
- Example: -001_dot_product



Exercise: calculate $\vec{p}$ using the dot product



Exercise: calculate $\vec{p}$ using the dot product



$\vec{a} = (7, 1)$

$\vec{b} = (8, -6)$

$|\vec{b}| = \sqrt{8^2 + (-6)^2} = \sqrt{100} = 10$

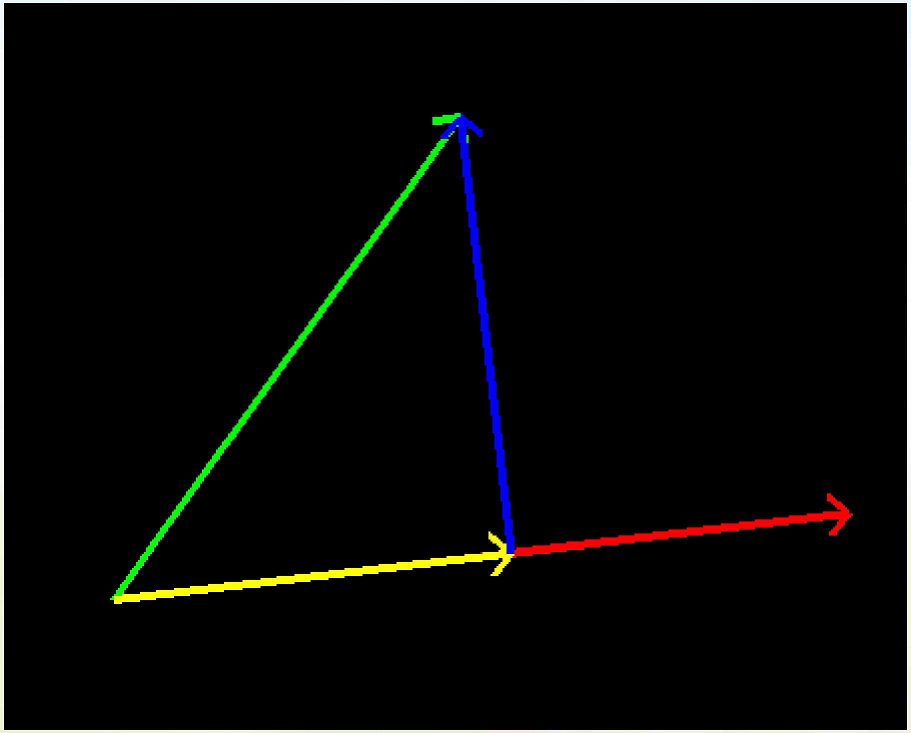
$\hat{b} = \frac{\vec{b}}{|\vec{b}|} = (\frac{4}{5}, -\frac{3}{5})$

$|\vec{p}| = \vec{a} \bullet \hat{b} = \frac{7 \cdot 4 - 1 \cdot 3}{5} = 25/5 = 5$

$\vec{p} = (\vec{a} \bullet \hat{b})\, \hat{b} = 5 \cdot (\frac{4}{5}, -\frac{3}{5}) = (4, -3)$

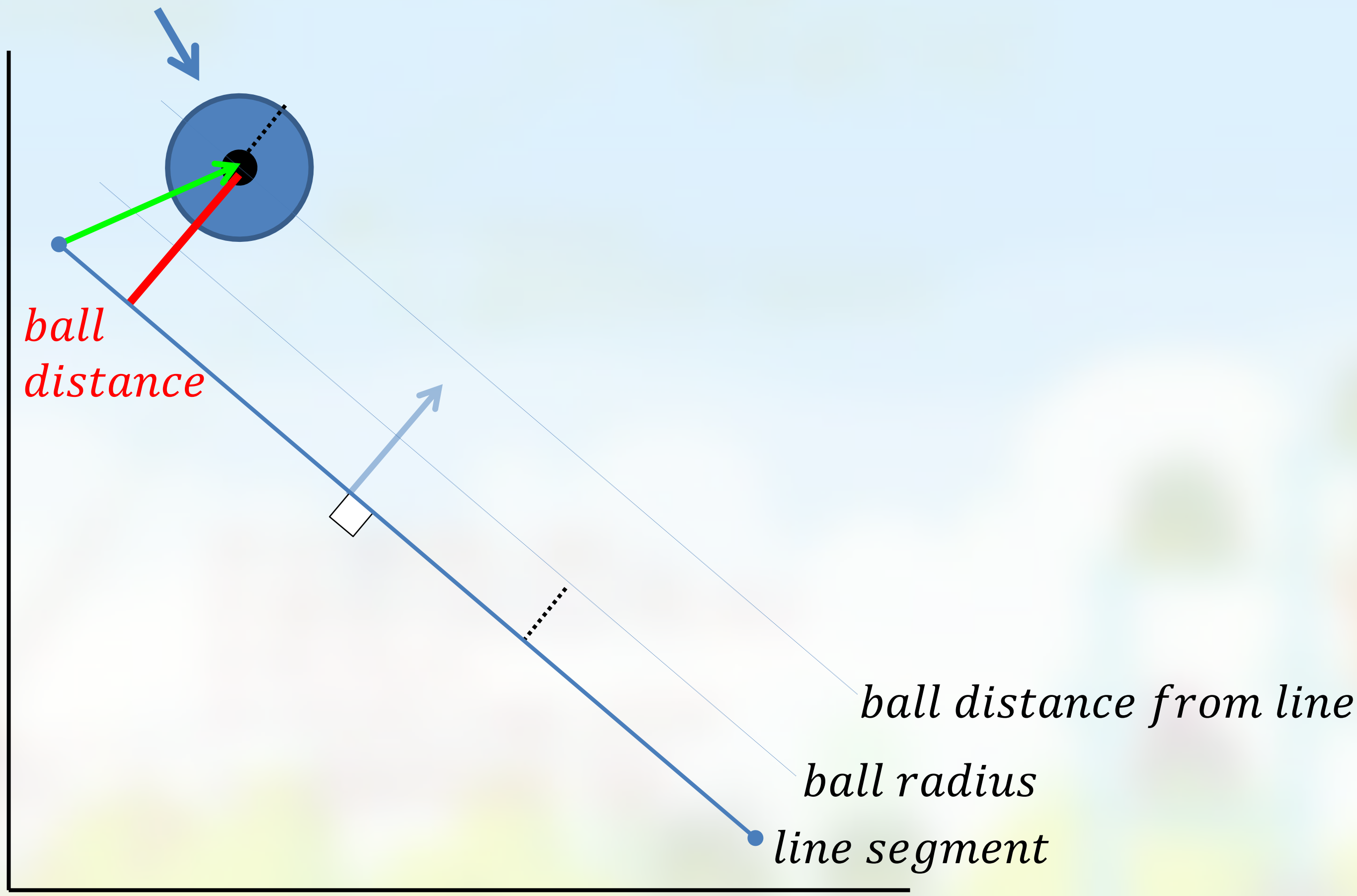
Test your setup!

- Using +001_the_dot_product:
  - Left and right click to set the green/red vectors
- When correctly implemented it looks like:



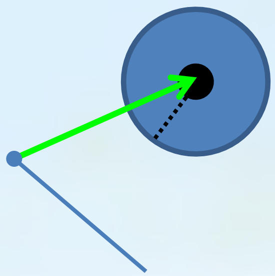
Ball/Line collision **detection** using scalar projection (the dot product)

Ball/Line collision detection

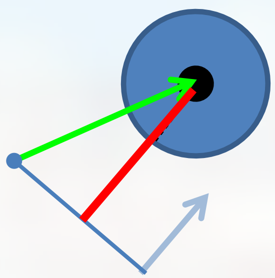


Ball/Line collision detection overview

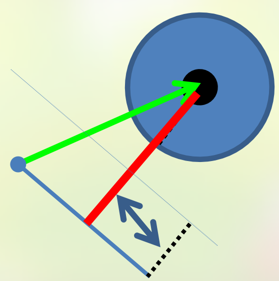
Get difference vector of ball and line start/end:



Scalar project diff. vector onto line normal with dot product:



Compare projection length with ball radius:



Ball/Line collision detection overview

- For now we are ignoring the fact that we are dealing with only a segment. We act as if we are bouncing on an infinite line, not just a segment.
  - limiting to segments: lecture 5.
- This is discrete collision detection: are we currently overlapping/below the line?
- We can use *any* point on the line to compute a difference vector (=green) to start with

Ball/Line collision detection code

- Base code: +002_line_collision_detection

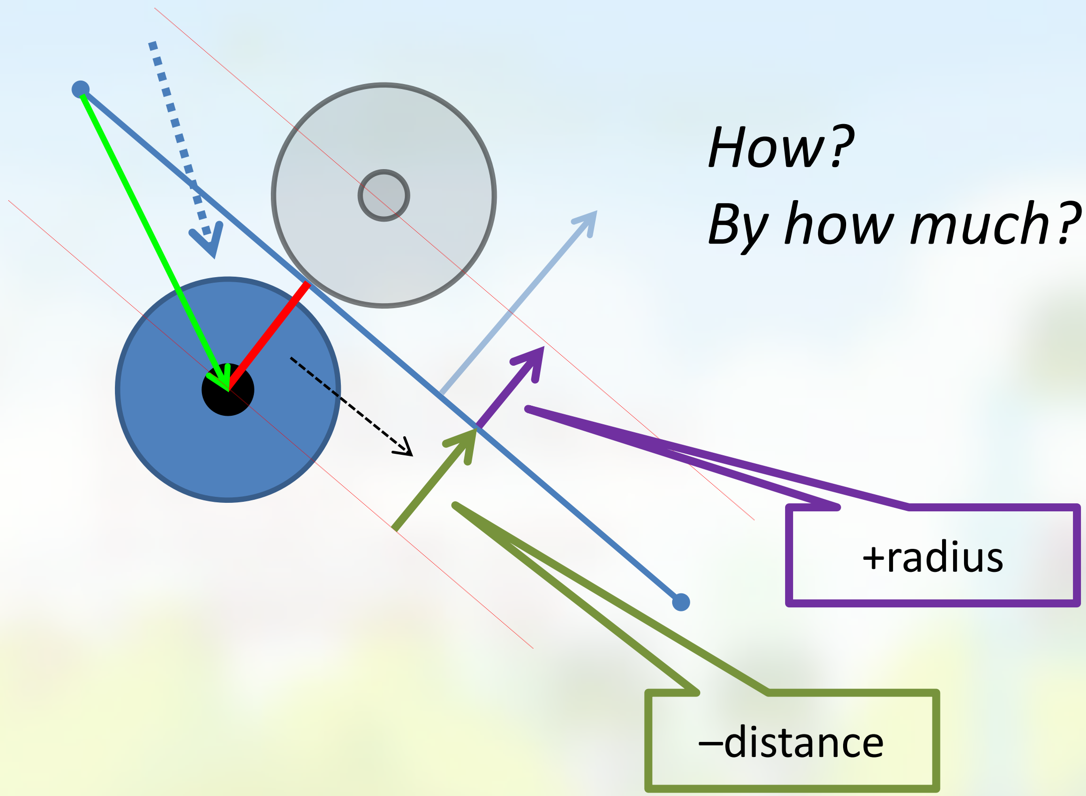
- When implemented correctly,
  the ball turns red when closer than r to the line
  (or only when **above** the line, that is ok too)
- Basically for now, lines have only 1 positive/ok side

Position reset

Simple position reset

If the ball is below the line / touching the line:

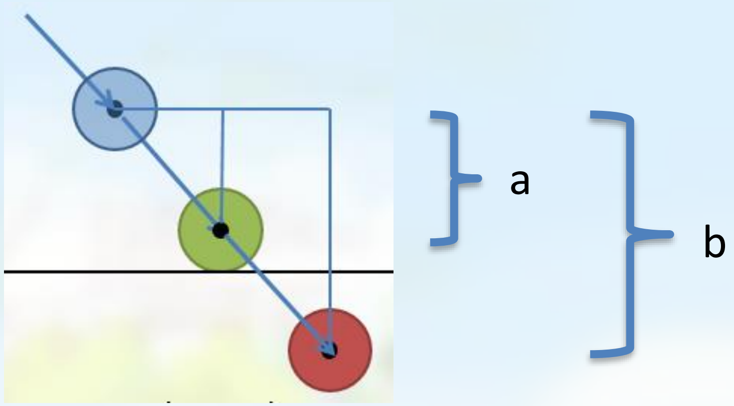
Move it back in the direction of the normal



Example -002 / -003

Point of Impact (POI) Calculation

- Consider three values:
  - oldDistance →
  - Radius →
  - newDistance →

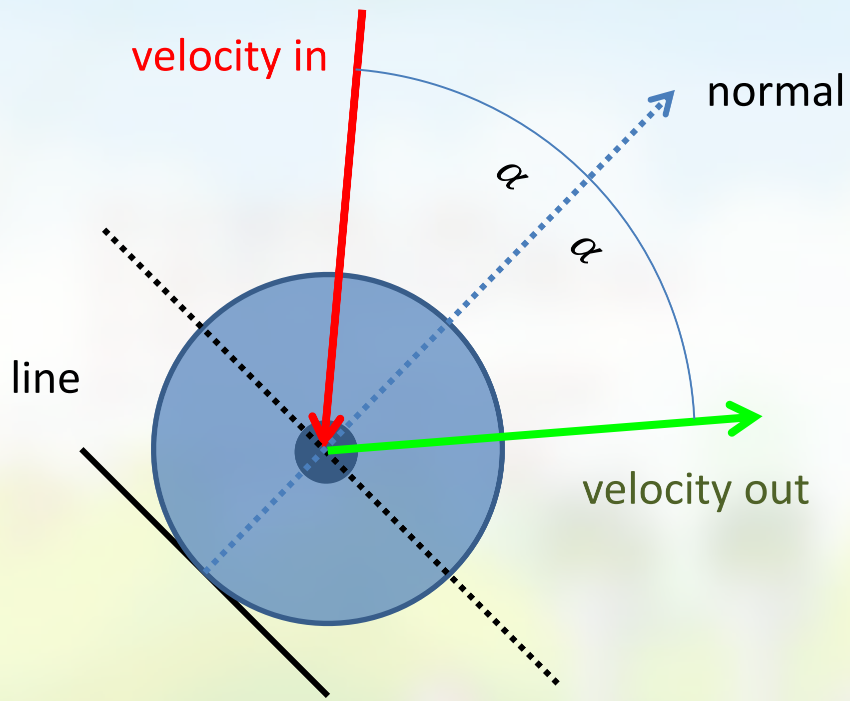


- They define lengths a and b. (with a<b)
- Define *time of impact $t = a/b$.*
  - If t=0: impact at "start of current frame"
  - If t=1: impact at "end of current frame"
- Set: POI = oldPosition + t * velocity

Reflecting the velocity using The Law of Reflection

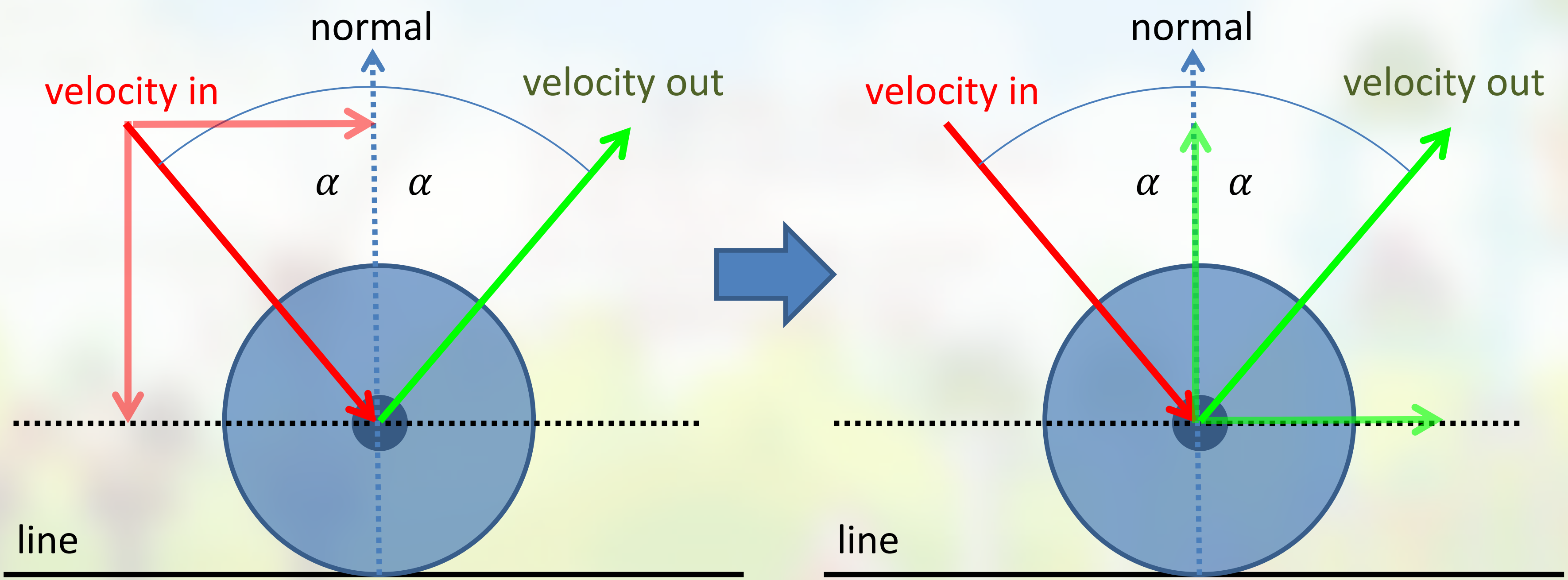
Law of Reflection

- the angle of incidence == the angle of reflection (in the case of *fully elastic collisions*)



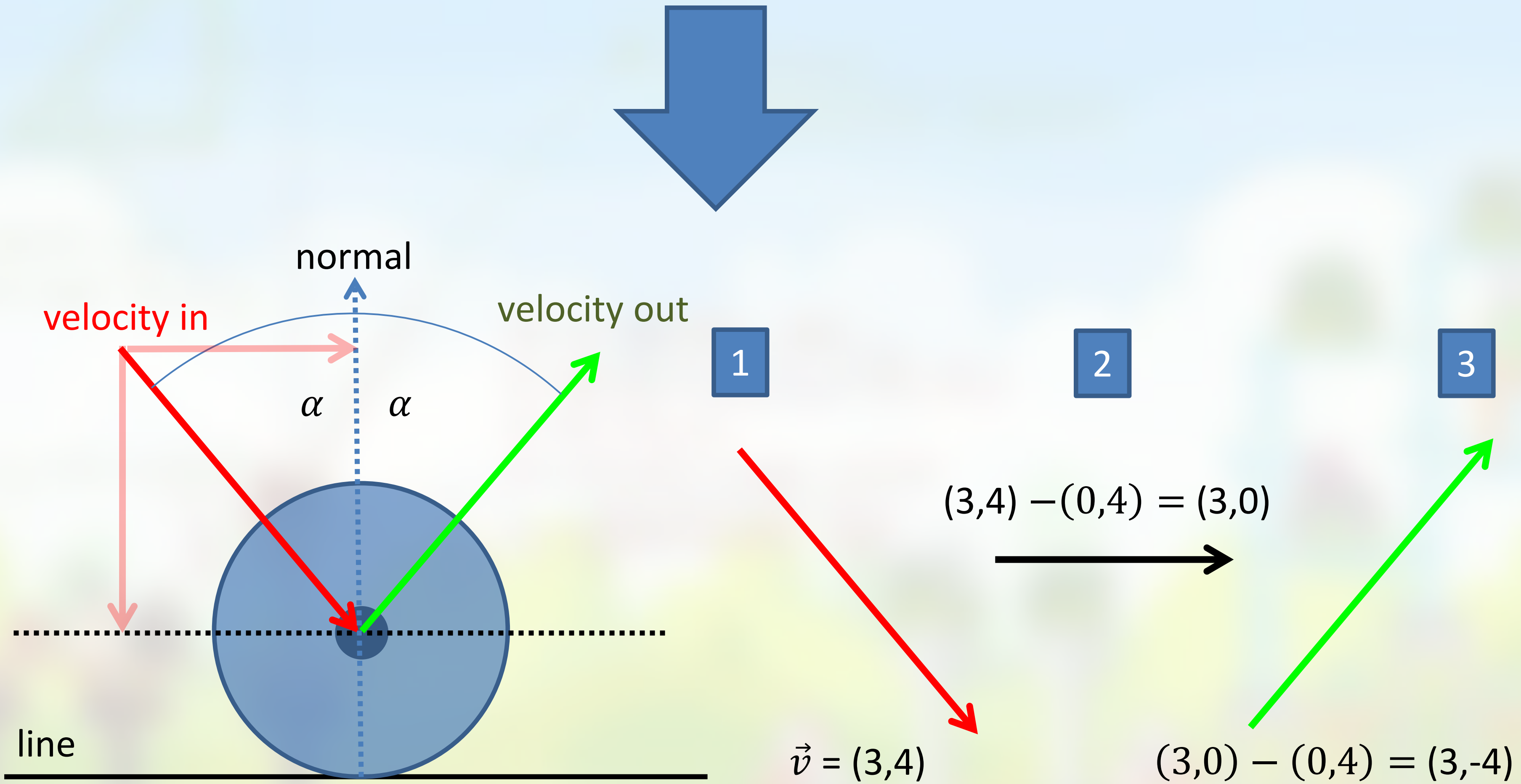
Law of Reflection

- In non angled case $\overrightarrow{v_{out}}.\,y$ = -$\overrightarrow{v_{in}}.\,y$



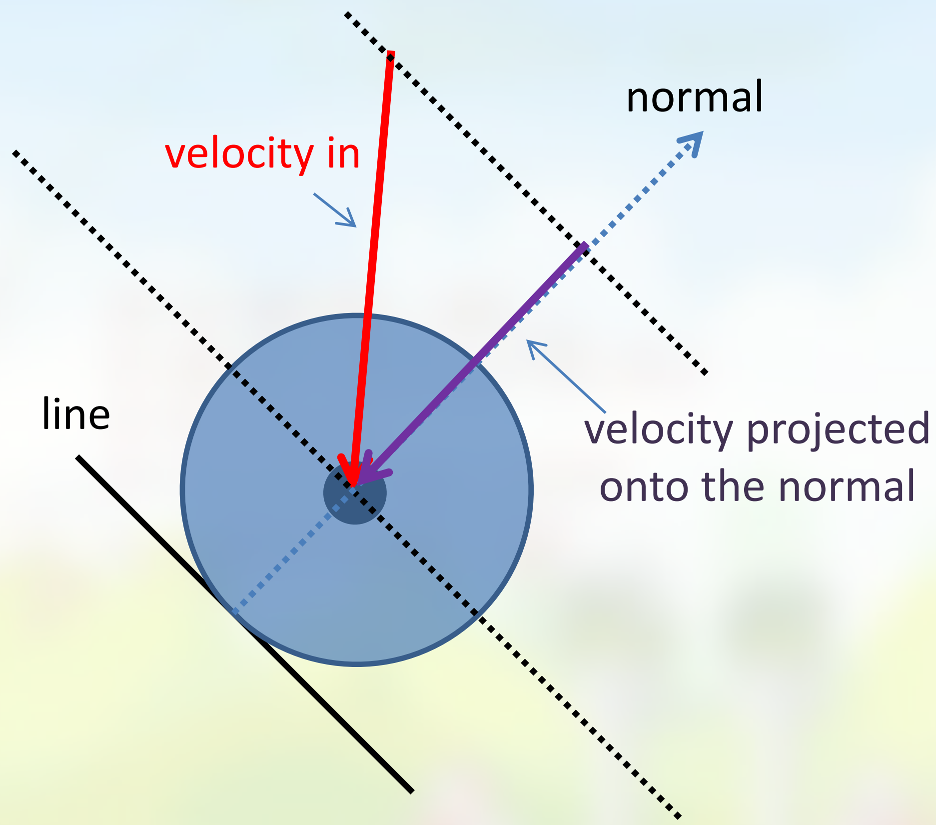
Law of Reflection: Non angled

$$\overrightarrow{v_{out}} = \overrightarrow{v_{in}} - 2 * (0, \overrightarrow{v_{in}}.y)$$



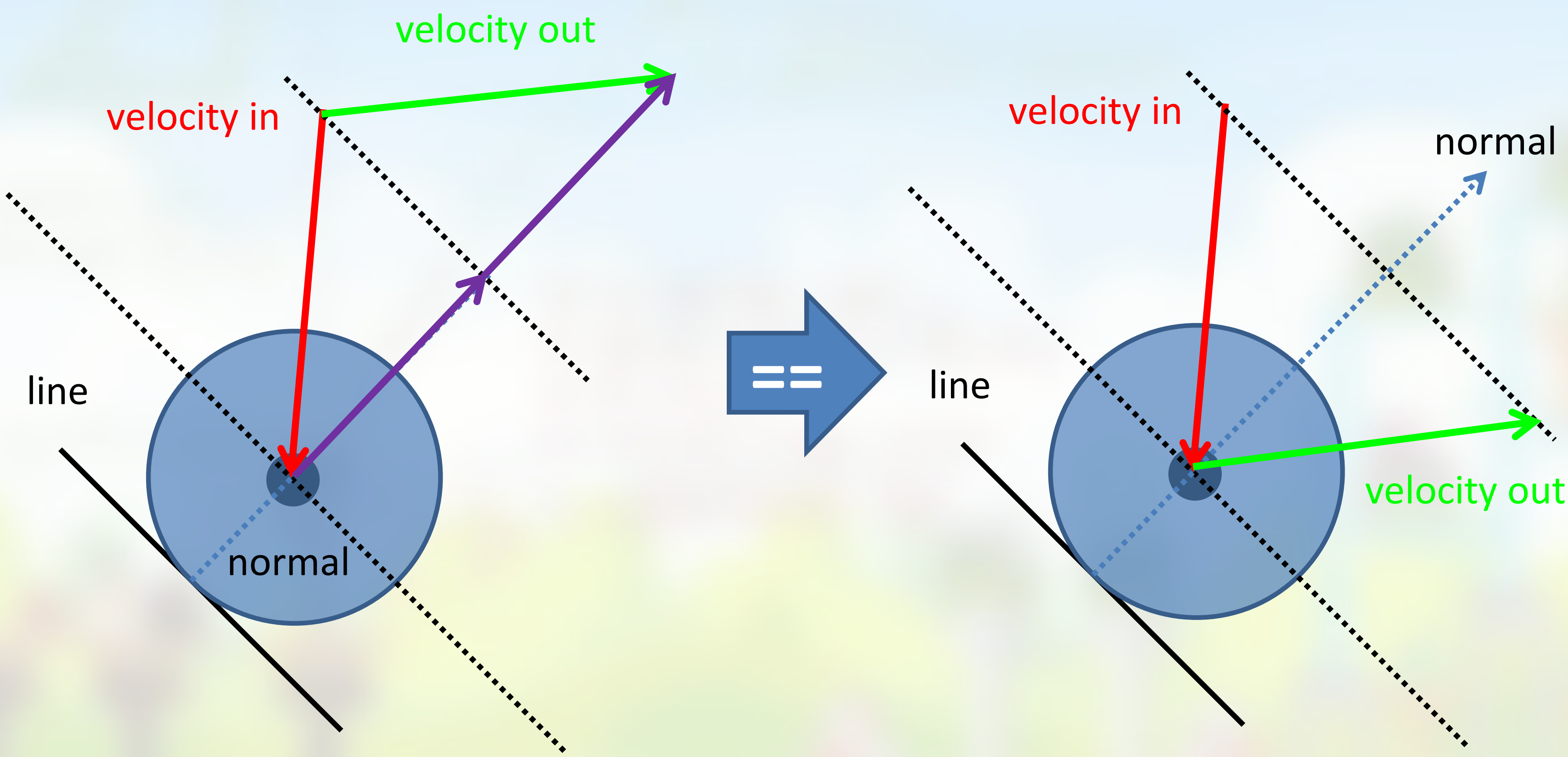
Law of Reflection, Angled: step by step

1. Find the part of the velocity parallel to the normal using vector projection.



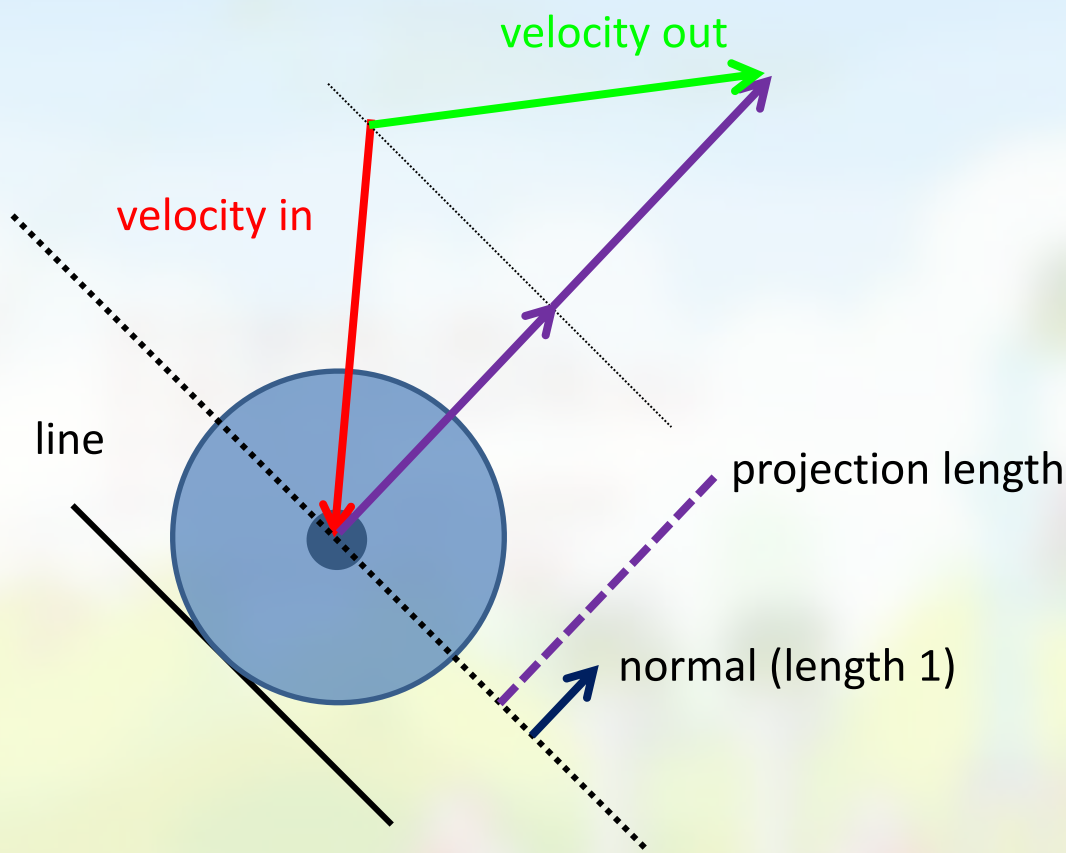
Law of Reflection, Angled: step by step

2. Subtract projected velocity twice from original velocity



Law of Reflection, Perfect elastic collision

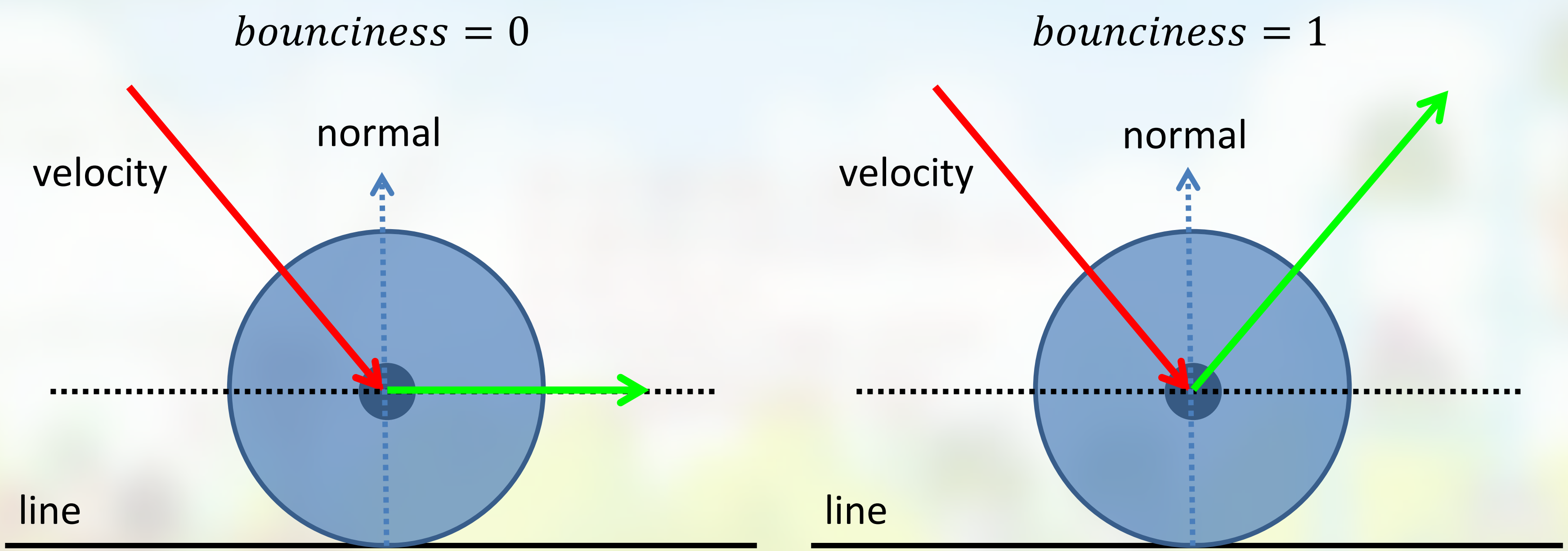
$$\overrightarrow{v_{out}} = \overrightarrow{v_{in}} - 2 \cdot (\overrightarrow{v_{in}} \bullet \hat{n}) \cdot \hat{n}$$



Law of Reflection, non perfect collisions

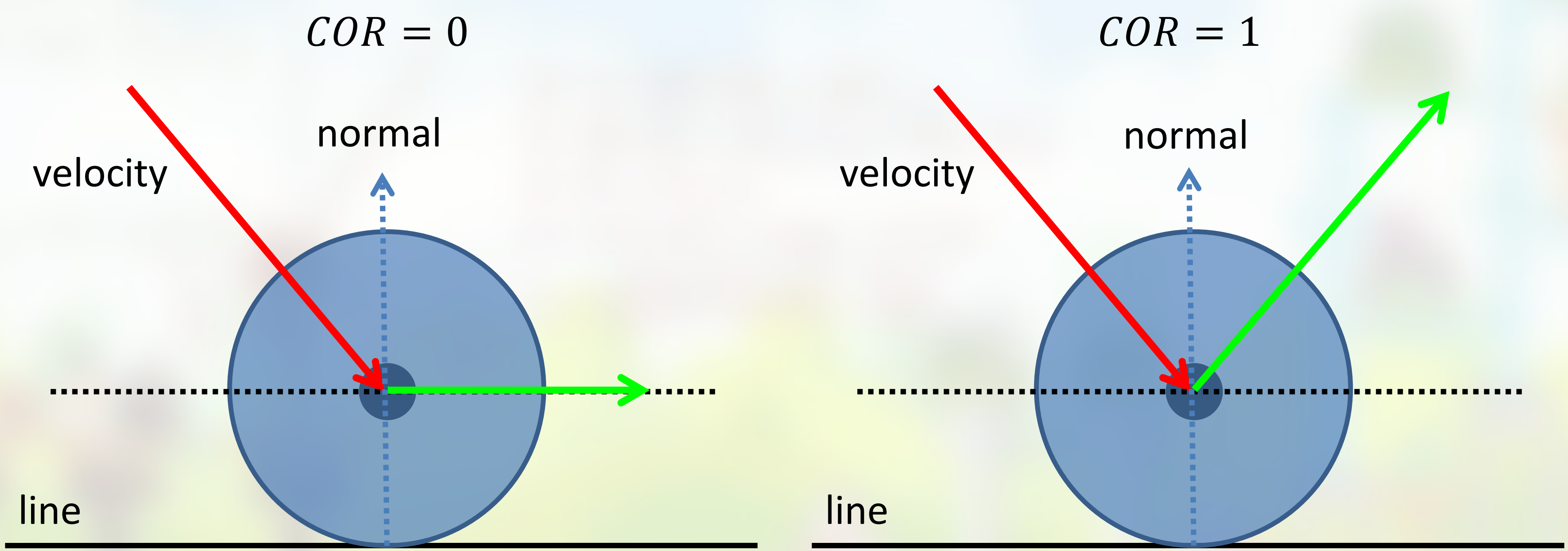
$$\overrightarrow{v_{out}} = \overrightarrow{v_{in}} - (1 + bounciness) \cdot (\overrightarrow{v_{in}} \bullet \hat{n}) \cdot \hat{n}$$
$$0 \leq bounciness \leq 1$$

(Perfect elastic collision: bounciness = 1)



COR : Coefficient Of Reflection

- Official word for bounciness:
  - coefficient of reflection COR, written as *C*:
  - $\overrightarrow{v_{out}} = \overrightarrow{v_{in}} - (1 + C) \cdot (\overrightarrow{v_{in}} \bullet \hat{n}) \cdot \hat{n}$



Implementing Reflect()

Implement Reflect(…)

$$\overrightarrow{v_{out}} = \overrightarrow{v_{in}} - (1 + \mathrm{C}) \cdot (\overrightarrow{v_{in}} \bullet \hat{n}) \cdot \hat{n}$$

```
public void Reflect(Vec2 pNormal, float pBounciness = 1) {
        //…. reflection code here ….
}
```

Demo (-003 / -004)

Dot product - Summary

- Main uses of Dot product:
  - Projecting a vector onto another vector
    - Used for velocity reflection, distance computation, …
  - Computing angle between two vectors
    - Really easy to determine: is the angle bigger or smaller than 90 degrees?
- Everything presented holds also for 3 (or 4, 5…) dimensions:
  - 3D: $\vec{a} \bullet \vec{b} = \vec{a}.x * \vec{b}.x + \vec{a}.y * \vec{b}.y + \vec{a}.z * \vec{b}.z$
  - $\vec{\boldsymbol{a}} \bullet \vec{\boldsymbol{b}} = |\vec{a}||\vec{b}|\,\boldsymbol{cos}(\boldsymbol{\alpha})$ still true

Assignment 4

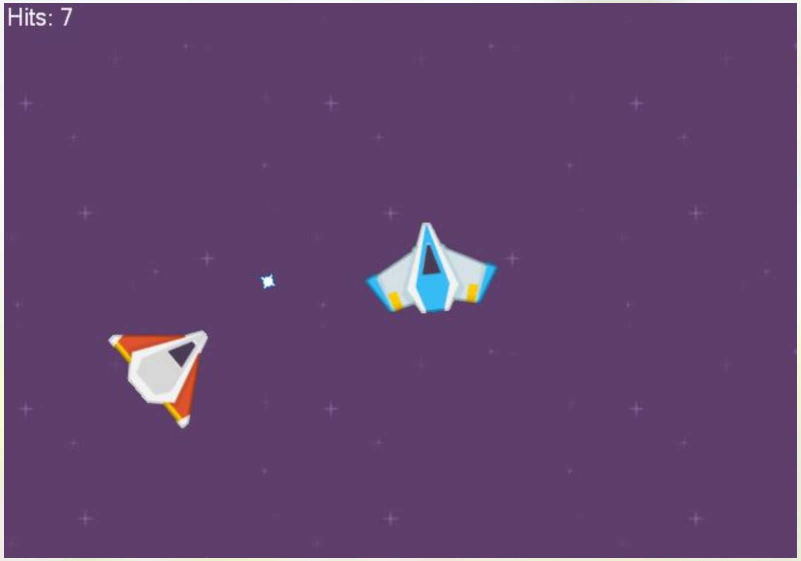
- Download the assignment from blackboard for details

- In short:
  - Bouncing a ball off multiple angled lines *(demo)*

Other Applications

- Note that the dot product and formulas from the last two lectures have many more (unexpected) applications!
- Next year: 3D Math
- Example for now: smart aiming (you might use this to score `excellent' on aiming)
- (demo)

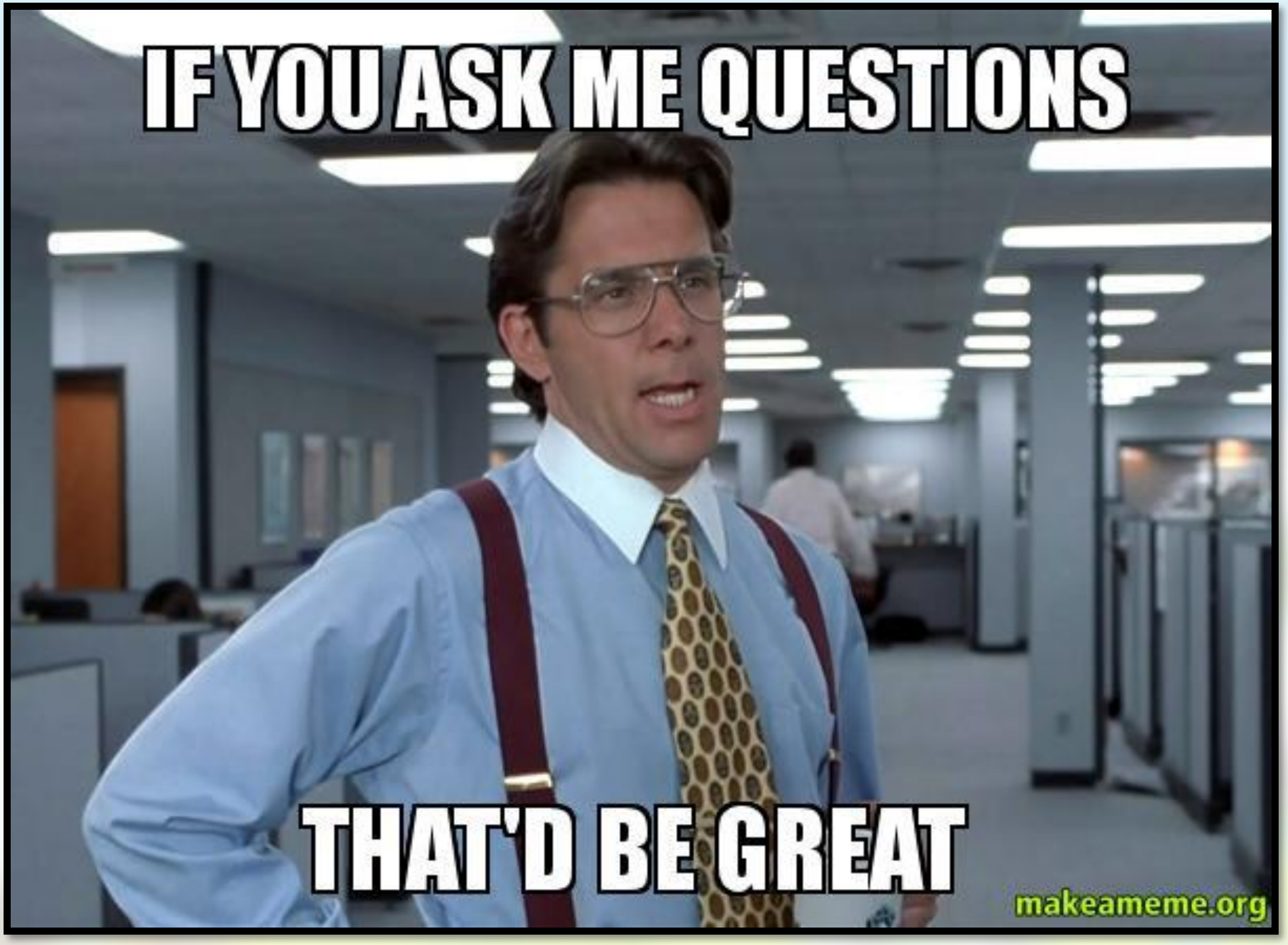
(If you score well on the Blackboard tests for week 3, 4, you might find some tips for this somewhere on Blackboard…)



Next time

- Fixing line segment collision issues:
  - collision is not limited to the line segment yet
  - ball cannot pass behind line segment without being reset
  - ball cannot bounce on the line ends/caps
  - bounce is one sided

- And more exciting stories from the world of physics ☺


IF YOU ASK ME QUESTIONS
THAT'D BE GREAT
makeameme.org

Scalar projection proof

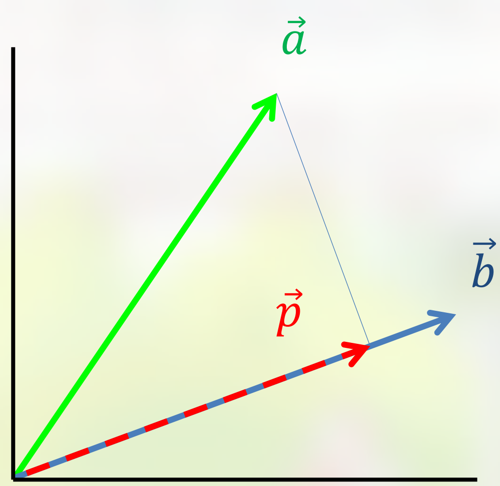
Background info

Scalar Projection

Given $\vec{a}$, $\vec{b}$ and $\vec{p}$ as shown below (see also earlier slides), this is what we want to prove:

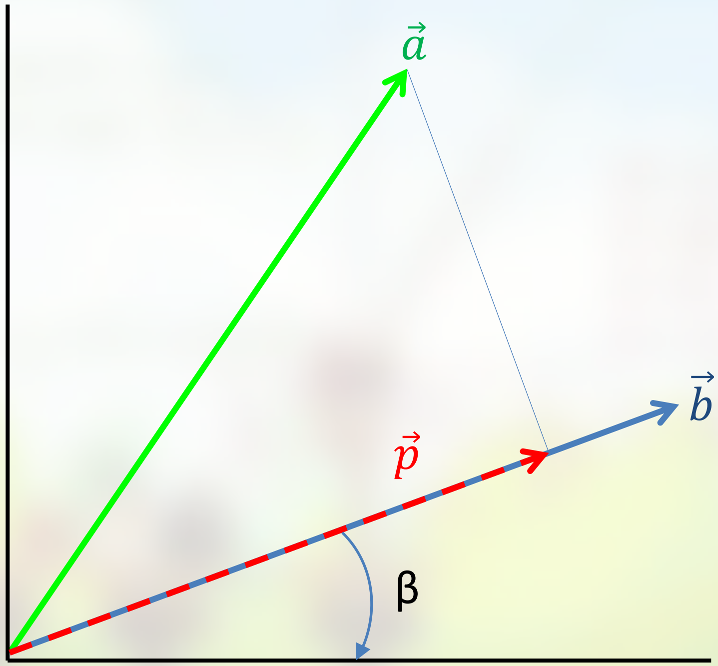
$$\vec{a} \bullet \hat{b} = \vec{a}.x * \hat{b}.x + \vec{a}.y * \hat{b}.y = |\vec{p}|$$

Note: must use the normalized version of *b!*

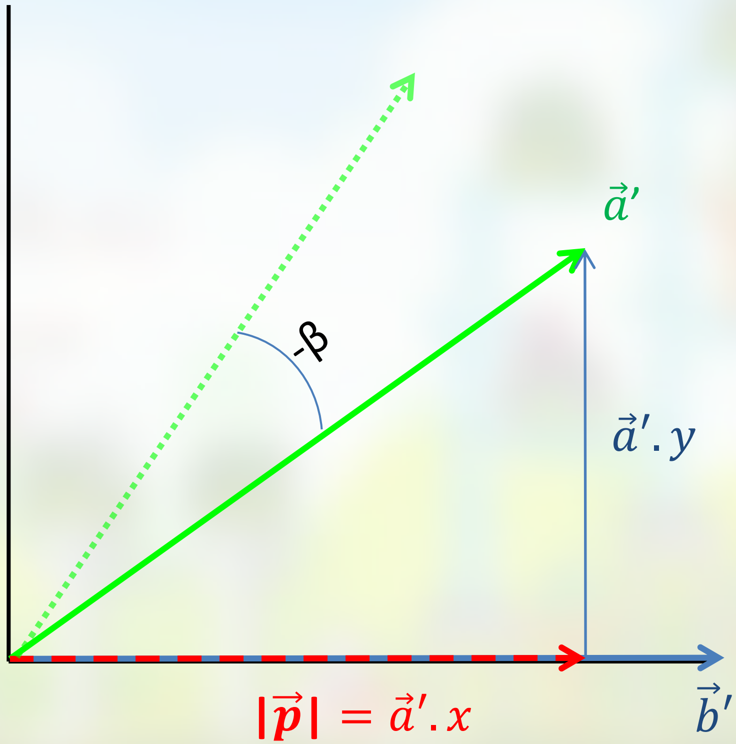


Simplify by rotating

- Observation:
  - if we rotate $\vec{a}$ over –β to $\vec{a}'$
  - we see $|\vec{p}|$ is equal to $\vec{a}'.x$

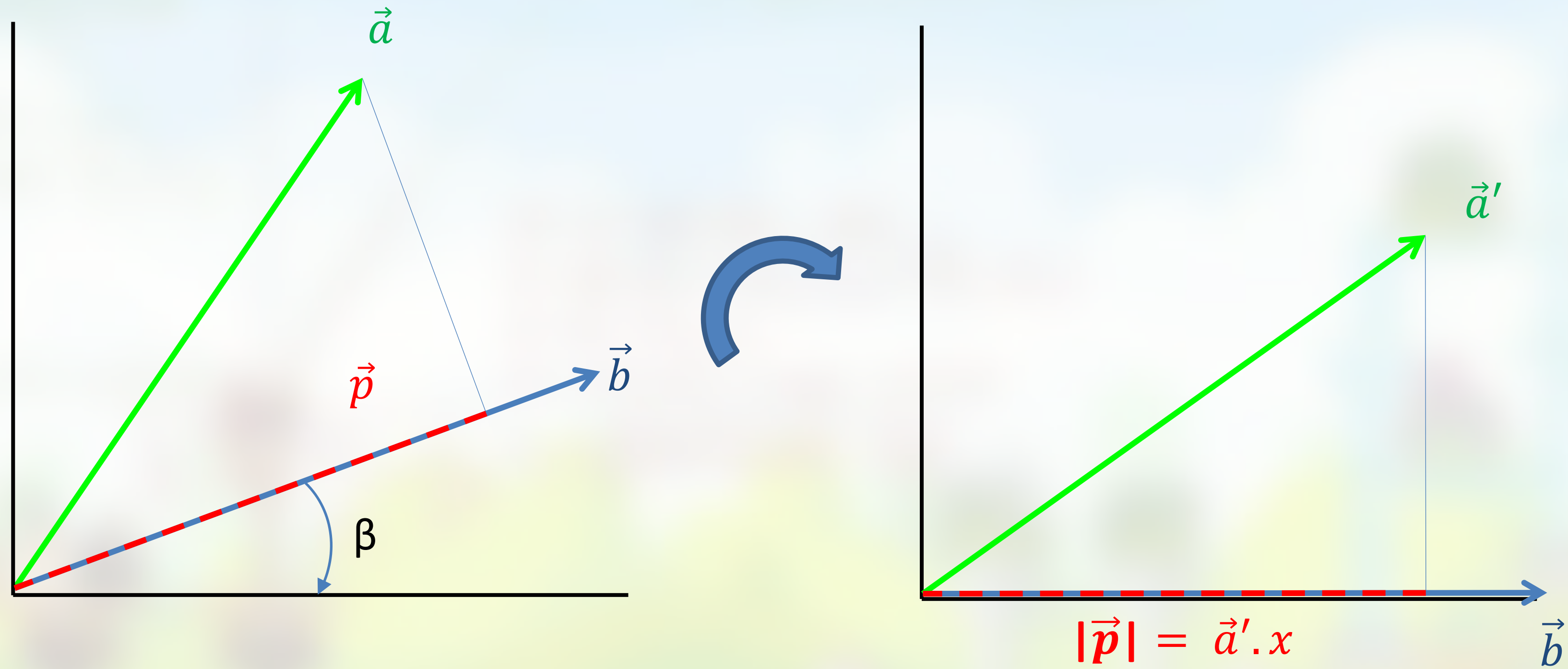


rotate
over -β



Calculate $\vec{a}'.x$ with 2d rotation formula

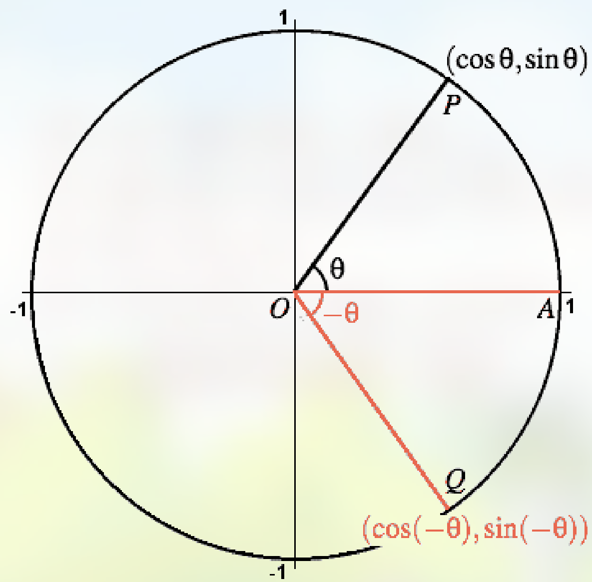
$$\vec{a}'.x = \vec{a}.x * \cos(-\beta) - \vec{a}.y * \sin(-\beta)$$



Simplify using angle “identities”

$$\vec{a}'.x = \vec{a}.x * \cos(-\beta) - \vec{a}.y * \sin(-\beta)$$

$$= \vec{a}.x * \cos(\beta) + \vec{a}.y * \sin(\beta)$$



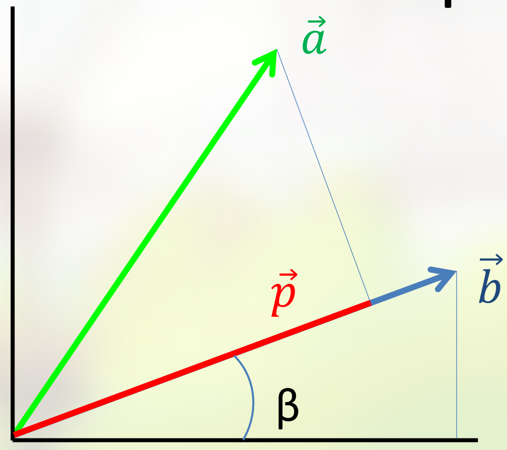
Simplify using Sohcahtoa

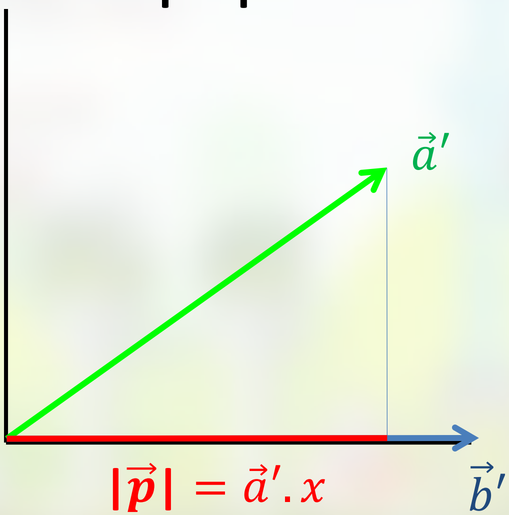
$$\cos(\beta) = \frac{\vec{b}.x}{|\vec{b}|} \;\&\; \sin(\beta) = \frac{\vec{b}.y}{|\vec{b}|}$$

Therefore:

$$\vec{a}'.x = \vec{a}.x * \cos(\beta) + \vec{a}.y * \sin(\beta)$$

$$= \vec{a}.x * \frac{\vec{b}.x}{|\vec{b}|} + \vec{a}.y * \frac{\vec{b}.y}{|\vec{b}|}$$





Simplify notation using definition of $\hat{b}$

$$\vec{a}'.x = \vec{a}.x * \frac{\vec{b}.x}{|\vec{b}|} + \vec{a}.y * \frac{\vec{b}.y}{|\vec{b}|}$$

$$\boxed{= \vec{a}.x * \hat{b}.x + \vec{a}.y * \hat{b}.y = \vec{\mathrm{a}} \bullet \hat{\mathrm{b}}}$$