Lecture 4: Vectors & the Dot Product

The Math Under Everything

CS 453 — Computer Graphics

Fakhir Shaheen

Forman Christian University

What is a Vector

Things Every Game Must Say

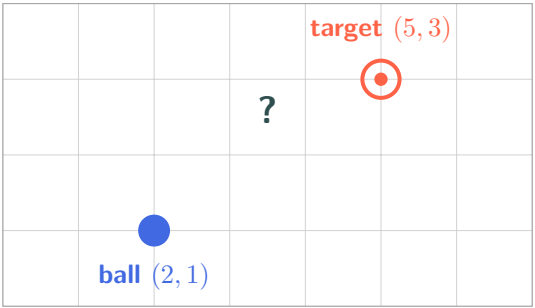
"move the ball **toward** the target"

"is the guard **facing** the player?"

"**bounce** off this wall"

Simple sentences. Let's try to write the first one in code.

Step 1: Draw It Before You Code It



The ball sits at $(2, 1)$.

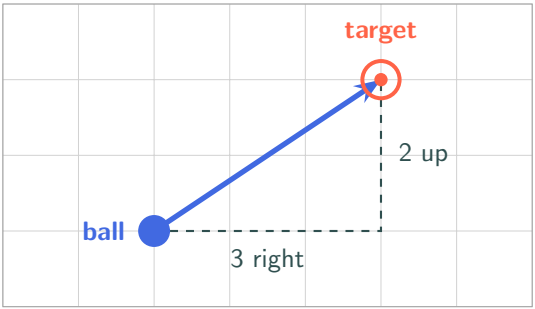
The target sits at $(5, 3)$.

Our job: nudge the ball **a little** toward the target.

Which way is "toward"?

In code we hold four plain numbers: `ball_x`, `ball_y`, `target_x`, `target_y`.

Step 2: How Far Right? How Far Up?



"Toward" means:

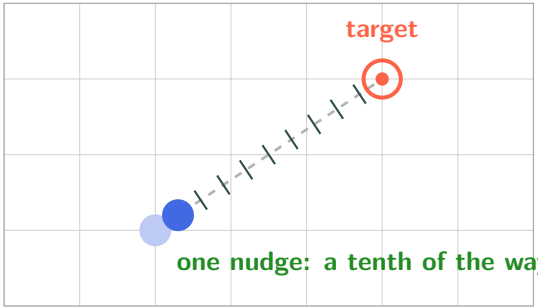
3 to the right, 2 up.

```
dx = target_x - ball_x    # = 3
dy = target_y - ball_y    # = 2
```

Two subtractions:
destination minus start.

Check the picture: start at the ball, walk 3 right, then 2 up — you land on the target.

Step 3: Take a Small Step



Don't jump the whole way — take a **tenth** of it.

```
ball_x = ball_x + dx * 0.1
ball_y = ball_y + dy * 0.1
```

A tenth of 3 right, a tenth of 2 up: the ball lands at $(2.3,\ 1.2)$.

Put it in a loop: each pass recomputes `dx`, `dy` from the new spot and nudges again. The ball creeps toward the target.

It Works. Now Look at the Shape of It

The whole thing, together:

```
dx = target_x - ball_x
dy = target_y - ball_y
ball_x = ball_x + dx * 0.1
ball_y = ball_y + dy * 0.1
```

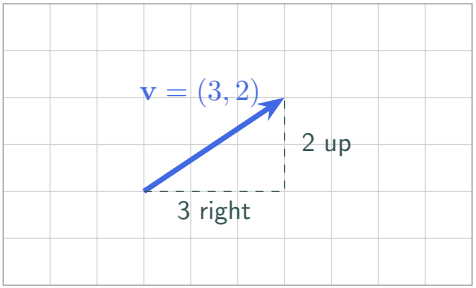
about x	about y
`dx = ...`	`dy = ...`
`ball_x = ...`	`ball_y = ...`

Every line about x has a **twin** about y. Forget one twin, and the bug hunts *you*.

And that was the *easy* sentence — "bounce" needs a dozen twins.

Today's goal: one word per idea, not two lines per idea.

A Vector Is an Arrow



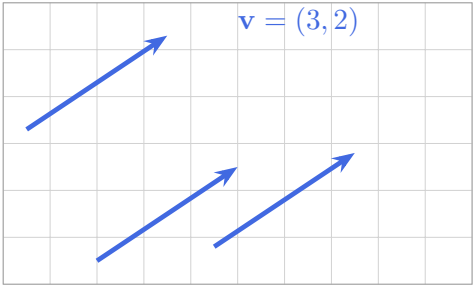
A vector is an **instruction**:

"**3 right, 2 up**"

In code: two floats,
`Vec2(3, 2)`

That pair `(dx, dy) = (3, 2)` from Step 2 was a vector all along.
Same coordinate plane as your A levels — x right, y up.

A Vector Has No Home



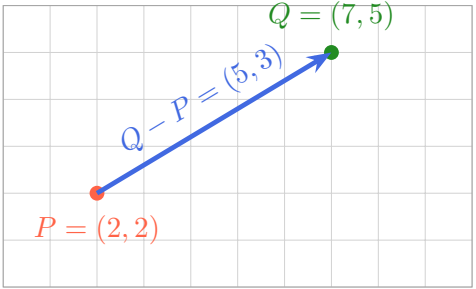
All three arrows are the **same** vector.

A vector remembers its **direction** and **length** —

not where it starts.

"3 right, 2 down" is the same instruction wherever you are standing.

Points vs. Vectors



A **point** is a location:
"*here*"

A **vector** is a trip:
"*that way*"

subtract two points $\Rightarrow$ the trip between them

Which Combinations Make Sense?

$$\text{point} - \text{point} = \text{vector} \quad \text{(the trip between them)}$$

$$\text{point} + \text{vector} = \text{point} \quad \text{(go there: arrive)}$$

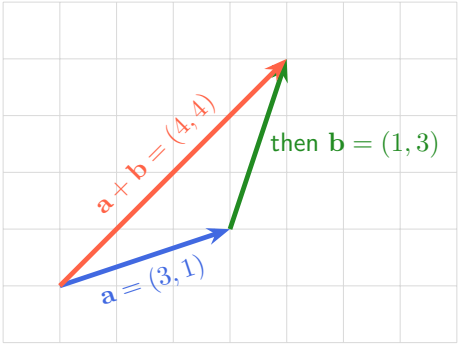
$$\text{vector} + \text{vector} = \text{vector} \quad \text{(two trips, chained)}$$

$$\text{point} + \text{point} = \textbf{nonsense} \quad \text{(add two } \textit{locations}\text{?)}$$

In A2 both are stored in one type, `Vec2` — the difference lives in your **naming**:

`pos`, `center` (points) vs. `dir`, `step`, `vel` (vectors)

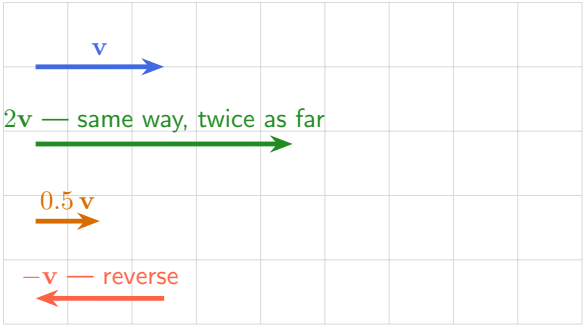
Adding Vectors Is Just Walking



Walk $\mathbf{a}$, then walk $\mathbf{b}$ — the sum is the shortcut.

Componentwise, no surprises: $(3, 1) + (1, 3) = (3{+}1,\ 1{+}3) = (4, 4)$.

Scaling: Stretch, Shrink, Flip



Multiply by a number: the direction stays put (or flips), only the length changes.

$2 \cdot (3, 1) = (6, 2)$ — again, componentwise.

The Sentences Come Back

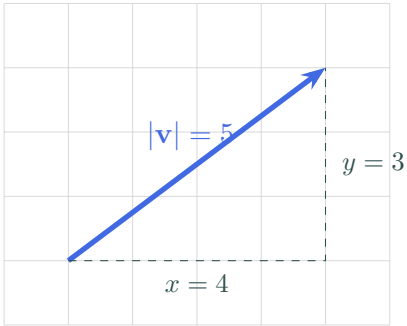
A2's `Vec2` spells these operations +, −, ∗ (that's task T1). Then:

```
# before: four lines of x/y twins
# after:
step = target - pos
pos  = pos + step * 0.1      # 10% of the way there
```

Code that reads like the sentence again.

One thing still missing: `step * 0.1` moves *slower when far, faster when near*? No — the opposite. To move at a **fixed speed**, we need to measure **length**.

Length: Pythagoras, Nothing More



$$|\mathbf{v}| = \sqrt{x^2 + y^2}$$

The arrow is the hypotenuse of its own components.

Distance between two points? $|Q - P|$ — subtract, then measure.

Week-1 Flashback: Skipping the Square Root

You already used this trick

`fill_circle` compared *squared* distances to avoid `sqrt`.

Now it has a name:

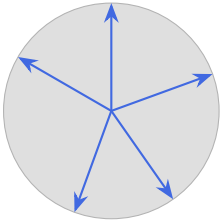
$$|\mathbf{v}|^2 = x^2 + y^2$$

When you only need *"which is bigger?"*, compare squared lengths — the `sqrt` changes nothing about the order.

Unit Vectors: Direction, Purified

Divide a vector by its own length:

$$\hat{\mathbf{v}} = \frac{\mathbf{v}}{|\mathbf{v}|}$$



all unit vectors end on a circle of radius 1

Its length becomes exactly 1:

"**one step that way**"

Keep the direction, throw away the length — so *you* choose the speed.

Fixed-Speed Motion in Two Lines

Now the fix for our "move toward" code:

```
step = (target - pos).normalized()
pos  = pos + step * speed
```

- `normalized()`: direction only, length 1
- `* speed`: exactly `speed` pixels each nudge, near or far

This is A2 Part B's beating heart.

One Vector Breaks It

What is the direction of `Vec2(0, 0)`?

```
step = (target - pos).normalized()    # target == pos ?
```

The zero vector has length $0 \Rightarrow$ **division by zero**.

No direction exists — so *define* the answer

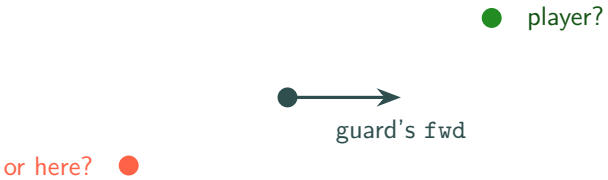
Return `Vec2(0, 0)` for the zero vector.

Guard with `EPSILON`, not `== 0.0` — floats are never exactly zero.

The Dot Product

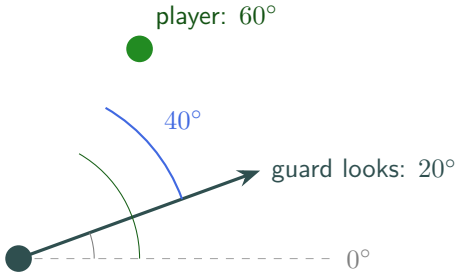
A Simple Question

"Is the guard **facing** the player?"



A yes/no question. How hard can it be?

First Idea: Measure the Angles



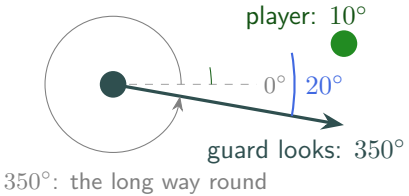
Measure three things:

1. where the guard looks: $20°$
2. where the player is: $60°$
3. the difference: $40°$

Less than a quarter turn $(90°)$
$\Rightarrow$ **facing**. ✓

Sounds fine. Now move the player a little.

Angles Wrap Around — and Bite



The picture says: $20°$ apart.

The subtraction says:

$$10° - 350° = -340°$$

After $359°$ comes $0°$ again.
The *number* jumps; the *picture* doesn't.

There is a way to answer "facing?" **without ever measuring an angle**. It needs one new operation.

The Dot Product: Multiply and Add

$$\mathbf{a} \cdot \mathbf{b} \;=\; a_x\, b_x \;+\; a_y\, b_y$$

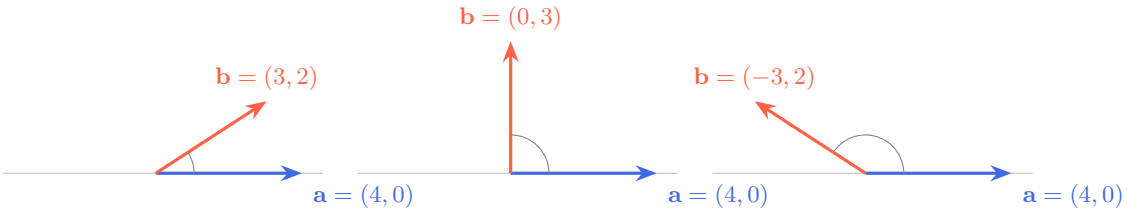
Two vectors go in.

A single **number** comes out — not a vector.

Example: $(4, 0) \cdot (3, 2) \;=\; 4{\cdot}3 + 0{\cdot}2 \;=\; 12$

Cheap: two multiplies, one add. But what does 12 *mean*?

The Sign Tells the Story



acute

$\mathbf{a} \cdot \mathbf{b} = 12 > 0$

roughly same way

right angle

$\mathbf{a} \cdot \mathbf{b} = 0$

perpendicular

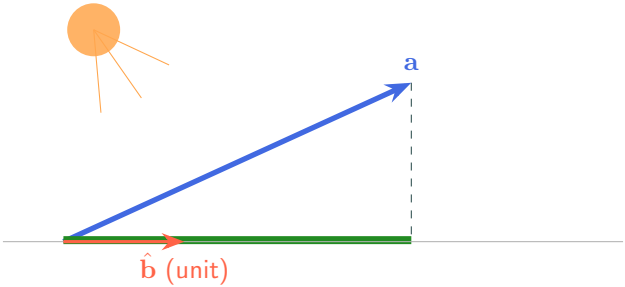
obtuse

$\mathbf{a} \cdot \mathbf{b} = -12 < 0$

roughly opposite

The **sign alone** answers: "same way or not?"

What the Number Means: the Shadow Reading



$$\mathbf{a} \cdot \hat{\mathbf{b}} \;=\; \text{length of } \mathbf{a}\text{'s } \textbf{shadow} \text{ on } \hat{\mathbf{b}}\text{'s line}$$

Sun directly overhead, $\mathbf{a}$ casts a shadow on the ground: that shadow's length *is* the dot product.

Reading the Shadow

shadow points *along* $\hat{\mathbf{b}}$ $\Rightarrow$ **positive**

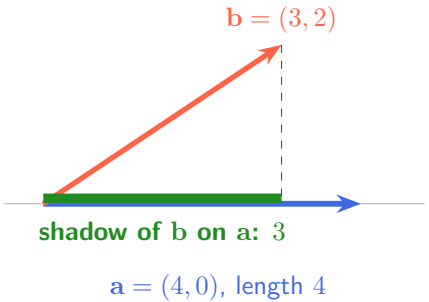
shadow points *against* $\hat{\mathbf{b}}$ $\Rightarrow$ **negative**

$\mathbf{a}$ pure sideways (no shadow) $\Rightarrow$ **zero**

Why believe it in *every* direction? With $\hat{\mathbf{b}} = (1, 0)$: $\mathbf{a} \cdot \hat{\mathbf{b}} = a_x$, literally "how far along x".

Rotating the *pair* together changes neither the picture nor the number — so the reading holds for any $\hat{\mathbf{b}}$.

So What *Was* 12?



$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| \times \text{(shadow of } \mathbf{b} \text{ on } \mathbf{a})$$
$$= 4 \times 3 = 12 \checkmark$$

Or the other way round: $|\mathbf{b}| \times$ (shadow of $\mathbf{a}$ on $\mathbf{b}$'s line) $= \sqrt{13} \times \frac{12}{\sqrt{13}} = 12$. Same number, either way.

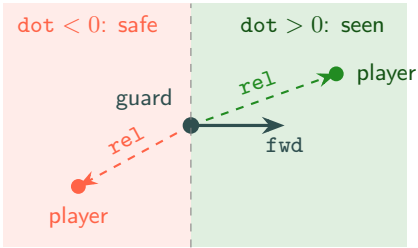
When one vector is **one step long**, its length factor is 1 — and the dot *is* the shadow. That was the last slide.

(For the trig-inclined: the shadow of $\mathbf{b}$ on $\mathbf{a}$ is $|\mathbf{b}| \cos\theta$, so $\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos\theta$ — same statement, more symbols.)

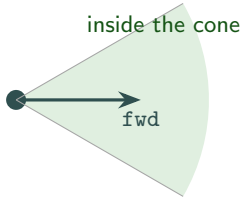
The Payoff: One Line, No Trig

"Is the guard facing the player?"

```
rel = player_pos - guard_pos
facing = fwd.dot(rel) > 0
```



Bonus: a Narrower Cone of Vision

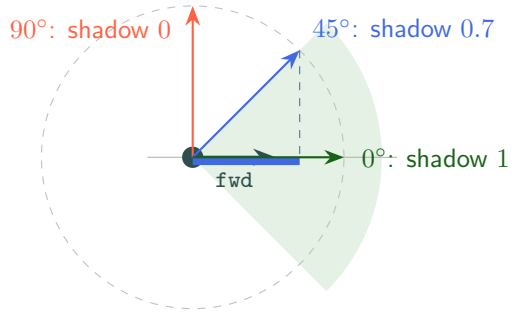


Compare against a threshold instead of 0:

`fwd.dot(rel.normalized()) > 0.7` $\approx$ within $45°$

Bigger threshold, narrower cone. Still no `acos`. But why 0.7?

Why 0.7? Shadows of a One-Step Arrow



every coloured arrow is one step long

Make `rel` **one step long** (that's `normalized()`).

Now the dot is just the **shadow** of a one-step arrow on `fwd`:

- straight ahead: shadow 1
- $45°$ off: shadow 0.7 (the 1–1–$\sqrt{2}$ triangle: $1/\sqrt{2} \approx 0.707$)
- sideways: shadow 0

So "shadow > 0.7" *means* "within $45°$".

Why one step long? A far-away player has a long arrow and a long shadow — distance would leak into the answer. We want direction only.

Perpendicularity: the School Test Breaks

School's test: slopes multiply to -1

$$m_1 \cdot m_2 = -1 \;\Rightarrow\; \text{perpendicular}$$

× a **vertical** line has no slope — divide by zero before you even start

× near-vertical: slopes explode into the millions — float misery

Graphics is *full* of vertical and near-vertical lines. We need a test with no special cases.

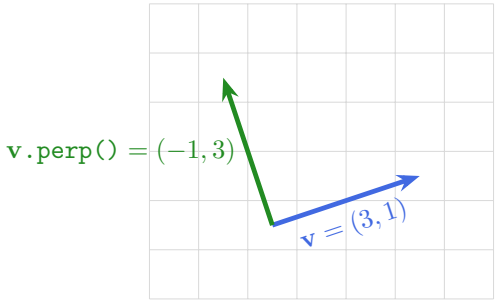
The Dot Product Doesn't Care

$$\mathbf{a} \cdot \mathbf{b} = 0 \iff \text{perpendicular}$$

Works for vertical, horizontal, everything.

(You already saw why: perpendicular $\Rightarrow$ no shadow $\Rightarrow$ zero.)

Making a Perpendicular: the Quarter Turn



$$(x, y) \rightarrow (-y, x)$$

swap, negate one:
a quarter turn.

Verify: $(x, y) \cdot (-y, x) = -xy + yx = 0$ ✓

The Dot's Sibling: the 2D Cross Product

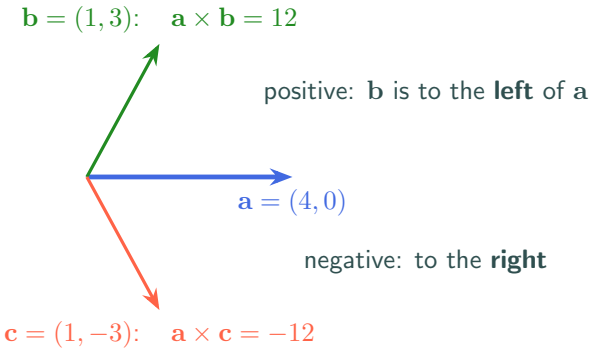
$$\mathbf{a} \times \mathbf{b} = a_x\, b_y - a_y\, b_x$$

Also a single number. Different question:

dot asks: same way, or not?

cross asks: which *side* are you on?

Cross Product: the Side Test



sign $\Rightarrow$ which side zero $\Rightarrow$ parallel

When You Truly Need the Angle

Rarely. But when you do, run the shadow reading backwards:

$$\theta = \arccos\left(\frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}|\,|\mathbf{b}|}\right)$$

The fraction is $\cos\theta$; `acos` recovers $\theta \in [0, \pi]$.

Looks innocent. Let's implement it.

The Innocent Implementation Crashes

```
t = a.dot(b) / (a.length()
                * b.length())
return math.acos(t)
```

```
t = 1.0000000000000002
ValueError:
math domain error
```

Feed it two **parallel** vectors:

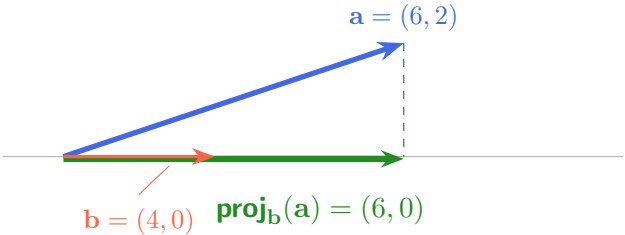
`Vec2(5, 6)` and `Vec2(25, 30)` ...

$\cos\theta$ can only be $-1 \ldots 1$, but float rounding lands a hair outside — and `acos` refuses.

The fix

`t = max(-1.0, min(1.0, t))` before `acos`.
Also: a zero-length input has no angle — return `0.0`. Both are checker cases.

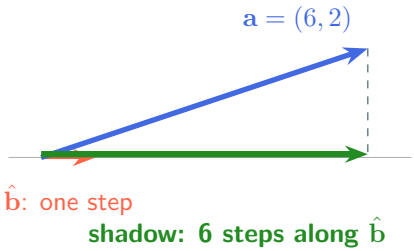
Projection: Keeping Only the "Along" Part



The projection is the **shadow itself, as a vector** —
not just its length.

Here: $\mathbf{a} = (6, 2)$ on the x-axis keeps its "along" part $(6, 0)$ and drops the rest.

Projection, Step by Step: Unit $\hat{\mathbf{b}}$ First



1. How long is the shadow?
The shadow reading: $\mathbf{a} \cdot \hat{\mathbf{b}}$
$(6, 2) \cdot (1, 0) = 6$ — a number

2. Turn it into an arrow.
Walk that many steps along $\hat{\mathbf{b}}$:

$$(\mathbf{a} \cdot \hat{\mathbf{b}})\,\hat{\mathbf{b}}$$

$6 \cdot (1, 0) = (6, 0)$ — a vector ✓

That is the whole idea: *measure* the shadow, then *walk* that far along $\hat{\mathbf{b}}$. One catch: it needs $\hat{\mathbf{b}}$ to be one step long.

Same Recipe When $\mathbf{b}$ Is Any Length

Usually we hold $\mathbf{b} = (4, 0)$, not $\hat{\mathbf{b}}$. But we know how to shrink it: $\hat{\mathbf{b}} = \mathbf{b}/|\mathbf{b}|$.

Put that into step 2, twice — once for the shadow, once for the walk:

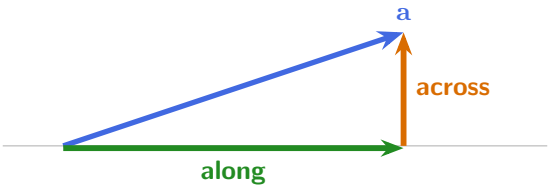
$$\left(\mathbf{a} \cdot \frac{\mathbf{b}}{|\mathbf{b}|}\right)\frac{\mathbf{b}}{|\mathbf{b}|} \;=\; \frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{b}|^2}\,\mathbf{b} \;=\; \frac{\mathbf{a}\cdot\mathbf{b}}{\mathbf{b}\cdot\mathbf{b}}\,\mathbf{b}$$

The last step is Pythagoras from earlier: $|\mathbf{b}|^2 = b_x^2 + b_y^2 = \mathbf{b}\cdot\mathbf{b}$.

Check with the same numbers

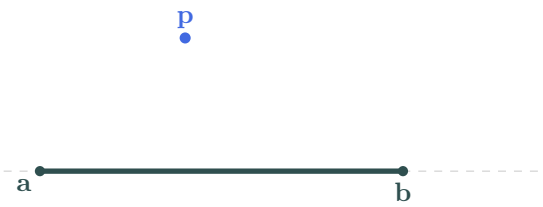
$\mathbf{a}\cdot\mathbf{b} = 6{\cdot}4 + 2{\cdot}0 = 24, \quad \mathbf{b}\cdot\mathbf{b} = 16, \quad \dfrac{24}{16} = 1.5, \quad 1.5\cdot(4,0) = (6,0)$ ✓

Every Vector Splits in Two



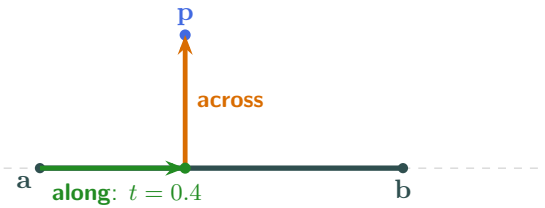
$$\mathbf{a} = \underbrace{\text{proj}_{\mathbf{b}}(\mathbf{a})}_{\text{along } \mathbf{b}} + \underbrace{\big(\mathbf{a} - \text{proj}_{\mathbf{b}}(\mathbf{a})\big)}_{\text{across}}$$

The Split, Used Once More: Nearest Point on a Segment



Which point of the segment $\mathbf{ab}$ is nearest to $\mathbf{p}$?

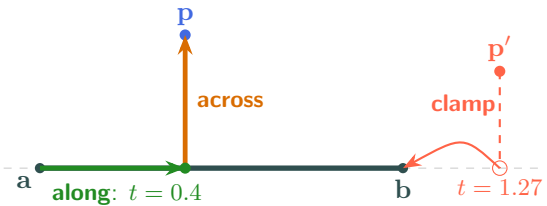
The Split, Used Once More: Nearest Point on a Segment



Which point of the segment $\mathbf{ab}$ is nearest to $\mathbf{p}$?

Split $\mathbf{p} - \mathbf{a}$ against $\mathbf{b} - \mathbf{a}$. The **along** part ends at the answer; the **across** part *is* the distance.

The Split, Used Once More: Nearest Point on a Segment

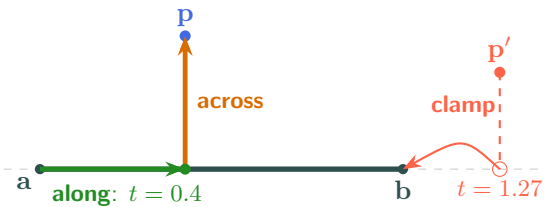


Which point of the segment $\mathbf{ab}$ is nearest to $\mathbf{p}$?

Split $\mathbf{p} - \mathbf{a}$ against $\mathbf{b} - \mathbf{a}$. The **along** part ends at the answer; the **across** part *is* the distance.

One new step. For $\mathbf{p}'$ the shadow lands *past* $\mathbf{b}$. A segment ends, so **clamp** t **to** $[0, 1]$: the answer is $\mathbf{b}$ itself.

The Split, Used Once More: Nearest Point on a Segment



Which point of the segment $\mathbf{ab}$ is nearest to $\mathbf{p}$?

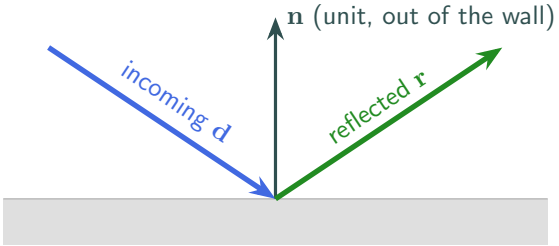
Split $\mathbf{p} - \mathbf{a}$ against $\mathbf{b} - \mathbf{a}$. The **along** part ends at the answer; the **across** part *is* the distance.

One new step. For $\mathbf{p}'$ the shadow lands *past* $\mathbf{b}$. A segment ends, so **clamp** t **to** $[0, 1]$: the answer is $\mathbf{b}$ itself.

```
seg = b - a
t   = (p - a).dot(seg) / seg.dot(seg)    # projection, as a fraction
t   = max(0.0, min(1.0, t))              # segment, not line
return a + seg * t                       # lerp; distance = |p - that|
```

`seg.dot(seg)` can be 0 ($\mathbf{a} = \mathbf{b}$) — guard it, return $\mathbf{a}$.

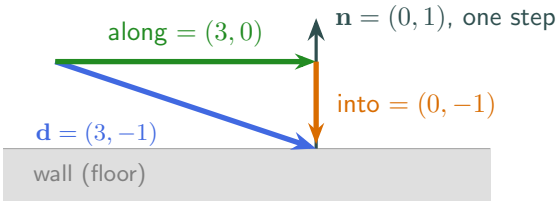
The Grand Payoff: Bouncing Off a Wall



Split $\mathbf{d}$ against the wall's normal:

along the wall $\Rightarrow$ **keep it** *into the wall* $\Rightarrow$ **flip it**

Bouncing Off a Wall: Split $\mathbf{d}$ Against $\mathbf{n}$



Into the wall: project $\mathbf{d}$ onto $\mathbf{n}$.
$\mathbf{n}$ is one step long, so the shadow recipe is just:

$$\text{into} = (\mathbf{d} \cdot \mathbf{n})\,\mathbf{n}$$

$\mathbf{d} \cdot \mathbf{n} = 3 \cdot 0 + (-1) \cdot 1 = -1$, so into $= -1 \cdot (0, 1) = (0, -1)$.

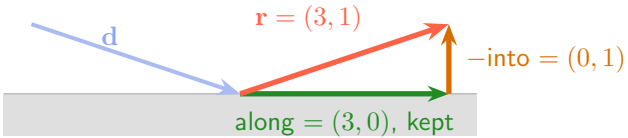
Along the wall: whatever is left.

$$\text{along} = \mathbf{d} - \text{into}$$

$(3, -1) - (0, -1) = (3, 0)$.

Same split as the last slide: $\mathbf{d} =$ along $+$ into. Walk 3 right, then 1 down — you trace $\mathbf{d}$. ✓

Keep the Along Part, Flip the Into Part



Keep "along", reverse "into":

$$\mathbf{r} = \text{along} - \text{into}$$

Now write along as $\mathbf{d}-$ into:

$$\mathbf{r} = (\mathbf{d} - \text{into}) - \text{into} = \mathbf{d} - 2\,\text{into}$$

And into is $(\mathbf{d} \cdot \mathbf{n})\,\mathbf{n}$:

$$\boxed{\mathbf{r} = \mathbf{d} - 2\,(\mathbf{d} \cdot \mathbf{n})\,\mathbf{n}}$$

Check: $(3,-1) - 2(-1)(0,1) = (3,-1) + (0,2) = (3,1)$ ✓ — 3 right as before, but now 1 *up*.

Where this formula goes (T6)

A2 balls bounce with it **this week** — and your ray tracer renders **mirrors** with it, verbatim, in Week 6.

Your Turn

The Cheat Sheet: One Operation, Five Questions

You want to know	**Use**
same way or not?	the **sign** of $\mathbf{a} \cdot \mathbf{b}$ — no angle needed
how long?	$\lvert\mathbf{a}\rvert = \sqrt{\mathbf{a} \cdot \mathbf{a}}$
the actual angle?	$\arccos$ of the ratio — *clamped*
how much of $\mathbf{a}$ along $\mathbf{b}$?	$\dfrac{\mathbf{a} \cdot \mathbf{b}}{\mathbf{b} \cdot \mathbf{b}}\,\mathbf{b}$
how do I bounce?	$\mathbf{d} - 2(\mathbf{d} \cdot \mathbf{n})\mathbf{n}$

Stuck on a geometry problem? Ask *"what would a dot product say here?"*
It is right more often than it has any business being.

Before Wednesday

1. Re-read the **shadow reading** — the whole dot product lives on that one picture
2. Work these **by hand**, no computer:
   - $|(3, 4)|$
   - $(4, 0) \cdot (0, 3)$ — and say *why* in one word
   - the ball $\mathbf{d} = (2, -3)$ bounces off a floor, $\mathbf{n} = (0, 1)$: find $\mathbf{r}$
3. **one flip to remember**: on screen, y grows *down* (Week 1!). Every formula today still holds verbatim — only your *drawings* appear mirrored top-to-bottom

Exit Ticket

Today's questions *(2 minutes)*

(a) $\mathbf{v} = (3, 4)$. What is $|\mathbf{v}|$?

(b) Guard faces $\mathbf{f} = (1, 0)$; the player sits at displacement $(-2, 5)$ from the guard. Compute the dot product — can the guard see the player?

CS 453 — Exit Ticket #4

Name: _____

Roll no: _____

(a) _____

(b) _____

hand in on your way out

Wednesday: give every curve a dial named t — parametric lines, circles & ellipses, and motion.