Getting There in Style

A brief introduction to interpolation
and control systems

Fletcher Dunn

VALVE

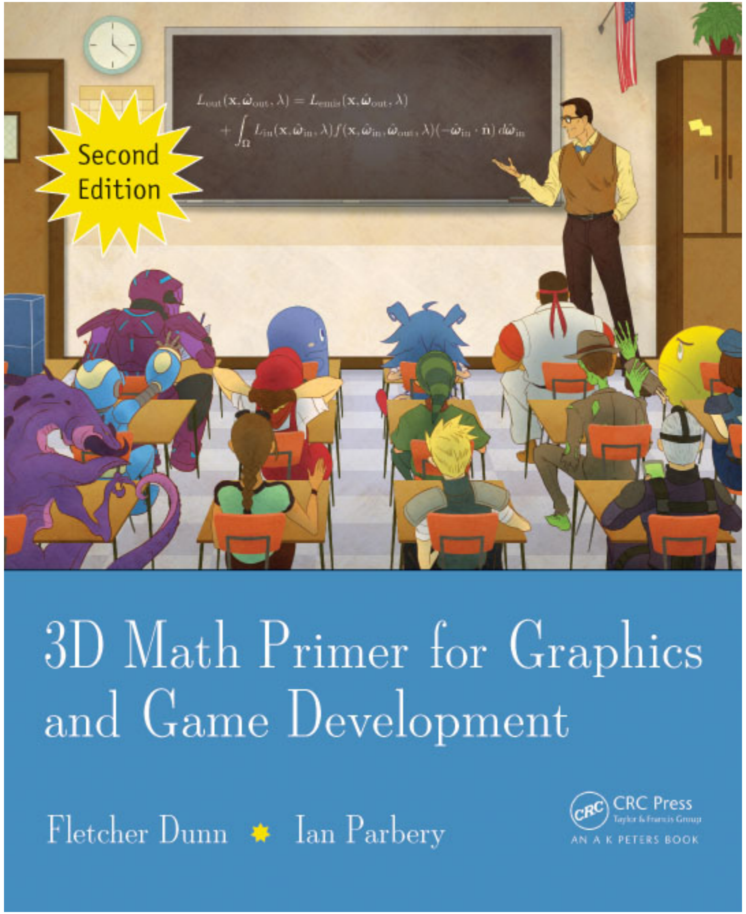
For more info

gamemath.com/gdc2021

Interactive demos
Stuff that wouldn't fit here

Free 800-page
math book!

gamemath.com



Common task: animate something from raw input

Raw Input	Output
UI element is highlighted Y/N	Intensity of highlight
WASD key state	Character velocity
Door open/closed	Door position
Angle to targeted enemy	Angle of arms / turret
Actual HP	HP bar length in HUD
Ideal camera position	Camera position

Discrete-valued input

Goals

Smoth over "pops"

Make it look cool

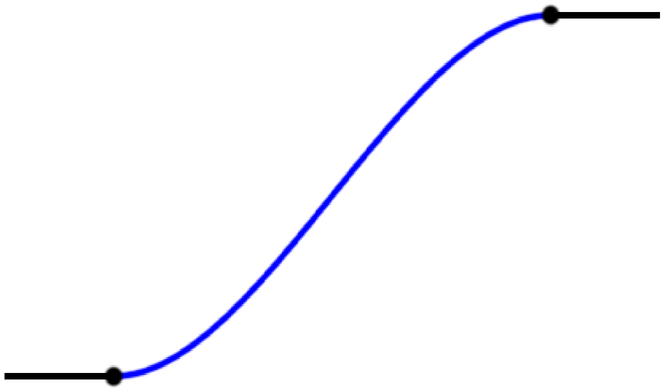
Continuous-valued input

Transition vs control system

Timed Transitition

Fixed duration or finish time

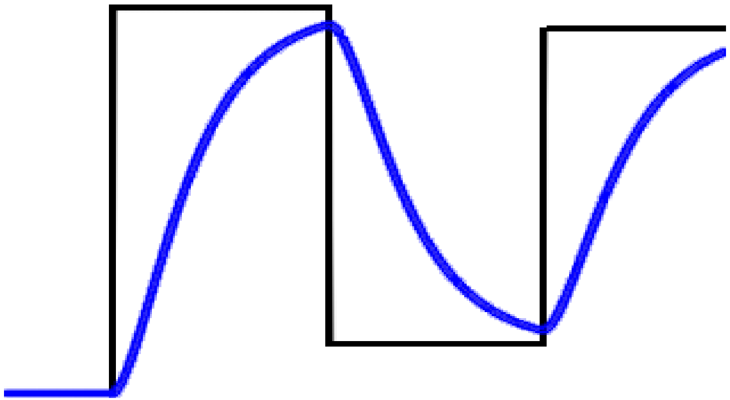
Inactive most of the time



Control System

Indefinite duration

Active even in steady state



Where are we going?

Timed transitions

- Basic lerp
- Smoothstep
- Chase transitions

Control systems

- First-order lag
- PD controller

Review of lerp() function

Linear Int**erp**olation

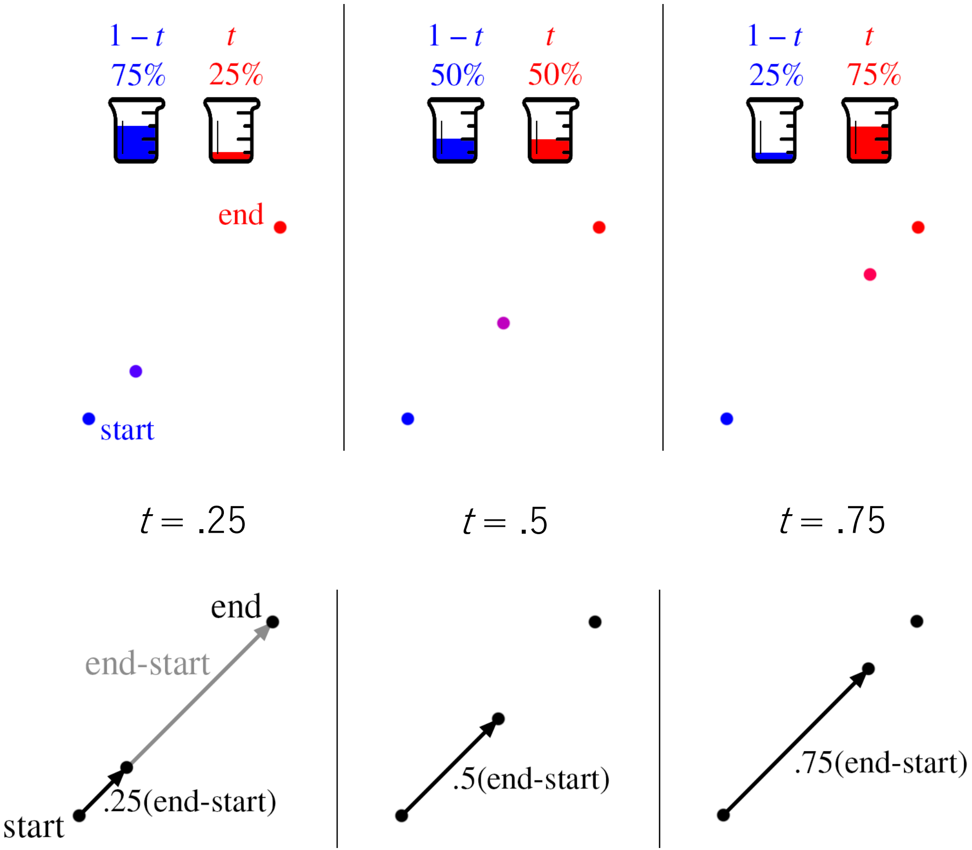
lerp(start, end, t)

$$(1-t)(\text{start}) + t(\text{end})$$

$1-t$ and t are blend weights
("barycentric coords")

$$\text{start} + t(\text{end} - \text{start})$$

t is fraction of delta to apply

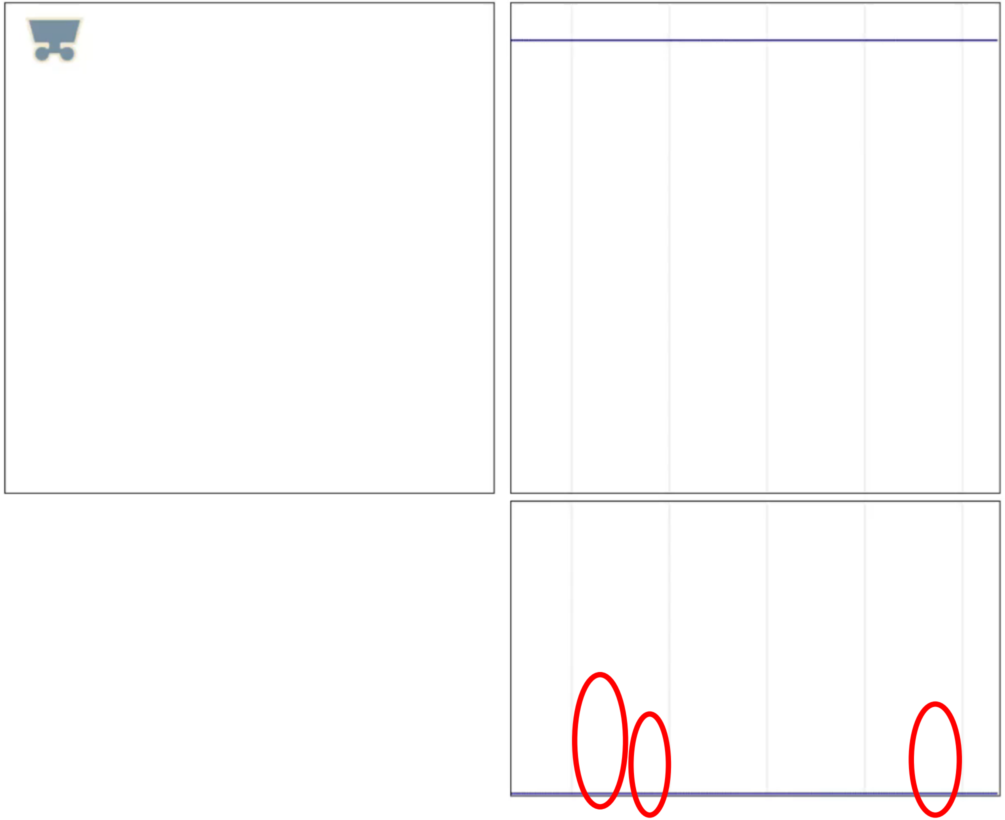


Lerp transition - Basic implementation

```
Vec cur;

float t; // 0 ... 1
Vec start; // Starting pos


// Initialize transtion from
// current position
void Begin() {
  start=cur;
  t=0.0;
}
```

```
// Update transition towards target.
// dt is simulation timestep (seconds)
// Return true if transition finished
bool Update(Vec target, float dt) {

  // Normalize to get fraction of total
  // transition consumed this frame
  dt /= kTransitionDuration;

  // Advance time.  Transition done?
  t += dt;
  if (t>=1.0) { cur=target; return true; }

  // Set new position
  cur = lerp(start, target, t);
  return false; // Transition not complete

}
```

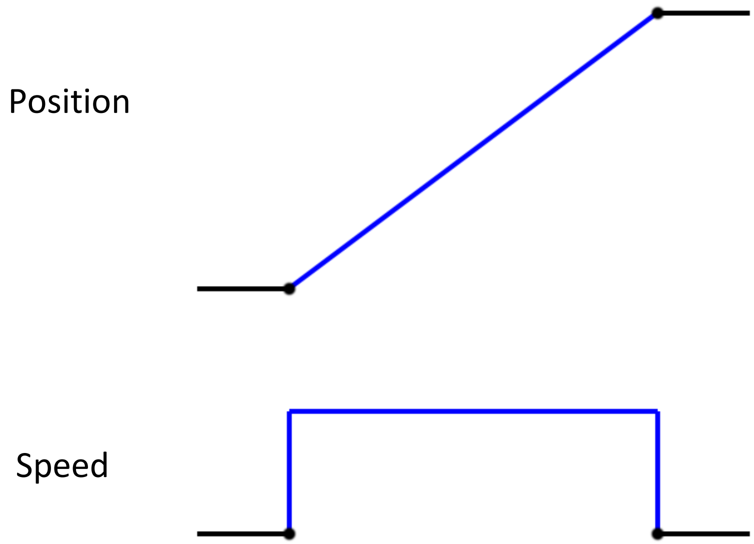
Lerp transition - Behaviour

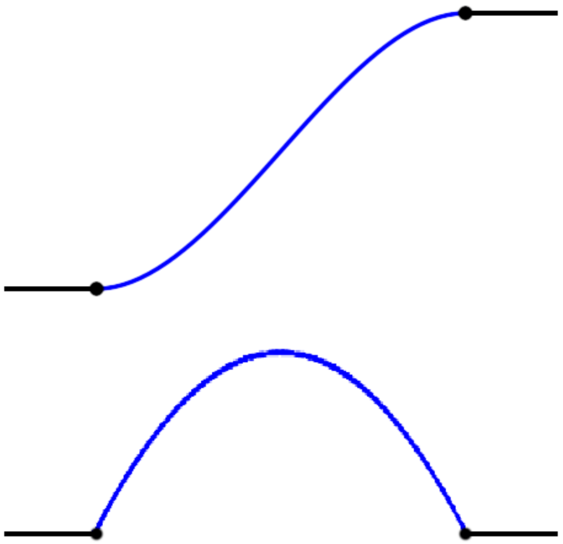


Position

Speed

Lerp transition exhibits "lurch"





Instantaneous change in velocity.
Infinite acceleration is non-physical.

We would like velocity
to be continuous.

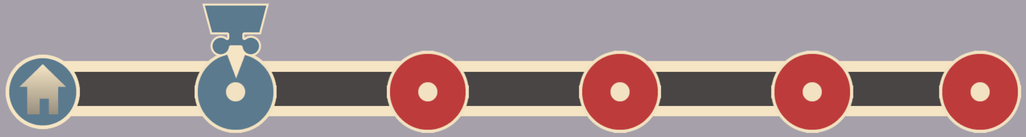
We’re on our way

Timed transitions

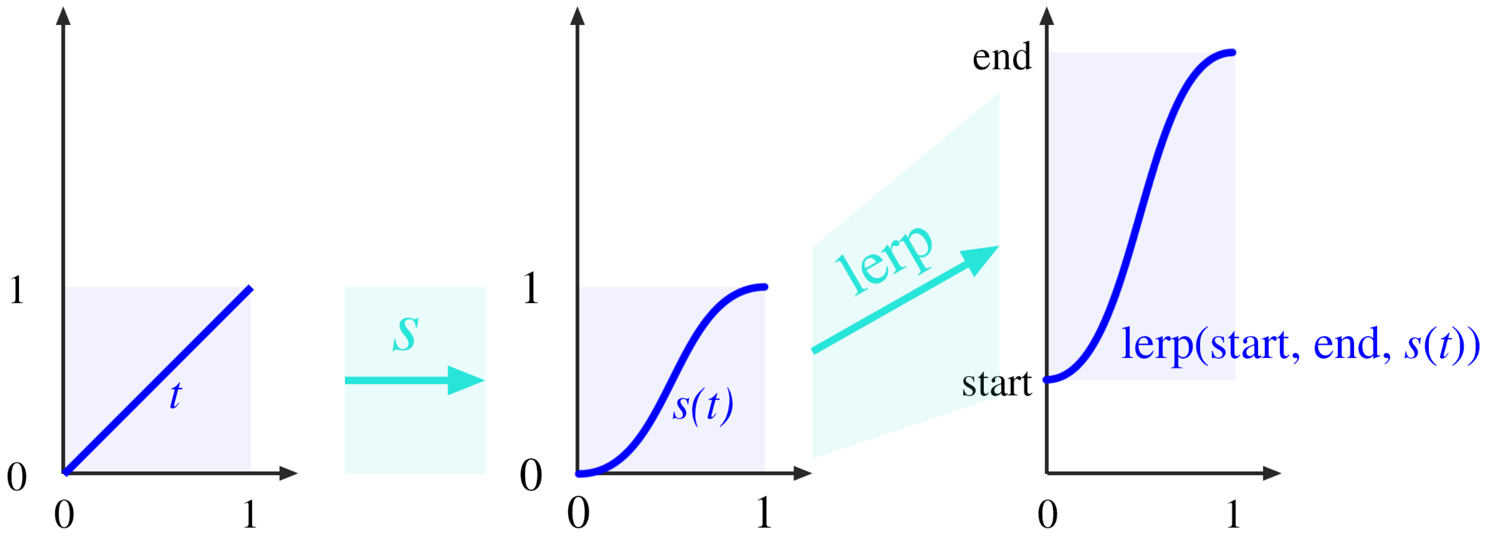
- Basic lerp
- **Smoothstep**
- Chase transitions

Control systems

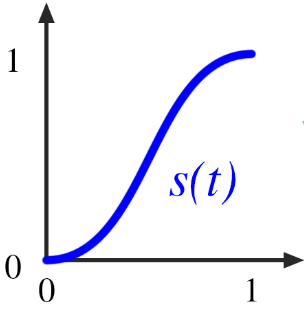
- First-order lag
- PD controller



Smoothstep – Remap unit interval



Smoothstep – Discovering



Goals

$s(0) = 0 \qquad s(1) = 1$

$s'(0) = 0$

$s'(1) = 0$

"Hermite form"

What form? Let's try polynomial.

$s(t) = at^3 + bt^2 + ct + d$

"Monomial form"

Algebra go brrrrr......

$$\boxed{s(t) = 3t^2 - 2t^3}$$

"The smoothstep function"

Transition using lerp

```
Vec cur;
float t;
Vec start;
void Begin() {
  start=cur; t=0.0;
}
bool Update(Vec target, float dt) {
  dt /= kTransitionDuration;

  // Advance time.  Transition done?
  t += dt;
  if (t>=1.0) { cur=target; return true; }

  // Set new position
  cur = lerp(start, target, t);
  return false; // Transition not complete
}
```

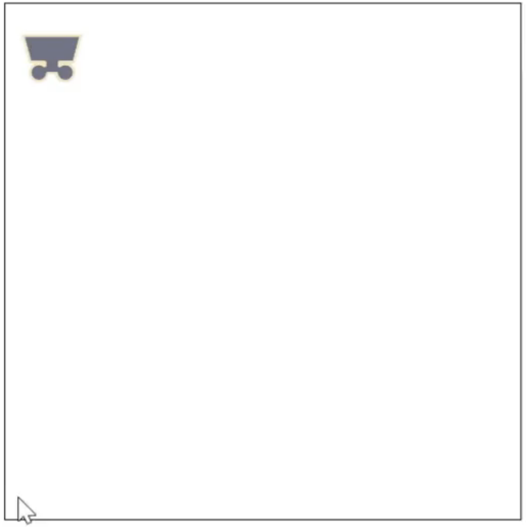
Transition using smoothstep

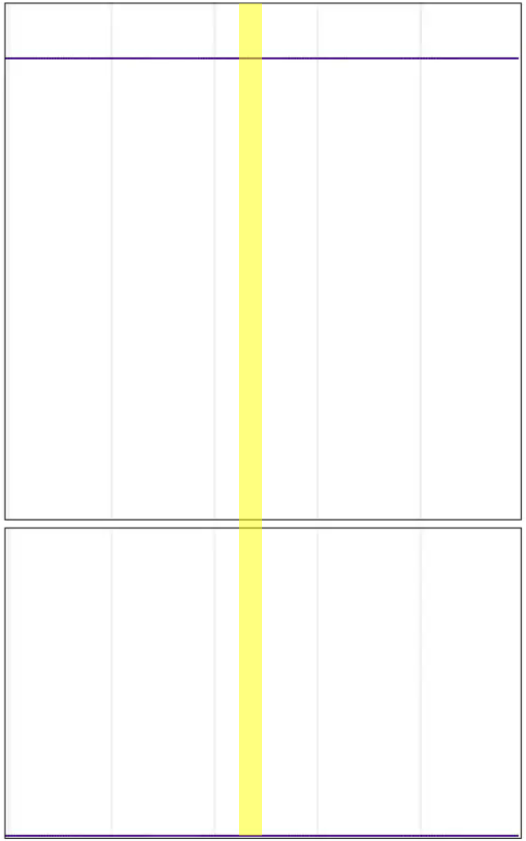
```
Vec cur;
float t;
Vec start;
void Begin() {
  start=cur; t=0.0;
}
bool Update(Vec target, float dt) {
  dt /= kTransitionDuration;

  // Advance time.  Transition done?
  t += dt;
  if (t>=1.0) { cur=target; return true; }

  // Set new position
  cur = lerp(start, target, 3*t*t-2*t*t*t);
  return false; // Transition not complete
}
```

Smoothstep transition - Behaviour

Smoothstep

Lerp

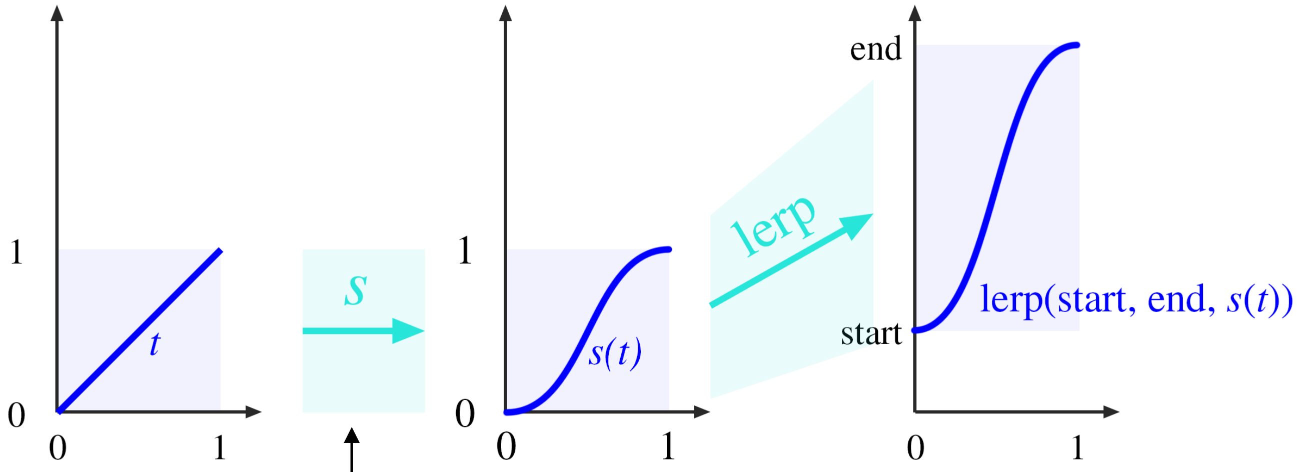


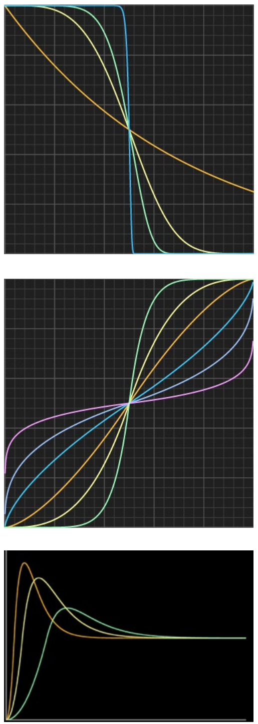


Position

Speed

Remapping the unit interval is fun!





Try other functions here

iquilezles.org/www/articles/functions/functions.htm

Next stop

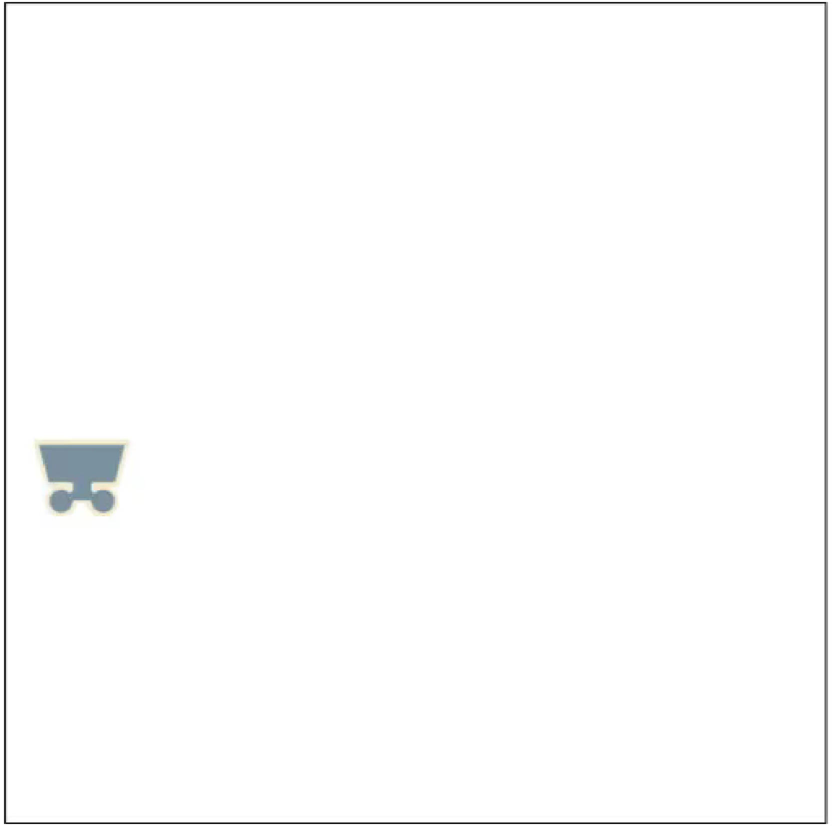
Timed transitions

- Basic lerp
- Smoothstep
- **Chase transitions**

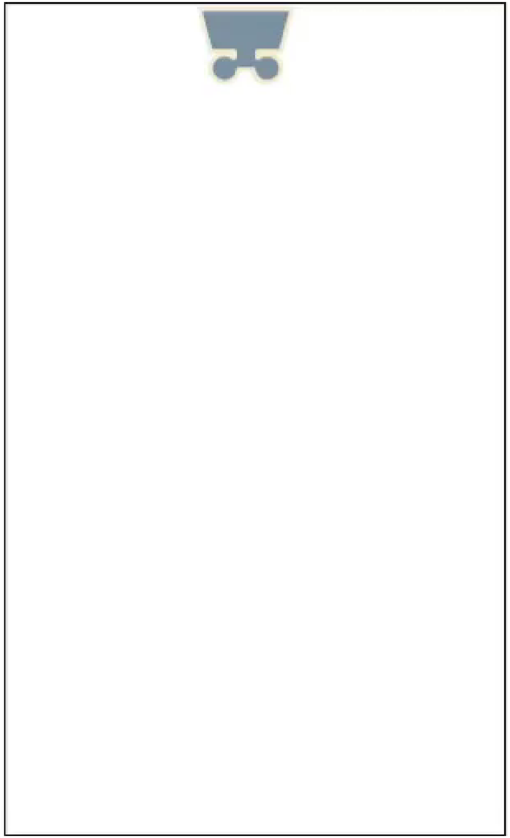
Control systems

- First-order lag
- PD controller

Moving targets – It works....sort of



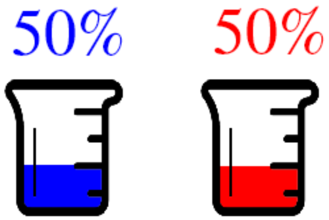
Moving targets – The target approacheth



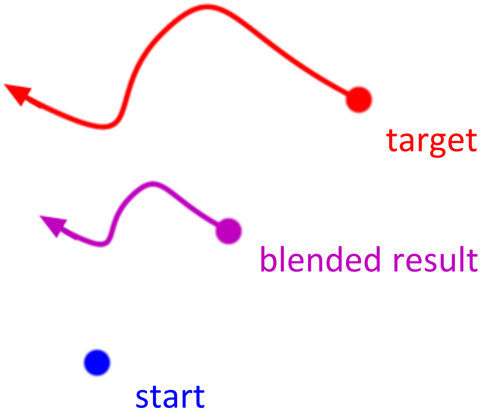
Moving targets – Jump around

Moving targets – Understanding the Problem

Lerp is “blending” start & target



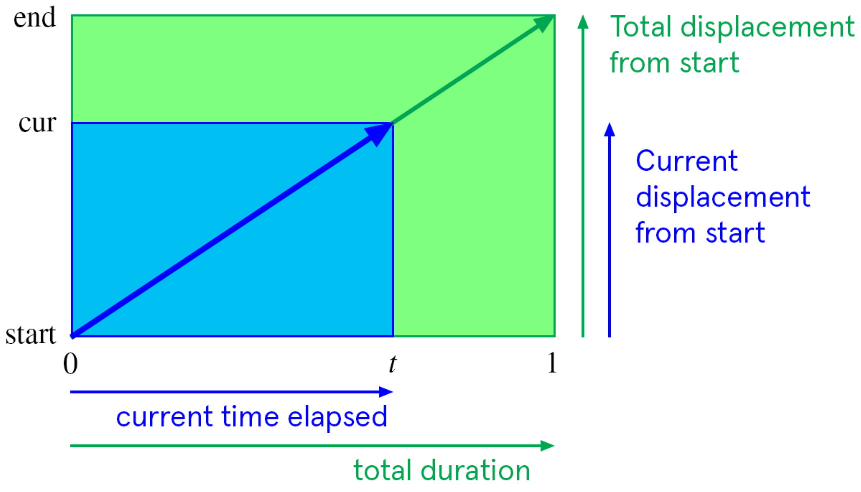
We are blending the motion, too!



“Chase-style” transition

Standard interpolation transition

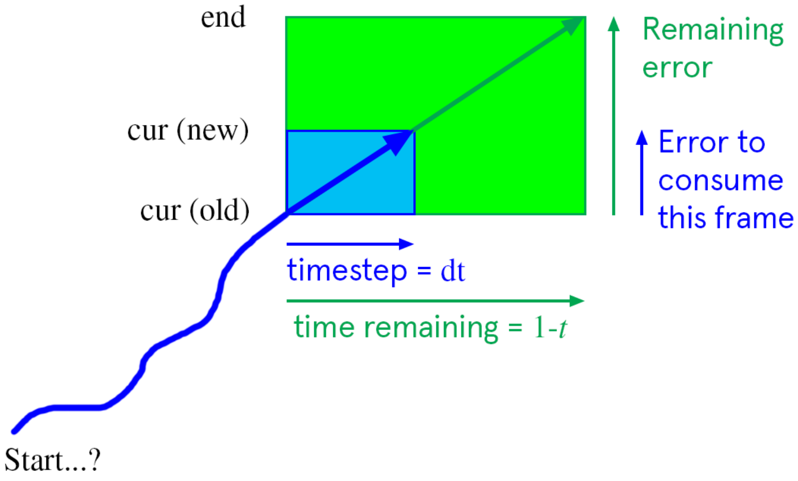
Remember where we started



What fraction of **total** displacement should have been consumed **at the current time**?

Chase-style transition

Only know where we are now



What fraction of **remaining** displacement should we consume **during this timestep**?

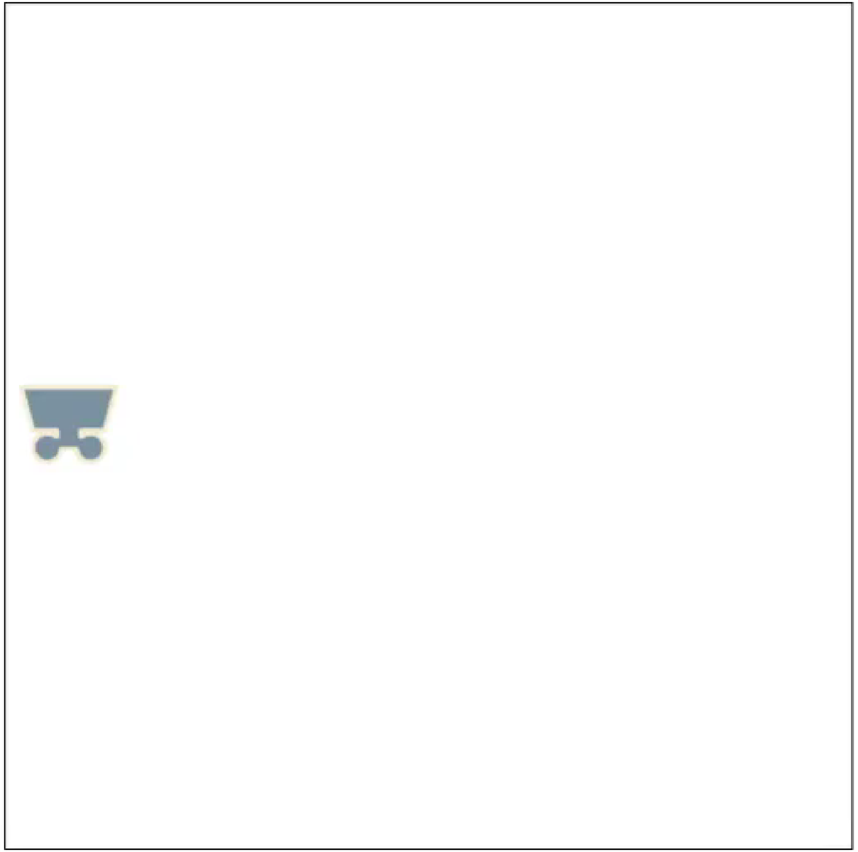
Standard lerp

```
Vec cur;
float t;
Vec start;
void Begin() {
  start=cur; t=0.0;
}
bool Update(Vec target, float dt) {
  dt /= kTransitionDuration;


  // Advance time.  Transition done?
  t += dt;
  if (t>=1.0) { cur=target; return true; }

  // Set new position
  cur = lerp(start, target, t);
  return false; // Transition not complete
}
```

Chase-style lerp

```
Vec cur;
float t;

void Begin() {
  t=0.0;
}
bool Update(Vec target, float dt) {
  dt /= kTransitionDuration;
  float frac = dt/(1-t); // Frac to consume

  // Advance time.  Transition done?
  t += dt;
  if (t>=1.0) { cur=target; return true; }

  // Consume fraction of remaining error
  cur += (target-cur) * frac;
  return false; // Transition not complete
}
```

Chase-style transitions in action



If target doesn't move, same result

Look better when target moves?
You decide

Always moves towards target
Speed is more variable

Doesn't "run away"

Motion always continuous

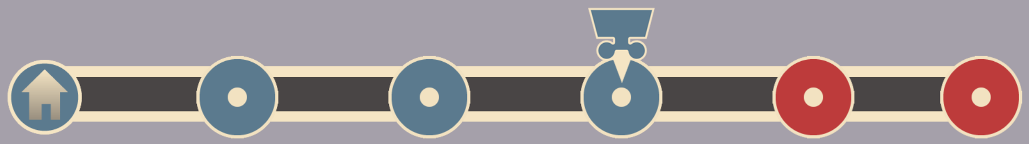
Are we there yet?

Timed transitions

- Basic lerp
- Smoothstep
- Chase transitions

Control systems

- **First-order lag**
- PD controller



First-order lag – You've probably used it before

Ever seen code like this?

```
Vec cur; // state variable
void Update(Vec target) {
  cur += (target - cur) * k;
}
```

With frame rate compensation?

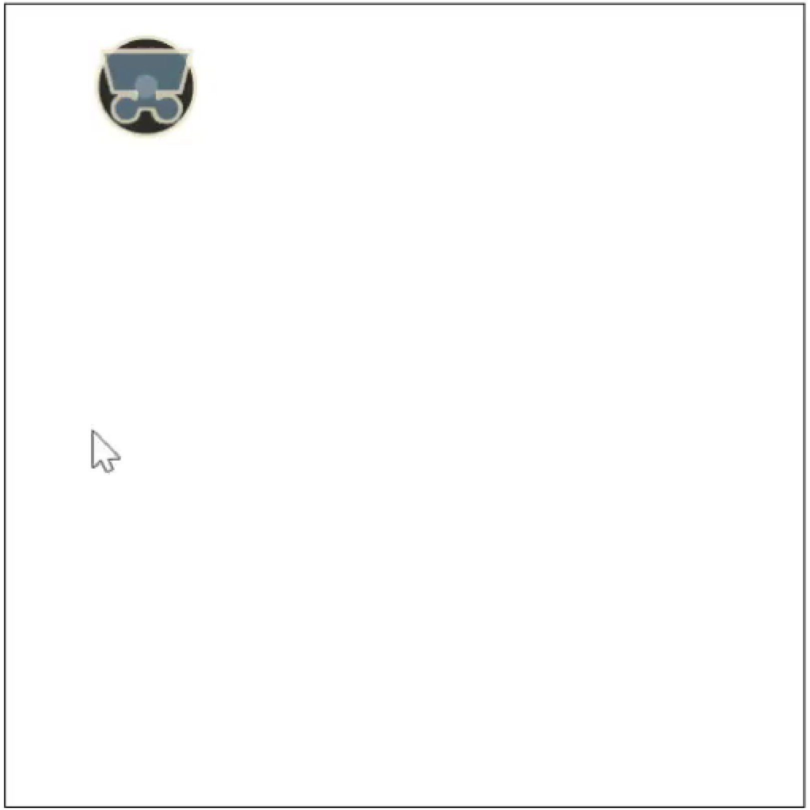
```
void Update(Vec target, float dt) {
  cur += (target - cur) * k*dt;
}
```

Or maybe something like this:

```
void Update(Vec target) {
  cur = lerp(cur,target,k);
}
```

```
void Update(Vec target, float dt) {
  cur = lerp(cur,target,k*dt);
}
```

First-order lag - Behaviour



Characteristic behaviours:

"Rubber band" feel

Target jump causes "lurch"

Decelerate as we approach

Relatively long time to "settle"

First-order lag - Math

```
cur += (target - cur) * k * dt
```

```
y += (x - y) * k * dt
```

$$dy = k(x - y)\ dt$$

$$\boxed{dy/dt = k(x - y)}$$

$x(t)$ = input signal / “target”

$y(t)$ = output signal / “cur”

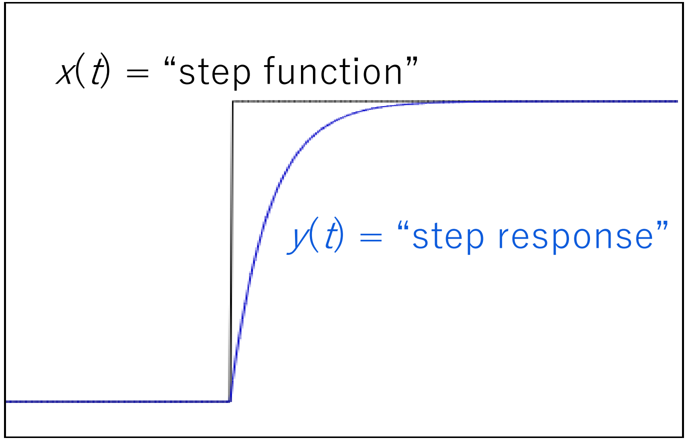
dy = “change in y”

dy/dt = “velocity”

(“First order” because only 1st derivatives appear in diff eq’s.)

First-order lag - Step response

Position



Velocity

Characteristic behaviours

Response *value* is continuous

But velocity is not.

There’s a “lurch”

Decellerate as approach target.

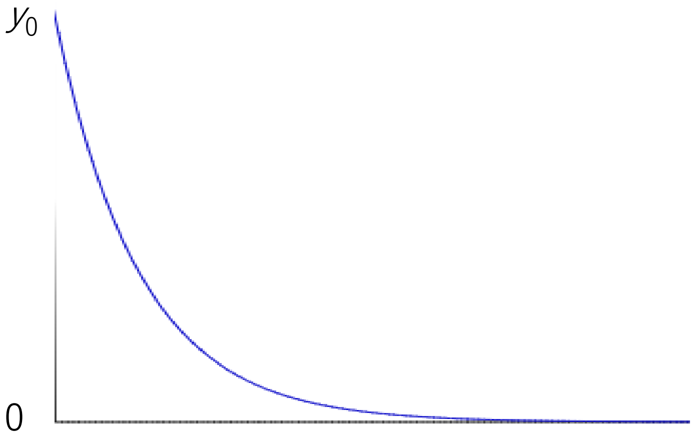
In theory never reach target!

First-order lag - Analysis of step response

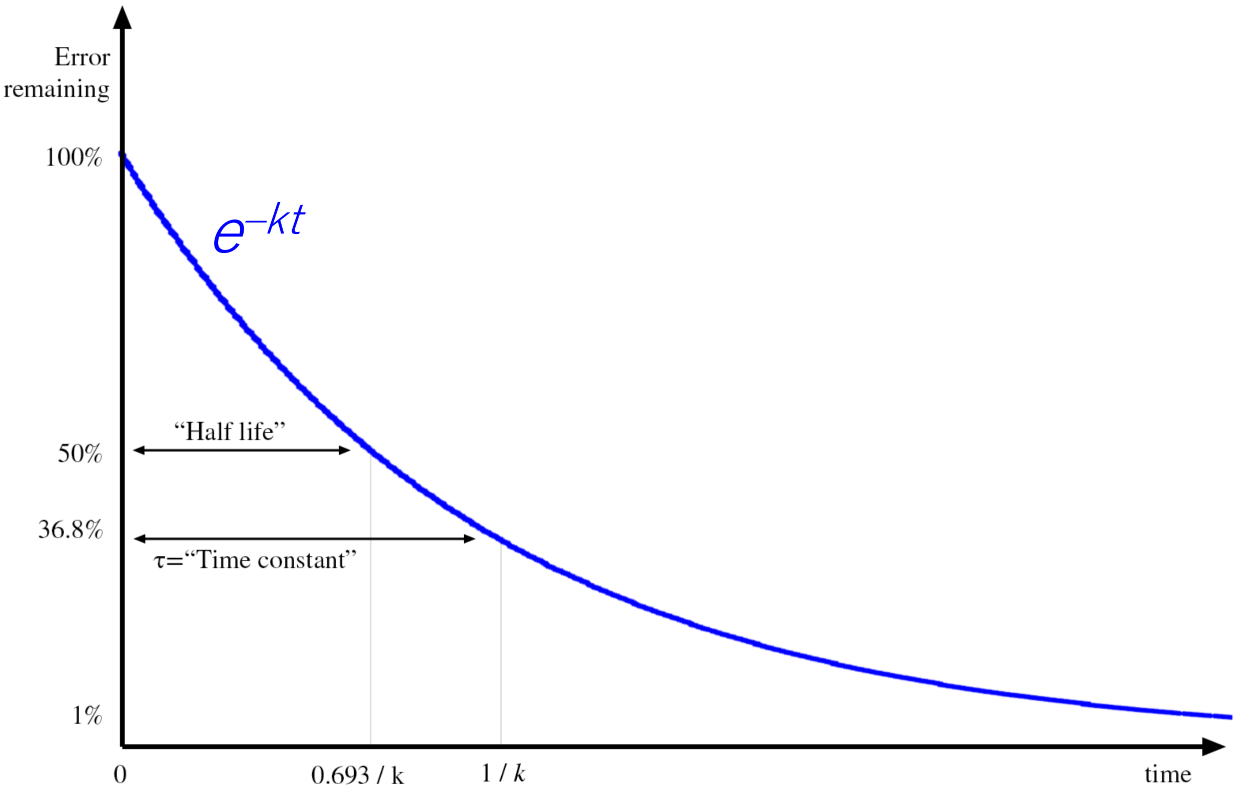
$$dy/dt = k(x - y)$$

$$dy/dt = -ky \qquad \text{(Assume } x = 0\text{)}$$

$$\boxed{y(t) = y_0 e^{-kt}}$$



First-order lag – Choosing k **using Math™**



First-order lag - Variable frame rate

```
cur += (target-cur) * k
```

```
cur += (target-cur) * k*dt
```

```
cur += (target-cur) * (1 - exp(-k*dt))
```

$$y(t) = e^{-kt}$$

First-order lag - Key points

Control system w/ velocity proportional to error

Characteristic behaviours:

- Step response: velocity lurch, then exponential decay.
- Never fully reaches target (in theory)

Use Math™ to understand k!

Please don’t call it “lerp”.

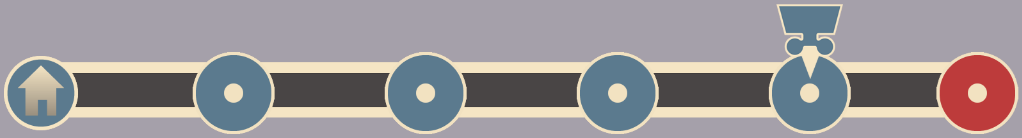
Last stop

Timed transitions

- Basic lerp
- Smoothstep
- Chase transitions

Control systems

- First-order lag
- **PD controller**



PD Controller — a.k.a. “spring-damper”

Problem: 1st order system jarring / non-physical

A commonly re-invented “wheel”!

Solution: 2nd order system

$$d^2y/dt^2 = F(x, y, dy/dt)$$ 2nd order system

$$d^2y/dt^2 = k_p(x-y) + k_d(dy/dt)$$ PD controller

Proportional
Spring

Derivative
Damper

PD Controller - Parameters

$d^2y/dt^2 = k_p(x-y) + k_d(dy/dt)$

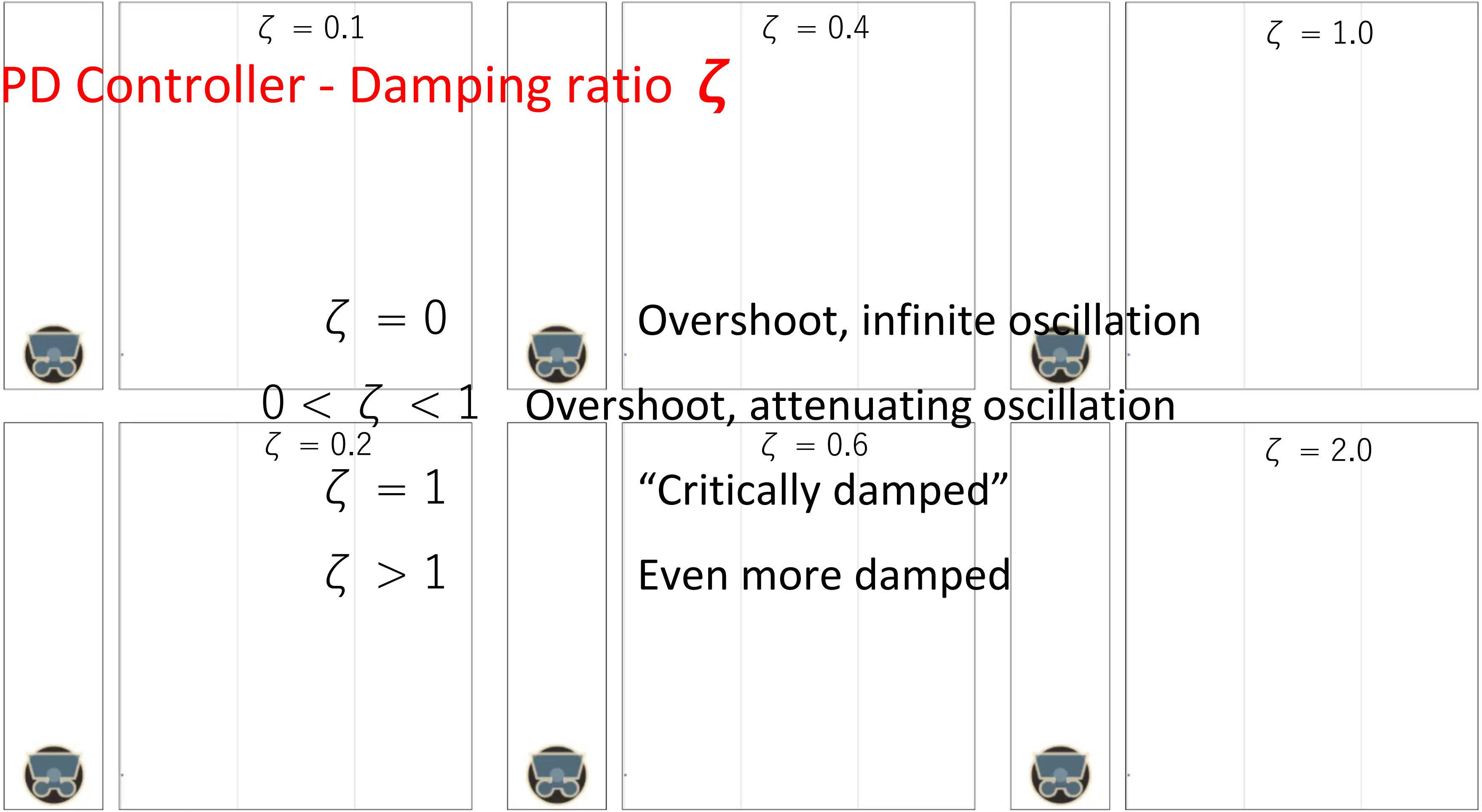
$d^2y/dt^2 = \omega^2(x-y) - 2\,\zeta\,\omega(dy/dt)$

k_p and k_d not intuitive

ζ = **Damping ratio** Amount of overshoot / oscillation

ω = **Natural frequency** "Tightness"

PD Controller - Damping ratio ζ



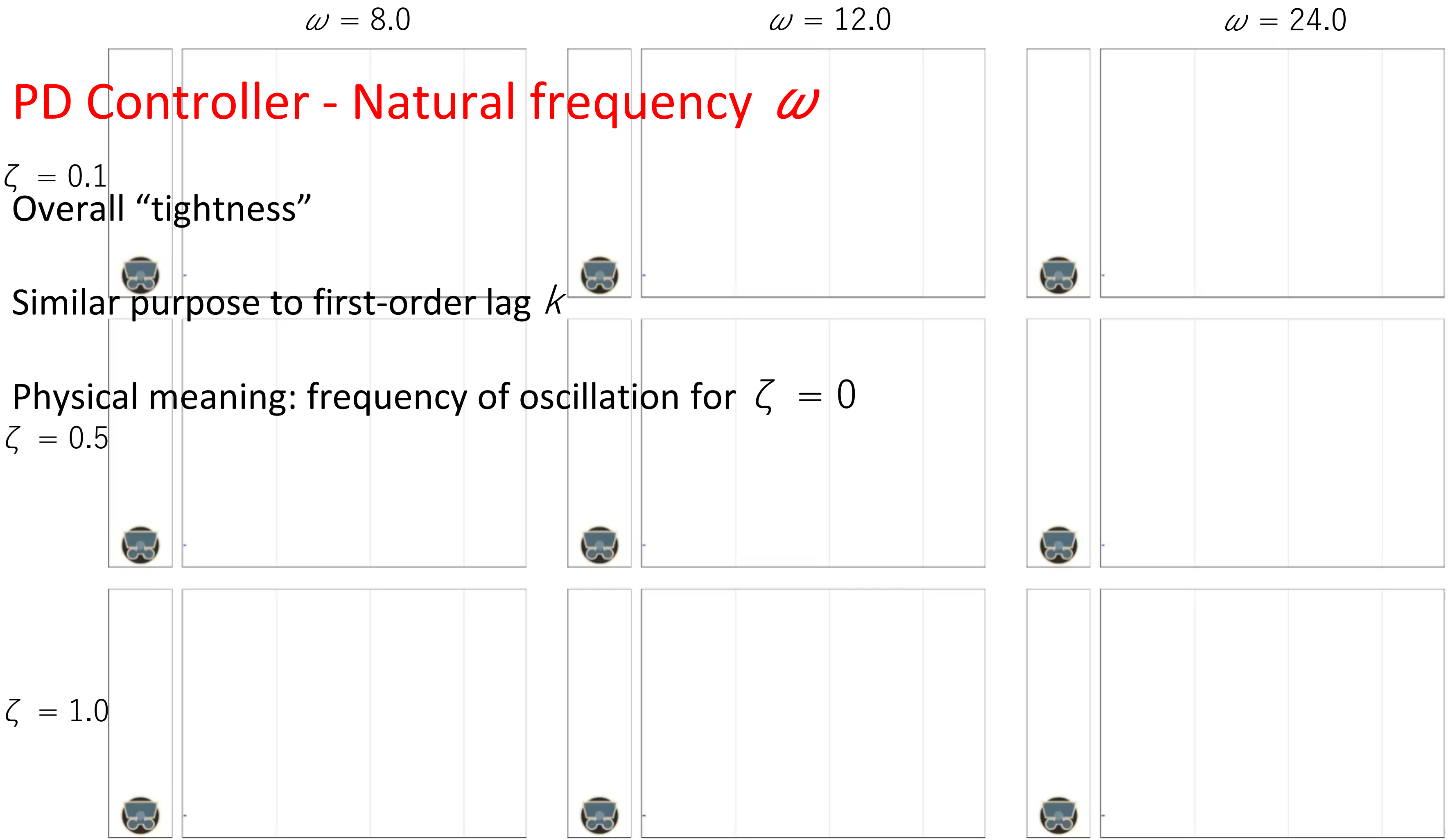
$\zeta = 0$	Overshoot, infinite oscillation
$0 < \zeta < 1$	Overshoot, attenuating oscillation
$\zeta = 1$	"Critically damped"
$\zeta > 1$	Even more damped

PD Controller - Natural frequency ω

Overall “tightness”

Similar purpose to first-order lag k

Physical meaning: frequency of oscillation for $\zeta = 0$



PD Controller - Code

```
// State variables
Vec cur; // current value (y)
Vec vel; // current velocity (dy/dt)

void Update(Vec target, float dt) {

  // Calculate acceleration
  Vec acc = k_p*(target-cur) + k_d*vel;

  // Step forward in time (“Euler integration”)
  vel += acc*dt;
  cur += vel*dt;

}
```

$$d^2y/dt^2 = k_p(x-y) + k_d(dy/dt)$$

PD Controller - Key points

Great default method

Tune using two parameters:

- Damping ratio: how much overshoot / oscillation
- Critically-damped common choice
- Natural frequency: how “tight” do you want the system to be?

Easy to implement

Payload Delivered

For fixed-duration transitions:

- Try smoothstep if lerp feels “mechanical”
- Try chase-style if target moves

No “finish time”? Use a “control system.”

- First-Order Lag is a simple / solid method. Suffers from “lurch”.
- PD controller is a good default method

Don’t “fiddle”. Use these well-studied methods!

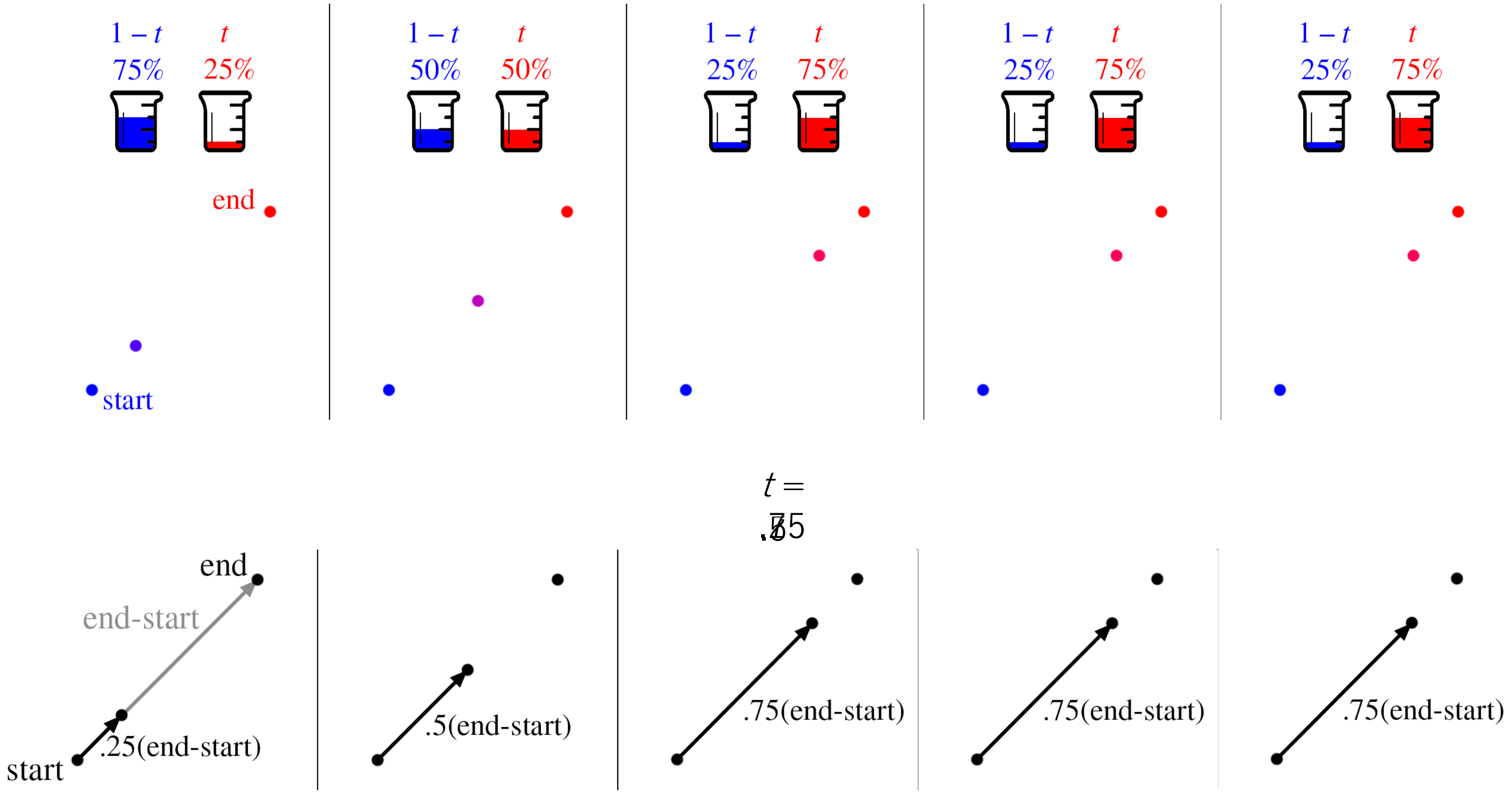
Thank you

fletcherd at valvesoftware dot com

@zpostfacto

gamemath.com/gdc2021

www.valvesoftware.com/jobs



1 – t
t
75%
25%
end
start
1 – t
t
50%
50%
1 – t
t
25%
75%
1 – t
t
25%
75%
1 – t
t
25%
75%
t =
.75
end
end-start
.25(end-start)
start
.5(end-start)
.75(end-start)
.75(end-start)
.75(end-start)