SPECIAL EDITION 2026

FLUTTER ANIMATION

Concepts, Math & Mechanisms

From Basic Motion to Advanced Interaction

Understand how Flutter turns time, curves, tweens and controllers into fluid user experiences.

- **Implicit vs Explicit** — Know when to use which
- **Tween & Lerp** — Interpolate anything
- **Curves & Easing** — Shape the feeling
- **Physics & Springs** — Natural. Expressive. Real.
- **Performance at 120 FPS** — Build silky smooth UIs

$$P(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t)t^2 P_2 + t^3 P_3$$

```
AnimationController
  .vsync(this)
  ..duration = Duration
    (milliseconds: 800)
  ..repeat();
```

Flutter
Build beautiful, animate everything.

flutterfever.com

02

SPECIAL EDITION 2026

ANIMATION FOUNDATIONS

How motion works inside Flutter

Everything in Flutter animation flows through a simple pipeline. A ticker drives time, a controller produces values, tweens map those values, curves shape the motion, and widgets rebuild to reflect the change. Master these building blocks and you can create any motion you imagine.

THE ANIMATION PIPELINE

Ticker / Vsync	Animation Controller	Animation <double>	Tween	Curve	Widget Rebuild
Provides the heartbeat (time) of the animation.	Drives values from 0.0 to 1.0 over a duration.	A stream of values that change over time.	Maps values from start to end.	Eases the motion for natural feel.	UI updates with new values.

CORE BUILDING BLOCKS

Controller
AnimationController orchestrates time. It has a duration, direction, and a current value.

Tween
Defines a range (begin → end) and maps controller values to useful output.

Curve
Modifies the rate of change. Curves make motion feel natural and expressive.

Status
Controllers emit status changes (forward, reverse, completed, dismissed).

Listener
Listens to value or status changes so you can react or trigger side effects.

IMPLICIT vs EXPLICIT

Implicit (Animated*)
Quick, concise and perfect for simple UI transitions with minimal code.

Explicit (Controller)
Full control over time, curves and sequences. Ideal for complex and coordinated animations.

CODE IN ACTION

Implicit Example — AnimatedContainer

```
AnimatedContainer(
  duration: const Duration(milliseconds: 600),
  curve: Curves.easeInOutCubic,
  width: expanded ? 220 : 80,
  height: expanded ? 220 : 80,
  decoration: BoxDecoration(
    color: expanded ? Colors.blue : Colors.cyan,
    borderRadius: BorderRadius.circular(24),
  ),
)
```

Explicit Example — AnimationController

```
late final AnimationController controller;
late final Animation<double> opacityAnim;

@override
void initState() {
  super.initState();
  controller = AnimationController(
    vsync: this,
    duration: const Duration(milliseconds: 800),
  );
  opacityAnim = CurvedAnimation(
    parent: controller,
    curve: Curves.easeInOut,
  );
}
```

MOTION OVER TIME

Start	Entering	Centering	Emphasizing	Complete
Opacity: 0.0	Opacity: 0.35	Opacity: 0.75	Opacity: 0.95	Opacity: 1.0
Scale: 0.90	Scale: 0.95	Scale: 1.00	Scale: 1.05	Scale: 1.00
Rotate: -15°	Rotate: -5°	Rotate: 0°	Rotate: 5°	Rotate: 0°
Position: Off-screen	Position: Moving in	Position: Center	Position: Settle	Position: Final

Tip: Think in 0.0 to 1.0. The controller drives the progress, everything else translates that progress into beautiful, meaningful motion.

WHEN TO CHOOSE WHAT

Quick UI Polish
Use implicit animations like AnimatedContainer, AnimatedOpacity or AnimatedSwitcher.

State-driven Transition
Use explicit animations when the transition is triggered by state changes or business logic.

Custom Choreography
Use explicit controllers to sequence, stagger and synchronize multiple animations with precision.

Flutter
Build beautiful, animate everything.

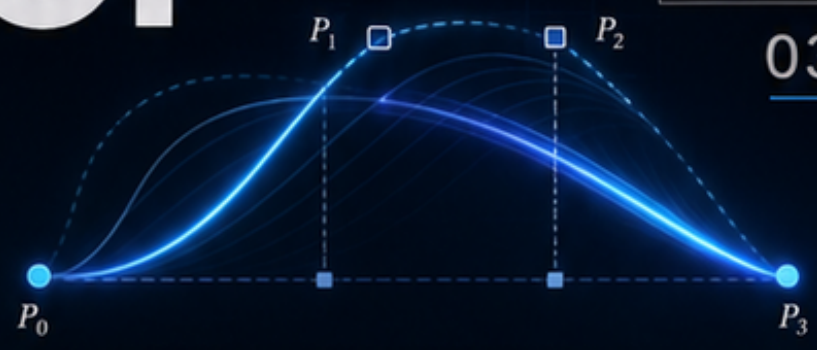
flutterfever.com

THE MATH OF MOTION

— Time, interpolation, easing and spring behavior —

SPECIAL EDITION 2026

03



1 Normalized Time $t \in [0, 1]$

Animation progresses from start (0) to end (1). Everything is a function of time t.

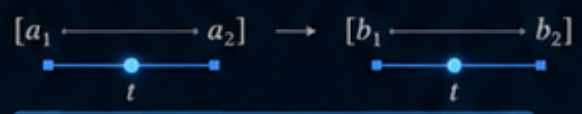
2 Linear Interpolation (*lerp*)

$$\text{lerp}(a, b, t) = a + (b - a)t$$

a = start b = end t = time

The simplest type of motion. Produces a straight-line change between values.

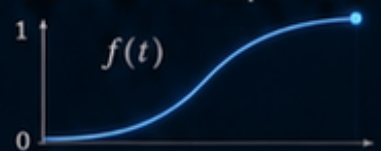
3 Tween: Map One Range to Another



$$value = b_1 + (b_2 - b_1) \cdot \frac{t - a_1}{a_2 - a_1}$$

Tweens convert an input range $[a_1, a_2]$ into an output range $[b_1, b_2]$.

4 Curves: Reshape Time

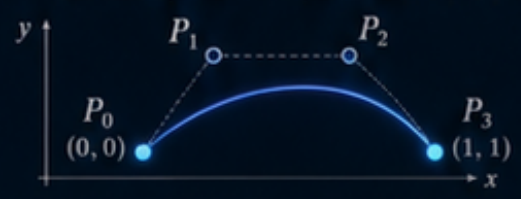


Curves remap t into $f(t)$ to control speed. Ease-in, ease-out, bounce, elastic and more.

$$value = \text{lerp}(a, b, f(t))$$

5 Cubic Bézier: Intuition

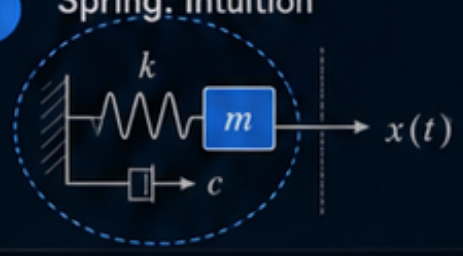
Defined by 4 control points P_0, P_1, P_2, P_3



The curve starts at P_0 and ends at P_3. P_1 and P_2 pull the curve, shaping its acceleration profile.

$t \in [0, 1] \longrightarrow$ position on curve

6 Spring: Intuition



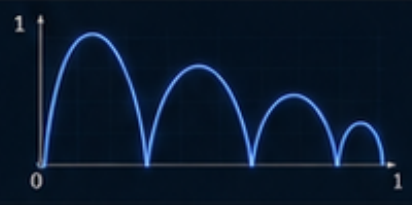
m	Mass
k	Stiffness (spring strength)
c	Damping (resistance)
x	Displacement from rest
$v = x'$	Velocity

Common Curves: Feel Comparison

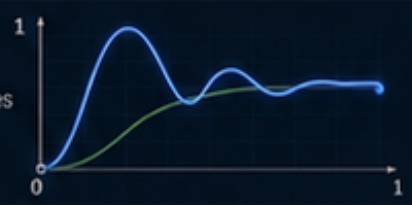
Linear
Constant speed from start to end.

EaseInOut
Slow start, speeds up, then slows down.

Bounce
Hits the end and bounces a few times before settling.



Elastic
Overshoots and oscillates like a stretched spring before settling.



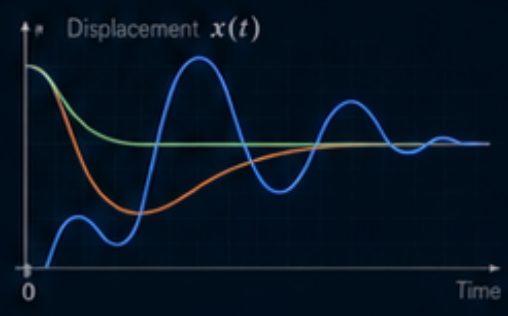
Spring Model Intuition

A damped harmonic oscillator.

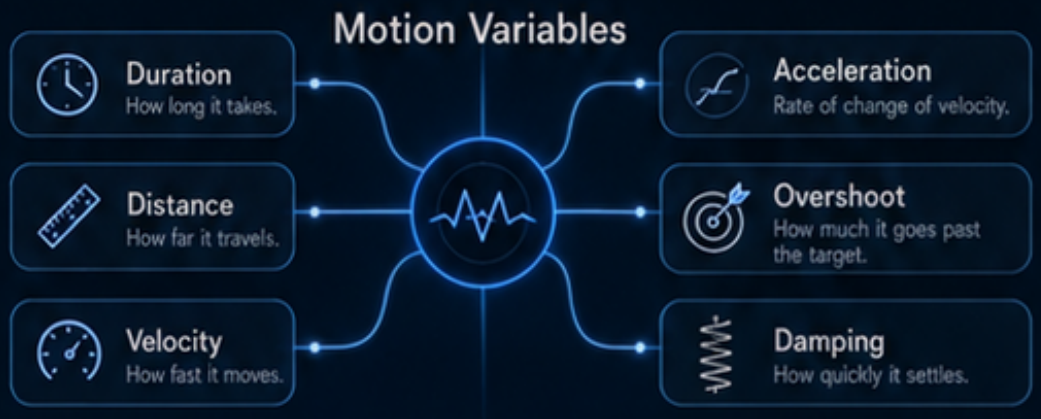
$$x'' + 2\zeta\omega x' + \omega^2 x = 0$$

ω = natural frequency
ζ = damping ratio

$\zeta < 1$	Underdamped (oscillates)
$\zeta = 1$	Critically damped (fastest to rest)
$\zeta > 1$	Overdamped (sluggish)



Motion Variables



Why It Matters

Feel
Great motion communicates state changes, creates delight and builds trust.

Realism
Physics-inspired motion feels natural and grounded.

User Guidance
Motion directs attention, clarifies hierarchy and reduces cognitive load.

Flutter
Build beautiful, animate everything.

flutterfever.com

FLUTTER ANIMATION

SPECIAL EDITION 2026

MECHANISMS & APIs

— From widgets to orchestration —

> *Great animation isn't magic. It's motion with meaning.*

Flutter gives you multiple ways to bring interfaces to life. Choose the right mechanism, compose with precision, and orchestrate seamless experiences.

LIFECYCLE ESSENTIALS

initState()
Create controller, animation, and configuration.

vsync
Provide a TickerProvider (vsync) for smooth ticks.

addListener()
Listen for value changes or use a Builder.

setState() or Builder
Update UI efficiently on each frame.

dispose()
Dispose controllers to free resources.

MECHANISM MODULES

AnimatedContainer
Implicitly animates changes to its properties over a duration.

Use case: Smoothly expand a card when it's tapped.

AnimatedOpacity
Implicitly fades a widget in or out by animating opacity.

Use case: Fade in content after an async load.

TweenAnimationBuilder
A simple explicit animation from one value to another.

Use case: Animate a number or color change on demand.

AnimatedBuilder
High-performance builder that listens to any animation.

Use case: Rebuild complex parts of the tree efficiently.

Hero
Create shared element transitions between routes.

Use case: Seamlessly animate images or cards across screens.

Physics-based Animations
Use SpringSimulation and friction for natural motion.

Use case: Draggable interactions and bounce effects.

HOW ANIMATIONS WORK TOGETHER

From state change to pixels on screen—every frame.

CURVED ANIMATION EXAMPLE

DART

```dart
late AnimationController _controller;
late Animation<double> _animation;

@override
void initState() {
  super.initState();
  _controller = AnimationController(
    vsync: this,
    duration: const Duration(milliseconds: 800),
  );
  _animation = CurvedAnimation(
    parent: _controller,
    curve: Curves.easeInOutCubic,
    reverseCurve: Curves.easeInCubic,
  );
  _controller.forward();
}

@override
void dispose() {
  _controller.dispose();
  super.dispose();
}
```

Tip: Curves shape perceived motion. Ease in, ease out, or make it elastic.

STAGGERED ANIMATION TIMELINE

Sequence elements with delays and offsets.

QUICK CHOOSER

Small property changes	→ Use Implicit (Animated*)
Complex sequences	→ Use Explicit (Controller)
Shared transitions	→ Use Hero
Natural / physical feel	→ Use Physics
High performance builds	→ Use AnimatedBuilder

Flutter
Build beautiful, animate everything.

"Motion is the language of experience."

flutterfever.com

05

SPECIAL EDITION 2026

ADVANCED CHOREOGRAPHY

Performance, patterns and production thinking

Staggered Animations
Orchestrate lists and grids with intervals, delays and curves for natural, readable motion.

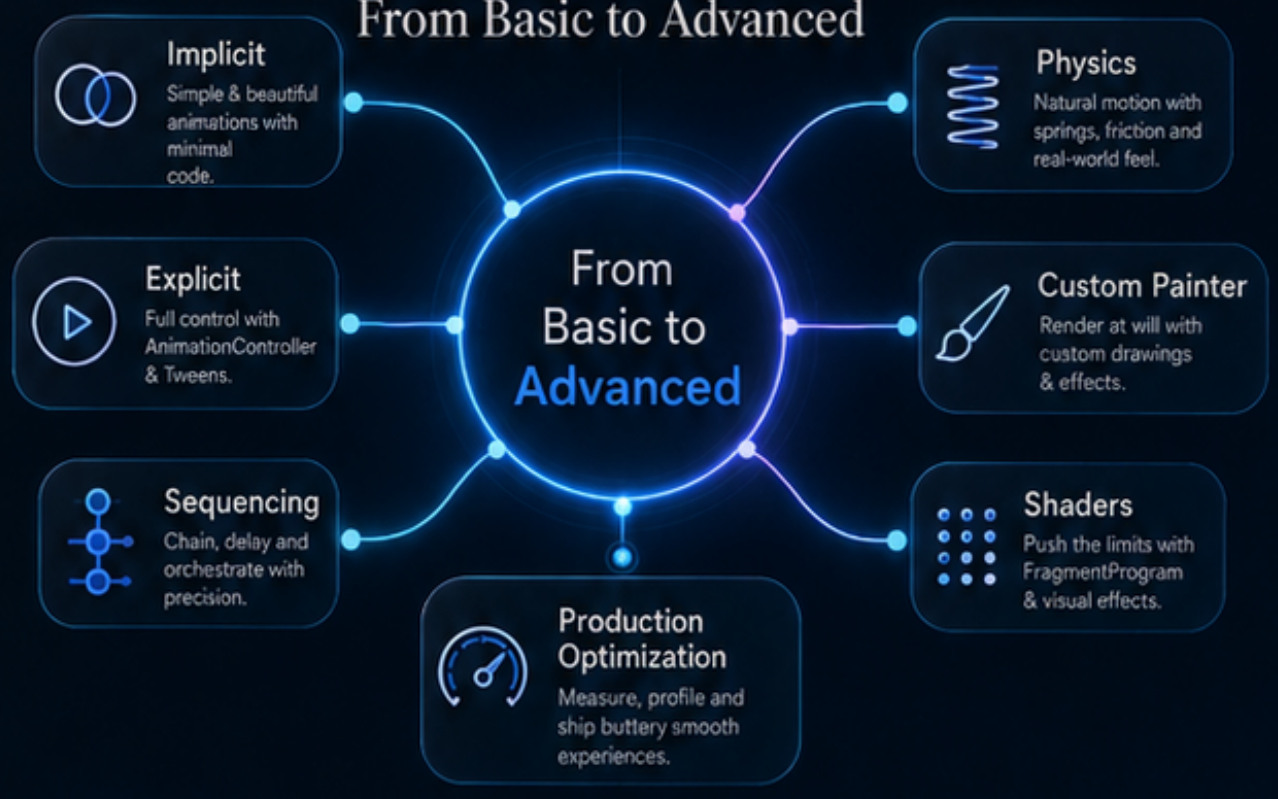
Route Transitions
Design seamless page transitions with shared axis, fade through and custom transitions.

Hero Animations
Create spatial continuity between screens using *Hero*, *flightShuttle* builder and custom flights.

Scroll-Linked Motion
Connect UI to scroll with ScrollController, AnimatedBuilder and slivers for immersive feels.

Micro-Interactions
Elevate affordances and feedback with subtle, fast and purposeful micro-animations.

From Basic to Advanced



Performance Checklist

- Target 60 / 120 FPS consistently
- Avoid unnecessary rebuilds
- Use child in AnimatedBuilder when possible
- Dispose controllers & listeners
- Prefer lightweight effects (over blurs & shadows)
- Profile on real devices, not just simulators

Frame Performance (Goal)
120 FPS

Case Study

A polished onboarding flow

1 Opacity In 300ms → 2 Slide Up 400ms → 3 Scale In 350ms → 4 Delayed CTA 600ms

Coordinated animations create clarity, momentum and a sense of delight.

Common Mistakes

Too Much Motion
Over-animating hurts clarity and distracts users.

Mismatched Durations
Inconsistent timing breaks rhythm and polish.

Rebuilding Expensive Trees
Unnecessary rebuilds drop frames and drain battery.

Ignoring Accessibility
Not respecting user preferences alienates and excludes.

Great motion is invisible. You feel it, not notice it.

> *"Advanced animation isn't about showing off — it's about designing experiences that feel effortless, intuitive and human."*

— Build with purpose. Animate with meaning.