How to Animate Elements When in View on Scroll with Framer Motion

Author
Will Johnson

Animate React Apps with Framer Motion

Will released a full-length course on Framer Motion. Check it out if you're interested!

What is Framer Motion?

Framer Motion is a declarative animation library for React. It makes adding animations to React apps feel simple, even magical. Framer Motion basically hides away CSS transitions from you. You just say what you want, and BOOM the library will handle the crs details of the animation.

The heart & soul of Framer Motion is Motion components. You can turn anything HTML or SVG element into a motion component by adding `motion.` to the beginning of the element.

Example

```
<motion.div>Motion Component</motion.div>
```

Once an element becomes a motion component, it gets access to a bunch of new props like `animate`, `variants` and `transition` to name a few.

Prerequisites

You would know how to create small applications in React and use features like Hooks. You are also familiar with CSS properties like opacity, scale, and transition. While this isn't an introduction to Framer Motion from scratch. You're not expected to have any Framer Motion knowledge. You will learn Framer Motion by building with it, and I'll explain new concepts along the way.

What You'll Be Building

You'll use Framer Motion with react-intersection-observer to trigger squares and scale animation to grow bigger when they come into view.

> *"react-intersection-observer is a React flavored version of the Intersection Observer API to tell you when an element enters or leaves the viewport."*

You will not build this application from scratch the starter code is here

You can view the final demo here

Set Up

You'll start with a React app with four `square` components rendered on screen vertically.

App.js

```
import React from "react";
import "./styles.css";
function App() {
  return <div className="square">Hello,
</div>;
}
export default function App() {
  return (
    <div className="App">
      <H1 className="title">Hello!</H1> Demo</h1>
      <Square />
      <Square />
      <Square />
      <Square />
    </div>
  );
}
```

The first thing you'll do is install Framer Motion and react-intersection-observer

```
npm install framer-motion react-intersection-observer --save
```

Next, you'll bring in everything you need to create the animation.

- Import `motion` and the `useAnimation` hook from Framer motion
- `useInView` from React
- `useInView` hook from react-intersection observer

App.js

```
import React, { useEffect } from "react";
import { useAnimation, motion } from "framer-motion";
import { useInView } from "react-intersection-observer";
```

Don't worry. You will get an explanation on what each one does when it's time to use them.

Creating the Animation with Variants

To create the animation, you'll use a feature of Framer Motion call Variants (not be confused with CAN variants). Variants are objects that you use to define how you want the animation to look and reference the animation by name.

App.js

```
const squareVariants = {
  visible: { opacity: 1, scale: 4, transition: { duration: 1 } },
  hidden: { opacity: 0, scale: 0 }
};
function Square() {

}
```

Here you created a variant object called `squareVariants`. Inside the object, you have the property `visible` that you passed an object as the value. When the element is `visible` you want it to:

- have opacity be 1
- scale to 4x it's size
- have the transition from hidden to visible to have a duration of 1 second.

And when the element is `hidden` you change `opacity` and `scale` to 0 so it's invisible.

Variants are dope because they free up your motion components from being overcrowded with a whole lot of props. You can put all the details of your animation in the variants object and reference it by adding the `variants` prop and to a motion component.

App.js

```
function Square() {
  return (
    <motion.div
      variants={squareVariants}
      className="square"
    />motion.div
  );
}
```

Inside of the `Square` component you:

- add `motion.` in the div
- adding `variants` prop
- set it to the `squareVariants` object you created

Just adding the variants prop won't do anything. Your motion components need to know when you want these animations to start and what state you want them to end up in when the animation is finished.

For this, you need to add the `initial` and the `animate` prop.

App.js

```
function Square() {
  return (
    <motion.div
      animate={{ scale: 2 }}
      initial="hidden"
      variants={squareVariants}
      className="square"
    />motion.div
  );
}
```

Set the `initial` prop to `"hidden"` this tells Frame you want this div to have an opacity and scale of 0 (which is the value you define for `hidden` in the `squareVariants` object.) when the pages loads. The `animate` prop describes what state you want the animation to go.

The `animate` prop is the final state of the animation. Framer Motion automatically takes care of how the animation looks from initial to `animate`. The `animate` prop takes objects. Temporarily you'll pass in `{ scale: 2 }`. This causes Square to scale to twice its size on page loads, `animate` will be changed later.

Ready to pick up the pace?
Enter your email and receive regular updates on our latest articles and courses
Email
mail@example.com
What do you want to take to the next level?
Portfolio Building
SIGN UP
egghead.io
We're here to help.
Sign up for our **FREE** email course
Portfolio Building: Learn how to build a badass developer portfolio so you can land that next job

Add useAnimation and useInView to the Square Component

By default, Framer Motion runs animations once the page loads. The `useAnimation` hook gives you access to the helper methods `start` and `stop` to give you more control over when the animation begins and ends. You'll use the `start` method to set what animation gets get trigged when the element is in view.

Create a `controls` variable and set to `useAnimation()` and change the `animate` prop to `controls`.

App.js

```
function Square() {
  const controls = useAnimation();
  return (
    <motion.div
      animate={{ scale: 2 }}
      initial="hidden"
      variants={squareVariants}
      className="square"
    />motion.div
  );
}
```

Next, you'll bring in the `useInView` hook from `react-intersection-observer`. Use array destructing to pull out `ref` and `inView` from `useInView` return.

App.js

```
function Square() {
  const controls = useAnimation();
  const [ref, inView] = useInView();
  return (
    <motion.div
      animate={{ scale: 2 }}
      initial="hidden"
      variants={squareVariants}
      className="square"
    />motion.div
  );
}
```

The `useInView` hook returns an array with a `ref` and a Boolean `inView` that checks if the status of an element being in view is true or `false`.

Trigger Animation with useEffect

Inside of the Square component `useEffect` with an if statement. If `inView` is `true` let it run the `controls.start()` method and pass it the `squareVariants` property of `"visible"`. Use `controls` and `inView` as the dependency array. Finally, add a `ref` attribute to the `motion.div` and assign it the `ref` from `useInView`.

App.js

```
function Square() {
  const controls = useAnimation();
  const [ref, inView] = useInView();
  useEffect(() => {
    if (inView) {
      controls.start("visible");
    }
  }, [controls, inView]);
  return (
    <motion.div
      ref={ref}
      animate={controls}
      initial="hidden"
      variants={squareVariants}
      className="square"
    />motion.div
  );
}
```

When the value of `inView` changes also when the element we referenced is in view, and `inView` changes from `false` to `true`. It will call `useEffect` to trigger `controls.start()` to run the `visible` animation.

This is a look at the full code

App.js

```
import React, { useEffect } from "react";
import { useAnimation, motion } from "framer-motion";
import { useInView } from "react-intersection-observer";
import "./styles.css";
const squareVariants = {
  visible: { opacity: 1, scale: 4, transition: { duration: 1 } },
  hidden: { opacity: 0, scale: 0 }
};
function Square() {
  const controls = useAnimation();
  const [ref, inView] = useInView();
  useEffect(() => {
    if (inView) {
      controls.start("visible");
    }
  }, [controls, inView]);
  return (
    <motion.div
      ref={ref}
      animate={controls}
      initial="hidden"
      variants={squareVariants}
      className="square"
    />motion.div
  );
}
export default function App() {
  return (
    <div className="App">
      <H1 className="title">Hello!</H1> Demo</h1>
      <Square />
      <Square />
      <Square />
      <Square />
    </div>
  );
}
```

Conclusion

The demo app should have four boxes that grow to 4x their size when they are visible on the screen. You created this with minimal tweaking of the animation properties. Framer handled the escaping parts for you.

You learned how to create motion components, create variants to clean up your code, add the `animate` and `initial` and `animate` props to tell Framer Motion how you want animations to start and end. Also, you used the `useAnimation` hook to override Framer Motions default behavior of when the animation starts.

You also used react-intersection-observer to make it a lot easier to tell when our element is in view or not.