Framer Motion Animation - How to animate scroll position in React - Make delightful animations with useViewportScroll

Written by Seth Corker on Benevolent Bytes

What we're animating in React

I'll walk you through how I created the following animation with Framer Motion and React. It's shows an envelope which as you scroll down, a letter slides out before sliding down over the envelope.

See the {" "} letter animation example in Framer Motion

If the iframe above isn't working, you can also see the letter animation example in Framer Motion here. We're creating a simple effect where a letter looks like it's being drawn from an envelope. The animation happens when scrolling up or down.

If you're looking for some more guides with examples of how to use Framer Motion, take a look at how you can create a spinner loading animation or a tap to expand animation

How to control animations on scroll with `useViewportScroll`

The `useViewportScroll` hook is one of my favourite features of Framer Motion. It allows us to control animations based on the scroll position. The way achieve this simple letter opening animation is by mapping time to the scroll position so when the scrollbar is at the top of the page, our animation is at the first frame. When the scrollbar is at the bottom of the page, our animation is at the last frame. Scrolling back up the page will reverse the animation. This is a simple effect but it's possible to create complex animations and base them entirely on where the X and Y scroll position is.

How do we use it?

For our letter example, we are basing the animation on just the Y position so we just use the hook like so:

```
const { scrollYProgress } = useViewportScroll()
```

This hook returns `scrollY`, `scrollX`, `scrollYProgress` and `scrollXProgress`. I chose not to use `scrollY` because I don't really care about how many pixels we've scrolled down the page. I just want to know where we are in the animation from 0 to 100% complete that's why I use `scrollYProgress` which gives us a value between 0 and 1, then we transform that however we need to with the `useTransform` hook below.

What `useTransform` does and how to use it

The `useTransform` hook is useful for transforming one value to another. What does this mean? It allows us to take our `scrollYProgress` which is between 0 and 1 and get a different set of values like so:

```
const { scrollYProgress } = useViewportScroll()
const scaleAnim = useTransform(scrollYProgress, [0, 0.5, 1], [1, 1, 1.5])
const yPosAnim = useTransform(scrollYProgress, [0, 0.4, 1], [0, -250, -100])
```

What you'll notice is that the first argument is `scrollYProgress`, this is what we want to transform. The second argument is a series of numbers that we want to transform between. We know `scrollYProgress` can be 0 at the minimum (we haven't scrolled down the page at all) and 1 at the maximum (we reached the bottom of the page). So why do we use three values? Well, it's because the animation we want to create has three distinct keyframes. So if look at his example for our Y position, `useTransform(scrollYProgress, [0, 0.4, 1], [0, -250, -100])`, we are saying at the beginning Y position 0 should transform to 0. No change. When we are 40% down the page (0.4), the Y position should be -250px upwards. Finally, when we reach the end of the page, the Y position should be at -100px.

Tip for using the `useTransform` hook

When you use the `useTransform` hook this way, make sure you have the same number of input values as output values so if you need three keyframes then make sure you use two arrays of values as the second and third arguments.

Breakdown of the letter animation example

The structure of the example

Let's walk through the example, I'll breakdown the steps used to create letter animation and why certain decisions were made. I'm using CSS-in-JS to keep the style as close to the code as possible, you don't have to do this and Framer Motion doesn't demand you do it either. It's possible to use CSS modules, plain old CSS or a combination of all the above.

Laying out the scene First, I created some high level components to match the mental model of what we're trying to achieve. An envelope with a letter inside it.

```
<div style={letterSceneStyle}>
  <Envelope>
    <Letter />
  </Envelope>
</div>
```

The style for the container div is to give us enough space to scroll. A larger height will make the animation slower because the user has to scroll more for it to reach 100%. I chose a comfortable speed/viewport size of `200vh`.

```
const letterSceneStyle = {
  height: "200vh",
}
```

Creating the envelope Now we get into how the animation works. The only prop the envelope accepts is `children`, in our case. This is the `<Letter />`. A benefit of having the letter within the envelope is any animation we apply to the envelope will affect the letter.

```
function Envelope({ children }) {
  const [ffLayer, setFfLayer] = useState(0)
  const { scrollYProgress } = useViewportScroll()
  const scaleAnim = useTransform(scrollYProgress, [0, 0.5, 1], [1, 1.2, 0.8])
  const yPosAnim = useTransform(scrollYProgress, [0, 0.5, 1], [0, 100, 200])
  const zRotAnim = useTransform(scrollYProgress, [0, 0.5, 1], [0, 3, 0])

  scrollYProgress.onChange(x => {
    setFfLayer(x > 0.4 ? -1 : 0)
  })

  return (
    <motion.div
      style={{
        ...envelopeStyle,
        scale: scaleAnim,
        y: yPosAnim,
        rotateZ: zRotAnim,
      }}
    >
      {children}
      <div style={{ ...frontfaceStyle, zIndex: ffLayer }}>
        <button onClick={() => window.scrollTo(0, 1500)}>Open Me</button>
      </div>
    </motion.div>
  )
}
```

We make extensive use of the `useTransform` hook to transform `scrollYProgress` into the values we need. In the case of `scaleAnim`, we want it to start at 100% scale then get larger halfway through the animation and small at the end. We also move the envelope down the screen with `yPosAnim` and perform a slight tilt with `zRotAnim`. To hook up the animation, we simply use a `<motion.div/>` component and set the `style` prop. If any of the values change, we'll see a smooth animation thanks to Framer Motion. There are a few other things we need to do to achieve the effect which isn't immediately obvious.

What's the `ffLayer` state? The effect I wanted to achieve is a letter being pulled out of an envelope so `ffLayer` is to keep track of the front face of the envelope. The reason we need this is so that after the letter is 'pulled' out, it can then slide down over the envelope. The `scrollYProgress` is a motion value so we can add an `onChange` handler to trigger the state change 50% though the animation. Once we are half-way though, we change the `z-index` of the envelope front face so it will be behind the letter.

How does the button scroll cause a smooth scroll? As a way to showcase the animation, I added `<button onClick={() => window.scrollTo(0, 1500)}>Open Me</button>` to the envelope to allow for a smooth scroll. This works by scrolling the page down 1500px. Just this alone *won't* give us the effect we need. The next step is to make sure we add the following CSS.

```
html {
  scroll-behavior: smooth;
}
```

Creating the letter The letter animation makes use of the same hooks we're familiar with from before. This time we animate the scale and the Y position.

```
function Letter() {
  const { scrollYProgress } = useViewportScroll()
  const scaleAnim = useTransform(scrollYProgress, [0, 0.5, 1], [1, 1, 1.5])
  const yPosAnim = useTransform(scrollYProgress, [0, 0.4, 1], [0, -250, -100])

  return (
    <motion.div
      style={{
        ...letterStyle,
        scale: scaleAnim,
        y: yPosAnim,
      }}
    >
      {/* The contents of the letter goes here */}
    </motion.div>
  )
}
```

Setup styles with CSS-in-JS

The most important thing to achieve the envelope and letter effect are some basic styles before we apply the animation.

Envelope Styles We set a defined with and height of the envelope and position it on the page. Notice, we always want to see the animation take place in the center of the screen regardless of where the user has scrolled to so we use `position: fixed`. The front face style is very simple too. It just needs to be the same size as the envelope and have a `backgroundColor` so it hides the letter while still inside the envelope.

```
const envelopeStyle = {
  width: "28rem",
  height: "15rem",
  scale: 1,
  position: "fixed",
  top: "10rem",
  left: "calc(50% - 14rem)",
  boxShadow: `rgba(0, 0, 0, 0.5) 0px 0px 150px 10px`,
}

const frontfaceStyle = {
  width: "100%",
  height: "100%",
  backgroundColor: "#debda1",
  position: "absolute",
  left: 0,
```

```
  top: 0,
  display: "flex",
  justifyContent: "center",
  alignItems: "center",
}
```

Letter style The only important properties of the letter is the dimensions and position. The position is absolute so we can position it within the envelope. The size is also slightly smaller so it looks as though the letter fits within the envelope.

```
const letterStyle = {
  width: "calc(100% - 1rem)",
  height: "calc(100% - 1rem)",
  top: "0.5rem",
  left: "0.5rem",
  backgroundColor: "#f8efd5",
  overflow: "hidden",
  padding: "1rem",
  boxSizing: "border-box",
  position: "absolute",
}
```

Where to go from here

Although it's a silly example, a letter coming out of an envelope, this is a good example to see how to take advantage of the capabilities Framer Motion offers. If you want a more practical application of `useViewportScroll`, you could do one of the following:

- Show a progress bar of how far through someone is reading a webpage.
- Highlight headings in a sidebar depending on if they are reading that section or not.
- Have elements appear and disappear easily on scroll

The possibilities are boundless, think of some cool ways you can orchestrate animations based on the scroll position.

Resources

- `useViewportScroll` on Framer Motion API docs
- `useTransform` on Framer Motion API docs
- The letter animation example in Framer Motion
- The full source code for the letter example on GitHub