Trigger Animations On Scroll With GSAP

marmelab.com/blog/2024/04/11/trigger-animations-on-scroll-with-gsap-scrolltrigger.html

Jean-Baptiste Kaiser

This blog post is the second of a series we started to share our experience with the GSAP animation library:

We recommend, if you haven't already, that you check out the first post of this series before reading this one, as it introduces some of the library's main concepts.

If you are still there, it means you are ready to dive into `ScrollTrigger` with us! 🙂

Animating on Scroll

In the previous post, we have seen animations that trigger on page load, or after a given delay. It would be nice if we could start animating elements only when they become visible on the page. This is where `ScrollTrigger` comes into play.

`ScrollTrigger` is a plugin allowing to create scroll-based animations. It can either be used as a simple *trigger*, to control when an animation should start (or stop), or it can be used to control the animation *playhead*, i.e. it will bind the animation's overall progress to the position of the scrollbar. Coupled with the (built-in) *pinning* feature, i.e. the ability to pin the container in place while the scroll-based animation is playing, it can allow for many creative effects, such as:

- Progressively replace an image
- Slide-in panels
- Parallax effect
- Progressively draw a path
- Make content scroll horizontally instead of vertically

Syntax

`ScrollTrigger` is a plugin, and as such needs to be registered before usage:

```
gsap.registerPlugin(ScrollTrigger);
```

`ScrollTrigger` can be used as a trigger to start either a tween or a timeline:

```
gsap.to(".box", {
  scrollTrigger: ".box", // start the tween animation when ".box" enters the
viewport (once)
  x: 500,
});

const tl = gsap.timeline({
  scrollTrigger: ".container", // start the timeline animation when ".container"
enters the viewport (once)
});
tl.from(".logo", { duration: 2.5, opacity: 0, scale: 0.3 });
tl.from(".circle", { duration: 1, opacity: 0, y: 150 });
```

Actually, the `scrollTrigger` property accepts many options, allowing to customize when the trigger should activate and how it will affect the animation. Providing a string value (like `".container"`) to `scrollTrigger` is equivalent to passing `scrollTrigger: { trigger: ".container" }`.

Let's go over some of the most useful options.

trigger: Specifying the Element to Watch

As we saw, `trigger` allows us to specify the element whose position is watched by the `ScrollTrigger`. It accepts either a (string) CSS selector or a DOM element object.

```
gsap.to(".box", {
  scrollTrigger: { trigger: ".box" },
  x: 500,
});
```

start: Controlling When the Animation Starts

The `start` option allows you to specify exactly when the `ScrollTrigger` will start the animation.

By default its value is `"top bottom"`, which means: *start the animation when the top of the trigger element hits the bottom of the screen*.

```
gsap.to(".box", {
  scrollTrigger: {
    trigger: ".box",
    start: "top bottom"
  },
  x: 500,
});
```

It also supports more complex values, like `"bottom 80%"`, which would mean: *when the bottom of the trigger element hits 80% down from the top of the viewport*.

Tip: You can use pixel values or percentages which are always relative to the top.

Tip: Use a function to force refresh the value whenever the viewport size changes (e.g. to make it depend on responsive values).

toggleActions: Controlling the Animation Lifecycle

`toggleActions` is an important concept to master when working with `ScrollTrigger` animations. It allows you to specify what should happen to your animation in the following cases:

- If the user scrolls too fast, and the element leaves the viewport before completing its animation
- If the user scrolls backward, and the elements come into view again
- If the user scrolls back to the top, and the element leaves the viewport again but from the bottom -- should we restart the animation?

It consists of 4 different values, meaning, in that order, what to do when the element triggers the toggles:

1. `onEnter`: Element enters the viewport forward
2. `onLeave`: Element exits the viewport forward
3. `onEnterBack`: Element comes back in the viewport backwards
4. `onLeaveBack`: Element exits the viewport backward

The default value for `toggleActions` is `"play none none none"`, meaning: play the animation (once) when the element enters the view, and do nothing else. This means the animation will never restart, nor pause if the user scrolls too fast.

In each slot you can use one of the following keywords: `"play"`, `"pause"`, `"resume"`, `"reset"`, `"restart"`, `"complete"`, `"reverse"`, and `"none"`.

Let's see a more complex example.

```
gsap.to(".box", {
  scrollTrigger: {
    trigger: ".box",
    toggleActions: "play pause resume reset",
  },
  x: 500,
});
```

In this example, the animation will pause if the user scrolls too fast and does not let the animation play until the end. It will resume when the user scrolls back up and brings the element into view again. Lastly, it will reset the animation if the user scrolls up so far that the element is no longer in view. This allows the animation to be ready to play another time if the user scrolls back down again.

scrub: Linking the Progress of the Animation to the Scrollbar

`scrub` is also one of the most important options that can be applied to a `ScrollTrigger`, as it will change the way the `ScrollTrigger` operates. Instead of simply triggering the animation, it will now bind the progress of the animation directly to the position of the scrollbar.

This allows to run an animation effect progressively (like replacing an image or drawing a path), giving the user total control over the pace, and even the direction, of the animation.

Setting `scrub` to `true` will bind the playhead strictly to the scrollbar.

```
gsap.to(".box", {
  scrollTrigger: {
    trigger: ".box",
    scrub: true
  },
  x: 500,
});
```

Alternatively, you can pass a number to `scrub` to smooth things out a bit:

```
gsap.to(".box", {
  scrollTrigger: {
    trigger: ".box",
    scrub: 1 // Means it will take 1 sec for the playhead to catch up with the
scrollbar
  },
  x: 500,
});
```

end: Controlling When the Animation Ends

The `end` option allows you to specify where the `ScrollTrigger` should end. It uses the same syntax as `start`.

Its most useful usage is when used in conjunction with `scrub`, as it allows to specify for how long the user has to scroll to complete the animation. In other words, it allows to control the animation speed.

```
gsap.to(".box", {
  scrollTrigger: {
    trigger: ".box",
    start: "top center",
    scrub: true,
    end: "+=300", // means "300px beyond where the start is"
  },
  x: 500,
});
```

It can also be useful to set the `end` option to have more control over when an animation should pause, or reset, when used in conjunction with `toggleActions`.

```
gsap.to(".box", {
  scrollTrigger: {
    trigger: ".box",
    toggleActions: "play pause resume reset",
    end: "bottom center"
  },
  x: 500,
});
```

pin: Pinning an Element in Place

Use the `pin` option to pin an element in place while the `ScrollTrigger` is active. This is best used with the `scrub` option, and allows for effects such as slide-in panels, making content scroll horizontally instead of vertically, before/after images revealed on scroll, and tons of others...

Setting `pin` to `true` will cause it to pin the `trigger` element.

```
gsap.to(".box", {
  scrollTrigger: {
    trigger: ".box",
    scrub: true,
    pin: true,
  },
  x: 500,
});
```

Setting `pin` to a string will pin the element targeted by a CSS selector. This is useful to pin the parent container for instance, while the `trigger` element is animating.

```
gsap.to(".box", {
  scrollTrigger: {
    trigger: ".box",
    scrub: true,
    pin: ".container",
  },
  x: 500,
});
```

snap: Automatically Snapping to Progress Values

`snap` is yet another useful option. It allows you to pick specific progress values (between 0 and 1), on which you would like the scrollbar (and playhead if you are using `scrub`) to reposition automatically when the user stops scrolling.

This is for example very useful if you are creating slide-in panels, or a carousel, where you wouldn't want to let the user stuck *in-between* two transition states, and instead would like to automatically redirect them to the closest *stable* state.

There are various ways to enable this feature. You can for instance simply pass it a number, equal to the increment between steps. If you have a certain number of sections, you can simply pass `1 / (sections - 1)`.

```
const sections = gsap.utils.toArray('section');
gsap.to(".container", {
  scrollTrigger: {
    trigger: ".container",
    scrub: true,
    snap: 1 / (sections.length - 1),
  },
  x: 500,
});
```

You can also pass it an array of numbers (containing all progress values) or a function (to compute the closest progress value via a formula).

If you are using timeline labels, you can simply pass `"labels"` (or `"labelsDirectional"`) to automatically snap to the closest label in the timeline.

It's also worth noting there are plenty of options you can pass to `snap` to further customize its behavior, like `delay`, `directional`, `duration`, `ease`, `inertia`, etc.

markers and id: Debugging Your ScrollTrigger

The last options I'd like to mention are `markers` and `id`. They are very simple but can help you a great deal when debugging your `ScrollTrigger` animation.

Setting `markers` to `true` will have GSAP display marker lines on the page to help you visualize when the `ScrollTrigger` starts and ends, and where the `trigger` element starts and ends.

Providing a unique string to `id` will display it next to the corresponding markers, which can be useful when working with several `ScrollTrigger` or several `trigger` elements.

```
gsap.to(".box", {
  scrollTrigger: {
    trigger: ".box",
    scrub: true,
    markers: true,
    id: "box-with-scrub",
  },
  x: 500,
});
```

`markers` were enabled on all demos embedded in this post.

Additional Tips

`ScrollTrigger` is not limited to animating DOM Elements, it can also be used to update JS variables, thanks to its callbacks.

- `onEnter`
- `onLeave`
- `onEnterBack`
- `onLeaveBack`
- `onUpdate` (gives the scroll position)
- `onToggle` (gives the `isActive` value)

```
gsap.to(".box", {
  scrollTrigger: {
    trigger: ".box",
    scrub: true,
    onUpdate: self => console.log("progress", self.progress)
  },
  x: 500,
});
```

We can also use `toggleClass` to toggle a CSS class while the scroll trigger is active.

```
gsap.to(".box", {
  scrollTrigger: {
    trigger: ".box",
    scrub: true,
    toggleClass: { targets: ".my-selector", className: "active" }
  },
  x: 500,
});
```

Conclusion

`ScrollTrigger` is another very powerful tool offered by GSAP, allowing you to create animations that trigger on scroll, or follow the scrollbar, with only a few lines of code.

But this power comes at the cost of some **complexity**: `ScrollTrigger` comes with many options that can sometimes get tricky to master, and whose names are in my opinion not always self-explanatory. Fortunately, the `ScrollTrigger` documentation is very complete and includes tons of examples, as well as video tutorials, which help mitigate this issue.

I learned two key lessons while working with `ScrollTrigger`:

1. Understanding that `scrub` will completely change the way the animation operates, along with the effect of other options like `end` (which then serves to control the animation speed).
2. Realizing I should have started using `markers` right away, as they are a huge help in understanding what the `ScrollTrigger` does exactly and what you need to change to fix your animation.

Nevertheless, to me, **it's well worth investing some time** to set up a `ScrollTrigger` properly, as you will then be able to fine-tune your scroll-based animations with ease and get creative with them.

To conclude this series, I'd like to bring some of the difficulties I faced while using GSAP to build animations for the react-admin landing page. This will be the topic of the next and final post of this series: GSAP in practice.