

# How to create high-performance CSS animations

---



- [Browser compatibility](#)
  - [Move an element {:#move }](#)
  - [Resize an element {:#resize }](#)
  - [Change an element's visibility {:#visibility }](#)
  - [Avoid properties that trigger layout or paint {:#triggers }](#)
  - [Force layer creation {:#force }](#)
  - [Debug slow or janky animations {:#debug }](#)
    - [Check if an animation triggers layout {:#layout }](#)
      - [Chrome DevTools {:#layout-chrome }](#)
      - [Firefox DevTools {:#layout-firefox }](#)
    - [Check if an animation is dropping frames {:#fps }](#)
    - [Check if an animation triggers paint {:#paint }](#)
      - [Chrome DevTools {:#paint-chrome }](#)
      - [Firefox DevTools {:#paint-firefox }](#)
  - [Conclusion](#)
- 

This guide teaches you how to create high-performance CSS animations.

See [Why are some animations slow?](#) to learn the theory behind these recommendations.

## Browser compatibility

All of the CSS properties that this guide recommends have good cross-browser support.

- `transform`

- `opacity`
- `will-change`

# Move an element { : #move }

To move an element, use the `translate` or `rotation` keyword values of the `transform` property.

For example to slide an item into view, use `translate`.

```
.animate {
  animation: slide-in 0.7s both;
}

@keyframes slide-in {
  0% {
    transform: translateY(-1000px);
  }
  100% {
    transform: translateY(0);
  }
}
```

{% Glitch 'animation-slide-in' %}

Items can also be rotated, in the example below 360 degrees.

```
.animate {
  animation: rotate 0.7s ease-in-out both;
}

@keyframes rotate {
  0% {
    transform: rotate(0);
  }
  100% {
    transform: rotate(360deg);
  }
}
```

{% Glitch 'animation-rotate' %}

# Resize an element { : #resize }

To resize an element, use the `scale` keyword value of the `transform` property.

```
.animate {  
  animation: scale 1.5s both;  
}  
  
@keyframes scale {  
  50% {  
    transform: scale(0.5);  
  }  
  100% {  
    transform: scale(1);  
  }  
}
```

{% Glitch 'animation-scale' %}

# Change an element's visibility { : #visibility }

To show or hide an element, use `opacity`.

```
.animate {  
  animation: opacity 2.5s both;  
}  
  
@keyframes opacity {  
  0% {  
    opacity: 1;  
  }  
  50% {  
    opacity: 0;  
  }  
}
```

```
}  
100% {  
  opacity: 1;  
}  
}
```

{% Glitch 'animation-opacity' %}

{% Aside %} Find copy and paste examples of various animations at [Animista](#). {% endAside %}

## Avoid properties that trigger layout or paint {:#triggers }

Before using any CSS property for animation (other than `transform` and `opacity`), determine the property's impact on the [rendering pipeline](#). Avoid any property that triggers layout or paint unless absolutely necessary.

{% Aside 'warning' %} If you must use a property that triggers layout or paint, it is unlikely that you will be able to make the animation smooth and high-performance. {% endAside %}

## Force layer creation {:#force }

As explained in [Why are some animations slow?](#), by placing elements on a new layer they can be repainted without also requiring the rest of the layout to be repainted.

Browsers will often make good decisions about which items should be placed on a new layer, but you can manually force layer creation with the `will-change` property. As the name suggests, this property tells the browser that this element is going to be changed in some way.

{% Aside 'caution' %} As layer creation can cause other performance issues, this property should not be used as a premature optimization. Instead, you should only use it when you are seeing jank and think that promoting the element to a new layer may help. {% endAside %}

In CSS this property can be applied to any selector:

```
body > .sidebar {  
  will-change: transform;  
}
```

However, [the specification](#) suggests this approach should only be taken for elements that are always about to change. If the above example was a sidebar the user could slide in and out, that might be the case. Some items on your page may not frequently change, and so it would be better to apply `will-change` using JavaScript at a point where it becomes likely the change will occur. You'll need to make sure to give the browser enough time to perform the optimizations needed and then remove the property once the changing has stopped.

{% Aside %} For more information and examples of correct use of `will-change` read [Everything You Need To Know About The CSS `will-change` Property](#). {% endAside %}

If you need a way to force layer creation in one of the rare browsers that doesn't support `will-change` (most likely Internet Explorer at this point), you can set `transform: translateZ(0)`.

## Debug slow or janky animations {:#debug }

Chrome DevTools and Firefox DevTools have lots of tools to help you figure out why your animations are slow or janky.

## Check if an animation triggers layout {:#layout }

An animation that moves an element using something other than `transform`, is likely to be slow. In the following example, I have achieved the same visual result animating `top` and `left`, and using `transform`.

{% Compare 'worse' %}

```
.box {
  position: absolute;
  top: 10px;
  left: 10px;
  animation: move 3s ease infinite;
}

@keyframes move {
  50% {
```

```
    top: calc(90vh - 160px);
    left: calc(90vw - 200px);
  }
}
```

{% endCompare %}

{% Compare 'better' %}

```
.box {
  position: absolute;
  top: 10px;
  left: 10px;
  animation: move 3s ease infinite;
}

@keyframes move {
  50% {
    transform: translate(calc(90vw - 200px), calc(90vh - 160px));
  }
}
```

{% endCompare %}

You can test this in the following two Glitch examples, and explore performance using DevTools.

- [Before](#).
- [After](#).

## Chrome DevTools {: #layout-chrome }

1. Open the **Performance** panel.
2. [Record runtime performance](#) while your animation is happening.
3. Inspect the **Summary** tab.

If you see a nonzero value for **Rendering** in the **Summary** tab, it may mean that your animation is causing the browser to do layout work.

{% Img src="image/admin/cMNQR2jBEwa6ku5POXtZ.jpg", alt="The Summary panel shows 37ms for rendering and 79ms for painting.", width="800", height="699" %}

The [animation-with-top-left](#) example causes rendering work.

```
{% Img src="image/admin/3bn44P9h6lR93uBNRXY3.jpg", alt="The Summary panel show zero values for rendering and painting.", width="800", height="639" %}
```

The [animation-with-transform](#) example does not cause rendering work.

## Firefox DevTools { `#layout-firefox` }

In Firefox DevTools the [Waterfall](#) can help you to understand where the browser is spending time.

1. Open the **Performance** panel.
2. In the panel Start Recording Performance while your animation is happening.
3. Stop the recording and inspect the **Waterfall** tab.

If you see entries for [Recalculate Style](#) then the browser is having to begin at the start of the [rendering waterfall](#).

## Check if an animation is dropping frames { `#fps` }

1. Open the [Rendering tab](#) of Chrome DevTools.
2. Enable the **FPS meter** checkbox.
3. Watch the values as your animation runs.

At the top of the **FPS meter** UI you see the label **Frames**. Below that you see a value along the lines of `50% 1 (938 m) dropped of 1878`. A high-performance animation will have a high percentage, e.g. `99%`. A high percentage means that few frames are being dropped and the animation will look smooth.

```
{% Img src="image/admin/i9Cg7nswyO7jB768kpdQ.jpg", alt="The fps meter shows 50% of frames were dropped", width="710", height="469" %}
```

The [animation-with-top-left](#) example causes 50% of frames to be dropped

```
{% Img src="image/admin/FGROZ0i15tCAoilOoEdG.jpg", alt="The fps meter shows only 1% of frames were dropped", width="710", height="468" %}
```

The [animation-with-transform](#) example causes only 1% of frames to be dropped.

## Check if an animation triggers paint { `#paint` }

When it comes to painting, some things are more expensive than others. For example, anything that involves a blur (like a shadow, for example) is going to take longer to paint than drawing a red box. In terms of CSS, however, this isn't always obvious: `background: red;` and `box-shadow: 0, 4px, 4px, rgba(0,0,0,0.5);` don't necessarily look like they have vastly different performance characteristics, but they do.

Browser DevTools can help you to identify which areas need to be repainted, and performance issues related to painting.

## Chrome DevTools { : #paint-chrome }

1. Open the [Rendering tab](#) of Chrome DevTools.
2. Select **Paint Flashing**.
3. Move the pointer around the screen.

```
{% Img src="image/admin/MzAeQc5PvClcm3gWaNV.jpg", alt="A UI element highlighted in green to demonstrate it will be repainted", width="708", height="185"%}
```

In this example from Google Maps you can see the elements that will be repainted.

If you see the whole screen flashing, or areas that you don't think should change highlighted then you can do some investigation.

If you need to dig into whether a particular property is causing performance issues due to painting, the [paint profiler](#) in Chrome DevTools can help.

## Firefox DevTools { : #paint-firefox }

1. Open **Settings** and add a Toolbox button for [Toggle paint flashing](#).
2. On the page you want to inspect, toggle the button on and move your mouse or scroll to see highlighted areas.

# Conclusion

Where possible restrict animations to `opacity` and `transform` in order to keep animations on the compositing stage of the rendering path. Use DevTools to check which stage of the path is being affected by your animations.

Use the paint profiler to see if any paint operations are particularly expensive. If you find anything, see if a different CSS property will give the same look and feel with better performance.

Use the `will-change` property sparingly, and only if you encounter a performance issue.

