Aligned Input Events

By Dave Tapuska

Dave is a contributor to Web**Fundamentals**

TL;DR

- Chrome 60 reduces jank by lowering event frequency, thereby improving the consistency of frame timing.
- The `getCoalescedEvents()` method, introduced in Chrome 58 provides the same wealth of event information you've had all along.

Providing a smooth user experience is important for the web. The time between receiving an input event and when the visuals actually update is important and generally doing less work is important. Over the past few releases of Chrome we have driven down input latency across these devices.

In the interest of smoothness and performance, in Chrome 60 we are making a change that causes these events to occur at a lower frequency while increasing the granularity of the information provided. Much like when Jelly Bean was released and brought the Choreographer which aligns input on Android, we are bringing frame aligned input to the web on all platforms.

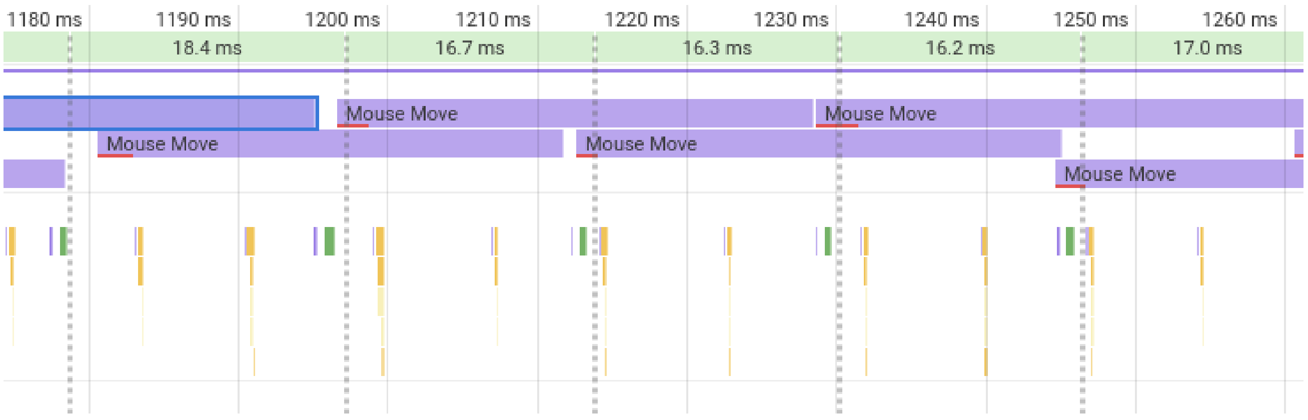
But sometimes you need more events. So, in in Chrome 58 we implemented a method called `getCoalescedEvents()`, which lets your application retrieve the full path of the pointer even while it's receiving fewer events.

Let's talk about event frequency first.

Lowering event frequency

Let's understand some basics: touch-screens deliver input at 60-120Hz and mice deliver input typically at 100Hz (but can be anywhere up to 2000Hz). Yet the typical refresh rate of a monitor is 60Hz. So what does that actually mean? It means that we receive input at a higher rate than we actually update the display at. So let's look at a performance timeline from devtools for a simple canvas painting app.

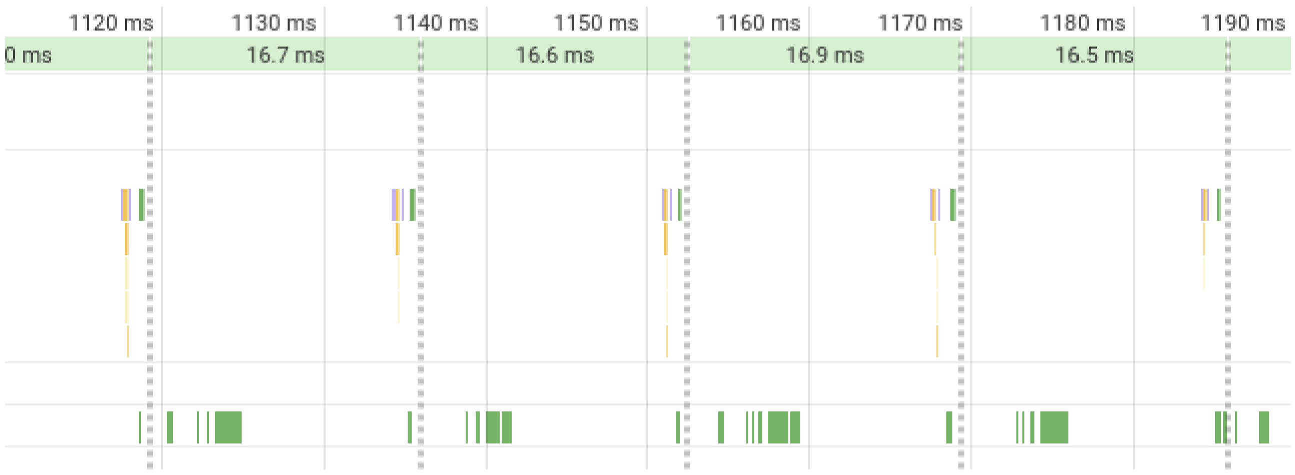
In the picture below with `requestAnimationFrame()`-aligned input disabled you can see multiple processing blocks per frame with an inconsistent frame time. The small yellow blocks indicate hit testing for such things as the target of the DOM event, dispatching the event, running javascript, updating the hovered node and possibly re-calculating layout and styles.



So why are we doing extra work that doesn't cause any visual updates? Ideally we don't want to do any work that doesn't ultimately benefit the user. Starting in Chrome 60 the input pipeline will delay dispatching continuous events ([`wheel`](), [`mousewheel`](), [`touchmove`](), [`pointermove`](), [`mousemove`]()) and dispatch them right before the [`requestAnimationFrame()` callback]() occurs. In the picture below (with the feature enabled) you see a more consistent frame time and less time processing events.

We have been running an experiment with this feature enabled on the Canary and Dev channels and have found that we perform 35% less hit tests which allows the main thread to be ready to run more often.

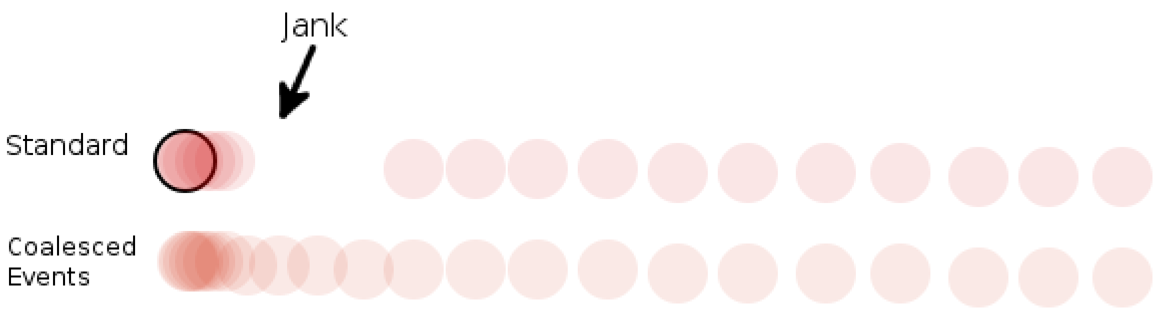
An important note that web developers should be aware of is that any discrete event (such as [keydown](), [keyup](), [mouseup](), [mousedown](), [`touchstart`](), [touchend]()) that occurs will be dispatched right away along with any pending events, preserving the relative ordering. With this feature enabled a lot of the work is streamlined into the normal [event loop]() flow, providing a consistent input interval. This brings continuous events inline with [`scroll`]() and [`resize`]() events which have already been streamlined into the event loop flow in Chrome.



We've found that the vast majority of applications consuming such events have no use for the higher frequency. Android has already aligned events for a number of years so nothing there is new, but sites may experience less granular events on desktop platforms. There has always been a problem with janky main threads causing hiccups for input smoothness meaning that you might see jumps in position whenever the application is doing work, making it impossible to know how the pointer got from one spot to the other.

The getCoalescedEvents() method

As I said, there are rare scenarios where the application would prefer to know the full path of the pointer. So to fix the case where you see large jumps and the reduced frequency of events, in Chrome 58 we launched an extension to pointer events called `getCoalescedEvents()`. And below is an example of how jank on the main thread is hidden from the application if you use this API.



Instead of receiving a single event you can access the array of historical events that caused the event. Android , iOS and Windows all have very similar APIs in their native SDKs and we are exposing a similar API to the web.

Typically a drawing app may have drawn a point by looking at the offsets on the event:

```
window.addEventListener("pointermove", function(event) {
  drawPoint(event.pageX, event.pageY)
});
```

This code can easily be changed to use the array of events:

```
window.addEventListener("pointermove", function(event) {
  var events = 'getCoalescedEvents' in event ? event.getCoalescedEvents() : [even
  for (let e of events) {
    drawPoint(e.pageX, e.pageY)
  }
});
```

Note that not every property on the coalesced events is populated. Since the coalesced events are not really dispatched but are just along for the ride they aren't hit tested. Some fields such as currentTarget, and eventPhase will have their default values. Calling dispatch related methods such as `stopPropagation()` or `preventDefault()` will have no effect on the parent event.