Improved vsync scheduling for Chrome on Android

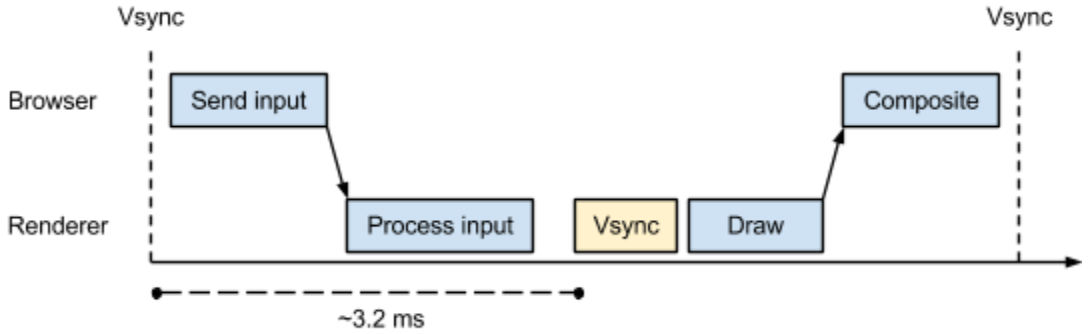
Author: skyostil@

Motivation

Currently the renderer process uses an internal timer to determine when to draw. This timer is tuned to approximate the time base and interval of the system's vsync signal based on a periodic IPC message from the browser. The timer is activated on demand so that it only ticks when the renderer is going to produce at least one new frame.

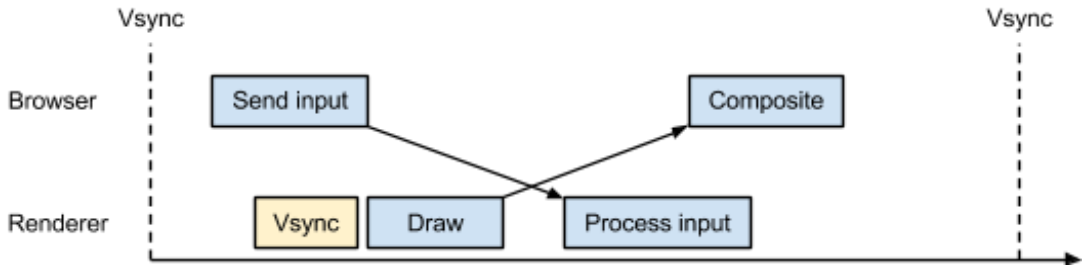
The use of an independent timer to approximate vsync is problematic because of the way it interacts with input event processing. On Android[1], input events are queued and sent by the system immediately at the start of a vsync interval. This means that the renderer should not attempt to draw until it has received the input events for the new frame; otherwise the effects of those input events will not show up until one frame later.

We currently encode this dependency on input event delivery as a 3.2 ms phase delay on the vsync timer. This delay causes the timer to fire after all input events have generally had time to reach the renderer. The delay was chosen experimentally so that the vsync would get triggered soon after the renderer has started processing input. The relative positioning of the events is illustrated in the following diagram.



Since this phase adjustment is done statically based on a magic constant, it does not account for delayed input events, timer jitter or other instability in system scheduling. It is therefore possible for drawing and input event processing to happen out of order, causing input event processing to be delayed by a vsync interval as illustrated below.

[1] Since the Jellybean (API 16) version of Android. On Ice Cream Sandwich we resort to approximating vsync with a timer in the browser.



Another problem with the vsync timer is that when it is switched on, it will generally pulse at the next frame instead of the current one. This means that after processing input events in an idle state, the renderer can be delayed up to a frame before it is allowed to draw in response to that input.

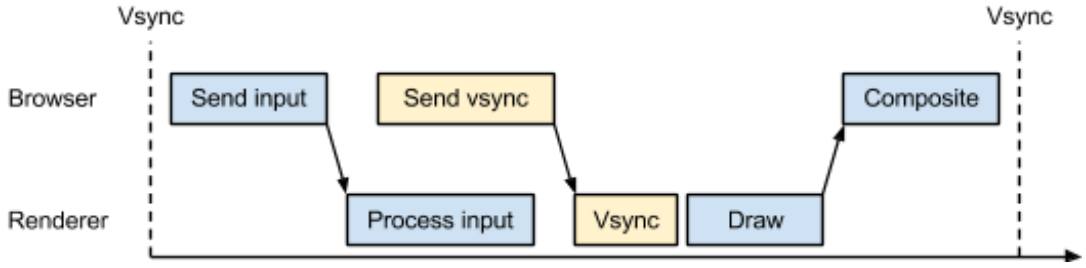
The vsync timer also makes the renderer less efficient, because the time between processing input events and waiting for the timer to fire is spent idling. Also, because of the phase adjustment, input events and frame timestamps are not directly comparable.

Improved vsync scheduling

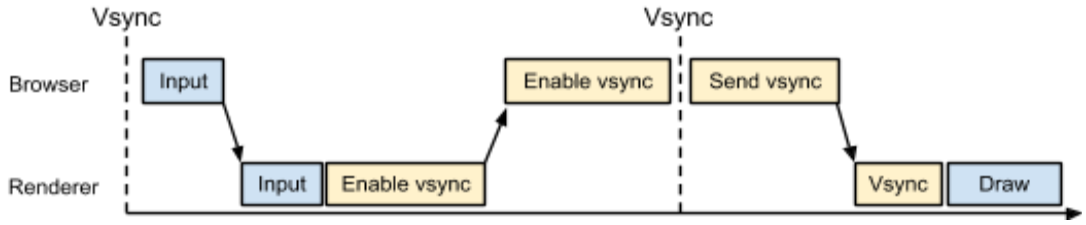
The improved vsync scheduling scheme described here is based on two changes. First, the vsync timer is replaced with an explicit vsync notification message from the browser to the renderer. Second, we use input events that double as vsync notifications when feasible.

Vsync notification message

To guarantee the ordering between input processing and drawing, the browser sends the renderer an explicit vsync notification message. This message is used to initiate drawing in the renderer as shown below.



Since the vsync signal is generated in the browser, we also need to introduce a mechanism for the renderer to enable and disable the notification on demand to avoid sending it needlessly. As the renderer generally leaves vsync enabled over several frames, the signal is controlled with an enable/disable toggle instead of requesting it separately for every frame.

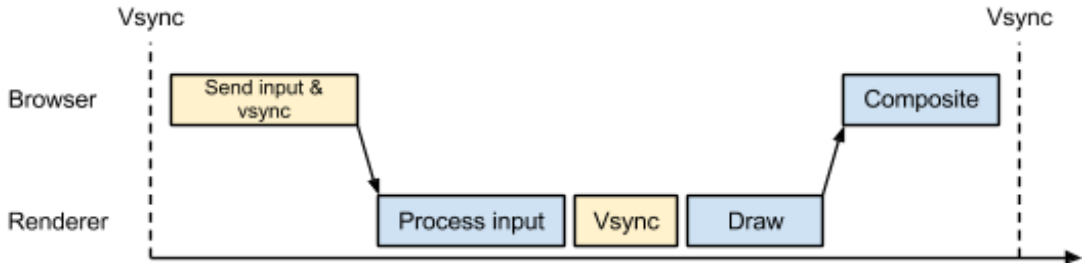


It is important to note that if the notification is enabled in frame N, it will only take effect at frame N+1. This is similar to how the existing timer-based vsync works. The following section describes a further improvement to alleviate this problem.

Triggering vsync based on input

While the vsync notification makes input processing more deterministic, it suffers from some inefficiencies. First, it requires two different IPC message to be sent for frames that contain input. Second, because the notification can only be requested for the next frame instead of the current one, the latency of responding to input events from an idle state is not improved.

We fix these problems by combining both the input event and the vsync notification to a single IPC message. This is done by marking the final input event with a flag that indicates no more input will be received for the current frame. As mentioned, we can do this because Android delivers input events at the start of a vsync interval, which means the renderer can start drawing as soon as this has happened.



If the actual vsync notification was also enabled for the same frame, it is suppressed by the browser to avoid having the renderer draw the same frame twice.

There are some limitations to this optimization. First, we can only do it for frames that contain input; other types of rendering will continue to use the vsync notification. Second, it is only useful with input events[2] that are processed atomically on the renderer cc thread. This is because the effects of main-thread input event processing flow back through a separate commit operation, which generally completes much later than at the nominal vsync time. Effectively the results of such input events will only become visible at the following frame, i.e., with an extra

[2] Specifically GestureScrollUpdate and GesturePinchUpdate

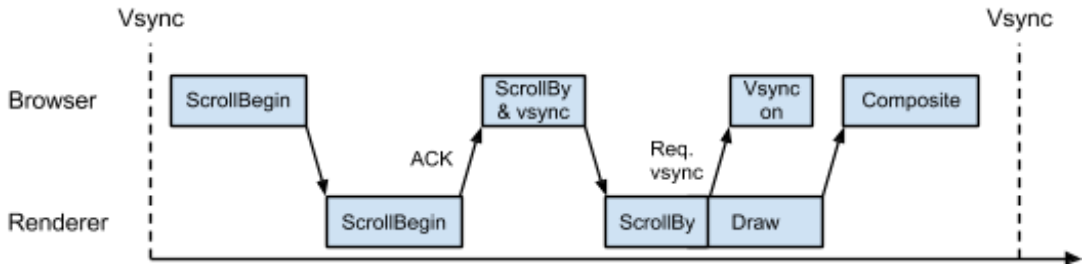
frame of latency.

We could likely react to commit-induced vsync requests earlier than at the next frame, but this is left as future work.

Case studies

In this section we examine how the proposed improvements work in common input latency related use cases.

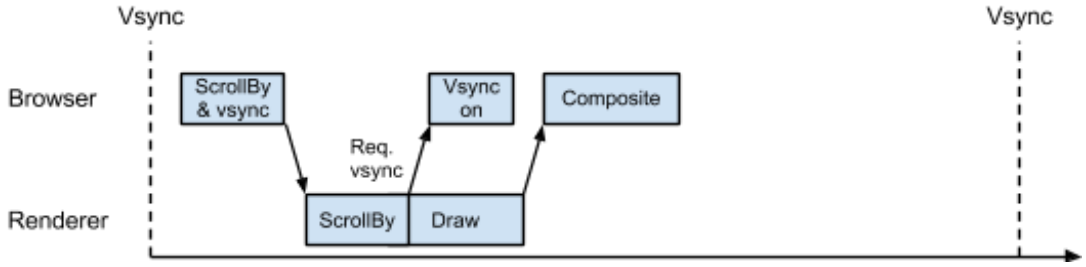
Starting to scroll



Responding to a user scroll gesture happens entirely within a single vsync interval as the renderer can process both the ScrollBegin and ScrollBy messages without waiting for an intermediate vsync tick.

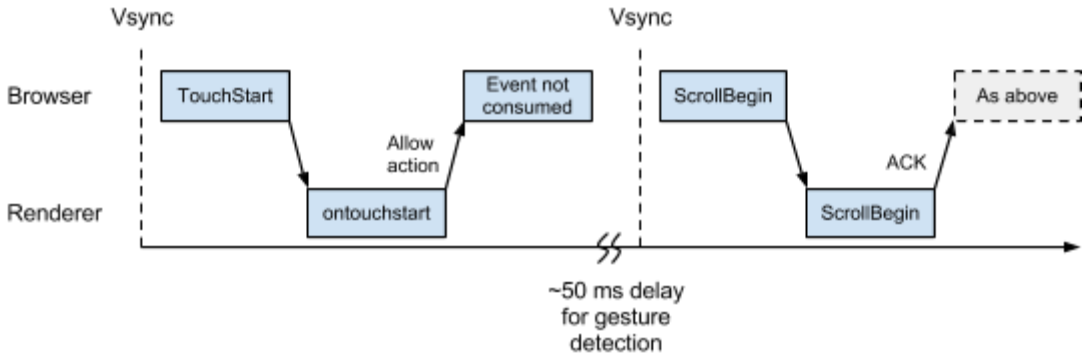
A further improvement would be to merge the ScrollBegin and the first ScrollBy messages to avoid the initial round-trip.

Scrolling steady state



Each ScrollBy message doubles as a vsync, so both input processing and the subsequent drawing are done as an atomic operation in the renderer.

Scrolling a page with a JavaScript touch handler



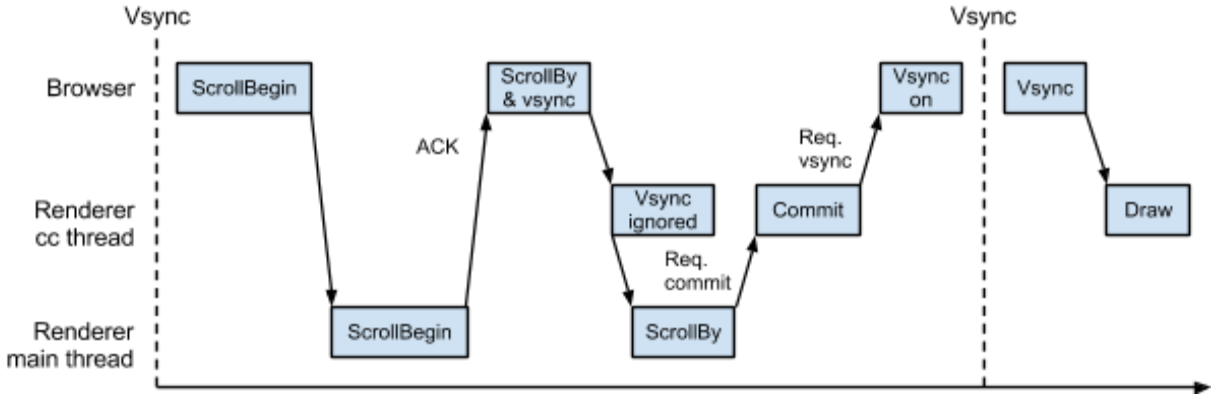
When the user starts scrolling a page with a JavaScript touch handler, the browser consults the handler with the initial touch event (“TouchStart”). There are two possible results:

1. The event is allowed to propagate. Scroll gesture recognition and processing proceeds normally. No further touch events are sent to JavaScript.

   Note that because TouchStart and ScrollBegin are distinct input events that generally are separated in time by >=2 vsync intervals because of the gesture recognition delay, the initial round trip to the renderer has no impact on scroll touch latency. In other words, simply having a touch event handler on a page does not impact scroll latency. However, latency may be increased due to the side effects of the touch handler execution.

2. The event is consumed. Gesture recognition is disabled for the duration of the touch transaction.

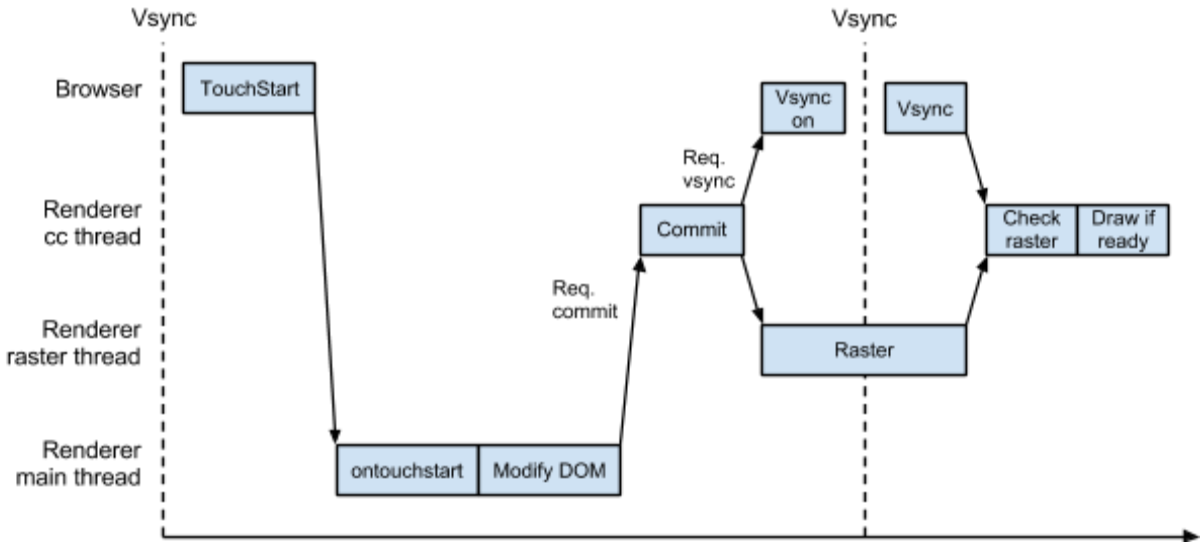
Scrolling a page that uses main-thread scrolling



When a page requires main-thread scrolling, e.g., because of wheel event handlers or non-layer viewport dependent objects, scroll gesture processing is delegated to the renderer's main thread. When properties of the page, such as the scroll position, are modified on the main thread, a separate commit operation is needed to apply those change to the cc thread. This means that processing the ScrollBy message is no longer atomic and we cannot make use of the vsync signal that is bundled with it. What happens instead is that during the commit operation the renderer requests a vsync notification from the browser. The notification arrives at the start of next vsync interval and the renderer proceeds to draw the new frame. Thus, any input processing that flows through a commit has an extra frame of latency compared to input processing that happens on the cc thread.

Note that the current timer-based vsync has the same problem, because the offset vsync pulse generally arrives before the main thread has finished processing input events.

JavaScript Touch handler latency



The diagram above illustrates the processing flow of a page that modifies its DOM as a response to touch events. Similarly to main thread scrolling, the DOM modification causes a commit, the effects of which become visible at the following frame at the earliest. An additional complication is the raster work prompted by the DOM changes; currently the compositor polls for raster work completion at vsync intervals and decides whether it should draw a new frame. This means that if rasterization takes even a little longer than a vsync interval, it will introduce one full frame of extra delay. Correspondingly short raster tasks may finish much earlier than the check for their completion.

Since we cannot make use of a vsync notification within the TouchStart event, the changes outlined here do not improve this use case.

Conclusion

This document describes improvements to vsync scheduling which allows Chrome on Android to generally respond to scroll gestures within a single vsync interval. These improvements apply to regular page scrolling, while lowering the latency of main thread and JavaScript-driven updates are left as future work.

References

- Master bug for tracking Chrome on Android latency improvements: http://crbug.com/230759
- ZILCh (Zero input latency Chrome), which is the umbrella project for input touch latency improvements.
- Scheduler overhaul, phase 1, which is the continuation of this work.