VSync-Aligned Buffered Input

Status: Prototype implemented, shipping on hold
Authors: jdduke@chromium.org
Last Updated: 5/16/2014

Objective

Android is currently the only Chromium platform for which input event delivery is synchronized with the vsync signal. Events on other platforms can arrive at any time, a phenomenon resulting in suboptimal scheduling, increased work and undesirable *motion aliasing*[1]. This proposal outlines a basic design in which all platforms enjoy the benefits of vsync aligned input with minimal interference to the existing input pipeline.

Background

Buffered, vsync-aligned touch input was introduced on Android with Jelly Bean. This addition provided several immediate benefits:

- **Predictable event timing** - All events are delivered to the application just before the vsync signal, minimizing message loop interruption.

- **Reduced event processing costs** - Batching all touch events within a given frame into a single event reduces the total amount of processing required to handle the event stream.

- **Smooth and consistent visual output** - Using short term prediction, each frame can be logically provided a similar duration of input, e.g., if frames alternate between receiving 2 and 3 touch events, the stream of touch events delivered to the app can be generated so as to give each frame equal “weighting” of input[2].

These benefits are desirable for all platforms, not just Android, motivating adoption of a similar vsync-aligned buffering strategy in Chromium. Initial attempts at implementing such a system were more ambitious, and ultimately failed under the weight of excessive complexity and overly

[1] *Motion aliasing* occurs when *input* sampling and *visual* sampling misalign in such a way that logically smooth input and output signals produce janky and erratic visual output.
[2] Here is an example that illustrates the problem: both boxes animate at 60hz, but the top box receives discrete input movement deltas at 100hz while the bottom box receives an interpolated movement delta.

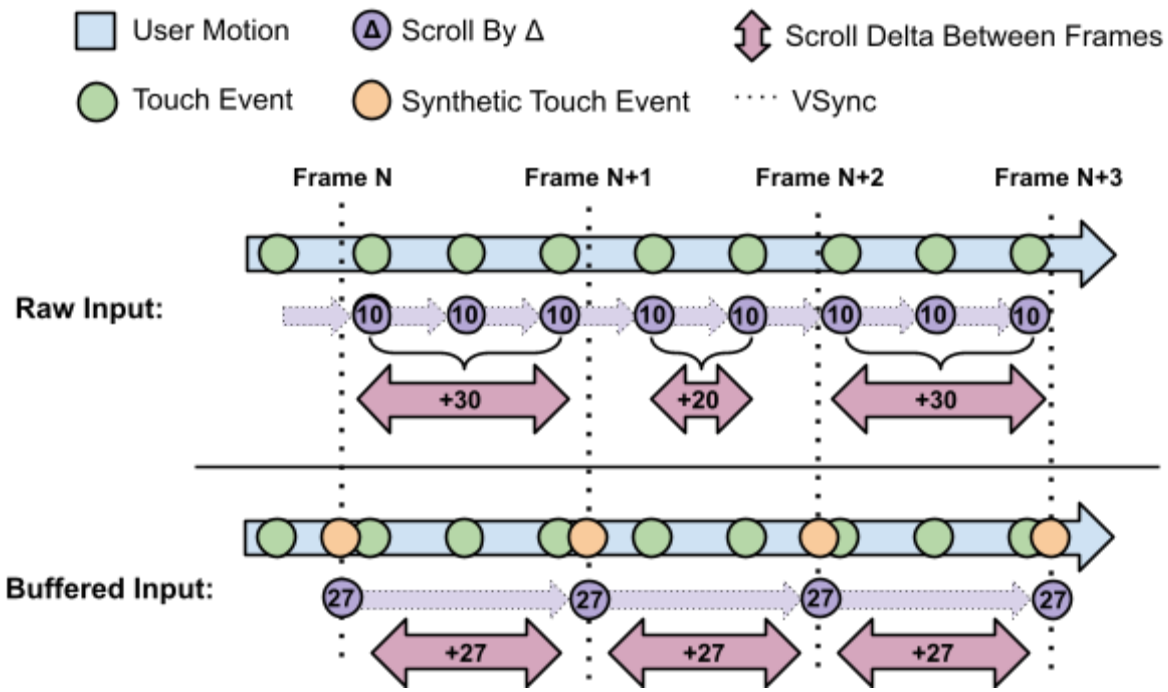
opinionated design. This approach retains the spirit of that design with primary goals of simplicity and flexibility.

Figure A illustrates the common scenario in which a touch digitizer discretely samples a smooth touch stream, generating events at a rate both higher than and unaligned with the vsync signal (150hz vs 60hz). **Raw input** is the current system, and **buffered input** is the proposed system.

- **Raw Input** - Each event is processed singly, generating multiple scroll events per frame. However, because the touch scan frequency is not an even multiple of the vsync frequency, adjacent frames receive an alternating number of discrete touch samples. Even if rendered at a full 60hz, the resulting output will appear jittery.

- **Buffered Input** - Each event is stored in a buffering pool. At the vsync tick, a synthetic touch event is generated from the buffer using events yet fully sampled. The synthetic event is forwarded, generating a single scroll event for that frame. Adjacent frames receive similarly sized sample "windows" of the underlying touch signal.

Figure A: Raw vs Buffered Input[3]



[3] This model is slightly simplified, ignoring the small negative time offset used for synthetic event creation. See Figure C for additional details.

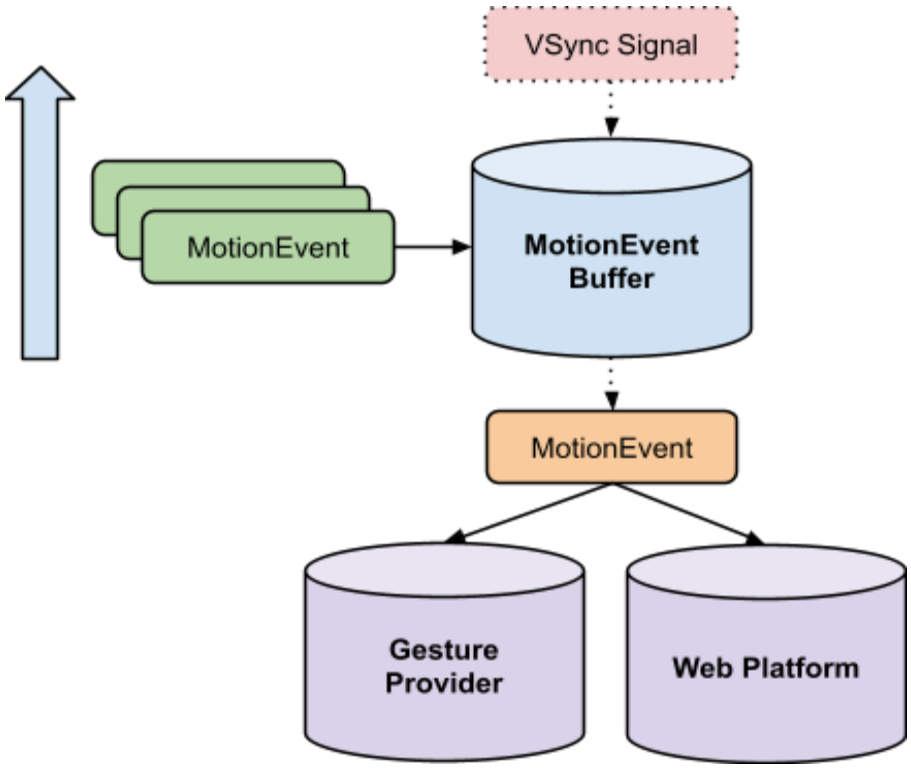
Design

Initially, the vsync-aligned buffering system will be used for touch events. However, its design will be general enough that it can operate on any discretely varying scalar or vector sequence of samples (e.g., touchpad or even mouse events?). This buffering approach dovetails nicely with future efforts[4] toward more intelligent touch prediction, where the event generation strategy can be swapped with more sophisticated models to produce motion events at a given point in time.

MotionEventBuffer

The basic components of the buffered model are ***MotionEvents***[5], a ***MotionEventBuffer*** and a subscribable, regular time signal from the platform[6].

Figure B: Buffered MotionEvent Flow



Event Flow for Input Unaligned With Vsync

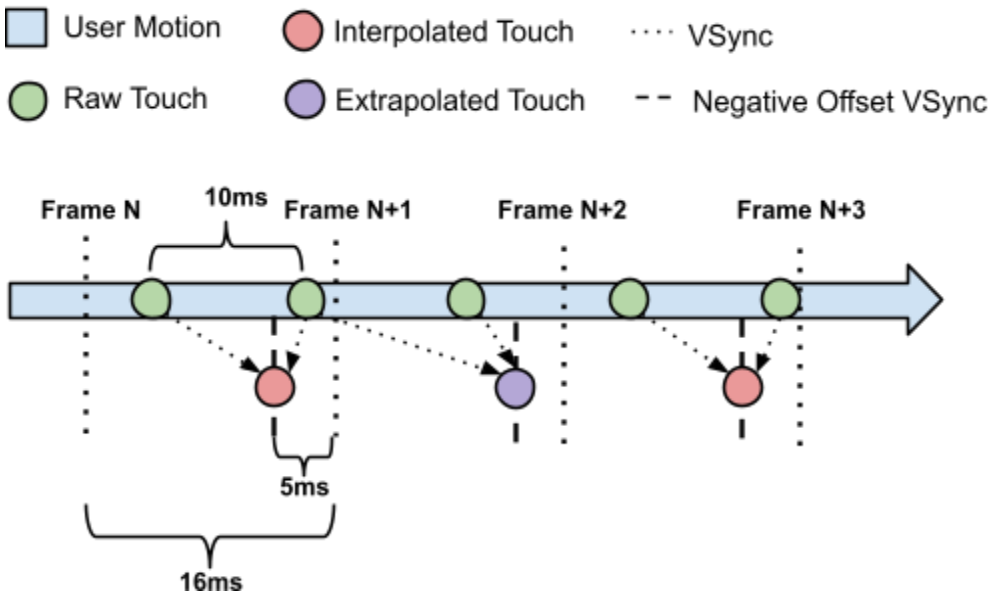
---

[4] Motion Vectors - Likely a Q3 2014 endeavor.
[5] Note that this design mirrors that of the Android platform, but we have the freedom to adjust appropriately for platform-specific requirements and differences.
[6] For now, this is simply the vsync signal time, but in the future this might be the predicted display time.

Input that is unaligned with the display's vsync (e.g., 100hz touch input vs 60hz display output) is the most common case in current devices[7]. In this model, the vsync signal triggers a resampling of buffered events.

1. Raw *MotionEvents* generated by the platform are fed into a *MotionEventBuffer*.
2. A time signal from the platform triggers a query from the *MotionEventBuffer* for a *MotionEvent* sample.
3. The *MotionEventBuffer* synthesizes a *MotionEvent* according to:
    a. If no events are buffered, return immediately without generating an event.
    b. Offset the provided sample time by a small negative delta[8].
    c. If the buffer has an event both before and after the offset sample time, generate an event using linear interpolation[9].
    d. If the buffer has two events before the offset sample time, generate an event using (bounded)[10] linear extrapolation.
    e. Otherwise use the most recent event.
    f. Append raw historical events (those older than the offset sample time) to the returned MotionEvent, flushing such events from the buffer.
4. The synthesized *MotionEvent* is fed to the gesture provider and dispatched to the web platform for consumption.

Figure C: Buffered MotionEvent Resampling



[7] The Nexus 5 is the only Android device of which we are aware for which input is aligned with vsync.
[8] Offsetting by a small negative delta (~5ms) reduces the uncertainty in event extrapolation when the input frequency is either unknown or unpredictable. See Android's implementation for details.
[9] Note that devices like the Nexus 5, where the touch frequency is a phase locked integral multiple of the display frequency, require neither extrapolation nor interpolation; the latest sample at vsync is used directly.
[10] Android caps the extrapolation horizon to 8ms from the latest event.

Event Flow for Input Aligned With Vsync
For devices where it is known that vsync is aligned with input (e.g., the Nexus 5), or such alignment can be reliably and deterministically, no event resampling is required.

1. Raw *MotionEvents* generated by the platform are fed into a *MotionEventBuffer*.
2. If the scan rate is equivalent to the vsync rate, no buffering or resampling occurs and events are always forwarded immediately.
3. Otherwise, the scan rate is an integral multiple (call it N) of the vsync rate:
    a. Events remain buffered until either N events have arrived (a full frame of input), or the vsync tick arrives (a partial frame of input), whichever comes first.
    b. The latest event is forwarded to the system, with preceding buffered events bundled as historical entries in that event.
    c. No resampling (either interpolation or extrapolation) is performed.
4. The latest *MotionEvent* is fed to the gesture provider and dispatched to the web platform for consumption.

Note that in both the aligned and unaligned input cases, MotionEvent's will buffer *only as long as they remain "move-only" type events*. MotionEvent's that involve pointer cancellation, activation or release will *never* buffer. Instead, they will flush any buffered move events and be forwarded immediately. This minimizes potential synthetic latency for discrete gestures such as tapping.

Testing

TODO(jdduke): Describe discrepancy measurement and any other analysis and/or benchmarks used in validating the result.

Implementation

Plan:
1. (Completed) Implement the MotionEventBuffer.
2. (In-progress) Implement a unified vsync signal.
3. Wire the MotionEventBuffer to the vsync signal.

Appendix

```
class MotionEventBufferClient {
 public:
  // Signal from the MotionEventBuffer that an event has been received
  // (and consequently requires sampling).
  virtual void OnMotionEventReceived() = 0;
};

class MotionEventBuffer[11] {
 public:
  explicit MotionEventBuffer(MotionEventBufferClient* client)[12];

  // Called upon receipt of raw MotionEvents from the system.  The
  // event remains buffered until consumed by |SampleMotionEvent()|.
  void OnMotionEvent(scoped_ptr<MotionEvent> event);

  // Generate a (potentially synthetic) MotionEvent for the given time
  // using buffered events, flushing events used in generating the
  // returned event.
  // Note that this may return NULL if no buffered events exist, and
  // it is *not* idempotent[13].
  scoped_ptr<MotionEvent> Sample(gfx::TimeTicks sample_time);

  …

};
```

[11] In practice, this structure should live close to the |FilteredGestureProvider|.
[12] An optional prediction strategy may be supplied in the future, with which event generation can be tailored using more sophisticated motion curves (e.g., using 2nd order least squares for prediction beyond 8-10ms).
[13] Idempotency will likely be a desirable feature for more sophisticated prediction models, but for now flushing the buffer upon sample request is sufficient.