Modeling and Reducing Spatial Jitter caused by Asynchronous Input and Output Rates

Axel Antoine, Mathieu Nancel, Ella Ge, Jingje Zheng, Navid Zolghadr, Géry Casiez

Modeling and Reducing Spatial Jitter caused by Asynchronous Input and Output Rates

Axel Antoine[1,2,3,4,5], **Mathieu Nancel**[5,2,3], **Ella Ge**[1], **Jingje Zheng**[1], **Navid Zolghadr**[1], **Géry Casiez**[2,3,4,5,6]
[1]Google, Kitchener, Canada [2]Univ. Lille, UMR 9189 - CRIStAL, Lille, France
[3]CNRS, UMR 9189, Lille, France [4]Centrale Lille, Lille, France
[5]Inria, France [6]Institut Universitaire de France (IUF)
mathieu.nancel@inria.fr, {eirage, jingjiezheng, nzolghadr}@google.com,
{axel.antoine, gery.casiez}@univ-lille.fr

ABSTRACT
Jitter in interactive systems occurs when visual feedback is perceived as unstable or trembling even though the input signal is smooth or stationary. It can have multiple causes such as sensing noise, or feedback calculations introducing or exacerbating sensing imprecisions. Jitter can however occur even when each individual component of the pipeline works perfectly, as a result of the differences between the input frequency and the display refresh rate. This asynchronicity can introduce rapidly-shifting latencies between the rendered feedbacks and their display on screen, which can result in trembling cursors or viewports. This paper contributes a better understanding of this particular type of jitter. We first detail the problem from a mathematical standpoint, from which we develop a predictive model of jitter amplitude as a function of input and output frequencies, and a new metric to measure this spatial jitter. Using touch input data gathered in a study, we developed a simulator to validate this model and to assess the effects of different techniques and settings with any output frequency. The most promising approach, when the time of the next display refresh is known, is to estimate (interpolate or extrapolate) the user's position at a fixed time interval before that refresh. When input events occur at 125 Hz, as is common in touch screens, we show that an interval of 4 to 6 ms works well for a wide range of display refresh rates. This method effectively cancels most of the jitter introduced by input/output asynchronicity, while introducing minimal imprecision or latency.

Author Keywords
jitter; spatial jitter; noise; input frequency; output frequency; resampling; asynchronicity.

CCS Concepts
•**Human-centered computing** → **Graphics input devices;**

INTRODUCTION
Jitter is defined as "*irregular random movement (as of a pointer or an image on a television screen); also : vibratory motion*" [20]. In signal processing, jitter is defined as a form of timing noise, which can have deterministic and random components. In the HCI literature however, it tends to denote any form of perceivable tremor [24]. Jitter in interactive systems is often observed visually, as it typically translates into e.g. cursor or viewport displacements, or '*spatial jitter*' [27]. Spatial jitter in interactive systems occurs when feedback in response to some input is perceived as unstable or trembling when the input signal is smooth or stationary [10].

Jitter is known to affect human performance [27] and subjective preferences [24], and is usually assumed to originate from imprecise input sensing, scaling factors introduced by interaction techniques, and human limbs tremor [27, 10]. For instance, interaction techniques like Raycasting can introduce large scaling factors that amplify input device noise and human tremor [6]. 'Next-point prediction,' used to compensate interactive latency on a software level, can also introduce jitter by exacerbating minute imprecision and quantization effects in mouse or finger input [24] through extrapolation. These causes, however, are all random in nature: noise in the input movement, or in the treatment of that input. They all convey that some aspects of the interactive loop do not behave as well as they could.

This paper highlights another family of jitter, one that can occur even when every individual component of the interactive loop functions well, and that denotes design rather than technical flaws. In particular, we describe, discuss, and evaluate how the discrepancy between input and output frequencies can generate spatial jitter, as a result of shifting delays between input and output events, and even when every (distinct) hardware and software component works as intended.

This issue is exacerbated in recent devices. As we discuss below, the typical combination of 120-125 Hz input sensing and 60 Hz output refresh rate found in most mobile devices causes little to no extra jitter in theory. However, new mobile devices have started to stray from that trend, especially in terms of output frequency: 80 Hz (Oculus), 90 Hz (Google Pixel 4, HTC Vive), 120 Hz (iPad Pro), or 144 Hz (Valve Index) refresh rates

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

can be observed in mobile or entertainment devices. We will focus on the case of the Google Pixel 4 (120 Hz input, 90 Hz output) released recently. The jitter resulting from this combination of frequencies can be directly compared to its previous version (Pixel 3), and compelled its designers to implement additional software processes to smooth out the phenomenon. We will also discuss these solutions, and characterize their trade-offs using input data gathered from 12 participants using Pixel 3 and Pixel 4 devices.

This paper contributes to a better understanding of the impact of different input and output frequencies on spatial jitter. We first detail the problem from a mathematical standpoint, then introduce a new metric to measure spatial jitter. Based on a data collection we build a simulator to evaluate the impact of display refresh rate on spatial jitter. We used it to compare different techniques and parameters for each technique.

RELATED WORK

Impact of frame rate

End-to-end latency in interactive systems has been shown to increase as display frame rates decrease. For example, switching from a 60 Hz to a 120 Hz screen decreases end-to-end latency by 16 ms on average [9]. Increased latency can in turn degrade performance in 3D selections [31] and touch interaction [18]. This has motivated the development of input and output devices with very high frequencies to reduce latency [25]. Claypool and Claypool showed that (display) frame rate has a much greater impact on user performance than does resolution in games, especially below 15 Hz [14]. This has been confirmed by Janzen and Teather in the context of moving target selection [17], suggesting that frame rate more strongly affects moving target selection than latency.

Impact of noise on performance and user perception

In pointing tasks, Pavlovych and Stuerzlinger identified a trade-off between spatial jitter and latency, with spatial jitter having a strong effect on error rate, roughly inversely proportional to the target size [27]. Teather *et al.* studied the effect of latency and jitter in 2D and 3D pointing tasks [30]. They showed that latency has a much stronger effect on performance than low levels of jitter, but that erratic 'spikes' in jitter bring significant performance costs. Spatial jitter is also known to affect user perception in touch interaction, and is more noticeable in panning and dragging tasks than in drawing tasks [24].

Characterizing jitter

Jitter is often represented as an estimation of the variation between an output signal and a reference value or input signal. In interactive systems, it has been measured *e.g.* as a maximum mean-to-peak value [25] or as an average Euclidean distance between the signal and its reference [24]. However, establishing that reference can be challenging when the input signal is not fully synchronized with the output, *e.g.* as in direct touch. One way is to measure output noise when the device or limb is held steady [25]. This method however does not exclude the noise stemming from the user's own jitter. Another approach consists in obtaining a reference signal from another, more reliable source. For example, the static and dynamic tracking precisions of the Leap Motion were analyzed using a high-precision optical tracking system following the user's input movement [16]. While very precise, this method requires careful hardware setup and a way to track markers to precisely measure the signals of interest.

When input-to-output transformations make it too challenging to use the input signal directly, a reference can be approximated by smoothing the output signal. LaViola used a zero phase shift filter to remove high frequency noise in trajectories with jitter [19]. Nancel *et al.* followed the same approach to estimate the noise introduced by next-point prediction algorithms [24]. However this approach requires tuning parameters for the filter, which is most of the time performed by hand and hard to generalize. As with above, it also "blends" the user's own limb jitter together with the system's.

Finally, interactive jitter is usually modeled as a stochastic process. Pavlovych and Stuerzlinger evaluated the influence of spatial jitter by adding artificial noise to a mouse pointer, modeling spatial jitter as a uniformly distributed noise with a maximum offset expressed in pixels [25]. Taranta *et al.* followed a similar approach to test jitter-filtering techniques [29].

Reducing jitter

Similar to estimating a reference signal, a common way to reduce jitter is to filter the signal. Input devices like computer mice measure relative displacements at high frame rates, typically between 1,500 and 12,000 Hz [8]. This high time resolution appears to be used to average out jitter and other noise sources [27]. Other filtering mechanisms may exist, such as assigning a threshold speed or displacement under which the device or limb is considered static, to smooth out sensing noise. Other techniques include moving average [33], Kalman filter [32], single and double exponential smoothing [19], or the 1€ filter used in interactive systems to reduce jitter while minimizing perceived latency [10, 29].

Resampling techniques

Most displays have a fixed refresh rate, typically set at 60 Hz for desktop computers. Graphic cards most of the time enable vertical synchronization (V-sync) to synchronize the timing of the frame buffer swap with the start of a new scanout. This allows to display whole frames instead of having new frames partially overlapping previous ones, creating visual glitches like fractures in straight lines (tearing).

Some recent screens have variable refresh rates, which allows to synchronize the frequencies of the display and the graphics card, up to the display's maximum refresh rate, while avoiding tearing and stuttering issues (frames displayed multiple times). AMD's FreeSync [1] and Nvidia's G-Sync [26] technologies provide this feature to compatible displays and graphic cards. However it is not available on mobile devices. In practice, the next framebuffer swap timestamp can be estimated from the previous one, using `CVDisplayLink` on macOS [4], `WaitForVBlank` on Windows [22], `CADisplayLink` on iOS [3], and `FrameCallback` on Android [2].

In comparison to most displays, input devices report events at a frequency bound to a nominal (maximum) value. Input devices like touch screens or computer mice monitor position or displacements, and emit events when changes are detected above a predefined threshold amplitude. When such changes are detected more often than the nominal frequency, the events can be coalesced by the operating system to match it [5].

Google Android uses linear interpolation and extrapolation to resample touch input event coordinates 5 ms before frame time for touch devices, as a way to provide a smoother scrolling experience [15]. Chromium recently re-implemented the same solution to be more accurate and reduce noise [13]. Besides these specific solutions, it remains unclear how different input and output frequencies affect visual jitter and extrapolation techniques affect the quality of re-sampling.

Next-point prediction techniques

Next-point prediction techniques provide means for predicting a likely path for the next few input locations [24]. Most of these techniques have been designed for touch interaction. Depending on the underlying principle they use, these techniques can either offer predictions at any point in the future, or only at fixed time intervals. Techniques based on polynomial models like first [11] and second-order Taylor series or curve-fitting can predict over a variable time horizon, as their underlying model includes a time parameter. Android and Chromium, for instance, use first-order Taylor series. Other techniques like Kalman filters, or machine learning approaches trained for fixed latencies [23], are designed to predict fixed time horizons. Nancel *et al.* provide a detailed review of each technique [24].

THEORETICAL EFFECTS OF IN-OUT ASYNCHRONICITY

Spatial jitter in next-point prediction [24] is a consequence of (1) input sensing noise and quantization that can make speed and direction estimation inaccurate, and (2) errors in extrapolating an ongoing trajectory over several frames [23]. However, jitter can occur even with zero input noise and a perfect prediction model — or with no prediction at all. Calculating a visual feedback as soon as an input event is received, and rendering it to be displayed as soon as possible, can introduce jitter when input frequency and output (display) frame rate are asynchronous.

Example of Asynchronicity

Consider the situation depicted in Fig. 1-top with a 100 Hz input stream (10 ms between input events) and a 125 Hz frame rate (8 ms between screen updates), both frequencies assumed constant for simplicity. Let us also assume that sensing, computation, rendering, and display are instantaneous (this point is discussed later).

t=0 ms An input event is received, and a feedback is calculated. That feedback is scheduled to be displayed as soon as possible, and since all computations are instantaneous, it is displayed at $t = 0$.

t=8 ms The next frame time occurs. No new event was received, so the current feedback still corresponds to $t = 0$ ms. *A latency is introduced*, despite all typical sources of delay (sensing, rendering, etc.) being currently null.

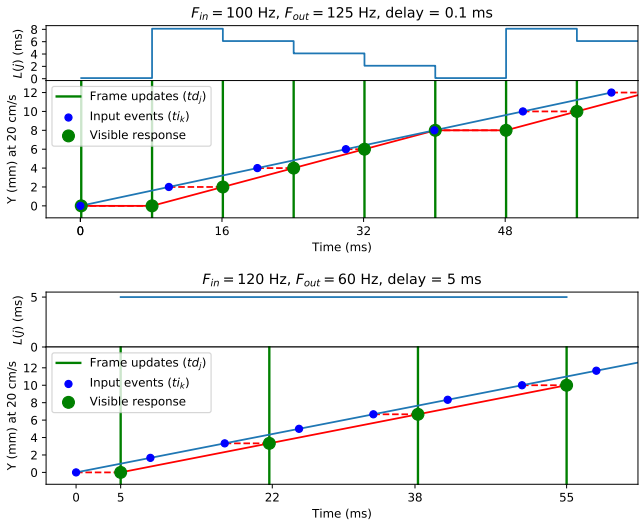

Figure 1: Two examples of input (blue dots) / output (green lines) asynchronicity, and how it affects visual trajectories (red line) during pointing tasks. *Top*: 100 Hz input, 125 Hz output: a repetitive pattern of shifting lag translates into visual jitter. *Bottom*: 120 Hz input, 60 Hz output: the lag is constant and therefore does not cause jitter.

t=10 ms A new input event is received, and its feedback is calculated and ready to be displayed as soon as possible.

t=16 ms The feedback is displayed instantaneously. The instantaneous latency is now $16 - 10 = 6$ ms.

t=20 ms New input event and feedback rendering.

t=24 ms Feedback is displayed, now with 4 ms latency.

t=30 ms New input event and feedback rendering.

t=32 ms Feedback is displayed, now with 2 ms latency.

t=40 ms New input event and feedback rendering. Feedback is displayed, now with zero latency.

t=48 ms Feedback is displayed again, without new input event, so *instantaneous latency* is back to $48 - 40 = 8$ ms.

and so on.

In this example, a temporal lag of varying amplitude occurs as a consequence of the asynchronicity between input and output rates, even in the absence of any other noise or delay. This particular lag follows a pattern of steadily decreasing from 8 to 0 ms, then jumping back to 8 ms, and so on. This pattern occurs every 40 ms, i.e. at 25 Hz.

Up until now we assumed, for simplicity, that sensing, computation, rendering, and display were instantaneous. However, non-null hardware and software latencies, assuming they are roughly constant, would only offset this pattern without changing its effect in user experience. In the example above, visual feedback would still be produced at 100 Hz, even though it would occur some time after each input event; and the visual output would still occur at 125 Hz, despite rendering and display delays. Therefore a pattern equal or similar to $8, 6, 4, 2, 0, 8, 6, 4$,... (ms) would still be observed.

Note that the phenomenon of asynchronicity-induced jitter does not occur systematically as soon as the input and output frequencies differ. For instance if a touch-screen senses finger position at 120 Hz and the screen updates at 60 Hz (see Figure 1-bottom), then the latency at each display event nevertheless remains constant, because the delay between a display update and the previous input event (or available feedback) remains the same.

Note also that the presence and amplitude of this phenomenon *in time units* does not necessarily mean that it is always a usability issue, or even noticeable, as it does not necessarily translates into perceivable cursor or viewport jumps. To be so, the resulting *spatial* jitter needs to be (1) large enough, and (2) on screen for a sufficient amount of time. This introduces an interesting theoretical trade-off, since the amplitude of jitter is likely proportional to speed: fast input movements might create larger jumps, but are usually shorter in time than slow ones. Slow input movements can happen for a longer period of time, *e.g.* when scrolling text while skimming to locate a particular word, but possibly with smaller viewport jumps. One also needs to consider that noise occurring at high velocity can be less noticeable than at low velocity [10, 23]. This theoretical trade-off between amplitude and duration is discussed later in the context of a real-case example.

Generalization

Let us denote F_d and F_i respectively the display and input frequencies. We also denote td_j the j[th] display-update time, and ti_k the k[th] input-event time, with $(j,k) \in \mathbb{I}^2$. Since $td_x \neq ti_x$ for any $x > 0$ unless $F_\mathrm{d} = F_\mathrm{i}$, let us also define that in the following, $k(j)$ denotes the last input event *that occurred before* td_j:

$$\forall j \in \mathbb{I} : k(j) = \{\max(i \in \mathbb{I}) : ti_i < td_j\} \tag{1}$$

The amount of asynchronicity-induced lag $L(j)$ between the time $t = td_j$ of the j^{th} output event, and the time $ti_{k(j)}$ of the most recent input event prior to td_j, is expressed in time units, e.g. milliseconds. Assuming that F_d and F_i are roughly constant for simplicity, $L(j)$ is necessarily smaller than the input *period* $1/F_\mathrm{i}$, and is formulated as:

$$L(j) = td_j - ti_{k(j)} \qquad \text{with } 0 \leq L(j) < \frac{1}{F_\mathrm{i}} \tag{2}$$

In a simple case with constant hardware and software latency, we can approximate *td* and *ti* as follows:

$$td_j = S_\mathrm{d} + \frac{j}{F_\mathrm{d}} \qquad \text{with } -\frac{1}{F_\mathrm{d}} < S_\mathrm{d} < \frac{1}{F_\mathrm{d}} \tag{3}$$

$$ti_{k(j)} = S_\mathrm{i} + \frac{k}{F_\mathrm{i}} \qquad \text{with } -\frac{1}{F_\mathrm{i}} < S_\mathrm{i} < \frac{1}{F_\mathrm{i}} \tag{4}$$

with S_d and S_i some constants representing starting latency values at $j = k = 0$.

At any display time td_j, we can calculate k as follows using Equations (1, 2, 3, 4):

$$\begin{aligned} k &= F_\mathrm{i} \times (ti_{k(j)} - S_\mathrm{i}) \\ &= \lfloor F_\mathrm{i} \times (td_j - S_\mathrm{i}) \rfloor \text{ since } td_{j-1} \leq ti_{k(j)} < td_j \\ &= \left\lfloor F_\mathrm{i} \times \left(S_\mathrm{d} - S_\mathrm{i} + \frac{j}{F_\mathrm{d}} \right) \right\rfloor \end{aligned} \tag{5}$$

$$\text{and thus, } ti_{k(j)} = Si + \left\lfloor F_\mathrm{i} \times \left(S_\mathrm{d} - S_\mathrm{i} + \frac{j}{F_\mathrm{d}} \right) \right\rfloor \times \frac{1}{F_\mathrm{i}} \tag{6}$$

Combining Equations (2-6), we can deduce that:

$$L(j) = S + \frac{j}{F_\mathrm{d}} - \left\lfloor F_\mathrm{i} \times \left(S + \frac{j}{F_\mathrm{d}} \right) \right\rfloor \times \frac{1}{F_\mathrm{i}} \tag{7}$$

$$\text{with } -\frac{1}{F_\mathrm{d}} - \frac{1}{F_\mathrm{i}} \leq S = S_\mathrm{d} - S_\mathrm{i} < \frac{1}{F_\mathrm{d}} + \frac{1}{F_\mathrm{i}} \tag{8}$$

We can reformulate the above by considering that:

$$\left\lfloor z + \frac{xi}{y} \right\rfloor = \left\lfloor \frac{xi + yz}{y} \right\rfloor = \frac{xi + yz - (xi + yz) \bmod y}{y}$$

We then obtain

$$\begin{aligned} L(j) &= S + \frac{j}{F_\mathrm{d}} - \left(\frac{S \cdot F_\mathrm{i} \cdot F_\mathrm{d} + F_\mathrm{i} \cdot j - (S \cdot F_\mathrm{i} \cdot F_\mathrm{d} - F_\mathrm{i} \cdot j) \bmod F_\mathrm{d}}{F_\mathrm{i} \cdot F_\mathrm{d}} \right) \\ &= \frac{[F_\mathrm{i}(S \cdot F_\mathrm{d} - j)] \bmod F_\mathrm{d}}{F_\mathrm{i} \cdot F_\mathrm{d}} \end{aligned} \tag{9}$$

If $L(j)$ is constant, regardless of its value, then there is no visual noise resulting from it, only constant lag. Variations in $L(j)$ cause *e.g.* the cursor or viewport to jump, as illustrated in the previous subsection. Asynchronicity-induced noise therefore manifests itself as a *differential* of $L(j)$.

We define the temporal amplitude of one (asynchronicity-induced) cursor jump, for the j^{th} display event, as

$$\Delta L(j) = L(j) - L(j-1) \tag{10}$$

$$= \frac{1}{F_\mathrm{d}} + \frac{1}{F_\mathrm{i}} \left(\left\lfloor F_\mathrm{i} \left(S + \frac{j-1}{F_\mathrm{d}} \right) \right\rfloor - \left\lfloor F_\mathrm{i} \left(S + \frac{j}{F_\mathrm{d}} \right) \right\rfloor \right) \tag{11}$$

$$= \frac{[F_\mathrm{i}(S \cdot F_\mathrm{d} - j)] \bmod F_\mathrm{d} - [F_\mathrm{i}(S \cdot F_\mathrm{d} - j + 1)] \bmod F_\mathrm{d}}{F_\mathrm{i} \cdot F_\mathrm{d}} \tag{12}$$

with Equation (11) expressed using the "floor" notation similar to Equation (7), and Equation (12) expressed using the "modulo" notation as in Equation (9).

Like $L(j)$, $\Delta L(j)$ is expressed in time units, so the visual amplitude of the resulting jitter is obtained by multiplying $\Delta L(j)$ to finger speed. This effect also depend on the screen's refresh rate F_d, as higher rates mean more frequent jumps.

How this phenomenon translates in actual cursor or viewport jumps during a pointing or scrolling action is complex to model, as pointing gestures happen in multiple bursts of velocity depending on a number of task and setup factors [21, 7]. To simplify, we make the assumption that averaging $\Delta L(j)$

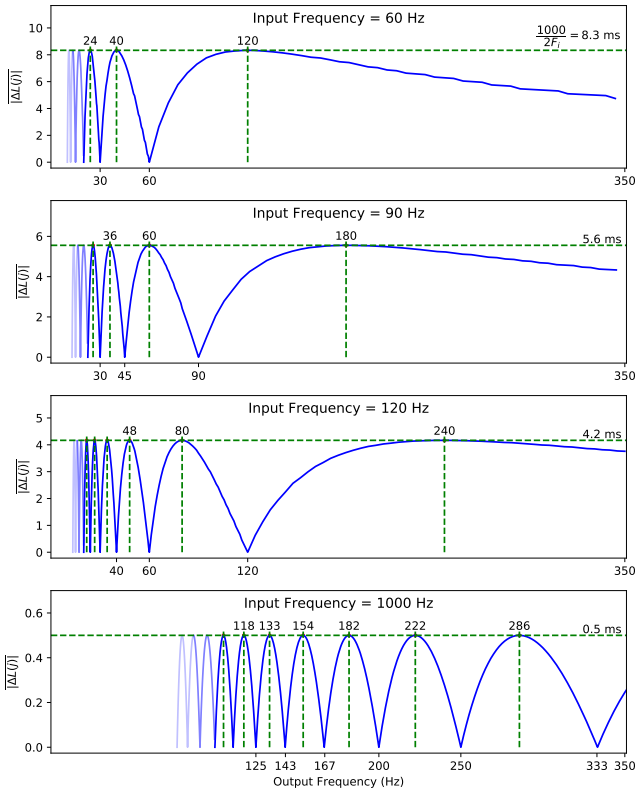

Figure 2: Examples of input/output frequency combinations and the average jitter that they induce, according to our model.

over a stroke provides a usable proxy for the visual amplitude of the noise caused by input-output asynchronicity. We use $\overline{|\Delta L|} = \frac{1}{n}\sum_{j=1}^{n}|\Delta L(j)|$, the mean of the absolute value of $\Delta L(j)$ over a pointing stroke, as an indication of overall noise. It can be formulated as follows (developed in the appendix):

$$\overline{|\Delta L|} = \frac{2a(1-a)}{F_\mathrm{i}} \text{ with } a = \frac{F_\mathrm{i}}{F_\mathrm{d}} - \left\lfloor \frac{F_\mathrm{i}}{F_\mathrm{d}} \right\rfloor \qquad (13)$$

Figure 2 shows the average $\overline{|\Delta L|}$ for combinations of input F_i and display F_d rates, averaged over a simulated stroke lasting 3 seconds ($n = (3 - S_\mathrm{d})/F_\mathrm{d}$). Note that giving random values to S yields the exact same graph. We observe several phenomena.

① First, $\overline{|\Delta L|}$ increases and decreases as a function of F_d in right-skewed 'bumps' (concave spikes) whose minimum seems to be zero.

$$\overline{|\Delta L|} = 0 \iff \Delta L(j) = 0$$
$$\iff [F_\mathrm{i}(S\cdot F_\mathrm{d} - j)] \bmod F_\mathrm{d} - [F_\mathrm{i}(S\cdot F_\mathrm{d} - j + 1)] \bmod F_\mathrm{d} = 0$$

The modulo operator is distributive, so we know that

$$[(a\%n) + (b\%n)]\%n = (a+b)\%n$$

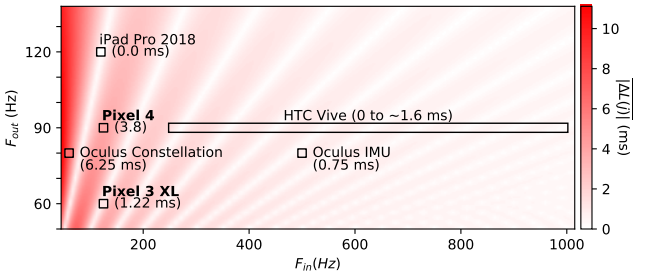

Figure 3: $\overline{|\Delta L|}$ for a subset of F_i and F_d values. The HTC Vive specifications mention a variable range of input frequencies. The Oculus uses an 1000-Hz IMU for input, whose drifting is compensated by cameras (Constellation) at 60 Hz.

(here % denotes the mod operator to save space), and so:

$$([F_\mathrm{i}(S\cdot F_\mathrm{d} - j)]\%F_\mathrm{d} - [F_\mathrm{i}(S\cdot F_\mathrm{d} - j + 1)]\%F_\mathrm{d})\%F_\mathrm{d} = 0$$
$$[F_\mathrm{i}(S\cdot F_\mathrm{d} - j) - F_\mathrm{i}(S\cdot F_\mathrm{d} - j + 1)]\%F_\mathrm{d} = 0$$
$$F_\mathrm{i}\%F_\mathrm{d} = 0$$
$$\iff F_\mathrm{d} = \frac{F_\mathrm{i}}{c}, \text{with } c \in \mathbb{I} \qquad (14)$$

The 'bumps' happen every $F_{\mathrm{d}min} = \frac{F_\mathrm{i}}{c+1} - \frac{F_\mathrm{i}}{c}$ Hz, $c \in \mathbb{I}^+$ when $F_\mathrm{d} \leq F_\mathrm{i}$ (Fig. 2). One last 'bump' occurs for $F_\mathrm{d} \geq F_\mathrm{i}$, that seems to tend towards zero when F_d becomes infinitely high, which fits the expected limit behavior of $F_{\mathrm{d}min}$ when $c = 0$.

② Second, the maximum of $\overline{|\Delta L|}$ seems to be constant throughout all 'bumps' for a given input frequency, but decreases as input frequency F_i increases. This maximum value is $\max(\overline{|\Delta L|}) = \frac{1}{2F_\mathrm{i}}$, and occurs when $F_\mathrm{d} = \frac{F_\mathrm{i}}{c+\frac{1}{2}}$, with $c \in \mathbb{I}$ (the mathematical demonstration can be found in appendix).

According to this model and metric, for any given display refresh rate F_d, the amount of noise induced by asynchronicity is null when the input rate $F_\mathrm{i} = c \times F_\mathrm{d}$, with $c \in \mathbb{I}$, and reaches its worst when $F_\mathrm{i} = F_\mathrm{d}(c + \frac{1}{2})$, with jumps of $|\Delta L(j)| = \frac{1}{F_\mathrm{d}(2\times c+1)}$, in time units. As shown in Fig. 3, the effects of asynchronicity should disappear for high values of input frequencies.

As discussed above, *time* jitter is only a usability issue when its consequences are noticeable, which $\overline{|\Delta L|}$ doesn't express directly. To do so, it needs to be expressed in terms of spatial rather than temporal lag, *e.g.* by multiplying $L(j)$ by the instantaneous input speed at td_j. Instantaneous speed profiles are however notoriously tricky to model—at least predictively [21, 7]—so we discuss below how to measure this empirically.

PRACTICAL ASPECTS OF ASYNCHRONICITY JITTER

Re-sampling Input Events

Smoothing jitter can be done in a number of ways, the most common being filtering the output signal directly, using *e.g.* moving average or the 1€ filter [10, 29].

Another approach, specifically relevant to asynchronicity-induced jitter, consists in estimating input events at a frequency that matches the display's refresh rate, which in effect

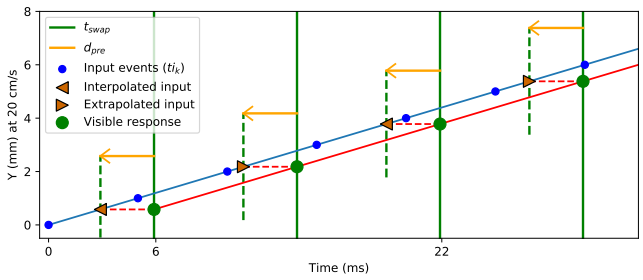

Figure 4: An idea case of smoothing asynchronicity-induced jitter by re-sampling the input signal to match the display refresh rate, using interpolation and extrapolation.

should make $L(j)$ constant. This approach is currently in use in Android [15] and Chromium [13]. It computes the visual feedback for the incoming frame using a virtual input event *re-sampled* to occur at a fixed duration d_{pre} before the next frame. The timing of the next frame t_{swap} is defined as the timestamp of the next GPU swap between the screen buffer and the frame buffer. As illustrated in Figure 4, if the last input event occurred before $t_{\text{swap}} - d_{\text{pre}}$, the location of the re-sampled input event is *extrapolated* using the last few input events; otherwise, it can be *interpolated* between them. Small d_{pre} values therefore increase the chances of extrapolation, which in turn can introduce "prediction" side-effects [24, 23] (including more jitter). Interpolation offers smooth transitions between known input values, but increases the delay between input and output signals.

Measuring spatial jitter

Jitter is usually measured as a noise, *i.e.* as a metric of variation around a reference signal. This reference signal can represent a neutral state, *e.g.* measuring the amount of spatial jitter when a user holds a motion-tracked hand in mid-air. This is however not applicable in our situation, since when a finger is in contact with the screen but not moving, the touch surface will not report any event in many cases.

The reference signal can also be inferred from the output trajectory itself, *e.g.* the viewport positions in a scrolling task, using filtering to produce a "smooth" trajectory for comparison [19]. However, this often requires to tune smoothing parameters by hand, as the distinction between noise vs. meaningful trajectory can sometimes only be distinguished via human expertise. When trying to use that approach, we noticed that these parameters can be quite sensitive to input and output frequencies. For example, our best guesses for the cutoff frequency of a zero-phase shift filter were respectively 0.95 Hz and 0.5 Hz for output frequencies of 30 Hz and 90 Hz. While a feasible approach, hand-tuned parameters also decrease replicability, as different practitioners can have different subjective thresholds for what constitutes noise vs. trajectory variation.

Finally, in some situations a reference signal can be obtained directly from the input signal. This is trivial when the input and output events are fully synchronized, but requires adjustments when not, as is our case here. We use a method similar to $\Delta L(j)$, but applied to position instead of time: given a reference time t_{i}, we calculate the signed difference vector $D(t_{\text{i}})$ between the positions in the input signal $P_{\text{in}}(t_{\text{i}})$ and in the resulting output signal $P_{\text{out}}(t_{\text{i}})$:

$$D(t_{\text{i}}) = P_{\text{out}}(t_{\text{i}}) - P_{\text{in}}(t_{\text{i}}) \tag{15}$$

This is the *spatial* equivalent of $L(j)$ in Eq. (2, 9). If this difference is the same at every frame time, there is no *asynchronicity-induced* jitter to observe—only possibly the one already present in the input signal.

Asynchronicity-induced spatial jitter results from variations in that difference, which will translate into cursor or viewport jumps. Similar to $\Delta L(j)$ above, we express instantaneous spatial jitter *introduced by input-output asynchronicity* in terms of variations of D, *i.e.* :

$$\Delta D(t_{\text{i}}) = D(t_{\text{i}}) - D(t_{\text{i-1}}) \tag{16}$$

A general estimation of the asynchronicity-induced spatial noise over an entire gesture, measured at times $t_{\text{i}} \in \mathbb{T}$, can be formulated similarly to $\overline{|\Delta L|}$ as a mean of absolute differences:

$$\overline{|\Delta D|} = \frac{1}{n}\sum_{i=1}^{n-1} |\Delta D(t_{\text{i}})| \tag{17}$$

$$= \frac{1}{n}\sum_{i=1}^{n-1} |P_{\text{out}}(t_{\text{i}}) - P_{\text{in}}(t_{\text{i}}) - P_{\text{out}}(t_{\text{i-1}}) + P_{\text{in}}(t_{\text{i-1}})| \tag{18}$$

However, in most setups P_{in} and P_{out} are not continuous signals but collections of measurements, and they might not always contain values corresponding to t_{i}. In this case we need to estimate these positions, either using the most recent available value, or through interpolation or extrapolation.

For our purposes, a spatial jitter metric does not need to be usable in real time, so the reference signal $P_{\text{in}}(t_{\text{i}})$ can be interpolated from values prior and posterior to t_{i}. In what follows we denote this $\text{interp}[P_{\text{in}}(t_{\text{i}})]$. The signal P_{out} to which this reference is compared, on the other hand, is calculated throughout the movement in real time, and therefore cannot take future events into account. In the general case without re-sampling, and since in the context of direct touch we do not apply any spatial transformation like C-D gains, this means that for each timestamp $t_{\text{swap},i}$ the value $P_{\text{out}}(t_{\text{swap},i})$ is the position of the last sensed *input* event prior to $t_{\text{swap},i}$, as illustrated in Fig. 1. Reusing the notations in Equation (1), we denote this as follows: $P_{\text{out}}(t_{\text{i}}) = P_{\text{in}}(t_{\text{k(i)}})$.

In the case of re-sampling, the reference times are defined as $t'_i = t_{\text{swap},i} - d_{\text{pre}}$. Since input events can occur prior to $t_{\text{swap},i}$ but after t'_i, some P_{out} values can be interpolated, and others need to be extrapolated:

$$P_{\text{out}}(t'_i) = \begin{cases} \text{extrap}[P_{\text{in}}(t'_i)] & \text{if } t_{\text{k(i)}} < t'_i \\ \text{interp}[P_{\text{in}}(t'_i)] & \text{otherwise} \end{cases} \tag{19}$$

Since reference values $P_{\text{in}}(t_{\text{i}})$ are always interpolated, as explained above, it results that

$$D(t_{\text{i}}) = P_{\text{out}}(t_{\text{i}}) - P_{\text{in}}(t_{\text{i}})$$

$$= \begin{cases} \text{extrap}[P_{\text{in}}(t'_i)] - \text{interp}[P_{\text{in}}(t'_i)] & \text{if } t_{\text{k(i)}} < t'_i \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

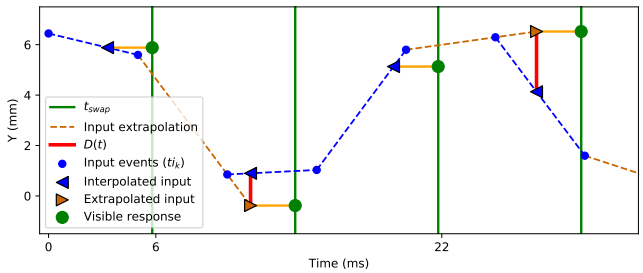

Figure 5: $D(t)$ calculation in a realistic setup with re-sampling. When the last input event before a frame happens after $t'_i = t_{\text{swap}} - d_{\text{pre}}$, interpolation is used, otherwise the input is extrapolated to match t'_i.

Interpolation and extrapolation can be computed using different models, linear being the most straightforward (see *e.g.* in Figure 5). Other, more complex models can be used, typically using higher-level polynomials and prediction algorithms. We will evaluate candidate techniques in the following study.

SIMULATOR STUDY WITH GOOGLE PIXEL DEVICES

To evaluate the effects of re-sampling methods on input/output asynchronicity-induced jitter, we developed a program that can calculate the visual output of a simple scrolling task on any display frame rate, based on existing input data, available at ns.inria.fr/loki/async. We first assess the validity of our model using a real-case example of similar devices with equal input frequency and different frame rates, for which we could collect both input and output event timestamps. Then, using existing re-sampling techniques, we varied the delay d_{pre} between the expected buffer swap time t_{swap} and the targeted input estimation, and observed the resulting trade-offs between reduced spatial jitter and added latency.

We selected the Google Pixel 3 and Pixel 4 as use cases, as the latter adopts a 90-Hz display compared to the more common 60 Hz used in the Pixel 3, while keeping the same touch sensing frequency. Note that Pixel 4 display frequency is 90 Hz only during interactions, and normal refresh rate is 60 Hz. Internal testing revealed that users could see some unwanted jitter before input re-sampling (see above) was implemented in the Chromium app.

Simulator

Given a set of input strokes, the simulator iterates over each individual stroke and artificially replays the received scroll events at a controlled output frequency $F_{\text{d}} = \frac{1}{\Delta_t}$. The simulator generates artificial buffer swap times (t_{swap}) every Δ_t. For realism, it also applies a random (uniform) phase shift $\Delta_i <= \Delta_t$ to determine the initial swap time after the first input event. For each artificial frame time *td*, events with timestamps anterior to the current t_{swap} are unstacked and processed to generate an output coordinate. In our use-case of direct touch input, we do not apply any geometric or scale transformation—such as C-D gains, so input and output coordinates remain the same. This processing also includes re-sampling with varying d_{pre} values, when needed.

Using these resulting "frames" (coordinates), we applied Equations (18, 20) to estimate the spatial jitter introduced by the asynchronicity-induced lag for each individual stroke, as a function of the output frequency parameter F_{d}, and of d_{pre} when relevant.

Data collection

In a controlled experiment, we collected a set of strokes from participants in a series of scrolling tasks, using the touchscreen of two mobile devices: a Google Pixel 3 XL and a Google Pixel 4, which share the same input frequency but differ in their display frame rate. These strokes were later used to simulate the amount of instantaneous, asynchronicity-induced lag using real input data, and estimate the amount and amplitude of the resulting spatial jitter under different conditions.

Method and Apparatus

Both devices have an input frequency of 120Hz (one event every 8.33 ms). The Pixel 3 XL has a 6.3" OLED display with a definition of 2960×1440 pixels (523 dots per inch), an output refresh rate of 60 Hz (one frame every 16.67 ms), a Qualcomm Snapdragon 845 processor with 8 cores at 2.7 Ghz and was running Android 9 Pie. Pixel 4 has a 5.7" OLED display, with a definition of 1920×1080 pixels (444 dpi), an output refresh rate going up to 90 Hz (one frame every 11.11 ms), a Qualcomm Snapdragon 855 processor with 8 cores at 2.84 Ghz and was running Android 10. Scrolling events were recorded locally on both devices, using a web application running in Android Chromium. Each recorded event conveyed enough information to reproduce the interaction sequence as it happened during data collection: coordinates, touch-up and touch-down events, and input timestamp provided by the driver/OS. Automatic scrolling events for inertia were ignored. Logging events using Chromium also allowed us to know the buffer swap timestamp (t_{swap}) for each event, *i.e.* the scheduled display time of each dispatched event's visual feedback. The event dispatcher that unstacks incoming input events is then aware of the next frame's timestamp (See Chromium sources [12], where `args.frame_time` is sent

(a) *Search* task: participants search and click on the green button

(b) *Navigation* task: participants scroll to the opposite side of the page as fast as they can, and click the green button

(c) *Read* task: participants click on all the green buttons dispatched between the *N* sections

Figure 6: Scrolling tasks mimicking the different scroll behaviors on touch devices.

to the `scroll_predictor_` instance). Native re-sampling mechanisms were disabled. We did not ask participants whether they noticed any issue in particular, but in our own pilot tests we did observe some visual jitter on the Pixel 4 when scrolling, especially at low speeds. This would indicate that the theoretical duration/amplitude trade-off mentioned earlier in **Example of Asynchronicity** is tipped in favor of duration, at least in this context of direct touch input.

Procedure, Design, and Task

Similar to Quinn *et al.* [28], we asked participants to scroll long-content web pages on mobiles devices. Web pages were composed of N sections. Each section consisted of a paragraph of text, an image and another paragraph of text. Sections were designed to take a full screen height of the mobile device, as illustrated in Fig. 6.

We asked participants to scroll over the web pages in two *directions* (top, bottom), on both *devices*, and under 3 different *tasks*: *search*, *navigation* and *read* (Figure 6). *Search* consisted in finding and clicking a green button in the middle of the web page (at $N/2$). *Navigation* consisted in scrolling to the opposite side of the web page as fast as the participant could, and click a green button. *Read* consisted in scrolling to the opposite side of the web page while clicking all the green buttons displayed between each section.

The experiment used a $2 \times 3 \times 3 \times 2$ within-subjects design for factors: *device* (Pixel 3 XL or Pixel 4), *task* (search, navigation, read), N (10, 20, 30 sections), and *direction* (top-to-bottom or bottom-to-top). The order of devices, tasks, and directions was counterbalanced across participants. We obtained a total of 432 trials, each consisting of 19.4 strokes on average (min=2, max=114, SD=20.7).

Participants

Twelve participants (5 females, 7 males) participated in the experiment, all software engineers in a mobile phone technology company. We did not instruct them on how to hold the smartphones, but to hold and use them like they normally would. 3 participants put the smartphones on the table and scrolled with their index finger. 1 participant used only one hand to hold the devices and scrolled with the thumb. 3 participants held the devices in one hand and scrolled with the index finger of the other hand. Finally, 5 held the devices with 2 hands and scrolled with the thumb. Participants were instructed to keep the same posture throughout the 20 min study.

Results

The distributions of all input periods indicates that 96% of input events occurred either 8 ms (63%) or 9 ms (33%) after their predecessor for the Pixel 3 XL, and 91% (resp. 80% for 8 ms and 11% for 9 ms) for the Pixel 4. The distributions of all display refresh events (through t_{swap}) indicates that 98% of the output events of the Pixel 3 XL occurred at 60 Hz, and 96 % at 90 Hz for the Pixel 4. Overall, the assumption of constant output frequency formulated for the simulator's design is confirmed, and the assumption of constant input frequency in our theoretical model is mostly confirmed.

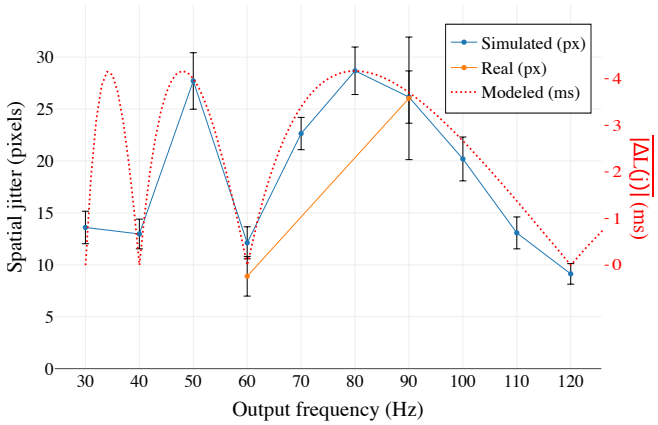

Figure 7: Average spatial jitter (pixels) per output frequency (Hz), computed from the Pixel 3 and 4 (orange) and using the simulator (blue). Error bars represent 95% CI. The jitter in time units, as measured by our model, is overlaid in red on the figure.

Impact of the Display Rate on Spatial Jitter

We measured the spatial jitter (Equations (18, 20)) for the Pixel 3 XL and Pixel 4, using the recorded input and t_{swap} information. As shown in Figure 7, average absolute lag differential increases from 9 pixels at 60 Hz, to 26 pixels at 90 Hz. This difference is in line with direct observations and with our model.

We then used all recorded input from the two devices to simulate different output frequencies. The resulting simulated spatial jitter displays the same trends as our temporal model, with a horizontal shift (intercept) of about 10 pixels likely due to variable input speed, which our theoretical model does not account for. The simulated average spatial jitter at 90 Hz is very similar to the one measured in our data collection (26 px), and about 4 px off for 60 Hz. The levels of spatial jitter obtained through simulation seem coherent both with our analysis and with our real-life measures, confirming that some asynchronous frequencies increase visual jitter.

Effects of Linear Re-sampling

Temporal re-sampling at fixed intervals d_{pre} before new frames (t_{swap}) is generally performed using linear interpolation or extrapolation, as in Android and Chromium at $d_{\text{pre}} = 5$ ms. Therefore, we first evaluated the effects of linear re-sampling. We also varied d_{pre} from 0 to 10 ms, to explore the trade-offs between spatial jitter and latency that we expect to occur with re-sampling.

As illustrated in Figure 8, all values of d_{pre} decrease spatial jitter for all output frequencies F_{d} except when jitter was already minimal at $F_{\text{d}} = F_{\text{i}}/c$, $c \in \mathbb{I}$, as predicted in Equation (14). This suggests that higher values of d_{pre} nearly cancel asynchronicity-induced spatial jitter. They however come at the cost of increased latency. Figure 9 illustrates this trade-off, compared to a baseline without re-sampling. For $d_{\text{pre}} = 10$ ms (green symbols in Fig. 9), re-sampling can add up to 6 ms of latency in average, while $d_{\text{pre}} = 0$ (blue) can reduce the latency of the pipeline by nearly 8 ms using extrapolation. Such differences can be considered small, but Deber *et al.*

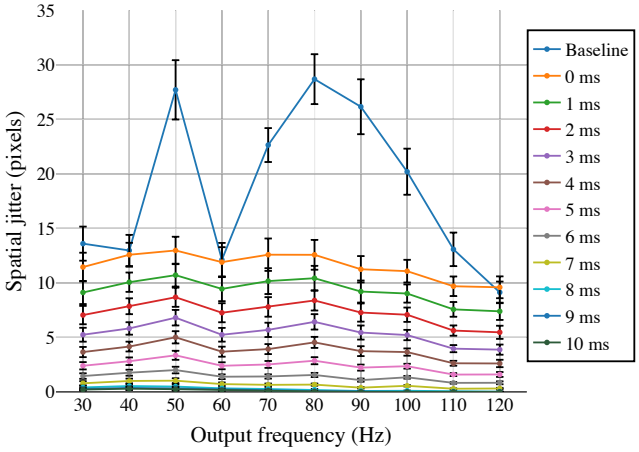

Figure 8: Average spatial jitter (pixels) over output frequency (Hz) for the baseline and different values of linear resampling. Error bars represent 95% CI.

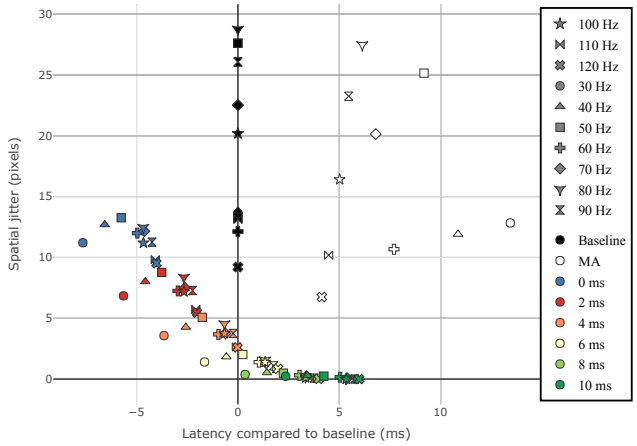

Figure 9: Trade-off between spatial jitter (pixels) and latency (ms) for the baseline technique, moving average (MA) and linear resampling for 0, 2, 4, 6, 8 and 10 ms, and different frequencies. Negative latency corresponds to some latency compensation compared to the baseline and positive values correspond to latency added to the baseline.

have shown that improvements in latency as small as 8 ms are noticeable from a wide range of baseline latencies [18]. $d_{\text{pre}} = 4$ ms (orange in Fig. 9) appears to offer a good trade-off for $F_{\text{i}} = 125\ Hz$, as it systematically reduces both spatial jitter and latency. Based on our results, the choice of $d_{\text{pre}} = 5$ ms in Android and Chromium is a good default value, with only a small increase in latency.

Standard Filtering Techniques

We also assessed how general-purpose filtering techniques fare with asynchronicity-induced jitter, in comparison to re-sampling. We chose simple moving average (MA) as a standard approach, which has the advantages of being straightforward to implement, requiring no parameter tuning, and introducing a deterministic and constant amount of latency $l = 0.5\,(N-1)/F_{\text{i}}$ where N is the number of samples used in the moving average and F_{i} is assumed constant.

As shown in Figure 9, a moving average with $N = 2$ introduces overall more latency than linear resampling, but reduces spatial jitter compared to the baseline, for all output frequencies. Linear re-sampling systematically reduces spatial jitter better than moving average except for $d_{\text{pre}} = 0$ ms (prediction at frametime) for output frequencies of 30, 60, and 120 Hz.

More sophisticated filtering techniques like the 1€ filter could introduce less latency for the same reduction of spatial jitter. However, the exact latency introduced by such techniques is variable and difficult to quantify. Without an accurate estimation of this added latency, we cannot apply our spatial jitter metric. Taken together, filtering techniques do not seem best suited to solve the problem compared to linear re-sampling.

Comparison of Extrapolation Techniques

To re-sample input events, coordinates in the near-future of a trajectory can be estimated in a number of ways in addition to linear extrapolation. Such techniques can be used for instance to predict the next few points of a movement in order to compensate latencies [19, 23], allowing for different degrees of accuracy and negative side-effects depending on

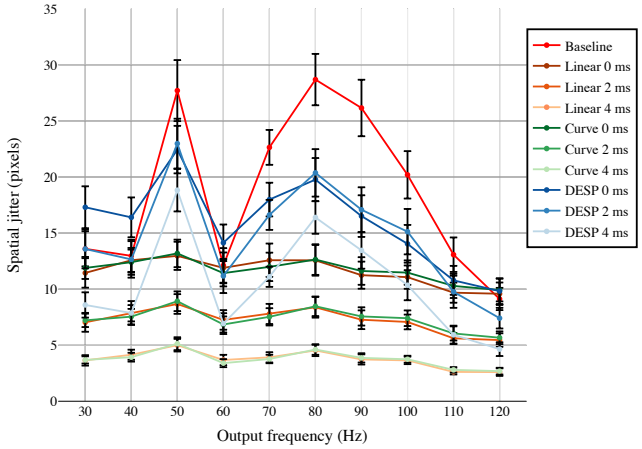

Figure 10: Average visual jitter (pixels) over output frequency (Hz) for the baseline and different predictors. Error bars represent 95% CI.

the predicted duration [24], or "horizon". In particular, second order polynomial curve fitting (dubbed CURVE in [24]) and Double-Exponential Smoothing Predictor (DESP) from LaViola [19] were found to generate fewer side-effects overall compared to other predictors [23]. Other prediction techniques such as Kalman filters or the TurboTouch Predictor [23] can also offer reliable predictions, but are designed to predict at fixed horizons. In our case, the horizon to predict $h = t_{\text{swap}} - t$ is variable, so we did not include these techniques in our tests.

We compared CURVE and DESP to predict 0, 2 and 4 ms before frame time (Figure 10). CURVE shows performance very similar to linear extrapolation, reducing spatial jitter even when predicting at frame time ($d_{\text{pre}} = 0$). DESP also reduces spatial jitter, compared to no re-sampling, but to a lower extent. These results could indicate a lower robustness of DESP to predict at any time in the future instead of fixed time intervals.

DISCUSSION

This work describes and characterizes the phenomenon that asynchronous input and output (display) rates can generate visual jitter in interactive systems, even when every individual component of the interaction pipeline functions perfectly. We proposed a mathematical model of this jitter in time units, and a metric to evaluate it in spatial units. Both were validated using real data from devices with equal input rates and different display rates, which provided a first confirmation of our model and of the validity of the proposed metric. We also explored the strengths and weaknesses of various spatial jitter-reduction methods, the most promising so far being to re-sample input events at fixed delays before frame time. This method is now a default input component in Chromium[1], with a re-sampling parameter that matches our simulated findings.

Our results offer guidelines for system design, in particular Fig. 3 that can be used to estimate how much spatial jitter to expect from a given combination of input/output rates and guide the choice of hardware frequencies. For situations wherein such a choice is not possible, we propose a simulator to explore re-sampling parameters and assess the jitter/latency trade-off.

This work does not aim to offer a definitive solution to this usability issue, but to initiate an effort to characterize and address what seems to be an avoidable problem. The next steps of our research will focus around three goals.

1. Consolidating our theoretical contributions

While promising, our empirical findings need to be extended to more combinations of input/output rates, in order to reinforce the validation of our model and simulator. This is not trivial, as many input or output devices offer little to no variability on their frequencies. We will experiment with high-frequency displays that can be "slowed-down" to commercial standards, and input devices with controllable rates like gaming mice or motion trackers. While we are confident that our theoretical contributions will hold in more varied situations, populating Figure 7 remains useful to assess their variability in real setups. We will also assess the robustness of our findings with scenarios wherein input processing is neither negligible nor constant (*e.g.* 3D games with significant computation upon inputs), thereby introducing variable amounts of latency.

2. Generalizing our findings to noisy interactive pipelines

To isolate the specific phenomenon of asynchronicity-induced jitter, we narrowed our empirical explorations to direct-touch interaction. These setups typically have little to no other sources of noise, such as limb tremor (*e.g.* holding an input device in mid-air), input transformations (*e.g.* applying C-D gains), or additional extrapolations (*e.g.* to compensate system latency), making them a good control testbed. However, we need to validate our findings in noisier environments, to confirm whether our temporal jitter model and spatial jitter metric remain usable amidst other sources of noise, and whether re-sampling can still help.

3. User perception of jitter

We focused this work on scrolling tasks, for simplicity and as a frequent example of jitter issues in direct touch. Previous work showed that there exist perceptive thresholds for jitter, for instance in terms of amplitude [24] and input speed [10]. Jitter can also be perceived differently, and generate different degrees of frustration, depending on the task at hand [24]. The fact that Android implemented a specific software solution to tackle this phenomenon may indicate that it was reported as a significant issue; however, our work so far is limited to geometric characterizations of asynchronicity-induced jitter, and would become more applicable with clear perceptive thresholds depending on interaction context. As of now, our model (see Fig. 3) indicates that increasing input frequency (F_i) is an efficient 'brute force' method to decrease the maximum amplitude of asynchronicity-induced jitter. Increasing output frequency (F_d) is a more complex matter, however, because output rates higher than the input rate can still increase jitter significantly (*e.g.* the last 'bumps' in Fig. 2). We will test our findings on different tasks such as pointing, drawing, and writing, and gather systematic subjective feedback to assess the perception thresholds above which asynchronicity-induced jitter needs to be addressed.

[1] chrome://flags/#enable-resampling-scroll-events

CONCLUSION

This work addresses the problem of visual jitter caused by asynchronous input and output rates in interactive systems. We first describe the phenomenon from a mathematical standpoint, from which we formulate a predictive model of temporal jitter amplitude as a function of input and output frequencies. From this we introduce a metric to measure the corresponding spatial jitter. This metric was validated on a real-case setup, through the collection of scrolling data on two similar devices having the same input rate and different output rates. We further used the collected data to simulate a range of output frequencies and compare different approaches to reduce ansynchronicity-induced jitter. Our results validate our model and show that re-sampling input events to a timestamp before frame-display time is an effective way to cancel asynchronicity-induced jitter, while introducing minimal imprecision or latency. In particular, we show that for devices with 120 Hz input, *i.e.* most touch-screens today, linear re-sampling at 4-6 ms before frame time offers the best results. This technique and parameters are now used in Chromium on most touch devices. Future work will include investigating these issues in different setups, especially in the context of virtual reality.

ACKNOWLEDGMENTS

This work was partially supported by ANR (Causality, ANR-18-CE33-0010-01) and the Google Faculty Research Awards Program.

REFERENCES

[1] AMD. 2015. FreeSync. (2015). Retrieved April 13th, 2020 from https://www.amd.com/en/technologies/free-sync

[2] Android. 2020. Choreographer.FrameCallback. (2020). Retrieved April 13th, 2020 from https://developer.android.com/reference/android/view/Choreographer.FrameCallback

[3] Apple. 2020a. CADisplayLink. (2020). Retrieved April 13th, 2020 from https://developer.apple.com/documentation/quartzcore/cadisplaylink?language=objc

[4] Apple. 2020b. CVDisplayLink. (2020). Retrieved April 13th, 2020 from https://developer.apple.com/documentation/corevideo/cvdisplaylink?language=objc

[5] Apple. 2020c. Getting High-Fidelity Input with Coalesced Touches. (2020). Retrieved April 30th, 2020 from https://developer.apple.com/documentation/uikit/touches_presses_and_gestures/handling_touches_in_your_view/getting_high-fidelity_input_with_coalesced_touches?language=objc

[6] Marc Baloup, Thomas Pietrzak, and Géry Casiez. 2019. RayCursor: A 3D Pointing Facilitation Technique Based on Raycasting. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, Article Paper 101, 12 pages. DOI: http://dx.doi.org/10.1145/3290605.3300331

[7] Robin T Bye and Peter D Neilson. 2008. The BUMP model of response planning: variable horizon predictive control accounts for the speed-accuracy tradeoffs and velocity profiles of aimed movement. *Hum Mov Sci* 27, 5 (oct 2008), 771–798. DOI: http://dx.doi.org/10.1016/j.humov.2008.04.003

[8] Géry Casiez, Stéphane Conversy, Matthieu Falce, Stéphane Huot, and Nicolas Roussel. 2015. Looking through the Eye of the Mouse: A Simple Method for Measuring End-to-End Latency Using an Optical Mouse. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (UIST '15)*. Association for Computing Machinery, New York, NY, USA, 629–636. DOI: http://dx.doi.org/10.1145/2807442.2807454

[9] Géry Casiez, Thomas Pietrzak, Damien Marchal, Sébastien Poulmane, Matthieu Falce, and Nicolas Roussel. 2017. Characterizing Latency in Touch and Button-Equipped Interactive Systems. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. Association for Computing Machinery, New York, NY, USA, 29–39. DOI:http://dx.doi.org/10.1145/3126594.3126606

[10] Géry Casiez, Nicolas Roussel, and Daniel Vogel. 2012. 1€ Filter: A Simple Speed-Based Low-Pass Filter for Noisy Input in Interactive Systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. Association for Computing Machinery, New York, NY, USA, 2527–2530. DOI: http://dx.doi.org/10.1145/2207676.2208639

[11] Elie Cattan, Amélie Rochet-Capellan, Pascal Perrier, and François Bérard. 2015. Reducing Latency with a Continuous Prediction: Effects on Users' Performance in Direct-Touch Target Acquisitions. In *Proc. 2015 International Conference on Interactive Tabletops and Surfaces (ITS '15)*. ACM, 205–214. DOI: http://dx.doi.org/10.1145/2817721.2817736

[12] Chromium. 2019a. InputHandlerProxy (Line 1265). (2019). Retrieved May 5th, 2020 from https://chromium.googlesource.com/chromium/src.git/+/refs/tags/84.0.4136.2/ui/events/blink/input_handler_proxy.cc

[13] Chromium. 2019b. Linear resampling. (2019). Retrieved May 5th, 2020 from https://chromium.googlesource.com/chromium/src.git/+/refs/tags/84.0.4136.2/ui/events/blink/prediction/linear_resampling.cc

[14] Mark Claypool and Kajal Claypool. 2009. Perspectives, Frame Rates and Resolutions: It's All in the Game. In *Proceedings of the 4th International Conference on Foundations of Digital Games (FDG '09)*. Association for Computing Machinery, New York, NY, USA, 42–49. DOI:http://dx.doi.org/10.1145/1536513.1536530

[15] Google. 2010. Android Input Transport (Line 53). (2010). Retrieved May 5th, 2020 from https://android.googlesource.com/platform/frameworks/native/+/refs/tags/android-10.0.0_r36/libs/input/InputTransport.cpp

[16] Jože Guna, Grega Jakus, Matevž Pogačnik, Sašo Tomažič, and Jaka Sodnik. 2014. An Analysis of the Precision and Reliability of the Leap Motion Sensor and Its Suitability for Static and Dynamic Tracking. *Sensors* 14, 2 (Feb. 2014), 3702–3720. DOI: http://dx.doi.org/10.3390/s140203702

[17] Benjamin F. Janzen and Robert J. Teather. 2014. Is 60 FPS Better than 30? The Impact of Frame Rate and Latency on Moving Target Selection. In *CHI '14 Extended Abstracts on Human Factors in Computing Systems (CHI EA '14)*. Association for Computing Machinery, New York, NY, USA, 1477–1482. DOI: http://dx.doi.org/10.1145/2559206.2581214

[18] Ricardo Jota, Albert Ng, Paul Dietz, and Daniel Wigdor. 2013. How Fast is Fast Enough? A Study of the Effects of Latency in Direct-Touch Pointing Tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. Association for Computing Machinery, New York, NY, USA, 2291–2300. DOI: http://dx.doi.org/10.1145/2470654.2481317

[19] Joseph J. LaViola. 2003. Double Exponential Smoothing: An Alternative to Kalman Filter-Based Predictive Tracking. In *Proceedings of the Workshop on Virtual Environments 2003 (EGVE '03)*. Association for Computing Machinery, New York, NY, USA, 199–206. DOI:http://dx.doi.org/10.1145/769953.769976

[20] Merriam-Webster. 2020. Jitter. (2020). Retrieved April 6th, 2020 from https://www.merriam-webster.com/dictionary/jitter

[21] D E Meyer, R A Abrams, S Kornblum, C E Wright, and J E Smith. 1988. Optimality in human motor performance: ideal control of rapid aimed movements. *Psychol Rev* 95, 3 (jul 1988), 340–370. DOI: http://dx.doi.org/10.1037/0033-295x.95.3.340

[22] Microsoft. 2020. WaitForVBlank. (2020). Retrieved April 13th, 2020 from https://docs.microsoft.com/en-us/windows/win32/api/dxgi/nf-dxgi-idxgioutput-waitforvblank

[23] Mathieu Nancel, Stanislav Aranovskiy, Rosane Ushirobira, Denis Efimov, Sebastien Poulmane, Nicolas Roussel, and Géry Casiez. 2018. Next-Point Prediction for Direct Touch Using Finite-Time Derivative Estimation. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology (UIST '18)*. Association for Computing Machinery, New York, NY, USA, 793–807. DOI: http://dx.doi.org/10.1145/3242587.3242646

[24] Mathieu Nancel, Daniel Vogel, Bruno De Araujo, Ricardo Jota, and Géry Casiez. 2016. Next-Point Prediction Metrics for Perceived Spatial Errors. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. ACM, New York, NY, USA, 271–285. DOI: http://dx.doi.org/10.1145/2984511.2984590

[25] Albert Ng, Julian Lepinski, Daniel Wigdor, Steven Sanders, and Paul Dietz. 2012. Designing for Low-Latency Direct-Touch Input. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12)*. Association for Computing Machinery, New York, NY, USA, 453–464. DOI:http://dx.doi.org/10.1145/2380116.2380174

[26] NVidia. 2014. G-Sync. (2014). Retrieved April 13th, 2020 from https://www.nvidia.com/en-us/geforce/products/g-sync-monitors/

[27] Andriy Pavlovych and Wolfgang Stuerzlinger. 2009. The Tradeoff between Spatial Jitter and Latency in Pointing Tasks. In *Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '09)*. Association for Computing Machinery, New York, NY, USA, 187–196. DOI: http://dx.doi.org/10.1145/1570433.1570469

[28] Philip Quinn, Sylvain Malacria, and Andy Cockburn. 2013. Touch Scrolling Transfer Functions. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST '13)*. ACM, New York, NY, USA, 61–70. DOI: http://dx.doi.org/10.1145/2501988.2501995

[29] Eugene M. Taranta, Seng Lee Koh, Brian M. Williamson, Kevin P. Pfeil, Corey R. Pittman, and Joseph J. LaViola. 2019. Pitch Pipe: An Automatic Low-Pass Filter Calibration Technique for Pointing Tasks. In *Proceedings of the 45th Graphics Interface Conference on Proceedings of Graphics Interface 2019 (GI'19)*. Canadian Human-Computer Communications Society, Waterloo, CAN, Article Article 27, 8 pages. DOI:http://dx.doi.org/10.20380/GI2019.27

[30] Robert J. Teather, Andriy Pavlovych, Wolfgang Stuerzlinger, and I. Scott MacKenzie. 2009. Effects of tracking technology, latency, and spatial jitter on object movement. In *2009 IEEE Symposium on 3D User Interfaces*. IEEE. DOI: http://dx.doi.org/10.1109/3dui.2009.4811204

[31] Colin Ware and Ravin Balakrishnan. 1994. Reaching for Objects in VR Displays: Lag and Frame Rate. *ACM Trans. Comput.-Hum. Interact.* 1, 4 (Dec. 1994), 331–356. DOI: http://dx.doi.org/10.1145/198425.198426

[32] Greg Welch and Gary Bishop. 2001. An Introduction to the Kalman Filter. (2001). Retrieved April 30th, 2020 from http://www.cs.unc.edu/~tracker/ref/s2001/kalman/index.html

[33] Andrew Wilson. 2012. Sensor- and Recognition-Based Input for Interaction. In *Human-Computer Interaction Handbook : Fundamentals, Evolving Technologies, and Emerging Applications*, Jan Fagerberg, David C. Mowery, and Richard R. Nelson (Eds.). CRC Press, Chapter 7, 133–156.

Appendix: Characterizing $\overline{|\Delta L|}$

We develop the formulation of $\overline{|\Delta L|}=f(F_\mathrm{i},F_\mathrm{d})$ in a simple case.

First, let us define $r\in\mathbb{R}$ such that

$$F_\mathrm{i}=F_\mathrm{d}\times r \tag{21}$$

We distinguish r's floor $c=\lfloor r\rfloor=\left\lfloor\frac{F_\mathrm{i}}{F_\mathrm{d}}\right\rfloor\in\mathbb{I}$, and r's fractional part $a=\{r\}=\left\{\frac{F_\mathrm{i}}{F_\mathrm{d}}\right\}\in[0,1)$:

$$\frac{F_\mathrm{i}}{F_\mathrm{d}}=\lfloor r\rfloor+\{r\}=a+c \tag{22}$$

We can re-express $\overline{|\Delta L|}$ from Equation (11) as

$$\begin{aligned}
\overline{|\Delta L|} &= \frac{1}{n}\sum_{j=1}^{n}|\Delta L(j)| \\
&= \frac{1}{n}\sum_{j=1}^{n}\left|\frac{1}{F_\mathrm{d}}+\frac{1}{F_\mathrm{i}}\left(\left\lfloor F_\mathrm{i}\left(S+\frac{j-1}{F_\mathrm{d}}\right)\right\rfloor-\left\lfloor F_\mathrm{i}\left(S+\frac{j}{F_\mathrm{d}}\right)\right\rfloor\right)\right| \\
&= \frac{1}{nF_\mathrm{i}}\sum_{j=1}^{n}|c+a+c(j-1)+\lfloor F_\mathrm{i}\cdot S+a(j-1)\rfloor-cj-\lfloor F_\mathrm{i}\cdot S+aj\rfloor| \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\text{since } c,j\in\mathbb{I} \text{ and } F_\mathrm{i}>0 \\
&= \frac{1}{nF_\mathrm{i}}\sum_{j=1}^{n}|a+\lfloor F_\mathrm{i}\cdot S+a(j-1)\rfloor-\lfloor F_\mathrm{i}\cdot S+aj\rfloor| \\
&= \frac{1}{nF_\mathrm{i}}\sum_{j=1}^{n}|\,|\lfloor F_\mathrm{i}\cdot S+aj\rfloor-\lfloor F_\mathrm{i}\cdot S+a(j-1)\rfloor-a| \qquad (23)
\end{aligned}$$

$S=S_\mathrm{d}-S_\mathrm{i}$ represents the difference of the offsets of the output and input signals, assuming their frequency is constant for simplicity. If we consider:

(i) that S_i and S_d are defined null at $j=0$, then $F_\mathrm{i}\cdot S=0$;

(ii) or, that their difference itself S is very small—which would likely be the case if all values are expressed in the International System of Units, *i.e.* in seconds and hertz—then we could also discard the $F_\mathrm{i}\cdot S$ component;

(iii) or, that S and F_i are expressed in *integer* values, e.g. integral amounts of milliseconds and kiloHertz to maintain orders of magnitude, then the two $F_\mathrm{i}\cdot S$ components can be taken out of the floor functions, and cancel each other.

In all three cases, we can simplify Equation (23) as:

$$\overline{|\Delta L|}=\frac{1}{nF_\mathrm{i}}\sum_{j=1}^{n}|\,|\lfloor aj\rfloor-\lfloor a(j-1)\rfloor-a| \tag{24}$$

By definition, $\lfloor aj\rfloor$ is the largest integer $m\le aj$:

$$\lfloor aj\rfloor \;=\; m\in\mathbb{I}:\frac{m}{a}\le j<\frac{m+1}{a} \tag{25}$$

$$\lfloor a(j-1)\rfloor \;=\; m_2\in\mathbb{I}:\frac{m_2}{a}\le j-1<\frac{m_2+1}{a} \tag{26}$$

Since $a\in[0,1)$, $\lfloor aj\rfloor-\lfloor a(j-1)\rfloor$ is either 0 or 1. In particular, it is equal to 0 if the intervals defined in Equations (25) and (26) are the same, *i.e.* if $m=m_2$, and equal to 1 otherwise.

$$\begin{aligned}
\lfloor aj\rfloor-\lfloor a(j-1)\rfloor=0 &\iff \frac{m}{a}\le j-1<j<\frac{m+1}{a} \\
&\iff \frac{m}{a}+1\le j<\frac{m+1}{a} \qquad (27)
\end{aligned}$$

Complementarily, $\lfloor aj\rfloor-\lfloor a(j-1)\rfloor=1$ if $m_2=m-1$:

$$\begin{aligned}
\lfloor aj\rfloor-\lfloor a(j-1)\rfloor=1 &\iff \frac{m}{a}\le j \text{ and } j-1<\frac{m_2+1}{a} \\
&\iff \frac{m}{a}\le j<\frac{m}{a}+1 \qquad (28)
\end{aligned}$$

Thus, for any integer x in any interval $\mathscr{I}=\left[\frac{m}{a},\frac{m+1}{a}\right)$ with $m\in\mathbb{I}$ and $a\in[0,1)$,

$$\lfloor ax\rfloor-\lfloor a(x-1)\rfloor=\begin{cases}1 & \text{if } x\in\left[\frac{m}{a},\frac{m}{a}+1\right)\iff x=\left\lfloor\frac{m}{a}\right\rfloor\\ 0 & \text{if } x\in\left[\frac{m}{a}+1,\frac{m+1}{a}\right)\end{cases} \tag{29}$$

Furthermore,

$$\begin{aligned}
&\lfloor aj\rfloor-\lfloor a(j-1)\rfloor\in\{0;1\} \\
\iff\;& \lfloor aj\rfloor-\lfloor a(j-1)\rfloor-a\in\{-a;1-a\} \\
\iff\;& |\,\lfloor aj\rfloor-\lfloor a(j-1)\rfloor-a|\in\{a;1-a\} \qquad (30)
\end{aligned}$$

To generalize, we can partition any interval $I=[1..n]$ into intervals $\mathscr{I}_k=\left[\frac{k}{a},\frac{k+1}{a}\right)$ with $k\in[0..\lfloor na\rfloor]$. According to Equation (29)-top, $\lfloor ax\rfloor-\lfloor a(x-1)\rfloor=1$ once in every $\mathscr{I}_k$ interval, *i.e.* $\lfloor na\rfloor$ times overall within I. Consequently, $\lfloor ax\rfloor-\lfloor a(x-1)\rfloor=0$ the rest of the time, i.e. $n-\lfloor na\rfloor$ times within I. Combining this with Equation (30), we obtain that:

$$\begin{aligned}
\overline{|\Delta L|} &= \frac{1}{nF_\mathrm{i}}\sum_{j=1}^{n}|\,|\lfloor aj\rfloor-\lfloor a(j-1)\rfloor-a| \\
&= \frac{(1-a)\lfloor na\rfloor+a(n-\lfloor na\rfloor)}{nF_\mathrm{i}} \\
&= \frac{(1-2a)\lfloor na\rfloor+na}{nF_\mathrm{i}} \qquad (31)
\end{aligned}$$

When n is very large, we consider that $\frac{\lfloor na\rfloor}{n}\sim a$, and thus:

$$\overline{|\Delta L|}\sim\frac{2a(1-a)}{F_\mathrm{i}} \tag{32}$$

The resulting curve perfectly overlaps the ones obtained through repeated simulation (Figure 2). From this, we can easily characterize the 'bumps' formed by the curve of $\overline{|\Delta L|}(F_\mathrm{d})$, and in particular their maximum:

$$a_{max}=\arg\max_a(\overline{|\Delta L|})=\arg\max_a[a(1-a)]=\frac{1}{2} \tag{33}$$

$$F_{\mathrm{d}max}=\frac{F_\mathrm{i}}{c+a_{max}}=\frac{F_\mathrm{i}}{c+\frac{1}{2}} \text{ for any } c\in\mathbb{I} \tag{34}$$

Using Equation (32), we can calculate

$$\max(\overline{|\Delta L|})=\frac{1}{2F_\mathrm{i}} \text{ at } F_\mathrm{d}=F_{\mathrm{d}max} \tag{35}$$