Forget Synchrony for Low Latency

Min Hong Yun, Songtao He, and Lin Zhong
Rice University, Houston, TX

Modern systems have an interactive latency of over 50 ms. While humans generally cannot perceive a latency of 50 ms [3], touchscreen latency manifests spatially [1, 2]. For example, when a user draws a line on a screen at 50 cm/sec, a latency of 10 ms would lead to a visible, annoying gap of 5 cm between the finger tip and the line [2]. In this poster, we report a promising design, Asynchrony, that reduces the mean latency by 48%.

Synchrony Contributes to Latency

A significant portion of latency stems from the synchrony in how a mobile system deals with touchscreen use. Modern mobile systems have four objectives in user interaction: consistent frame rate, no frame drops, concurrent screen updates by multiple apps, and no tearing effects. To achieve these objectives, they must synchronize multiple subsystems that operate asynchronously, i.e., input, output, and display as depicted in Fig. 1. They do so by batching user events, buffering graphical output, and using the sync pulse fired by the display subsystem when refreshing.

For a consistent frame rate, the input subsystem sends batched user events to an app on a sync pulse as shown in Fig. 1. To ensure no frame drops, the output subsystem changes buffer ownership at a sync pulse. To support concurrent screen updates by multiple apps, the output subsystem in the figure collects buffers from apps at a sync pulse. To ensure no tearing effects, the display subsystem prohibits apps from drawing on the buffers transferred from the output subsystem until the display subsystem releases them with firing a sync pulse.

In summary, the synchrony enforces waiting time on the subsystems. The input subsystem waits for a sync pulse instead of immediately delivering user events. The output subsystem waits for a sync pulse to collect buffers from apps even when an app finishes drawing. The display subsystem waits for a sync pulse to release buffers.

Design of Asynchrony

To minimize the waiting time, we reexamine the four objectives. Hardware improvements, such as faster CPU, GPU, and larger memory, enable stable 60 fps which, in turn, makes the objectives evolve: ① a consistent frame rate is not necessary when it is above 30 fps; ② frame drops are allowed if they are not consecutive, and frame rate is kept above 30 fps; ③ concurrent screen updates by multiple apps are not necessary for an immersive app, encompassing the entire screen alone; ④ tearing effects can be avoided not by blocking all drawings, but by allowing 'tearing-free' drawings. These changes lead us to develop a new design less reliant on the sync pulse.

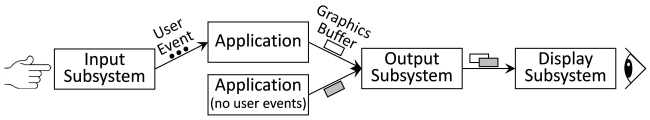

Figure 1: An app receives user events batched by the input subsystem, draws on a graphics buffer, and ships it to the display subsystem via the output subsystem.

We report Asynchrony, a design for low latency, and its implementation on Android. It eliminates synchrony through two core subdesigns: just-in-time trigger and position-aware buffer manager. As a result, Asynchrony achieves a mean latency of 34.5 ms compared to that of 66.2 ms in existing systems. This reduction has a significant, noticeable effect. More importantly, the effect is orthogonal to that of known techniques, including faster hardware and event predictions.

The **just-in-time trigger** achieves the first three objectives by setting two variable deadlines. It predicts two execution times of an app and the output subsystem to set the deadlines to be as late as possible. The input subsystem uses one to deliver more recent events, and the output subsystem uses the other to display a newer frame. A simple prediction algorithm achieves 99.5% accuracy out of 10,000 samples with various CPU and GPU workload configurations. For the remaining 0.5% error, Asynchrony provides a recovery mechanism that drops a delayed buffer while displaying recent one.

The **position-aware buffer manager** achieves the last objective by allowing tearing-free accesses to a buffer currently being displayed. It uses dirty region (changing pixels) information obtained from the display subsystem or an app. When the display has not yet exposed dirty regions and the app can finish drawing before the display reads the regions, the buffer manager allows the app to access the buffer currently being displayed.

References

[1] A. Ng, M. Annett, P. Dietz, A. Gupta, and W. F. Bischof. In the blink of an eye: investigating latency perception during stylus interaction. In *ACM CHI*, 2014.

[2] A. Ng, J. Lepinski, D. Wigdor, S. Sanders, and P. Dietz. Designing for low-latency direct-touch input. In *ACM UIST*, 2012.

[3] S. C. Seow. *Designing and engineering time: The psychology of time perception in software*, chapter 3. Addison-Wesley Professional, 2008.