Technical Disclosure Commons

Defensive Publications Series

October 05, 2017

Low-Latency Rendering

Thomas Buckley

Hannia Zia

Dennis Kempin

David Reveman

LOW-LATENCY RENDERING

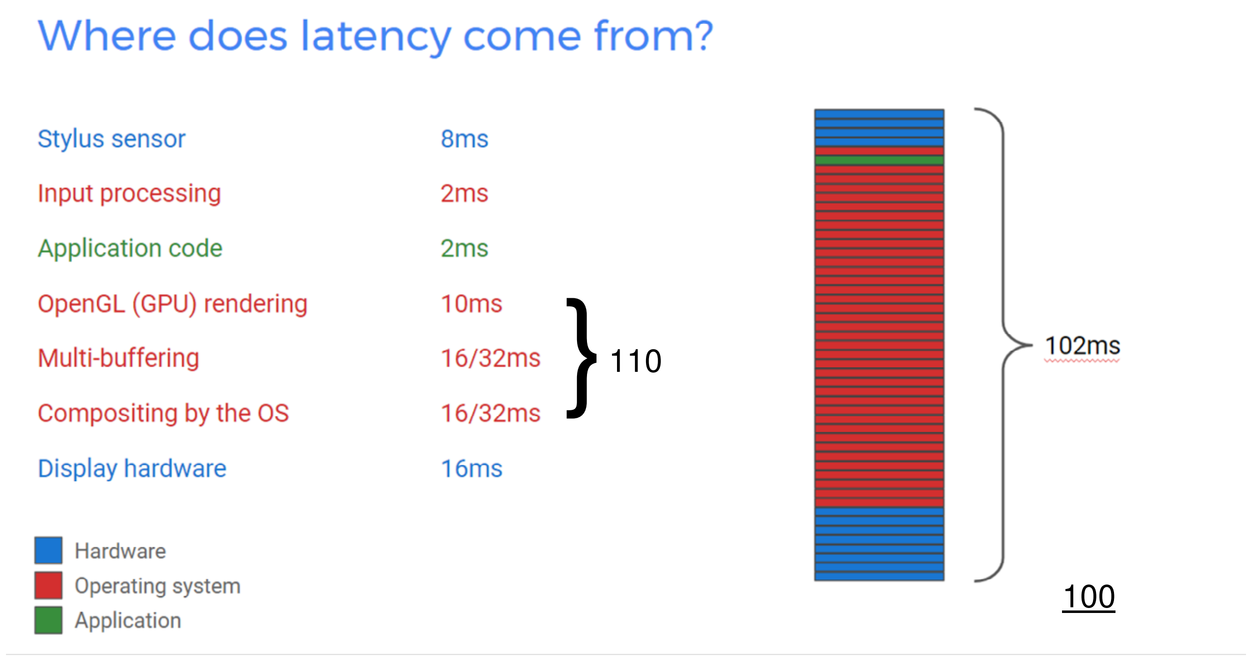
ABSTRACT

An Application Programming Interface (API) on a pen enabled device provides applications with a low latency rendering path for quickly updating a display in response to pen input, for example from a capacitive or resistive digitizer. The API allows applications to define a hardware overlay that bypasses standard rendering pipeline steps such as graphics processing unit (GPU) rendering, multi-buffering, and operating system compositing. Combined with input prediction, the API enables applications to provide low or near zero latency for pen input. Accordingly, pen enabled applications can provide a higher quality user experience that more closely matches traditional pen and paper. Since the API can be used without changes to underlying hardware, reduced latency is possible even for commodity off-the-shelf parts, obviating the need for costly specialized hardware such as high refresh rate displays.

BACKGROUND

Prior to the widespread availability of operating systems with high levels of hardware abstraction, applications may have interfaced with display hardware in a direct, low level manner. Accordingly, an application may provide visual feedback in response to a user input with little to no latency. However, this requires the application to be tailored to a specific hardware configuration, limiting the reusability of the application code. Further, due to the low level hardware access, the application may demand exclusive access to hardware resources, preventing the application from being used in a modern multi-tasking operating system.

As operating systems started to provide standard rendering APIs, application development became highly modular and convenient for developers, but conversely a number of mandatory pipeline steps increased between receiving an input and displaying a response to that input. As illustrated in FIG. 1 below, a standard rendering API may, for example, include conventional rendering steps 110 which include OpenGL (GPU) rendering, multi-buffering (e.g. double or triple buffering), and operating system compositing. While this pipeline may implement best practices for image quality, usability, and system performance in most general cases, a significant latency penalty may be incurred (e.g. 102 ms total latency as shown in FIG. 1). This high latency negatively impacts the user experience for specific use cases, such as drawing or writing with pen input.



Where does latency come from?

Stylus sensor	8ms
Input processing	2ms
Application code	2ms
OpenGL (GPU) rendering	10ms
Multi-buffering	16/32ms
Compositing by the OS	16/32ms
Display hardware	16ms

🟦 Hardware
🟥 Operating system
🟩 Application

FIG. 1

FEATURES OF THE SUBJECT TECHNOLOGY

The subject technology proposes a low latency rendering API for pen enabled devices, such as a laptop, tablet, smartphone, or other device. The API enables application developers to specify a portion of a display for a hardware overlay, for example a portion corresponding to a canvas of the application. When the application is actively in operation, the application may use the low latency rendering API instead of a standard rendering API to update a state of the display within the hardware overlay. The low latency rendering API provides a single-buffered low latency path for updating the display, bypassing the conventional rendering steps 110 shown in FIG. 1. Note that the use of a single buffer allows page copy operations to be avoided, further reducing latency. Thus, latencies between pen inputs and corresponding display updates may be principally limited by hardware latencies, as shown by the 28 ms total latency in FIG. 2.

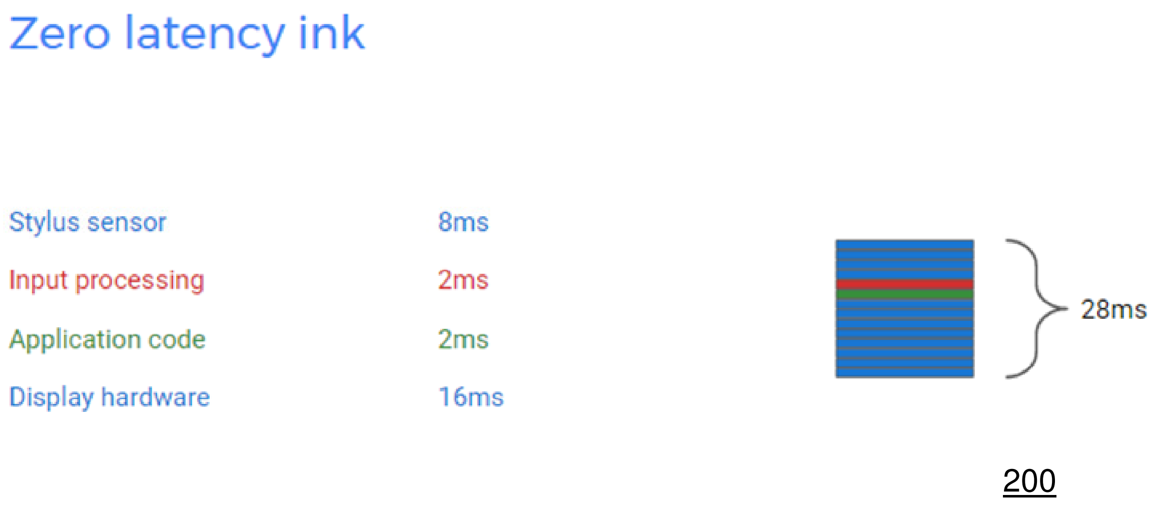


FIG. 2

By utilizing the low latency rendering API in combination with pen input prediction techniques, the latency can be further minimized such that the user perceives a zero or near-zero latency for pen input. For example, by predicting pen input for 2 frames or approximately 32 ms into the future (assuming a refresh rate of 60 Hz), the 28 ms of total latency in FIG. 2 can be effectively zeroed.

Further, since the pen input is provided according to the movement of the user’s hand, an observation can be made that only a small portion of the hardware overlay is typically updated at a time. Thus, the benefits of using a standard rendering API, such as tearing protection, are of less importance when compared to the low latency benefits provided by the hardware overlay. Meanwhile, the remaining display area outside the hardware overlay can be updated normally using a standard rendering API. Thus, other running applications with less demanding latency requirements can still reap standard rendering pipeline benefits such as GPU accelerated rendering, screen tearing protection, Z-culling, and others.

Accordingly, the low latency rendering API combines the low latency advantages of direct hardware access with the numerous benefits provided by the longer pipeline of standard rendering APIs. Note that while the examples above discuss a hardware overlay that is only a portion of the display, the hardware overlay can also be specified to cover the entire display area, for example if the user maximizes a pen enabled application or uses a full screen mode.

Further latency reduction may be achievable by encapsulating the above process within the low latency rendering API, which may have access to low level hardware statistics that are not exposed to the application. For example, display hardware may update per scanline rather than per frame. In this case, a further reduction of latency is possible if hardware overlay updates can be scheduled to meet scanline output timing deadlines. By having the low latency rendering API handle the reading, predicting, and drawing of pen strokes, the low latency rendering API can coordinate these steps to meet the scanline timing deadlines.