Intel® Xe Low Latency (XeLL) Developer Guide 1.1

Intel® Xe Low Latency (XeLL) enables latency reduction in applications to enhance user experience and utilize the GPU in the most optimal way.

Table of Contents

Introduction 3

Behavior 4

Requirements 4

Integration considerations 4

Integration checklist 5

Features 5

Low Latency 5

Frame limiter 6

XeSS-FG support 7

Deployment 8

Naming conventions and branding guidance 8

Programming Guide 8

Initialization 8

Logging Callback 9

Enable Features 9

Sleep 9

Markers 10

Simulation 10

RenderSubmit 10

Present 10

Shutdown 10

Versioning 10

Notices 12

Introduction

An application’s latency depends on multiple factors, such as its design and performance, monitor’s specification (variable or constant refresh rate) and software settings (e.g. VSync). This document focuses on three types of latency metrics, which are visible in Figure 1. The period between frame being submitted to a GPU and rendering start is what we call input-to-render latency. Time that a rendered frame was waiting after render has completed until it was presented on a screen (arrives to a blue vertical line) is called render-to-screen latency. Together with rendering time they create a combined input-to-screen latency.

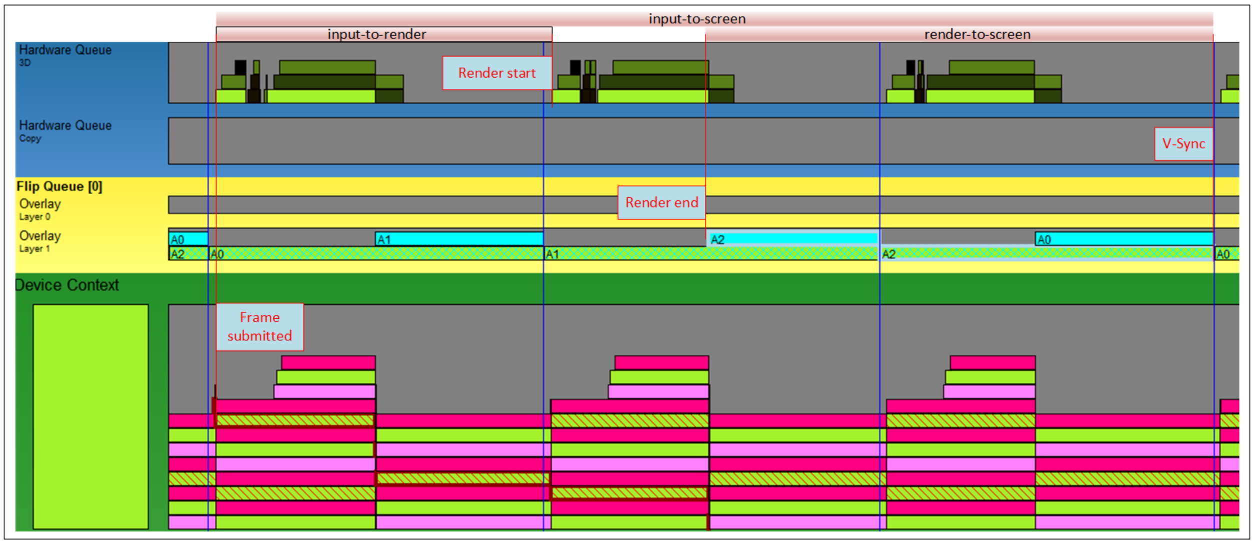

Figure 1 Latency types visible in GpuView.

The input-to-render metric describes the delay between GPU work submission and actual rendering start. Every new submission will be added to a render software queue before it reaches the GPU hardware queue. In the ideal scenario GPU work submission should start around the time when previous frame finishes its execution, while input sampling should be as close to this moment as possible. Input sampling done earlier is adding latency to the application, causing user experience degradation due to an “input lag” feeling.

The render-to-screen metric describes the delay between the finish of GPU rendering and actual presentation on the screen. Optimizing this latency is important when application runs with "no tearing" mode, where present request waits for the next VSync interval to be displayed on a screen. In the ideal scenario GPU rendering should finish as close to this moment as possible. Having a frame ready earlier adds to increased latency.

XeLL, when integrated into the application, receives per-frame timing information that allows the library to calculate the CPU delay that an application should apply before work on the next frame begins. By doing so, the time from when the application begins work on a CPU to presentation on a screen is minimized, that is input-to-screen latency is decreased, allowing for best user experience. The additional delay is calculated so that latency is minimized, while the application’s performance is preserved.

Behavior

The `xellSleep` performs a CPU-side wait before the frame begins and implements two kinds of latency optimizations.

The first is render queue reduction, which is always available. This optimization instructs the application to wait before CPU work for a next frame begins until a last possible moment, so that when command lists pertaining to the next frame are submitted to a GPU, it will not be queued, but its GPU processing will start almost immediately. This results in a decrease of input-to-render latency.

The second mode of latency reduction makes sure that the moment when a frame is ready for presentation is aligned with VSync intervals of a monitor while preserving the minimal latency. This mode is only active when the application's performance is sufficient and applies when DXGI Present has Sync Interval set to 1.

Requirements

- Intel® Arc A-Series GPU devices or later,
- Monitor connected directly to the rendering GPU,
- Windows 10/11 x64 - 10.0.19043/22000 or later,
- DirectX12,
- DXGI flip model used.

Any other forms of latency reduction or wait inducing technologies like frame limiters should be disabled when XeLL is in use. If the requirement is not satisfied, then timings gathered by XeLL may become inaccurate and might result in undefined behavior.

Passing frame limit to XeLL allows it to enable certain optimizations which would be not available in case of a separate frame limiter. If an application implements its own frame limiter the wait operation should happen before `xellSleep` gets called. Having both, application frame limiter and XeLL active at the same time is discouraged.

When it comes to scheduling the work, application should not implement any additional submission logic, like waiting for previous frame to finish. With XeLL enabled, work submission will be controlled by `xellSleep`, as it will ensure that each frame starts submitting at the right time to utilize the GPU in the optimal way and create best user experience by reducing the latency.

Applications should allow VSync when XeLL is enabled, but only allow to use Sync Interval set to 1 in DXGI Present call.

Integration considerations

It is recommended that an application calls XeLL API each frame even if all the features are currently disabled. XeLL ensures that minimal work is done in the system when running in pass-through mode. Features should be controlled only via `xellSetSleepMode`. Such approach should simplify the integration and minimize work that needs to be done during settings change.

In the future, integration might bring additional benefits to the user using Arc Software and driver cooperation. In other words, it is recommended for the application's behavior to be the same with XeLL active or inactive. The only difference would be the state provided in `xellSetSleepMode`.

Integration checklist

Simple integration checklist, ensuring that crucial requirements are fulfilled for XeLL:

- `xellSleep` and `xellAddMarkerData` are always called,
- features are controlled only via `xellSetSleepMode`,
- when VSync is enabled SyncInterval in Present must be always 1 when Low Latency is active,
- FPS Cap is controlled via `xellSetSleepMode` API when Low Latency mode is active,
- application's behavior should be the same with XeLL active or inactive,
- application must ensure any GPU activity is finished before `xellSetSleepMode` is called,
- application must ensure that XeSS-FG object is destroyed before XeLL object.

Features

Feature Name	*GPU Vendor*	*GPU Gen*
Latency reduction	Intel only	Intel Arc GPUs
Frame limiter	Intel only	Intel Arc GPUs

Low Latency

Enabling this feature causes XeLL to delay frames to improve latency in applications. The exact behavior depends on the application's frame time, monitor specification and current display settings (tearing or no-tearing). Any other form of latency reduction should be disabled when the Low Latency feature is enabled.

In tearing mode XeLL will ensure that the render queue is almost empty, meaning the first submission in each frame happens around the time when previous work is completed, maintaining the same level of GPU utilization. As the presentation happens almost immediately after the frame is ready, there is no render-to-screen delay, only input-to-render is lowered.

In no-tearing mode XeLL will additionally try to align render end with VSync intervals of the monitor if the performance of the application is equal or higher than monitor's refresh rate. Such approach, combined with ensuring that the render queue is almost empty, enhances both input-to-render and render-to-screen latency. In figure 2 each frame has minimal input-to-render latency, work starts immediately. After the frame is rendered, XeLL maintains a small safety band to ensure the frame does not miss its VSync Interval as the present would have to wait for the next one, increasing render-o-screen latency.

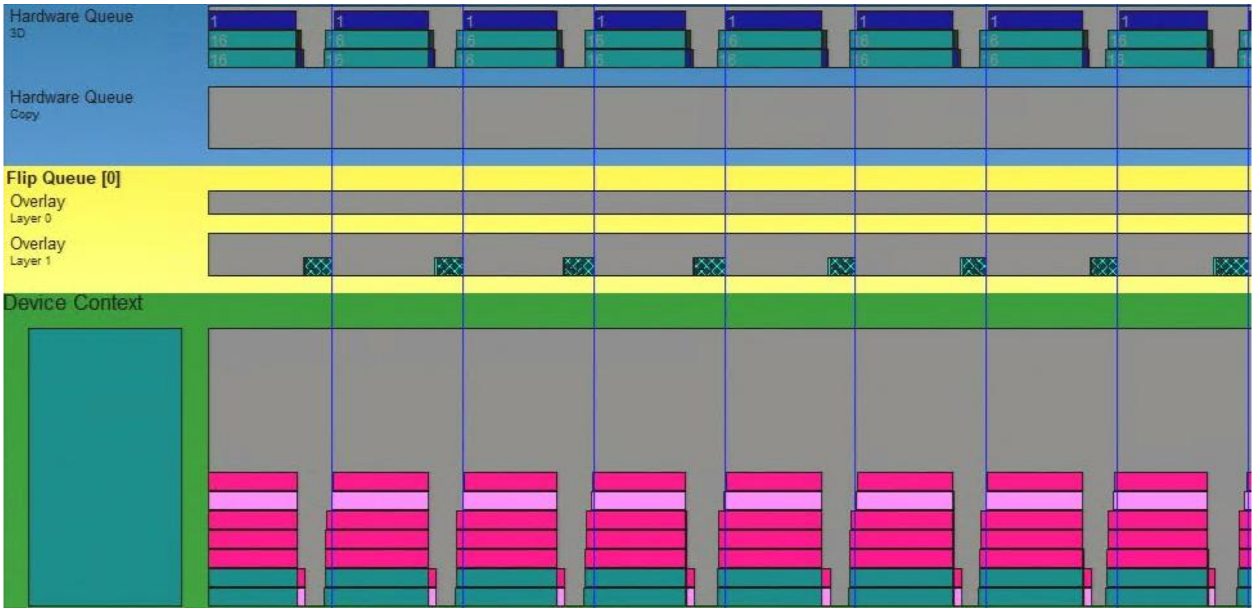

Figure 2 GpuView showing application running with VSync and Low Latency Enabled

Frame limiter

Frame limiter can be used without Latency Reduction. The application should ensure that any internal frame limiter should be disabled if enabled in XeLL.

Enabling Frame Limiter with Latency Reduction gives XeLL the possibility to optimize latency where it was earlier not possible due to not sufficient performance in no-tearing modes. In such case aligning rendering end with VSync intervals will happen if applications performance is equal or higher than Frame Limit, enhancing render-to-screen. Frame limiter does not affect tearing mode, XeLL will continue to ensure that there is no render queue to minimize input-to-render.

In Figure 3 you can see an application with frame limit set to half of the screen refresh rate (e.g. 30 FPS on 60 Hz screen). With VSync enabled each frame was aligned with every other blue vertical line, while maintaining safety net.

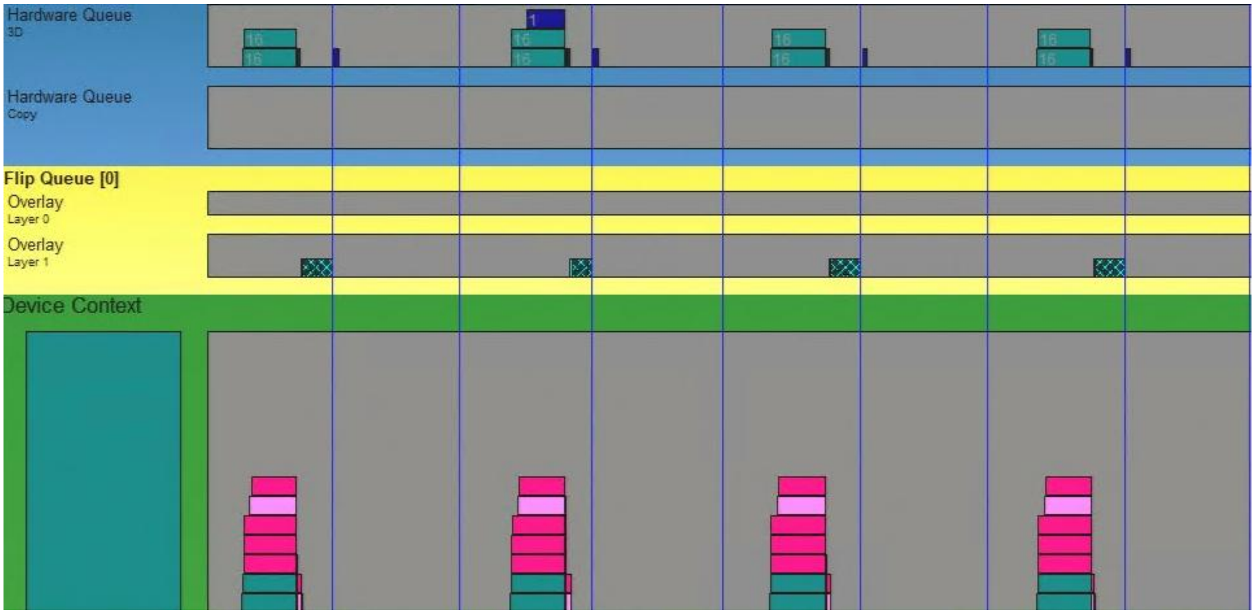

Figure 3 GpuView showing application running with VSync and Low Latency enabled with frame limit set to half of the screen frequency

XeSS-FG support

XeLL is required when using XeSS-FG. If Frame Generation is detected XeLL will account for additional Present calls. Frame Generation affects no-tearing mode as one frame will produce two Presents utilizing two VSync intervals. Due to this, if the application's performance is equal or higher than half of the monitor's refresh rate then XeLL will limit the application to run at half the frequency of the monitor and align rendering end with VSync interval. The goal is to limit render-to-screen for both generated and rendered frame.

In Figure 4 an application with XeSS-FG and VSync enabled renders at half of the screen's refresh rate, but due to the generated frames each VSync interval is utilized. This minimizes the input-to-render and render-to-screen latency for both Presents in each frame.

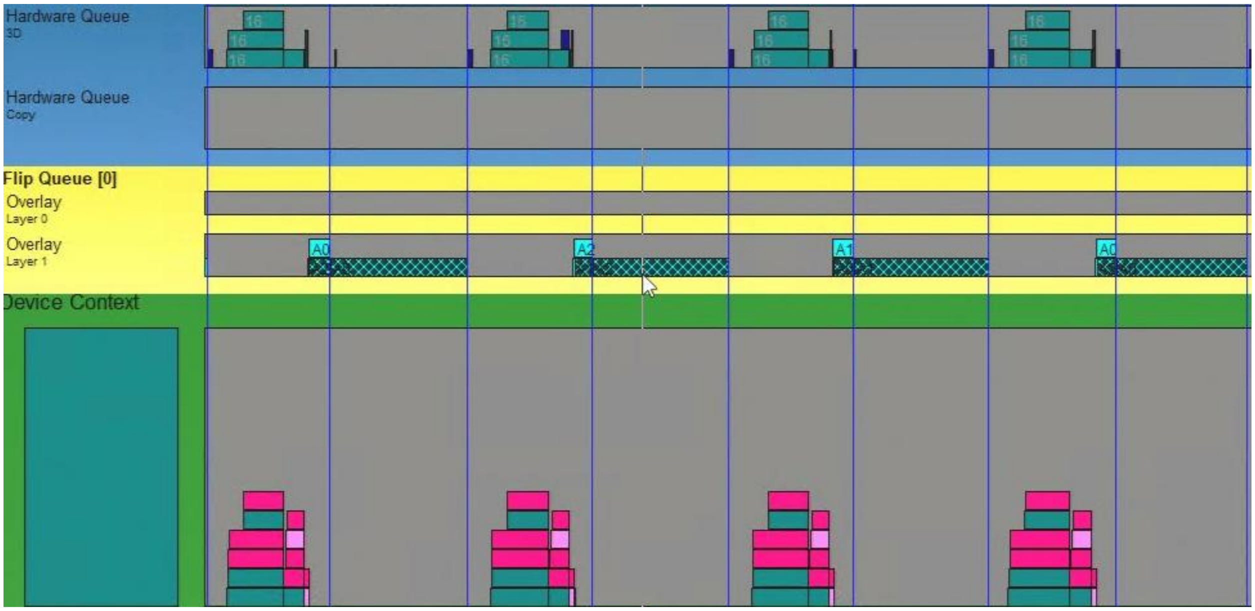

Figure 4 GpuView showing application running with VSync, Low Latency and XeSS-FG enabled.

With Frame Limiter, if application's performance is equal or higher than half of the FPS target, XeLL will limit the frame rate to half of the target ensuring that the total count of presented frames (both generated and rendered) matches requested FPS. For example, if user specifies 60 FPS on a 120 Hz screen, then XeLL will target 30 FPS (each frame calling Present two times) resulting in 60 FPS. As the screen is 120 Hz, every other VSync will be used.

Deployment

To use XeLL in a project:

- Add "inc" folder to the include path
- Include xell.h and xell_d3d12.h
- Link with lib/libxell.lib

The following file must be placed next to the executable location or in the DLL search path:

- Libxell.dll

Naming conventions and branding guidance

Please see "XeSS 2 Naming Structure and Examples.pdf" for approved naming conventions, branding guidance and settings menu examples.

Programming Guide

Initialization

XeLL is initialized by creating a context object. This is the only operation which is API specific. For DirectX12, `xellD3D12CreateContext` method should be used:

```
xell_result_t xellD3D12CreateContext(ID3D12Device* device,
      xell_context_handle_t* out_context);
```

The function will return `XELL_RESULT_ERROR_UNSUPPORTED_DEVICE` if device is not supported, `XELL_RESULT_ERROR_UNSUPPORTED_DRIVER` if the driver installed is not supported. Context will not be created in such cases. The application should validate if context is not a null pointer and if the returned code was `XELL_RESULT_SUCCESS`.

Logging Callback

Receiving API logs can be done by registering a log callback:

```
static void LogCallback(const char* message, xell_logging_level_t level);
xell_logging_level_t log_level =
      xell_logging_level_t::XELL_LOGGING_LEVEL_WARNING;
xellSetLoggingCallback(m_hXell, log_level,
      (xell_app_log_callback_t)LogCallback);
```

There are four levels of logging:

- `XELL_LOGGING_LEVEL_DEBUG`,
- `XELL_LOGGING_LEVEL_INFO`,
- `XELL_LOGGING_LEVEL_WARNING`,
- `XELL_LOGGING_LEVEL_ERROR`.

We recommend using error level for production and warning or debug level for development.

Enable Features

After context creation an application should turn on required features using `xellSetSleepMode`. The function call needs to be done only once after settings change. There is no need to recreate context if settings change. Currently, latency reduction and frame limiter features are supported, represented by `bLowLatencyMode` and `minimumIntervalUs` fields in `xell_sleep_params_t` structure.

```
xell_result_t xellSetSleepMode(xell_context_handle_t context, const
      xell_sleep_params_t* param);
```

`bLowLatencyMode` - when set to ‘true’ enables latency reduction.

`minimumIntervalUs` - use this parameter to force a desired frame limit by passing the time in us, for example for a cap of 30 FPS set this to 33333.

Sleep

At the beginning of each frame, before sampling user input, an application should call `xellSleep` which calculates and executes a delay for a current frame based on previous statistics and markers. Frames are uniquely identified by an application by assigning them increasing `frame_id` values. `xellSleep` must be the first XeLL API function called with newly assigned `frame_id` value if Low Latency is enabled.

```
xell_result_t xellSleep(xell_context_handle_t context, uint32_t
frame_id);
```

Markers

An application provides XeLL with timing information by means of markers. When given marker is delivered to XeLL, it allows the library to know what phase in a frame application starts or finishes with. All markers enumerated below must be delivered per each frame for XeLL to correctly calculate and execute `xellSleep` API call.

Available markers are defined by `xell_latency_marker_type_t` enum. Delivery of a marker is done by `xellAddMarkerData` API function.

```
xell_result_t xellAddMarkerData(xell_context_handle_t context,
      uint32_t frame_id, xell_latency_marker_type_t marker);
```

Markers define the following phases of the frame. It's expected that each *END marker comes after corresponding *START marker.

Simulation

`XELL_SIMULATION_START` and `XELL_SIMULATION_END` markers delimit the period when application does CPU calculations and updates required to prepare GPU work for current frame. During this time application samples input devices.

After sleep was called `XELL_SIMULATION_START` is expected to be the first marker in each frame.

RenderSubmit

`XELL_RENDERSUBMIT_START` and `XELL_RENDERSUBMIT_END` markers delimit the period when application populates command lists and submits them to execution.

Present

`XELL_PRESENT_START/XELL_PRESENT_END` markers delimit call to DXGI Present.

Shutdown

For a clean shutdown the application must call `xellDestroyContext` after making sure that all GPU activities stopped.

```
xell_result_t xellDestroyContext(xell_context_handle_t context);
```

If frame generation is used, application must ensure that XeLL object is destroyed after XeSS-FG object.

Versioning

The library uses a major.minor.patch <major>.<minor>.<patch> versioning format, and Numeric 90+ scheme, for development stage builds. The version is specified by a 64-bit sized structure (xell_version_t), in which:

- A major version increment indicates a new API, and potentially a break in functionality.
- A minor version increment indicates incremental changes such as optional inputs or flags. This does not change existing functionality.
- A patch version increment may include performance or quality tweaks, or fixes, for known issues. There is no change in the interfaces. Versions beyond 90 are used for development builds to change the interface for the next release.

The version is baked into the XeLL SDK release and can be accessed using the function xellGetVersion.

Notices

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

Performance varies by use, configuration, and other factors. Learn more at www.Intel.com/PerformanceIndex.

No product or component can be absolutely secure.

All product plans and roadmaps are subject to change without notice.

Your costs and results may vary.

Intel technologies may require enabled hardware, software, or service activation.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at intel.com.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications, and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Microsoft, Windows, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.