Intel® Xe Low Latency (XeLL)

开发者指南 1.1

Intel® Xe Low Latency (XeLL)，也叫低延迟技术，它通过减少应用程序的延迟来增强用户体验，并以最优方式利用 GPU。

目录

介绍

应用程序的延迟取决于多个因素，比如其架构设计和性能、显示器的规格（可变或恒定刷新率）以及软件设置（如 VSync）。本文档重点介绍图 1 中显示的三种延迟指标。帧提交到 GPU 和开始渲染之间的时间被称为输入到渲染延迟（input-to-render）。渲染完成后帧等待呈现到屏幕上的时间（到达蓝色垂直线）被称为渲染到屏幕延迟（render-to-screen）。它们与渲染时间一起构成了组合的输入到屏幕延迟（input-to-screen）。

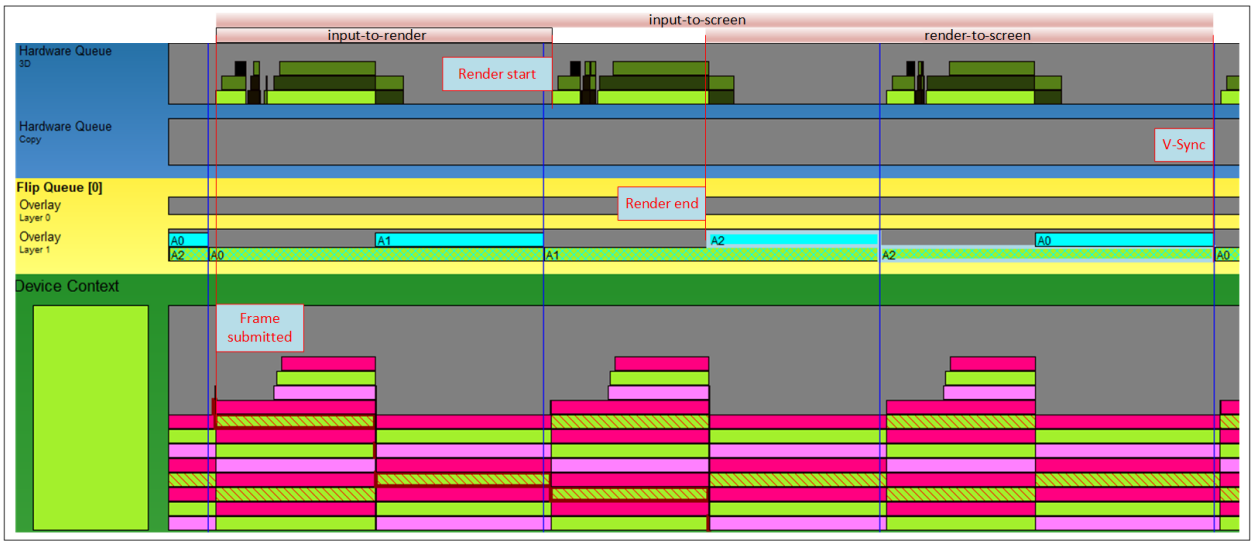

图 1 GpuView 中的延迟类型

输入到渲染指标描述了 GPU 工作提交和实际渲染开始之间的延迟。每个新提交的渲染任务将在到达 GPU 硬件队列之前被添加到渲染软件队列中。在理想情况下，GPU 工作提交应在上一帧执行完毕时开始，同时输入采样应尽可能接近这一时刻。较早的输入采样会增加应用程序的延迟，导致用户体验下降，产生“输入滞后”的感觉。

渲染到屏幕指标描述了 GPU 渲染完成和实际呈现在屏幕上的延迟。当应用程序以“无撕裂”模式运行时优化此延迟非常重要，此时 Present 请求需要等待下一个 VSync 间隔才能显示在屏幕上。在理想情况下，GPU 渲染应尽可能接近这一时刻完成。过早的完成 GPU 渲染会增加延迟。

XeLL 集成到应用程序中后，会接收每帧的定时信息，使 XeLL 能够计算应用程序在下一帧开始之前应施加的 CPU 延迟。这样做，可以将从应用程序在 CPU 上开始工作到屏幕上呈现的时间最小化，即输入到屏幕延迟减少，从而提供最佳用户体验。采用这样的计算，可以在最小化延迟的同时还能保持应用的性能。

行为

xellSleep 在开始帧之前执行 CPU 端等待，并实现两种延迟优化。

第一种是减少渲染队列的排队，该优化始终有效。此优化指示应用程序在下一帧的 CPU 工作开始之前等待到最后一刻，这样当与下一帧相关的命令列表提交到 GPU 时，它不用排队，而是几乎立即开始 GPU 处理。这可以减少输入到渲染的延迟。

第二种延迟减少模式是，确保帧准备好 Present 的时间与显示器的 VSync 间隔对齐，从而保持最小延迟。此模式仅在应用程序性能足够时才会激活，并适用于 DXGI Present 的 Sync Interval 设置为 1 时。

配置要求

- Intel® Arc A-Series GPU 或更新的设备
- 显示器直接连接到执行渲染的 GPU
- Windows 10/11 x64 - 10.0.19043/22000 或更高版本
- DirectX12
- 使用 DXGI flip 模式

当使用 XeLL 时，任何其他形式的延迟减少或等待技术（如帧率限制器）应禁用。如果不这样，则 XeLL 收集的定时信息可能不准确，并可能导致不确定的行为。

将帧率限制传递给 XeLL，让它可以启用在其它的帧率限制器情况下不可用的某些优化。如果应用程序实现了自己的帧率限制器，则应在调用 xellSleep 之前进行等待操作。强烈不建议同时激活应用程序的帧率限制器和 XeLL。

关于工作调度，应用程序不应实现任何其它的提交逻辑，比如等待上一帧完成。启用 XeLL 时，工作提交将由 xellSleep 控制，以确保每帧在正确的时间开始提交，从而以最优方式利用 GPU 并通过减少延迟提供最佳用户体验。

启用 XeLL 时，应用程序应允许打开或者关闭 VSync，但在 DXGI Present 调用中仅允许 Sync Interval 设置为 1。

集成

建议应用程序每帧调用 XeLL API，即使所有功能当前都被禁用。XeLL 确保在穿透（pass-through）模式下，系统中执行的工作负载最少。所有功能应通过 xellSetSleepMode 来控制。此方法可以简化集成并减少设置更改时需要做的工作。

将来，集成 XeLL 可能通过 Arc 软件和驱动程序的配合为用户带来额外好处。换句话说，建议应用程序的行为在 XeLL 激活或未激活时保持相同。唯一的不同是 xellSetSleepMode 中提供的状态。

集成清单

集成 XeLL 需要满足以下关键要求：

- 始终调用 xellSleep 和 xellAddMarkerData
- 功能仅通过 xellSetSleepMode 控制
- 启用 VSync 时，Low Latency 激活时 Present 中的 SyncInterval 必须始终为 1
- 当 Low Latency 模式激活时，FPS 上限应该通过 xellSetSleepMode API 控制
- 应用程序行为在 XeLL 激活或未激活时应相同
- 应用程序必须确保在调用 xellSetSleepMode 之前完成所有 GPU 活动
- 应用程序必须确保在销毁 XeLL 对象之前销毁 XeSS-FG 对象

功能

Feature Name	*GPU Vendor*	*GPU Gen*
低延迟 Low Latency	Intel only	Intel Arc GPUs
帧率限制器 Frame limiter	Intel only	Intel Arc GPUs

低延迟

启用此功能后，XeLL 会延迟帧计算以改进应用程序中的响应速度。确切的行为取决于应用程序的帧计算时间、显示器规格和当前显示设置（撕裂或无撕裂）。启用 Low Latency 功能时应禁用任何其他形式的延迟减少技术。

在撕裂模式下，XeLL 将确保渲染队列几乎为空，这意味着每帧的第一次渲染任务提交的时候，之前的 GPU 工作已经基本完成，这样可以充分利用 GPU。由于帧在准备好后几乎立即 Present，因此没有渲染到屏幕的延迟，只有输入到渲染的延迟被降低了。

在无撕裂模式下，如果应用程序的性能等于或高于显示器的刷新率，XeLL 还将尝试将渲染结束的时间与显示器的 VSync 间隔对齐。这种方式结合确保渲染队列几乎为空，减少了输入到渲染和渲染到屏幕的延迟。在图 2 中，每帧都有最小的输入到渲染延迟，计算工作立即开始。帧渲染完成后，XeLL 保持一个小的安全带，以确保帧不会错过其 VSync 间隔，因为如果错过的话，Present 将不得不等待下一个，导致渲染到屏幕的延迟变大。

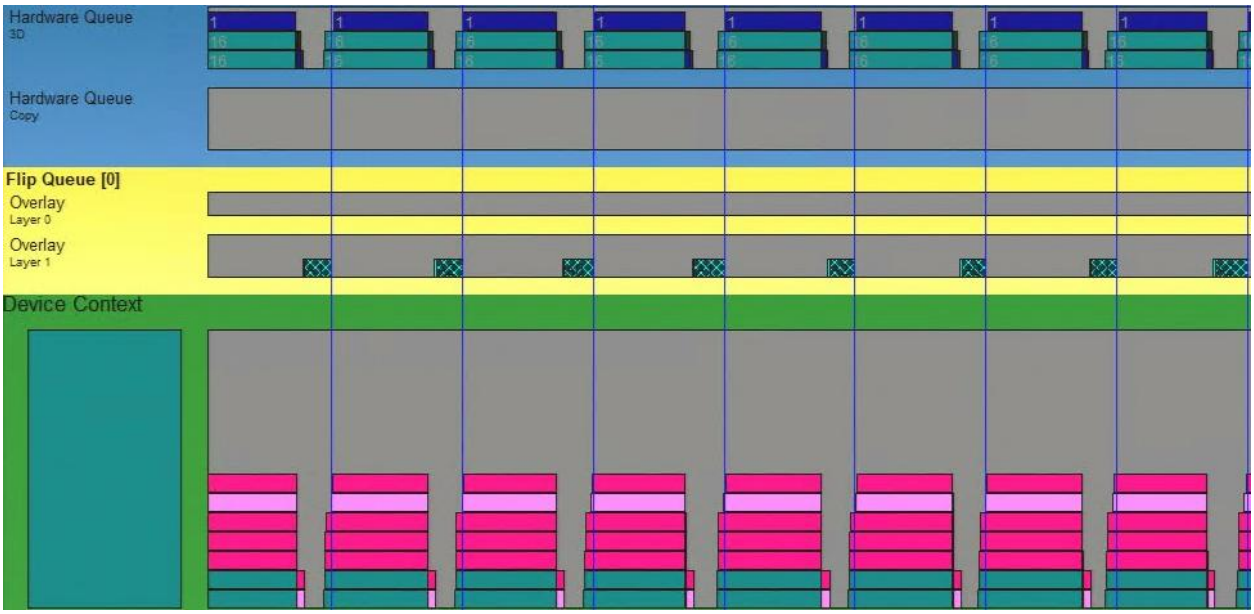

图 2 GpuView 中应用程序在 Vsync 和 Low Latency 都开启的情况下的行为

帧率限制器

帧率限制器可以在没有延迟减少技术的情况下使用。启用 XeLL 时，应用程序应确保禁用任何内部的帧率限制器。

同时启用延迟减少和帧率限制器，使 XeLL 能够做之前无撕裂模式下由于性能不足而无法实现的延迟优化。在这种情况下，如果应用程序性能等于或高于帧率限制，则渲染结束将与 VSync 间隔对齐，减少渲染到屏幕的延迟。帧率限制器不会影响撕裂模式，XeLL 将继续确保渲染队列为空以最小化输入到渲染的延迟。

在图 3 中，可以看到帧率限制设置为屏幕刷新率一半（例如 60 Hz 屏幕上的 30 FPS）的应用程序，在启用 VSync 时，每帧与每 2 个蓝色垂直线对齐，同时保持安全带。

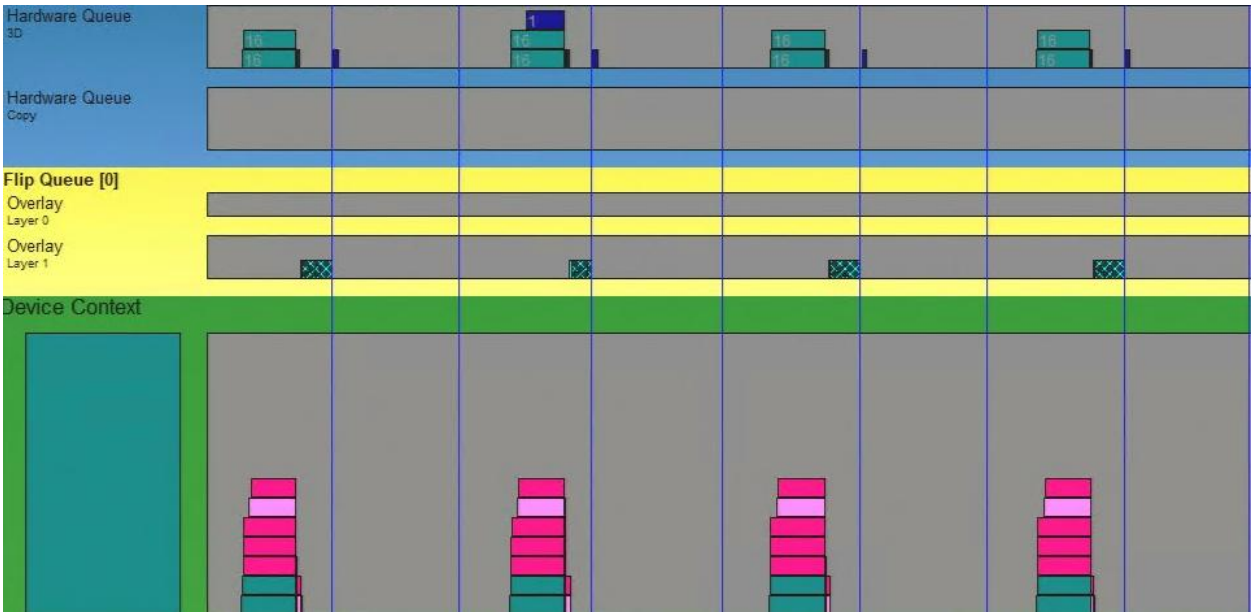

图3 GpuView 中应用程序运行在 VSync 和Low Latency 同时开启且帧率限制设为显示器刷新率的一半

XeSS-FG 支持

使用 XeSS Frame Generation （XeSS-FG）时需要开启 XeLL。如果检测到 XeSS-FG，XeLL 将负责增加的那个 Present 调用。帧生成影响无撕裂模式，因为一个帧将产生两个 Present，利用两个 VSync 间隔。因此，如果应用程序的性能等于或高于显示器刷新率的一半，则 XeLL 将限制应用程序以显示器刷新率一半的帧率运行并将渲染结束与 VSync 间隔对齐。目标是减少生成帧和渲染帧的渲染到屏幕延迟。

在图 4 中，启用 XeSS-FG 和 VSync 的应用程序以屏幕刷新率的一半渲染，但由于生成的帧，每个 VSync 间隔都被利用。这最小化了两次 Present 中输入到渲染和渲染到屏幕的延迟。

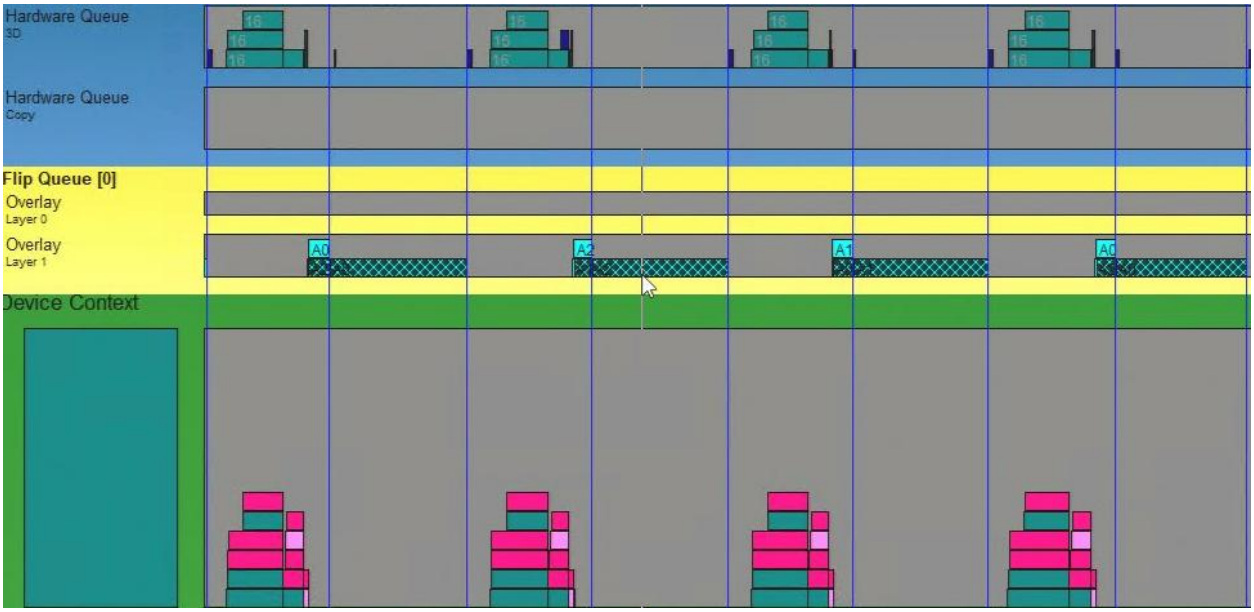

图4 GpuView 中应用程序运行在 VSync，Low Latency 和 XeSS-FG 同时开启的情况

使用帧率限制器时，如果应用程序性能等于或高于帧速目标的一半，XeLL 将限制应用程序帧率为目标的一半，确保呈现的总帧数（包括生成帧和渲染的帧）匹配帧率限制器的 FPS。例如，如果用户在120 Hz 屏幕上指定 60 FPS，则 XeLL 将目标设为 30 FPS（每帧调用两次 Present），结果为 60 FPS。由于屏幕为 120 Hz，那么每隔一个 VSync 显示一个新的帧。

部署

在项目中使用 XeLL：

- 将“inc”文件夹添加到包含路径中
- 包含头文件 xell.h 和 xell_d3d12.h
- 链接 lib/libxell.lib

以下文件必须放置在可执行文件旁边或 DLL 搜索路径中：

- Libxell.dll

命名规范和品牌指南

请参阅“XeSS 2 Naming Structure and Examples.pdf” 以获取批准的命名规范、品牌指南和设置菜单示例。

编程指南

初始化

通过创建上下文对象初始化 XeLL。对于 DirectX12，应使用 xellD3D12CreateContext 方法：

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

该函数将在设备不受支持时返回 XELL_RESULT_ERROR_UNSUPPORTED_DEVICE，如果驱动程序不受支持则返回 XELL_RESULT_ERROR_UNSUPPORTED_DRIVER。在这种情况下不会创建上下文。应用程序应验证上下文是否为空指针以及返回代码是否为 XELL_RESULT_SUCCESS。

日志回调

应用程序可以通过注册日志回调来接收 API 日志：

```
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);
```

日志记录有四个级别：

- XELL_LOGGING_LEVEL_DEBUG
- XELL_LOGGING_LEVEL_INFO
- XELL_LOGGING_LEVEL_WARNING
- XELL_LOGGING_LEVEL_ERROR

我们建议在生产中使用错误级别，在开发中使用警告或调试级别。

启用功能

在创建上下文之后，应用程序应使用 xellSetSleepMode 打开所需的功能。设置更改后只需调用一次此函数。更改设置时无需重新创建上下文。目前支持的功能包括延迟减少和帧率限制器，由 xell_sleep_params_t 结构中的 bLowLatencyMode 和 minimumIntervalUs 字段表示。

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

bLowLatencyMode - 设置为‘true’时启用延迟减少

minimumIntervalUs - 使用该参数通过传递时间（单位为微秒）来强制设置所需的帧率限制，例如设置为 30 FPS 上限时，该参数应设置为 33333

睡眠

在每帧开始时，在采样用户输入之前，应用程序应调用 xellSleep，它根据先前的统计数据和标记计算并执行当前帧的延迟。应用程序通过分配递增的 frame_id 值唯一标识每帧。如果启用了低延迟，xellSleep 必须是首次调用的 XeLL API 函数，并带有新分配的 frame_id 值。

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

标记

应用程序通过标记向 XeLL 提供定时信息。当给定标记传递给 XeLL 时，XeLL 可以知道应用程序处在帧开始或完成的阶段。为了 XeLL 正确计算和执行 xellSleep API 调用，每帧必须传递所有枚举的标记给 XeLL。

可用标记由 xell_latency_marker_type_t 枚举定义。标记的传递通过 xellAddMarkerData API 函数完成。

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

标记定义了帧的以下阶段，预期每个 END 标记在相应的 START 标记之后出现。

模拟

XELL_SIMULATION_START 和 XELL_SIMULATION_END 标记限定了应用程序执行 CPU 计算和更新以准备当前帧的 GPU 工作的时间段。在此期间应用程序会采样输入设备。

调用 sleep 之后，XELL_SIMULATION_START 应是每帧的第一个标记。

渲染提交

XELL_RENDERSUBMIT_START 和 XELL_RENDERSUBMIT_END 标记限定了应用程序填充命令列表并提交执行的时间段。

Present

XELL_PRESENT_START/XELL_PRESENT_END 标记限定了对 DXGI Present 的调用。

关闭

为了干净地关闭，应用程序必须在确保所有 GPU 活动停止后调用 xellDestroyContext。

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

如果使用了 XeSS-FG，应用程序必须确保在销毁 XeLL 对象之前销毁 XeSS-FG 对象。

版本控制

XeLL 库使用了 <major>.<minor>.<patch> 版本格式和 Numeric 90+ 方案用于开发阶段版本。版本号由 64 位大小的结构体（xell_version_t）指定，其中：

- major 增量表示新的 API，并可能导致功能中断
- minor 增量表示增量更改，例如可选输入或标志。这不会改变现有功能
- patch 增量可能包括性能或质量调整，或已知问题的修复，无接口变化。超过 90 的版本用于开发版本，以更改下一版本的接口。

版本信息嵌入到了 XeLL SDK 发布中，可以通过 xellGetVersion 函数访问。

声明

您不得使用或协助使用本文档连接到任何侵犯或其他法律分析有关的 Intel 产品。您同意授予 Intel 非排他性、免版税的许可，以使用本文所披露的主题撰写的任何专利声明。

性能因使用、配置和其他因素而异。了解更多信息，请访问 www.Intel.com/PerformanceIndex。

没有产品或组件是绝对安全的。

所有产品计划和路线图如有更改，恕不另行通知。

您的成本和结果可能会有所不同。

Intel 技术可能需要启用硬件、软件或服务激活。

Intel 技术的功能和好处取决于系统配置，可能需要启用硬件、软件或服务激活。性能因系统配置而异。请与您的系统制造商或零售商联系，了解更多信息，请访问 intel.com。

本文档未授予任何知识产权权利（明示或暗示，通过禁止反言或其他方式）。

Intel 不承担任何明示或暗示的保证，包括但不限于适销性、适合特定用途和不侵权的暗示保证，以及任何因履行、交易过程或贸易使用而产生的保证。

本文档包含有关开发中产品、服务和/或流程的信息。本文提供的所有信息如有更改，恕不另行通知。请联系您的 Intel 代表，获取最新的预测、时间表、规格和路线图。

描述的产品和服务可能包含被称为“已知问题”的缺陷或错误，可能导致与发布的规格有所偏离。当前已知问题的详细信息可供请求。

可通过致电 1-800-548-4725 或访问 www.intel.com/design/literature.htm 获取本文档中引用的具有订单编号的文件副本。

Microsoft、Windows 和 Windows 徽标是 Microsoft Corporation 在美国及/或其他国家的商标或注册商标。