Intel® XeSS Super Resolution (XeSS-SR)

开发者指南 2.0

Intel® XeSS Super Resolution (XeSS-SR) 提供了创新的、提升帧率的技术，支持 Intel® Arc™ 显卡和其他 GPU 供应商的显卡。XeSS-SR 通过 AI 深度学习进行图像放大，它可以在不影响图像质量的情况下提供更高的帧率。

目录

简介

Intel® XeSS 超级分辨率（XeSS-SR）是一种基于 AI 的时域超级采样和抗锯齿技术，由一系列计算着色器批次实现，在后处理阶段之前执行（详见“TAA 和 XeSS-SR”部分）。它通常接受未经反走样且带抖动的低分辨率颜色、游戏中的运动矢量以及深度作为输入，生成目标分辨率的上采样抗锯齿颜色缓冲区。

从 XeSS SDK 1.2 及更早版本的更新记录

XeSS SDK 1.3 版本增加了新的超级性能(3.0 倍放大)，超高质量+(1.3 倍)，原生抗锯齿 (1.0 倍) 等质量预设，同时增加了已有质量预设的放大倍数：

预设	以前版本 XeSS-SR 的分辨率放大倍数	XeSS-SR 1.3 版本的分辨率放大倍数
原生抗锯齿	N/A	1.0x（原生分辨率）
超高质量+	N/A	1.3x
超高质量	1.3x	1.5x
质量	1.5x	1.7x
平衡	1.7x	2.0x
性能	2.0x	2.3x
超级性能	N/A	3.0x
关闭	关闭 Intel XeSS-SR	关闭 Intel XeSS-SR

从之前的 XeSS-SR 版本更新到 XeSS-SR 1.3 及以后版本时，务必验证 XeSS-SR 在游戏中的集成满足以下几点：

- 分辨率放大倍数、额外增加的 mip bias 和抖动序列长度不应按质量预设硬编码在游戏代码中，而应该在运行时调用相应的函数和公式来获得。
- 应使用 xessGetOptimalInputResolution 函数查询输入分辨率。
- 应使用公式$\log_2(\frac{Input\ Width}{Target\ Width})$来计算额外增加的纹理 mip bias。例如：XeSS 1.3 的性能模式应该使用-1.202 的 mip bias，超级性能模式应该使用-1.585 的 mip bias。
- 抖动序列的长度必须至少是 $8*\left(\frac{Target\ Width}{Input\ Width}\right)^2$。例如：XeSS 1.3 的超级性能模式，抖动序列长度应该至少是 72。

请参考本文档的相关章节获得更详细的信息。

XeSS-SR 组件

XeSS-SR 通过 XeSS-SR SDK 提供接口以集成到游戏引擎中，包括以下组件：

- 一个基于 HLSL 的跨供应商实现，适用于支持 SM 6.4 的任何 GPU。推荐使用支持 DP4a 硬件加速的设备。
- 一个 Intel 实现，优化 Intel® Arc™ Graphics 和 Intel® Iris® Xe Graphics 上的运行。
- 一个实现调度器，它会选择性地加载游戏附带的 XeSS-SR 运行时、Intel® 显卡驱动程序提供的版本或跨供应商实现。

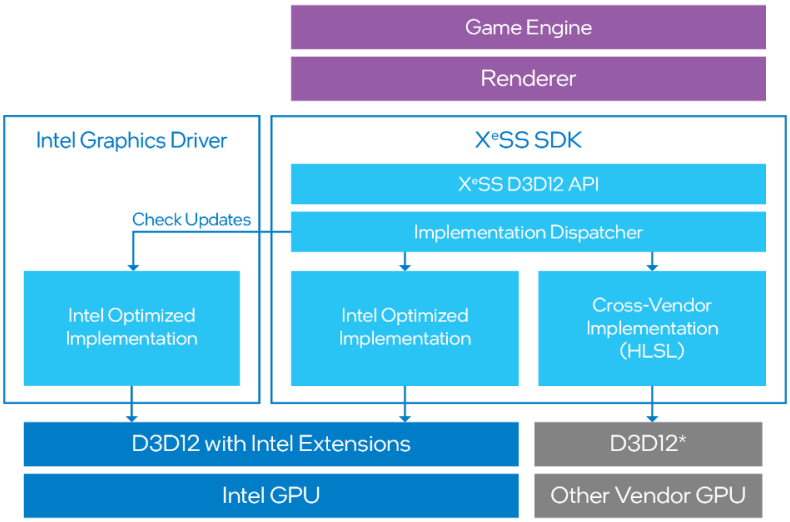

图 1. XeSS-SR SDK D3D12 组件，包含针对 Intel 和跨供应商的实现

兼容性

所有未来的 Intel® 显卡驱动程序版本都将兼容之前的 XeSS-SR 版本。

在初始化期间，XeSS-SR 将检查驱动程序兼容性并选择使用应用程序分发的版本或驱动程序捆绑的版本。

配置要求

- Windows 10/11 x64 - 10.0.19043/22000 或更高版本
- DirectX12
    - Intel® Iris® Xe GPU 或更新的设备

    - 其他支持 SM 6.4 并具有 DP4a 硬件加速的 GPU
- DirectX11
    - Intel® Arc™ Graphics 或更新的设备
- Vulkan 1.1（使用 SDK 函数查询所需的实例和设备扩展以及设备功能）
    - Intel® Iris® Xe GPU 或更新的设备
    - 其他支持 shaderIntegerDotProduct, shaderStorageReadWithoutFormat 和 mutableDescriptorType 功能的 GPU

TAA 和 XeSS-SR

XeSS-SR 是一种时域分摊的超级采样/上采样技术，它取代了游戏渲染器中的时域抗锯齿（TAA）阶段，图像质量显著优于当前游戏中的先进技术。

图 2 显示了带有 TAA 的渲染器。渲染器在每一帧中抖动摄像机，以在屏幕空间采样不同的坐标。TAA 阶段通过在时域上累积这些样本以生成超采样图像。之前累积的帧（历史帧）使用渲染器生成的运动向量进行形变，以在累积之前将其与当前帧对齐。然而，由于帧与帧之间的可见性和着色变化或运动向量的误差，变形后的样本历史可能会与当前像素不匹配。这通常会导致伪影。TAA 实现使用诸如邻域钳制（Neighborhood Clamping）等启发式方法来检测不匹配并拒绝历史数据。然而，这些启发式方法常常失败，产生明显的重影、过度模糊或闪烁现象。

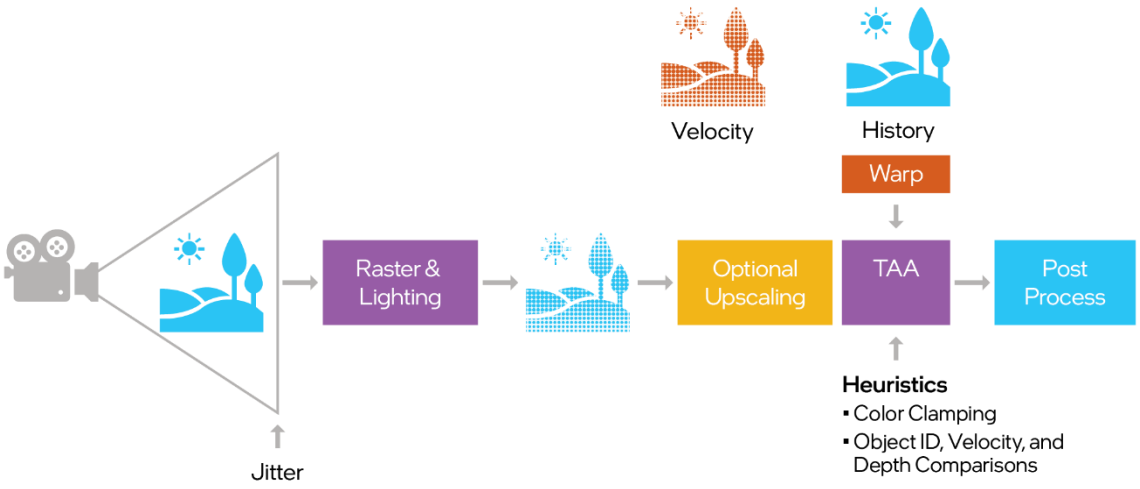

图 2. 带有 TAA 的典型渲染管线流程图

XeSS-SR 取代了 TAA 阶段，采用基于神经网络的方法，如下图所示，XeSS-SR 具有与 TAA 相同的输入和输出。请参阅此报告以了解 TAA 技术的概述。

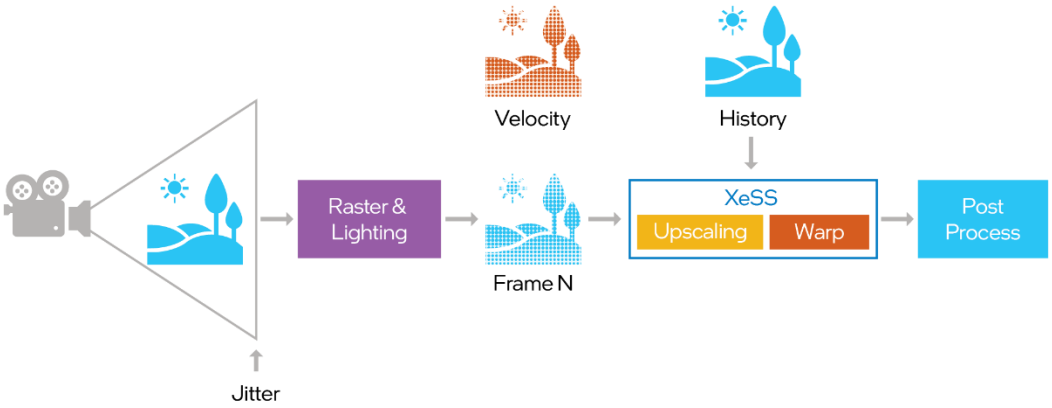

图 3. 包含 XeSS-SR 的渲染管线

游戏设置推荐

在您的游戏中集成 XeSS-SR 时，请确保遵循以下指南，以便用户在修改 XeSS-SR 选项时有一致的体验。

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。

XeSS-SR 相关的官方字体是 IntelOneText-Regular。

标签	Intel XeSS-SR
简短描述	Intel® XeSS 超级分辨率（XeSS-SR）技术使用 AI 提供更高的性能和卓越的图像质量。XeSS 针对具有 AI 加速硬件的 Intel® Arc™ GPU 进行了优化。
最简描述	Intel® XeSS 超级分辨率（XeSS-SR）技术使用 AI 提供更高的性能和卓越的图像质量。

游戏图形设置菜单/游戏安装程序/启动器设置

游戏图形设置应清晰显示 XeSS-SR 选项名称，并允许用户选择不同的质量/性能级别，如下所示：

预设	描述	推荐分辨率
原生抗锯齿	基于 AI 的抗锯齿，最大限度提高视觉质量	1080p 及以上
超高质量+	最高质量视觉放大	1080p 及以上
超高质量	更高质量视觉放大	1080p 及以上
质量	高质量视觉放大	1080p 及以上
平衡	性能与视觉质量之间的最佳平衡	1080p 及以上
性能	高性能提升	1440p 及以上
超级性能	最高性能提升	1440p 及以上
关闭	关闭 Intel XeSS-SR	N/A

注意：要启用 XeSS-SR，您的游戏需要禁用其他放大技术，例如 NVIDIA DLSS* 和 AMD FSR*，以及时域抗锯齿（TAA）技术，以减少任何不兼容问题的可能性。

图形预设默认推荐

游戏菜单中选择的 XeSS-SR 预设应基于用户设置的目标分辨率。以下是推荐的默认设置：

默认 XeSS-SR 推荐	描述	推荐设置
分辨率特定	您的游戏根据输出分辨率调整 XeSS-SR 默认预设	1080p 及以下设置为“平衡”，1440p 及以上设置为“性能”
通用	您的游戏选择一个 XeSS-SR 预设作为默认设置。	Intel XeSS-SR 开启，设置为“性能”

注意：所有支持的 Intel XeSS-SR 设置应通过选项菜单暴露给用户，以鼓励用户自主选择。

部署

要在项目中使用 XeSS-SR：

- 将“inc”文件夹添加到包含路径中
- 包含 xess.h 和一个 API 特定头文件：xess_d3d12.h、xess_d3d11.h 或 xess_vk.h

- 对于 D3D12 和 Vulkan：链接 lib/libxess.lib
- 对于 D3D11：链接 lib/libxess_dx11.lib

以下文件必须放置在可执行文件旁边或 DLL 搜索路径中：

- 对于 D3D12 和 Vulkan：libxess.dll
- 对于 D3D11：libxess_dx11.dll
- Microsoft Visual C++ 可再发行版 14.40.33810 或更高版本。需要以下库：
  - msvcp140.dll
  - vcruntime140.dll
  - vcruntime140_1.dll

请注意， 部署 D3D11 后端所需要的库不同于 D3D12 和 Vulkan，必须小心链接和调用正确库函数。特别是 xess.h 头文件中的公用函数，如 xessDestroyContext，这两个库都有暴露，但 libxess.dll 库中的版本不支持被 D3D11 的 XeSS-SR 上下文调用。同样，libxess_dx11.dll 库中的这个函数也不支持被 D3D12 或 Vulkan 的 XeSS-SR 上下文调用。

编程指南

线程安全

XeSS-SR 不是线程安全的。客户端应用程序必须确保对 XeSS-SR 函数的调用是线程安全的。一般来说，所有对 XeSS-SR API 的调用都必须在 XeSS-SR 初始化的那个线程中进行。

输入和输出

XeSS-SR 每帧需要一组最小的输入：

- 抖动(Jitter)
- 输入颜色(Input color)
- 扩展的高分辨率运动矢量(Dilated high-res motion vectors)

如果无法提供高分辨率运动矢量，渲染器可以提供输入分辨率的尺寸的运动矢量以及深度：

- 未扩展的低分辨率运动矢量(Undilated low-res motion vectors)
- 深度(Depth)

在后一种情况下，运动矢量将在 XeSS-SR 内部进行扩展和上采样。

XeSS-SR 生成单一输出到应用程序提供的纹理中。

抖动

作为一种时域超级采样技术，XeSS-SR 需要对投影矩阵在每一帧应用次像素抖动偏移（J_x, J_y）。这一过程中，每帧生成一个新的次像素样本位置，并保证即使在静态场景中也能实现时域收敛。抖动偏移值应在 [-0.5, 0.5] 范围内。我们可以通过在摄像机投影矩阵中添加剪切变换来应用抖动：

```
ProjectionMatrix.M[2][0] += Jx * 2.0f / InputWidth
ProjectionMatrix.M[2][1] -= Jy * 2.0f / InputHeight
```

摄像机的抖动产生帧中采样点的位移，如下图所示，其中目标图像在宽度和高度上放大了 2 倍。请注意，有效抖动相对于（J_x, J_y）是负的，因为投影矩阵应用于几何体，所以它对应于负的摄像机抖动。

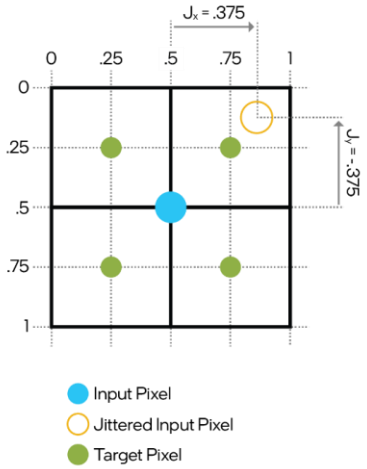

图 4. 采样点的抖动位移

抖动序列

为了获得最佳的 XeSS-SR 算法质量，需要一个具有良好空间分布特性的准随机采样序列（Halton 序列是一个不错的选择）。在使用这样的序列时，应考虑缩放因子以修改重复模式的长度。例如：若游戏在原生渲染中使用长度为 8 的 Halton 序列，则在 XeSS-SR 上采样时的长度应为 $8 * \left(\frac{Target\ Width}{Input\ Width}\right)^2$，这样可以确保在单个低分辨率像素覆盖的区域内样本的良好分布。有时候，进一步增加长度会带来额外的

质量提升，因此我们鼓励对序列的长度进行实验。此外，我们应该避免使用那些可能导致抖动样本分布偏移输入像素的采样技术。

颜色

XeSS-SR 接受任何线性颜色格式的 LDR 和 HDR 输入颜色，例如：

- R16G16B16A16_FLOAT
- R11G11B10_FLOAT
- R8G8B8A8_UNORM

只能使用 UNORM 整数格式作为颜色输入，不支持其他整数格式。

应用程序应通过设置输入标志 XESS_INIT_FLAG_LDR_INPUT_COLOR 来通知 XeSS 是否提供的是 LDR 输入颜色。

输入颜色应为 scRGB 颜色空间，这是场景参照的(Scene-referred)，例如颜色值表示亮度级别。值（1.0, 1.0, 1.0）编码 D65 白色在 80 尼特并表示 SDR 显示器的最大亮度。对于 HDR 内容，颜色值可以超过（1.0, 1.0, 1.0）。

建议提供 HDR 输入颜色给 XeSS-SR，因为 XeSS-SR 可以内部应用调优的色调映射，以优化 XeSS-SR AI 模型的视觉质量。如果提供了 LDR 输入颜色，XeSS-SR 质量可能会下降。如果 HDR 输入颜色值尚未调整曝光，或者它们与 sRGB 空间的比例不同，则可以通过以下方式提供单独的比例值：

- 如果没有可用的曝光值，可以在初始化时使用 XESS_INIT_FLAG_ENABLE_AUTOEXPOSURE 标志让 XeSS-SR 计算曝光值。注意，使用此标志会对性能产生可测量的影响。
- 可以通过在 xess_*_execute_params_t 结构中传递值，或提供可以由 GPU 更新的曝光比例纹理来提供输入曝光值。

如果应用程序向 XeSS-SR 提供 LDR 颜色，建议将曝光值设置为“1”并避免使用自动曝光。

如果应用程序的输入颜色是预曝光的，建议通过以下方式之一提供预曝光：

- 提供除以预曝光的曝光值
- 提供预曝光值的倒数作为 xessSetExposureMultiplier 的输入

曝光倍增值将应用于所有曝光值，包括由内置自动曝光功能生成的值。

曝光值应用于输入颜色的方式如下：

```
if (useAutoexposure)
{
    scale = XeSSCalculatedExposure(…)
}
else if (useExposureScaleTexture)
{
    scale = exposureScaleTexture.Load(int3(0, 0, 0)).x
}
else
{
    scale = inputScale
}

inputColor *= scale * exposure_multiplier
```

如果对输入应用了比例值，如上所示，则此比例的反比将应用于输出颜色。

XeSS-SR 维护一个内部历史状态以执行传入样本的时间累积。这意味着如果场景或视图突然变化，历史记录应被丢弃。这通过在 xess_xxx_execute_params_t 中设置 historyReset 标志来实现。

输出必须与输入在同一颜色空间中。它可以是与输入类似的任何线性颜色格式。

运动矢量

运动矢量指定从当前帧指向前一帧的屏幕空间运动（以像素为单位）。如果应用程序使用归一化设备坐标（NDC）进行运动矢量计算，则应在 XeSS-SR 上下文初始化期间传递一个额外的标志（XESS_INIT_FLAG_USE_NDC_VELOCITY）。XeSS-SR 接受 R16G16_FLOAT 格式的运动矢量，其中 R 通道编码 x 方向的运动，G 通道编码 y 方向的运动。这些运动矢量不包括因摄像机抖动引起的运动。运动矢量可以是低分辨率（默认）或高分辨率（XESS_INIT_FLAG_HIGH_RES_MV）。低分辨率运动矢量由输入分辨率的 2D 纹理表示，而高分辨率运动矢量由目标分辨率的 2D 纹理表示。

对于高分辨率运动矢量，由摄像机动画产生的速度分量在延迟渲染通道中使用摄像机变换和深度值在目标分辨率下计算。然而，与粒子和物体动画相关的速度分量通常在输入分辨率下计算并存储在 G-Buffer 中。这个速度分量被上采样并与摄像机速度组合以生成高分辨率运动矢量的纹理。XeSS-SR 需要扩展后的高分辨率运动矢量。例如，运动矢量表示输入像素的小邻域（如 3×3）的最前面表面的运动。用户可以在单独的渲染通道中实现高分辨率运动矢量的计算。

低分辨率运动矢量未被扩展，直接表示在每个抖动像素位置采样的速度。XeSS-SR 在内部将运动矢量上采样到目标网格，并使用深度纹理来扩展它们。图 5 显示了同一运动矢量在低分辨率和高分辨率下的表示。

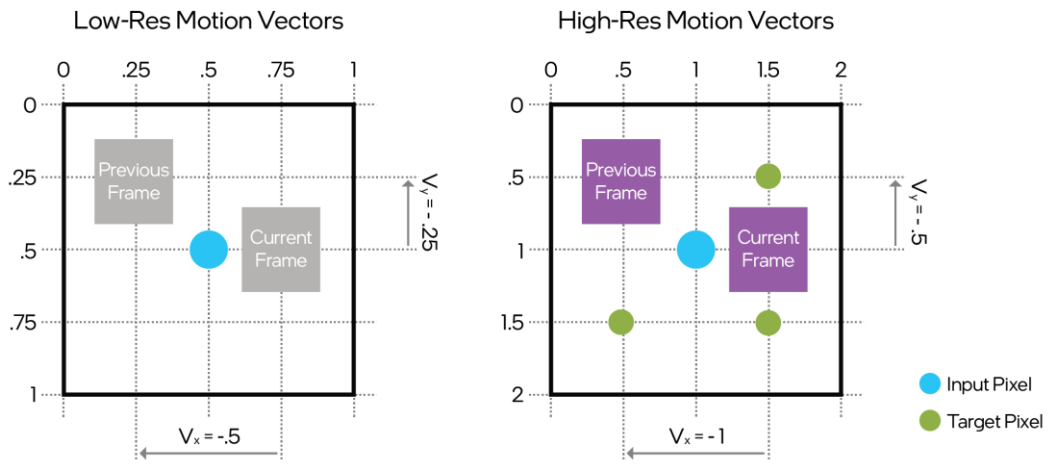

图 5. XeSS-SR 指定低分辨率和高分辨率运动矢量的约定

一些游戏引擎仅将对象渲染到 G-Buffer 中，并在 TAA 着色器中快速计算摄像机速度。在这种情况下，在执行 XeSS-SR 之前需要额外的一次处理，以合并对象和摄像机速度并生成一个展平的速度缓冲区。在这种情况下，高分辨率运动向量可能是更好的选择，因为展平处理可以在目标分辨率下执行。

深度

如果 XeSS-SR 与低分辨率运动向量一起使用，还需要一个用于速度扩展的深度纹理。任何深度格式，如 D32_FLOAT 或 D24_UNORM，都被支持。默认情况下，XeSS-SR 假定较小的深度值离摄像机更近。然而，一些游戏引擎使用反转深度，可以通过设置 XESS_INIT_FLAG_INVERTED_DEPTH 来启用这一特性。

响应式像素遮罩

尽管 XeSS-SR 是一种广泛适用的技术，应该能处理各种渲染场景，但在某些情况下，没有有效运动矢量的对象（例如粒子）可能会产生伪影。在这种情况下，可以提供响应像素遮罩纹理对这些对象进行遮罩。响应遮罩允许显式控制历史记录对被遮罩像素的影响程度。

遮罩应是范围在 [0.0, 1.0] 之间的浮点值。遮罩像素值 1.0 表示应完全忽略相应颜色像素的累积历史，直接使用当前帧。遮罩像素值 0.0 表示不修改 XeSS-SR AI 模型的决策，相当于没有提供响应遮罩的情况。默认情况下，XeSS-SR 在应用值时将内部响应遮罩值钳制在 0.8；这是一个合理的值，可以减少对象的遮罩被设置成 1.0 时常见的意外锯齿和各种闪烁。默认钳制值应仅在验证更高值提供更好结果后由应用程序覆盖。可以使用 xessSetMaxResponsiveMaskValue 函数更改钳制值。

应用程序可以以任何可以读取为浮点数的纹理格式提供响应遮罩纹理，响应遮罩值应保存在 R 通道中。响应像素遮罩应为输入分辨率且不应包含任何抖动。

应用程序应仅在明确提高视觉质量时尝试使用响应像素遮罩。

输出

XeSS-SR 对输出纹理的要求取决于颜色输入格式。输出必须与输入在同一格式和颜色空间中。它必须足够大以容纳输出分辨率大小的图像。

如果输出纹理大于输出分辨率——输出图像将放置在输出纹理的左上角。应用程序可以在调用 xess*Execute 时提供偏移量以更改输出图像位置。

XeSS 不保留输出纹理的 alpha 通道——它将被填充为值 1.0。

资源状态

XeSS-SR 期望所有输入纹理处于以下状态：

- D3D12
  - 输入纹理为 D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE。
  - 输出纹理为 D3D12_RESOURCE_STATE_UNORDERED_ACCESS。
- Vulkan
  - 输入纹理为 VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL
  - 输出纹理为 VK_IMAGE_LAYOUT_GENERAL

在 Vulkan 中，XeSS-SR 在 VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT 阶段访问资源，使用 VK_ACCESS_SHADER_READ_BIT 访问输入纹理，使用 VK_ACCESS_SHADER_READ_BIT 和 VK_ACCESS_SHADER_WRITE_BIT 访问输出纹理。

在 D3D12 和 Vulkan 中，XeSS-SR 不对输入和输出资源执行任何内存访问同步。应用程序须负责准备相应的资源布局。

资源格式

XeSS-SR 期望所有输入纹理为类型化（Typed）格式。

针对 D3D12 和 D3D11

XeSS-SR 可以接受一些无类型（Typeless）的输入纹理。在无类型格式的情况下，它们将根据以下表格在内部解释：

输入无类型格式	内部类型化解析
DXGI_FORMAT_R32G32B32A32_TYPELESS	DXGI_FORMAT_R32G32B32A32_FLOAT
DXGI_FORMAT_R32G32B32_TYPELESS	DXGI_FORMAT_R32G32B32_FLOAT
DXGI_FORMAT_R16G16B16A16_TYPELESS	DXGI_FORMAT_R16G16B16A16_FLOAT
DXGI_FORMAT_R32G32_TYPELESS	DXGI_FORMAT_R32G32_FLOAT
DXGI_FORMAT_R32G8X24_TYPELESS	DXGI_FORMAT_R32_FLOAT_X8X24_TYPELESS
DXGI_FORMAT_R10G10B10A2_TYPELESS	DXGI_FORMAT_R10G10B10A2_UNORM
DXGI_FORMAT_R8G8B8A8_TYPELESS	DXGI_FORMAT_R8G8B8A8_UNORM
DXGI_FORMAT_R16G16_TYPELESS	DXGI_FORMAT_R16G16_FLOAT
DXGI_FORMAT_R32_TYPELESS	DXGI_FORMAT_R32_FLOAT
DXGI_FORMAT_R24G8_TYPELESS	DXGI_FORMAT_R24_UNORM_X8_TYPELESS
DXGI_FORMAT_R8G8_TYPELESS	DXGI_FORMAT_R8G8_UNORM
DXGI_FORMAT_R16_TYPELESS	DXGI_FORMAT_R16_FLOAT
DXGI_FORMAT_R8_TYPELESS	DXGI_FORMAT_R8_UNORM
DXGI_FORMAT_B8G8R8A8_TYPELESS	DXGI_FORMAT_B8G8R8A8_UNORM
DXGI_FORMAT_B8G8R8X8_TYPELESS	DXGI_FORMAT_B8G8R8X8_UNORM
DXGI_FORMAT_D16_UNORM	DXGI_FORMAT_R16_UNORM
DXGI_FORMAT_D32_FLOAT	DXGI_FORMAT_R32_FLOAT
DXGI_FORMAT_D24_UNORM_S8_UINT	DXGI_FORMAT_R24_UNORM_X8_TYPELESS
DXGI_FORMAT_D32_FLOAT_S8X24_UINT	DXGI_FORMAT_R32_FLOAT_X8X24_TYPELESS

Mip Bias

为了在目标分辨率下保留纹理细节，XeSS-SR 需要额外的 mip bias。建议使用以下公式：

$$\log_2(\frac{Input\ Width}{Target\ Width})$$

例如，对于 2.0 倍分辨率缩放（对应于平衡质量预设），需应用 -1 的 mip bias。在某些情况下，进一步增加 mip bias 可以带来额外的视觉质量提升；然而，这可能会带来潜在的性能开销，因为增加的内存带

宽需求和可能导致的时域稳定性降低（闪烁和摩尔纹）。建议实验不同的纹理 LOD 偏移以找到最佳值，但 mip bias 应接近推荐值，并始终根据输入和输出分辨率计算。

在动态分辨率场景中，每次 XeSS-SR 输入分辨率变化时，建议更新 mip bias。如果每次输入分辨率变化时都无法更改 mip bias，请考虑保留一组估计的 mip bias。

初始化

D3D12 相关

首先创建一个 XeSS-SR 上下文，如下所示。在 Intel GPU 上，这一步加载了 XeSS-SR 的最新 Intel 优化实现。返回的上下文句柄可以用于初始化和执行。

```
xess_context_handle_t context;
xessD3D12CreateContext(pD3D12Device, &context)
```

在初始化 XeSS-SR 之前，用户可以请求管线预编译过程，以避免在初始化期间进行代价高昂的内核编译和管线创建。

```
xessD3D12BuildPipelines(context, NULL, false, initFlags);
```

应用程序可以通过调用 xessGetPipelineBuildStatus 检查管线构建状态。

然后调用 xessD3D12Init 函数初始化 XeSS-SR。在初始化期间，XeSS-SR 可能创建临时缓冲区和复制队列以上传权重数据。这些将在初始化结束时被销毁。XeSS-SR 的存储和层特化由目标分辨率确定。因此，在初始化期间必须设置目标宽度和高度。

```
xess_d3d12_init_params_t initParams;
initParams.outputWidth = 3840;
initParams.outputHeight = 2160;
initParams.initFlags = XESS_INIT_FLAG_HIGH_RES_MV;
initParams.pTempStorageHeap = NULL;

xessD3D12Init(&context, &initParams);
```

XeSS-SR 使用三种类型的存储：

- **持久的输出无关存储**：持久存储如网络权重在初始化期间由 XeSS-SR 内部分配和上传。
- **持久的输出相关存储**：持久存储如内部历史纹理。
- **临时存储**：临时存储如网络激活，只在 XeSS-SR 执行期间有有效数据。

持久存储始终由 XeSS-SR 拥有。应用程序可以通过 xess_d3d12_init_params_t 结构的 pTempTextureHeap 和 pTempBufferHeap 字段选择提供临时存储的资源堆。分配的资源堆必须使用默认堆类型 (D3D12_HEAP_TYPE_DEFAULT)。为了确保最佳性能，该堆应具有高内存驻留优先级。这些堆在 XeSS-SR 执行后可以复用。可以通过调用 xessGetProperties 获取这些堆的所需大小。

```
// Get required heap sizes
xess2d_t output_res {width, height};
xess_properties_t props;
xessGetProperties(context, &output_res, &props);

// Create buffer heap
ComPtr<ID3D12Heap> pBufferHeap;
CD3DX12_HEAP_DESC bufferHeapDesc(props.tempBufferHeapSize,
  D3D12_HEAP_TYPE_DEFAULT);
d3dDevice->CreateHeap(&bufferHeapDesc , IID_PPV_ARGS(&pBufferHeap));

// Create texture heap
ComPtr<ID3D12Heap> pTextureHeap;
CD3DX12_HEAP_DESC textureHeapDesc(props.tempTextureHeapSize,
  D3D12_HEAP_TYPE_DEFAULT);
d3dDevice->CreateHeap(&textureHeapDesc, IID_PPV_ARGS(&pTextureHeap));

// Pass heaps to XeSS
initParams.pTempBufferHeap = pBufferHeap.Get();
initParams.bufferHeapOffset = 0;
initParams.pTempTextureHeap= pTextureHeap.Get();
initParams.textureHeapOffset = 0;

xessD3D12Init(&context, &initParams);
```

对于 D3D12，应用程序可以选择提供外部描述符堆，XeSS-SR 在其中创建所有内部资源描述符。您必须通过调用 xessGetProperties 获取所需描述符的数量。然后在初始化时指定 XESS_INIT_FLAG_EXTERNAL_DESCRIPTOR_HEAP 标志，并在执行期间传递描述符堆和偏移量。描述符堆必须是 D3D12_SRV_UAV_CBV_DESCRIPTOR_HEAP 类型，并且可见于着色器。 如果目标分辨率或任何其他初始化参数发生变化，可以重新初始化 XeSS-SR。然而，必须在重新初始化之前完成所有待执行的 XeSS-SR 命令列表。当分配了临时的 XeSS-SR 存储时，您需要释放或重新分配堆。质量预设的更改开销较小，但任何其他参数的更改可能导致较长的 xessD3D12Init 执行时间。

D3D11 相关

目前，在 D3D11 上的 XeSS-SR 仅支持 Intel® Arc™ Graphics 或更高版本。仅提供针对 Intel 优化的 XeSS-SR 实现。

在不支持的设备上创建上下文，尤其是非 Intel 设备或 Intel® Iris® Xe，可能会失败，并返回 XESS_RESULT_ERROR_UNSUPPORTED_DEVICE 错误。

要在 D3D11 上使用 XeSS-SR，必须按照如下所示创建上下文：

```
xess_context_handle_t context;
xessD3D11CreateContext(pD3D11Device, &context);
```

然后调用 xessD3D11Init 函数初始化 XeSS-SR。在初始化期间，XeSS-SR 可以创建临时缓冲区和复制队列以上传权重。这些将在初始化结束时被销毁。XeSS-SR 的存储和层特化由目标分辨率确定。因此，在初始化期间必须设置目标宽度和高度。

```
xess_d3d11_init_params_t initParams;
initParams.outputWidth = 3840;
initParams.outputHeight = 2160;
initParams.initFlags = XESS_INIT_FLAG_HIGH_RES_MV;
xessD3D11Init (&context, &initParams);
```

如果目标分辨率或初始化标志发生变化，可以重新初始化 XeSS-SR。然而，必须在重新初始化之前完成所有待执行的 XeSS-SR 命令。质量预设的更改开销较小，但任何其他参数的更改可能导致较长的 xessD3D11Init 执行时间。

与 D3D12 不同，在 D3D11 中，XeSS-SR 执行所需的数据的临时存储由库管理，初始化期间无法由用户提供临时存储堆。

另一个与 D3D12 路径的区别是 D3D11 路径不支持 XESS_INIT_FLAG_EXTERNAL_DESCRIPTOR_HEAP 标志。

此外，D3D11 中不提供管线构建函数，例如 xessD3D*BuildPipelines 或 xessGetPipelineBuildStatus。

Vulkan 相关

要使用 XeSS-SR，必须支持 Vulkan 1.1。此外，XeSS-SR 还需要 Vulkan 扩展和功能，应按照如下所示进行查询。对于基础操作，XeSS-SR 目前需要：

- shaderStorageReadWithoutFormat feature.
- mutableDescriptorType feature from VK_EXT_mutable_descriptor_type extension.
- shaderIntegerDotProduct feature.

然而，在运行时，XeSS-SR 可能需要其他功能和扩展以提高性能，例如 shaderInt8。因此，XeSS-SR 提供了一组函数来查询所需的扩展和功能。

首先，在创建 VkInstance 对象之前，用户应查询 XeSS-SR 以获取所需的实例扩展和 Vulkan API 版本，如下所示：

```
uint32_t instance_extension_count;
const char* const* instance_extensions;
uint32_t api_version;
xessVKGetRequiredInstanceExtensions(&instance_extension_count,
  &instance_extensions, &api_version);
```

如果返回的扩展数量不为 0，则必须在传递给 xessVKCreateContext 的 VkInstance 对象中启用返回的扩展。在创建 VkInstance 时，必须将 VkApplicationInfo.apiVersion 字段设置为等于或大于 xessVKGetRequiredInstanceExtensions 返回的 api_version。

保证在给定平台上支持返回的扩展和 API 版本。如果不支持所需扩展，函数将失败并返回 XESS_RESULT_ERROR_UNSUPPORTED_DRIVER 错误。

返回的扩展列表的内存由 XeSS-SR 拥有，应用程序不应释放它。

在创建 VkInstance 对象后，在创建 VkDevice 对象之前，用户应查询 XeSS-SR 以获取所需的设备扩展和设备功能，如下所示：

```
uint32_t device_extension_count;
const char* const* device_extensions;
xessVKGetRequiredDeviceExtensions(instance, physical_device,
 &device_extension_count, &device_extensions);


VkPhysicalDeviceFeatures2 physical_device_features2 = {...};
void* features = &physical_device_features2; // or NULL
xessVKGetRequiredDeviceFeatures(instance, physical_device, &features);
```

如果返回的扩展数量不为 0，则必须在传递给 xessVKCreateContext 函数的 VkDevice 对象中启用返回的扩展。

xessVKGetRequiredDeviceFeatures 通过将它们添加到给定的 VkPhysicalDevice...Features 结构链中返回所需的功能。链应通过 pNext 字段连接结构形成。xessVKGetRequiredDeviceFeatures 函数通过向现有结构写入字段值并在必要时链接新结构返回所需的功能。

或者，可以将 features 设置为 NULL 调用此函数，在这种情况下，它将返回新的功能结构链，应用程序应将其合并到传递给 vkCreateDevice 的 VkDeviceCreateInfo 结构链中。用户必须注意可能的重复结构类型，如果 XeSS-SR 和应用程序都希望链接相同的结构类型。

传递非空 pEnabledFeatures 字段的 VkDeviceCreateInfo 结构是错误的，因为此字段是 const，不能由 XeSS-SR 修改。应该通过 pNext 字段链接 VkPhysicalDeviceFeatures2 结构，而不是使用 pEnabledFeatures 字段。

必须将返回的功能链接到传递给 vkCreateDevice 调用的 VkDeviceCreateInfo 结构，以创建传递给 xessVKCreateContext 函数的 VkDevice 对象。

保证返回的扩展和功能在给定的设备上支持。如果不支持所需扩展或功能，函数将失败并返回 XESS_RESULT_ERROR_UNSUPPORTED_DRIVER 错误。

返回的扩展列表和添加到链中的功能结构的内存由 XeSS-SR 拥有，应用程序不应释放它。

要使用 XeSS-SR，必须按照如下所示创建一个上下文：

```
xess_context_handle_t context;
xessVKCreateContext (vulkan_instance, physical_device, device,
  enabled_features2, enabled_extensions_count, enabled_extensions,
  &context)
```

在初始化 XeSS-SR 之前，用户可以请求管线预编译过程，以避免在初始化期间进行代价高昂的内核编译和管道创建。

```
xessVKBuildPipelines(context, pipeline_cache, false, initFlags);
```

应用程序可以通过调用 xessGetPipelineBuildStatus 检查管道构建状态。

然后调用 xessVKInit 函数初始化 XeSS-SR。在初始化期间，XeSS-SR 可以创建临时缓冲区和复制队列以上传权重。这些将在初始化结束时被销毁。XeSS-SR 的存储和层特化由目标分辨率确定。因此，在初始化期间必须设置目标宽度和高度。

```
xess_d3d12_init_params_t initParams;
initParams.outputWidth = 3840;
initParams.outputHeight = 2160;
initParams.initFlags = XESS_INIT_FLAG_HIGH_RES_MV;
initParams.tempBufferHeap = VK_NULL_HANDLE;
initParams.tempTextureHeap = VK_NULL_HANDLE;
xessVKInit (&context, &initParams);
```

XeSS-SR 使用三种类型的存储：

- **持久的输出无关存储**：持久存储如网络权重在初始化期间由 XeSS-SR 内部分配和上传。

- **持久的输出相关存储**：持久存储如内部历史纹理。
- **临时存储**：临时存储如网络激活，只在 XeSS-SR 执行期间有有效数据。

持久存储始终由 XeSS-SR 拥有。应用程序可以通过 xess_d3d12_init_params_t 结构的 pTempTextureHeap 和 pTempBufferHeap 字段选择提供临时存储的资源堆。分配的资源堆应从设备本地内存中分配。如果应用程序分配这些存储——它们可以在 XeSS-SR 执行后重新使用。可以通过调用 xessGetProperties 获取这些堆的所需大小。如果目标分辨率或任何其他初始化参数发生变化，可以重新初始化 XeSS-SR。然而，必须在重新初始化之前完成所有挂起的 XeSS-SR 命令列表。当分配了临时的 XeSS-SR 存储时，应用程序有责任释放或重新分配堆。质量预设的更改是便宜的，但任何其他参数的更改可能导致较长的 xessVKInit 执行时间。

与 D3D12 路径的一个区别是 Vulkan 路径不支持 XESS_INIT_FLAG_EXTERNAL_DESCRIPTOR_HEAP 标志。

日志回调

应用程序可以通过注册日志回调（xessSetLoggingCallback ）获取额外的诊断信息。回调的要求：

- 回调可以从不同的线程调用。
- 回调可以从多个线程同时调用。
- 消息指针仅在函数内部有效，并可能在返回调用后立即无效。
- 消息是一个以 null 结尾的 UTF-8 字符串。

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。

执行

在 D3D12 和 Vulkan 中，XeSS-SR 执行函数不涉及任何 GPU 工作负载，而是将 XeSS-SR 命令记录到指定的命令列表中。然后由用户将命令列表排队执行。这意味着应用程序有责任确保所有输入/输出资源在实际 GPU 执行过程中保持有效。

D3D12 相关

默认情况下，XeSS-SR 创建一个内部描述符堆，但如果在初始化阶段指定了 XESS_INIT_FLAG_EXTERNAL_DESCRIPTOR_HEAP，您可以在执行参数中传递外部描述符堆及其偏移量。此标志仅支持 D3D12 路径。

如果在 xessD3D12Init 参数中指定了 EXTERNAL_DESCRIPTOR_HEAP 标志，必须在与内部描述符相同的描述符堆中的连续位置创建输入和输出缓冲区的描述符。通过 xess_d3d12_execute_params_t 结构的 pDescriptorHeap 字段传递外部描述符堆。DescriptorHeapOffset 应指向 XeSS-SR 描述符表。

D3D11 相关

在 D3D11 中，XeSS-SR 执行函数为 GPU 准备工作负载，然后与用户 D3D11 命令一起排队并最终执行。输入/输出资源的访问与用户 D3D11 命令隐式同步。

Vulkan 相关

对于 Vulkan 集成，应用程序必须传递附加信息以及资源指针。结构 xess_vk_execute_params 使用附加结构来保存资源描述：

```
typedef struct _xess_vk_image_view_info
{
    VkImageView imageView;
    VkImage image;
    VkImageSubresourceRange subresourceRange;
    VkFormat format;
    unsigned int width;
    unsigned int height;
} xess_vk_image_view_info;
```

由于 Vulkan 不提供任何从 VkImage/VkImageView 检索该数据的方法，因此该结构的所有字段都很重要。

subresourceRange 和 view 参数应指向 XeSS-SR 使用的图像子资源（级别和层）。仅支持 2D 视图类型。对于深度图像和深度模板图像，subresource 和 view 应仅指向 VK_IMAGE_ASPECT_DEPTH_BIT 方面的图像。

固定输入分辨率

使用 XeSS-SR 的默认场景是使用所需质量和目标分辨率的固定输入分辨率。调用 xessGetOptimalInputResolution 并使用 pInputResolutionOptimal 值根据质量和目标分辨率来确定输入分辨率。在每次调用 xessExecute *时，必须*在 xess__execute_params_t 结构中提供实际的输入分辨率值。

请注意，从 XeSS 1.2 开始，xessGetInputResolution 函数已被弃用，仅为兼容性保留。

请注意，从 XeSS 1.3 开始，质量预设和分辨率缩放的映射已更改。请参考以下 XeSS 1.3 的表格：

预设	分辨率放大倍数
原生抗锯齿	1.0x（原生分辨率）
超高质量+	1.3x
超高质量	1.5x
质量	1.7x
平衡	2.0x
性能	2.3x
超级性能	3.0x
关闭	关闭 Intel XeSS-SR

XeSS-SR 提供了一种恢复到旧版质量和分辨率缩放映射的方法。可以通过使用 xessForceLegacyScaleFactors 函数来实现。调用此函数后将在下一次调用 xessGetOptimalInputResolution 和 xessD3D12Init 时使用以下缩放因子：

预设	分辨率放大倍数（旧版）
原生抗锯齿	1.0x（原生分辨率）
超高质量+	1.3x
超高质量	1.3x
质量	1.5x

平衡	1.7x
性能	2.0x
超级性能	3.0x
关闭	关闭 Intel XeSS-SR

为了使用旧版缩放因子，应用程序应按照以下代码示例的顺序初始化 XeSS-SR：

```
// 1. Create context
xess*CreateContext(device, &context);

// 2. Request legacy scale factors
xessForceLegacyScaleFactors(context, true);

// 3. Get optimal input resolution and initialize context
xessGetOptimalInputResolution(context, …);
xess*Init(context, …);
```

动态输入分辨率

XeSS-SR 支持动态输入分辨率同时固定目标分辨率的方式。在这种情况下，建议忽略 pInputResolutionOptimal 值，并在支持的范围内（[pInputResolutionMin; pInputResolutionMax]）自由更改输入分辨率，让应用程序在保持固定 FPS 的同时达到最佳视觉质量。必须在每次调用 xess*Execute 时作为 xess*__execute_params_t 结构的一部分提供实际的输入分辨率值。在变化输入分辨率时，建议应用程序尽可能让渲染分辨率的长宽比接近目标分辨率的长宽比，以确保 XeSS-SR 的最佳和稳定的视觉质量。

例如，对于 D3D12 路径，但对于 Vulkan 也是类似的：

```
xess_d3d12_execute_params_t params;
params.jitterOffsetX    = 0.4375f;
params.jitterOffsetY    = 0.3579f;

params.inputWidth = 1920;
params.inputHeight = 1080;

// xess records commands into the command list
xessD3D12Execute(&context, pd3dCommandList, &params);
```

```
// Application may record more commands as needed
pD3D12GraphicsCommandList->Close();

// Application submits the command list for GPU execution
pCommandQueue->ExecuteCommandLists(1, &pCommandLists);
```

请注意，从 XeSS 1.3 开始，质量预设到分辨率缩放的映射已更改。请参考以下 XeSS 1.3 的表格：

预设	动态分辨率放大范围
原生抗锯齿	N/A
超高质量	1.0x - 1.5x
质量	1.0x - 1.7x
平衡	1.0x - 2.0x
性能	1.0x - 2.3x
超级性能	1.0x - 3.0x
关闭	关闭 Intel XeSS-SR

抖动比例

函数 xessSetJitterScale 将缩放因子应用于抖动偏移。这对于应用程序以非像素为单位存储抖动时可能有用。例如：可以通过设置适当的比例将 NDC 抖动转换为像素抖动。

速度比例

函数 xessSetVelocityScale 将缩放因子应用于速度值。这对于应用程序以非像素为单位存储速度时可能有用。例如，可以通过设置适当的比例将归一化视口速度转换为像素速度。

曝光倍增值

函数 xessSetExposureMultiplier 将倍增值应用于曝光值。此倍增值适用于所有曝光模式：

- 自动曝光
- 通过 xess_d3d12_execute_params_t 结构传递曝光值
- 曝光纹理

推荐做法

视觉质量

我们建议在后处理刚开始，色调映射计算之前运行 XeSS-SR，以最大程度地提高图像质量。在某些情况下，也可以在色调映射后执行；然而，这种模式是实验性的，不能保证良好的质量。

为了最大化图像质量，请考虑以下几点：

- 使用高或超高质量设置的屏幕空间环境遮蔽（SSAO）和阴影。
- 关闭任何用于降低着色率和渲染分辨率的技术，例如可变速率着色（VRS）、自适应着色、棋盘格渲染、Dither 抖动等。
- 避免在 XeSS-SR 放大之前使用四分之一分辨率特效。
- 不要依赖 XeSS-SR 进行任何类型的去噪；噪声信号会显著损害图像重建质量。
- 在场景线性 HDR 空间中使用 fp16 精度的颜色缓冲区。
- 使用 fp16 精度的速度缓冲区。
- 调整 mip bias 以最大化图像质量并控制开销。
- 提供适当的场景曝光值。正确的曝光对于最小化运动物体的重影、模糊和精确的亮度重建至关重要。

驱动程序验证

为了获得最佳性能和质量，请安装最新的驱动程序。在使用 xess*CreateContext 初始化后，调用函数 xessIsOptimalDriver 验证安装的驱动程序是否可提供最佳体验。如果此函数返回 XESS_RESULT_WARNING_OLD_DRIVER，则应向用户显示建议安装较新驱动程序的提示或通知。XESS_RESULT_WARNING_OLD_DRIVER 不是致命错误，用户仍然可以继续使用。

调试技巧

运动矢量调试

如果 XeSS-SR 生成锯齿或不稳定的图像，可在静态场景中集中调试：

- 在引擎中模拟帧间的零时间增量，以保持完全静态的场景。
- 设置 0 运动矢量缩放，以排除与运动矢量相关的潜在问题。
- 显著增加重复抖动模式的长度。

XeSS-SR 应生成高质量的超级采样图像。如果没有发生这种情况，可能是抖动序列或输入纹理内容存在问题；否则，问题可能出在运动矢量的解码上。确保运动矢量缓冲区内容与当前设置的单位（NDC 或像素）相对应，并且轴方向正确。试试把运动矢量缩放因子设置成 1 或-1，以正确地对齐坐标轴。

抖动偏移调试

如果静态场景看起来不好，试试把运动矢量缩放因子设置成 1 或-1，以正确对齐坐标轴。确保抖动范围不超出 [-0.5, 0.5]。

版本控制

XeSS-SR 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（xess_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。

版本嵌入到 XeSS-SR SDK 发布中，可以通过调用函数 xessGetVersion 得到。版本号同时也包含在 zip 文件和其随附的 README，以及代码示例的头文件中。

其他资源

Survey of temporal anti-aliasing techniques

Intel® Arc™ landing page

Intel® XeSS Plugin for Unreal Engine on GitHub

DirectX download page

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。