VR Rendering Performance

Tunning and Optimizations

Contents

1 Introduction

Achieving an optimal VR experience on resource-limited hardware is key to delivering a smooth and comfortable user experience. If the frame rate of content rendering drops or unstable below the device’s refresh rate, it will lead to frame judder and stutting, motion sicknes, etc,. finally negatively impact the user experience. Therefore, optimizing the content performance is very important for ensuring an enjoyable experience.

Before starting performance tuning, it is important to understand where the performance bottlenecks are to avoid inefficient tunning. This document is designed to help developers identify performance bottlenecks and offer solutions to resolve the rendering performance issues.

The document is organized into the following sections:

- **Chapter 2: Identify the Bottleneck** – This section assists developers in identifying where the bottlenecks are.
- **Chapter 3 and 4: VIVE Wave and VIVE OpenXR Settings** – These sections outline specific settings that may impact CPU/GPU performance for VIVE Wave and OpenXR apps. Developers can experiment with enabling or disabling these features based on the performance bottlenecks encountered to determine if there is any improvement.
- **Chapter 5: Common Optimization** – This section shares some common optimization practices and experiences.

2 Identify The Bottleneck

When HMD is moving, if the VR/MR app has frame jitter or black edge, etc., it usually caused by bad rendering performance issue. Typically, rendering performance problems can be categorized into 2 types: CPU-bound or GPU-bound. Understand which types of bound for your app is very important at beginning to avoid inefficient tunning.

In this chapter, we provide simple steps that let you to quickly identify where the performance issues are.

2.1 Check Content Rendering FPS

First, we start by checking the content FPS that is the number of frames the content renders per second. It should be maintained to the display framerate and kept stable. Otherwise, it might cause frame jitters.

If your application SDK are using **VIVE WAVE SDK 6.0.0 or later**, you can use the following adb command to check the FPS.

```
$adb Logcat -s VRMetric
```

You will see the following log data.

```
VRMetric:FPS=89.8/89.8,CPU=27/1,GPU=72/3,GpuBd=0,LrCnt=1,2Stag=1,Pstat=2,AQ=1,FOVED=0/
0, FSE=1,TWS=2,PT=0(0),RndrBK=0,GLTA=2D,EB=1720x1720
```

“FPS=89.8/89.8” The first number represents the content FPS, while the second number represents the display framerate.

If your **Wave SDK version is below 6.0.0**, it is recommended to upgrade to the latest version to enhance rendering performance and others optimization.

If your application SDK built with **VIVE OpenXR**. You can use the following adb command to check the FPS.

```
$adb Logcat -s RENDER_ATW
```

You will see the following log data

```
RENDER_ATW: [FPS] new texture:90.00
RENDER_ATW: [FPS] R present:90.00 skip:0 (0.592317, -0.015515, 0.805527, 0.006788)
RENDER_ATW: [FPS] L present:90.00 skip:0 (0.592301, -0.015502, 0.805539, 0.006773)
```

The number following “new texture” represents currently content FPS. The number following “R present” and “L present” represents display framerate.

Sometimes, the content FPS and the display framerate might have a slight discrepancy. For example, in the case above, 89.8 FPS can be considered as 90 FPS.

If the app's content FPS is consistently lower than the display framerate or remains unstable, it indicates a rendering performance issue. Therefore, the next step is to identify whether the bottleneck is coming from the CPU or the GPU.

2.2 Check CPU and GPU usage

If your application SDK are using **VIVE WAVE SDK 6.0.0 or later**, you can use the following adb command to check the FPS.

```
$adb Logcat -s VRMetric
```

You will see the following log data.

```
VRMetric:FPS=89.8/89.8,CPU=27/1,GPU=72/3,GpuBd=0,LrCnt=1,2Stag=1,Pstat=2,AQ=1,FOVED=0
/0, FSE=1,TWS=2,PT=0(0),RndrBK=0,GLTA=2D,EB=1720x1720
```

As you can see in above log result, the CPU usage is 27% and the GPU usage is 72%

If your **Wave SDK version is below 6.0.0**, it is recommended to upgrade to the latest version to enhance rendering performance and others optimization.

For the VIVE OpenXR app, you can use the following command to check the CPU and GPU usage.

```
# on linux/ubuntu
$ adb logcat | grep CPU_USAGE
# on powershell
$ adb logcat | Select-String -Pattern CPU_USAGE
```

You will see the following log

```
                 CPU Avg.  CPU0    CPU1    CPU2    CPU3    CPU4    CPU5    CPU6    CPU7   GPU
CPU_USAGE [LOAD] 25.67%  32.22%  25.29%  30.77%  29.35%  21.35%  22.09%  18.39%  24.14%  73 %
```

If you observe that the FPS cannot maintain the display frame rate and the GPU usage is also very high, typically exceeds 85%, you can try adjusting the **Eyebuffer Resolution** (section 3.1.2, section 4.1.2) to see if it improves FPS. If this adjustment leads to better performance, we can conclude that the issue is GPU-bound and focus our optimization efforts accordingly.

On the other hand, if adjusting the Eyebuffer Resolution does not result in a noticeable performance improvement, the bottleneck is likely CPU-bound, and we should focus on optimizing CPU performance.

It's also possible that the application is both CPU-bound and GPU-bound simultaneously. In such cases, optimization efforts should be applied to both the CPU and GPU to achieve balanced performance improvements.

2.3 GPU-bound

When a VR app is GPU-bound, it means that the GPU is the primary bottleneck, and it's unable to keep up with the rendering demands of the application. To mitigate GPU-bound issues, consider the following recommendations:

First, use profiling tools like RenderDoc or Game Engine profiler (Unity Profiler, Unreal Insights) to analyze where the GPU is spending most of its time. Identify the costliest operations and focus on optimizing them.

- For Native Developer, you can use RenderDoc to identify which draw call is causing excessive GPU load.
- For Unity Developer, you can follow Unity this document or use RenderDoc to analyze rendering performance issue, and follow the Unity graphics optimization documentation for guidance to optimize your application.
- For Unreal Developer, you can use GPU Visualizer or use RenderDoc to analyze rendering performance issue, and follow the Unreal Performance Guidelines for guidance to optimize your application.

Second, you can also try adjusting certain Wave features or settings to reduce GPU loading.

1. Set **Display Refresh Rate** slower (section 3.1.1, section 4.1.1)
2. Adjust **Eyebuffer Resolution** (section 3.1.2, section 4.1.2),
3. Try to enable **Foveation** (section 3.1.4, section 4.1.4).

If your app is also an MR app, you can adjust the Passthrough settings as well.

1. Adjust **Passthrough Image Quality** lower. (section 3.2.1)
2. Adjust **Passthrough Framerate** slower. (section 3.2.2).

For more other settings about GPU performance, you can refer to Chapter 2.6.

2.4 CPU-bound

When a VR app is CPU-bound, it means that the CPU is the primary bottleneck, consider the following recommendations:

First, use profiling tools like Systrace or Game Engine profiler (Unity Profiler, Unreal Insights) to analyze and identify which parts of your code are consuming the most CPU resources. Focus on optimizing these areas and refactor computationally intensive algorithms to reduce CPU load.

- For Native Developer, you can use Systrace to profiler your project.
- For Unity Developer, you can use CPU Usage Profiler module to find CPU performance issue.
- For Unreal Developer, you can use Unreal's Insights to find CPU performance issue.

Second, you can also try adjusting certain Wave features or settings to reduce GPU loading.

1. Set **Display Refresh Rate** slower (section 3.1.1, section 4.1.1)
2. Use **Multi-View Rendering** (section 3.1.4, section 4.1.4)

If your app is also an MR app, you can adjust the Passthrough settings as well.

1. Adjust **Passthrough Framerate** slower (section 3.2.2).

For more other settings about CPU performance, you can refer to Chapter 2.6.

2.5 Summary

Finally, we've organized the above performance checking workflow into Figure 2-5-1. Start by checking the content's FPS. If it's lower than the display framerate or remains unstable, then analyze the GPU/CPU usage to determine if it's GPU-bound or CPU-bound. Lastly, use a profiler to identify potential performance issues or adjust Wave features or settings to optimize CPU performance.

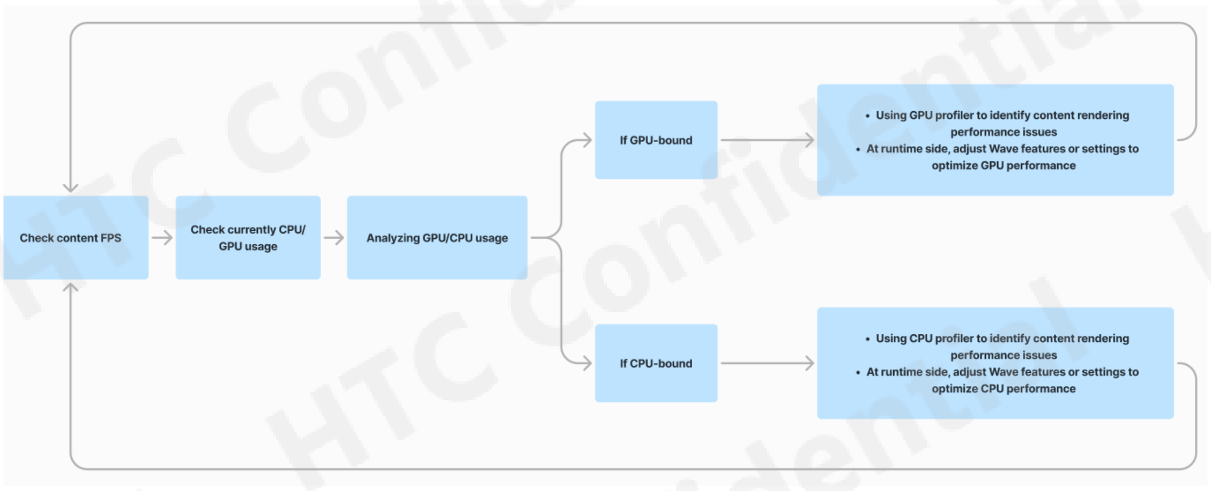
Check content FPS
Check currently CPU/GPU usage
Analyzing GPU/CPU usage
If GPU-bound
If CPU-bound
• Using GPU profiler to identify content rendering performance issues
• At runtime side, adjust Wave features or settings to optimize GPU performance
• Using CPU profiler to identify content rendering performance issues
• At runtime side, adjust Wave features or settings to optimize CPU performance

Figure 2-5-1 Check performance workflow

2.6 Quick Reference Which Settings Can Improve CPU/GPU loading

List the settings of SDK that related to the CPU/GPU loading as below. You can be based on the app's bottleneck to check the relevant optimization settings.

Related to CPU:

- VIVE Wave SDK Setting
  - VR Content
    - 3.1.1 Display Refresh Rate
    - 3.1.4 Multi-View Rendering
    - 3.1.6 Adaptive Quality
    - 3.1.7 Adaptive Motion Compositor
  - MR Content
    - 3.2.2 Adjust Passthrough Frame Rate
- VIVE OpenXR SDK Setting
  - VR Content
    - 4.1.1 Display Refresh Rate
    - 4.1.4 Multi-View Rendering
- Common Optimization

3 VIVE Wave Setting

VIVE Wave is an open platform and toolset that lets you easily develop VR content and provides high-performance device optimization for third-party partners. VIVE Wave support Unity and Unreal game engines.

We continuously optimize and resolve various bugs, so we recommended keeping the SDK up to date.

Currently, VIVE Wave only supports OpenGL ES. Here lists the features ordered by the influence to GPU performance. We will divide this into two parts: VR content and MR content.

3.1 VR Content

3.1.1 Display Refresh Rate

Higher refresh rates offer smoother visuals, but come at the cost of increased system load. Conversely, lower refresh rates reduce system load, but result in less smooth visuals. If App has CPU/GPU bound issue, you can try decreasing the display refresh rate to alleviate the issue.

- For Native developer, refer to WVR_SetFrameRate.
- For Unity developer, refer to this guide.
- For Unreal developer, refer to this guide.

3.1.2 Eyebuffer Resolution

Eyebuffer resoultion is the texture size that content App to be renderd, rendered texture will be submmited to the runtime to do posting process and present on the HMD display.

While a larger eye buffer size can result in clearer and more detailed visuals, but it also imposes a significant load on the GPU. Therefore, finding the right balance between visual quality and performance is essential.

If App has GPU bound issue, you can try decreasing the eyebuffer size by multiply a scale factor. Howerver, **we recommend not reducing the scale factor below 0.7**, as this may result in unacceptable visual quality.

- For Native developer, refer to WVR_ObtainTextureQueue. When adjusting the size, you should multiply the width and height by a ratio.

- For Unity developer, refer to WaveXRSettings.
  Alternatively, you can make changes via code as belwoe.

```
XRSettings.eyeTextureResolutionScale = ResolutionScaleValue;  // C#
```

- For Unreal developer, refer to SetPixelDensity.

3.1.3 Multi-View Rendering

In traditional rendering, we draw the left and right eyes separately, which requires two draw calls for the same scene. Multi-View Rendering addresses this issue by performing only one draw call.

This feature reduces CPU load by decreasing the number of draw calls. The GPU also has some benefits, vertex shader's workload is also reduced as it doesn't need to run an additional shader for the other eye, but the fragment shader's workload remains unchanged since it still needs to evaluate each pixel for both eyes. We recommand enabling this feature.

- For Native developer, you can refer to wvr_native_hellovr sample.
- For Unity developer, refer to the Render Mode, single pass is multi-view feature.
- For Unreal developer, refer to this guide.

3.1.4 Foveation

Foveated rendering is primarily designed to reduce the GPU load. It reduces the frame detail in the peripheral of display and maintaining high resolution detail in the center of the field of view. If App has GPU bound issue, you can try eanbling Foveation rendering.

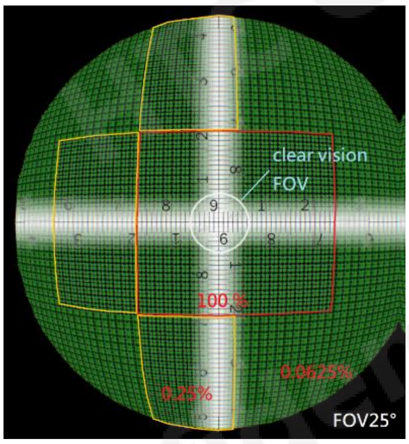

Figure 3-1-1 Foveated rendering

There are something need to notice while using foveation:

- ➢ Users typically don’t notice the reduced detail in peripheral regions applying default foveation mode. But if the peripheral quality of foveation is set too low, it could become noticeable to the user.
- ➢ The effects of foveation may be more noticeable with certain materials of textures, which could catch the user's attention. Developers should be aware of this and evaluate it accordingly.
- ➢ Enabling foveated rendering feature incurs a fixed GPU performance cost, which can vary between 1% to 6% depending on the size of the eye buffer. When using a simple shader in the scene, the performance gain from saving resources might be lower than the fixed GPU performance cost, resulting in a performance drop.

- For Native developer, refer to this guide.
- For Unity developer, refer to this guide. Notably, when you enable post-processing or HDR, foveation cannot be fully utilized. Because Unity will render objects onto its own generated render texture, rather than the runtime-generated present’s render texture that supports foveation.
- For Unreal developer, refer to this guide. Notably, foveation cannot be fully utilized on Multi-View Rendering, because Unreal cannot directly render objects onto the runtime-generated render texture that supports foveation.

3.1.5 Frame Sharpness Enhancement (FSE)

The FSE providing sharpen rendering result via introducing sharpen filter, it can make content more clear and be quite helpful for improving the clarity of the text in the scene. If App has GPU bound issue, you can consider disabling FSE if it is not essential.

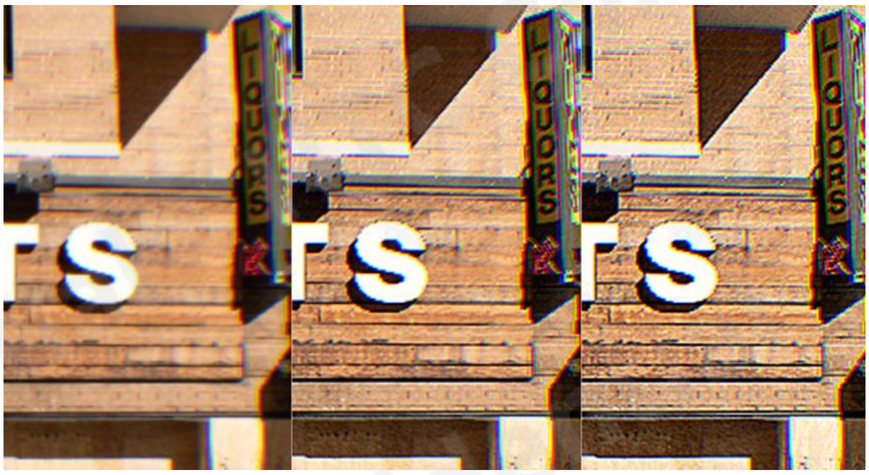

Figure 3-1-2 FSE effect

- For Native developer, refer to this guide.
- For Unity developer, refer to this guide.

- For Unreal developer, refer to this guide.

3.1.6 Adaptive Quality

To conserve battery and maintain the device's rendering performance, this feature automatically adjusts the performance levels of the CPU/GPU clock based on their usage. Additionally, other strategies can be implemented to enhance performance, such as automatically enable/disable Foveation or content can adjust itself if receiving high/low load events.

- For Native developer, refer to this guide.
- For Unity developer, refer to this guide. In our Unity plugin, the eye buffer size can be automatically adjusted based on current performance; Text size will filter out scale values that are too small in the Resolution list. We recommend text of size at least 20 dmm or larger.
- For Unreal developer, refer to this guide.

3.1.7 Adaptive Motion Compositor

This feature is experimental feature that includes UMC and PMC. UMC will reduce the Frame Rate by half and extrapolate new frame in real time to keep visual smoothness. However, it comes with some latency, artifacts and GPU loading.

PMC primarily uses the Depth Buffer to allow ATW to account for HMD translation, extend to a 6-dof compensation. This feature can reduce translation latency by 1~2 frames, but increase GPU loading.

- For Native developer, refer to this guide.
- For Unity developer, refer to this guide.
- For Unreal developer, refer to this guide.

3.1.8 Render Mask [Not Support Unreal]

Pixels at the edges become nearly invisible after distortion, the render mask modifies the depth buffer values of these invisible pixels. If you enable depth testing, due to early-z, these invisible pixels will not be rendered, thereby reducing the GPU load. This feature is useful if there are heavy-loading rendering objects in these invisible areas; otherwise, if there are no rendering objects in these areas, recommended to disable it because it will consume a small GPU usage..

- For Native developer, refer to this guide. You must binding the depth buffer before call RenderMask; otherwise, it will be ineffective.
- For Unity developer, refer to this guide.
- For Unreal developer, currently does not support the Render Mask feature.

3.2 MR Content

3.2.1 Adjust Passthrough Quality

There are 3 levels for passthrough image quality:

- **WVR_PassthroughImageQuality_DefaultMode** - suitable for the MR content without the specific demand.
- **WVR_PassthroughImageQuality_PerformanceMode** - suitable for the MR content that needs more GPU resource for virtual scene rendering.
- **WVR_PassthroughImageQuality_QualityMode** - suitable for the MR content that allows the users see the surrounding environment clearly, but the virtual scene of content must have the more fine tuning for performance.

You can adjust Passthrough quality to PerformanceMode to reduce GPU usage.

- For Native, Uunity or Unreal developer, refer to this guide.

3.2.2 Adjust Passthrough Frame Rate

Like the Display refresh rate, higher Passthrough framerate offer smoother visuals, but come at the cost of increased system load. Conversely, lower refresh rates reduce system load, but result in less smooth visuals. There are 2 modes of passthrough framerate: Boost and Normal.

- For Native developer, can adjust the passthrough quality using the WVR_SetPassthroughImageRate.
- For Unity developer, can change via code, example settings are as follows

```
// C#
Interop.WVR_SetPassthroughImageQuality(WVR_PassthroughImageQuality.PerformanceMode);
```

- For Unreal developer, setting method see the blueprint node in the Figure 3-2-2.

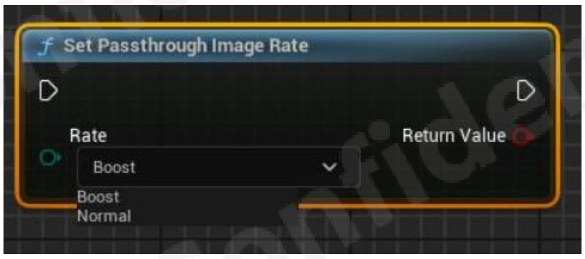

Figure 3-2-2 Passthrough image rate blueprint node

4 VIVE OpenXR Setting

OpenXR is open standard that provides a common set of APIs for developing XR applications that run across a wide range of VR devices, developed by the Khronos Group. The VIVE Focus 3 and VIVE XR Elite also support OpenXR, VIVE OpenXR SDK provides comprehensive support for HTC VR devices, allowing developers to build All-in-One and content with Unity and Unreal engine on HTC VR devices.

We continuously optimize and resolve various bugs, so it is recommended that developers update their device's FOTA version to keep it up to date.

Currently, VIVE OpenXR SDK supports OpenGL ES and Vulkan.

4.1 VR Content

4.1.1 Display Refresh Rate

The concept here is similar to **3.1.1 Display Refresh Rate**.

- For Native developer, refer to XrEventDataDisplayRefreshRateChangedFB.
- For Unity developer, refer to this guide.
- For Unreal developer, refer to this guide.

4.1.2 Eyebuffer Resolution

The concept here is similar to **3.1.2 Eyebuffer Resolution**. we recommend not reducing the scale factor below 0.7, as this may result in unacceptable visual quality.

- For Native developer, refer to xrCreateSwapchain. When adjusting the size, you should multiply the width and height by a ratio. ,
- For Unity developer, refer to the following example

```
// C#
XRSettings.eyeTextureResolutionScale = 0.7f;   //recommended 1.0f~0.7f
```

- For Unreal settings, refer to this guide.

4.1.3 Multi-View Rendering

The concept here is similar to **3.1.3 Multi-View Rendering**. This feature reduces the load on the CPU, GPU also has some benefits. We recommand enabling this feature.

- For Native developer, KhronosGroup provides an OpenXR Multi-View example, refer to this guide.
- For Unity developer, refer to the Render Mode, single pass is multi-view feature.
- For Unreal developer, as with VIVE Wave settings, refer to this guide.

4.1.4 Foveation [Not Support Unreal]

The concept here is similar to **3.1.4 Foveation**. Foveated rendering is primarily designed to reduce the GPU load but enabling it will incur a fixed GPU performance cost and if foveation is set too low and certain materials or textures are used, it can become very noticeable to the user. Therefore, it is advisable to enable or disable the feature based on your specific requirements and performance considerations

Currently, Foveated functionality is only supported in OpenGL ES on VIVE OpenXR SDK.

- For Native developer, this feature is available, but currently, no examples are provided.
- For Unity developer, refer to this guide.
- For Unreal developer, does not support this feature at the moment.

4.1.5 Render Mask [Not Support Unreal]

The concept here is similar to **3.1.8 Render Mask**.

- For Native developer, use XrVisibilityMaskKHR to get the Mesh. Before rendering the scene, use this Mesh to populate the depth buffer values before rendering the scene.
- For Unity developer, the Render Mask feature is enabled by default for OpenGL ES, and can be disabled with the following code; Vulkan currently does not support this feature.

```
    //C#
UnityEngine.XR.XRSettings.occlusionMaskScale = 0.0f;
```

- For Unreal developer, currently does not support the Render Mask feature.

4.2 MR Content

OpenXR currently does not support setting Passthrough Quality and Frame Rate. We will continue to optimize and fix the Passthrough feature, so recommended that developers update device's FOTA version to keep it up to date.

5 Common Optimization

5.1 Turn off High Performance Mode

Turning off "High performance mode" can reduce the display size of the device, thereby reducing GPU usage. The drawback is a decrease in screen resolution. You can balance quality and performance to decide whether to enable it.

The setting location for VIVE Focus 3 is shown Figure 5-1-1:

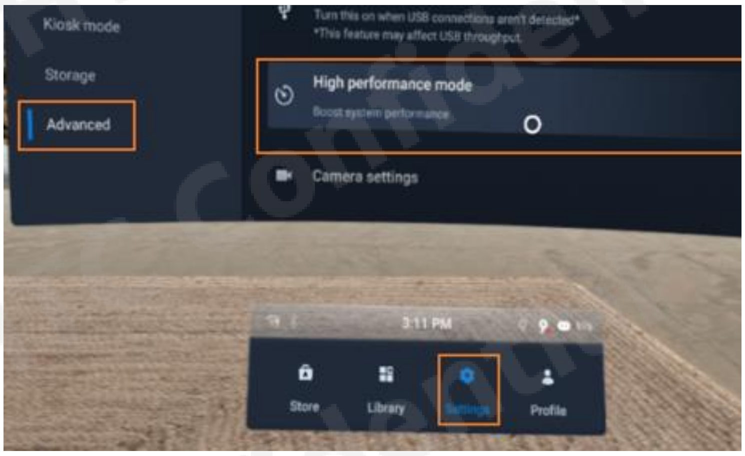

Figure 5-1-1 High Performance Mode on VIVE Focus 3

The setting location for VIVE XR Elite is shown Figure 5-1-2:

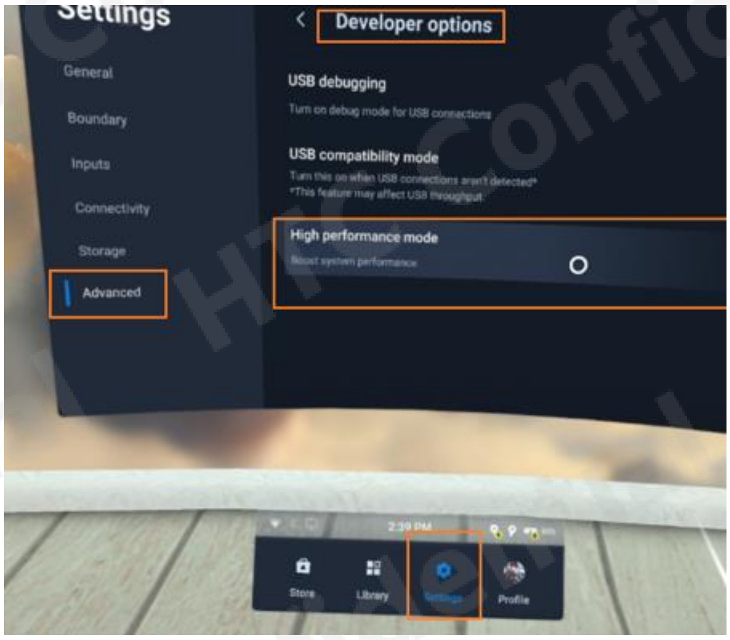

Figure 5-1-2 High Performance Mode on VIVE XR Elite

5.2 Multisampling Anti-Aliasing

Multisampling is an anti-aliasing technique used to smooth out jagged edges, usually is accelerated through hardware, which incurs GPU performance cost. We recommend not setting MSAA higher than 2x because more hight value will consume more gpu usage.

- For Native developer, MSAA OpenGL ES exsample can refer to this;
  MSAA Vulkan exampler can refer to this.

  The Adreno GPU provides an extension that optimizes MSAA.
- For Unity developer, refer to this guild.
- For Unreal developer, refer to this guild. Unreal also has provide post processing anti-aliasing, refer to this guild.

5.3 GMEM Load/Store

In the Adreno GPU architecture, there is a feature where, when binding a Render Target, if the Render Target don't clear or invalidate, each time rendering occurs, the values in the Render Target are loaded into Graphics Memory, which is called GMEM Load. If the previous values are not needed, clear or invalidate Render Target befaure rendering, can avoid this situation to improve GPU performance.

You can avoid GMEM Load using the following methods. In OpenGL ES, after binding the FBO, you can call glClear and glClearDepth to clear Color, Depth, and Stencil buffer, or call glInvalidateFramebuffer to invalidate the specified Render Target. In Vulkan, additional instructions are not necessary; you can explicitly set whether to clear the attachment before use in VkAttachmentDescription.loadOp.

Similarly, storing the result of a Tile Render back to Main Memory from Graphics Memory is called GMEM Store; this operation is also expensive for the GPU. To avoid this, we recommend binding only the required Render Targets to prevent unnecessary Store operations.

5.4 Composition Layer(Multi Layer)

Textures displayed using Multi-Layer have a better visual quality. However, this feature significantly increases GPU performance with the number of layers and the size of the textures. We recommend not over three layers.

- For Native developer,
  - VIVE Wave SDK uses WVR_SubmitFrameLayers to pass in data for each layer.

    - VIVE OpenXR SDK places layer data into XrFrameEndInfo and submits it via xrEndFrame.
- For Unity developer,
    - VIVE Wave SDK settings, refer to this guide,
    - VIVE OpenXR settings, refer to this guide.
- For Unreal developer,
    - VIVE Wave SDK settings, refer to this guide.
    - VIVE OpenXR settings, refer to this guide.

5.5 CPU Spike

When the CPU loading is heavier, some background processes threads having high priority, it might interrupt native execution. We cannot guarantee that the Content Application will not be interrupted by other thread.

If such issues arise, you can try increasing the thread priority to see if it resolves the problem. But if you change the thread configuration to optimize for devices, you need to check if this has any negative impact.

- For Unity Developer, refer to Android thread configuration feature. If you are using the VIVE Wave SDK, we have a feature in WaveXRSettings that allows you to adjust the priority, as shown in the Figure 5-5-2. The smaller value represents higher priority.

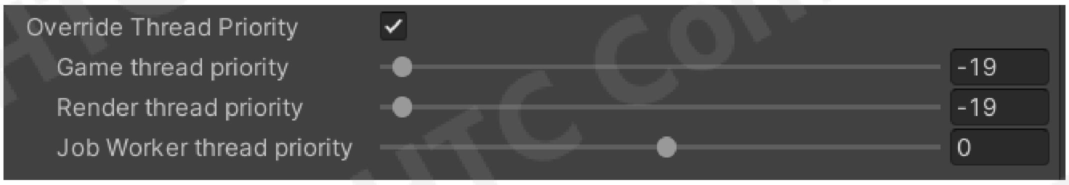
Override Thread Priority ✓
Game thread priority -19
Render thread priority -19
Job Worker thread priority 0

Figure 5-5-2 Override Thread Pirority

- Unreal no method to change game thread、rendering thread and RHI thread priority through external settings unless you modify the engine code.