Virtual Reality Application Developer Guide

The developer guidelines provided in this document are created with input from Katen Shah, Bill Sadler, Prasoon Surti, Mike Apodaca, Rama Harihara, Travis Schuler, and John Gierach.

Get general guidelines for developing and designing your virtual reality (VR) application, and learn how to obtain optimal performance. This guide is based on performance characterizations across several VR workloads, and defines common bottlenecks and issues. Find solutions to address black and white-bound choice-of-texture formats, fusing shader passes, and how to use post-anti-aliasing techniques to improve the performance of VR application workloads.

Goals

Define general design point and budget recommendations for developers on creating VR content using 7th generation Intel® Core™ i7 processors and Intel® HD Graphics 615 (GT2)

Provide guidance and considerations for obtaining optimal graphics performance on 7th generation Intel® Core™ i7 processors

Provide suggestions for optimal media, particularly 3D media

Get tips on how to design VR apps for sustained power, especially for mobile devices.

Identify tools that help developers identify VR issues in computer graphics on VR-ready hardware

Developer Recommended Design Points

GENERAL GUIDELINES ON DESIGN POINTS AND BUDGETS TO ISVs

CATEGORY	GUIDELINE
Triangles/Frame	200 K –300 K visible triangles in a given frame.*
	Use aggressive culling of view frustum, back face, and occlusion to reduce the number of actual triangles sent to the GPU.
Draws/Frame	500–1000*
	Reduce number of draw calls to improve performance and power. Batch draws by shader and draw front-to-back with 3D workloads (refer to 3D guide).
Target Refresh	At least 60 frames per second (fps), 90 fps for best experience.

Resolution	Resolution of head-mounted display (HMD) can downscale if needed to hit 60 fps but cannot go below 80 percent of HMD resolution.* Dynamic scaling of render target resolution can also be considered to meet frame rate requirements.*
Texture Budget	*TBD
Memory	180 MB – 200 MB per frame (DDR3, 1600 MHz) for 90 fps.*

* Data marked is to be used as a place holder and is WIP.

Considerations for Optimal Performance on General Hardware

Texture Formats and Filtering Modes

- Texture formats and filtering modes can have a significant impact on bandwidth.
- Generally 32-bit and 64-bit image formats are recommended for most filtering modes (bilinear and so on).
- Filtering trilinear and volumetric surfaces with standard red green blue and high-dynamic range (sRGB/HDR) formats will be slower compared to 32-bit cases.

Uncompressed Texture Formats

Uncompressed formats—sRGB and HDR —consume greater bandwidth. Use linear formats if the app becomes heavily bandwidth-bound.

HDR Formats

The use of R10G10B10A2 over R16G16B16A16 and floating point formats is encouraged.

Filtering Modes

Filtering modes, like anisotropic filtering, can have a significant impact on performance, especially with uncompressed formats and HDR formats.

Anisotropic filtering is a trade-off between performance and quality. Generally anisotropic level two is recommended based on our performance and quality studies. Mip-Mapping textures along with anisotropic levels add overhead to the filtering and hardware pipeline. If you chose anisotropic filtering, we recommend using bc1–5 formats.

Anti-Aliasing

Temporally stable anti-aliasing is crucial for a good VR experience. Multisample anti-aliasing (MSAA) is bandwidth intense and consumes a significant portion of the rendering budget. Anti-aliasing algorithms that are temporally stable post-process, like TSCMAA, can provide equivalent functionality at half the cost and should be considered as alternatives.

Low-Latency Preemption

Gen hardware supports object-level preemption, which usually translates into preemption on triangle boundaries. For effective scheduling of the compositor, it is important that the primitives can be preempted in a timely fashion. To enable this, draw calls that take more than 1 ms should usually have more than 64–128 triangles. Typically, full-screen post-effects should use a grid of at least 64 triangles as opposed to 1 or 2.

APP SCHEDULING

1. **Recommendation: Nothing additional is required.**

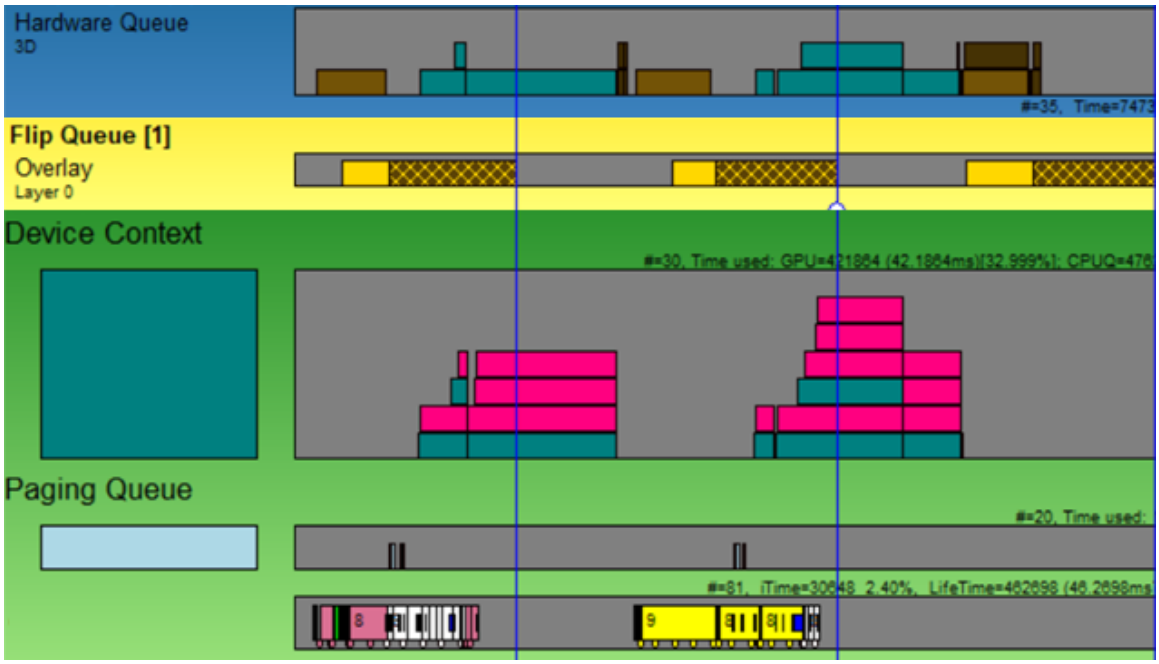
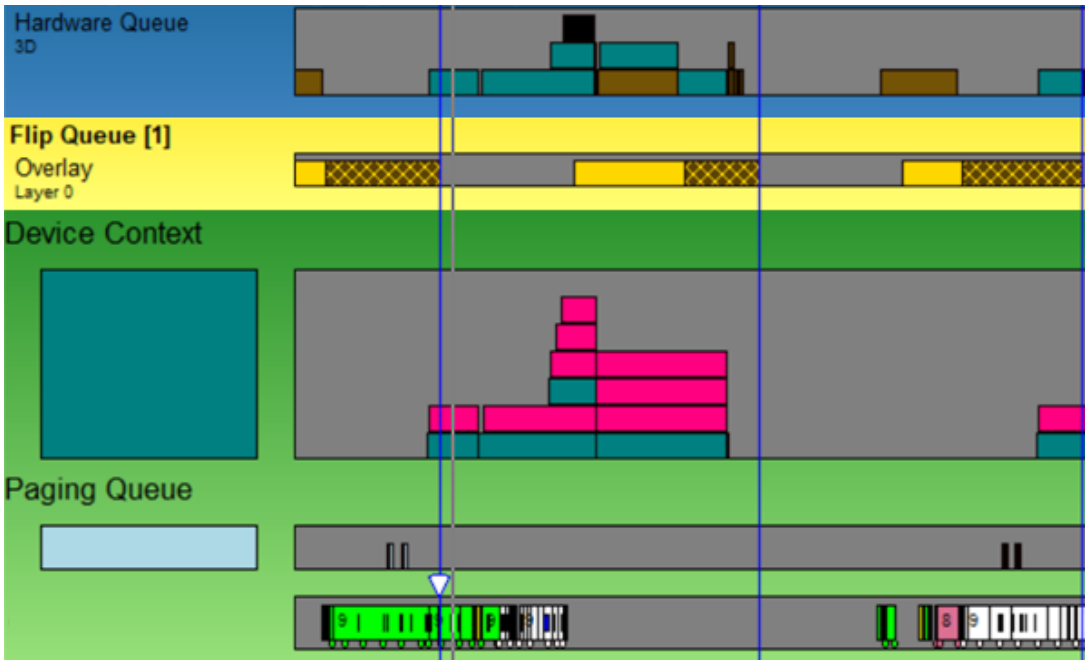


Table 1: Recommendation: Nothing additional is required.

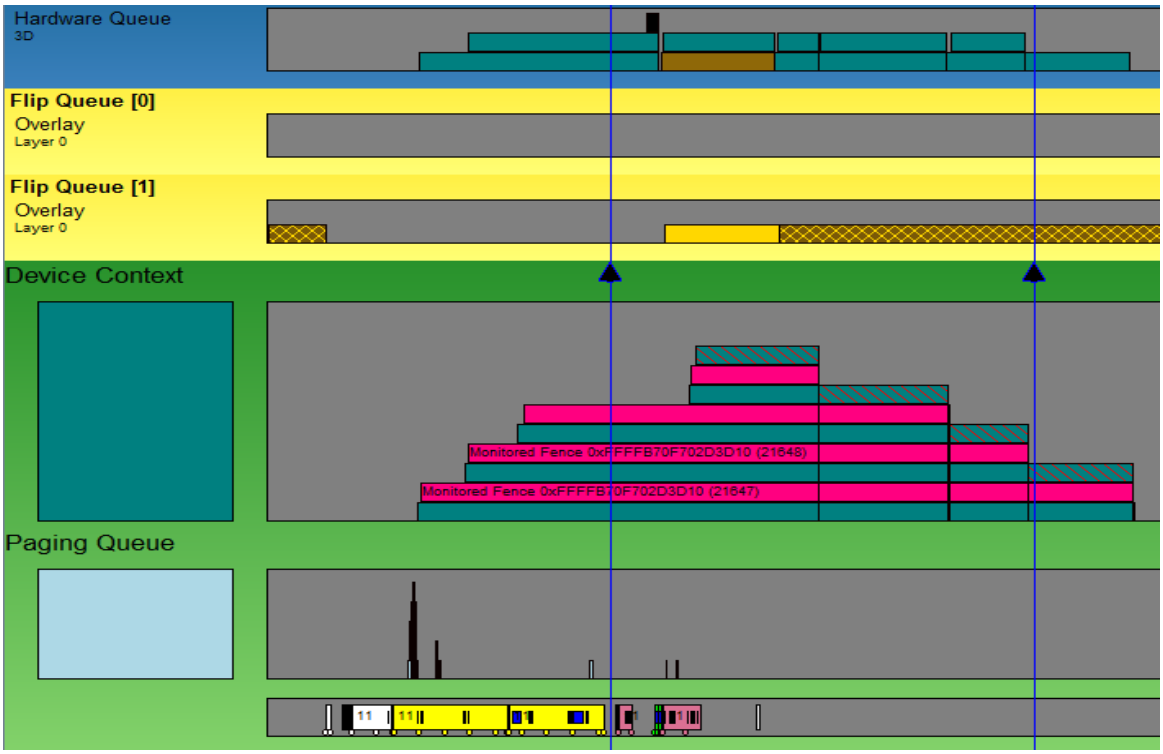
In the ideal case for a given frame, the app will have ample time to complete its rendering work between the vsync and before the link state request (LSR) packet is submitted. In this case, it is best that the app synchronize on the vsync itself, so that rendering is performed on the newest HMD positional data. This helps to minimize motion sickness.

2. **Recommendation: Start work earlier by syncing on the compositor starting work rather than vsync.**



When the frame rendering time no longer fits within this interval, all available GPU time should be reclaimed for rendering the frame before the LSR occurs. If this interval is not met, the compositor can block the app from rendering the next frame by withholding the next available render target in the swap chain. This results in entire frames being skipped until the present workload for that frame has finished, causing a degradation of fps for the app. The app should synchronize with its compositor so that new rendering work is submitted as soon as the present/LSR workload is submitted. This is typically accomplished via a wait behavior provided by the compositor API.

3. Recommendation: Present asynchronously.



In the worst case, when the frame rendering time exceeds the vsync, the app should submit rendering work as quickly as possible to fully saturate the GPU to allow the compositor to use the newest frame data available, whenever that might occur relative to the LSR. To accomplish this, do not wait on any vsync or compositor events to proceed with rendering, and if possible build your application so that the presentation and rendering threads are decoupled from the rest of the state update.

For example, on the Holographic API, pass DoNotWaitForFrameToFinish to PresentUsingCurrentPrediction, or in DirectX*, pass SyncInterval=0 to Present.

4. Recommendation: Present asynchronously.

Use GPU analysis tools, such as GPUView, to see which rendering performance profile you have encountered, and make the necessary adjustments detailed above.

Other design considerations

Half float versus float: For compute-bound workloads, half floats can be used to increase throughput where precision is not an issue. Mixing half and full resolution results in performance penalties and should be minimized.

Tools

The following tools will help you identify issues with VR workloads.

GPU View: Gives specifics on identifying issues with scheduling and dropped frames.

Intel® Graphics Performance Analyzers: Gives specifics on analyzing VR workloads and the expected patterns we see. For example, two sets of identical calls for the left and right eyes.

Additional Resources

A Graphics API Developer Guide for 6th Generation Graphics Processors

https://software.intel.com/en-us/articles/6th-gen-graphics-api-dev-guide

A Unity* Optimization Guide for Processor Graphics

https://software.intel.com/en-us/android/articles/unity-optimization-guide-for-x86-android-part-1

Compute Architecture for 6th Generation Graphics Processors

https://software.intel.com/sites/default/files/managed/c5/9a/The-Compute-Architecture-of-Intel-Processor-Graphics-Gen9-v1d0.pdf

Summary

The biggest challenge to performance for VR workloads comes from being bandwidth-bound. The texture format, fusing shader passes, and using post anti-aliasing techniques help reduce the pressure on bandwidth.

Notes

The features and benefits of Intel® technologies 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.
Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of others.
© 2017 Intel Corporation