Examining Vulkan video extensions for XR applications

Emmanouil Potetsianakis
Emmanuel Thomas, Xiaomi
Outline

1. XR application examples and requirements
2. Vulkan video extension
3. MPEG-I Part 13 Video Decoding Interface (VDI)
4. VDI & Vulkan
5. Performance tests of multiple decoding instances in mobile devices
6. Future Work & Conclusion
XR application examples and their requirements

- VR immersive virtual environment
  - 3D models (geometry, texture, atlases), video overlays
  - High-throughput, assets synchronization, *multiple decoders*, media orchestration

- AR conferencing
  - 2.5D (RGB+D) streams
  - Low-latency, *multiple decoders*

- XR gaming
  - 2.5D streams (for remote rendering) and/or 3D models (for local rendering)
  - Low-latency, high-throughput, assets synchronization, *multiple decoders*, media orchestration
XR + multiple decoders (e.g. RGB+D)

- Multiple decoders are often needed
- Even “simple” use-cases that provide RGB+D (for stereo, pose correction etc.) require at least 2 decoders
- Uncertainties of such systems:
  - Number of instances available at runtime
  - Different resolutions per stream
  - Different codec per stream
  - Output sync
RGB frame & Depth frame misalignment

Example of a capture of a moving hand with a misalignment of 1 frame between the RGB and the Depth frames

Capture Source: https://github.com/IntelRealSense/librealsense/issues/5675
Vulkan Video for video-decoding.

Parsing, Resource management and Synchronization to “application”.

**VkBuffer** for input bitstream

**VkImageView** (of **VkImage**) for decoded output and DPB images (inside decoder)

Source: https://www.khronos.org/blog/an-introduction-to-vulkan-video
MPEG-I VDI: Video Decoding Engine and Interfaces

Legend
- MDS: media stream
- ES: elementary stream
- MTS: metadata stream
- DS: decoded sequence
- m: number of input metadata streams
- n: number of media streams
- j: number of video decoder instances
- p: number of output metadata streams
- q: number of decoded sequences

Application configuration and capability query
MPEG-I VDI

- Group decoder instances together for more efficient management
- Inputs media streams to merge/split bitstream
- Outputs decoded image sequences in synchronized fashion
- Provides global decoding capabilities and configuration
- Builds on existing technologies (e.g. Khronos Vulkan)
VDI and Vulkan

Input Video Decoding Interface
Input Formatting
Input Buffers
Khronos Vulkan Video
Video Decoding Instances
Hardware Video Decoders
Output Video Decoding Interface
Output Formatting
Output Buffers

Data exchange
Control commands
Decoding in mobile devices

Decoding in mobile devices is relevant because they share some characteristics with (standalone) XR devices such as:

• Have limited processing capabilities
• Energy consumption is essential
• Used for immersive communication
• Share same family of (mobile) chipsets – thus having similar APIs

Therefore decoder instance management is imperative for both device types.
Multiple decoding instances performance

**Motivation:** demonstrating the benefits of Vulkan and VDI for XR applications

**Target:** measuring performance and scalability of multiple decoders in AR-enabled devices

**Methodology:** we measure the decoding times of frames using the Android video decoder (since it is the current alternative to the Vulkan/VDI stack)

**Tool:** *VidBench* – in-house video decoding benchmark tool for Android

**Device:** we selected smartphones as they are comparable to hardware capabilities of other XR devices (e.g. AR Glasses)

**Content:** AVC-encoded FullHD video of test sequence Big Buck Bunny. Duration: 2mins. Two versions: 1. I-Frame only (live-like) 2. I-Frames and P-Frames (low-delay)
The VidBench tool

• In-house Android application using the Android MediaCodec and MediaFormat APIs

• Characteristics:
  • single-thread per decoder
  • user defines the number of decoders (per test)
  • user defines the number of runs (per test)

• Benchmark settings:
  • 12 input buffer slots observed
  • each test has 20 runs
VidBench logic extract (pseudocode)

1. Getting the input buffer
2. Writing NAL to the input buffer
3. Queue the input buffer
4. Getting the output buffer
5. Release output buffer (without rendering)
Measuring performance of decoding in mobile devices (number of decoders)
Measuring performance of decoding in mobile devices (coding parameters)

![Graphs showing mean decoding time vs. number of decoder instances for different coding parameters.](image-url)
Cross-device decoding performance

Experiment 3: Performance of different devices

**Note 8T**
- **Chipset**: Qualcomm SDM665 Snapdragon 665 (11 nm)
- **CPU**: Octa-core (4x2.0 GHz Kryo 260 Gold & 4x1.8 GHz Kryo 260 Silver)
- **GPU**: Adreno 610
- **OS**: Android 9

**Note 9 Pro**
- **Chipset**: Qualcomm SM7125 Snapdragon 720G (8 nm)
- **CPU**: Octa-core (2x2.3 GHz Kryo 465 Gold & 6x1.8 GHz Kryo 465 Silver)
- **GPU**: Adreno 618
- **OS**: Android 11
Measuring performance of decoding in mobile devices (device)

Mean Decoding Time / Decoder Instances

5 Decoders / R.A.

Note8

Note9Pr
Future Work

On benchmarking

• Tests with different codecs.

• Port VidBench on a Vulkan-based platform (if and when available in Android)
  • Include mixed assets in the tests

On VDI status

• MPEG is finalizing the first edition of the VDI specification (Q1 ‘23).

• Discussion of 2nd edition of VDI has already started, focusing on extending support for other codecs, defining metadata streams and validation/conformance framework.