Vulkan Video
Core API Introduction

Tony Zlatinski, NVIDIA
April 2021
Vulkan Video Design Goals

- Low-level stateless management of hardware for efficiency and flexibility
  - Low-level synchronization for lower processing latency and efficient hardware scheduling
  - Low execution overhead
  - Low CPU/GPU/HW and memory resource utilization
- Suitable for low-power/memory embedded devices to high-performance servers
- Distribution of video processing across multiple CPU cores and video-codec devices
- Closer integration with Vulkan Graphics and Displays
Vulkan Video Core and Codec Extensions

- Vulkan Video Decode Core
  - Vulkan Video Decode h.264
  - Vulkan Video Decode h.265
- Vulkan Video Encode Core
  - Vulkan Video Encode h.264
  - Vulkan Video Encode h.265
- Planned for future release
  - Vulkan Video Decode VP9
  - Vulkan Video Decode AV1
  - Vulkan Video Encode AV1
Vulkan Video Profiles

- Vulkan Video Profiles are containers of formats describing the compressed bitstream
- VkVideoProfile describes:
  - videoCodecOperation - codec operations such as h264 encode, h265 encode, etc.
  - chromaSubsampling - YCBCr 4:4:4, 4:2:2, 4:2:0, 4:0:0 color subsampling mode
  - lumaBitDepth and chromaBitDepth describe luminance & chroma channel bit depth - 8, 10, 12-bit

- Video Profile structure must be included when obtaining device properties or creating Vulkan objects that will be used with Vulkan Video

<table>
<thead>
<tr>
<th>Vulkan API</th>
<th>Vulkan Structure to Extend</th>
</tr>
</thead>
<tbody>
<tr>
<td>vkGetPhysicalDeviceFormatProperties2</td>
<td>VkFormatProperties2</td>
</tr>
<tr>
<td>vkCreateImage</td>
<td>VkImageCreateInfo</td>
</tr>
<tr>
<td>vkCreateImageView</td>
<td>VkImageViewCreateInfo</td>
</tr>
<tr>
<td>vkCreateBuffer</td>
<td>VkBufferCreateInfo</td>
</tr>
<tr>
<td>vkCreateQueryPool</td>
<td>VkQueryPoolCreateInfo</td>
</tr>
</tbody>
</table>
Video Session Object

- VkVideoSession object contain (read-only) stream configuration parameters and maintains the context associated with the stream
  - One session object per video stream
- Created before using any video decode or encode operations
  - Specifies the video profile and maximum parameters for the video stream
- A video session instance supports a single compression standard only
  - H.264, HEVC, VP9, AV1, etc.
- Video Session object maintains the device memory heaps
  - The application allocates and binds VkDeviceMemory objects to the Video Session object which uses it for its memory heaps
Video Session Parameters Object

- VkVideoSessionParameters object contains processing parameters
  - Created against and belongs to a Video Session object

- Use multiple VkVideoSessionParameters objects to process a stream
  - An object can apply to the whole stream or a portion
  - Session Parameter object is provided with the vkCmdBeginVideoCoding command and remains in effect until the next vkCmdEndVideoCoding command

- Can add parameters to a the VkVideoSessionParameters object
  - Previously parameters cannot be modified
  - Can clone all video parameters into a new Session Parameter object
Example of a Set of HEVC Codec Parameters

Video Session Parameters Object

SPS parameters ID 0

VPS parameters ID 0

Other Parameters

PPS parameters ID 0

PPS parameters ID 1

PPS parameters ID 2

PPS parameters ID N

Implementation-specific stream context (opaque)
Video Decode/Encode DPB Picture Resources

DPB* stand for Decoded Picture Buffer

Legend
- Externally visible objects
- Implementation-specific data

Global metadata
- VkImage Opaque Metadata
- VkImage Metadata

Common Device Memory
- Video Session Internal Device Memory Heap Implementation

Per-slot session metadata
- Reference Picture Metadata
- Per-slot session metadata

Per-slot Device Memory

Picture Resource
- VkImageView
- VkImage
- VkDeviceMemory

Slot State

DPB
- 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

This work is licensed under a Creative Commons Attribution 4.0 International License

© The Khronos® Group Inc. 2021 - Page 8
Video and General Vulkan Objects

Video and General Vulkan Objects

Vulkan Video API

Vulkan API

VkVideoSession

H.265  H.264
VP9  AV1

VkVideoSession Parameters

VkQueues

VK_QUEUE_VIDEO_DECODE
VK_QUEUE_VIDEO_ENCODE
VK_QUEUE_TRANSFER

 VkEvent
 VkBuffer
 VkFence
 VkImage
 VkSemaphore
 VkImageView
 VkDeviceMemory
 VkMemoryBarrier
 VkBufferMemoryBarrier
 VkImageMemoryBarrier
 VkQueryPool

Vulkan Core
Typical Vulkan Video Decode & Encode App

Video decoder application
- Parse and process
- Stream

Video decode queues
- Fetch YUV->RGB
- YCbCr

GFX rendering application
- Fetch YCbCr
- Render
- Convert RGB -> YCbCr
- Cross-process queues

Video encode application
- Convert RGB -> YCbCr
- FB

Compositor

Video encode queues

Sync with Vulkan WSI
Sync with 3D

VULKAN VIDEO APIS
Video Encode Processing In Vulkan

Vulkan Video Application

- Encode frames
- Populate Parameters
- Record the Command Buffers
- Submit frame to HW queue
- Handle Ref slots management
- VkImageView(s)
- VkVideoSession
- VkVideoSessionParameters

Input VkImageView(s)

Most encode use cases do not require additional bitstream data insertion. Implementation can generate Non-VCL data.

More advanced video compression use cases may require some bitstream assembly.

The implementation can report bitstream offset and size for each output buffer via video bitstream encode queries.
Video Decode Processing In Vulkan

1. Copy or map bitstream to device memory
2. Populate Command Buffer
3. Submit frame to HW
4. Handle DPB slots management
5. Ref VkImageView(s)
6. VkVideoSession
7. VkVideoSessionParameters
8. VkVideoSessionParameters
9. User API
10. IHV Implementations
11. Wait for a GPU Semaphore
12. Present frame
13. Submit to graphics
14. GFX Render

Video Decode Application:
- Vulkan video session is required for all video operations
- Multiple video session parameters objects are supported
- Low-level memory management enables reduced memory footprint
- Closer integration with the graphics APIs allows for lower presentation latency

Vulkan video session is required for all video operations. Multiple video session parameters objects are supported. Low-level memory management enables reduced memory footprint. Closer integration with the graphics APIs allows for lower presentation latency.
Optimizing Memory Usage

- Create Sessions with the maximum parameters required for video content
  - Max resolution, max number of DPB, etc.
- Allocate image and buffer resources on demand
  - When the content requires those resources
- Free image or buffer resources that are not required
- Strip the resources of their physical memory backing
  - Using sparse memory binding if supported
- Enable the output of the decoded images to be directly consumed by Vulkan graphics and display processing pipelines
- Enable for the output of the Vulkan graphics or display processing to be consumed directly by the encoder’s input
Vulkan Sparse Resources with Video

- 4k/8k and 10/12-bit video content requires significant memory resources for stream buffers and picture images
  - One frame can be bigger than 2 MB. Video session may require 3-8 input and/or output images and 4 to 16 references that requires hundreds of megabytes of memory
- IHVs should support Vulkan Sparse binding for buffers and images for memory efficient resource management
  - Sparse Partially-Resident Buffers
  - Support for both Sparse Buffers with VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT enables for portions of the Vulkan buffers used for the input or output stream to be unmapped
- Sparse Partially-Resident Images
  - VK_IMAGE_CREATE_SPARSE_BINDING_BIT and VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT to support an efficient memory use during content resolution change
  - Use vkQueueBindSparse() before or after the queuing video commands
Reusing images without reallocation on decode size change

If Sparse Residency is supported, this unused area may not use any physical memory.

Applications can save memory by removing physical memory residency, if supported by the implementation.
Vulkan Video API Advantages

Parse and record command buffers (in parallel and out of order).

Command buffers and bitstream data are built directly in the video device memory ahead of time before submission to the hardware.

Command buffers can be built in parallel (using multiple host threads).

Command buffers can be scheduled at the device (not host) side.

Submit in decode order or order frames at the device side using Timeline Semaphores.
Vulkan Video API Advantages (2)

One video queue instance can decode (or encode) commands representing different video streams in parallel.

Submit in decode order or order frames at the device side using Timeline Semaphores.
Vulkan Video API Advantages (3)

- Application can optimize use of system resources
  - Allocate required decode or encode resources only when needed: Input, Output Picture Images, DPB, Stream, and Command buffers
  - Delete objects and remove the backing physical memory as soon as possible
  - Reuse resources when content size changes
Video VkCommandBuffers Queue Submission

- Regular Vulkan Queue Submit Sequence:
  - One or more Recorded Vulkan Video Command Buffers can be submitted
  - The command buffer sequences can be synchronized by binary or timeline semaphores
  - The command buffer sequences can be synchronized with the host CPU via semaphores or a fence
**VkCommandBuffers Video Recording Sequence**

- All Vulkan Video command sequences start with `vkCmdBeginVideoCoding` and end with `vkCmdEndVideoCodingKHR`.
- Multiple Video Start/End command sequences are supported.
- Implicit ordering guarantees also apply to the video and other commands that belong to the same command buffer.
VkCommandBuffers Recording Context Setup

- `vkCmdBeginVideoCoding` via the `VkVideoBeginCodingInfo` parameters establishes a context for the subsequent video decode and/or encode commands.
- `vkCmdEndVideoCodingKHR` terminates the context established by the last `vkCmdBeginVideoCoding`.

**VkVideoBeginCodingInfo**

- **Selects Active Video Session Object**
- **Sets Stream Codec Parameters**
- **Sets codec quality preset**
- **Sets Reference Picture Resources and Slots**
Recording VkCommandBuffer Commands

Only commands for decode/encode, barriers/events/query and transfer** operation are supported between VkBeginVideoCoding and VkEndVideoCoding.

- **Handle Global Memory Barriers**
  - VkMemoryBarrier

- **Handle Image Layouts Transition**

- **Handle buffer barriers for stream buffers**
  - VkBufferMemoryBarrier

- **Handle image barriers for input, output and DPB images and buffers**
  - VkImageMemoryBarrier

- **Set per frame or slice queries***
  - vkCmdResetQueryPool()
  - vkCmdBeginQuery()
  - vkCmdEndQuery()

- **Set frame events**
  - vkCmdWaitEvents()
  - vkCmdSetEvent()
  - vkCmdResetEvent()

- **One or more Video Decode operations**
  - vkCmdDecodeVideoKHR

- **One or more Video Encode operations**
  - vkCmdEncodeVideoKHR

- **Transfer and/or blit Operations**
  - vkCmdCopyBuffer
  - vkCmdCopyImage
  - vkCmdCopyBufferToImage
  - vkCmdCopyImageToBuffer
  - vkCmdBlitImage

* Some implementations may NOT support video queries.

** Transfer operation are only allowed if the video decode and/or encode queue advertises transfer operations via the VK_QUEUE_TRANSFER_BIT
Special Image Layout Transitions

- **DPB image special handling**
  - DPB images implicitly transition to VK_IMAGE_LAYOUT_UNDEFINED when:
    - Image is used for the first time with a video session
    - Content size or other parameters change within a video session
    - VkVideoSession object is reset
    - DPB slot is assigned for the first time with the image view representing the image
    - Video may need a structure like VkSampleLocationsInfoEXT to simplify those rules?
  - DPB images layout should not be affected when:
    - Transitioning reference images from VK_IMAGE_LAYOUT_VIDEO_DECODE_DPB or VK_IMAGE_LAYOUT_VIDEO_DECODE_DST to Gfx/compute friendly layouts

- **Video Input images transition**
  - When the content size or parameters change encode input images implicitly transition from VK_IMAGE_LAYOUT_VIDEO_ENCODE_SRC to VK_IMAGE_LAYOUT_UNDEFINED or VK_IMAGE_LAYOUT_PREINITIALIZED
DPB Slot Management

- Allocating/Associating DPB reference slots with *slotId*
  - Add entry with slotId and associated VkImage resource in the array of VkVideoBeginCodingInfoKHR::pReferenceSlots within the vkCmdBeginVideoCoding command

- Making DPB reference slot with slotId valid
  - Decode (vkCmdDecodeVideoKHR) or Encode (vkCmdEncodeVideoKHR) commands targeting slotId within the pSetupReferenceSlot

- Invalidating DPB slot with slotId
  - Replace association of the reference slot with slotId with a different VkImage resource
  - Decode or Encode commands targeting the reference slot with slotId (with pSetupReferenceSlot)
  - Reset the decoder/encoder
  - Replace the content of the associated VkImage resource or unbind the backing memory
  - Change the layout of the associated VkImage resource to an incompatible layout
DPB Slot Management Example

Allocating slots with associated picture resources

_frames__|__B__ |__1__ |__2__ |__3__ |__4__ |__5__ |__6__ |__7__ |__8__ |__9__ |__10__ |__11__ |__12__ |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Available DPB slot  Allocated Target  Reference  Current FB  Image and buffer Memory Barriers  New Begin/end sequence  Manage Image/buffer Sparse Bindings

How to deal with Gaps and “non-existent” references?

Note: Sparse may require a different queue family to submit, if the video queue families do not advertise sparse capabilities
DPB Slots
Multi-threaded cmdBuffer Recording

Frames

<table>
<thead>
<tr>
<th>Frames</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

FB

- Thread 0
- Thread 1
- Thread 2
- Thread 3
- Thread 4
- Thread 5
- Thread 6
- Thread 7

DPB slots

- Available DPB slot
- Allocated Target
- Reference
- Current FB
- Image and buffer Memory Barriers
- New Begin/end sequence
- Manage Image/buffer Sparse Bindings

Application record buffers with multiple (8) threads

Is it valid to perform the above multi-threaded command buffer recording?
Video Queries

- **Result Status Query (optional)**
  - Used to check whether a set of operations has been completed successfully
  - Type is VK_QUERY_RESULT_WITH_STATUS_BIT_KHR
  - Can be used with other than video queue families

- **Encode Bitstream Range Query**
  - Describes range of bytes written in the bitstream buffer by video encode commands
  - Type is VK_QUERY_TYPE_VIDEO_ENCODE_BITSTREAM_BUFFER_RANGE_KHR

- **Queries supported with Video**
  - Host side vkGetQueryPoolResults()

- **Queries not supported with Video**
  - Device side: vkCmdCopyQueryPoolResults()
Video Properties and Capabilities

- **Supported codecs for a particular Vulkan video queue**
  - Queried through vkVideoQueueFamilyProperties2KHR, chained to vkGetPhysicalDeviceQueueFamilyProperties() function

- **Supported video decode and encode capabilities**
  - Queried through vkGetPhysicalDeviceVideoCapabilitiesKHR() function

- **Supported video output, input and DPB image formats**
  - Enumerated through vkGetPhysicalDeviceVideoFormatPropertiesKHR() function