OPTIMISING A AAA VULKAN TITLE ON DESKTOP

LOU KRAMER

LOU KRAMER

DEVELOPER TECHNOLOGY ENGINEER
AMD

THE GAME

First Vulkan game using the engine

Engine had existing DX11 and DX12 support on top of an internal rendering API

Once the Vulkan version was somewhat stable, we started to look at the performance side of things ☺

THE GAME

- Best practices

-> hopefully minor changes only

- Other optimization opportunities?

-> require probably a bit more work

-> start early enough, can introduce new problems

BEST PRACTICES

- Is compression enabled for the G-buffer render targets?
- How do the barriers look?
- Can we make use of the copy queue?
- What about the shader building infrastructure?
- … usage flags, use of correct layouts, etc.

BEST PRACTICES

- Is compression enabled for the G-buffer render targets?
- How do the barriers look?
- Can we make use of the copy queue?
- What about the shader building infrastructure?
- … usage flags, use of correct layouts, etc.

☑ This is a checklist you can follow through and verify for your own engine

OTHER OPTIMIZATION OPPORTUNITIES

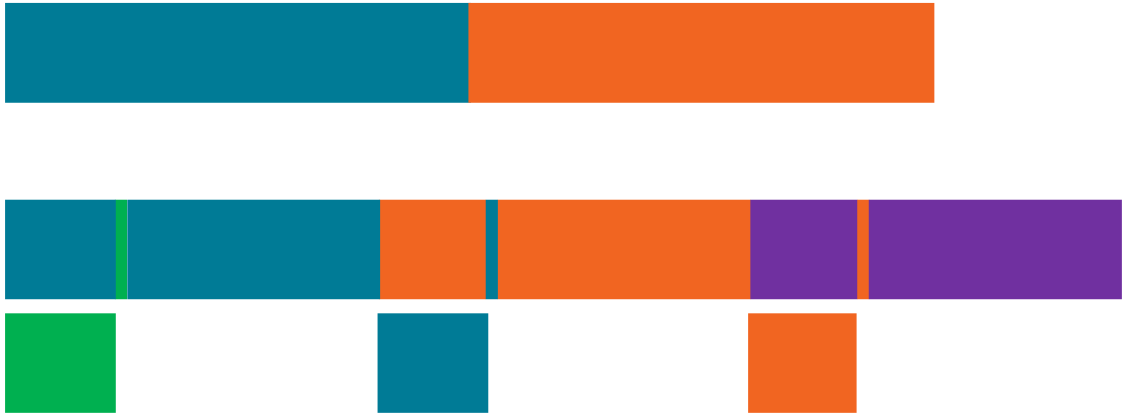
- Very engine specific
- In this particular case, there was a great **async compute** opportunity

OTHER OPTIMIZATION OPPORTUNITIES

- Very engine specific
- In this particular case, there was a great **async compute** opportunity

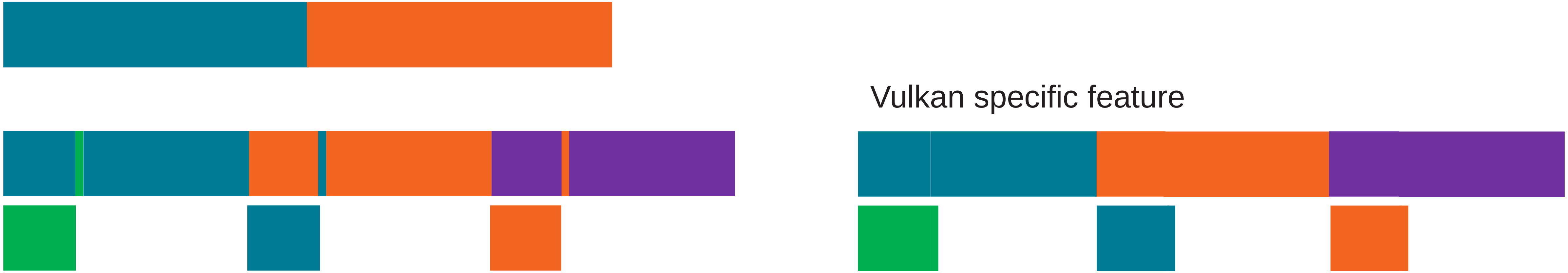
OTHER OPTIMIZATION OPPORTUNITIES

- Very engine specific
- In this particular case, there was a great **async compute** opportunity



OTHER OPTIMIZATION OPPORTUNITIES

- Very engine specific
- In this particular case, there was a great **async compute** opportunity



Vulkan specific feature

AGENDA

- DCC – Delta Color Compression
- Barriers 🤓 and other synchronization hassles
- Other small things
- Q&A

AGENDA

OR THE PREVIOUSLY MENTIONED CHECKLIST

- DCC – Delta Color Compression
- Barriers 🤓 and other synchronization hassles
- Other small things
- Q&A

AGENDA

OR THE PREVIOUSLY MENTIONED CHECKLIST

- DCC – Delta Color Compression
- Barriers 🤓 and other synchronization hassles
- Other small things
- Q&A

+ async compute opportunity

DCC – DELTA COLOR COMPRESSION

- What is DCC?
- Why do we want it

-> Performance impact

- How to enable DCC?

-> the journey of enabling DCC for this game 🏞

WHAT IS DCC?

- DCC – Delta Color Compression
- Takes advantage of the fact that render targets tend to store slowly varying data
    - E.g. a blue sky will have little variance between the pixels

WHAT IS DCC?

- DCC – Delta Color Compression
- Takes advantage of the fact that render targets tend to store slowly varying data
  - E.g. a blue sky will have little variance between the pixels

WHAT IS DCC?

- DCC – Delta Color Compression
- Takes advantage of the fact that render targets tend to store slowly varying data
  - E.g. a blue sky will have little variance between the pixels

- Stores whole blocks – one value is stored with full precision, rest is stored as delta
- It‘s lossless

WHY DO WE WANT DCC?

- It‘s a bandwidth saver
- Take a special emphasis in enabling DCC for the G-buffer render targets
  - they usually benefit a lot from bandwidth savings

WHY DO WE WANT DCC?

- It‘s a bandwidth saver
- Take a special emphasis in enabling DCC for the G-buffer render targets
  - they usually benefit a lot from bandwidth savings

- How much?
- Depends on workload and varies between graphics card
- But in this particular game title, we observed speed-ups on all tested AMD GPUs, ranging between
  ~5 – 10%

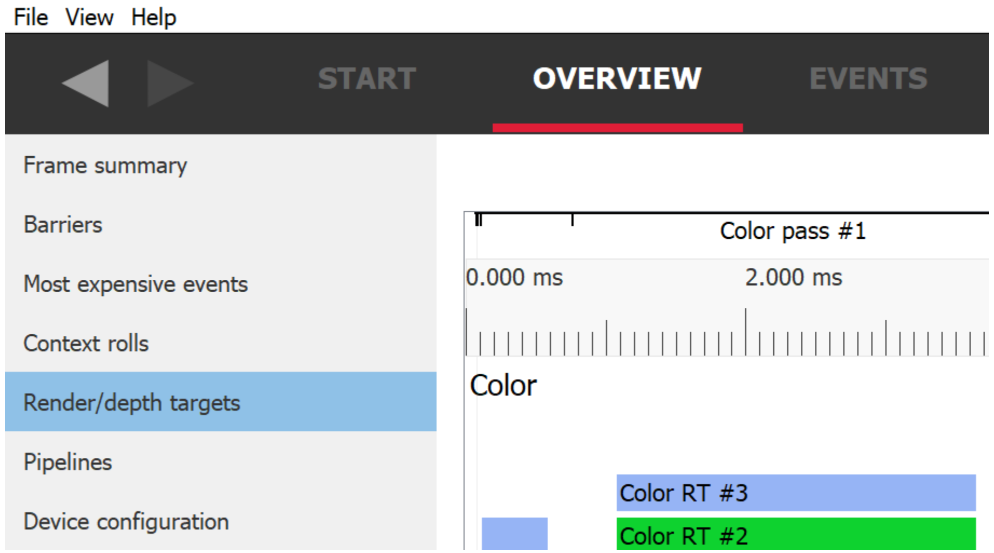
WHY DO WE WANT DCC?

- It‘s a bandwidth saver
- Take a special emphasis in enabling DCC for the G-buffer render targets
  - they usually benefit a lot from bandwidth savings

- How much?
- Depends on workload and varies between graphics card
- But in this particular game title, we observed speed-ups on all tested AMD GPUs, ranging between
  ~5 – 10%

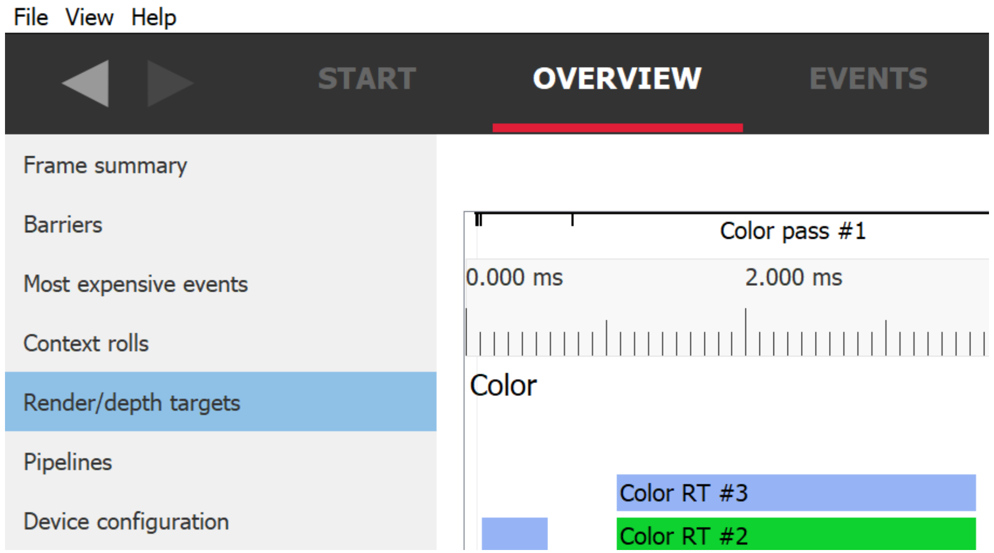
HOW DO I KNOW DCC IS ENABLED?

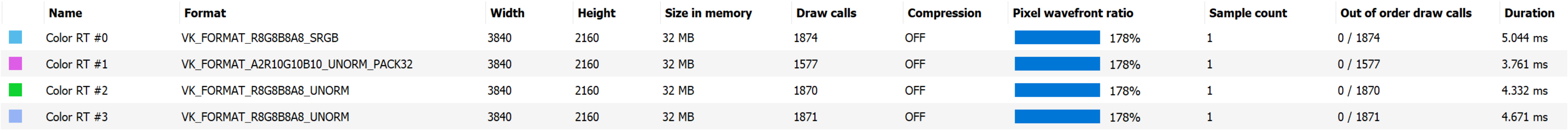
Use Radeon GPU Profiler (RGP):



HOW DO I KNOW DCC IS ENABLED?

Use Radeon GPU Profiler (RGP):

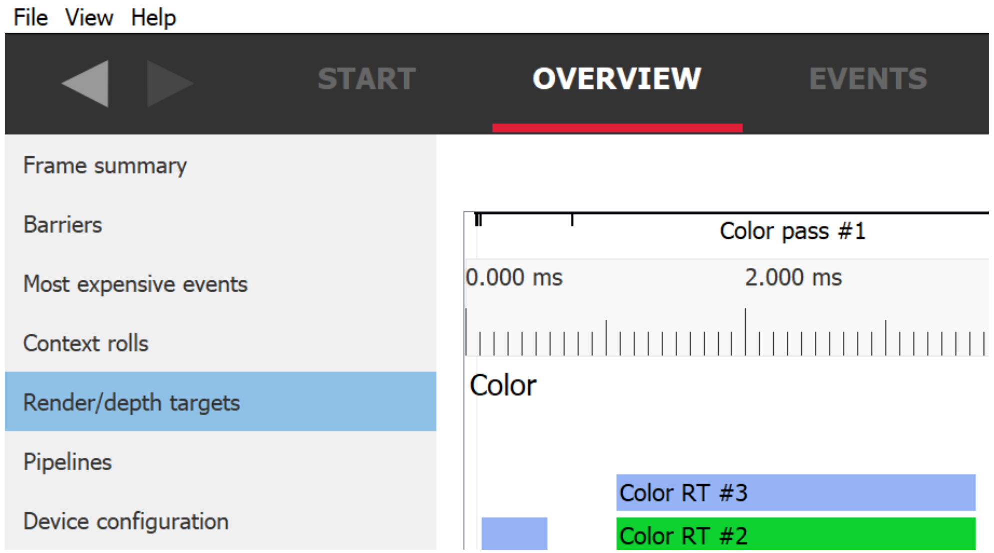


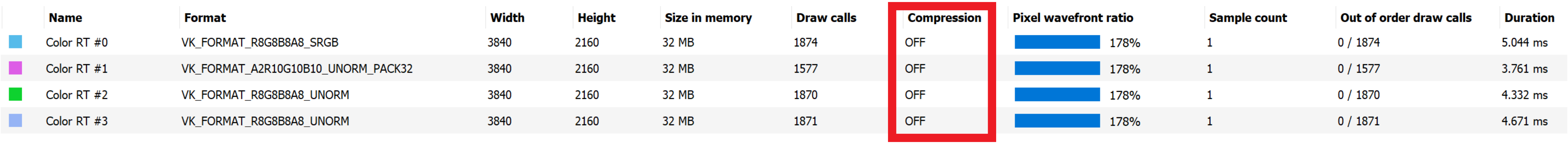


	Name	Format	Width	Height	Size in memory	Draw calls	Compression	Pixel wavefront ratio	Sample count	Out of order draw calls	Duration
🟦	Color RT #0	VK_FORMAT_R8G8B8A8_SRGB	3840	2160	32 MB	1874	OFF	178%	1	0 / 1874	5.044 ms
🟪	Color RT #1	VK_FORMAT_A2R10G10B10_UNORM_PACK32	3840	2160	32 MB	1577	OFF	178%	1	0 / 1577	3.761 ms
🟩	Color RT #2	VK_FORMAT_R8G8B8A8_UNORM	3840	2160	32 MB	1870	OFF	178%	1	0 / 1870	4.332 ms
🟦	Color RT #3	VK_FORMAT_R8G8B8A8_UNORM	3840	2160	32 MB	1871	OFF	178%	1	0 / 1871	4.671 ms

HOW DO I KNOW DCC IS ENABLED?

Use Radeon GPU Profiler (RGP):

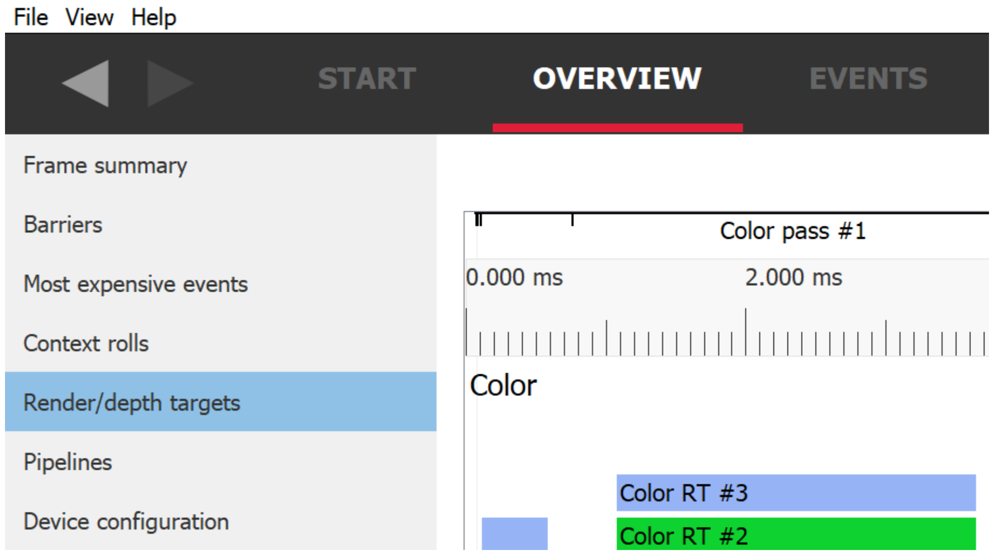


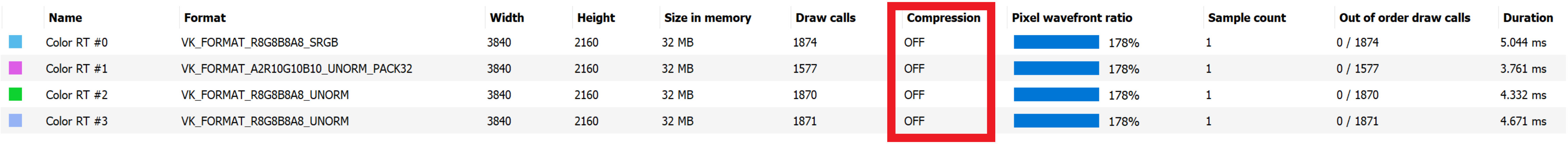


	Name	Format	Width	Height	Size in memory	Draw calls	Compression	Pixel wavefront ratio	Sample count	Out of order draw calls	Duration
🟦	Color RT #0	VK_FORMAT_R8G8B8A8_SRGB	3840	2160	32 MB	1874	OFF	178%	1	0 / 1874	5.044 ms
🟪	Color RT #1	VK_FORMAT_A2R10G10B10_UNORM_PACK32	3840	2160	32 MB	1577	OFF	178%	1	0 / 1577	3.761 ms
🟩	Color RT #2	VK_FORMAT_R8G8B8A8_UNORM	3840	2160	32 MB	1870	OFF	178%	1	0 / 1870	4.332 ms
🟦	Color RT #3	VK_FORMAT_R8G8B8A8_UNORM	3840	2160	32 MB	1871	OFF	178%	1	0 / 1871	4.671 ms

HOW DO I KNOW DCC IS ENABLED?

Use Radeon GPU Profiler (RGP):



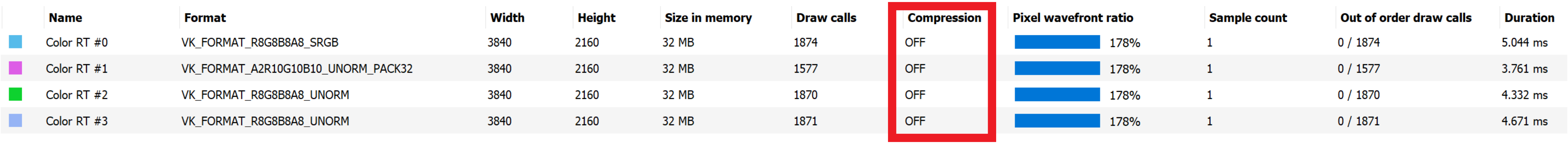


	Name	Format	Width	Height	Size in memory	Draw calls	Compression	Pixel wavefront ratio	Sample count	Out of order draw calls	Duration
🟦	Color RT #0	VK_FORMAT_R8G8B8A8_SRGB	3840	2160	32 MB	1874	OFF	178%	1	0 / 1874	5.044 ms
🟪	Color RT #1	VK_FORMAT_A2R10G10B10_UNORM_PACK32	3840	2160	32 MB	1577	OFF	178%	1	0 / 1577	3.761 ms
🟩	Color RT #2	VK_FORMAT_R8G8B8A8_UNORM	3840	2160	32 MB	1870	OFF	178%	1	0 / 1870	4.332 ms
🟦	Color RT #3	VK_FORMAT_R8G8B8A8_UNORM	3840	2160	32 MB	1871	OFF	178%	1	0 / 1871	4.671 ms

☹

DCC IS TURNED OFF – WHY?

- You can check the format
  - Float format
  - Integer format



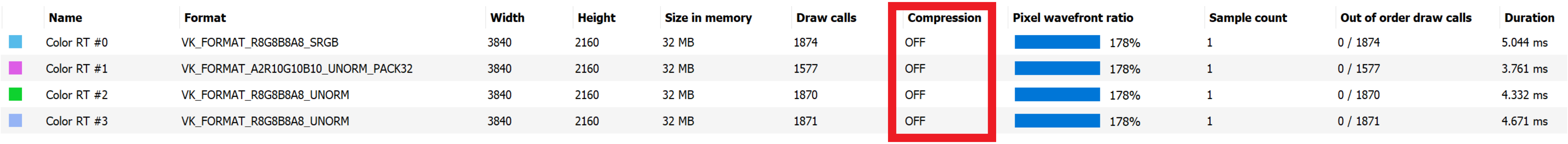
	Name	Format	Width	Height	Size in memory	Draw calls	Compression	Pixel wavefront ratio	Sample count	Out of order draw calls	Duration
🟦	Color RT #0	VK_FORMAT_R8G8B8A8_SRGB	3840	2160	32 MB	1874	OFF	178%	1	0 / 1874	5.044 ms
🟪	Color RT #1	VK_FORMAT_A2R10G10B10_UNORM_PACK32	3840	2160	32 MB	1577	OFF	178%	1	0 / 1577	3.761 ms
🟩	Color RT #2	VK_FORMAT_R8G8B8A8_UNORM	3840	2160	32 MB	1870	OFF	178%	1	0 / 1870	4.332 ms
🟦	Color RT #3	VK_FORMAT_R8G8B8A8_UNORM	3840	2160	32 MB	1871	OFF	178%	1	0 / 1871	4.671 ms

☹

DCC IS TURNED OFF – WHY?

- You can check the format
  - Float format
  - Integer format

- All of the below are supported

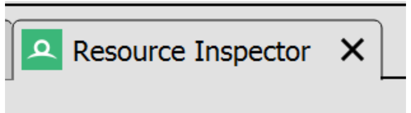


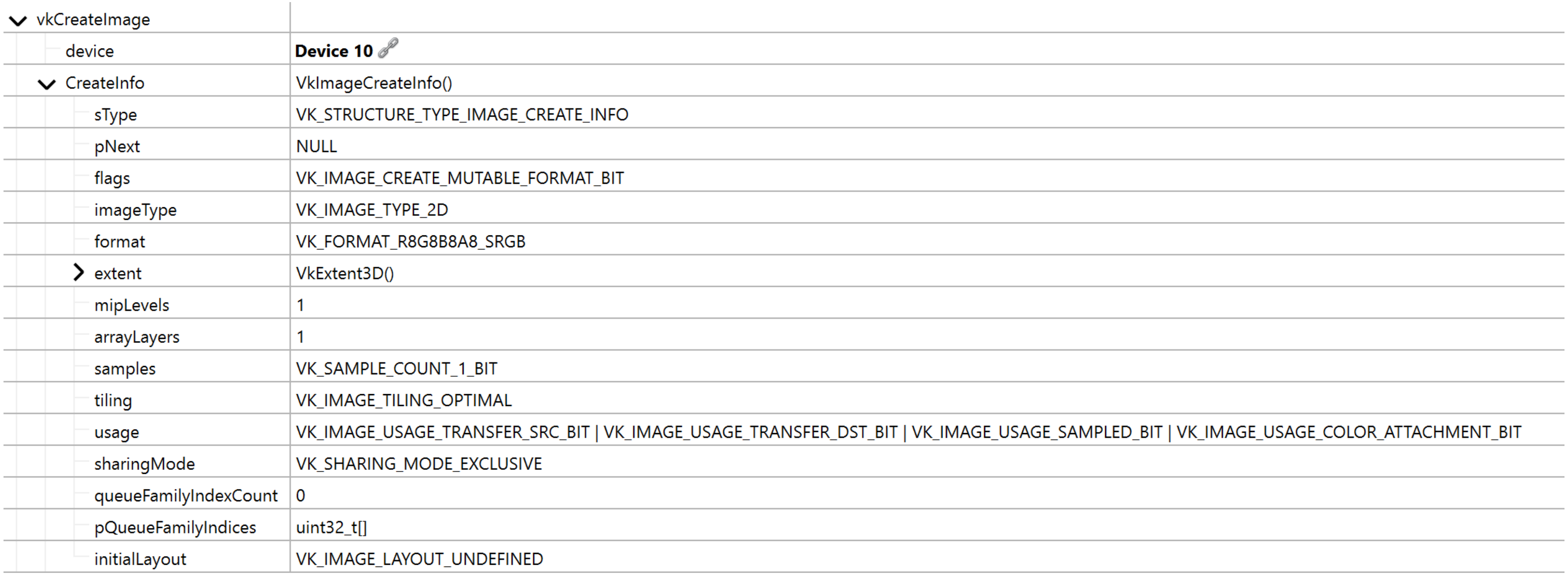
	Name	Format	Width	Height	Size in memory	Draw calls	Compression	Pixel wavefront ratio	Sample count	Out of order draw calls	Duration
🟦	Color RT #0	VK_FORMAT_R8G8B8A8_SRGB	3840	2160	32 MB	1874	OFF	178%	1	0 / 1874	5.044 ms
🟪	Color RT #1	VK_FORMAT_A2R10G10B10_UNORM_PACK32	3840	2160	32 MB	1577	OFF	178%	1	0 / 1577	3.761 ms
🟩	Color RT #2	VK_FORMAT_R8G8B8A8_UNORM	3840	2160	32 MB	1870	OFF	178%	1	0 / 1870	4.332 ms
🟦	Color RT #3	VK_FORMAT_R8G8B8A8_UNORM	3840	2160	32 MB	1871	OFF	178%	1	0 / 1871	4.671 ms

☹

DCC IS TURNED OFF – WHY?

Retrieve some more resource details from RenderDoc:

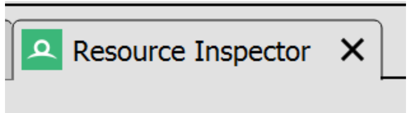


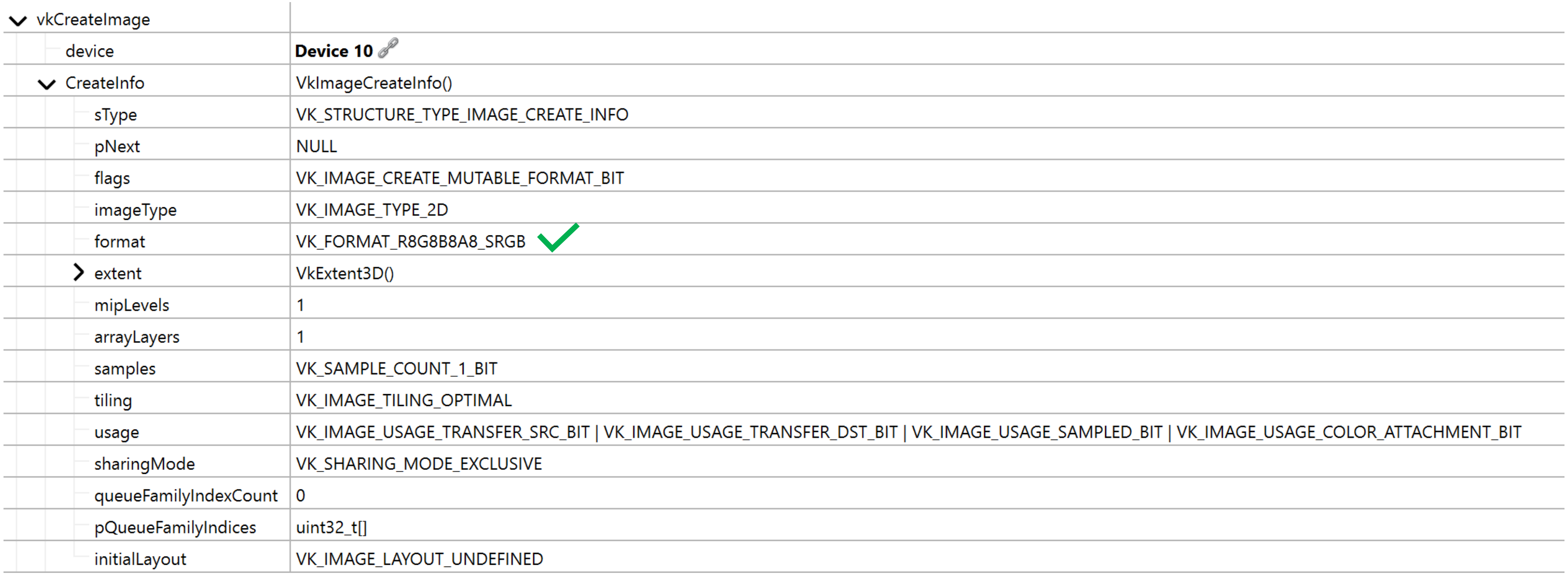


vkCreateImage	
device	**Device 10**
CreateInfo	VkImageCreateInfo()
sType	VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO
pNext	NULL
flags	VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
imageType	VK_IMAGE_TYPE_2D
format	VK_FORMAT_R8G8B8A8_SRGB
extent	VkExtent3D()
mipLevels	1
arrayLayers	1
samples	VK_SAMPLE_COUNT_1_BIT
tiling	VK_IMAGE_TILING_OPTIMAL
usage	VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_USAGE_SAMPLED_BIT | VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
sharingMode	VK_SHARING_MODE_EXCLUSIVE
queueFamilyIndexCount	0
pQueueFamilyIndices	uint32_t[]
initialLayout	VK_IMAGE_LAYOUT_UNDEFINED

DCC IS TURNED OFF – WHY?

Retrieve some more resource details from RenderDoc:

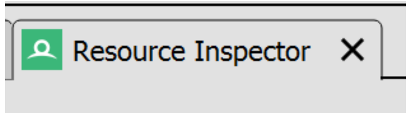


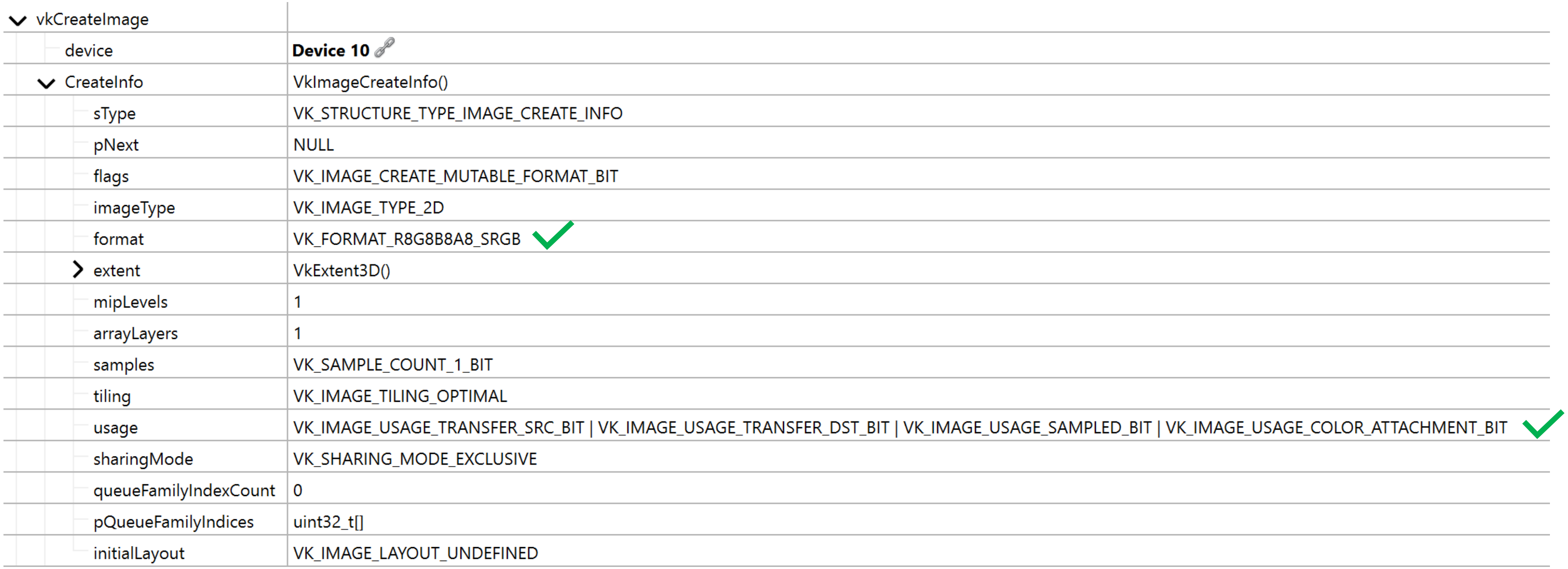


vkCreateImage	
device	**Device 10**
CreateInfo	VkImageCreateInfo()
sType	VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO
pNext	NULL
flags	VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
imageType	VK_IMAGE_TYPE_2D
format	VK_FORMAT_R8G8B8A8_SRGB ✓
extent	VkExtent3D()
mipLevels	1
arrayLayers	1
samples	VK_SAMPLE_COUNT_1_BIT
tiling	VK_IMAGE_TILING_OPTIMAL
usage	VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_USAGE_SAMPLED_BIT | VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
sharingMode	VK_SHARING_MODE_EXCLUSIVE
queueFamilyIndexCount	0
pQueueFamilyIndices	uint32_t[]
initialLayout	VK_IMAGE_LAYOUT_UNDEFINED

DCC IS TURNED OFF – WHY?

Retrieve some more resource details from RenderDoc:

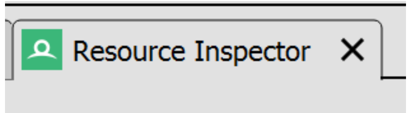


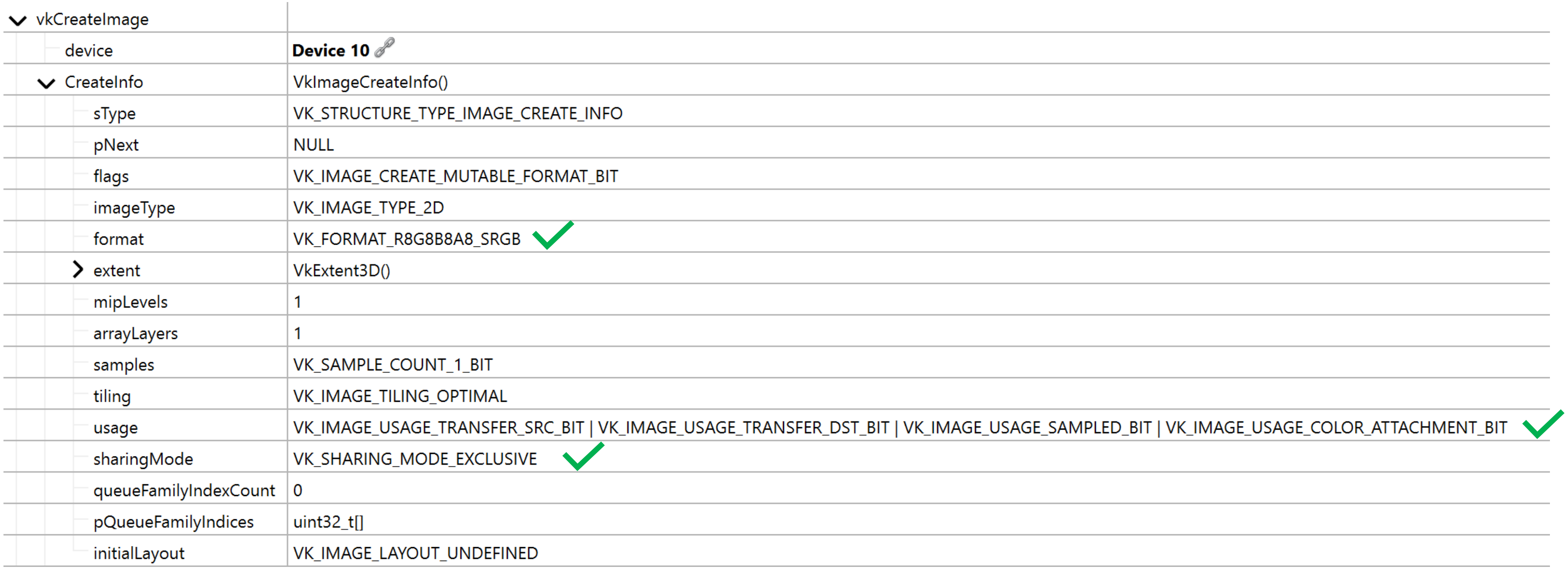


vkCreateImage	
device	**Device 10**
CreateInfo	VkImageCreateInfo()
sType	VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO
pNext	NULL
flags	VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
imageType	VK_IMAGE_TYPE_2D
format	VK_FORMAT_R8G8B8A8_SRGB ✓
extent	VkExtent3D()
mipLevels	1
arrayLayers	1
samples	VK_SAMPLE_COUNT_1_BIT
tiling	VK_IMAGE_TILING_OPTIMAL
usage	VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_USAGE_SAMPLED_BIT | VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT ✓
sharingMode	VK_SHARING_MODE_EXCLUSIVE
queueFamilyIndexCount	0
pQueueFamilyIndices	uint32_t[]
initialLayout	VK_IMAGE_LAYOUT_UNDEFINED

DCC IS TURNED OFF – WHY?

Retrieve some more resource details from RenderDoc:

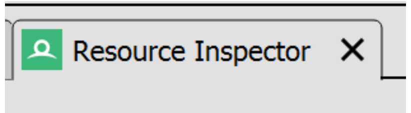


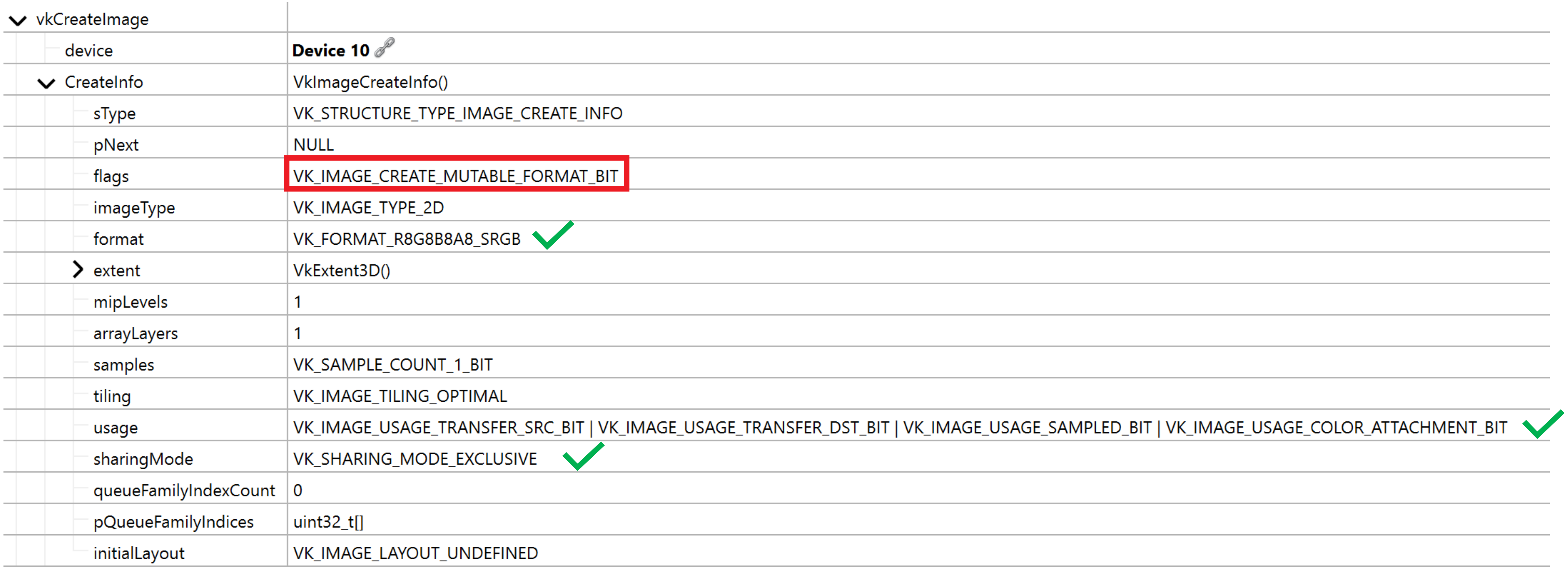


vkCreateImage	
device	**Device 10**
CreateInfo	VkImageCreateInfo()
sType	VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO
pNext	NULL
flags	VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
imageType	VK_IMAGE_TYPE_2D
format	VK_FORMAT_R8G8B8A8_SRGB ✓
extent	VkExtent3D()
mipLevels	1
arrayLayers	1
samples	VK_SAMPLE_COUNT_1_BIT
tiling	VK_IMAGE_TILING_OPTIMAL
usage	VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_USAGE_SAMPLED_BIT | VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT ✓
sharingMode	VK_SHARING_MODE_EXCLUSIVE ✓
queueFamilyIndexCount	0
pQueueFamilyIndices	uint32_t[]
initialLayout	VK_IMAGE_LAYOUT_UNDEFINED

DCC IS TURNED OFF – WHY?

Retrieve some more resource details from RenderDoc:

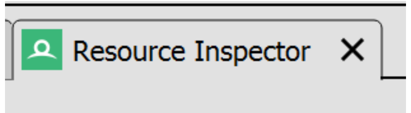


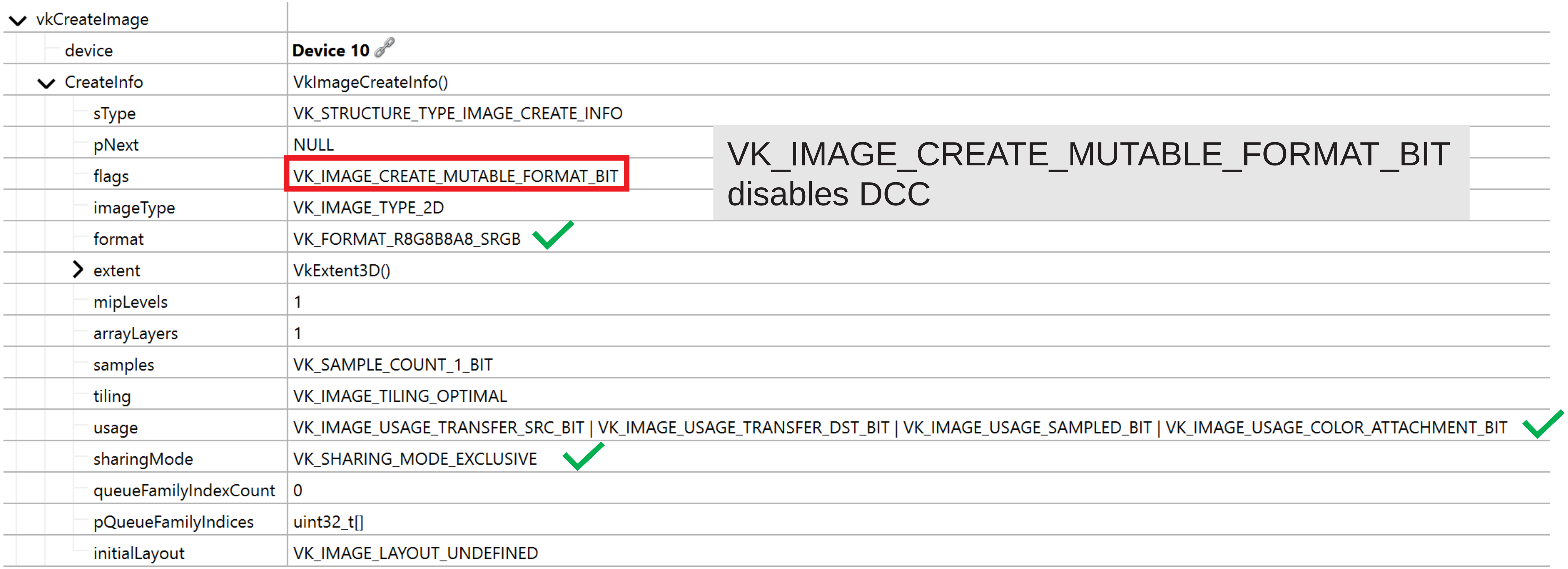


vkCreateImage	
device	**Device 10**
CreateInfo	VkImageCreateInfo()
sType	VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO
pNext	NULL
flags	VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
imageType	VK_IMAGE_TYPE_2D
format	VK_FORMAT_R8G8B8A8_SRGB ✓
extent	VkExtent3D()
mipLevels	1
arrayLayers	1
samples	VK_SAMPLE_COUNT_1_BIT
tiling	VK_IMAGE_TILING_OPTIMAL
usage	VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_USAGE_SAMPLED_BIT | VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT ✓
sharingMode	VK_SHARING_MODE_EXCLUSIVE ✓
queueFamilyIndexCount	0
pQueueFamilyIndices	uint32_t[]
initialLayout	VK_IMAGE_LAYOUT_UNDEFINED

DCC IS TURNED OFF – WHY?

Retrieve some more resource details from RenderDoc:



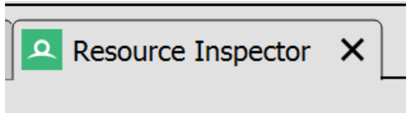


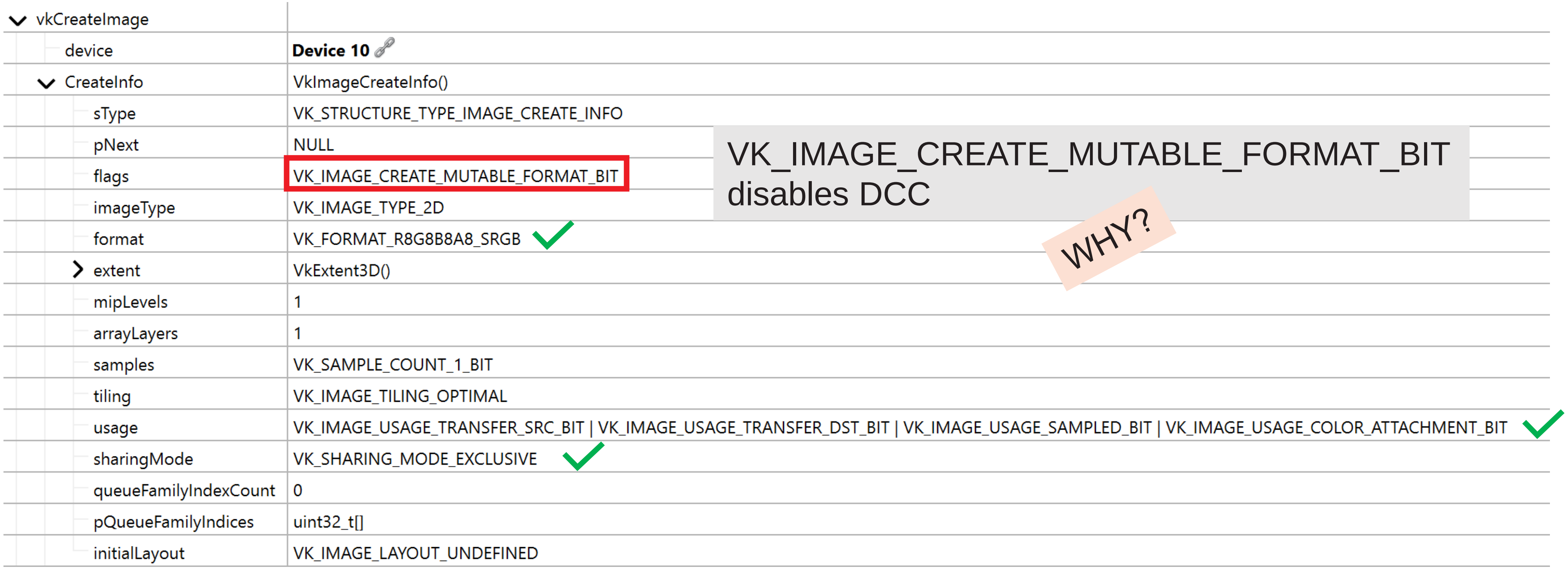
vkCreateImage	
device	**Device 10**
CreateInfo	VkImageCreateInfo()
sType	VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO
pNext	NULL
flags	VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
imageType	VK_IMAGE_TYPE_2D
format	VK_FORMAT_R8G8B8A8_SRGB ✓
extent	VkExtent3D()
mipLevels	1
arrayLayers	1
samples	VK_SAMPLE_COUNT_1_BIT
tiling	VK_IMAGE_TILING_OPTIMAL
usage	VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_USAGE_SAMPLED_BIT | VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT ✓
sharingMode	VK_SHARING_MODE_EXCLUSIVE ✓
queueFamilyIndexCount	0
pQueueFamilyIndices	uint32_t[]
initialLayout	VK_IMAGE_LAYOUT_UNDEFINED

VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT disables DCC

DCC IS TURNED OFF – WHY?

Retrieve some more resource details from RenderDoc:





vkCreateImage	
device	**Device 10**
CreateInfo	VkImageCreateInfo()
sType	VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO
pNext	NULL
flags	VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
imageType	VK_IMAGE_TYPE_2D
format	VK_FORMAT_R8G8B8A8_SRGB ✓
extent	VkExtent3D()
mipLevels	1
arrayLayers	1
samples	VK_SAMPLE_COUNT_1_BIT
tiling	VK_IMAGE_TILING_OPTIMAL
usage	VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_USAGE_SAMPLED_BIT | VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT ✓
sharingMode	VK_SHARING_MODE_EXCLUSIVE ✓
queueFamilyIndexCount	0
pQueueFamilyIndices	uint32_t[]
initialLayout	VK_IMAGE_LAYOUT_UNDEFINED

VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT disables DCC

WHY?

VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT

DCC only works for float **XOR** integer formats

-> R16G16B16A16_SFLOAT, DCC is supported

-> R16G16B16A16_UNORM, DCC is supported

Etc.

VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT

DCC only works for float **XOR** integer formats

-> R16G16B16A16_SFLOAT, DCC is supported

-> R16G16B16A16_UNORM, DCC is supported

Etc.

How does the driver know the format of the image?

```
VkImageCreateInfo imageCreateInfo = {};
imageCreateInfo.format = VK_FORMAT_R8G8B8A8_SRGB;
```

VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT

DCC only works for float **XOR** integer formats

-> R16G16B16A16_SFLOAT, DCC is supported

-> R16G16B16A16_UNORM, DCC is supported

Etc.

How does the driver know the format of the image?

```
VkImageCreateInfo imageCreateInfo = {};
imageCreateInfo.format = VK_FORMAT_R8G8B8A8_SRGB;
```

What happens when the mutable bit is set?

VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT

```
VkImageCreateInfo imageCreateInfo = {};
imageCreateInfo.format = VK_FORMAT_R8G8B8A8_SRGB;
imageCreateInfo.flags = VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT;
```

Spec:

> “VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT specifies that the image can be used to create a VkImageView with a **different format** from the image.”

VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT

```
VkImageCreateInfo imageCreateInfo = {};
imageCreateInfo.format = VK_FORMAT_R8G8B8A8_SRGB;
imageCreateInfo.flags = VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT;
```

Spec:

> “VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT specifies that the image can be used to create a VkImageView with a **different format** from the image.”

-> The driver can‘t rely on the format information from the VkImageCreateInfo struct anymore

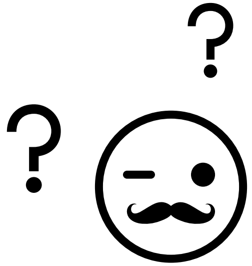
VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT

-> The driver can‘t rely on the format information from the VkImageCreateInfo struct anymore

For **float XOR integer**, the driver needs to distinguish between:

1. Image views with integer **AND** float formats are used on the image -> DCC must be **disabled**
2. Unsupported format is used -> DCC must be **disabled**
3. **Only integer** formats are used, e.g. UNORM and SRGB -> DCC can be **enabled**
4. **Only float** formats are used -> DCC can be **enabled**

The driver can’t know if enabling DCC is safe by simply looking at the mutable bit.



VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT

-> The driver can‘t rely on the format information from the VkImageCreateInfo struct anymore

For **float XOR integer**, the driver needs to distinguish between:

1. Image views with integer **AND** float formats are used on the image -> DCC must be **disabled**
2. Unsupported format is used -> DCC must be **disabled**
3. **Only integer** formats are used, e.g. UNORM and SRGB -> DCC can be **enabled**
4. **Only float** formats are used -> DCC can be **enabled**

The driver can’t know if enabling DCC is safe by simply looking at the mutable bit.

VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT

-> The driver can‘t rely on the format information from the VkImageCreateInfo struct anymore

For **float XOR integer**, the driver needs to distinguish between:

1. Image views with integer **AND** float formats are used on the image -> DCC must be **disabled**
2. Unsupported format is used -> DCC must be **disabled**
3. **Only integer** formats are used, e.g. UNORM and SRGB -> DCC can be **enabled**
4. **Only float** formats are used -> DCC can be **enabled**

The driver can’t know if enabling DCC is safe by simply looking at the mutable bit.

VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT

The driver can’t know if enabling DCC is safe by simply looking at the mutable bit.

-> provide additional information by using

VK_KHR_image_format_list

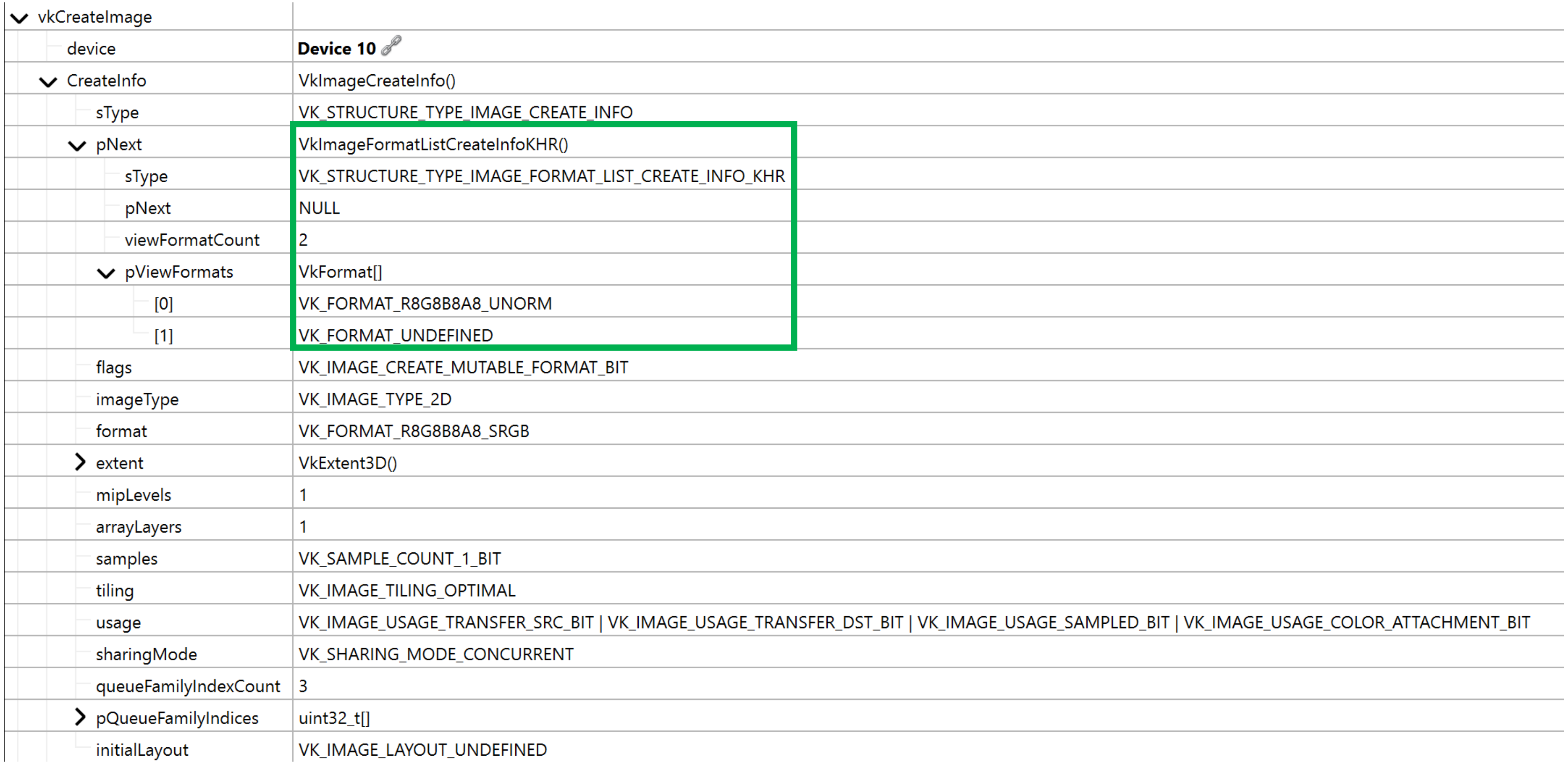
```
typedef struct VkImageFormatListCreateInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           viewFormatCount;
    const VkFormat*    pViewFormats;
} VkImageFormatListCreateInfoKHR;
```

VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT

```
VkImageFormatListCreateInfoKHR imageFormatList = {};
imageFormatList.sType = VK_STRUCTURE_TYPE_IMAGE_FORMAT_LIST_CREATE_INFO_KHR
imageFormatList.pNext = … ;
imageFormatList.viewFormatCount = formatCount;
imageFormatList.pViewFormats = formats; // array of VkFormat

VkImageCreateInfo imageCreateInfo = {};
imageCreateInfo.format = VK_FORMAT_R8G8B8A8_SRGB;
imageCreateInfo.flags = VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT;
imageCreateInfo.pNext = &imageFormatList;
…
```

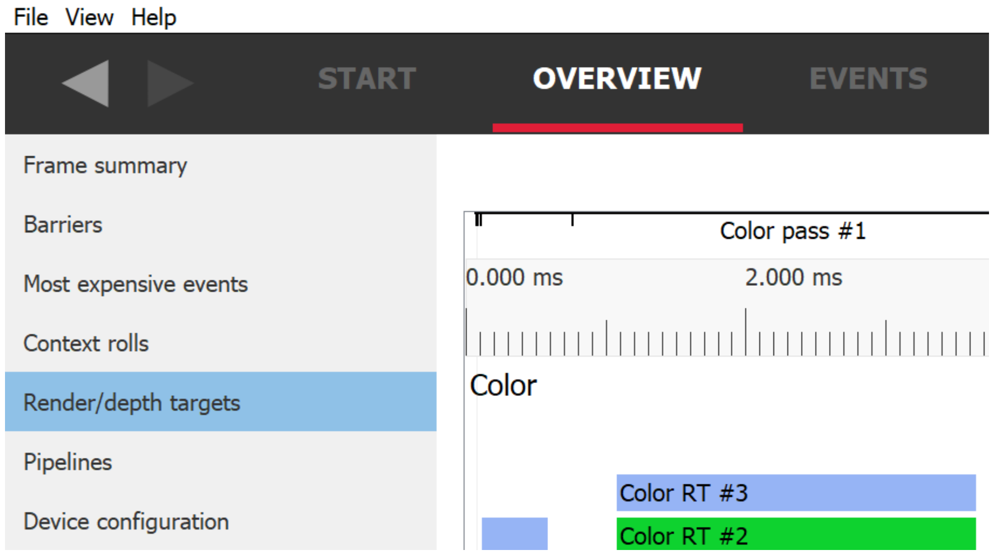
VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT

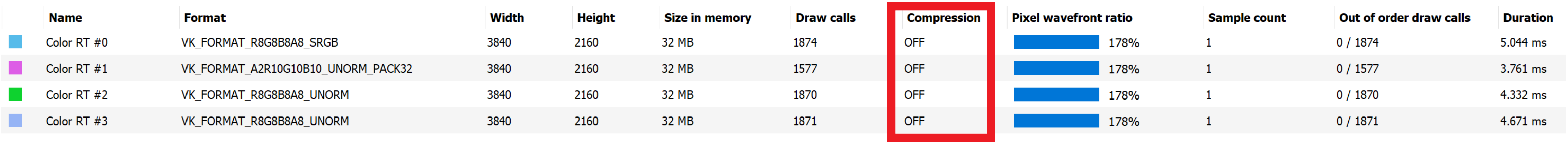


vkCreateImage	
device	**Device 10**
CreateInfo	VkImageCreateInfo()
sType	VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO
pNext	VkImageFormatListCreateInfoKHR()
sType	VK_STRUCTURE_TYPE_IMAGE_FORMAT_LIST_CREATE_INFO_KHR
pNext	NULL
viewFormatCount	2
pViewFormats	VkFormat[]
[0]	VK_FORMAT_R8G8B8A8_UNORM
[1]	VK_FORMAT_UNDEFINED
flags	VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
imageType	VK_IMAGE_TYPE_2D
format	VK_FORMAT_R8G8B8A8_SRGB
extent	VkExtent3D()
mipLevels	1
arrayLayers	1
samples	VK_SAMPLE_COUNT_1_BIT
tiling	VK_IMAGE_TILING_OPTIMAL
usage	VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_USAGE_SAMPLED_BIT | VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
sharingMode	VK_SHARING_MODE_CONCURRENT
queueFamilyIndexCount	3
pQueueFamilyIndices	uint32_t[]
initialLayout	VK_IMAGE_LAYOUT_UNDEFINED

DOUBLE-CHECK IF THE CHANGE HAD THE INTENDED EFFECT …

Use Radeon GPU Profiler (RGP):



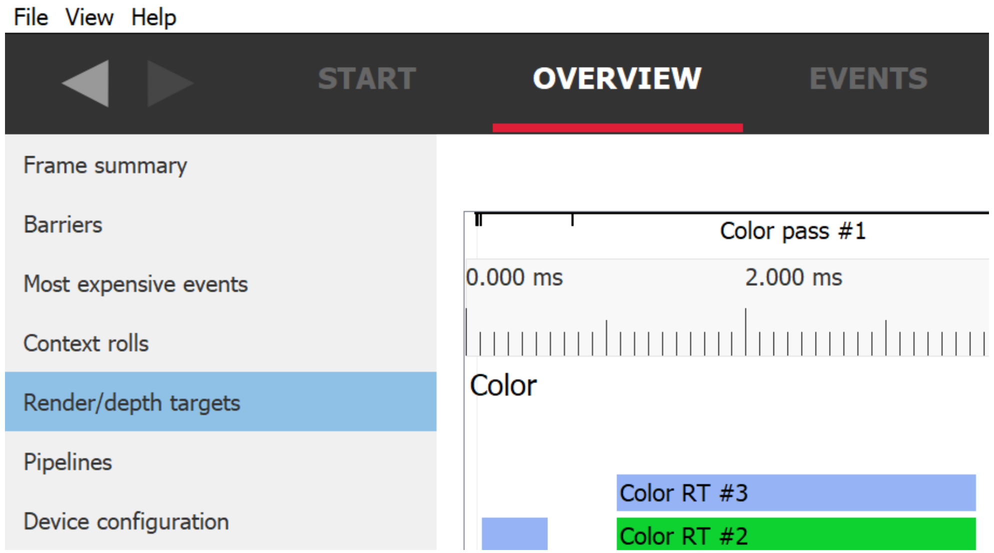


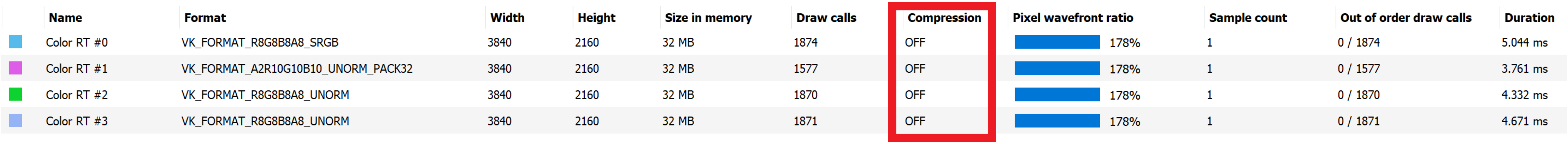
	Name	Format	Width	Height	Size in memory	Draw calls	Compression	Pixel wavefront ratio	Sample count	Out of order draw calls	Duration
🟦	Color RT #0	VK_FORMAT_R8G8B8A8_SRGB	3840	2160	32 MB	1874	OFF	178%	1	0 / 1874	5.044 ms
🟪	Color RT #1	VK_FORMAT_A2R10G10B10_UNORM_PACK32	3840	2160	32 MB	1577	OFF	178%	1	0 / 1577	3.761 ms
🟩	Color RT #2	VK_FORMAT_R8G8B8A8_UNORM	3840	2160	32 MB	1870	OFF	178%	1	0 / 1870	4.332 ms
🟦	Color RT #3	VK_FORMAT_R8G8B8A8_UNORM	3840	2160	32 MB	1871	OFF	178%	1	0 / 1871	4.671 ms

DOUBLE-CHECK IF THE CHANGE HAD THE INTENDED EFFECT …

Use Radeon GPU Profiler (RGP):

It did not …





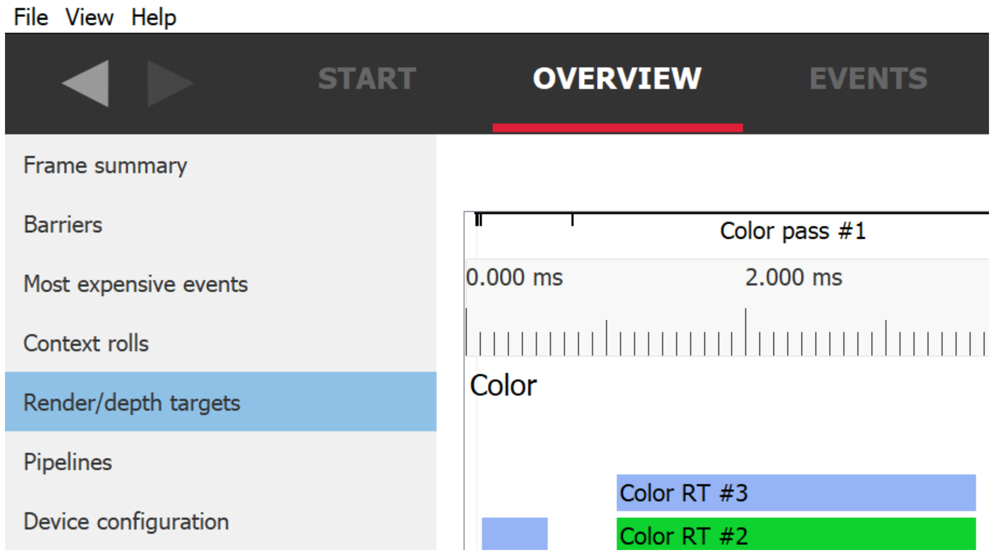
	Name	Format	Width	Height	Size in memory	Draw calls	Compression	Pixel wavefront ratio	Sample count	Out of order draw calls	Duration
🟦	Color RT #0	VK_FORMAT_R8G8B8A8_SRGB	3840	2160	32 MB	1874	OFF	178%	1	0 / 1874	5.044 ms
🟪	Color RT #1	VK_FORMAT_A2R10G10B10_UNORM_PACK32	3840	2160	32 MB	1577	OFF	178%	1	0 / 1577	3.761 ms
🟩	Color RT #2	VK_FORMAT_R8G8B8A8_UNORM	3840	2160	32 MB	1870	OFF	178%	1	0 / 1870	4.332 ms
🟦	Color RT #3	VK_FORMAT_R8G8B8A8_UNORM	3840	2160	32 MB	1871	OFF	178%	1	0 / 1871	4.671 ms

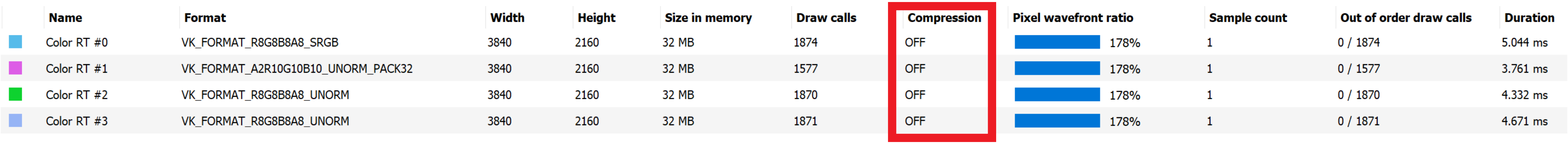
DOUBLE-CHECK IF THE CHANGE HAD THE INTENDED EFFECT …

Use Radeon GPU Profiler (RGP):

It did not … ☹

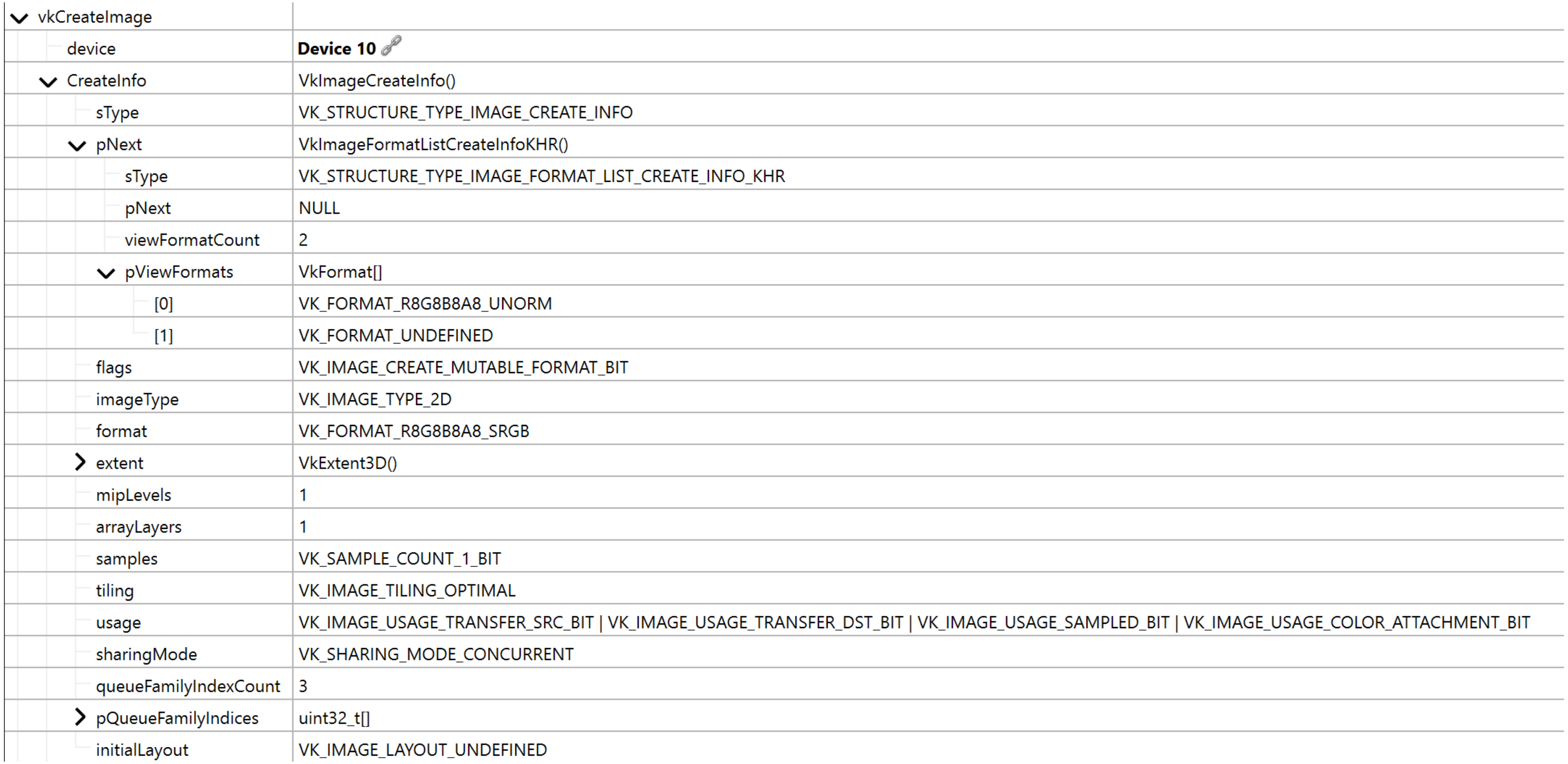
WHY?





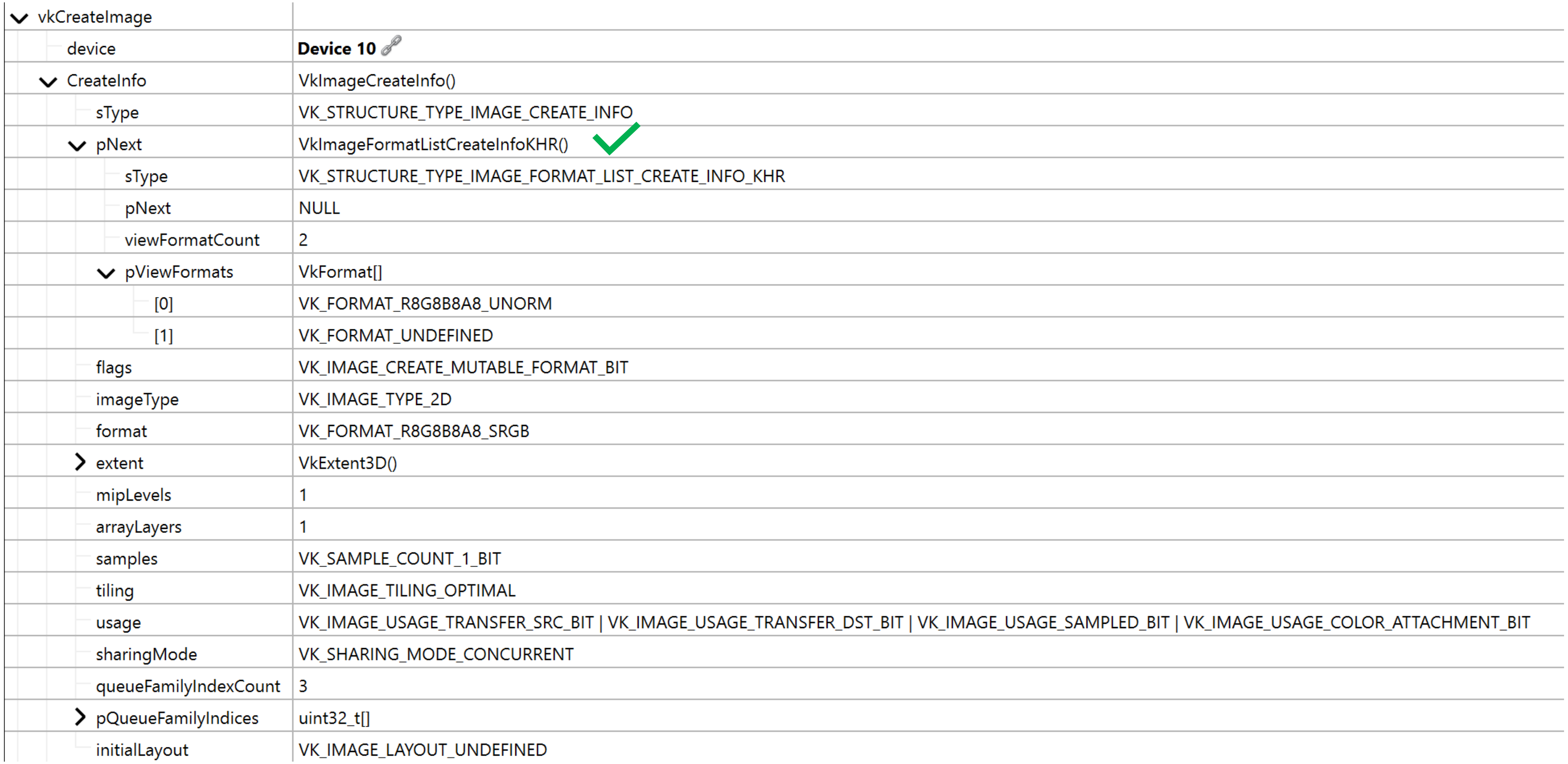
	Name	Format	Width	Height	Size in memory	Draw calls	Compression	Pixel wavefront ratio	Sample count	Out of order draw calls	Duration
🟦	Color RT #0	VK_FORMAT_R8G8B8A8_SRGB	3840	2160	32 MB	1874	OFF	178%	1	0 / 1874	5.044 ms
🟪	Color RT #1	VK_FORMAT_A2R10G10B10_UNORM_PACK32	3840	2160	32 MB	1577	OFF	178%	1	0 / 1577	3.761 ms
🟩	Color RT #2	VK_FORMAT_R8G8B8A8_UNORM	3840	2160	32 MB	1870	OFF	178%	1	0 / 1870	4.332 ms
🟦	Color RT #3	VK_FORMAT_R8G8B8A8_UNORM	3840	2160	32 MB	1871	OFF	178%	1	0 / 1871	4.671 ms

LET‘S EXAMINE CREATE IMAGE INFO AGAIN



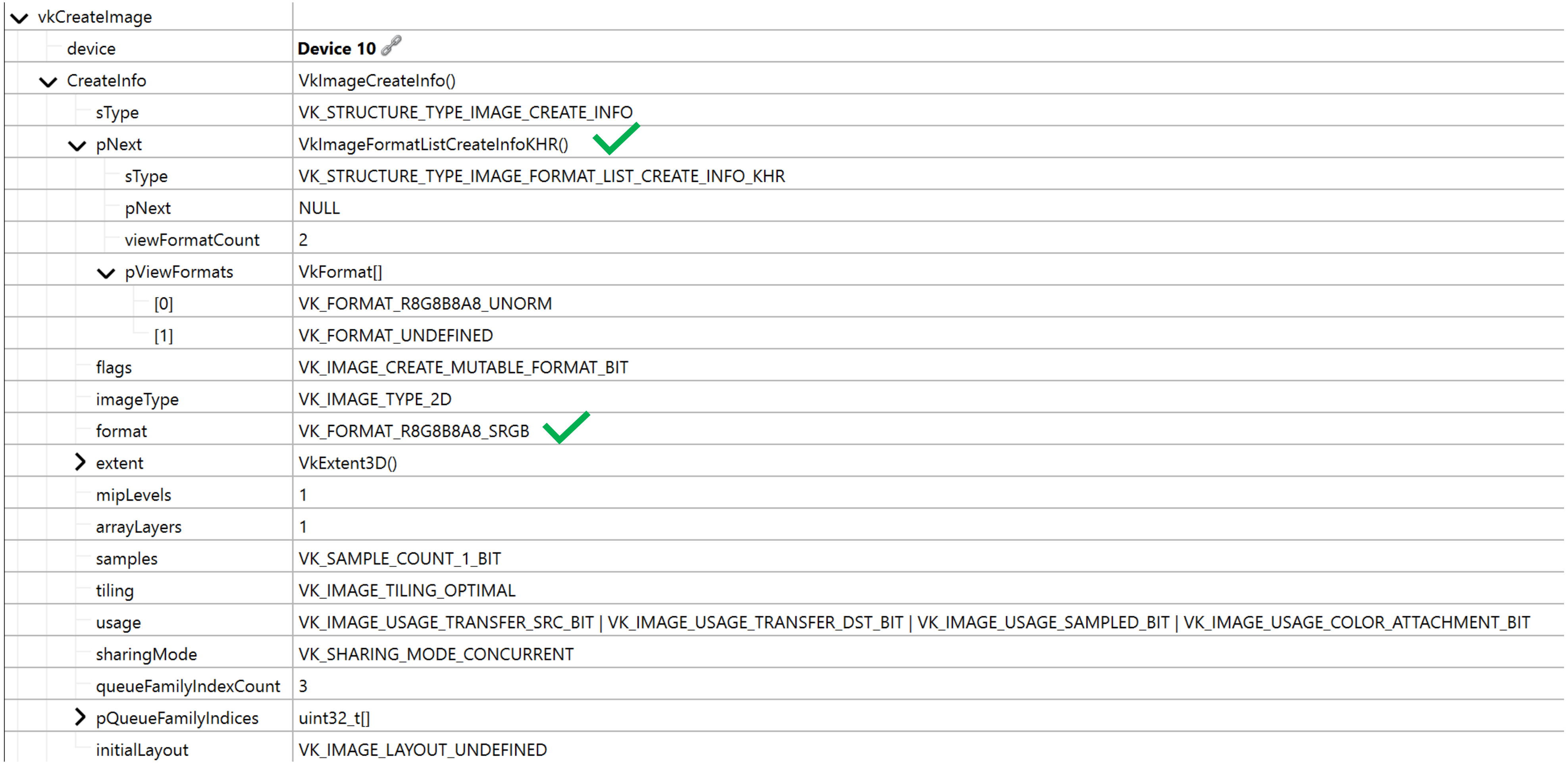
vkCreateImage	
device	**Device 10**
CreateInfo	VkImageCreateInfo()
sType	VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO
pNext	VkImageFormatListCreateInfoKHR()
sType	VK_STRUCTURE_TYPE_IMAGE_FORMAT_LIST_CREATE_INFO_KHR
pNext	NULL
viewFormatCount	2
pViewFormats	VkFormat[]
[0]	VK_FORMAT_R8G8B8A8_UNORM
[1]	VK_FORMAT_UNDEFINED
flags	VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
imageType	VK_IMAGE_TYPE_2D
format	VK_FORMAT_R8G8B8A8_SRGB
extent	VkExtent3D()
mipLevels	1
arrayLayers	1
samples	VK_SAMPLE_COUNT_1_BIT
tiling	VK_IMAGE_TILING_OPTIMAL
usage	VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_USAGE_SAMPLED_BIT | VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
sharingMode	VK_SHARING_MODE_CONCURRENT
queueFamilyIndexCount	3
pQueueFamilyIndices	uint32_t[]
initialLayout	VK_IMAGE_LAYOUT_UNDEFINED

LET‘S EXAMINE CREATE IMAGE INFO AGAIN



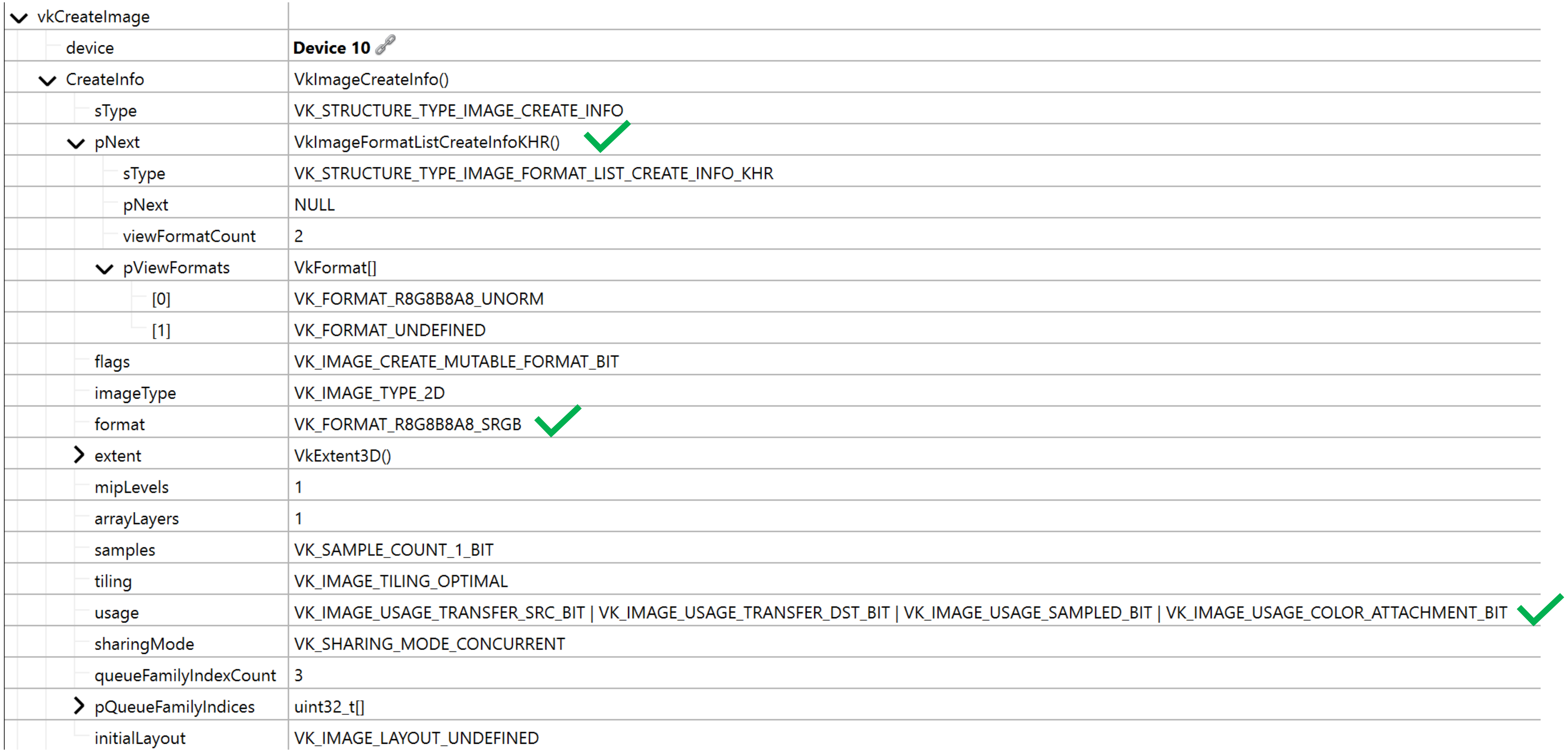
vkCreateImage	
device	**Device 10**
CreateInfo	VkImageCreateInfo()
sType	VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO
pNext	VkImageFormatListCreateInfoKHR() ✓
sType	VK_STRUCTURE_TYPE_IMAGE_FORMAT_LIST_CREATE_INFO_KHR
pNext	NULL
viewFormatCount	2
pViewFormats	VkFormat[]
[0]	VK_FORMAT_R8G8B8A8_UNORM
[1]	VK_FORMAT_UNDEFINED
flags	VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
imageType	VK_IMAGE_TYPE_2D
format	VK_FORMAT_R8G8B8A8_SRGB
extent	VkExtent3D()
mipLevels	1
arrayLayers	1
samples	VK_SAMPLE_COUNT_1_BIT
tiling	VK_IMAGE_TILING_OPTIMAL
usage	VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_USAGE_SAMPLED_BIT | VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
sharingMode	VK_SHARING_MODE_CONCURRENT
queueFamilyIndexCount	3
pQueueFamilyIndices	uint32_t[]
initialLayout	VK_IMAGE_LAYOUT_UNDEFINED

LET‘S EXAMINE CREATE IMAGE INFO AGAIN



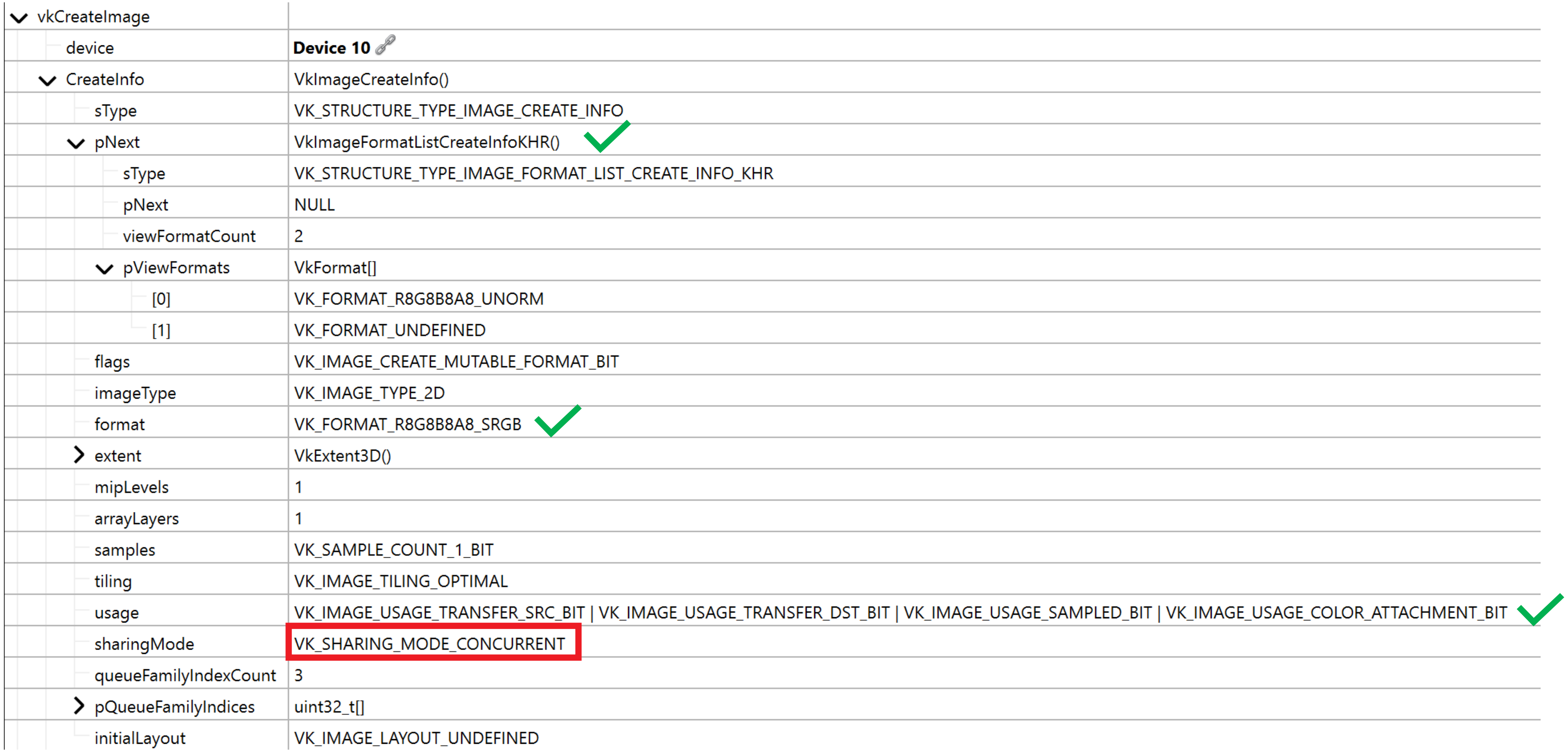
vkCreateImage	
device	**Device 10**
CreateInfo	VkImageCreateInfo()
sType	VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO
pNext	VkImageFormatListCreateInfoKHR() ✓
sType	VK_STRUCTURE_TYPE_IMAGE_FORMAT_LIST_CREATE_INFO_KHR
pNext	NULL
viewFormatCount	2
pViewFormats	VkFormat[]
[0]	VK_FORMAT_R8G8B8A8_UNORM
[1]	VK_FORMAT_UNDEFINED
flags	VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
imageType	VK_IMAGE_TYPE_2D
format	VK_FORMAT_R8G8B8A8_SRGB ✓
extent	VkExtent3D()
mipLevels	1
arrayLayers	1
samples	VK_SAMPLE_COUNT_1_BIT
tiling	VK_IMAGE_TILING_OPTIMAL
usage	VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_USAGE_SAMPLED_BIT | VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
sharingMode	VK_SHARING_MODE_CONCURRENT
queueFamilyIndexCount	3
pQueueFamilyIndices	uint32_t[]
initialLayout	VK_IMAGE_LAYOUT_UNDEFINED

LET‘S EXAMINE CREATE IMAGE INFO AGAIN



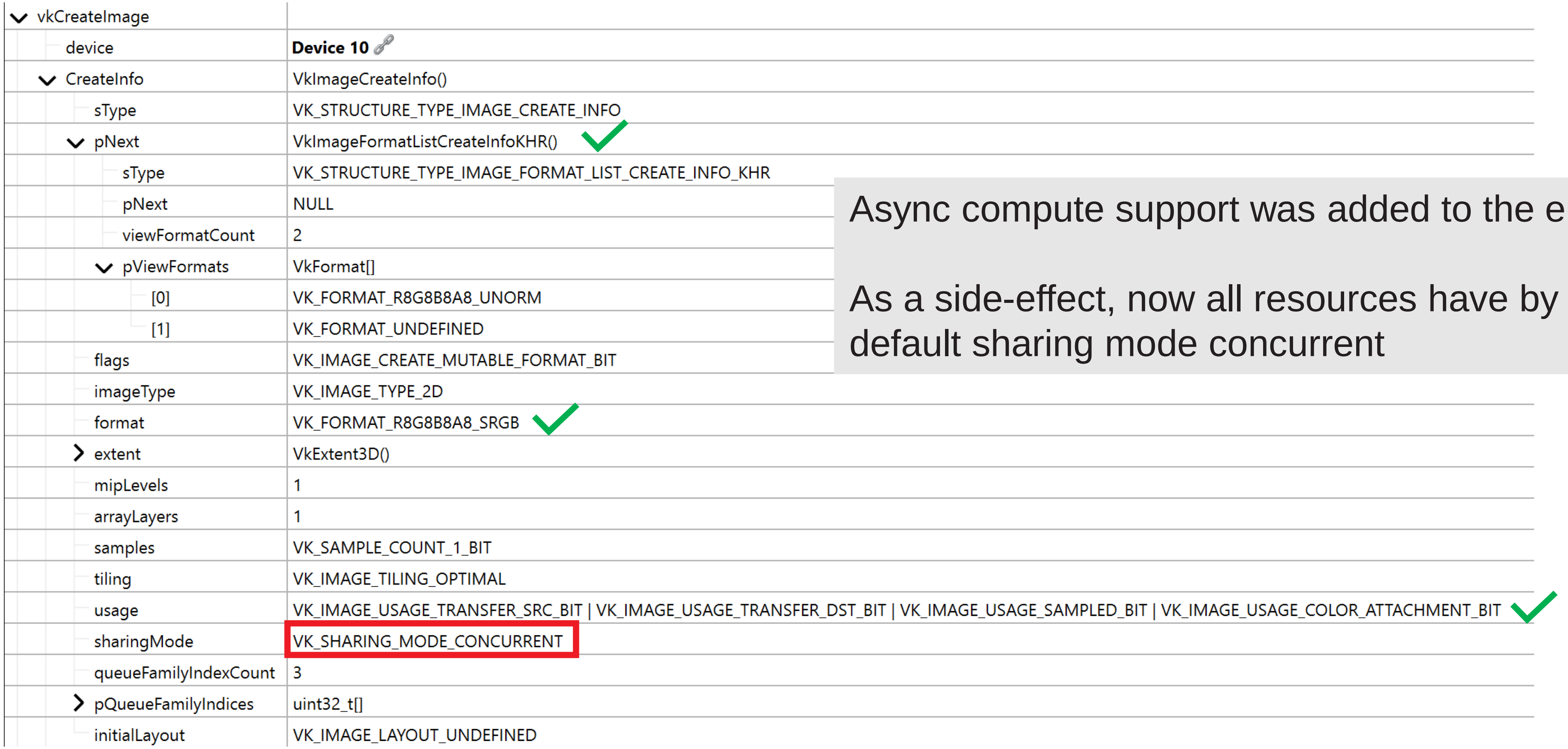
vkCreateImage	
device	**Device 10**
CreateInfo	VkImageCreateInfo()
sType	VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO
pNext	VkImageFormatListCreateInfoKHR() ✓
sType	VK_STRUCTURE_TYPE_IMAGE_FORMAT_LIST_CREATE_INFO_KHR
pNext	NULL
viewFormatCount	2
pViewFormats	VkFormat[]
[0]	VK_FORMAT_R8G8B8A8_UNORM
[1]	VK_FORMAT_UNDEFINED
flags	VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
imageType	VK_IMAGE_TYPE_2D
format	VK_FORMAT_R8G8B8A8_SRGB ✓
extent	VkExtent3D()
mipLevels	1
arrayLayers	1
samples	VK_SAMPLE_COUNT_1_BIT
tiling	VK_IMAGE_TILING_OPTIMAL
usage	VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_USAGE_SAMPLED_BIT | VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT ✓
sharingMode	VK_SHARING_MODE_CONCURRENT
queueFamilyIndexCount	3
pQueueFamilyIndices	uint32_t[]
initialLayout	VK_IMAGE_LAYOUT_UNDEFINED

LET‘S EXAMINE CREATE IMAGE INFO AGAIN



vkCreateImage	
device	**Device 10**
CreateInfo	VkImageCreateInfo()
sType	VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO
pNext	VkImageFormatListCreateInfoKHR() ✓
sType	VK_STRUCTURE_TYPE_IMAGE_FORMAT_LIST_CREATE_INFO_KHR
pNext	NULL
viewFormatCount	2
pViewFormats	VkFormat[]
[0]	VK_FORMAT_R8G8B8A8_UNORM
[1]	VK_FORMAT_UNDEFINED
flags	VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
imageType	VK_IMAGE_TYPE_2D
format	VK_FORMAT_R8G8B8A8_SRGB ✓
extent	VkExtent3D()
mipLevels	1
arrayLayers	1
samples	VK_SAMPLE_COUNT_1_BIT
tiling	VK_IMAGE_TILING_OPTIMAL
usage	VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_USAGE_SAMPLED_BIT | VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT ✓
sharingMode	VK_SHARING_MODE_CONCURRENT
queueFamilyIndexCount	3
pQueueFamilyIndices	uint32_t[]
initialLayout	VK_IMAGE_LAYOUT_UNDEFINED

LET‘S EXAMINE CREATE IMAGE INFO AGAIN

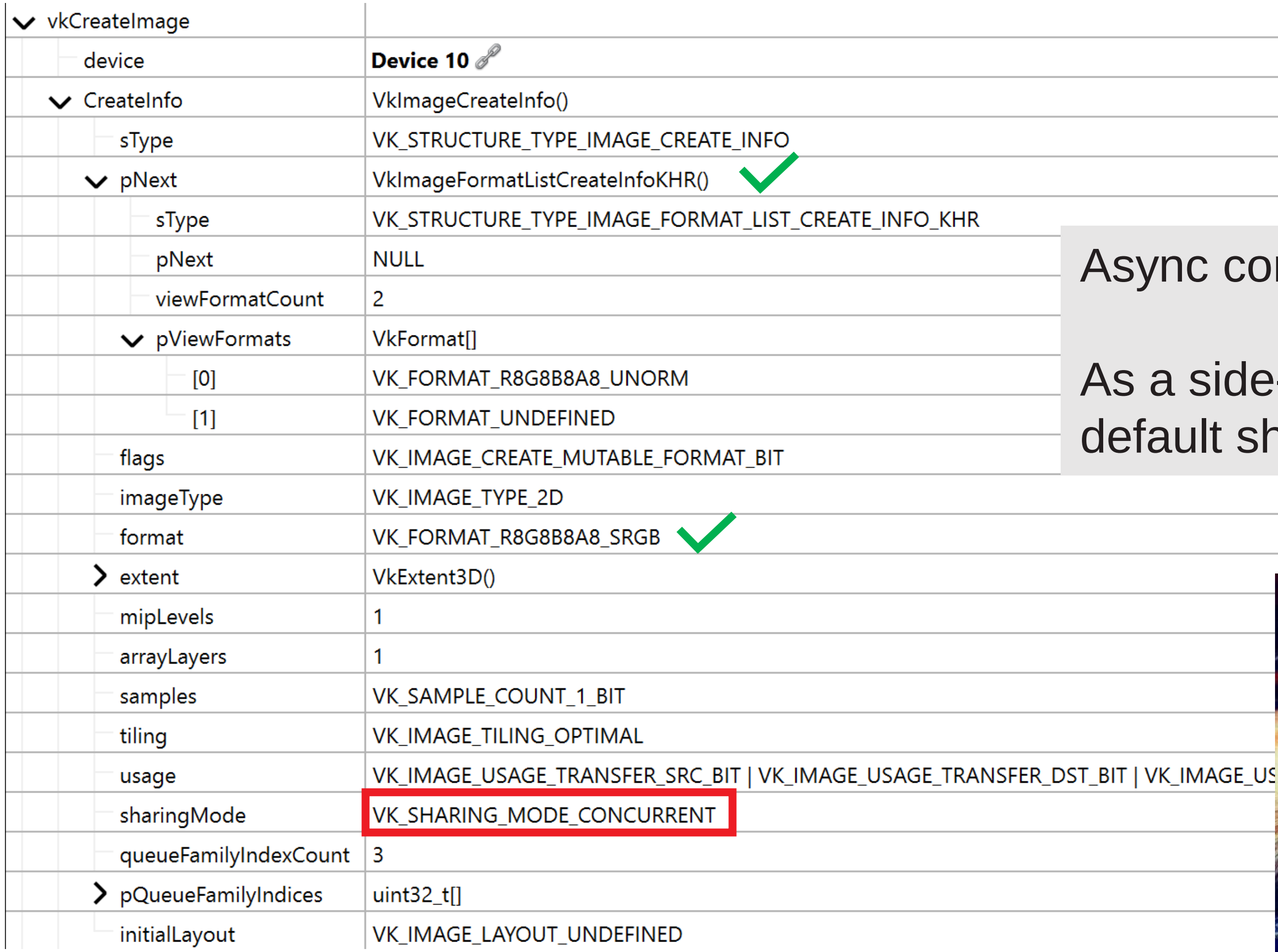


vkCreateImage	
device	**Device 10**
CreateInfo	VkImageCreateInfo()
sType	VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO
pNext	VkImageFormatListCreateInfoKHR() ✓
sType	VK_STRUCTURE_TYPE_IMAGE_FORMAT_LIST_CREATE_INFO_KHR
pNext	NULL
viewFormatCount	2
pViewFormats	VkFormat[]
[0]	VK_FORMAT_R8G8B8A8_UNORM
[1]	VK_FORMAT_UNDEFINED
flags	VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
imageType	VK_IMAGE_TYPE_2D
format	VK_FORMAT_R8G8B8A8_SRGB ✓
extent	VkExtent3D()
mipLevels	1
arrayLayers	1
samples	VK_SAMPLE_COUNT_1_BIT
tiling	VK_IMAGE_TILING_OPTIMAL
usage	VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_USAGE_SAMPLED_BIT | VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT ✓
sharingMode	VK_SHARING_MODE_CONCURRENT
queueFamilyIndexCount	3
pQueueFamilyIndices	uint32_t[]
initialLayout	VK_IMAGE_LAYOUT_UNDEFINED

Async compute support was added to the engine!

As a side-effect, now all resources have by default sharing mode concurrent

LET‘S EXAMINE CREATE IMAGE INFO AGAIN



vkCreateImage	
device	**Device 10**
CreateInfo	VkImageCreateInfo()
sType	VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO
pNext	VkImageFormatListCreateInfoKHR() ✓
sType	VK_STRUCTURE_TYPE_IMAGE_FORMAT_LIST_CREATE_INFO_KHR
pNext	NULL
viewFormatCount	2
pViewFormats	VkFormat[]
[0]	VK_FORMAT_R8G8B8A8_UNORM
[1]	VK_FORMAT_UNDEFINED
flags	VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
imageType	VK_IMAGE_TYPE_2D
format	VK_FORMAT_R8G8B8A8_SRGB ✓
extent	VkExtent3D()
mipLevels	1
arrayLayers	1
samples	VK_SAMPLE_COUNT_1_BIT
tiling	VK_IMAGE_TILING_OPTIMAL
usage	VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_US IT ✓
sharingMode	VK_SHARING_MODE_CONCURRENT
queueFamilyIndexCount	3
pQueueFamilyIndices	uint32_t[]
initialLayout	VK_IMAGE_LAYOUT_UNDEFINED

Async compute support was added to the engine!

As a side-effect, now all resources have by default sharing mode concurrent

VK_SHARING_MODE_CONCURRENT

Spec:

> “VK_SHARING_MODE_CONCURRENT specifies that concurrent access to any range or image subresource of the object from multiple queue families is supported.”

VK_SHARING_MODE_CONCURRENT

Spec:

> “VK_SHARING_MODE_CONCURRENT specifies that concurrent access to any range or image subresource of the object from multiple queue families is supported.”

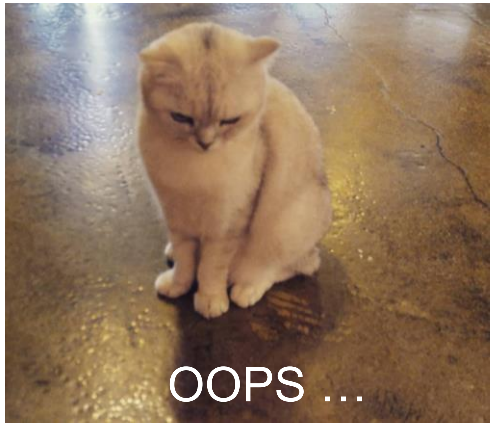
With VK_SHARING_MODE_CONCURRENT **DCC is disabled**

VK_SHARING_MODE_CONCURRENT

Spec:

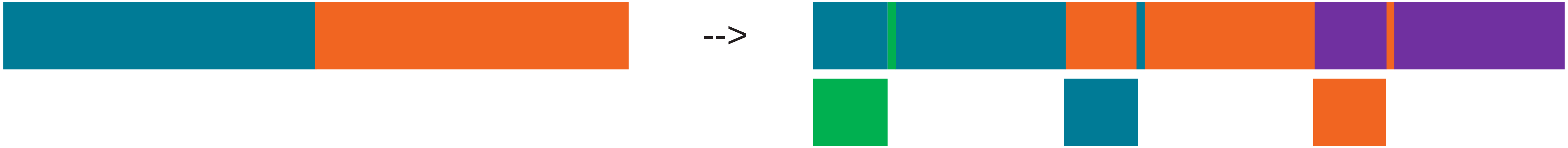
> "VK_SHARING_MODE_CONCURRENT specifies that concurrent access to any range or image subresource of the object from multiple queue families is supported."

With VK_SHARING_MODE_CONCURRENT **DCC is disabled**



VK_SHARING_MODE_CONCURRENT

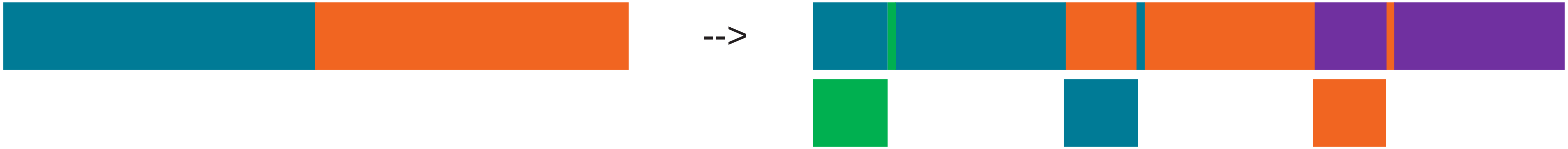
Quick side note on async compute ☺



Improved performance of up to ~10%

VK_SHARING_MODE_CONCURRENT

Quick side note on async compute ☺



Improved performance of up to ~10%

What about DCC?

VK_ SHARING_MODE_CONCURRENT

How to go back to VK_SHARING_MODE_EXCLUSIVE to get DCC enabled?

-> Obviously, if a resource is accessed only by **one** queue, just switch back to EXCLUSIVE

But what about resources, which are accessed by several queue families?

-> transfer queue family ownership

TRANSFER QUEUE FAMILY OWNERSHIP

Done in 2 steps

1. **Release** the exclusive ownership from the **source** queue family
2. **Acquire** the exclusive ownership for the **destination** queue family

TRANSFER QUEUE FAMILY OWNERSHIP

Done in 2 steps

1. **Release** the exclusive ownership from the **source** queue family
2. **Acquire** the exclusive ownership for the **destination** queue family

Example:

Queue family 0 holds currently the exclusive ownership of image A

Queue family 1 wants to acquire exclusive ownership of image A

RELEASE THE EXCLUSIVE OWNERSHIP

```
VkImageMemoryBarrier imageMemoryBarrier = {};
imageMemoryBarrier.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
imageMemoryBarrier.srcAccessMask = …
imageMemoryBarrier.dstAccessMask = 0;
imageMemoryBarrier.oldLayout = oldLayoutImageA;
imageMemoryBarrier.newLayout = newLayoutImageA;
imageMemoryBarrier.srcQueueFamilyIndex = 0;
imageMemoryBarrier.dstQueueFamilyIndex = 1;
imageMemoryBarrier.image = imageA;
imageMemoryBarrier .subresourceRange = subresourceRangeImageA;
…
vkCmdPipelineBarrier(cmdBuf, …);
…
vkQueueSubmit(queueFamily0,…, submitInfo, …);
```

RELEASE THE EXCLUSIVE OWNERSHIP

```
VkImageMemoryBarrier imageMemoryBarrier = {};
imageMemoryBarrier.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
imageMemoryBarrier.srcAccessMask = …
imageMemoryBarrier.dstAccessMask = 0;
imageMemoryBarrier.oldLayout = oldLayoutImageA;
imageMemoryBarrier.newLayout = newLayoutImageA;
imageMemoryBarrier.srcQueueFamilyIndex = 0;
imageMemoryBarrier.dstQueueFamilyIndex = 1;
imageMemoryBarrier.image = imageA;
imageMemoryBarrier .subresourceRange = subresourceRangeImageA;
…
vkCmdPipelineBarrier(cmdBuf, …);
…
vkQueueSubmit(queueFamily0,…, submitInfo, …);
```

Associated to a commandPool

RELEASE THE EXCLUSIVE OWNERSHIP

```
VkImageMemoryBarrier imageMemoryBarrier = {};
imageMemoryBarrier.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
imageMemoryBarrier.srcAccessMask = …
imageMemoryBarrier.dstAccessMask = 0;
imageMemoryBarrier.oldLayout = oldLayoutImageA;
imageMemoryBarrier.newLayout = newLayoutImageA;
imageMemoryBarrier.srcQueueFamilyIndex = 0;
imageMemoryBarrier.dstQueueFamilyIndex = 1;
imageMemoryBarrier.image = imageA;
imageMemoryBarrier .subresourceRange = subresourceRangeImageA;
…
vkCmdPipelineBarrier(cmdBuf, …);
…
vkQueueSubmit(queueFamily0,…, submitInfo, …);
```

Associated to a commandPool → Associated to queue family **0**

RELEASE THE EXCLUSIVE OWNERSHIP

```
VkImageMemoryBarrier imageMemoryBarrier = {};
imageMemoryBarrier.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
imageMemoryBarrier.srcAccessMask = …
imageMemoryBarrier.dstAccessMask = 0;
imageMemoryBarrier.oldLayout = oldLayoutImageA;
imageMemoryBarrier.newLayout = newLayoutImageA;
imageMemoryBarrier.srcQueueFamilyIndex = 0;
imageMemoryBarrier.dstQueueFamilyIndex = 1;
imageMemoryBarrier.image = imageA;
imageMemoryBarrier .subresourceRange = subresourceRangeImageA;
…
vkCmdPipelineBarrier(cmdBuf, …);
…
vkQueueSubmit(queueFamily0,…, submitInfo, …);
```

→ Semaphore to sync across queues

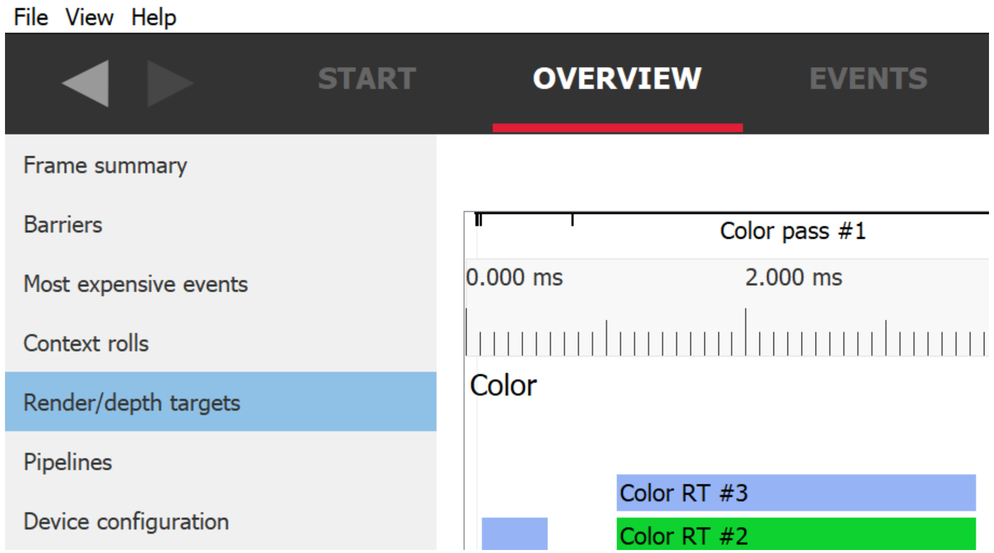
ACQUIRE THE EXCLUSIVE OWNERSHIP

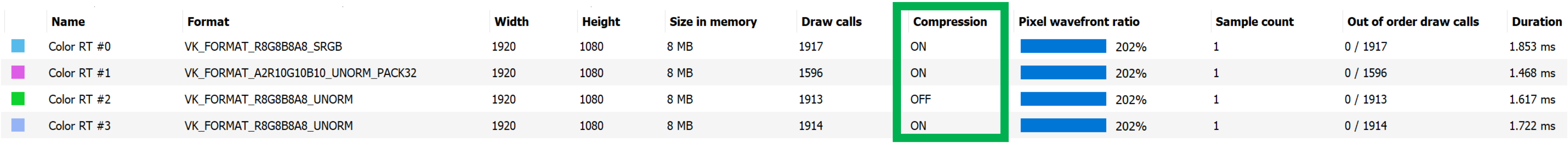
```
VkImageMemoryBarrier imageMemoryBarrier = {};
imageMemoryBarrier.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
imageMemoryBarrier.srcAccessMask = 0;
imageMemoryBarrier.dstAccessMask = …
imageMemoryBarrier.oldLayout = oldLayoutImageA;
imageMemoryBarrier.newLayout = newLayoutImageA;
imageMemoryBarrier.srcQueueFamilyIndex = 0;
imageMemoryBarrier.dstQueueFamilyIndex = 1;
imageMemoryBarrier.image = imageA;
imageMemoryBarrier .subresourceRange = subresourceRangeImageA;
…
vkCmdPipelineBarrier(cmdBuf, …);
…
vkQueueSubmit(queueFamily1,…, submitInfo, …);
```

Associated to a commandPool → Associated to queue family **1**

LET‘S CHECK AGAIN ☺

Use Radeon GPU Profiler (RGP):

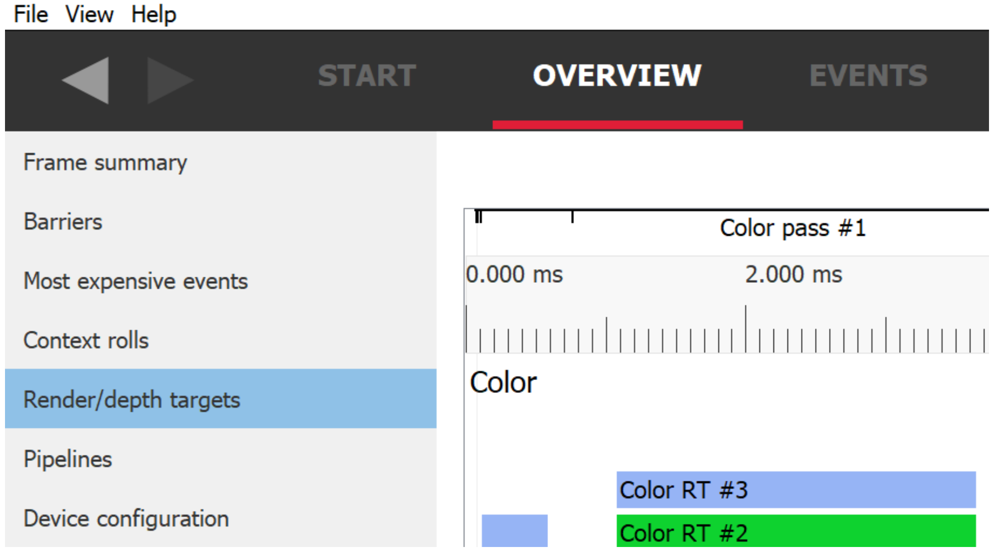


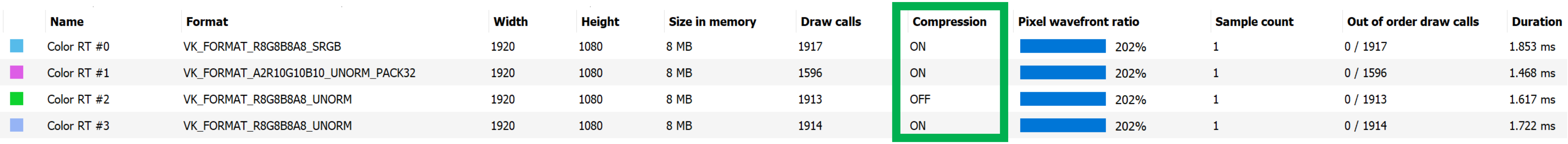


	Name	Format	Width	Height	Size in memory	Draw calls	Compression	Pixel wavefront ratio	Sample count	Out of order draw calls	Duration
🟦	Color RT #0	VK_FORMAT_R8G8B8A8_SRGB	1920	1080	8 MB	1917	ON	202%	1	0 / 1917	1.853 ms
🟪	Color RT #1	VK_FORMAT_A2R10G10B10_UNORM_PACK32	1920	1080	8 MB	1596	ON	202%	1	0 / 1596	1.468 ms
🟩	Color RT #2	VK_FORMAT_R8G8B8A8_UNORM	1920	1080	8 MB	1913	OFF	202%	1	0 / 1913	1.617 ms
🟦	Color RT #3	VK_FORMAT_R8G8B8A8_UNORM	1920	1080	8 MB	1914	ON	202%	1	0 / 1914	1.722 ms

LET‘S CHECK AGAIN ☺

Use Radeon GPU Profiler (RGP):

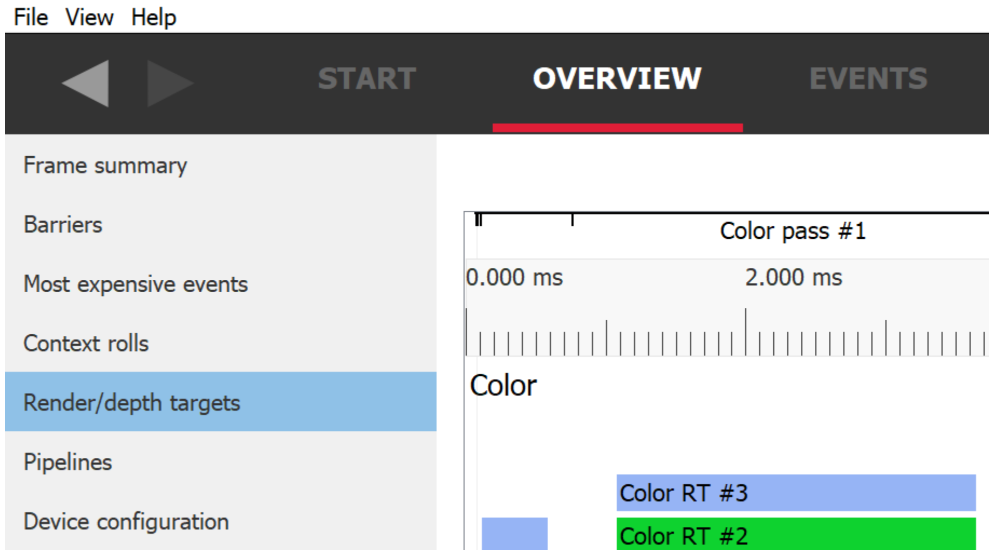




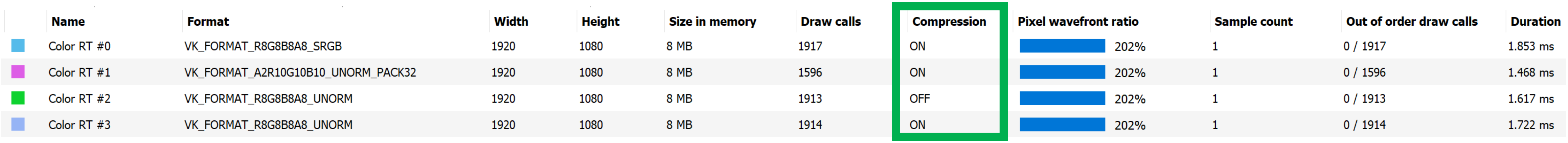
	Name	Format	Width	Height	Size in memory	Draw calls	Compression	Pixel wavefront ratio	Sample count	Out of order draw calls	Duration
🟦	Color RT #0	VK_FORMAT_R8G8B8A8_SRGB	1920	1080	8 MB	1917	ON	202%	1	0 / 1917	1.853 ms
🟪	Color RT #1	VK_FORMAT_A2R10G10B10_UNORM_PACK32	1920	1080	8 MB	1596	ON	202%	1	0 / 1596	1.468 ms
🟩	Color RT #2	VK_FORMAT_R8G8B8A8_UNORM	1920	1080	8 MB	1913	OFF	202%	1	0 / 1913	1.617 ms
🟦	Color RT #3	VK_FORMAT_R8G8B8A8_UNORM	1920	1080	8 MB	1914	ON	202%	1	0 / 1914	1.722 ms

LET'S CHECK AGAIN ☺

Use Radeon GPU Profiler (RGP):



The performance increased about ~5-10%, depending on AMD graphics card

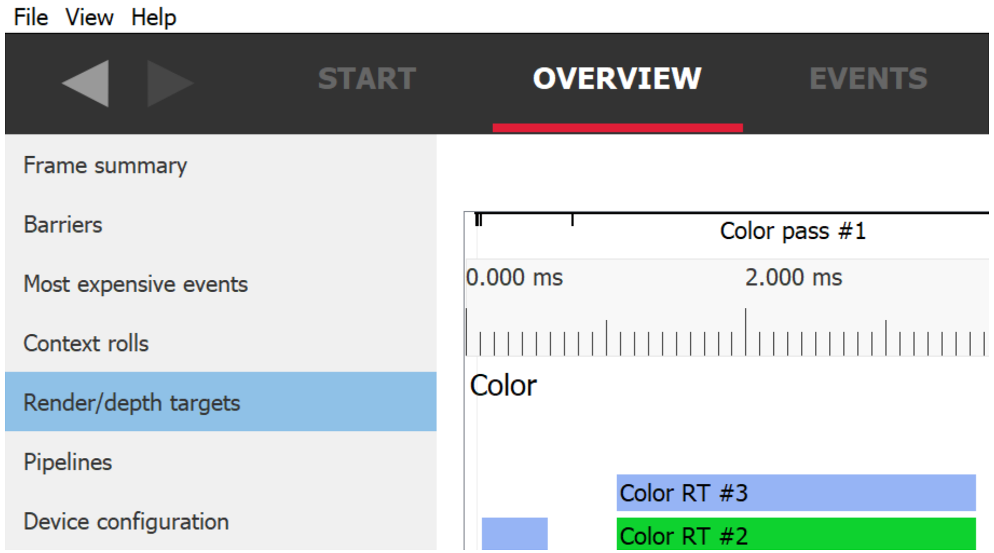


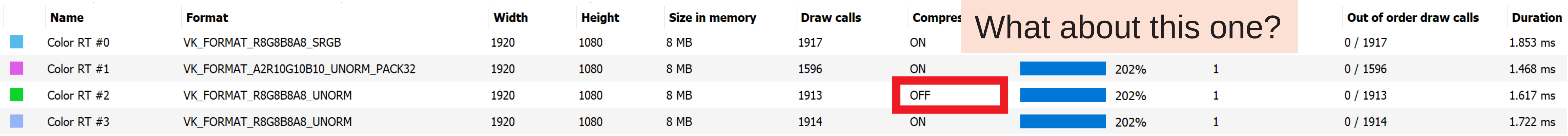
	Name	Format	Width	Height	Size in memory	Draw calls	Compression	Pixel wavefront ratio	Sample count	Out of order draw calls	Duration
🟦	Color RT #0	VK_FORMAT_R8G8B8A8_SRGB	1920	1080	8 MB	1917	ON	202%	1	0 / 1917	1.853 ms
🟪	Color RT #1	VK_FORMAT_A2R10G10B10_UNORM_PACK32	1920	1080	8 MB	1596	ON	202%	1	0 / 1596	1.468 ms
🟩	Color RT #2	VK_FORMAT_R8G8B8A8_UNORM	1920	1080	8 MB	1913	OFF	202%	1	0 / 1913	1.617 ms
🟦	Color RT #3	VK_FORMAT_R8G8B8A8_UNORM	1920	1080	8 MB	1914	ON	202%	1	0 / 1914	1.722 ms

LET'S CHECK AGAIN ☺

Use Radeon GPU Profiler (RGP):

The performance increased about ~5-10%, depending on AMD graphics card



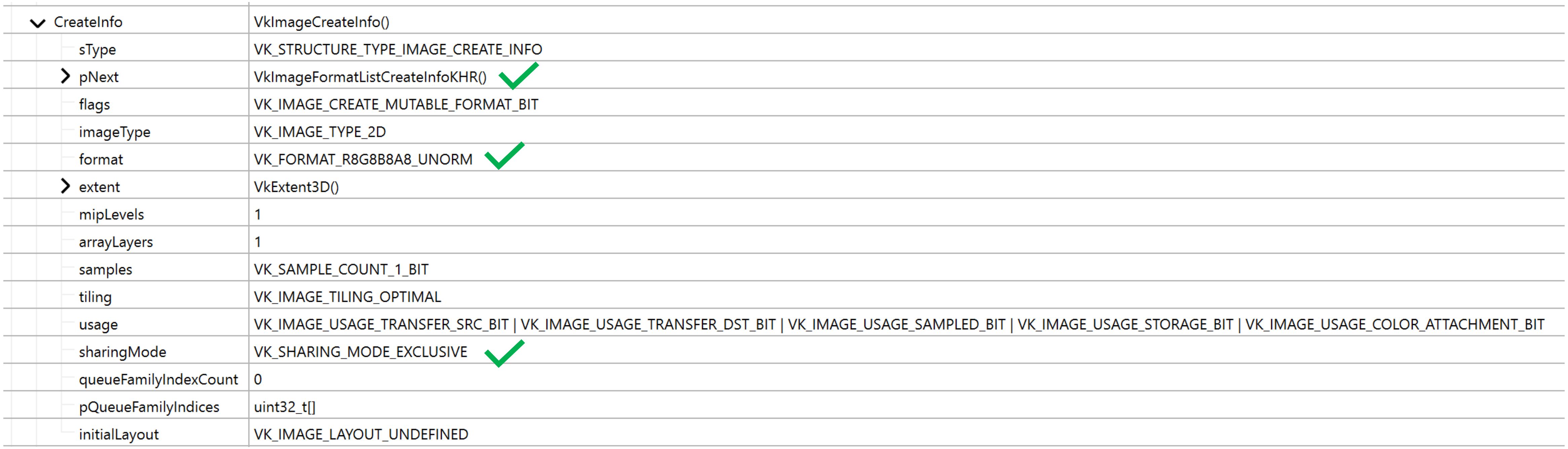


	Name	Format	Width	Height	Size in memory	Draw calls	Compres			Out of order draw calls	Duration
🟦	Color RT #0	VK_FORMAT_R8G8B8A8_SRGB	1920	1080	8 MB	1917	ON			0 / 1917	1.853 ms
🟪	Color RT #1	VK_FORMAT_A2R10G10B10_UNORM_PACK32	1920	1080	8 MB	1596	ON	202%	1	0 / 1596	1.468 ms
🟩	Color RT #2	VK_FORMAT_R8G8B8A8_UNORM	1920	1080	8 MB	1913	OFF	202%	1	0 / 1913	1.617 ms
🟦	Color RT #3	VK_FORMAT_R8G8B8A8_UNORM	1920	1080	8 MB	1914	ON	202%	1	0 / 1914	1.722 ms

What about this one?

AND ONCE AGAIN … ☺

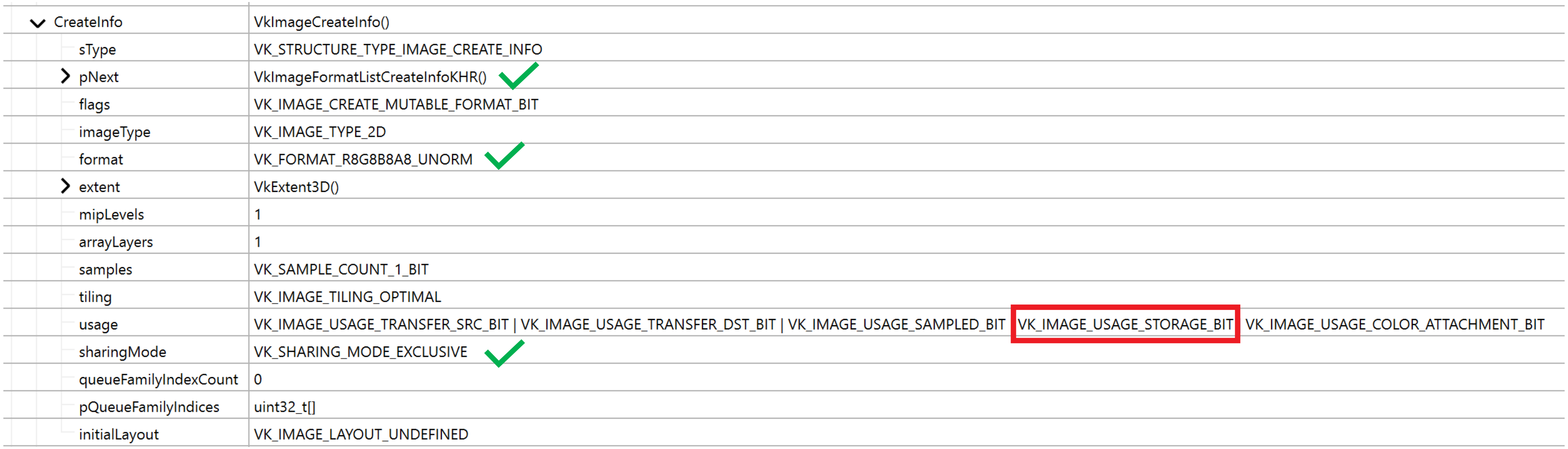
Color RT #2 – G-buffer resource #2



⌄ CreateInfo	VkImageCreateInfo()
sType	VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO
› pNext	VkImageFormatListCreateInfoKHR() ✓
flags	VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
imageType	VK_IMAGE_TYPE_2D
format	VK_FORMAT_R8G8B8A8_UNORM ✓
› extent	VkExtent3D()
mipLevels	1
arrayLayers	1
samples	VK_SAMPLE_COUNT_1_BIT
tiling	VK_IMAGE_TILING_OPTIMAL
usage	VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_USAGE_SAMPLED_BIT | VK_IMAGE_USAGE_STORAGE_BIT | VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
sharingMode	VK_SHARING_MODE_EXCLUSIVE ✓
queueFamilyIndexCount	0
pQueueFamilyIndices	uint32_t[]
initialLayout	VK_IMAGE_LAYOUT_UNDEFINED

AND ONCE AGAIN … ☺

Color RT #2 – G-buffer resource #2



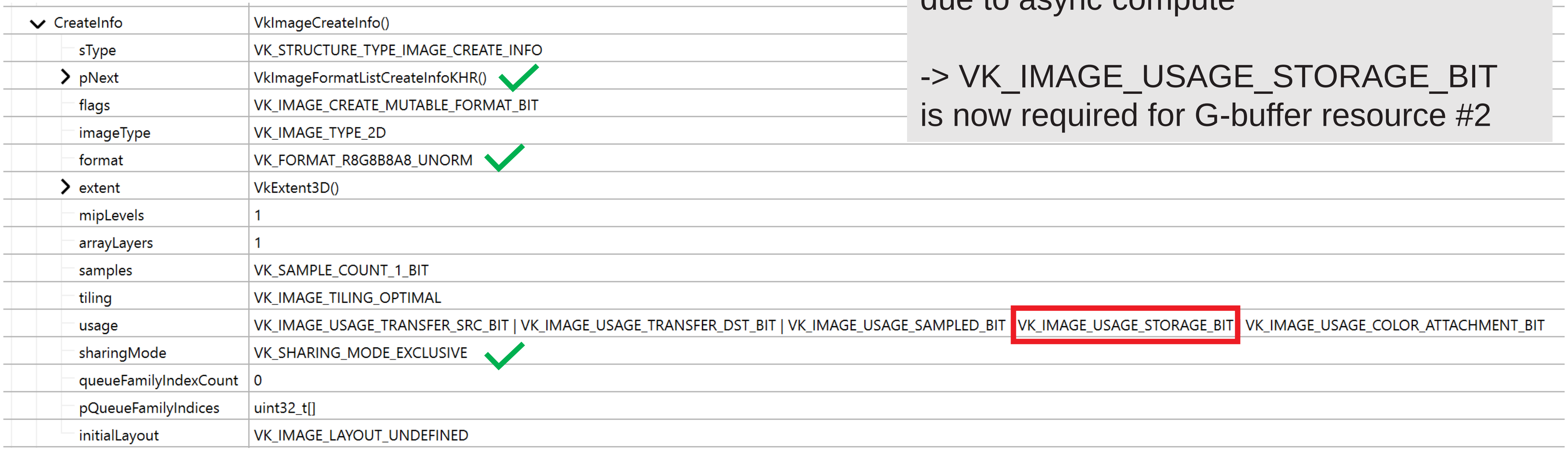
CreateInfo	VkImageCreateInfo()
sType	VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO
pNext	VkImageFormatListCreateInfoKHR() ✓
flags	VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
imageType	VK_IMAGE_TYPE_2D
format	VK_FORMAT_R8G8B8A8_UNORM ✓
extent	VkExtent3D()
mipLevels	1
arrayLayers	1
samples	VK_SAMPLE_COUNT_1_BIT
tiling	VK_IMAGE_TILING_OPTIMAL
usage	VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_USAGE_SAMPLED_BIT VK_IMAGE_USAGE_STORAGE_BIT VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
sharingMode	VK_SHARING_MODE_EXCLUSIVE ✓
queueFamilyIndexCount	0
pQueueFamilyIndices	uint32_t[]
initialLayout	VK_IMAGE_LAYOUT_UNDEFINED

USAGE FLAGS

Color RT #2 – G-buffer resource #2

Post process moved to the compute queue due to async compute

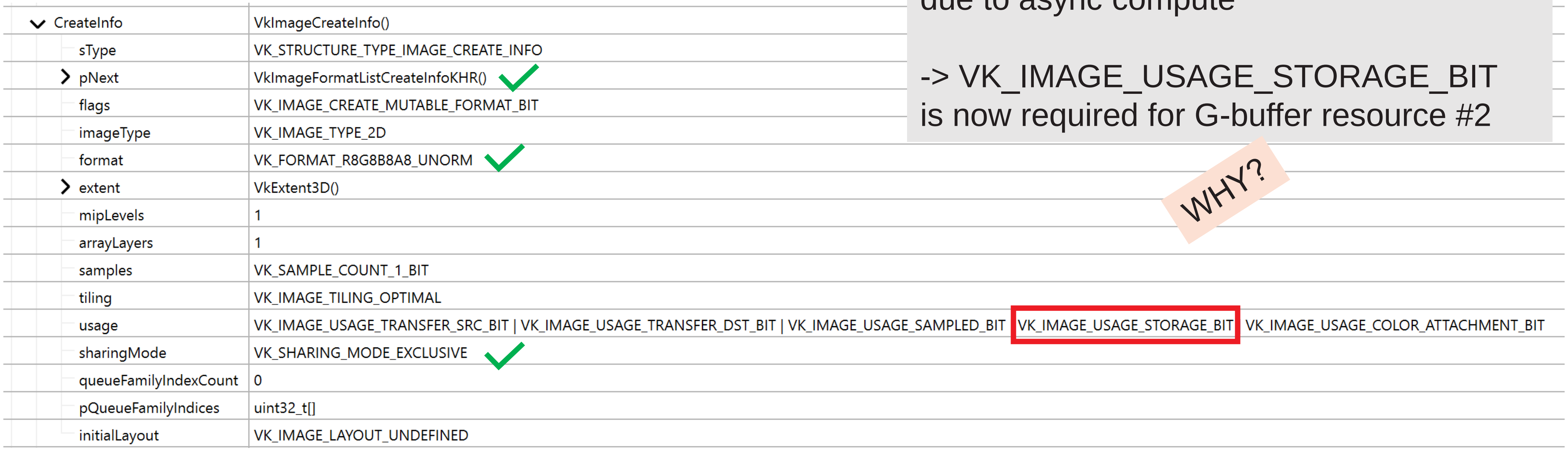
-> VK_IMAGE_USAGE_STORAGE_BIT is now required for G-buffer resource #2



CreateInfo	VkImageCreateInfo()
sType	VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO
pNext	VkImageFormatListCreateInfoKHR() ✓
flags	VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
imageType	VK_IMAGE_TYPE_2D
format	VK_FORMAT_R8G8B8A8_UNORM ✓
extent	VkExtent3D()
mipLevels	1
arrayLayers	1
samples	VK_SAMPLE_COUNT_1_BIT
tiling	VK_IMAGE_TILING_OPTIMAL
usage	VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_USAGE_SAMPLED_BIT VK_IMAGE_USAGE_STORAGE_BIT VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
sharingMode	VK_SHARING_MODE_EXCLUSIVE ✓
queueFamilyIndexCount	0
pQueueFamilyIndices	uint32_t[]
initialLayout	VK_IMAGE_LAYOUT_UNDEFINED

USAGE FLAGS

Color RT #2 – G-buffer resource #2



CreateInfo	VkImageCreateInfo()
sType	VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO
pNext	VkImageFormatListCreateInfoKHR() ✓
flags	VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
imageType	VK_IMAGE_TYPE_2D
format	VK_FORMAT_R8G8B8A8_UNORM ✓
extent	VkExtent3D()
mipLevels	1
arrayLayers	1
samples	VK_SAMPLE_COUNT_1_BIT
tiling	VK_IMAGE_TILING_OPTIMAL
usage	VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_USAGE_SAMPLED_BIT VK_IMAGE_USAGE_STORAGE_BIT VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
sharingMode	VK_SHARING_MODE_EXCLUSIVE ✓
queueFamilyIndexCount	0
pQueueFamilyIndices	uint32_t[]
initialLayout	VK_IMAGE_LAYOUT_UNDEFINED

Post process moved to the compute queue due to async compute

-> VK_IMAGE_USAGE_STORAGE_BIT is now required for G-buffer resource #2

WHY?

VK_IMAGE_USAGE_STORAGE_BIT

Spec:

> „VK_IMAGE_USAGE_STORAGE_BIT specifies that the image can be used to create a VkImageView suitable for occupying a VkDescriptorSet slot of type VK_DESCRIPTOR_TYPE_STORAGE_IMAGE “

Spec:

> „ A storage image (VK_DESCRIPTOR_TYPE_STORAGE_IMAGE) is a descriptor type associated with an image resource via an image view that load, **store**, and atomic operations can be performed on.“

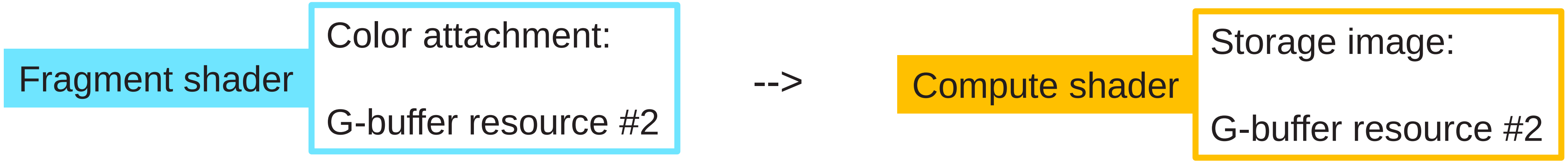
VK_IMAGE_USAGE_STORAGE_BIT

Spec:

> „VK_IMAGE_USAGE_STORAGE_BIT specifies that the image can be used to create a VkImageView suitable for occupying a VkDescriptorSet slot of type VK_DESCRIPTOR_TYPE_STORAGE_IMAGE “

Spec:

> „ A storage image (VK_DESCRIPTOR_TYPE_STORAGE_IMAGE) is a descriptor type associated with an image resource via an image view that load, **store**, and atomic operations can be performed on.“



Fragment shader

Color attachment:

G-buffer resource #2

-->

Compute shader

Storage image:

G-buffer resource #2

USAGE FLAGS

Usage flags influencing DCC:

- VK_IMAGE_USAGE_STORAGE_BIT – **disables** DCC
- VK_IMAGE_USAGE_SAMPLED_BIT – makes DCC **less** efficient

USAGE FLAGS

Usage flags influencing DCC:

- VK_IMAGE_USAGE_STORAGE_BIT – **disables** DCC
- VK_IMAGE_USAGE_SAMPLED_BIT – makes DCC **less** efficient

Always use what you need, but not more ☺

SUMMARY

VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT

- use VK_KHR_image_format_list

VK_SHARING_MODE_EXCLUSIVE

- don‘t use sharing mode concurrent in production ready code
- use SHARING_MODE_EXCLUSIVE and transfer queue family ownership when required

USAGE FLAGS

- set all the usage flags you need, but not more

OTHER NIT-PICKS CONCERNING DCC

Decompression

- During transfer operations
- General layout

Depth targets

- Compressed differently
- Above guidelines don‘t apply here

There is no rule without expection 😈

- There might be some tweaks in the driver for specific cards

OTHER NIT-PICKS CONCERNING DCC

Decompression

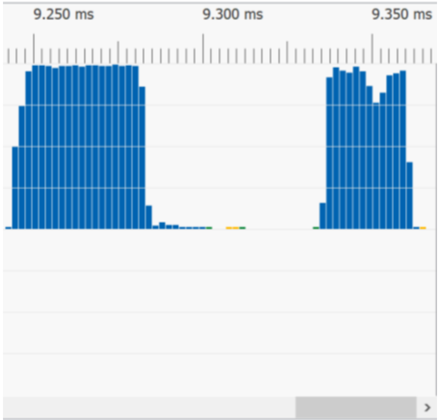
- During transfer operations
- General layout

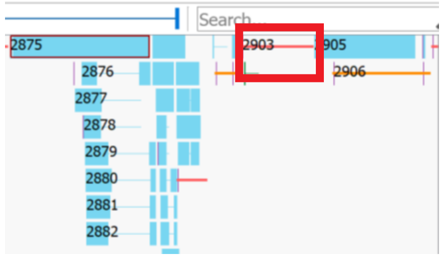
Depth targets

- Compressed differently
- Above guidelines don‘t apply here

There is no rule without expection 😈

- There might be some tweaks in the driver for specific cards





SYNCHRONIZATION

Barriers

- Placing
- Batching
- Pipeline stage masks

Cross queue synchronization

BARRIERS

- Experience with barriers in this particular game
- Most of the issues likely have their roots in the original engine structure, which is DX11-based

BARRIERS

- Experience with barriers in this particular game
- Most of the issues likely have their roots in the original engine structure, which is DX11-based

-> Rearranging barriers to get more overlap between the drawcalls / passes

-> Batching barriers to save some additional time

BARRIERS

- Experience with barriers in this particular game
- Most of the issues likely have their roots in the original engine structure, which is DX11 based

-> Rearranging barriers to get more overlap between the drawcalls / passes

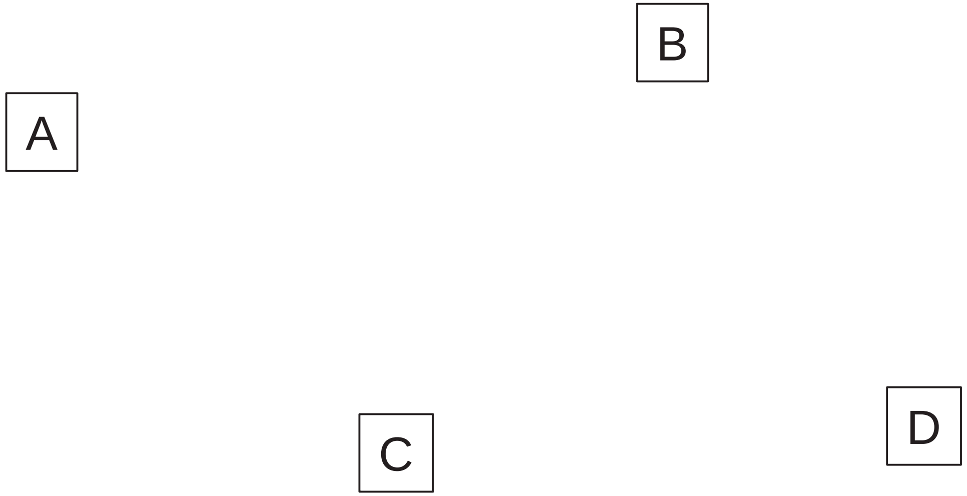
-> Batching barriers to save some additional time

- Other findings

-> Where specifying barriers as precise as possible really pays of

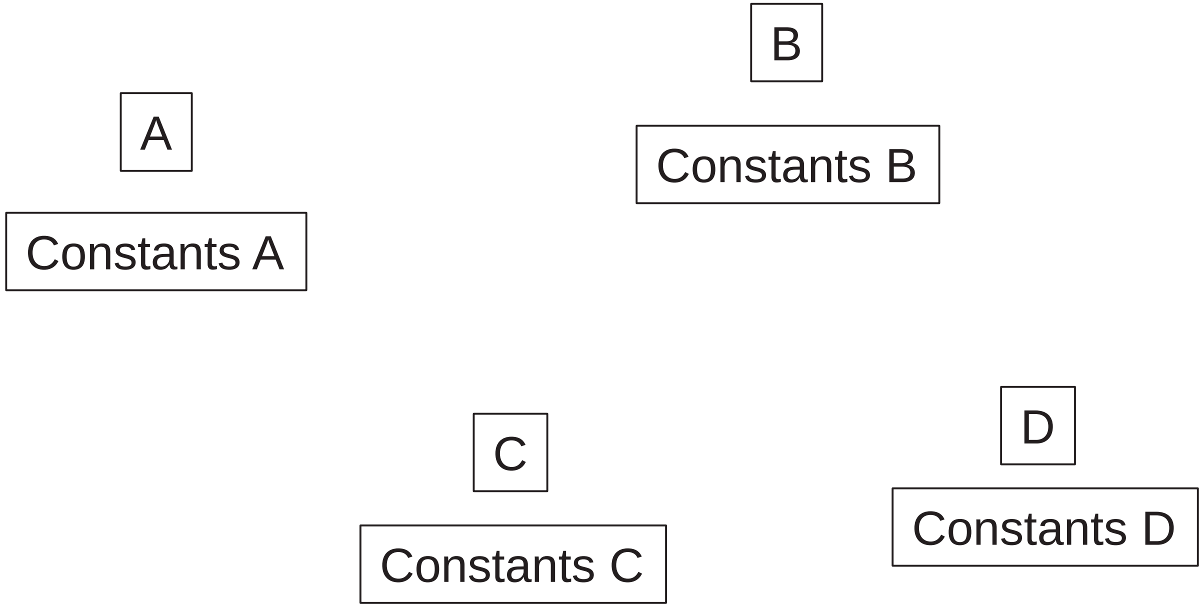
BARRIERS – ORIGINAL SETUP

- The rendering work is logically organized in components – e.g. one shadow map component, one lighting component etc.



BARRIERS – ORIGINAL SETUP

- The rendering work is logically organized in components – e.g. one shadow map component, one lighting component etc.



A

Constants A

B

Constants B

C

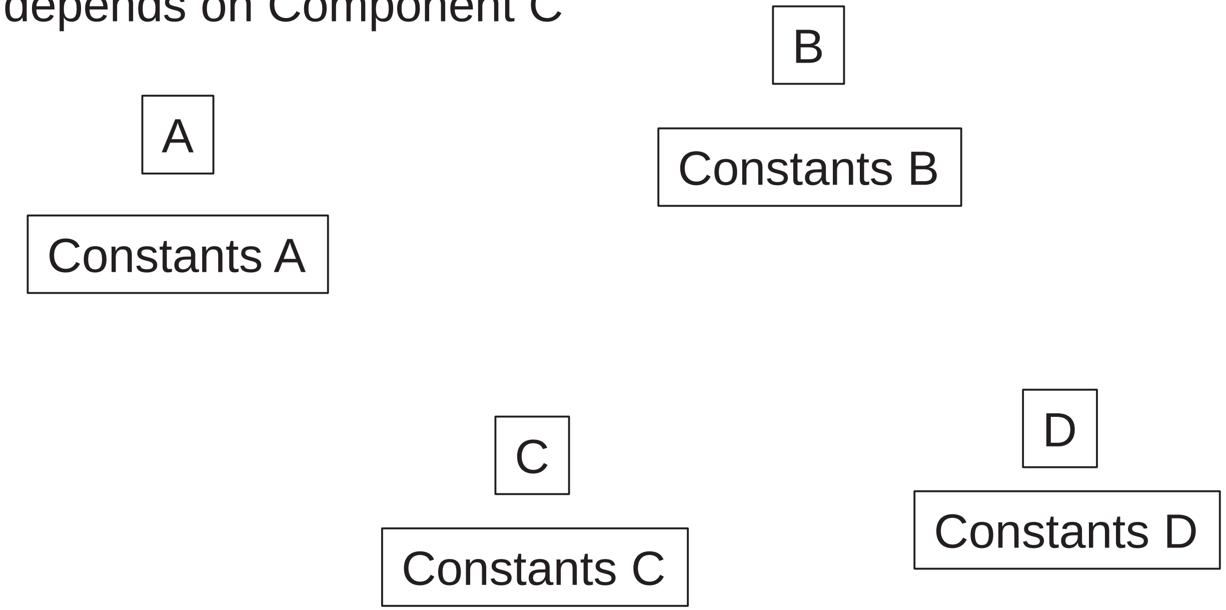
Constants C

D

Constants D

BARRIERS – ORIGINAL SETUP

- Constants information is gathered on the CPU side in the beginning of each frame
- Constants A, B and C are equal, constants D are different
- Component A is independent from Component B
- Component C depends on Component A and B
- Component D depends on Component C



B

A

Constants B

Constants A

C

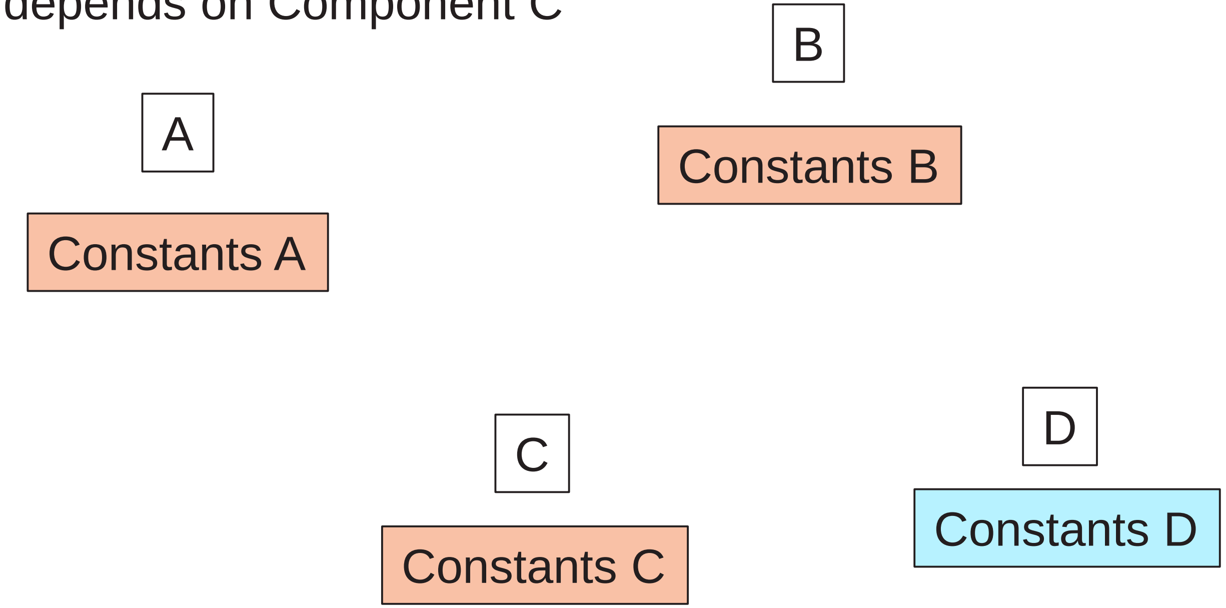
D

Constants D

Constants C

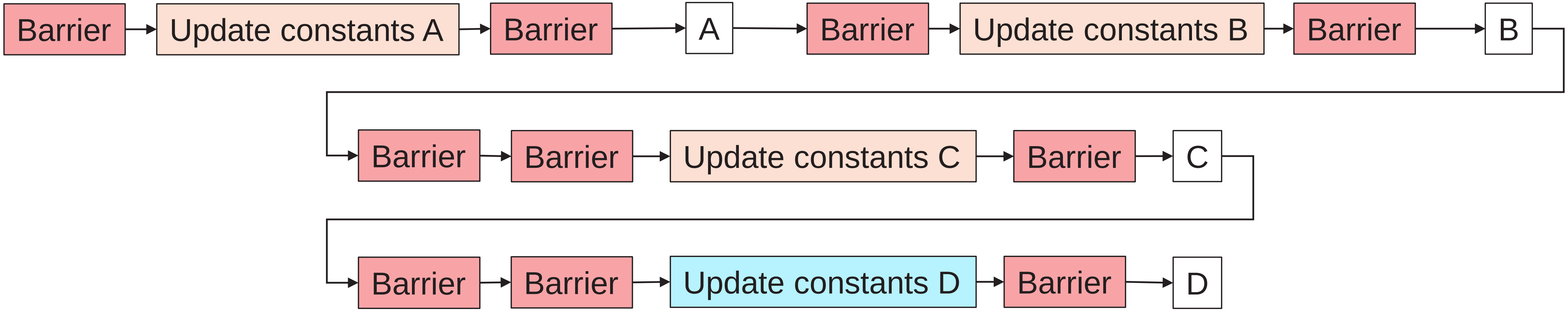
BARRIERS – ORIGINAL SETUP

- Constants information is gathered on the CPU side in the beginning of each frame
- Constants A, B and C are equal, constants D are different
- Component A is independent from Component B
- Component C depends on Component A and B
- Component D depends on Component C



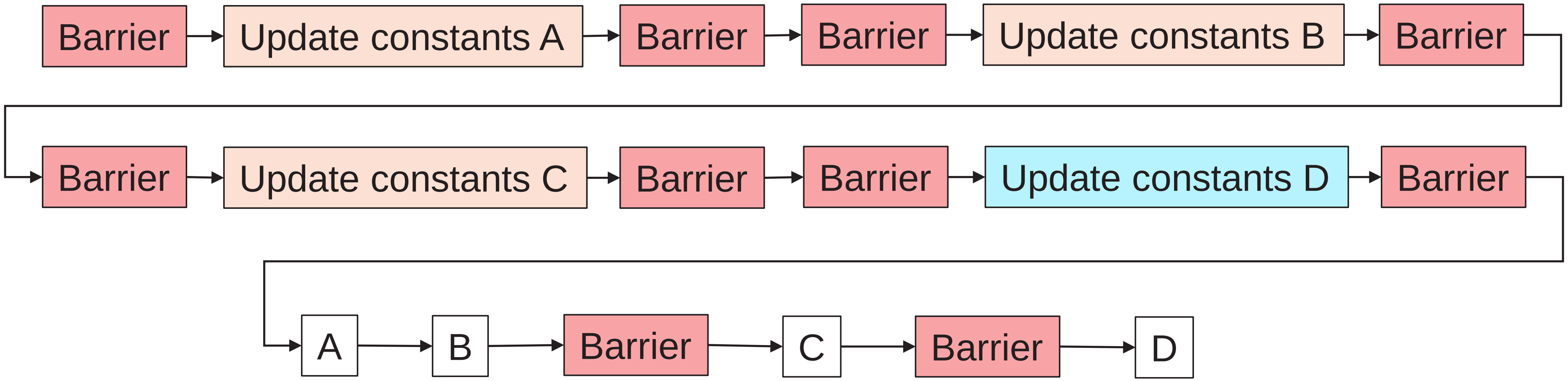
BARRIERS – ORIGINAL SETUP

- Constants information is gathered on the CPU side in the beginning of each frame
- Constants A, B and C are equal, constants D are different
- Component A is independent from Component B
- Component C depends on Component A and B
- Component D depends on Component C



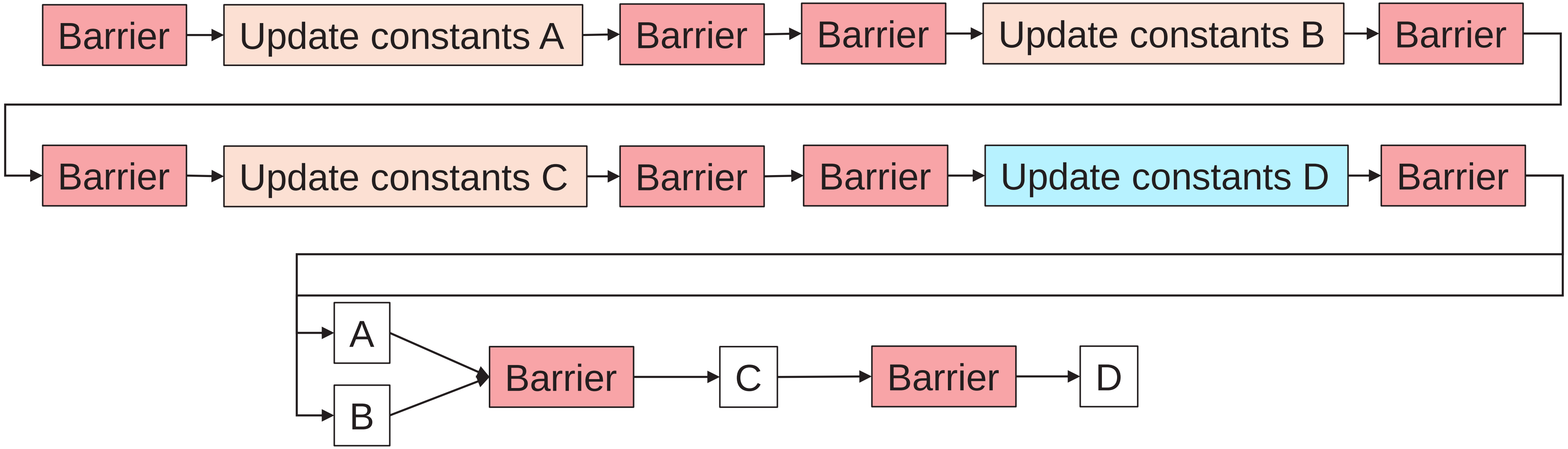
BARRIERS – OPTIMIZED

- Constants information is gathered on the CPU side in the beginning of each frame
- Constants A, B and C are equal, constants D are different
- Component A is independent from Component B
- Component C depends on Component A and B
- Component D depends on Component C



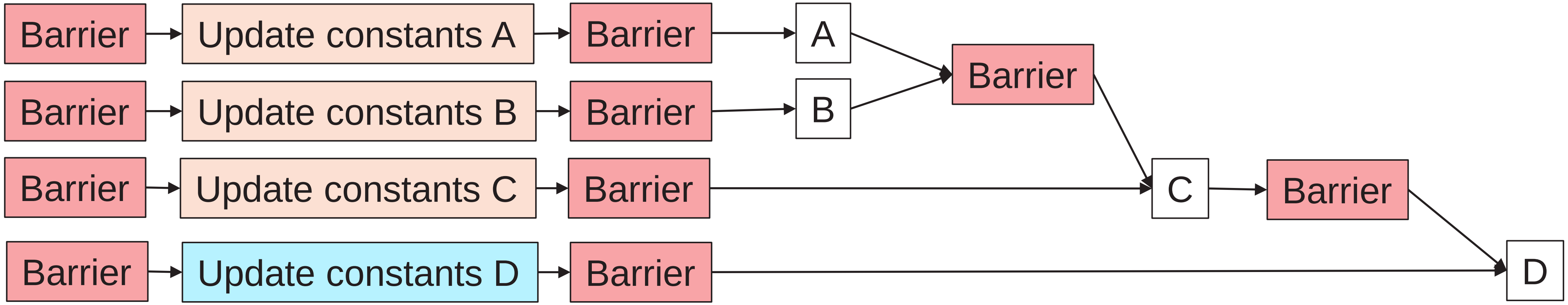
BARRIERS – OPTIMIZED

- Constants information is gathered on the CPU side in the beginning of each frame
- Constants A, B and C are equal, constants D are different
- Component A is independent from Component B
- Component C depends on Component A and B
- Component D depends on Component C



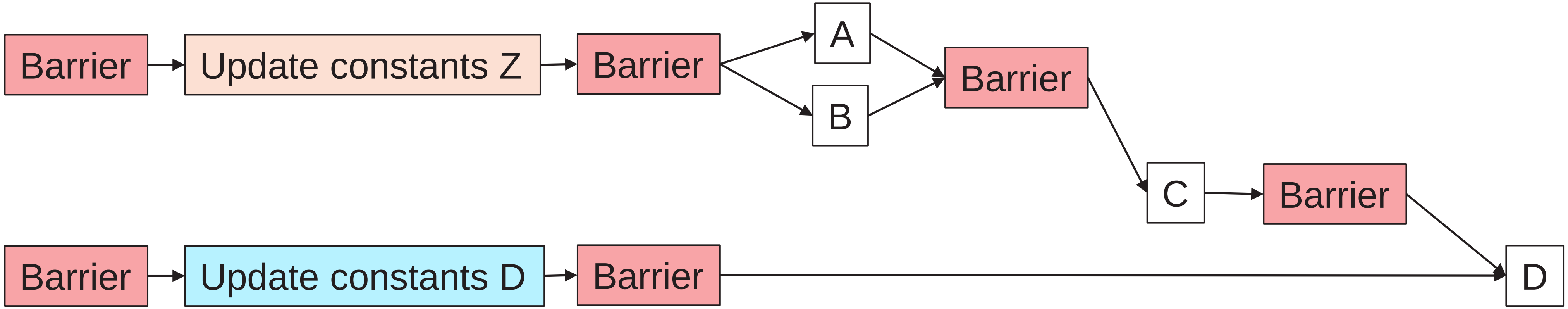
BARRIERS – OPTIMIZED

- Constants information is gathered on the CPU side in the beginning of each frame
- Constants A, B and C are equal, constants D are different
- Component A is independent from Component B
- Component C depends on Component A and B
- Component D depends on Component C



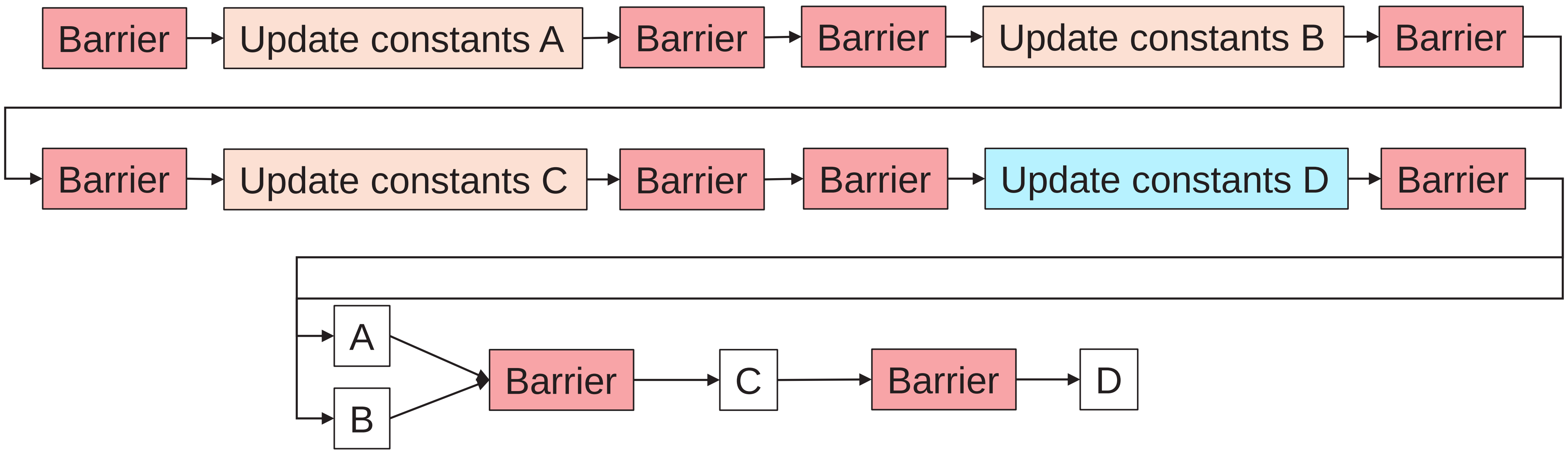
BARRIERS – OPTIMIZED

- Constants information is gathered on the CPU side in the beginning of each frame
- Constants A, B and C are equal, constants D are different
- Component A is independent from Component B
- Component C depends on Component A and B
- Component D depends on Component C

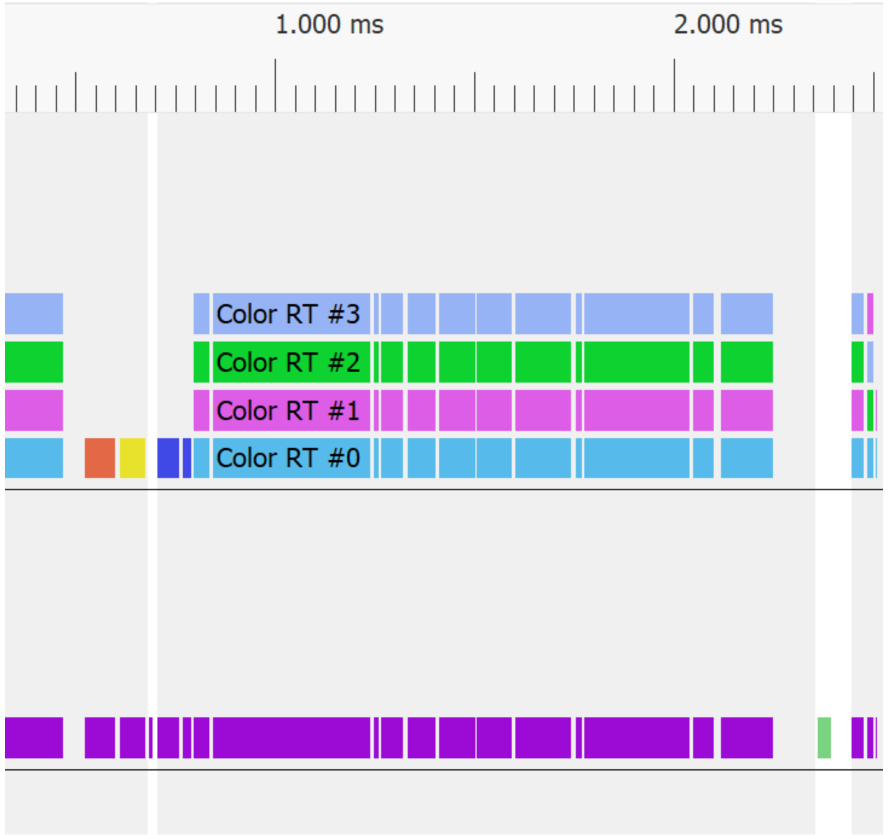


BARRIERS – OPTIMIZED

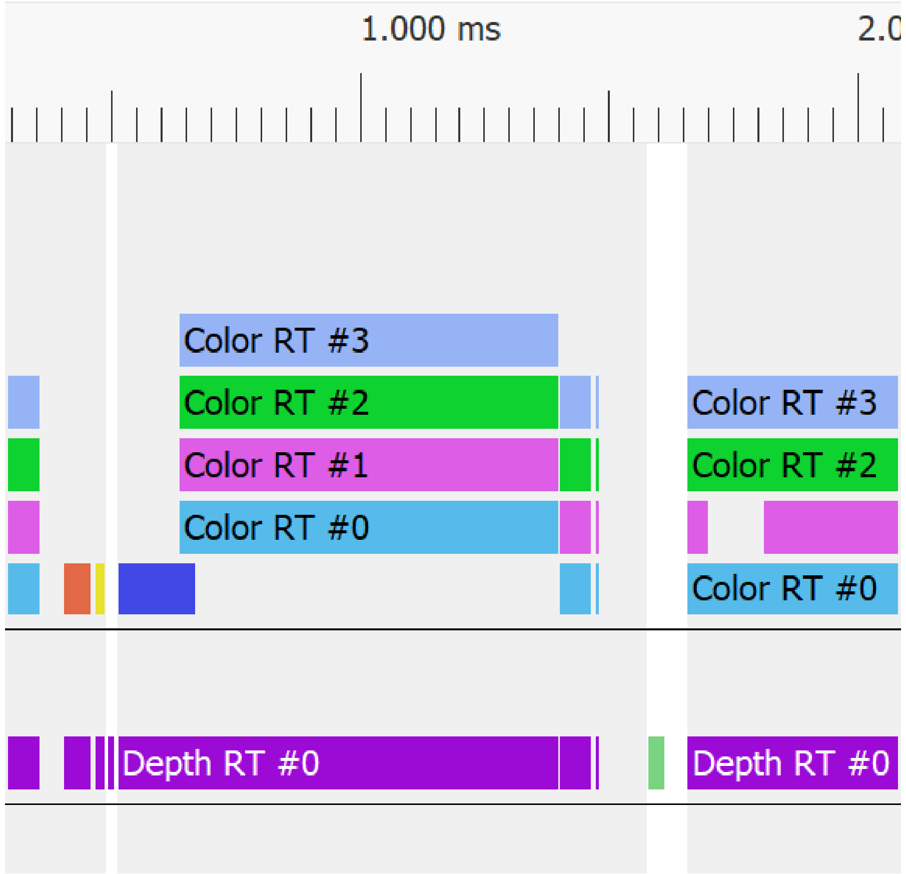
- This is what we ended up with – but it already had observable changes



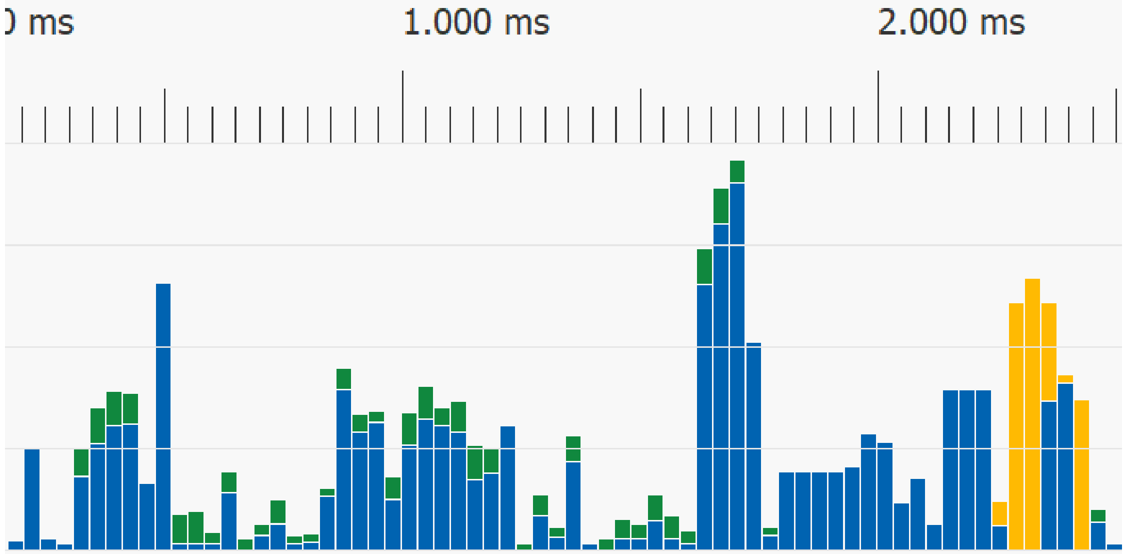
BARRIERS – OPTIMIZED



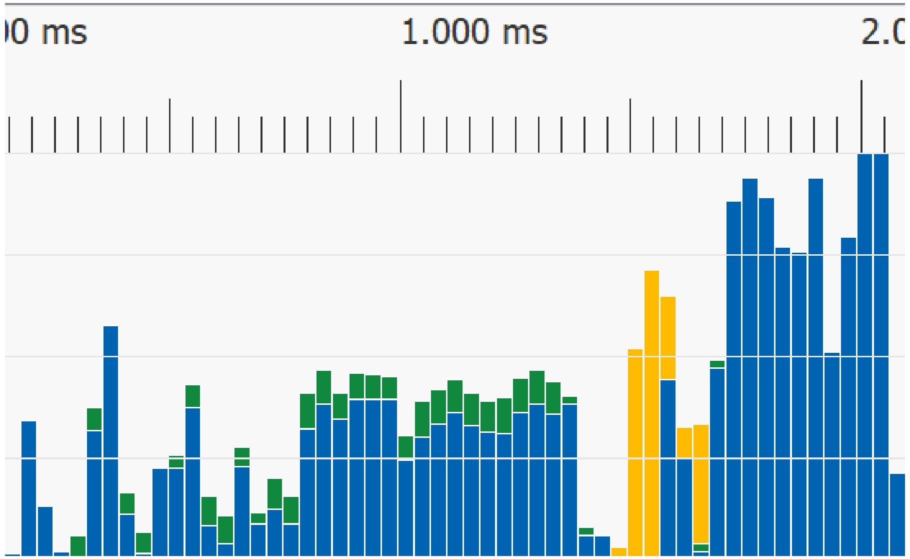
~15%
-->



BARRIERS – OPTIMIZED

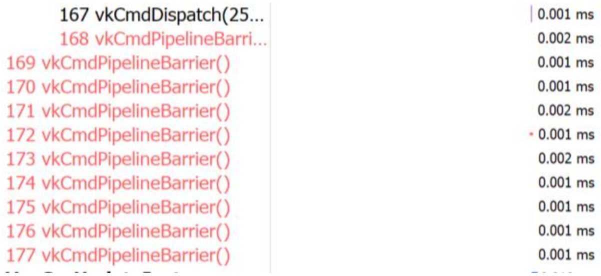


~15%
-->



BARRIER BATCHING

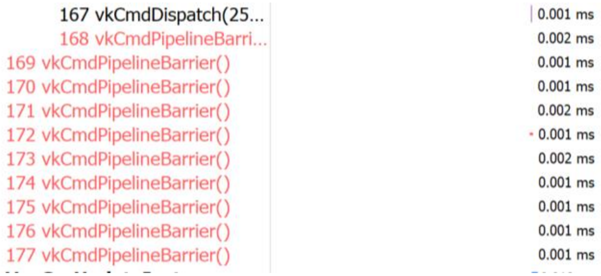
Early builds had several consecutive barriers:



167 vkCmdDispatch(25...	0.001 ms
168 vkCmdPipelineBarri...	0.002 ms
169 vkCmdPipelineBarrier()	0.001 ms
170 vkCmdPipelineBarrier()	0.001 ms
171 vkCmdPipelineBarrier()	0.002 ms
172 vkCmdPipelineBarrier()	0.001 ms
173 vkCmdPipelineBarrier()	0.002 ms
174 vkCmdPipelineBarrier()	0.001 ms
175 vkCmdPipelineBarrier()	0.001 ms
176 vkCmdPipelineBarrier()	0.001 ms
177 vkCmdPipelineBarrier()	0.001 ms

BARRIER BATCHING

Early builds had several consecutive barriers:

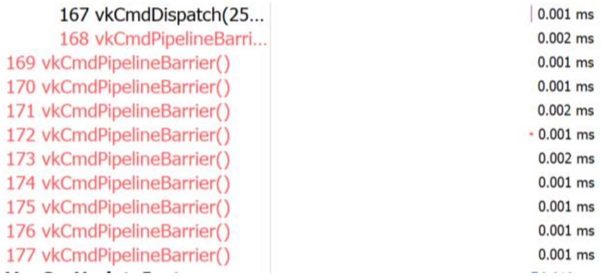


```
    167 vkCmdDispatch(25...            0.001 ms
    168 vkCmdPipelineBarri...          0.002 ms
169 vkCmdPipelineBarrier()             0.001 ms
170 vkCmdPipelineBarrier()             0.001 ms
171 vkCmdPipelineBarrier()             0.002 ms
172 vkCmdPipelineBarrier()             0.001 ms
173 vkCmdPipelineBarrier()             0.002 ms
174 vkCmdPipelineBarrier()             0.001 ms
175 vkCmdPipelineBarrier()             0.001 ms
176 vkCmdPipelineBarrier()             0.001 ms
177 vkCmdPipelineBarrier()             0.001 ms
```

```
void vkCmdPipelineBarrier(
VkCommandBuffer                commandBuffer,
VkPipelineStageFlags           srcStageMask,
VkPipelineStageFlags           dstStageMask,
VkDependencyFlags              dependencyFlags,
uint32_t                       memoryBarrierCount,
const VkMemoryBarrier*         pMemoryBarriers,
uint32_t                       bufferMemoryBarrierCount,
const VkBufferMemoryBarrier*   pBufferMemoryBarriers,
uint32_t                       imageMemoryBarrierCount,
const VkImageMemoryBarrier*    pImageMemoryBarriers);
```

BARRIER BATCHING

Early builds had several consecutive barriers:



```
167 vkCmdDispatch(25...          0.001 ms
168 vkCmdPipelineBarri...        0.002 ms
169 vkCmdPipelineBarrier()       0.001 ms
170 vkCmdPipelineBarrier()       0.001 ms
171 vkCmdPipelineBarrier()       0.002 ms
172 vkCmdPipelineBarrier()       0.001 ms
173 vkCmdPipelineBarrier()       0.002 ms
174 vkCmdPipelineBarrier()       0.001 ms
175 vkCmdPipelineBarrier()       0.001 ms
176 vkCmdPipelineBarrier()       0.001 ms
177 vkCmdPipelineBarrier()       0.001 ms
```

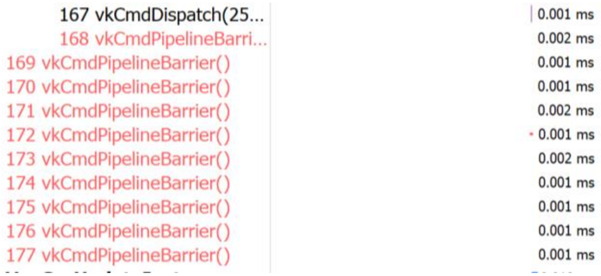
```
void vkCmdPipelineBarrier(
VkCommandBuffer                    commandBuffer,
VkPipelineStageFlags               srcStageMask,
VkPipelineStageFlags               dstStageMask,
VkDependencyFlags                  dependencyFlags,
uint32_t                           memoryBarrierCount,
const VkMemoryBarrier*             pMemoryBarriers,
uint32_t                           bufferMemoryBarrierCount,
const VkBufferMemoryBarrier*       pBufferMemoryBarriers,
uint32_t                           imageMemoryBarrierCount,
const VkImageMemoryBarrier*        pImageMemoryBarriers);
```

Example: 2 image layout transitions

```
vkCmdPipelineBarrier(…, 0, NULL, 0, NULL, 1, &imageBarrierA);
vkCmdPipelineBarrier(…, 0, NULL, 0, NULL, 1, &imageBarrierB);
```

BARRIER BATCHING

Early builds had several consecutive barriers:



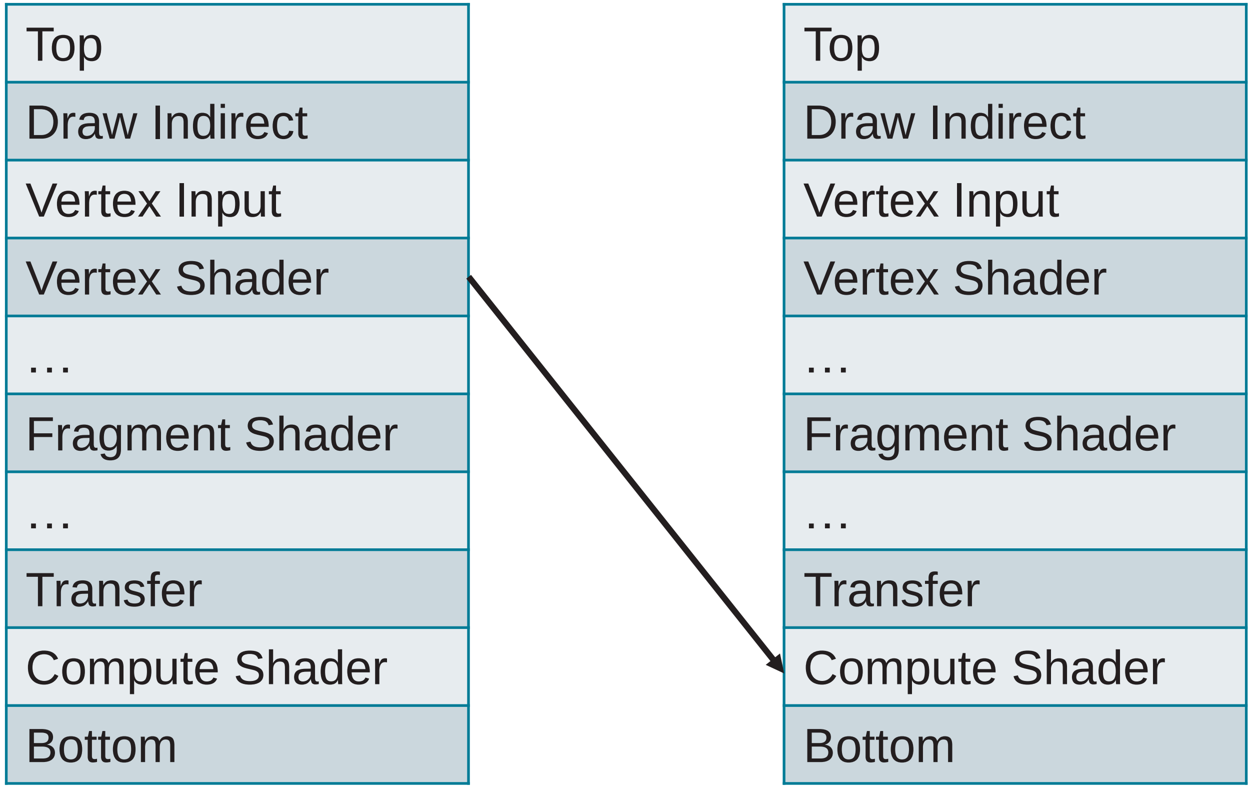
```
167 vkCmdDispatch(25...          0.001 ms
  168 vkCmdPipelineBarri...      0.002 ms
169 vkCmdPipelineBarrier()       0.001 ms
170 vkCmdPipelineBarrier()       0.001 ms
171 vkCmdPipelineBarrier()       0.002 ms
172 vkCmdPipelineBarrier()       0.001 ms
173 vkCmdPipelineBarrier()       0.002 ms
174 vkCmdPipelineBarrier()       0.001 ms
175 vkCmdPipelineBarrier()       0.001 ms
176 vkCmdPipelineBarrier()       0.001 ms
177 vkCmdPipelineBarrier()       0.001 ms
```

```
void vkCmdPipelineBarrier(
VkCommandBuffer                 commandBuffer,
VkPipelineStageFlags            srcStageMask,
VkPipelineStageFlags            dstStageMask,
VkDependencyFlags               dependencyFlags,
uint32_t                        memoryBarrierCount,
const VkMemoryBarrier*          pMemoryBarriers,
uint32_t                        bufferMemoryBarrierCount,
const VkBufferMemoryBarrier*    pBufferMemoryBarriers,
uint32_t                        imageMemoryBarrierCount,
const VkImageMemoryBarrier*     pImageMemoryBarriers);
```

Example: 2 image layout transitions

```
  vkCmdPipelineBarrier(…, 0, NULL, 0, NULL, 1, &imageBarrierA);
  vkCmdPipelineBarrier(…, 0, NULL, 0, NULL, 1, &imageBarrierB);
->
  VkImageMemoryBarrier[2] imageBarriers = {imageBarrierA, imageBarrierB};
  vkCmdPipelineBarrier(…, 0, NULL, 0, NULL, 2, &imageBarriers);
```

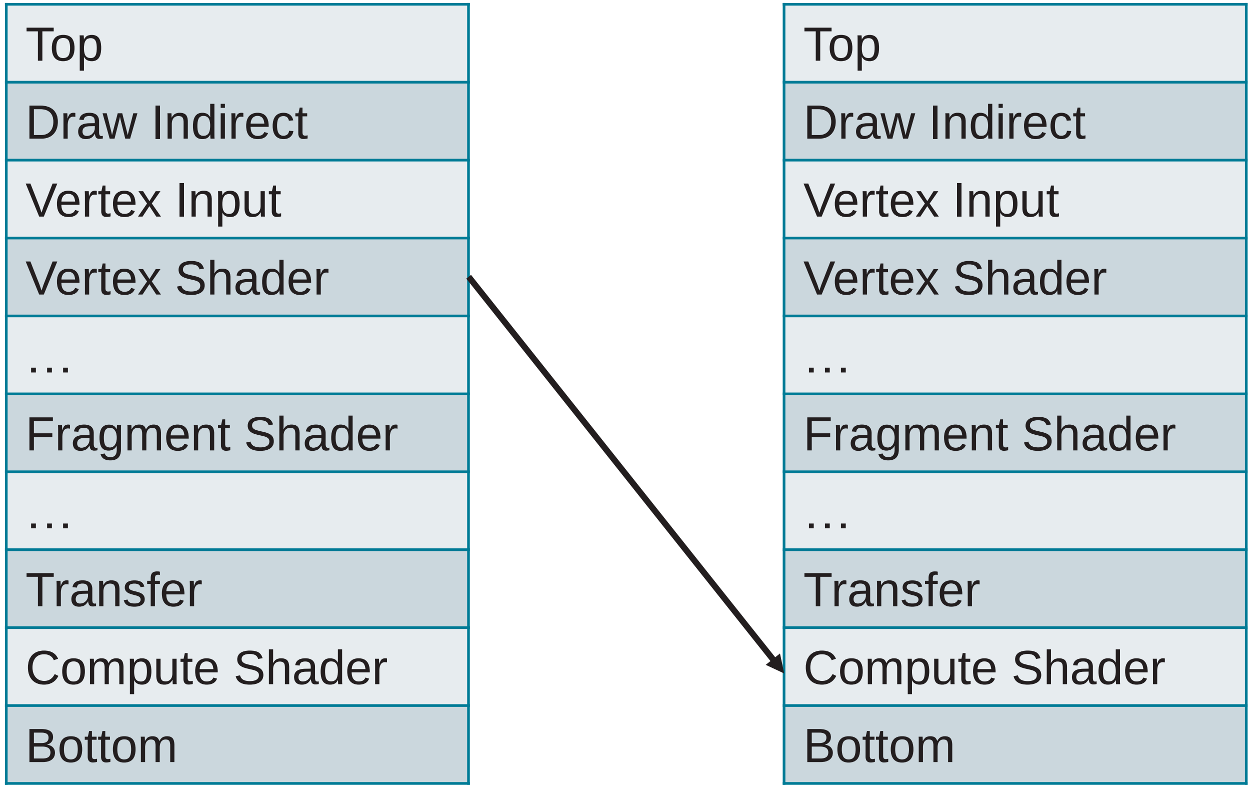
PIPELINE STAGE MASKS



Top
Draw Indirect
Vertex Input
Vertex Shader
…
Fragment Shader
…
Transfer
Compute Shader
Bottom

Top
Draw Indirect
Vertex Input
Vertex Shader
…
Fragment Shader
…
Transfer
Compute Shader
Bottom

PIPELINE STAGE MASKS



ALL_COMMANDS_BIT

Spec:

> “VK_PIPELINE_STAGE_ALL_COMMANDS_BIT is equivalent to the logical OR of every other pipeline stage flag that is supported on the queue it is used with.”

ALL_COMMANDS_BIT – COMPUTE PIPELINE

Top
Draw Indirect
Vertex Input
Vertex Shader
…
Fragment Shader
…
Transfer
Compute Shader
Bottom

ALL_COMMANDS_BIT

Spec:

> “VK_PIPELINE_STAGE_ALL_COMMANDS_BIT is equivalent to the logical OR of every other pipeline stage flag that is supported on the queue it is used with.”

ALL_COMMANDS_BIT – COMPUTE PIPELINE

Top
Draw Indirect
Vertex Input
Vertex Shader
…
Fragment Shader
…
Transfer
Compute Shader
Bottom

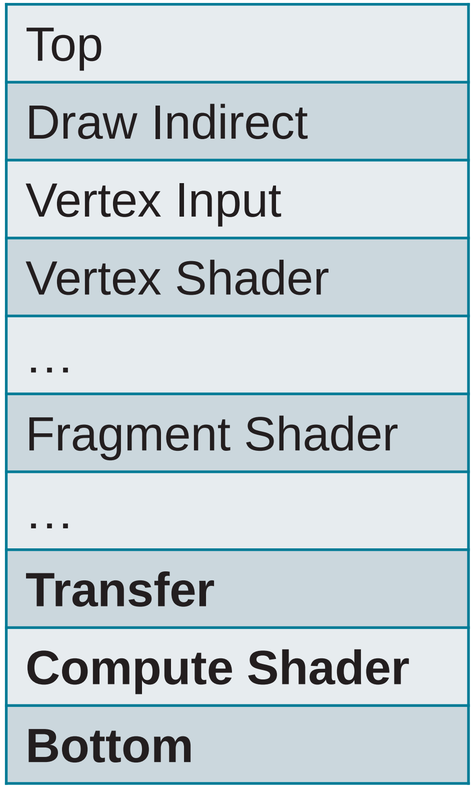
ALL_COMMANDS_BIT

Spec:

> “VK_PIPELINE_STAGE_ALL_COMMANDS_BIT is equivalent to the logical OR of every other pipeline stage flag that is supported on the queue it is used with.”

The bottom bit adds a wait on end of pipe + timestamp
-> can take up to ~64k cycles on the async queue ☹

ALL_COMMANDS_BIT – COMPUTE PIPELINE



Top

Draw Indirect

Vertex Input

Vertex Shader

…

Fragment Shader

…

Transfer

Compute Shader

Bottom

ALL_COMMANDS_BIT

Spec:

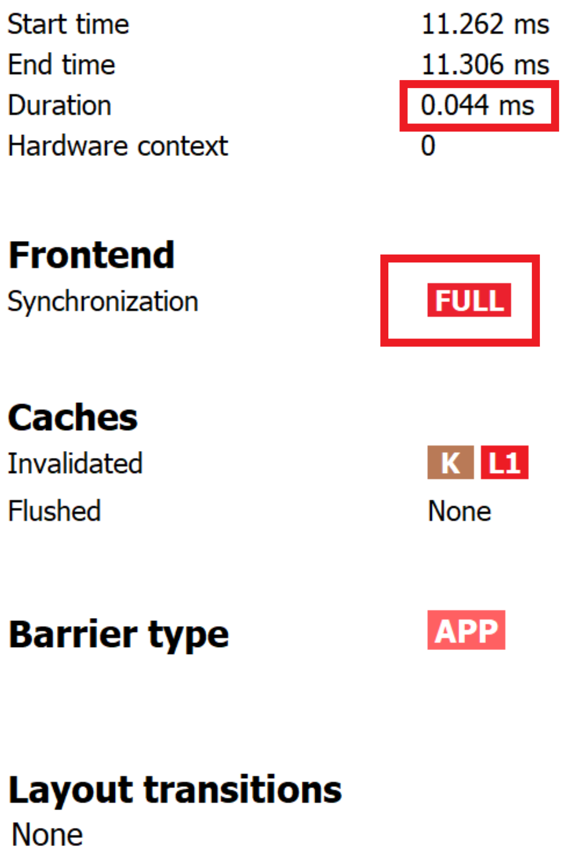
> “VK_PIPELINE_STAGE_ALL_COMMANDS_BIT is equivalent to the logical OR of every other pipeline stage flag that is supported on the queue it is used with.”

->

Use the specific pipeline stage mask instead of all_commands, e.g.:
VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT |
VK_PIPELINE_STAGE_TRANSFER_BIT

The bottom bit adds a wait on end of pipe + timestamp
-> can take up to ~64k cycles on the async queue ☹

ALL_COMMANDS_BIT – COMPUTE PIPELINE



Start time	11.262 ms
End time	11.306 ms
Duration	0.044 ms
Hardware context	0

Frontend

Synchronization	FULL

Caches

Invalidated	K L1
Flushed	None

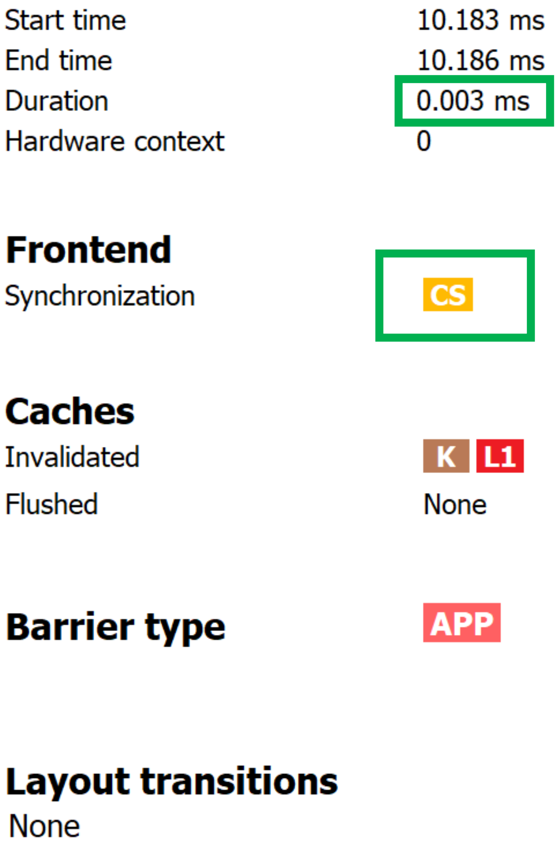
Barrier type	APP

Layout transitions

None

-> VK_PIPELINE_STAGE_ALL_COMMANDS_BIT on async compute queue

ALL_COMMANDS_BIT – COMPUTE PIPELINE



Start time	10.183 ms
End time	10.186 ms
Duration	0.003 ms
Hardware context	0

Frontend

Synchronization	CS

Caches

Invalidated	K L1
Flushed	None

Barrier type APP

Layout transitions

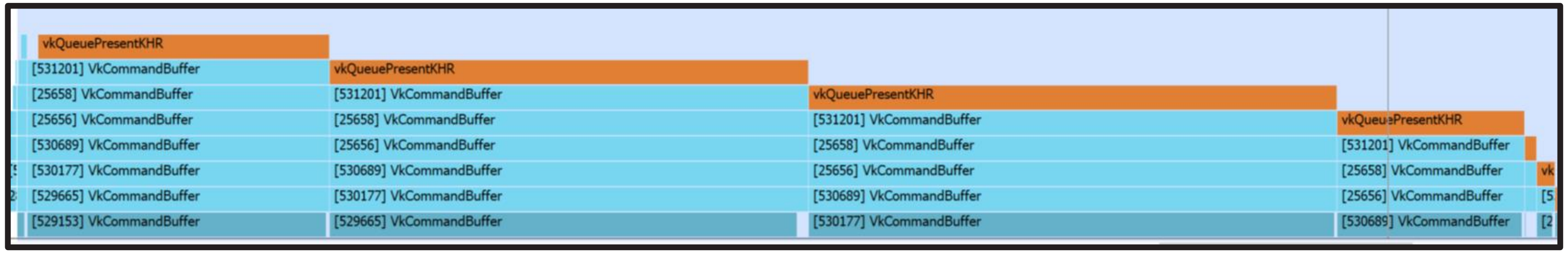
None

->

VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT |
VK_PIPELINE_STAGE_TRANSFER_BIT
on async compute queue

CROSS QUEUE SYNCHRONIZATION

The engine used to have ~7 command buffers per frame



vkQueuePresentKHR
[531201] VkCommandBuffer
[25658] VkCommandBuffer
[25656] VkCommandBuffer
[530689] VkCommandBuffer
[530177] VkCommandBuffer
[529665] VkCommandBuffer
[529153] VkCommandBuffer

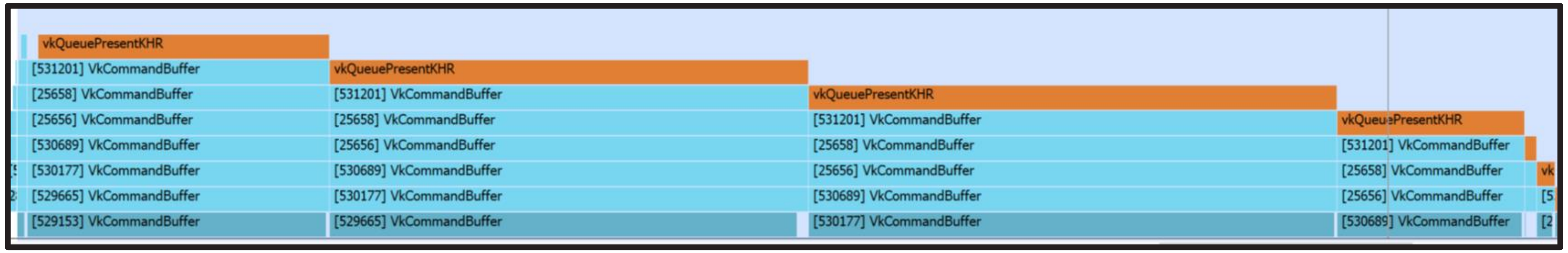
vkQueuePresentKHR
[531201] VkCommandBuffer
[25658] VkCommandBuffer
[25656] VkCommandBuffer
[530689] VkCommandBuffer
[530177] VkCommandBuffer
[529665] VkCommandBuffer

vkQueuePresentKHR
[531201] VkCommandBuffer
[25658] VkCommandBuffer
[25656] VkCommandBuffer
[530689] VkCommandBuffer
[530177] VkCommandBuffer

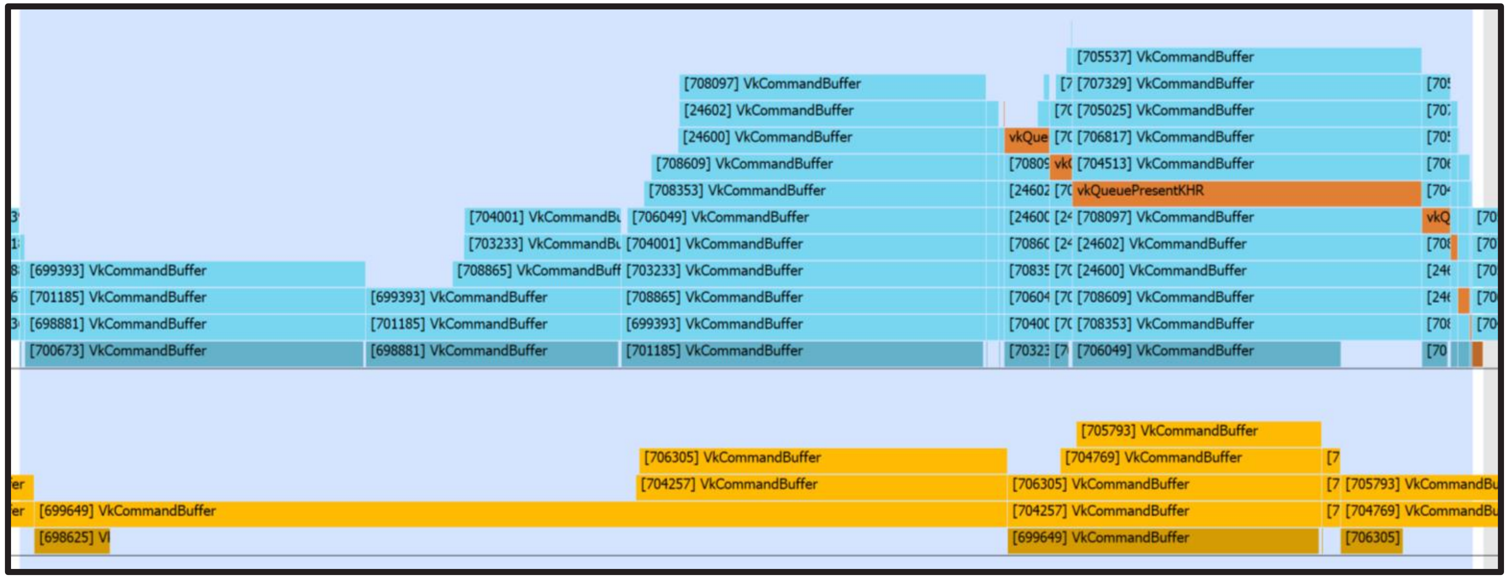
vkQueuePresentKHR
[531201] VkCommandBuffer
[25658] VkCommandBuffer
[25656] VkCommandBuffer
[530689] VkCommandBuffer

CROSS QUEUE SYNCHRONIZATION

The engine used to have ~7 command buffers per frame

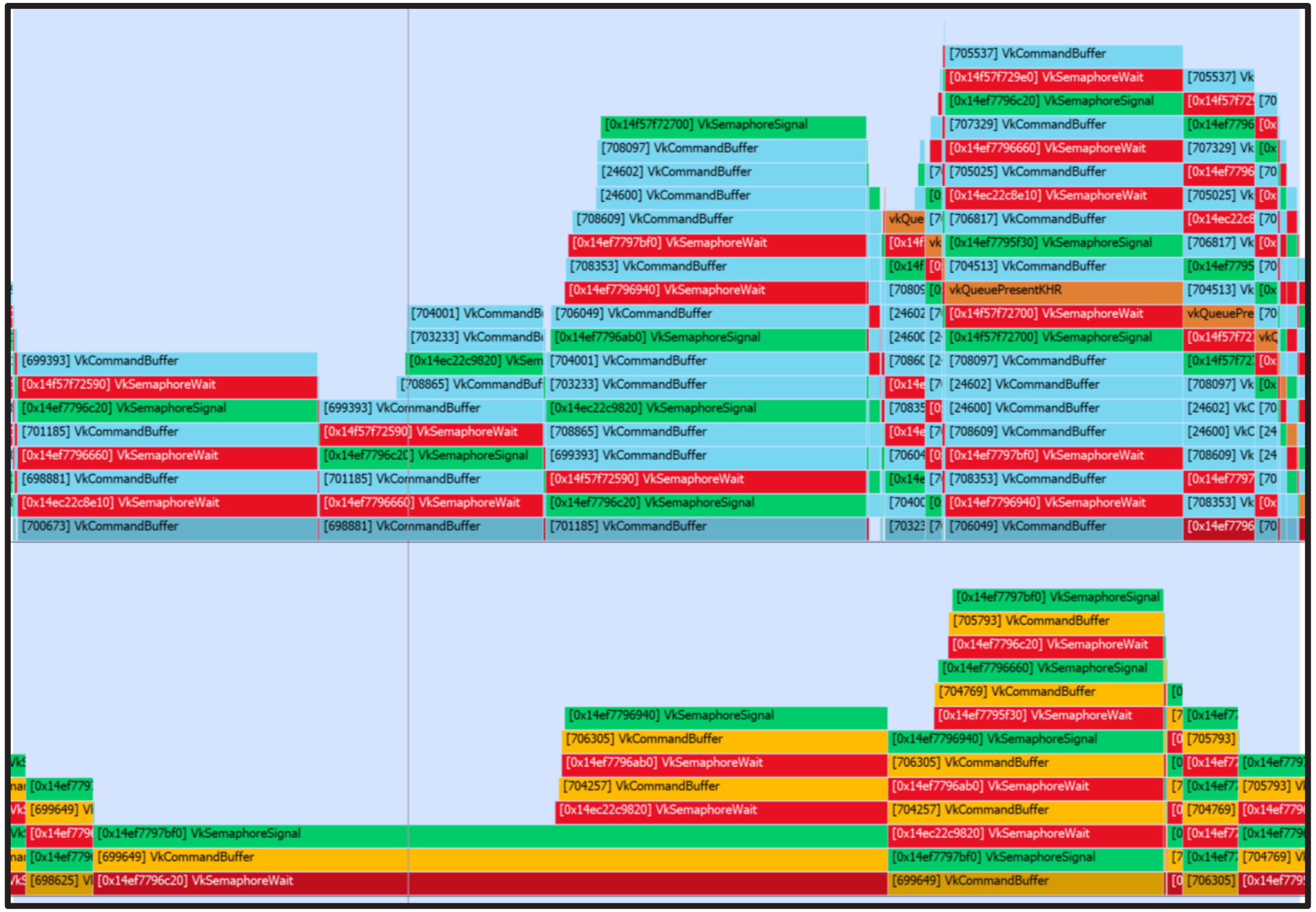


After async compute support was added, the number of command buffers doubled



CROSS QUEUE SYNCHRONIZATION

Cross queue synchronization is only possible at submission boundaries


[705537] VkCommandBuffer
[0x14f57f729e0] VkSemaphoreWait
[0x14ef7796c20] VkSemaphoreSignal
[707329] VkCommandBuffer
[0x14ef7796660] VkSemaphoreWait
[705025] VkCommandBuffer
[0x14ec22c8e10] VkSemaphoreWait
[706817] VkCommandBuffer
[0x14ef7795f30] VkSemaphoreSignal
[704513] VkCommandBuffer
vkQueuePresentKHR
[0x14f57f72700] VkSemaphoreWait
[0x14f57f72700] VkSemaphoreSignal
[708097] VkCommandBuffer
[24602] VkCommandBuffer
[24600] VkCommandBuffer
[708609] VkCommandBuffer
[0x14ef7797bf0] VkSemaphoreWait
[708353] VkCommandBuffer
[0x14ef7796940] VkSemaphoreWait
[706049] VkCommandBuffer
[0x14f57f72700] VkSemaphoreSignal
[708097] VkCommandBuffer
[24602] VkCommandBuffer
[24600] VkCommandBuffer
[708609] VkCommandBuffer
[0x14ef7797bf0] VkSemaphoreWait
[708353] VkCommandBuffer
[0x14ef7796940] VkSemaphoreWait
[706049] VkCommandBuffer
[0x14ef7796ab0] VkSemaphoreSignal
[704001] VkCommandBuffer
[703233] VkCommandBuffer
[0x14ec22c9820] VkSemaphoreSignal
[708865] VkCommandBuffer
[699393] VkCommandBuffer
[0x14f57f72590] VkSemaphoreWait
[0x14ef7796c20] VkSemaphoreSignal
[701185] VkCommandBuffer
[699393] VkCommandBuffer
[0x14f57f72590] VkSemaphoreWait
[0x14ef7796c20] VkSemaphoreSignal
[701185] VkCommandBuffer
[0x14ef7796660] VkSemaphoreWait
[698881] VkCommandBuffer
[0x14ec22c8e10] VkSemaphoreWait
[700673] VkCommandBuffer
[0x14ef7797bf0] VkSemaphoreSignal
[705793] VkCommandBuffer
[0x14ef7796c20] VkSemaphoreWait
[0x14ef7796660] VkSemaphoreSignal
[704769] VkCommandBuffer
[0x14ef7795f30] VkSemaphoreWait
[0x14ef7796940] VkSemaphoreSignal
[706305] VkCommandBuffer
[0x14ef7796ab0] VkSemaphoreWait
[704257] VkCommandBuffer
[0x14ec22c9820] VkSemaphoreWait
[0x14ef7797bf0] VkSemaphoreSignal
[699649] VkCommandBuffer
[0x14ef7796c20] VkSemaphoreWait

SUMMARY

- Check your barriers if you can rearrange them
- Batch consecutive barriers to a single barrier
- Specify your barriers as precise as possible
- Cross queue synchronization is only possible at submission boundaries

OTHER SMALL THINGS

- Copy queue
- Compute queue & the swapchain
- Shader building infrastructure

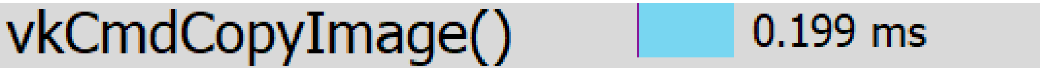
COPY QUEUE

Resource was copied from GPU to CPU

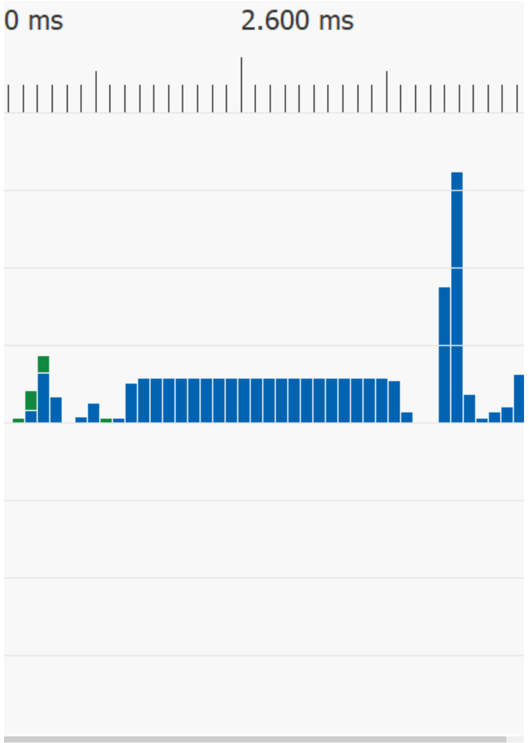
- Generated on GPU during previous frame
- After the copy overwritten with updated data from current frame

This copy blocked the whole GPU.

-> ~1-2% of frame time

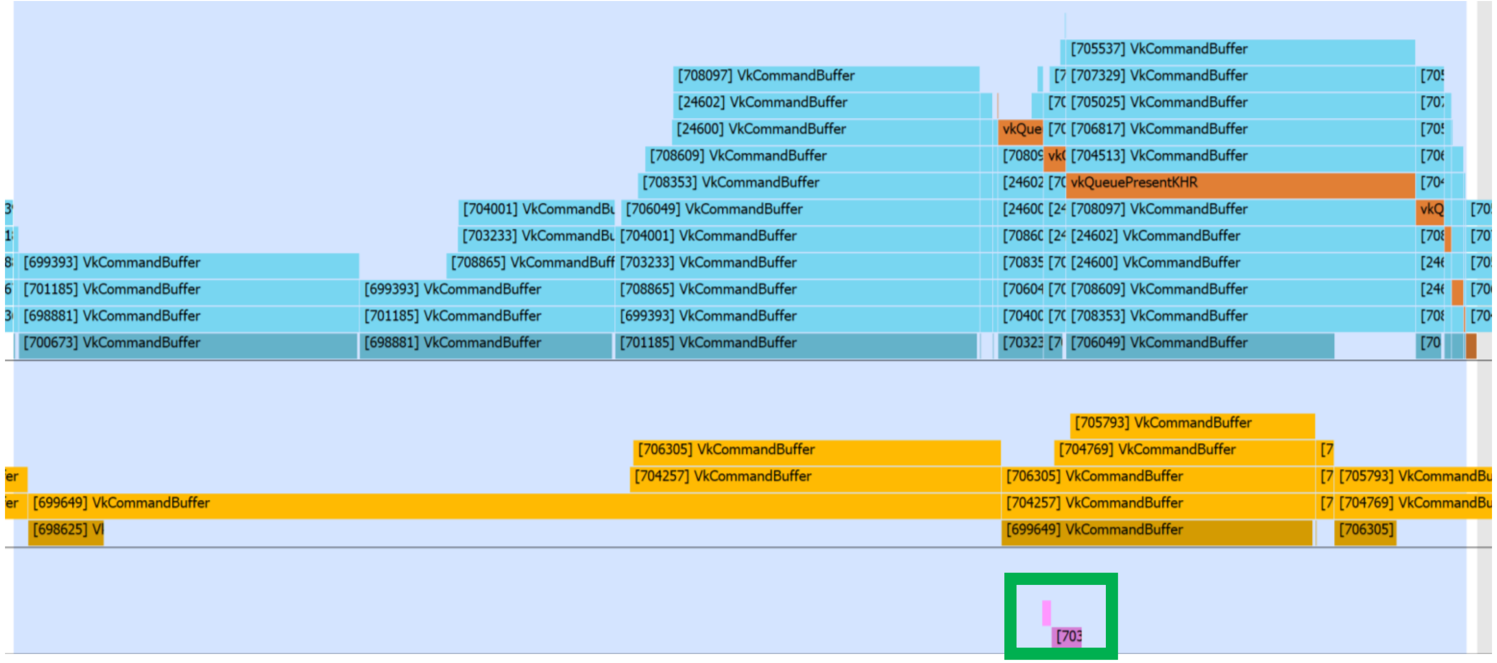


vkCmdCopyImage() 0.199 ms



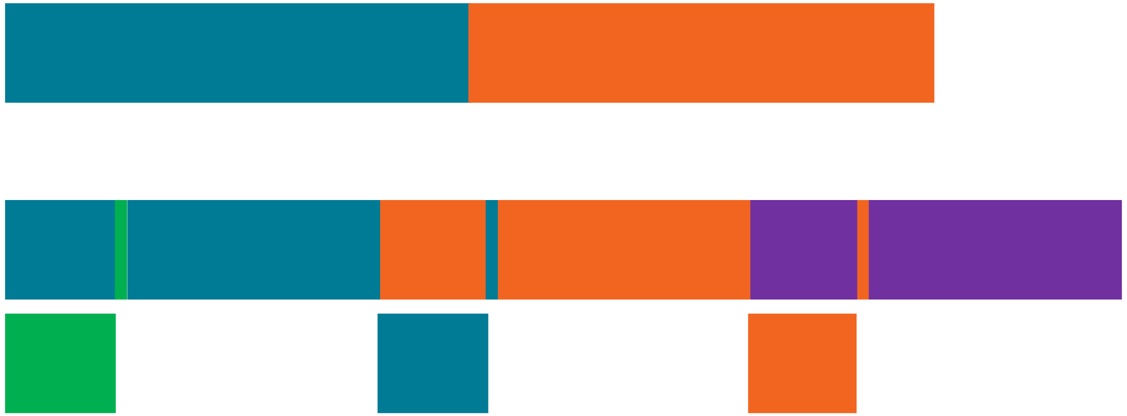
COPY QUEUE

By using the copy queue, we won the time previously spend for vkCmdCopyImage() back.



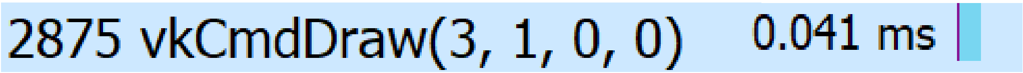
COMPUTE QUEUE & SWAPCHAIN

Write directly from compute to the swapchain

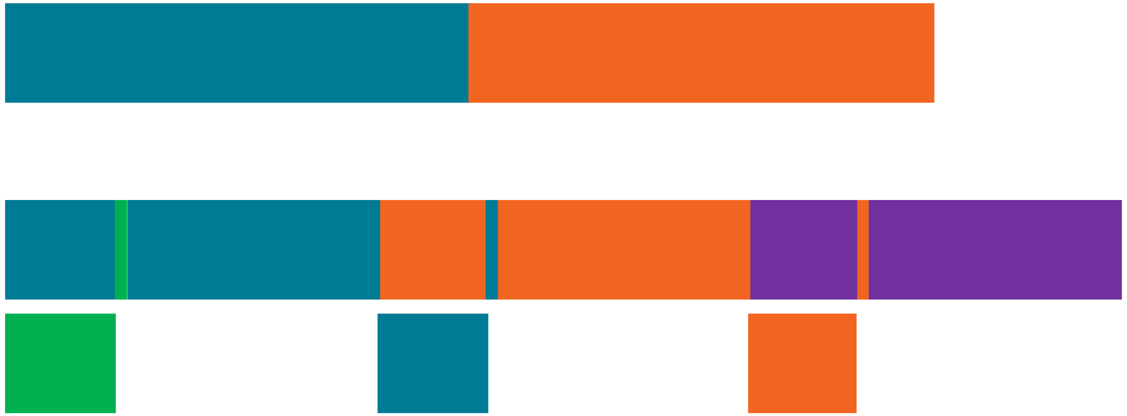


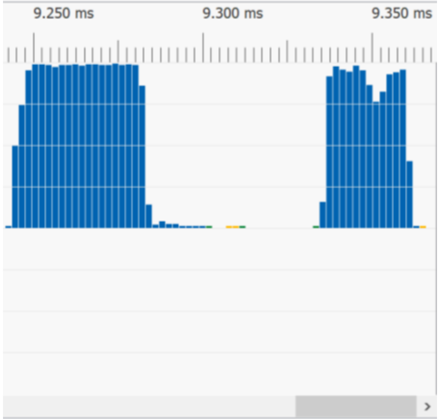
COMPUTE QUEUE & SWAPCHAIN

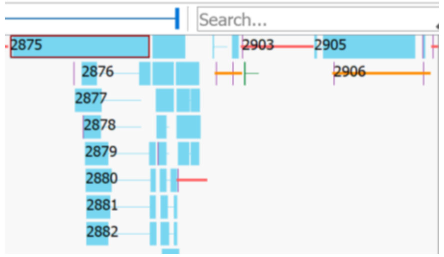
Write directly from compute to the swapchain



2875 vkCmdDraw(3, 1, 0, 0) 0.041 ms



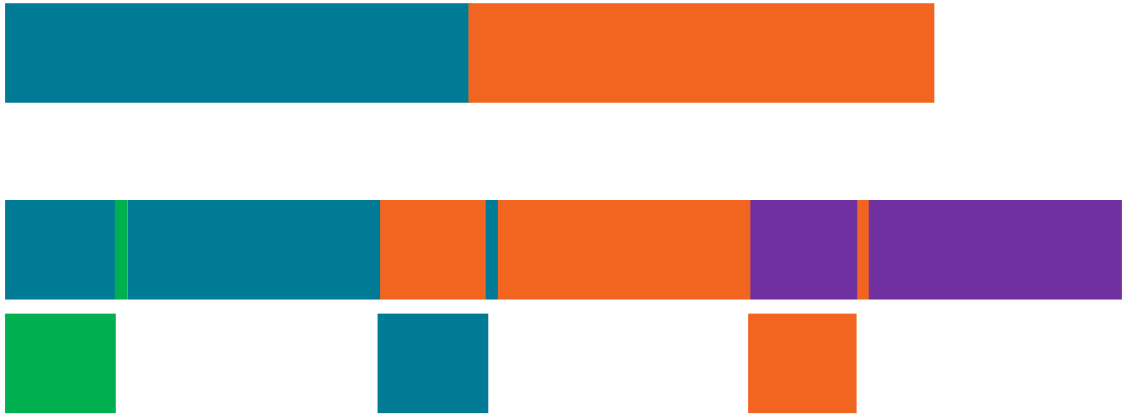




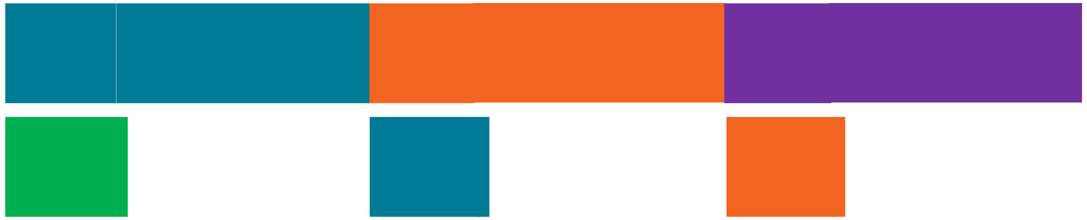
COMPUTE QUEUE & SWAPCHAIN

Write directly from compute to the swapchain

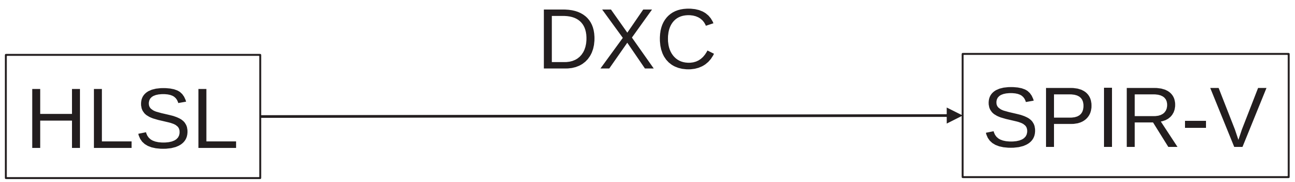
Possibly present from compute



Vulkan specific feature

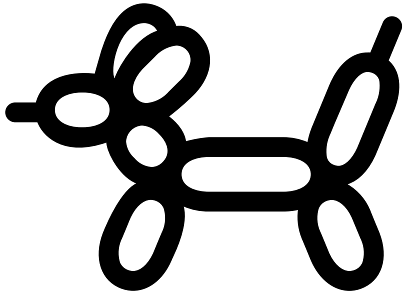


SHADER BUILDING INFRASTRUCTURE



SUMMARY

- Check for compression, especially for the G-buffer render targets
- Take special care of the barriers ☺
- Can you make good use of the copy queue?
- The compute queue can write directly to the swapchain
- Use the DXC compiler



THANKS TO

Dominik Baumeister

Matthäus Chajdas

Tobias Hector

Adam Sawicki

Rys Sommefeldt

Steven Tovey

Marco Weber

REFERENCES

https://www.khronos.org/registry/vulkan/specs/1.1-extensions/html/

https://gpuopen.com/dcc-overview/

https://gpuopen.com/vulkan-barriers-explained/

https://github.com/GPUOpen-LibrariesAndSDKs/VulkanMemoryAllocator

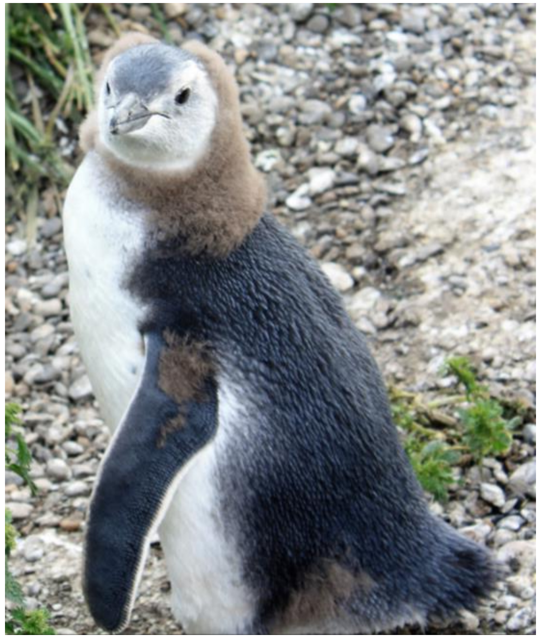
https://gpuopen.com/reducing-vulkan-api-call-overhead/

Q&A

✉ lou.kramer@amd.com

@lou_auroyup

https://gpuopen.com/



DISCLAIMER & ATTRIBUTION

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

AMD