VULKAN TECHNOLOGY UPDATE

Christoph Kubisch, NVIDIA   GTC 2017
Ingo Esser, NVIDIA

NVIDIA

AGENDA

Device Generated Commands

API Interop

VR in Vulkan

NSIGHT Support

VK_NVX_device_generated_commands

DEVICE GENERATED COMMANDS

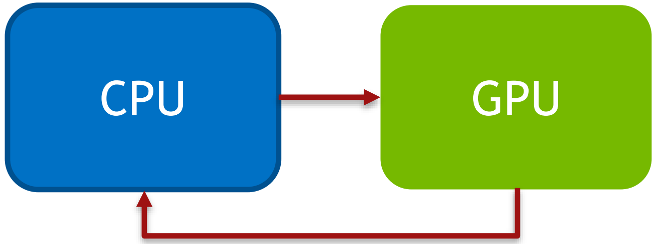
GPU creates its own work (drawcalls and compute)

Define the work-load in-pipeline, in-frame

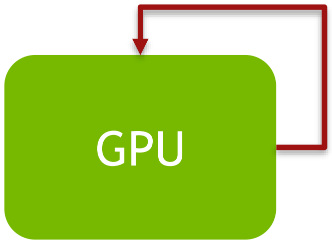
Reduce latency as no CPU roundtrip is required (VR!)

Use any GPU accessible resources to drive decision making (zbuffer etc.)

Select level of detail, cull by occlusion, classify work into different state usage, …



1-2 frames latency



DEVICE GENERATED COMMANDS

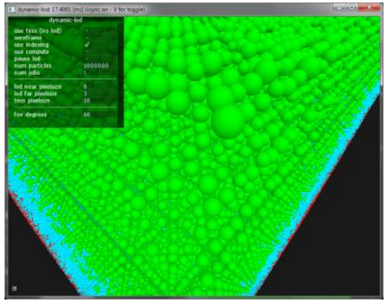
OpenGL Examples

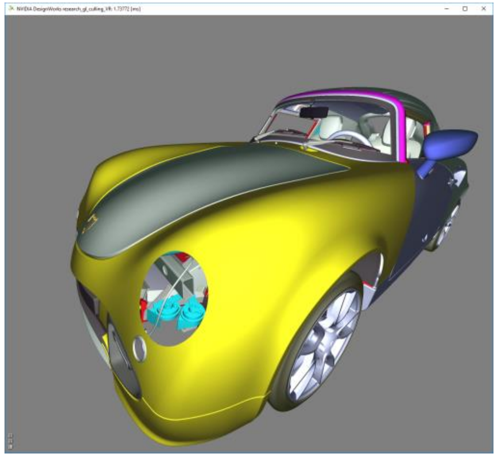
https://github.com/nvpro-samples/gl_dynamic_lod

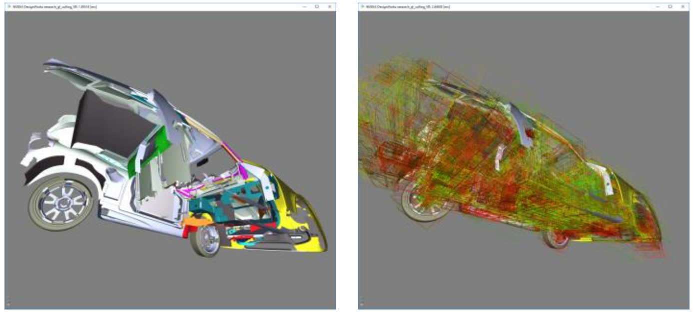
ARB_draw_indirect to classify how particles are drawn (point, mesh, tessellation)

https://github.com/nvpro-samples/gl_occlusion_culling

ARB_multi_draw_indirect / NV_command_list to do shader-based occlusion culling

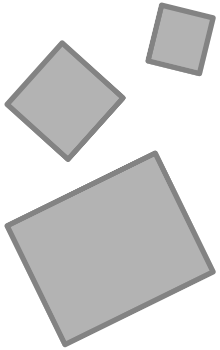






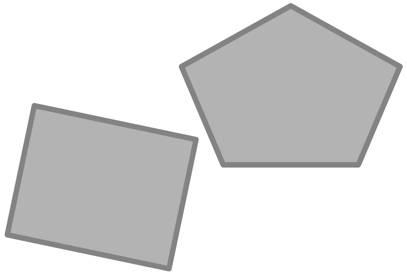
Reverse angle & bboxes of culled
Model courtesy of PGO Automobiles

EVOLUTION

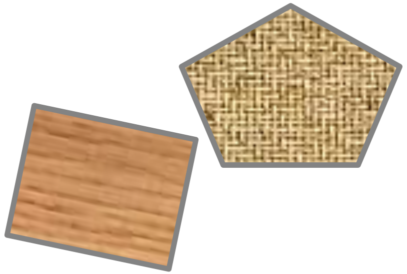


Draw Indirect:
Typically change
primitives,
instances

```
DrawElements
{
  GLuint     indexCount;
  GLuint     instanceCount;
  GLuint     firstIndex;
  GLuint     baseVertex;
  GLuint     baseInstance;
}
```

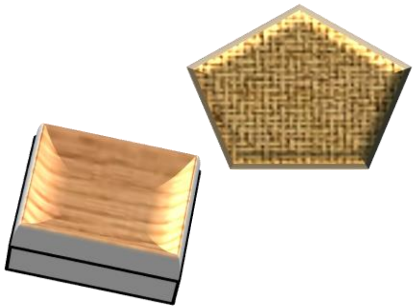


Multi Draw Indirect:
Multiple draw calls with different index/vertex offsets



GL_NV_command_list & DX12 ExecuteIndirect:
Change shader input bindings for each draw

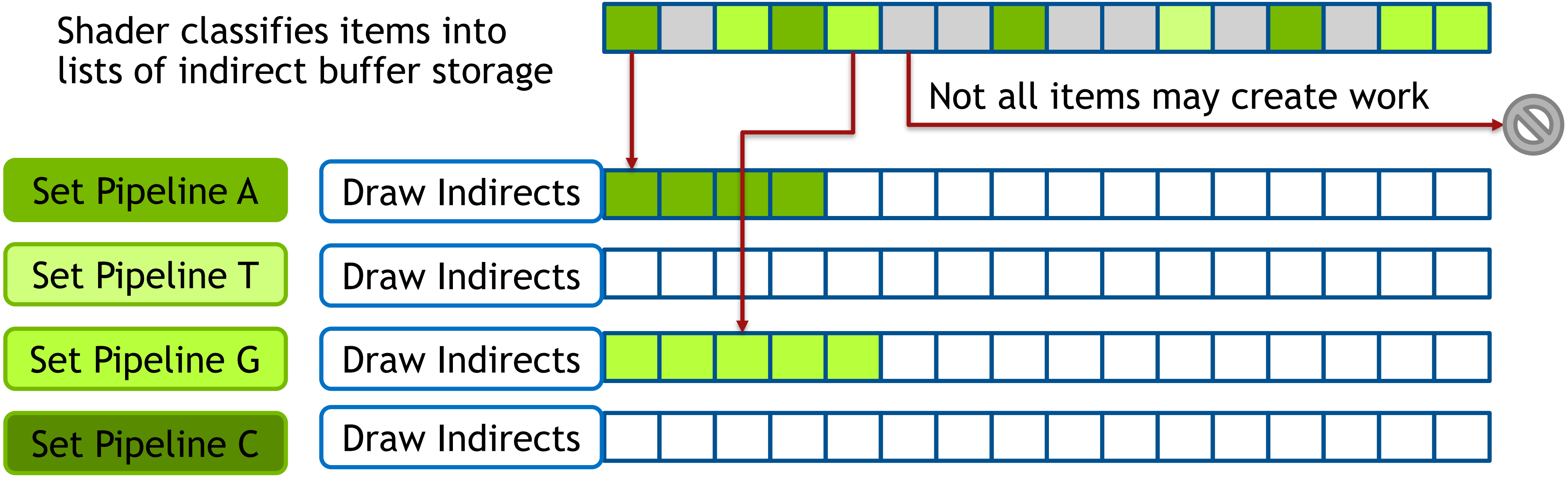
```
UniformAddressCommandNV
{
  GLuint     header;
  GLushort   index;
  GLushort   stage;
  GLuint64   address;
}
```



VK_NVX_device_generated_commands
Change shader (pipeline state) per draw call

```
DescriptorSetToken
{
  GLuint     objectTableIndex;
  Gluint     offsets[];
}
```

TRADITIONAL SETUP

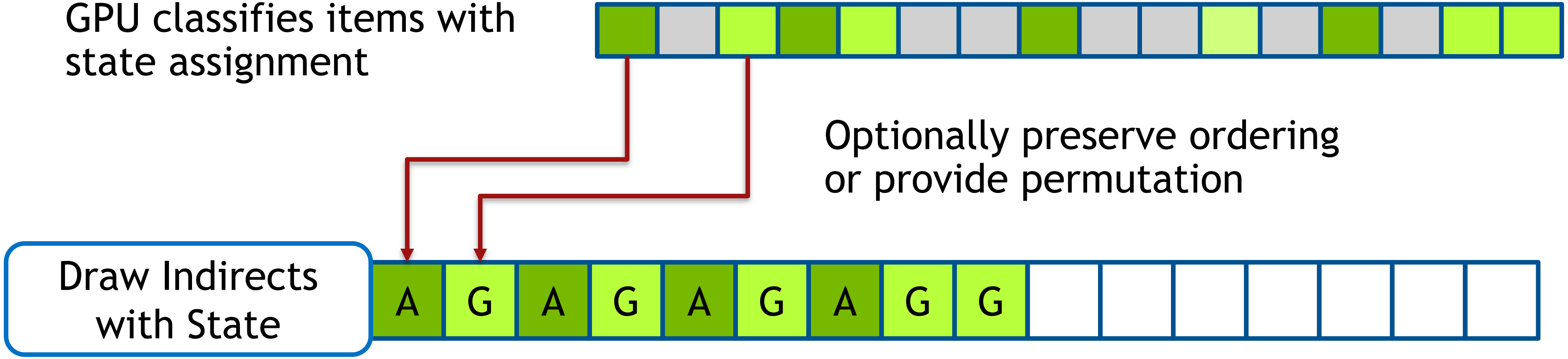


CPU-driven state setup is for worst-case distribution of indirect work

May yield lots of needless state setup (imagine 100s of potentially-used Pipelines)

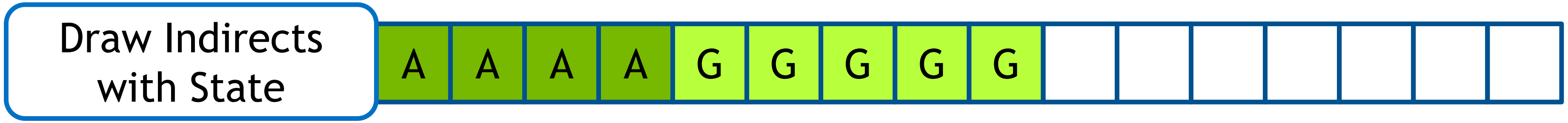
NEW VULKAN ABILITY

GPU classifies items with
state assignment



Compact stream without unnecessary state setup or data overfetching

Grouping by state is still recommended



PIPELINE CHANGES

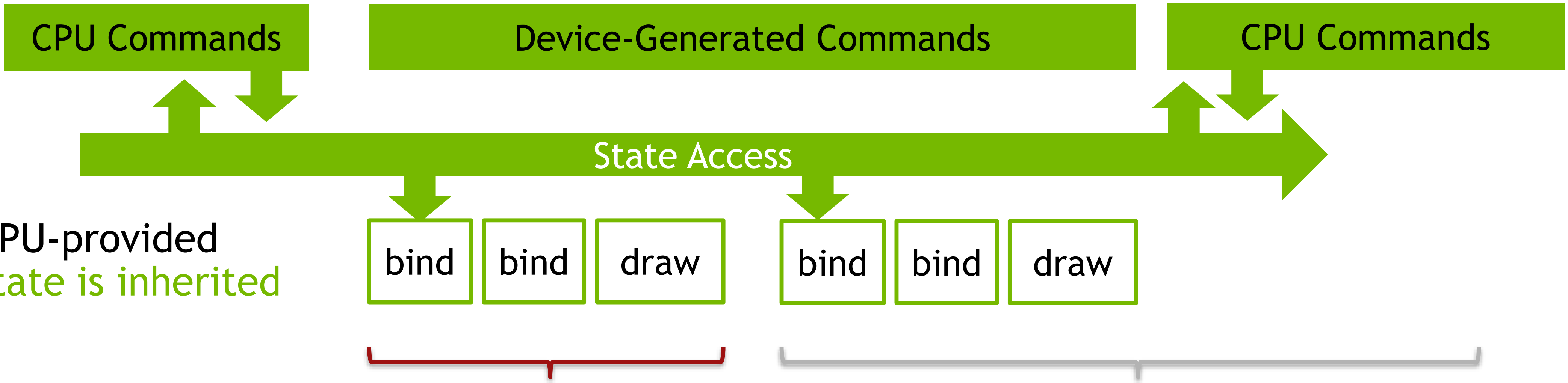
Add command-related work on the GPU to be more efficient at the actual tasks

Make use of shader specialization (less dynamic branching, more aggressive compile-time optimizations...)

Shader level of detail

Partition & organize work by shader permutation or usage pattern

STATELESS DESIGN

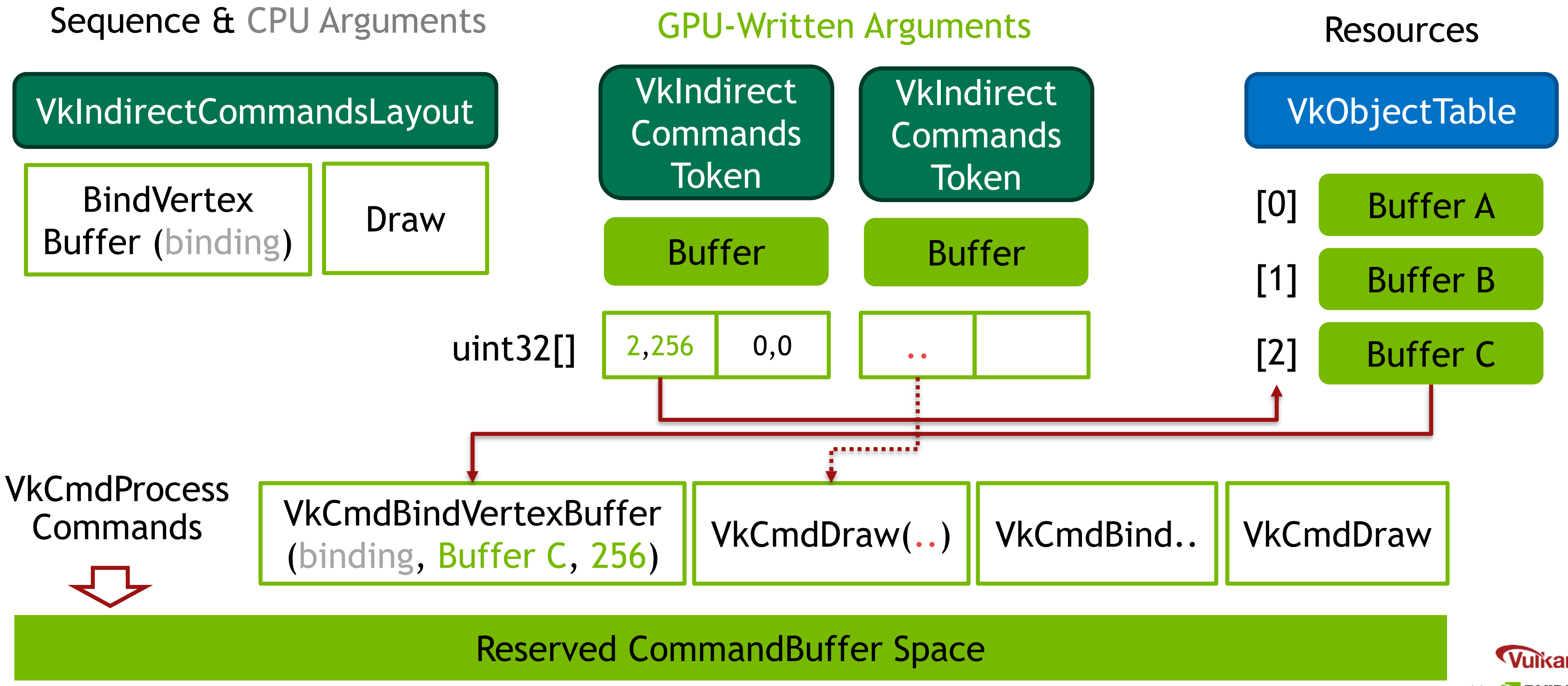


CPU-provided
state is inherited

Stateful within single command sequence

Modified state is undefined for subsequent sequences or CPU commands

OVERVIEW



Sequence & CPU Arguments
GPU-Written Arguments
Resources
VkIndirectCommandsLayout
BindVertex Buffer (binding)
Draw
VkIndirect Commands Token
VkIndirect Commands Token
Buffer
Buffer
uint32[]
2,256
0,0
..
VkObjectTable
[0]
Buffer A
[1]
Buffer B
[2]
Buffer C
VkCmdProcess Commands
VkCmdBindVertexBuffer (binding, Buffer C, 256)
VkCmdDraw(..)
VkCmdBind..
VkCmdDraw
Reserved CommandBuffer Space

WORKFLOW

Define a stateless sequence of commands as VkIndirectCommandsLayout

Register Vulkan resources (VkBuffer, VkDescriptorSet, VkPipeline) in VkObjectTable at developer-managed index

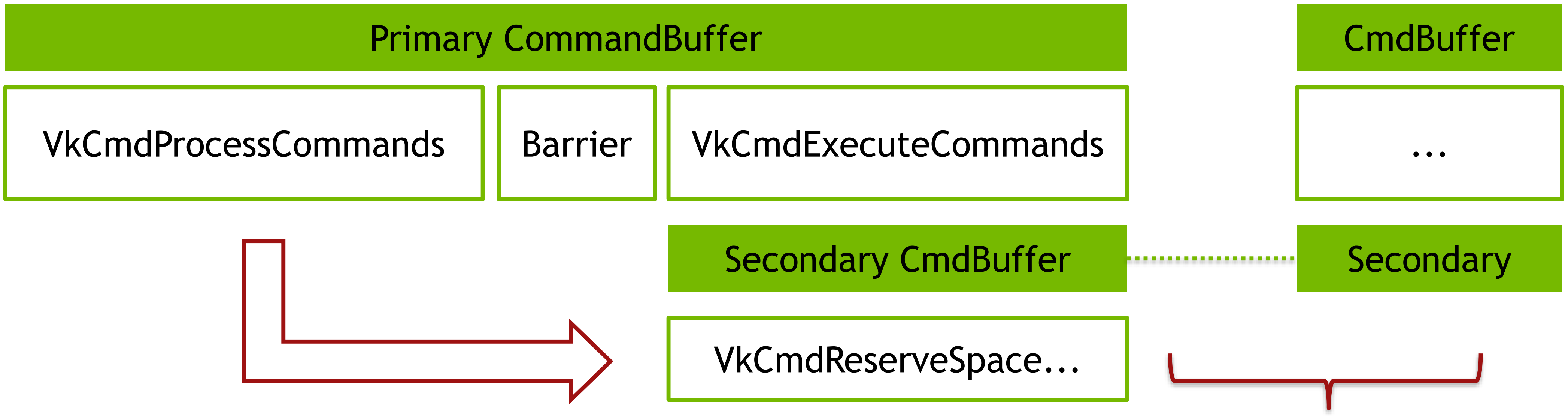
Fill & modify VkBuffers with command arguments and object table indices for many sequences

Use VkCmdReserveSpaceForCommands to allocate command buffer space

Generate the commands from token buffer content via VkCmdProcessCommands

Execute via VkCmdExecuteCommands

SEPARATE GENERATION & EXECUTION



Record an array of command sequences into the reserved space

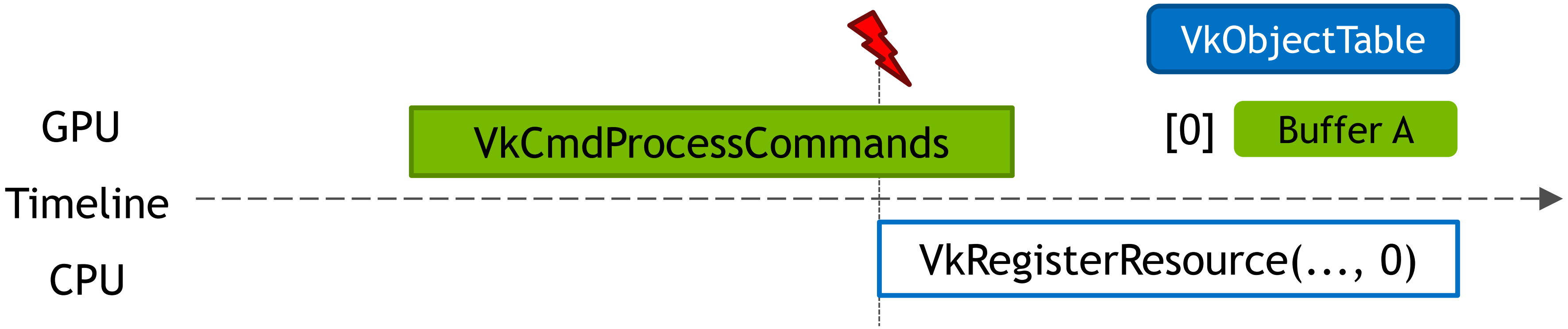
Generate & Execute as single action is also supported

Reuse commands, or reuse reserved space for another generation

OBJECT TABLE

ObjectTable behaves similar to DescriptorPool

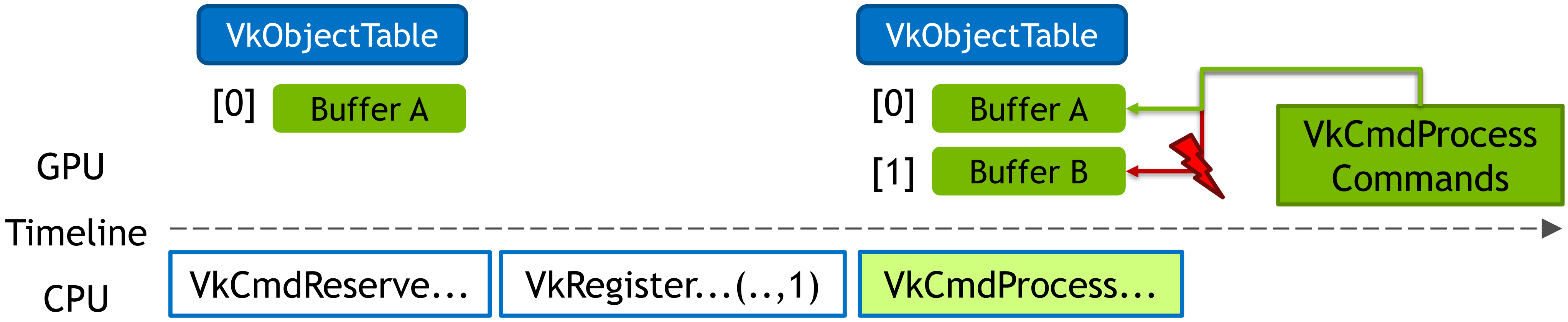
Do not delete it, nor modify resource indices that may be in-flight



OBJECT TABLE

CommandBuffer reservation depends on ObjectTable‘s state

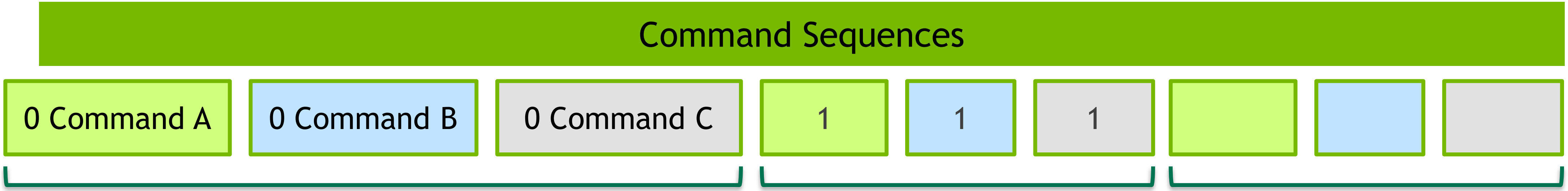
Use only those resources, that were registered at reservation time



INDIRECT COMMANDS

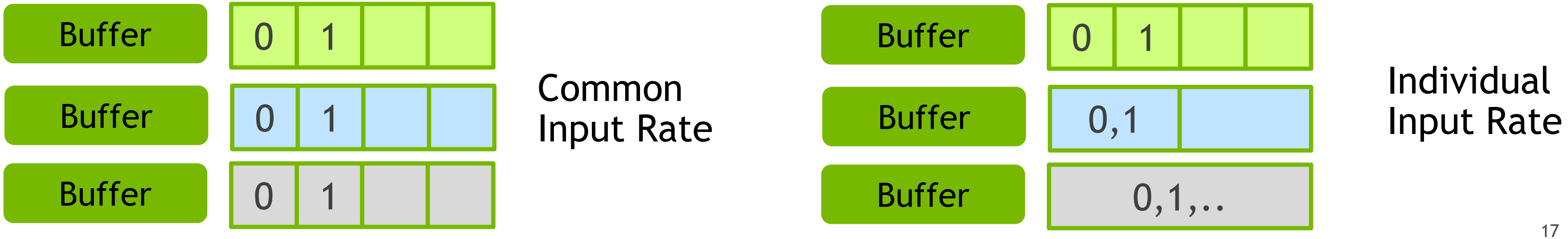
VK_INDIRECT_COMMANDS_TOKEN	EQUIVALENT COMMAND & GPU-WRITTEN ARGUMENTS
_PIPELINE_NVX	vkCmdBindPipeline(… pipeline)
_DESCRIPTOR_SET_NVX	vkCmdBindDescriptorSets(… descrSet, offsets)
_INDEX_BUFFER_NVX	vkCmdBindIndexBuffer(… buffer, offset)
_VERTEX_BUFFER_NVX	vkCmdBindVertexBuffer (… buffer, offset)
_PUSH_CONSTANT_NVX	vkCmdPushConstants(… data)
_DRAW_INDEXED_NVX	vkCmdDrawIndexed(*all*)
_DRAW_NVX	VkCmdDraw(*all*)
_DISPATCH_NVX	VkCmdDispatch(*all*)

MULTIPLE INPUT STREAMS



Traditional approaches used single interleaved stream (array of structures AoS)

VK extension uses input streams (SoA), allows individual re-use and efficient updates on input



FLEXIBLE SEQUENCING

Ordered Sequences

0	1	2	3	4	5	6	7

Default monotonic order of command sequences

CPU Argument | **8**

Number of sequences by CPU

Unordered / Subset

3	2	0	1				

Allow impl.-dependent ordering (incoherent)

Buffer | **4**

Actual number provided by GPU Buffer

Custom Subset

2	5	1	4				

Provide sequence indices as additional GPU buffer

Buffer | **2** | **5** | **1** | **4**

Buffer | **4**

TEST BENCHMARK

200.000 Drawcalls (few triangles/lines)

45.000 Pipeline switches (lines vs triangles)

6 Tokens:
Pipeline
DescriptorSet (1 ubo + 1 offset)
DescriptorSet (1 ubo + 1 offset)
VertexBuffer + 1 offset
IndexBuffer + 1 offset
DrawIndexed

https://github.com/nvpro-samples/gl_vk_threaded_cadscene/blob/master/doc/vulkan_nvxdevicegenerated.md

TEST BENCHMARK

200 000 DRAWCALLS 45 000 PSO CHANGES	GENERATE	EXECUTE
Driver (CPU 1 thread)	8.74 ms (async, on CPU)	14.74 ms
Device Gen. Cmds	0.35 ms	8.12 ms

100 000 DRAWCALLS NO PSO	GENERATE	EXECUTE
Driver (CPU 1 thread)	3.8 ms (async, on CPU)	1.8 ms
Device Gen. Cmds	0.20 ms	1.8 ms

Test benchmark is very simplified scenario, your milage will vary

NVIDIA IMPLEMENTATION

Currently experimental extension, feedback welcome (design, performance etc.)

VkIndirectCommandsLayout generates internal compute shader

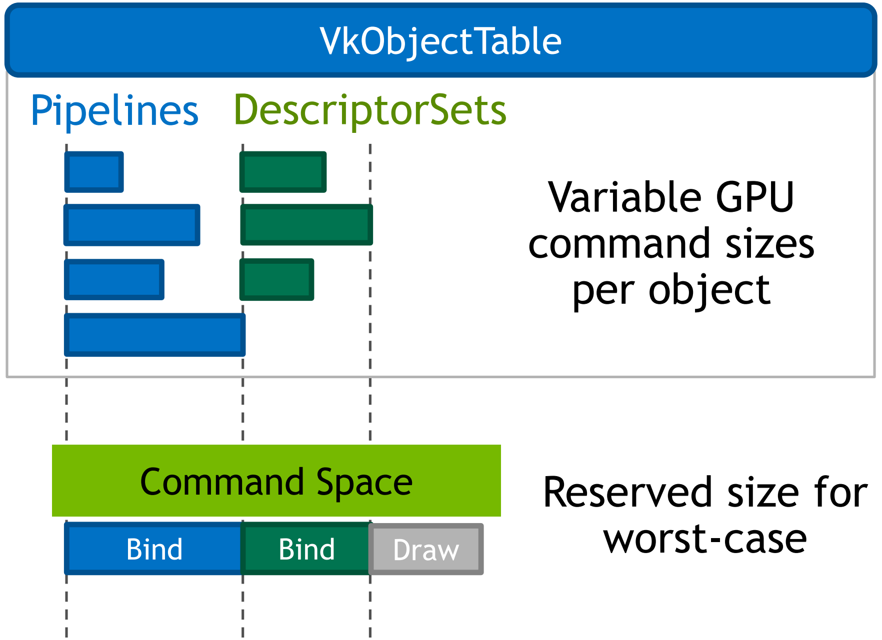
Compute shader stitches the command buffer from data stored in the VkObjectTable

Implements redundant state filter within local workgroup

Reserved command buffer space has to be allocated for worst-case scenario

NVIDIA IMPLEMENTATION

Previous 200.000 drawcall example reserved ~35 and generated ~15 megs



Global memory used internally to stitch command buffer

```
struct ObjectTable {
  uint                          pipelinesCount;
  uint                          descriptorsetsCount;
  uint                          vertexbuffersCount;
  uint                          indexbuffersCount;
  uint                          pushconstantCount;
  uint                          pipelinesetsCount;

  ResourcePipeline*             pipelines;
  ResourceDescriptorSet*        descriptorsets;
  ResourceVertexBuffer*         vertexbuffers;
  ResourceIndexBuffer*          indexbuffers;
  ResourcePushConstant*         pushconstants;
  ResourcePipelineSet*          pipelinesets;

  uint*                         rawPipelines;
  uint*                         rawDescriptorsets;
  uint*                         rawVertexbuffers;
  uint*                         rawIndexbuffers;
  uint*                         rawPushconstants;
  uint*                         rawPipelinesets;

  uvec2*                        pipelinediffs;
  uint*                         rawPipelinediffs;
};
```

```
struct GeneratingTask {
  uint        maxSequences;
  uvec4       sequenceRawSizes;
  uint*       outputBuffer;
  uint*       inputBuffers[MAX_INPUTS];
  ...
};

layout(std140,binding=0) uniform tableUbo {
  ObjectTable table;
};

layout(std140,binding=1) uniform taskUbo {
  GeneratingTask task;
};
```

CONCLUSION

GPU-generating will get slower with divergent resource usage

Still important to group by state, helps both CPU and GPU

CPU-generating is asynchronous to device, may not add to frame-time

GPU-generating is on device, best used to save work, not to offload work

CROSS API INTEROP

CROSS API INTEROP

Generic framework lead by Khronos

Share device memory & synchronization primitives across APIs and processes

Created in context of Vulkan, but not exclusive to it

Vulkan, OpenGL, DirectX (11,12), others may follow

EXTERNAL MEMORY

VK_KHX_external_memory (& friends)

New extensions to share memory objects across APIs

VkMemoryAllocateInfo was extended

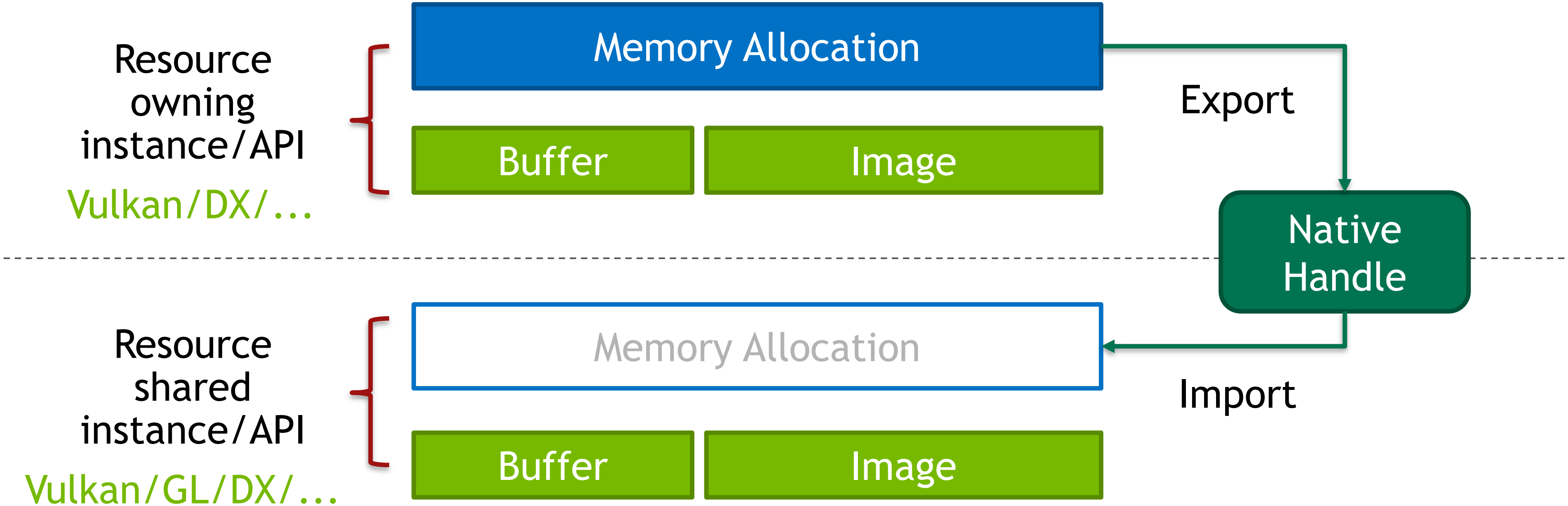
VkImportMemory*Platform*HandleInfoKHX to reference memory owned by other instances of the same device

VkExportMemory*Platform*HandleInfoKHX to make memory accessible to other instances

VkGetMemory*Platform*KHX to query platform handle

EXTERNAL MEMORY

VK_KHX_external_memory (& friends)



Memory offsets for resources are provided by original instance

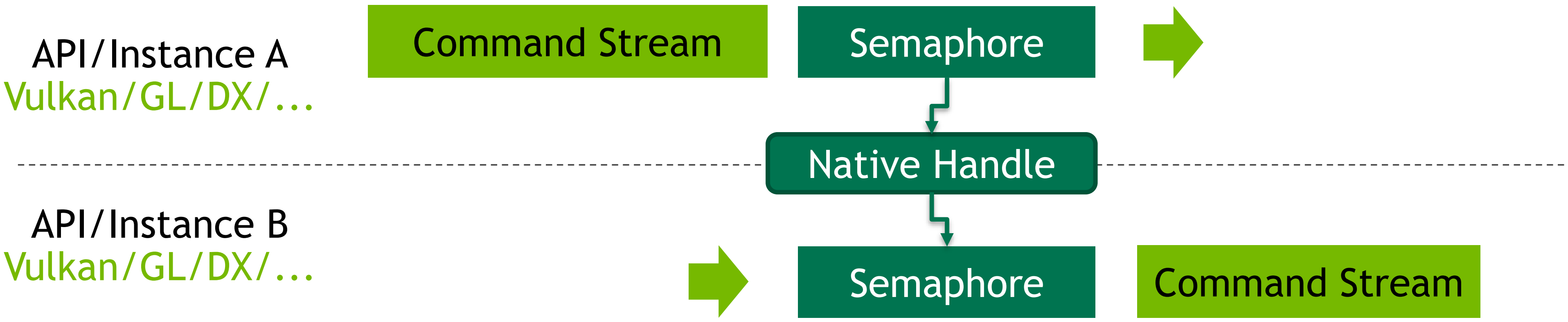
EXTERNAL SYNCHRONIZATION

VK_KHX_external_semaphore (& friends)

Same principle as with memory

Allows sharing device synchronization primitives

Control command flow and dependencies on the same device



CROSS API INTEROP

May allow adding Vulkan (or other APIs) to host applications not designed for it

OpenGL extension to import Vulkan memory is in progress (but not to export from it)

Synchronization across (or within) APIs should not be very frequent (Frankenstein API usage)

VULKAN VR

NVIDIA VRWORKS

Comprehensive SDK for VR Developers

GRAPHICS	HEADSET	TOUCH & PHYSICS	AUDIO
LENS MATCHED SHADING	CONTEXT PRIORITY	PHYSX	VRWORKS AUDIO
SINGLE PASS STEREO	DIRECT MODE	**PROFESSIONAL**	**VIDEO**
MULTIRES SHADING	FRONT BUFFER RENDERING	WARP & BLEND	VRWORKS 360 VIDEO
VR SLI		SYNCHRONIZATION	GPUDIRECT FOR VIDEO
		GPU AFFINITY	

NVIDIA VRWORKS

Comprehensive SDK for VR Developers

GRAPHICS

- **LENS MATCHED** SHADING
- **SINGLE PASS** STEREO
- **MULTI**RES SHADING
- **VR** SLI

HEADSET

- **CONTEXT** PRIORITY
- **DIRECT** MODE
- **FRONT BUFFER** RENDERING

TOUCH & PHYSICS

- **PHYSX**

PROFESSIONAL

- **WARP &** BLEND
- **SYNCHRONIZATION**
- **GPU** AFFINITY

AUDIO

- **VRWORKS** AUDIO

VIDEO

- **VRWORKS** 360 VIDEO
- **GPUDIRECT** FOR VIDEO

Vulkan

GRAPHICS PIPELINE

VR Workloads

1080

1920

124M Pix/s
N vertices
60 Hz

1680

1512 1512

457M Pix/s
2N vertices
90 Hz

Preprocessing

Geometric Pipeline

Rasterization Fragment Shader

Postprocessing

3x

~3.6x

NVIDIA VRWORKS

Comprehensive SDK for VR Developers

GRAPHICS

LENS MATCHED SHADING

SINGLE PASS STEREO

MULTIRES SHADING

VR SLI

HEADSET

CONTEXT PRIORITY

DIRECT MODE

FRONT BUFFER RENDERING

TOUCH & PHYSICS

PHYSX

PROFESSIONAL

WARP & BLEND

SYNCHRONIZATION

GPU AFFINITY

AUDIO

VRWORKS AUDIO

VIDEO

VRWORKS 360 VIDEO

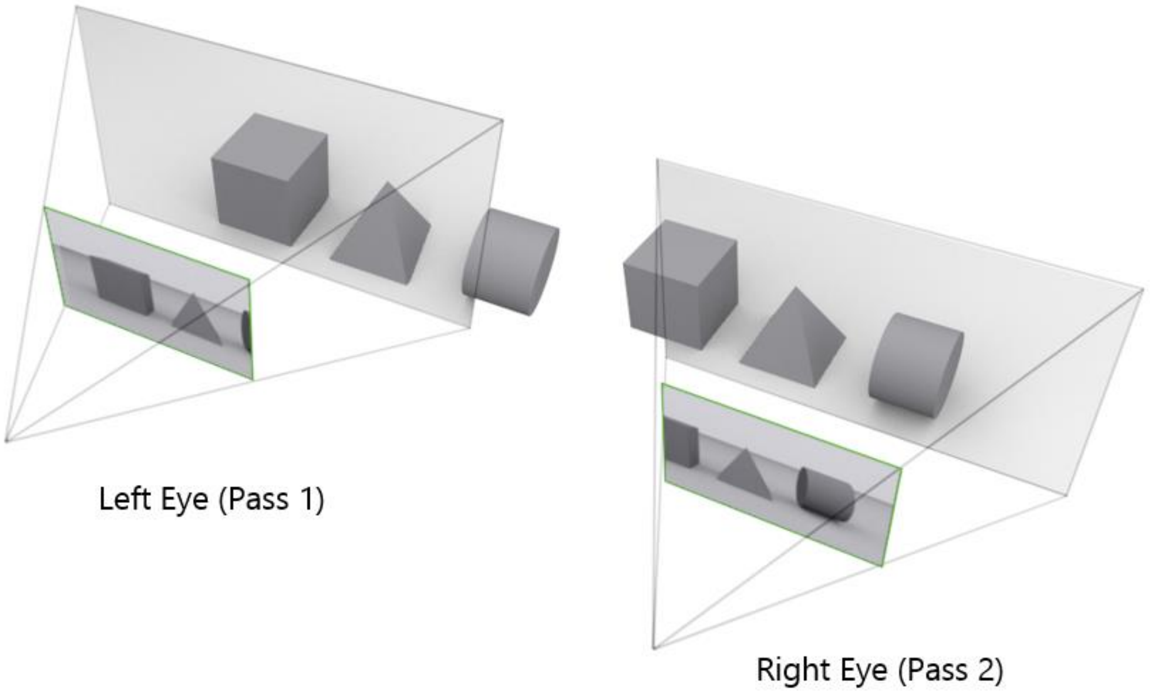
GPUDIRECT FOR VIDEO

SINGLE PASS STEREO

Traditional Rendering

Render eyes separately

Doubles CPU and GPU load



SINGLE PASS STEREO

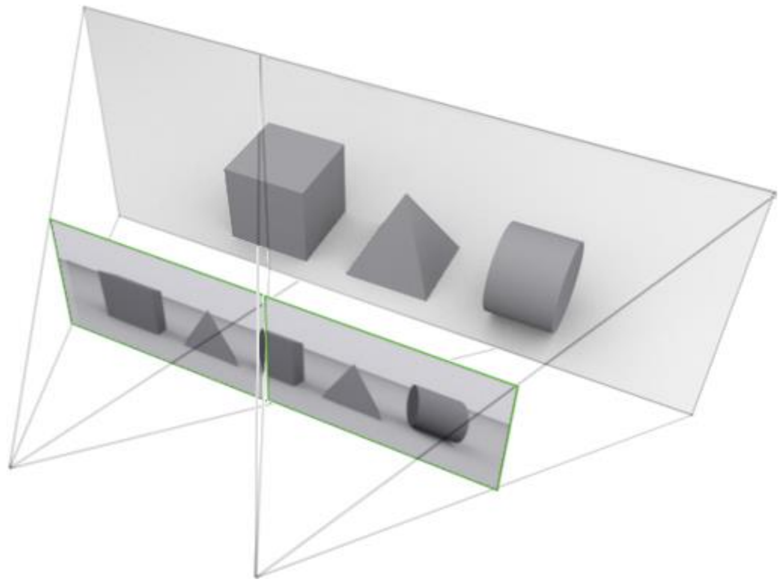
Using SPS to improve rendering performance

Single Pass Stereo uses Simultaneous Multi-Projection architecture

Draw geometry only once

Vertex/Geometry stage runs once
Outputs two positions for left/right

Only rasterization is performed per-view



More Detail:
GTC2017 - S7578 - ACCELERATING YOUR VR APPLICATIONS WITH VRWORKS

SINGLE PASS STEREO

Vulkan

In Vulkan via `VK_NVX_multiview_per_view_attributes`

Requires `VK_KHX_multiview` and `VK_NV_viewport_array2` extensions

Check support using `vkGetPhysicalDeviceFeatures2KHR` with a `VkPhysicalDeviceMultiviewPerViewAttributesPropertiesNVX` struct

Spec distinguishes between extension support in one or all components of position attribute

We only need support for the X component for VR

SINGLE PASS STEREO

Setup

Create layered texture image and view for rendering left and right simultaneously

Set up render pass with MultiView support

Broadcast rendering to both viewports

```
VkRenderPassMultiviewCreateInfoKHX::pViewMasks -> 0b0011
```

Hint to render both views concurrently, if possible

```
VkRenderPassMultiviewCreateInfoKHX::pCorrelationMasks -> 0b0011
```

Fill UBO with offsets for left and right eye

SINGLE PASS STEREO

Vertex Shader

Calculate projection space position

```
proj_pos = (proj * view * model * inPosition).xyz;
```

Standard MultiView – specify once, may execute shader twice

```
gl_Position = proj_pos + UBO.offsets[gl_ViewIndex];
```

With per-view attributes - also specify positions explicitly, execute shader only once

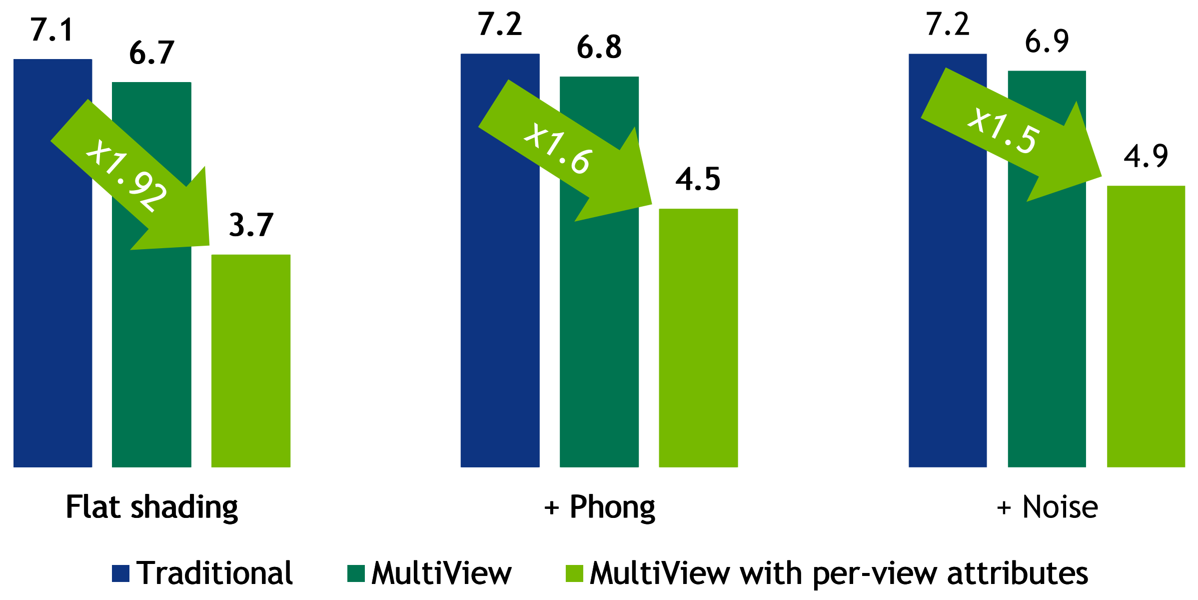
```
gl_PositionPerViewNV[0] = proj_pos + UBO.offsets[0];
gl_PositionPerViewNV[1] = proj_pos + UBO.offsets[1];
```

GRAPHICS PIPELINE

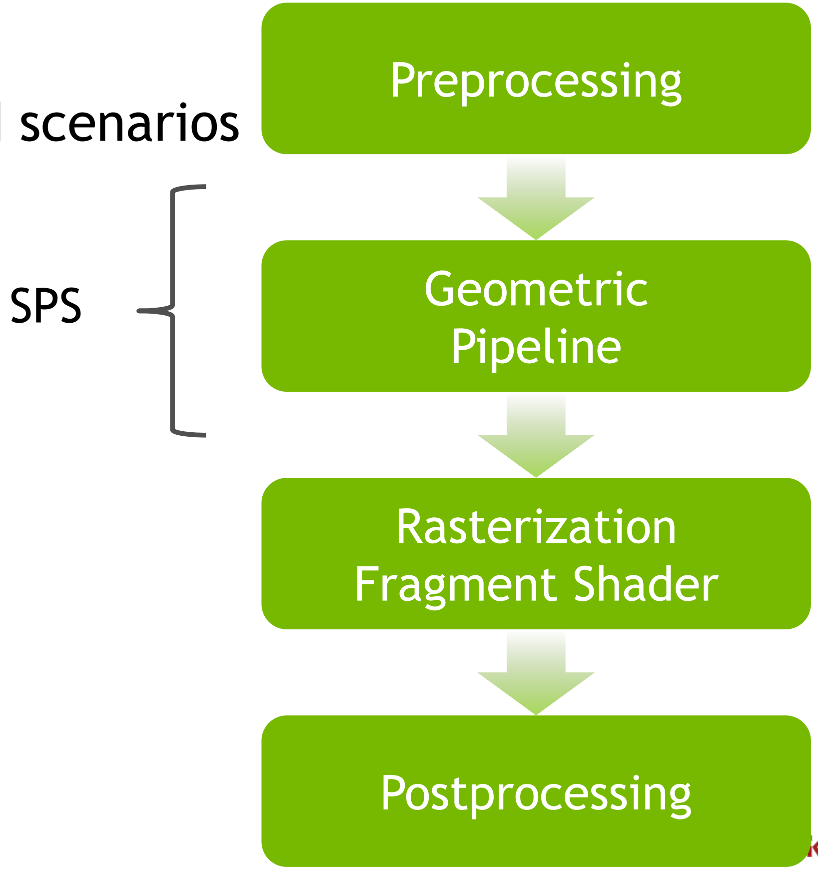
Single Pass Stereo Performance Results

Single Pass Stereo brings benefits in geometry bound scenarios

Heavy fragment shaders will reduce scaling



NVIDIA Quadro P6000, Scene with 17.6M faces, frame times in ms



NVIDIA VRWORKS

Comprehensive SDK for VR Developers

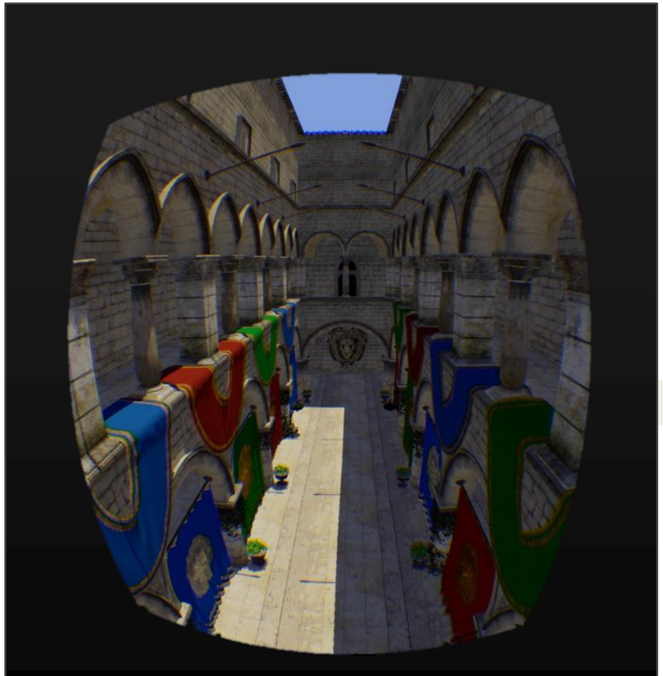
GRAPHICS	HEADSET	TOUCH & PHYSICS	AUDIO
LENS MATCHED SHADING	CONTEXT PRIORITY	PHYSX	VRWORKS AUDIO
SINGLE PASS STEREO	DIRECT MODE	**PROFESSIONAL**	**VIDEO**
MULTIRES SHADING	FRONT BUFFER RENDERING	WARP & BLEND	VRWORKS 360 VIDEO
VR SLI		SYNCHRONIZATION	GPUDIRECT FOR VIDEO
		GPU AFFINITY	

Vulkan

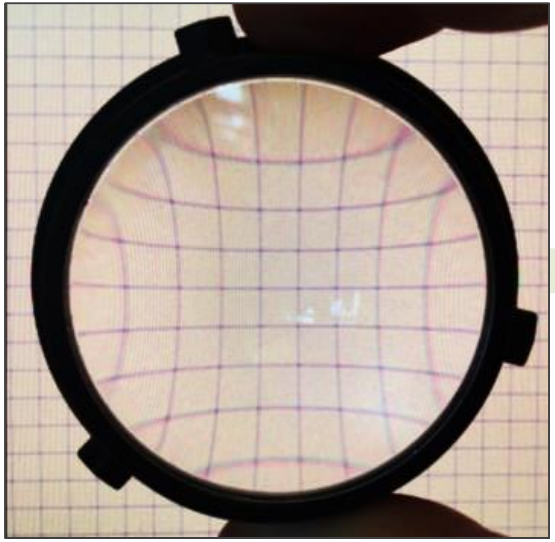
NVIDIA

LENS MATCHED SHADING

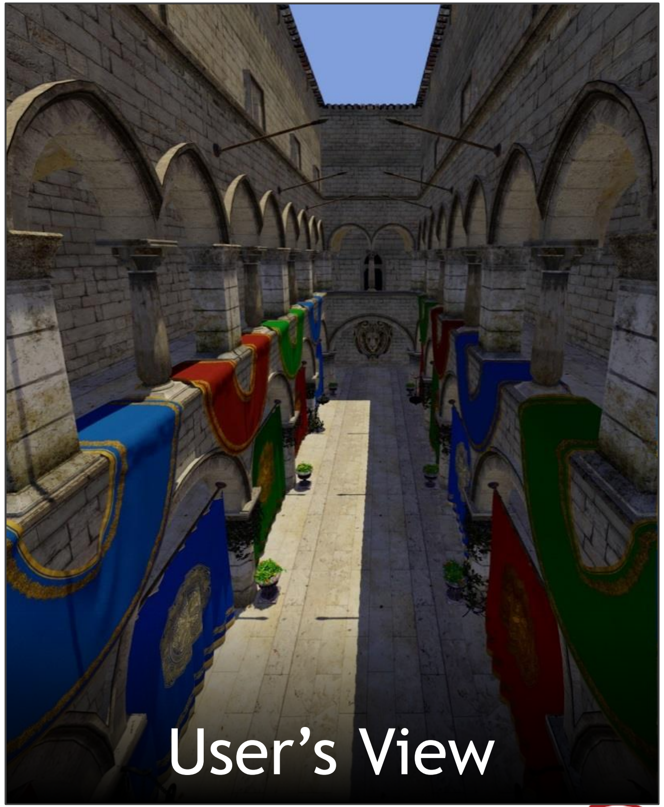
Countering Lens Distortion



Displayed Image



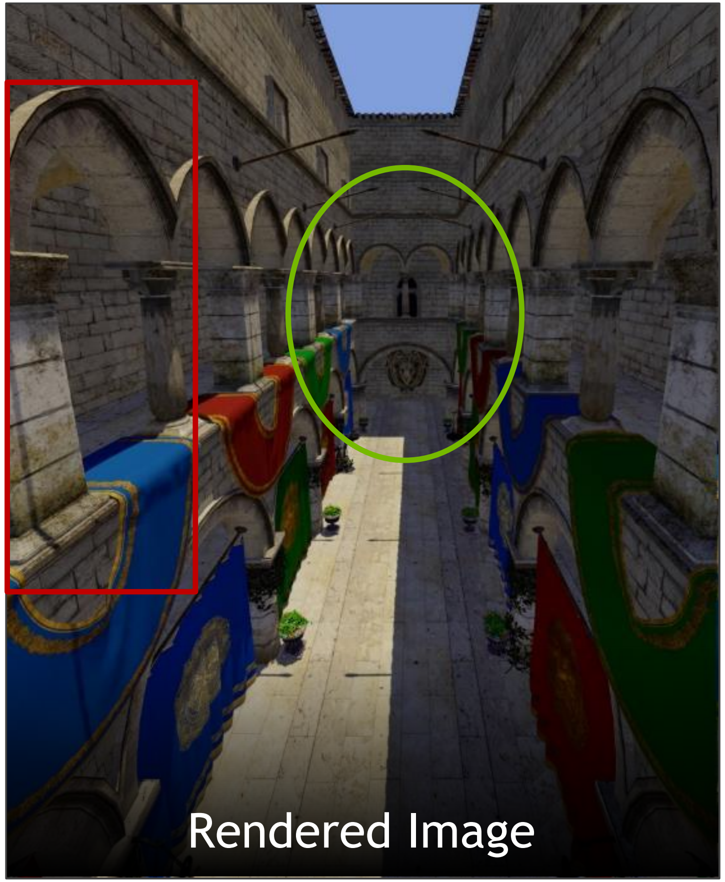
Optics

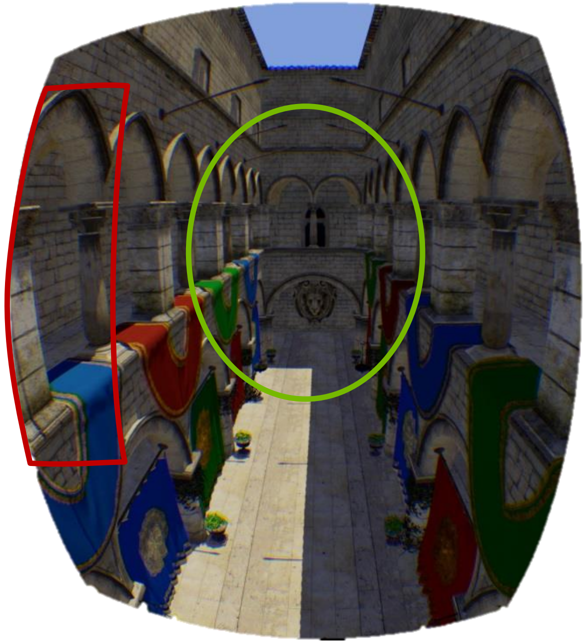


User’s View

LENS MATCHED SHADING

Oversampling near the borders

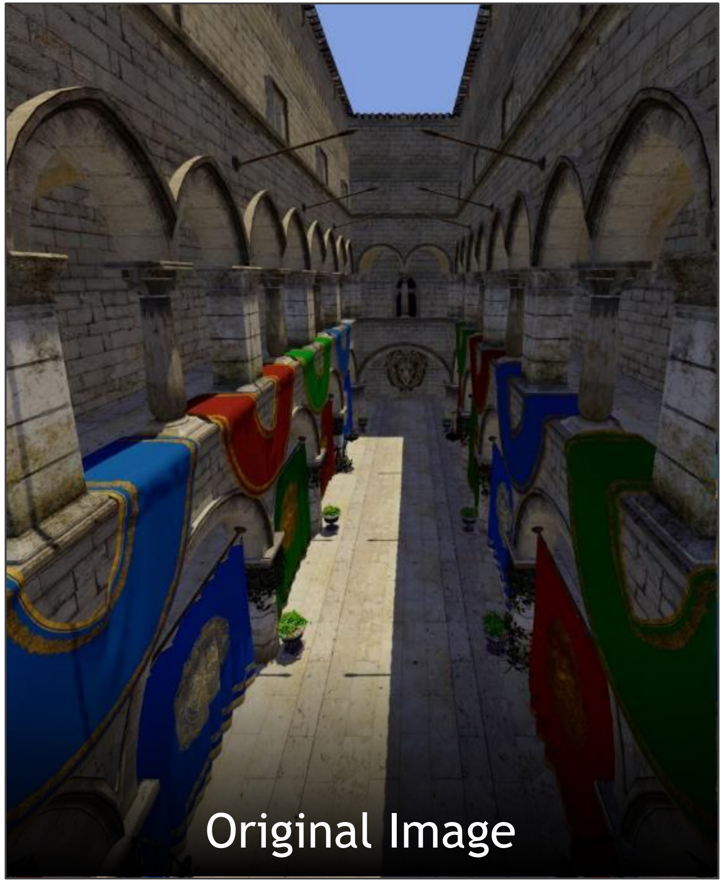


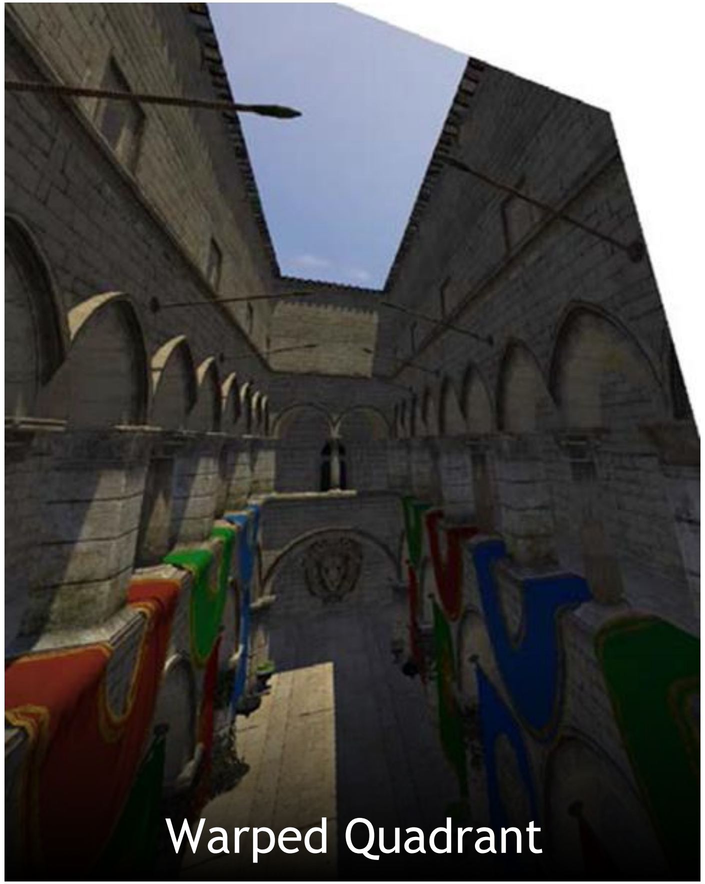


Displayed Image

LENS MATCHED SHADING

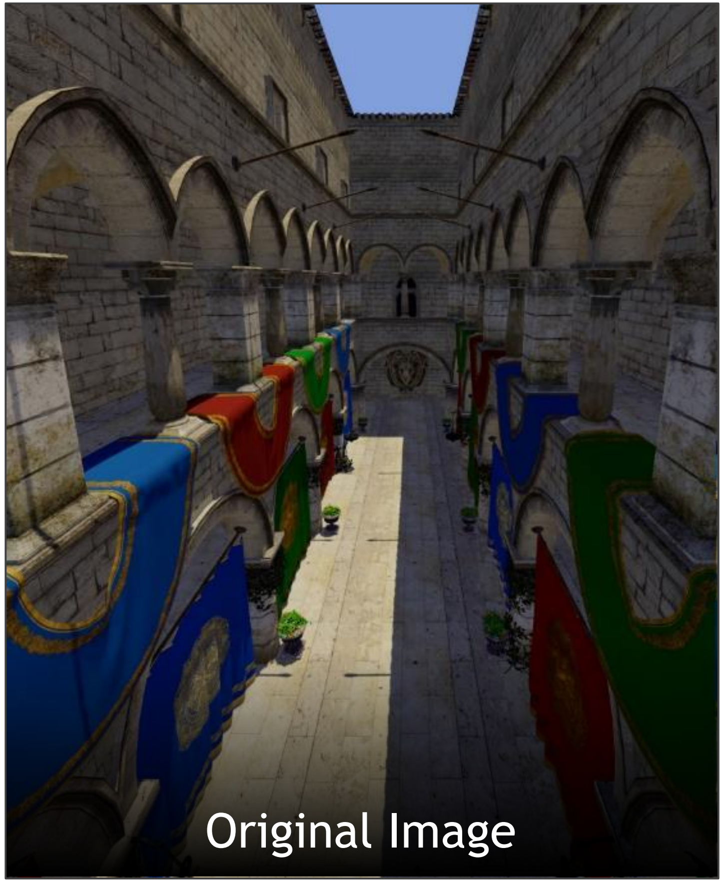
w’ = w + Ax + By

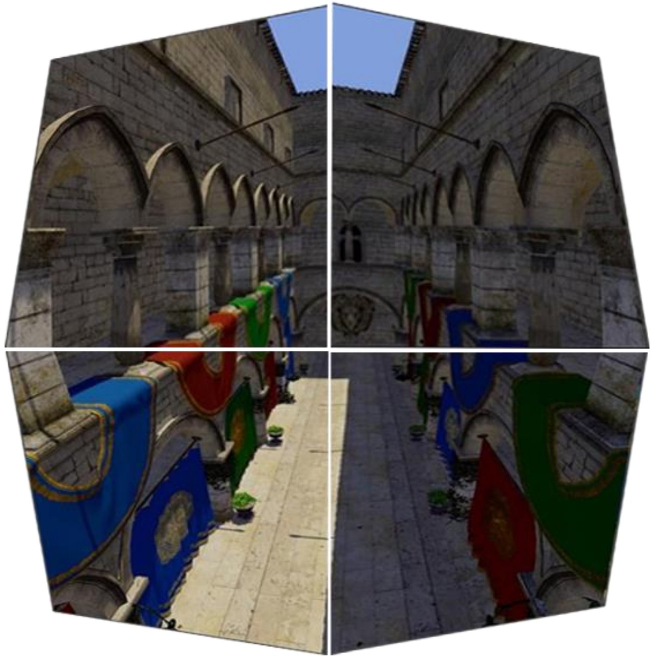




LENS MATCHED SHADING

Four Viewports





LMS Image

LENS MATCHED SHADING

Vulkan

In Vulkan via `VK_NV_clip_space_w_scaling` extension

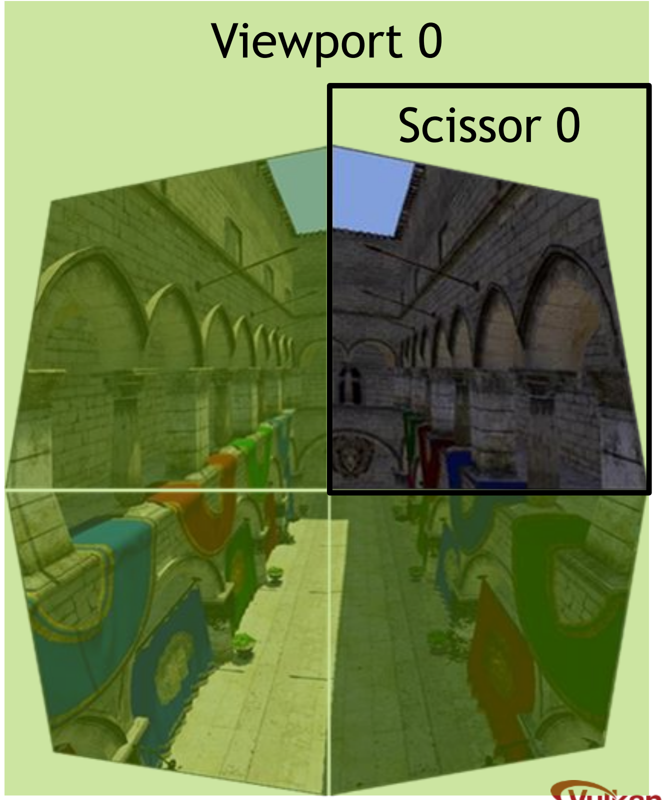
Set up four viewports, rendering full resolution

Set scissors to each quadrant

`VkPipelineViewportWScalingStateCreateInfoNV`

W scaling parameters:

- Use the viewport struct / set on creation
- Dynamic state & `vkCmdSetViewportWScalingNV`



LENS MATCHED SHADING

Shaders

`gl_ViewportMask[0]` controls broadcasting of vertices and primitives

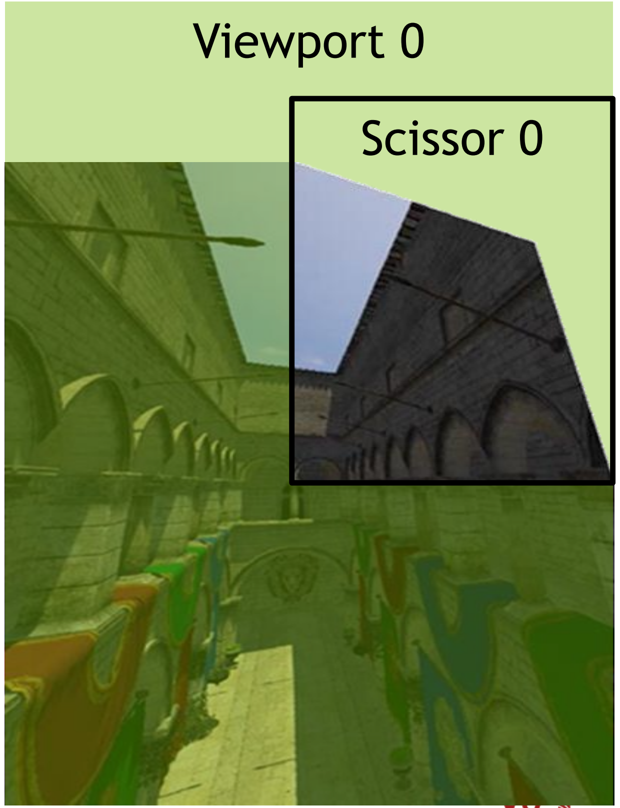
Inefficient - set mask in vertex shader

```
gl_ViewportMask[0] = 15;
```

More efficient - filter in pass through geometry shader

Determine quadrant(s) for each primitive

Set bit(s) in `gl_ViewportMask[0]`



Vulkan.

NVIDIA.

LENS MATCHED SHADING

Scaling and Unscaling

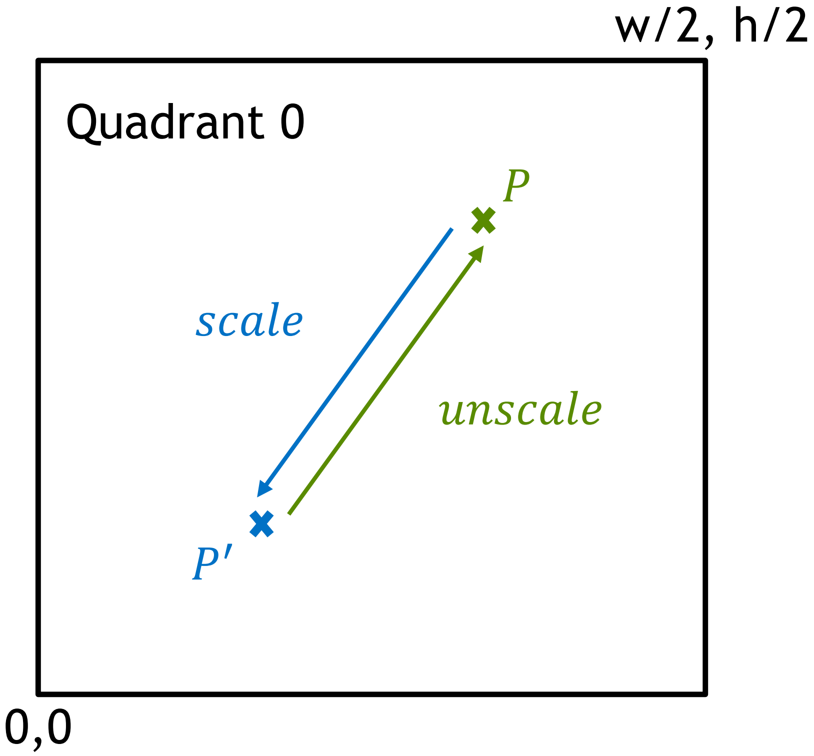
HMD runtime can‘t consume w warped images yet, need to unscale before submit

$$scale = \frac{1}{1 - w_x * P'_x - w_y * P'_y}$$

$$P' = scale \; * P$$

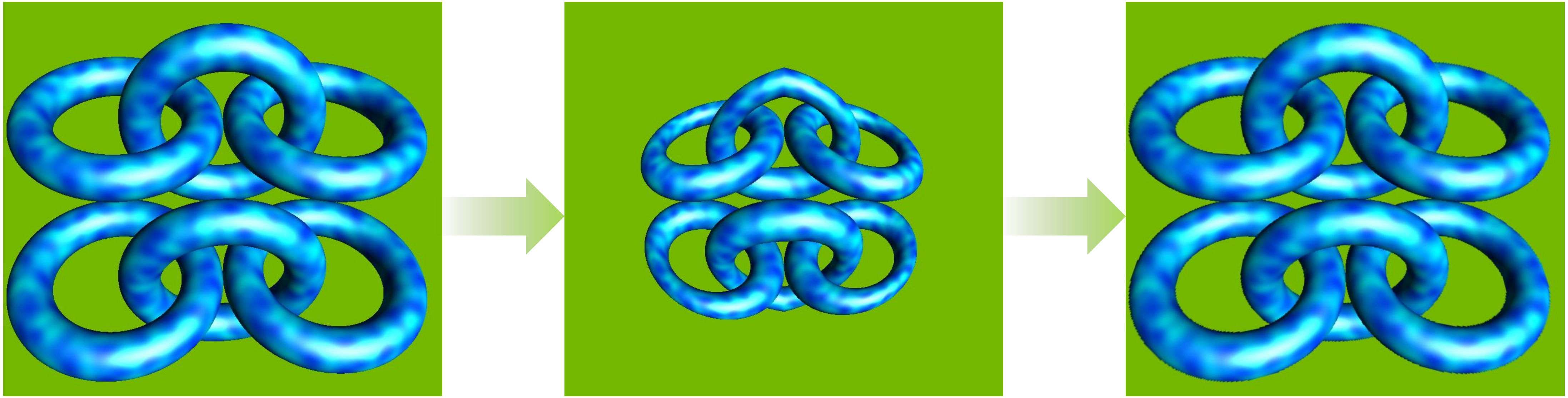
$$unscale = \frac{1}{1 + w_x * P_x + w_y * P_y}$$

$$P = unscale \; * P'$$



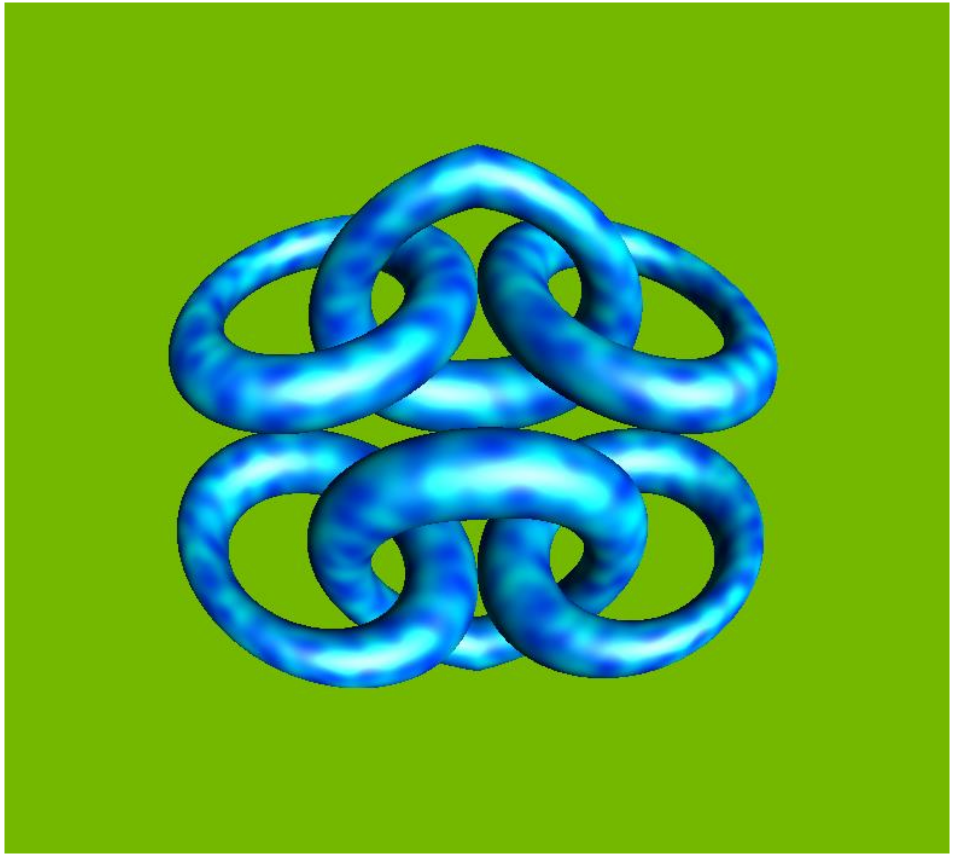
LENS MATCHED SHADING

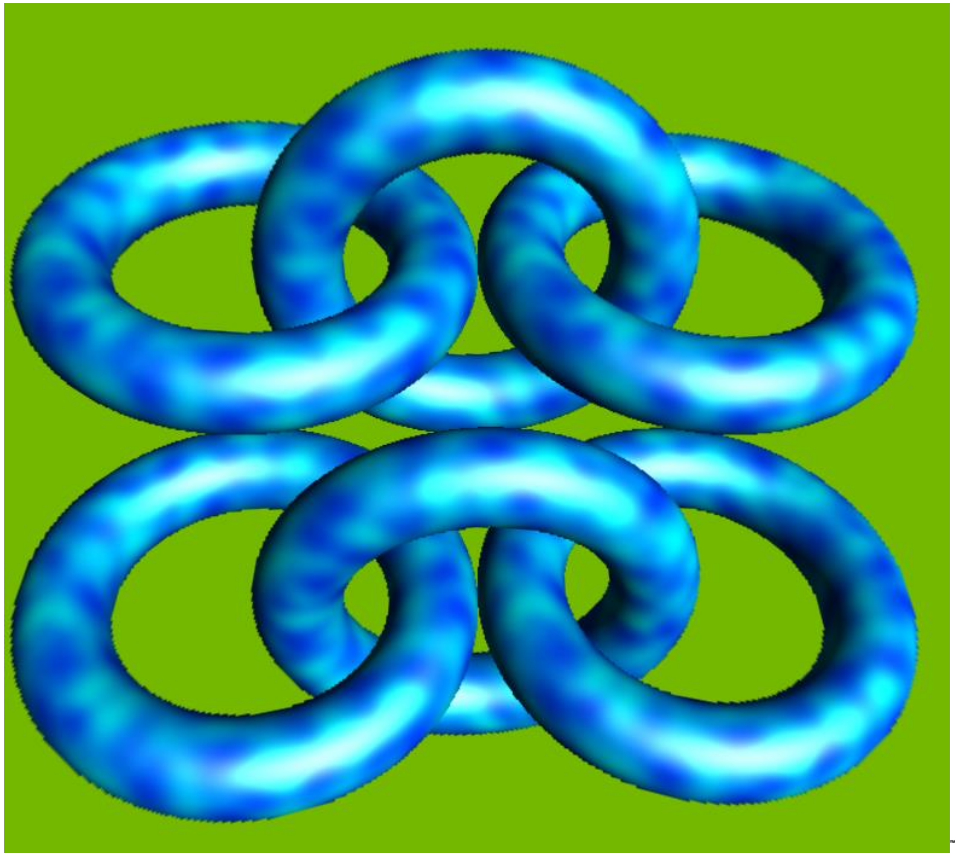
Scaling and Unscaling



LENS MATCHED SHADING

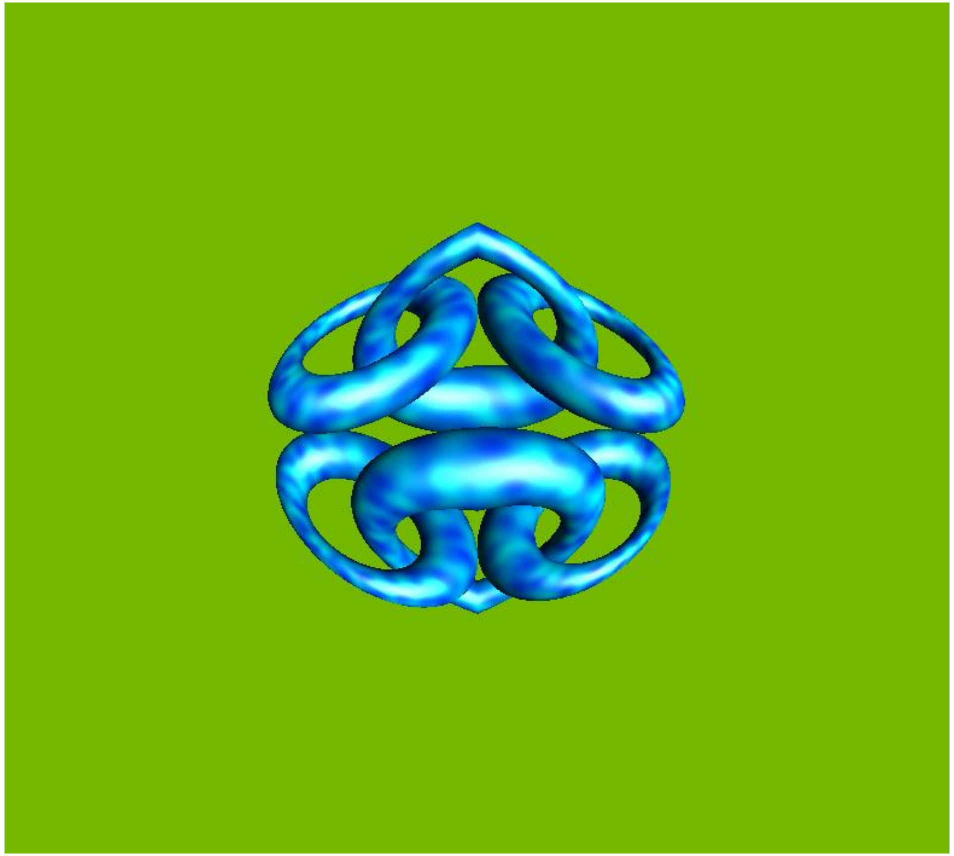
Wx = 0.4 Wy = 0.4 24.2ms -> 11.3ms

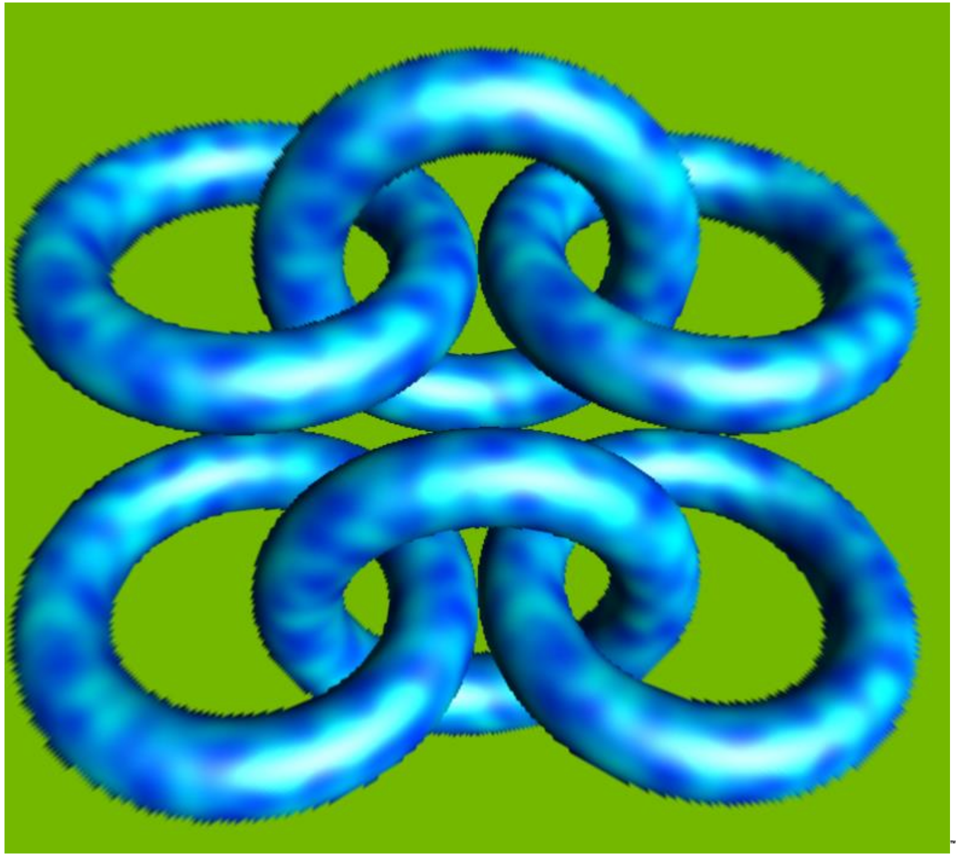




LENS MATCHED SHADING

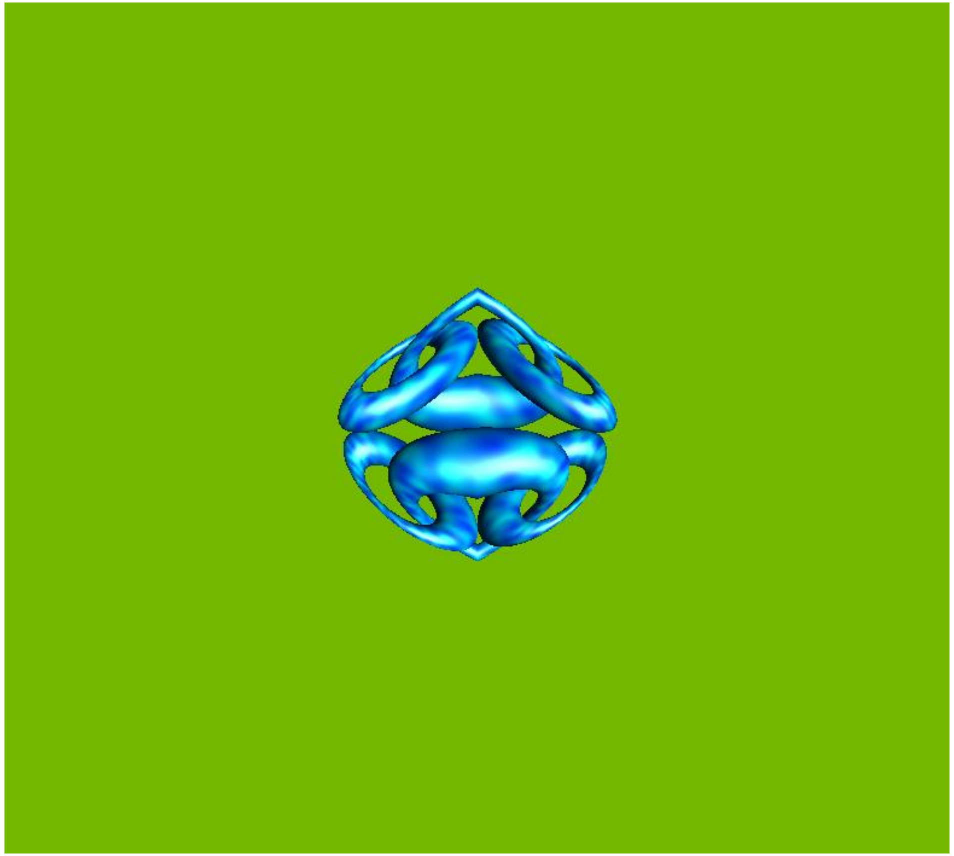
Wx = 1.0 Wy = 1.0 24.2ms -> 5.9ms

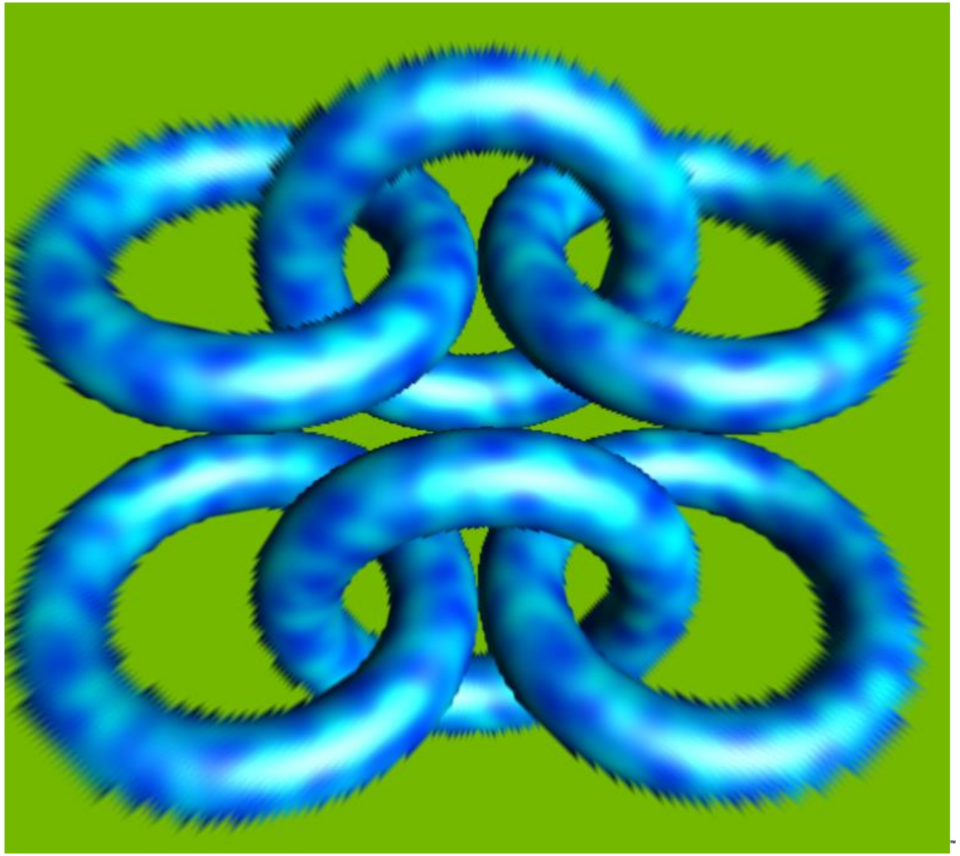




LENS MATCHED SHADING

Wx = 2.0 Wy = 2.0 24.2ms -> 3.3ms



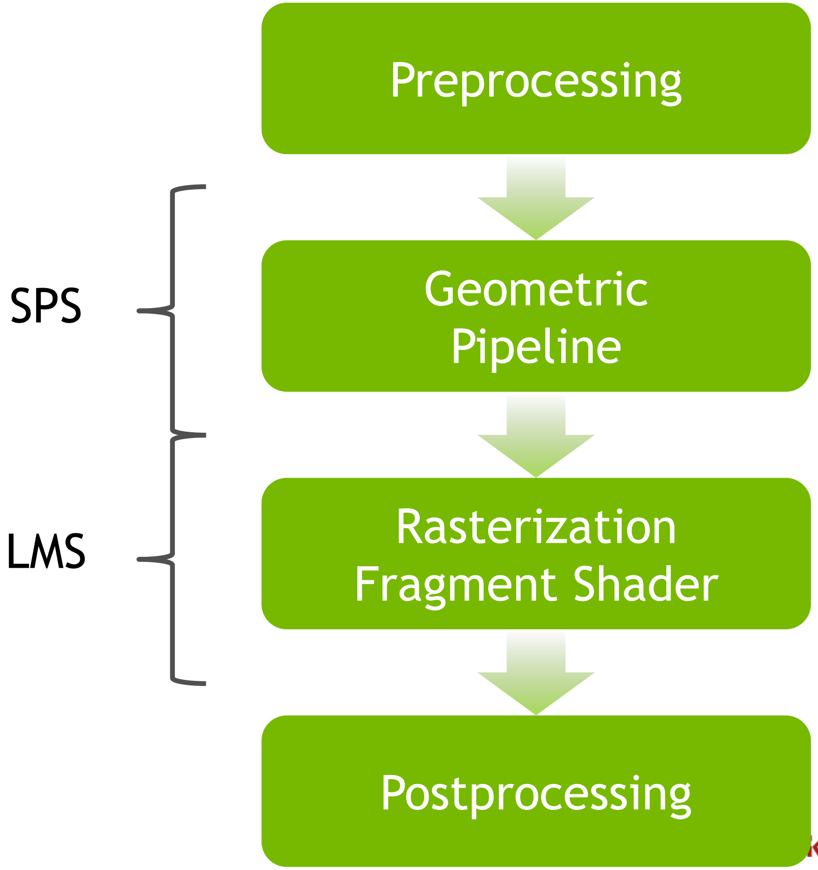


GRAPHICS PIPELINE

Lens Matched Shading Results

LMS can improve performance of
Raster / Fragment stage

Trade-off between quality and performance



NVIDIA VRWORKS

Comprehensive SDK for VR Developers

GRAPHICS

- **LENS MATCHED** SHADING
- **SINGLE PASS** STEREO
- **MULTI**RES SHADING
- **VR** SLI

HEADSET

- **CONTEXT** PRIORITY
- **DIRECT** MODE
- **FRONT BUFFER** RENDERING

TOUCH & PHYSICS

- **PHYSX**

PROFESSIONAL

- **WARP &** BLEND
- **SYNCHRONIZATION**
- **GPU** AFFINITY

AUDIO

- **VRWORKS** AUDIO

VIDEO

- **VRWORKS** 360 VIDEO
- **GPUDIRECT** FOR VIDEO

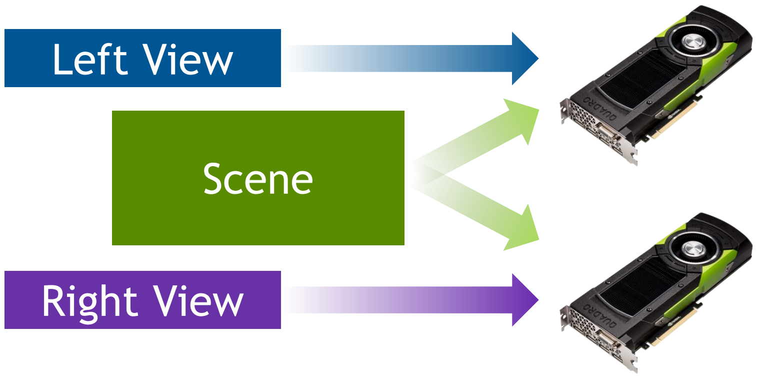
Vulkan

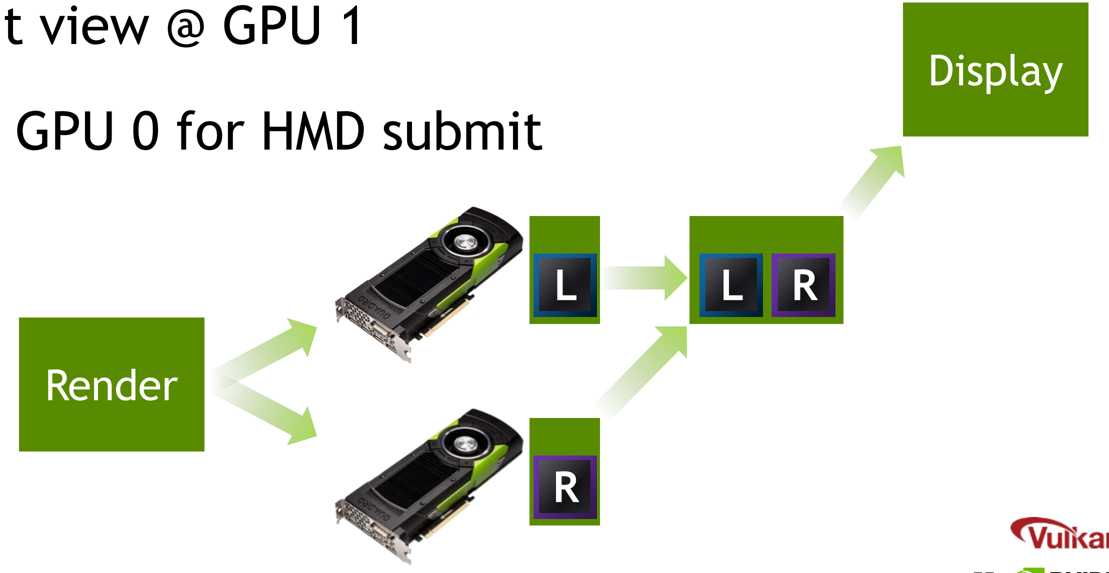
VR SLI

Overview

Common HMD VR use case, realized through `VK_KHX_device_group` extension

1. Broadcast scene data, upload separate view data
2. Render left view @ GPU 0, right view @ GPU 1
3. Transfer right view @ GPU 1 to GPU 0 for HMD submit





VR SLI

Enumerate devices, create device group

Create VkInstance using `VK_KHX_device_group_creation`

Use `vkEnumeratePhysicalDeviceGroupsKHX`
to enumerate device groups

Check that devices in a candidate group
support `VK_KHX_device_group`

Make sure the device group supports peer access via
`vkGetDeviceGroupPeerMemoryFeaturesKHX`

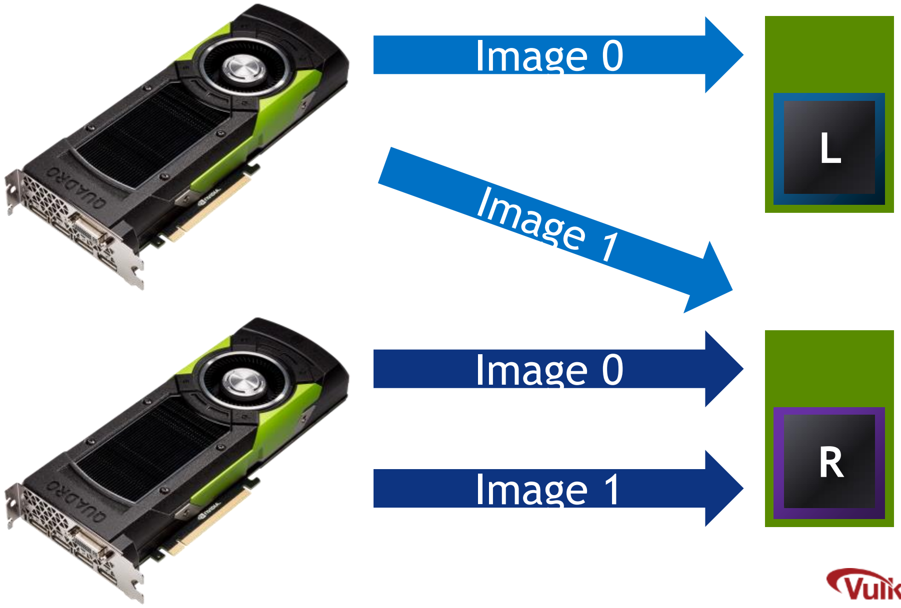
Create logical VkDevice using
`VkDeviceGroupDeviceCreateInfoKHX` struct

VR SLI

Prepare multi-GPU textures

Use `vkBindImageMemory2KHX` to bind memory to images across GPU boundaries

No direct texture copies in VK,
Use bindings to access memory




```
deviceIndices0[] = { 0, 1 };

deviceIndices1[] = { 1, 1 };
```

Make sure the formats match!

VR SLI

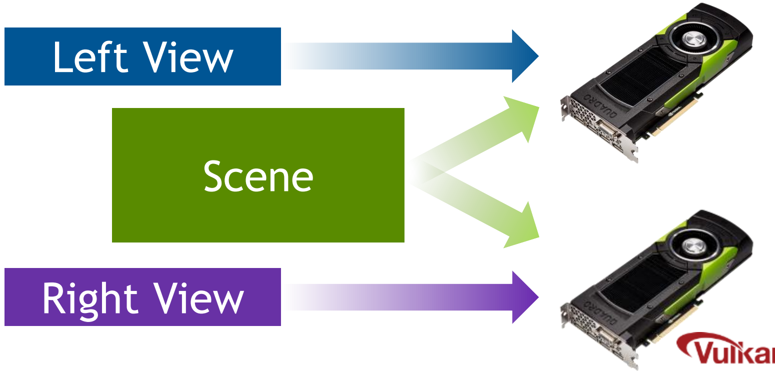
Data Upload

Upload data e.g. using `vkCmdUpdateBuffer` recorded in command buffer

Submit with a `VkDeviceGroupSubmitInfoKHX` struct, allowing device masks

Scene and other view independent data can be broadcast

View matrix and other view dependent uploads are limited to one GPU



VR SLI

Rendering

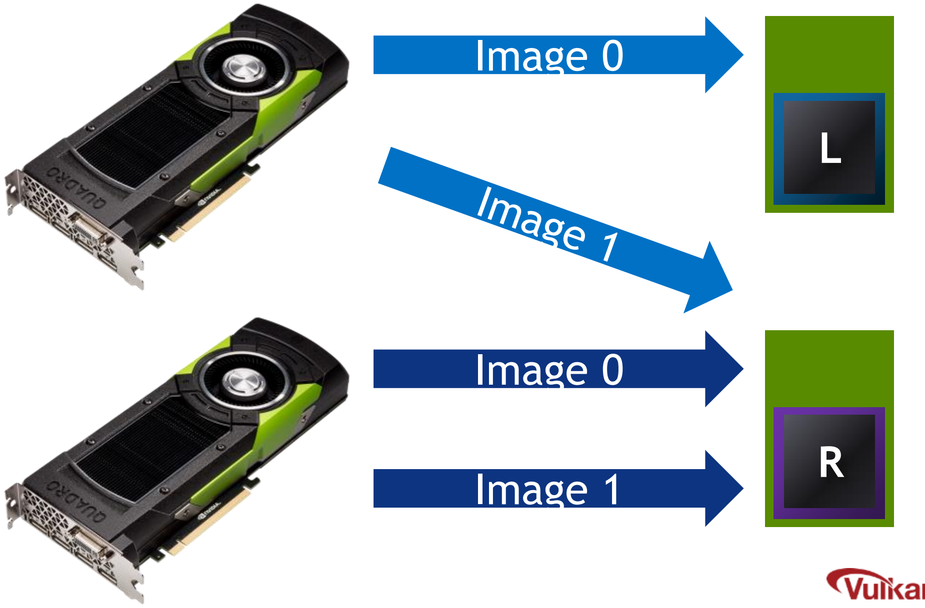
Submit one command buffer for rendering on both GPUs

Use Image 0 as render target

Broadcasting is the default

Restrict rendering using

- Command Buffer Info
- Render Pass Info
- `vkCmdSetDeviceMaskKHX`
- Submit Infos



VR SLI

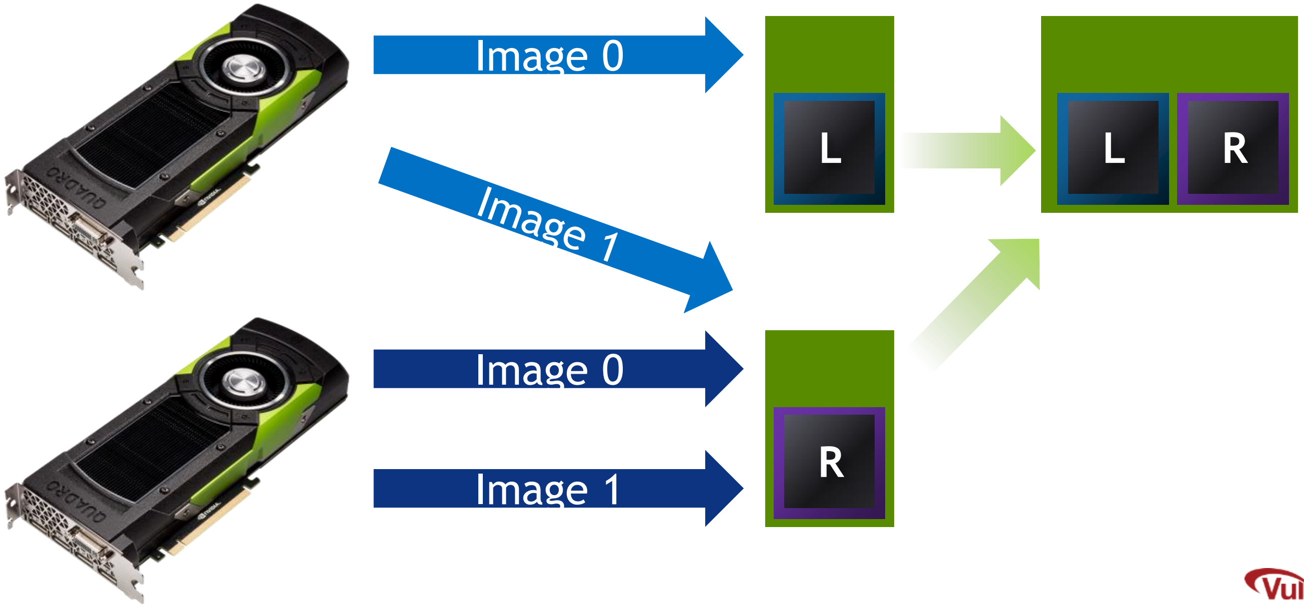
Texture Transfer

Texture transfer via `vkCmdCopyImage` or `vkCmdBlitImage` restricted to GPU 0

Transfer Image 0 and Image 1

Targets

- Swap Chain Image
- HMD textures
- Post-Process texture



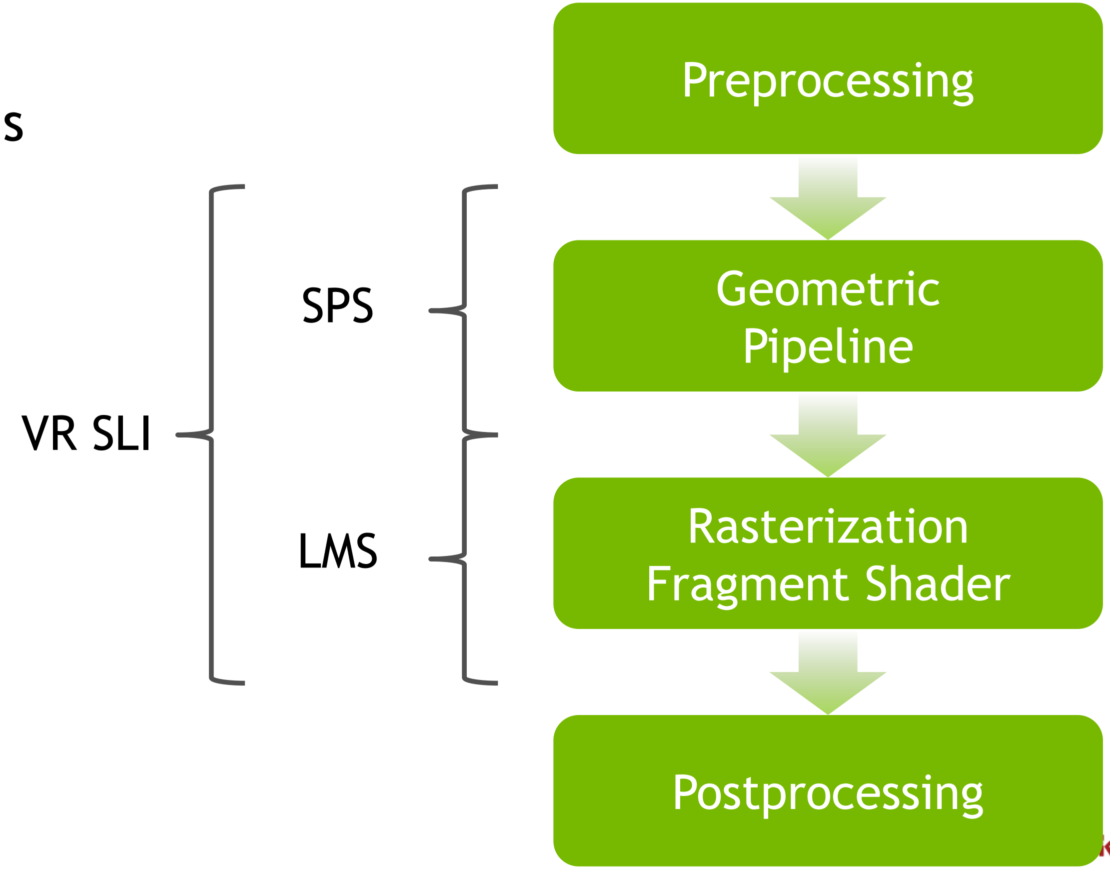
GRAPHICS PIPELINE

VR SLI impact

VR SLI covers a wide variety of workloads

Perfect load balancing between left/right eye and two GPUs

Copy overhead and view independent workloads limit scaling



TRY IT OUT!

VRWorks SDK: https://developer.nvidia.com/vrworks

SPS: vk_stereo_view_rendering

LMS: vk_clip_space_w_scaling

VR SLI: vk_device_group

Extensions

www.khronos.org/registry/vulkan/specs/1.0-extensions/html/vkspec.html

KHX and NVX are experimental, feedback welcome!

VULKAN NSIGHT SUPPORT

NSIGHT + VULKAN

What is Nsight Visual Studio Edition

Understand CPU/GPU interaction

Explore and debug your frame as it is rendered

Profile your frame to understand hotspots and bottlenecks

Save your frame for targeted analysis and experimentation

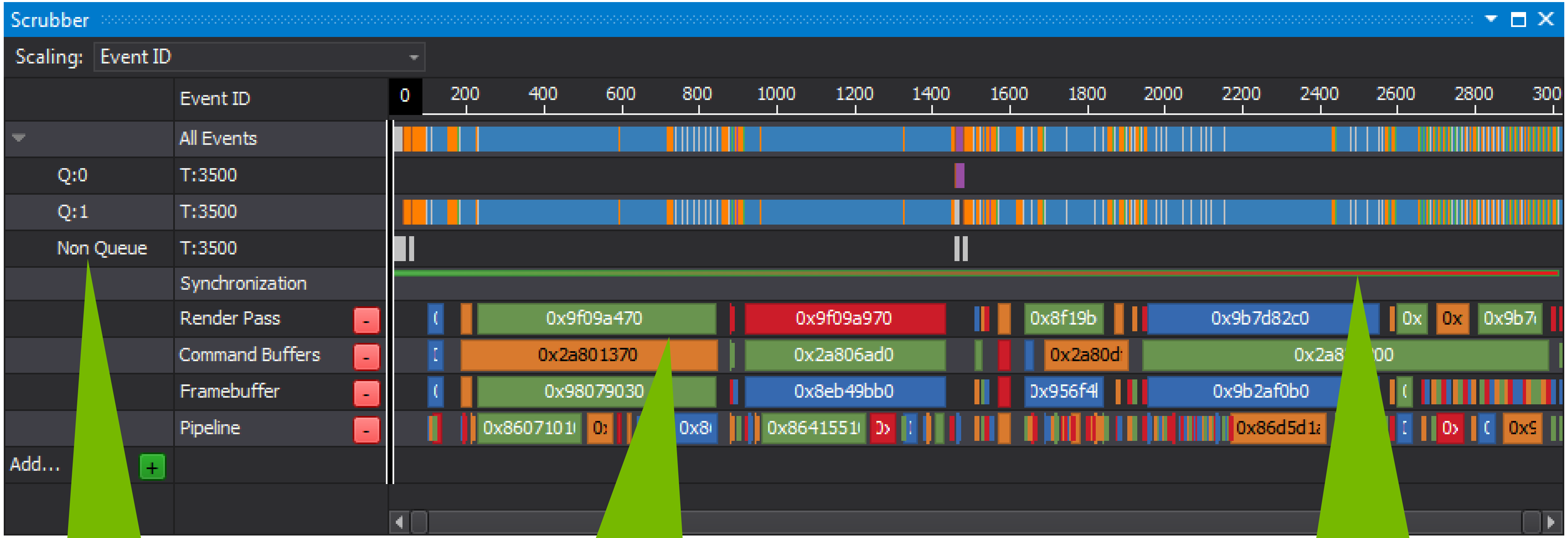
Debug & profile VR applications

Leverage the Microsoft Visual Studio platform

New in 5.3: Vulkan 1.0.42 support, extensions, serialization, shader reflection, and descriptor view

NSIGHT & VULKAN

Scrubber



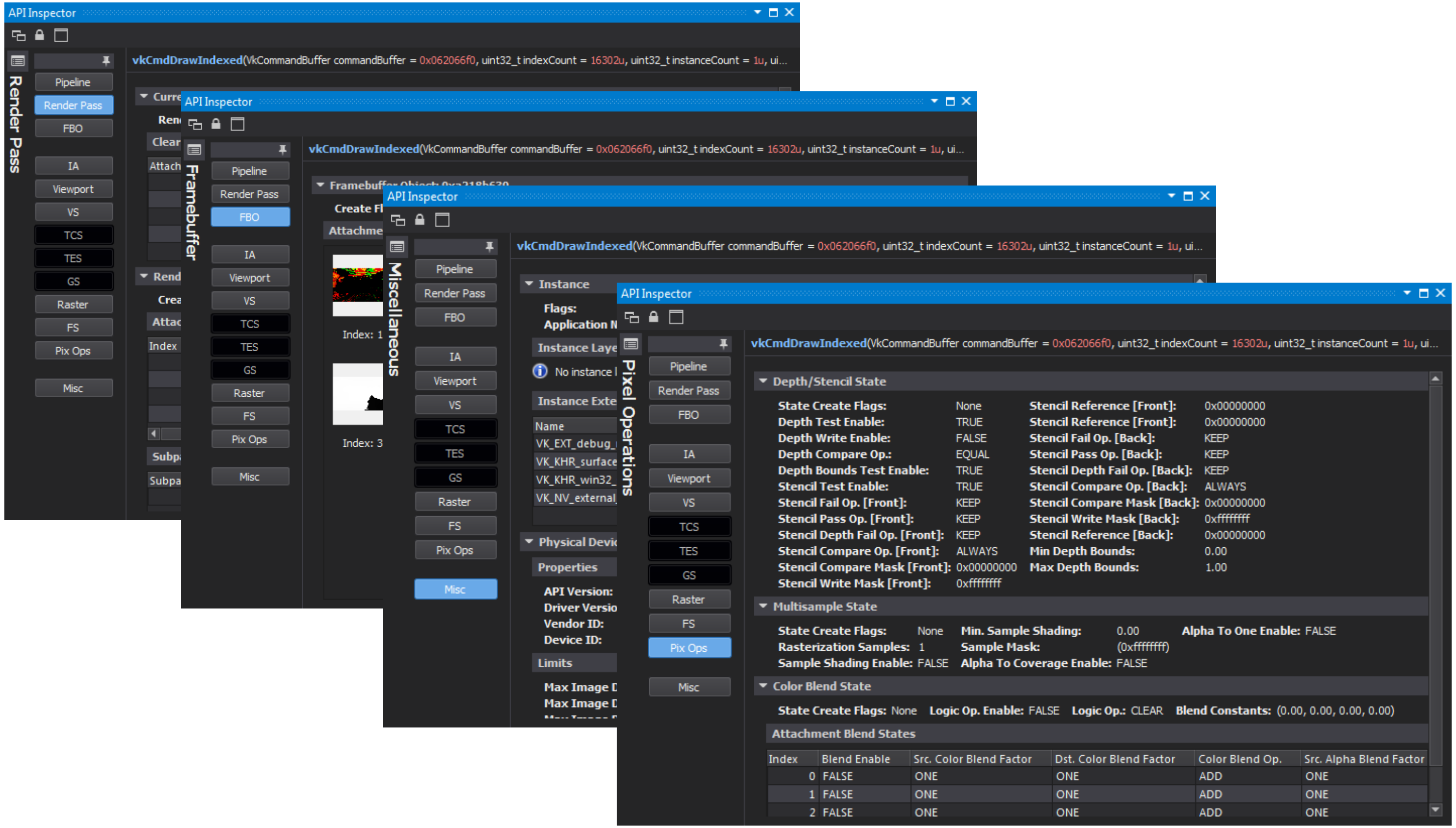
Multi-queue / multi-thread

State buckets & VK_EXT_debug_markers

Synchronization

NSIGHT + VULKAN

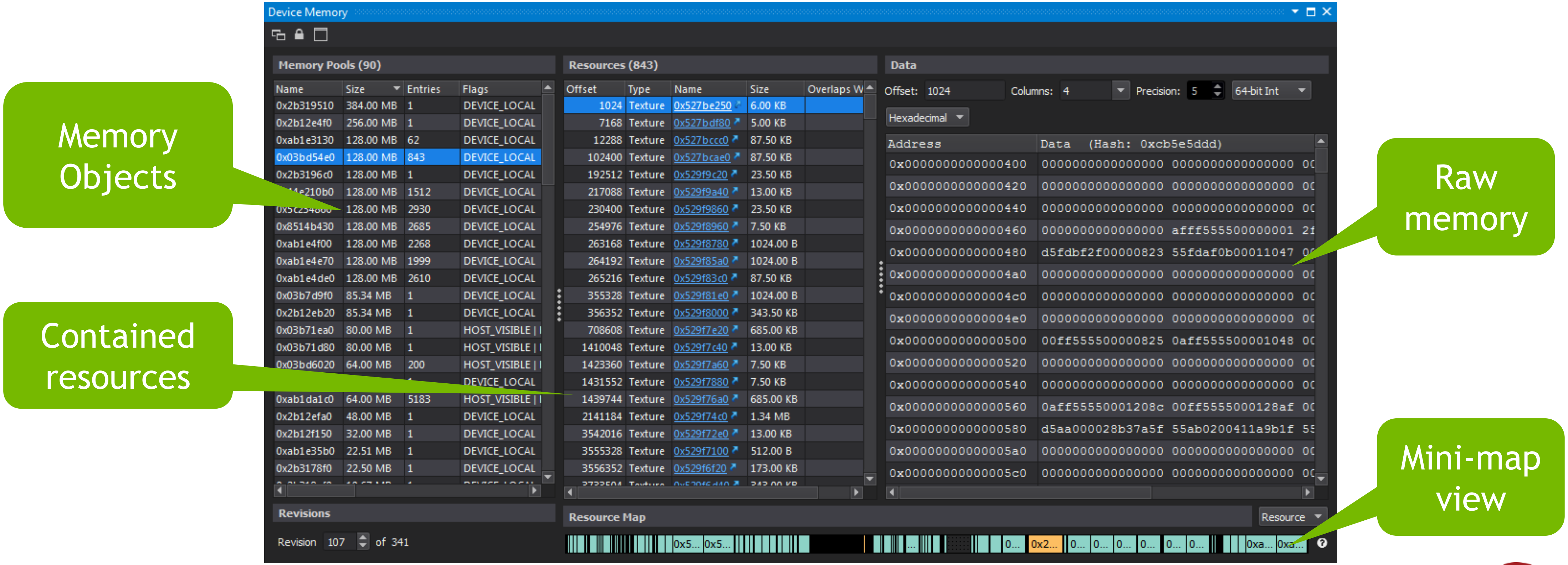
API Inspector – All of the render state



- Pipeline
- Render Pass
- Framebuffer
- Input Assembly
- Shaders
  - SPIRV Decorations
  - Uniform Values
- Viewport
- Raster
- Pixel Ops.
- Misc.

NSIGHT + VULKAN

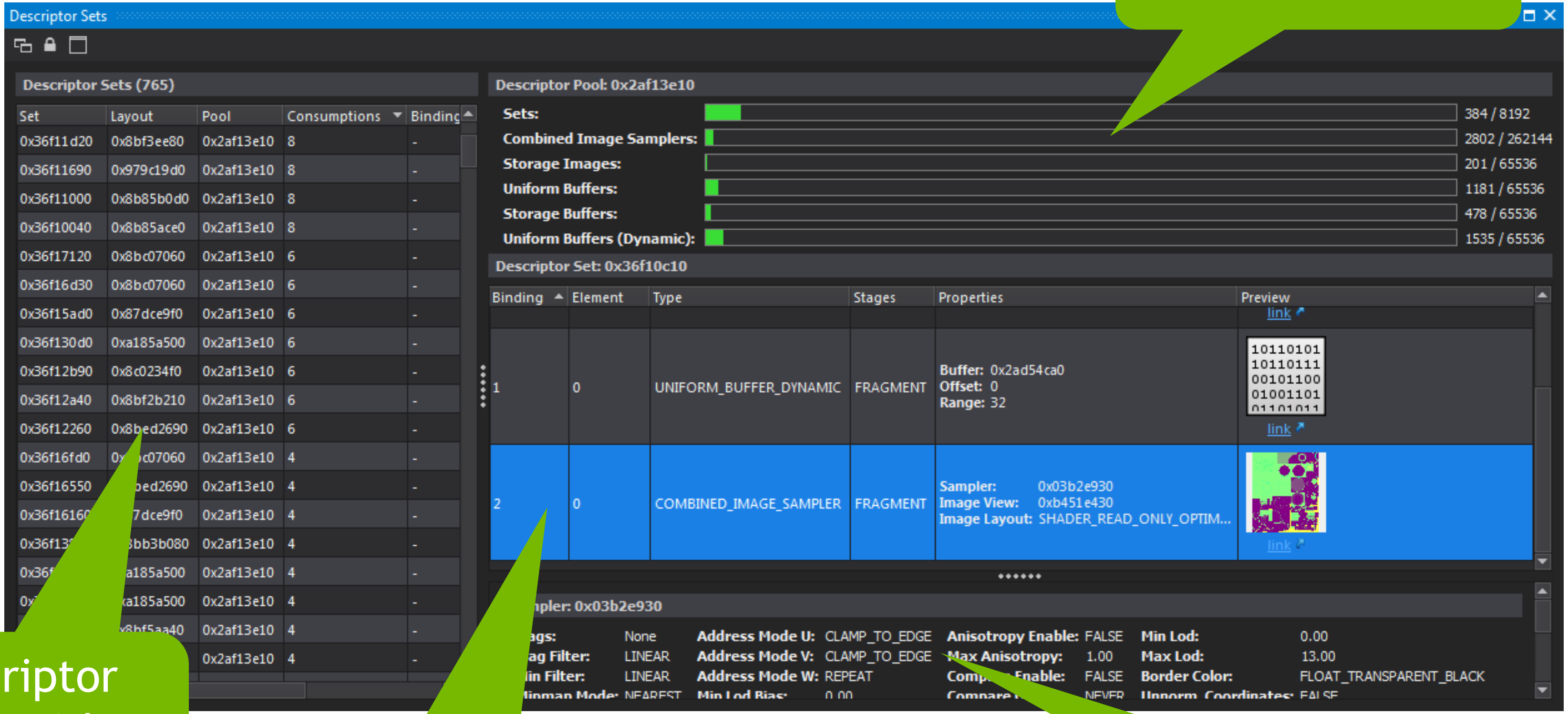
Device Memory



NSIGHT + VULKAN

Descriptor Sets

Pool information



All descriptor objects with usage counts

Associated resources

Selected resource information

Vulkan NVIDIA

NSIGHT + VULKAN

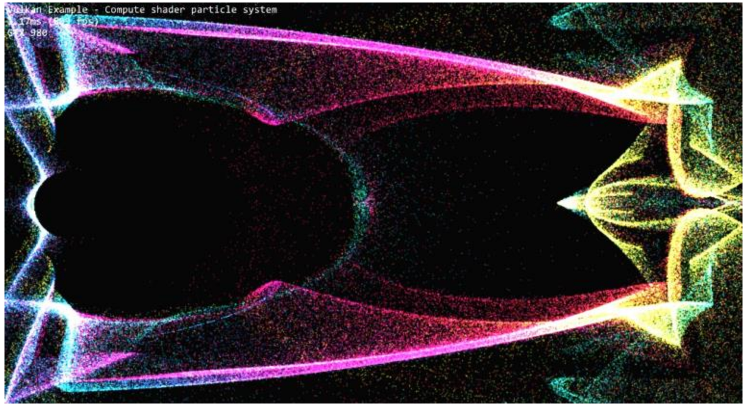
C/C++ Serialization – Challenges Solved

Portability

```c
typedef struct VkMemoryAllocateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceSize       allocationSize;
    uint32_t           memoryTypeIndex;
} VkMemoryAllocateInfo;
```

Frame looping

Where are my particles!?



Trace api

Convert trace into lightweight portable C/C++ project

Maybe useful to experiment with the project rather than full application

Supports original threads, queues etc.

NSIGHT + VULKAN

Roadmap

Profiler & Performance Analysis

Android & Linux Support

Shader Editing

Sparse Texture Support

Improved Resource Barrier Visualization

Future Extensions & Core Releases

JOIN THE NVIDIA DEVELOPER PROGRAM AT
developer.nvidia.com/join

THANK YOU

Christoph Kubisch (ckubisch@nvidia.com, @pixeljetstream)
Ingo Esser (iesser@nvidia.com)

NVIDIA

BACKUP

OBJECT TABLE

```
VkObjectTableCreateInfoNVX createInfo = {VK_STRUCTURE_TYPE_OBJECT_…};
createInfo.maxPipelineLayouts = 1;
createInfo.pObjectEntryTypes  = {VK_OBJECT_ENTRY_PIPELINE_NVX,… };
createInfo.pObjectEntryCounts = {4,… };
…
vkCreateObjectTableNVX(m_device, &createInfo, NULL, &m_table.objectTable);

VkObjectTablePipelineEntryNVX entry = {VK_OBJECT_ENTRY_PIPELINE_NVX};
entry.pipeline = pipelines.usingShaderA;

vkRegisterObjectNVX(m_table.objectTable, (VkObjectTableEntryNVX*)&entry,
                    developerChosenIndex);
```

INDIRECT COMMANDS

```
VkIndirectCommandsLayoutTokenNVX input;
input.type = VK_ INDIRECT_COMMANDS_TOKEN_PIPELINE_NVX;
input.bindingUnit = 0;
input.dynamicCount = 0;
input.divisor = 1;
inputInfos.push_back(input);

input.type = VK_OBJECT_ENTRY_DESCRIPTOR_SET_NVX;
input.bindingUnit = 0;
input.dynamicCount = 1;
input.divisor = 1;
inputInfos.push_back(input);
...
vkCreateIndirectCommandsLayoutNVX(m_device, genCreateInfo, NULL, &m_genLayout);
```

GENERATION

```
vkCmdReserveSpaceForCommandsNVX(cmdSecondary,{resourceTable, indirectLayout, maxCount});

VkIndirectCommandsTokenNVX input;
input.buffer = inputBuffer;
input.type = VK_INDIRECT_COMMANDS_TOKEN_PIPELINE_NVX;
input.offset = pipeOffset;
inputs.push_back(input);

input.type = VK_INDIRECT_COMMANDS_TOKEN_DESCRIPTOR_SET_NVX;
input.offset = matrixOffset;
inputs.push_back(input);

...
vkCmdProcessCommandsNVX(cmdPrimary, {resourceTable, indirectLayout,
                        inputs.size(), inputs.data(), count, cmdTarget, NULL, 0} );
```