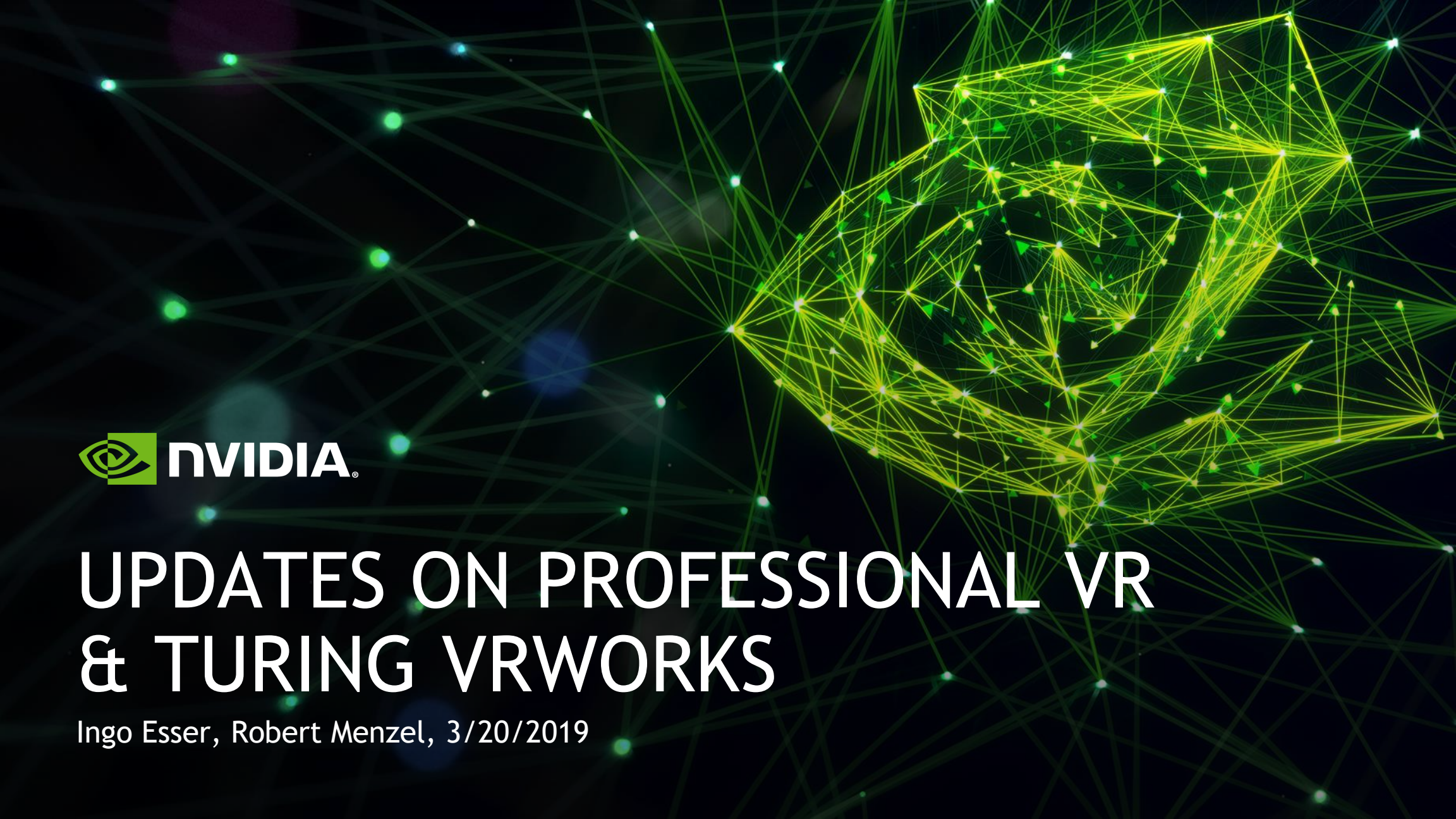


UPDATES ON PROFESSIONAL VR & TURING VRWORKS

Ingo Esser, Robert Menzel, 3/20/2019

AGENDA

Motivation

VR SLI – Multi-GPU Rendering

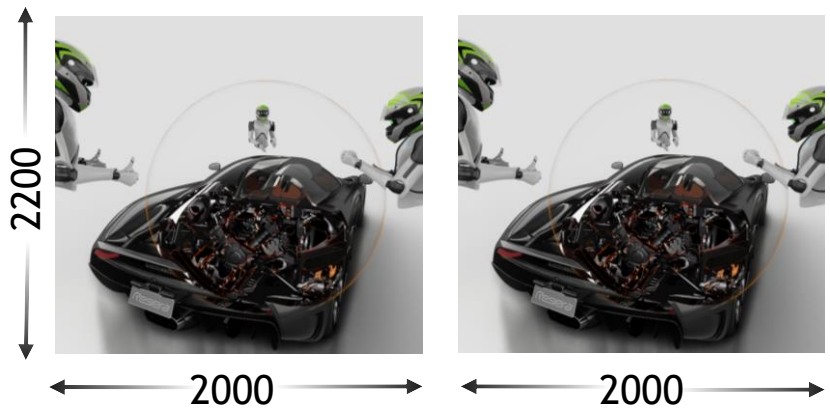
Multi-View Rendering (new in Turing)

Variable Rate Shading (new in Turing)

MOTIVATION

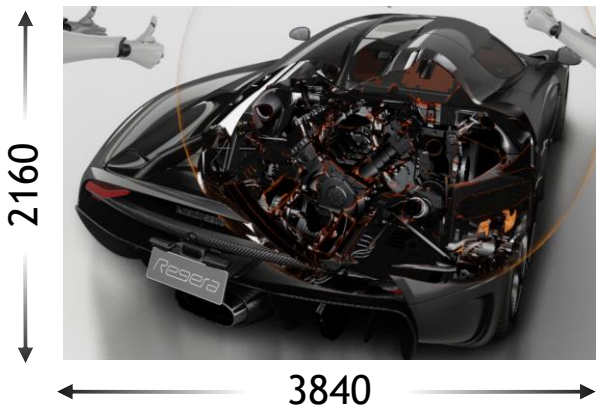
GRAPHICS PIPELINE

VR Workloads

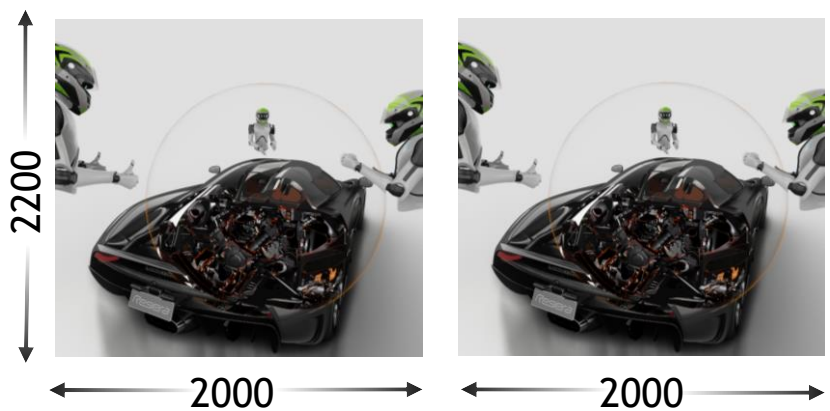


GRAPHICS PIPELINE

VR Workloads



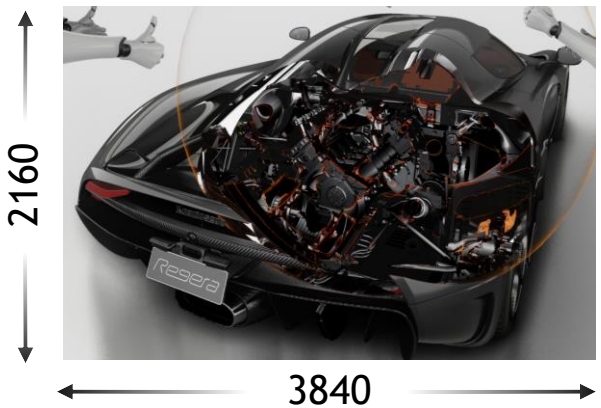
249M Pix/s
N vertices
30 Hz
(4K display)



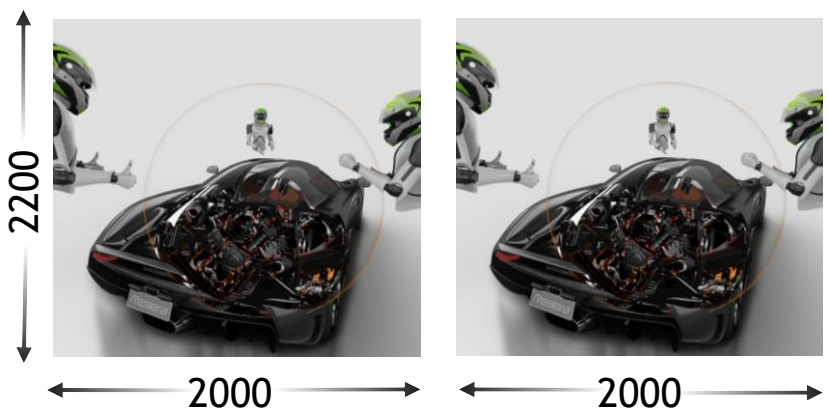
792M Pix/s
2N vertices
90 Hz
(Vive Pro /w oversampling)

GRAPHICS PIPELINE

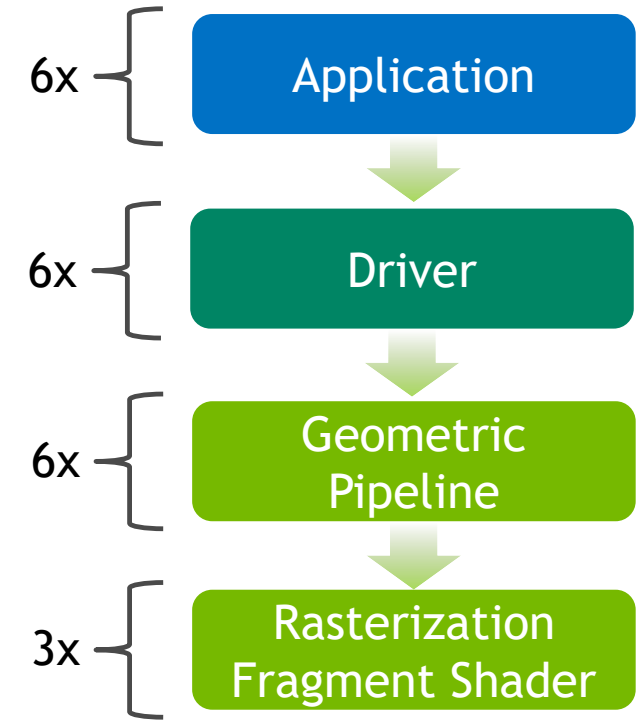
VR Workloads



249M Pix/s
N vertices
30 Hz
(4K display)



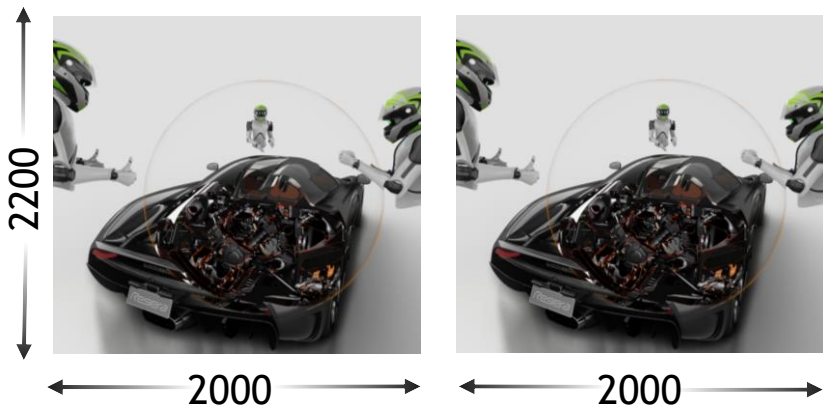
792M Pix/s
2N vertices
90 Hz
(Vive Pro /w oversampling)



GRAPHICS PIPELINE

VR Workloads

249M Pix/s
N vertices
30 Hz
(4K display)

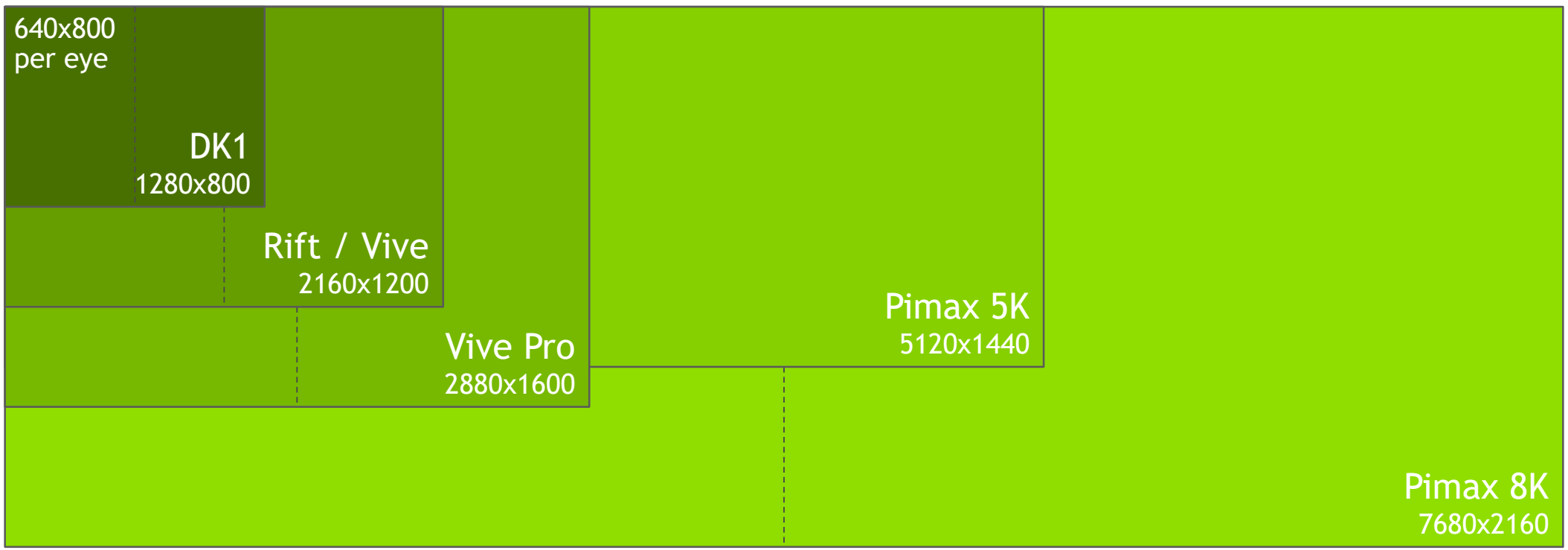


792M Pix/s
2N vertices
90 Hz
(Vive Pro /w oversampling)

VR SLI
- MVR
  - 6x Application
  - 6x Driver
  - 6x Geometric Pipeline
- VRS
  - 3x Rasterization Fragment Shader

HMD RESOLUTIONS

2013 to 2018



NVIDIA VRWORKS

Comprehensive SDK for VR Developers

GRAPHICS

SINGLE PASS STEREO

MULTI-VIEW RENDERING

MULTIRES SHADING

LENS MATCHED SHADING

VARIABLE RATE SHADING

VR SLI

HEADSET

CONTEXT PRIORITY

DIRECT MODE

FRONT BUFFER RENDERING

PROFESSIONAL

WARP & BLEND

SYNCHRONIZATION

GPU AFFINITY

SIMULATION

VRWORKS AUDIO

PHYSX

VIDEO

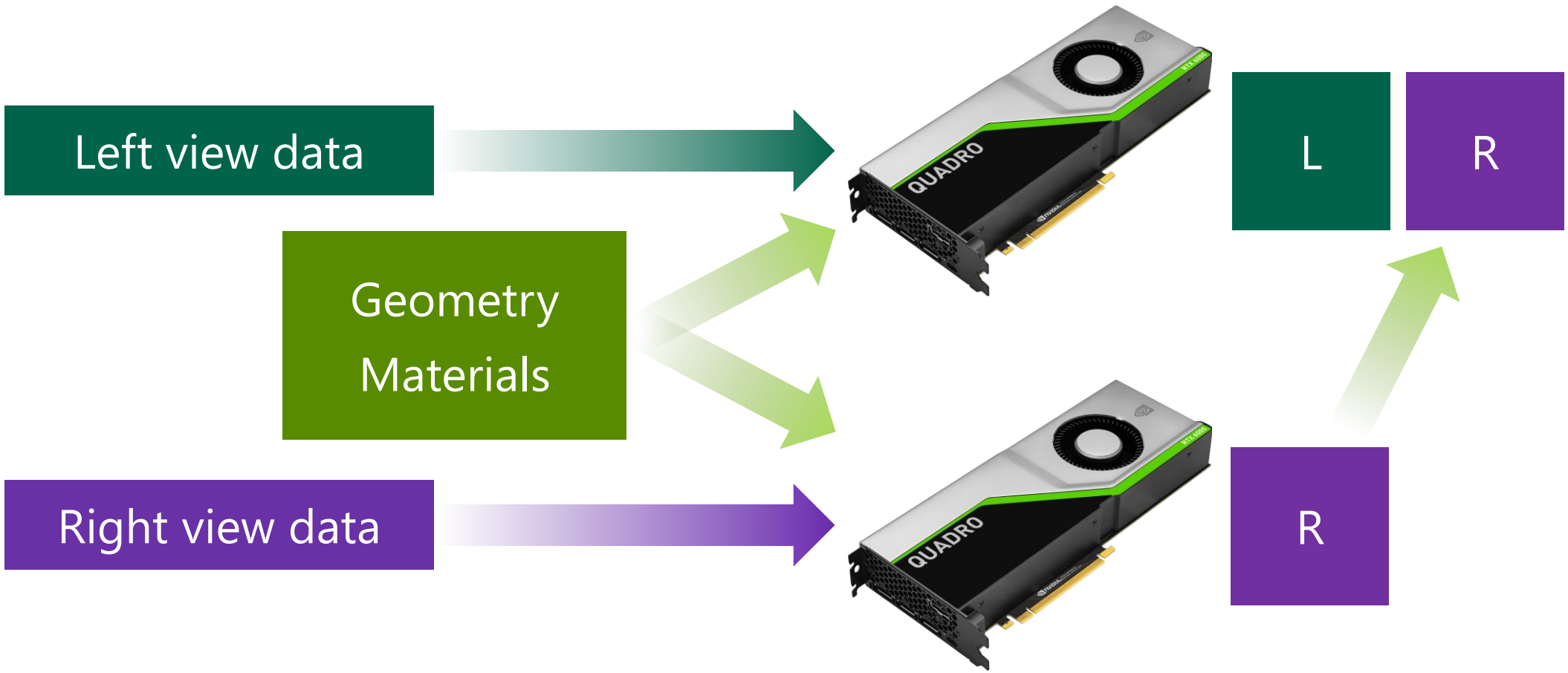
VRWORKS 360 VIDEO

GPUDIRECT FOR VIDEO


VR SLI
SCALING & NVLINK

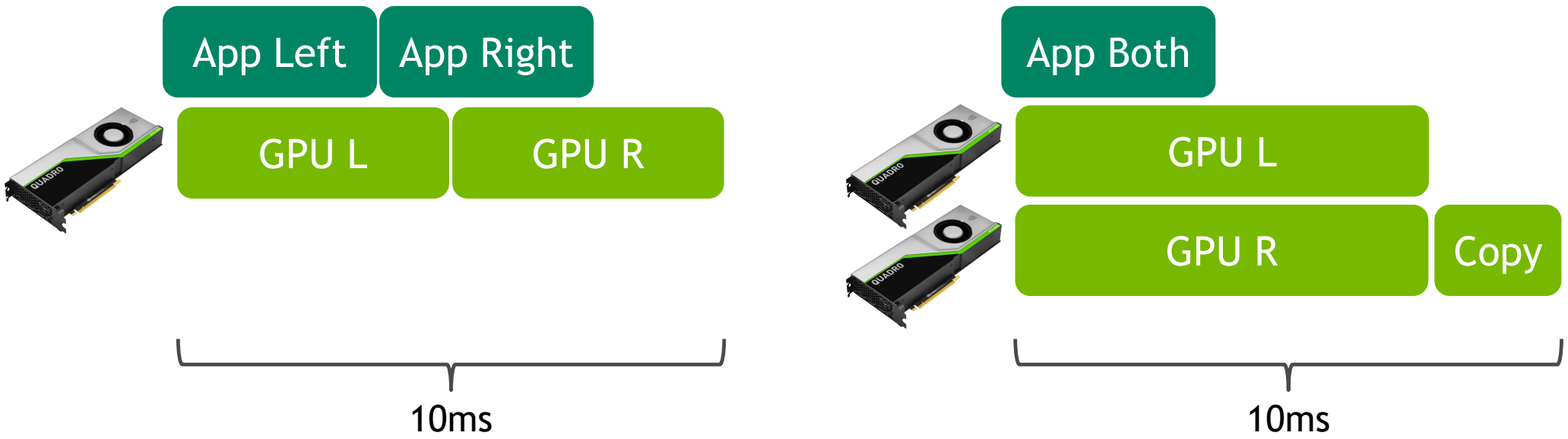
VR SLI

Crash course



VR SLI

Scaling 1 vs 2 GPUs



$$Scaling\ factor\ f = \frac{2 * (t\ - c)}{t}$$

$$frame\ time\ t = 10ms \qquad copy\ time\ c = \frac{frame\ size}{transfer\ speed}$$

VR SLI

Max scaling determined by copy time

$$Scaling\ f = \frac{2 * (t - c)}{t}$$

Typical render* resolution for Vive

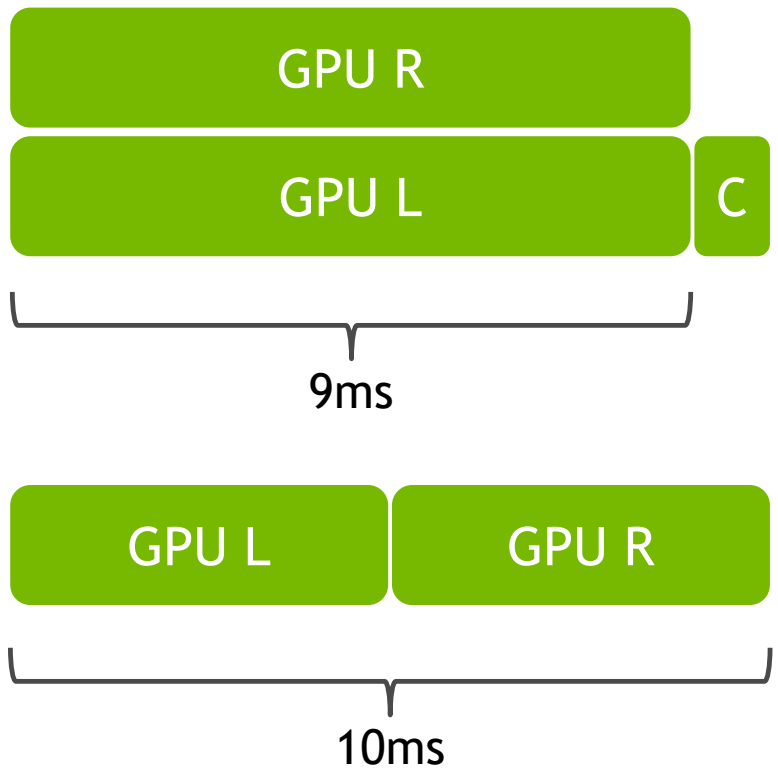
1512 x 1680 (per eye)

Copy time over PCIe3 (@10GB/s)

~1 ms

Max scaling with 11ms frame time

$$\frac{2 * (10ms - 1ms)}{10ms} = 1.8$$



* Vive HMD runtime requests 1.4^2 larger resolution than display resolution

VR SLI

Max scaling determined by copy time

$$Scaling\ f = \frac{2 * (t - c)}{t}$$

Typical render* resolution for Vive Pro

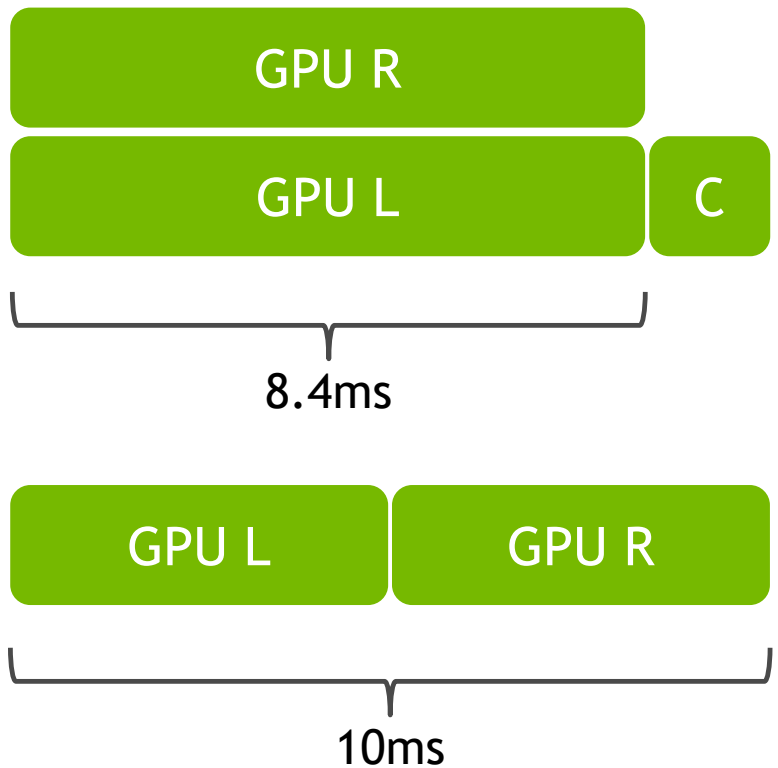
2016 x 2240 (per eye)

Copy time over PCIe3 (@10GB/s)

~1.6 ms

Max scaling with 11ms frame time

$$\frac{2 * (10ms - 1.6ms)}{10ms} = 1.68$$



* Vive HMD runtime requests larger resolution than display resolution

VR SLI

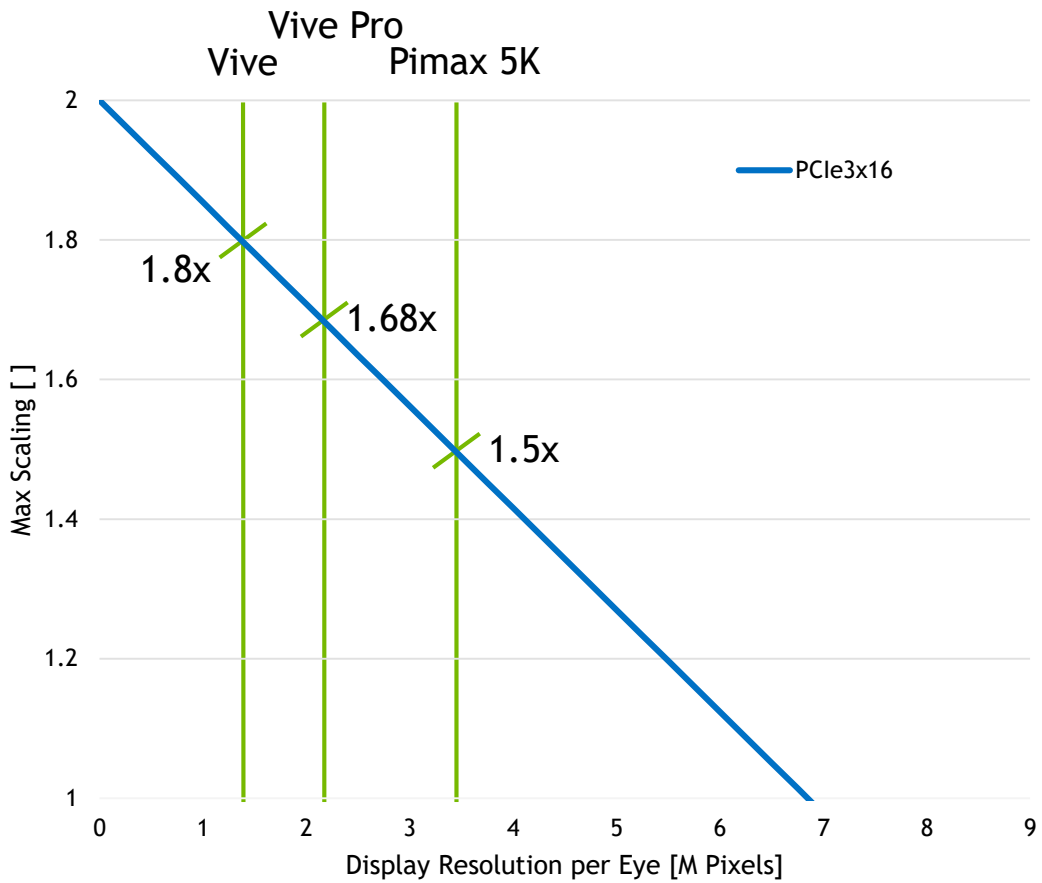
Higher resolutions limit scalability

Low-res HMDs show *screen door effect*

HMDs increase resolutions to improve experience

Vive Pro [Eye]: 1.6ms 1.68x

Pimax 5K Plus: 2.5ms 1.5x



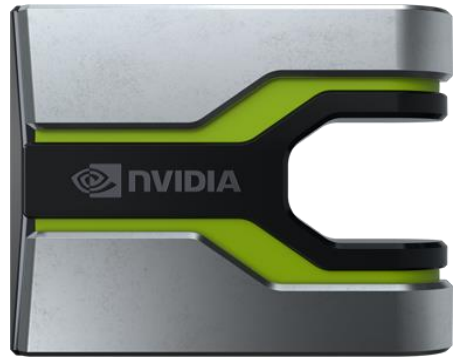
VR SLI

Improve scaling using NVLink

Copy times can hurt scaling with higer resolutions

Quadro RTX 6000 NVLINK: 50GB/s (100GB/s full duplex)

Quadro RTX 5000 NVLINK: 25GB/s (50GB/s full duplex)



NVLink is used automatically if present

No bandwidth sharing with other traffic

Independent of underlying hardware

VR SLI

NVLINK allows scaling with Hi-Res HMDs

NVLINK outperforms PCIe easily

Pimax 5K Plus: 2560 x 1440

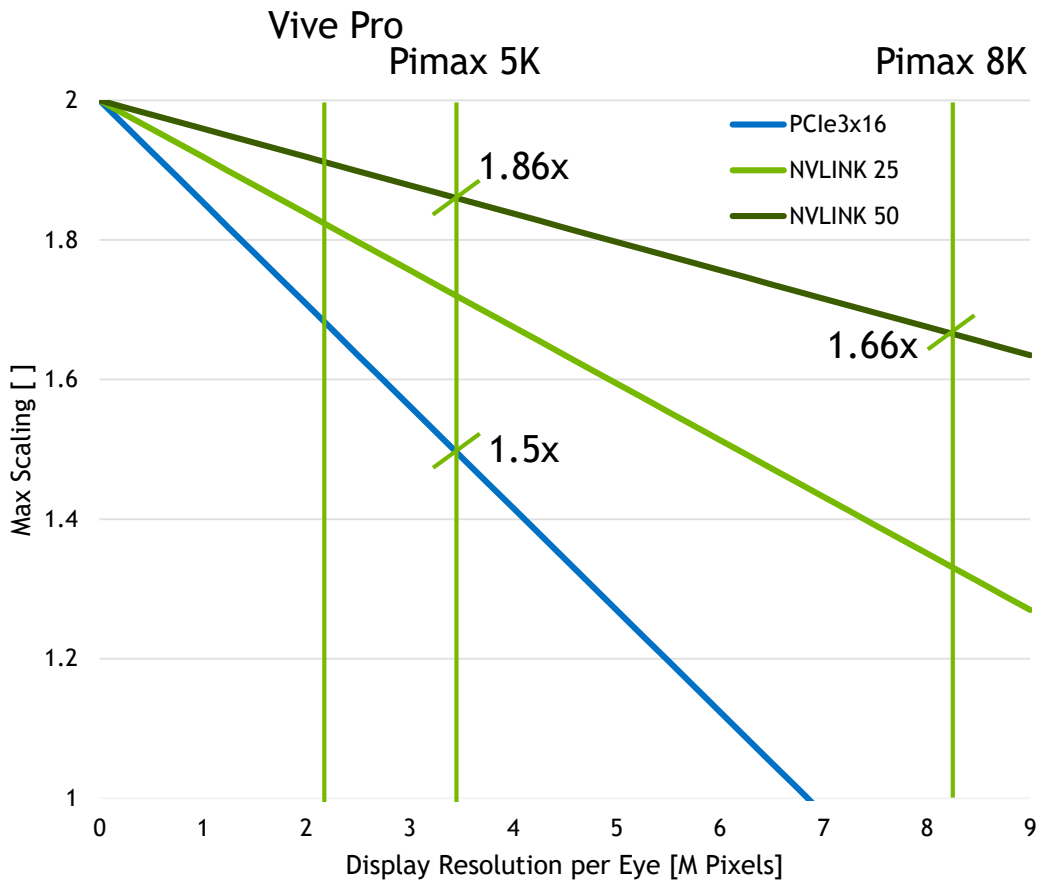
PCIe3x16: 2.5ms 1.5x

NVLINK 50: 0.7ms 1.86x

Pimax 8K: 3840 x 2160

PCIe3x16: 6.1ms 0.79x

NVLINK 50: 1.7ms 1.66x



NVIDIA

NVLINK

Side note: How to NVLINK

NVLINK is transparent – VR SLI automagically uses NVLINK if present

`nvidia-smi` allows to print link information

```
nvidia-smi nvlink
   -s              : Status
   -sc 0bz         : Set counter 0
   -r 0            : Reset counter 0
   -g 0            : Get value
```

Location:

$(ProgramFiles)\NVIDIA Corporation\NVSMI

DCH system: $(windir)\system32

```
c:\Program Files\NVIDIA Corporation\NVSMI>nvidia-smi nvlink -s
GPU 0: Quadro RTX 8000 (UUID: GPU-
         Link 0: 25.781 GB/s
         Link 1: 25.781 GB/s
GPU 1: Quadro RTX 8000 (UUID: GPU-
         Link 0: 25.781 GB/s
         Link 1: 25.781 GB/s

c:\Program Files\NVIDIA Corporation\NVSMI>nvidia-smi nvlink -g 0
GPU 0: Quadro RTX 8000 (UUID: GPU-
         Link 0: Rx0: 439342 KBytes, Tx0: 0 KBytes
         Link 1: Rx0: 437750 KBytes, Tx0: 0 KBytes
GPU 1: Quadro RTX 8000 (UUID: GPU-
         Link 0: Rx0: 0 KBytes, Tx0: 439342 KBytes
         Link 1: Rx0: 0 KBytes, Tx0: 437750 KBytes
```

NVLINK

NVML API support

NVML API (installed with CUDA SDK) allows to query NVLINK state & topology

Enumerate devices, get PCI info, get number of links

```
nvmlDeviceGetCount (&device_count);

nvmlDeviceGetHandleByIndex (i,      &device);
nvmlDeviceGetPciInfo       (device, &pci);
getUInt (device, NVML_FI_DEV_NVLINK_LINK_COUNT, &numLinks);
```

Get link state, speed, remote device PCI info (topology information)

```
nvmlDeviceGetNvLinkState (device, j, &isActive);
getUInt (device, NVML_FI_DEV_NVLINK_SPEED_MBPS_L0 + j, &speed);
nvmlDeviceGetNvLinkRemotePciInfo (device, j, &pci);
```

Additional API to query link capabilities, error/data counters, etc.

NVLINK

NVAPI access – under development

NVLINK API is getting comparable functionality

Enumerate devices

```
NvAPI_EnumPhysicalGPUs (
        NvPhysicalGpuHandle nvGPUHandle[NVAPI_MAX_PHYSICAL_GPUS],
        NvU32 *pGpuCount );
```

Get link number, speed, topology

```
NvAPI_GPU_NVLINK_GetStatus (
        NvPhysicalGpuHandle hPhysicalGpu,
        NVLINK_GET_STATUS* statusParams );
```

NVAPI also allows to query capabilities, error / data counters, etc.

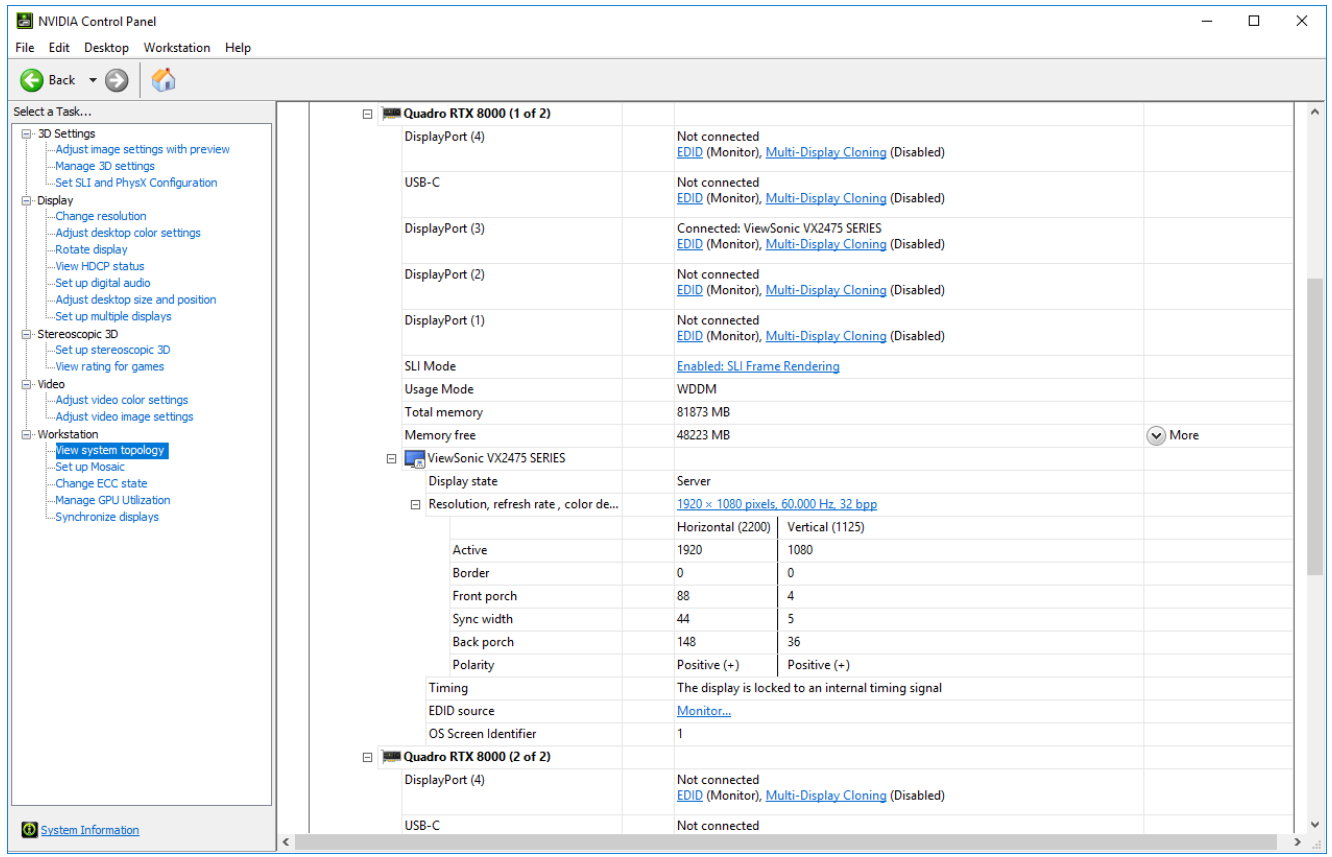
NVLINK

NVIDIA Quadro Control Panel – under development

NVIDIA Quadro Control Panel

Workstation
-> View System Topology

NVLINK information



VR SLI
OPENGL MULTICAST 2

OPENGL VR SLI: MULTICAST 2

Feedback on Multicast led to new functionality

Command & data broadcast

BufferSubData to specific GPU

CopyImageSubData & CopyBufferSubData

GPU-GPU Framebuffer Blit

Global barrier & directed sync functions

GPU Masks

Per-GPU sample locations

Per-GPU queries

Dynamic Multicast toggle (WGL_NV_multigpu_context)

GPU_ID built-in in GLSL shader

Per-GPU viewports & scissors

Texture & Buffer upload mask

Asynchronous copies

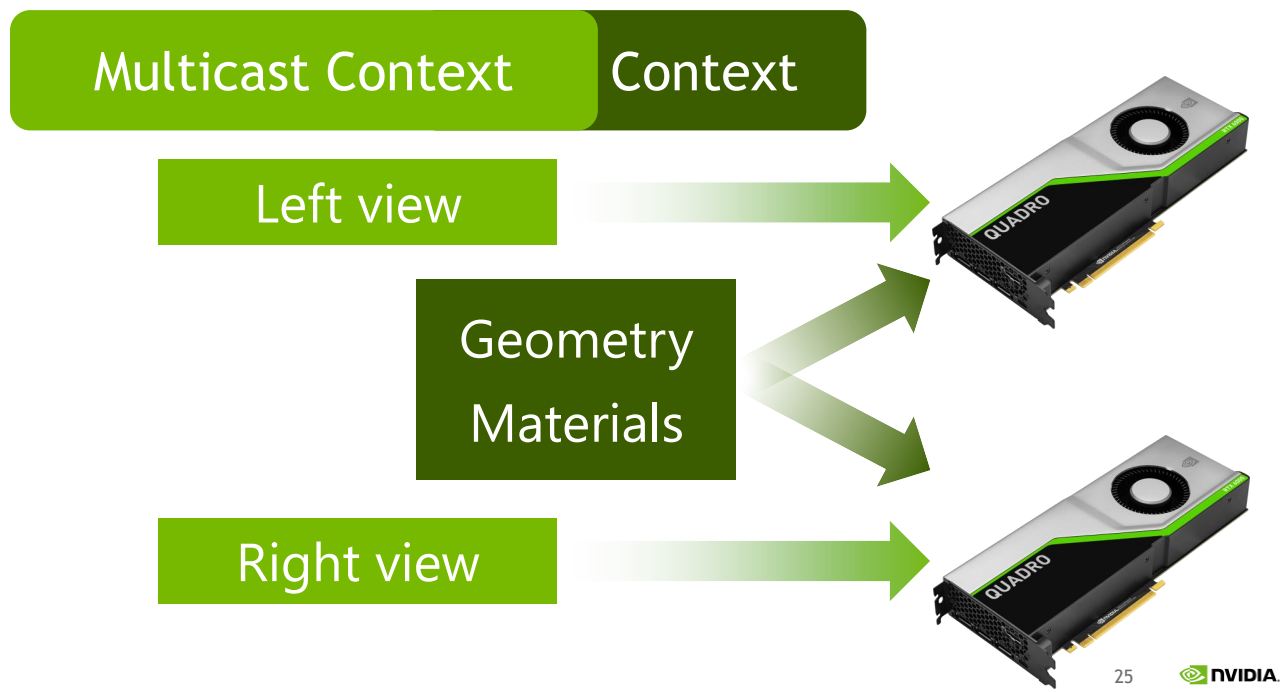
MULTICAST 2

Dynamic SLI mode

New extension WGL_NV_multigpu_context: Request SLI mode per context

No need to restart application

Possible to share resources between contexts



MULTICAST 2

Dynamic SLI mode

New extension WGL_NV_multigpu_context: Request SLI mode per context

No need to restart application

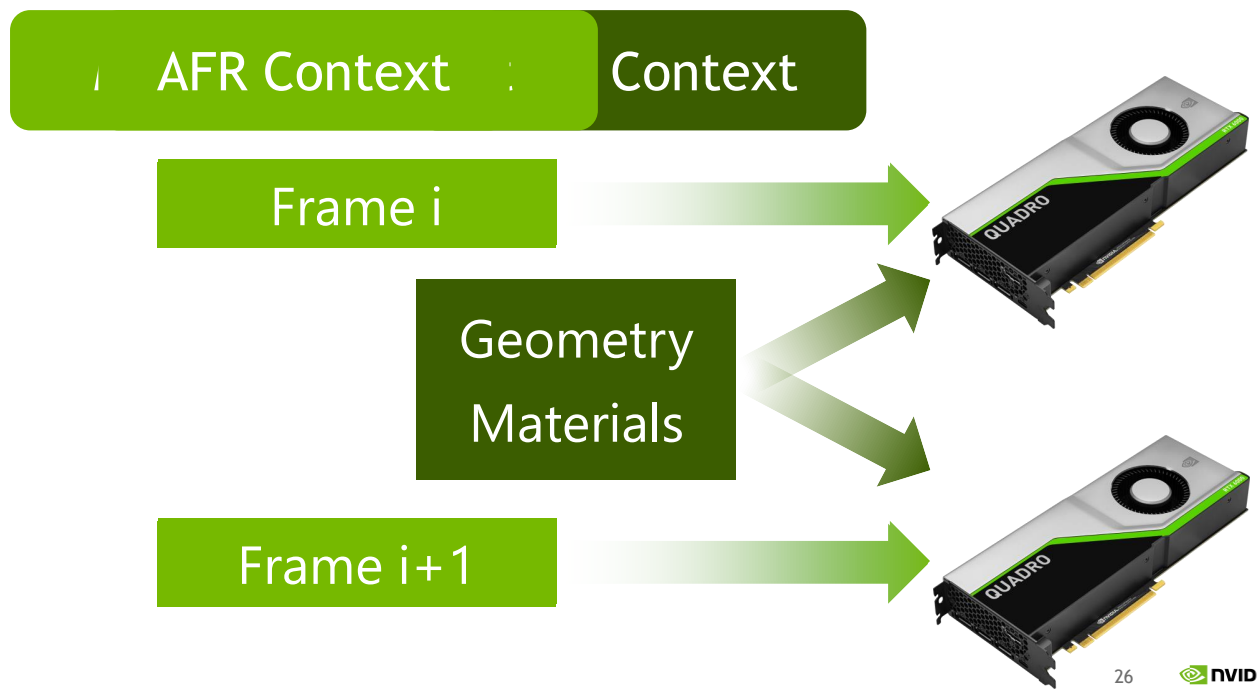
Possible to share resources between contexts

On toggle:

Clean up per-GPU resources

Keep scene data

Alternate Frame Rendering (AFR)



MULTICAST 2

GPU ID built-in: `gl_DeviceIndex`

Multicast v1 required per-GPU uploads

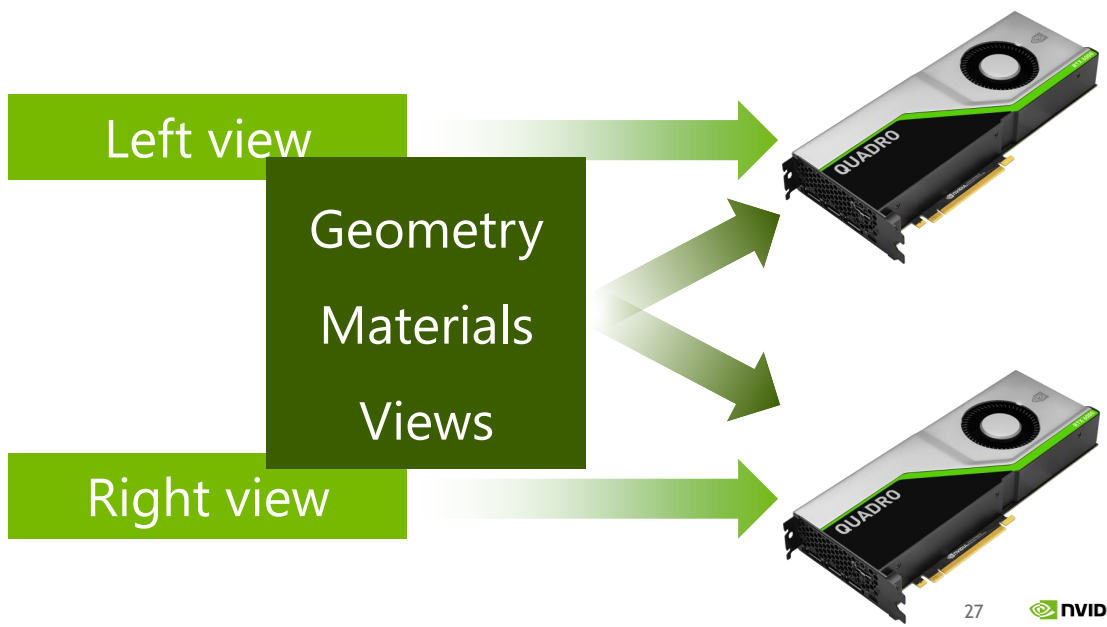
Larger code changes in some renderers

Add shader built-in: `gl_DeviceIndex`

- Upload all views to all GPUs
- Use per-GPU data in shaders

Renderer can remain unchanged

Just modify shaders instead



MULTICAST 2

Per-GPU Viewports & Scissors

Add new function to set viewports and scissors per GPU

```
glMulticastViewportArrayvNVX( ... );

glMulticastScissorArrayvNVX( ... );
```

Per-GPU Lens Matched Shading

MULTICAST 2

Per-GPU Viewports & Scissors

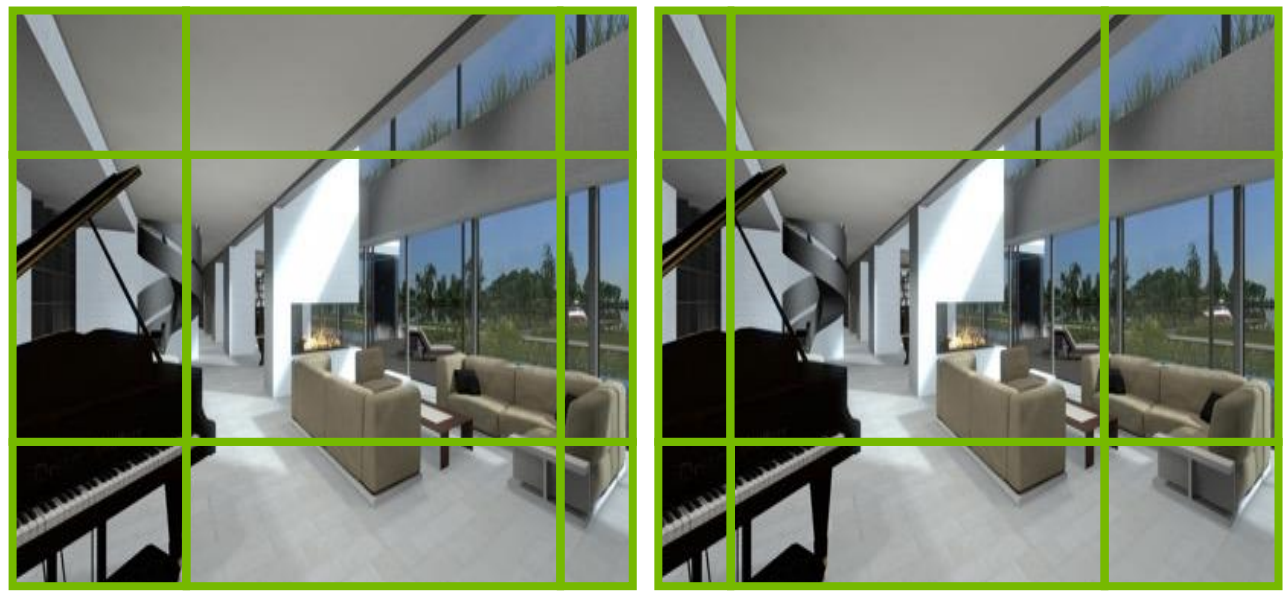
Add new function to set viewports and scissors per GPU

```
glMulticastViewportArrayvNVX( ... );

glMulticastScissorArrayvNVX( ... );
```

Per-GPU Lens Matched Shading

Per-GPU Multi Resolution Shading



MULTICAST 2

Per-GPU Viewports & Scissors

Add new function to set viewports and scissors per GPU

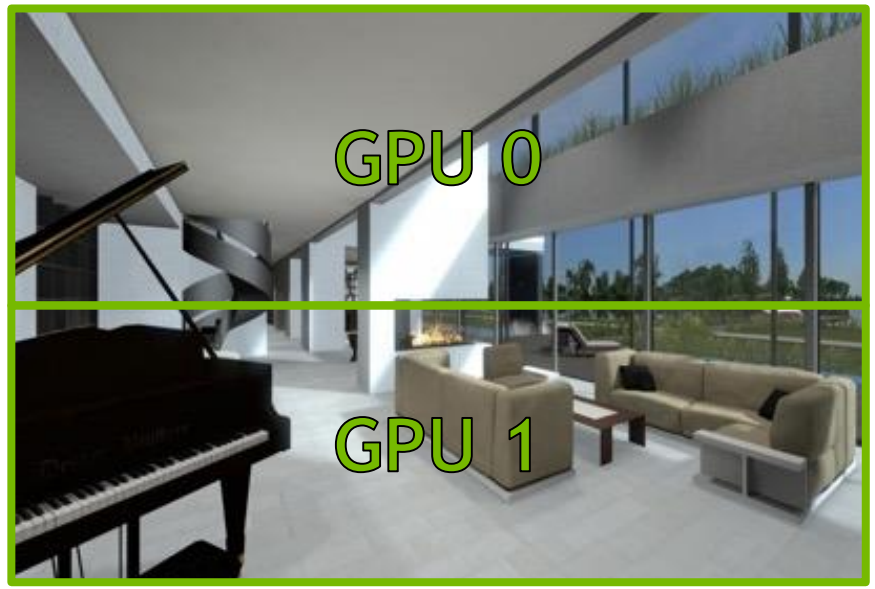
```
glMulticastViewportArrayvNVX( ... );

glMulticastScissorArrayvNVX( ... );
```

Per-GPU Lens Matched Shading

Per-GPU Multi Resolution Shading

Easily set up Split Frame Rendering (SFR)



MULTICAST 2

Texture & Buffer Upload Mask

Multicast provides per-GPU buffer uploads

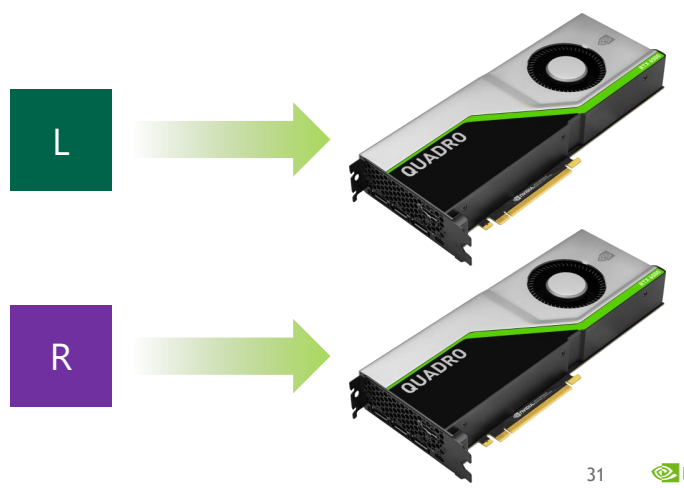
Asymmetrical functionality wrt texture upload functions

Add new mask function to modify texture & buffer uploads

```
glUploadGpuMaskNVX( GLbitfield mask );
```

Useful for simpler per-GPU texture streaming

Conserve PCIe bandwidth



MULTICAST 2

Asynchronous Copies

Multicast copies stall source GPU while copy takes place

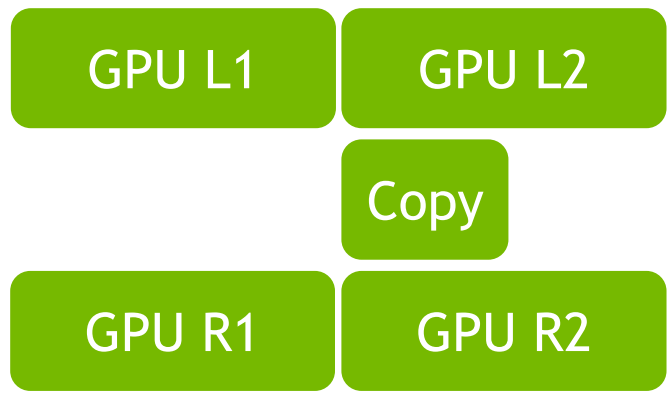
Easy to use because of implicit synchronization

New copy functions do not stall, but also need more synchronization

```
glAsyncCopyBufferSubDataNVX( ... );
glAsyncCopyImageSubDataNVX( ... );
```

Copy while both GPUs can continue rendering

Allows for more complex rendering algorithms



MULTICAST 2

Asynchronous Copies – Use case

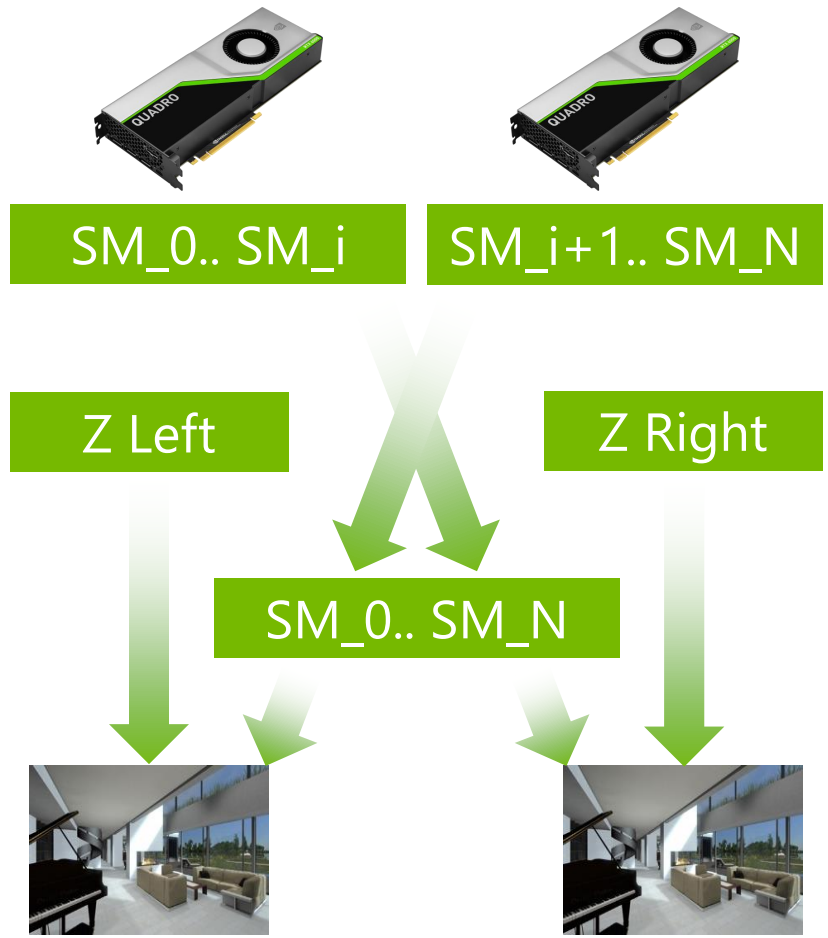
Render shadow maps (SM)

Start async copies of SMs to other GPU

Render z-prepass per GPU & eye

Wait for copy to finish

Render output images




VR SLI
+ QUADRO SYNC

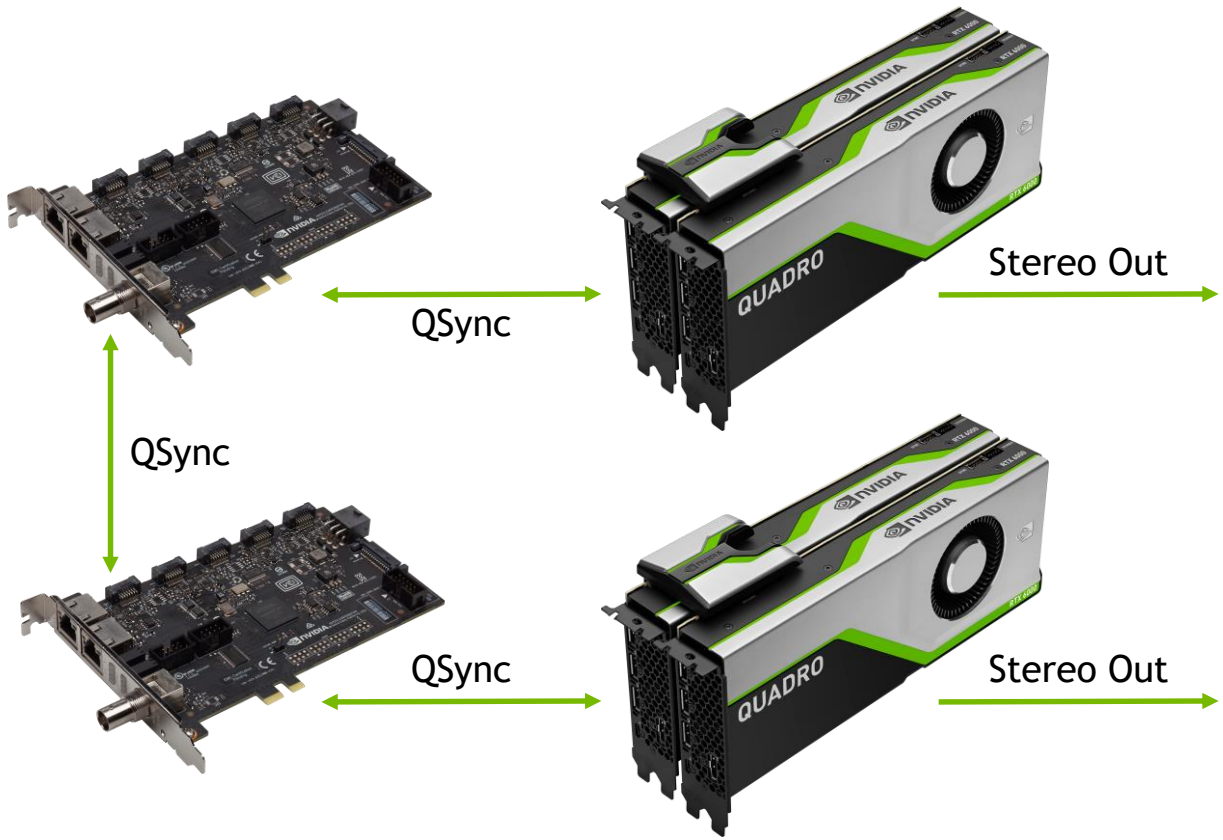
QUADRO SYNC + VR SLI

Support for new hardware configurations

Use case

- CAVE systems
- Each node generates L / R image
- Scan out through Quad Buffered Stereo
- Perfect for VR SLI

VR SLI + Quadro Sync + Quad Buffered Stereo supported with 418.81 and newer



QUADRO SYNC + VR SLI + QBS

Synthetic Speed-Of-Light Benchmark

Frame time for 0..800 M triangles

Render stereo, compare VR SLI on/off

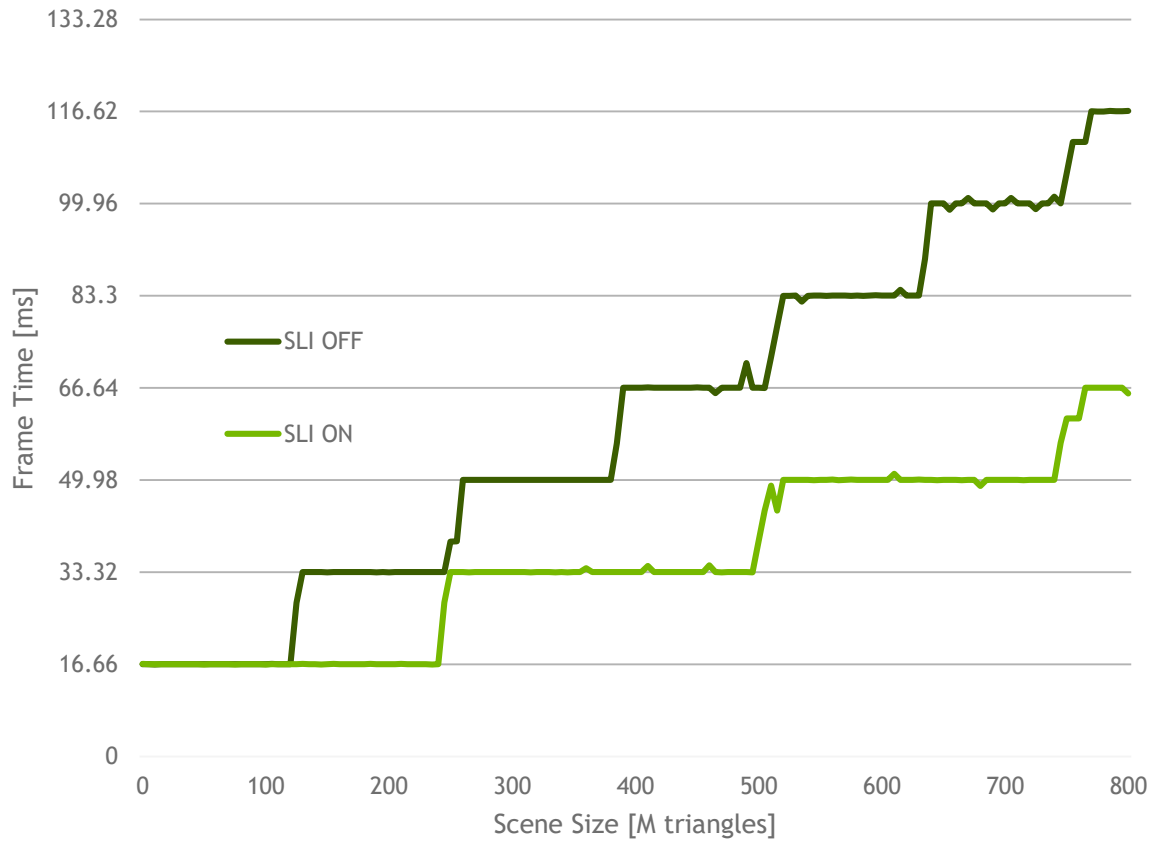
System performance nearly doubles:

16ms: 240M vs 125M triangles

32ms: 495M vs 250M triangles

Stereo: Rendering scene twice per frame

2x Quadro RTX6000 + NVLINK:
480M triangles in 16ms




VR SLI
VULKAN DEVICE GROUPS

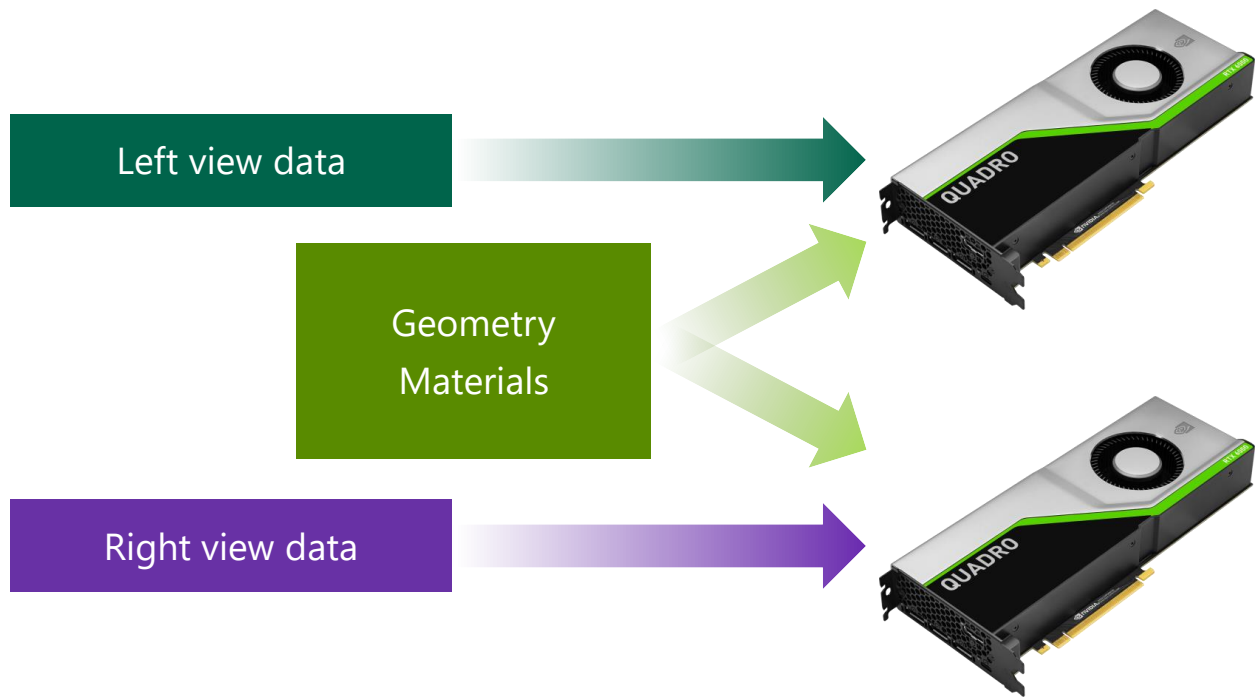
VR SLI

Vulkan - subsetAllocation

Vulkan provides VR SLI through the `VK_KHR_device_group` extension

Similar per-GPU functionality

- Uploads
- Render commands
- GPU-GPU transfers



VR SLI

Vulkan - subsetAllocation

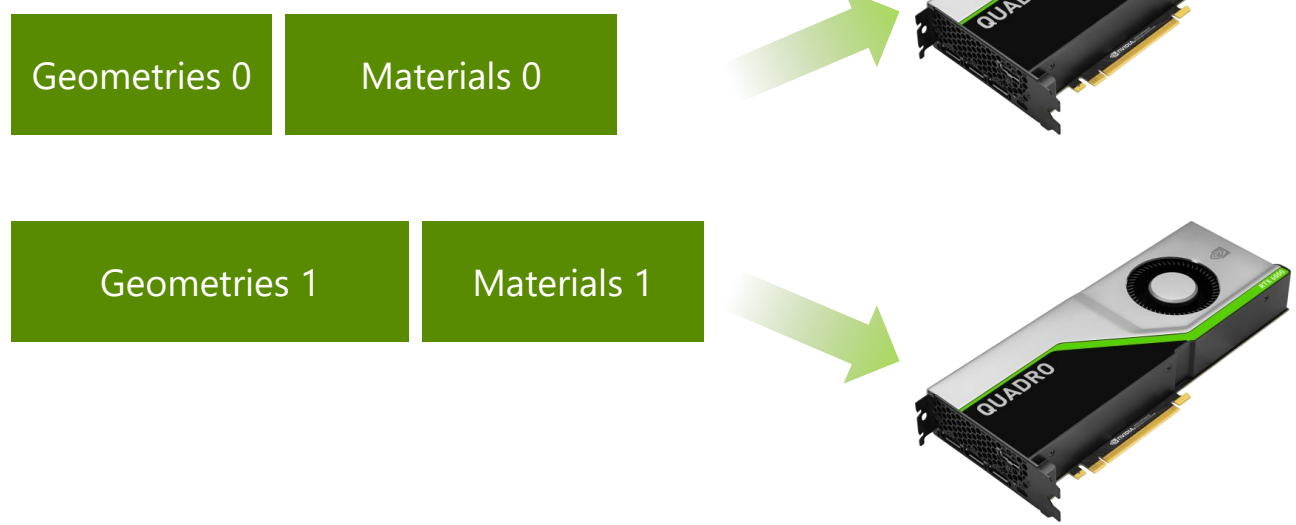
Vulkan provides VR SLI through the `VK_KHR_device_group` extension

Similar per-GPU functionality

- Uploads
- Render commands
- GPU-GPU transfers

Upcoming support:

- Per-GPU memory allocations



VR SLI

Recap

VR SLI covers a wide variety of workloads

Almost perfect load balancing between left/right eye and two GPUs

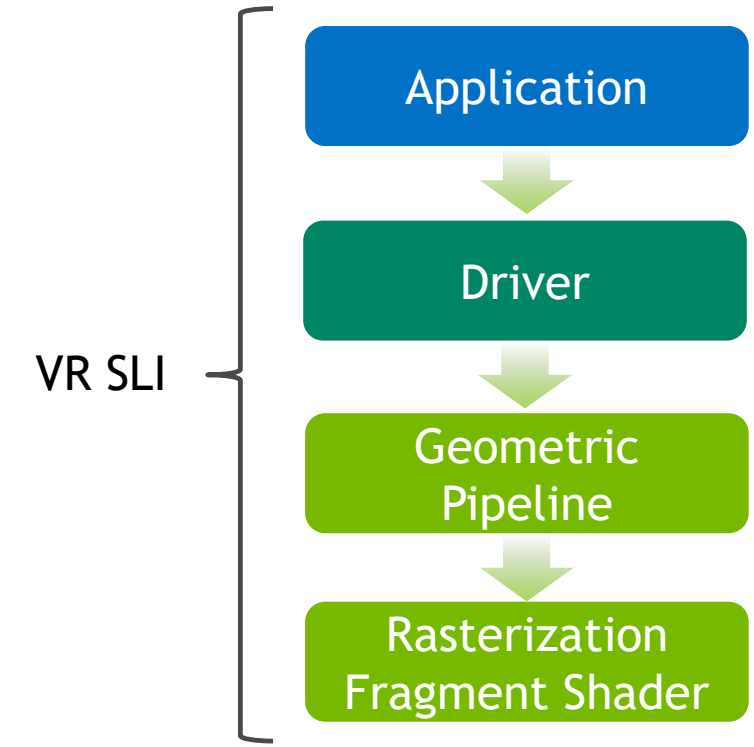
Copy overhead and view independent workloads limit scaling

NVLink can help improve scaling

OpenGL: `GL_NV_gpu_multicast / GL_NVX_gpu_multicast2`

Vulkan: `VK_KHR_device_group` (core in VK 1.1)

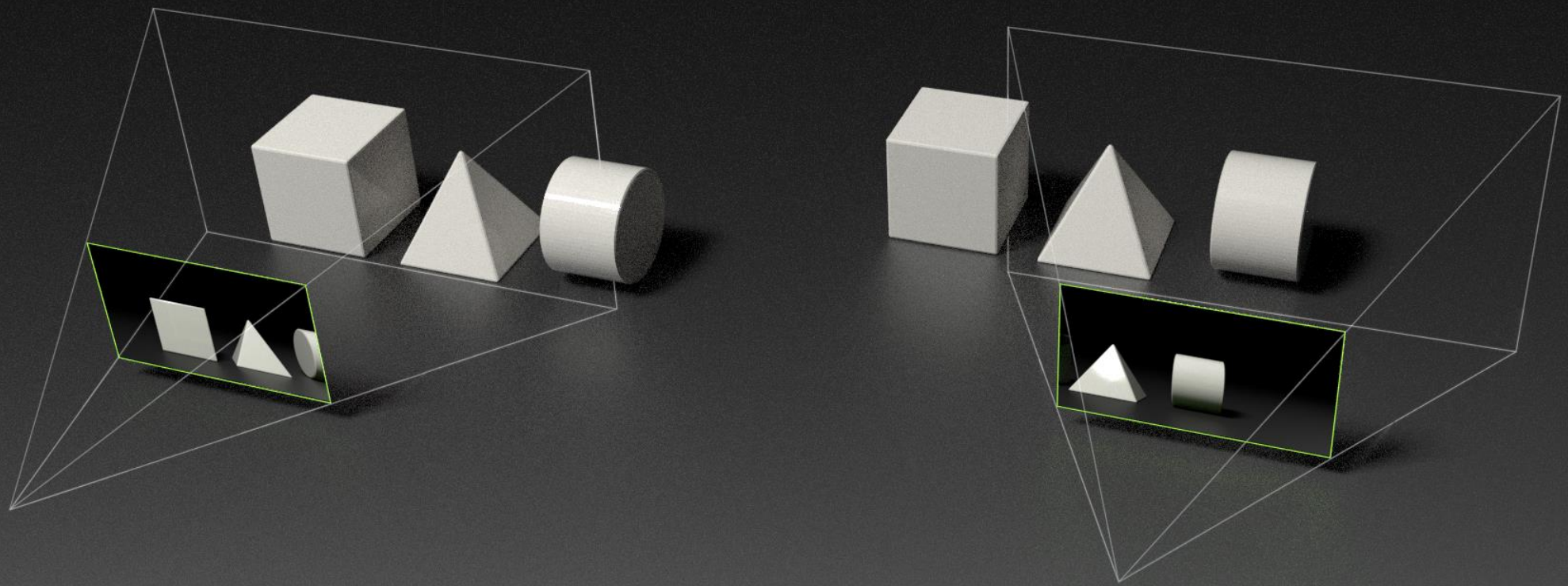
DX11: NVAPI



MULTI-VIEW RENDERING

TWO PASS STEREO RENDERING

2 Full Geometry Passes



Left Eye (Pass 1)

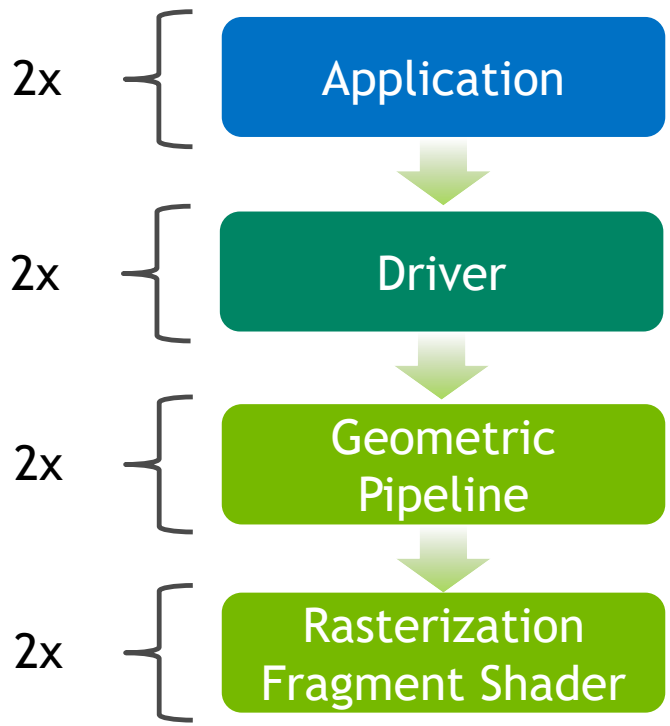
Right Eye (Pass 2)

TWO PASS RENDERING

Mono to Stereo

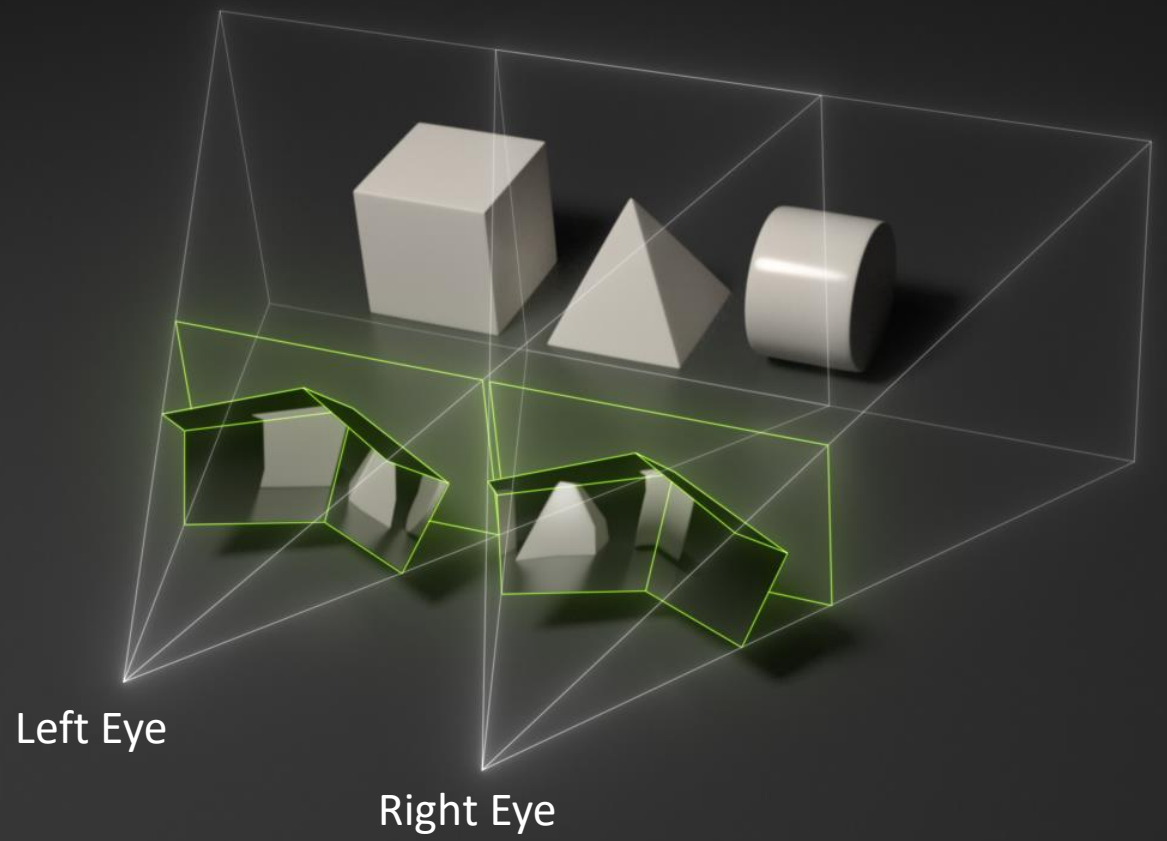
Workload in all steps of the pipeline double.

Getting CPU bound fast, especially in CAD!



SINGLE-PASS-STEREO

1 Pass on Pascal



SINGLE-PASS-STEREO

Mono to Stereo

Cut CPU time in half

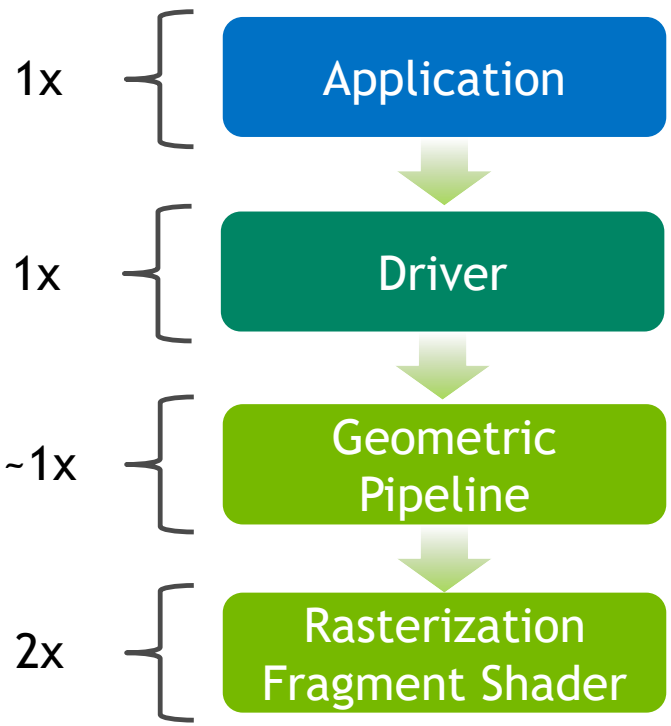
Cut VTG processing (nearly) in half

No change in raster & shading

DX: NVAPI

Vulkan: `VK_KHR_Multiview` (core in VK 1.1) & `VK_NVX_multiview_per_view_attributes`

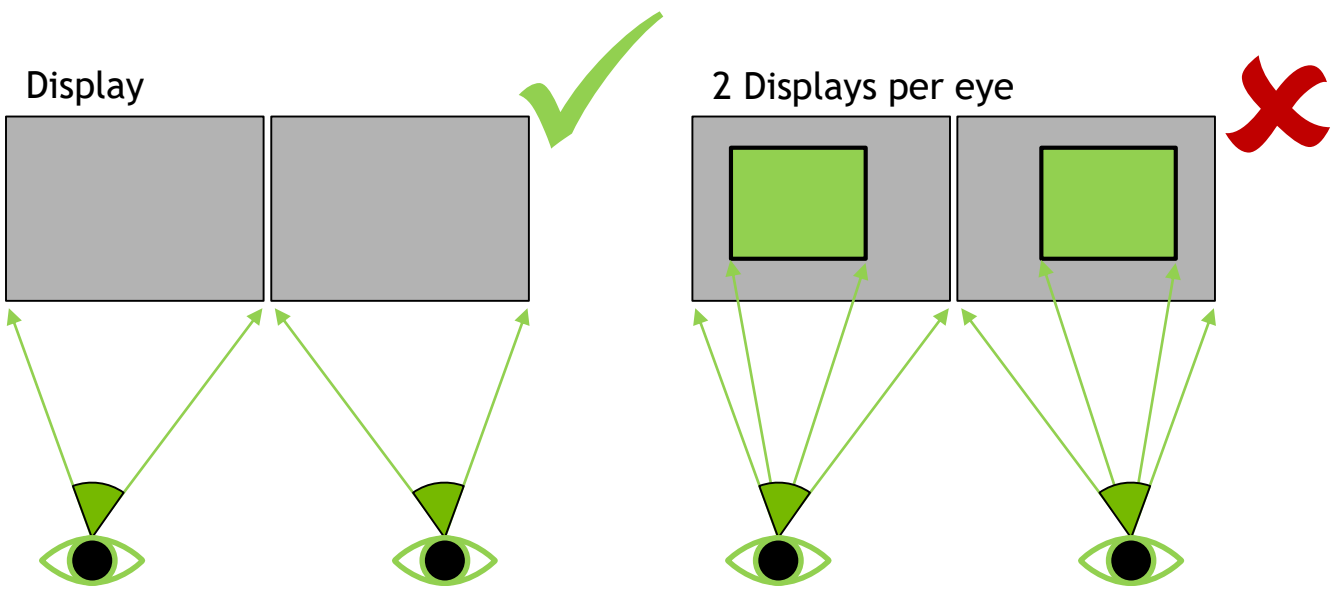
OpenGL: `GL_NV_stereo_view_rendering`

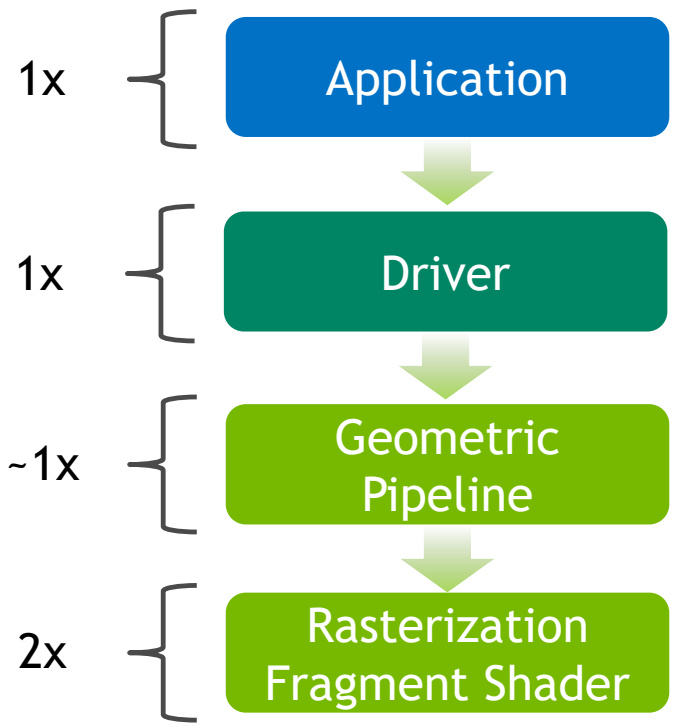


SINGLE-PASS-STEREO

Limitations

Two views only



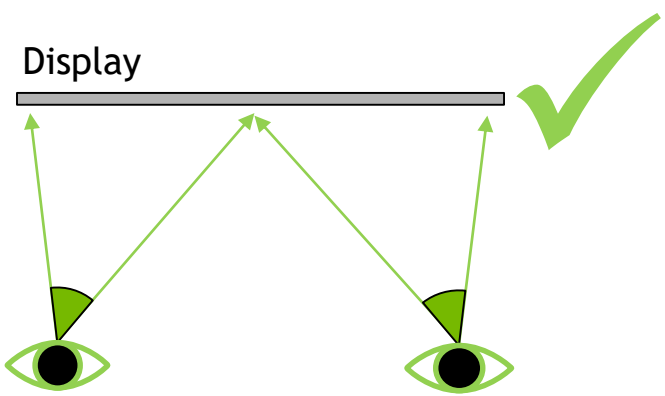


SINGLE-PASS-STEREO

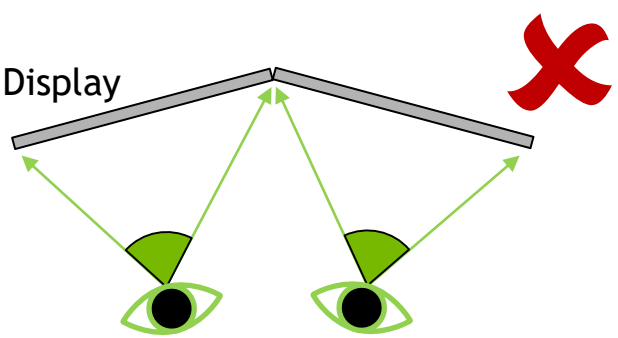
Limitations

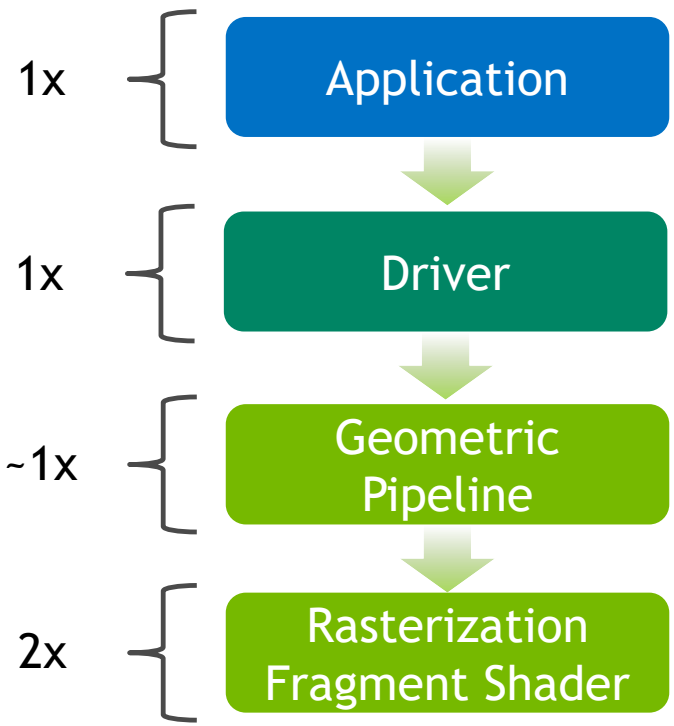
Two views only

Only change X-coordinate



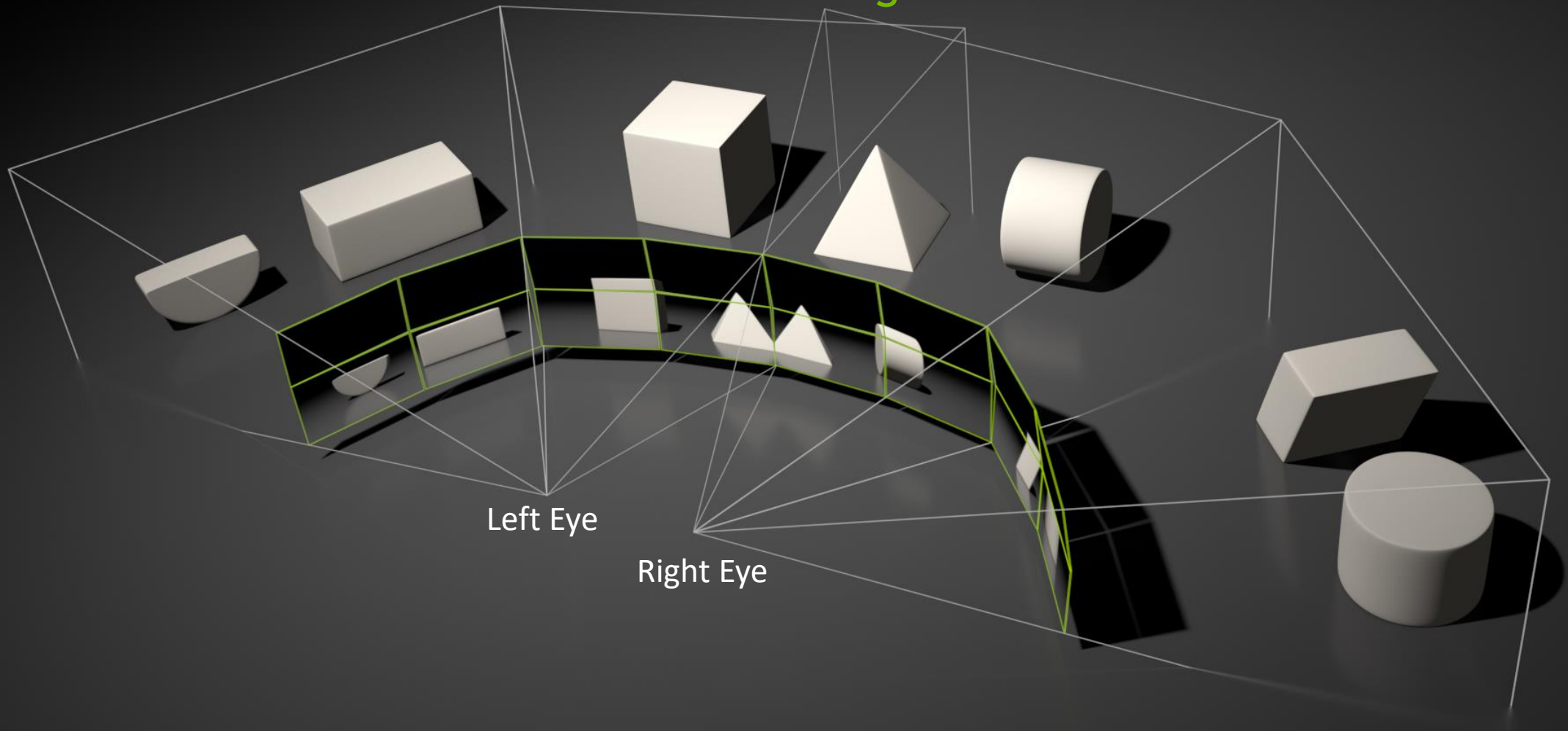
Canted displays (wide FoV)





MULTI-VIEW RENDERING

Next Generation Single-Pass-Stereo

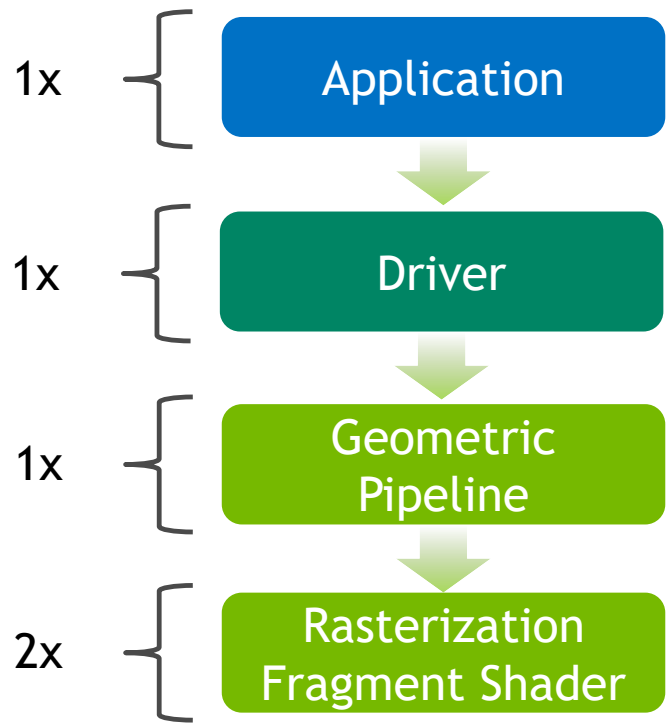


MULTI-VIEW RENDERING

Turing

Up to 4 arbitrary views in hardware.

Up to 32 arbitrary views in software.



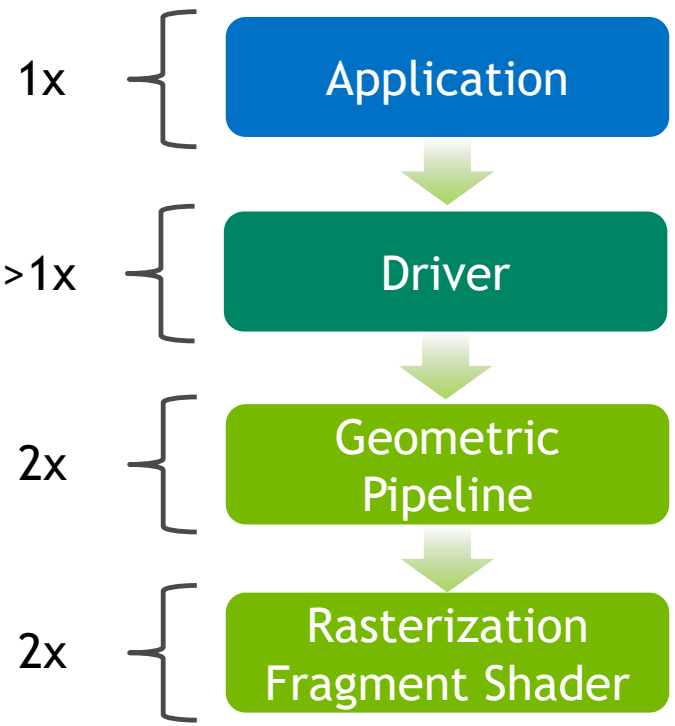
MULTI-VIEW RENDERING

Pre-Turing

Up to **32 arbitrary** views in software.

Still significant reduction in CPU overhead.

Reduces number of code paths.



MULTI-VIEW RENDERING

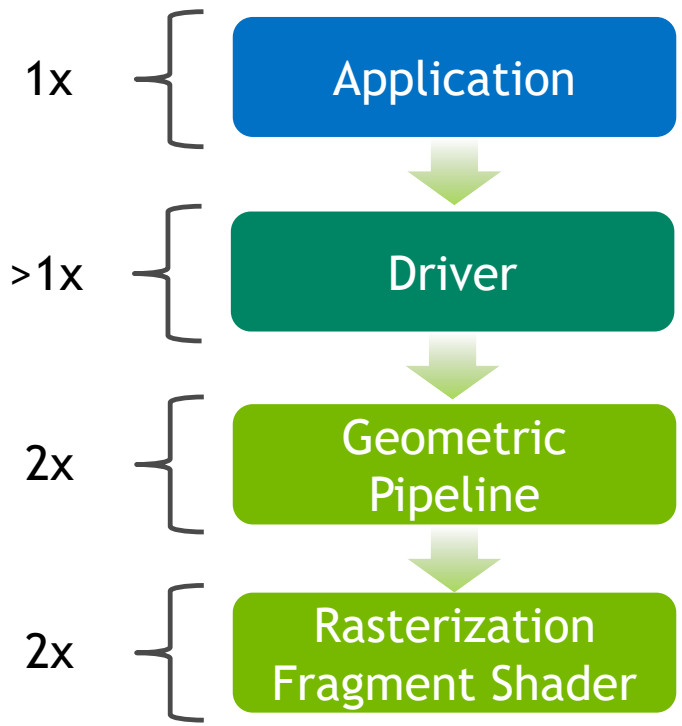
APIs

DX11: NVAPI

DX12: via View Instancing

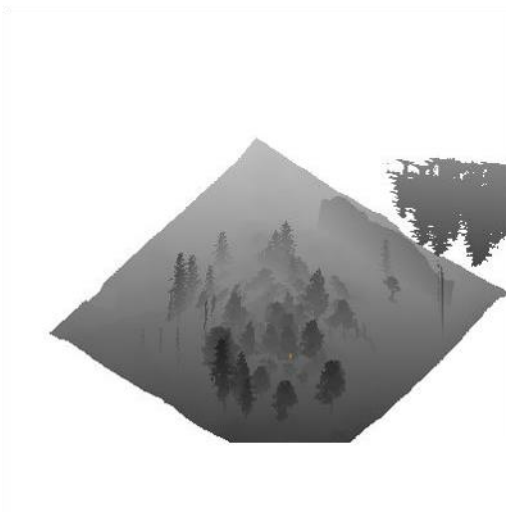
Vulkan: `VK_KHR_Multiview` (core in VK 1.1)

OpenGL: `GL_OVR_multiview & GL_OVR_multiview2`



MULTI-VIEW RENDERING

Non-VR Use-cases



Multiple Shadow Maps in one pass
(multiple light sources, cascaded shadow maps etc.)

MULTI-VIEW RENDERING

Example: OpenGL

Render to multiple layers (just like Single-Pass-Stereo)

Provide data for all views to Vertex Shader

Handle view dependent operations via new built-in `gl_ViewID_OVR`

Minimize number of varyings dependent on `gl_ViewID_OVR`!

MULTI-VIEW RENDERING

Example: OpenGL

```
mat4 modelViewProjection = viewProjMatrix[gl_ViewID_OVR] * model;
gl_Position = modelViewProjection * vertexPos;
```

MULTI-VIEW RENDERING

Example: OpenGL

```
mat4 modelViewProjection = viewProjMatrix[0] * model;
gl_Position = modelViewProjection * vertexPos;

if (gl_ViewID_OVR == 1) {
  mat4 modelViewProjection2 = viewProjMatrix[1] * model;
  vec4 pos = modelViewProjection2 * vertexPos;
  gl_Position.x = pos.x; // hint that only X depends on the viewID to mimic SPS
}
```

MULTI-VIEW RENDERING

Turing Mesh Shaders

Mesh Shaders can be used with Multi-View Rendering!

But:

not **implicitly** like Vertex/Tessellation/Geometry Shaders

but **explicitly** in the Mesh Shader

max 4 views

MULTI-VIEW RENDERING

Turing Mesh Shaders

Mesh Shader:

```
out gl_MeshPerVertexNV {

    vec4  gl_Position;

} gl_MeshVerticesNV[];



…

gl_MeshVerticesNV[i].gl_Position =
MVP * vertex;
```

Mesh Shader with explicit Multi-View Rendering

```
out gl_MeshPerVertexNV {

    perviewNV vec4 gl_PositionPerViewNV[];

} gl_MeshVerticesNV[];



…

gl_MeshVerticesNV[i].gl_PositionPerViewNV[ v ] =
MVP[ v ] * vertex;
```

MULTI-VIEW RENDERING

Limitations

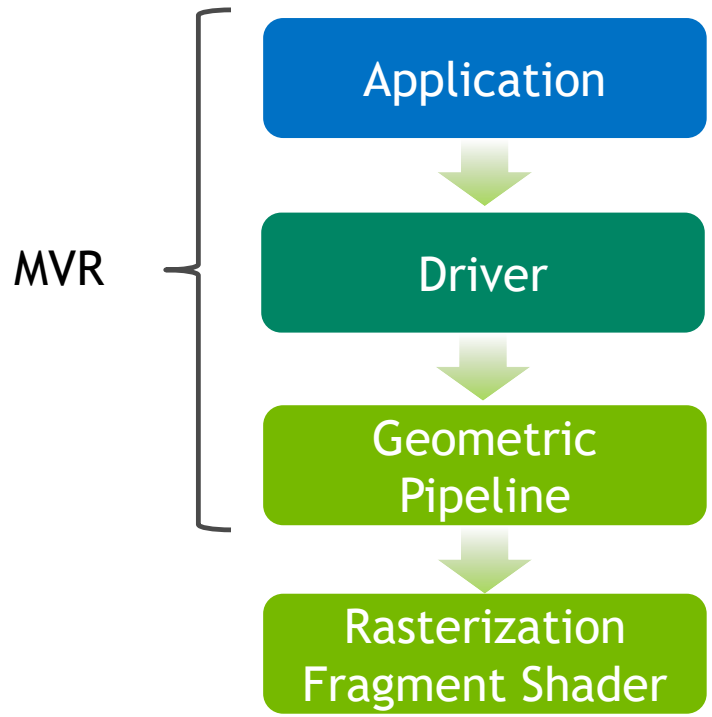
Only apply to OpenGL!

(Limitations come from GL_OVR_multiview/2)

No multisampling

No Geometry Shader

No Tessellation Shader



We‘re working on it!

MULTI-VIEW RENDERING

Recap

Reduces geometric load and CPU overhead

More flexible than SPS

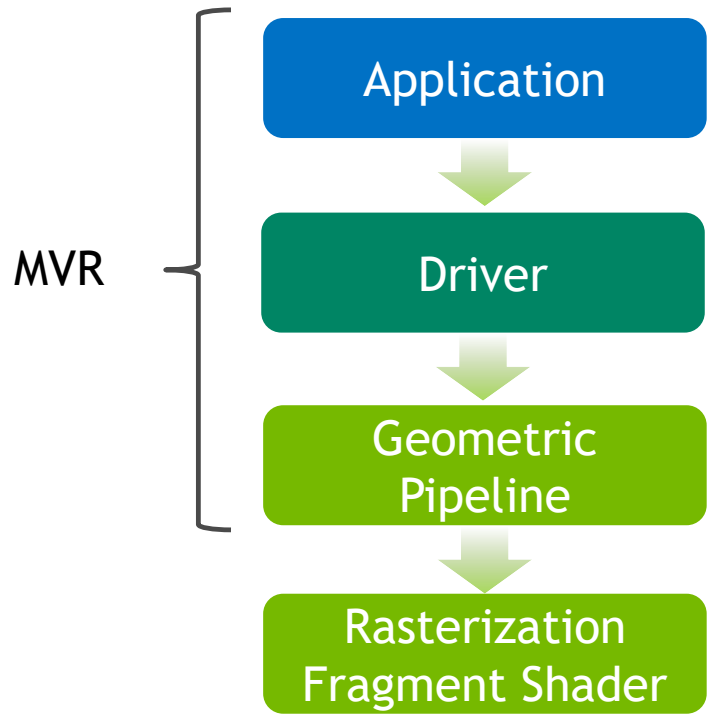
Software fallback for pre-Turing GPUs

Performance boost depends on number of view dependent attributes

DX11: NVAPI | DX12: via View Instancing

Vulkan: `VK_KHR_Multiview` (core in VK 1.1)

OpenGL: `GL_OVR_multiview & GL_OVR_multiview2`




VARIABLE RATE SHADING

VARIABLE RATE SHADING

Motivation

Due to the lens distortion the image is warped before sending it to the HMD.
Good opportunity to save unnecessary rendering work.

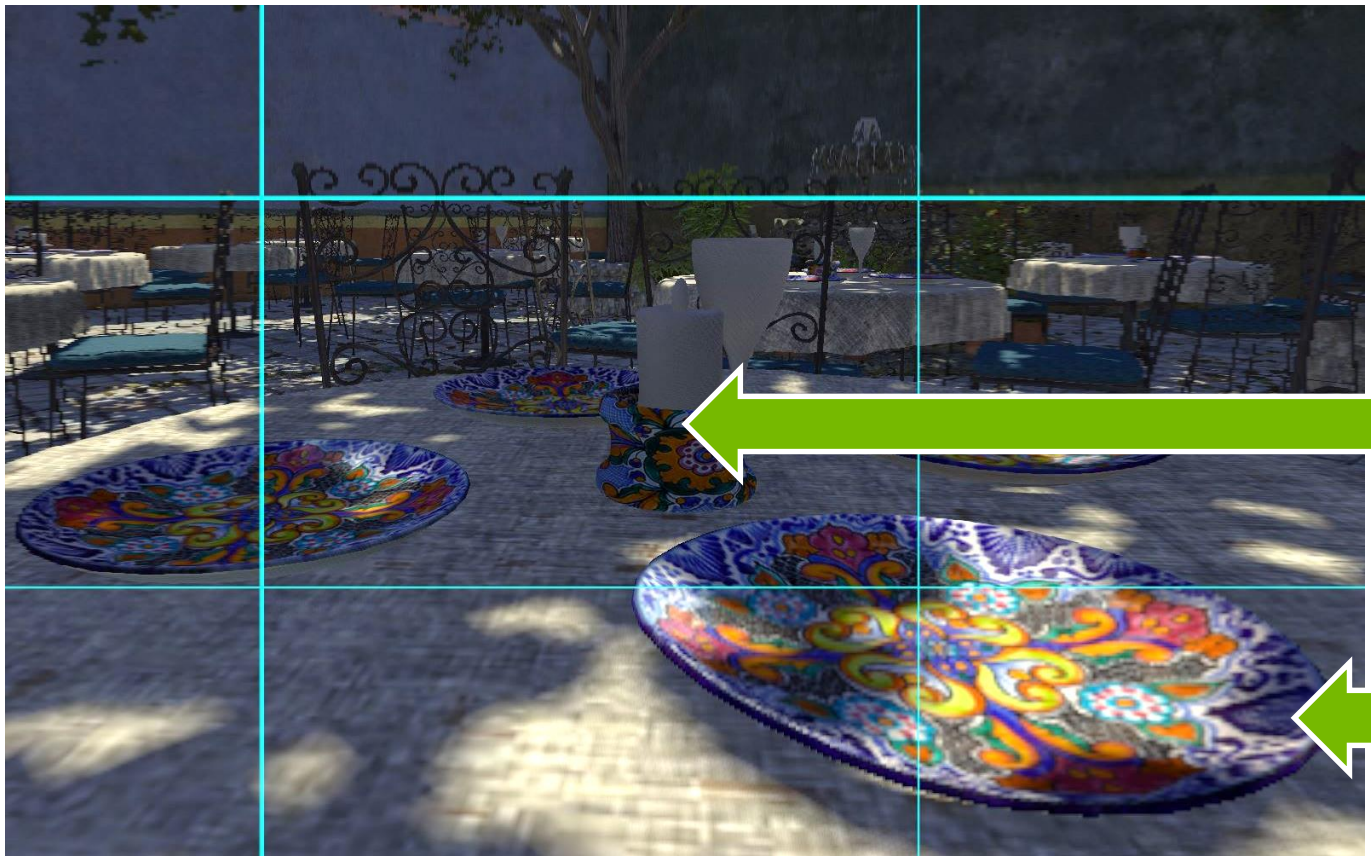
High Resolution

Medium Resolution

Low Resolution

RECAP: MAXWELL

Multi-Resolution Shading



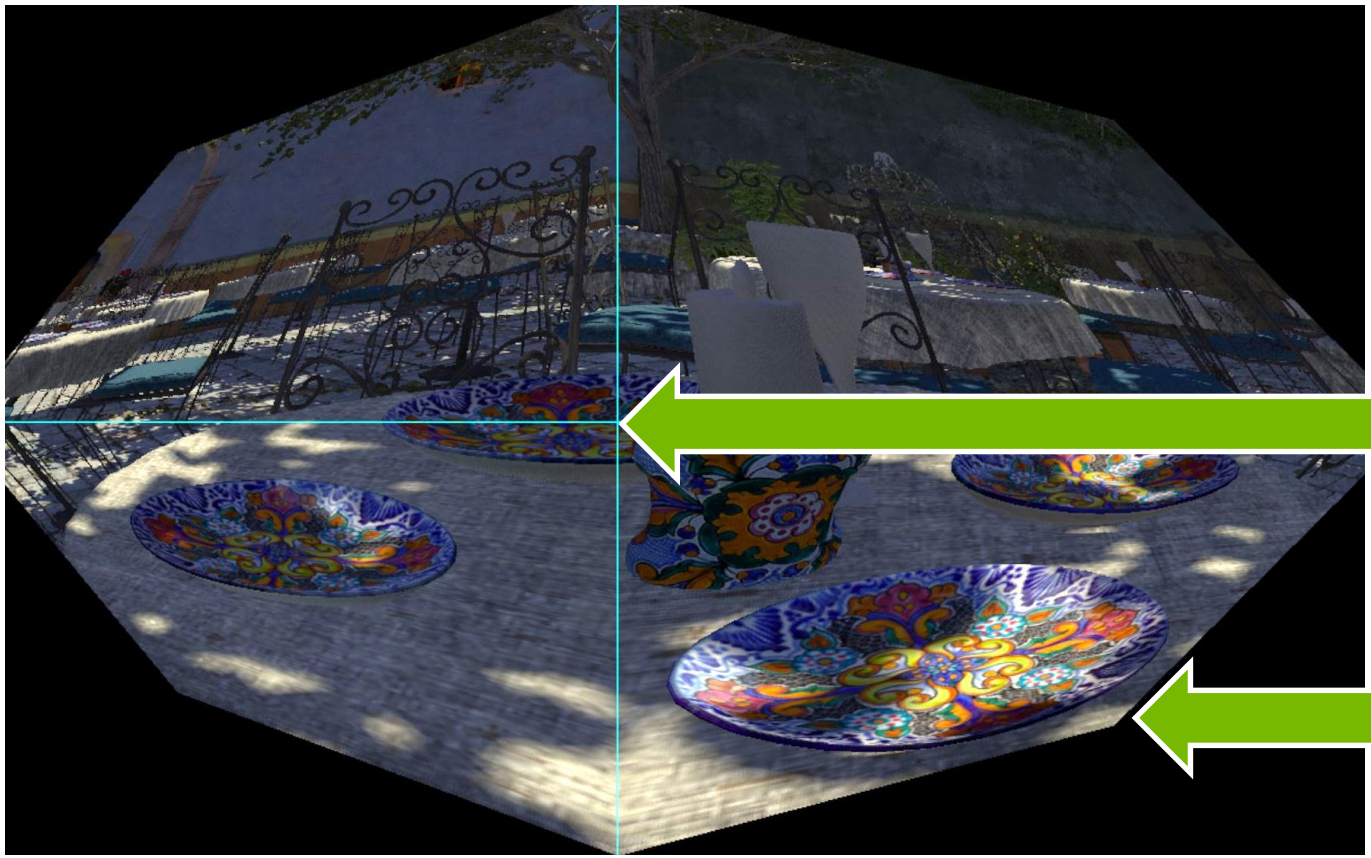
9 Viewports
9 areas in which the resolution is constant

High Resolution

Low Resolution

RECAP: PASCAL

Lens Matched Shading



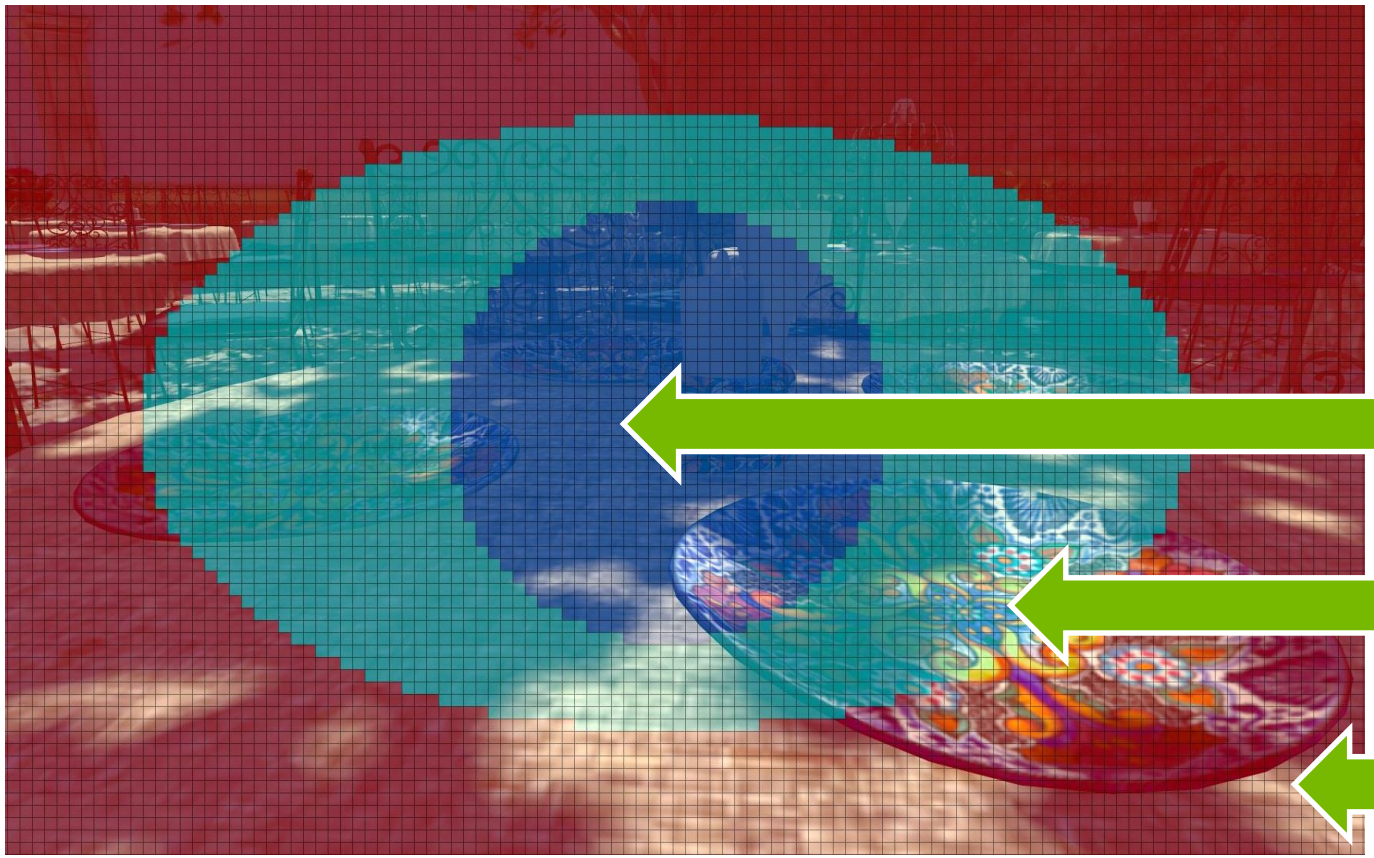
4 Viewports
4 areas in which the resolution gets reduced towards the corners

High Resolution

Low Resolution

NEW: TURING

Variable Rate Shading



1 Viewport
Many small areas in which the shading rate is constant

High Resolution

Medium Resolution

Low Resolution

COMPARING MRS, LMS, VRS

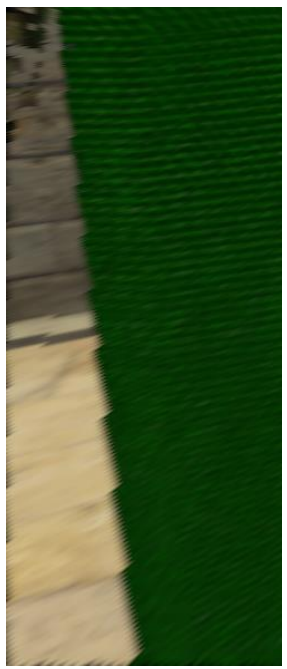
From our DX11 VRWorks Samples

COMPARING MRS, LMS, VRS

From our DX11 VRWorks Samples

MRS
Density: 0.25

LMS
Coefficient: 2.0



VRS
Shading Rate: 4x4

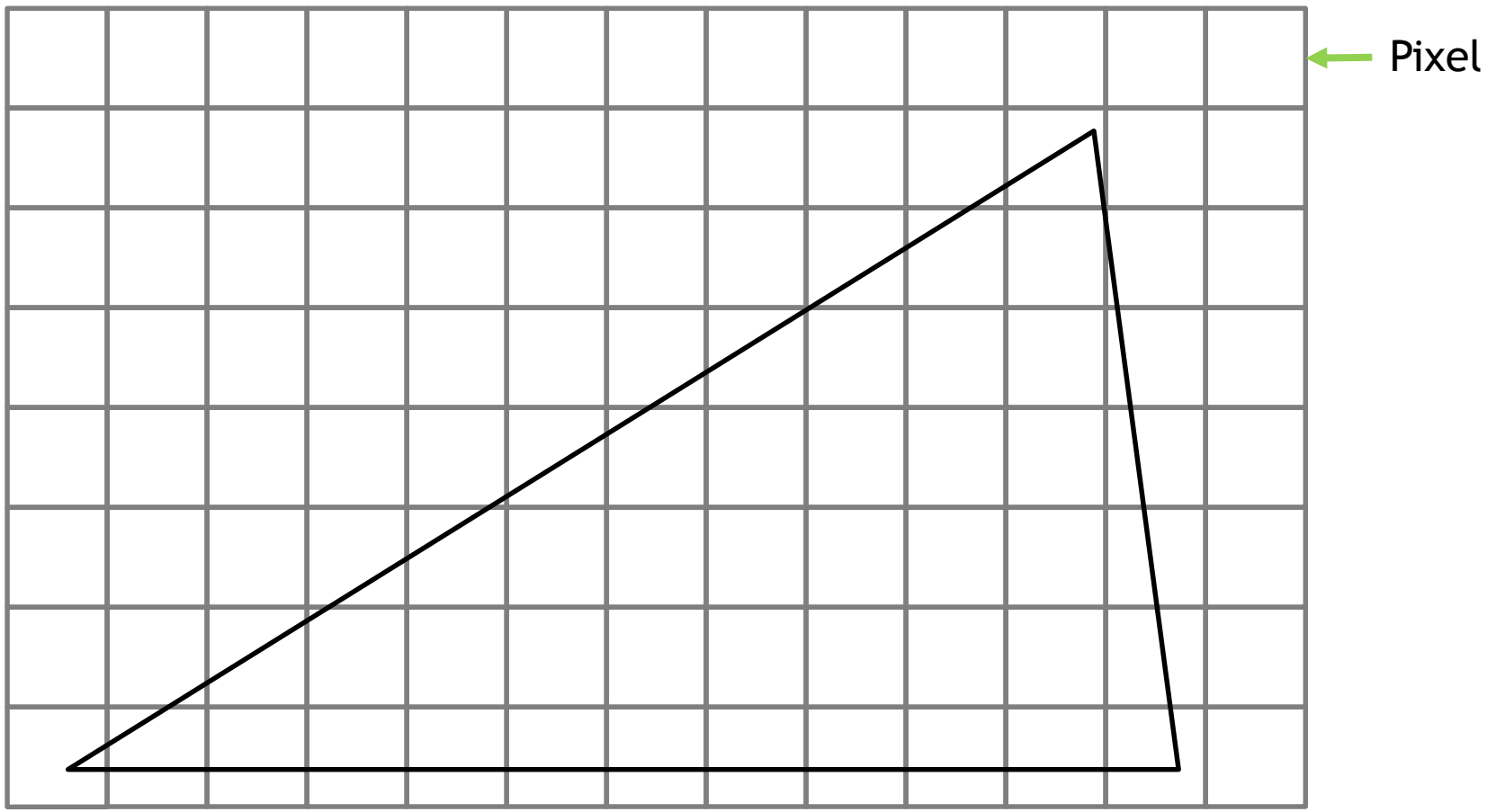
MRS

LMS

VRS

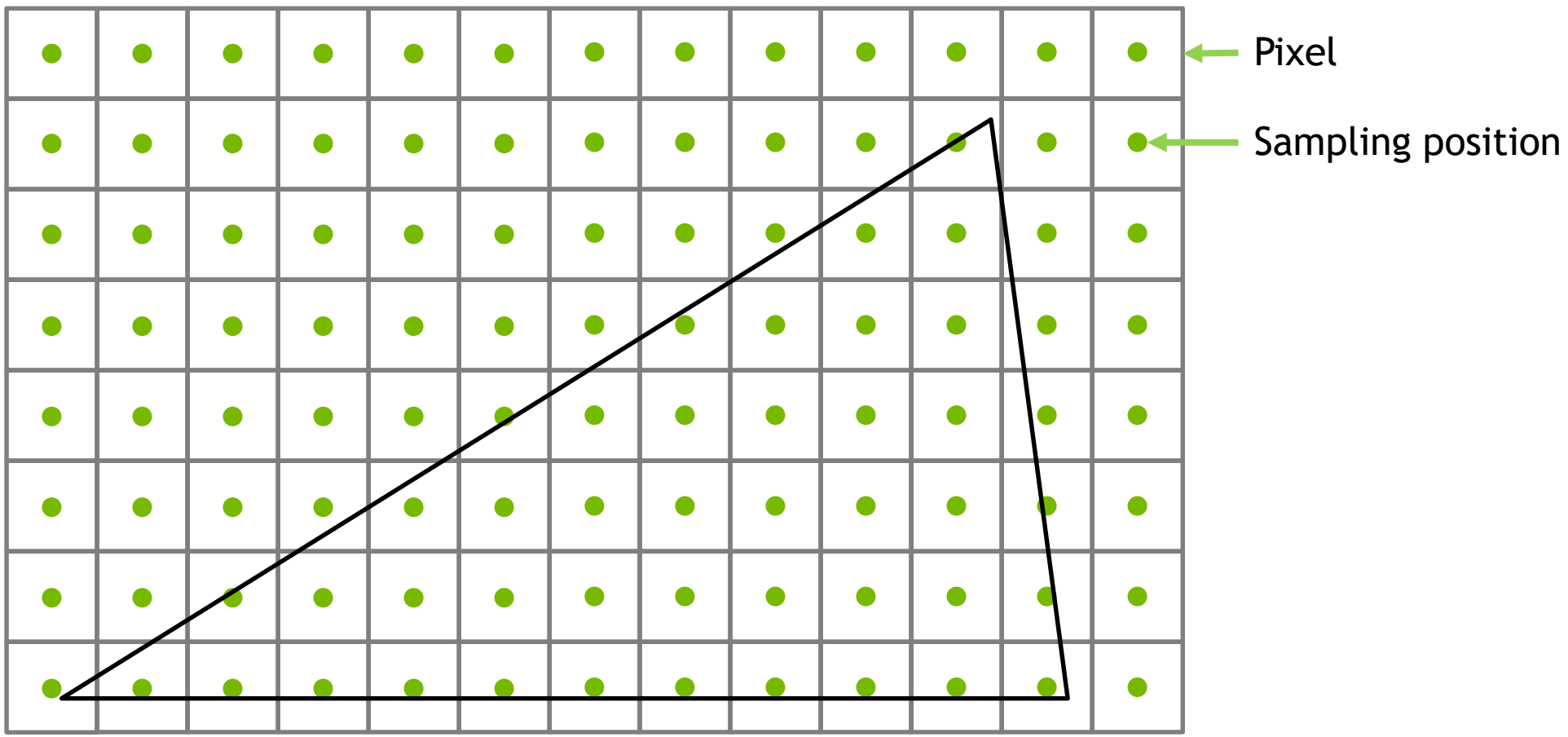
VARIABLE RATE SHADING

Rasterization



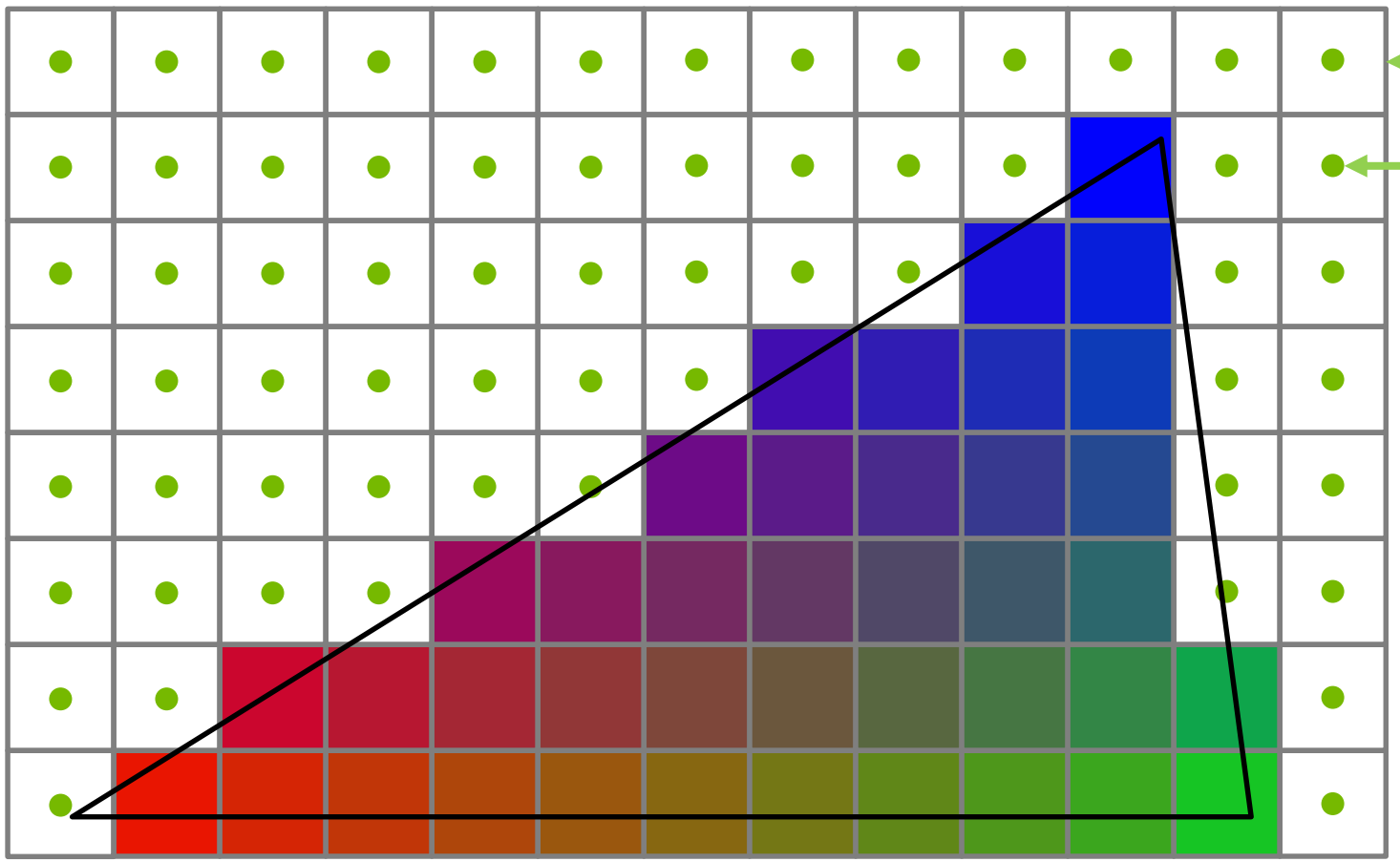
VARIABLE RATE SHADING

Rasterization



VARIABLE RATE SHADING

Rasterization



Pixel

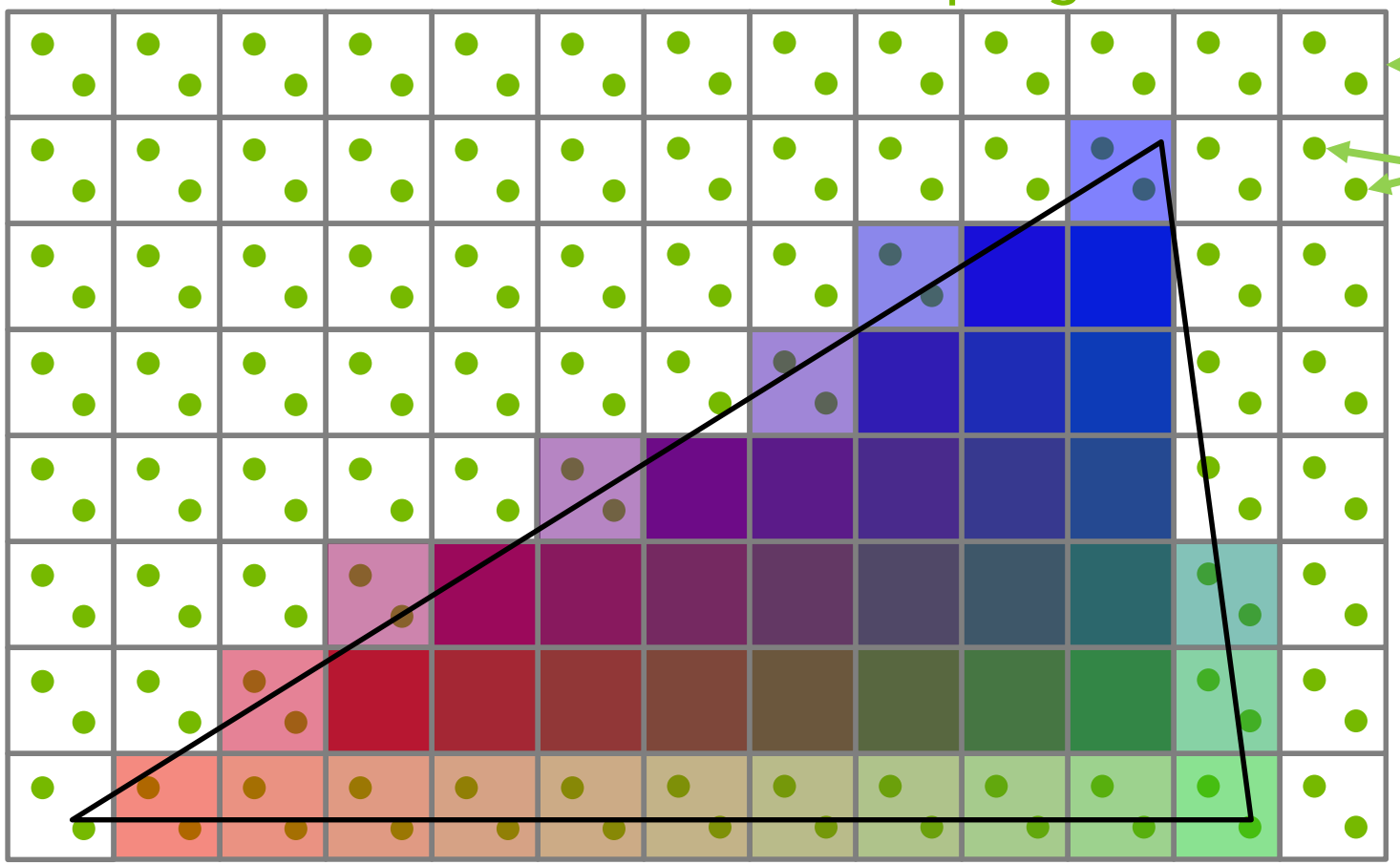
Sampling position

Pixels: 40
Samples covered: 40
F.Shader invocations*: 40

* (not counting helper threads)

VARIABLE RATE SHADING

Multi Sampling Rasterization



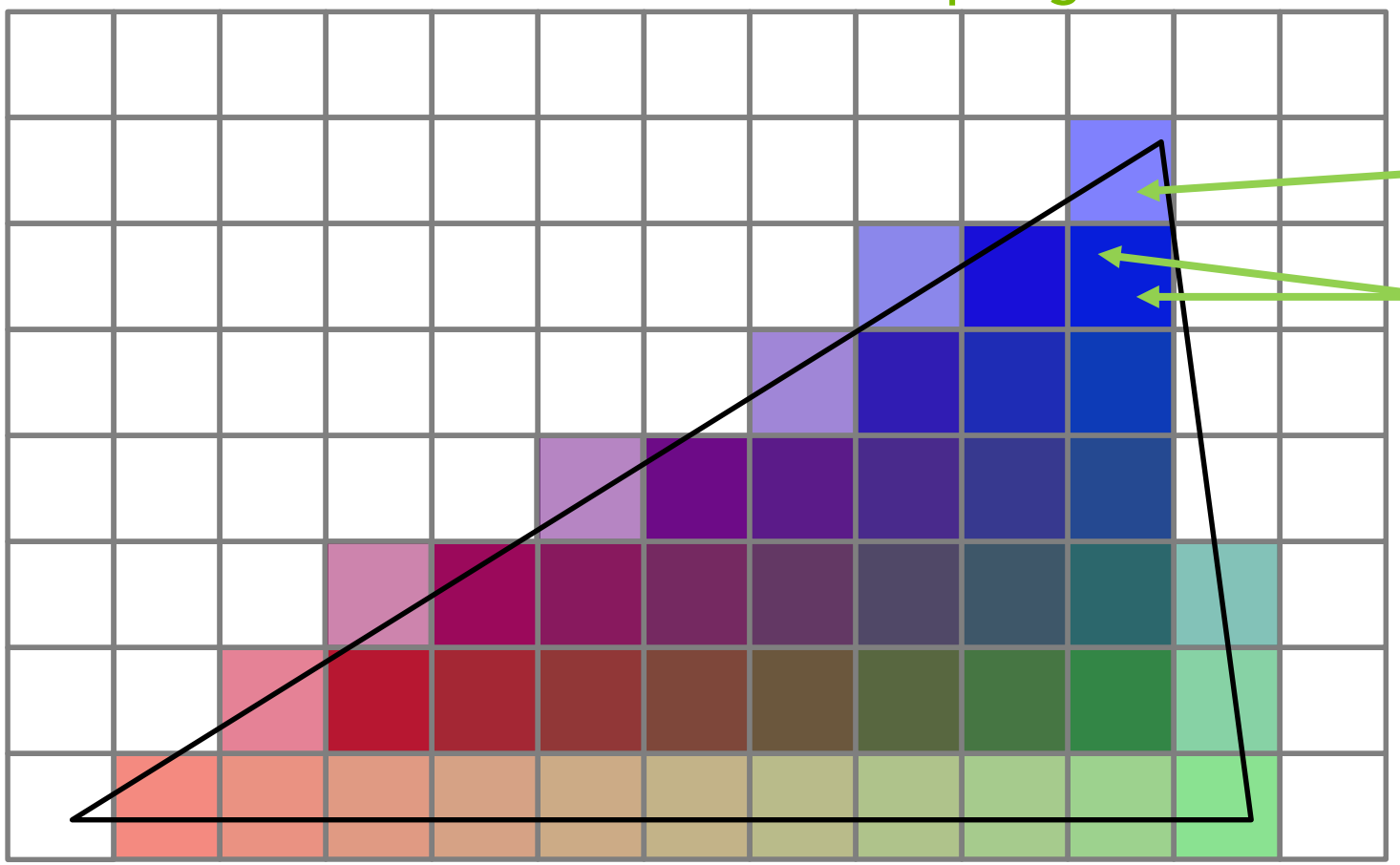
Pixel

Sampling position

Pixels:	44
Samples covered:	69
F.Shader invocations:	44

VARIABLE RATE SHADING

Multi Sampling Rasterization

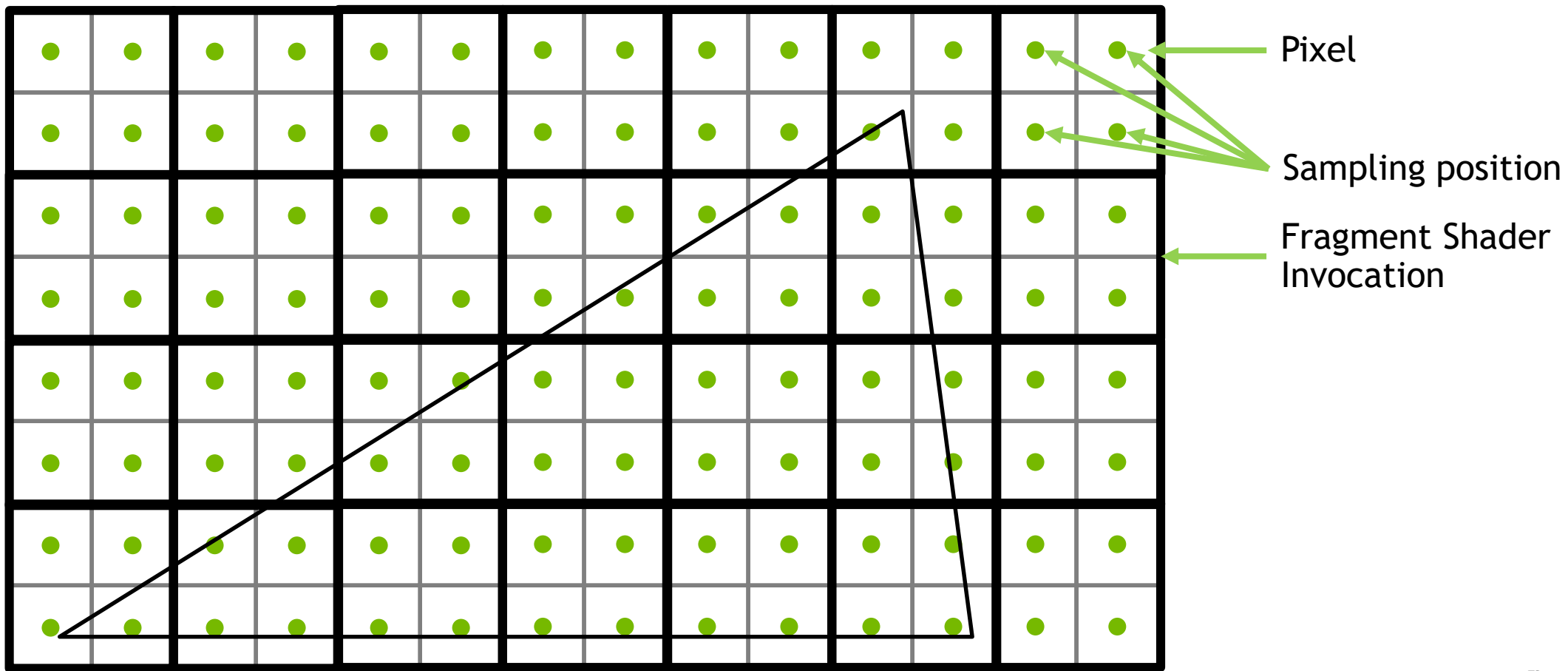


Shading result stored for **one** sampling position

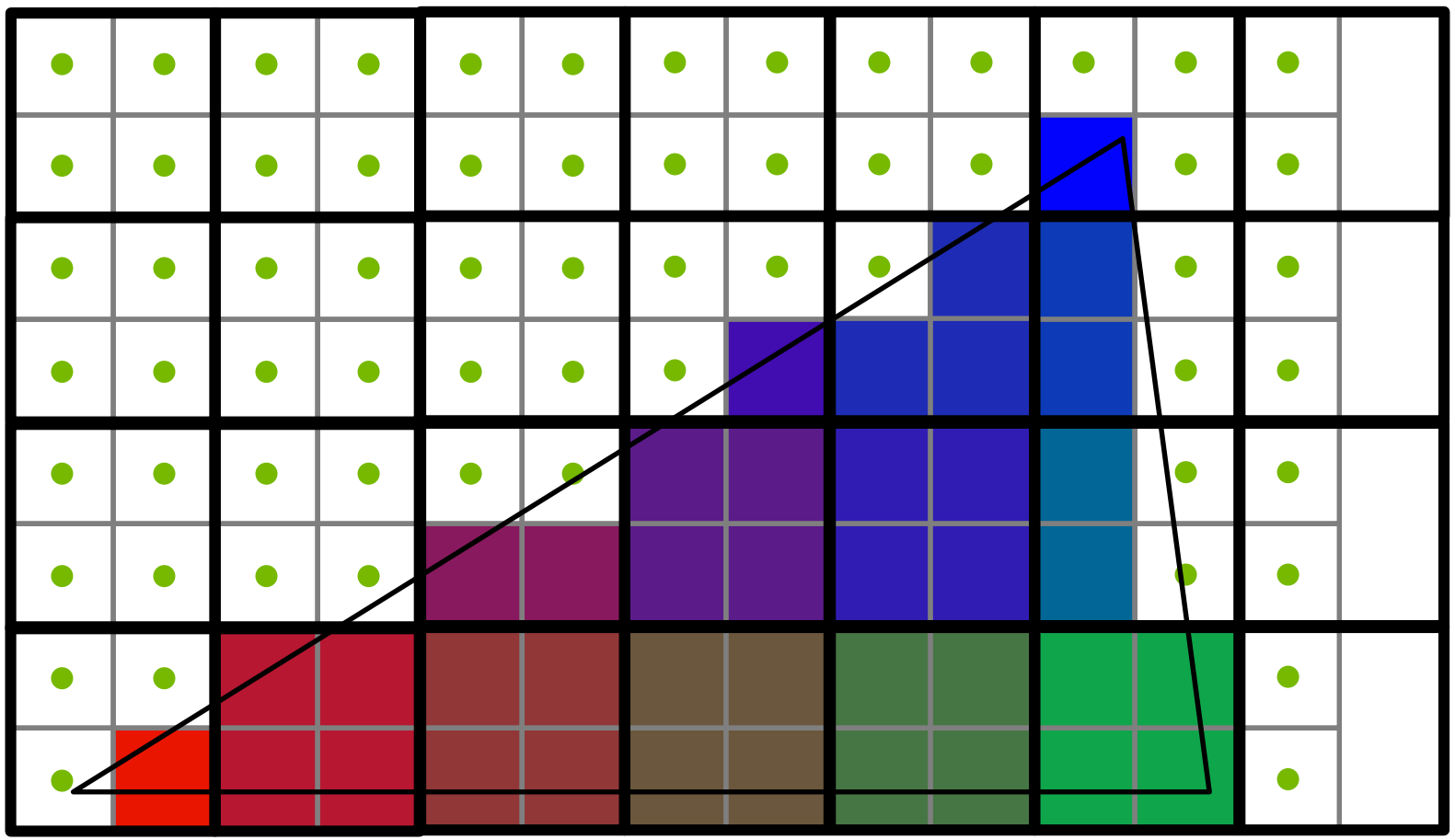
Shading result stored for **two** sampling position

Pixels:	44
Samples covered:	69
F.Shader invocations:	44

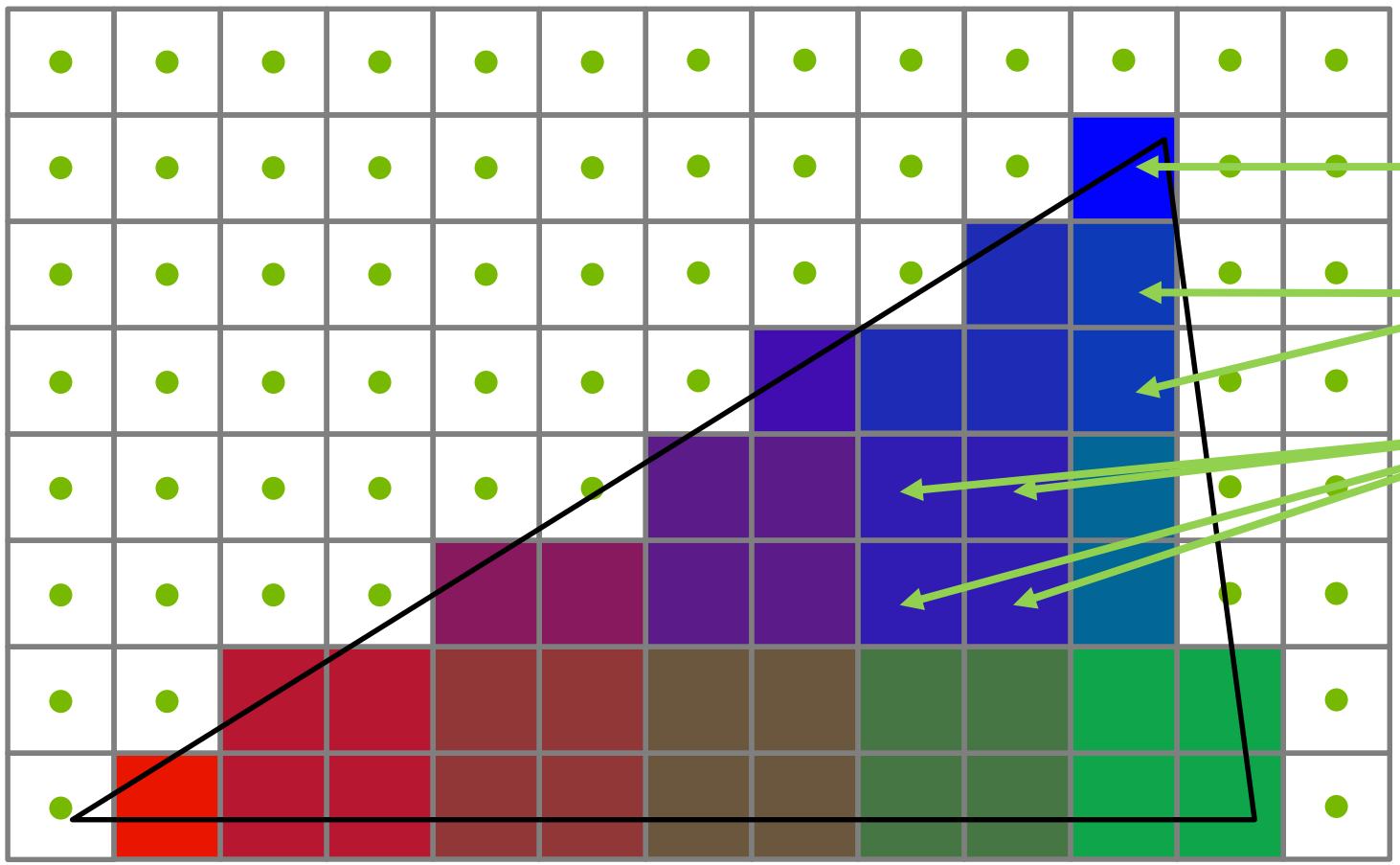
VARIABLE RATE SHADING



VARIABLE RATE SHADING



VARIABLE RATE SHADING

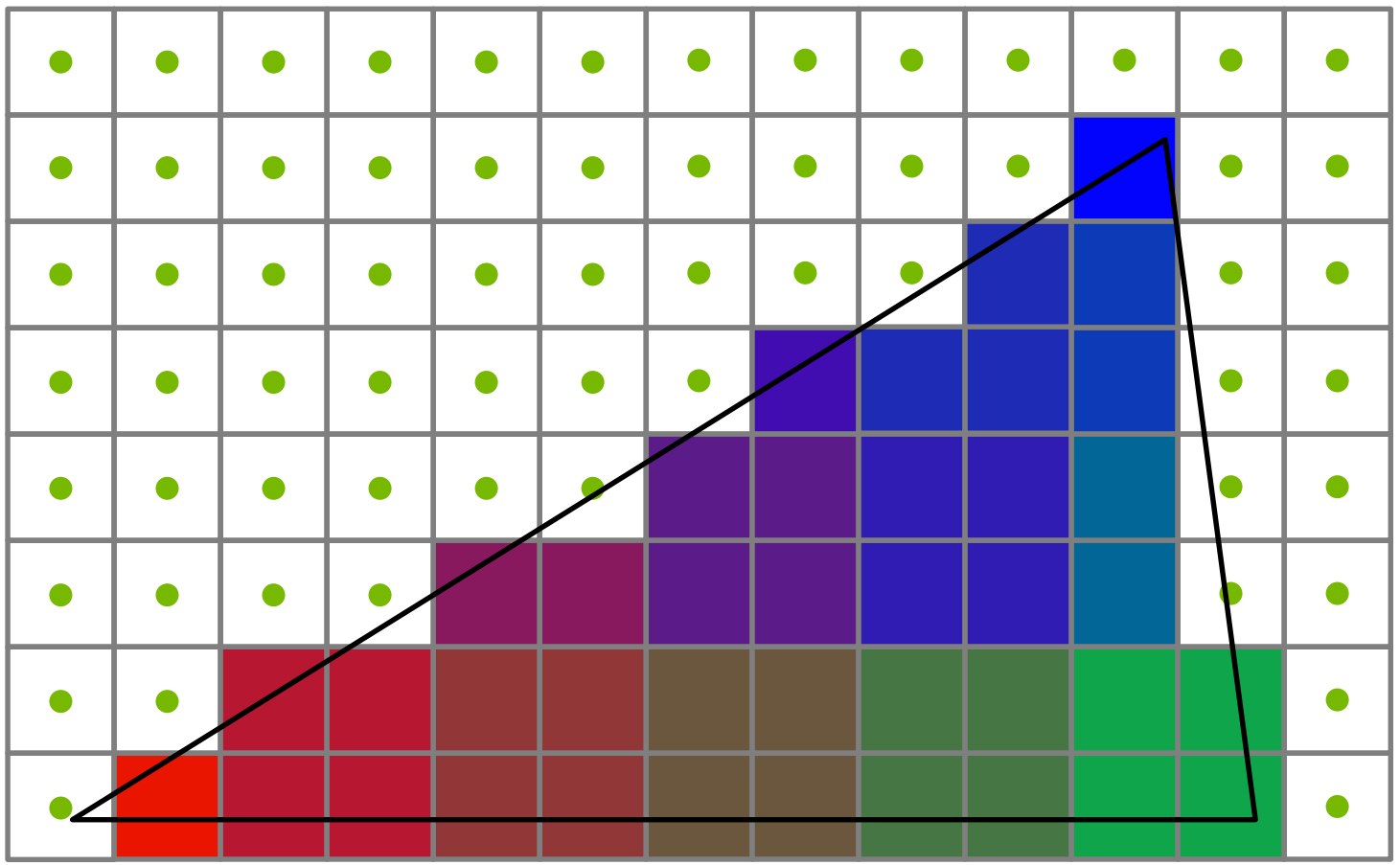


Shading result stored for one pixel

Shading result stored for two pixels

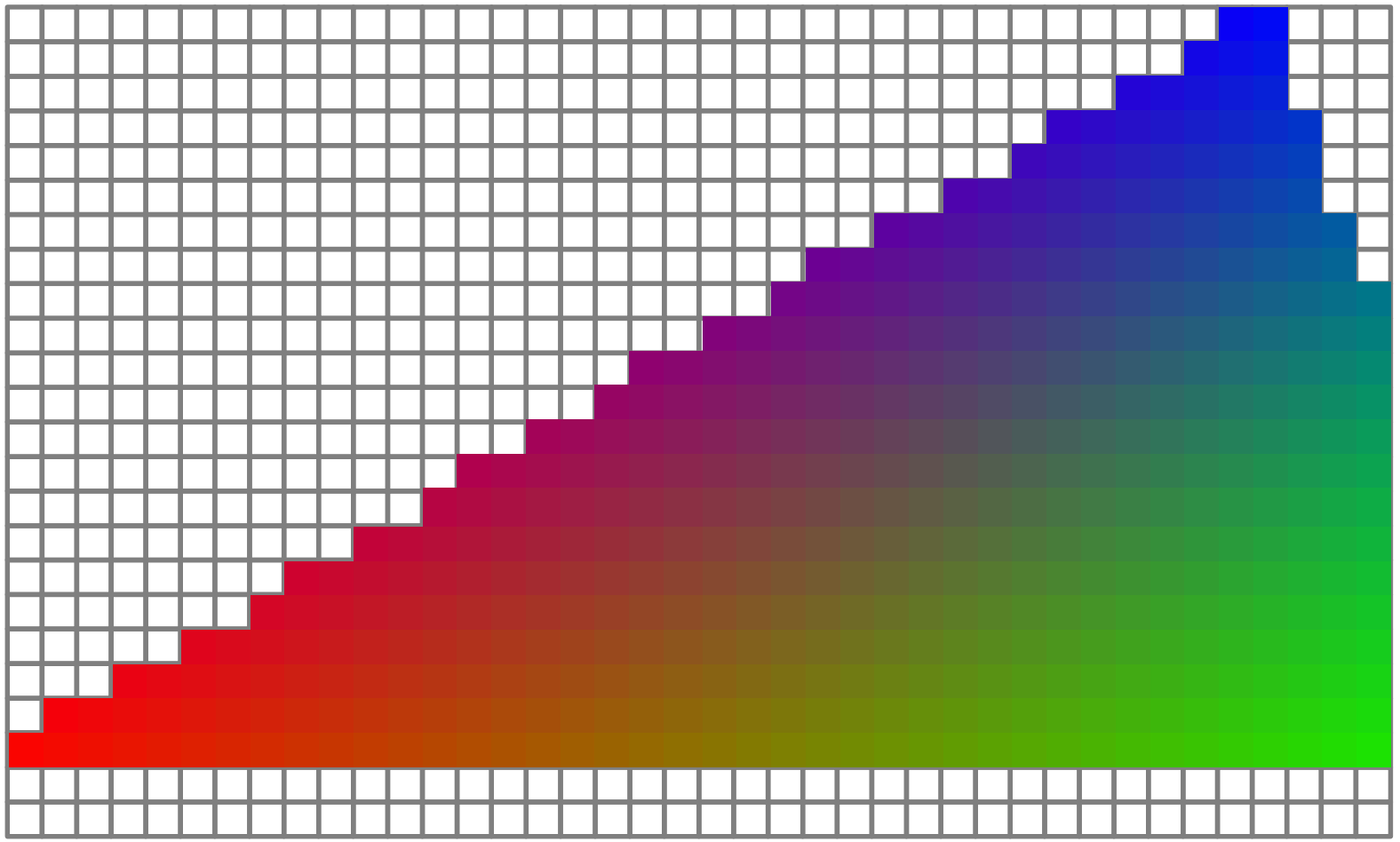
Shading result stored for four pixels

VARIABLE RATE SHADING



Pixels: 40
Samples covered: 40
F.Shader invocations: 14

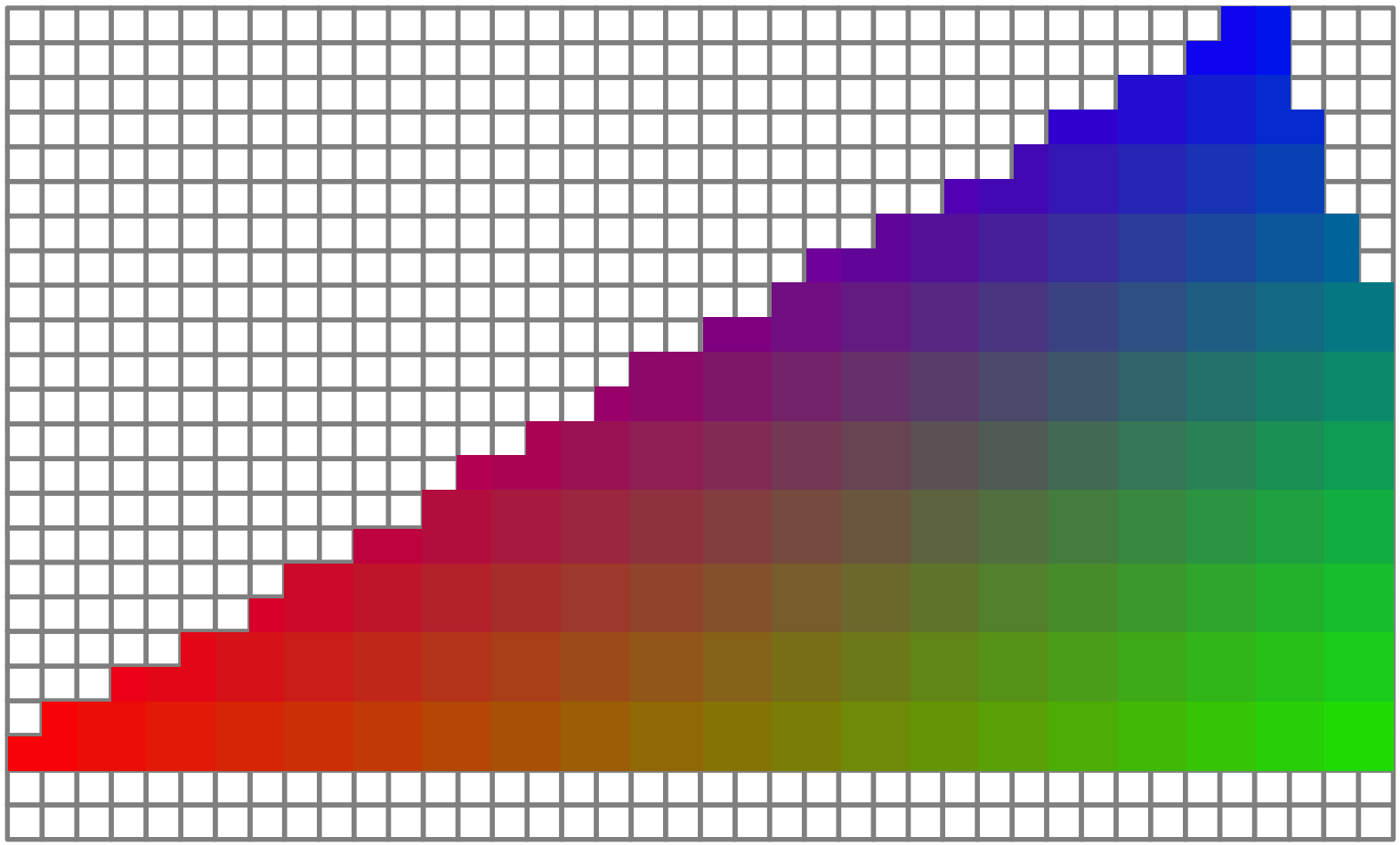
VARIABLE RATE SHADING



1x1 Shading Rate

Pixels:	477
Samples covered:	477
F.Shader invocations:	477

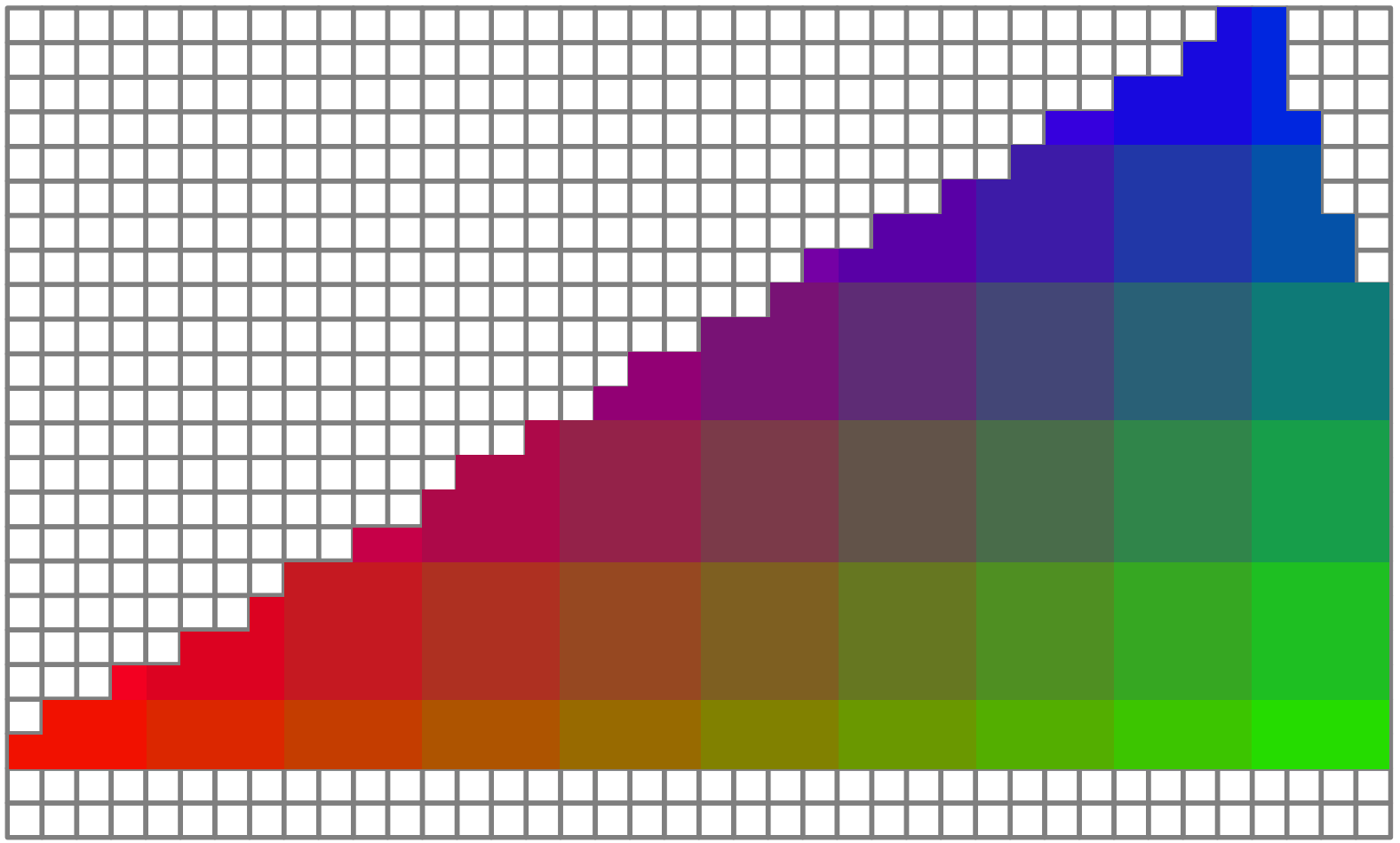
VARIABLE RATE SHADING



2x2 Shading Rate

Pixels:	477
Samples covered:	477
F.Shader invocations:	128

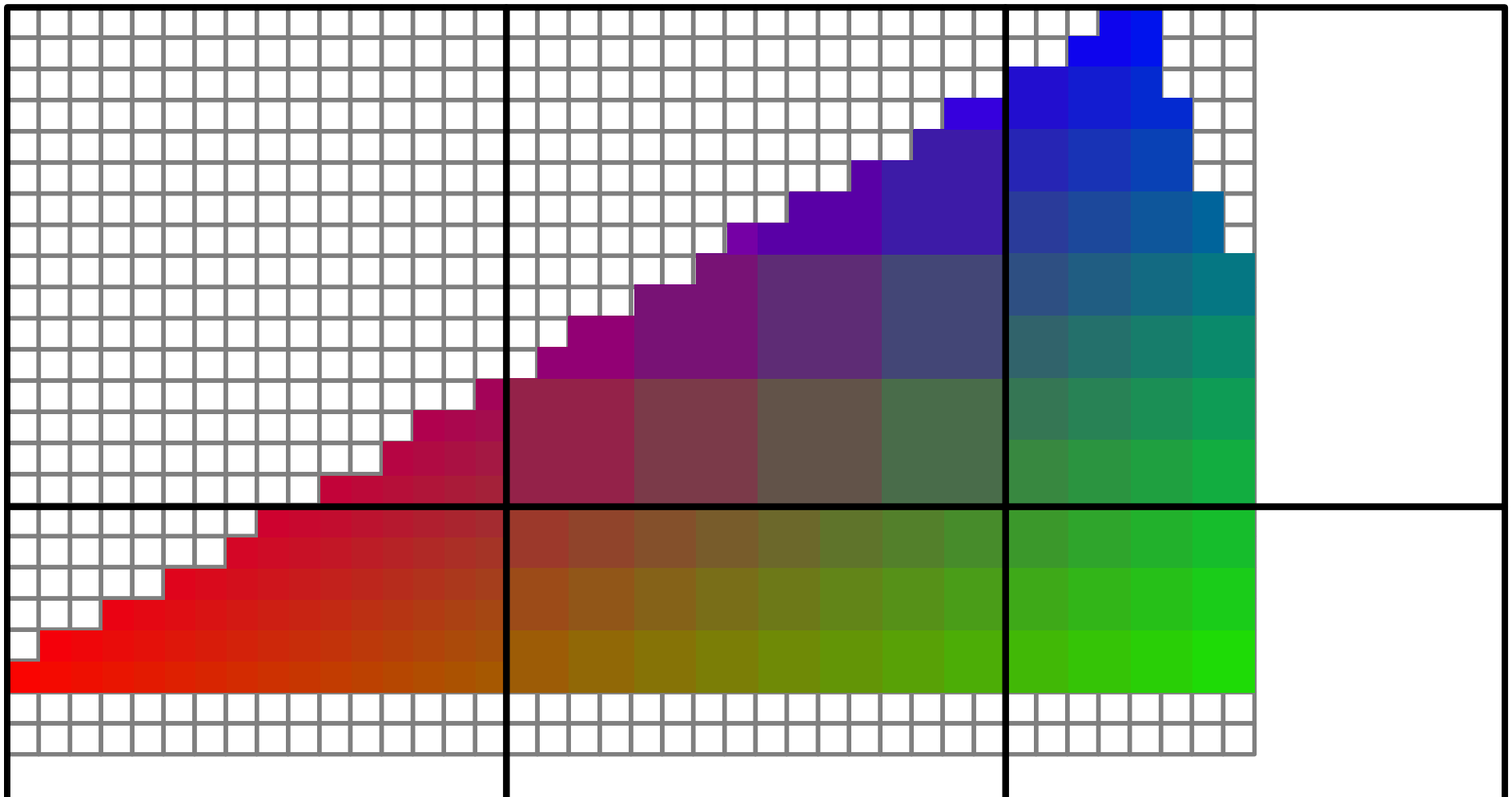
VARIABLE RATE SHADING



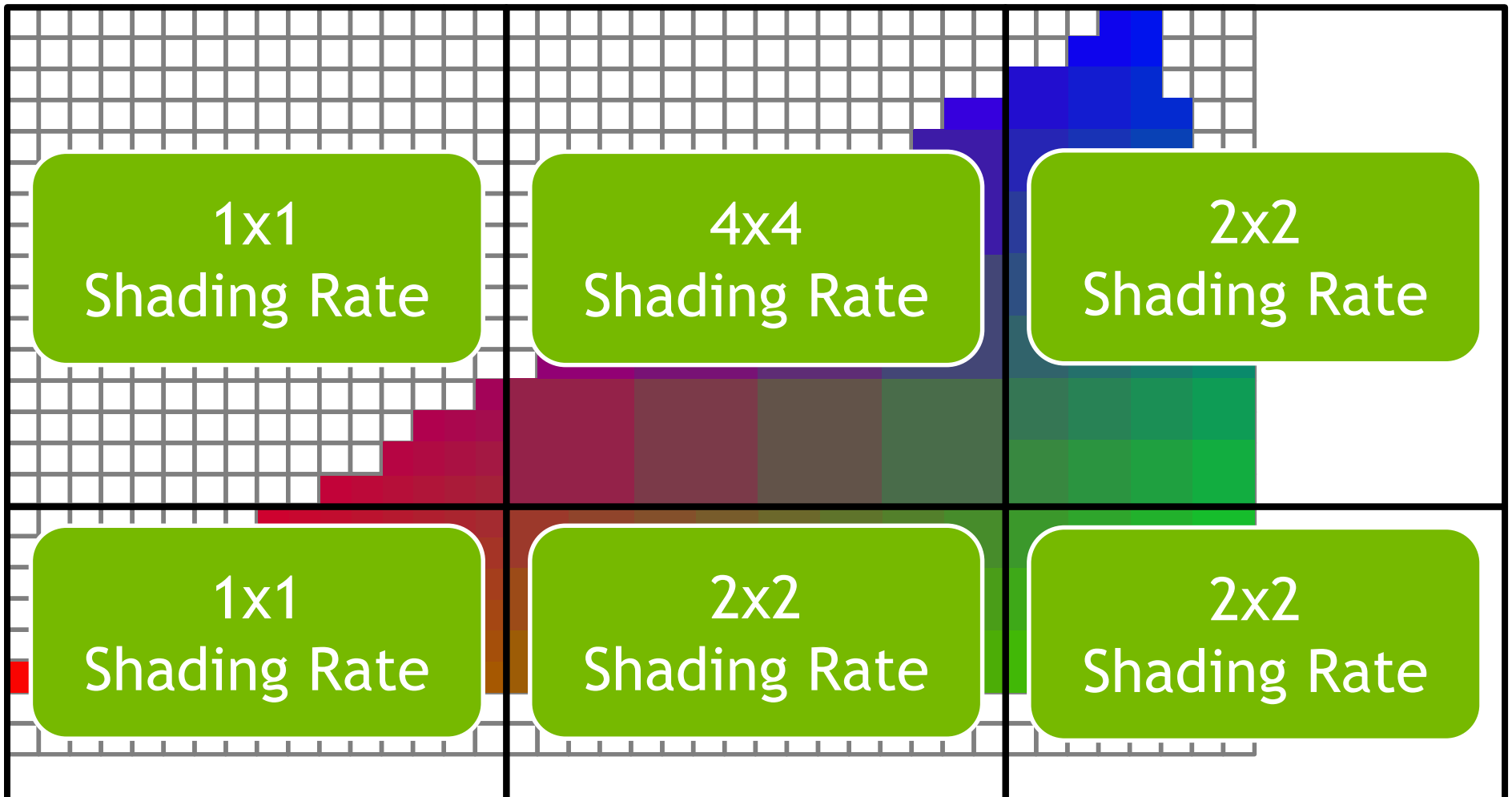
4x4 Shading Rate

Pixels:	477
Samples covered:	477
F.Shader invocations:	42

VARIABLE RATE SHADING

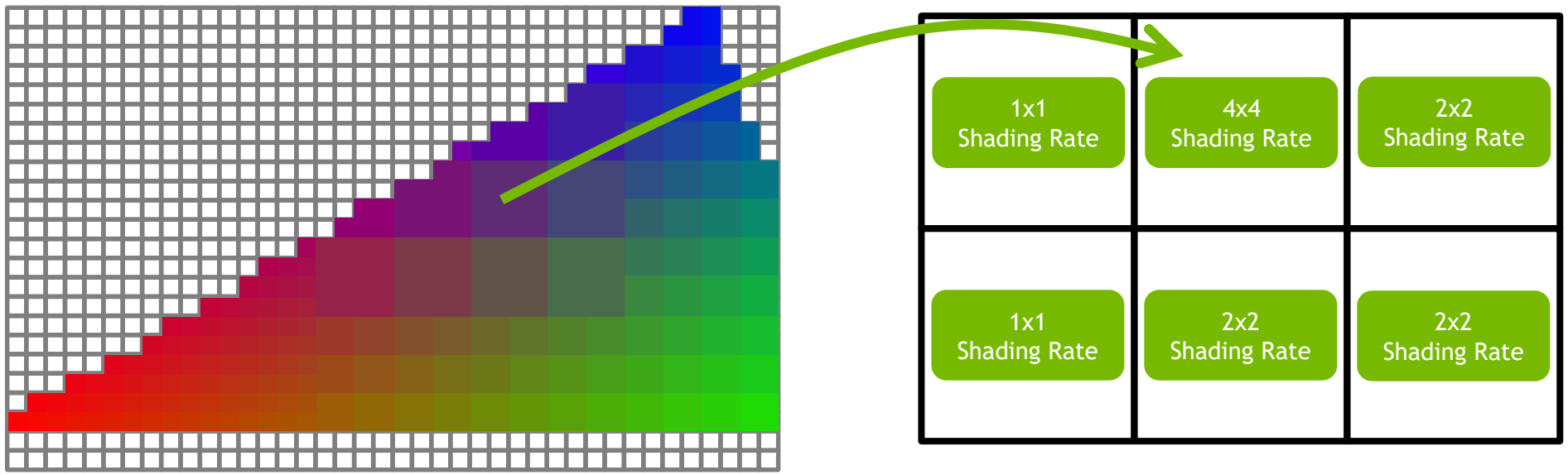


VARIABLE RATE SHADING



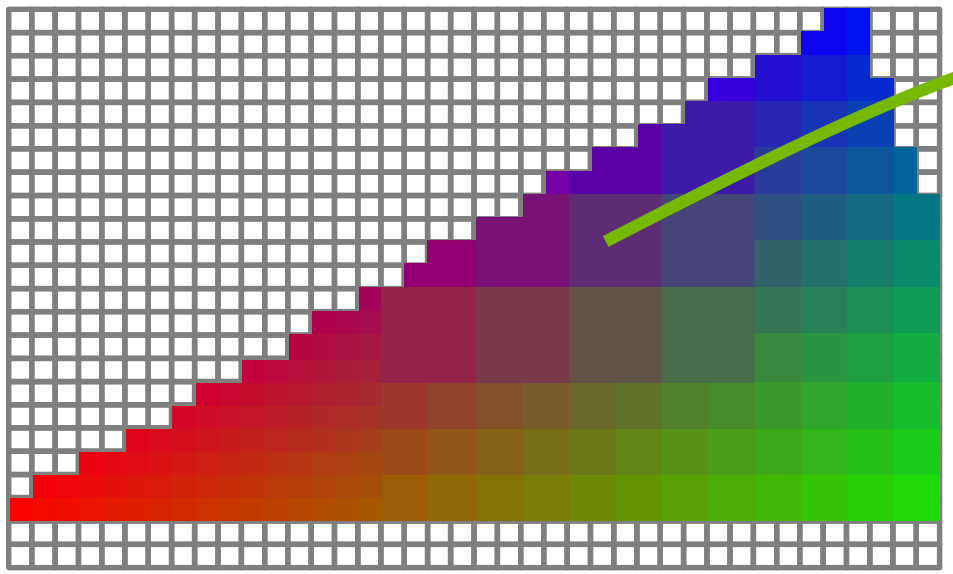
VARIABLE RATE SHADING

Shading Rate Lookup

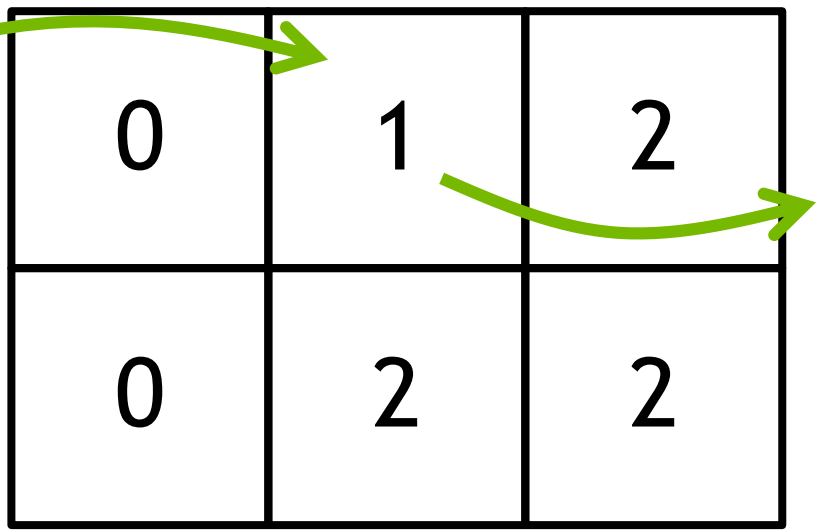


VARIABLE RATE SHADING

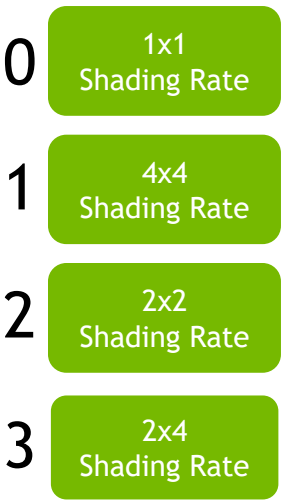
Shading Rate Lookup



Framebuffer



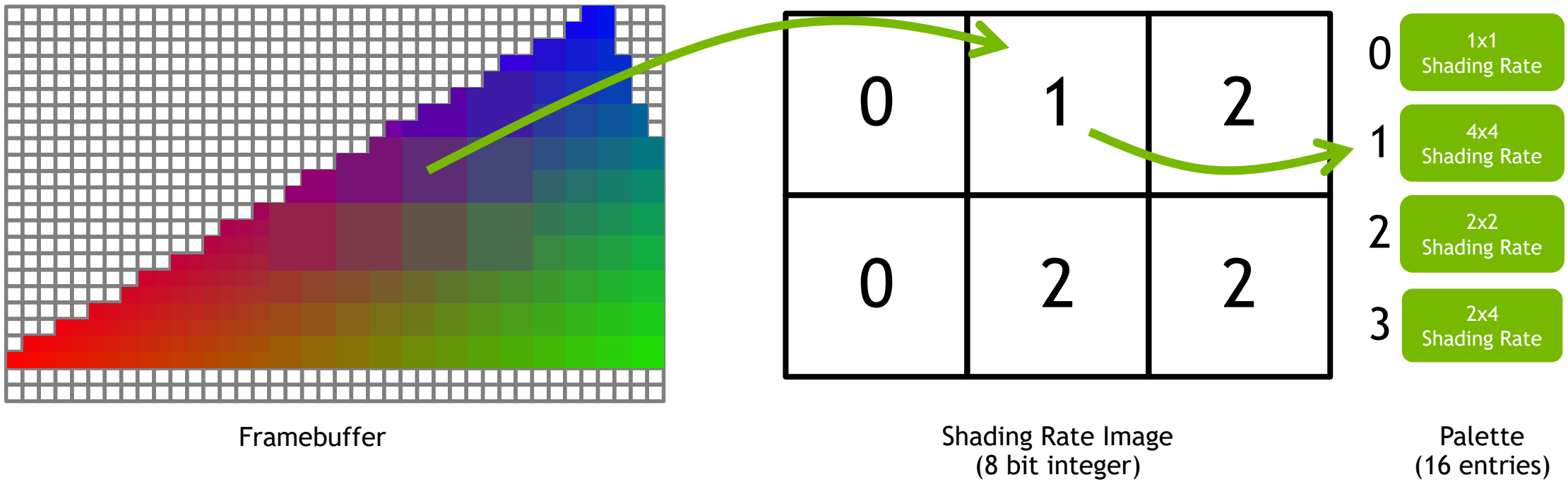
Shading Rate Image



Palette

VARIABLE RATE SHADING

Shading Rate Lookup



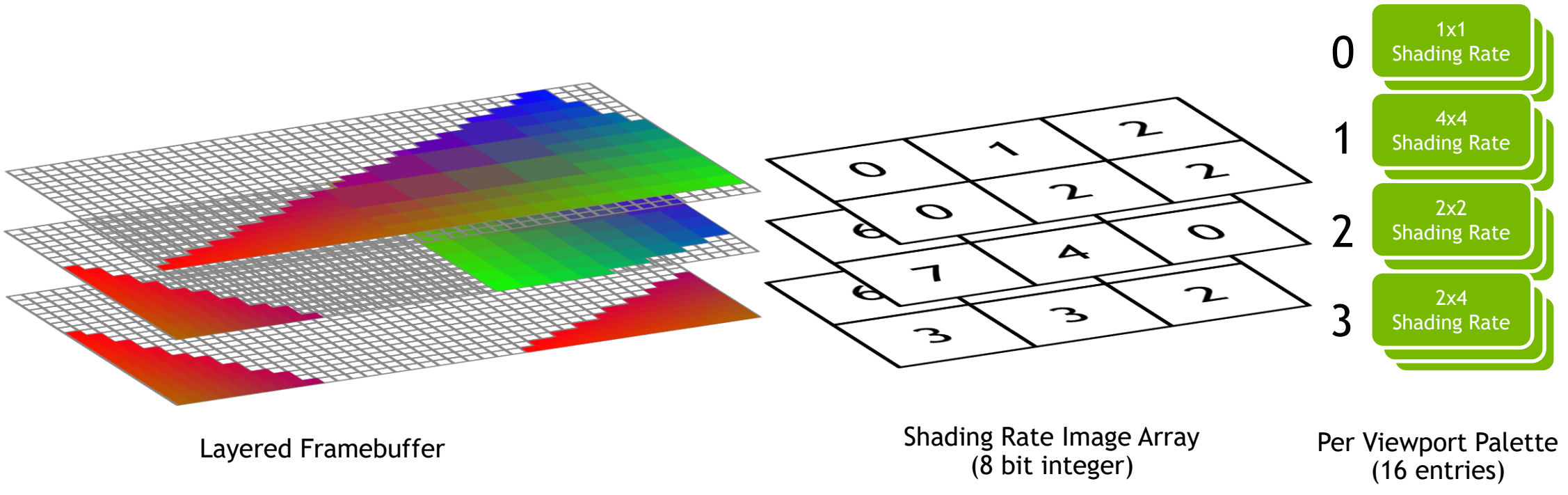
Framebuffer

Shading Rate Image
(8 bit integer)

Palette
(16 entries)

VARIABLE RATE SHADING

Shading Rate Lookup



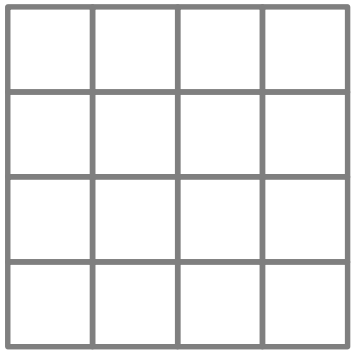
Layered Framebuffer

Shading Rate Image Array
(8 bit integer)

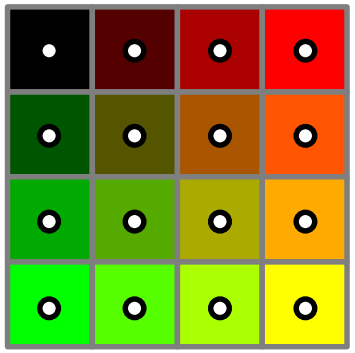
Per Viewport Palette
(16 entries)

VARIABLE RATE SHADING

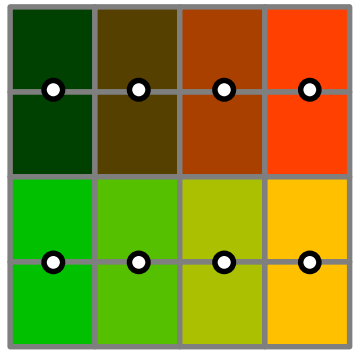
Shading Modes: GL_SHADING_RATE_



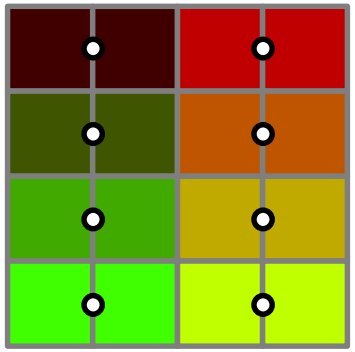
NO_INVOCATIONS_NV



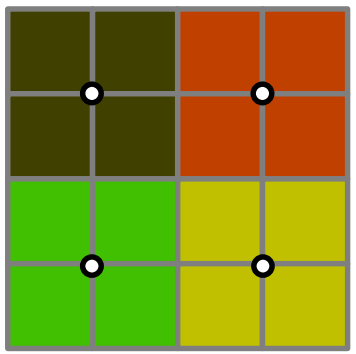
1_INVOCATION_PER_PIXEL_NV



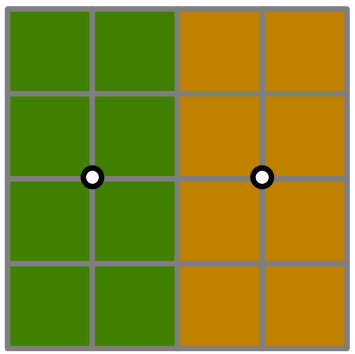
1_INVOCATION_PER_1X2_PIXELS_NV



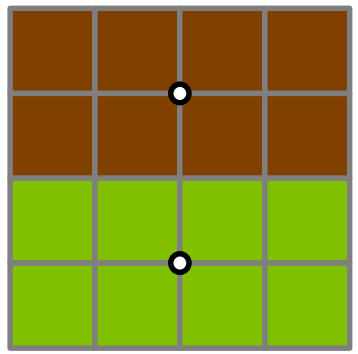
1_INVOCATION_PER_2X1_PIXELS_NV



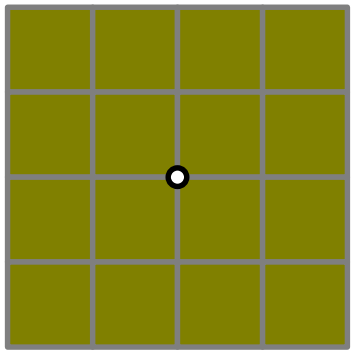
1_INVOCATION_PER_2X2_PIXELS_NV



1_INVOCATION_PER_2X4_PIXELS_NV



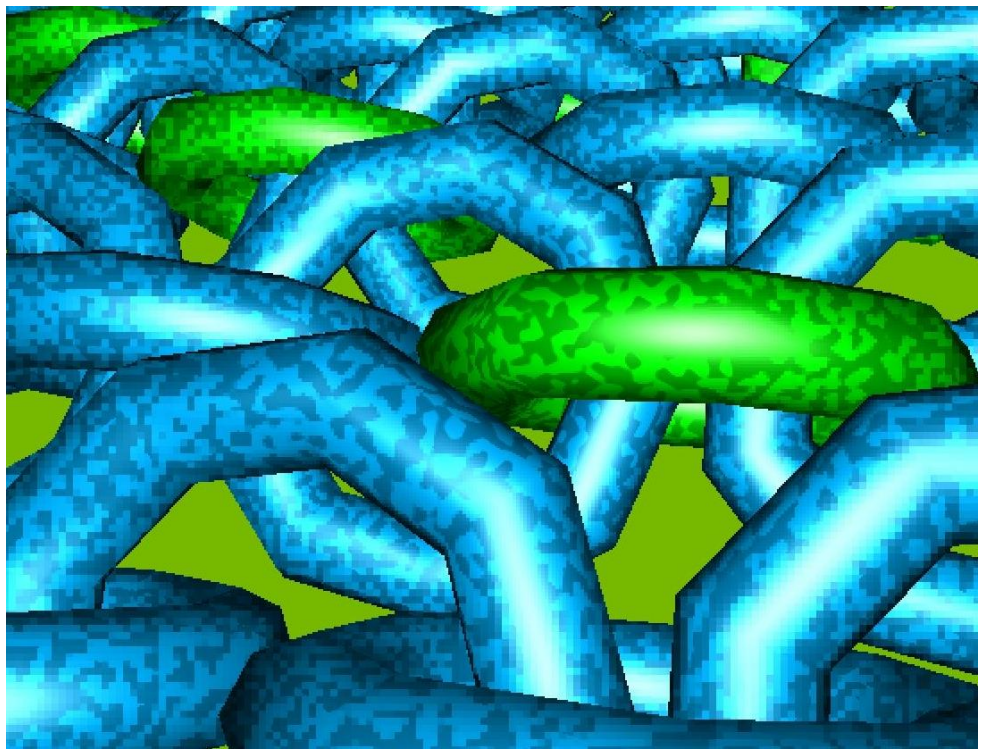
1_INVOCATION_PER_4X2_PIXELS_NV



1_INVOCATION_PER_4X4_PIXELS_NV

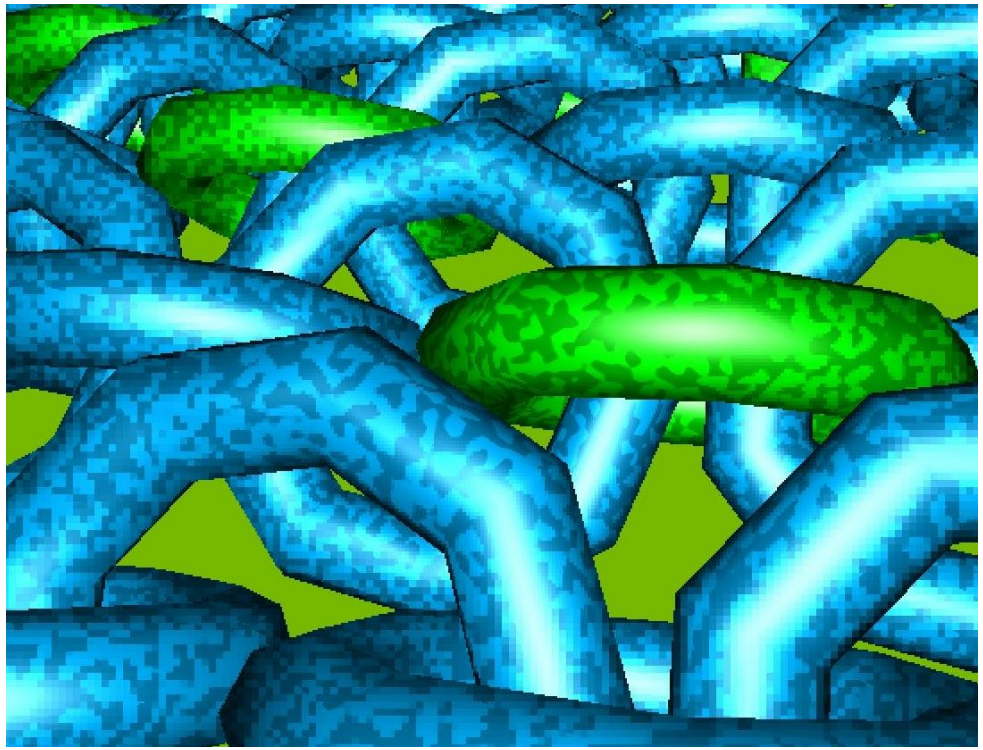
VARIABLE RATE SHADING

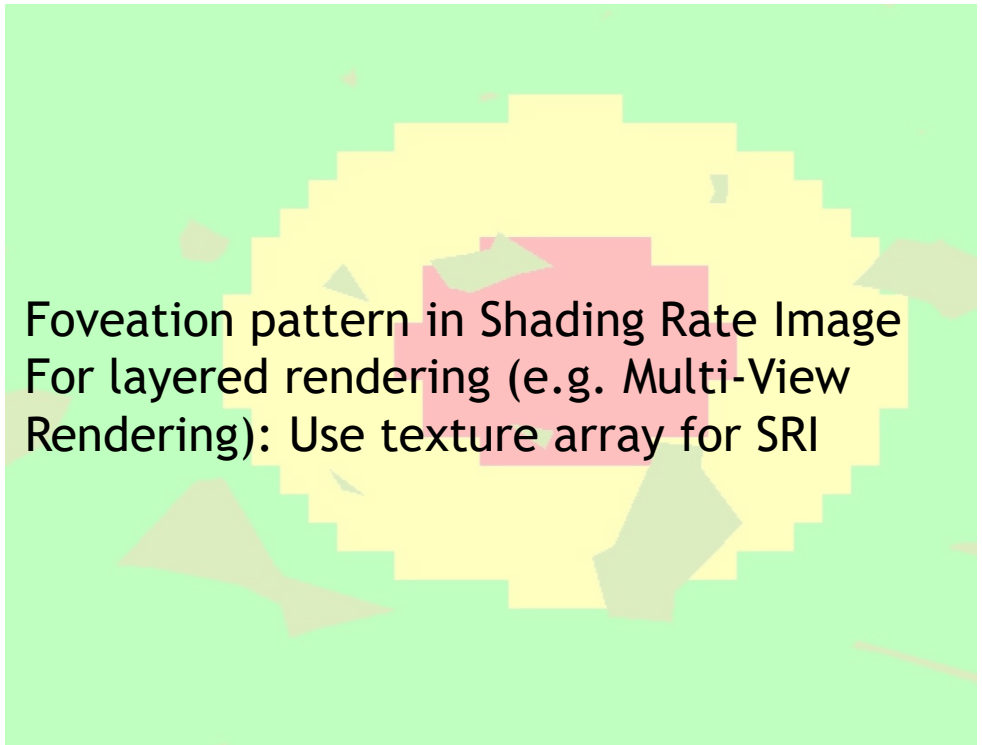
Foveated Rendering



VARIABLE RATE SHADING

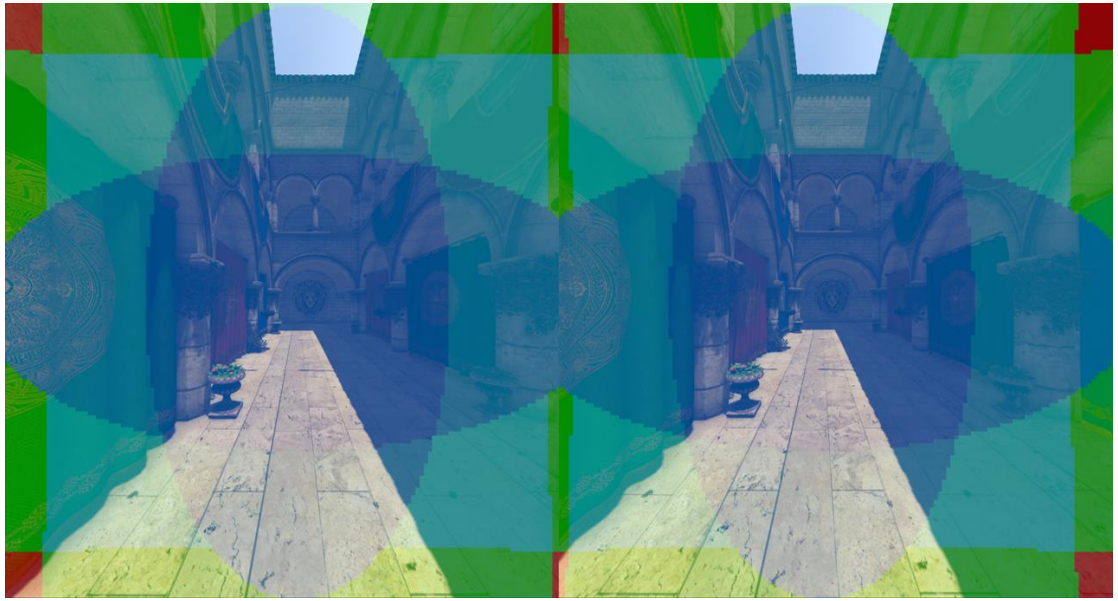
Foveated Rendering



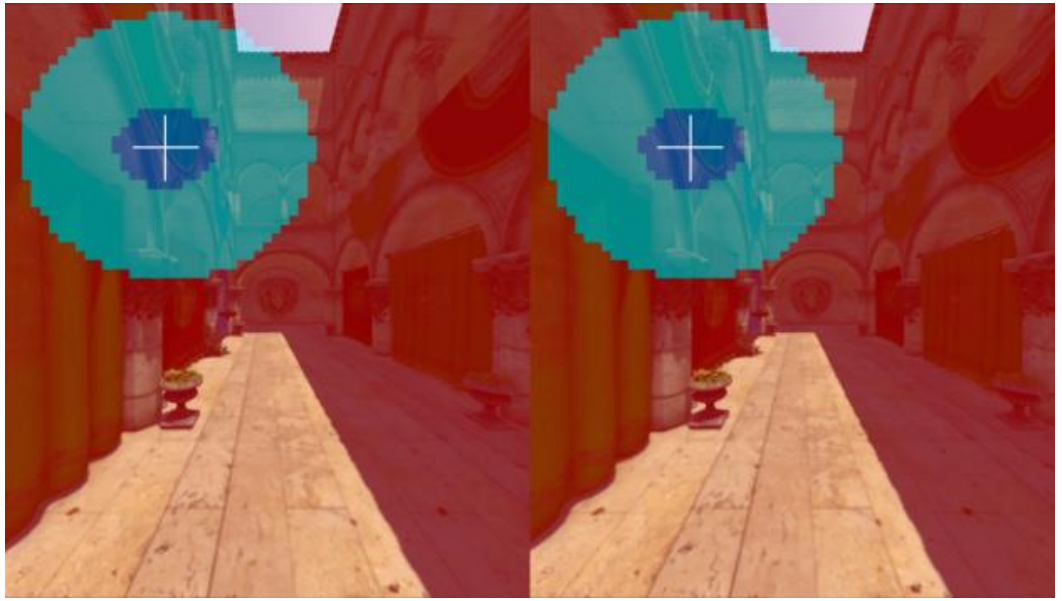


VARIABLE RATE SHADING

Foveated Rendering



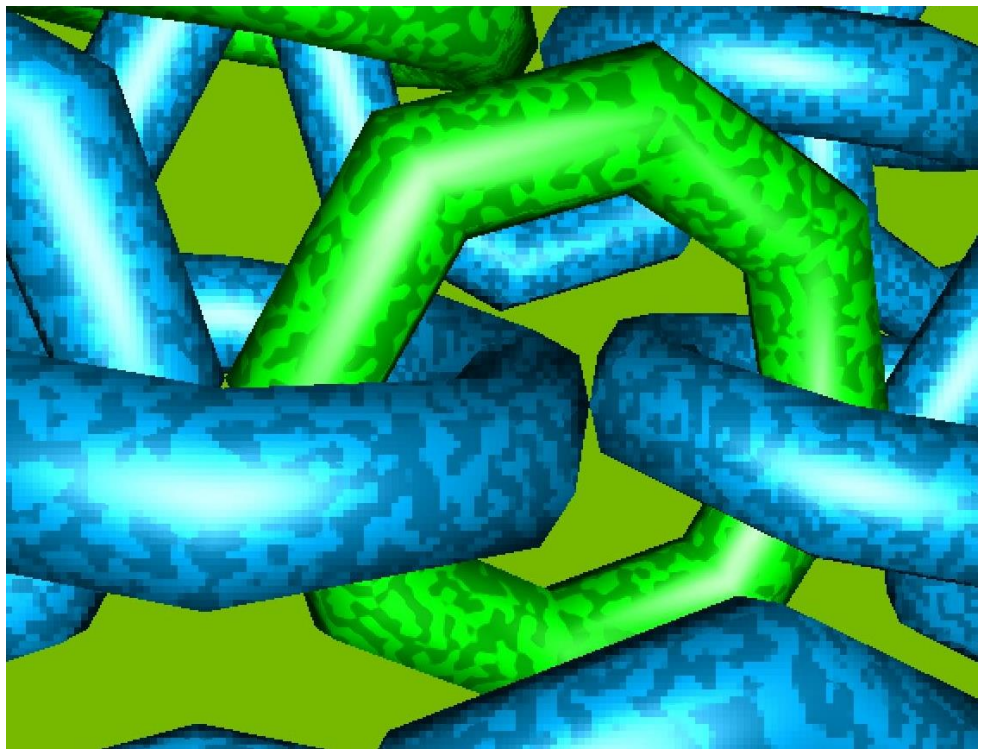
Lens Matched

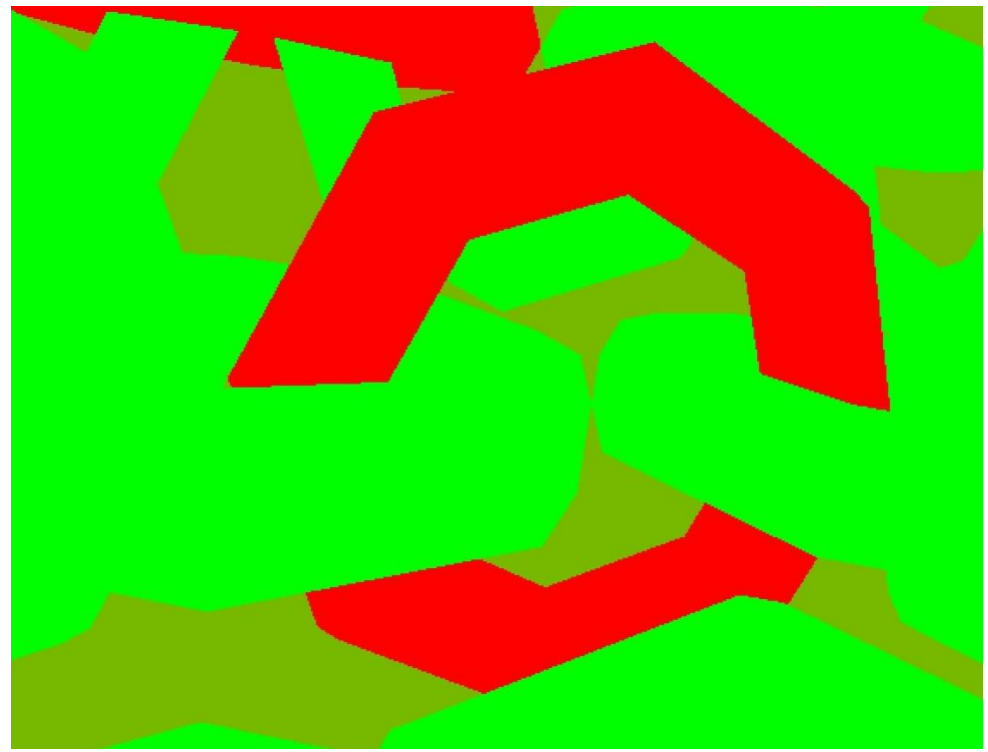


With Eye Tracking

VARIABLE RATE SHADING

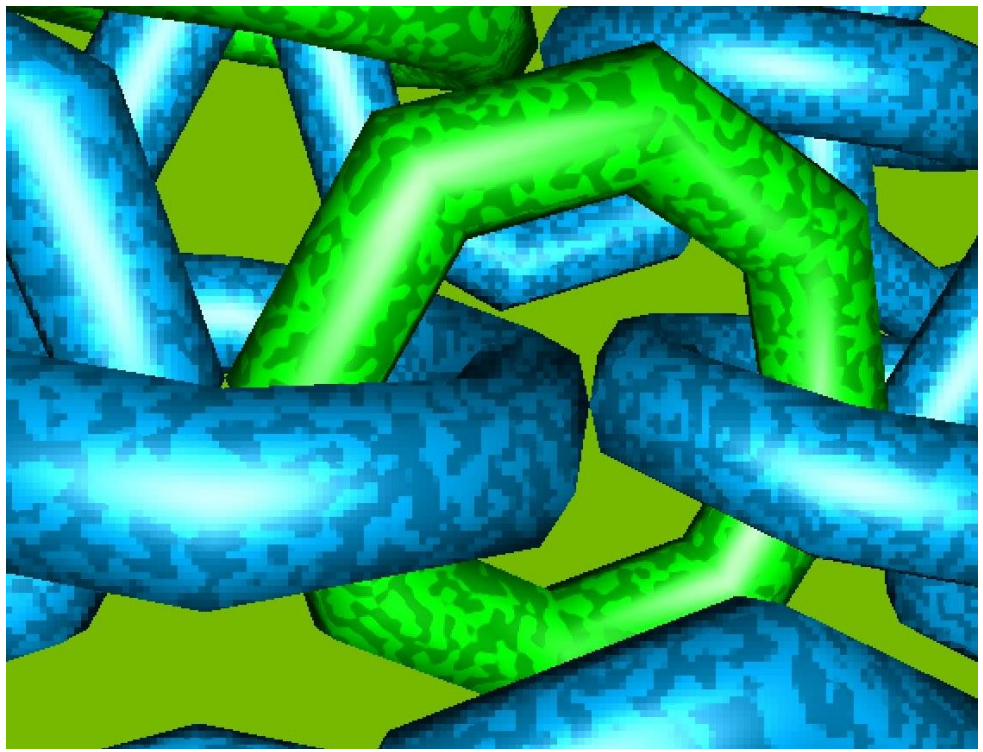
Content Adaptive Shading Rate

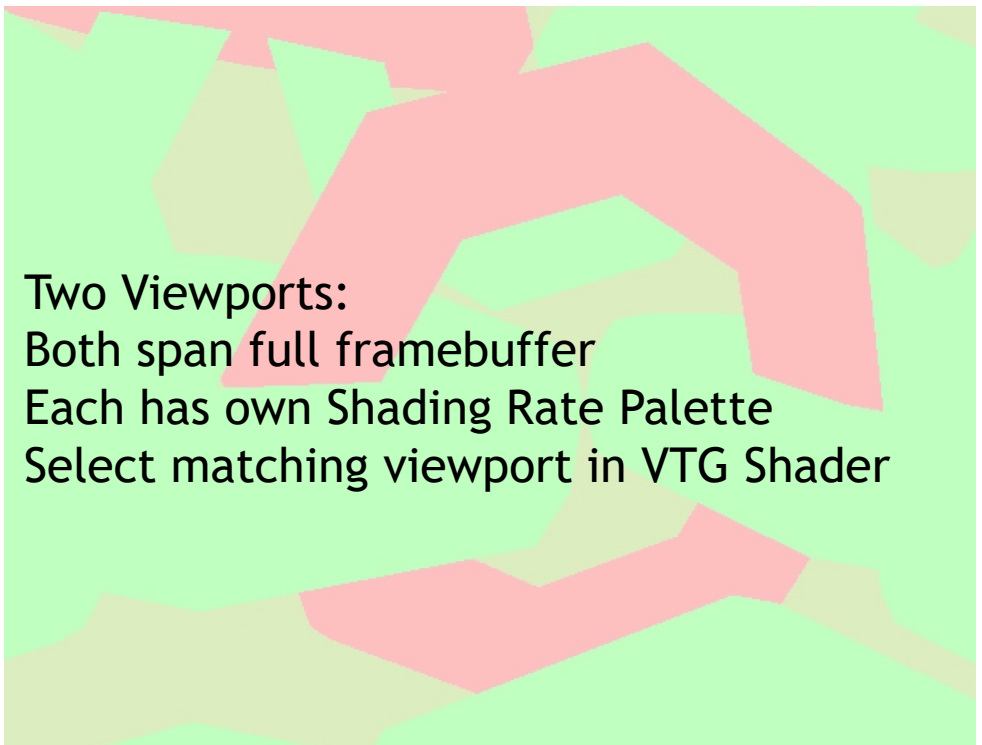




VARIABLE RATE SHADING

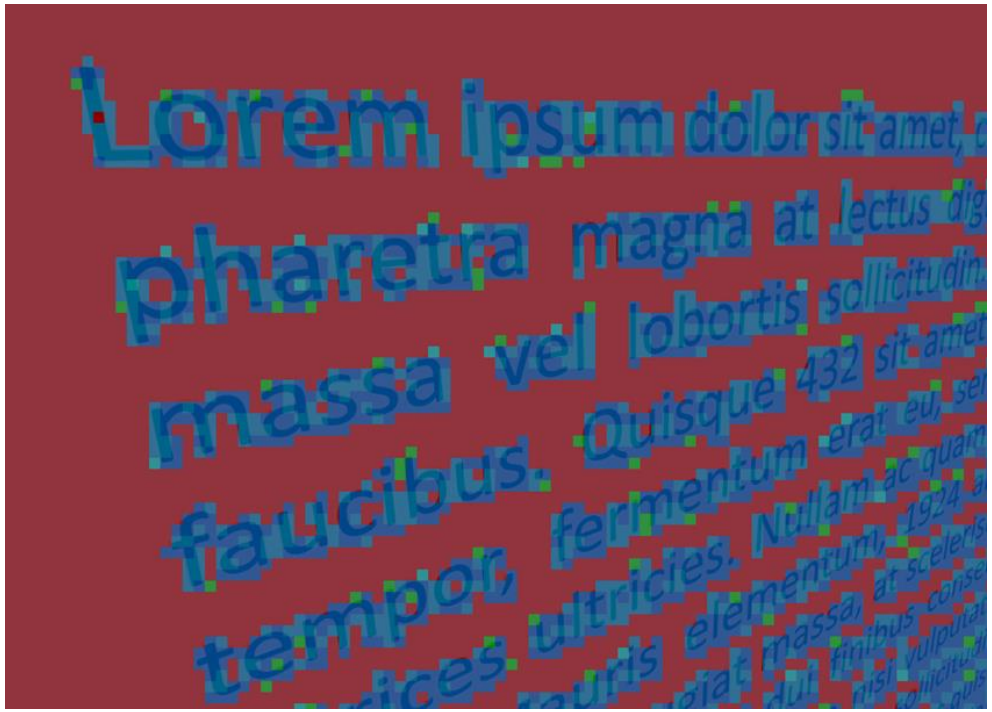
Content Adaptive Shading Rate





VARIABLE RATE SHADING

Content Adaptive Shading Rate



Legend:
Cold → Finer Shading
Hot → Coarse Shading

Content-adaptive Super Sampling for Text

VARIABLE RATE SHADING

Increased Shading Rate

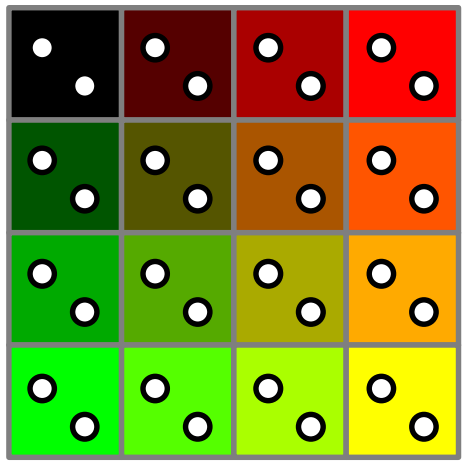
So far: reduced shading rate

Also possible: increase shading rate (where needed)

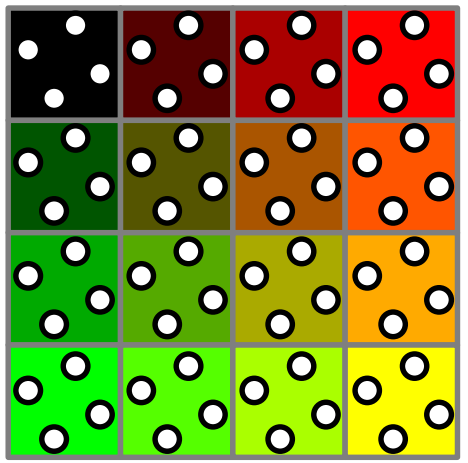
VARIABLE RATE SHADING

Shading Modes: Multi-Sample Framebuffers

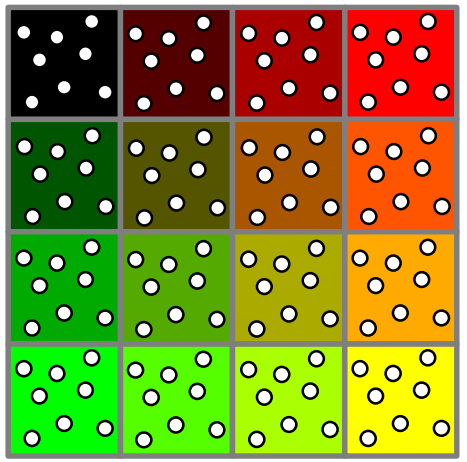
GL_SHADING_RATE_



2_INVOCATIONS_PER_PIXEL_NV



4_INVOCATIONS_PER_PIXEL_NV



8_INVOCATIONS_PER_PIXEL_NV

VARIABLE RATE SHADING

Increased Shading Rate

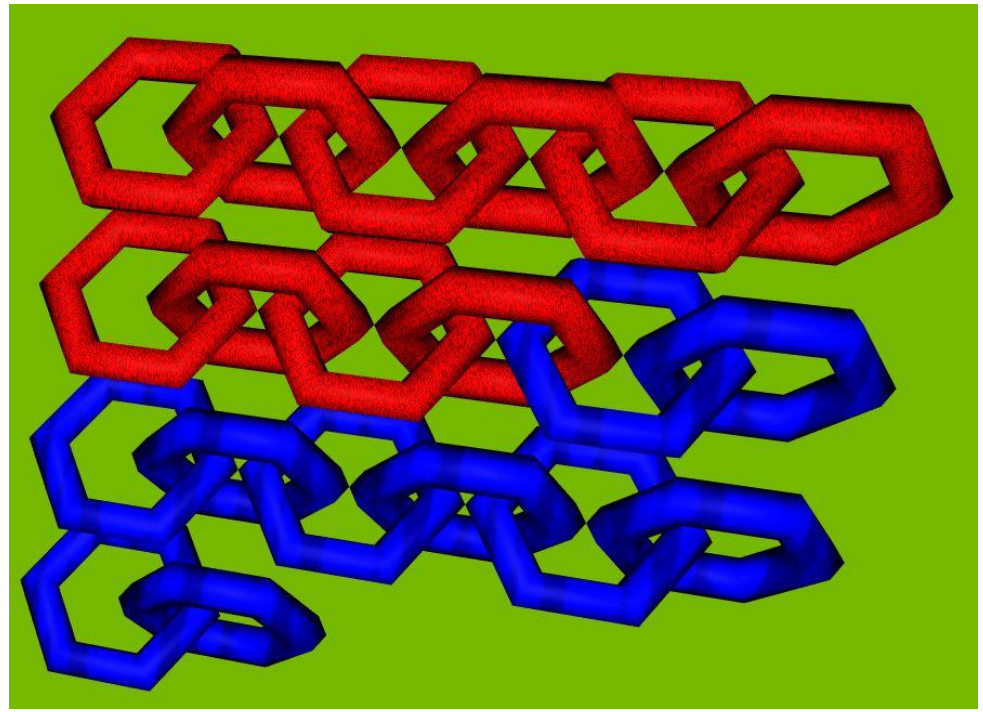
Idea:

Render to a MSAA buffer

1x shading for most of the scene (regular MSAA)

GL_SHADING_RATE_X_INVOCATIONS_PER_PIXEL_NV
for important objects or materials

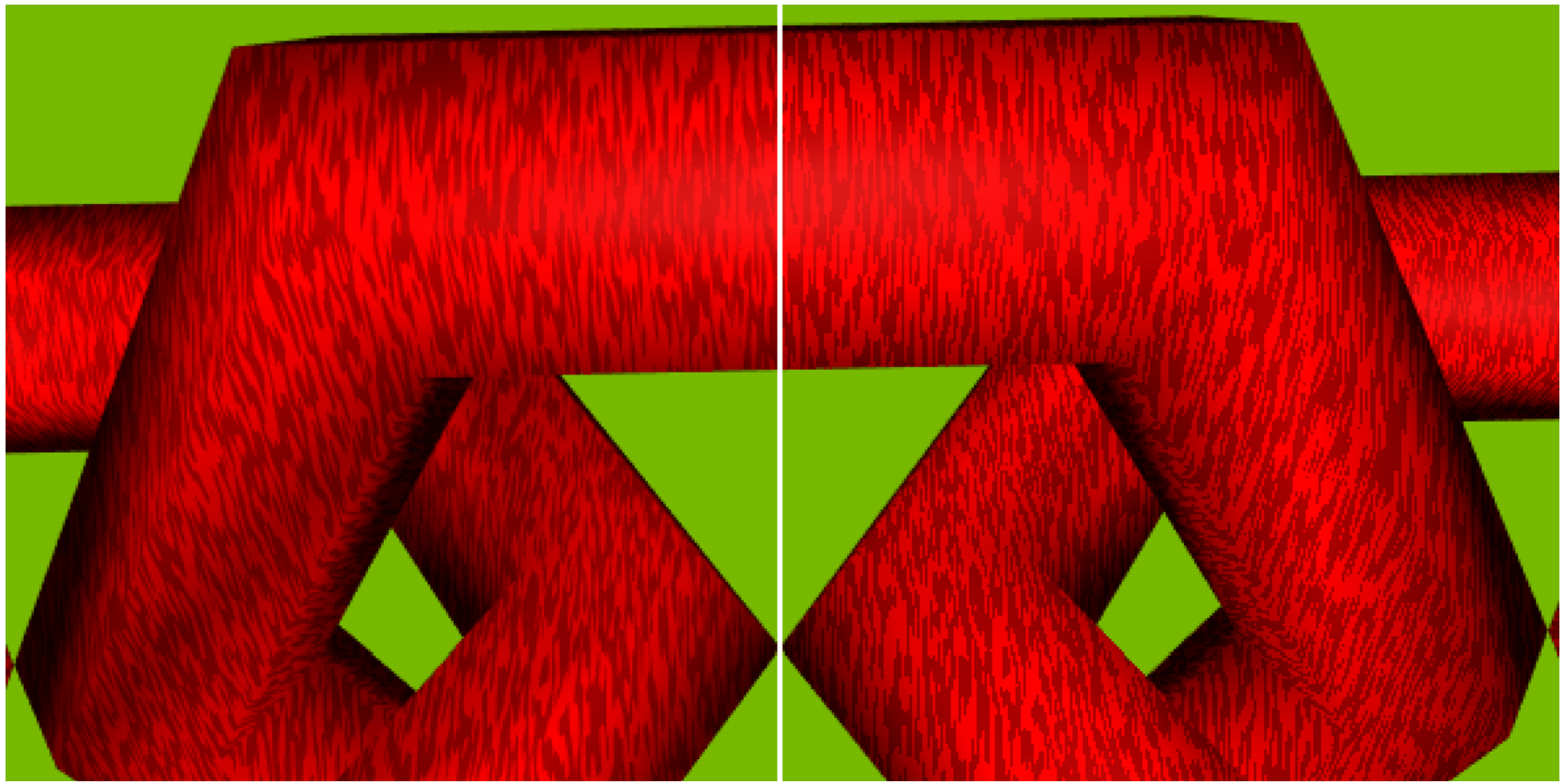
(X: 2,4,8)



(OpenGL) Sample from VRWorks

VARIABLE RATE SHADING

Increased Shading Rate: Animated Material

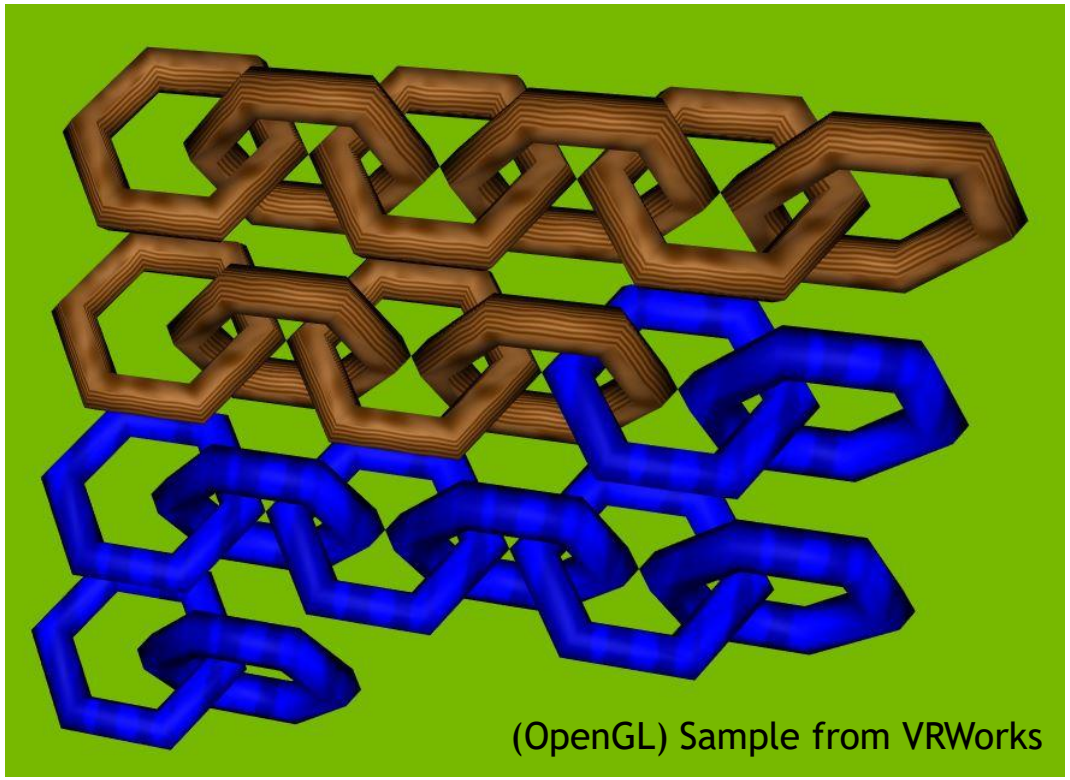


VRS

MSAA

VARIABLE RATE SHADING

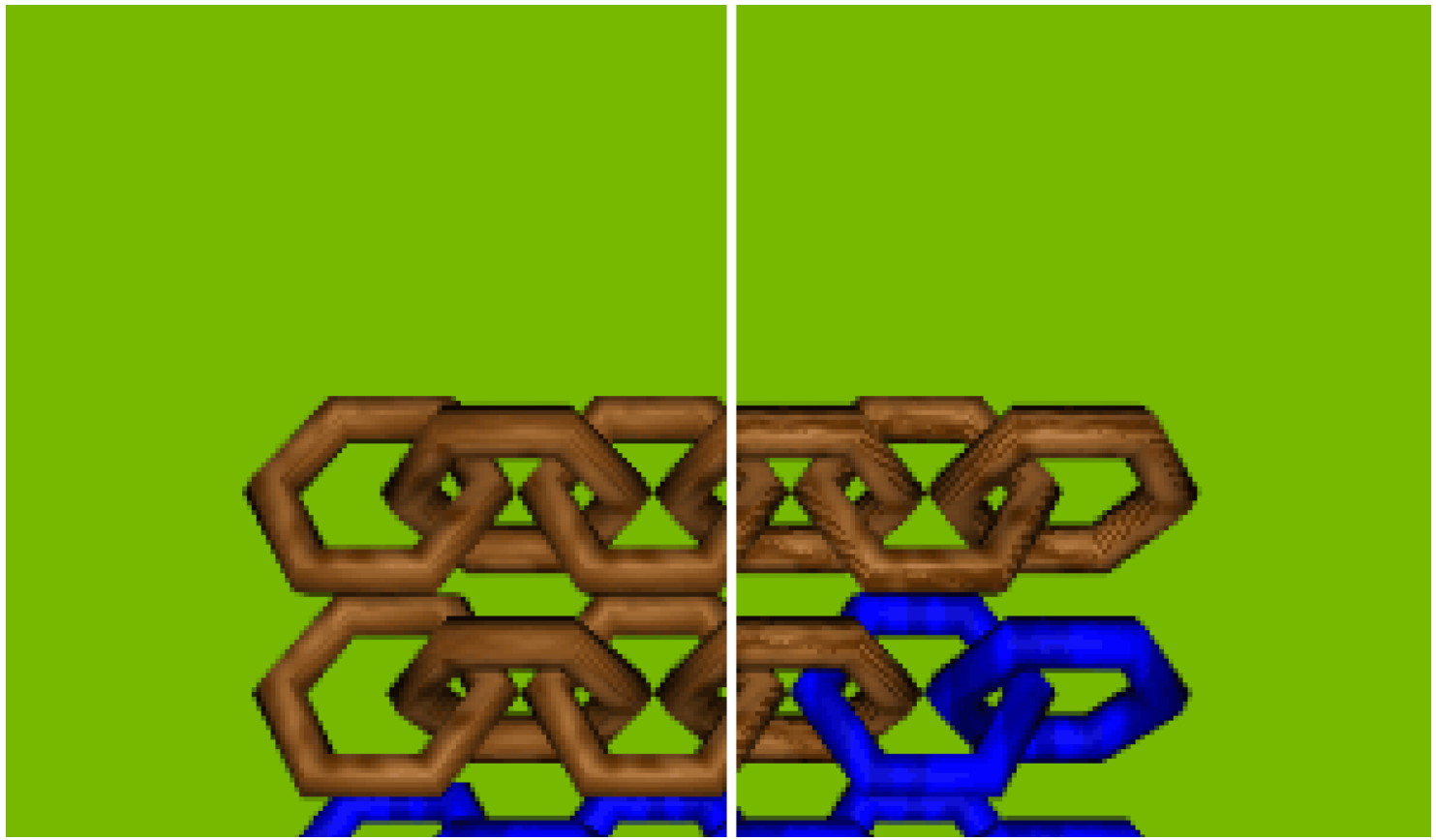
Increased Shading Rate: Procedural Material



VARIABLE RATE SHADING

Increased Shading Rate: Procedural Material

VRS



MSAA

VARIABLE RATE SHADING

Increased Shading Rate

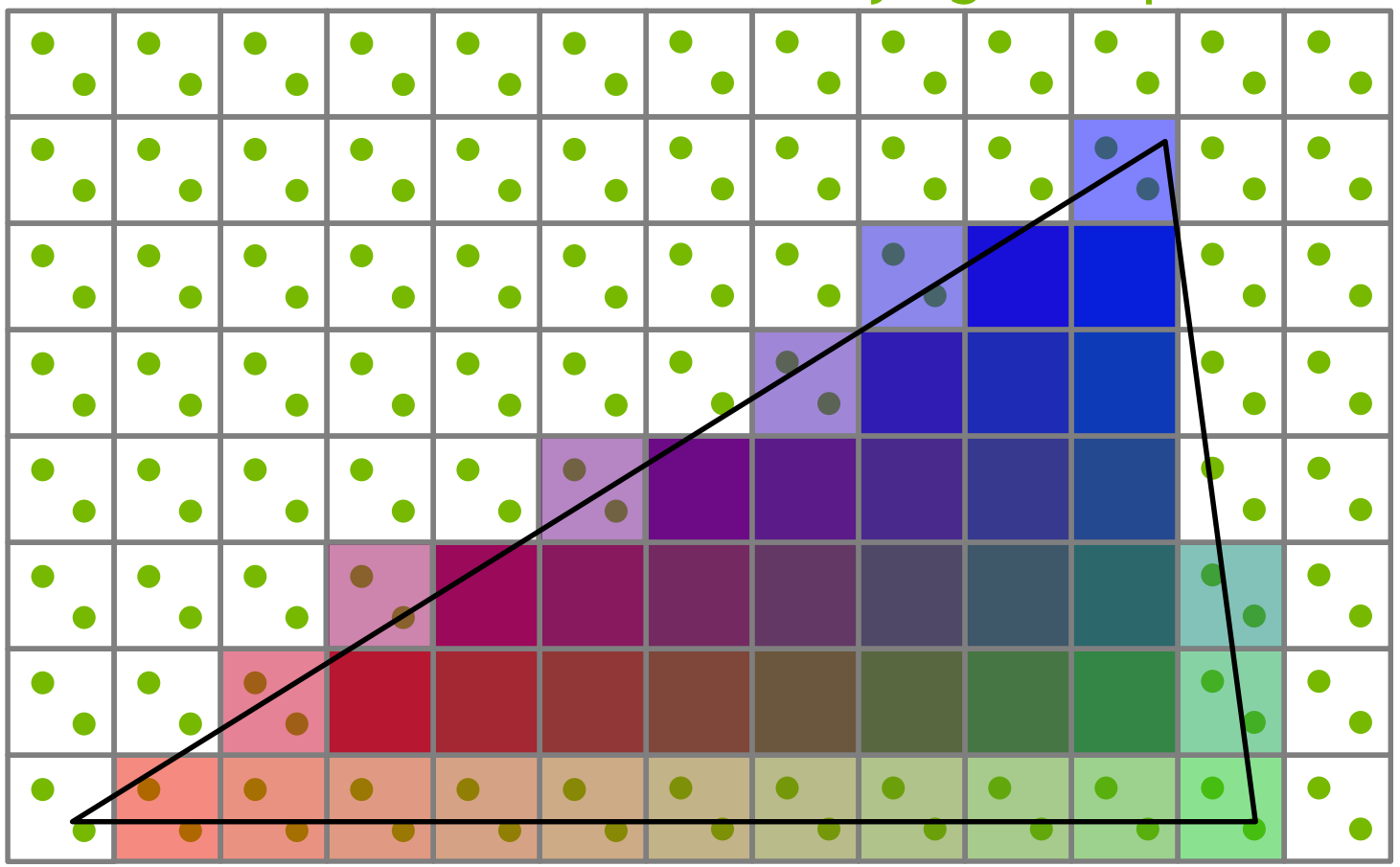
Edge quality: MSAA

Shading quality: MSAA **OR** like Super-Sampling (depending on requirement)

Performance: Adjustable between MSAA and Super-Sampling

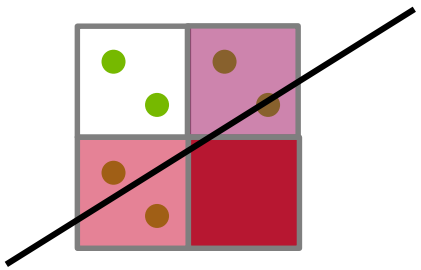
VARIABLE RATE SHADING

Varying Extrapolation



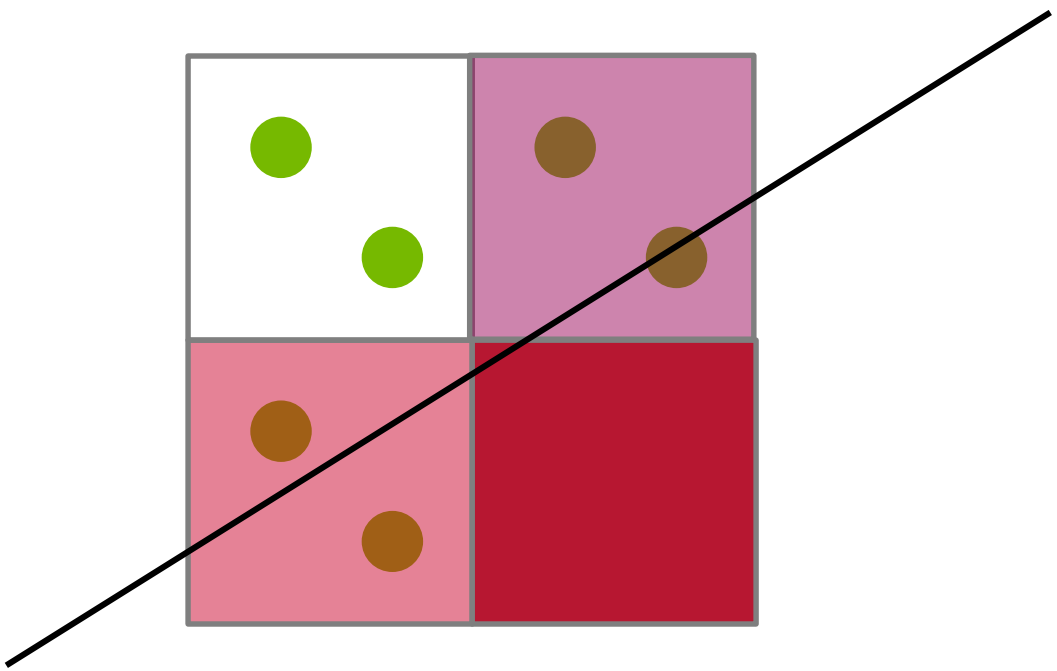
VARIABLE RATE SHADING

Varying Extrapolation



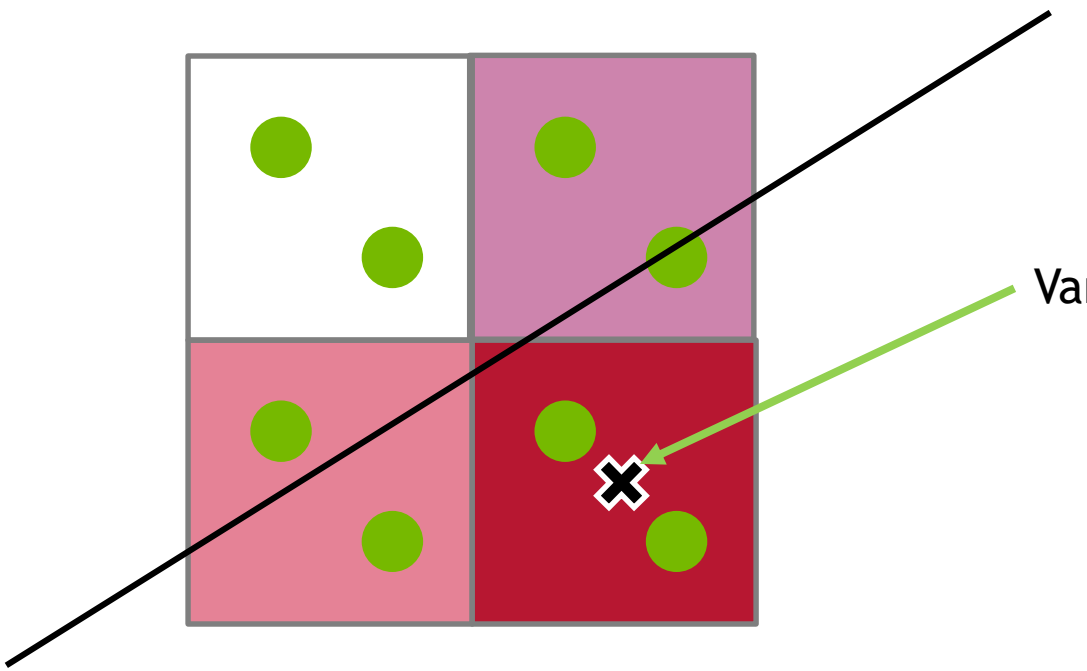
VARIABLE RATE SHADING

Varying Extrapolation



VARIABLE RATE SHADING

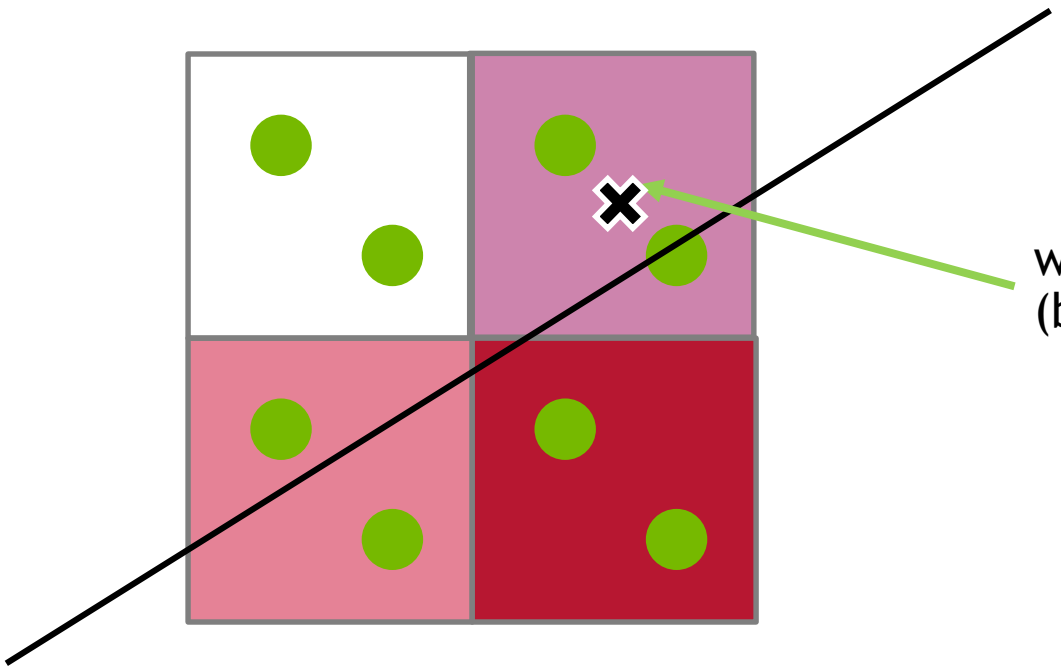
Varying Extrapolation



Varyings are interpolated in the Pixel center

VARIABLE RATE SHADING

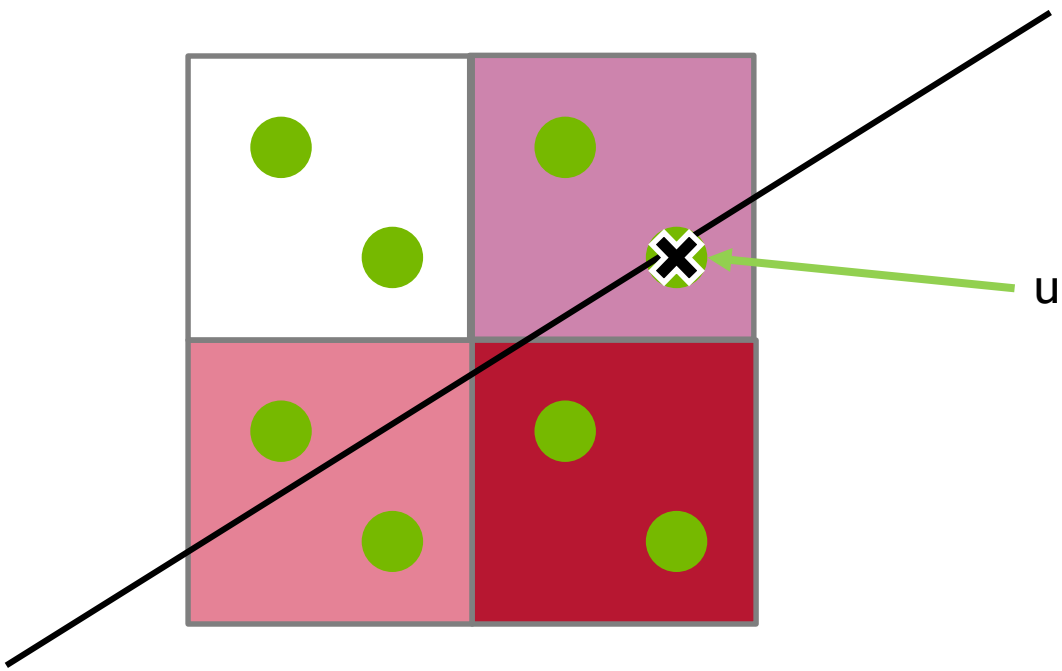
Varying Extrapolation



which means extrapolation for some
(but just a small amount)

VARIABLE RATE SHADING

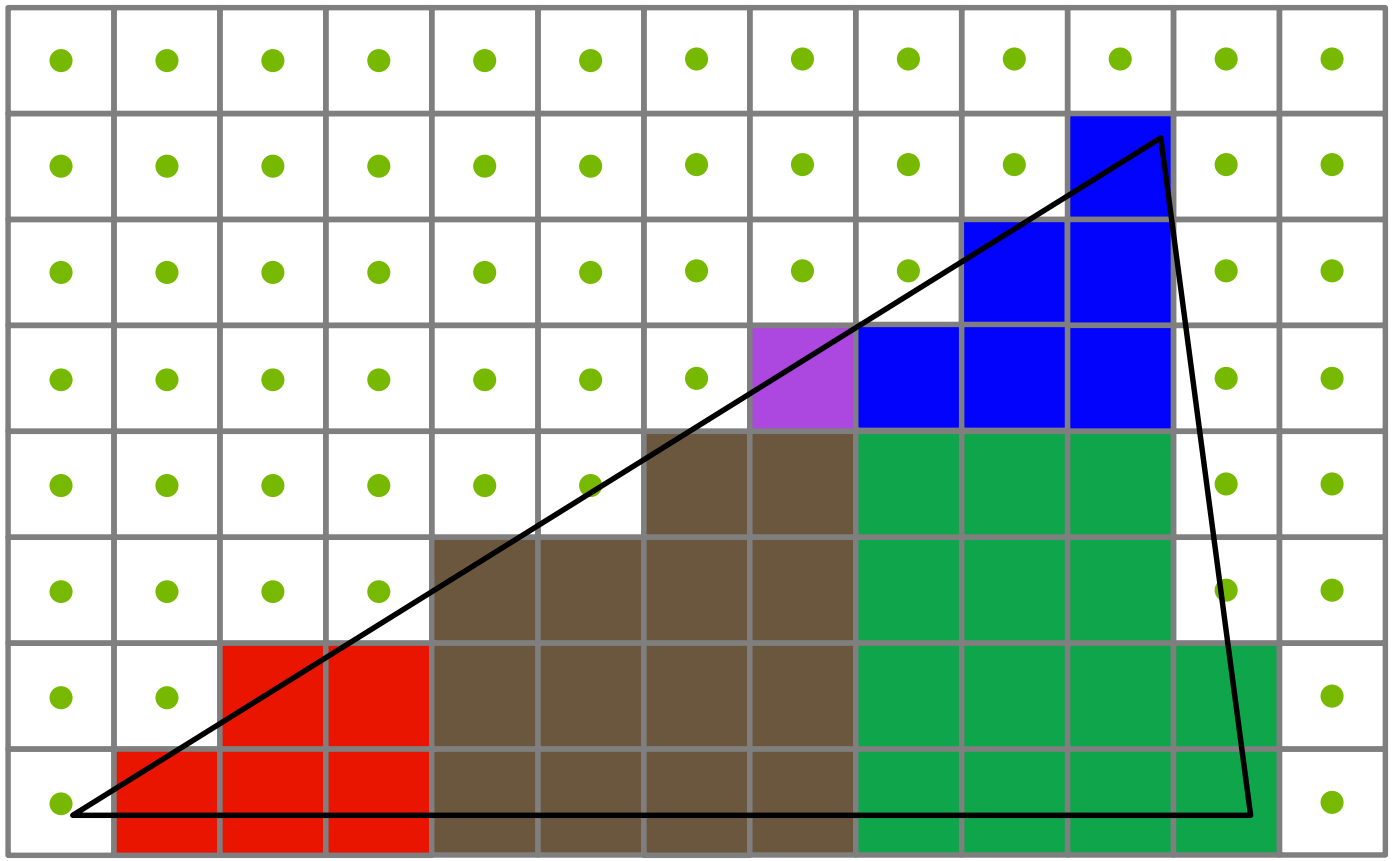
Varying Extrapolation



unless they are defined as centroid

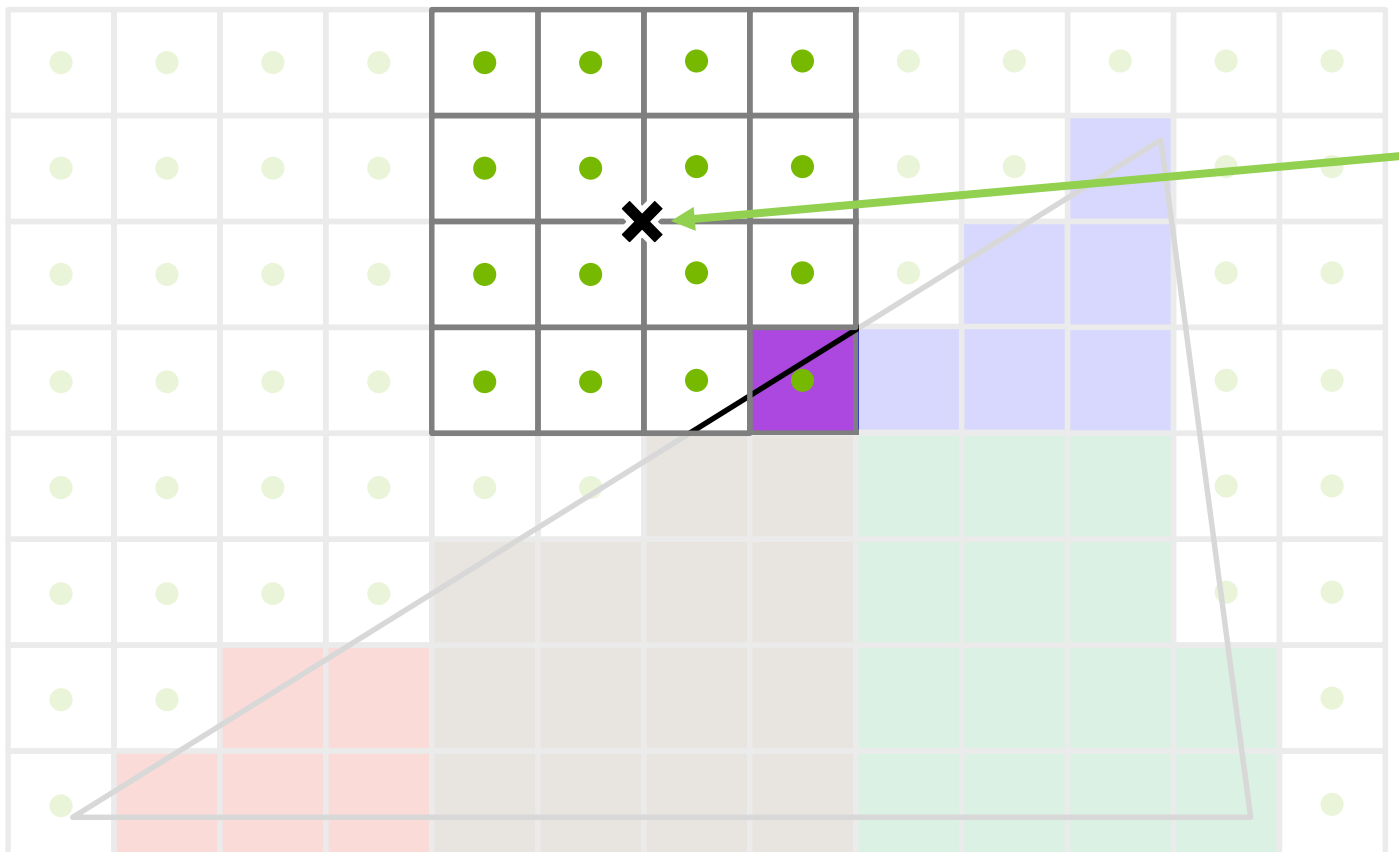
VARIABLE RATE SHADING

Varying Extrapolation



VARIABLE RATE SHADING

Varying Extrapolation

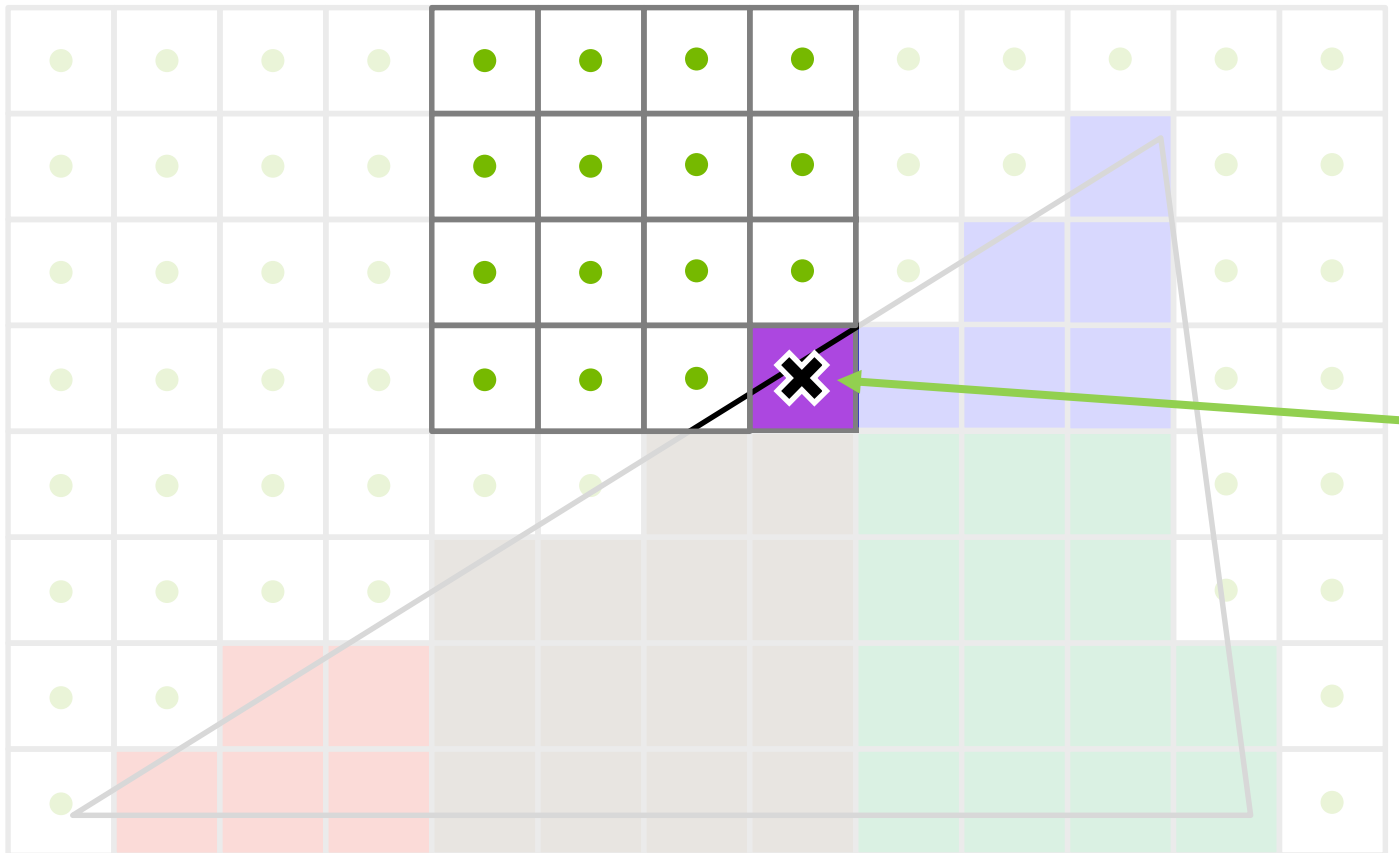


Varyings are interpolated in the coarse pixel center

Significantly more extrapolation compared to MSAA:
Use **centroid** to avoid artifacts!

VARIABLE RATE SHADING

Varying Extrapolation



Varyings are interpolated in the coarse pixel center

Significantly more extrapolation compared to MSAA:
Use **centroid** to avoid artifacts!

VARIABLE RATE SHADING

Recap

Reduces Fragment load

Allows to tailor workload to needs

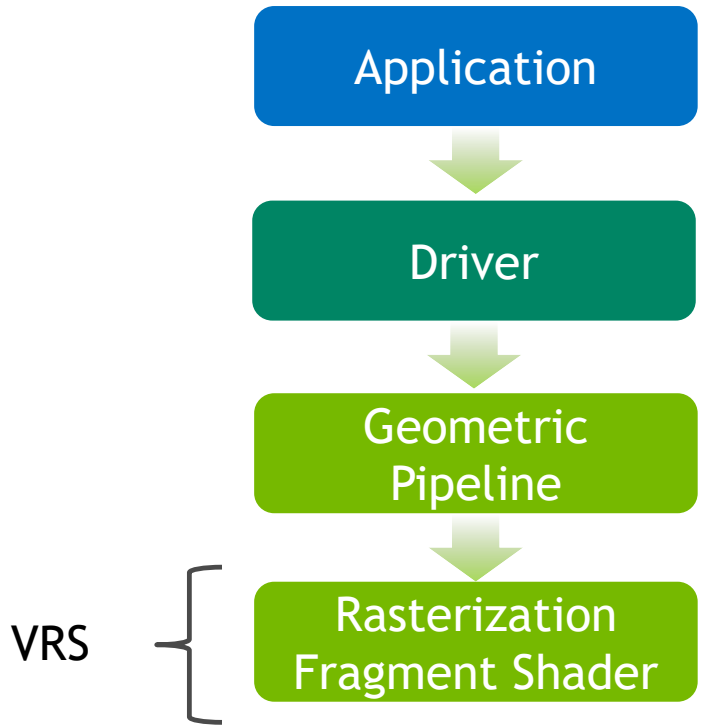
Fine-grained control over shading rate

Performance boost depends on shading complexity and triangle size

DX11: NVAPI

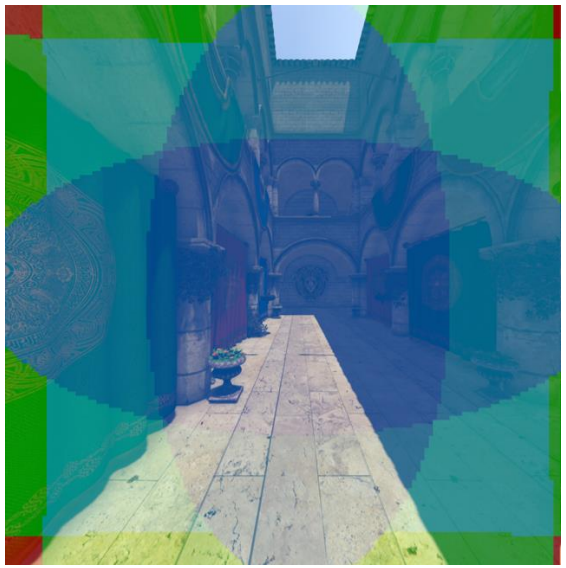
Vulkan: `VK_NV_shading_rate_image`

OpenGL: `GL_NV_shading_rate_image`

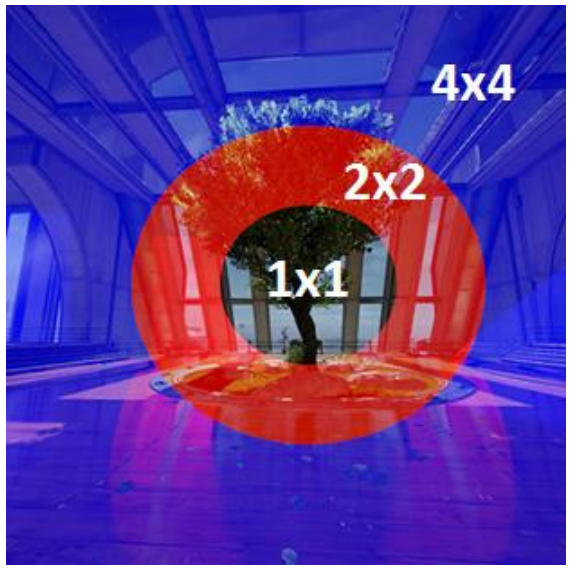


VARIABLE RATE SHADING

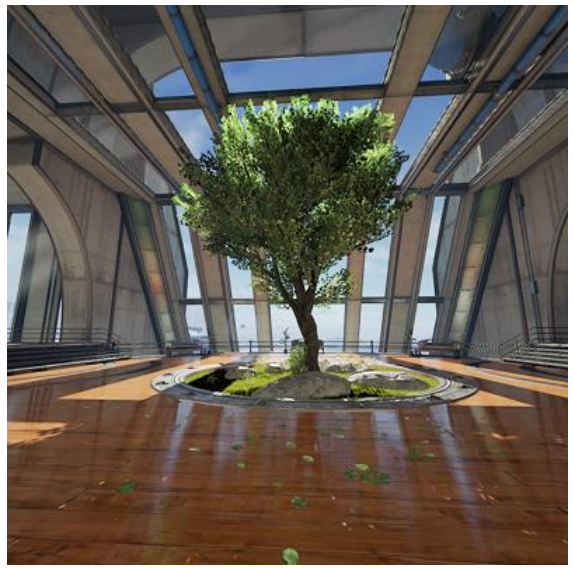
Recap



Lens Optimized Shading



Foveated Rendering



Content Adaptive Shading

See More VR on the Exhibition Floor

Expo Hall 3, Concourse Level

VR VILLAGE

Explore the VR Village to get hands-on with the latest advances in virtual reality

VR THEATER

Go to the VR Theater to see and experience narrated VR demos built by our partners

VR PARTNERS

Explore a great lineup of VR partners around the VR Village showcasing their groundbreaking technology

COME EXPLORE ALL THINGS VR AT GTC 2019

VR VILLAGE HOURS Wednesday: 12:00pm - 7:00pm Thursday: 11:00am - 2:00pm

TRY IT OUT!

..and more information

NVIDIA VRWorks SDK provides OpenGL, Direct3D & Vulkan samples

developer.nvidia.com/vrworks

Upcoming Zerolight VR talk discussing MVR, VRS and VR SLI

S9209 - Advances in Real-Time Automotive Visualisation – Thu, 11:00 – 11:50, Room 230A

More detail in our previous GTC talks:

2018 – S8695 – NVIDIA VR Update

2017 – S7191 – Vulkan Technology Update

2016 – S6338 – VR Multi GPU Acceleration Featuring Autodesk VRED

2015 – S5668 – VR Direct: How NVIDIA Technology Is Improving The VR Experience


NVIDIA