

Lei Yang

Sr. GPU Graphics Architect

Dmitry Zhdan

Sr. DevTech Engineer

Matthew Johnson

Sr. DevTech Engineer



# NVIDIA ADAPTIVE SHADING OVERVIEW

03/22/2019 (Fri), 1:30PM, Room 205, South Hall

[www.nvidia.com/GDC](http://www.nvidia.com/GDC)

# AGENDA

## Introduction to Variable Rate Shading (VRS)

### API Support

DirectX 11, 12, Vulkan and OpenGL

### NVIDIA Adaptive Shading (NAS)

Content adaptive shading

Motion adaptive shading

## Implementation in Wolfenstein II: the New Colossus

### Demo



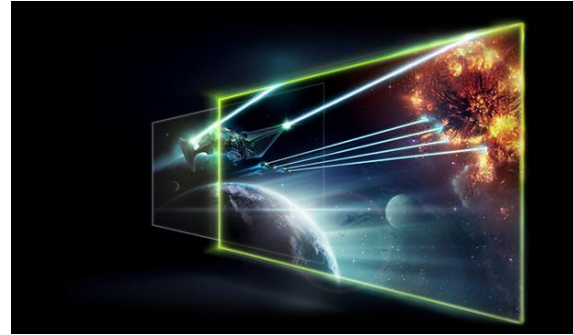
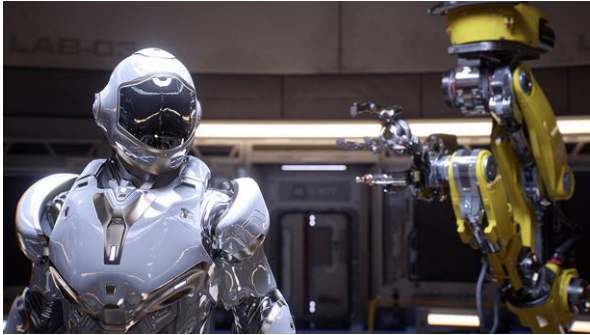
# INTRODUCTION TO VARIABLE RATE SHADING

# OVERVIEW

## Why do we need variable rate shading?

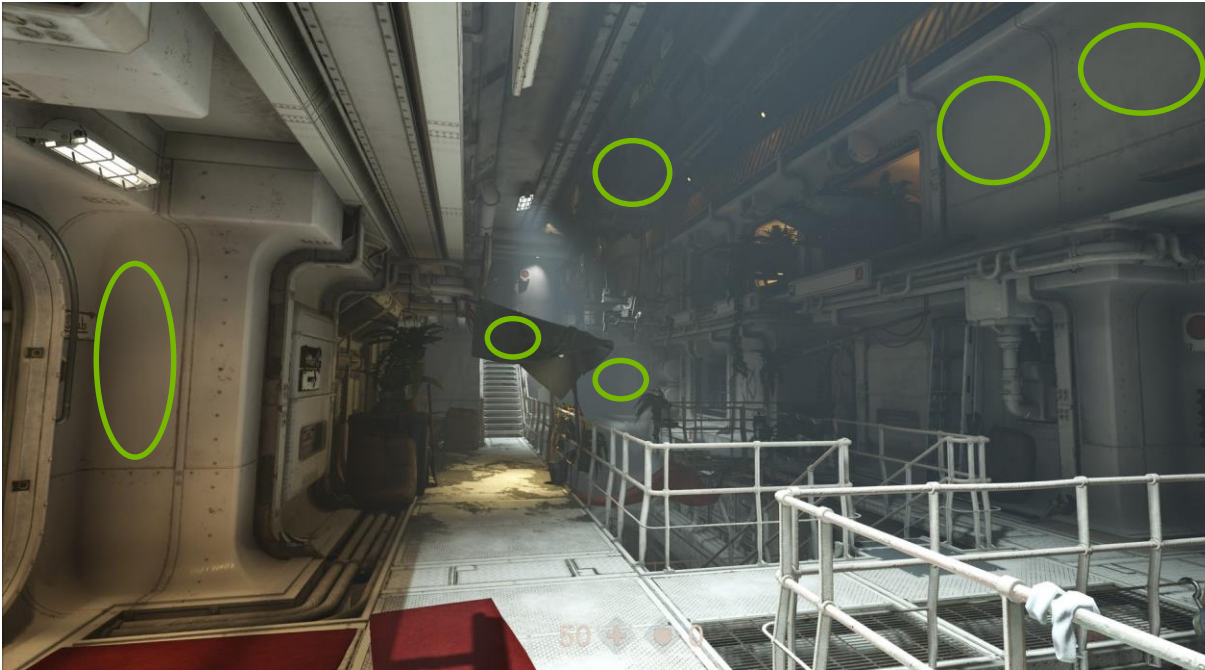
The cost of pixel shading increases dramatically in today's games

- Enhanced realism and special effects
- Higher resolution and framerate



# OVERVIEW

## Pixel Shading Inefficiencies



Shader gets run  
every single pixel,  
regardless of content

# OVERVIEW

## Variable Rate Shading (VRS)

Variable Rate Shading (VRS) is a new GPU feature to reduce shading cost adaptively

- Available on NVIDIA Turing GPUs
- Supported by multiple APIs

Basic idea:

Vary the shading rate, maintain the visibility rate

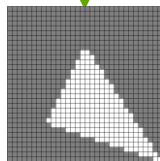
# Reduced Resolution Rendering + Upscaling

Vertex Shading  
Tessellation  
Geometry Shading

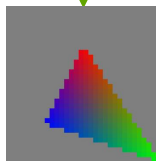
Rasterization

Pixel  
Shading

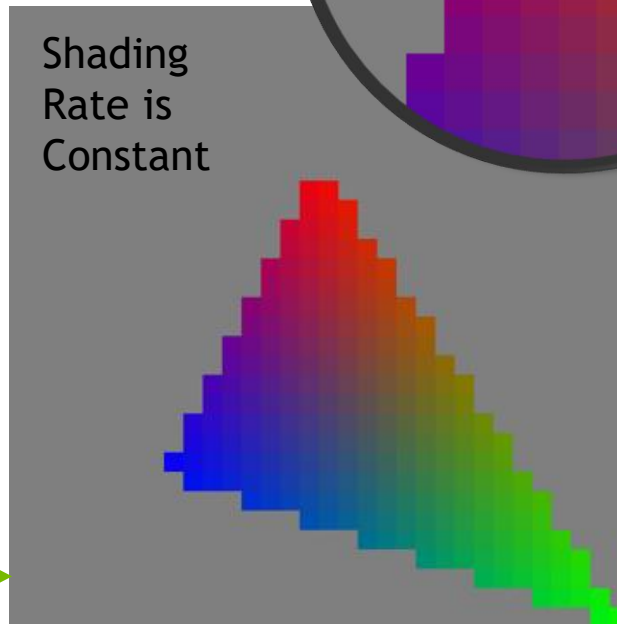
Upscaling  
Pass



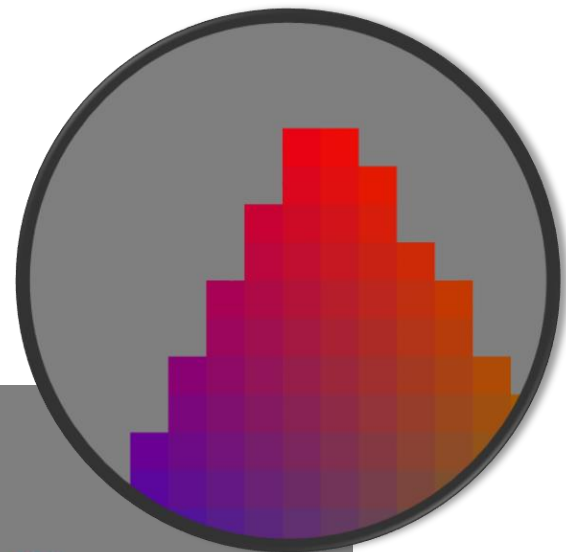
Lower  
Resolution  
Rasterization



Lower  
Resolution  
Shading



After Upscaling



Jagged  
Edges

# Variable Rate Shading

Vertex Shading  
Tessellation  
Geometry Shading

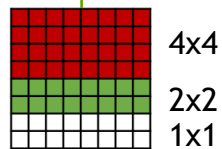
Rasterization

Pixel  
Shading

Output

No  
Upscaling  
Needed

Native  
Resolution  
Rasterization



Shading Rate Image  
(1 entry per 16x16  
screen tile)

Shading  
Rate is  
Variable

4x4

2x2

1x1

Edges are  
Preserved

4x4

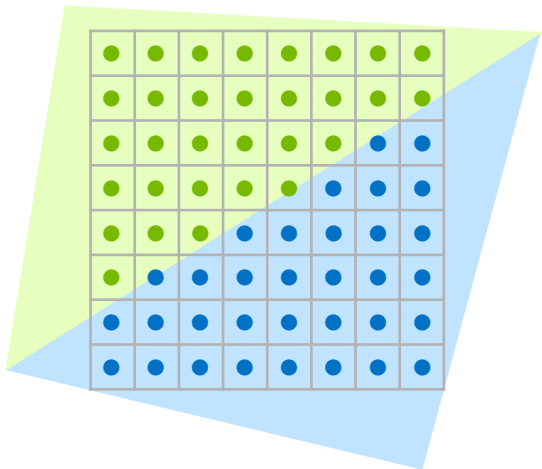
2x2

# OVERVIEW

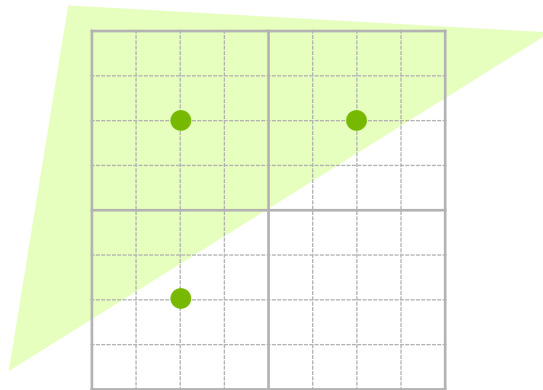
## Variable Rate Shading (VRS)

Rasterize at full sample rate, shade at specified shading rate, broadcast to all covered samples

1x1 shading



4x4 shading

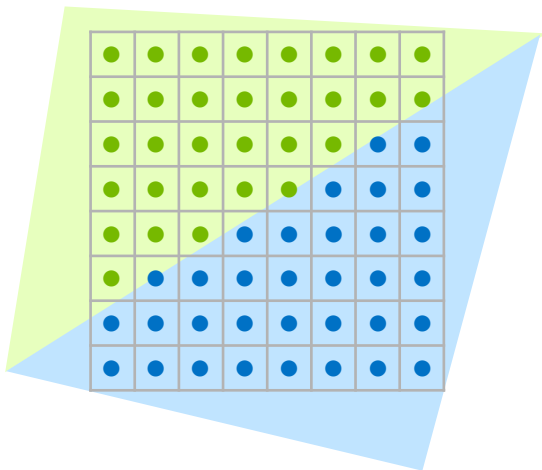


# OVERVIEW

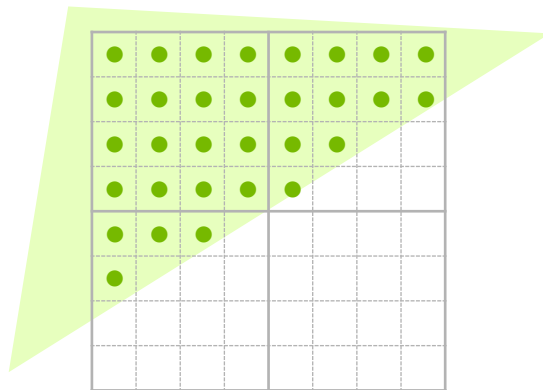
## Variable Rate Shading (VRS)

Rasterize at full sample rate, shade at specified shading rate, broadcast to all covered samples

1x1 shading



4x4 shading

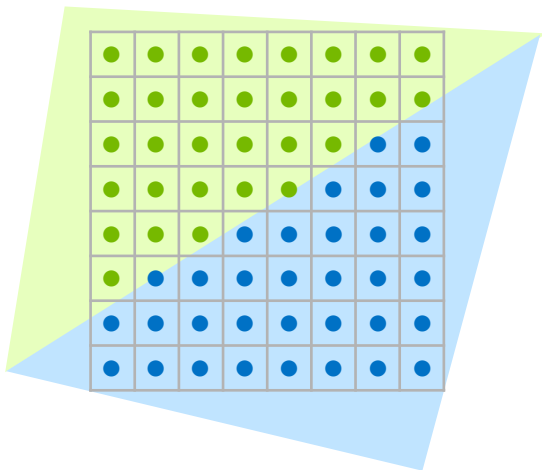


# OVERVIEW

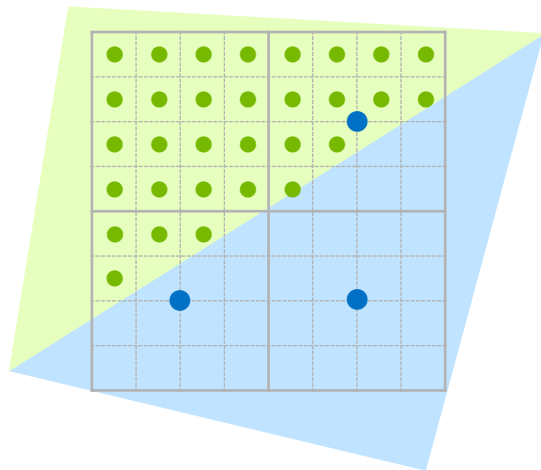
## Variable Rate Shading (VRS)

Rasterize at full sample rate, shade at specified shading rate, broadcast to all covered samples

1x1 shading



4x4 shading

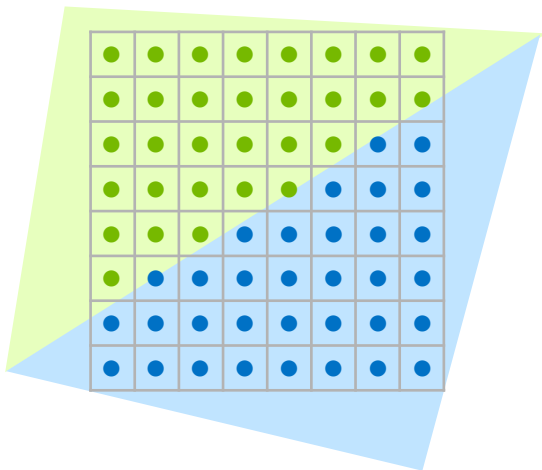


# OVERVIEW

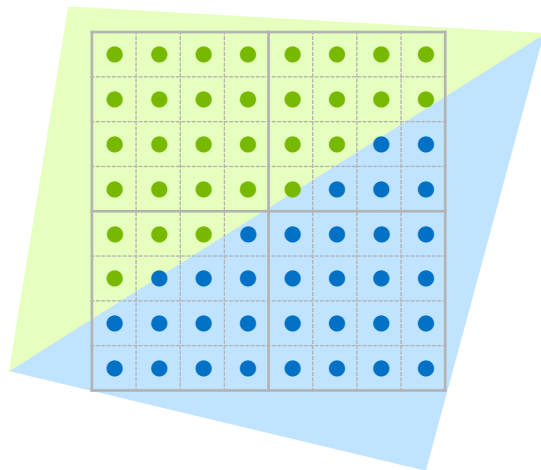
## Variable Rate Shading (VRS)

Rasterize at full sample rate, shade at specified shading rate, broadcast to all covered samples

1x1 shading



4x4 shading



# OVERVIEW

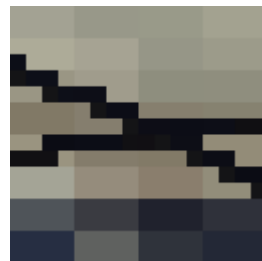
## Shading rate options



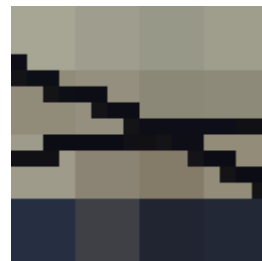
1x1



2x1



4x2



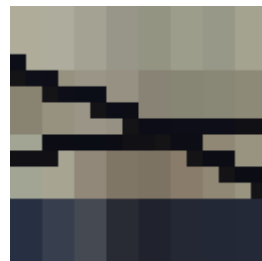
4x4



1x2



2x2



2x4

# GOALS

## Performance improvement

The goal of variable rate shading is to only shade at the detail when necessary

This saves shading costs in forward passes

This helps if the draw is limited by shading unit costs

Two cases it doesn't help - shouldn't hurt, either!

- The draw is framebuffer bandwidth or front-end limited
- The primitive-to-pixel ratio is close to or larger than 1

# VARIABLE RATE SHADING (VRS)

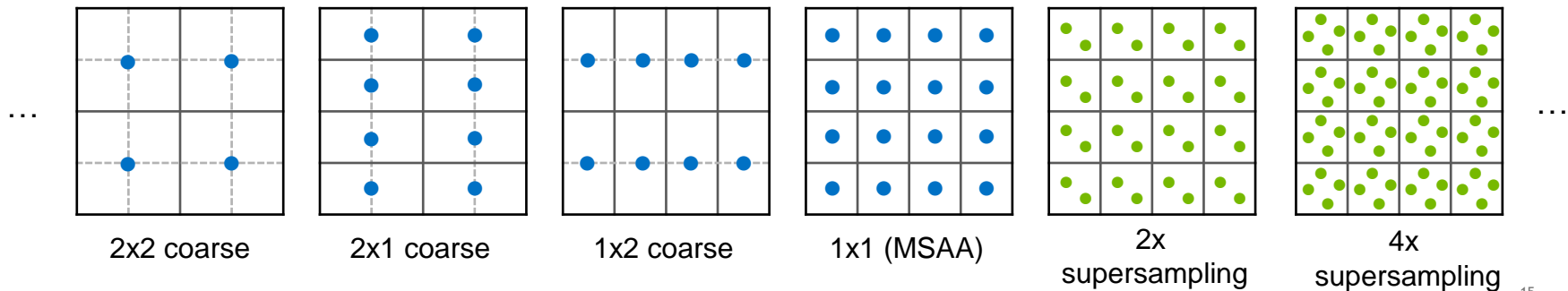
## VRS and MSAA

VRS is an extension to Multi-Sample Anti-Aliasing (MSAA):

Shading rate is decoupled from visibility sampling rate (fixed ratio in MSAA)

On Turing, VRS supports **hybrid supersampling**:

Shade  $>1$  per pixel (supersampling), or  $<1$  per pixel (coarse) on the same screen



# FLEXIBILITY

## VRS and MSAA modes

Twelve modes to choose from per 16x16 screen tile

Shading rate modes

Modes	Min Threads/Pixel	Max Pixels/Thread
Cull	0	-
Supersamp.	1, 2, 4, 8, 16	-
Coarse	-	2x1, 1x2, 2x2, 4x2, 2x4, 4x4

Number of pixels/samples to broadcast to

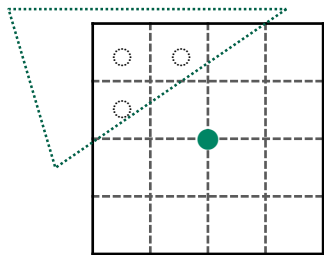
Coarse mode	MSAA Mode				
	Off (1x)	2x	4x	8x	16x
Off (1x1)	1	2	4	8	16
2x1/1x2	2	4	8	16	X
2x2	4	8	16	X	X
4x2/2x4	8	16	X	X	X
4x4	16	X	X	X	X

# INTERPOLATION

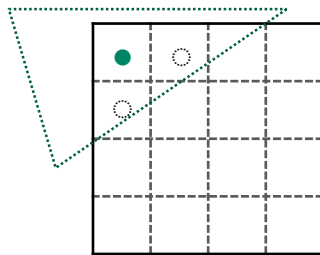
## Center and centroid modes

Like MSAA, coarse pixel attributes are interpolated in coarse pixel center

Centroid mode can help with out of gamut interpolation  
(gradients need to be adjusted manually)



Center

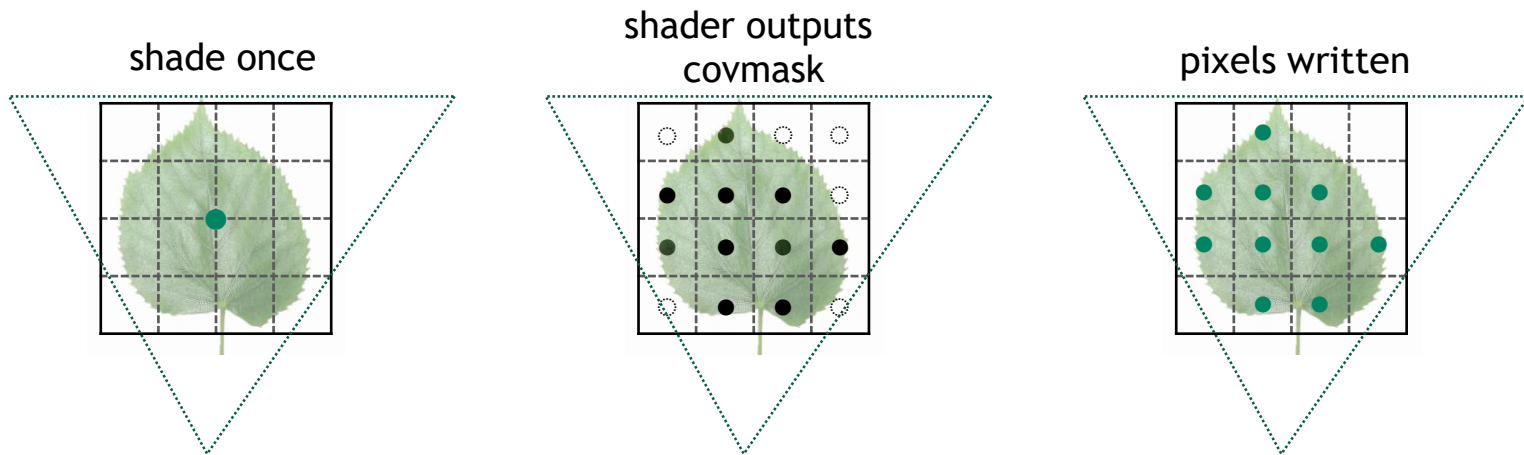


Centroid

# CUSTOMIZE COVERAGE

Write to selected pixels only

Similar to MSAA, coverage mask can be modified in the pixel shader  
Useful for per-pixel alpha test when shading coarse

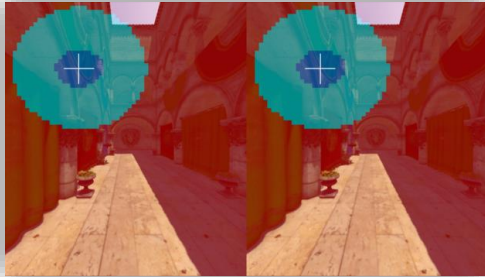


# APPLICATIONS

## Common use cases

Application has flexibility to change shading rate to whatever it likes

Foveated shading



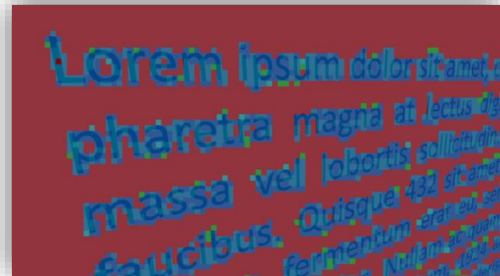
Lens-matched shading



Adaptive shading



Selective supersampling



The background is a solid black field. It is populated with numerous thin, light green lines that crisscross the frame in various directions. At the intersections of these lines, there are small, bright green circular dots. Some of these dots are slightly larger and more prominent than others. The overall effect is a complex, web-like pattern that suggests a network or a digital space.

**API SUPPORT**

# VARIABLE RATE SHADING

## Typical usage

App queries VRS support

App creates R8ui shading rate texture {w / tileSize, h / tileSize}

App fills the shading rate texture with desired pattern

App enables VRS in the graphics pipeline and binds shading rate texture

App issues draws

# DX11 NVAPI

## Typical usage

App queries VRS support

[ `NvAPI_D3D1x_GetGraphicsCapabilities` ]

App creates a regular Texture2D (R8\_UINT) for shading rate

App creates a new custom type of View (like RTV/SRV/UAV)

[ `NvAPI_D3D11_CreateShadingRateResourceView` ]

App populates this texture with shading rate pattern (draw/compute/upload)

App programs Shading Rate Lookup Table per viewport

[ `NvAPI_D3D11_RSSetViewportsPixelShadingRates` ]

App binds the shading rate resource view

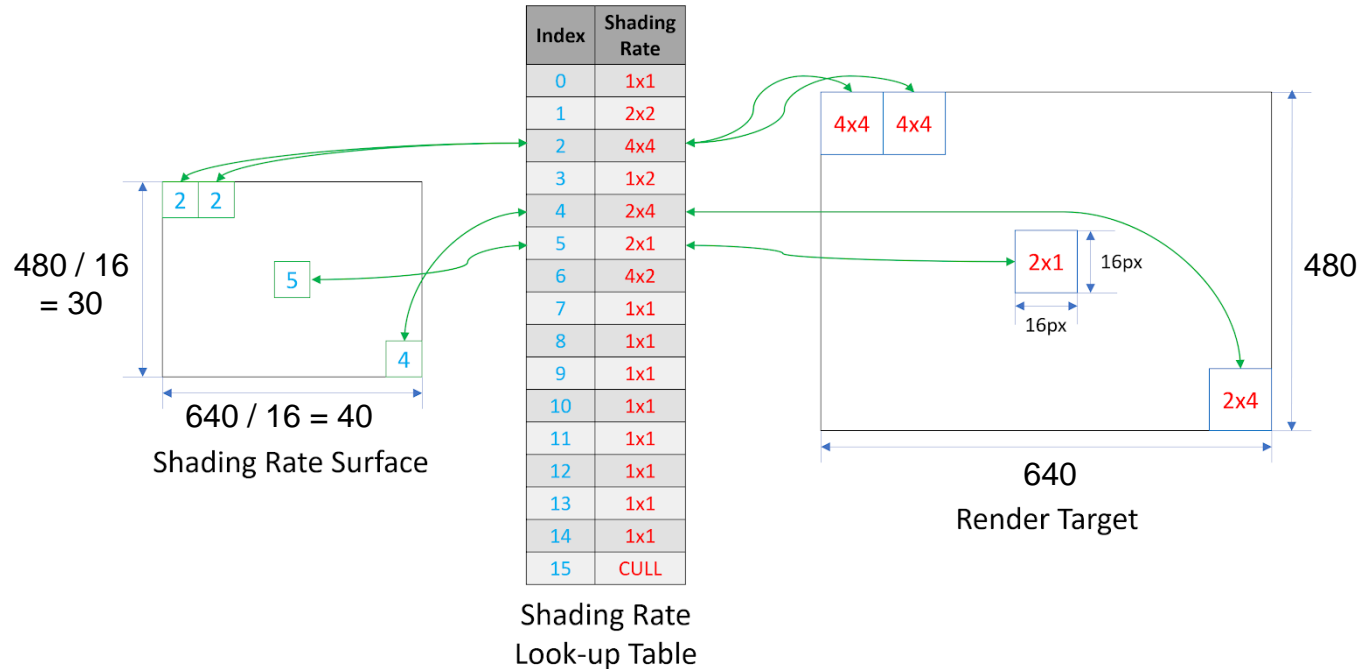
[ `NvAPI_D3D11_RSSetShadingRateResourceView` ]

App issues scene draw

[ `NvGetShadingRate` ] (optional, access shading rate in HLSL)

# DX11 NVAPI

## Shading rate lookup table



# VULKAN API

## VK\_NV\_shading\_rate\_image

App creates the shading rate image

[vkCreateImage, VK\_IMAGE\_USAGE\_SHADING\_RATE\_IMAGE\_BIT\_NV]

App creates image view

[vkCreateImageView]

App populates this texture with shading rate pattern (draw/compute/upload)

App enables shading rate image usage in the pipeline

[VkPipelineViewportShadingRateImageStateCreateInfoNV.shadingRateImageEnable]

App programs the per-viewport shading rate palette entries

[VkPipelineViewportShadingRateImageStateCreateInfoNV.pShadingRatePalettes/.viewportCount]

App binds the shading rate image

[vkCmdBindShadingRateImageNV]

App issues scene draws

[gl\_FragmentSize] (optional, access shading rate in GLSL)

# VULKAN API

## Additional info

Resource transitions are explicit (VRS specific - `VK_ACCESS_SHADING_RATE_IMAGE_READ_BIT_NV`, `VK_IMAGE_LAYOUT_SHADING_RATE_OPTIMAL_NV`)

[https://www.khronos.org/registry/vulkan/specs/1.1-extensions/html/vkspec.html#VK\\_NV\\_shading\\_rate\\_image](https://www.khronos.org/registry/vulkan/specs/1.1-extensions/html/vkspec.html#VK_NV_shading_rate_image)

# OpenGL API

## GL\_NV\_shading\_rate\_image

App creates shading rate image

```
[ TEXTURE_2D{ _ARRAY }, R8UI ]
```

App populates the image with shading rate pattern

```
[ Tex{Sub}Image2D ]
```

App programs the per viewport shading rate palette entries

```
[ ShadingRateImagePaletteNV ]
```

App binds the shading rate image

```
[ BindShadingRateImageNV ]
```

App enables shading rate image usage

```
[ Enable(SHADING_RATE_IMAGE_NV) ]
```

App draws the scene

```
[ gl_FragmentSize ] (optional, access shading rate in GLSL)
```

# DX12 API

## Typical usage

App queries device tier and caps

[ `CheckFeatureSupport`, `D3D12_FEATURE_D3D12_OPTIONS6`, `D3D12_SHADING_RATE_TIER` ]

App creates shading rate texture

[ `D3D12_RESOURCE_STATE_SHADING_RATE_SOURCE`, `DXGI_FORMAT_R8_UINT` ]

App populates the texture with shading rate pattern

[ `Compute/Copy/Clear` ]

App sets the base shading rate and combiner function

[ `RSSetShadingRate`, `D3D12_SHADING_RATE_COMBINER` ]

App binds the shading rate image

[ `RSSetShadingRateImage` ]

App draws the scene

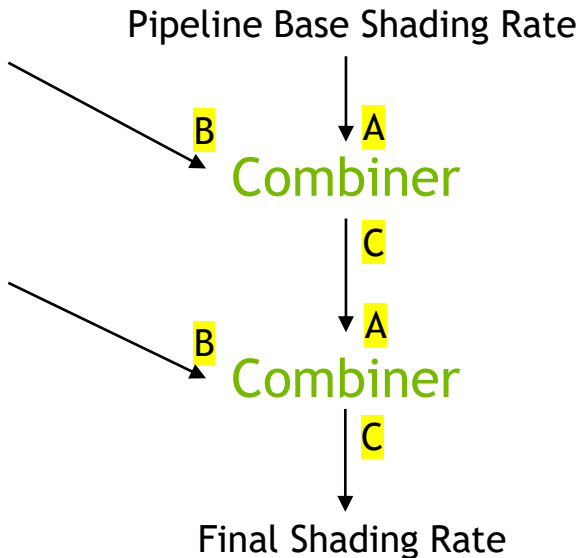
[ `SV_ShadingRate` ] (optional, read shading rate or set per-primitive shading rate in HLSL)

# DX12 API

## Using Combiners

Per-Primitive Shading Rate  
(*SV\_ShadingRate* VS/GS output)

Image-Based Shading Rate



Passthrough	$C = A$
Override	$C = B$
Higher quality	$C = \min(A, B)$
Lower quality	$C = \max(A, B)$
Relative	$C = A + B$ (in log2 space)

Where A, B, C are int2 shading rates

# DX12 API

## Query Device Capabilities

```
struct D3D12_FEATURE_DATA_D3D12_OPTIONS6
{
    // NVIDIA: True on Turing (4x2, 2x4, 4x4 supported)
    BOOL AdditionalShadingRatesSupported;

    // NVIDIA: False on Turing (support either per-prim rate or VP indexing)
    BOOL PerPrimitiveShadingRateSupportedWithViewportIndexing;

    // NVIDIA: Tier 2 on Turing
    D3D12_VARIABLE_SHADING_RATE_TIER VariableShadingRateTier;

    // NVIDIA: 16
    UINT ShadingRateImageTileSize;
}
```

# DX12 API

## Differences from DX11, Vulkan and OpenGL

DX12 allows setting per-primitive shading rate more easily  
(other APIs need to use multi-viewport)

DX12 offers fixed-function combiners for different shading rate sources  
(other APIs need to program the per-viewport shading rate LUT)

DX12 doesn't support hybrid supersampling yet (available in other APIs now)

DX12 has fixed coverage mask sample order and shading rate palette

# DX12 API

Further reading...

Spec: <http://aka.ms/vrsspec>

Microsoft's VRS GDC19 talks:

[DirectX: Boost Rendering Performance with Hardware Accelerated Variable Rate Shading \(VRS\)](#)

[Shawn Hargreaves](#) et al. Wednesday, March 20, 2019, 2:00pm - 3:00pm

[DirectX: Variable Rate Shading, a Deep Dive, Deriving Screen Space Shading Rate, Implementation Tips and Sparse Lighting \(Presented by Microsoft\)](#)

[Martin Fuller](#). Wednesday, March 20, 2019, 3:30pm - 4:30pm

The background is a dark, almost black, field filled with a complex network of thin, glowing green lines. These lines intersect at various points, creating a web-like structure. At many of these intersection points, there are small, bright green circular dots or nodes. Some of these dots have a soft, out-of-focus glow around them, making them stand out more prominently. The overall effect is one of a dynamic, interconnected system, possibly representing a neural network or a complex data structure.

**NVIDIA ADAPTIVE SHADING (NAS)**

**CONTENT ADAPTIVE SHADING**

# WHY ADAPTIVE SHADING?

## VRS and image quality

Variable rate shading introduces error to the results

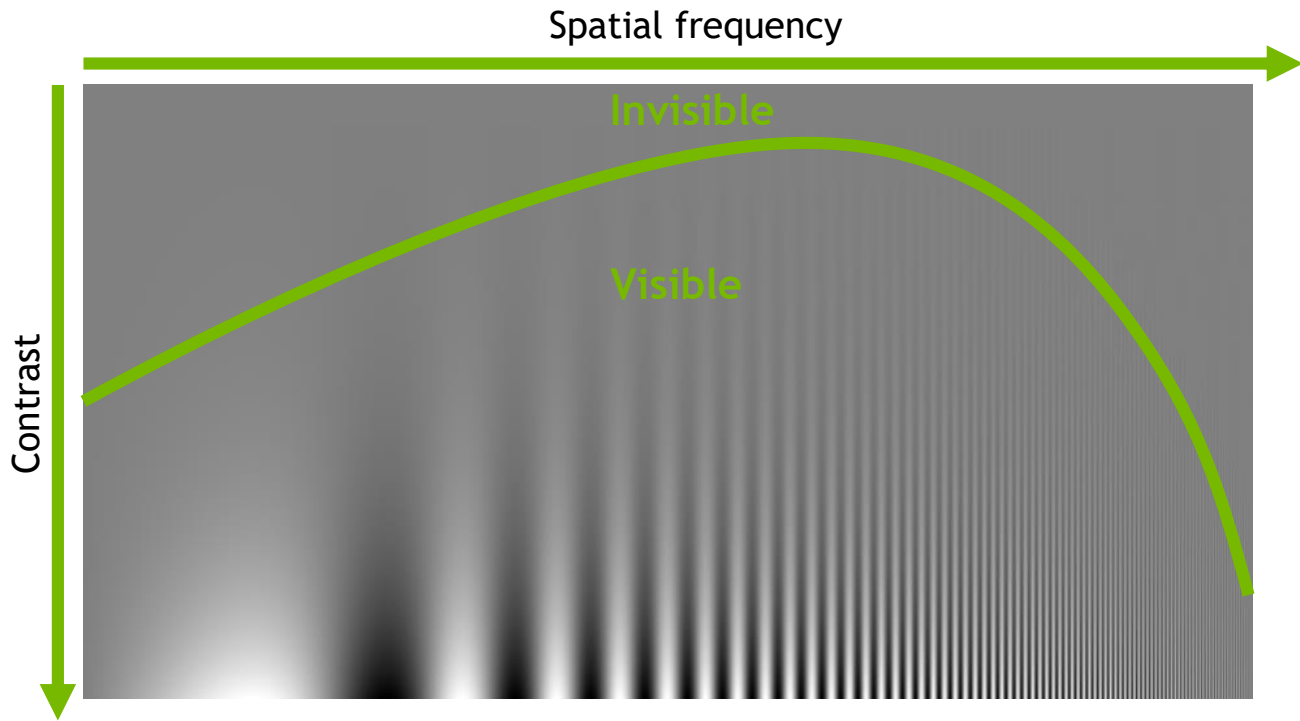
We don't want the error to be visible

Visibility of error depends on:

- Spatial frequency
- Contrast
- Motion

# ADAPTIVE SHADING

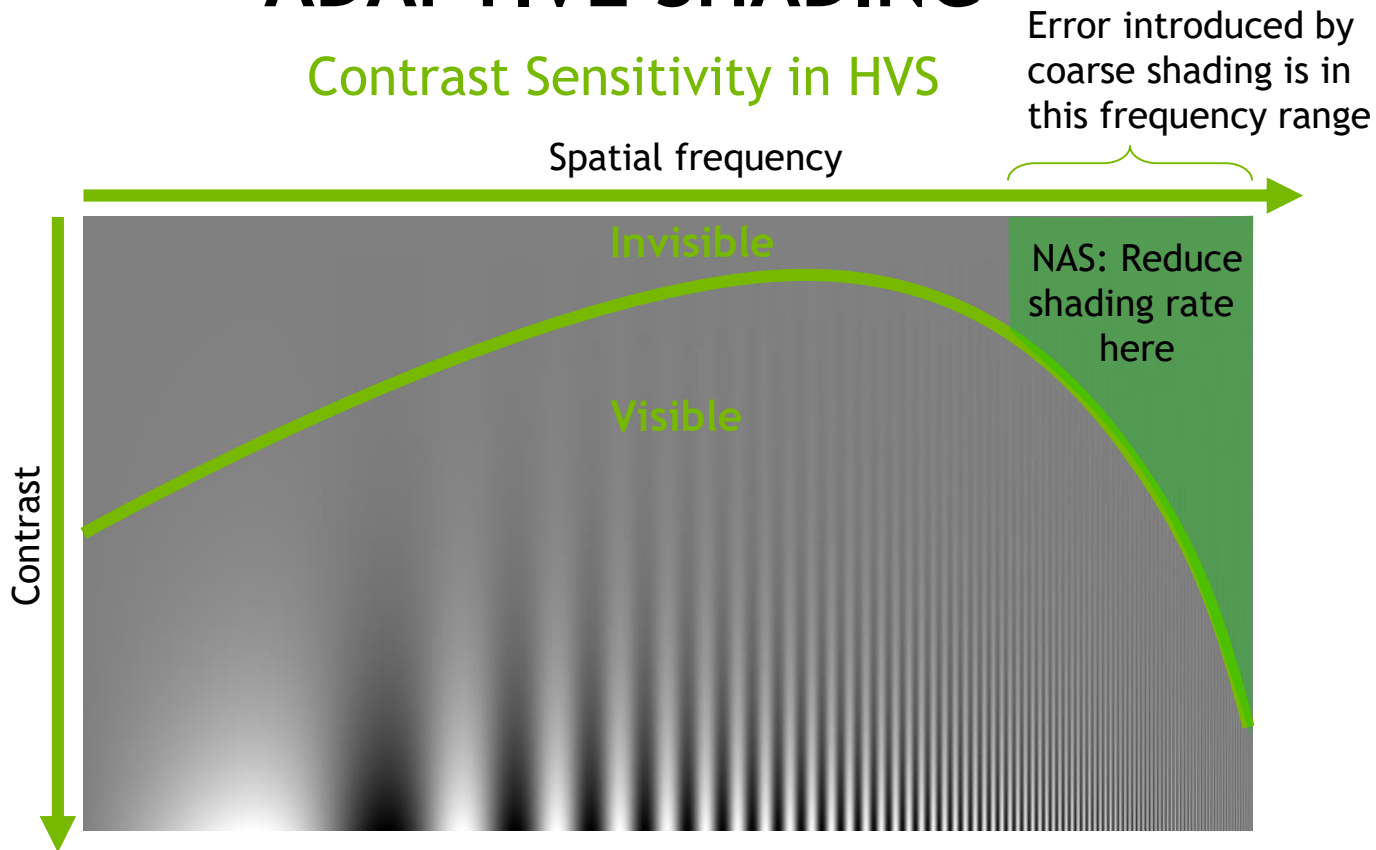
## Contrast Sensitivity in HVS



Campbell-Robson CSF chart  
Courtesy of [Izumi Ohzawa](#)

# ADAPTIVE SHADING

## Contrast Sensitivity in HVS



Campbell-Robson CSF chart  
Courtesy of [Izumi Ohzawa](#)

# ADAPTIVE SHADING

Works better in 4K and higher resolution!

Higher resolution -> Higher PPI

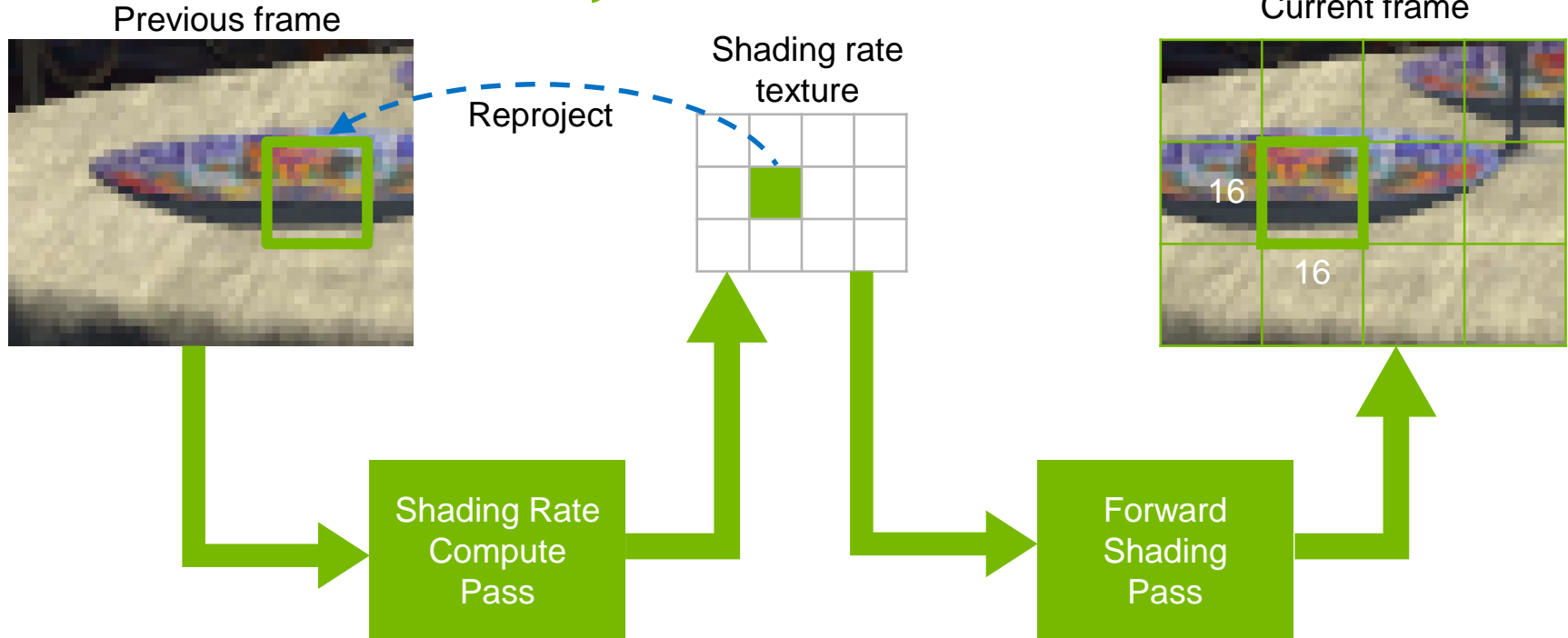
- > Details are in higher spatial frequency (lower sensitivity)

- > Larger headroom to enable VRS

Higher pixel-to-primitive ratio improves VRS performance

# ADAPTIVE SHADING FLOW

Analyze last rendered frame



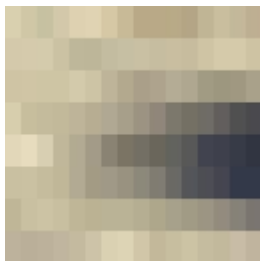
# CONTENT ADAPTIVE SHADING

How to predict the error?

16x16 tile  
from previous frame



1x1



1x2



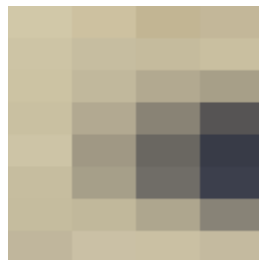
2x4



2x2



2x1



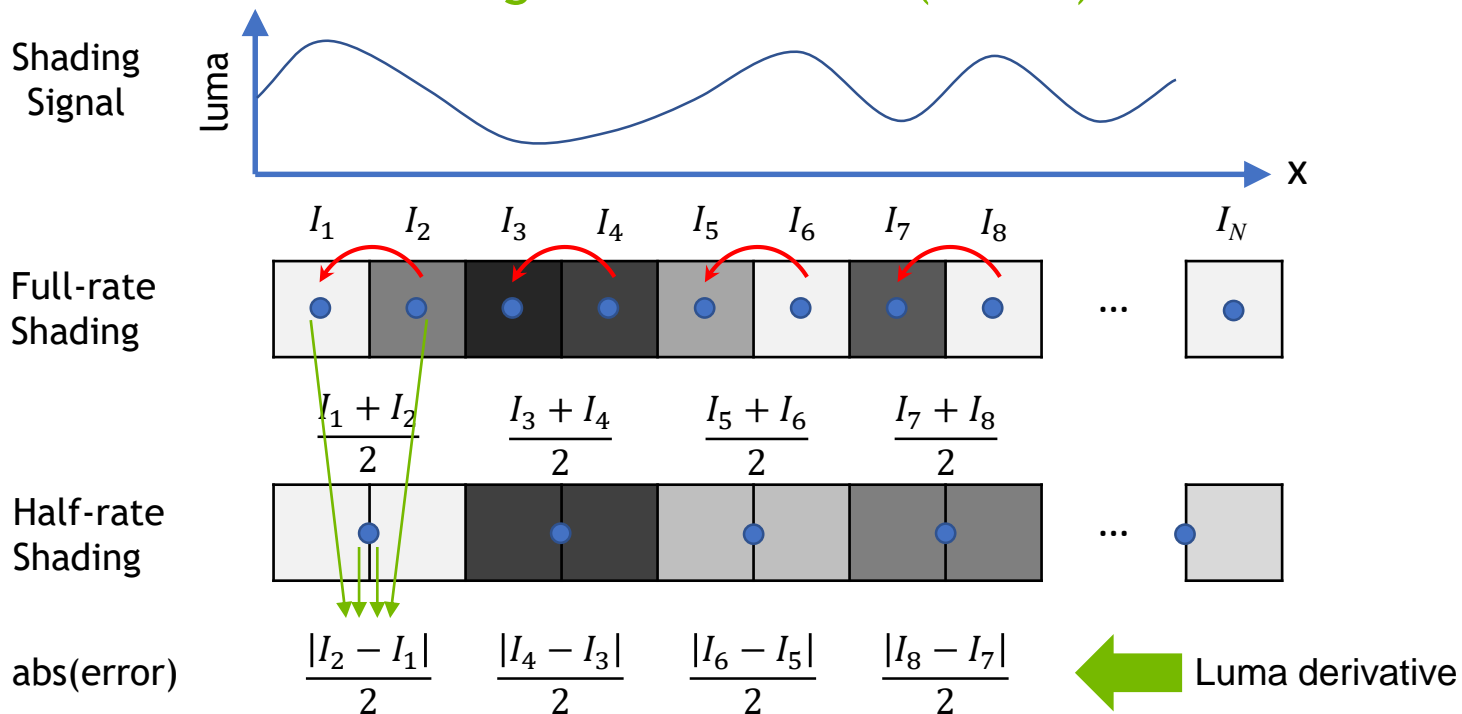
4x2



4x4

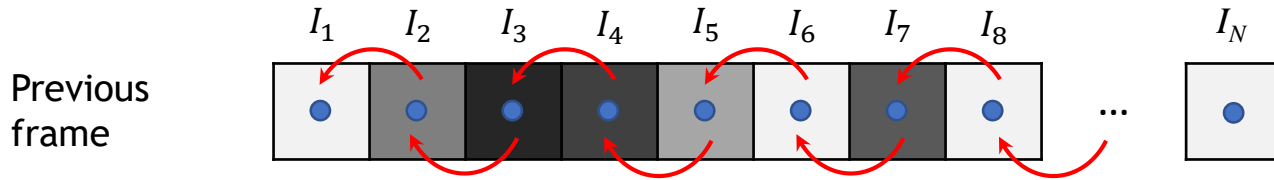
# CONTENT ADAPTIVE SHADING

Deriving the error in 1D (X or Y)



# CONTENT ADAPTIVE SHADING

Estimate the error per tile



Error is summed up per tile using an arithmetic mean, “mean squared error” or a “max”

We compute the derivative at all pixels, not just every other pixel

- Avoid unstable estimated error when image is shifted by 1 pixel
- More robust estimation when tile in previous frame was already shaded in lower rate

In 2D, we compute the derivative in both X and Y directions (for X, Y shading rates)

# CONTENT ADAPTIVE SHADING

Bounding error with a perceptual threshold

Enable half-rate shading (2x) when per-tile error  $E_H < \text{threshold}$

Using a fixed threshold is not sufficient!

# WEBER-FECHNER LAW



“The just-noticeable  
difference between two  
stimuli is proportional  
to the magnitude of the  
stimuli”

← Luminance

# CONTENT ADAPTIVE SHADING

## Bounding error with a perceptual threshold

Enable half-rate shading (2x) when per-tile error  $E_H < \text{threshold}$

Using a fixed threshold is not sufficient!

“Just Noticeable Difference” threshold:

$$\text{threshold} = \text{SENSITIVITY\_SCALE} * \text{BASE\_LUMINANCE}$$

Constant



Depends on the  
luminance level

# CONTENT ADAPTIVE SHADING

## Error of quarter-rate (4x coarse) shading

Quarter-rate shading needs a separate error estimator...

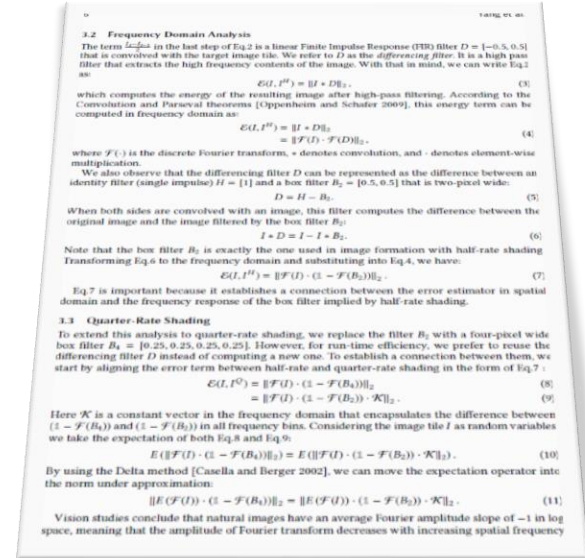
Approximate using luma derivative we just computed!

$$E_Q = 2.13 \cdot E_H$$

Detailed derivation can be found in our paper publishing in *ACM Symposium on I3D* this year (May 2019)

“Visually Lossless Content and Motion Adaptive Shading in Games” by Lei Yang, Dmitry Zhdan, Emmett Kilgariff, Eric Lum, Yubo Zhang, Matthew Johnson, and Henrik Rydgård

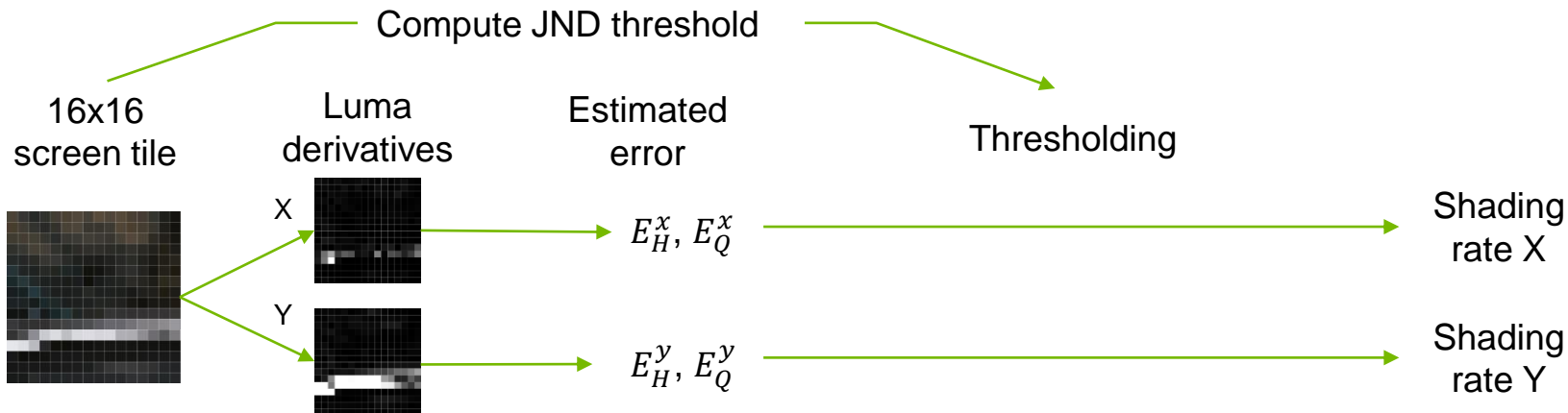
Use quarter-rate shading (4x) when  $E_Q < \text{threshold}$



# CONTENT ADAPTIVE SHADING

To summarize ...

For each 16x16 screen tile reprojected to previous frame:



The background is a dark, almost black, field filled with a complex network of thin, glowing green lines. These lines intersect at various points, creating a web-like structure. At many of these intersection points, there are small, bright green circular dots or nodes. Some of these dots have a soft, out-of-focus glow around them, making them stand out more prominently. The overall effect is one of dynamic energy and interconnectedness, typical of a digital or network-themed aesthetic.

**NVIDIA ADAPTIVE SHADING (NAS)**

**MOTION ADAPTIVE SHADING**

# THE EFFECT OF MOTION BLUR

## and why it helps VRS...

Motion blur hides the error caused by lowered shading rate

Two sources of motion blur:

- Engine simulated motion blur effect
- LCD display persistence blur

# LCD PERSISTENCE BLUR \*

IN REAL  
WORLD



ON THE  
SCREEN



# THE EFFECT OF MOTION BLUR

## and why it helps VRS...

Both types of motion blur reduce the high-frequency details on objects

Exactly where VRS error lies in!



# MOTION BLUR AND PERCEIVED ERROR

How motion blur hides VRS error

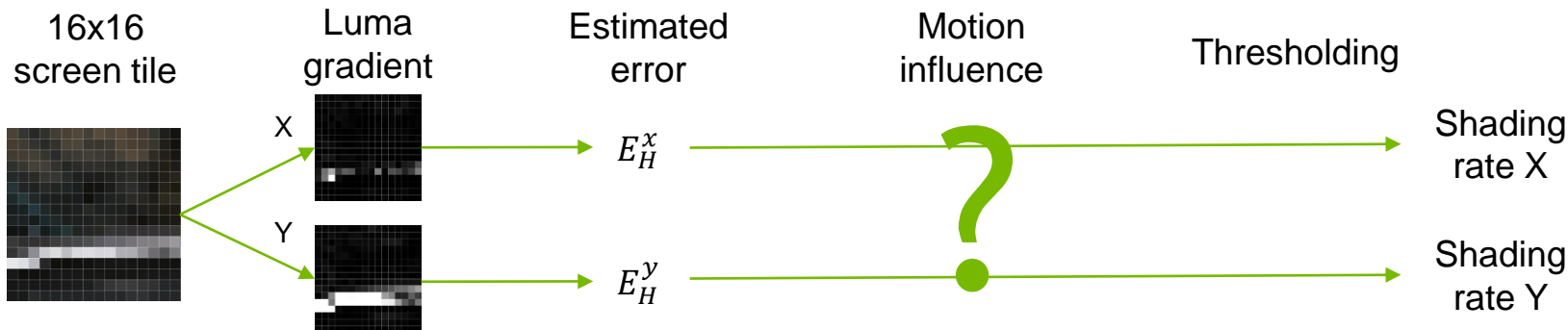


# MOTION ADAPTIVE SHADING

Add motion influence into adaptive shading

**Goal:** lower the shading rate in the presence of motion, without sacrificing quality

**Approach:** factor motion influence into the adaptive shading estimated error



# MOTION ADAPTIVE SHADING

## Motion-based VRS error estimate scaler

Motion scales down the estimated error per tile:

For half-rate:  $E_H(v) = b_H(v) \cdot E_H$

For quarter-rate:  $E_Q(v) = b_Q(v) \cdot E_H$

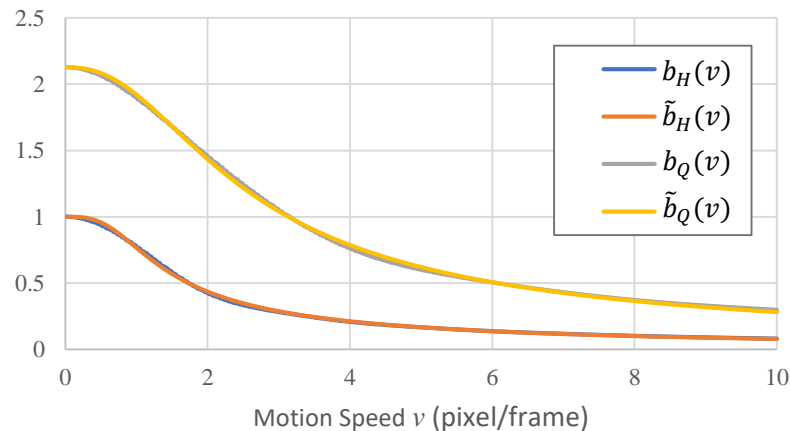
We compute  $b_H(v)$  and  $b_Q(v)$  numerically

Then fit two closed-form functions:

$$\tilde{b}_H(v) = \left( \frac{1}{1 + (1.05v)^{3.10}} \right)^{0.35},$$
$$\tilde{b}_Q(v) = 2.13 \left( \frac{1}{1 + (0.55v)^{2.41}} \right)^{0.49}.$$

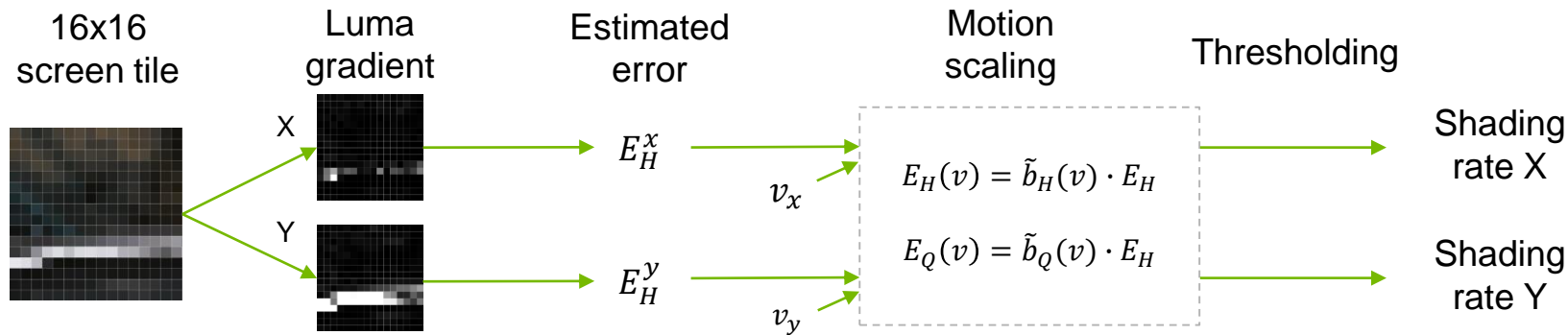
(Detailed derivation can be found in our I3D paper)

**This unifies motion and content adaptive shading**



# MOTION ADAPTIVE SHADING

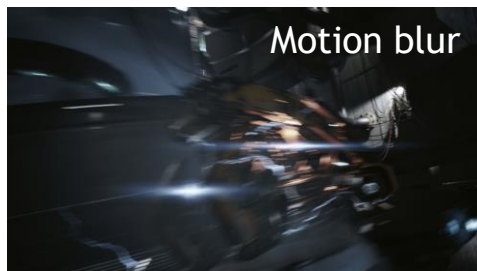
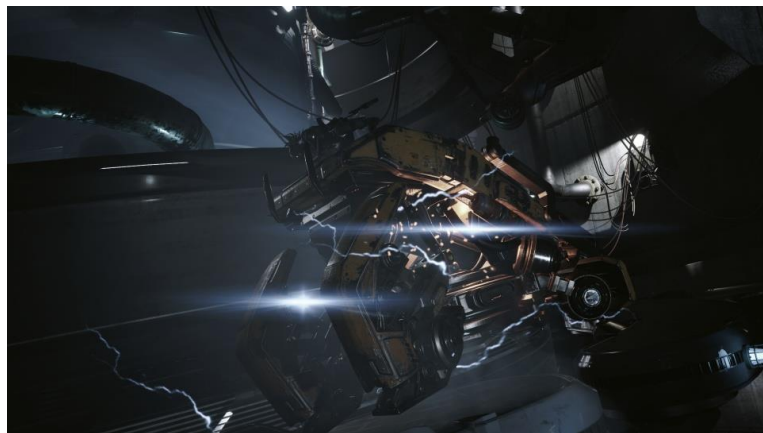
## Full pipeline



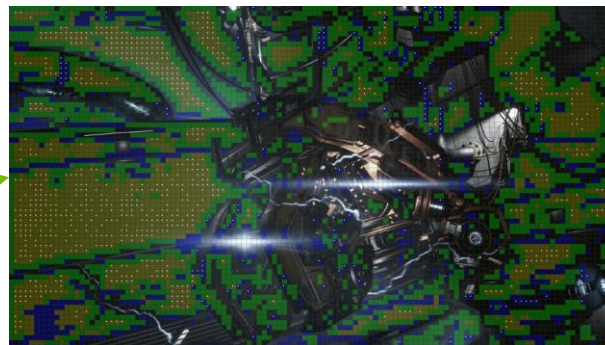
Per-tile motion speed  $v$  is the *minimum* motion vector across the tile  
(Use minimum motion because we need it to be conservative)

# MOTION ADAPTIVE SHADING

Effect on the shading rate pattern



Content  
Adaptive



Content +  
Motion  
Adaptive

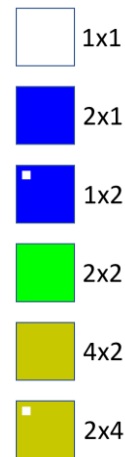
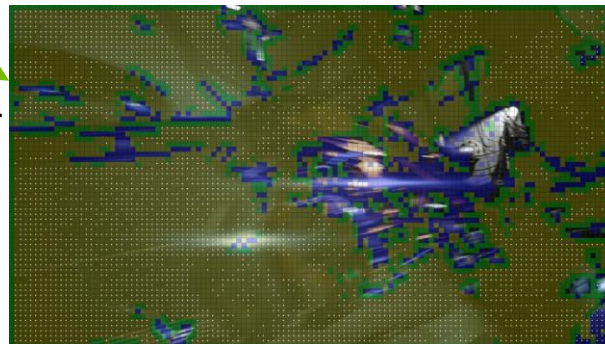


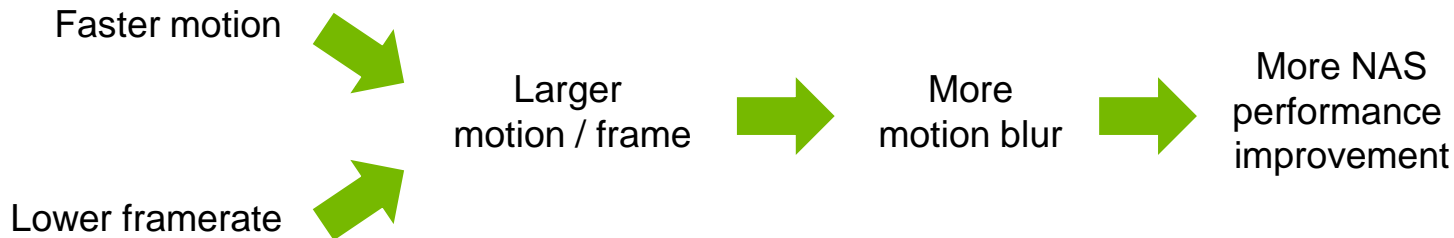
Image captured from "Infiltrator" demo rendered in Unreal Engine 4

# BENEFITS

## Motion adaptive shading

Motion adaptive shading delivers higher framerate **when it is needed**

Counteracts the unwanted slowdown due to wide motion blur filter in postprocessing





# NAS IMPLEMENTATION IN WOLFENSTEIN II: THE NEW COLOSSUS

# NAS IN WOLFENSTEIN II

Motivation: Extra Performance for free!

Based on VRS extension for Vulkan (**VK\_NV\_shading\_rate\_image**)

Average 2x perf boost on forward+ passes where applied (**2-20% saving in a 4K frame**)

Reap extra performance on a (already) heavily optimized engine (id Tech 6)

Offers performance for free without massive code changes and significant time investments (barely interferes with the engine code)

None or imperceptible image quality loss (strict IQ target!)

Opens **5K gaming** door wider...

NAS OFF

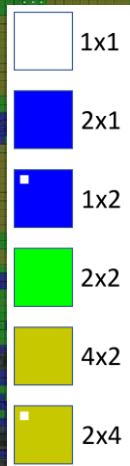
50 + 47

45/227

NAS ON

50 + 47

45/227



SHADING RATE  
VISUALIZATION

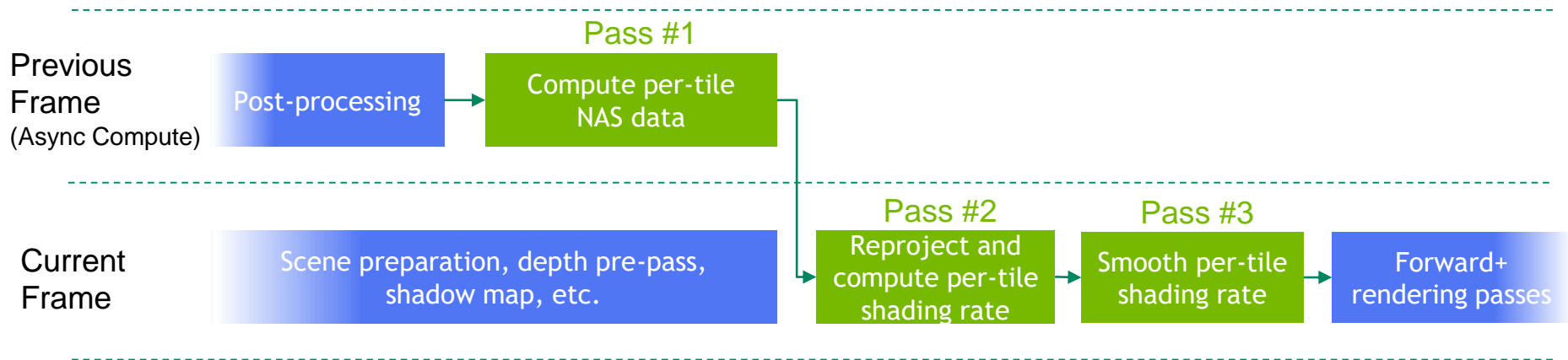
50 + 47

45/227

# NAS IN WOLFENSTEIN II

## Overview

Optimized pipeline for applying NAS (3 passes):



# MINIMIZING OVERHEAD

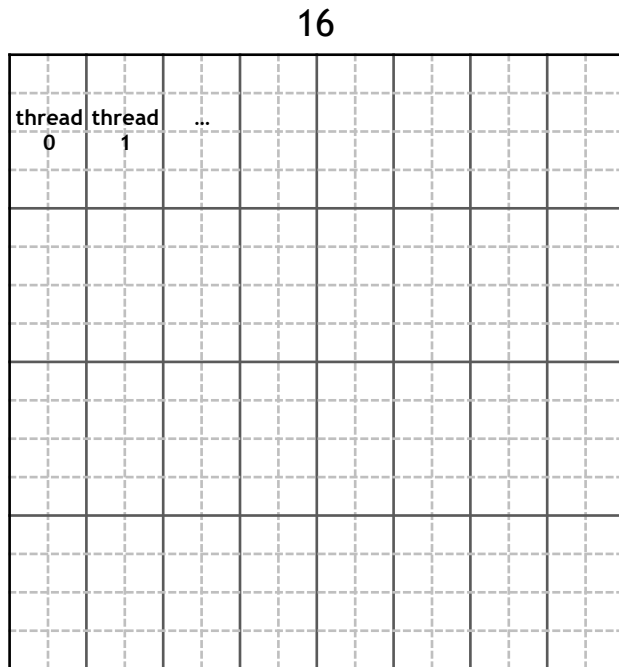
## Architecture-aware optimization

Shading rate calculation is a data reduction task  
(16x16 to 1, using min, max, or add)

- Process 16x16 tile with a **single warp** (32 threads)
- Each thread manages a 2x4 block of pixels
- Thread reduction 2x4 -> 1 first (on registers)
- Then intra-warp atomics (optimized into warp shuffles) 16

Result: Minimal performance overhead of additional passes  
(0.2ms @ 4K on RTX 2080 Ti)

**Async compute is your friend!**



# PASS 1: COMPUTE NAS DATA

## Luma derivatives

Compute per-pixel luma (input: final color is in LDR sRGB \*):

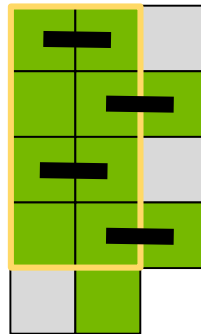
```
luma = dot(color * color, float3(0.299, 0.587, 0.114))
```

Compute pixel luma derivatives in X and Y

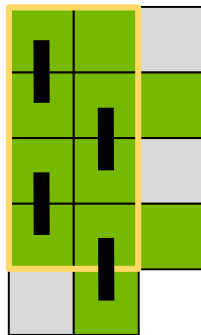
We save a few texture fetches and half of math ops by computing derivatives in an interleaved fashion

\*Note: Luma should be computed after tone-mapping, converted to linearRGB.  
Conversion is approximated by  $\text{color}^2$

dx:



dy:



# PASS 1: COMPUTE NAS DATA

## Correction to derivatives (Weber-Fechner Law)

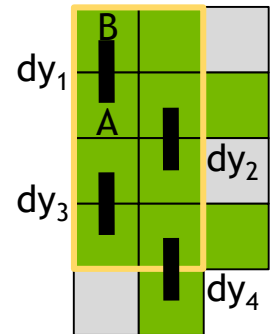
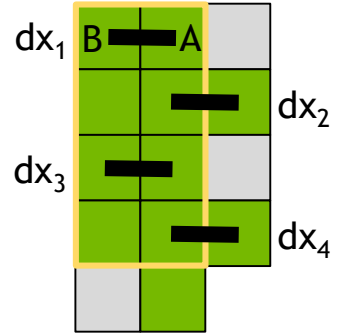
$$\text{LumaDeriv} = \text{abs}(A - B) / [\text{min}(A, B) + \text{BRIGHTNESS\_SENSITIVITY}]$$

Where:

- A and B - luma in neighboring pixels
- BRIGHTNESS\_SENSITIVITY corrects sensitivity in low luminance:

$\text{BRIGHTNESS\_SENSITIVITY} = C \cdot [1.0 - \text{satrate}(L * 50.0 - 2.5)]$ , where

- C - configurable constant
- L - min luma in 2x4 pixel block



# PASS 1: COMPUTE NAS DATA

## Compute per-tile luma derivatives

Each thread computes a sum of X/Y absolute derivatives over 2x4 pixel block

Use atomicAdd over the warp to get the sum of all 16x16

```
float4 dx = float4(dx1, dx2, dx3, dx4);
float4 dy = float4(dy1, dy2, dy3, dy4);
uint sumDx = uint( dot( dx, float4(255.0, 255.0, 255.0, 255.0) ) );
uint sumDy = uint( dot( dy, float4(255.0, 255.0, 255.0, 255.0) ) );

atomicAdd(groupAvgDxDy, (sumDy << 16) | sumDx);

float avgDx = float(groupAvgDxDy & 0xFFFF) / 65280.0;
float avgDy = float(groupAvgDxDy >> 16) / 65280.0;
```

Use atomics sparingly:

16x16 (pixels) \* 255  
(max derivative)  
< 65535 (0xFFFF)

Two atomic adds can  
be merged into one!

# PASS 1: COMPUTE NAS DATA

## Additional data: Material Properties

Find max smoothness and specular reflectance ( $R_f0$ ) per tile  
(for use in pass 2)

**Why?**

“NAS-TAA-Motion Blur complication”

(will be covered later)

For material properties, fetching 4 samples instead of 8 works well and improves performance:



# PASS 2: COMPUTE SHADING RATE

## Data reprojection

Reproject data from previous frame to the current to avoid 1 frame data latency

Motion



A common case where 1 frame data latency can cause visible artifacts

# PASS 2: COMPUTE SHADING RATE

## Data reprojection

Compute minimum motion vector per 16x16 screen tile:

- Find furthest point per tile from z-prepass
- Use camera matrices to reproject (object motion not considered)

Fetch NAS data to the current frame with linear interpolation

Motion vector is also used to determine motion influence

# PASS 2: COMPUTE SHADING RATE

## Image quality improvement

Specular aliasing in VRS can look bad, especially with postprocessing in HDR

- Large, blocky specular got smeared
- Lower sample count causing temporal artifacts

Two related practical problems we need to tackle...

# PASS 2: COMPUTE SHADING RATE

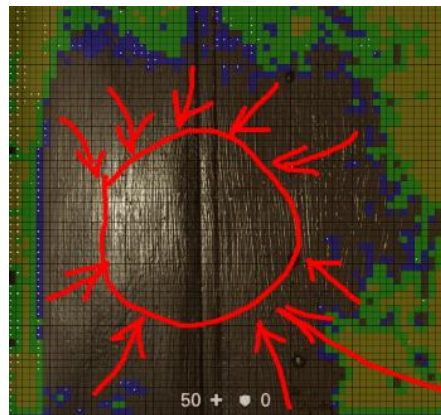
## A tale of two problems - #1

### Problem #1 Shimmering in slow motion

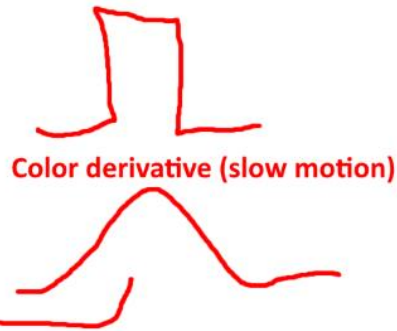
Pass 1 only has post-motion-blur input, causing it to underestimate derivatives in motion. The result is visible shimmering on reflections especially in slow motion

### Solution:

Increase error sensitivity under slow motion (use mixture of smoothness and  $Rf0$ )



Color derivative (no motion)



Color derivative (slow motion)

# PASS 2: COMPUTE SHADING RATE

## A tale of two problems - #2

### Problem #2 Motion blur + TAA/NAS feedback latency

When in fast motion, shading rate is lowered

When motion stops, NAS takes 3-4 frames to return to higher shading rate, due to unwanted feedback and TAA update latency

### Solution:

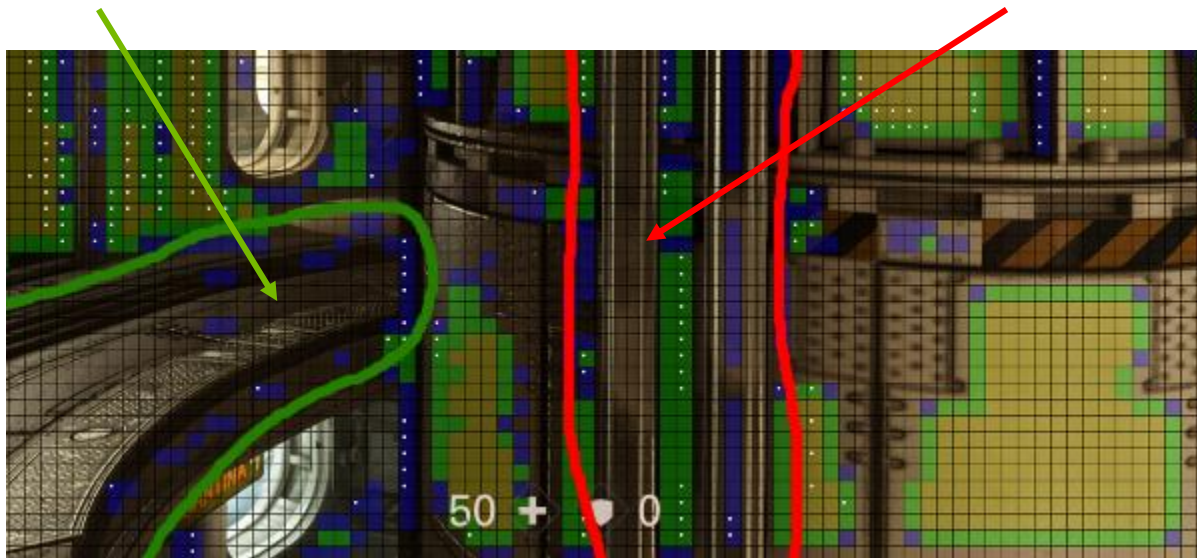
Increase shading rates on smooth metallic surfaces (using smoothness and  $Rf0$ )

# PASS 2: COMPUTE SHADING RATE

## A tale of two problems - #2

Mid-frequency specular can get reduced shading rates when not in motion

High-frequency specular always gets 1x1 shading



# REMAINING PROBLEM

Pass 2 outputs shading rate, but...

Shading rates can oscillate a lot!

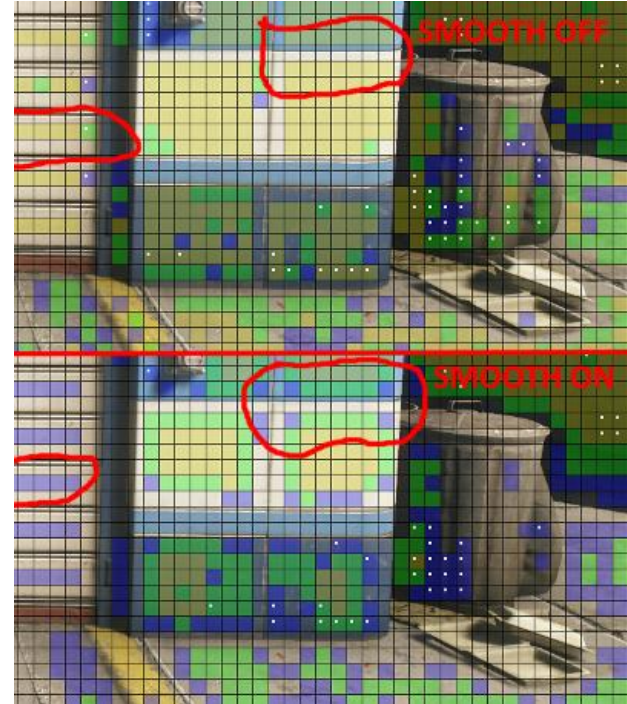
Usually happens at fine/coarse boundaries, esp. when shading rate in neighboring tiles differ by more than 1 level

## Solution?

Temporal stabilization? **No**. It introduces latency

**Spatial smoothing the shading rate? Yes!**

- -1x1
- -2x1
- -2x2
- -4x2



# PASS 3: SMOOTH SHADING RATE

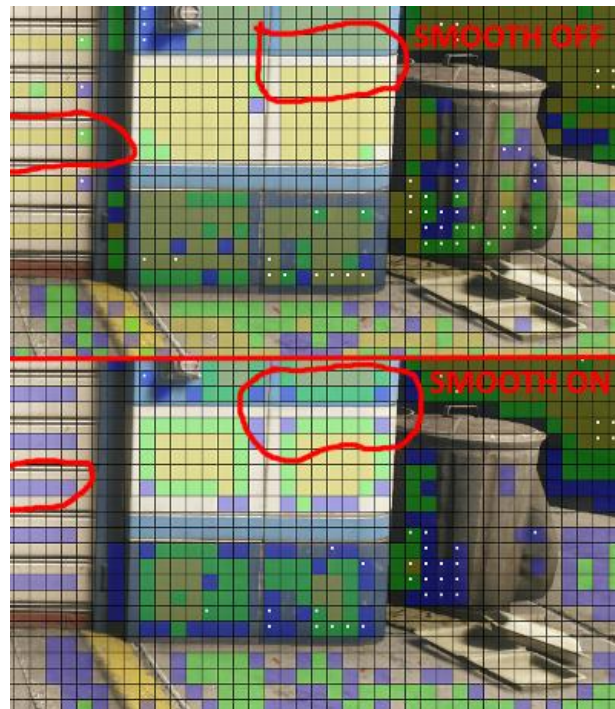
## Conservative spatial blurring

A simple 3x3 blur turned out to be a bad idea, because coarse tiles can lead astray fine ones

Instead we blur by only letting fine tiles blend into coarse ones

A tiny pass that post-processes the SR texture

- -1x1
- -2x1
- -2x2
- -4x2



# NAS IN WOLFENSTEIN II

## Lessons learned

HDR motion blur made adaptive shading harder

Avoid latency as much as you can!

Watch out for transition latency due to TAA/NAS feedback

Avoid introducing temporal smoothing to shading rate

TAA can be adapted to improve VRS blockiness and aliasing (not used due to extra complexity)

1080p requires a bit more tuning, due to large pixels and higher contrast (error) sensitivity



# WOLFENSTEIN II DEMO

# SUMMARY

## VRS and NAS

VRS: is a new GPU feature available on Turing and supported by major APIs

Very little change in engine code to achieve steady performance boost

API offers flexible options to configure shading rate

NAS: to unlock extra performance without perceivable quality loss

2x average, up to 5x performance gain in forward shading passes

Apply perception theory to best locate and avoid overshading

Help achieve best performance in motion

# SPECIAL THANKS!

**Jim Kjellin** and Machine Games - for source code guidance and familiarization with engine technologies

**Henrik Rydgard** - for VK coding and valuable ideas

**Swaroop Bhonde** - for excellent diagrams and slides that we adapted from

... and many staffs in NVIDIA GPU Architecture, ASIC, DevTech and software/drivers team - for their valuable contribution to this project



# QUESTIONS?



Lei Yang | [leyang@nvidia.com](mailto:leyang@nvidia.com)

Dmitry Zhdan | [dzhdan@nvidia.com](mailto:dzhdan@nvidia.com)

Matthew Johnson | [matthewj@nvidia.com](mailto:matthewj@nvidia.com)