# The Forge Interview

Christian Robles

November 2023

# Contents

- Feedback
- Project
- Q&A

# Feedback

# Feedback

## Framework Design

- Thin, 1 abstraction away from underlying type
- DX-like, easy to learn with Vulkan background
- Function linking system easy to use/extend, minorly inconvenient to find implementations

# Feedback

“Render Targets”/addRenderTarget

- Very convenient for common use cases
- Internally complex
- Overrides format choice (R8_UINT -> R8_TYPELESS)
- Expected this to be handled by ResourceLoader

Transition barriers, descriptor sets, pipelines, root signatures, buffers, shaders…

- Easy to use with ample examples
- Might have had similar experience if I’d had to modify

# Feedback

## FSL

- FSL generally easy to use but coarse documentation makes small syntax issues slow to remedy. Similar to HLSL but not exact
- Broad set of examples covers most needs
- Wave Ops feature set unclear – WaveActiveMax but not WaveActiveMin, WaveGetLaneIndex but not WaveGetLaneCount

# Feedback

- Nice to have: Clang format
- Subjective: Cmake, Handle system

# Project

# Project Overview

- Scoping/topic selection
- NAS Theory
- NAS Implementation
- Optionals
- Issues

# What to make?

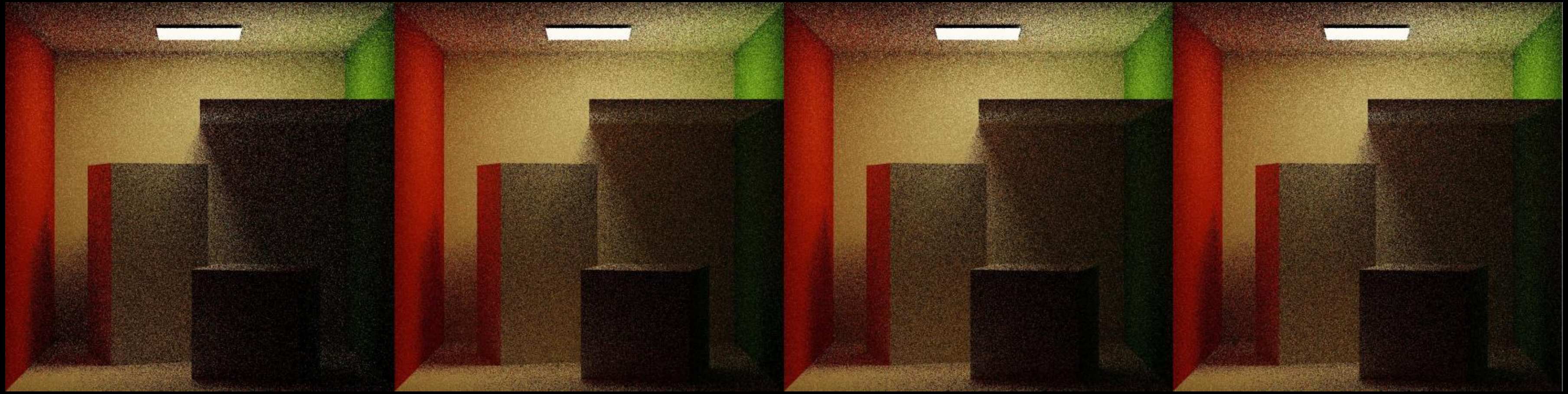
- Coverage over common systems (shaders, pipelines, UI, etc)
- Aim for 1 week ramp up + 2 weeks work + 1 week flex/wrap up
- Focus on what I know, implement something new

# What I was working on

- During MSc: light transport algorithms -> purpose-built research-oriented CPU path tracers
- Wanted to build a GPU rendering platform to use as testbed
    - Multi year project
    - Performant architecture before focusing on effects

# Interest in VRS

- Topical similarities to Directed Research project



- Works with visibility buffer
- Established industry support implies broad hardware support
  - (Wolfenstein II, Gears 5, CoD MW(2020))

# Research

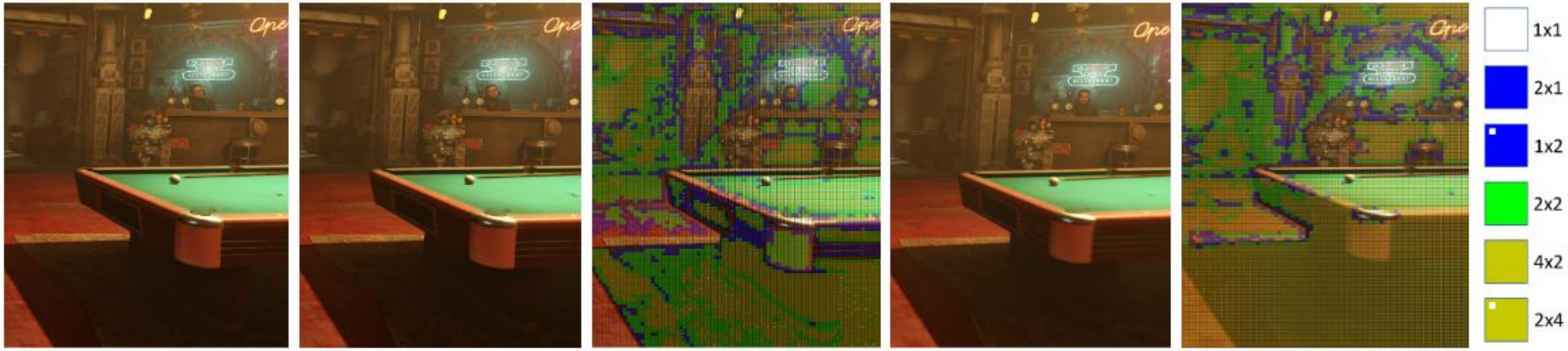
- Tier 2 Variable Rate Shading in Gears 5, Ms Game Dev 2021 https://www.youtube.com/watch?v=-exWLpgnOJ4
- Software-based Variable Rate Shading in Call of Duty: Modern Warfare (2020) https://research.activision.com/publications/2020-09/software-based-variable-rate-shading-in-call-of-duty--modern-war

➢Visually Lossless Content and Motion Adaptive Shading in Games, Ley Y. et al, NVIDIA 2019 http://leiy.cc/publications/nas/nas-pacmcgit.pdf

- Software VRS with Visibility Buffer Rendering, John Hable 2021 http://filmicworlds.com/blog/software-vrs-with-visibility-buffer-rendering/

# Nvidia Adaptive Shading (NAS) Quick View

- Tier-2 Hardware VRS
- Goal: improve performance with no loss in perceived quality



(a) Full rate (stationary)

(b) Adaptive (stationary)

(c) Shading Rate (stationary)

(d) Adaptive (in motion)

(e) Shading Rate (in motion)

# Why Hardware VRS

Pros:

- Minimal changes to render pipeline
- Opportunity to extend SDK
- No existing hardware VRS example

Cons:

- Have to update platform dependencies (Win SDK version, command list version)

# Why NAS

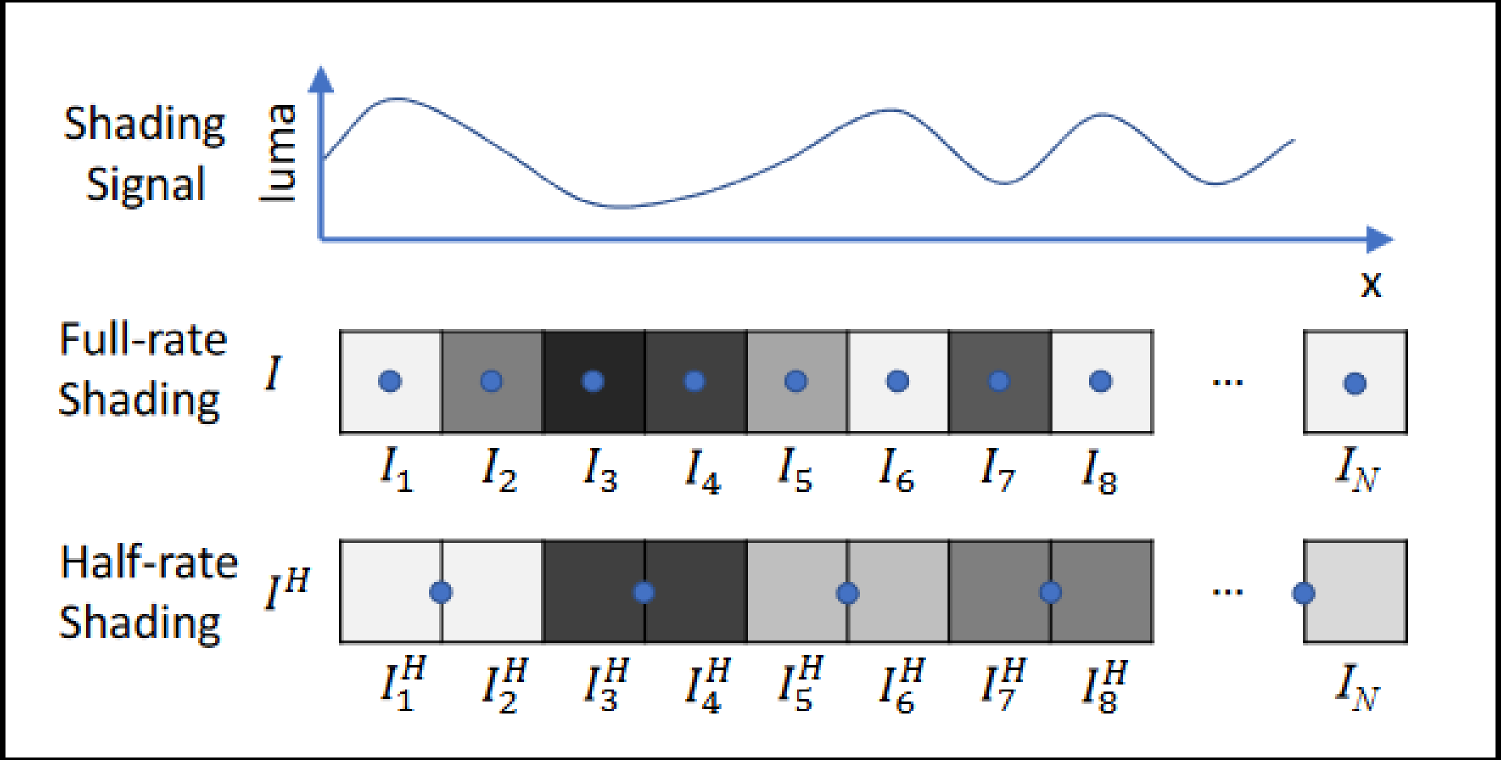
- Spending “error budget” clever approach
- Multifaceted – image-based and motion-adaptive shading
- Efficient – optimized thread group assignment and closed-form error equations
- Publication & reference repo

# NAS Introduction

Goal: maximize shading rate reduction while keeping error below perceptible threshold

1. Analyze previous frame with loss estimator
2. Predict error in current frame under reduced shading rates and motion velocity
3. Pick lowest shading rate under perception threshold

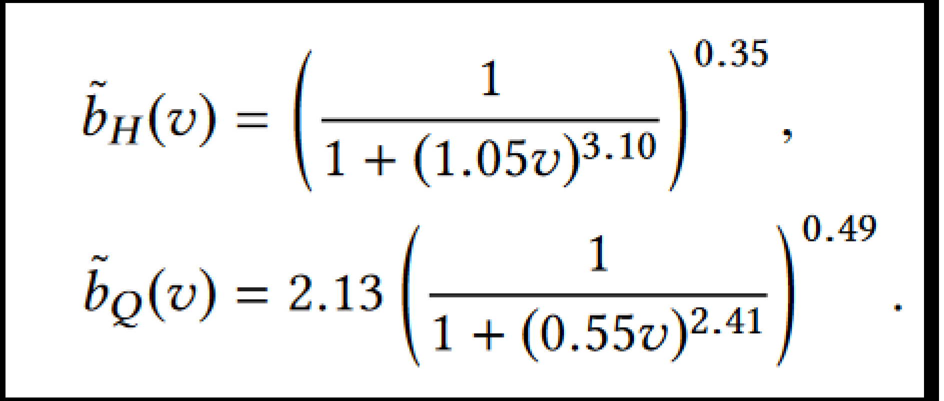
# Loss Estimator


Shading Signal
luma
x
Full-rate Shading
$I$
$I_1$ $I_2$ $I_3$ $I_4$ $I_5$ $I_6$ $I_7$ $I_8$ … $I_N$
Half-rate Shading
$I^H$
$I_1^H$ $I_2^H$ $I_3^H$ $I_4^H$ $I_5^H$ $I_6^H$ $I_7^H$ $I_8^H$ … $I_N$

- Block error can be evaluated using L1 (average absolute), L2 (RMSE) , or L∞ (max) norms
- Paper uses L2 to simplify derivation, implementation uses L∞ for fastest computation

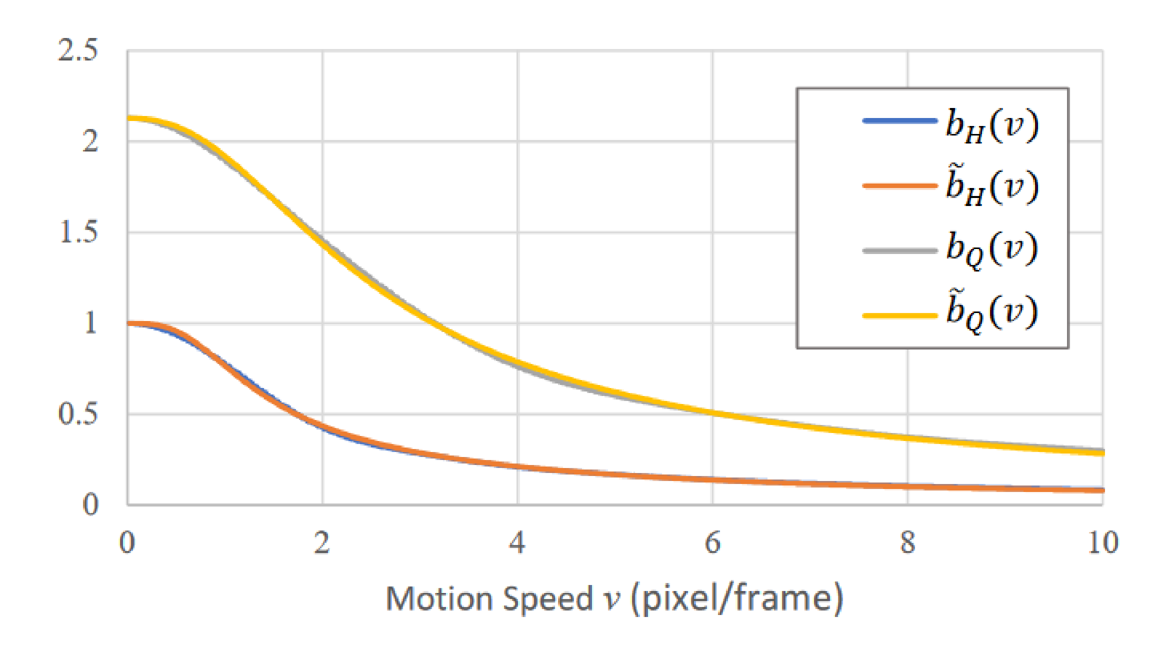
# Motion Adaptation

- Motion reduces perceived error
- Velocity vector (previous frame reprojection)
- Closed-form motion error scaling parameters derived from frequency analysis

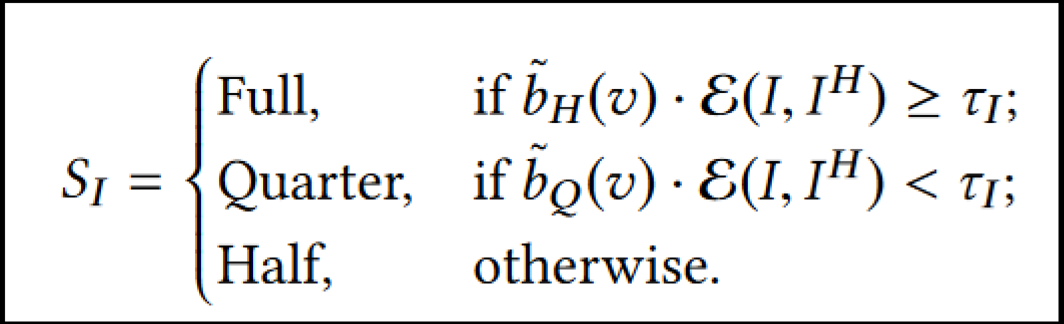


$$\tilde{b}_H(v) = \left( \frac{1}{1 + (1.05v)^{3.10}} \right)^{0.35},$$

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



# Shading Rate Selection



$$S_I = \begin{cases} \text{Full}, & \text{if } \tilde{b}_H(v) \cdot \mathcal{E}(I, I^H) \geq \tau_I; \\ \text{Quarter}, & \text{if } \tilde{b}_Q(v) \cdot \mathcal{E}(I, I^H) < \tau_I; \\ \text{Half}, & \text{otherwise.} \end{cases}$$

- Tuning parameters
  - Brightness Sensitivity – raises average luma (more reduction in dim regions)
  - Error sensitivity – error threshold τ
  - Motion sensitivity – increases motion scaling

# Implementation - Framework

- D3D12
  - Added cmdSetShadingRateImage
  - Added gpu setting query for VRS tile size
  - Updated addRenderTarget to allow shading rate images by texture creation flag

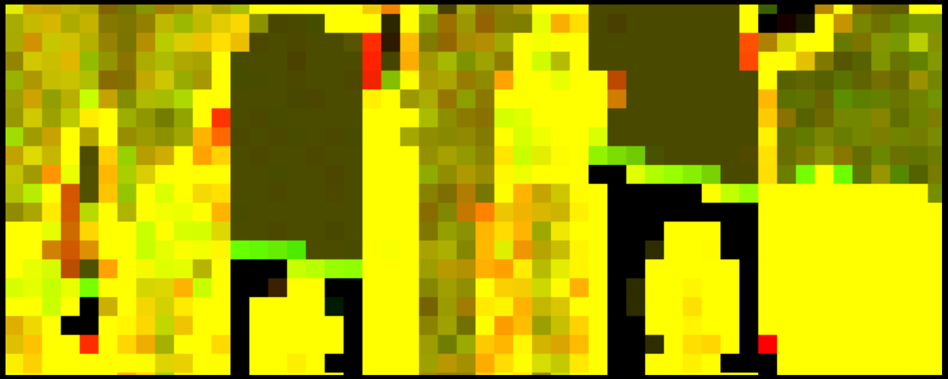
# Implementation

- Uber pixel shader -> single threaded compute shaders -> thread groups & wave ops

- NAS Data Surface
- Shading Rate Image
- Shading Rate Overlay
- Present Debug View

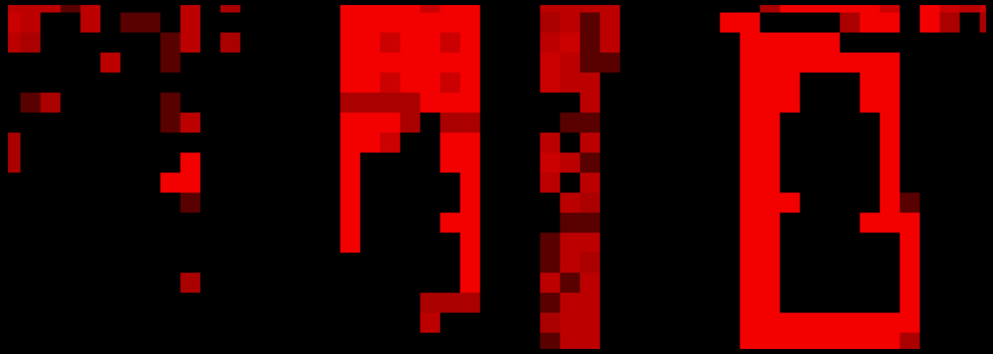
# Implementation

- NAS Data Surface Compute Shader
  - Performed at end of current frame
  - Inputs: current frame, brightness sensitivity
  - 1 dispatch per VRS tile
  - 8*4 threads * 8 samples per thread = 256 samples = 16 * 16 block
  - Computes estimated Qtr/Half rate block error from last frame result
  - Outputs: float2 X,Y error estimate UAV



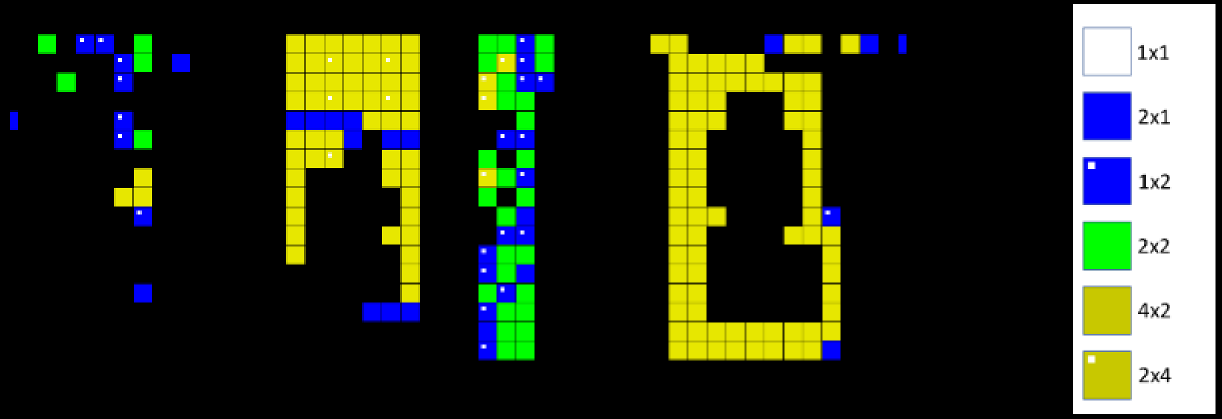
# Implementation

- Shading Rate Compute Shader
  - Performed just before shading pass
  - Inputs: NAS Data Surface, Depth Image, reprojection params, sensitivities
  - 1 dispatch per VRS tile
  - 8*4 threads * 8 samples per thread = 256 samples = 16 * 16 block
  - Computes motion adaptation and checks adapted error against threshold
  - Outputs: uint shading rate value UAV



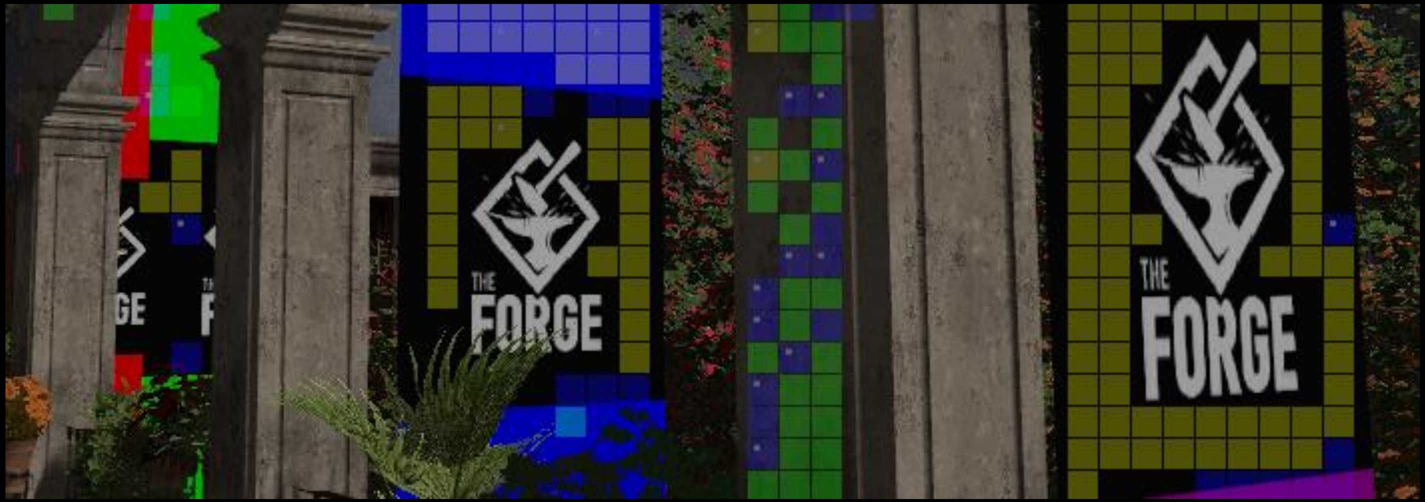
# Implementation

- Shading Rate Overlay Fragment Shader
  - Performed immediately after shading rate image
  - Inputs: Shading rate image
  - Visualizes shading rate



# Implementation

- Present Shader
  - Extended to view VRS debug images
  - Inputs: NAS data, Shading Rate Image, Shading Rate Overlay, debug view mode



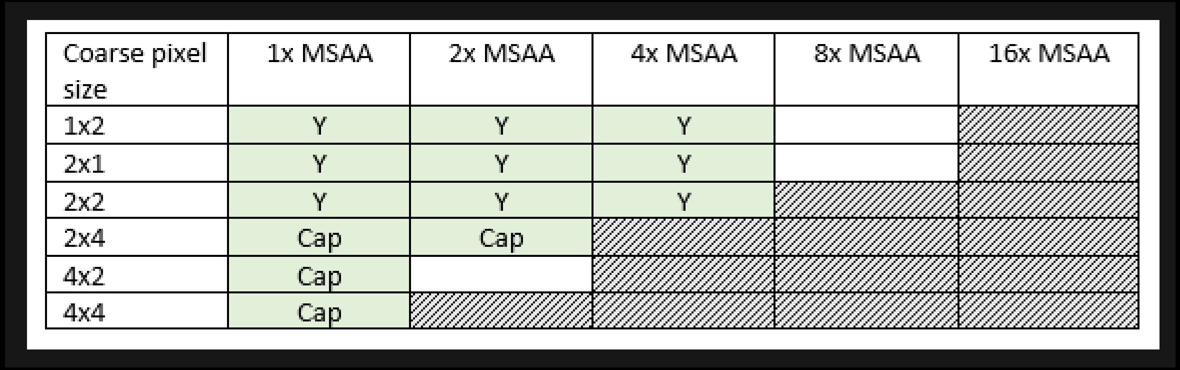
| | |
|---|---|
| □ | 1x1 |
| 🟦 | 2x1 |
| 🟦 | 1x2 |
| 🟩 | 2x2 |
| 🟨 | 4x2 |
| 🟨 | 2x4 |

# Optionals

- Shading rate smoothing
- Error stabilization (flicker reduction)
- Material based shading rate adaptation (aliasing from bright specular highlights)

# Issues

- No recommended values for parameters or recommendations on how to select values
- NAS data surface produces wrong error at partially covered tiles – many samples eval. 0 -> very small average -> boosts error estimate
- MSAA support matrix not honored in VRS rate selection



| Coarse pixel size | 1x MSAA | 2x MSAA | 4x MSAA | 8x MSAA | 16x MSAA |
|---|---|---|---|---|---|
| 1x2 | Y | Y | Y |  | ▨ |
| 2x1 | Y | Y | Y |  | ▨ |
| 2x2 | Y | Y | Y | ▨ | ▨ |
| 2x4 | Cap | Cap | ▨ | ▨ | ▨ |
| 4x2 | Cap |  | ▨ | ▨ | ▨ |
| 4x4 | Cap | ▨ | ▨ | ▨ | ▨ |

# Results/Demo

# Q&A

Thank you!