Microsoft

Microsoft Game Stack Live

Microsoft

Tier 2 Variable Rate Shading in Gears

Chris Wallis
Rendering Engineer

THE COALITION

Agenda

Introduction

Implementation

Results

Integration

Variable Rate Shading

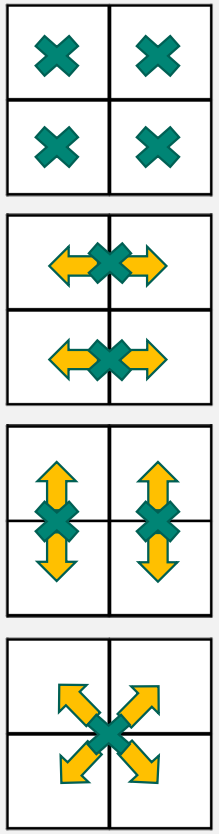
Variable Rate Shading (VRS) allows you to specify the resolution to run the pixel shader

4 types of shading rate:

- 1x1 (default, this is what happens if VRS is off)
- 1x2
- 2x1
- 2x2

Optional shading rates:

- 2x4/4x2/4x4



VRS in Gears Tactics for PC

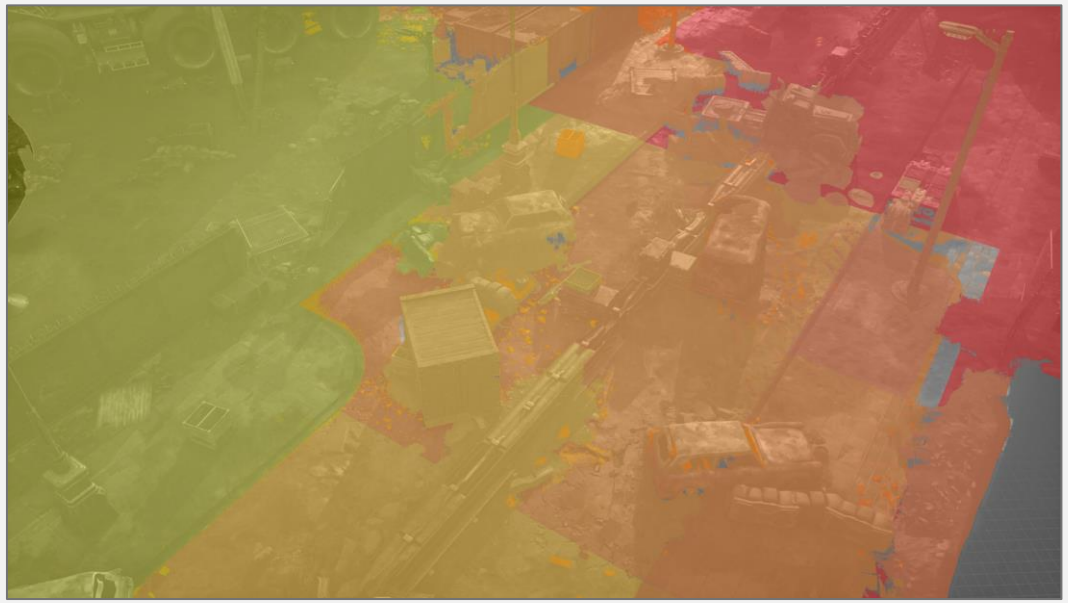
Tier 1 VRS

Specify shading rate per-draw

Prioritized intel

DX12 blog post with more details:

[Iterating on Variable Rate Shading in Gears Tactics]



Gears on Xbox Series X|S

Heavy-hitting GPU features

- Ultra PC feature set
- Screen Space Global Illumination
- Contact Shadows
- 60FPS cinematics

Wanted to keep dynamic resolution scaling high without compromising visuals

Tier 1 VRS was too aggressive and didn’t play nicely with dynamic resolution

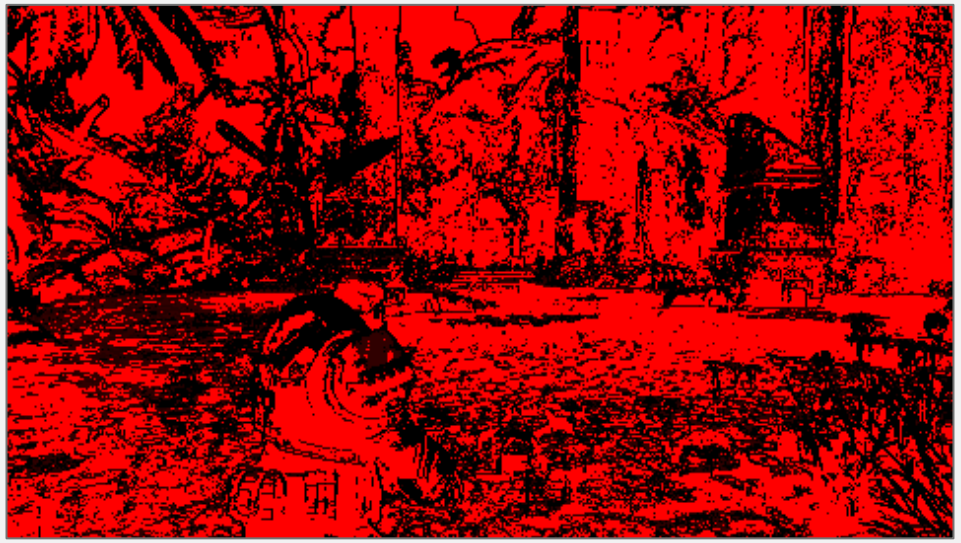
Tier 2 VRS

Shading rate specified in a screen-space texture

Shading rate stored in coarse "VRS tiles" of either 8x8 or 16x16 depending on HW

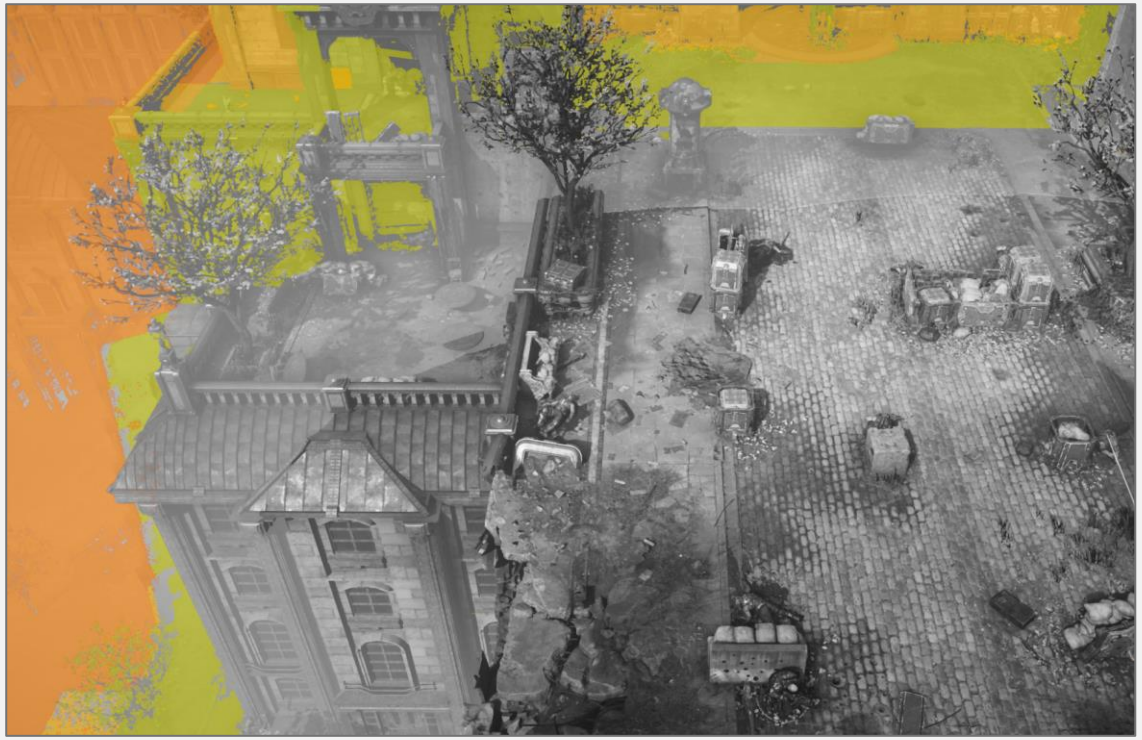
I.E. HW with 8x8 tiles and a 4k Render Target would be paired with a 480p shading rate texture

4k Scene Color

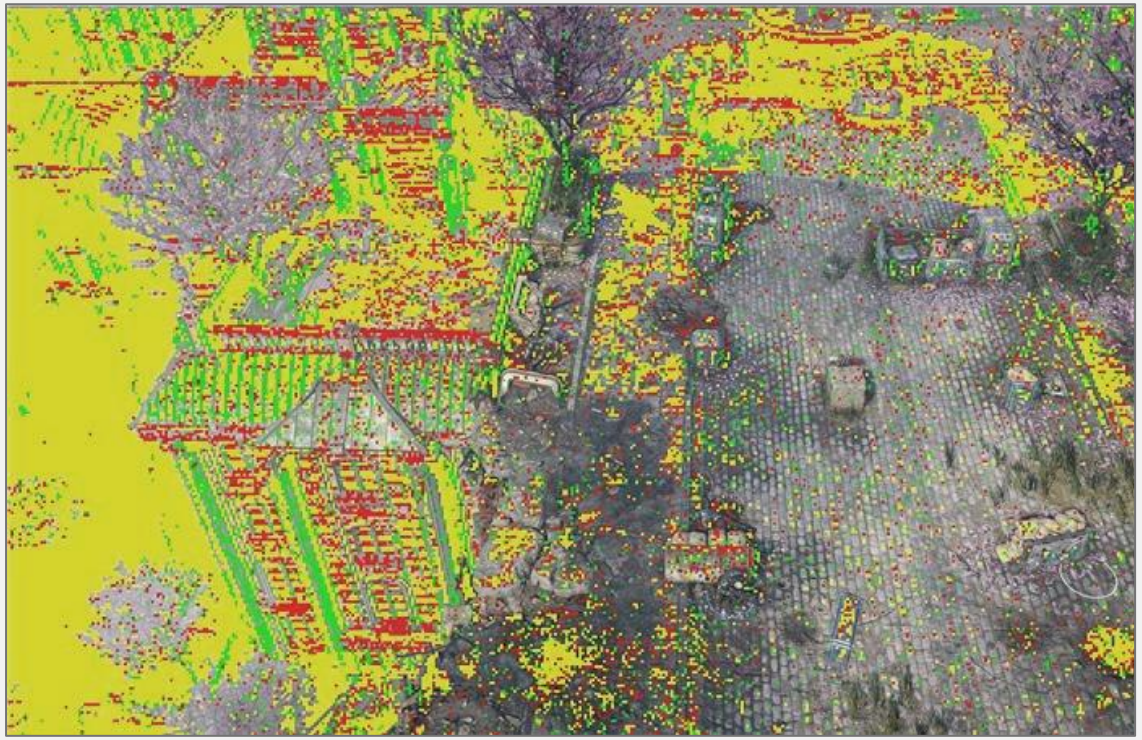


480p VRS Texture (R8_UINT)

VRS in Gears Tactics



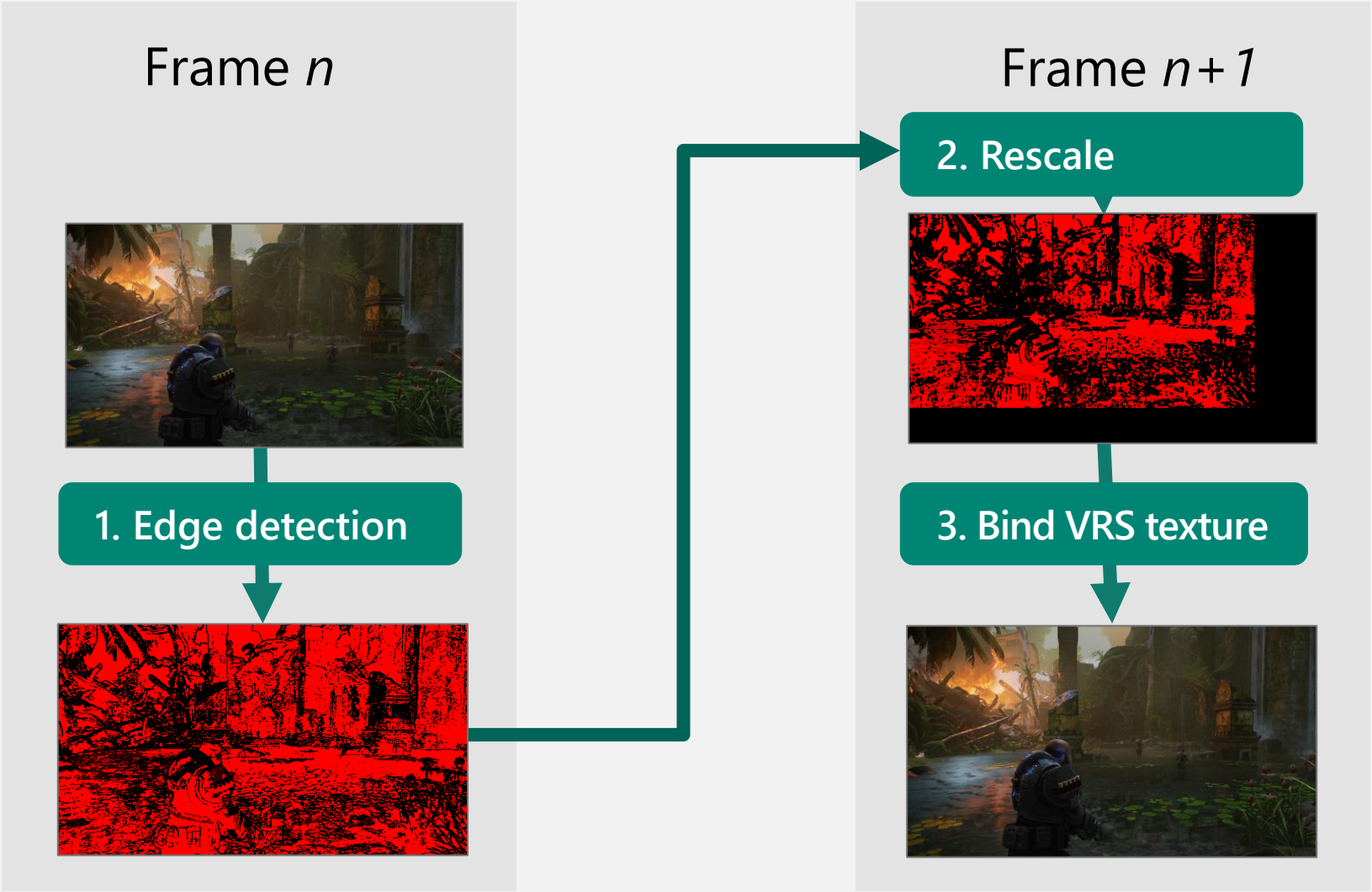
Tier 1 VRS



Tier 2 VRS

Implementation

VRS texture generation



Edge detection

Done via compute shader

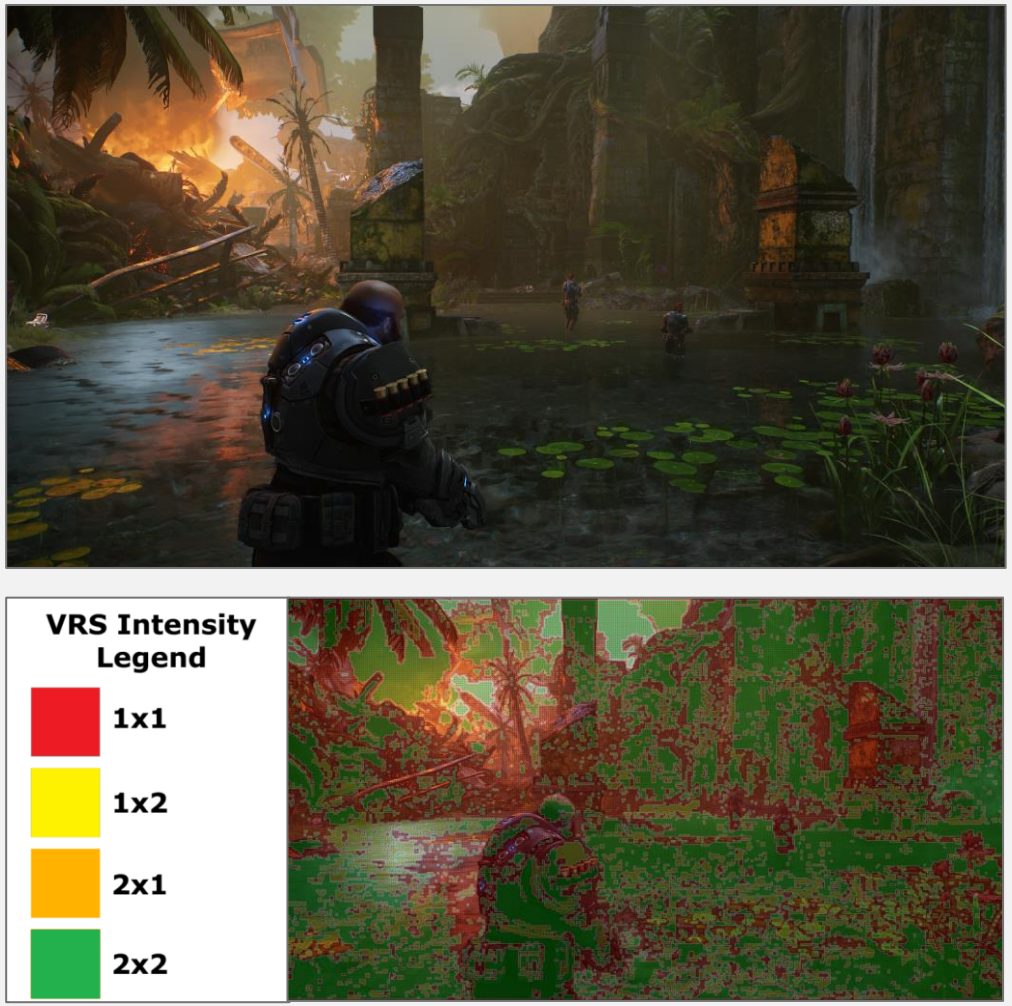
Threadgroup size matches the VRS tile size

Sobel Edge detection on the full resolution final post processing output

- Use luminance in **sRGB** color space

Each thread calculates the desired shading rate for a pixel

WaveAND to get the lowest shading rate from all threads



Edge detection

Naïve implementation @ 4k costs .4ms.

Key optimizations:

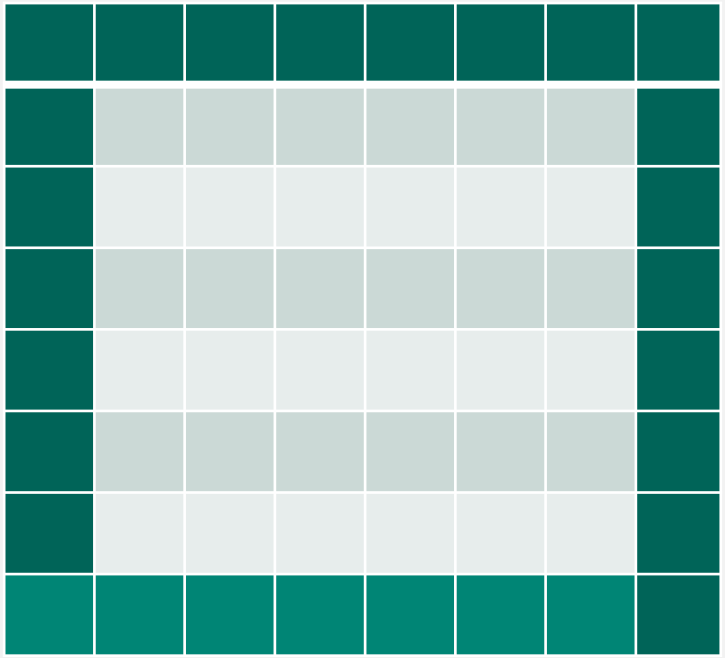
1. Skip border pixels: saves .2ms

   On an 8x8 VRS tile, reduces pixels that need edge detection from 64->36

2. Merge edge detection with the Tonemapping shader: saves ~.1ms

   Avoids writing the scene color to memory and immediately reading it back

3. Use async compute: saves < .1ms

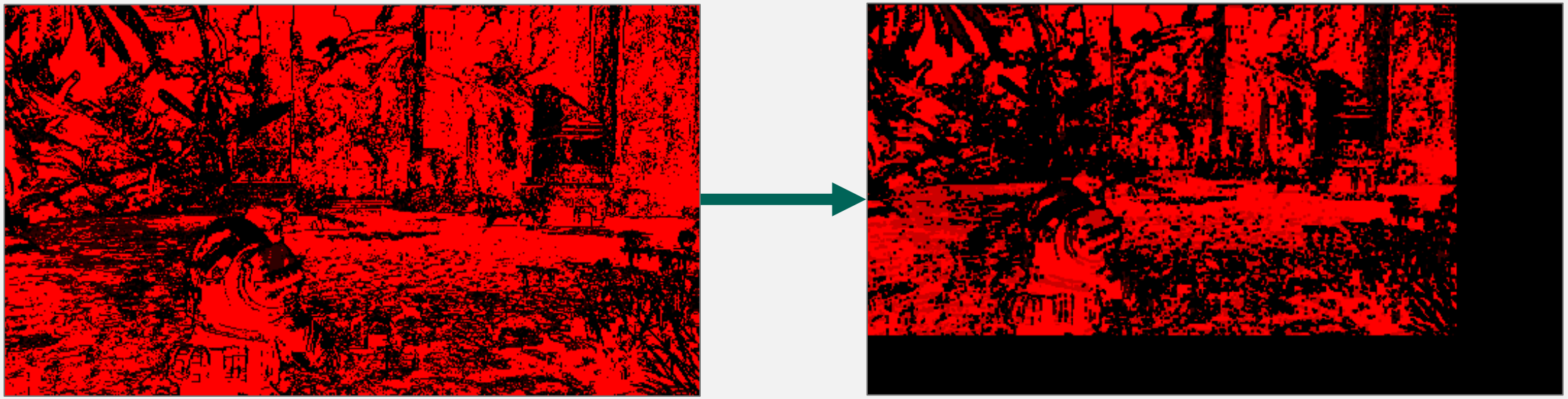


Rescale

Need to handle changes in render targets due to dynamic resolution

VRS texture is between an 8–16x downscale of the rendered buffer so the rescale compute shader is fast (0.02ms)

Previous frame reprojection done in the Rescale shader as well



Binding the VRS texture

Bind VRS texture via RSSetShadingRateImage

Applied to multiple passes:

- Base Pass
- Screen Space Ambient Occlusion
- Screen Space Global Illumination*
- Lighting
- Screen Space Reflections
- SSR Temporal
- Translucency

Not all passes VRS equally

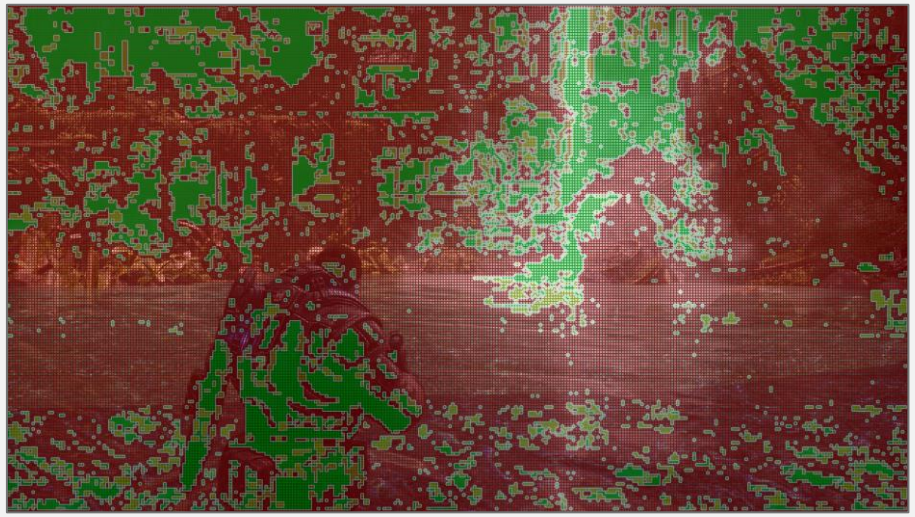
Some passes hide coarser shading rate better than others

- Translucency hides coarse shading rate well
- SSR not so much

Generated a second **conservative** VRS texture

"Free" to generate, re-uses edge detection results and just compare against a more conservative threshold value

Default VRS texture



Conservative VRS texture

VRS with Screen Space Global Illumination

Screen Space Global Illumination

- Great visual results but makes your GPU cry

Good candidate for VRS

- GI takes well to being done at lower resolutions
- But it’s compute shader driven…

VRS with Screen Space Global Illumination

That's okay, the VRS texture can be read from as an SRV! Emulate VRS behavior in a compute shader

Terminate threads/threadgroups based on shading rate

Hole-fill at the end of the shader based on shading rate

Shading rate fed into denoising

Had some minor quality issues, only used with "Aggressive" VRS settings for PC

SSRTDiffuseIndirect.usf

```
uint GroupWaveIndex = GroupThreadIndex / WaveSize;
uint TotalWaveCount = GroupThreadCount / WaveSize;
if ((ShadingRate & D3D12_SHADING_RATE_2X1) != 0) {
   TotalWaveCount /= 2;
}
if ((ShadingRate & D3D12_SHADING_RATE_1X2) != 0) {
   TotalWaveCount /= 2;
}
if (GroupWaveIndex >= TotalWaveCount) return;

.....

DiffIndUAV[Coord] = DiffuseColor;
if ((ShadingRate & D3D12_SHADING_RATE_1X2) != 0) {
    DiffIndUAV[Coord + uint2(0,1)] = DiffuseColor;
}
if ((ShadingRate & D3D12_SHADING_RATE_2X1) != 0) {
    DiffIndUAV[Coord + uint2(1,0)] = DiffuseColor;
}
if ((ShadingRate & D3D12_SHADING_RATE_2X2) != 0) {
    DiffIndUAV[Coord + uint2(1,1)] = DiffuseColor;
}
```

Results


VRS Off
Y
200


Y
VRS On
200


VRS Off


VRS On

VRS for Xbox Series X

Performance taken with dynamic resolution locked at 4K (i.e., off)

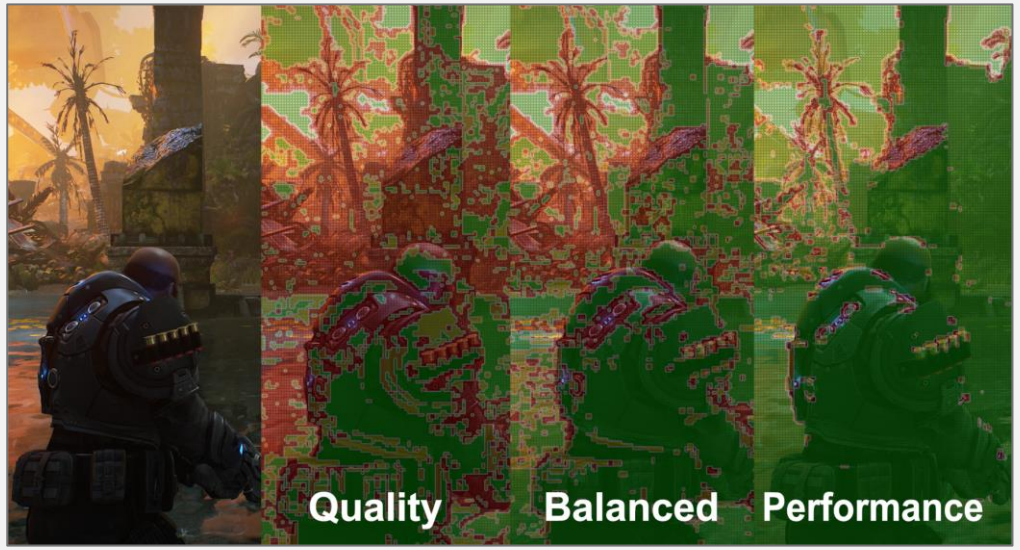
Increase dynamic resolution from 83%->92%

	VRS Off (ms)	VRS (ms)	Savings (ms)	Savings %
Base Pass	2.7	2.2	0.5	19%
SSAO	2.2	0.95	1.25	57%
Lighting	3.3	2.7	0.6	18%
SSR	0.73	0.51	0.22	30%
Translucency	0.22	0.17	0.05	23%
Total Frame	**20.4**	**17.8**	**2.6**	**13%**

VRS for PC

Ran at 4k on an AMD RX 6900 XT
with *Ultra* Settings

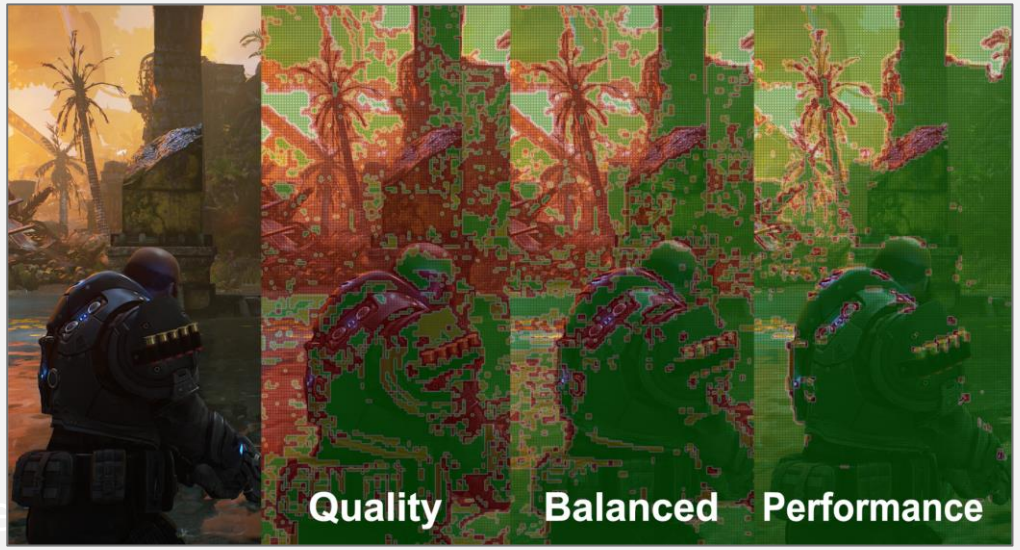
	Frametime (ms)	Savings (ms)	Savings (%)
VRS Off	13.3	-	-
VRS Quality	12.3	1	8%
VRS Balanced	12.1	1.2	10%
VRS Performance	11.9	1.4	12%



VRS for PC

Ran at 4k on an AMD RX 6900 XT with *Insane* Settings + Screen Space Global Illumination

	Frametime (ms)	Savings (ms)	Savings (%)
VRS Off	23.0	-	-
VRS Quality	19.8	3.21	14%
VRS Balanced	19.6	3.41	15%
VRS Performance	18.5	4.55	20%



Integration

Evaluating performance

Add RSSetShadingRate and 2x2 everything

Compare performance results before and after 2x2 for best case results

- Final performance is probably 30–50% of the savings.
  Varies wildly based on how aggressive VRS texture generation is

VRS integration

Used a RAII pattern for setting VRS state on construction and reverting it on destruction

Adding VRS to most passes becomes a 1-liner

ScreenSpaceReflections.cpp

```
void FRCPassPostProcessScreenSpaceReflections::Process(…)
{
  auto& RHICmdList = Context.RHICmdList;
  const FViewInfo& View = Context.View;
  FConditionalScopedVRS VRS(RHICmdList, VRSModeForSSR(View));
```

Bugs…

Fast and easy to get up and running with VRS

But plan for bugs, here’s some we hit to help evaluate

DDX/DDY

UE4 material system allows for artists to create node-based shaders

DDX/DDY material node:

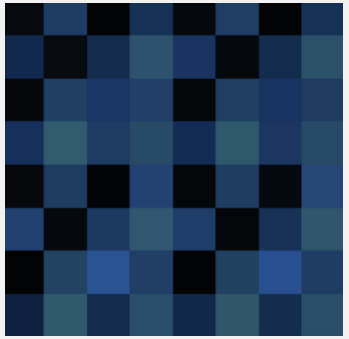
- Stretched when coarse

Solution: Correct DDX/DDY based on shading rate
(or disable VRS for materials using DDX/DDY)

Dithering artifacts

Screen space reflections rely on dithering rays within a 4x4 block of pixels

Solution: Stretch dither pattern when using coarser shading rates



SSR output before Temporal AA

Light aliasing

Coarse rate shading will interpolate between Gbuffer values. Leads to poorly interpolated normal when done on mesh edges

Solution: Snap pixel centroid to a single pixel center

HDR

Game originally authored in SDR

SDR converted to HDR after post processing

Found that doing VRS generation based on SDR results was “good enough”

Conclusions

VRS can make games look **better**

API allows for both easy evaluation and integration

DX12 Ultimate allows this to carry over across both console and PC

Acknowledgements

Key contributors

Jacob Nelson
Cam Mcrae

Tech art

Colin Penty
Geoff Lester

Rendering

Mike Perzel
Ian Wong
David Lucas
Simon Wong

Xbox/D3D Team

Ivan Nevraev
Xiang Li
Martin Fuller
Claire Andrews

References

Variable Rate Shading: a scalpel in a world of sledgehammers. Jacques van Rhyn

Variable Rate Shading | DirectX-Specs. Microsoft

Iterating on Variable Rate Shading in Gears Tactics. Jacob Nelson, Cam McRae

Moving Gears to Tier 2 Variable Rate Shading. Chris Wallis

Software-based Variable Rate Shading in Call of Duty: Modern Warfare. Michal Drobot

Questions?

Snapping the Pixel Centroid

Useful for lighting where centering the pixel centroid in the center of a 2x2 is undesirable

The selected pixel to snap to rotates based on the frame index

Any additional PS interpolants need to be snapped as well. Use of ddx/ddy makes it easy to copy-paste these in

Note: ddx/ddy will be stretched by coarse rate shading, so this needs to be accounted for

VariableRateShadingCommon.usf

```
void SnapAttributesBasedOnShadingRate(uint ShadingRate, inout,
float4 SVPos, inout float4 ScreenPosition)
{
   uint JitterIndex = Frame.StateFrameIndexMod8;
   float DerivativeMultiplier = 0.25f;
   if ((ShadingRate & D3D12_SHADING_RATE_2X1) != 0)
   {
      float JitterMultiplier = (JitterIndex % 2) == 0 ? -1.0 : 1.0;
      float OffsetMultiplier = DerivativeMultiplier * JitterMultiplier;
      SVPos += ddx(SVPos) * OffsetMultiplier;
      ScreenPosition += ddx(ScreenPosition) * OffsetMultiplier;
      JitterIndex = JitterIndex >> 1;
   }

   if ((ShadingRate & D3D12_SHADING_RATE_1X2) != 0)
   {
      float JitterMultiplier = (JitterIndex % 2) == 0 ? -1.0 : 1.0;
      float OffsetMultiplier = DerivativeMultiplier * JitterMultiplier;
      SVPos += ddy(SVPos) * OffsetMultiplier;
      ScreenPosition += ddy(ScreenPosition) * OffsetMultiplier;
   }
}
```

Querying Shading Rate without SM 6.0

Shading rate can be queried using SV_ShadingRate. Requires SM 6.0

However, shading rate can be implied on <SM 6.0 by looking at whether SV Position is on the center of a pixel or the edge (i.e., if the centroid has been shifted between 2 pixels, then coarse rate shading is being used)

VariableRateShadingCommon.usf

```
bool IsFloatEqual(float a, float b)
{
   const float Epsilon = 0.01;
   return abs(a - b) < Epsilon;
}

uint GetShadingRateFromSVPosition(float4 SVPosition)
{
   uint ShadingRate = D3D12_SHADING_RATE_1X1;
   if (IsFloatEqual(frac(SVPosition.x), 0.0))
   {
      ShadingRate |= D3D12_SHADING_RATE_2X1;
   }

   if (IsFloatEqual(frac(SVPosition.y), 0.0))
   {
      ShadingRate |= D3D12_SHADING_RATE_1X2;
   }
   return ShadingRate;
}
```

Microsoft