VARIABLE-RATE COMPUTE SHADERS

in DOOM: The Dark Ages

Martin Fuller
Principal Engineer
Xbox Advanced Technology Group

Philip Hammer
Principal Engine Programmer
id Software

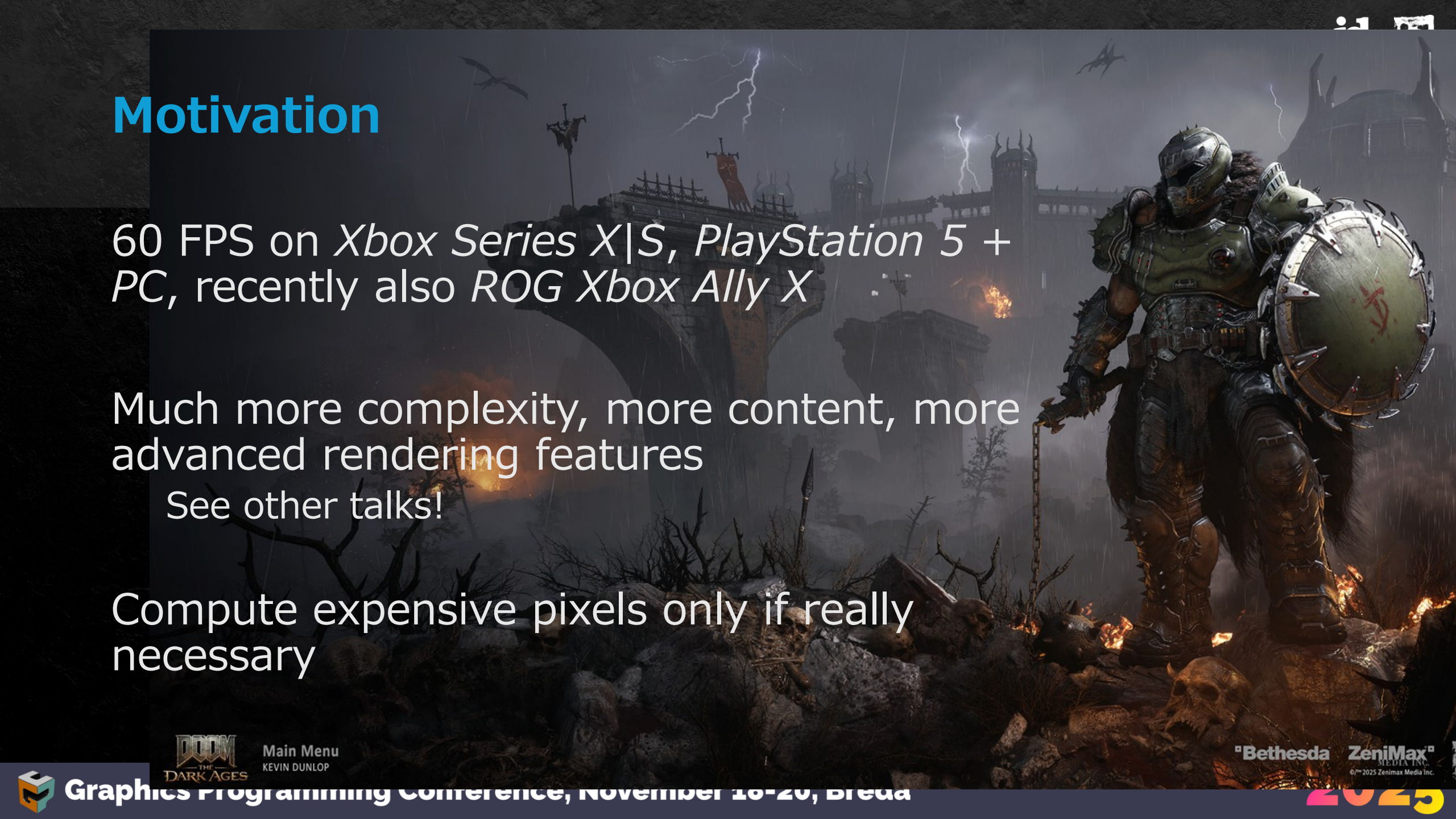
Motivation

60 FPS on *Xbox Series X|S*, *PlayStation 5* + *PC*, recently also *ROG Xbox Ally X*

Much more complexity, more content, more advanced rendering features

- See other talks!

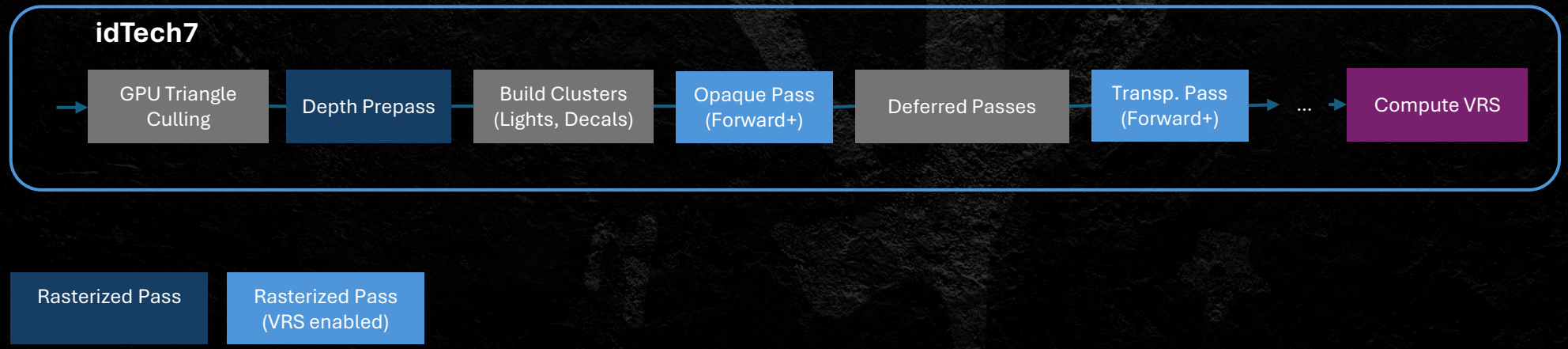
Compute expensive pixels only if really necessary



Recap: VRS in DOOM Eternal

idTech7 uses Clustered Forward+ rendering

- Hardware VRS on Xbox Series X|S (2022)
  - *(unsupported on base PS5 and many PC GPUs)*
- Added with the Gen9 update along with Raytraced reflections, 120 Hz mode, etc



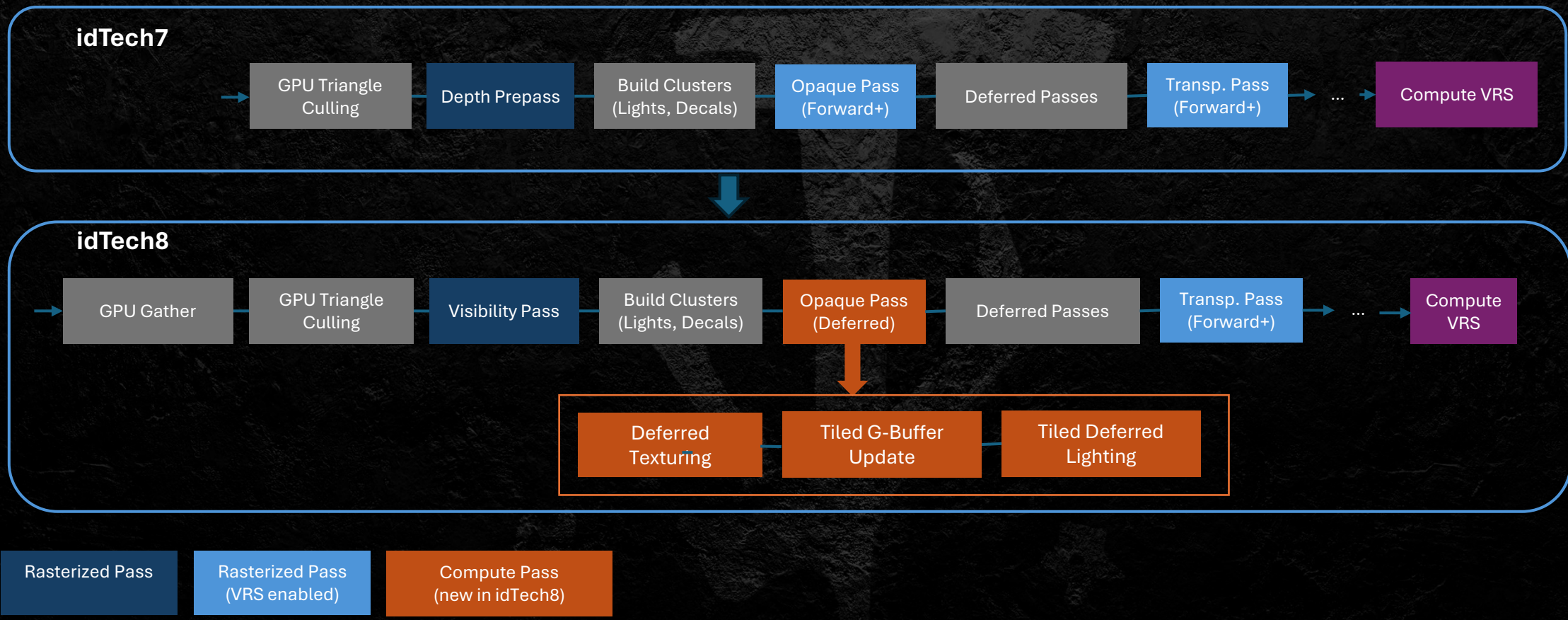
Recap: VRS in DOOM Eternal

VRS is a great fit for forward rendering

- Lots of expensive pixels avoided in DOOM Eternal
- Watch Variable Rate Shading Update Xbox Series Consoles

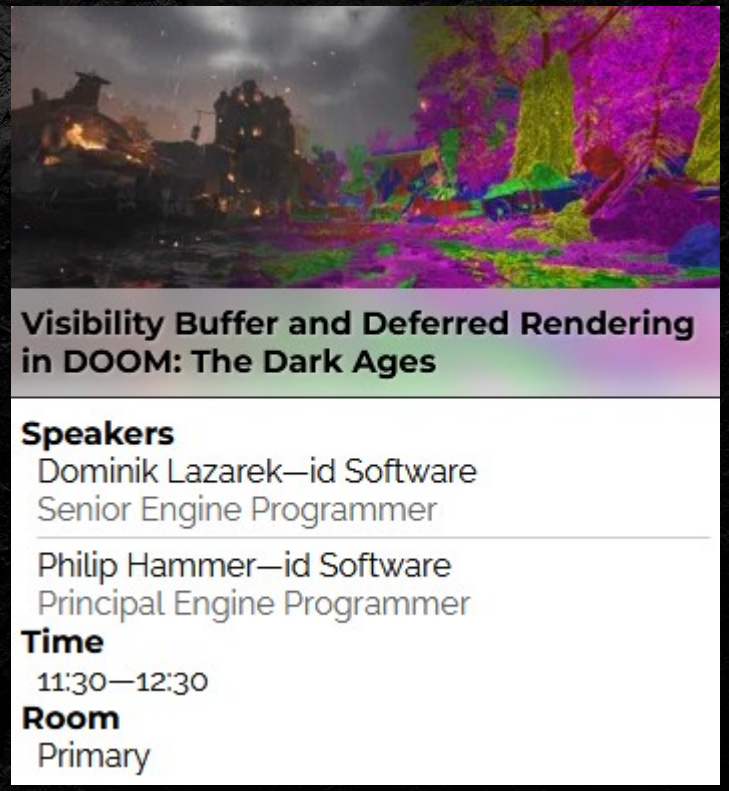
	VRS Off (ms)	VRS On (ms)	Savings (ms)	Savings %
Opaque	5.37	4.12	1.25	+23%
Blend	8.56	6.02	2.54	+29%
Total Frame	21.4	17.77	3.63	+17%

Recap: idTech7 vs. idTech8 pipeline



Recap: idTech8 Deferred Rendering

- Visibility Buffers + Deferred everything
  - Other DOOM:TDA talk today:
    Visibility Buffer And Deferred Rendering in DOOM: The Dark Ages
- Perf gain by minimizing raster work
  - Minimize Helper Pixel waste
  - Move heavy duty work to compute
  - Remaining raster work remained lightweight
    (Depth/Visibility Buffer, Shadows)



Recap: idTech8 Rendering vs. (hardware) VRS

Great results at first

But (hardware) VRS doesn't work with compute

Now what?

Forward+	**17.5 ms**
Visibility Buffer	**14.8 ms**

Forward+ (with VRS)	**14.5 ms**
Visibility Buffer (with VRS)	**14.6 ms**

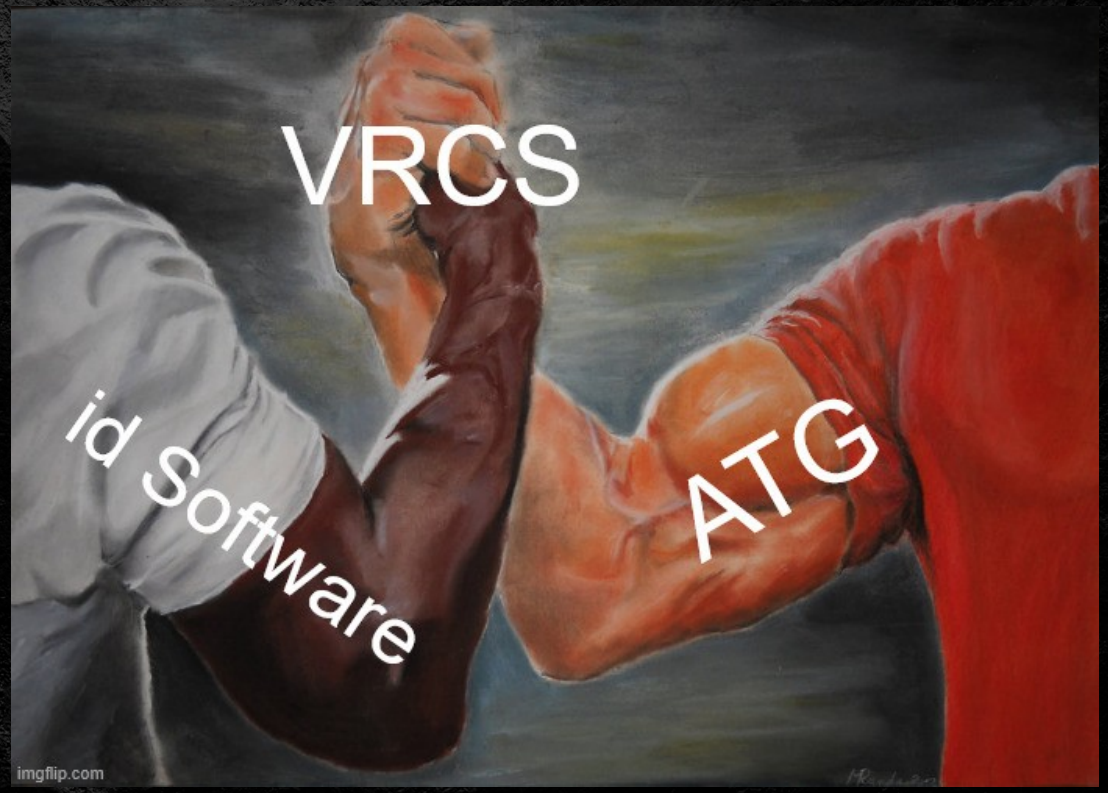
Variable-Rate Compute Shaders

VRCS to the rescue!

Initial implementation within 2 weeks from Martin's prior work

Proof-of-Concept != Production Ready

Direct collab at the id Frankfurt office



Variable-Rate Compute Shaders

Goal = Save performance!

(by retiring compute waves early)

Unique calculations only for a subset of pixels

Copy the result to the neighbor pixels

Apply to heavy-lifting compute passes replacing opaque forward rendering

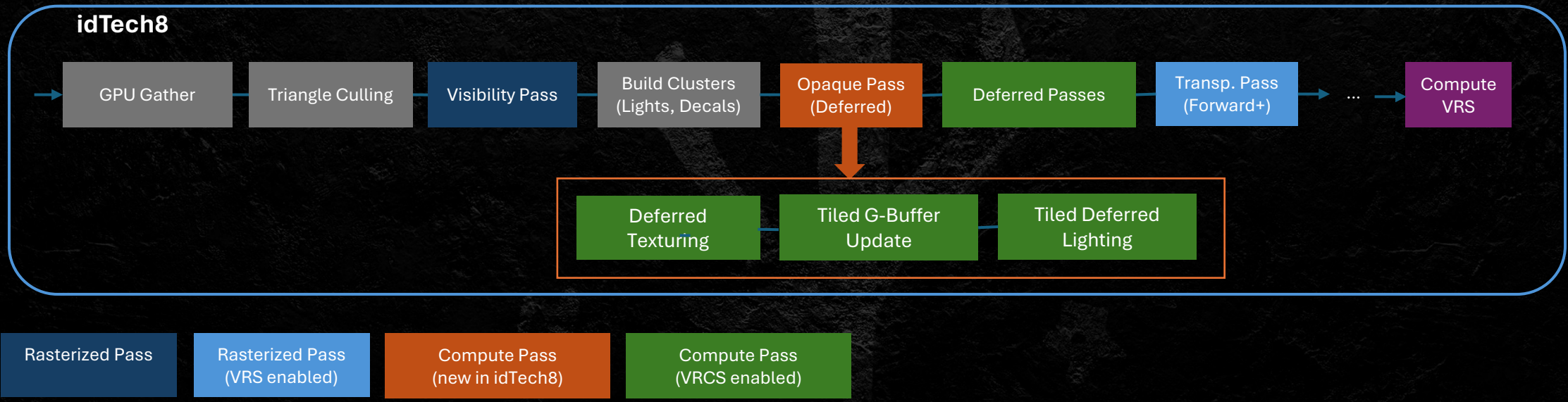
Variable-Rate Compute Shader Passes

Deferred Texturing (material evaluation)

Deferred G-Buffer Update (decals, rain, blood)

Deferred Lighting (lighting, shadows)

(initially) Deferred Composite (GI, Fog, Reflections, etc)



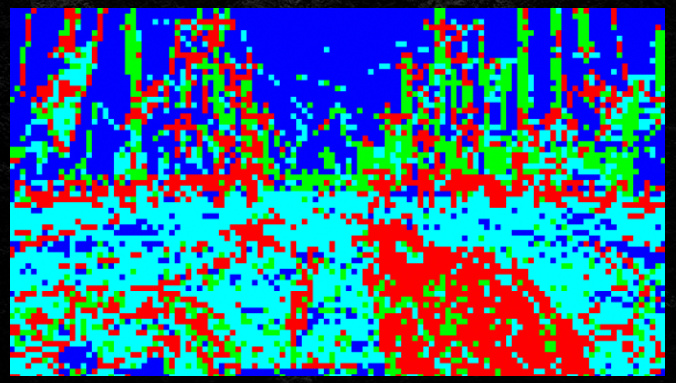
VRCS - Shading Rate Image

2x2 Shading Rate Image (SRI) 8bpp UINT

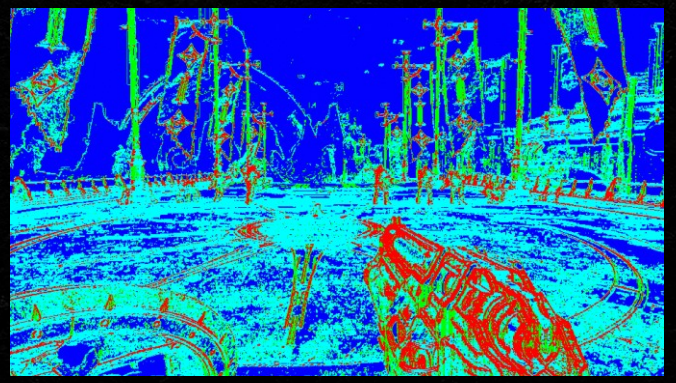
Analyzing luma gradients (think Sobel filter)

Assign shading rates to each 2x2 tile

Scene Luminance

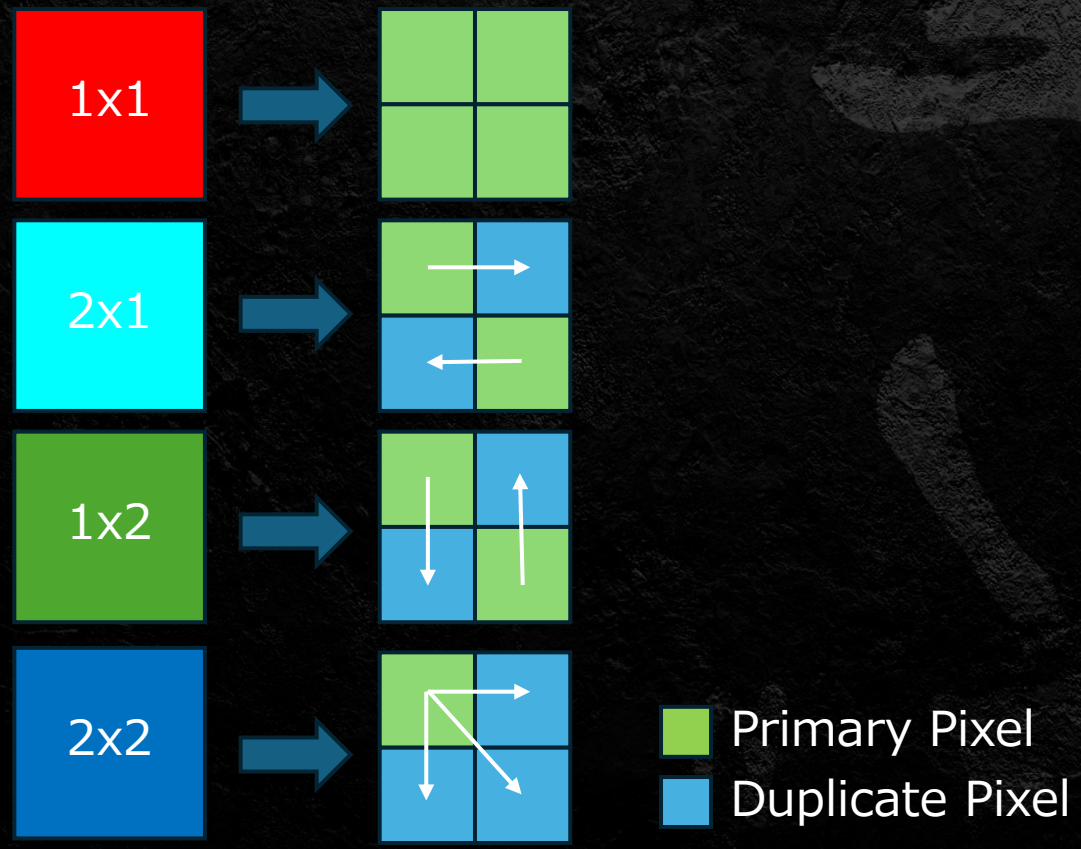


8x8 SRI for HW VRS (on AMD)



2x2 SRI for VRCS (any IHV)

Shading Rate + Coverage -> VRCS Buffer

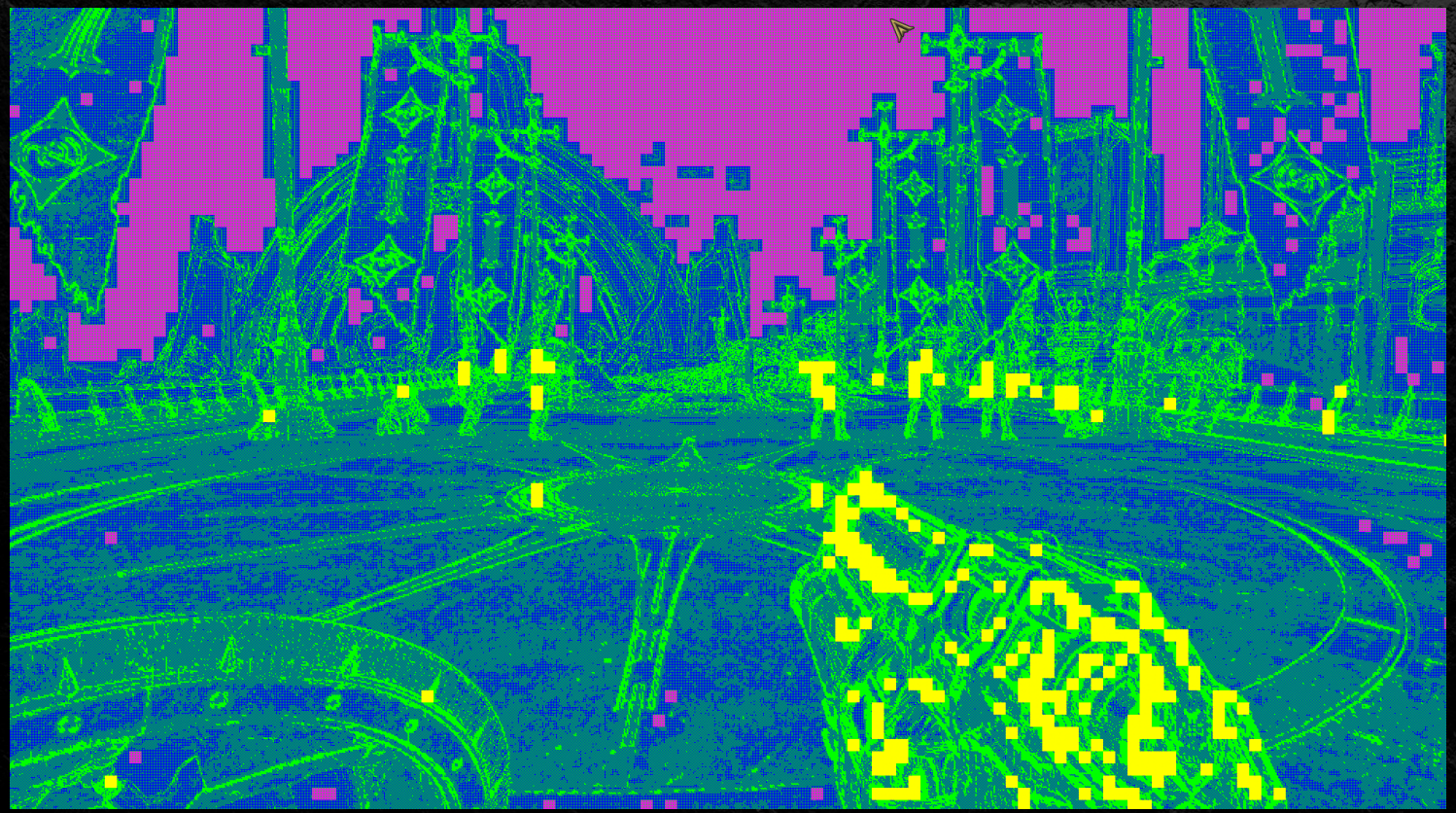


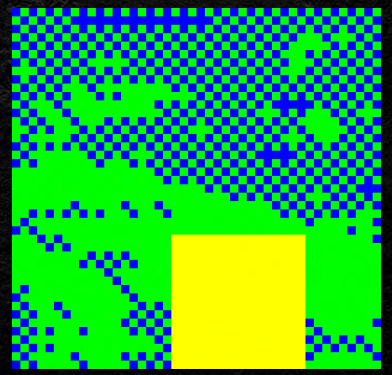
Primary vs. Duplicate Pixels

Shading Rate + Coverage determines uniquely calculated pixels

Primary Pixel result gets copied

VRCS 16x16 Tile Debug





Primary pixel

Duplicate pixel

Tile entirely 2x2 rate
(1 primary, 3 duplicates per 2x2)

Tile entirely 1x1 rate
(4 primary pixels per 2x2)

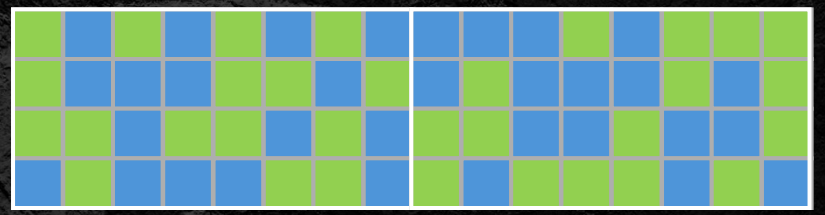
VRCS Compute Scheduling

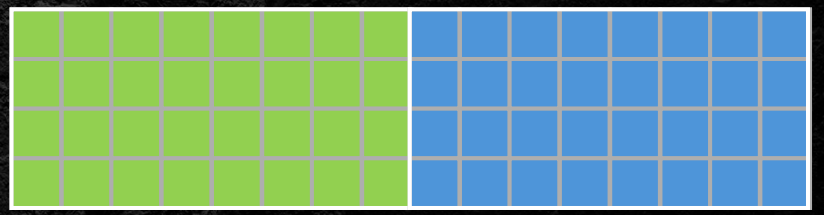
GPU work scheduling

No gain when mixing primaries and duplicates in the same wave

Must compute primary pixels together in waves

Goal: Uniform early-out for one or more waves





VRCS: Coordinate Remapping

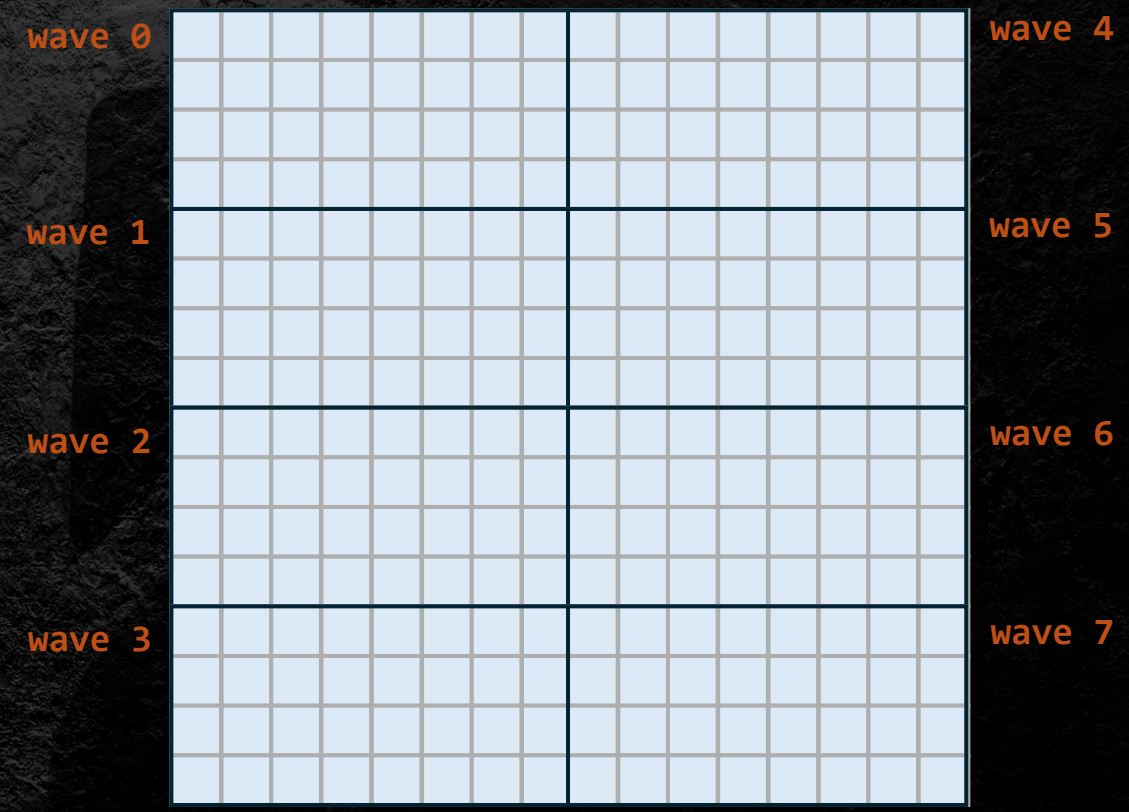
Segment screen into 16x16 pixel tiles

- 8x wave32 with 8x4 threads each
- All threads tile local => scalar light load

Store primary pixel count per tile

Store per pixel payload:

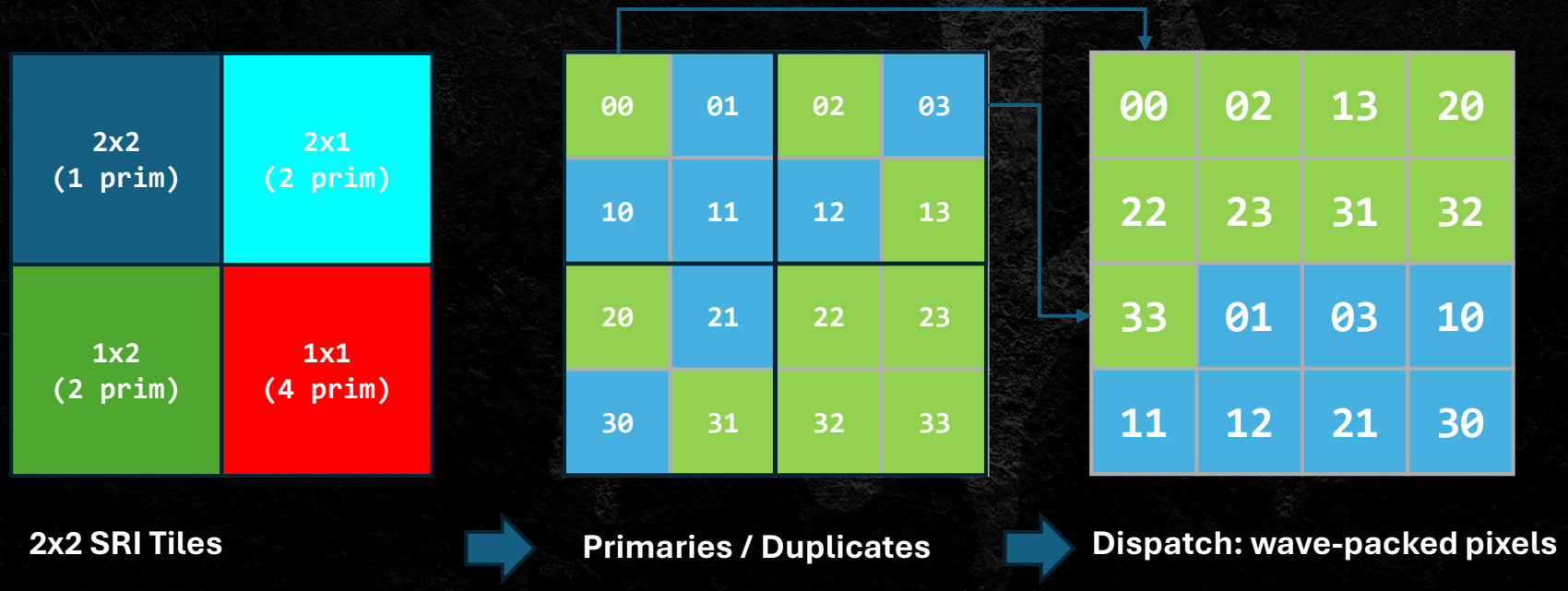
- 8 bits tile-relative XY offset (4 bits per dim)
- 3 bits copy commands (vert, horz, diag)



VRCS: Coordinate Remapping

Tile-relative offset to remap the pixel coordinate

Not moving any memory, just remapping threads



VRCS: Wave Packing Example

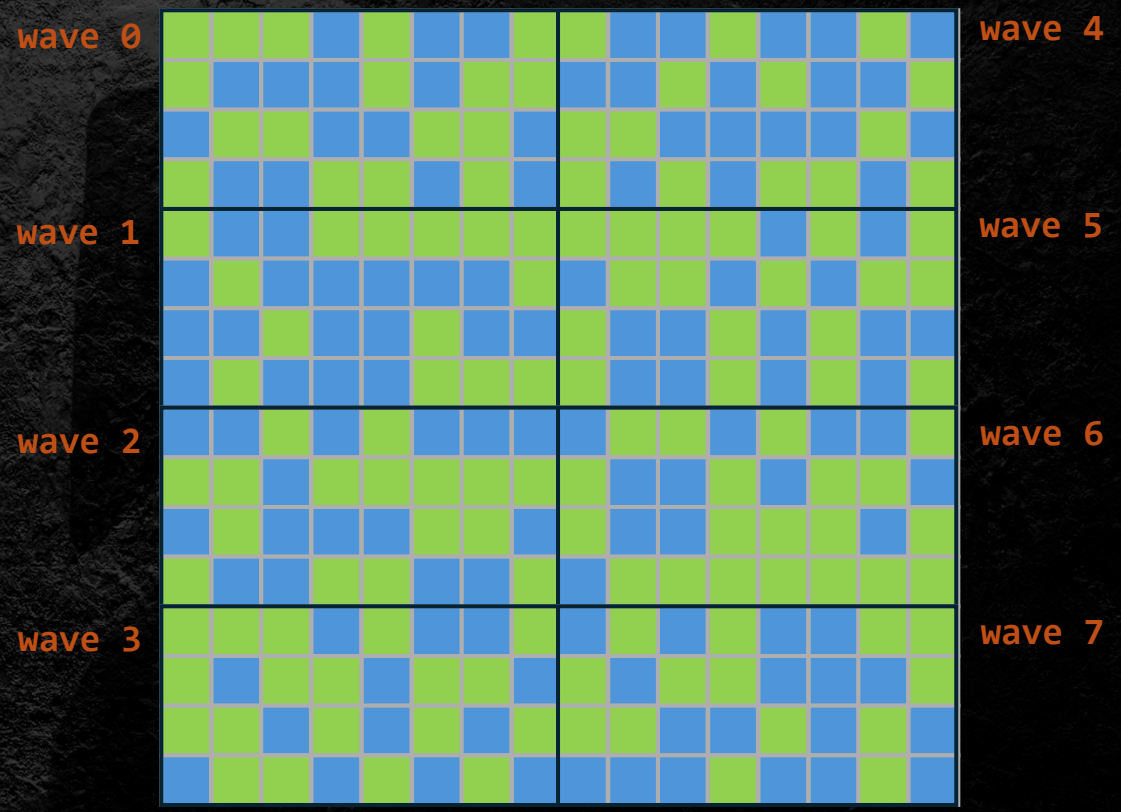
Example 16x16 tile:

132 primary pixels

124 duplicate pixels

Every wave processes both primary and duplicate pixels

-> Full cost for all 8 waves



VRCS: Wave Packing Example

Example 16x16 tile:

132 primary pixels

124 duplicate pixels

---

4 full waves of primary pixels

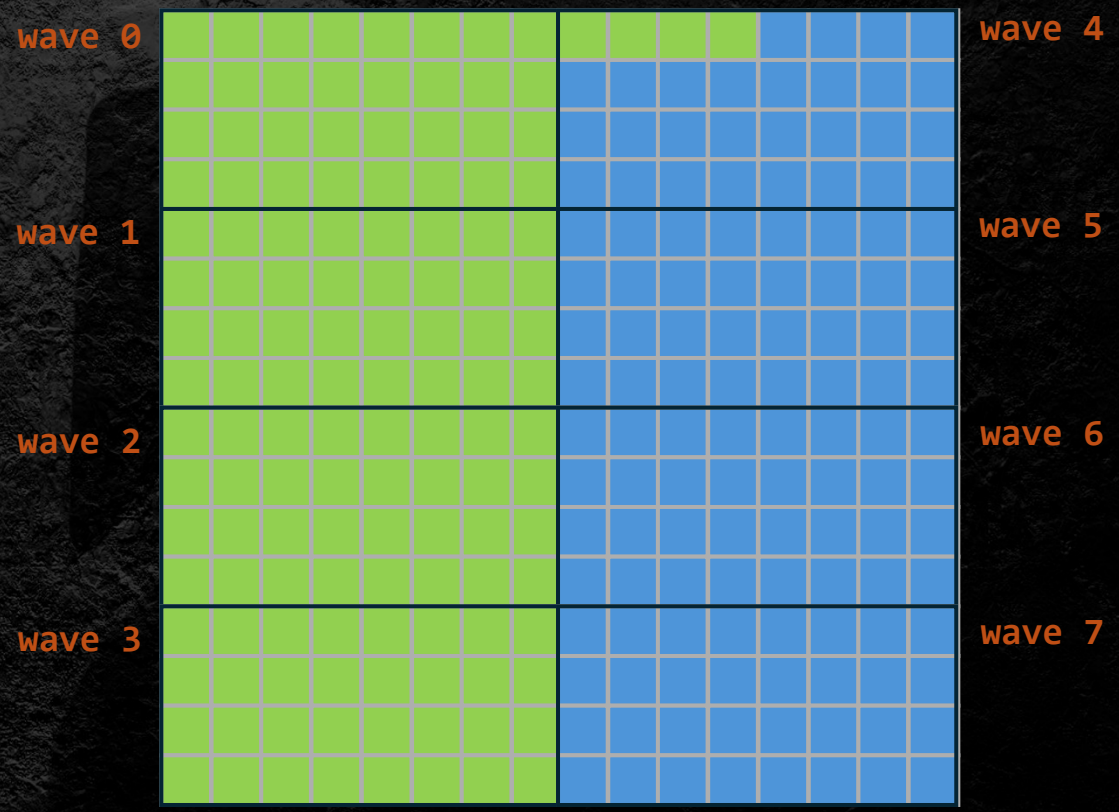
-> Full cost for shader execution

1 partial wave (4 primary pixels)

-> still full cost

3 full waves of duplicate pixels

-> very cheap (uniform early-out)



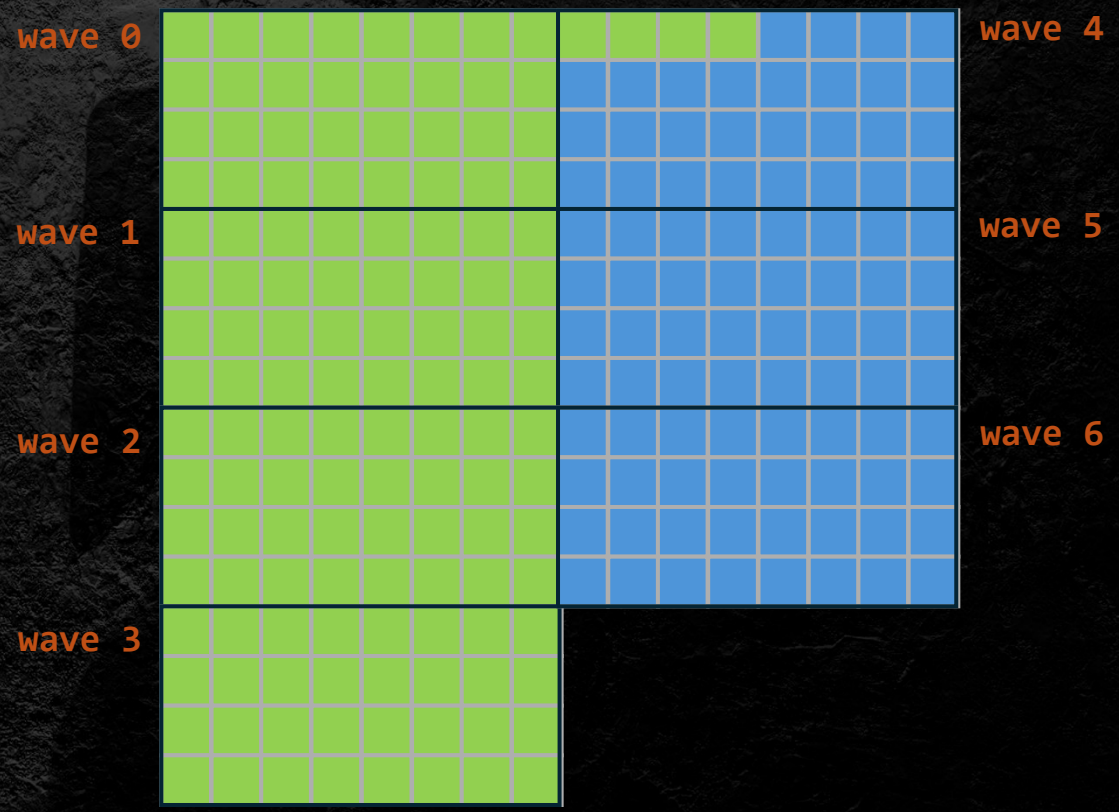
VRCS: Optimization

No storage need for wave 7

Need to save at least 1 wave to realize a benefit

Otherwise execute whole tile 1x1 rate

Storage per tile = 256−32 = 224 entries



VRCS: Implementation – Simple!

VRCSPrefix: Remap the pixel location and retrieve **vrcsData**

VRCSPostFix: store **outputValue**, potentially copy to duplicate pixel

```
void main() {
    int2 pixelLocation = compute.globalInvocationID.xy;
    uint vrcsData = 0;
    if ( constants.vrcsEnabled == true ) {
        uint vrcsCode = VRCSPrefix( pixelLocation, vrcsData );
        // early-out directly if there is nothing to do at all
        if ( ( vrcsCode == VRCS_TILE_SKY ) ||
             ( vrcsCode == VRCS_TILE_NO_WORK_FOR_THREAD ) ) {
            return;
        }
    }
    float4 outputValue;
    // do shader work, e.g. lighting calculations, etc.
    // ..

    if ( constants.vrcsEnabled == true ) {
        VRCSPostFix( pixelLocation, vrcsData, outputValue );
    }
}
```

VRCS: Implementation

Compute tile index

Evaluate early-out conditions

Remap pixel coords

```
uint VRCSPrefix( inout int2 coord, inout uint vrcsData ) {
    // there are 8 wave32s per 16x16 vrcs tile, calculate
    // which 16x16 tile we're dealing with
    const uint2 tileId = coord.xy / VRCS_TILE_SIZE;
    const uint tileIndex = tileId.y * constants.vrcsBufferWidth + tileId.x;
    const uint vrcsCount = constants.vrcsCountBuffer[ tileIndex ];
    const uint coordIndex = ( ( coord.y & 15 ) * 16 ) + ( coord.x & 15 );
    BRANCH if ( vrcsCount == 0 ) {
        return VRCS_TILE_SKY;
    }
    BRANCH if ( vrcsCount > ( 256 - 32 ) ) {
        return VRCS_TILE_LIGHT_ALL_PIXELS;
    }
    BRANCH if ( coordIndex >= vrcsCount ) {
        return VRCS_TILE_NO_WORK_FOR_THREAD;
    }
    vrcsData = vrcsCoordsBuffer[ tileIndex * MAX_COORDINATES_PER_SCREEN_TILE + coordIndex ];
    coord.x = int( VRCS_DATA_GET_X( vrcsData ) + ( VRCS_TILE_SIZE * tileId.x ) );
    coord.y = int( VRCS_DATA_GET_Y( vrcsData ) + ( VRCS_TILE_SIZE * tileId.y ) );
    return VRCS_TILE_WORK_FOR_THREAD;
}
```

```
#define VRCS_TILE_SKY                   0
#define VRCS_TILE_LIGHT_ALL_PIXELS      1
#define VRCS_TILE_WORK_FOR_THREAD       2
#define VRCS_TILE_NO_WORK_FOR_THREAD    3

#define VRCS_TILE_SIZE                 16

#define VRCS_DATA_GET_X(vrcsData)   ((vrcsData >> 0) & 0xf)
#define VRCS_DATA_GET_Y(vrcsData)   ((vrcsData >> 4) & 0xf)

#define VRCS_DATA_COPY_H(vrcsData)  ((vrcsData & 0x100) > 0)
#define VRCS_DATA_COPY_V(vrcsData)  ((vrcsData & 0x200) > 0)
#define VRCS_DATA_COPY_D(vrcsData)  ((vrcsData & 0x400) > 0)
```

VRCS: Implementation

Store primary pixel

Check copy bits & store to copy location(s)

```
void VRCSPostFix( int2 coord, uint vrcsData, float4 outputValue ) {
    imageStore( targetImageUAV, coord, outputValue );
    int2 copyCoord = coord.xy ^ 0x1;
    // do the pixel copies (hole filling)
    if ( VRCS_DATA_COPY_H( vrcsData ) ) {
        imageStore( targetImageUAV, int2( copyCoord.x, coord.y ), outputValue );
    }
    if ( VRCS_DATA_COPY_V( vrcsData ) ) {
        imageStore( targetImageUAV, int2( coord.x, copyCoord.y ), outputValue );
    }
    if ( VRCS_DATA_COPY_D( vrcsData ) ) {
        imageStore( targetImageUAV, int2( copyCoord.x, copyCoord.y ), outputValue );
    }
}
```

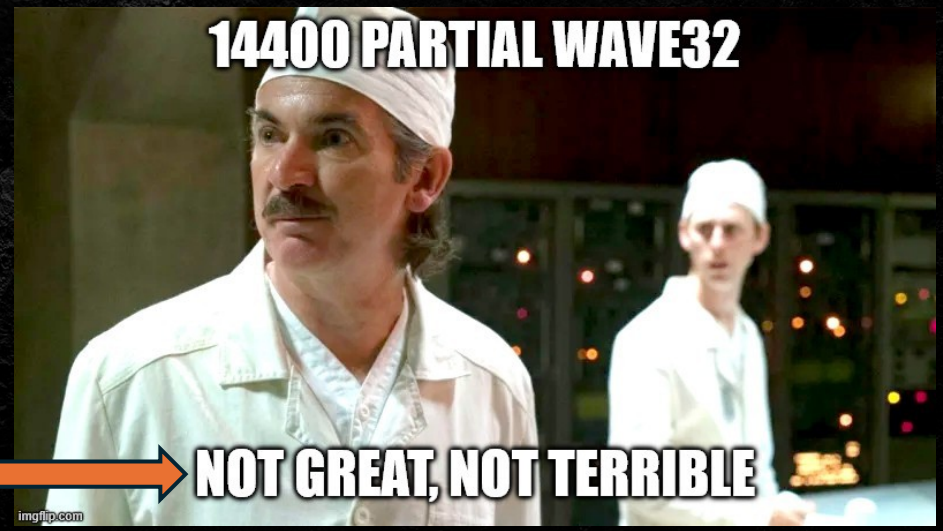
VRCS: Dispatch (Tiled Data)

Typical scenario: fullscreen dispatch
- Dispatch all threadgroups needed
- Early out for duplicate pixels

max. 1 partial wave per 16x16 tile (on average 1/8 of waves)

e.g. 14400/115200 partial waves @ 1440p

Nitpick - not 'quite' correct
- Sometimes `(numPrimaryPixels % 32) == 0`
- Including tiles flagged 1x1 or 2x2



VRCS: Dispatch (Pixel Commands)

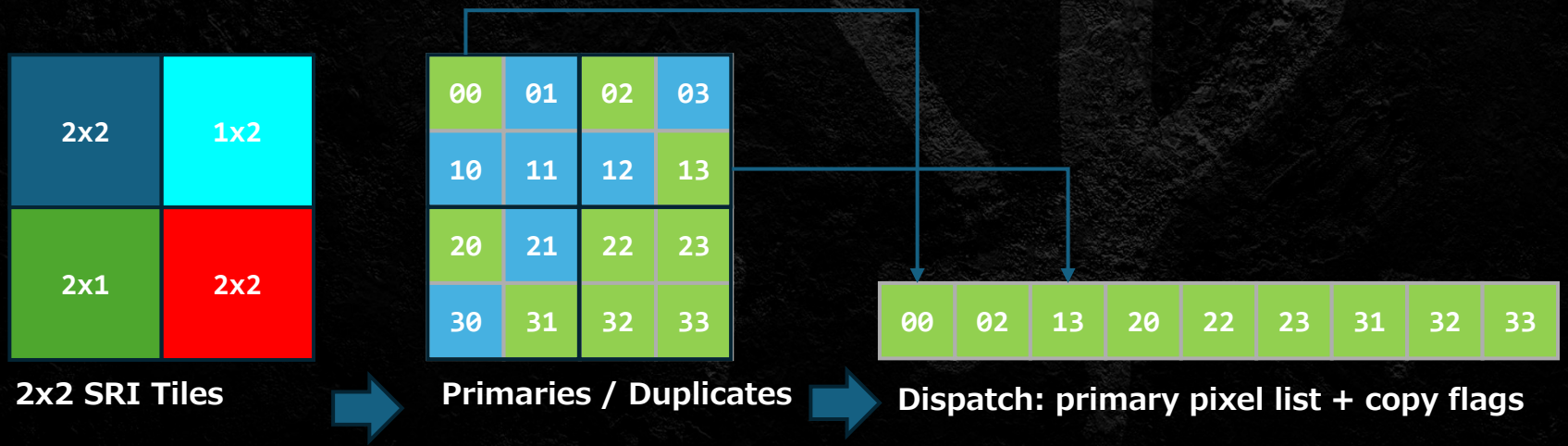
But we can do better ..

Deferred Texturing processes a single list of primary pixels („*Pixel Commands*“)

Only atomic-add primary pixels + copy flags

no early-out waves, no tile flags

= max. 1 partial wave per screen!



VRCS: Dispatch (The Future)

Observation: VRCS on Compact Pixel List turned out to be very efficient

- Max. 1 partial wave
- Duplicate pixel threads are not launched at all

Idea: Compact Pixel Lists also for other deferred stages (Lighting, G-Buffer, etc)

- Need more memory
- Need to solve uniform cluster reads (tile system improves uniform access to lights)

Idea+: Run Texturing, Lighting, G-Buffer update in one shader

- Similar shader setup like F+, but in compute
- Actually tried this, but ran out of time
- VGPR issues, difficulties with very long programs

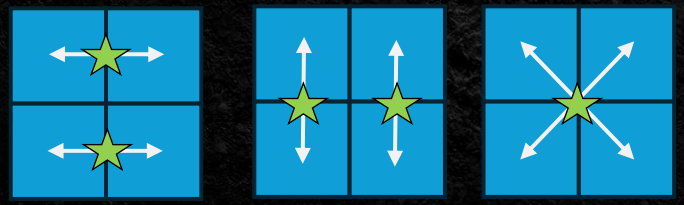
Fighting the Good Fight (Image Quality)

5 weeks to code lock, Rip and Tear until it's done!

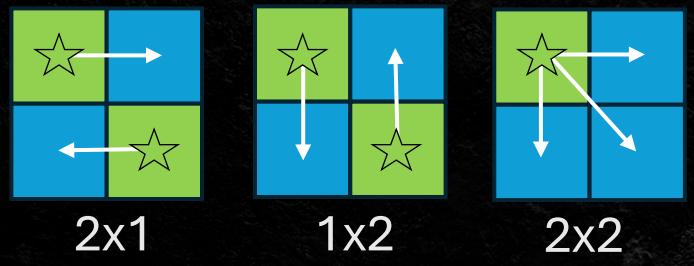
Pixel Offset Problem

Hardware VRS invokes the PS at the center of the ‘coarse pixel’

Covered pixels receive copies (★= PS invocation location)



VRCS shades first covered pixel, other pixels receive copies



Problem – VRCS introduces a 'half pixel offset'

Modify Shaders to HW VRS behaviour?

Problem could be corrected by sampling with an additional halfPixelOffset.xy, but

- Requires modification to every shader using VRCS
- Requires 2x extra long-life vector registers and extra math
- Not needed if running 1x1 rate...

Little enthusiasm*

Rotating Pixels

Per frame rotate which pixels are primary & converge?

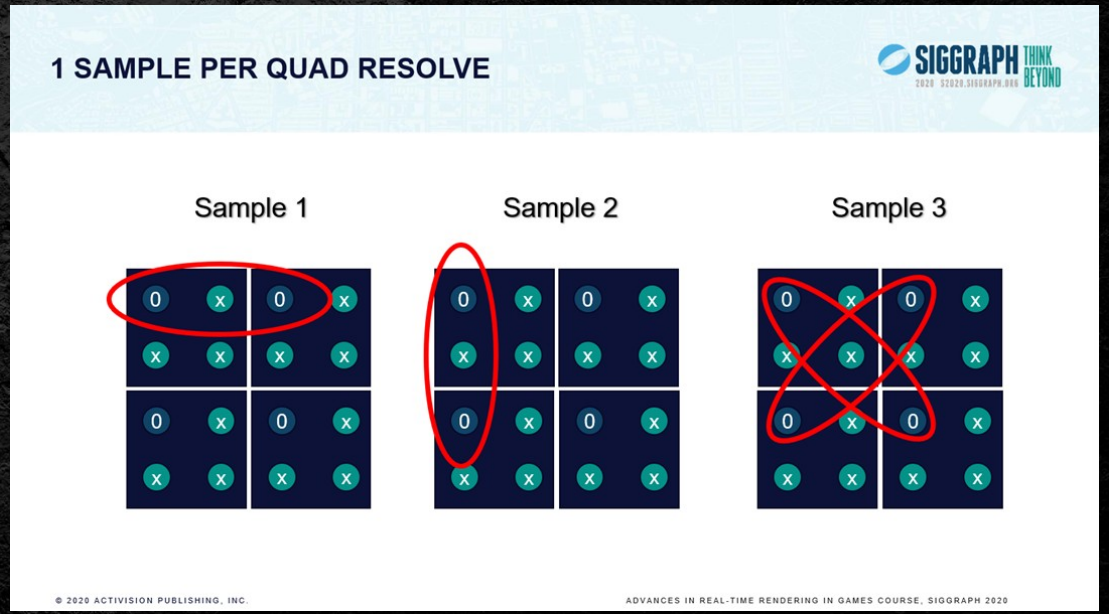
- Improvement for 1x2 and 2x1 rate, flip the checkerboard
- 2x2 rate requires 4 frames to converge = 15hz = too slow
  - Instead, alternate primary pixel between top left and bottom right
  - +ping pong sequence for 3 covered pixels

Improvement != fix

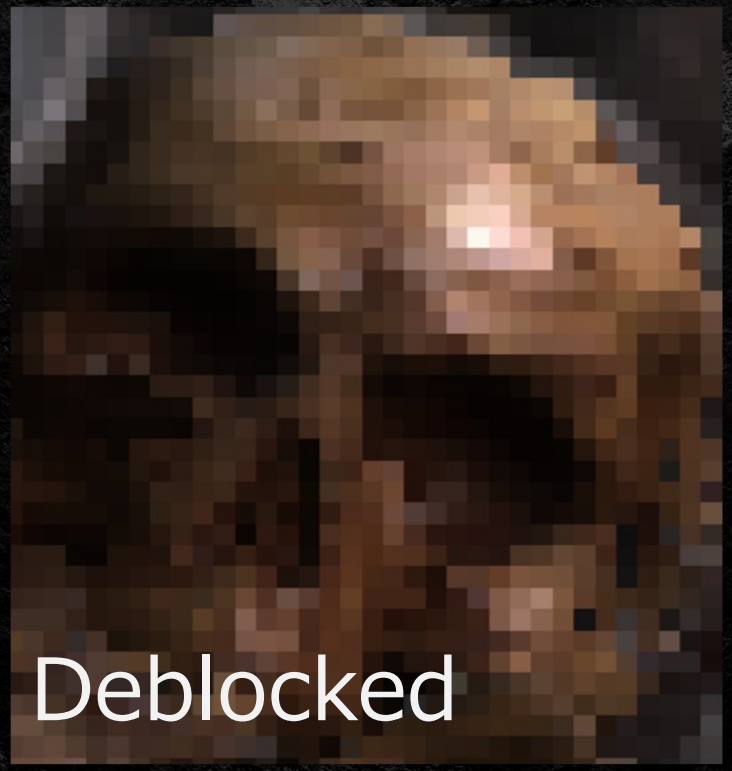
Activision's Deblocker - Michal Drobot Siggraph 2020

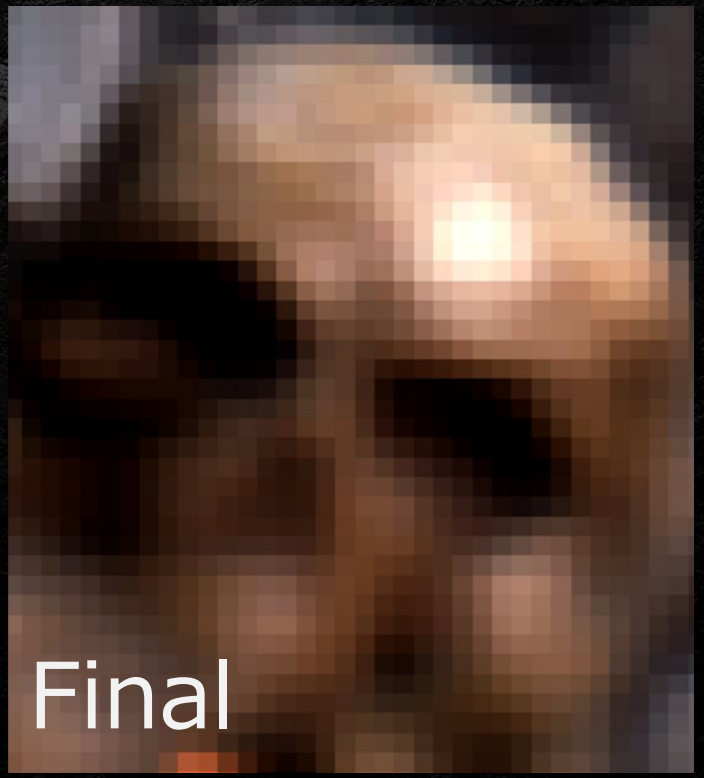
"Software-based Variable Rate Shading in Call of Duty: Modern Warfare."

- Execute a thread per entry in the VRCS coordinate buffer
  - 2x1 – average two pixels horizontally
  - 1x2 – average two pixels vertically
  - 2x2 – average 4 pixels diagonally
- Run before transparency
  - Solves half pixel offset



Results: XSS – Deblocker cost 97us @ 736p





Very little 1x1 rate on XSS at low resolutions, mostly half rate or 2x2

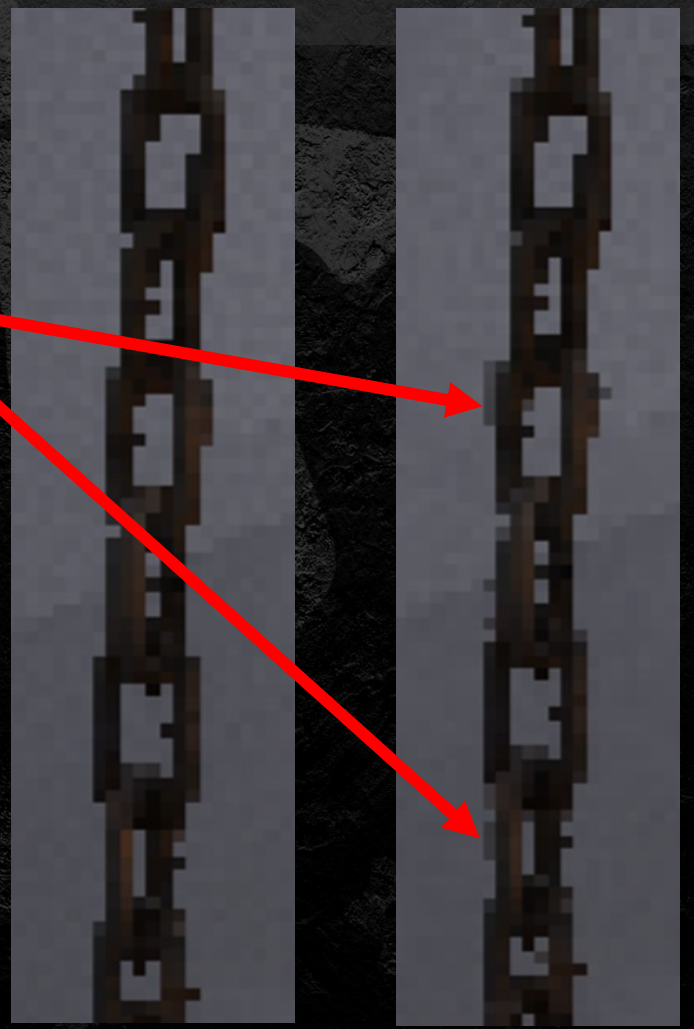
Coverage?

We let the deblock spill over triangle edges

- Incorrect for surface silhouette...

Did try respecting coverage:

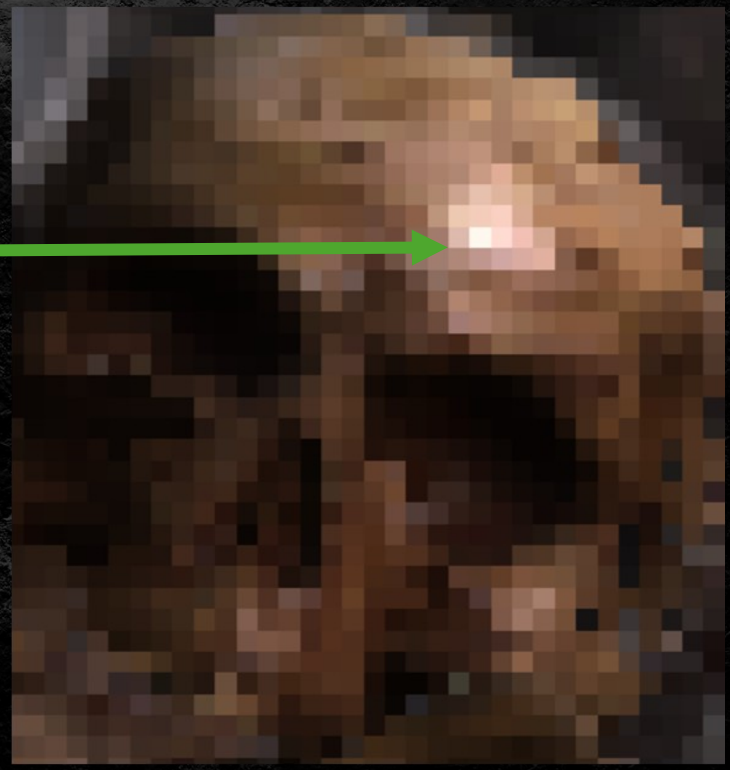
- Slow...
- Fixed silhouette, but interior problem



Coverage?

We let the deblock spill over triangle edges

- Incorrect for surface silhouette…
- But CORRECT for interior surfaces!
- e.g. deblock spec highlight across triangle edge



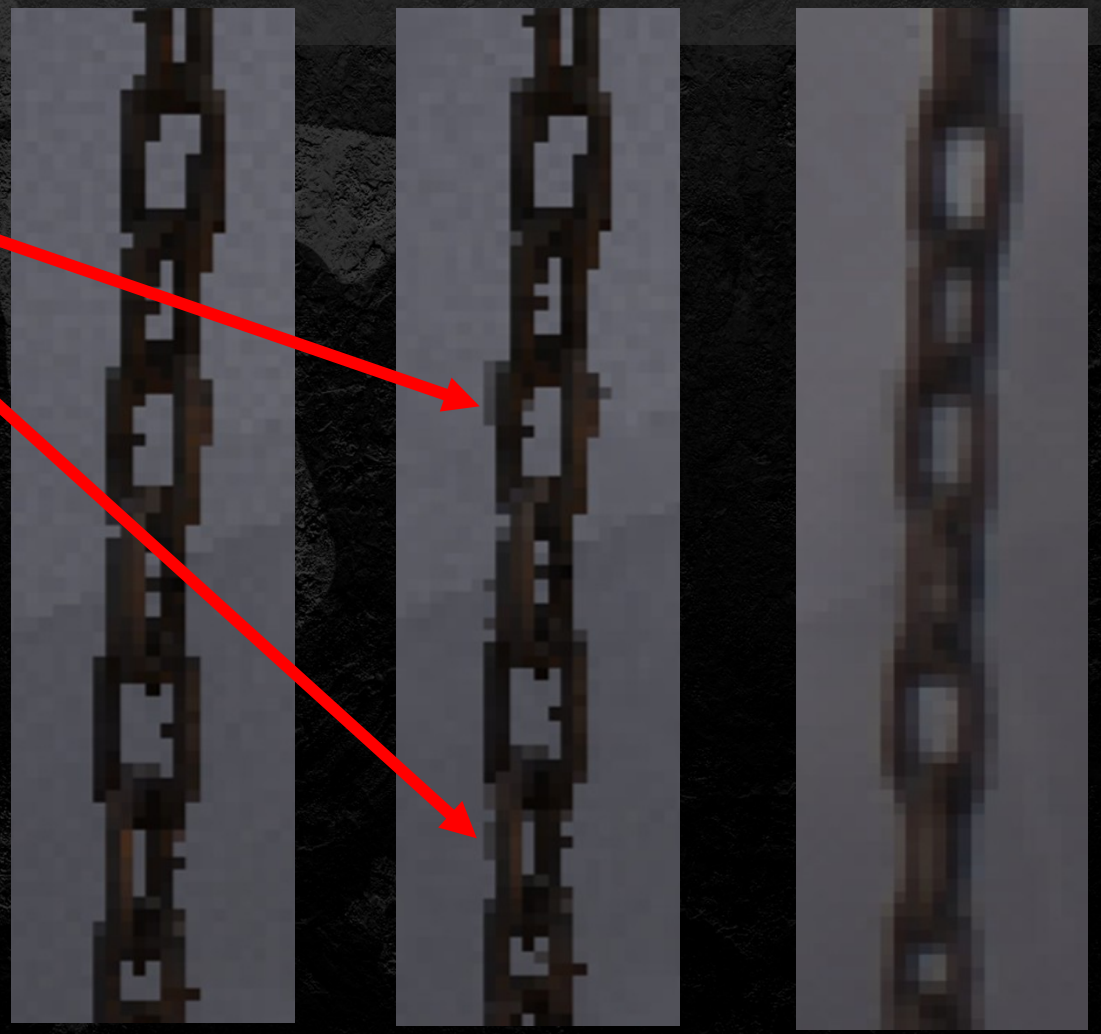
Coverage?

TAA fixes our sins!

- Or at least this sin...

Proper fix would be dissimilar depth

- But cost, shader B/W limited
- And time, ~5 weeks to ship, wasn't a priority...



Squeezing in Just One More Sin!

The Deblocker uses the same Input & Output UAV...

- Shader is bandwidth limited
- (also writes Luma for SRI generation)
- Same input & output buffer = big optimisation

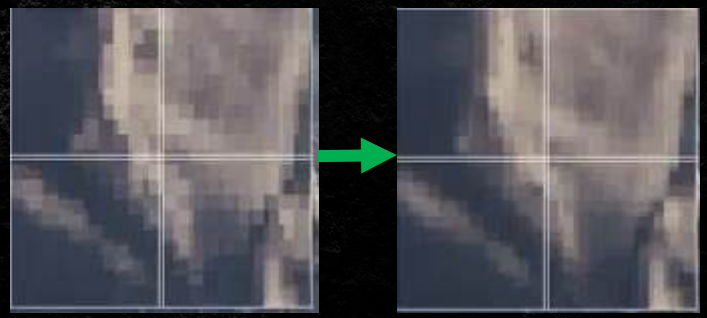
But

- Race condition - pixels changing value (output) which have yet to be used as inputs
- Imperceptible in practice

Activision's Deblocker Worked Extremely Well!

- Cheap!
- Fixed half pixel offset issue
  - No mod to existing shaders
- Allowed us to be WAY more aggressive on shading rate!
  - Especially on XSS
  - Especially at low DRS res

- Digital Foundry called out some blockiness
  - Strange, far distance only?!?
  - Oh... we accidentally turned off the deblocker for lightly fogged screen tiles....
  - Quickly fixed in post launch patch

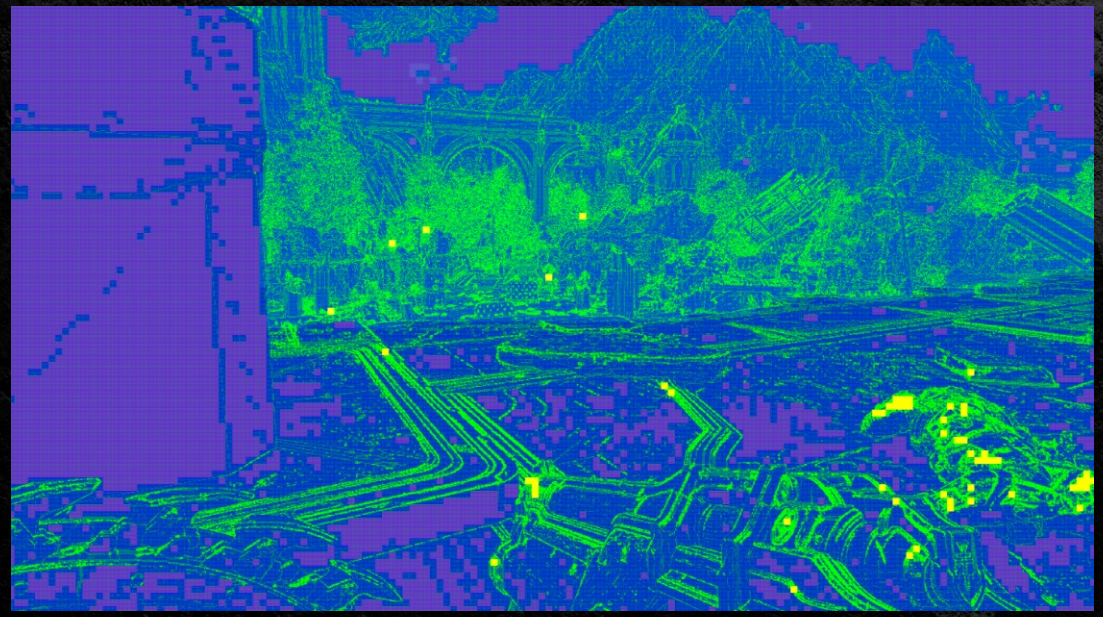


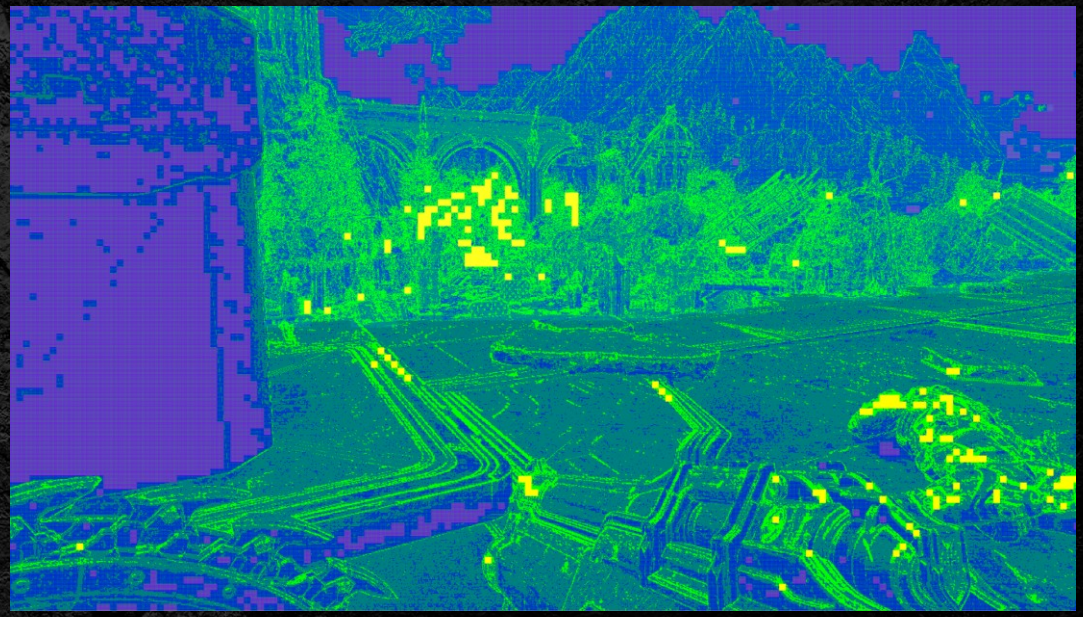
(The idTech8 Magnifying Glass!)

Noise is from Hell!

- idTech8 uses per-frame rotating blue-noise for 'Fog' and Shadow Cascade blending
  - Single pixel 'fizz' becomes objectionable at Macro scale (2x2 rate)
    - Randomness unpredictable by VRS/VRCS
    - TAA helps with single pxiel 'fizz', but not blown up to Macro scale
- Key observation - half rates are 'ok'
  - Added a LOT of 'extra half rate shading' to VRCS shading rate image (but not VRS image)
    - Compute shading rate as normal
    - If shading rate is 2x2, compute again, with reduced tolerances
      - But this time 1x1 is disallowed, 2x1, 1x2 or 2x2 only

Extra Half Rate Shading – But The Cost!





Also helps with SSR

Particularly sensitive to variation in normal direction

XSX 1440p

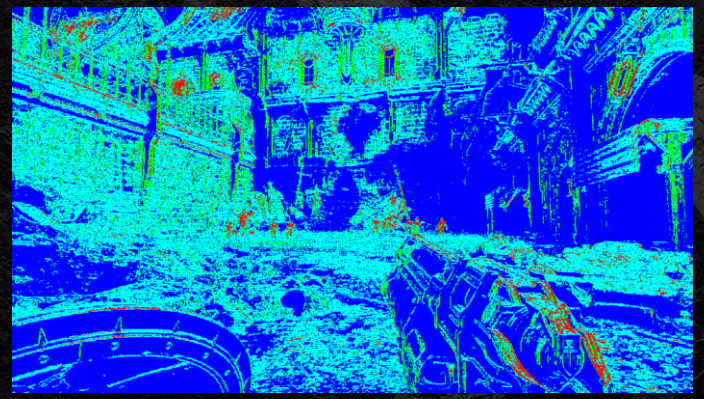
Normal Half Rate 17.4ms

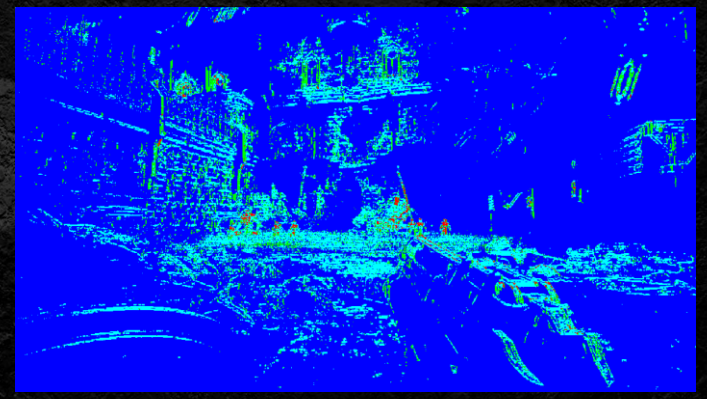
Extra Half rate 18.0ms

Clawing Back Performance

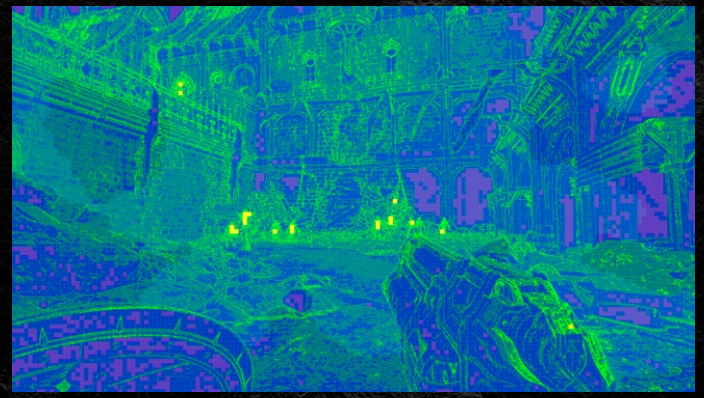
Because fixing the noise issue cost a lot of performance

Foveated Shading Rate linked to DRS (VRS & VRCS)

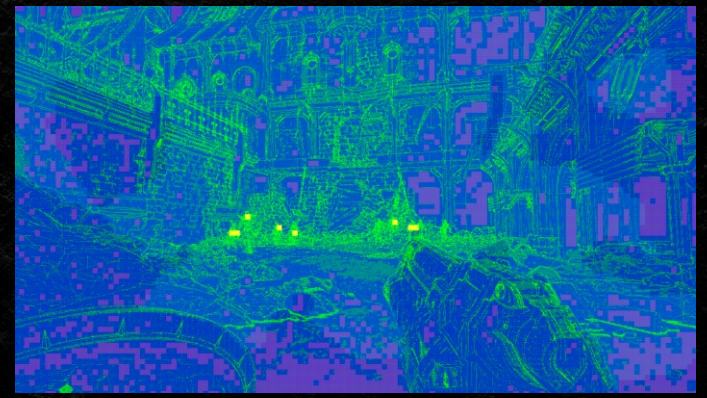




- Shading Rate linearly scales with DRS whole resolution range
- Foveated reduction kicks in at <= 75% res
- VRCS + DRS = large reduction in number of processed pixels!

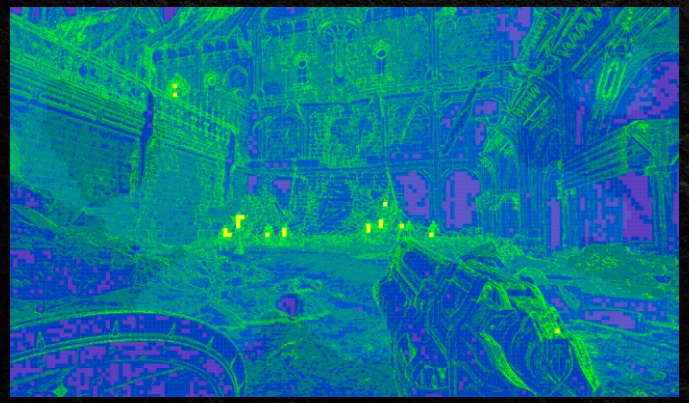


XSX 1440p

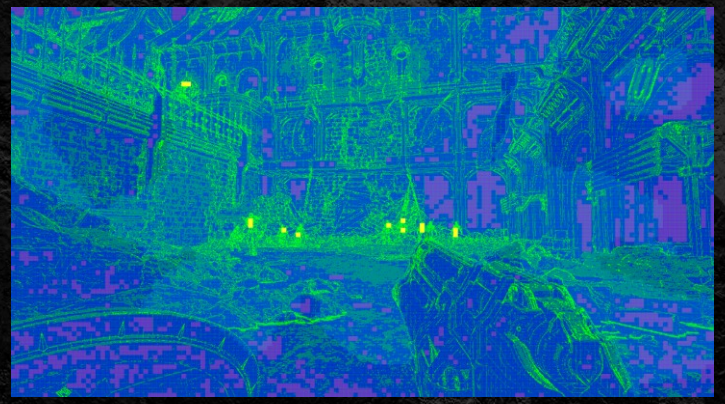


XSX 1170p

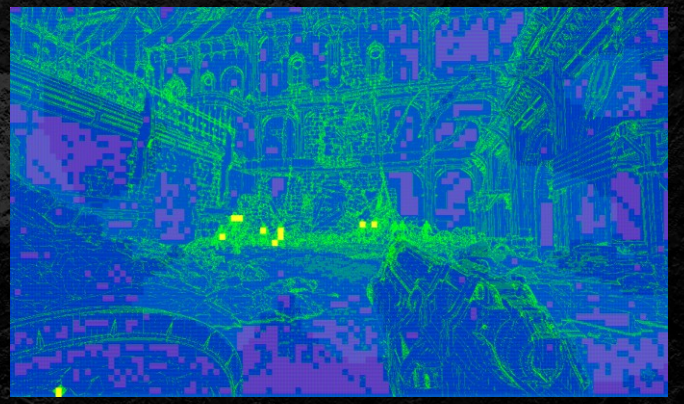
Foveated Shading Rate linked to DRS (VRS & VRCS)



XSX 1440p



XSX 1336p



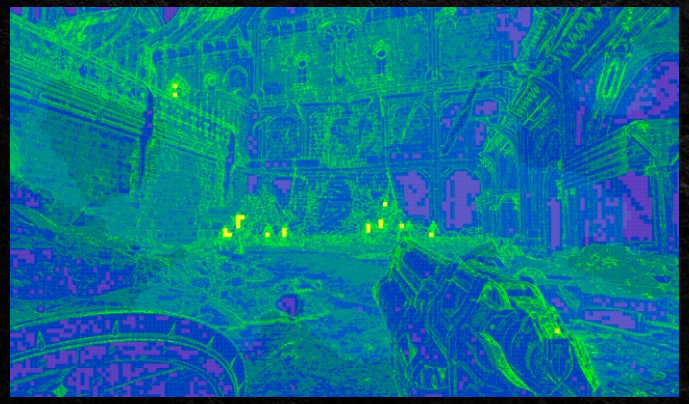
XSX 1020p

Q. What is VRCS worth?

A. Depends on resolution + we added a VRCS quality overshoot

- Above 85% resolution, increase VRCS quality above what we considered needed for image quality, because why not?
- Sometimes VRCS barely a win at max res, ok fine!

Best Wins are Not Necessarily at Min or Max Resolution



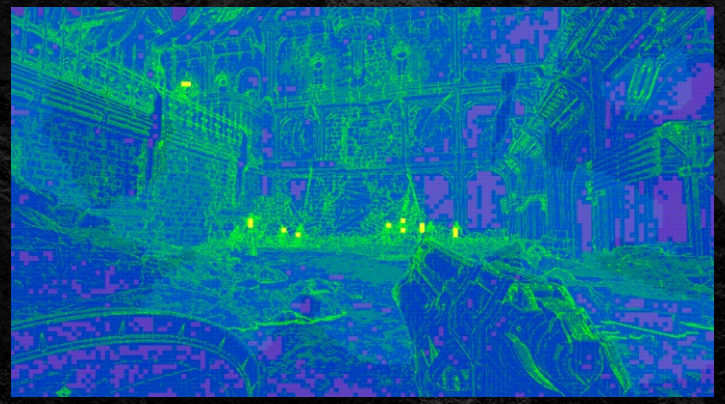
XSX 1440p

VRCS on 18.55ms

VRCS off 19.70ms

Saved 1150us

Quality overshoot active



XSX 1336p

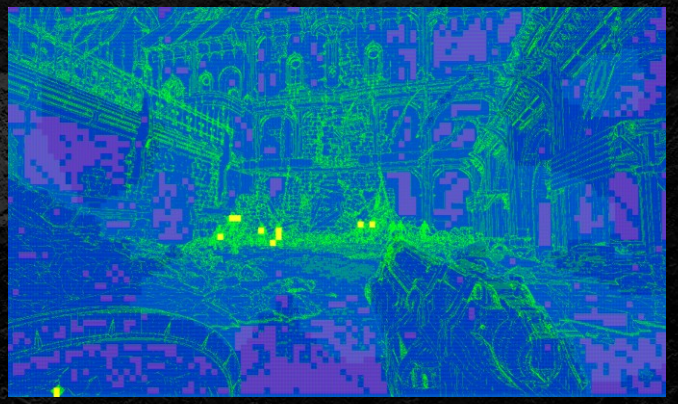
VRCS on 15.54ms

VRCS off 17.57ms

Saved 2030us

No quality overshoot

No foveated shading rate reduction yet



XSX 1020p

VRCS on 10.00ms

VRCS off 10.93ms

Saved 930us

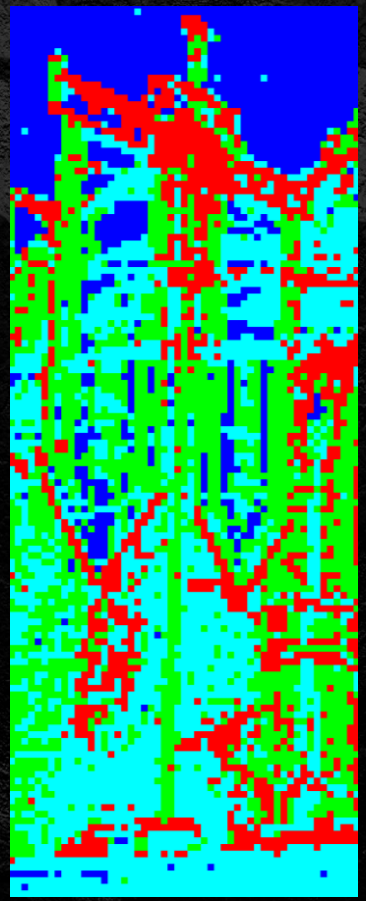
Full rate triangle edges limit win

Lowering resolution always provides diminishing returns

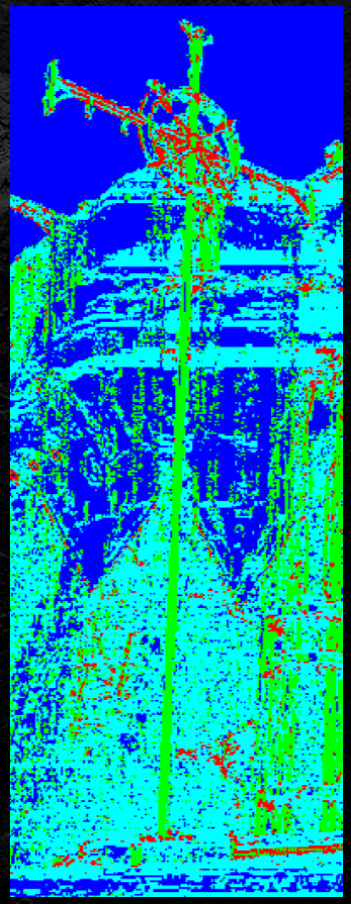
Transparencies Can Reduce Solid Shading Rate

- Luma stored twice
  - before and after transparent (R8G8)
  - VRCS shading rate computed twice
- VRS uses after transparent luma for shading rate
- VRCS uses the lowest of either shading rate
- Transparents can reduce shading rate on solid but cannot increase it
- = Profit!
- (Particularly in combat)

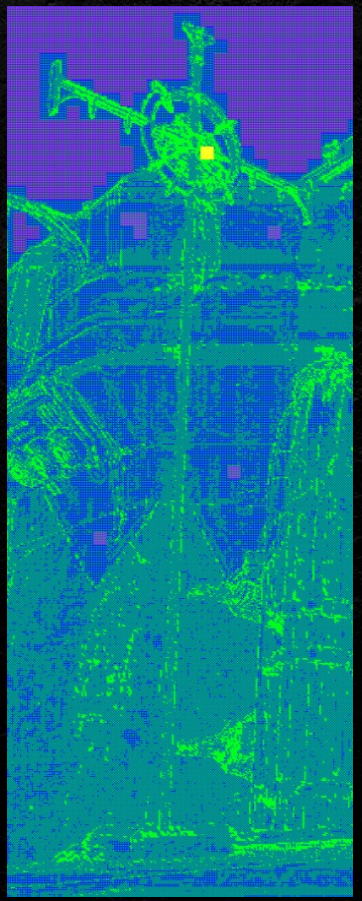
Final



VRS SRI

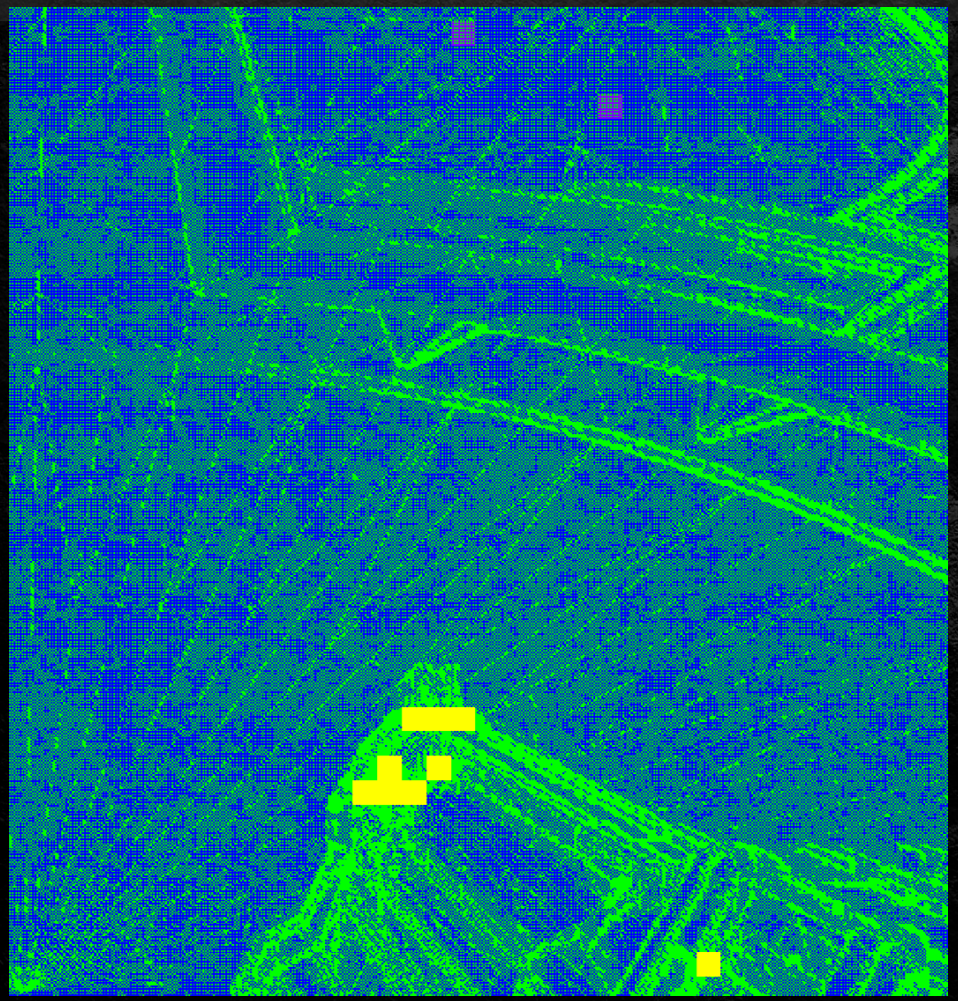


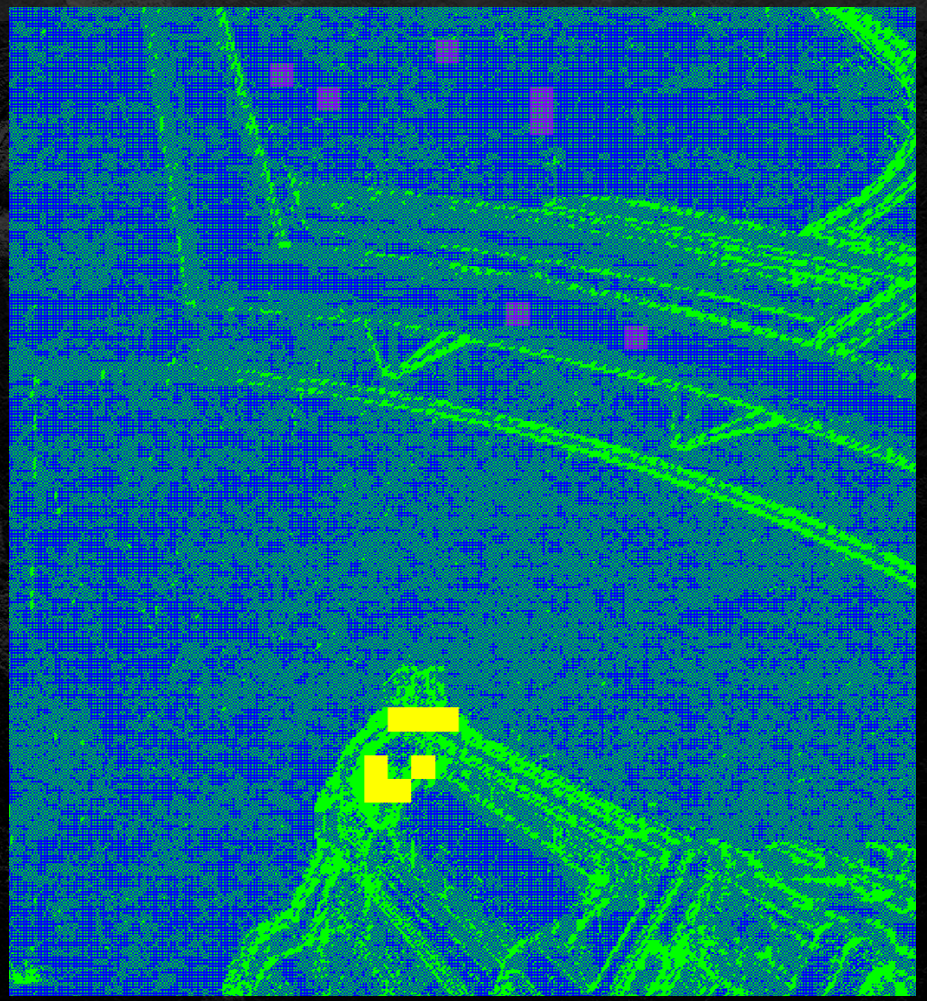
VRCS SRI



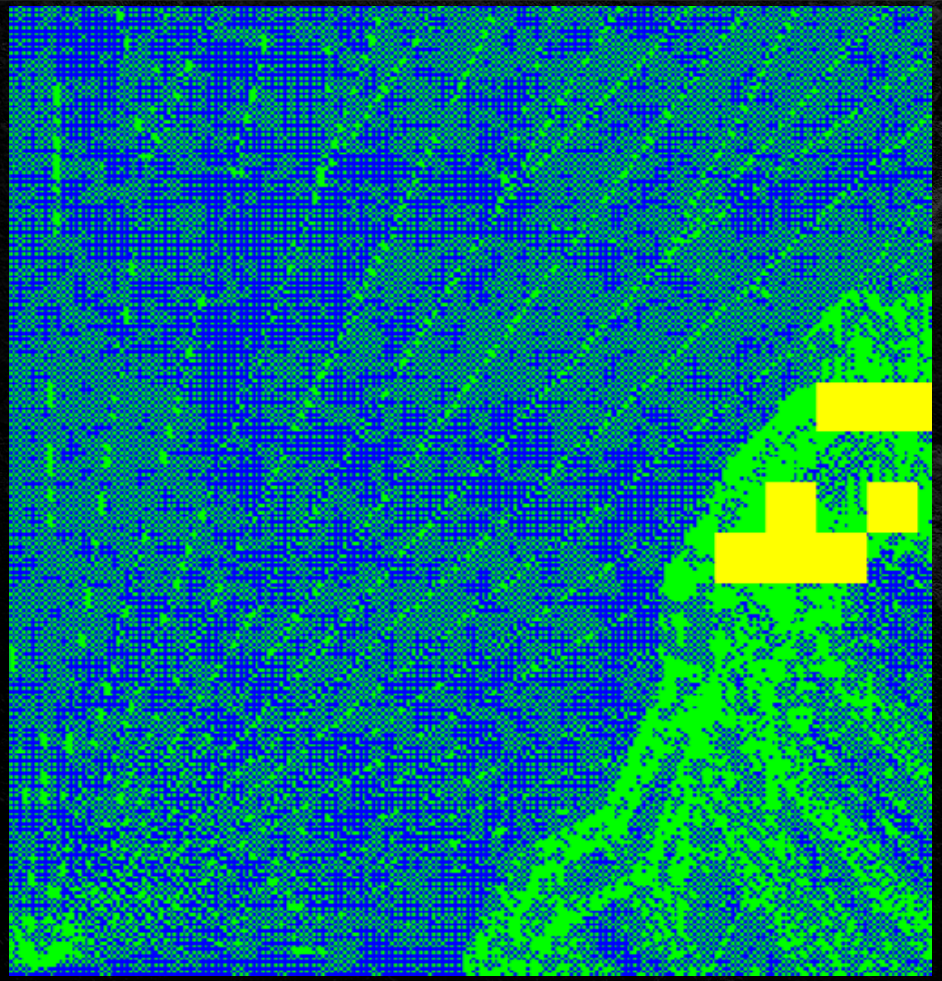
VRCS Buffers

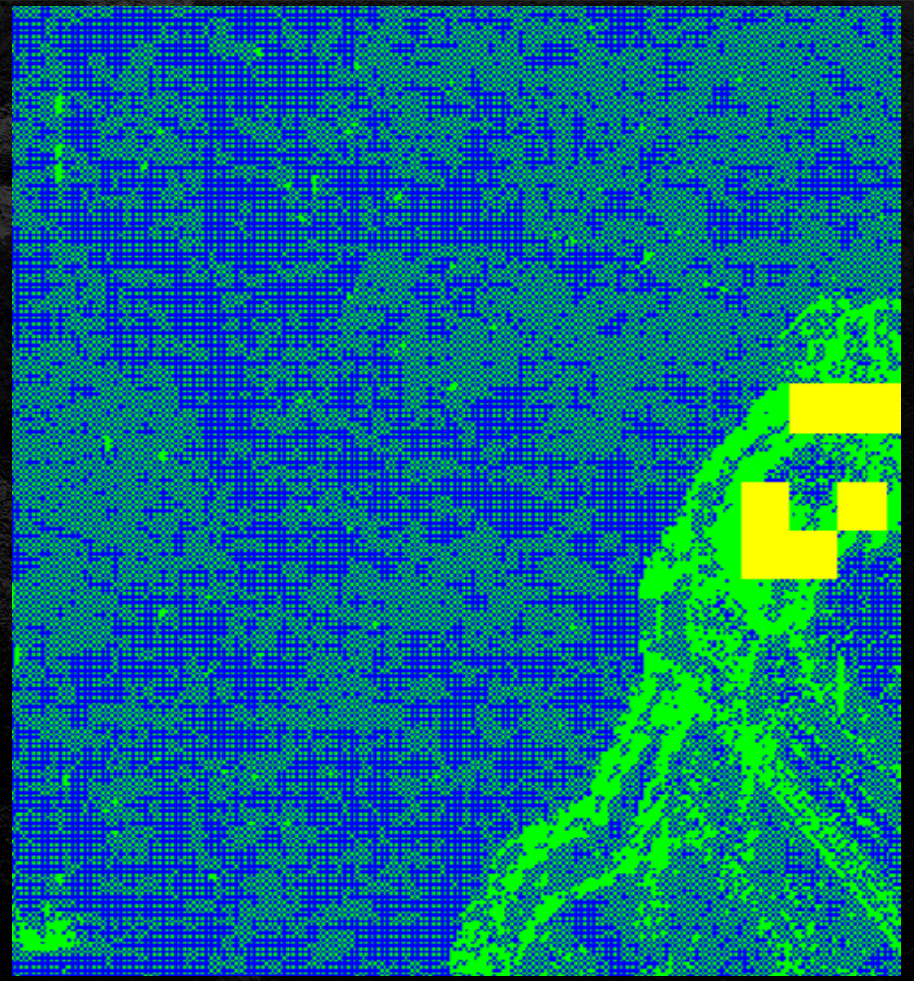
Remove Overshading of ~co-planar Triangle Edges





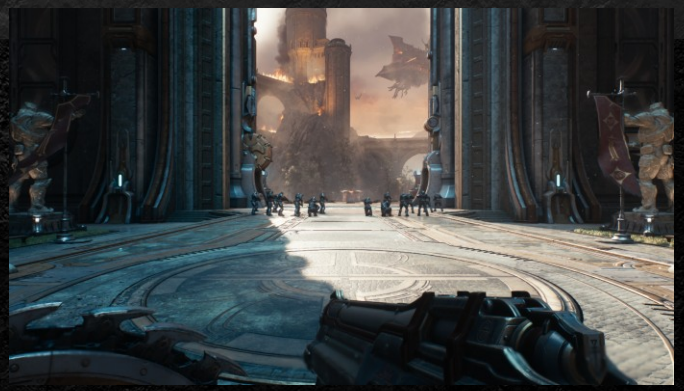
Zoomed in, in-case of Bad Projector Quality!

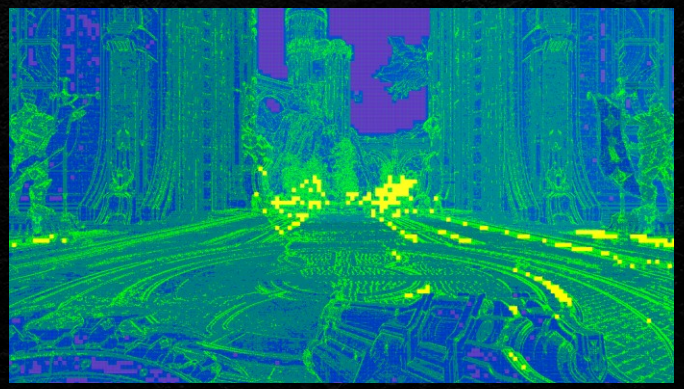




Remove Overshading of ~Co-planar Triangle Edges

- Dot product face normals and compare depth
  - Allow shared shading if close (and low rate)
  - Worked great except...
- Had to Reconstruct Face Normal
  - (Terrible 'knocked up in an hour' algorithm)
    - Load triangle ID + depth to 8x8 groupshared
    - Search for 3x3 for three identical triangle IDs
      - Cannot be in a straight line
    - Build face normal from 3x positions.xy+depth
    - Zero normal if not found





XSX 1440p

Normal VRCS time 18.10ms

Overshade reduced 17.95ms

Other Performance Wins

- Increased Shading Rate tolerance in 'Safe Area'
  - Border screen tiles have slightly lower shading rate = v.small win
  - Not forced 2x2, because noise...
- VRCS Tile Flags: (stored or derived from tile count of primary pixels)
  - 1x1 – process every pixel, pixel.xy = SV_DispatchThreadID
  - 2x2 – tile is entirely 2x2, pixel.xy computed, not loaded from buffer
  - Sky – tile is entirely sky, early out if shader doesn't want to process sky (Sky is not necessarily 2x2)
  - Fog – tile is fogged, composite shader runs 1x1 rate, other shaders run with stored pixel.xy (unless additionally flagged 1x1 or 2x2)

Fog Tile Flag Optimisation

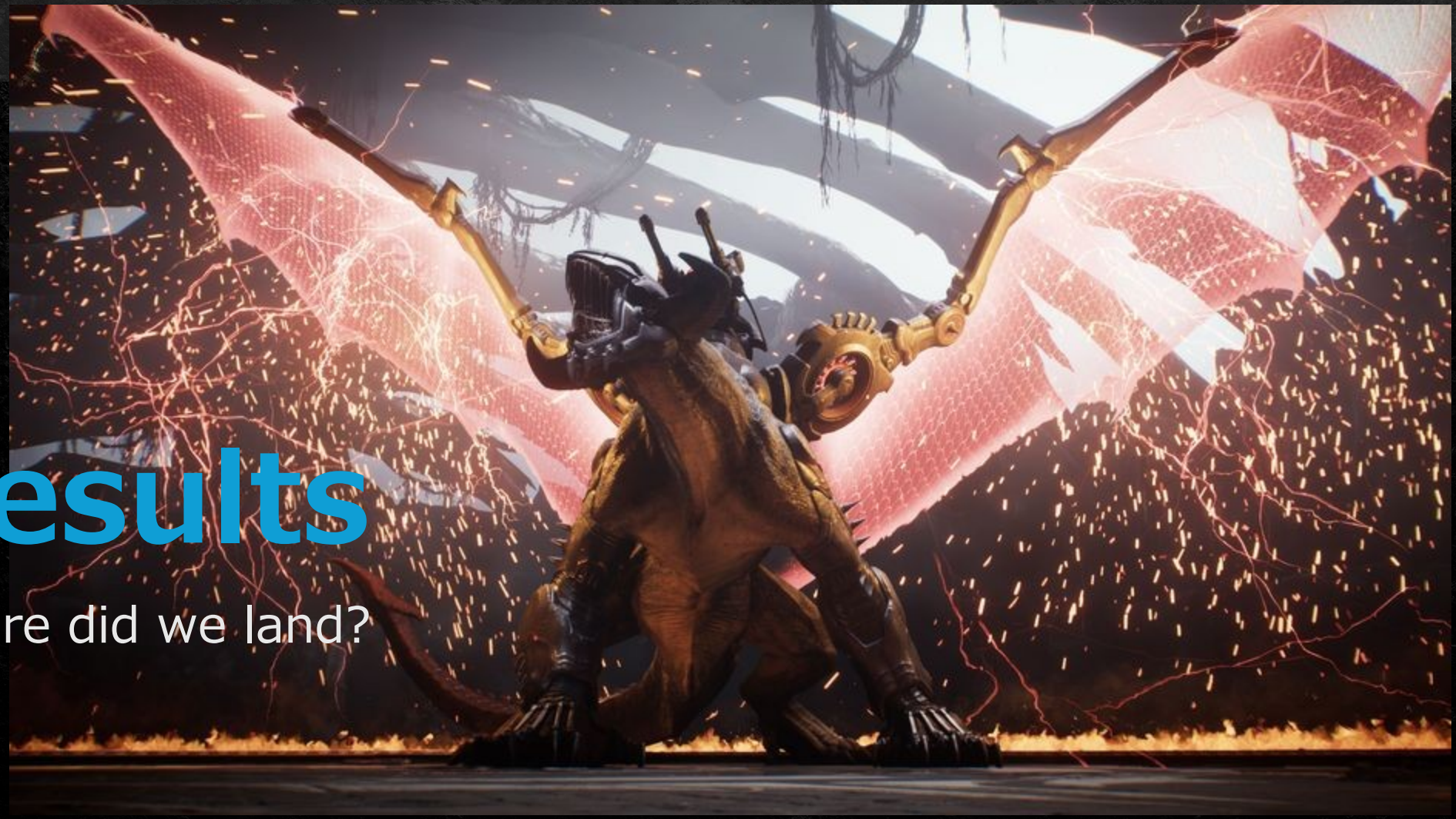
Fog Volumes are 3D

Building VRCS buffers runs 1 thread per 2x2 pixels

- Fog volume sampled once per thread = half res
- Still expensive though....

Post launch ended up dropping VRCS for the composite pass – which also calculates fog

- Meh,
- But - reclaim expense of calculating fog tile flag



Results

Where did we land?

Perf Numbers

Case with good gains

scarlett m2 rs 85%	FRAME	D-TEX	D-GBU	D-LGHT
VRCS on (ms)	13.64	1.24	0.63	1.245
VRCS off (ms)	15.74	2.196	1.05	2.04
delta (ms)	2.1	0.956	0.42	0.795
perf gain (%)	13.34%	43.53%	40.00%	38.97%

Case with average gains

scarlett m2a rs 85%	FRAME	D-TEX	D-GBU	D-LGHT
VRCS on (ms)	15.9	1.473	0.897	1.352
VRCS off (ms)	17.7	2.287	1.118	2.053
delta (ms)	1.8	0.814	0.221	0.701
perf gain (%)	10.17%	35.59%	19.77%	34.15%

Worst VRCS Gains

Always foliage!
1336p XSX
600us win
85% res

Worst scene I found
(non-exhaustive sampling, but DF called out the same area)

VRCS Performance Figures

- Xbox Series X|S and Playstation5
  - Huge gains in VRCS enabled passes
    - Frequently 33% saving for deferred texturing and lighting
  - Typically, 1-2ms overall GPU frame perf gain
  - Depends on content and DRS scale
    - Best gains at 85% resolution, by design
    - Worst gains in foliage heavy areas
- ROG Xbox Ally X
  - Similar gains without costly upscaler (simple TAA) as on Xbox/PS5
  - No great VRCS win with FSR upscaling
    - Cost of FSR pushes down internal resolution
    - VRCS win dependent on finding duplicate pixels
      - Overshading of triangle edges problem becomes much worse at low resolutions

Future Potential

Buckle up, there is a lot!

1. VRCS to Accelerate Ray Tracing

- Use VRCS to fire fewer rays originating in the view frustum
  - Reflections
  - AO
  - Shadow rays
  - Screen space probes
- Totally viable, potentially **MASSIVE** saving
  - As normal - fire rays from primary pixel, copy results
  - Or - reduced ray count per pixel
    - Non-binary use of VRCS!

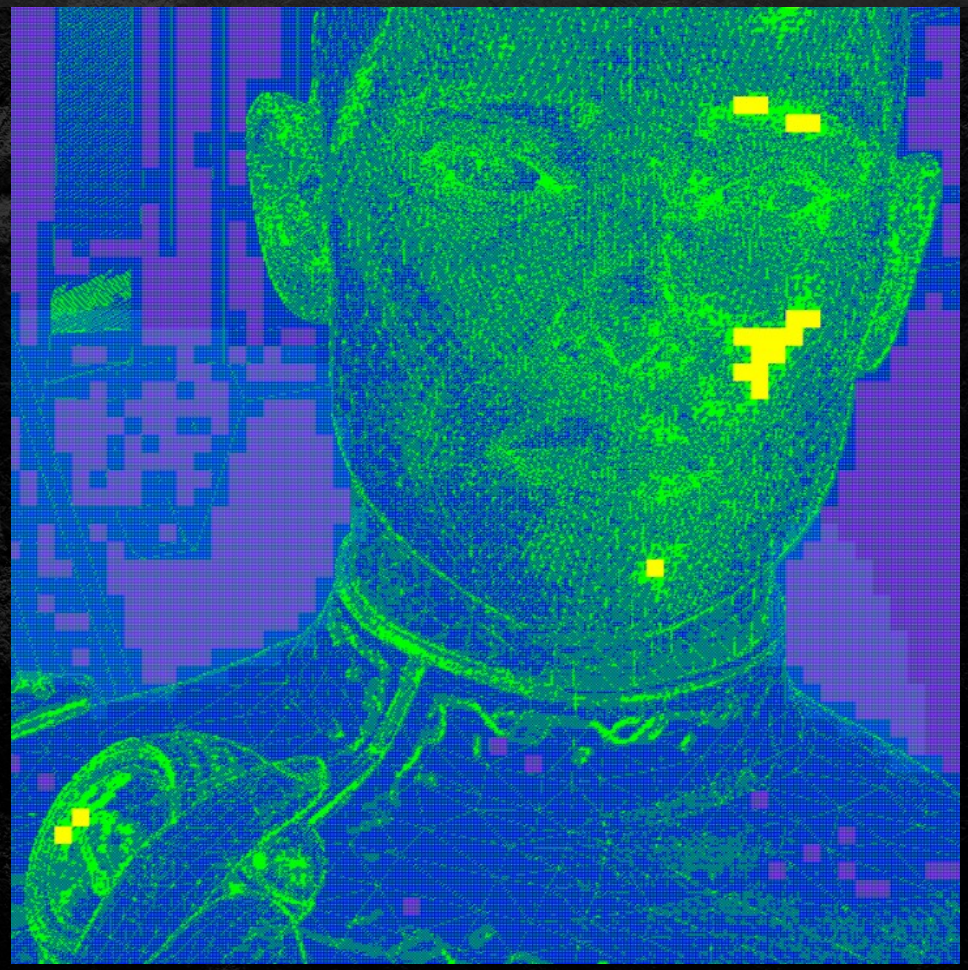
```
reducedNumRaysPerPixel = numRaysPerPixel / (1 + numPixelCopies);
```

    - (for GPUs that can fire multiple rays per pixel...)

2. Reduce Internal Shading of Triangle Edges

- Reconstructing face normal from adjacent samples ‘worked’
  - Requires finding three samples from the same triangle + precision issues
  - Not great, not terrible
- Improvements:
  - Store face normal with visibility buffer?
    - `OctEncodeLowPrecPosZOnly(cross(ddx(SV_Pos), ddy(SV_Pos)));`
  - Or output with barycentric buffer?
    - Pass accesses triangle data – visible triangles only
    - Tangent frame normal?
- We can do even better!

Unwanted Extra Shading on 'Internal Edges'



E.g.

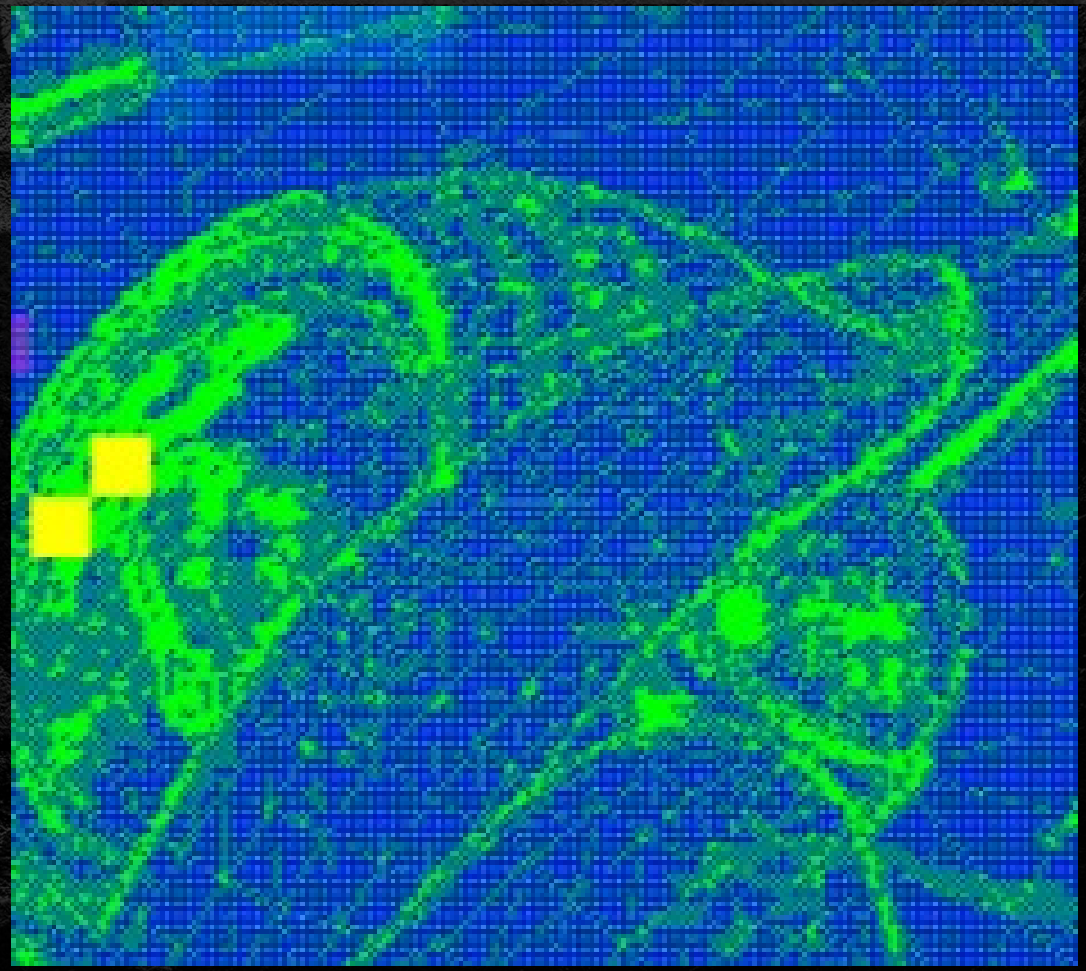
1. Face
2. Fingers
3. Cloak

XSX 1440p

Regular 13.9ms

VRCS 12.2ms

Not Co-planar Triangles, but 'Same Surface' Triangles



Solution : Stop using TriangleID !

Triangles are an artificial representation of the true geometry of a **surface**, we actually care about:

1. Increase quality on surface silhouette (where MSAA wins)
2. Surface is continuous (depth check)
3. Shading detail/lighting etc.. (shading rate image)

E.g.

```
shareShading = lowRate && zClose && (surfID == adjSurfID)
```

Would significantly reduce unnecessary additional shading

- Foliage, cloth, skin, floors, walls, any smooth surface!
- Particularly helpful at low resolution / detailed meshes / small triangle sizes

Tooling problem, how to assign surfaceID?

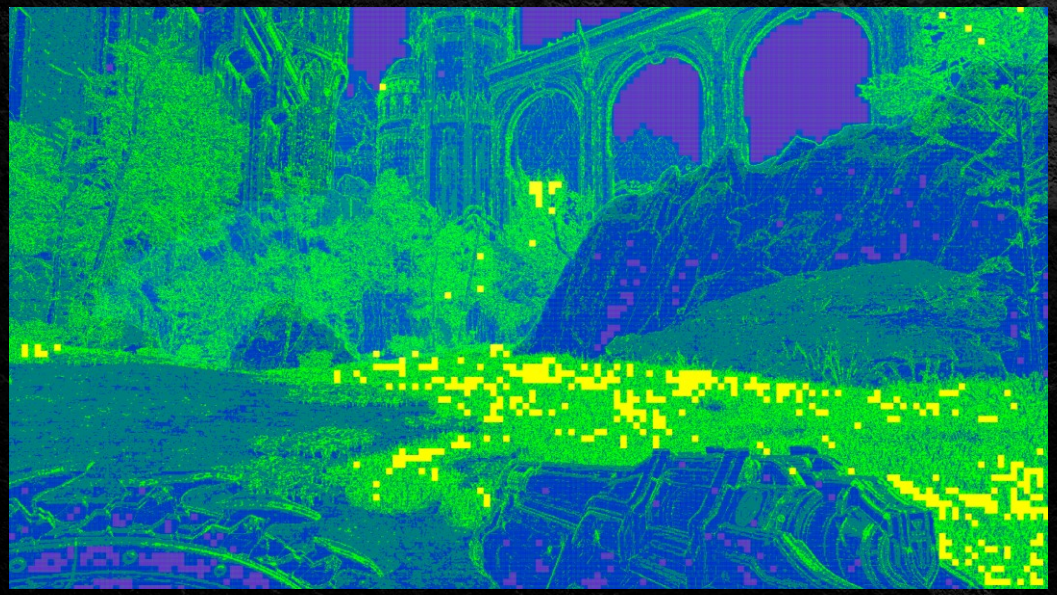
You could use VRCS without a Visibility Buffer!

- G-buffer SurfaceID

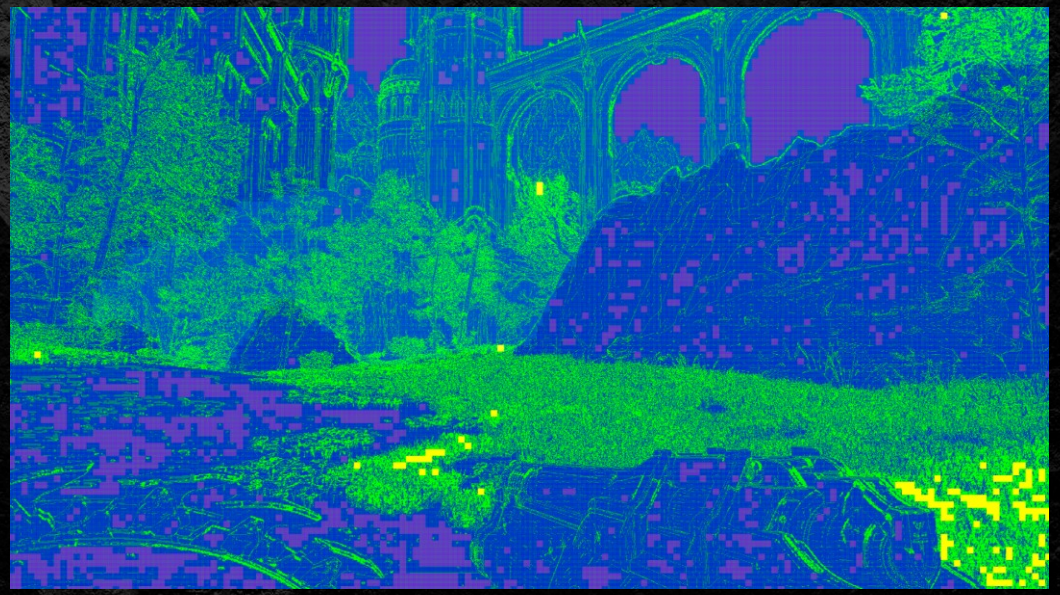
3. VRCS on Half Res?

- SSDO is half res, and expensive (~1.3ms)
  - Half res shading rate image
    - Additional output from current shader
  - Half res VRCS coordinate buffer
    - Simplified VRCS buffer gen shader
    - Requires half res visibility buffer or custom sample
    - Hide cost on async compute ofc!
  - No Deblocker?
    - Result is blurred anyway
  - Would VRCS be a win?
    - Guestimate net 100-200us saving with triangleID's
    - Guestimate net 250-300us saving with surfaceID's
    - (Triangles a lot smaller at half res! Surfaces still relatively large)

4. Foliage: Reduce/Remove Extra Half Rate



With Extra Half Rate



Fullscreen No Extra Half Rate

How much extra half rate on Foliage?

1. Less? Relax shading rate tolerance
2. None? :-)

XSX 1440p

Extra Half rate 17.2ms

No extra Half Rate 16.5ms

5. Sending Noise back to Hell!

Potential solutions:

- Produce a screen space mask of where noise is being applied (does not need to be binary)
  - e.g. where shadow cascades are transitioning
  - Next frame - boost half rate shading only on masked areas
- Disable extra half rate on certain materials
  - e.g. foliage
- Disable extra half rate when the camera is moving fast
  - Foveated?
  - (If only we had captured motion blur in the Shading Rate Image)

(Or remove use of noise..? #JustSaying...)

6. If We Can't Fix the Noise Issue?

- Observation – noise added in lighting pass, not texturing
- Per-pass SRI and VRCS buffers?
  - Deferred texturing – no half rate boost
  - Deferred lighting – half rate boost
- Problem?
  - Reflections particularly sensitive to normal direction
  - SSR runs thread per pixel - no VRCS
    - But it does use duplicate g-buffer normals
    - Not great, not terrible
- Solution:
  - Output normal g-buffer thread per pixel – no VRCS
  - Run VRCS for the rest of the shader, waves retire
  - Only for screen tiles with reflective materials - re-project from last frame

7. Post Process Blurs

Shading Rate Image doesn't know about:

- DOF, Motion Blur, Bloom etc..
- (Major VRS win on other titles => increased internal resolution)

Why?

- idTech8 overlaps frames => GREAT optimisation!
- But... post blurs not reliably done by the time Shading Rate Image required...

Solution?

- Coarse DOF calculation in Shading Rate generation shader?
  - Small VALU cost => but only when DOF is turned up high = profit!

VRCS Conclusions

- 1-2ms saving is huge for a 60hz title!
  - Increased internal resolution on PS5/XSX/XSS
  - Required on XSS to hit frame time+resolution targets
  - Worked well on ROG Xbox Ally X only with a cheap upscalar (same as console)
  - Best wins require a reasonable resolution
  - Deblocking is essential
- We have only scratched the surface!
  - Apply to RT and half resolution passes
  - Improvements
    - Pixel selection algorithm - surfaces not triangles
    - Better fix for the noise issue - reduce half rate shading boost
    - Foliage - remove internal triangle edges and remove/reduce half rate boost

Thank You For Listening!

We are Happy to Take Questions!

Acknowledgements

Thanks to the amazing team at idTech, especially for being open to big changes very late in the day!

idTech engineers:

Allen Bogue, Billy Khan, Bogdan Coroi, Carson Fee, Dominik Lazarek, Ian Malerich, John Roberts, Jean Geffroy, Johan Donderwinkel, Mel-Frederic Fidorra, Oliver Fallows, Dr. Peeter Parna, Philip Hammer, Regan Carver, Seth Hawkins, Stefan Pientka, Thorsten Lange, Tiago Sousa and Yixin Wang

id Software leadership (Marty Stratton, Hugo Martin)

Everyone else at id Software, ZeniMax and Xbox

DOOM THE DARK AGES

Bethesda ZeniMax MEDIA INC.

References

"Variable Rate Compute Shaders - Halving Deferred Lighting Time", Martin Fuller. Microsoft Game Dev YouTube Channel, 2022, https://www.youtube.com/watch?v=Sswuj7BFjGo

"Variable Rate Shading Update Xbox Series X|S", Martin Fuller, Philip Hammer, Christopher Wallis. Microsoft Game Dev YouTube Channel, 2022, https://www.youtube.com/watch?v=pPyN9r5QNbs

"Variable Rate Shading, A Deep Dive", Martin Fuller. GDC 2019. https://www.youtube.com/watch?v=2vKnKba0wxk.

"Software-based Variable Rate Shading in Call of Duty: Modern Warfare", Michal Drobot. SIGGRAPH 2020. https://research.activision.com/publications/2020-09/software-basedvariable-rate-shading-in-call-of-duty--modern-war

"Rendering the Hellscape of DOOM Eternal" Jean Geffroy, Axel Gneiting, and Yixing Wang. SIGGRAPH 2020. https://advances.realtimerendering.com/s2020/RenderingDoomEternal.pdf

"Visibility Buffer and Deferred Rendering in DOOM: The Dark Ages", Philip Hammer and Dominik Lazarek. Graphics Programmer Conference 2025

Bonus Slides

Results: XSX – Deblocker cost 119us @ 1320p

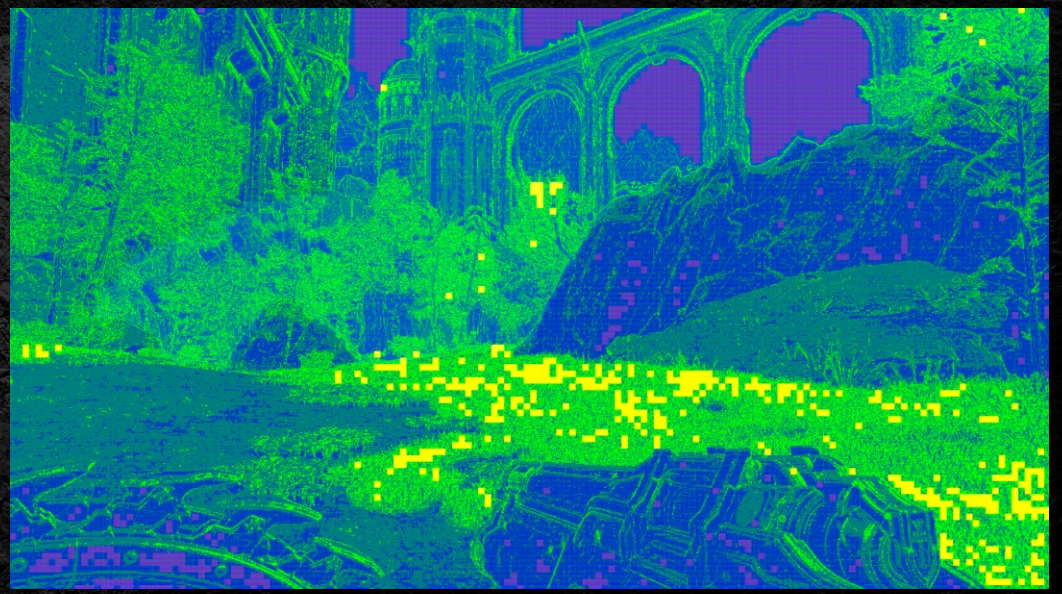
Future Potential

Buckle up, there's even more!

8. Reduce Single Buffer Coordinate Count

Build single compact VRCS coord buffer at the same time as tile relative VRCS coord buffer

- Tile relative coord buffer only stores max 224 entries (256-32) per 16x16 tile
  - Otherwise tile based system is not saving a wave, process all 256 pixels
  - (Yellow tiles in debug view)
- Leads to unwanted additional processing when executing single list per screen



Foliage heavy scenes generated the most full rate tiles, though that could be improved

Additional Minor Perf Wins Available

- Utilise 21 spare bits for linearZ
  - Tile relative coords have 21b free – enough for unorm linearZ (near plane to far plane)
  - Reading coord anyway, no point also reading depth and linearizing?
- VRCS buffer of tile count + tile flags is 32b
  - R8G8_UINT texture would be enough, 8b count | 8 flags
  - RAM saving + bonus saving on bitwise OPs
    - (yes, we are that obsessed)
- Optimise SRI and VRCS coordinate gen shaders (didn't get time!)
  - Removing face normal reconstruction hack + fog flag will help a lot
- Read smaller mip level for primary pixels with duplicates
  - HW VRS does this automatically, VRCS does not
  - Doesn't need sample divergence, Deferred Texturing can tweak length of gradients