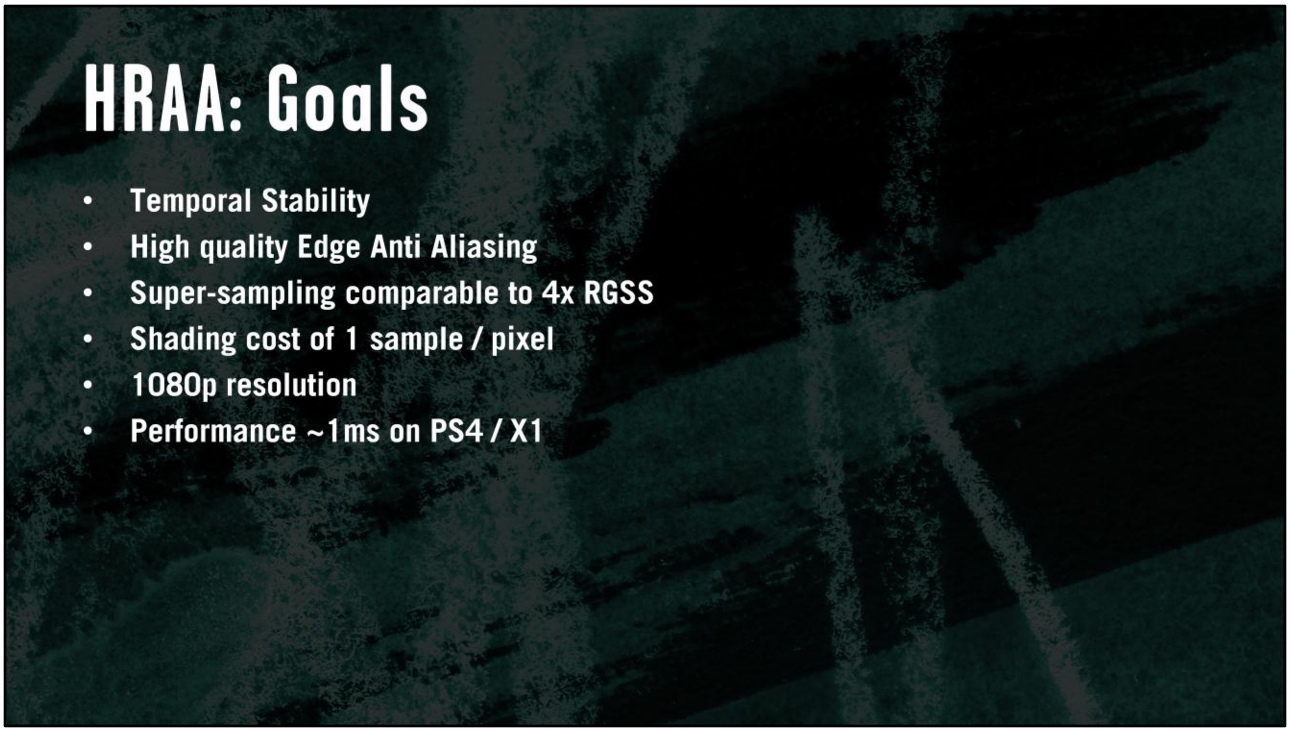
MICHAL.DROBOT

3D @ FAR CRY 4

HYBRID RECONSTRUCTION ANTI ALIASING

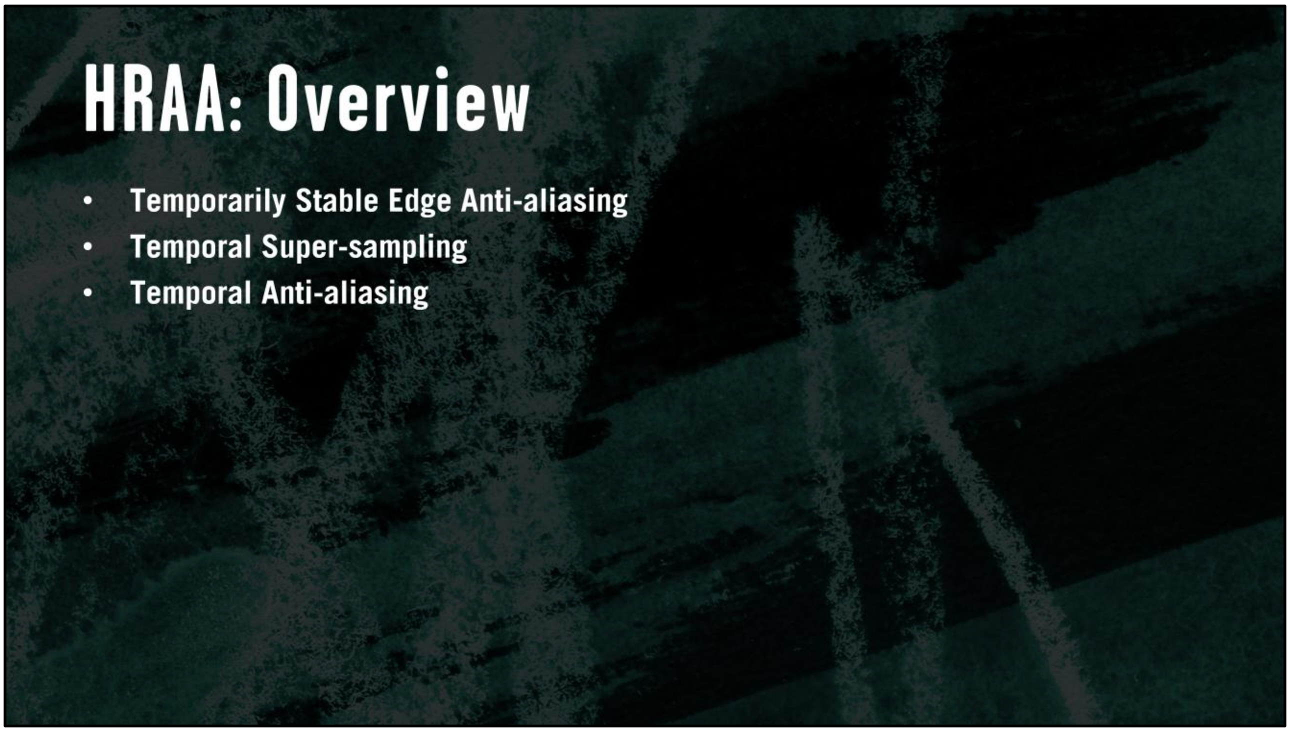
UBISOFT ENTERTAINMENT

2014-08-05



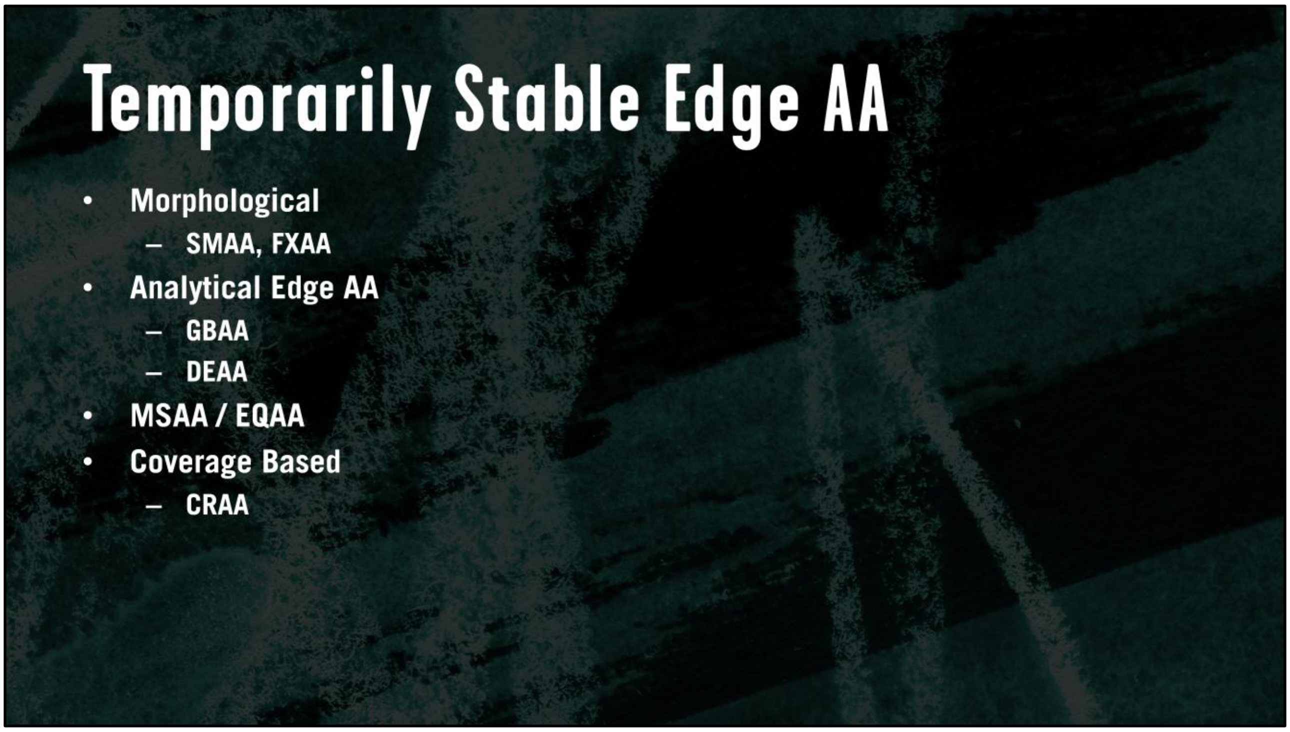
HRAA: Goals

- Temporal Stability
- High quality Edge Anti Aliasing
- Super-sampling comparable to 4x RGSS
- Shading cost of 1 sample / pixel
- 1080p resolution
- Performance ~1ms on PS4 / X1



HRAA: Overview

- Temporarily Stable Edge Anti-aliasing
- Temporal Super-sampling
- Temporal Anti-aliasing



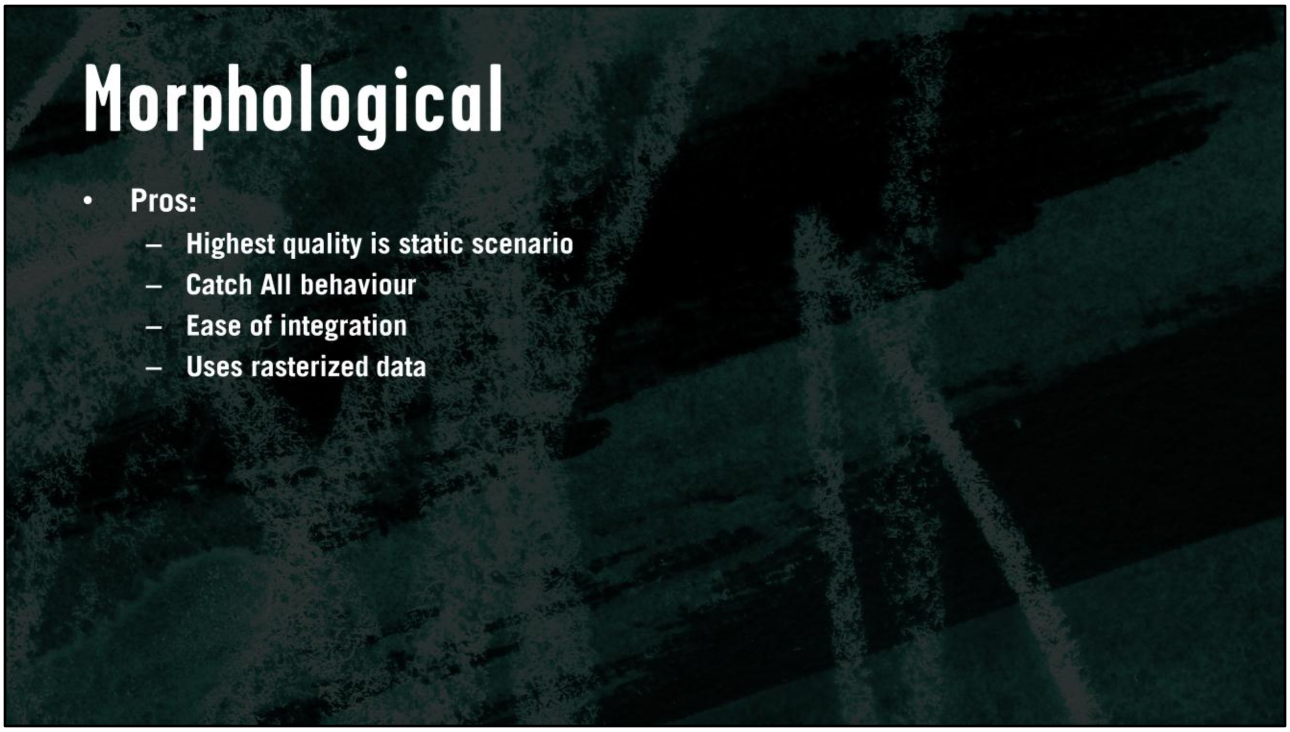
Temporarily Stable Edge AA

- Morphological
  - SMAA, FXAA
- Analytical Edge AA
  - GBAA
  - DEAA
- MSAA / EQAA
- Coverage Based
  - CRAA

Morphological:
Sub-pixel Morhpological Anti-Aliasing [Jimenez 11]
Fast AproXimatte Anti Aliasing [Lottes 09]

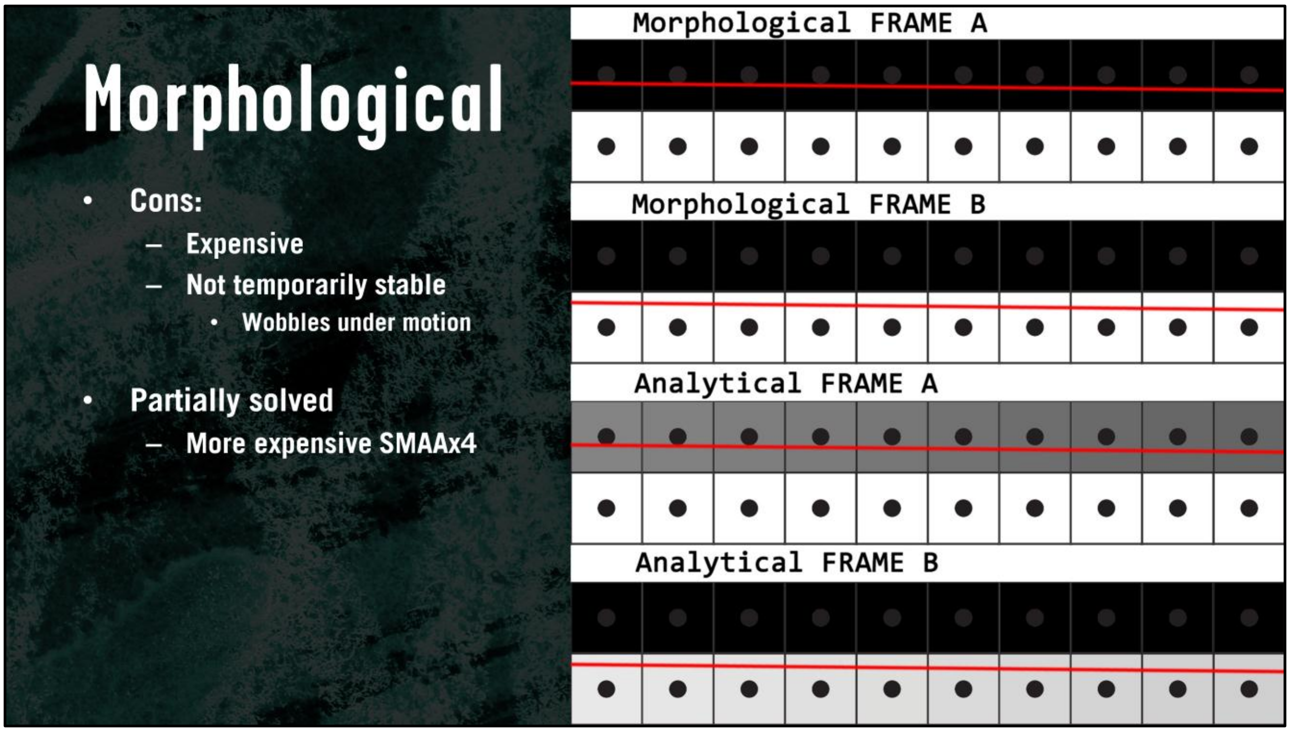
Analytical:
Geometric Buffer Anti Aliasing [Persson 11]
Distance to Edge Anti Aliasing [Malan 10]

Multi Sampled Anti Aliasing
Enhanced Quality Anti Aliasing



Morphological

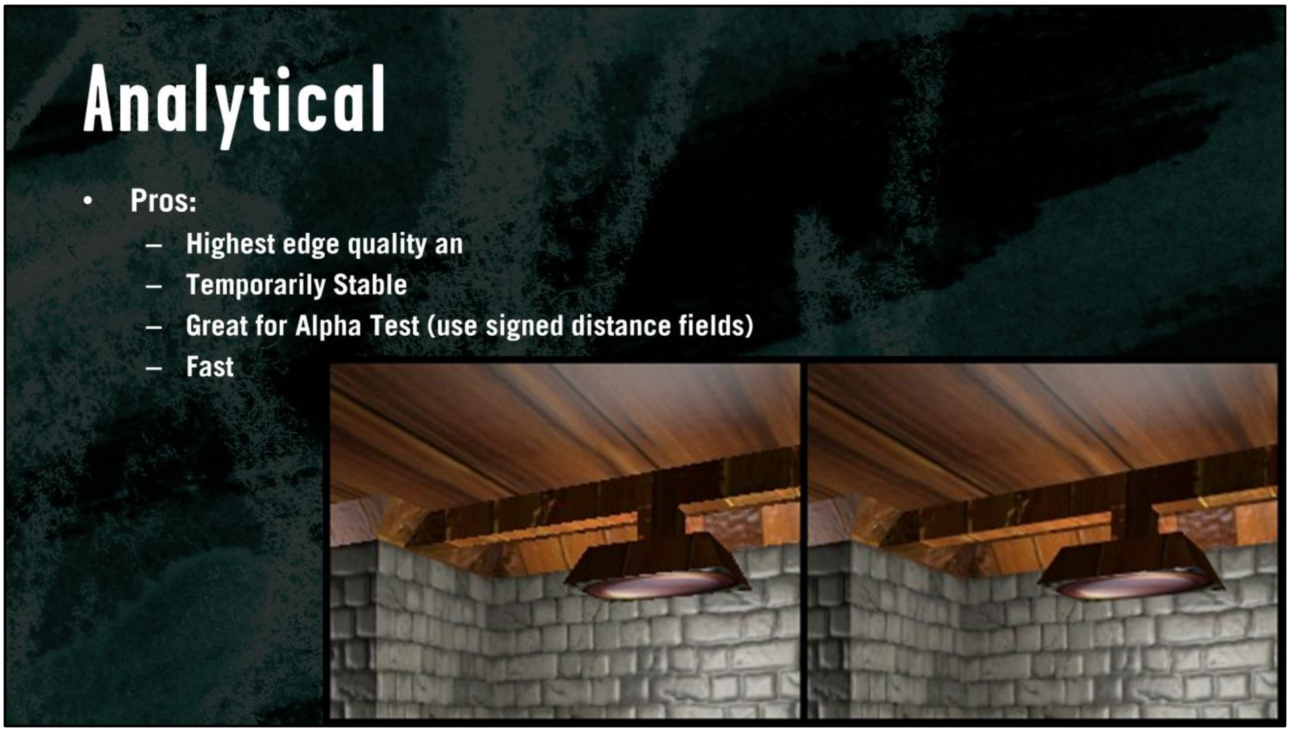
- Pros:
  - Highest quality is static scenario
  - Catch All behaviour
  - Ease of integration
  - Uses rasterized data



Morphological

- Cons:
  - Expensive
  - Not temporarily stable
    - Wobbles under motion
- Partially solved
  - More expensive SMAAx4

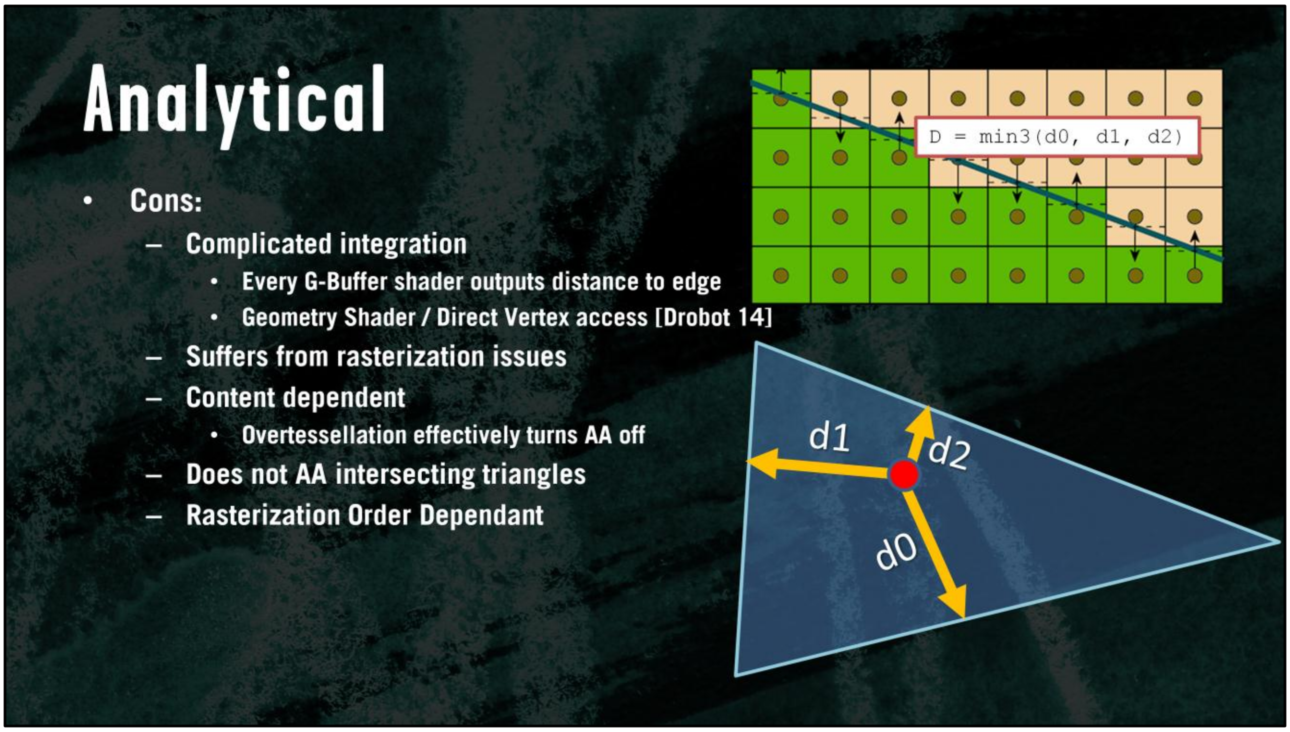
Morphological algorithm rely on rasterized pixel colour / geometric data changes. If those changes do not get 'rasterized' AA will be incorrect.
In case of long moving edges, AA will result in wobble effect, where edge will get correctly rasterized only at its unique rasterized positions.
All intermediate edge positions won't have any effect on AA, unlike Analytical methods.



Analytical

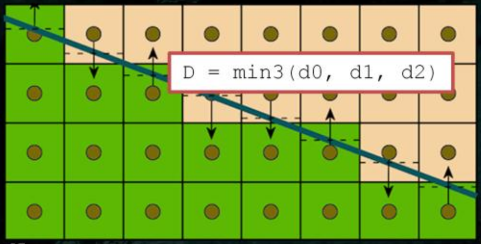
- Pros:
  - Highest edge quality an
  - Temporarily Stable
  - Great for Alpha Test (use signed distance fields)
  - Fast

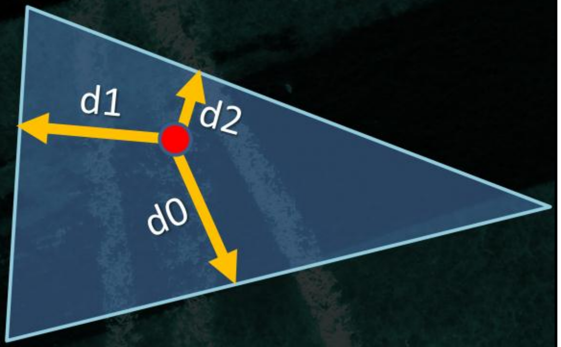
Image courtesy [Persson 11]



Analytical

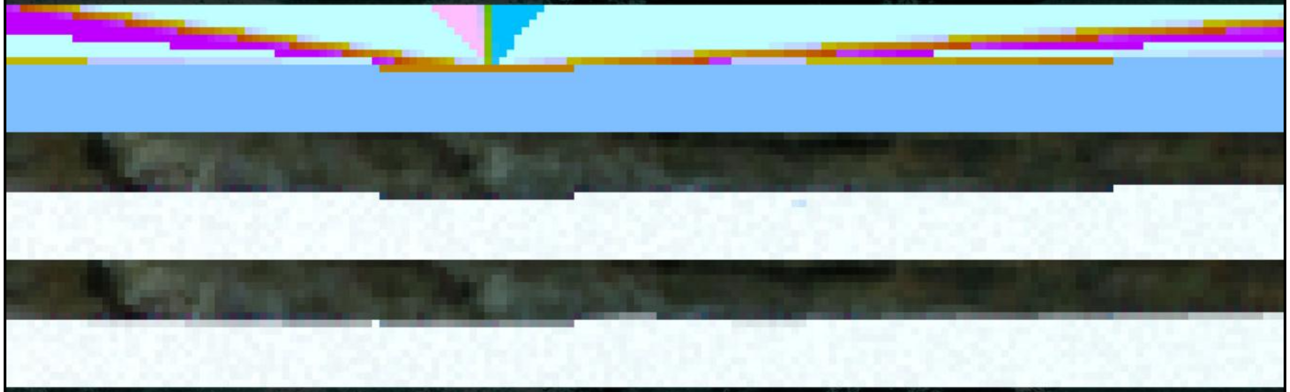
- Cons:
  - Complicated integration
    - Every G-Buffer shader outputs distance to edge
    - Geometry Shader / Direct Vertex access [Drobot 14]
  - Suffers from rasterization issues
  - Content dependent
    - Overtessellation effectively turns AA off
  - Does not AA intersecting triangles
  - Rasterization Order Dependant



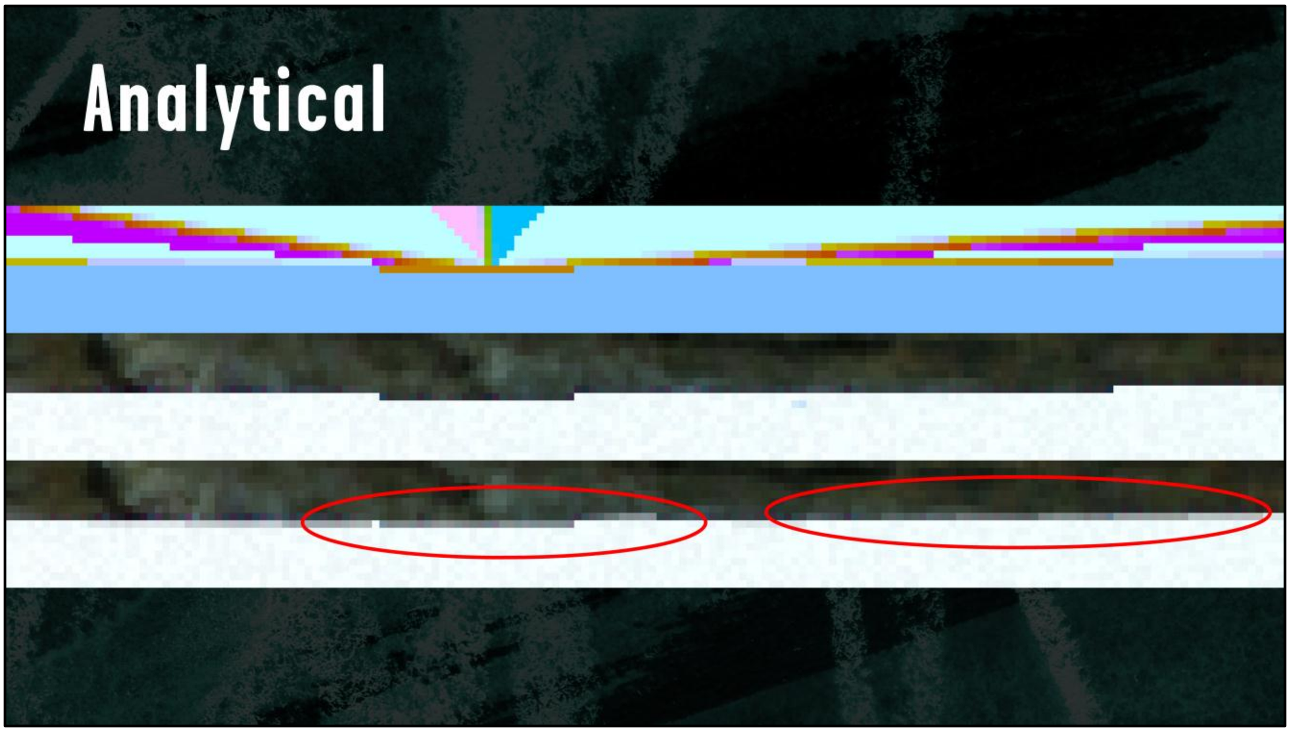


Note: Performance hit can be negligible on platforms supporting Direct Vertex Access in Pixel Shaders such as AMD GCN. See [Drobot 14]

Analytical



Top: Visualization of distance to edge. Color encodes 1bit direction (X, Y). Signed value encodes 4bit distance.
Middle: Result of 1x Centroid rasterization
Bottom: Result of analytical resolve.



Analytical

Note perfectly anti aliased edge on right side. Middle section shows incorrectly AA edge due to rasterization errors and subpixel triangles (where multiple triangle meet).

MSAA

- **Pros:**
  - Closest to ground truth
  - Resolves sub-pixel issues
- **Cons:**
  - Very expensive
  - Too prohibitive to reach Morphological quality level

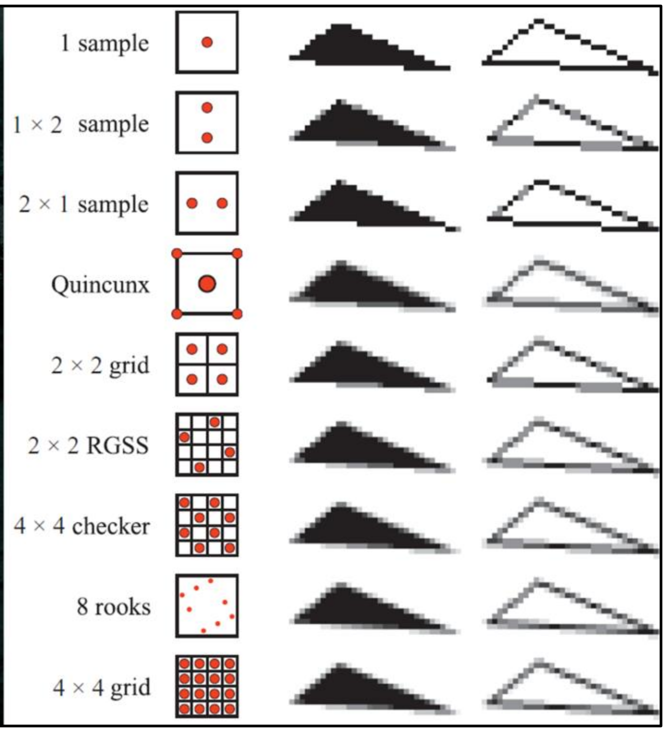
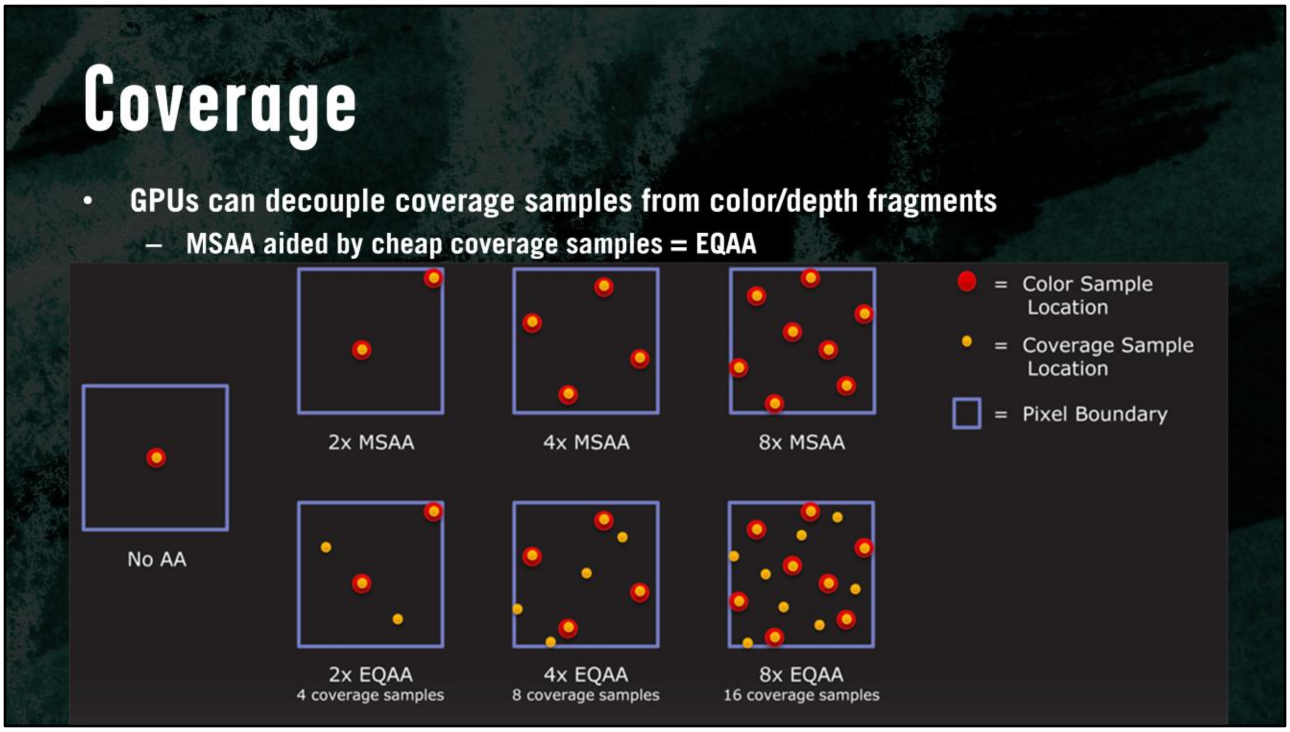


Image courtesy *Real-Time Rendering, 3rd Edition, A K Peters 2008*



Coverage

- GPUs can decouple coverage samples from color/depth fragments
  - MSAA aided by cheap coverage samples = EQAA

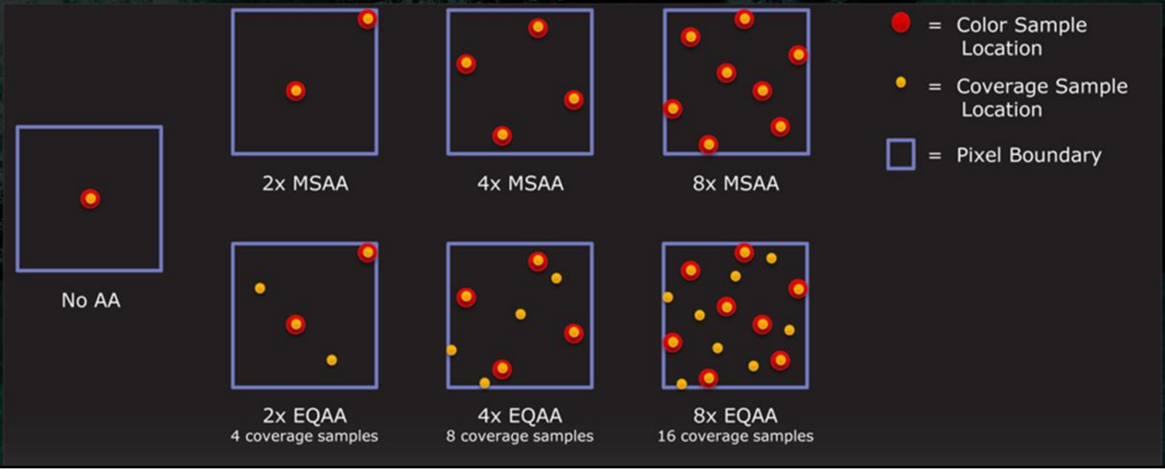
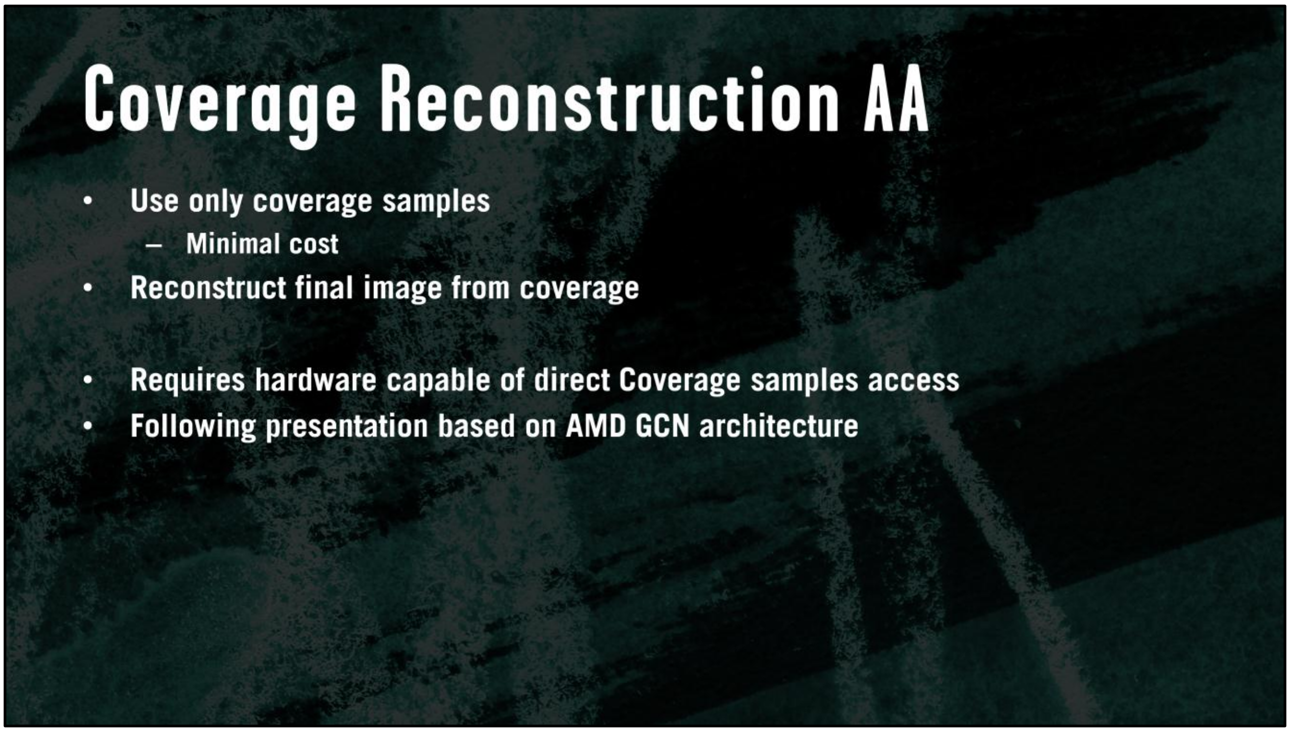


Image courtesy [AMD 11]

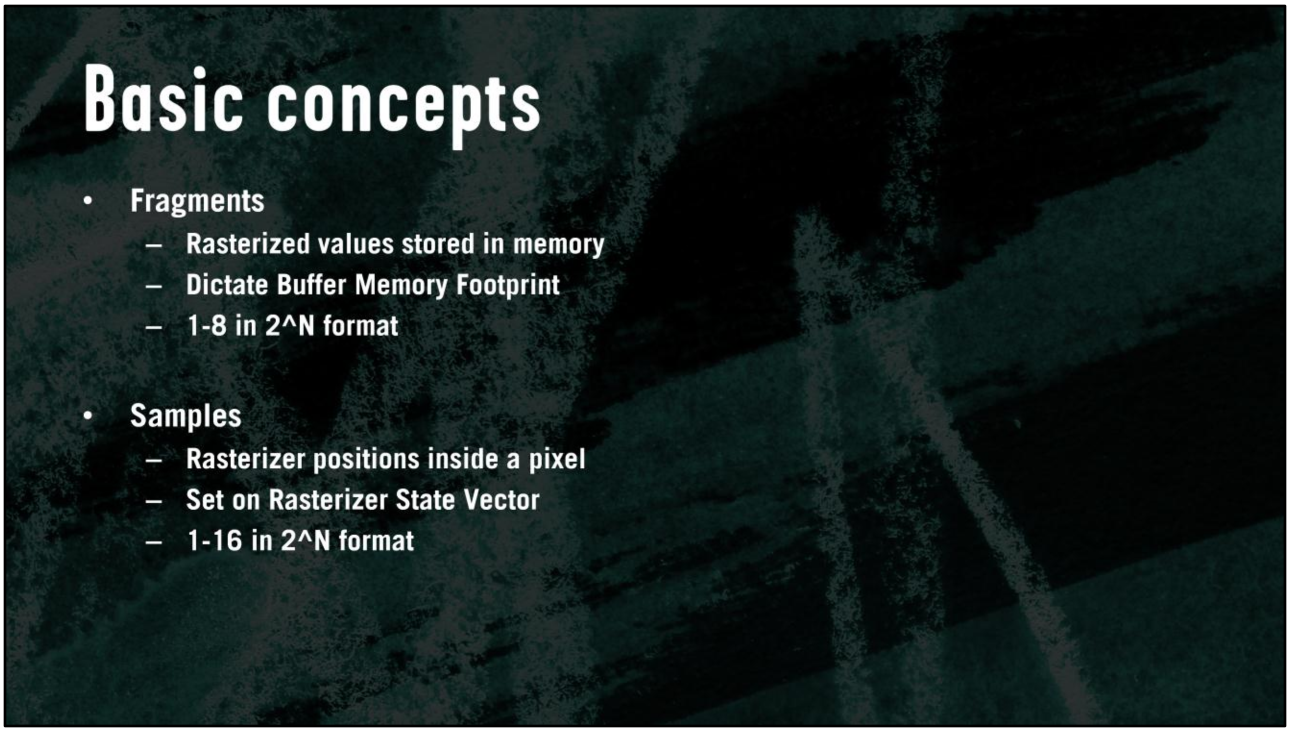


Coverage Reconstruction AA

- Use only coverage samples
  - Minimal cost
- Reconstruct final image from coverage

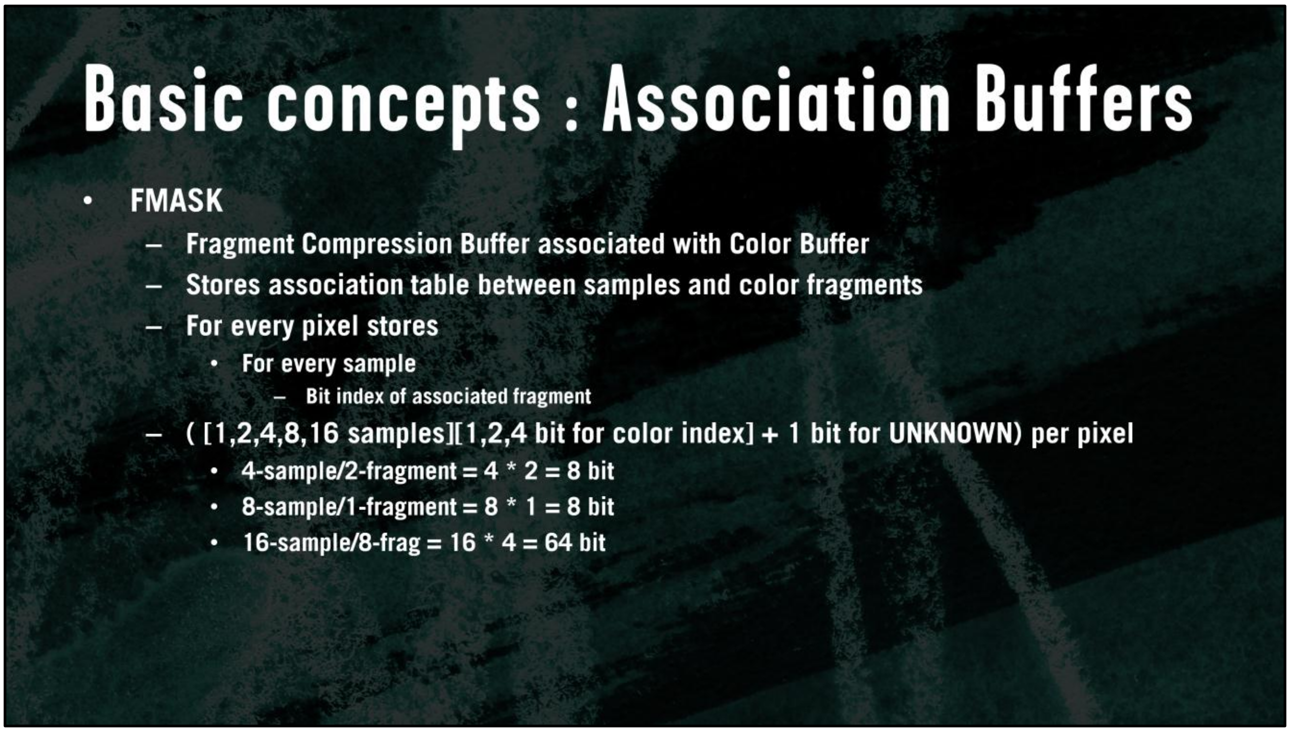
- Requires hardware capable of direct Coverage samples access
- Following presentation based on AMD GCN architecture

Most GPUs are very well optimized for Coverage sampling, unless you want to manually output coverage from Alpha Testing
AMD hardware allows manual access to Coverage pipeline intermediate buffers, thus allowing us more complex resolves.



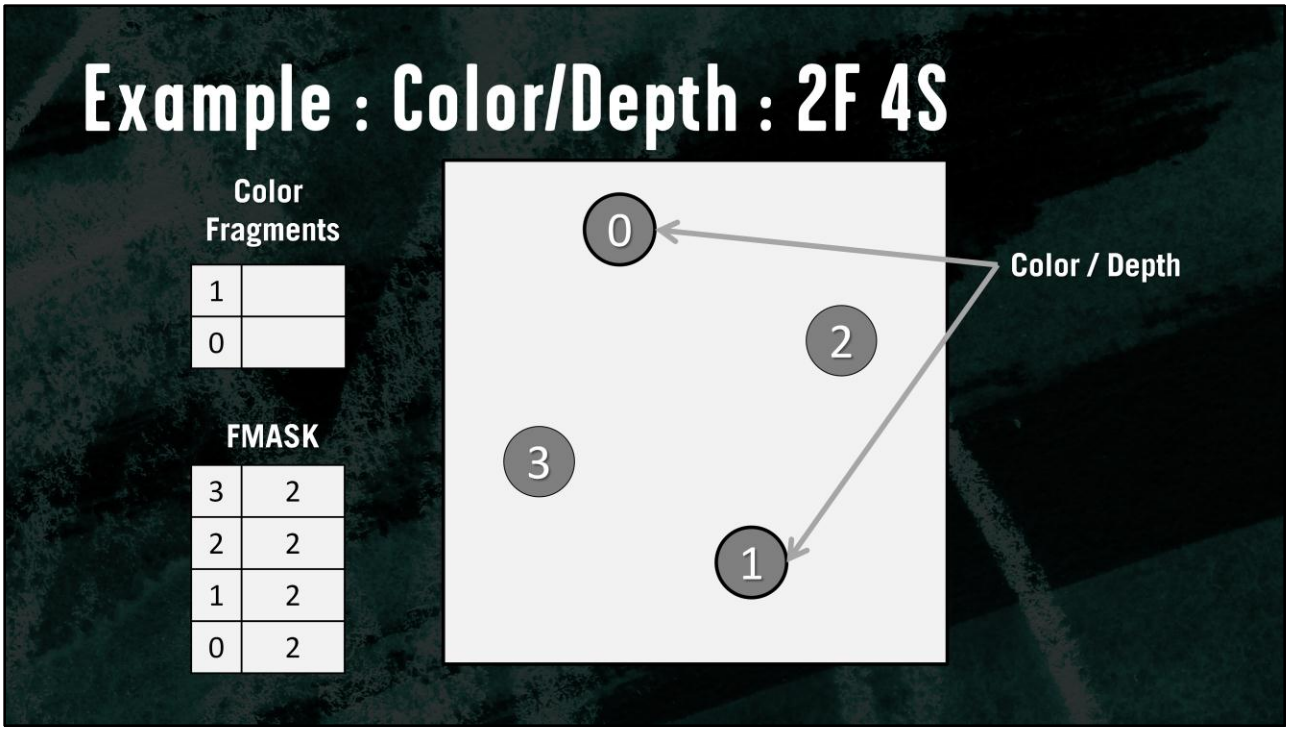
Basic concepts

- Fragments
  - Rasterized values stored in memory
  - Dictate Buffer Memory Footprint
  - 1-8 in 2^N format
- Samples
  - Rasterizer positions inside a pixel
  - Set on Rasterizer State Vector
  - 1-16 in 2^N format



Basic concepts : Association Buffers

- FMASK
  - Fragment Compression Buffer associated with Color Buffer
  - Stores association table between samples and color fragments
  - For every pixel stores
    - For every sample
      - Bit index of associated fragment
  - ([1,2,4,8,16 samples][1,2,4 bit for color index] + 1 bit for UNKNOWN) per pixel
    - 4-sample/2-fragment = 4 * 2 = 8 bit
    - 8-sample/1-fragment = 8 * 1 = 8 bit
    - 16-sample/8-frag = 16 * 4 = 64 bit



Example : Color/Depth : 2F 4S

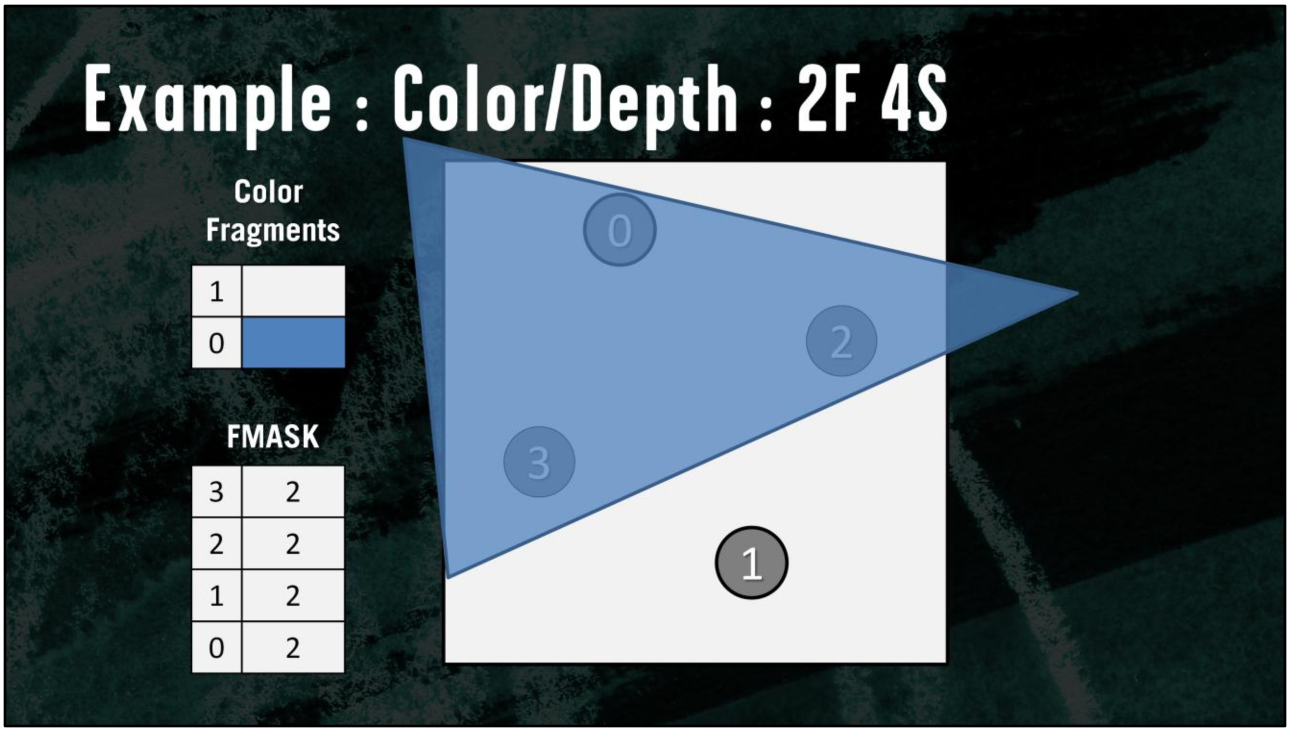
Color Fragments

1	
0	

FMASK

3	2
2	2
1	2
0	2

Samples 0 and 1 are anchored – have their own depth fragments for depth testing



Example : Color/Depth : 2F 4S

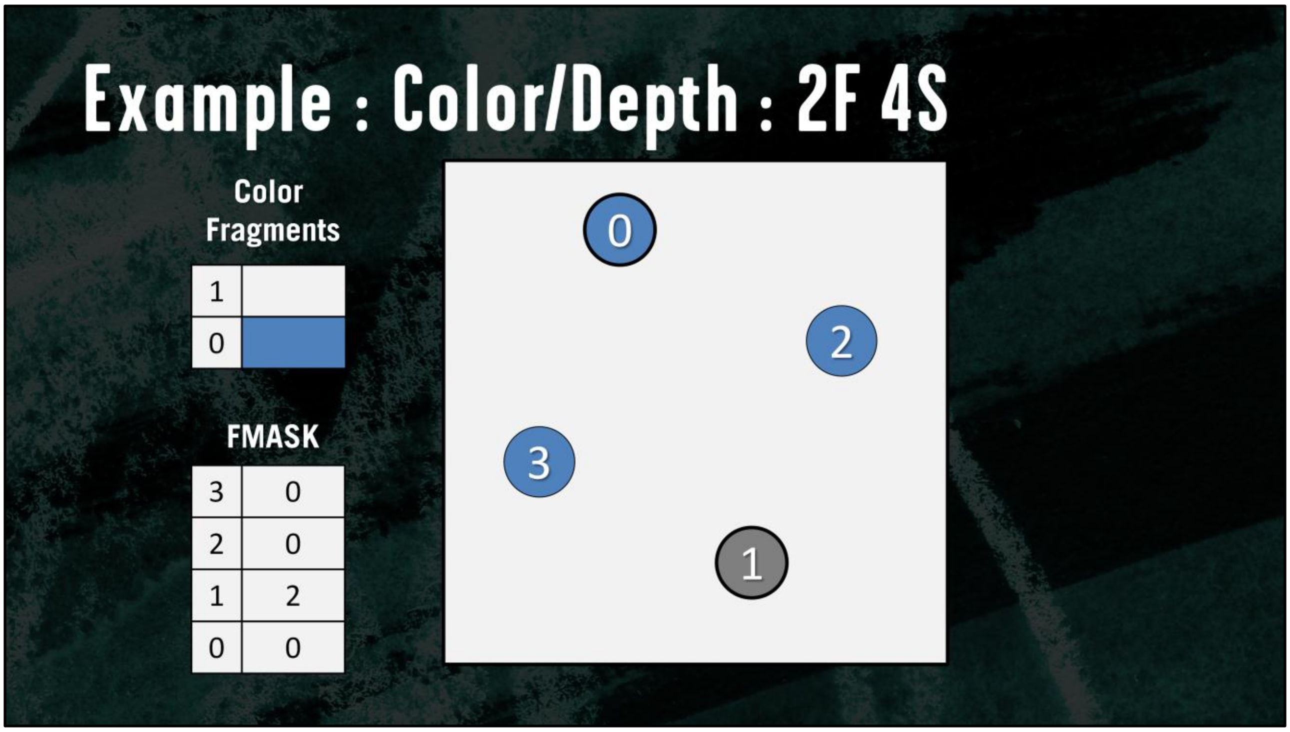
Color Fragments

1	
0	🟦

FMASK

3	2
2	2
1	2
0	2

BLUE triangle hits the ANCHORED SAMPLE 0 – BLUE is added to Color Fragments



Example : Color/Depth : 2F 4S

Color Fragments

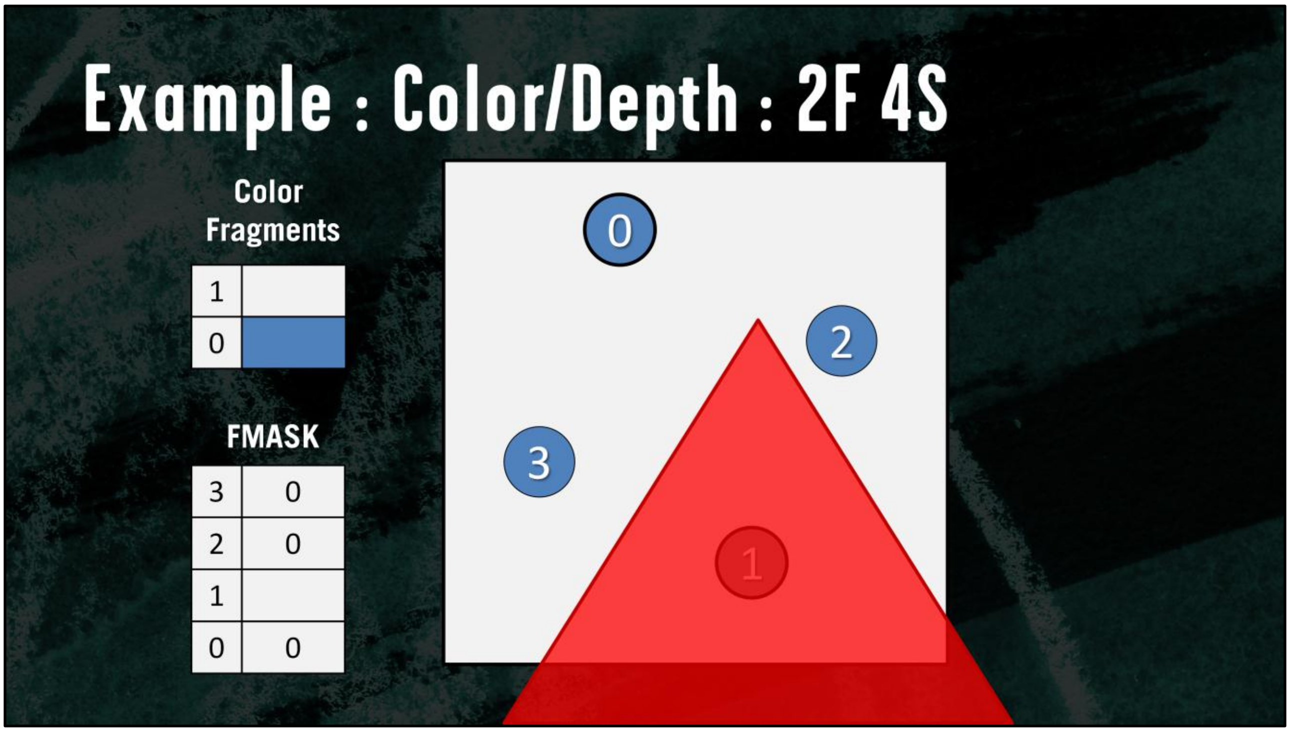
1	
0	🟦

FMASK

3	0
2	0
1	2
0	0

Samples covered by BLUE triangle gets associated

Example : Color/Depth : 2F 4S



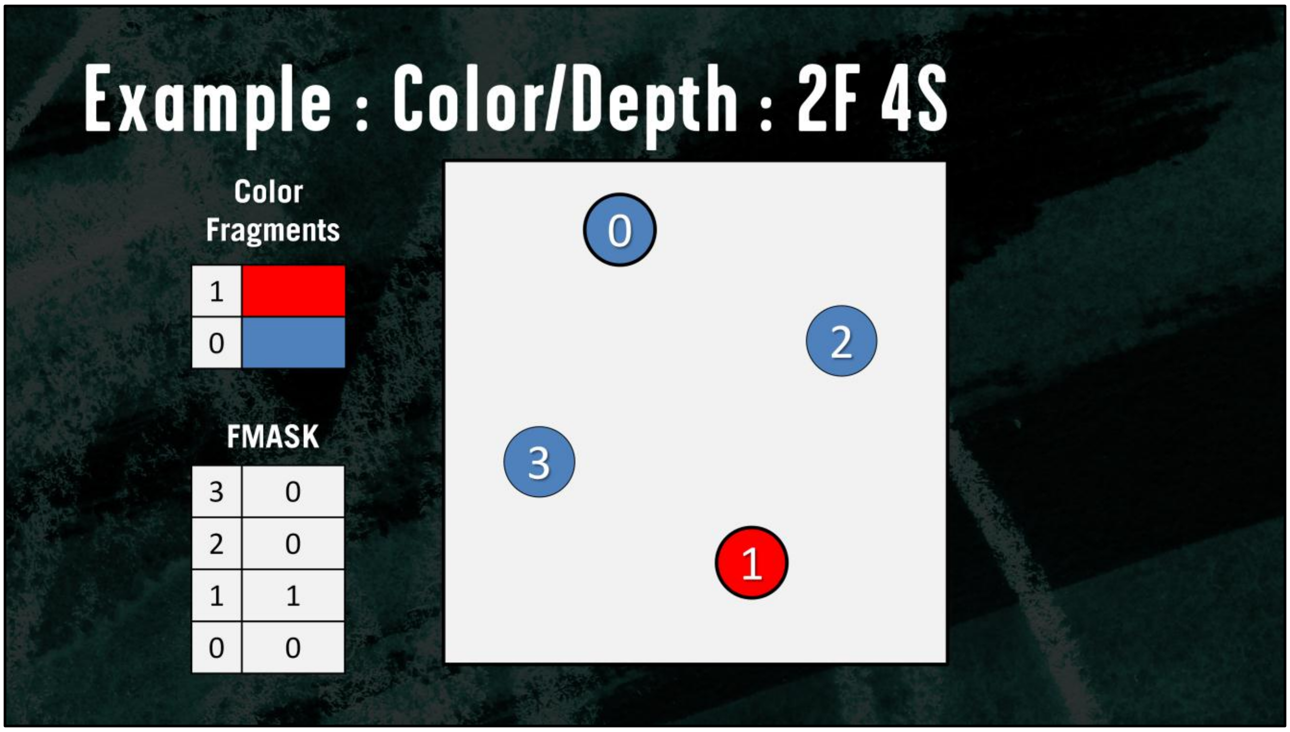
Color Fragments

1	
0	🟦

FMASK

3	0
2	0
1	
0	0

Incoming Red Triangle clears sample 1 association



Example : Color/Depth : 2F 4S

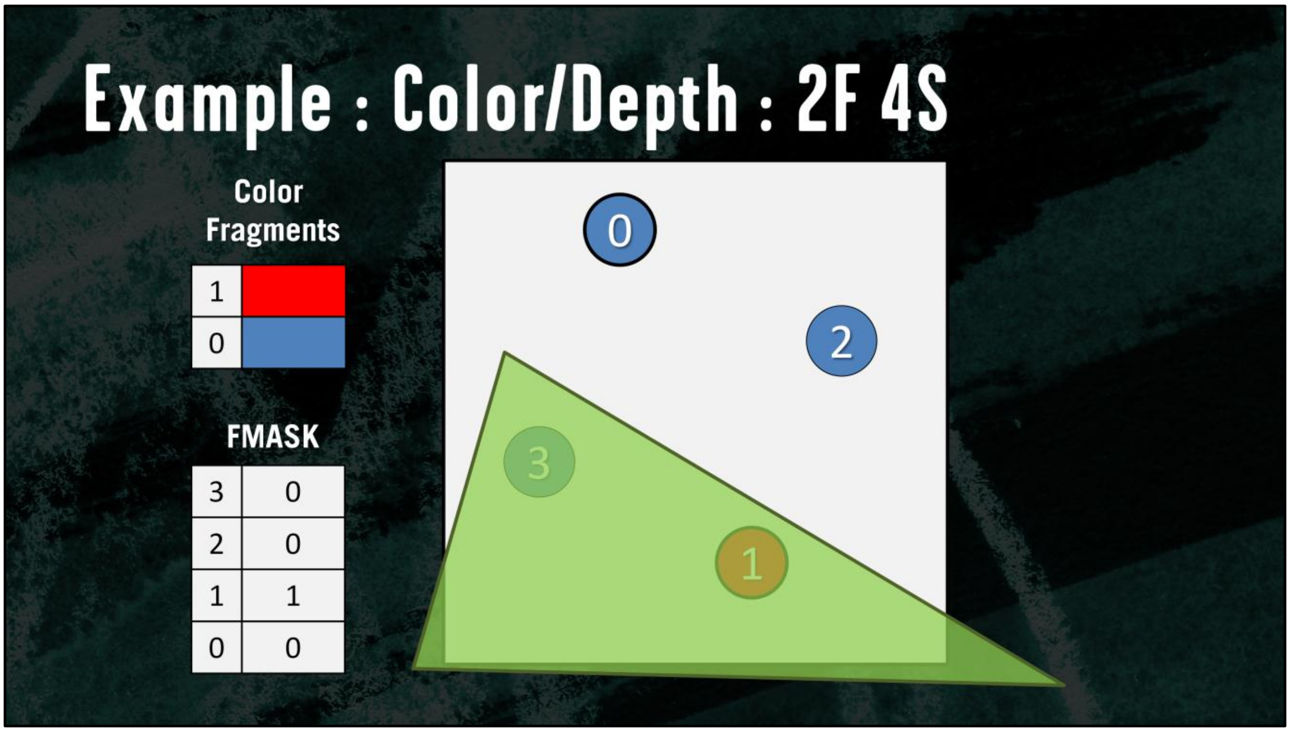
Color Fragments

1	🟥
0	🟦

FMASK

3	0
2	0
1	1
0	0

New color RED gets added to Fragment Color table. Sample 1 gets association with RED color stored at Color Fragment 1.



Example : Color/Depth : 2F 4S

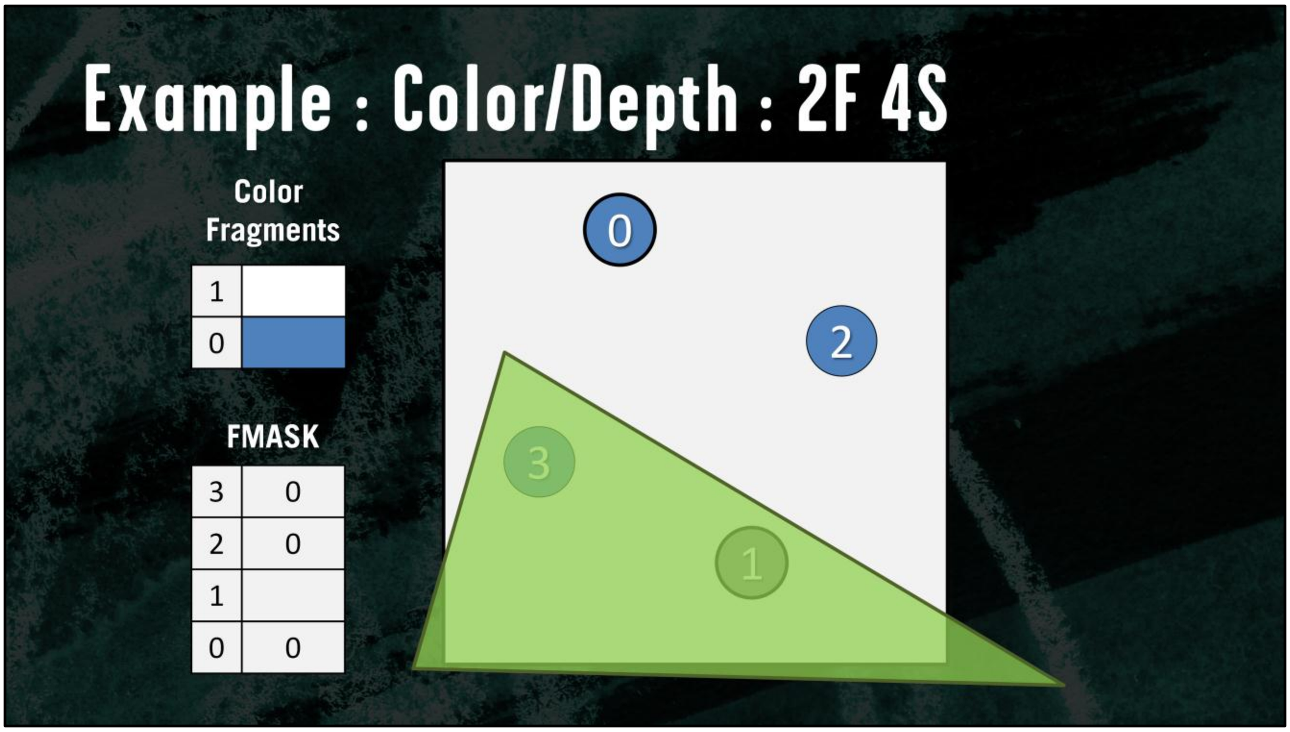
Color Fragments

1	🟥
0	🟦

FMASK

3	0
2	0
1	1
0	0

Incoming Green Triangle will need to clear Sample 1 and 3 association and evict RED or BLUE from Color Fragment Table



Example : Color/Depth : 2F 4S

Color Fragments

1	⬜
0	🟦

FMASK

3	0
2	0
1	
0	0

Arbitration rules:
During color eviction from Color Fragment table, Fragment with least priority is removed.
Priority depends on Sample Count associated with Color.
If a tie occurs, lower index has higher priority.

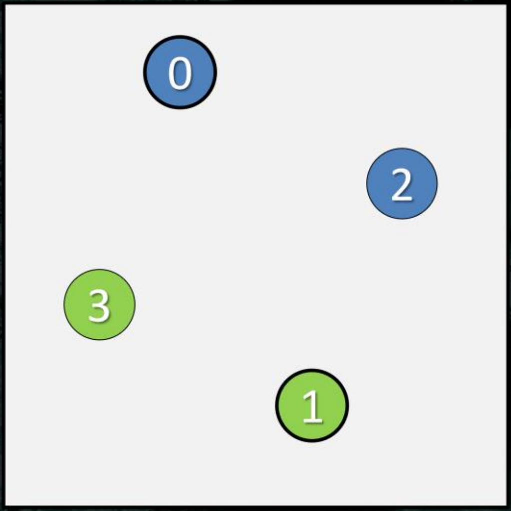
Example : Color/Depth : 2F 4S

Color Fragments

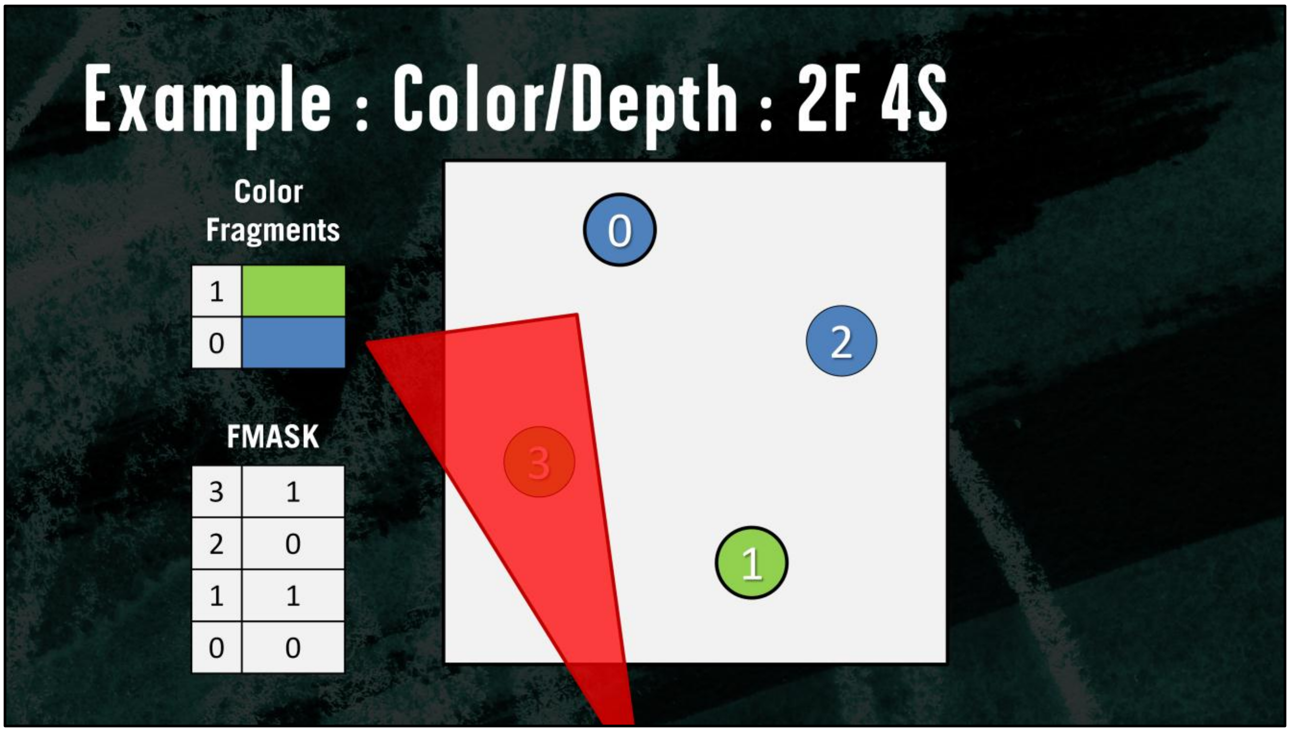
1	🟩
0	🟦

FMASK

3	1
2	0
1	1
0	0



GREEN gets added into Color Fragments, Samples 1 and 3 get new associations



Example : Color/Depth : 2F 4S

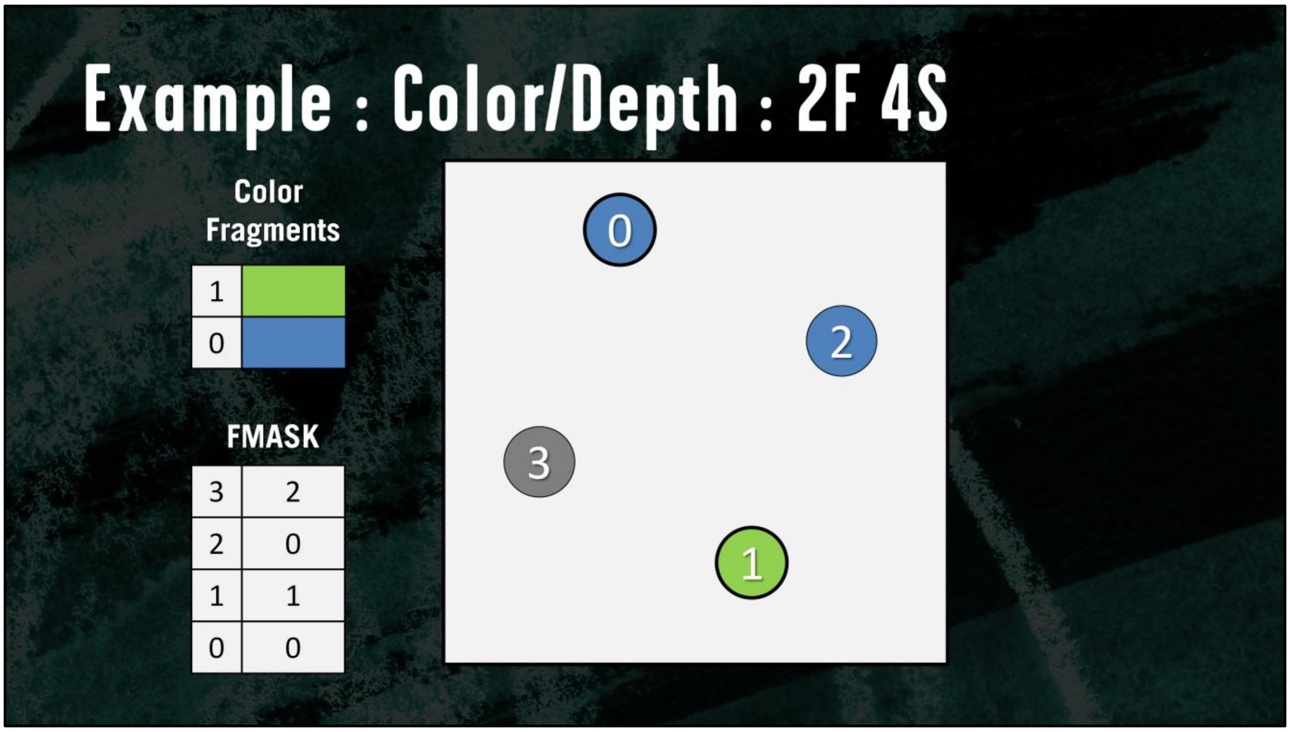
Color Fragments

1	🟩
0	🟦

FMASK

3	1
2	0
1	1
0	0

Sample 3 is unanchored - it has no depth thus can't be Z –tested.
To avoid color leakage it's association must be discarded.



Example : Color/Depth : 2F 4S

Color Fragments

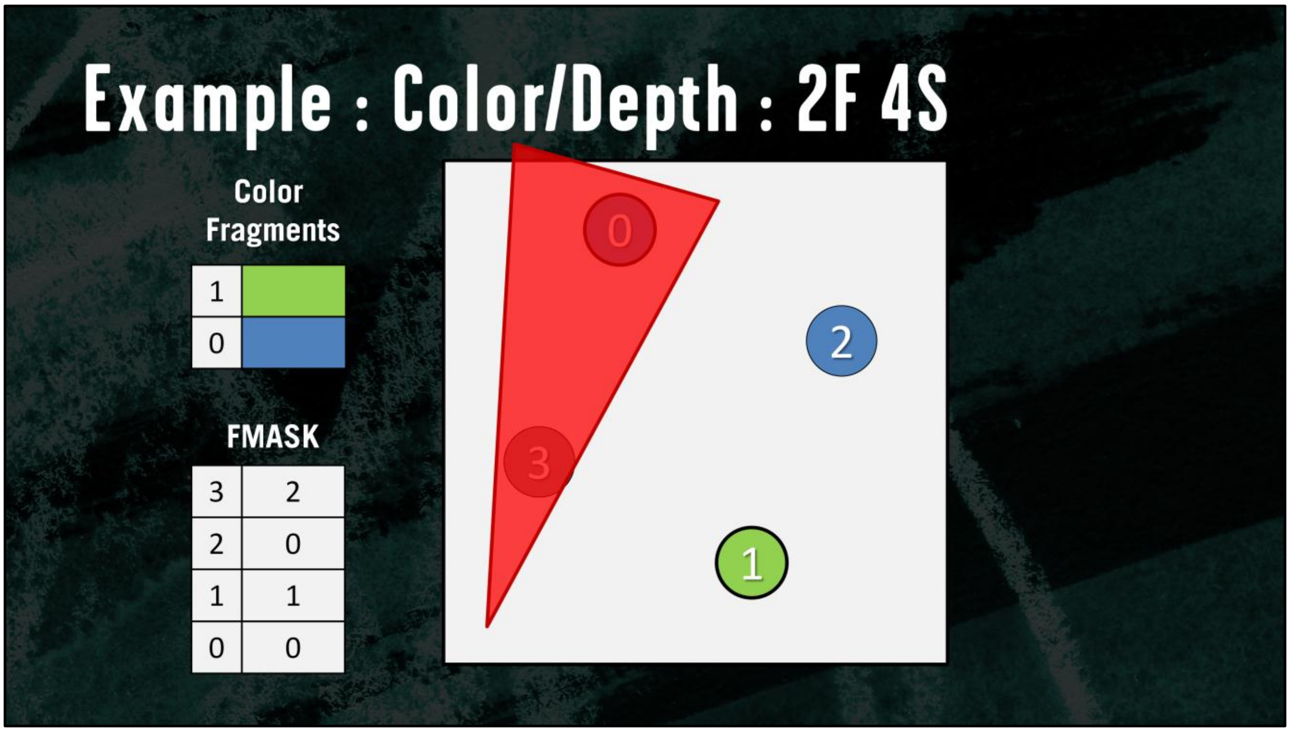
1	🟩
0	🟦

FMASK

3	2
2	0
1	1
0	0

Sample 3 goes UNKNOWN

NOTE: it’s possible to ‘save’ sample 3 association. HW can execute accurate testing by depth plane interpolation. If sample 3 would pass such test – would not be overwritten by RED triangle depth – it’s association would not be touched.
This is only possible if Depth Buffer is stored in Compressed Format using Zplanes [AMD GCN]
Otherwise it’s advised to use more anchor samples.



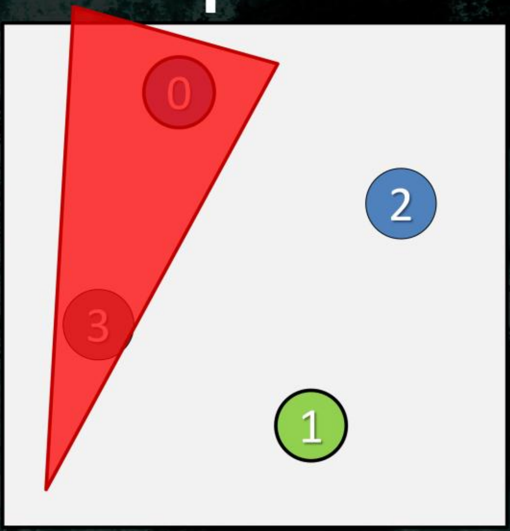
Example : Color/Depth : 2F 4S

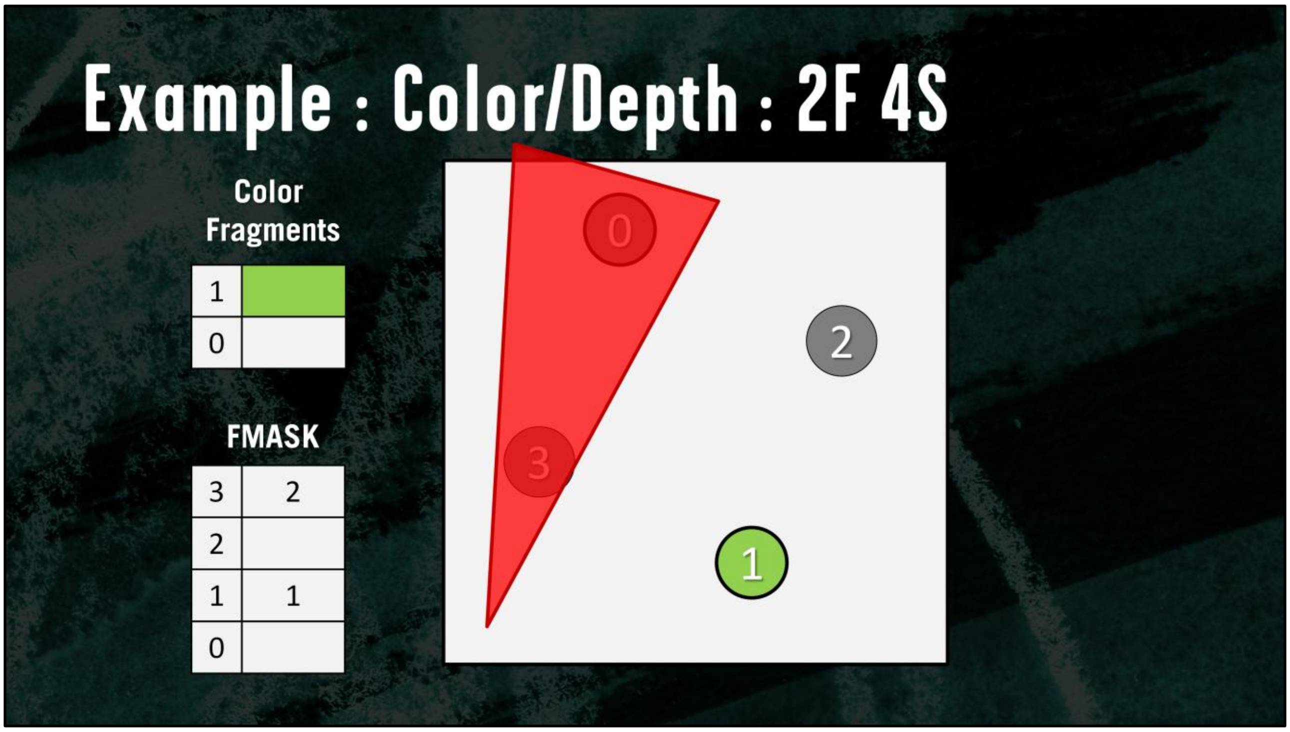
Color Fragments

1	🟩
0	🟦

FMASK

3	2
2	0
1	1
0	0





Example : Color/Depth : 2F 4S

Color Fragments

1	🟩
0	

FMASK

3	2
2	
1	1
0	

BLUE gets evicted. All samples associated with it will be set to UNKNOWN (including 2 not covered by RED triangle)

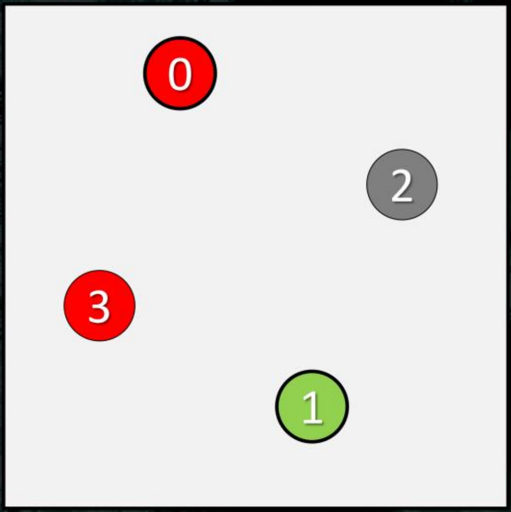
Example : Color/Depth : 2F 4S

Color Fragments

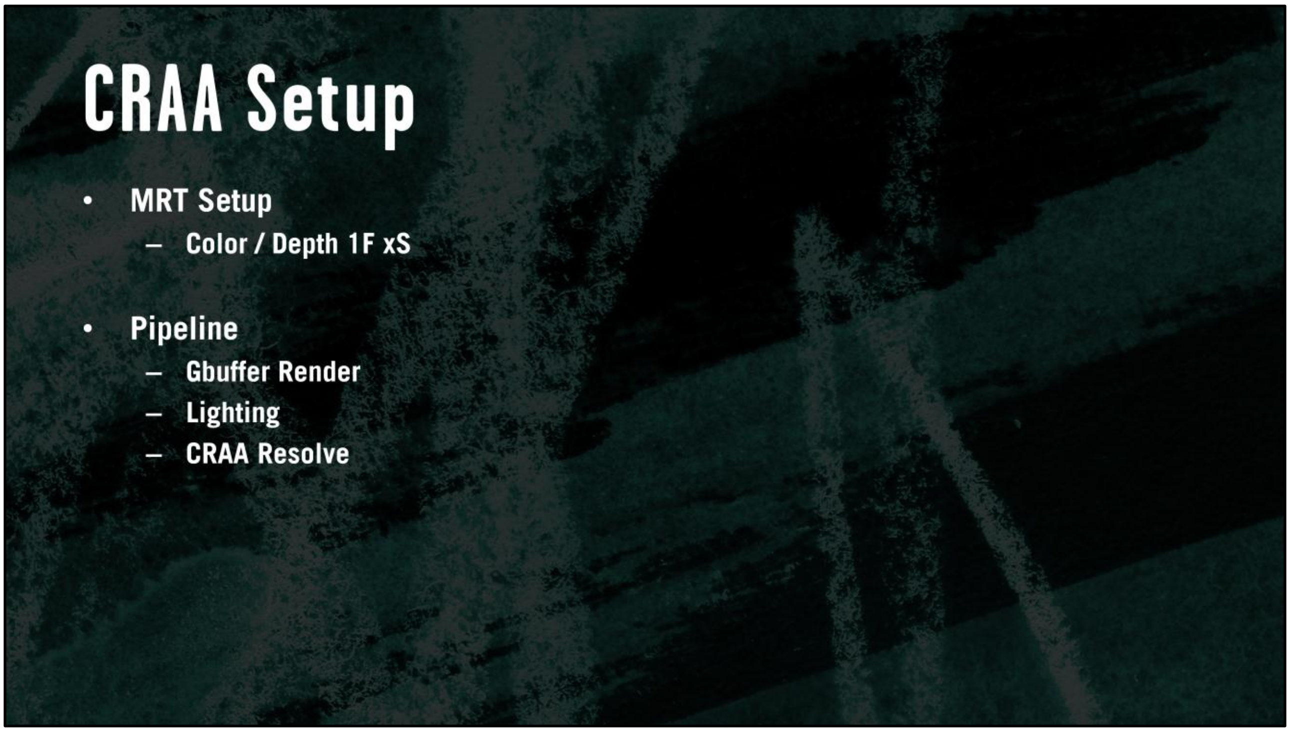
1	🟩
0	🟥

FMASK

3	0
2	2
1	1
0	0



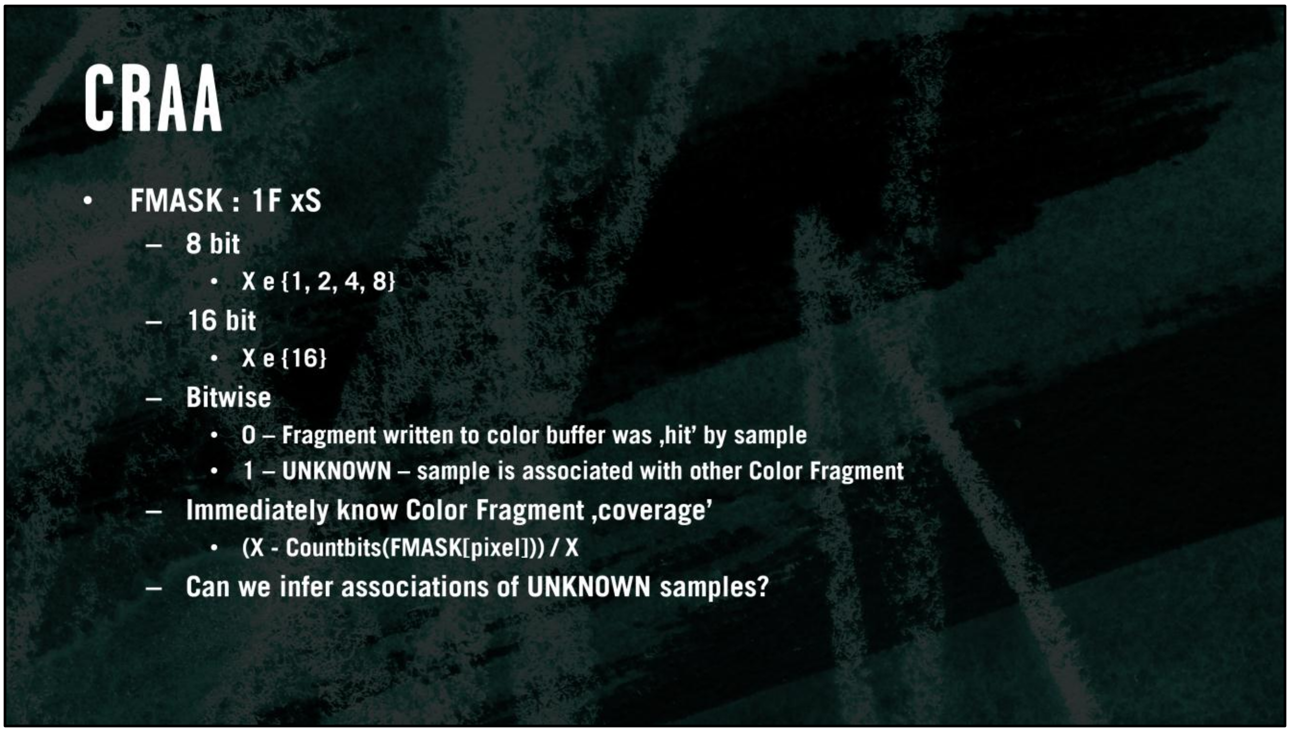
Sample 2 is left UNKNOWN



CRAA Setup

- MRT Setup
  - Color / Depth 1F xS
- Pipeline
  - Gbuffer Render
  - Lighting
  - CRAA Resolve

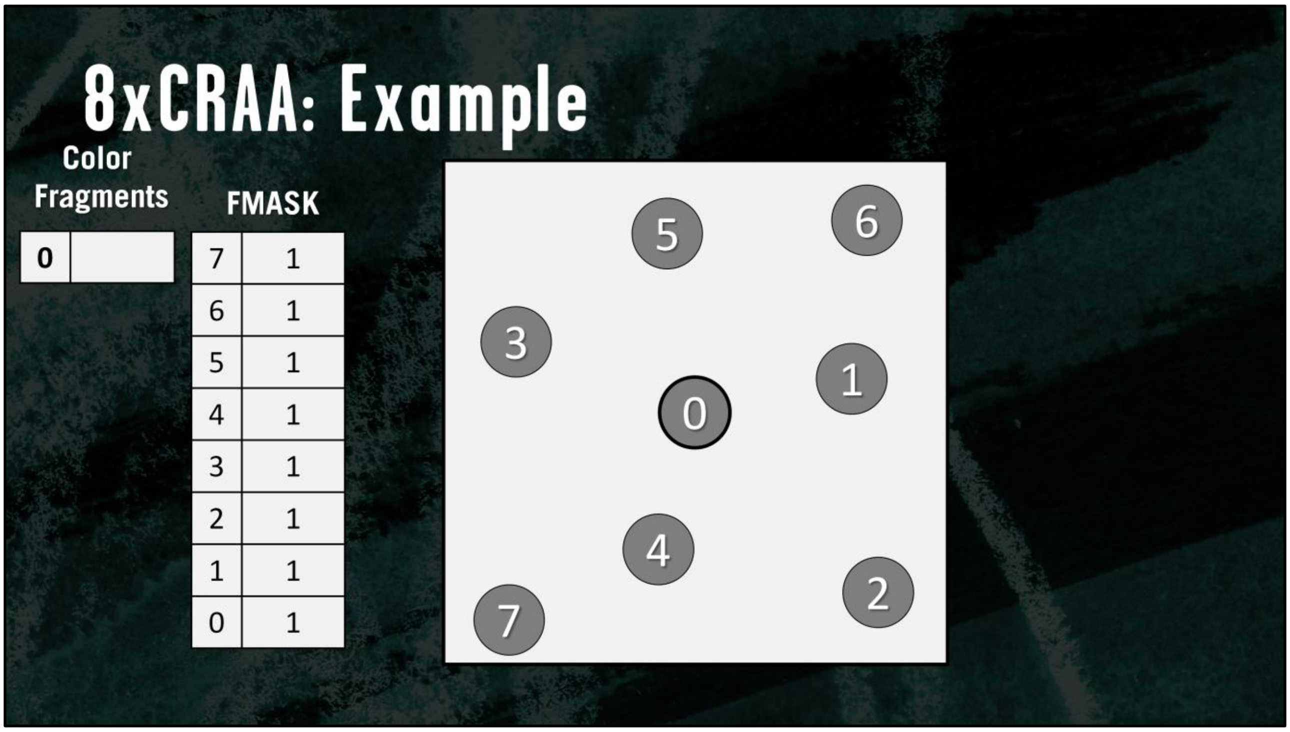
Simple setup that has low shading rate 1 shaded sample / pixel
Image reconstruction based on coverage data happens after lighting.



CRAA

- FMASK : 1F xS
  - 8 bit
    - X e {1, 2, 4, 8}
  - 16 bit
    - X e {16}
  - Bitwise
    - 0 – Fragment written to color buffer was ‚hit' by sample
    - 1 – UNKNOWN – sample is associated with other Color Fragment
  - Immediately know Color Fragment ‚coverage'
    - (X - Countbits(FMASK[pixel])) / X
  - Can we infer associations of UNKNOWN samples?

Specific case of 1Fragment FMask.
Every sample can only have two states, thus making sampling and recovering information very easy.



8xCRAA: Example

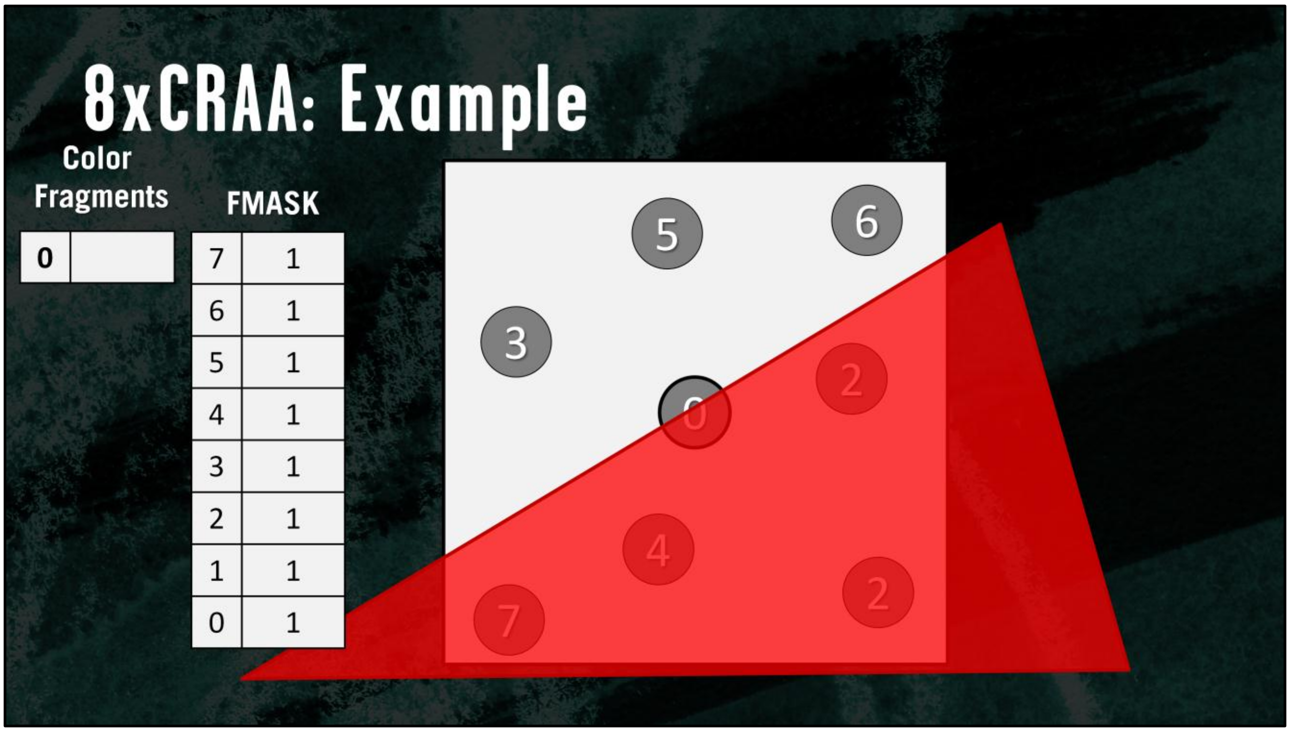
Color Fragments

0	

FMASK

7	1
6	1
5	1
4	1
3	1
2	1
1	1
0	1

1F xS sampling patterns should be handcrafted.
Sample 0 (depth / color) at center should behave better at reconstruction of sub-pixel details.
Sample 0 at edges / corners should behave better at reconstruction of edges.
Better sampling patterns require more work.



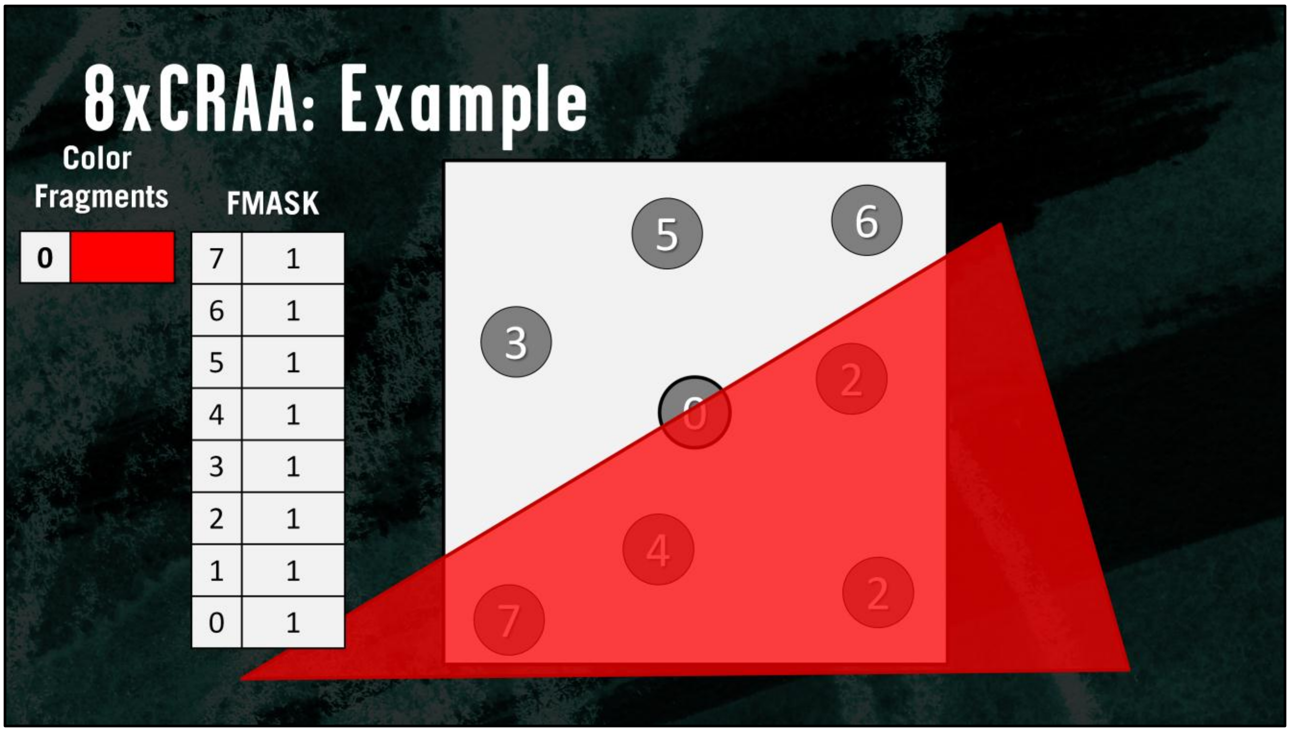
8xCRAA: Example

Color Fragments

0	

FMASK

7	1
6	1
5	1
4	1
3	1
2	1
1	1
0	1



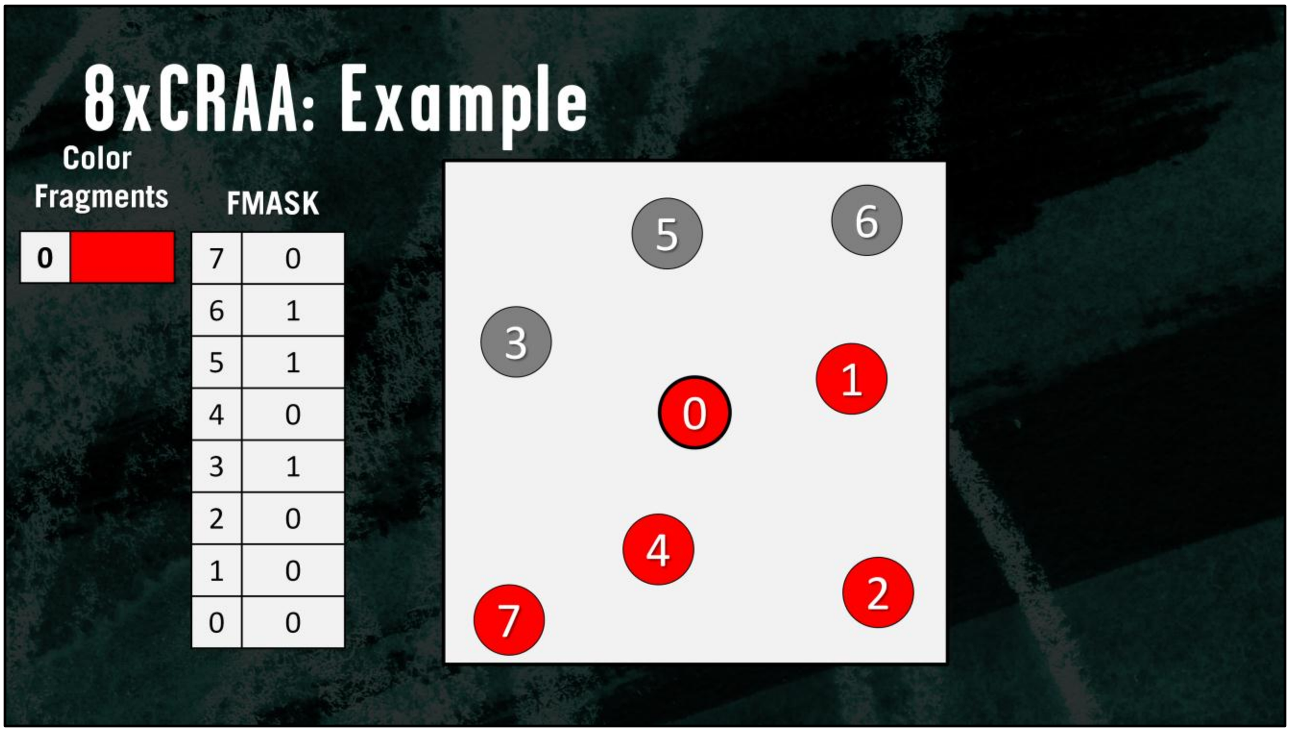
8xCRAA: Example

Color Fragments

0	🟥

FMASK

7	1
6	1
5	1
4	1
3	1
2	1
1	1
0	1



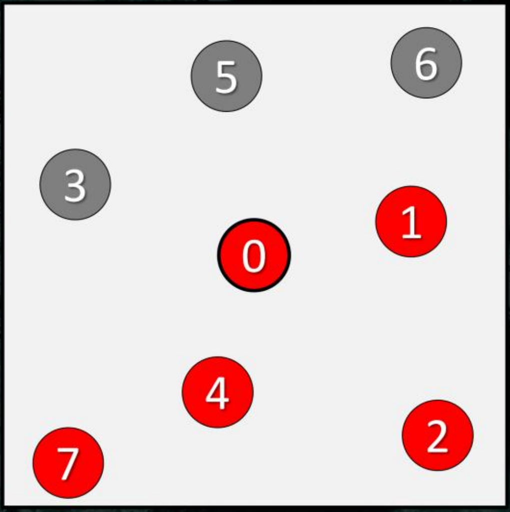
8xCRAA: Example

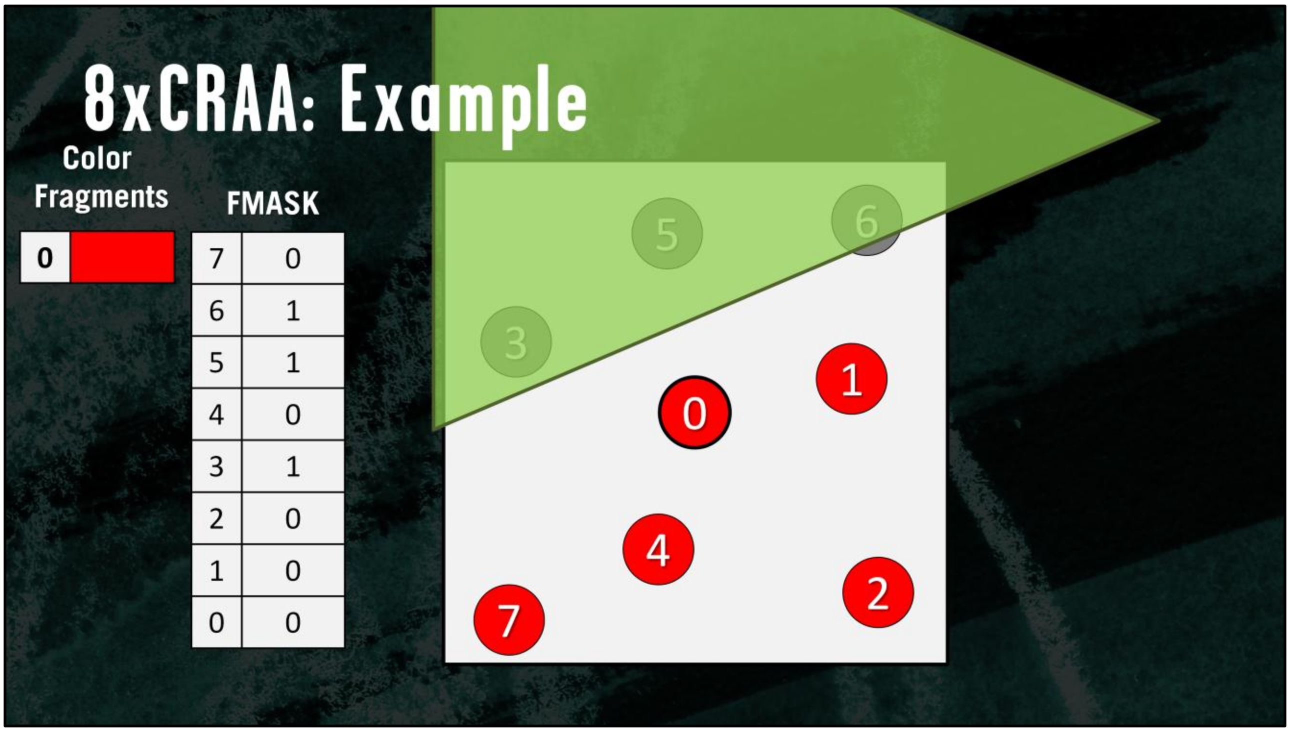
Color Fragments

0	🟥

FMASK

7	0
6	1
5	1
4	0
3	1
2	0
1	0
0	0





8xCRAA: Example

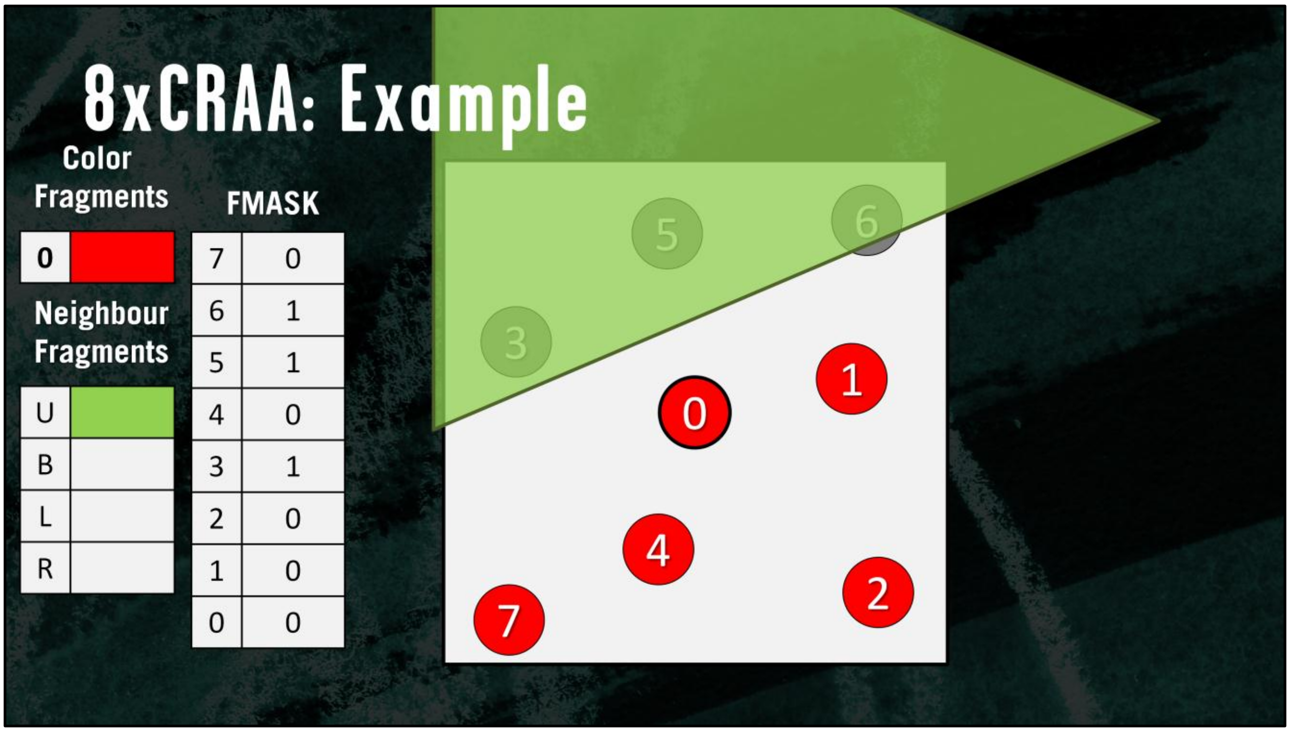
Color Fragments

0	🟥

FMASK

7	0
6	1
5	1
4	0
3	1
2	0
1	0
0	0

We see it will end up not being rasterized – thus set Samples to UNKNOWN.
But maybe the neighbour Pixel will get rasterized?



8xCRAA: Example

Color Fragments

0	🟥

Neighbour Fragments

U	🟩
B	
L	
R	

FMASK

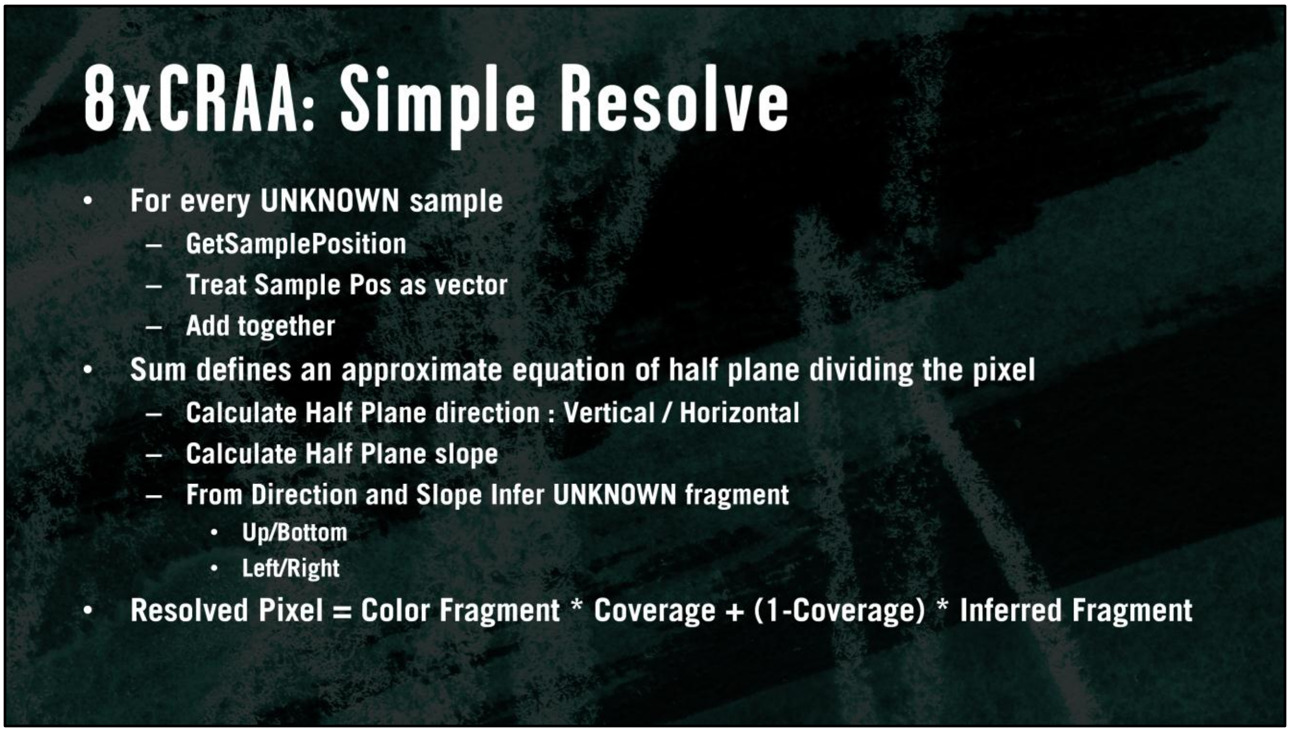
7	0
6	1
5	1
4	0
3	1
2	0
1	0
0	0

GREEN triangle is large enough to get rasterized in pixel neighborhood. We add its color GREEN to current pixel Neighborhood Fragments as UP neighbor.

8xCRAA: Example FMASK

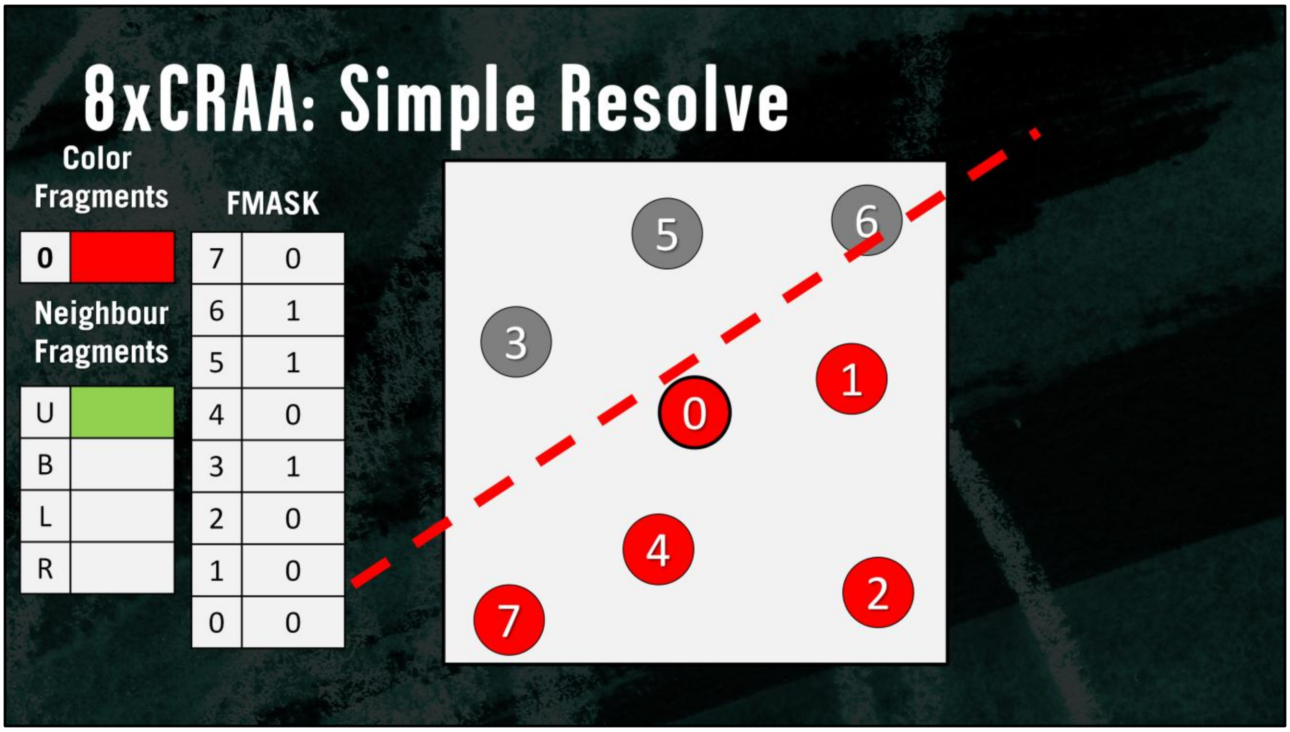
- FMASK : 00010110
- X = 8
- RED Coverage = CountBits(00010110^) / 8 = 5/8

- UNKNOWN
  - Infer them from neighbourhood
  - We know every Sample position



8xCRAA: Simple Resolve

- For every UNKNOWN sample
  - GetSamplePosition
  - Treat Sample Pos as vector
  - Add together
- Sum defines an approximate equation of half plane dividing the pixel
  - Calculate Half Plane direction : Vertical / Horizontal
  - Calculate Half Plane slope
  - From Direction and Slope Infer UNKNOWN fragment
    - Up/Bottom
    - Left/Right
- Resolved Pixel = Color Fragment * Coverage + (1-Coverage) * Inferred Fragment



8xCRAA: Simple Resolve

Color Fragments

0	🟥

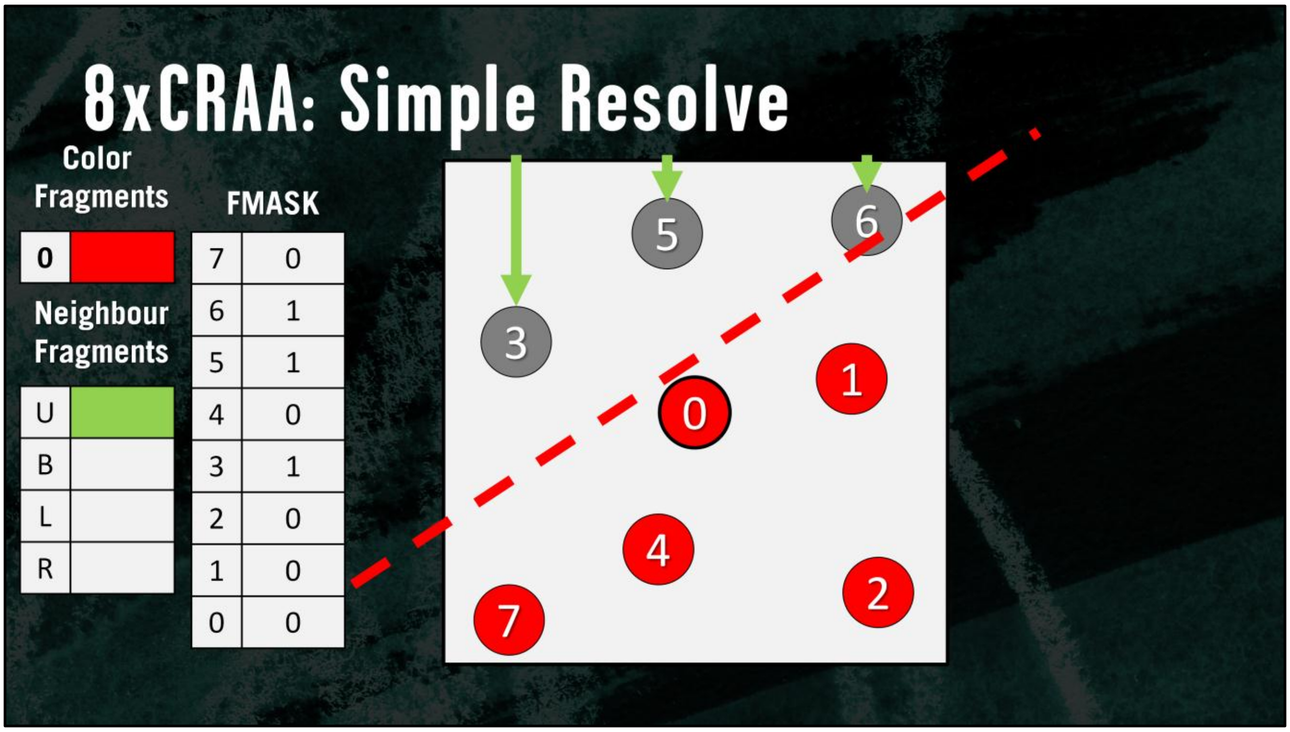
Neighbour Fragments

U	🟩
B	
L	
R	

FMASK

7	0
6	1
5	1
4	0
3	1
2	0
1	0
0	0

We find a half plane dividing the pixel.
Could also use more expensive (at runtime) line fitting between two subsets of samples.



8xCRAA: Simple Resolve

Color Fragments

0	🟥

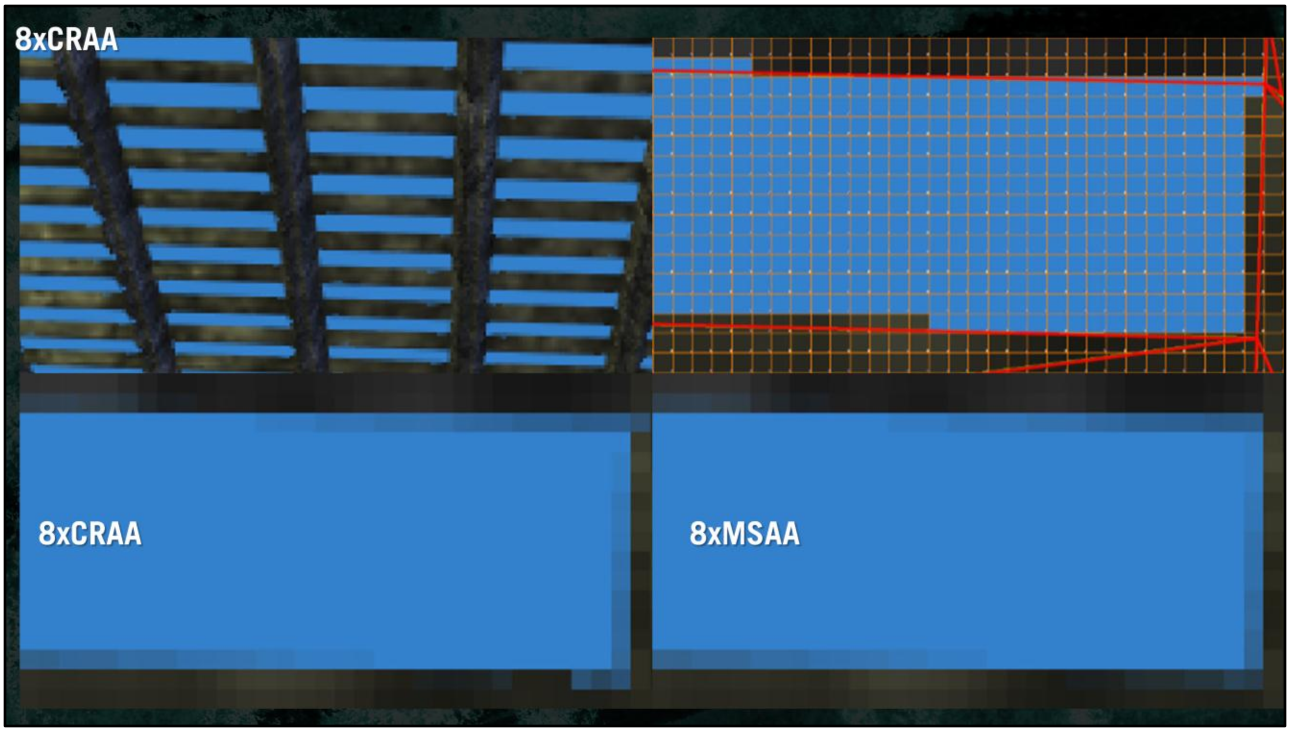
Neighbour Fragments

U	🟩
B	
L	
R	

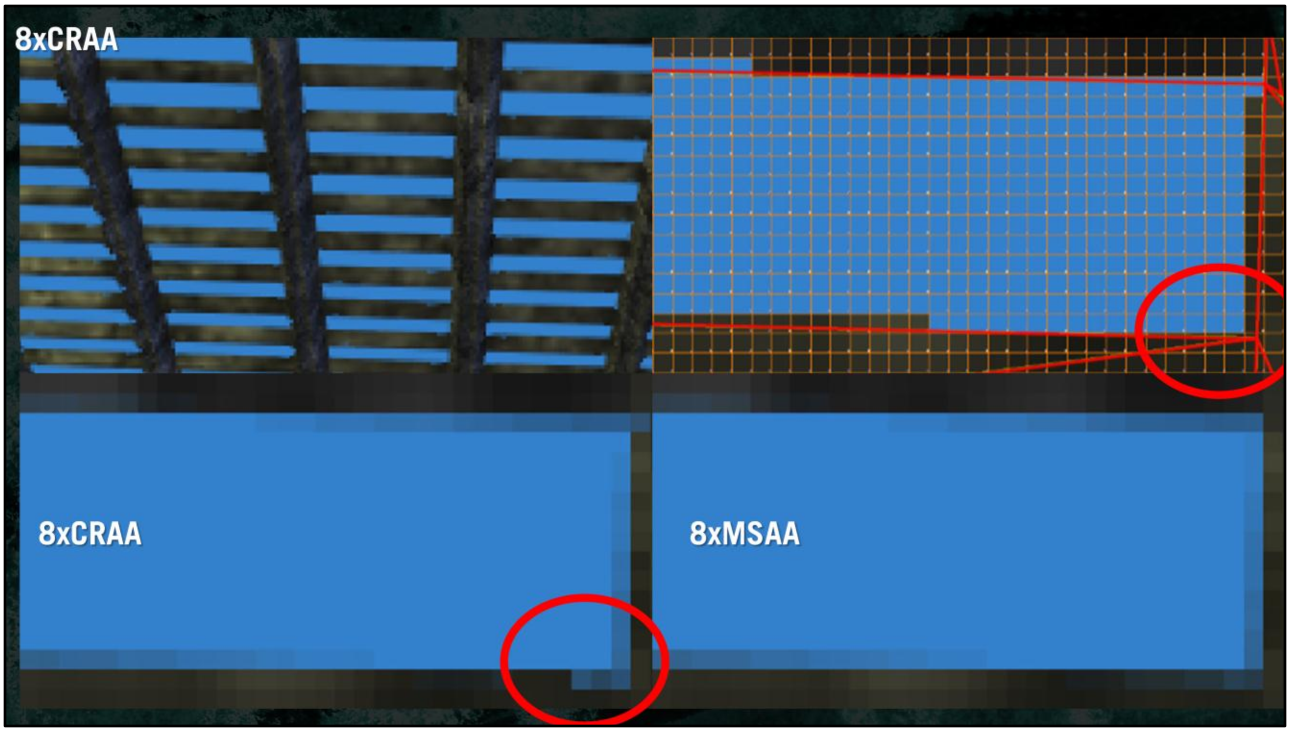
FMASK

7	0
6	1
5	1
4	0
3	1
2	0
1	0
0	0

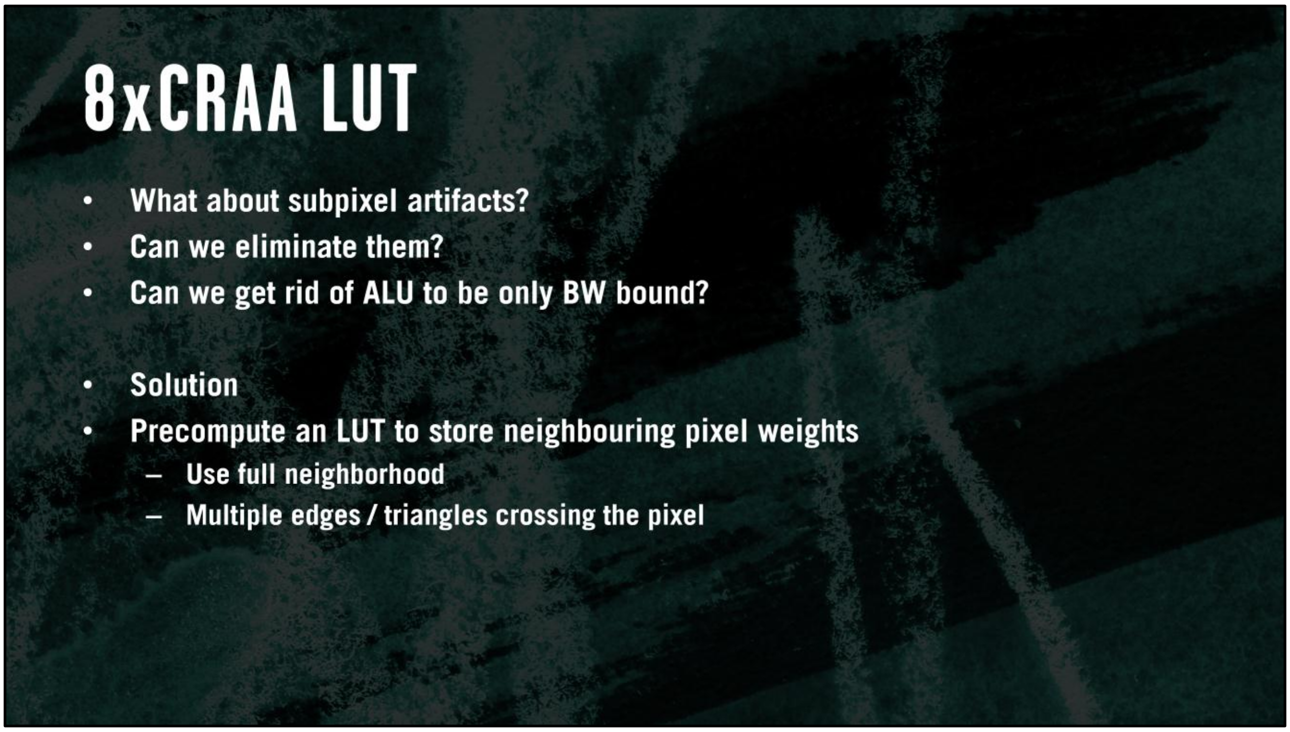
Infer UNKNOWN sample colors from the other side of derived edge.



Top Left : Overview of complex AA scenario using 8xCRAA
Top Right : Line layout for a crop of a complex corner case with multiple triangles intersecting one pixel
Bottom Left: 8xCRAA resolve results
Bottom Right: 8xMSAA resolve results



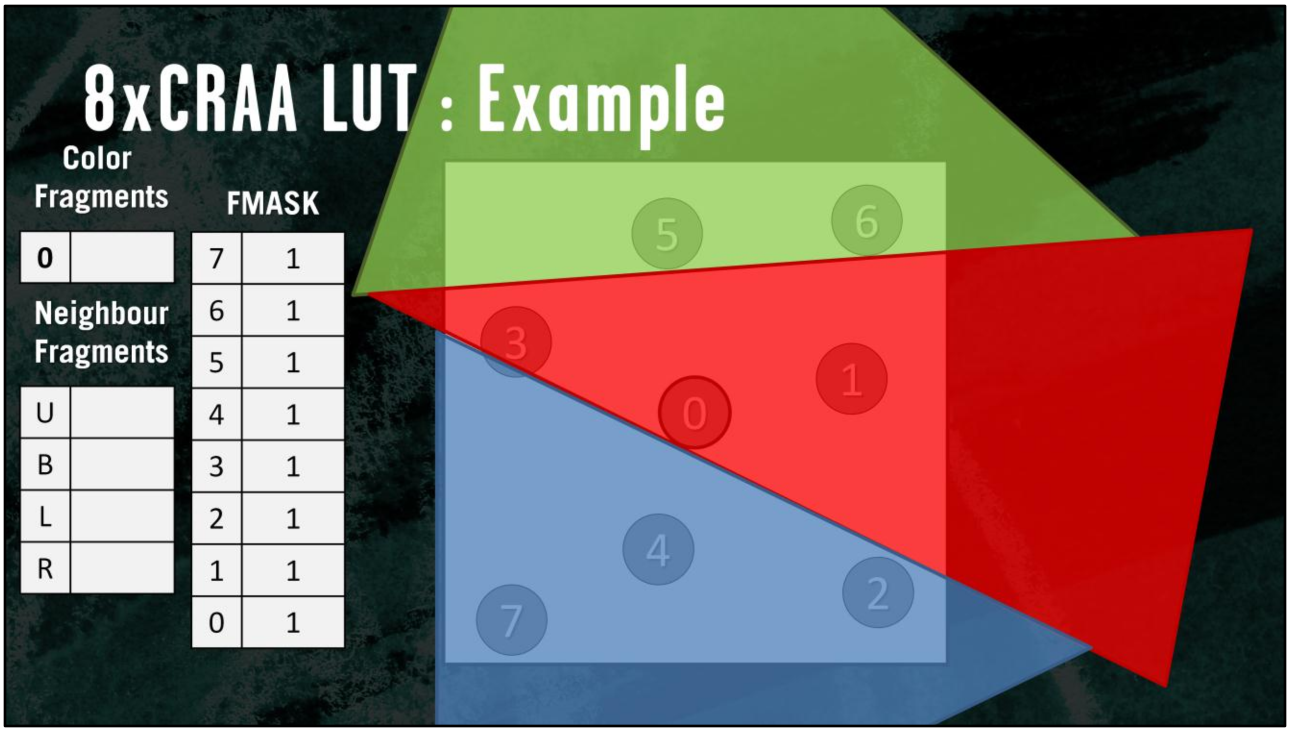
Note: 8xCRAA results are comparable with 8xMSAA apart from pixels that contain multiple triangle intersections.
In that complex case, a single edge estimation can't correctly resolve the edge.
Visible artifacts are similar to analytical methods.



8xCRAA LUT

- What about subpixel artifacts?
- Can we eliminate them?
- Can we get rid of ALU to be only BW bound?

- Solution
- Precompute an LUT to store neighbouring pixel weights
  - Use full neighborhood
  - Multiple edges / triangles crossing the pixel



8xCRAA LUT : Example

Color Fragments

0	

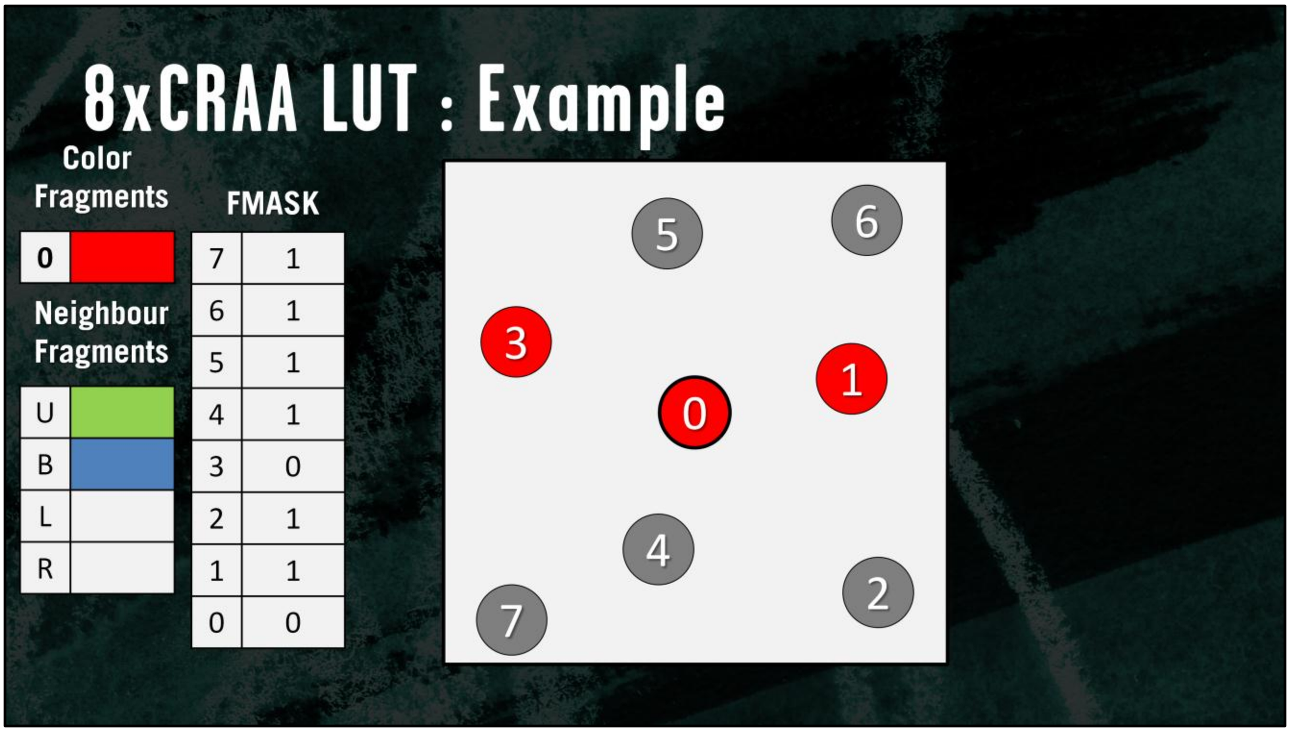
Neighbour Fragments

U	
B	
L	
R	

FMASK

7	1
6	1
5	1
4	1
3	1
2	1
1	1
0	1

In that case analytical would blend RED with BLUE



8xCRAA LUT : Example

Color Fragments

0	🟥

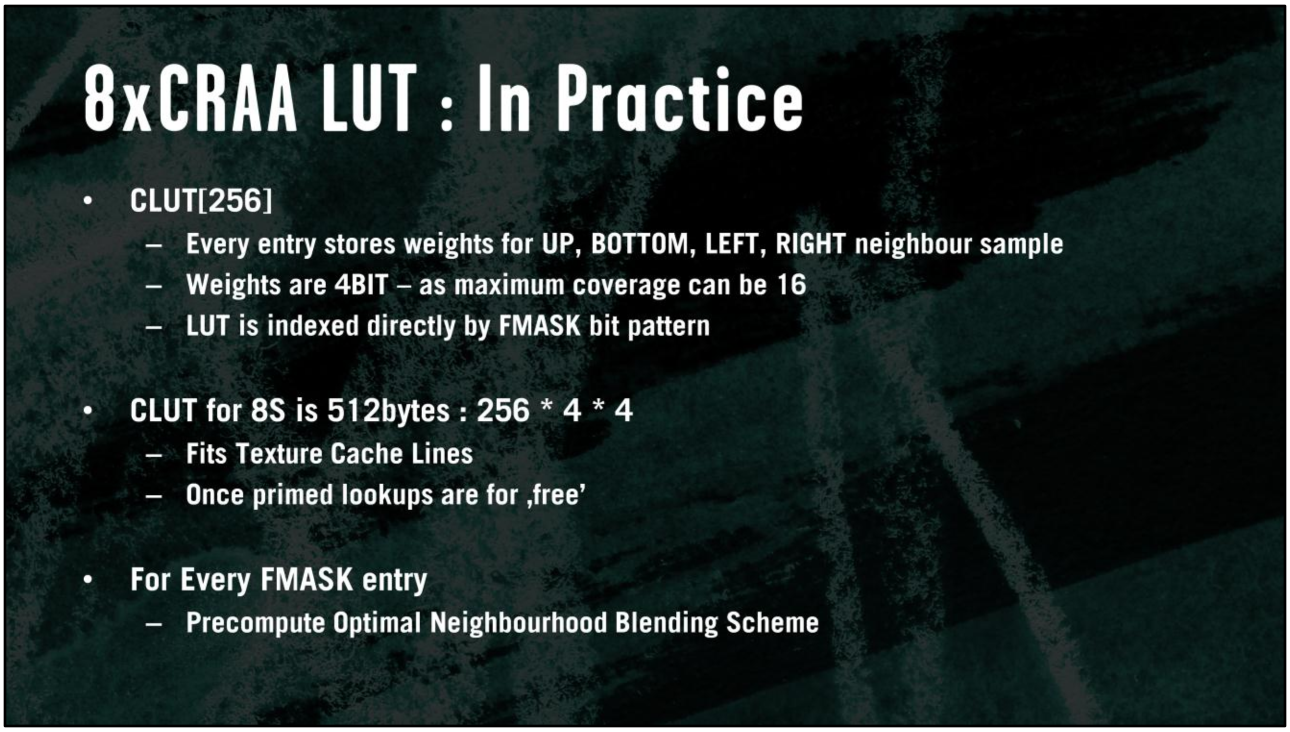
Neighbour Fragments

U	🟩
B	🟦
L	
R	

FMASK

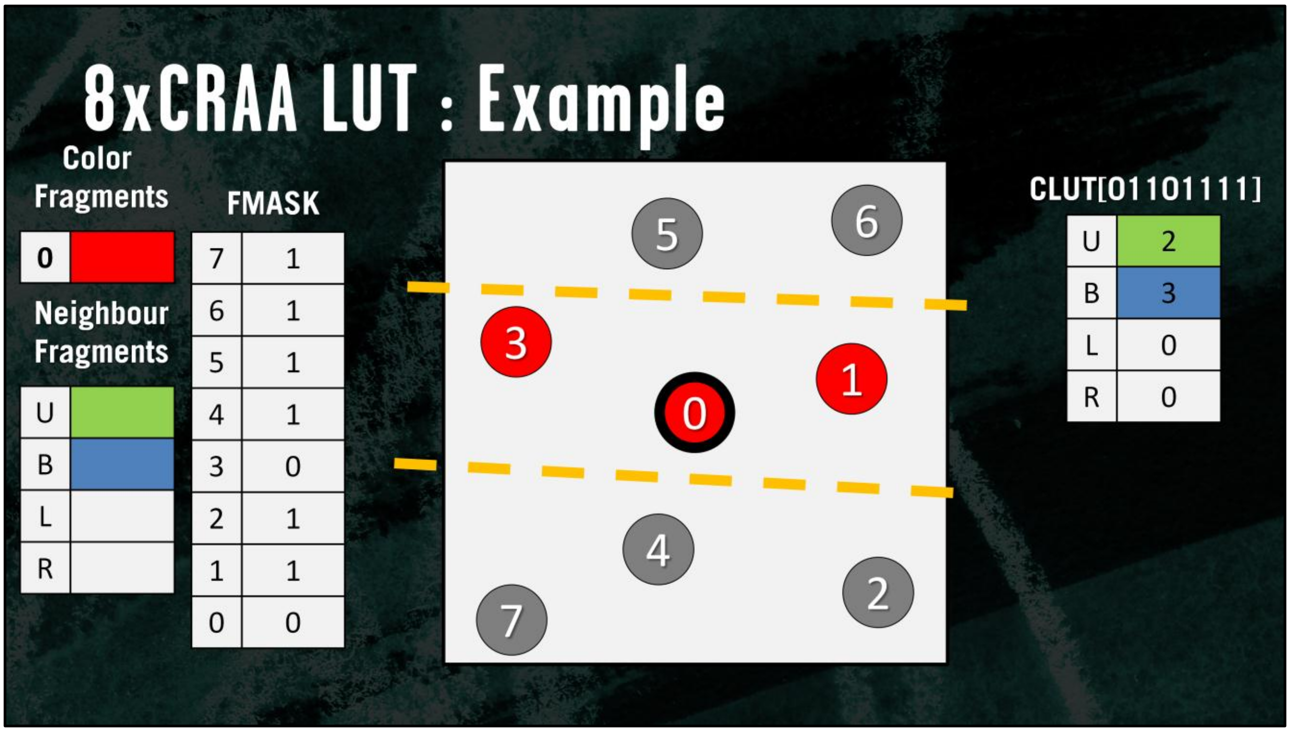
7	1
6	1
5	1
4	1
3	0
2	1
1	1
0	0

In that case analytical would blend RED with BLUE
We need to blend 3 RED with 2 GREEN and 3 BLUE
Only RED got rasterized in current pixel.
Fortunately BLUE and GREEN triangles were big enough to get rasterized in neighborhood (assumption).



8xCRAA LUT : In Practice

- CLUT[256]
  - Every entry stores weights for UP, BOTTOM, LEFT, RIGHT neighbour sample
  - Weights are 4BIT – as maximum coverage can be 16
  - LUT is indexed directly by FMASK bit pattern
- CLUT for 8S is 512bytes : 256 * 4 * 4
  - Fits Texture Cache Lines
  - Once primed lookups are for ‚free’
- For Every FMASK entry
  - Precompute Optimal Neighbourhood Blending Scheme



8xCRAA LUT : Example

Color Fragments

0	🟥

Neighbour Fragments

U	🟩
B	🟦
L	
R	

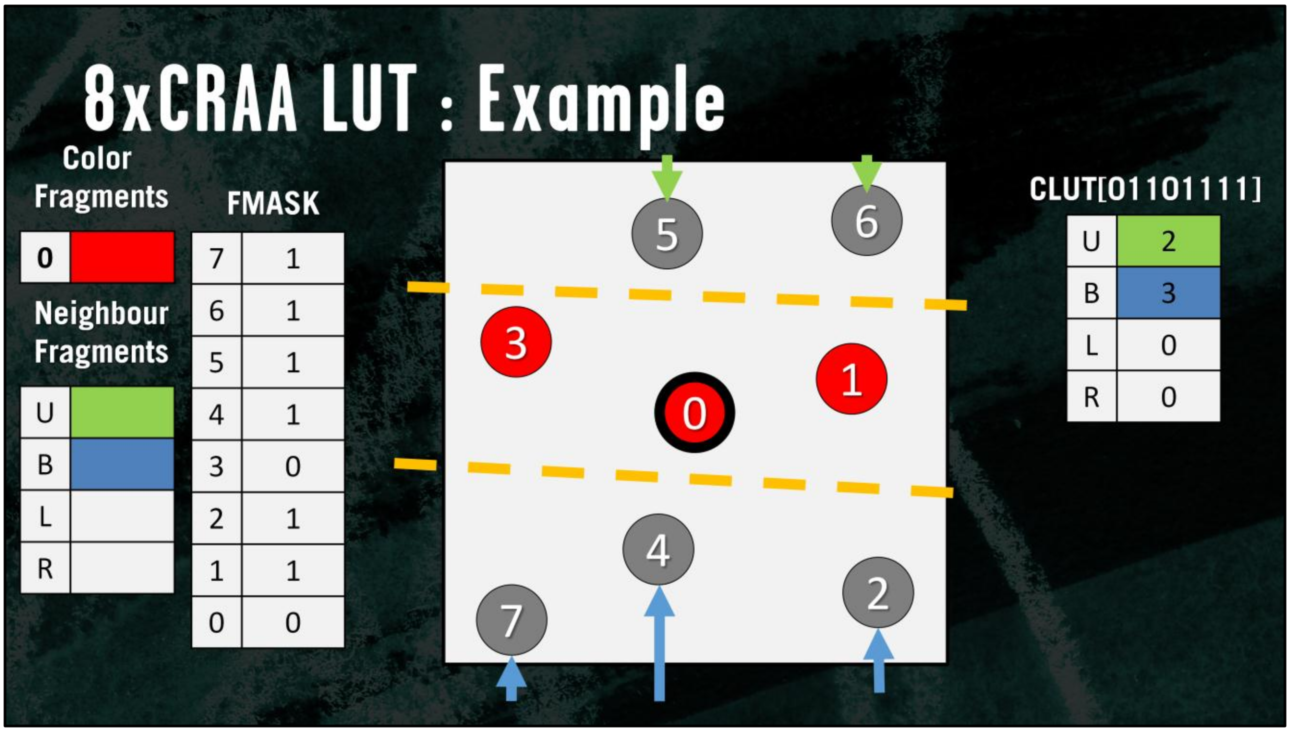
FMASK

7	1
6	1
5	1
4	1
3	0
2	1
1	1
0	0

CLUT[01101111]

U	2
B	3
L	0
R	0

With CLUT we have proper blending of subpixel details



8xCRAA LUT : Example

Color Fragments

0	🟥

Neighbour Fragments

U	🟩
B	🟦
L	
R	

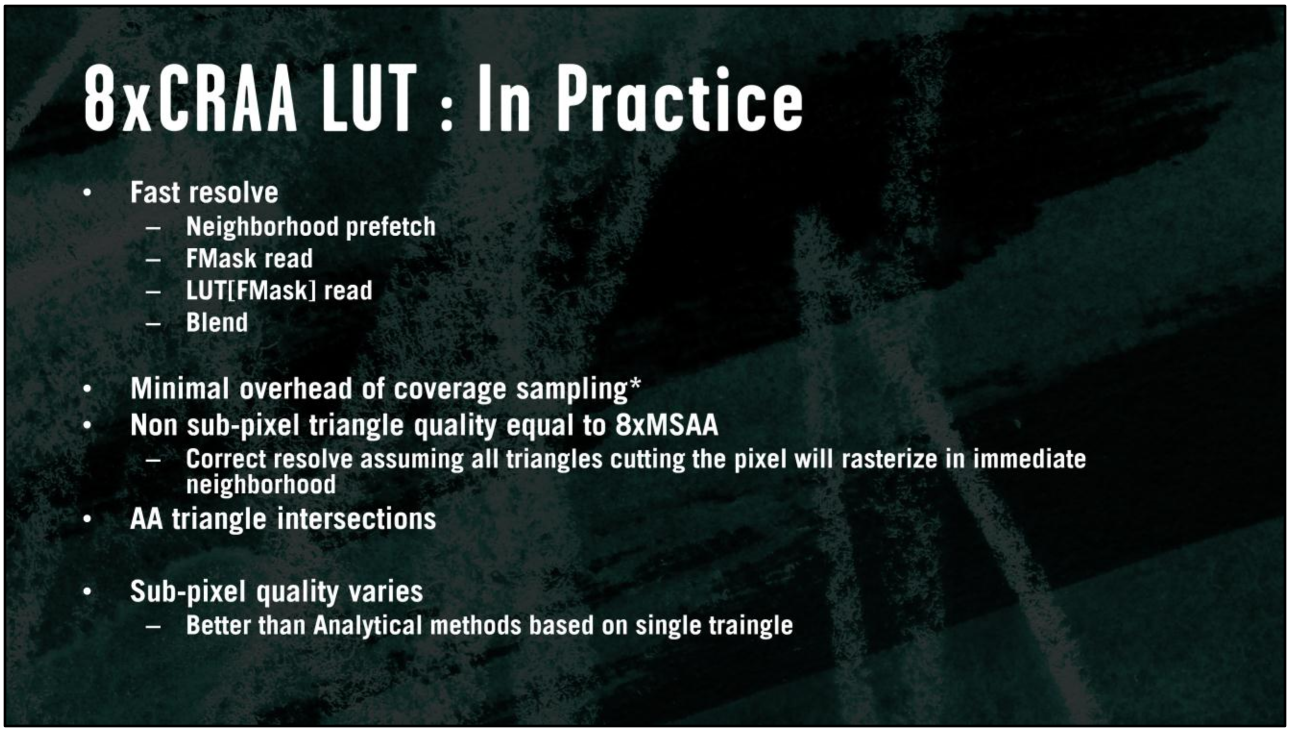
FMASK

7	1
6	1
5	1
4	1
3	0
2	1
1	1
0	0

CLUT[01101111]

U	2
B	3
L	0
R	0

Samples 5 and 6 pull data from UP neighbor.
7, 4, 2 pull data from BOTTOM.



8xCRAA LUT : In Practice

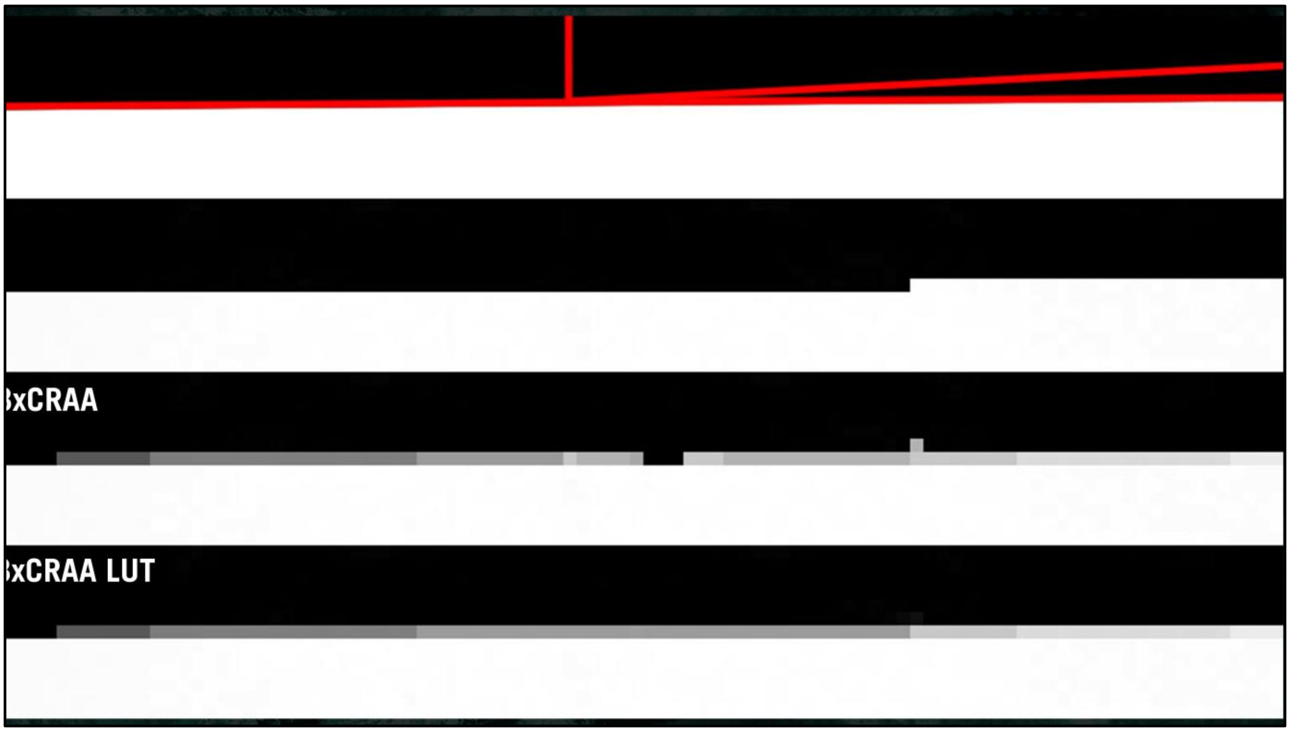
- Fast resolve
  - Neighborhood prefetch
  - FMask read
  - LUT[FMask] read
  - Blend
- Minimal overhead of coverage sampling*
- Non sub-pixel triangle quality equal to 8xMSAA
  - Correct resolve assuming all triangles cutting the pixel will rasterize in immediate neighborhood
- AA triangle intersections
- Sub-pixel quality varies
  - Better than Analytical methods based on single traingle

* You mileage may vary depending on HW, settings etc.

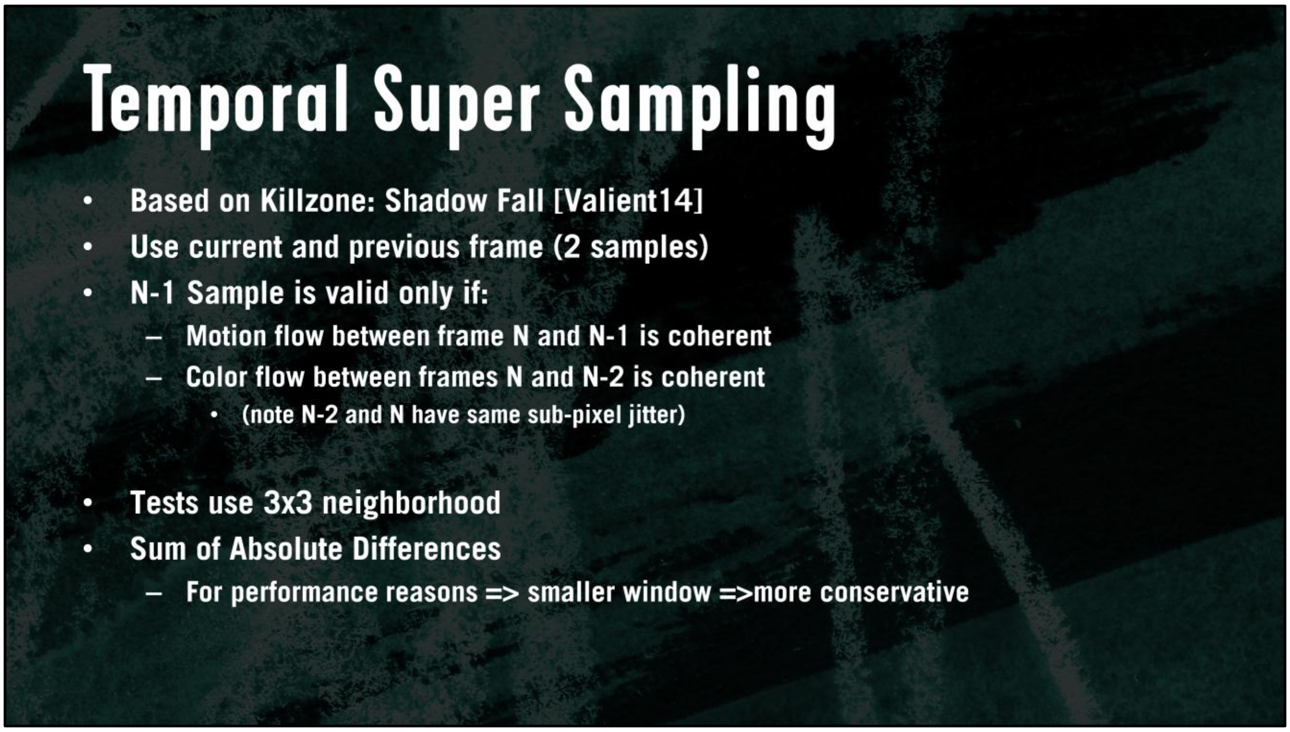
Note: It is still possible to get sub-pixel artifacts (similar to Analytical). However, the chance is lower (still not practical with certain content or tessellation).

Correct Coverage resolve requires specific HW modes settings
Check your IHV
For AMD GCN

    Compressed Depth Buffer
    Depth Testing High Quality Intersections
    Z sample interpolation
    Front to Back sorting



Top to bottom:
- Edge layout
- 1x Centroid Rasterization result
- 8xCRAA (single edge resolve) – fails at pixels cut by multiple edges
- 8xCRAA LUT – correctly resolves pixels cut by multiple edges, assuming that all triangles cutting the pixel will rasterize in immediate neighborhood.



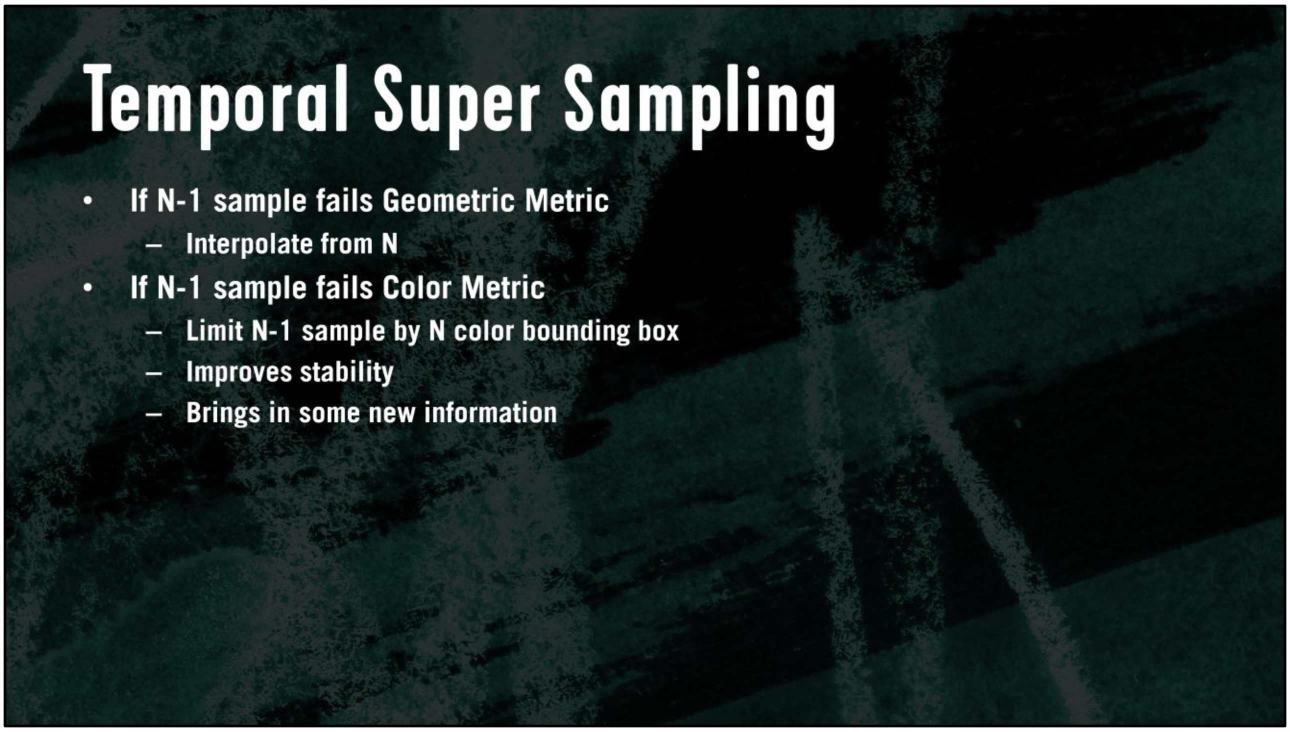
Temporal Super Sampling

- Based on Killzone: Shadow Fall [Valient14]
- Use current and previous frame (2 samples)
- N-1 Sample is valid only if:
  - Motion flow between frame N and N-1 is coherent
  - Color flow between frames N and N-2 is coherent
    - (note N-2 and N have same sub-pixel jitter)

- Tests use 3x3 neighborhood
- Sum of Absolute Differences
  - For performance reasons => smaller window =>more conservative

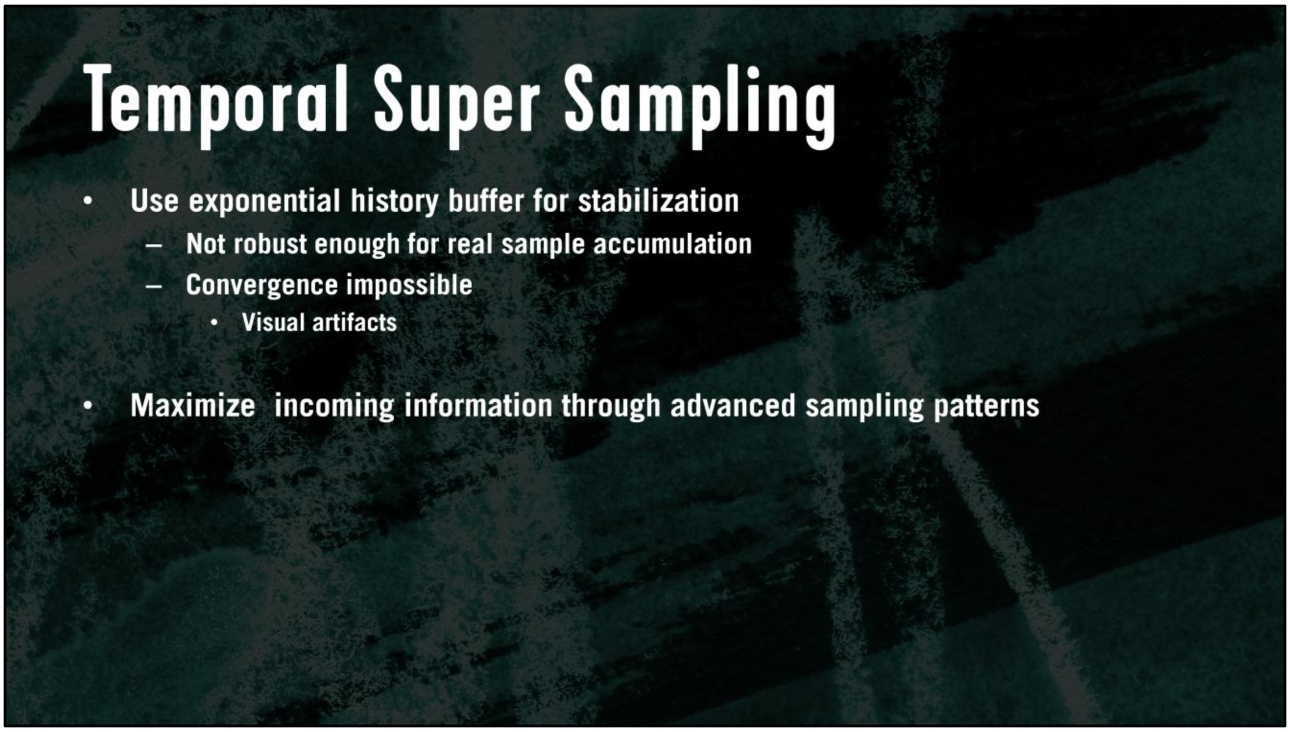
Extension of method from Killzone Shadow Fall.
Only two unique frames are used for actual super-sampling.
Motion Flow constraint guarantees dis-occlusion correctness.
Color Flow constraint guarantees color data freshness as well as lack of jitter in stationary position (this is a fail case for exponential buffer Super-sampling where convergence is impossible to due to cumulative weight).

For similarity test use SAD or other neighborhood similarity metric.
Also SAD based operations are natively supported in HW on AMD GCN architecture (as a part of HW video acceleration).
See **SAD4ShaderInstructions** Cap in DX11
See **msad4** (hlsl)
GCN allows much more useful operations on packed 8bit values : variations of sad, packed lerp…



Temporal Super Sampling

- If N-1 sample fails Geometric Metric
  - Interpolate from N
- If N-1 sample fails Color Metric
  - Limit N-1 sample by N color bounding box
  - Improves stability
  - Brings in some new information

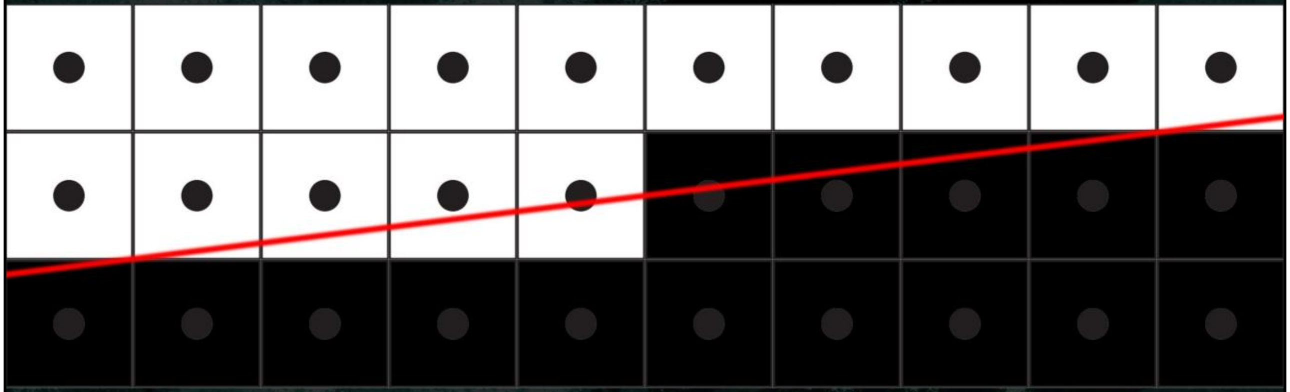


Temporal Super Sampling

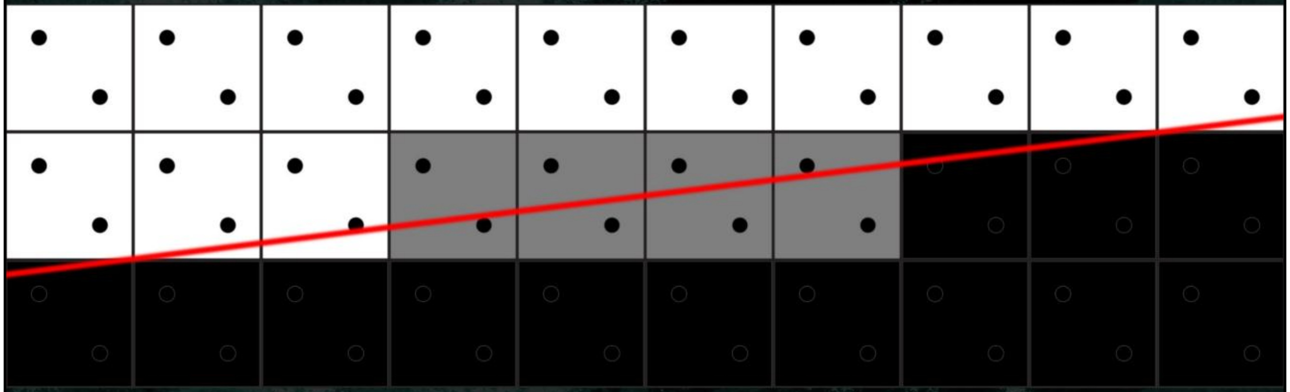
- Use exponential history buffer for stabilization
  - Not robust enough for real sample accumulation
  - Convergence impossible
    - Visual artifacts
- Maximize incoming information through advanced sampling patterns

Exponential history buffers are very convenient for image stabilization.
Unfortunately lack of possibility to remove a single sample from history, results in lack of convergence for finite length patterns.
In effect there is a very hard to remove artifact or high contrast details fading in/out – or 'fizzing'

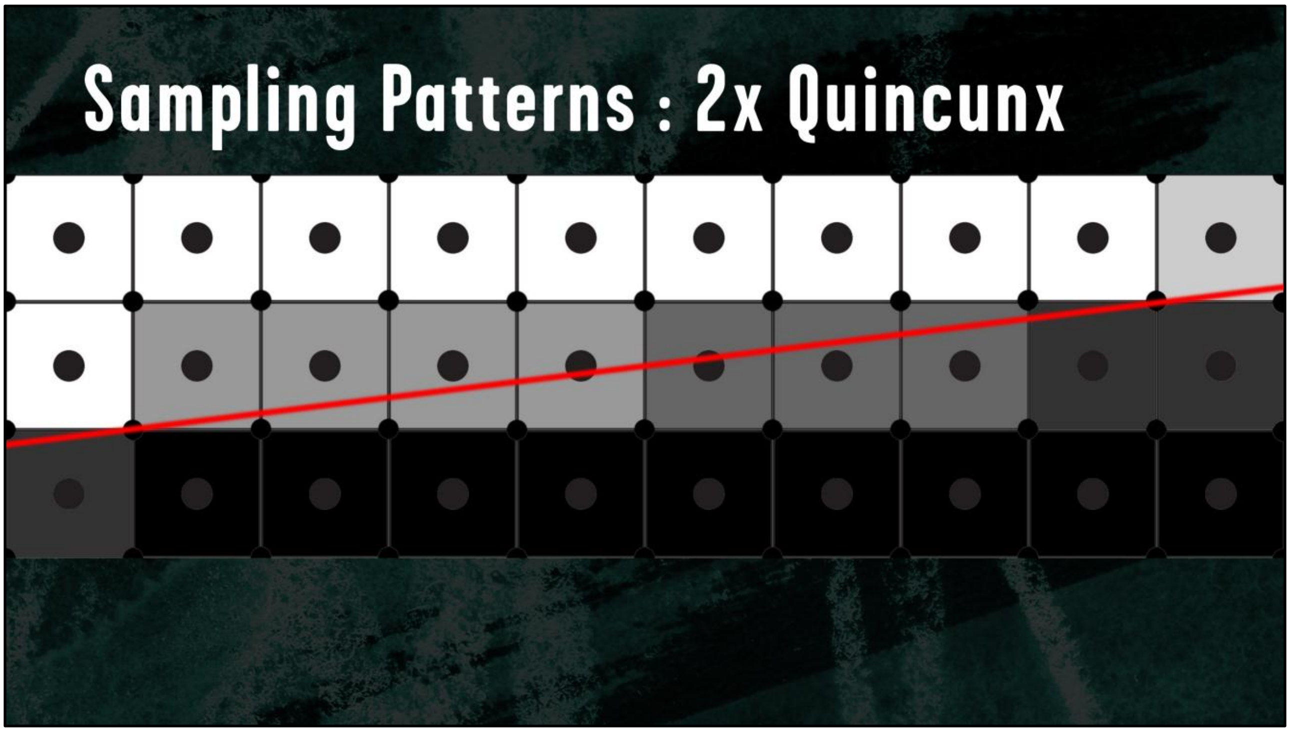
Sampling Patterns : 1x Centroid



Sampling Patterns : 2x Rotated Grid



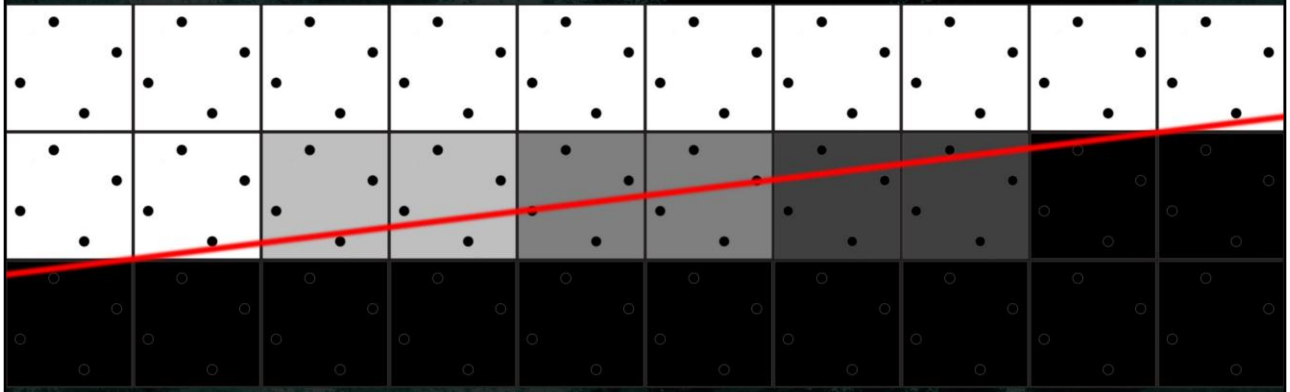
2xRG has 2 unique columns and 2 unique rows.



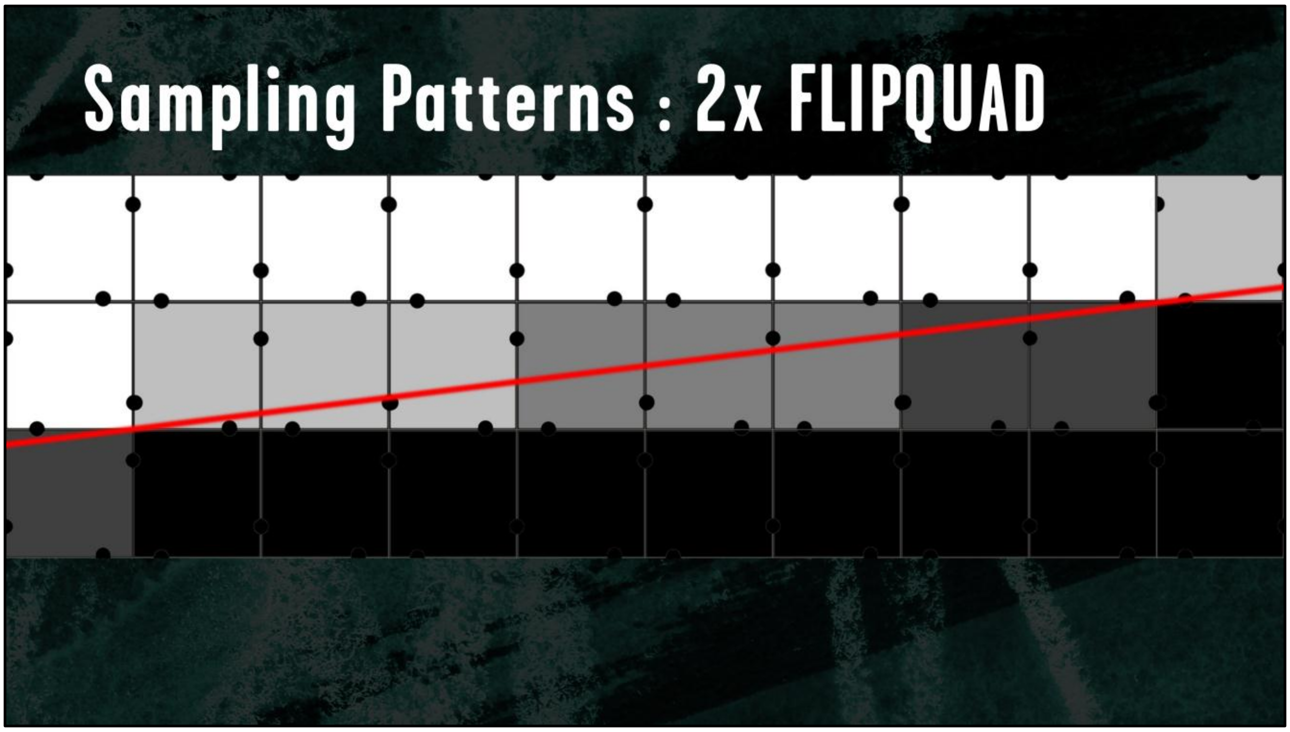
Sampling Patterns : 2x Quincunx

QUINCUNX optimizes the pattern by sharing corner samples with adjacent pixels.
It covers 3 unique rows and 3 unique columns, improving over 2xRG.
It adds a 0.5 radius blur (that is partially recoverable by 0.5 pixel unsharp mask processing).

Sampling Patterns : 4x Rotated Grid

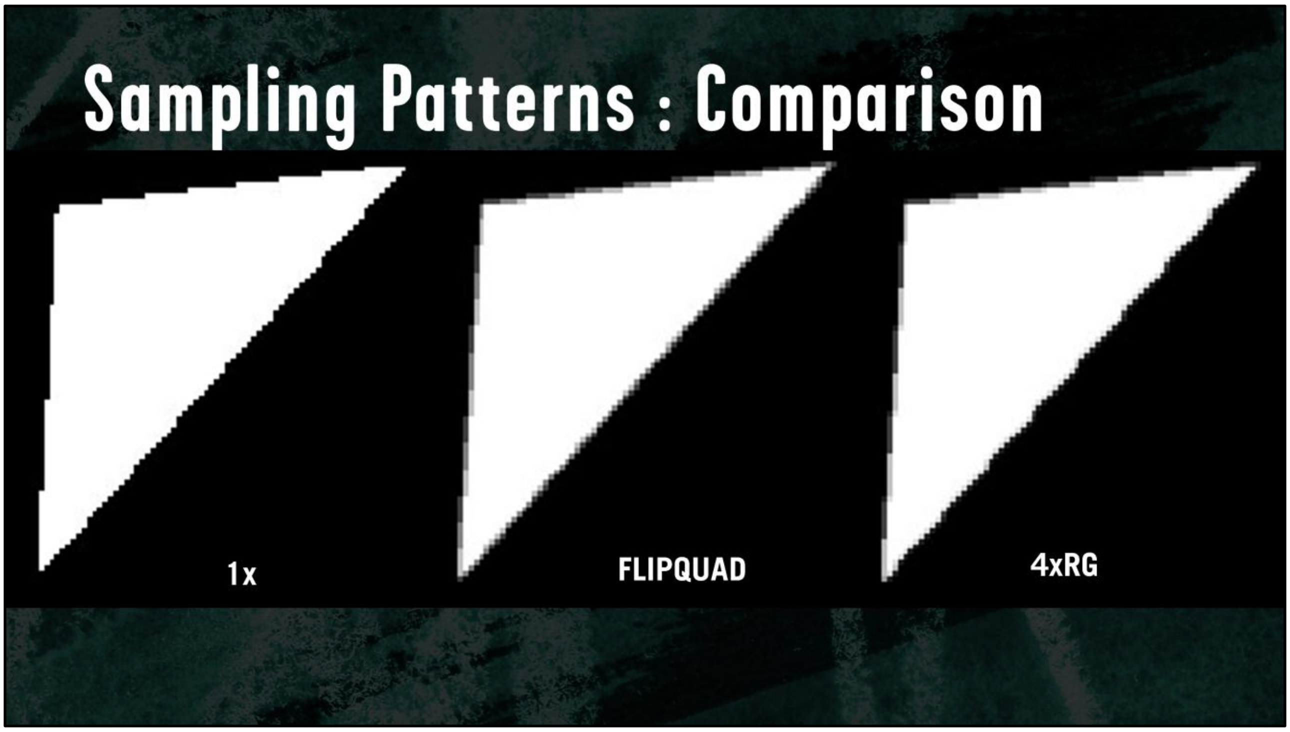


4xRG has 4 unique columns and 4 unique rows.



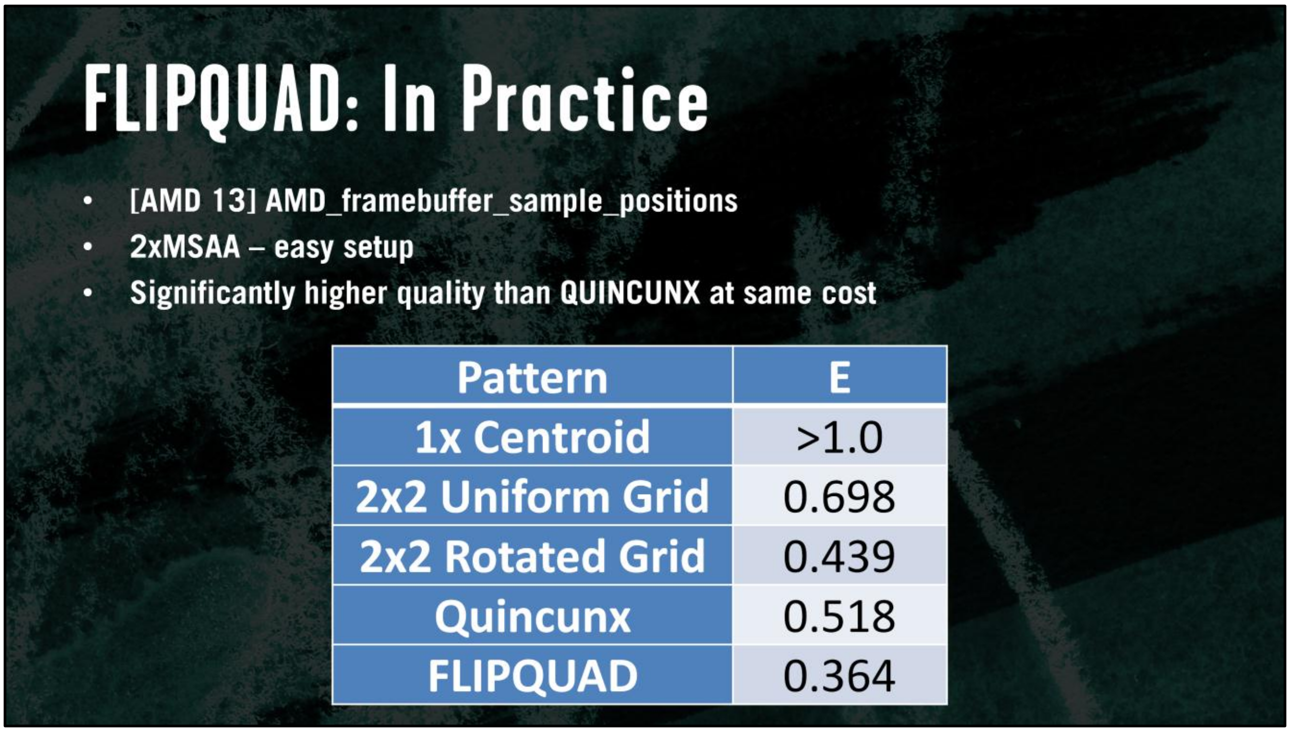
Sampling Patterns : 2x FLIPQUAD

FLIPQUAD is a efficient 2 sample / pixel scheme that allows effective 4x Super-sampling by sharing sampling points on pixel boundary edges.
It combines the benefits of QUINCUNX and Rotated Grid patterns.
Covers 4 unique rows and 4 unique columns, improving over 2xRG and QUINCUNX, matching 4xRG
It adds a 0.5 radius blur (that is partially recoverable by 0.5 pixel unsharp mask processing).



Sampling Patterns : Comparison

Image courtesy [Akenine 03]



FLIPQUAD: In Practice

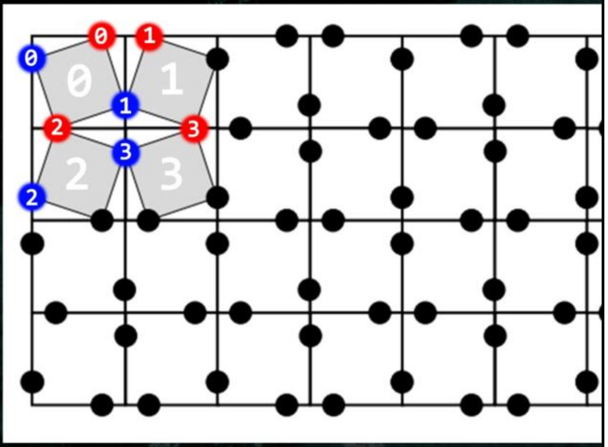
- [AMD 13] AMD_framebuffer_sample_positions
- 2xMSAA – easy setup
- Significantly higher quality than QUINCUNX at same cost

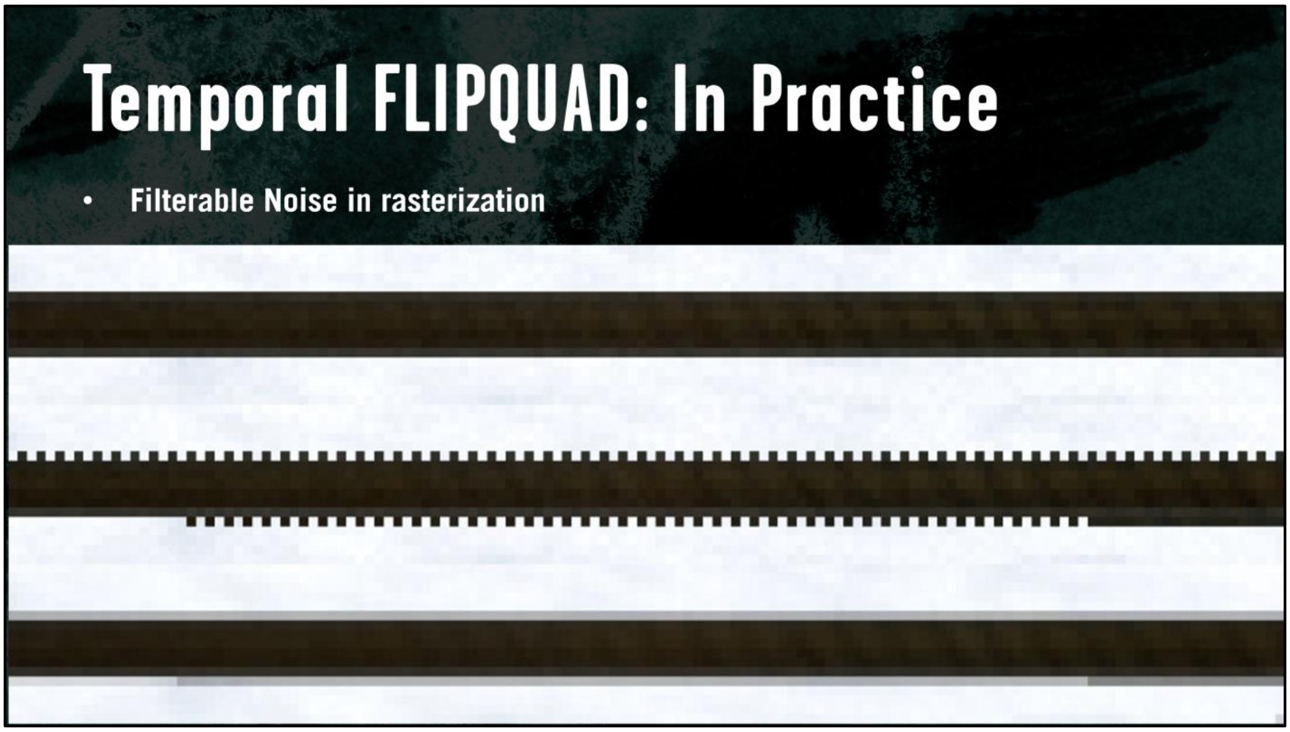
Pattern	E
1x Centroid	>1.0
2x2 Uniform Grid	0.698
2x2 Rotated Grid	0.439
Quincunx	0.518
FLIPQUAD	0.364

Lower error estimate E -> closer to 1024 super-sampled reference.
Image courtesy [Laine 06]

Temporal FLIPQUAD

- Split the pattern in half
- Frame A (BLUE) renders on part Frame B (RED) second
- Needs custom per pixel within quad resolve
  - Convenient blend on X or Y axis depending on frame
- Pixel0 = avg(BLUE(0,1), RED(0,2))

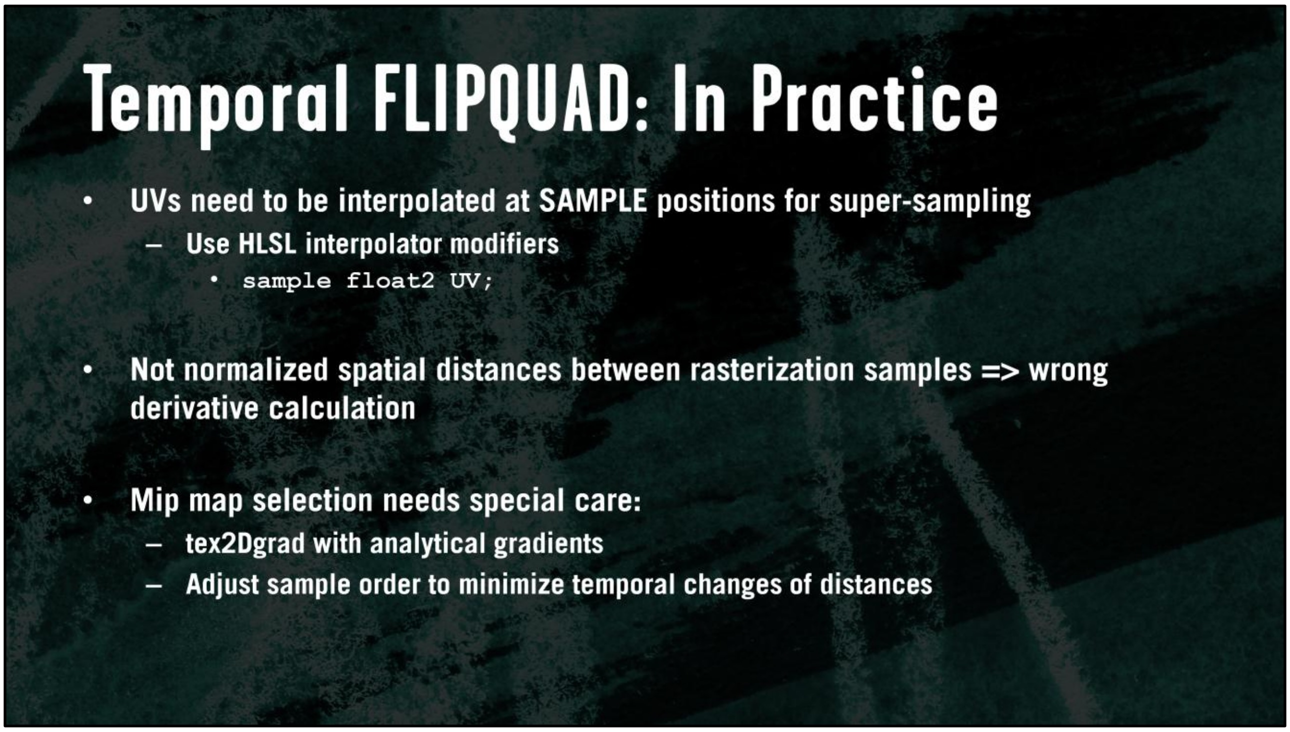




Temporal FLIPQUAD: In Practice

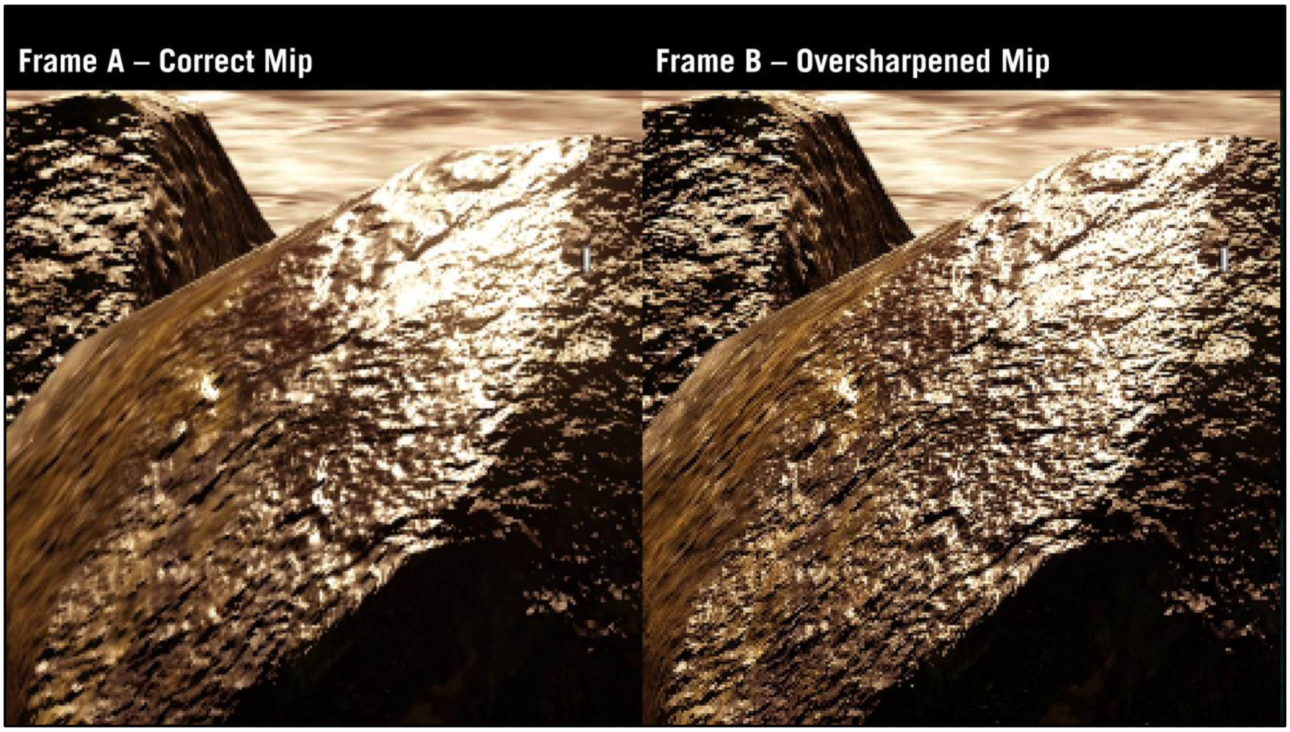
- Filterable Noise in rasterization

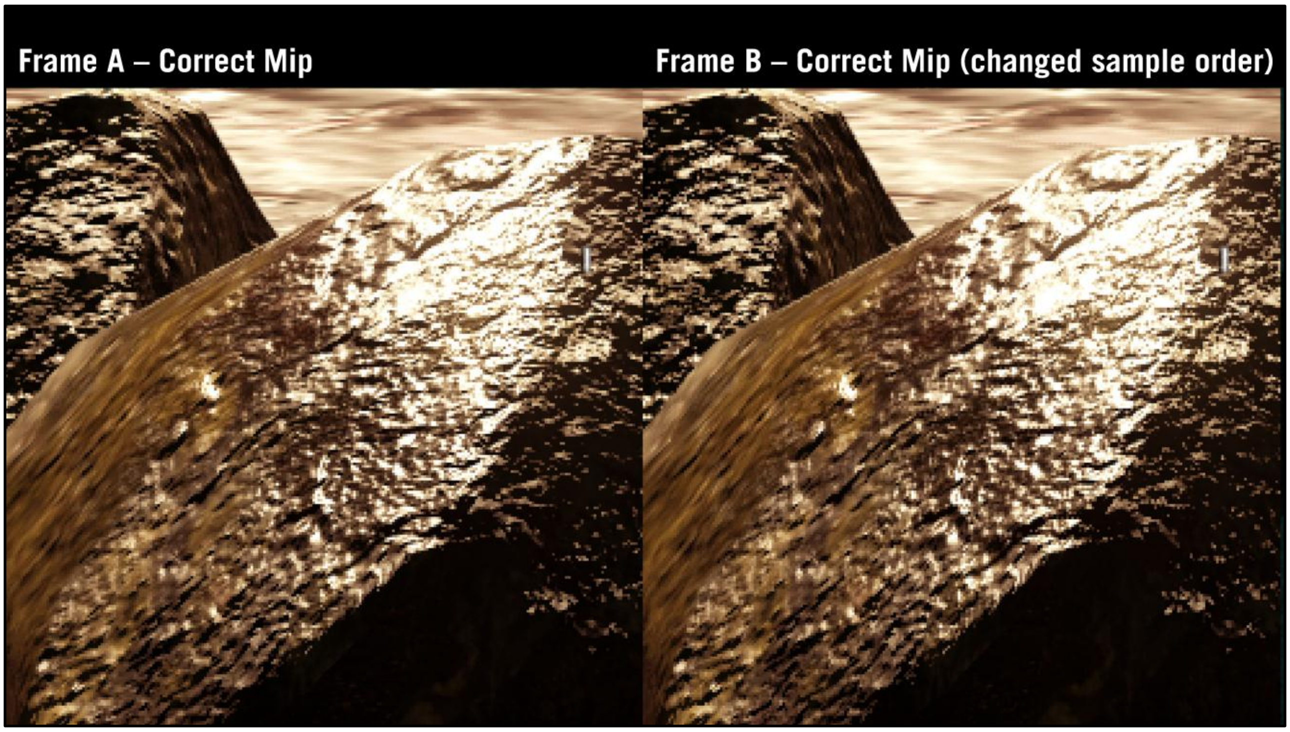
Noise is always contained within a pixel quad, thus easy to filter.

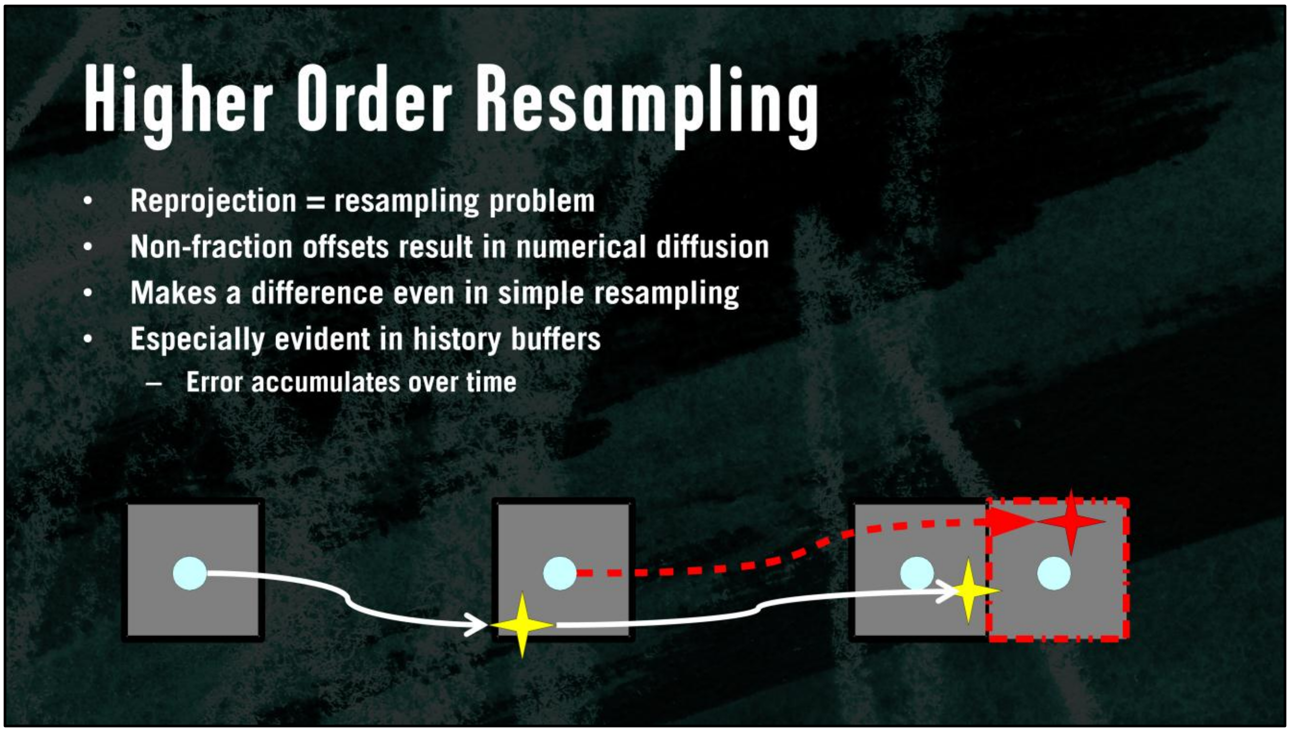

Temporal FLIPQUAD: In Practice

- UVs need to be interpolated at SAMPLE positions for super-sampling
  - Use HLSL interpolator modifiers
    - `sample float2 UV;`
- Not normalized spatial distances between rasterization samples => wrong derivative calculation
- Mip map selection needs special care:
  - tex2Dgrad with analytical gradients
  - Adjust sample order to minimize temporal changes of distances

Be default all texture coordinates will be interpolated at pixel center or centroid.
In case of our FLIPQUAD pattern we want to benefit from super-sampling, therefore UV should be evaluated at sample positions to match the rasterization grid.
Use HLSL interplator modifier : **sample**


Frame A – Correct Mip
Frame B – Oversharpened Mip


Frame A – Correct Mip
Frame B – Correct Mip (changed sample order)



Higher Order Resampling

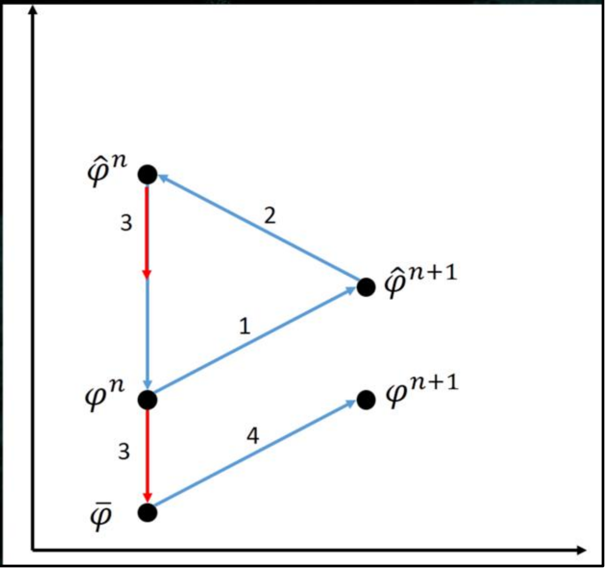
- Reprojection = resampling problem
- Non-fraction offsets result in numerical diffusion
- Makes a difference even in simple resampling
- Especially evident in history buffers
  - Error accumulates over time

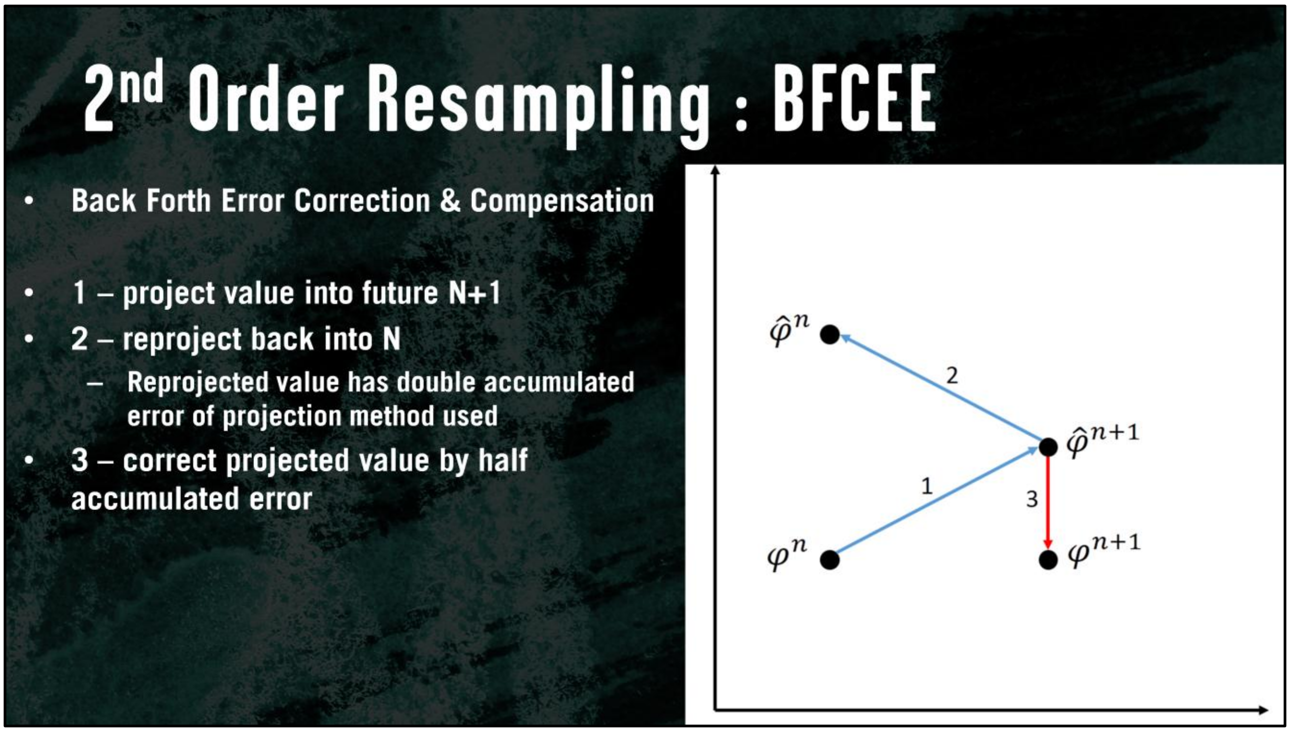
Due to grid quantization a value progression over time can be skewed.
Bilinear Sampling can be interpreted as semi-Lagrangian method (1st order). While unconditionally stable, it oversmoothens the results.

2nd Order Resampling: Mac Cormack

- Mac Cormack Scheme

- 1 – project value into future N+1
- 2 – reproject back into N
  - Reprojected value has double accumulated error of projection method used
- 3 – correct value by half accumulated error
- 4 – project corrected value into N+1

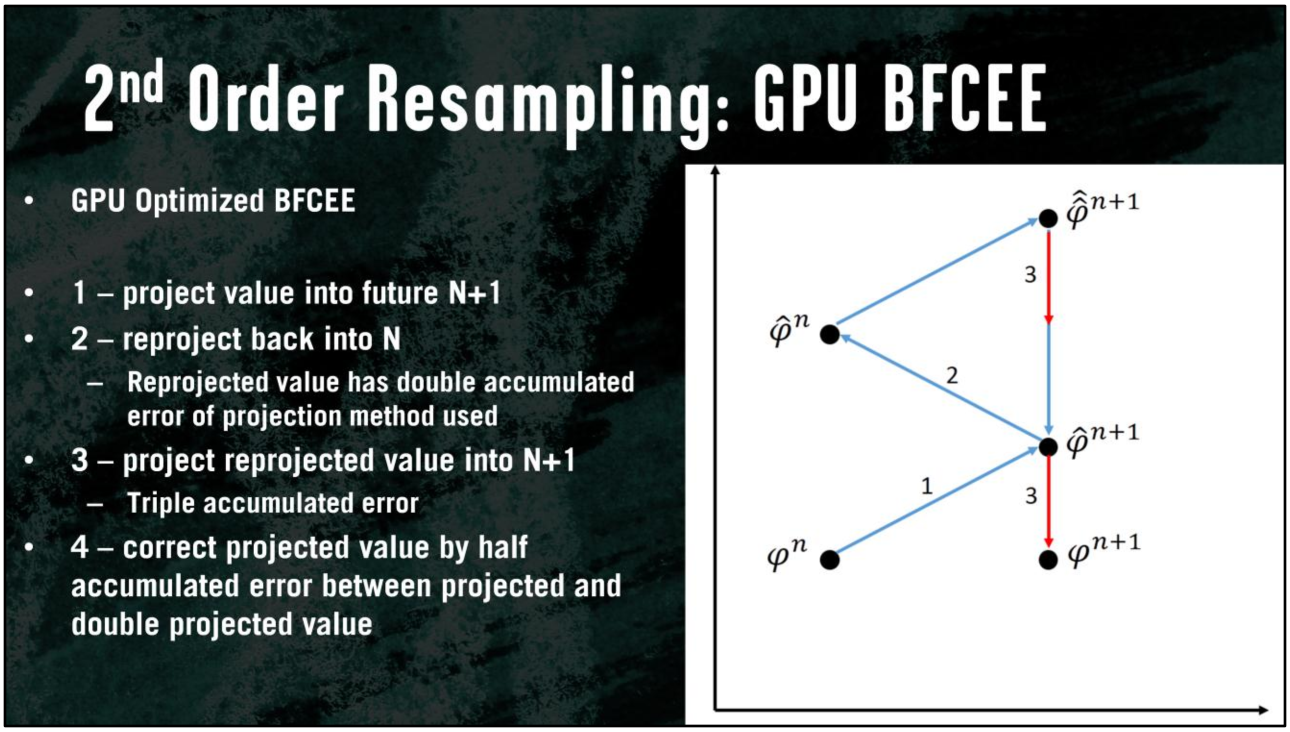




2nd Order Resampling : BFCEE

- Back Forth Error Correction & Compensation
- 1 – project value into future N+1
- 2 – reproject back into N
  - Reprojected value has double accumulated error of projection method used
- 3 – correct projected value by half accumulated error

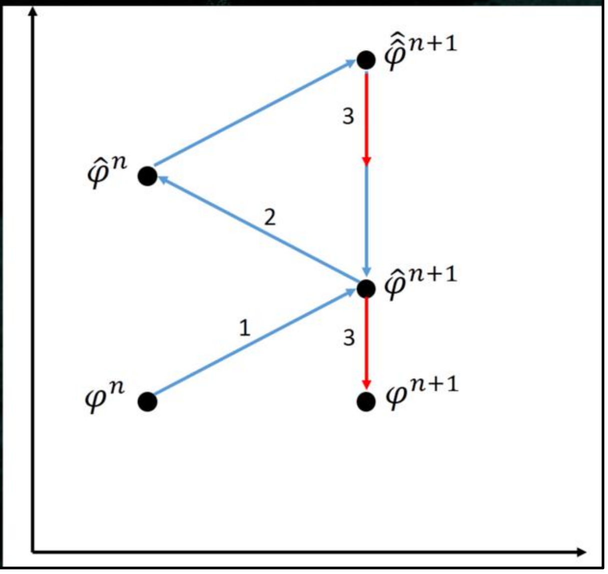
One full projection step is cut in comparison to Mac Cormack.
Unfortunately final value requires memory access to: phi_n, phi_hat_n and phi_hat_n+1



2nd Order Resampling: GPU BFCEE

- GPU Optimized BFCEE

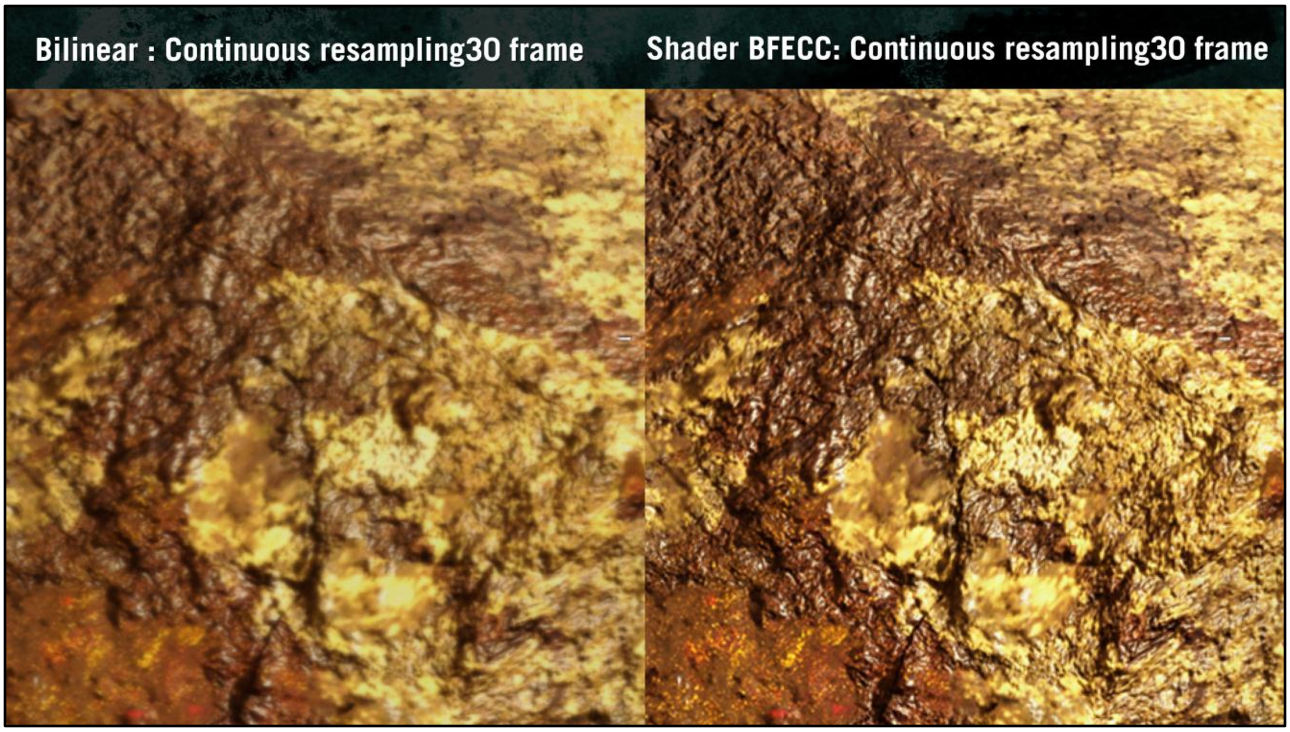
- 1 – project value into future N+1
- 2 – reproject back into N
  - Reprojected value has double accumulated error of projection method used
- 3 – project reprojected value into N+1
  - Triple accumulated error
- 4 – correct projected value by half accumulated error between projected and double projected value



Projections from N to N+1 are quick to evaluate at shader execution time by UV offsets.
Final value requires only access to phi_hat_hat_n+1 and phi_hat_n+1

Effectively this is equivalent to BFCEE in terms of steps, however memory consumption and bandwidth is cut.

Note:
BFCEE ⬄ GPU BFCEE ⬄ MacCormack only for linear projection functions!

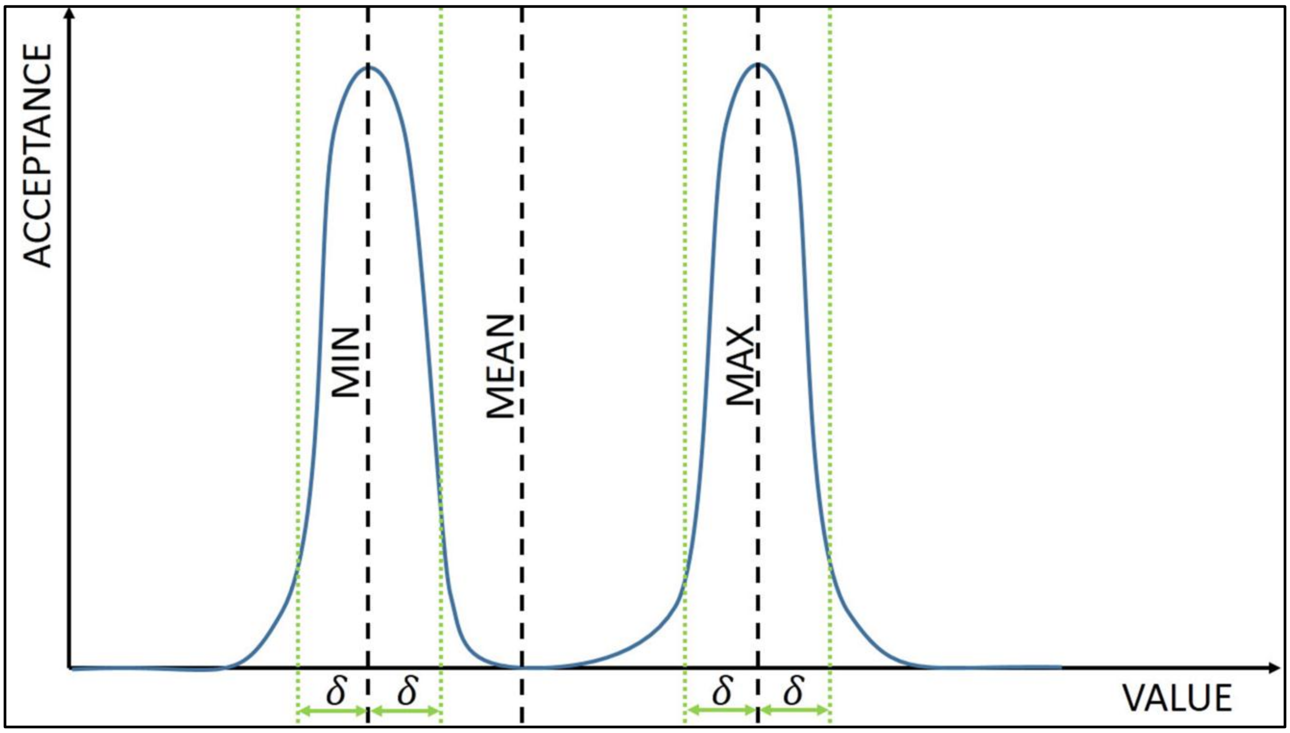

Bilinear : Continuous resampling30 frame
Shader BFECC: Continuous resampling30 frame

This is a result of Exponential History Buffer, resampled over 30 frames. Camera movement speed varied, therefore this can be assumed as average case.
If the relative pixel speed on screen would be exactly 0.5 texel a frame (thus bilinear resampling would always end up doing maximum blur), result would be much worse.

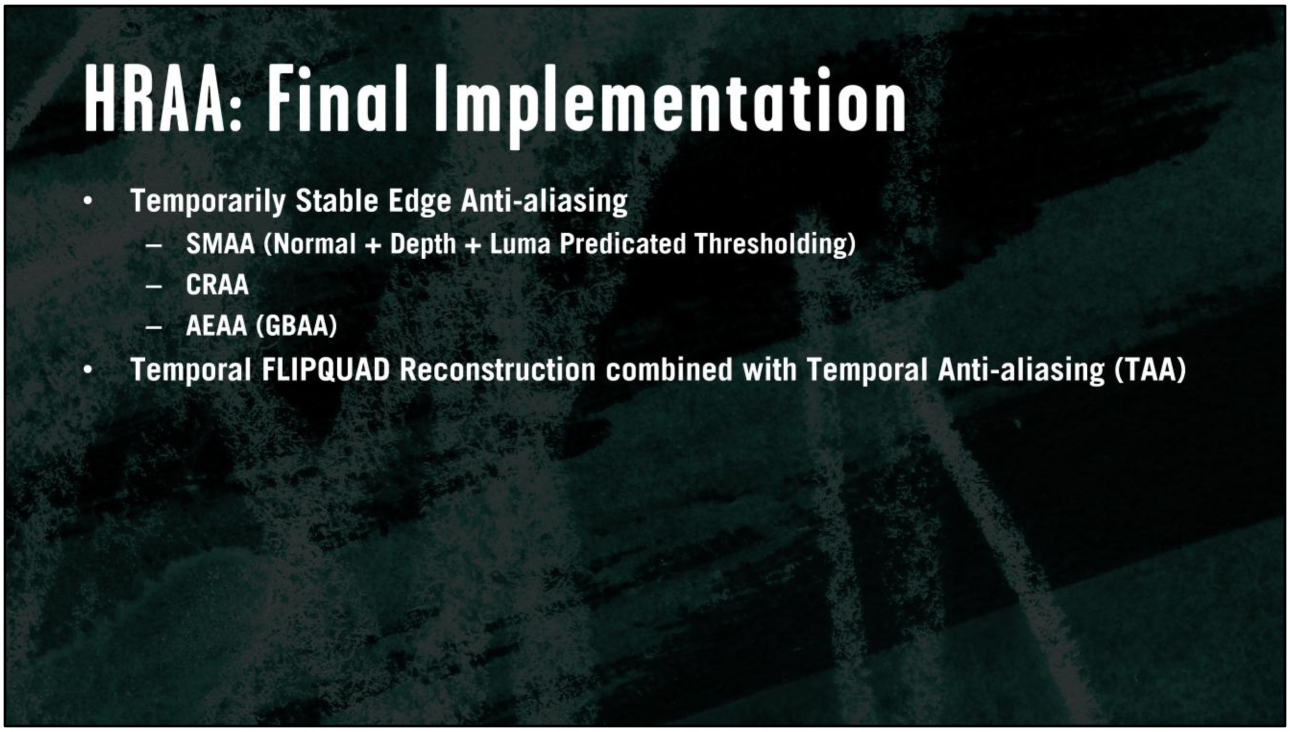
Temporal Anti Aliasing

- History exponential buffer
- Amortize sudden visual changes (flicker)
- Accumulate as much new ‘important’ data as possible

- Use frequency based acceptance metric
- Operate on fresh data neighborhood (3x3 window)
  - History sample close to mean doesn’t bring new information
  - History sample further away brings more information
  - History sample too far might be a fluctuation
- Use local minima / maxima for soft bounds

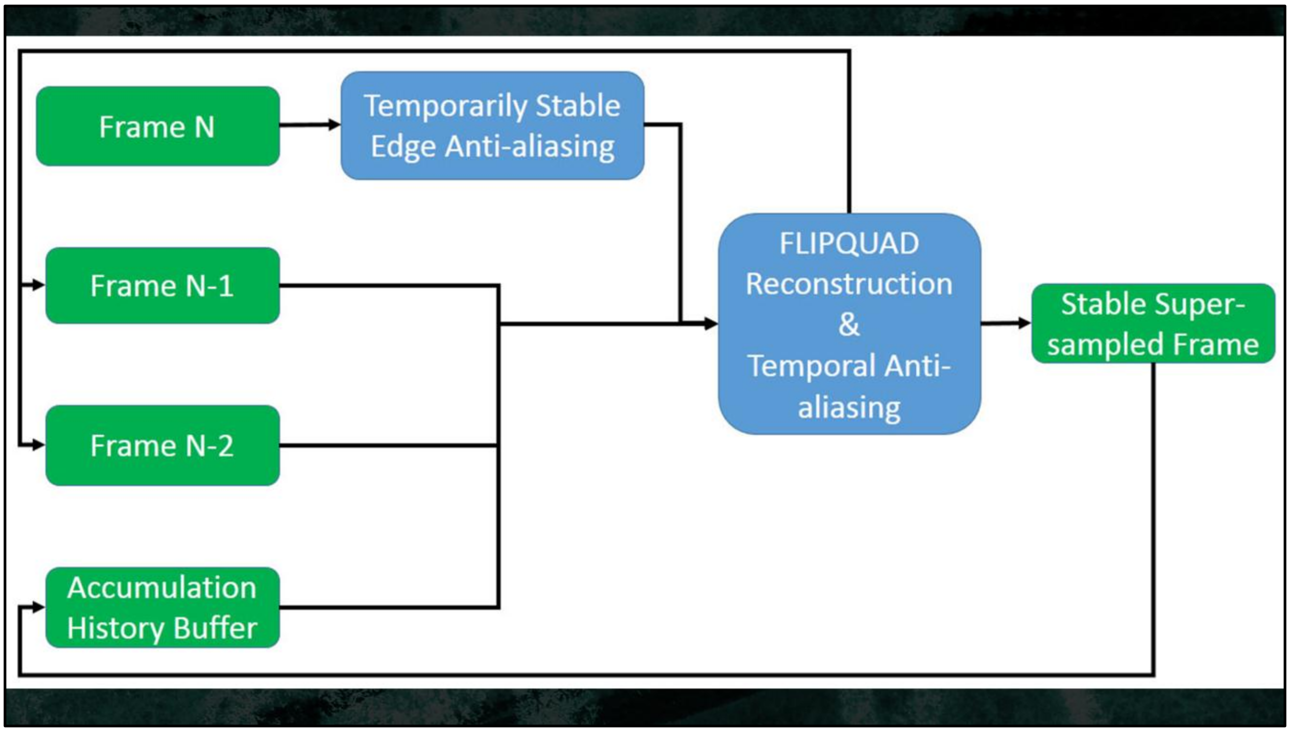


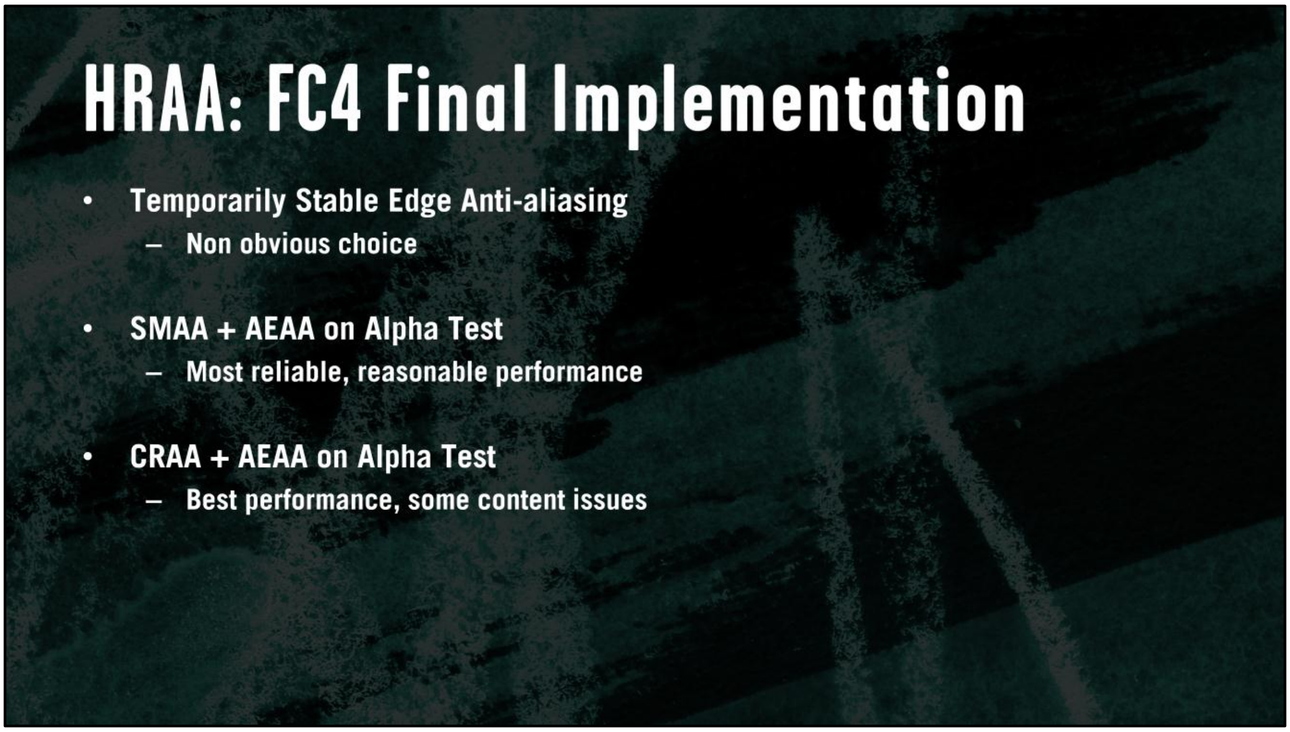
Frequency based acceptance metric
Operate on fresh data neighborhood (3x3 window)
    History sample close to mean doesn't bring new information
    History sample further away brings more information
    History sample too far might be a fluctuation
Local minima / maxima provide soft bounds for exponential weight function.



HRAA: Final Implementation

- Temporarily Stable Edge Anti-aliasing
  - SMAA (Normal + Depth + Luma Predicated Thresholding)
  - CRAA
  - AEAA (GBAA)
- Temporal FLIPQUAD Reconstruction combined with Temporal Anti-aliasing (TAA)

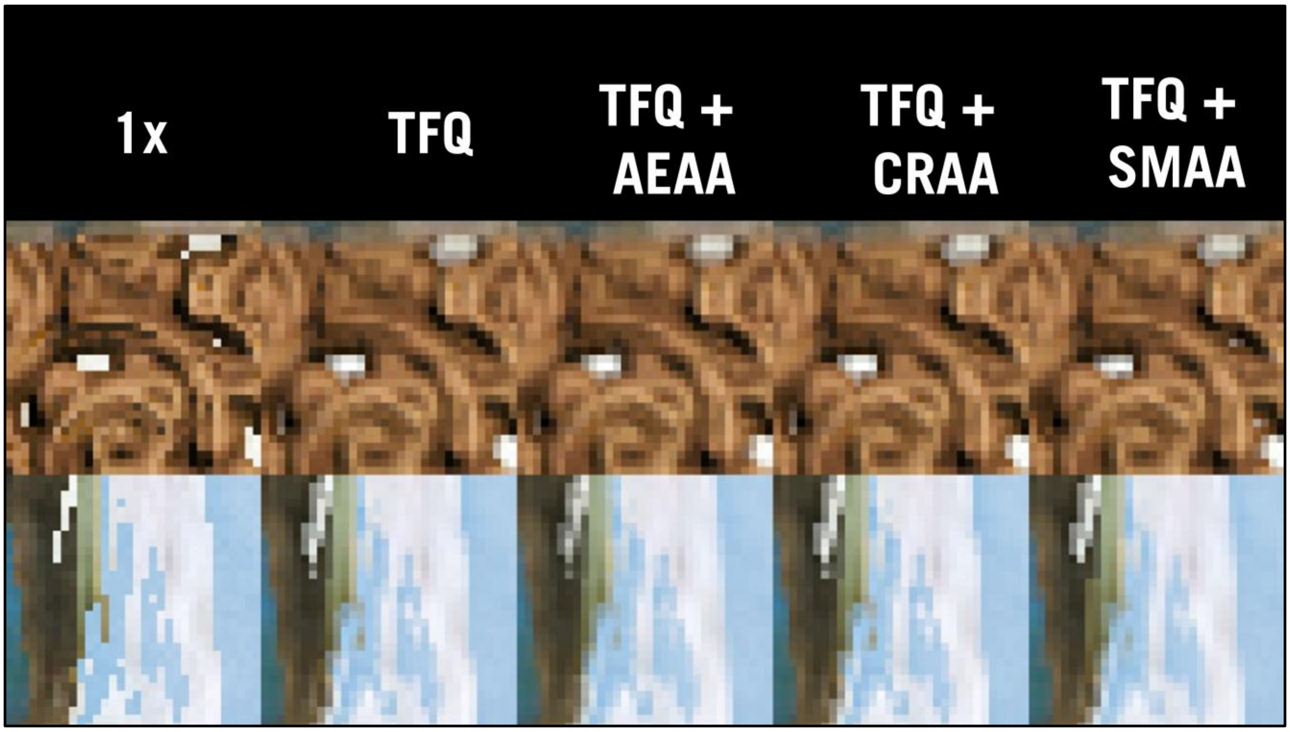

Frame N
Temporarily Stable Edge Anti-aliasing
Frame N-1
Frame N-2
Accumulation History Buffer
FLIPQUAD Reconstruction & Temporal Anti-aliasing
Stable Super-sampled Frame

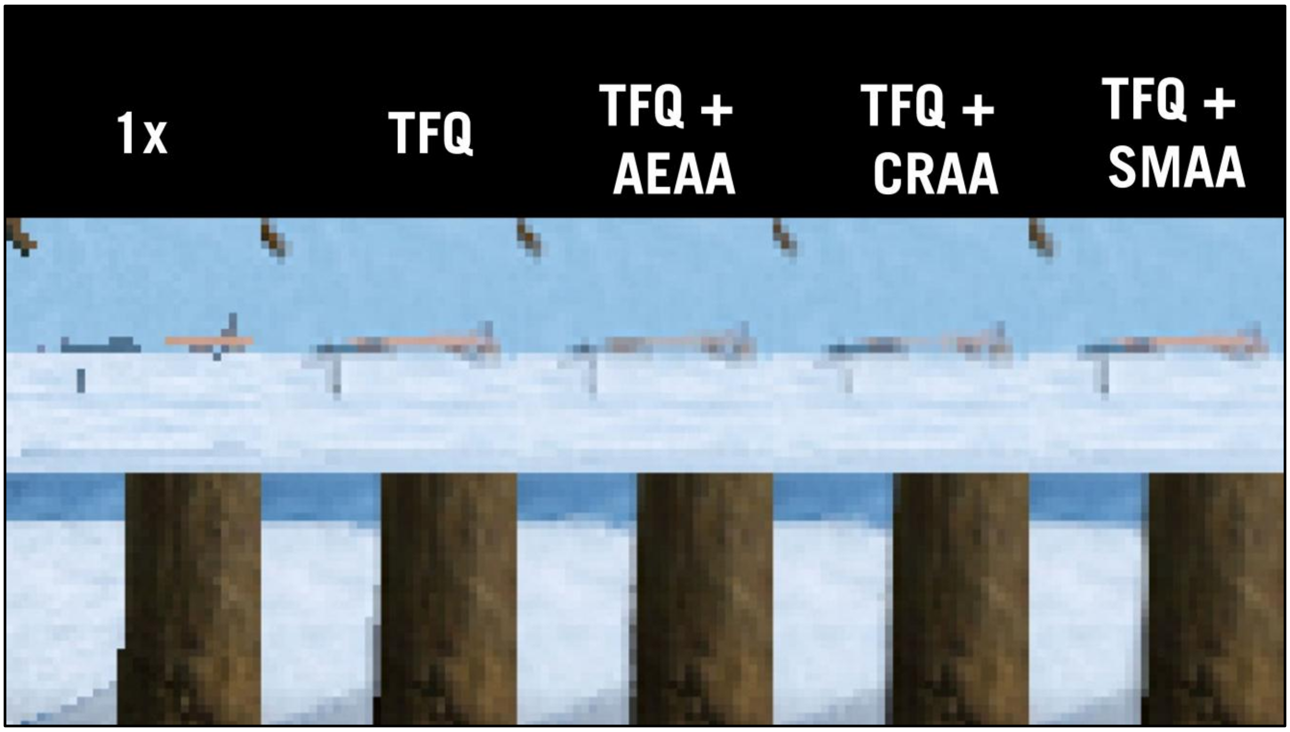


HRAA: FC4 Final Implementation

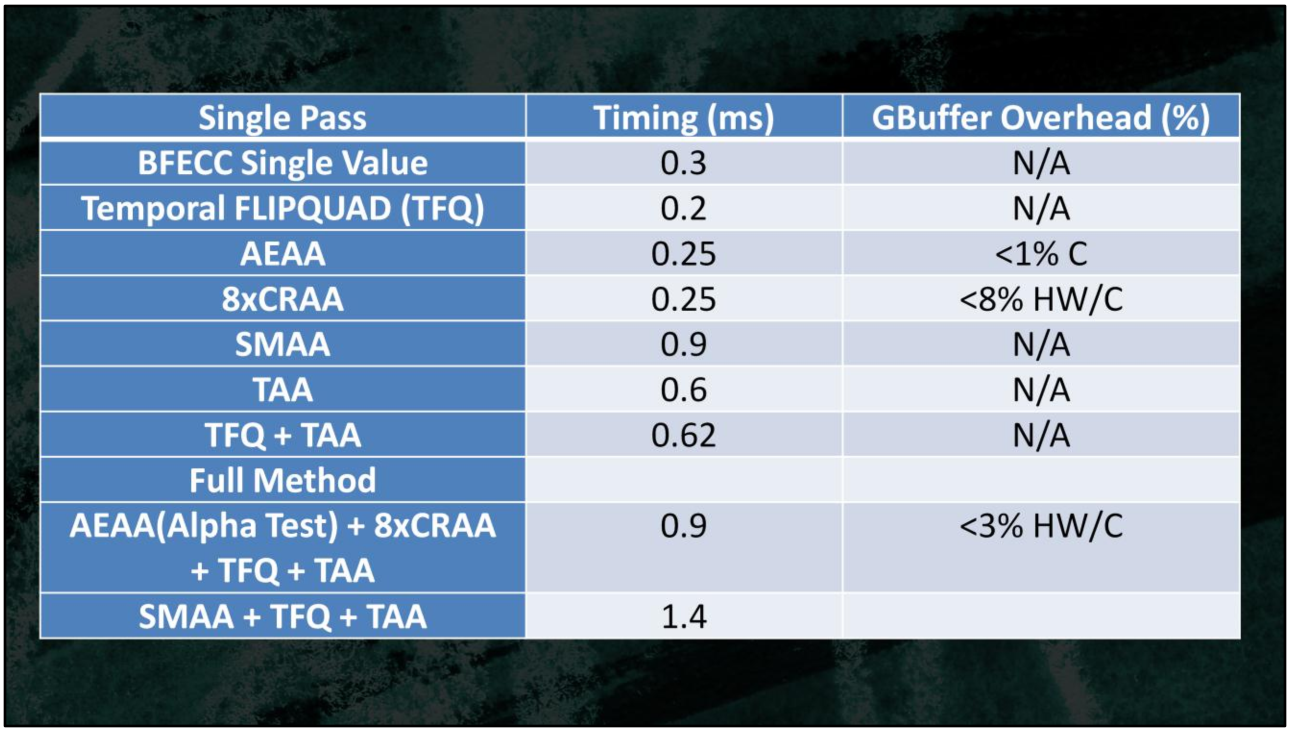
- Temporarily Stable Edge Anti-aliasing
  - Non obvious choice
- SMAA + AEAA on Alpha Test
  - Most reliable, reasonable performance
- CRAA + AEAA on Alpha Test
  - Best performance, some content issues

We are still on the fence with final method we will ship.
If we would live in perfect world with all content having perfect LODs , we would go CRAA.
Otherwise SMAA + AEAA on Alpha Test (this makes the whole pass cheaper and more stable on vegetation that is really important in our use case).


1x
TFQ
TFQ + AEAA
TFQ + CRAA
TFQ + SMAA



Note: Reconstruction capabilities of all TFQ based methods.

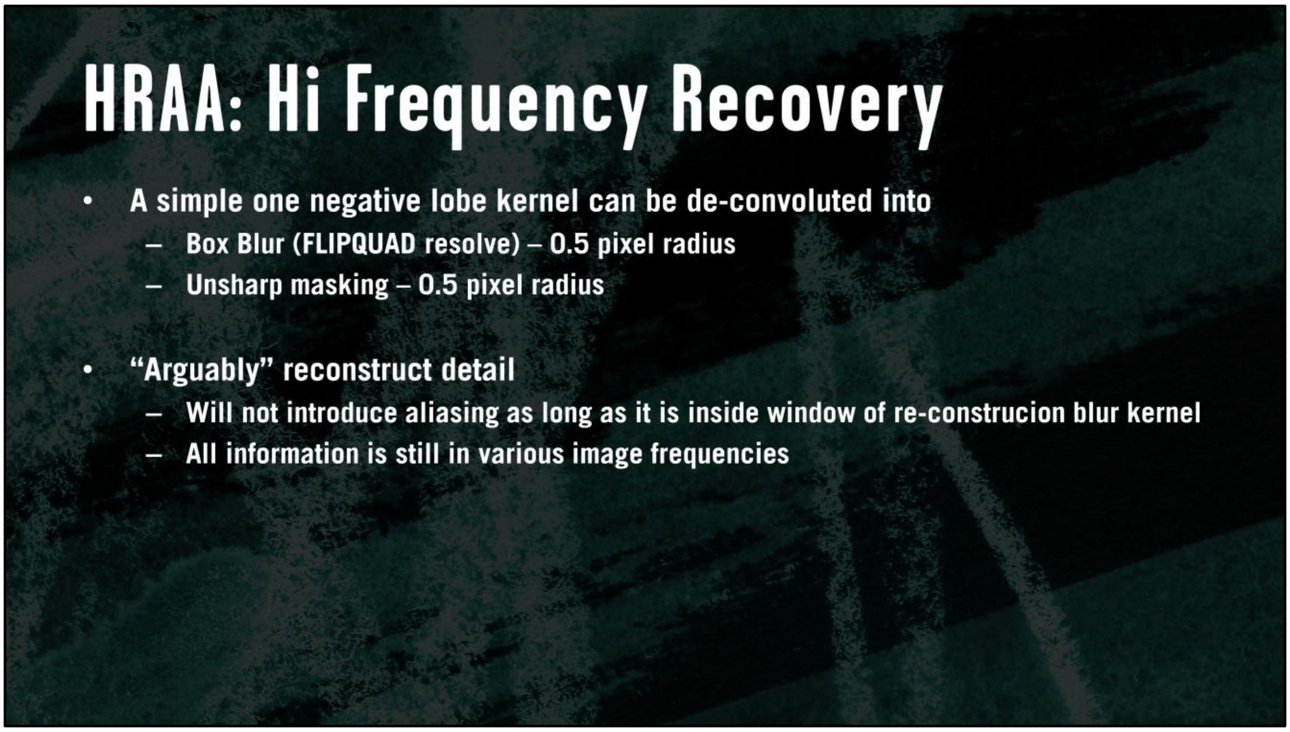


Single Pass	Timing (ms)	GBuffer Overhead (%)
BFECC Single Value	0.3	N/A
Temporal FLIPQUAD (TFQ)	0.2	N/A
AEAA	0.25	<1% C
8xCRAA	0.25	<8% HW/C
SMAA	0.9	N/A
TAA	0.6	N/A
TFQ + TAA	0.62	N/A
Full Method		
AEAA(Alpha Test) + 8xCRAA + TFQ + TAA	0.9	<3% HW/C
SMAA + TFQ + TAA	1.4	

HRAA: Hi Frequency Recovery

- FLIPQUAD BOX resolve kernel
  - Results in 0.5 blur
  - Art direction 'might' find it objectionable
- In real super-sampling a wider, complex kernel would be needed to preserve:
  - Anti-alias
  - Hi-Frequency
- Check [Burley 07]

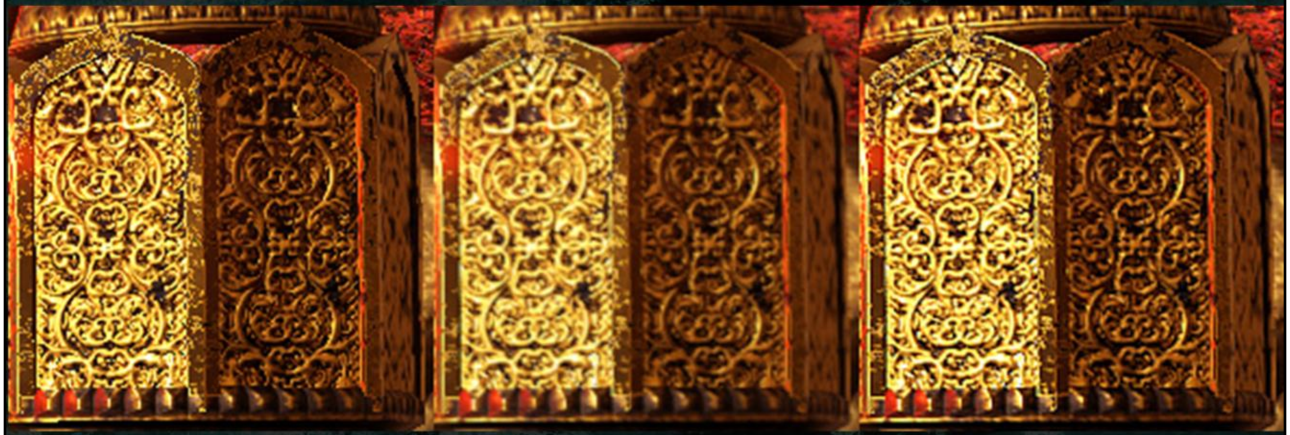
Moderne DINER
mitchell 4.0 (ShadingRate 1.0)
Moderne DINER
mitchell 4.0 (sharpness 1.5)



HRAA: Hi Frequency Recovery

- A simple one negative lobe kernel can be de-convoluted into
  - Box Blur (FLIPQUAD resolve) – 0.5 pixel radius
  - Unsharp masking – 0.5 pixel radius
- “Arguably” reconstruct detail
  - Will not introduce aliasing as long as it is inside window of re-construcion blur kernel
  - All information is still in various image frequencies

HRAA: Hi Frequency Recovery



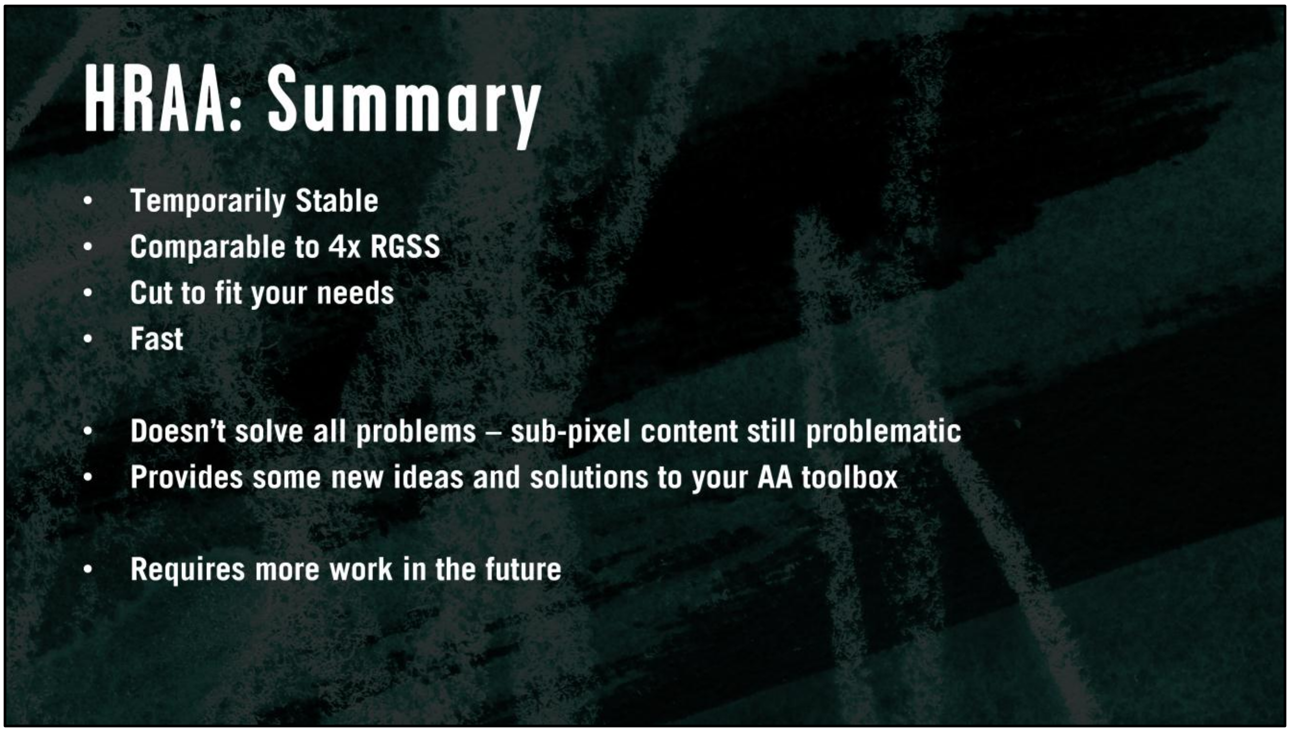
1x

FQ

FQ + Unsharp

(oversharpened for effect)

MOVIE



HRAA: Summary

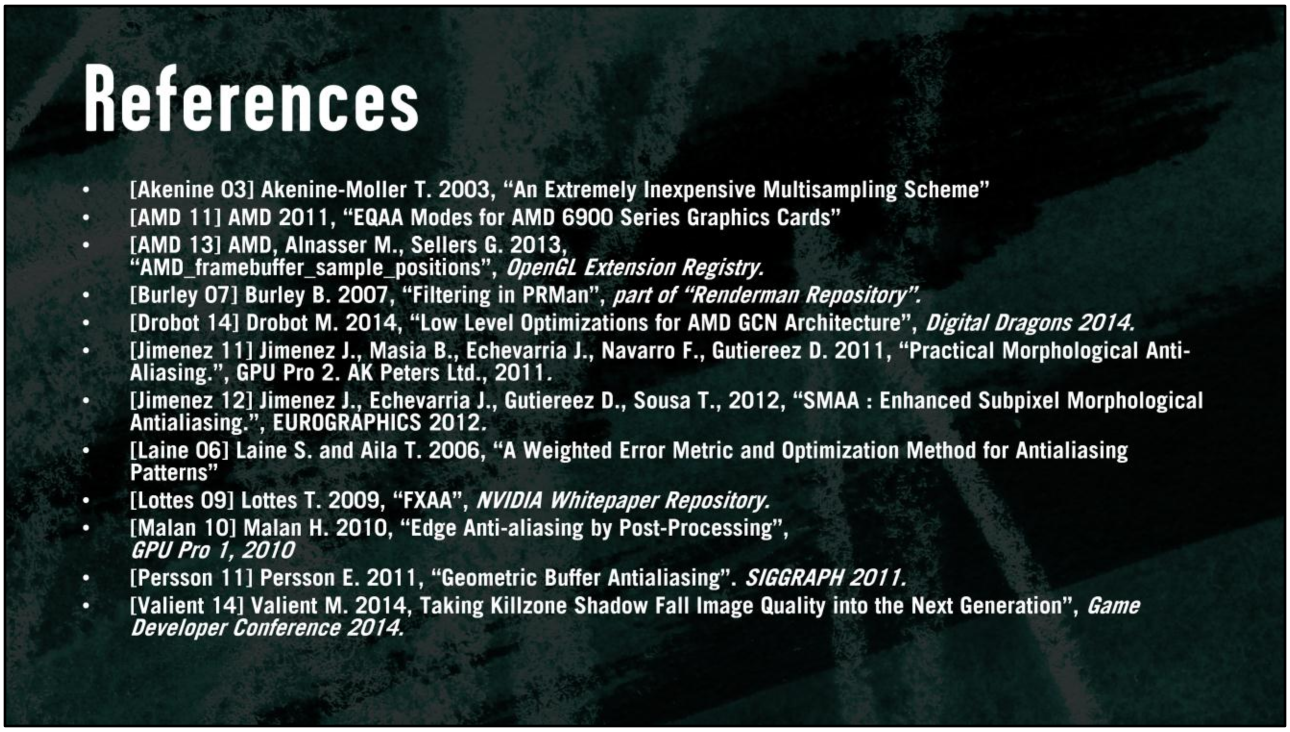
- Temporarily Stable
- Comparable to 4x RGSS
- Cut to fit your needs
- Fast

- Doesn't solve all problems – sub-pixel content still problematic
- Provides some new ideas and solutions to your AA toolbox

- Requires more work in the future

Q&A

More details, samples and pictures
in upcoming GPU Pro 6 article
GO grab it March 2015



References

- [Akenine 03] Akenine-Moller T. 2003, “An Extremely Inexpensive Multisampling Scheme”
- [AMD 11] AMD 2011, “EQAA Modes for AMD 6900 Series Graphics Cards”
- [AMD 13] AMD, Alnasser M., Sellers G. 2013, “AMD_framebuffer_sample_positions”, *OpenGL Extension Registry.*
- [Burley 07] Burley B. 2007, “Filtering in PRMan”, *part of “Renderman Repository”.*
- [Drobot 14] Drobot M. 2014, “Low Level Optimizations for AMD GCN Architecture”, *Digital Dragons 2014.*
- [Jimenez 11] Jimenez J., Masia B., Echevarria J., Navarro F., Gutiereez D. 2011, “Practical Morphological Anti-Aliasing.”, GPU Pro 2. AK Peters Ltd., 2011.
- [Jimenez 12] Jimenez J., Echevarria J., Gutiereez D., Sousa T., 2012, “SMAA : Enhanced Subpixel Morphological Antialiasing.”, EUROGRAPHICS 2012.
- [Laine 06] Laine S. and Aila T. 2006, “A Weighted Error Metric and Optimization Method for Antialiasing Patterns”
- [Lottes 09] Lottes T. 2009, “FXAA”, *NVIDIA Whitepaper Repository.*
- [Malan 10] Malan H. 2010, “Edge Anti-aliasing by Post-Processing”, *GPU Pro 1, 2010*
- [Persson 11] Persson E. 2011, “Geometric Buffer Antialiasing”. *SIGGRAPH 2011.*
- [Valient 14] Valient M. 2014, Taking Killzone Shadow Fall Image Quality into the Next Generation”, *Game Developer Conference 2014.*

Special Thanks

- In Random Order
- Ubisoft 3D Teams:
  - Urlich Haar
  - Jeremy Moore
  - Bartlomiej Wronski
- AMD:
  - Layla Mah
  - Chris Brennan
- Microsoft:
  - David Cook

- MY TURTLE
- BEER