
SIGGRAPH 2025
Vancouver+ 10-14 August
ADAPTIVE VOXEL-BASED ORDER INDEPENDENT TRANSPARENCY
MICHAL DROBOT
ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS
ACTIVISION


SIGGRAPH 2025
Vancouver+ 10-14 August
INTRODUCTION
ADVANCES IN
REAL-TIME RENDERING
IN GAMES
+ CELEBRATING 20 YEARS
ACTIVISION

THE OIT TEAM

ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS

MICHAL DROBOT

- Technology Fellow
- Activision Central Technology

PIOTR BRENDEL

- Expert Rendering Engineer
- Infinity Ward Poland

NICHOLAS JORDAN

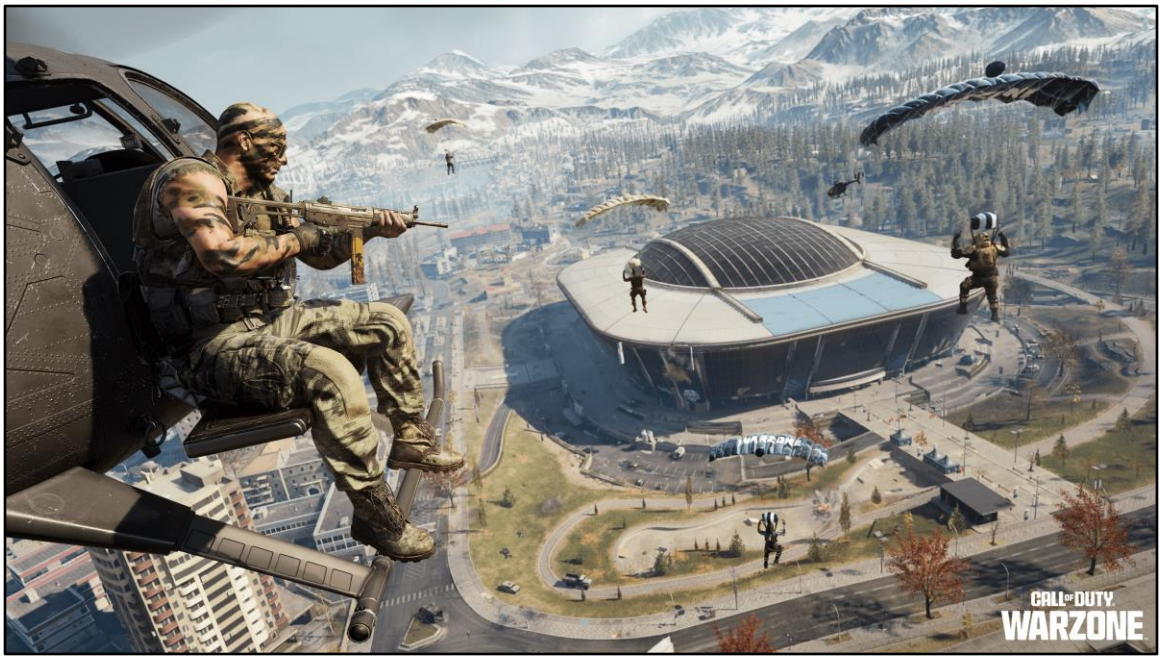
- Rendering Engineer
- Raven Software

ACKNOWLEDGEMENTS

ALL THE PEOPLE WHO HELPED

- Engineering : Piotr Brendel, Nicholas Jordan, Brandon Whitley, Francois Durand, Felipe Gomez
- Art & Tech Art : Patrick Hagar, Gavin Lerner, Rober Kowalchuk, Daniel Stern
- Treyarch : Robert Moffat, Yale Miller, Kevin Myers
- Central Technology : Natasha Tatarchuk, Michael Vance, Paul Edelstein
- Production : David Guo, Neal Nikaido, Christopher Reimschussel

Big shoutout to everyone who helped with this project!



Our games need technology to support vastly different scenarios and environments. This includes large open worlds with huge viewing distances.

Tight, close quarter combat running at highest possible framerate.

Crazy spectacular modes that are full of hundreds of NPCs with absolute visual mayhem on screen mixing explosions with magic.

As well as super high quality in-game cinematics that require highest precision and quality.

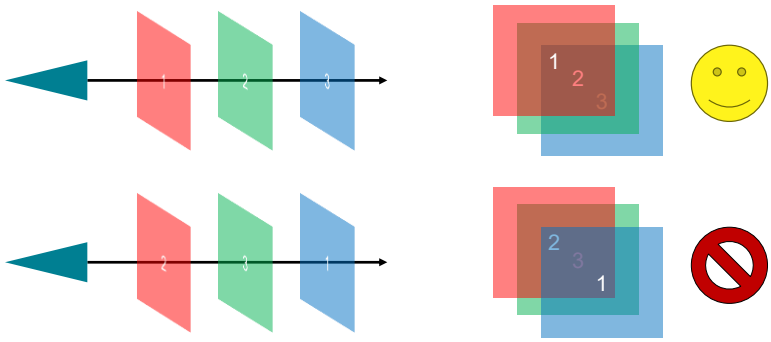
TRANSPARENCY IN GAMES

ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS

USUALLY REFERS TO

- Glass
- VFX meshes
- VFX cards

RENDERING IN RIGHT ORDER



Rendering transparencies is non-trivial.
We usually deal with transparencies when rendering effects and glass in games. Unfortunately drawing those surfaces in any order is not correct, and what is worse, switching order every frame will result in flicker and artifacts as alpha blend is order-dependent.

PAINTERS ALGORITHM

REFERENCE SOLUTION TO N EVENTS

- Render each event in back to front order
- Requires solving composition equation for each pixel

$$C_f = C_1 + (1 - \alpha_1)C_0 \ for\ i = 1$$

- In physical terms, this means solving a composition using the transmittance integral

$$C_f = [C_n + (1 - \alpha_j) \cdots [C_2 + (1 - \alpha_2)[C_1 + (1 - \alpha_1)C_0]] \cdots]$$

- Transmittance Integral at event i

$$\prod_{j=i+1}^{n} (1 - \alpha_j)$$

SOLUTIONS :
BIG FIELD OF RESEARCH [VAS20] [WYM16]

ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS

EXACT SOLUTIONS

- Sort back to front on CPU
  - For non overlapping geometry only
- Capture all events and sort
  - Linked Lists
  - Depth Peel
  - Bounded A / K buffers
    - Fallback to approximate on out of memory
  - Many more

PROS

- Exact

CONS

- High bandwidth
- High complexity
- High memory usage
- Requires synchronization primitives

SOLUTIONS :
BIG FIELD OF RESEARCH [VAS20] [WYM16]

ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS

STOCHASTIC SOLUTIONS

- Rasterize with stochastic dither
  - Based on pixel transparency [ESSL10][WYM17]
- Super sample
  - Spatially [ESSL10]
  - Over time
- Converge to solution
- Often used in games

PROS	CONS
• Cheap	• Situational

WEIGHTED BLENDING

APPROXIMATE SOLUTION TO N EVENTS

- Rewrites the equation to weighted components
- Normalize sum
- Converge to reference
  - Weight -> Integrated Transmittance up to Event

$$C_f = \sum_{i=1}^{n} C_i \cdot w(z_i, \alpha_i) \cdot \frac{1 - \prod_{i=1}^{n}(1-\alpha_i)}{\sum_{i=1}^{n} \alpha_i \cdot w(z_i, \alpha_i)} + C_0 \prod_{i=1}^{n}(1-\alpha_i)$$

Event Weight | Normalization Factor | Total Transmittance Integral

Event Weight — Approximates $\prod_{j=i+1}^{n}(1-\alpha_j)$

WEIGHTED BLENDING : TRANSPARENCY RENDERING SETUP

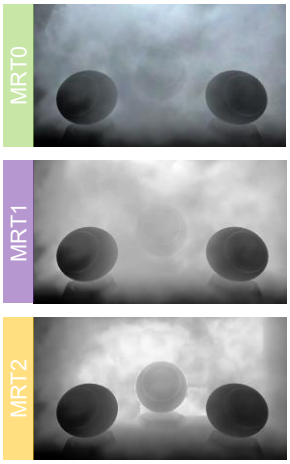
$$C_f = \sum_{i=1}^{n} C_i \cdot w(z_i, \alpha_i) \cdot \frac{1 - \prod_{i=1}^{n}(1-\alpha_i)}{\sum_{i=1}^{n} \alpha_i \cdot w(z_i, \alpha_i)} + C_0 \prod_{i=1}^{n}(1-\alpha_i)$$

Move from alpha to extinction matching volumetric rendering notation

$$\prod_{i=1}^{n}(1-\alpha_i) = \exp\left(-\sum_{i=1}^{n} -\log(1-\alpha_i)\right)$$

Draw all transparencies with following outputs

RT	Function	Format	Op
MRT0 RGB	$C_i \cdot w(z_i, \alpha^i)$	11R11G10B Float	Add
MRT1 R(GB)	$\alpha_i \cdot w(z_i, \alpha^i)$	11R11G10B Float	Add
MRT2 R(GB)	$-\log(1-\alpha_i)$	11R11G10B Float	Add



WEIGHTED BLENDING : TRANSPARENCY RESOLVE SETUP

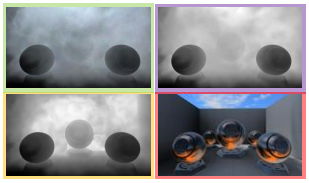
$$C_f = \sum_{i=1}^{n} C_i \cdot w(z_i, \alpha_i) \cdot \frac{1 - \prod_{i=1}^{n}(1-\alpha_i)}{\sum_{i=1}^{n} \alpha_i \cdot w(z_i, \alpha_i)} + C_0 \prod_{i=1}^{n}(1-\alpha_i)$$

$$\prod_{i=1}^{n}(1-\alpha_i) = \exp\left(-\sum_{i=1}^{n} -\log(1-\alpha_i)\right)$$

Full screen resolve pass

Inputs	Function	Op
Background	C_0	Replace
MRT0 RGB	$C_i \cdot w(z_i, \alpha^i)$	Add
MRT1 R(GB)	$\alpha_i \cdot w(z_i, \alpha^i)$	Add
MRT2 R(GB)	$-\log(1-\alpha_i)$	Add

$$C_f = MRT0 \cdot \frac{1 - \exp(-MRT2)}{MRT1} + Background \cdot \exp(-MRT2)$$



Resolve

SOLUTIONS :
BIG FIELD OF RESEARCH [VAS20] [WYM16]

ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS

APPROXIMATE BLEND SOLUTION

- Change “over blend” operator with order independent operator
  - Weighted Sum [Mes07]
  - Weighted Average [BM08]
  - Weighted Blended OIT [MB13]
    - Most common in games
    - Improvements in [KIR18]
- Use ad hoc arbitrary weights
  - Depth / luma based

PROS

- Low complexity
- Low bandwidth
- Low memory usage

CONS

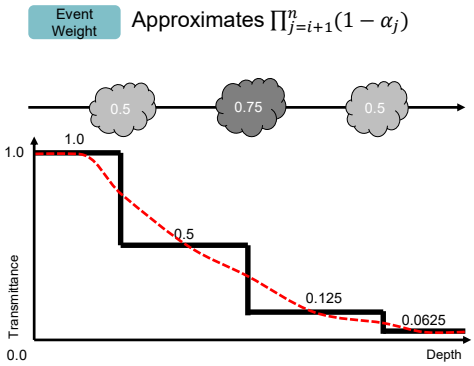
- Frequency soup
- Hard to tune
- Hard to represent occlusion

Approximate blend solution often suffer from a lot of ambiguous tuning values, making it hard to replicate results across content.

APPROXIMATE TRANSMITTANCE INTEGRAL

CONSTRUCT FUNCTION

- Approximate transmittance over depth
  - Exponential
  - Statistical
  - DCT
  - Piecewise
- Project all transmission events to generate function coefficients
  - Render all transparencies in prepass
  - Can be lower resolution



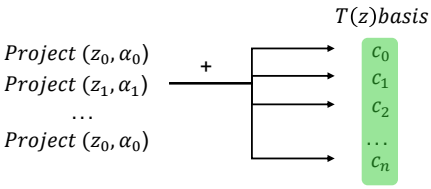
PROJECT TRANSMITTANCE ON FUNCTION BASIS

ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS

$$C_f = \sum_{i=1}^{n} C_i \cdot w(z_i, \alpha_i) \cdot \frac{1 - \prod_{i=1}^{n}(1-\alpha_i)}{\sum_{i=1}^{n} \alpha_i \cdot w(z_i, \alpha_i)} + C_0 \prod_{i=1}^{n}(1-\alpha_i)$$

Projection

Accumulate event transmittance projections on function basis



Coefficient stored pixel
Can be lower resolution

Reconstruction

Evaluate function at Z with accumulated coefficients

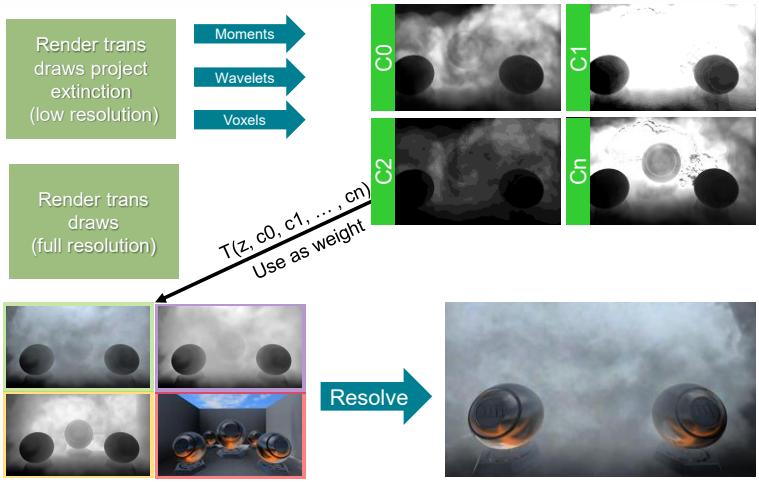
$$w\ (z_i, \alpha_i) = T(z, c_0, c_1, \ldots, c_n)$$

Approximates $\prod_{j=i+1}^{n}(1-\alpha_j)$

TRANSMITTANCE INTEGRAL WEIGHTED OIT

TRANSMITTANCE INTEGRAL

- Split composing into 2 passes
  - Transmittance integral calculation
  - Trans rendering with pre-integrated transmittance in any order
- Approximate transmittance integral
  - Transmittance Function Mapping [DEL11]
  - Moment Based OIT [PMWK17]
  - Wavelet Based OIT [ASM21]
  - Voxel Based OIT



TRANSMITTANCE INTEGRAL WEIGHTED OIT

ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS

PROS

- Medium complexity
- Low to Medium bandwidth
- Low to Medium memory usage
- Allows splitting event rendering resolution
  - from transmittance resolution

CONS

- Approximate
- Different tradeoffs
  - smoothness and correctness over depth
- Occlusion can be tricky


SIGGRAPH 2025
Vancouver+ 10-14 August
OIT JOURNEY
IN
CALL OF DUTY
9408
POLICE
ADVANCES IN
REAL-TIME RENDERING
IN GAMES
+ CELEBRATING 20 YEARS
ACTIVISION

CALL OF DUTY TRANSPARENCY RENDERING

ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS

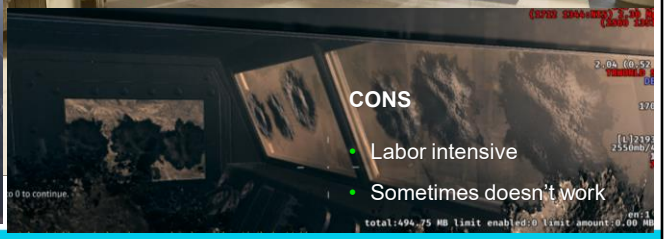
IW & CURRENT TECHNOLOGY

- Manual sort orders
  - Emitters
  - Materials
  - Meshes
- Sort on CPU
- Allow inter-emitter sorting
  - CPU
  - GPU
- Support Blend Add RGB mode
  - Hardware dual blend

PROS

- Fast
- No memory overhead
  - 8-bit accumulated alpha buffer for culling of flares

MANUAL / CPU SORT FAIL CASES



CONS

- Labor intensive
- Sometimes doesn't work

CALL OF DUTY TRANSPARENCY RENDERING

ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS

TA / BLACK OPS 4 [KOH16]

- Exact sorting
  - 9-layer A buffer
    - Meshes raster with UAVs
    - Blend into one layer on fallback
  - VFX rasterizes in software CS resolve
    - sort and resolve with A Buffer
    - Generate Adaptive Transparency OIT structure for other media composition [SAL11]
- No support for explicit sort orders

Layer Data	Monochrome	RGB
9 x Color	11R11G10B Float	11R11G10BF
9 x Transmittance	8UNORM	11R11G10BF
9 x Depth	16F	16F
1 x Sync Counter	8UINT |4x in U32	8UNT |4x in U32
Sum bits per pixel	**512**	**728**
Memory @4k	**~530MB**	**~754MB**

Bandwidth per pixel per event	Write	Read
Monochrome	88bits	88-512
RGB	112bits	112-728

CALL OF DUTY TRANSPARENCY RENDERING

ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS

TA / BLACK OPS 4 [KOH16]

- Pros
  - Shipped
  - Overflows not apparent <w/ that content>
- Cons
  - Heavy performance hit >2.5x vs non-OIT (current)
  - SW raster path requires a lot of maintenance
  - More modern VFX use less camera-oriented cards
    - Forcing A-Buffer overflows more often
    - Higher cost
  - No simple support for multi-resolution
  - 4K / RGB – memory / performance prohibitive

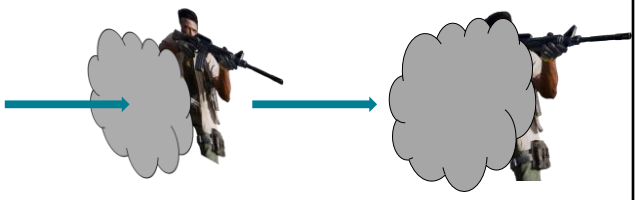
NEW SOLUTION

IMPORTANT

- Occlusion
  - Smoke grenades and visibility is an important gameplay feature
- Polychromic Transparency
  - *Call of Duty* games use plenty colored glass
- Robustness in long distance views
- Memory & Performance

WHAT OTHER GAMES USE

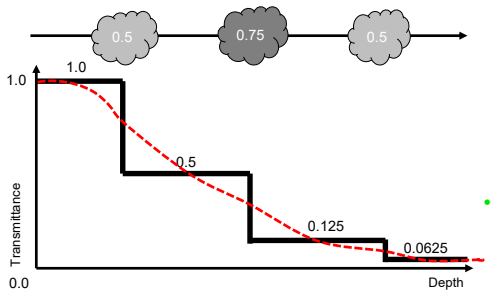
- Weight Blended OIT
- Moment-Based OIT
- Bound A-Buffer



MOMENT-BASED OIT [PMWK17][SHA18]

$T(z)$ Event Weight Approximates $\prod_{j=i+1}^{n}(1-\alpha_j)$

With **power moments** function
Or **trigonometric moments**



Prepass renders P channel MRT:

$z^1 \cdot w$
$z^2 \cdot w$
...
$z^p \cdot w$

Sampling reconstructs transmittance integral:

$$T(z) = \exp(-Hamburger(\frac{b}{d'}, z) \cdot d')$$

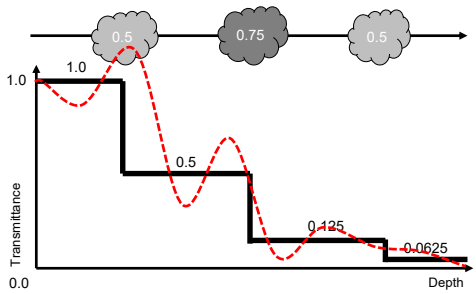
$$b = \sum_{i=1}^{n}(z_i, z_i^2, \ldots, z_i^p) \cdot w_i$$

- Potential issues
  - Hard to represent step function
    - **Thin opaque features smoothed out**
  - All projections change all coefficients
    - **Not depth complexity invariant**

WAVELET-BASED OIT [ASM21]

$T(z)$ Event Weight Approximates $\prod_{j=i+1}^{n}(1-\alpha_j)$

With **Haar Wavelets**
or other wavelet functions



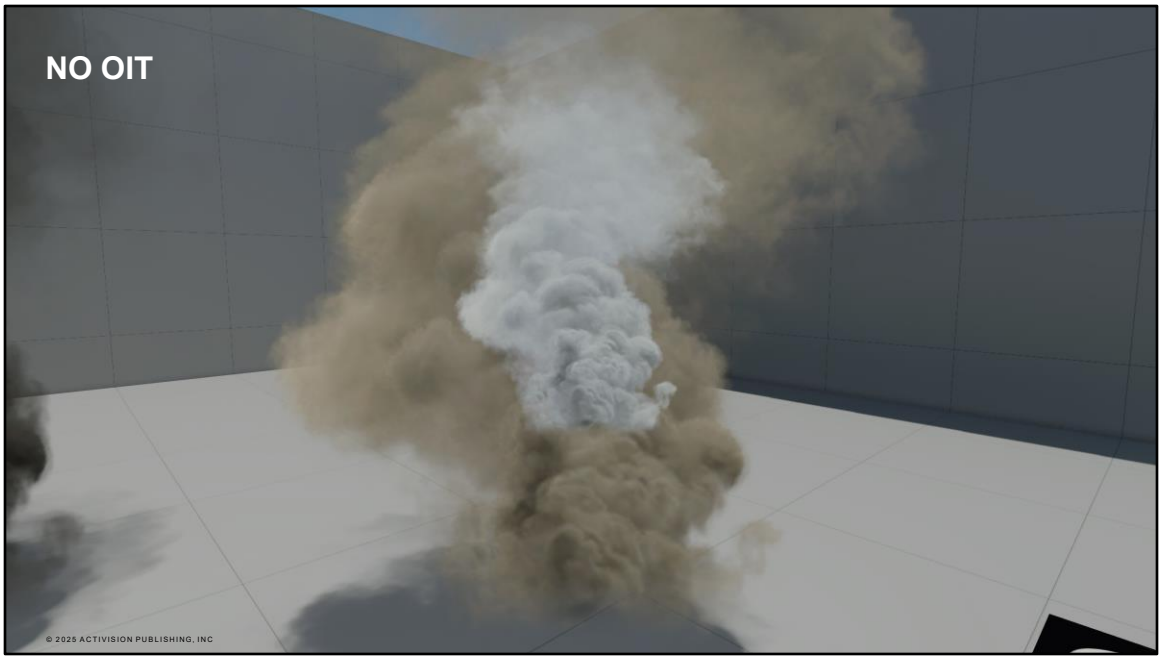
Prepass renders (n+1) channels to 2^n channel MRT:

$$\Psi_{n,k}(x) = 2^{-\frac{n}{2}} \cdot \begin{cases} 2^n x - k & 0 \le 2^n x - k < 0.5 \\ 1 + k - 2^n x & 0.5 \le 2^n x - k \le 1 \\ 0 & \text{Else} \end{cases}$$

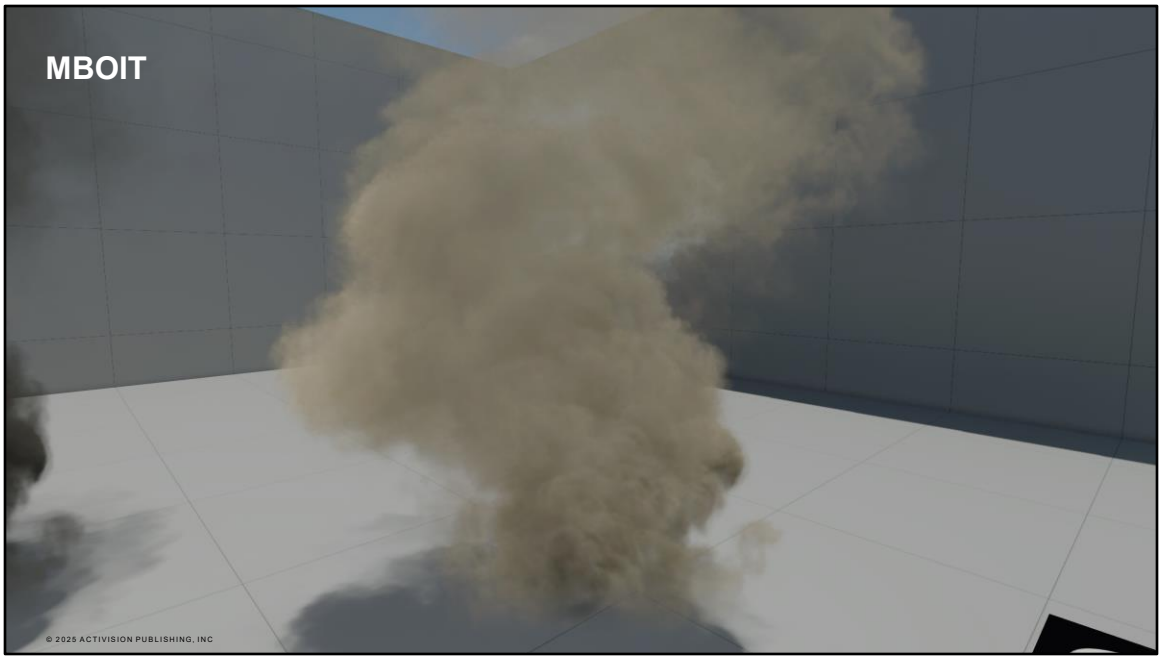
Sampling reconstructs transmittance integral:

$$-\sum \alpha_i \Psi_{n,k}(x_i)$$

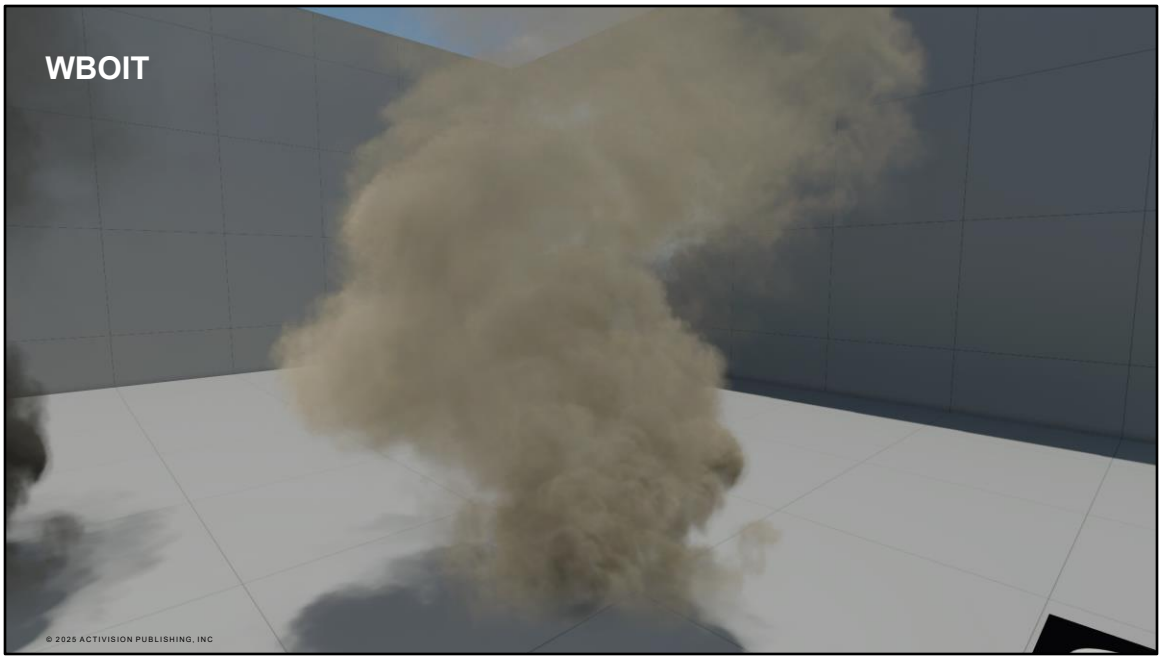
- Potential issues
  - Thin opaque feature projection
    - Ringing
    - Translation variance
      - **Not shift invariant**



First let’s compare no OIT – including an obvious fail case where we sort my particle emitter, yet actual sprites interleave resulting in visual artifacts.



MBOIT correctly resolves order dependencies resulting in stable good looking image.

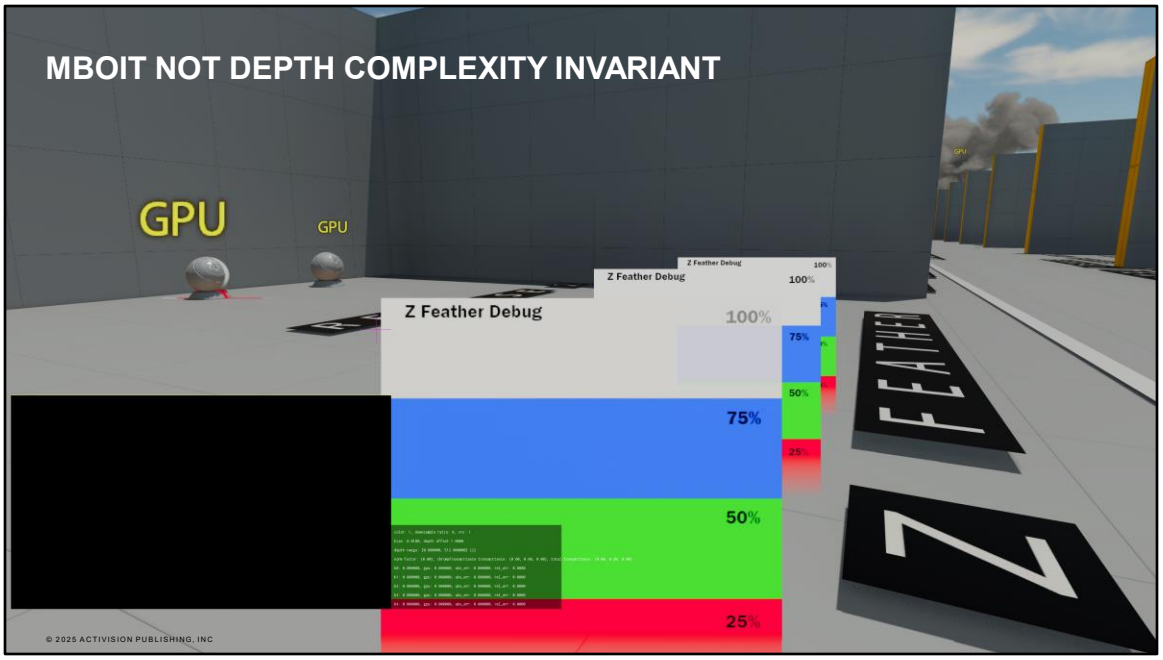


Similar with WBOIT.
So far so good.
Let’s try something more complex closer to in-game scenario.

Here we can observe no OIT – standard ordered version that exhibits some minor issues due to mixing z-feathered effects and distortion effects with imperfect back to front sorting (this is rather unusual setup for in-game).
One character silhouette shows a bit through smoke grenade (which is far from ideal because the main goal of smoke grenade is to obscure what is behind it).

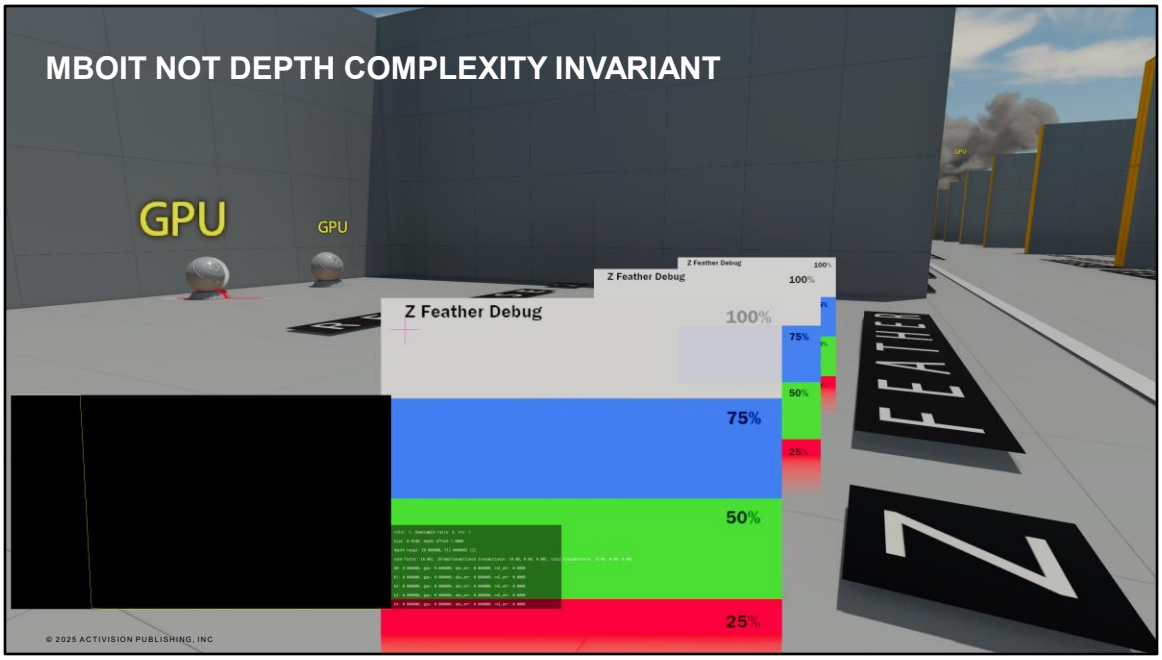
Switching to MBOIT generates even worse result when it comes to silhouettes. Suddenly we can see yet another character that was previously hiding in smoke and this time it is a perfectly legitimate opaque geometry character that should be 100% occluded.
Furthermore, we can also easily observe up sampling artifacts from MBOIT (as the function coefficients don’t behave well under linear filter at depth discontinuities which result in function discontinuities).

WBOIT exhibits same issues as MBOIT – with visibly better silhouette resolution preservation that is mostly due to better wavelet behavior under linear magnification (which stems from implicit depth partitioning by wavelet).

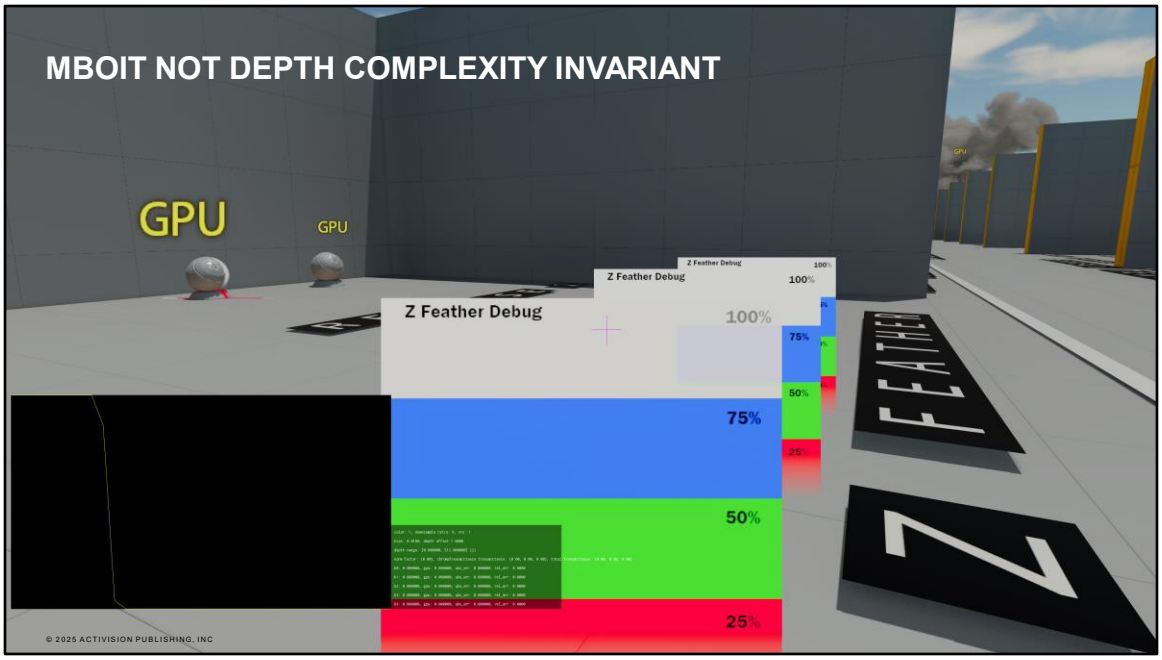


Let’s investigate this odd behavior.
First, this is out resting harness. Center left of screen you can see a magenta crosshair that is pointing at opaque background right next to top left corner of front most card.
Bottom left of screen, black rectangle draws the transmittance function, using chosen reconstruction, at crosshair.
Currently not drawing anything because we are pointing at opaque background.

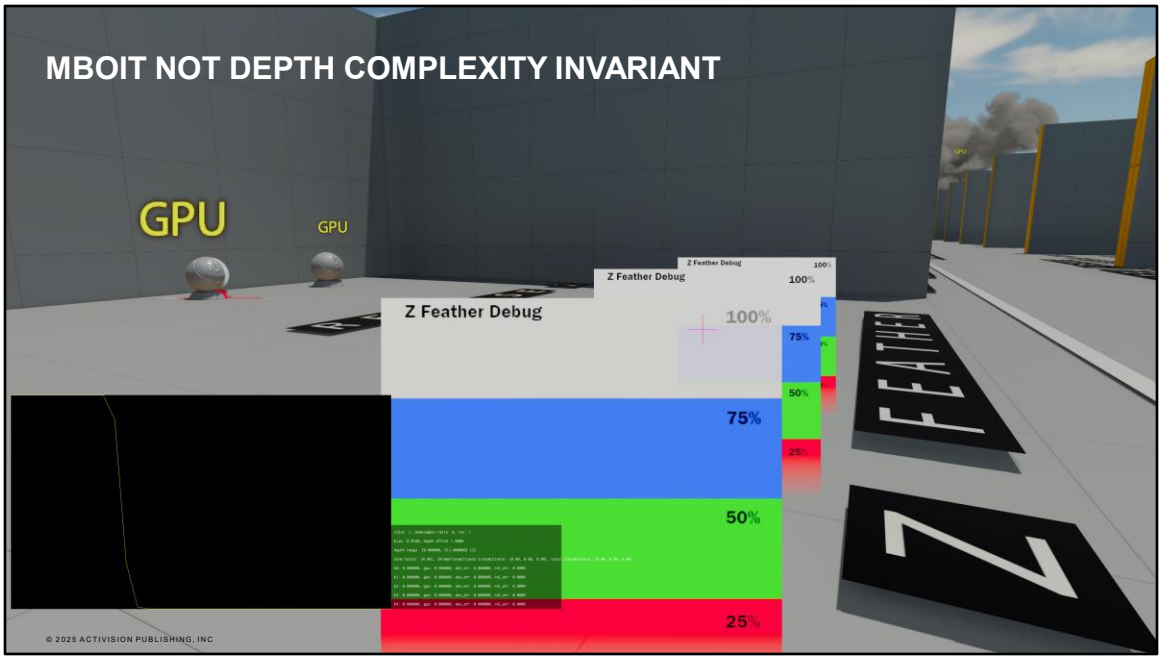
There are 3 cards drawn as transparent VFX with 0% transmittance. They are draws as opaque objects would, but with OIT, where we would expect 100% occlusion (as in smoke grenade case).



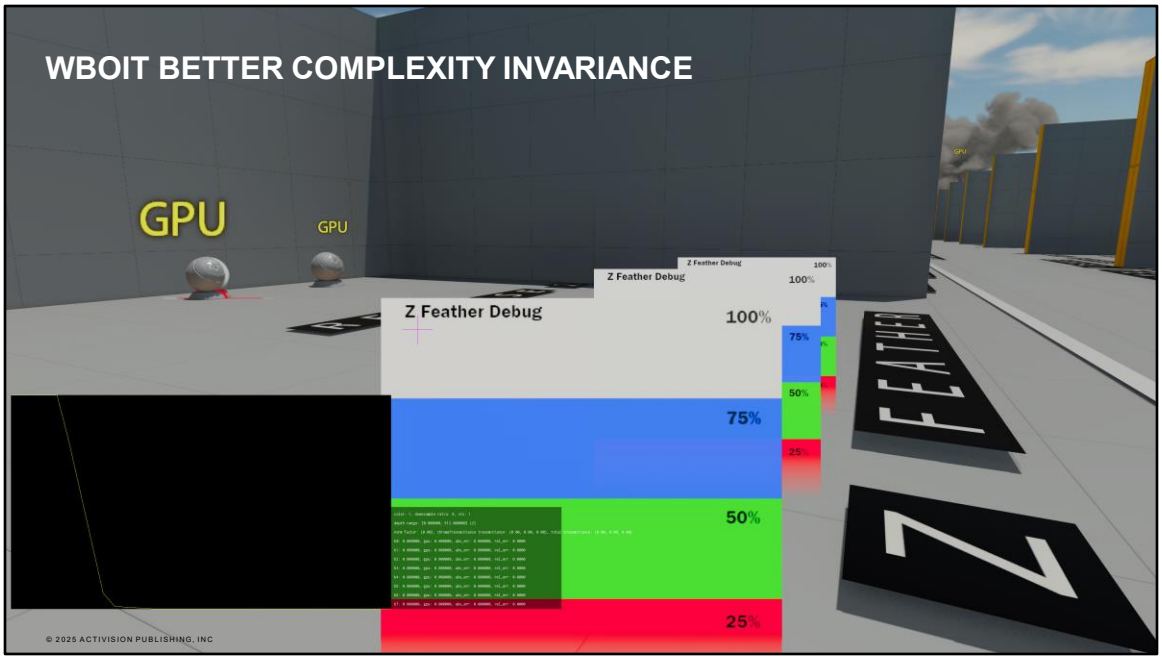
Moving right, we hit first single surface, overlapping the background. Function is steep enough (although ideally this would be a vertical line considering we are working with infinitely thin sprites / events).



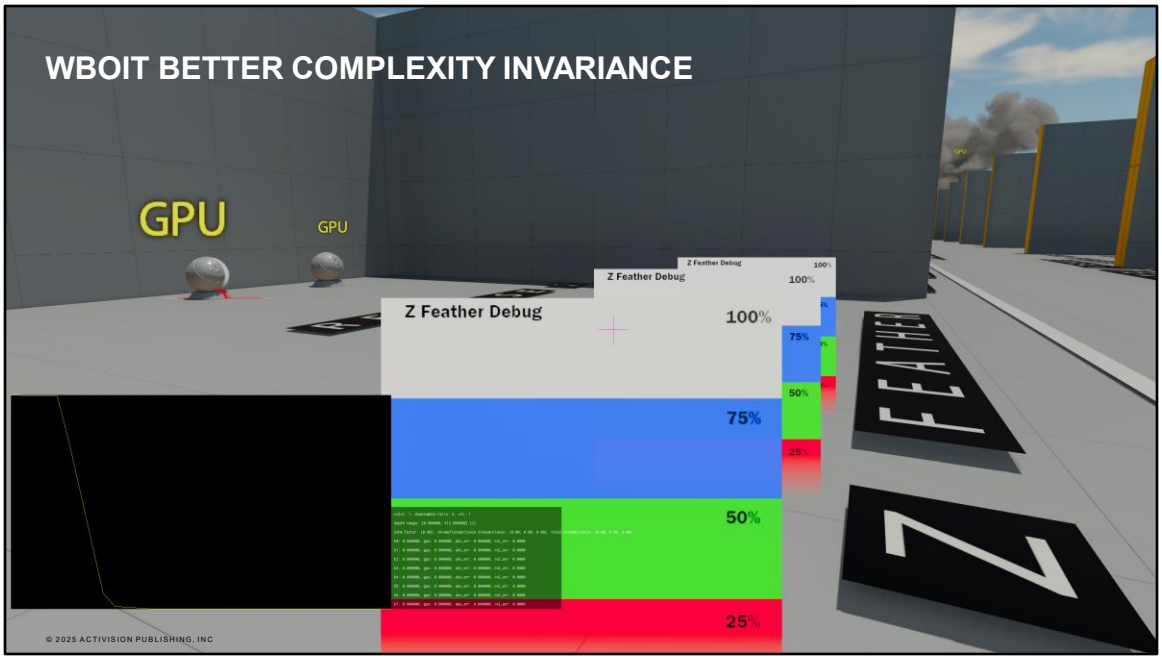
Moving further right, we hit a pixel that has two transparent sprites overlapping. At this point we start seeing the problem where the previously steep line is starting to skew losing its slope with further smoothing around toe and shoulder.
It also becomes offset in depth but not enough to cause trouble.



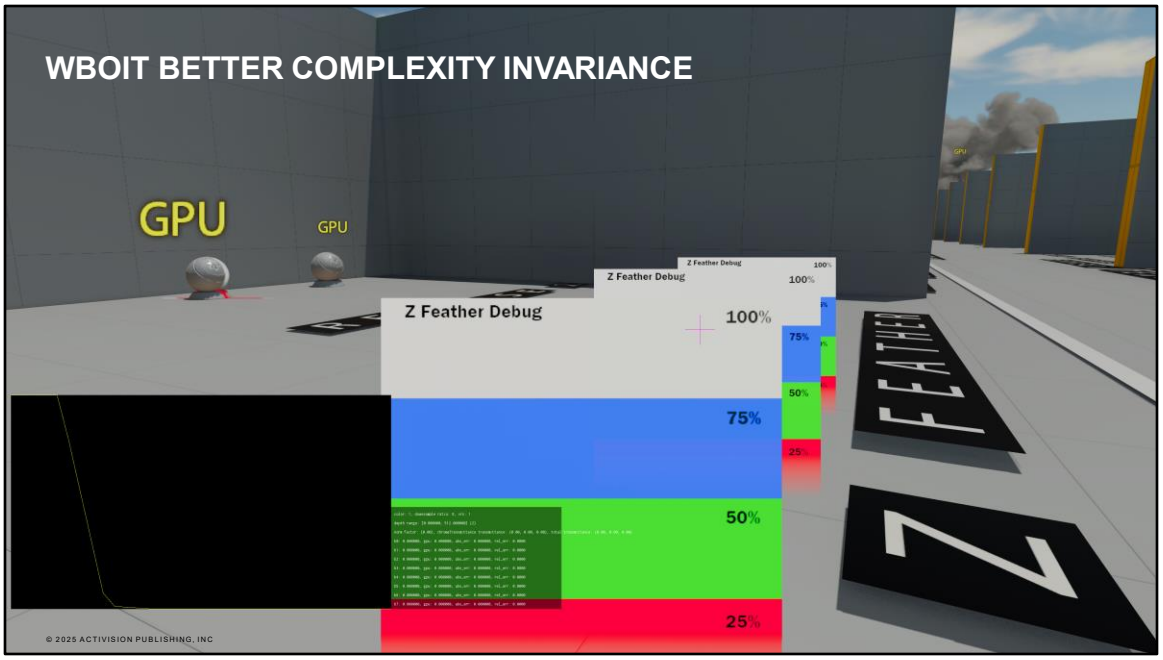
Moving pointer further right we see that pixel overlapping 3 opaque cards shows very significant transmittance function skew which is just too much for rather small depth difference between cards – which would explain why there is some blue piecing through from 2nd sprite.



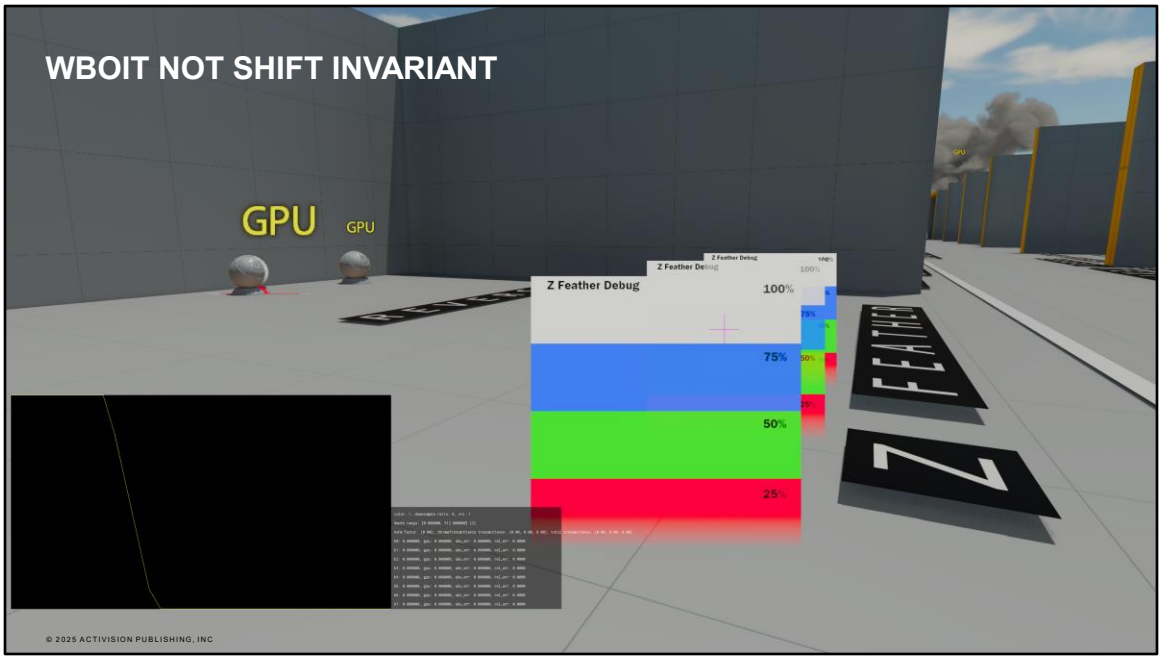
Switching to WBOIT we can immediately see that blue is not piercing through any card.
Actually things look pretty good. However there is some odd sloping visible on reconstructed transmittance graph.



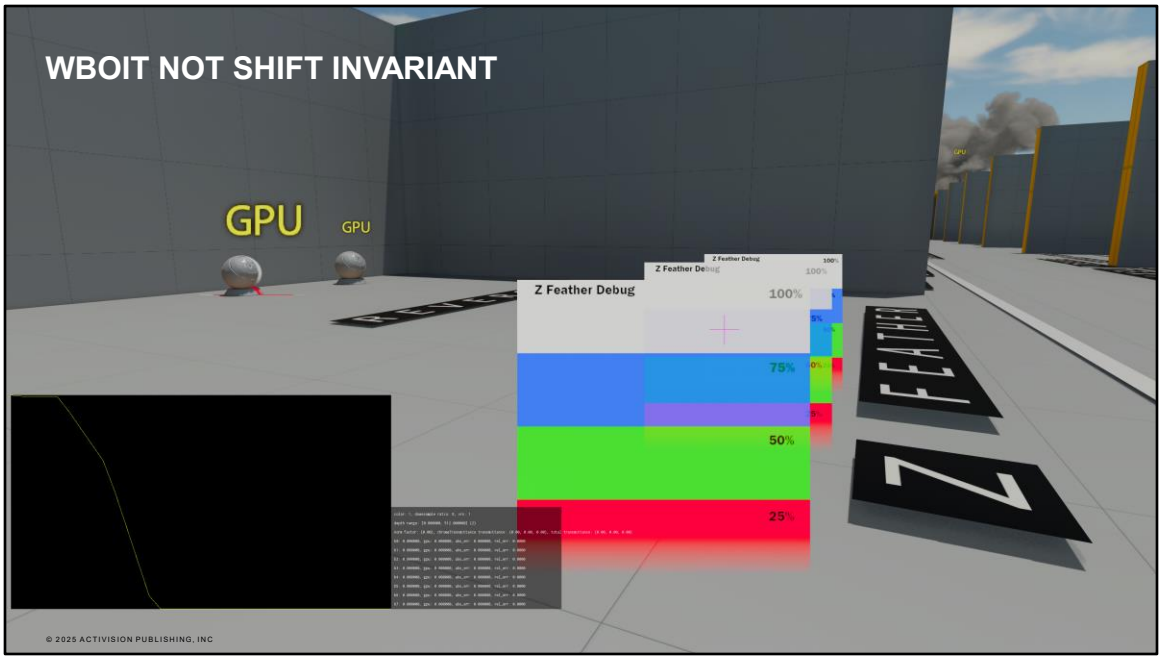
Moving further we see no change to transmittance curve. So far so good.



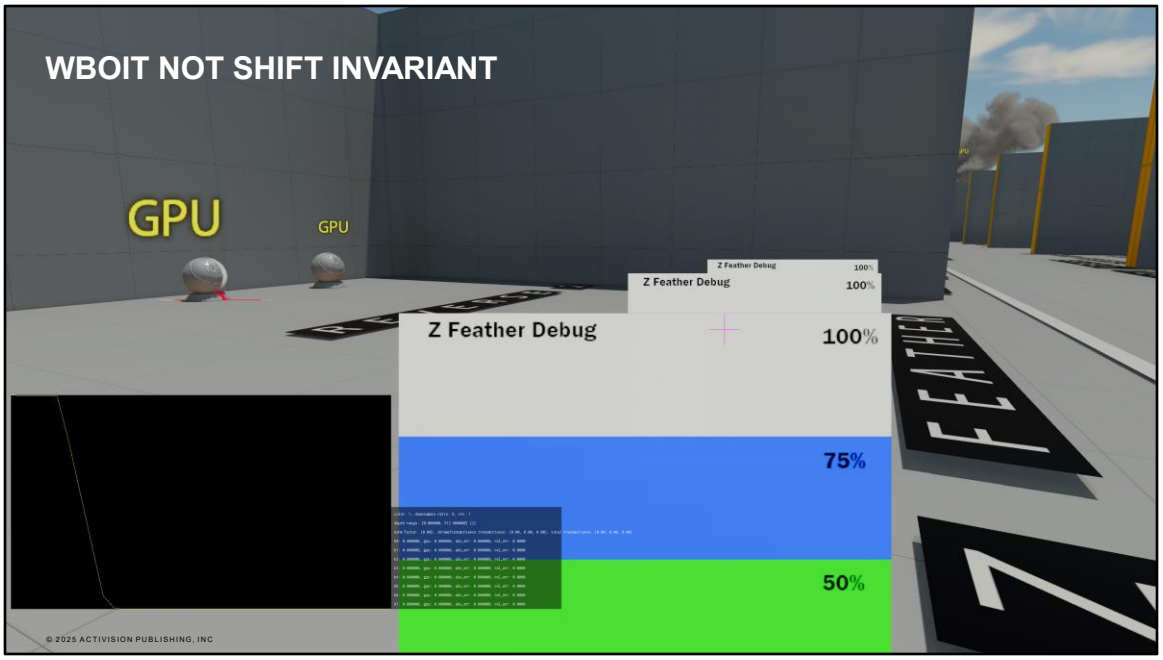
And same with final event.
This can be easily explained by thinking about wavelets as implicit separation over depth. Each wavelet frequency hits certain ‘depth ranges’ – and in this case we seem to be lucky enough to have good separation.
But let's see what happens if we start moving between those ranges.



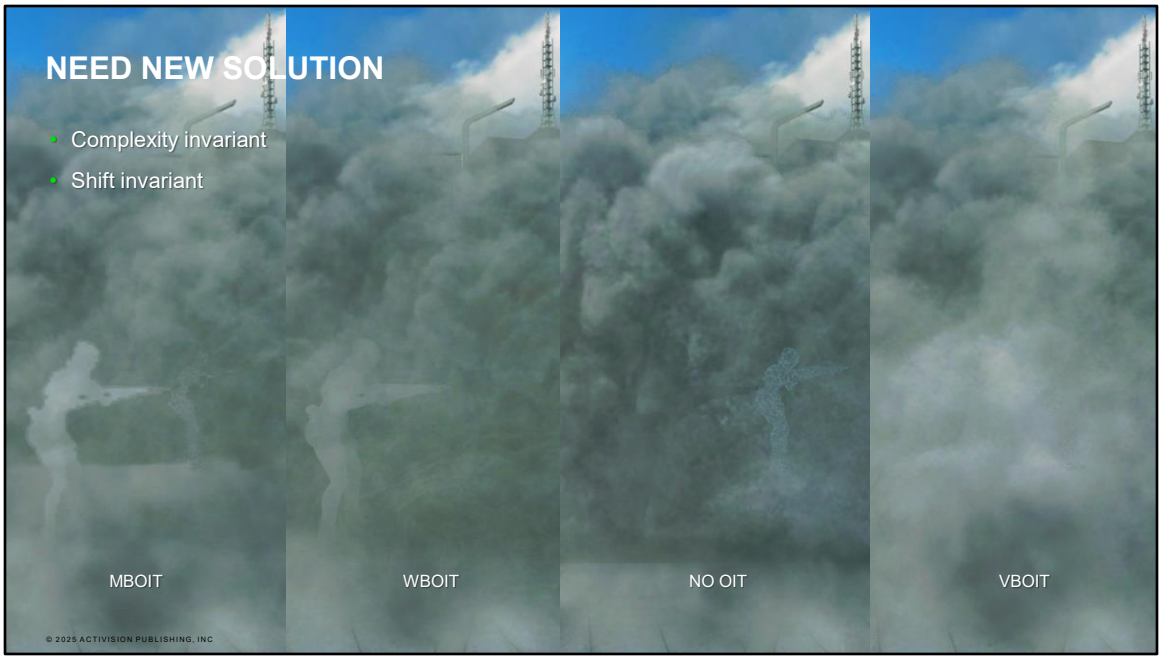
Moving the camera back and pointing at same overlap type (of 3 opaques) shows that reconstructed transmittance significantly shifted. There is not much slope change, but some smoothing at toe and shoulder is visible.
Also colors start mixing between sprites.



Moving the camera slightly in results in dramatic slope and offset shift – that in motion looks very much like ringing. Meaning we just hit a depth region between wavelet frequencies. This unfortunately results in significant visual oscillation of occlusion between sprites.



And then once we are past problematic depth range (frequency peaks) it all goes back to normal.
In the end this is an unreliable behavior.
There are other methods available to reduce ringing in frequency domain, but all of them trade signal smoothness and clipping for ringing prevention – thus with existing experience we decided not to pursue.



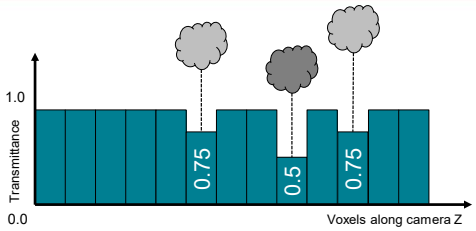
And finally, comparison of problematic are of smoke grenade across all methods. Last panel present our new proposed method using Voxels.

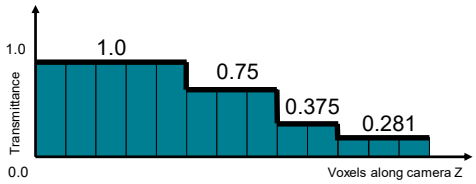
Before we get to final form of our OIT solution – Adaptive Voxel-Based OIT – we would like to present basic method version as building blocks necessary for fully featured solution.

VOXEL-BASED OIT

TRANSMITTANCE IN VOLUMETRIC RENDERING

- Solves same problem by voxelizing air density into froxels
- Compute job integrates transmittance over eye rays
- Generates a 3D LUT of transmittance integral from eye to point in question
- Render transparencies
  - sample 3D LUT for transmittance integral
  - linear or higher order sampling
- Similar framework used in volumetric fog [WRO14][DRO17]
- Deep Shadow Maps [LOK00][NGU05][SIN08][BAV13]





VBOIT : BASIC STEPS

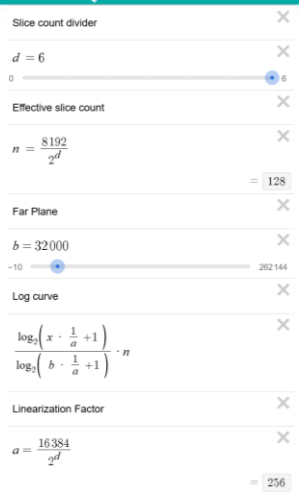
ALGORITHM

- Fit curve distribution for slices over depth
- Clear volumetric extinction buffer
- Splat transparency into volumetric extinction buffer at lower resolution
- Sum extinction along view rays into 3D extinction integral texture
- Draw opaque geometry
- Render transparency in any order
  - Lookup extinction integral at point of surface
  - Multiply surface opacity by transmittance integral
- Resolve opaque background with normalized transparency accumulated color and accumulated extinction

VBOIT : DEPTH DISTRIBUTION

LOG CURVE

- Adjustable
  - Far plane
  - Linearization factor
  - Slice count
- Implicit near plane

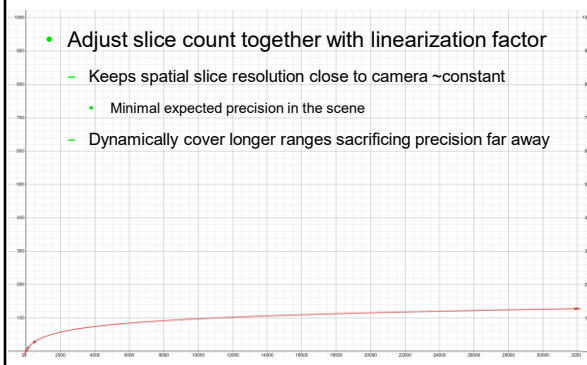


For depth distribution of slices of froxel buffer we use a parametric logarithmic curve that allows us to change precision at near plane depending on how many slices we have at our disposal.

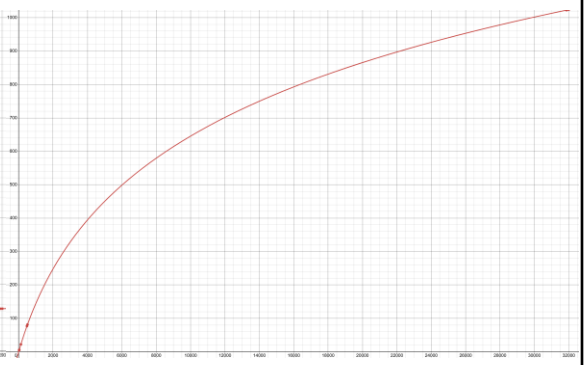
VBOIT : DEPTH DISTRIBUTION

- Adjust slice count together with linearization factor
  - Keeps spatial slice resolution close to camera ~constant
    - Minimal expected precision in the scene
  - Dynamically cover longer ranges sacrificing precision far away

128 Slices



1024 Slices

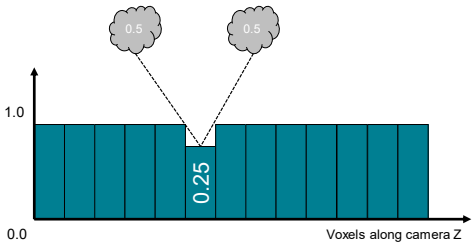


VOXELIZATION WITH RASTERIZATION

VFX & MESHES

- Accumulate transmittance via splatting
- [BAV13] suggests using SV_RenderTargetArrayIndex
  - Support on HW might require GS path
  - Fast path needs VPAndRTArrayIndexFromAnyShaderFeedingRasterizer
  - Only works on per-primitive basis
  - Can only voxelize camera facing quads
- Can we use atomics instead?

1. Clear slices to 1.0
2. VS assign VFX quad to slice
3. PS writes (1.0 – alpha) with MUL BLEND



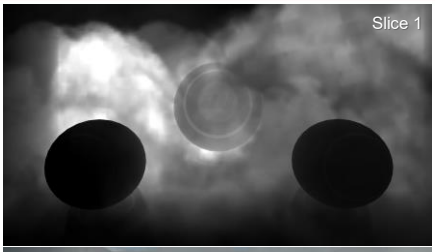
VOXELIZATION WITH ATOMICS

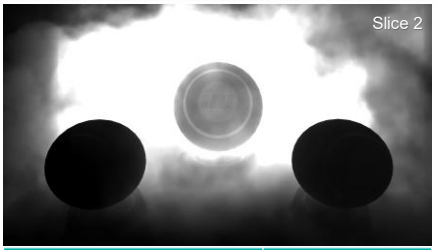
VFX & MESHES

- Atomics mesh voxelization [DRO17.2]
  - Splat from PS
  - Fast if contention is minimized
  - There is no InterlockedAtomicMultiply
- Move from transmittance to **extinction**
  - Log space
  - InterlockedAtomicAdd
- Work with UINT
  - Rescale extinction to normalized range
  - Scale by bit depth

```
float extinction = saturate(-log(1.0 - alpha) /
 -log(1.0 / ((float)VBT_EXTINCTION_BIT_MASK)));
uint depthSlice = depth * (float)VbtGetVoxelSliceCnt();
uint uintExtinction = ((float)VBT_EXTINCTION_BIT_MASK) * extinction + 0.5;
InterlockedAdd( vbt[uint3( coord, depthSlice) ], uintExtinction);
```

EXTINCTION VOXELIZATION
U32 ATOMICS





Splat Type	Time (ms)
MRT MUL	0.75
U32	0.91

Rendering 17mil pixels overdraw @ 1080p PS5

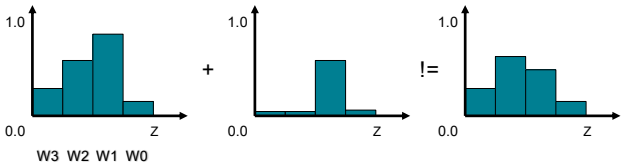
EXTINCTION VOXELIZATION PACKED 8BIT ATOMICS

PACKING INTO 8 BITS

- U32 for extinction accumulation is overkill
  - Memory and performance
- Pack slices into 8bit words of U32

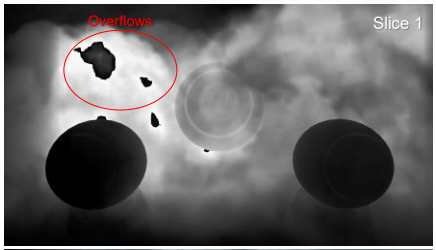
DWORD	Word3	Word2	Word1	Word0
	64	128	192	32
+	0	0	128	0
=		129	64	32
Correct	64	128	255	32

- 2 problems
  - Overflows
  - Carry over between words



EXTINCTION VOXELIZATION PACKED 8BIT ATOMICS

ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS



Splat Type	Time (ms)
MRT MUL	0.75
U32	0.91
U8	0.89

Rendering 17mil pixels overdraw @ 1080p PS5

Overflows result in unwanted function change – that will show as artifacts in final image.

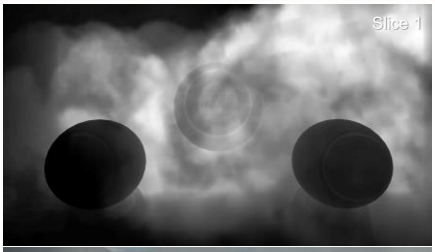
EXTINCTION WITH ATOMICS : OVERFLOW

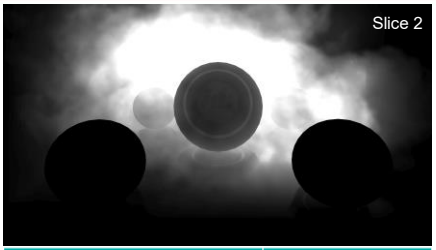
PACKING INTO 8 BITS

- Using InterlockedAtomicCompareExchange
  - Can implement any operation
  - Equivalent to spinlock
- Reference version
- 2x-10x more expensive

```
uint uintExtinctionPacked = pack( extinction, wordIdx );
uint current = 0;
uint expected = 0;
uint newData = uintExtinctionPacked;
while ( true )
{
 AtomicCmpExchange(vbt[uint3( coord, sliceIdx)], expected, newData, current);
 if ( expected == current )
  break;
 newData = pack( saturate( unpack( current, wordIdx) + extinction ), wordIdx );
 expected = current;
}
```

EXTINCTION VOXELIZATION
PACKED 8BIT ATOMICS /W SPINLOCK





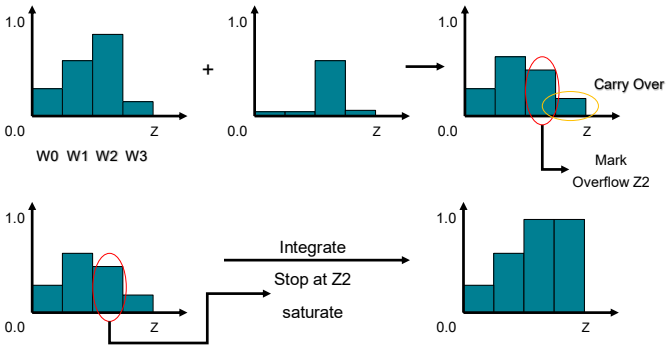
Splat Type	Time (ms)
MRT MUL	0.75
U32	0.91
U8	0.89
U8 Spinlock Overflow	1.95

Rendering 17mil overdraw pixels @ 1080p PS5

EXTINCTION WITH ATOMICS : OVERFLOW

OVERFLOW DETECTION

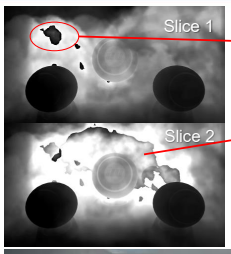
- Order words along integration direction
- Integrating along ray
  - Data past overflow does not matter
  - Don't care about carry over
- Detect overflow
  - On overflow store depth
    - Min over ray with InterlockedMin
- During integration
  - Read overflow min depth
  - Saturate
  - Stop Integrate

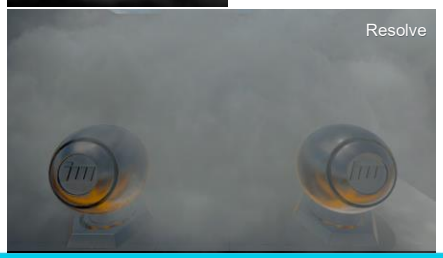



```
InterlockedAdd( vbt[uint3( coord, sliceIdx ) ], uintExtinction, preData );
bool isOverflow = ( unpack( prevData, wordIdx ) + extinction ) > 1.0;
if ( isOverflow )
{
 InterlockedMin( vbtOverflowDepth [uint3( coords, 0 )], sliceIdx );
}
```

EXTINCTION VOXELIZATION

PACKED 8BIT ATOMICS /W OVERFLOW MIN DEPTH DETECTION





Splat Type	Time (ms)
MRT MUL	0.75
U32	0.91
U8	0.89
U8 Spinlock Overflow	1.95
U8 Overflow Min	0.91

Rendering 17mil pixels overdraw @ 1080p PS5

Even though overflows are visible in individual slices, it does not matter for final integration, as we can saturate the results using overflow mind depth buffer data.

COLOR EXTINCTION WITH ATOMICS

SPLATTING COLOR

- Pack RGB into 32 bits as 8|(2)|8|(2)|8|(2)
  - Update with single atomic
  - Same execution speed as packed 8bit monochrome
  - Some loss due to worse caching
    - Less slice sharing -> Less atomic contention
    - First Overflow method does not protect against carry over to next ‘wavelength’
    - Pack 8bit values in 10bit fields
      - Allows 2 bits for overflows and carry over
    - Carry over only touches lowest bits of next ‘wavelength’
      - Not visible in practice unless all your VFX have heavy color skew
  - First Overflow stores first slice for each R G B
    - Faster with 3x InterlockedMin then spinlock updating single U32

0xFF	00	0xFF	00	0xFF	0000
RED	overflow	GREEN	overflow	BLUE	overflow

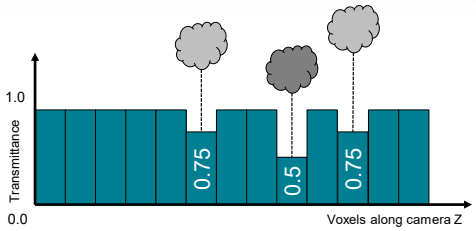
Splat Type	Time (ms)
MRT 8UNORM MUL	0.75
MRT 10R10G10B MUL	0.81
U8 Overflow Min	0.91
10|10|10 Overflow Min	0.95

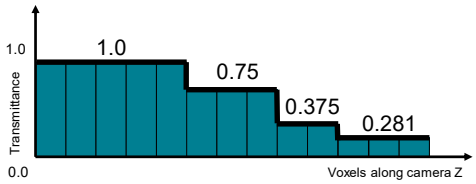
Rendering 17mil pixels overdraw @ 1080p PS5

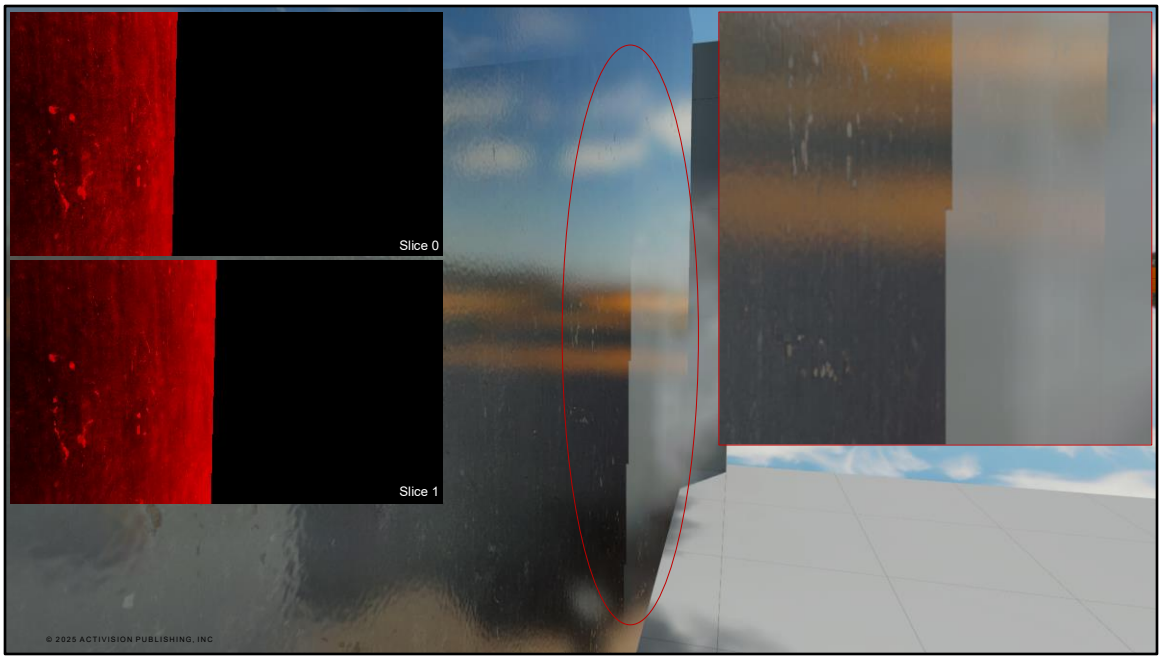
VBOIT : INTEGRATION

INTEGRATION CS JOB

- Generate Integral 3D LUT
  - Iterate over all slices
  - Add extinction
  - Stop ray when extinction integral saturates
    - Or overflow detected





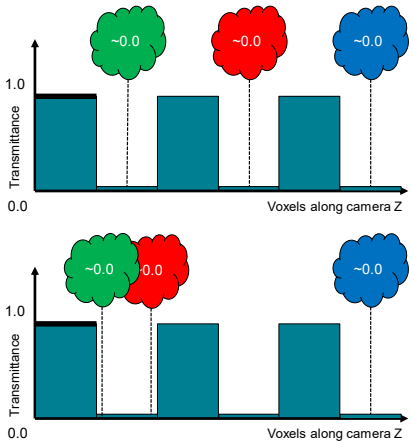


First try – works – but clear aliasing artifacts are visible.

VBOIT : ALIASING AT SPLAT

ALIASING AT SOURCE

- During splatting we discretize into slice resolution
- Changes over splat depth create aliasing

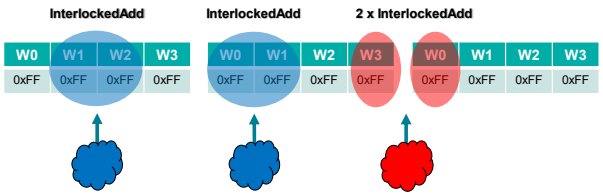


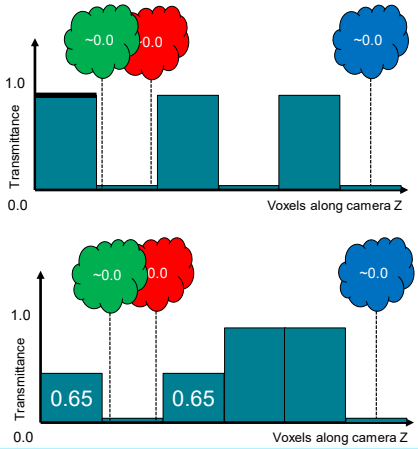
VBOIT : ALIASING AT SPLAT

ADVANCES IN REAL-TIME RENDERING IN GAMES + CELEBRATING 20 YEARS

LINEAR SPLATTING

- Use slice fraction to split partially between slices
- Monochrome
  - Merge splats to slice Z and Z + 1
  - Issue extra splat on straddling DWORDs





During linear splatting, if objects have actual thickness – it would be preferable to calculate exact volume of event intersecting a slice and only splat that amount of extinction. This can significantly improve visual dimensionality of vfx – however that specific improvement is orthogonal to discussed OIT algorithm.

VBOIT : ALIASING AT SPLAT

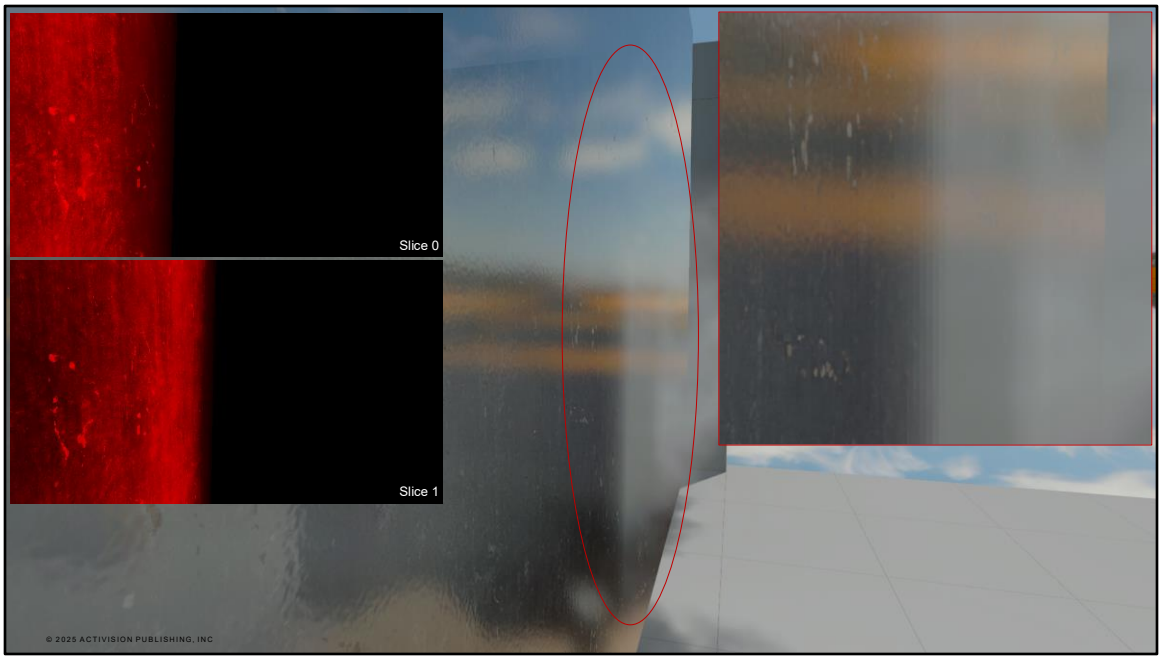
ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS

LINEAR SPLATTING

- RGB
  - Issue 2 splats
- No performance delta
  - Splats are within cache lines
  - They are independent

Splat Method	Time (ms)
Monochrome Point Splat	0.75
Monochrome Linear Splat	0.81
RGB Point Splat	0.91
RGB Linear Splat	0.95

Rendering 17mil pixels overdraw @ 1080p PS5

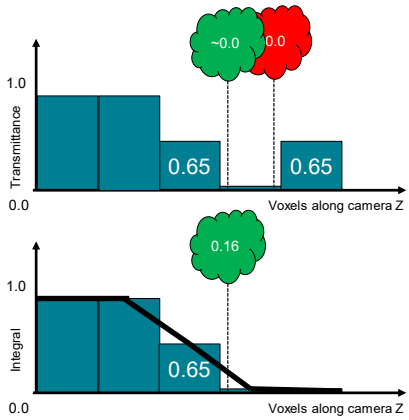

Slice 0
Slice 1
© 2025 ACTIVISION PUBLISHING, INC

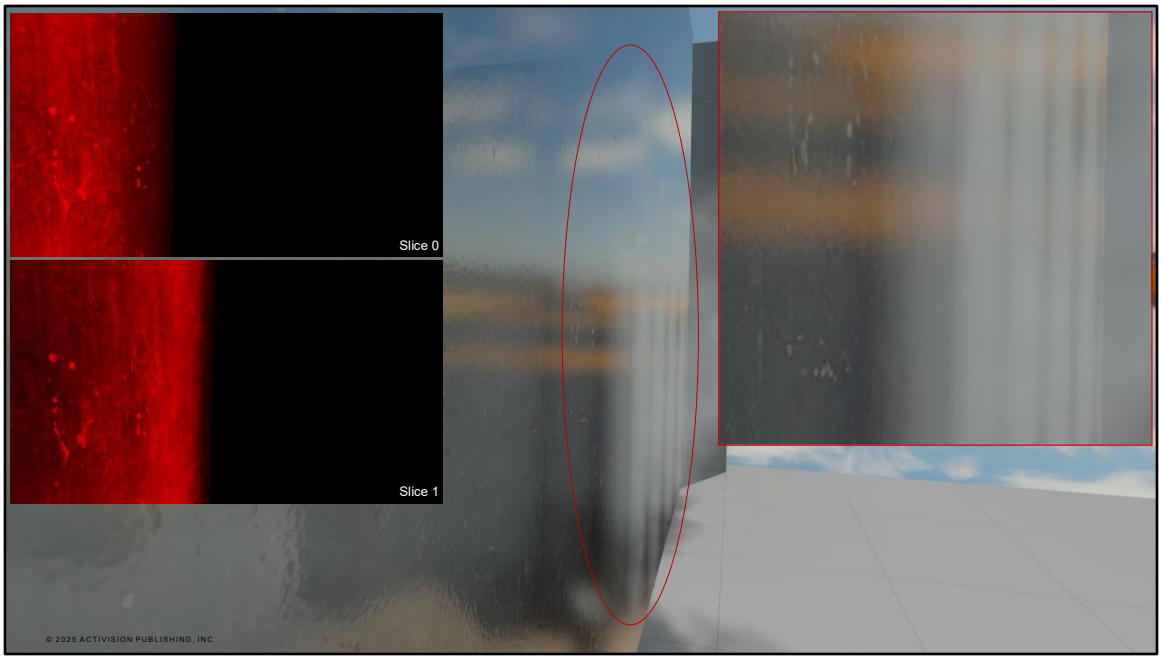
VBOIT : ALIASING AT SAMPLE

ALIASING AT SAMPLING

- Extinction can be linearly interpolated
- Use HW linear
- Sampling at splat point != value during splatting

- Chance for self occlusion?





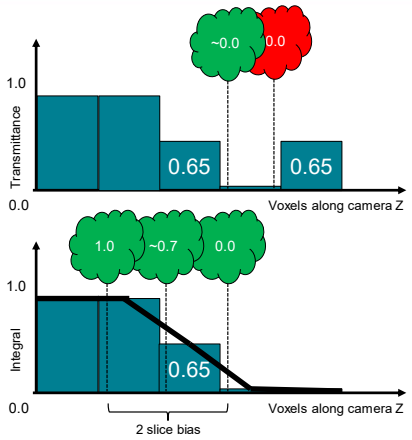
As we can see glass pane is starting intersect itself and self occlude due to imprecision of depth slicing. It is further exacerbated by lower resolution of integral buffer.

VBOIT : ALIASING AT SAMPLE

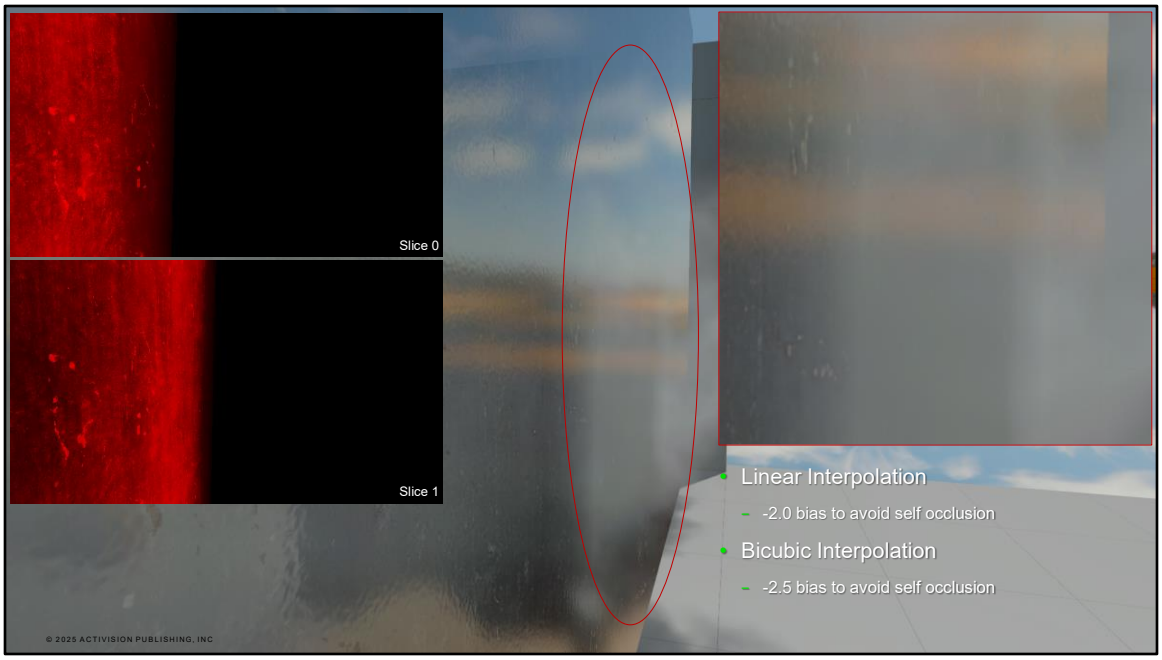
ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS

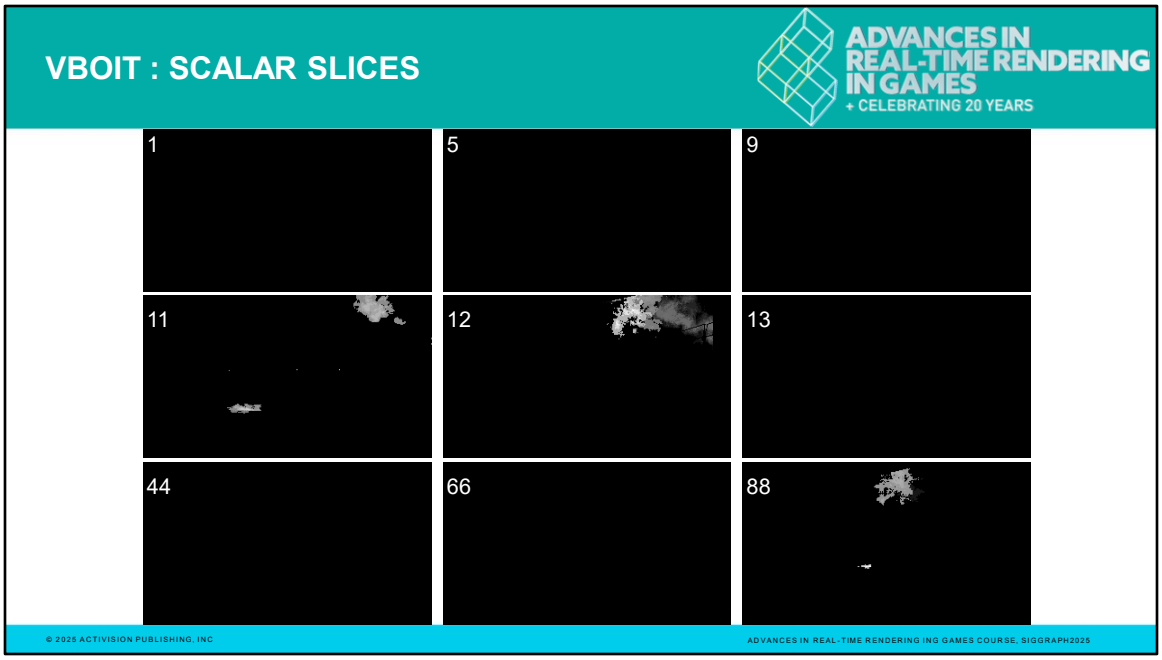
ALIASING AT SAMPLING

- Use HW linear
  - Requires 2 slices bias
- Linear interpolation over a linear splat
  - Not C1 continuous
  - Requires 2nd order sampling
    - Bicubic works well but needs 2.5 slice bias



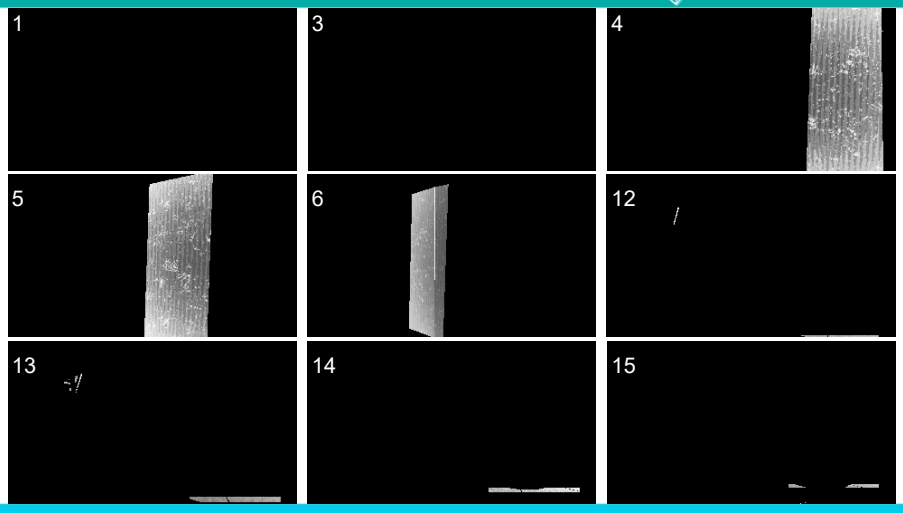
We need to make sure that sampling of extinction is offset in depth enough to avoid self occlusion.


Slice 0
Slice 1
Linear Interpolation
-2.0 bias to avoid self occlusion
Bicubic Interpolation
-2.5 bias to avoid self occlusion
© 2025 ACTIVISION PUBLISHING, INC

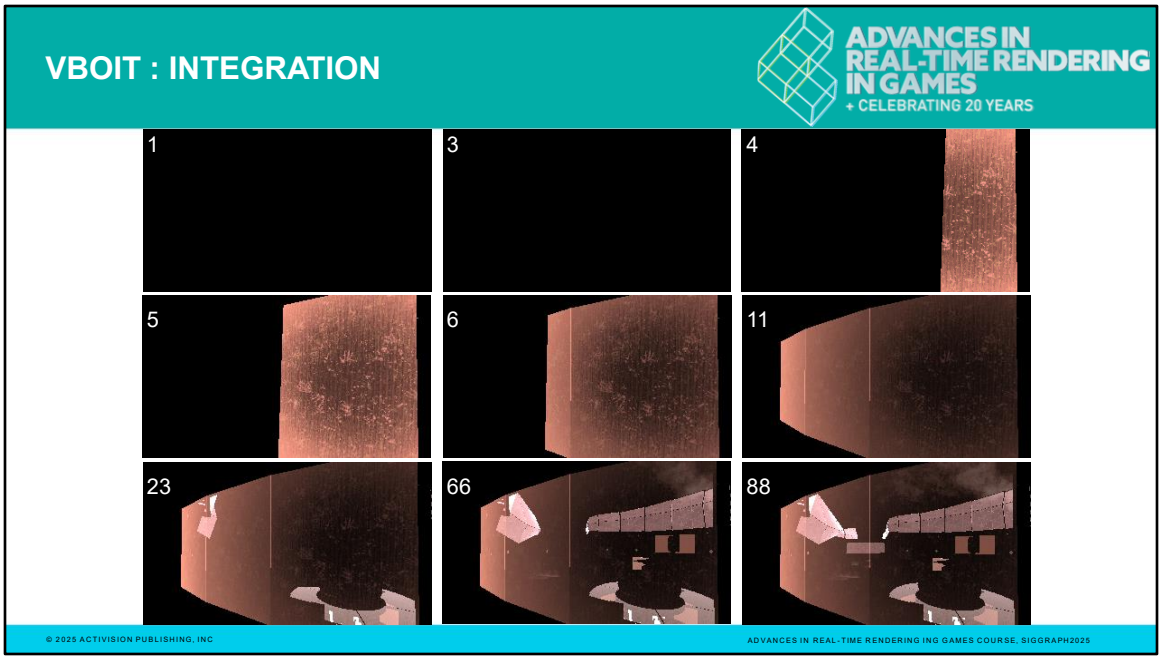


Before moving further let's look at some extinction slice captures from an example map.
Here I picked a random selection of slices, ordered in ascending order from camera.
Those slices represent any monochromatic event that splat.

VBOIT : COLOR SLICES



Similar here but this time around for RGB events.
As RGB buffers are split from scalar – you can see some overlapping at same depth but not visible together.
What is immediately clear is that occupancy of those buffers is quite sparse

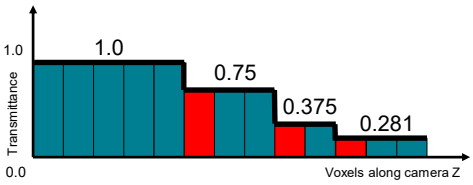


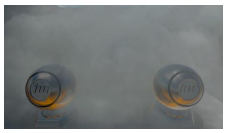
During extinction integration from both scalar and RGB sources we can still observe somewhat sparse nature of the buffer.
I specifically picked more interesting ranges here, but there is nothing happening in 0-3 as well as in 20-40 range etc
Also, empty space doesn't impact our extinction integral in any way (only costing us more).
We can also see some areas that saturate to full extinction earlier than end of buffer or even opaque scene behind it.
Maybe we can exploit this.

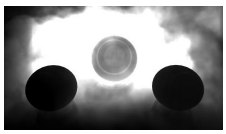
VBOIT : INTEGRATION

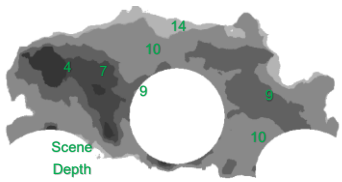
INTEGRATION CS JOB

- Observations
  - Reading empty slices is redundant
    - Can we skip them?
  - Extinction integral often saturates early
    - Before reaching opaque scene
    - Before reaching last event
    - Can we early out other rendering?





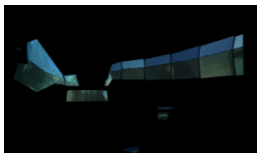


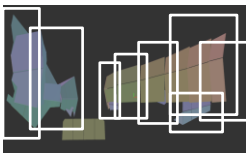


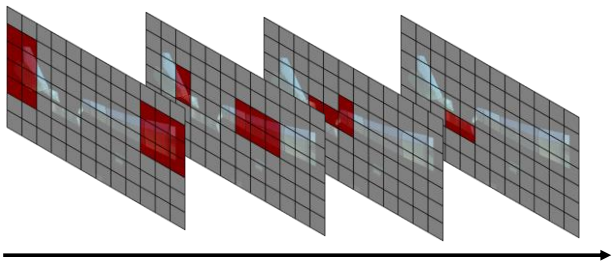
VBOIT : TILED DEPTH PREPASS

SOFTWARE RASTER BOUNDS

- Low resolution occupancy bit buffer
  - XY 1/8th resolution of Extinction resolution
  - Z 1 bit for each slice
    - Stored in U32
- CS job
  - Gather all bounding boxes and quads
  - Conservatively Rasterize / Voxelize
  - Separate for Scalar and RGB extinction
    - Most VFX use Scalar
    - Most Meshes use RGB







VBOIT : TILED DEPTH PREPASS OPTIMIZATION

ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS

SKIP EMPTY SPACES IN CLEAR AND INTEGRATION

- Empty space is not needed by any pass
  - Nothing writes to it
  - Nothing reads it
    - Integration can skip
- Clear & Integrate CS jobs
  - Read occupancy bit buffer per wavefront
    - Skip empty
- Can load balance for optimal performance
  - Not necessary at our resolutions
- Clear / Integration scale with spatial occupancy
- Good async compute candidates
- Use for resolve pass

1440p Target : RGB OIT

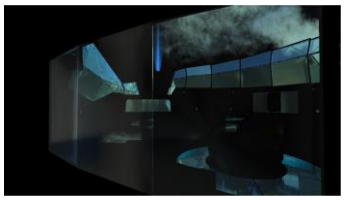
Scalar Extinction Buffer : 320x180x32 UINT32 (packed 8|8|8|8)

RGB Extinction Buffer : 320x180x128 UINT32 (packed 10|10|10)

Integrated Extinction Buffer : 320x180x128 RGBA8

Total : 66.5MB

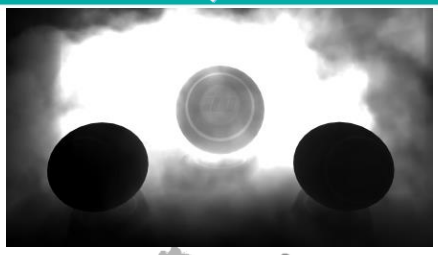
Pass	Full	Occupied
Clear	0.153	0.01
Integrate	0.2	0.04
Total	0.353	0.05

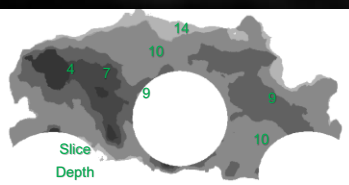


VBOIT : ZERO TRANSMITTANCE EARLY OUT

SKIP DRAWING FULLY EXTINCT PIXELS

- Once pixel transmittance goes to zero
  - We can stop drawing whatever is behind
    - Transparencies
    - Opaque geometry
- Extinction Integration job
  - Stops when extinction integral along ray reaches assumed zero transmittance
  - Write out slice index at which zero was reached
    - zeroTransmittanceDepth





VBOIT : ZERO TRANSMITTANCE EARLY OUT

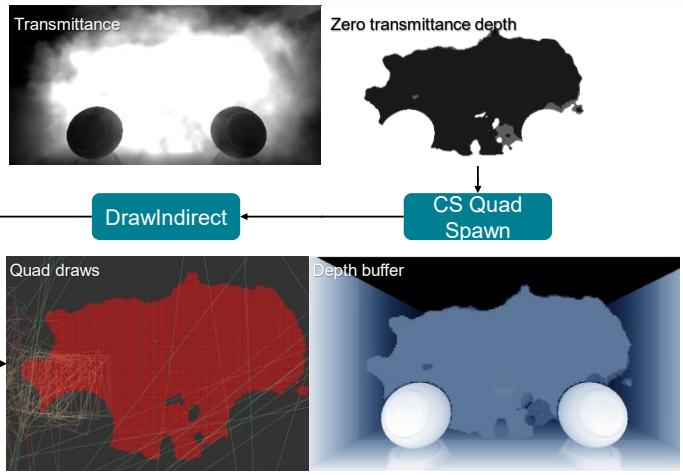
ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS

EARLY DEPTH PIPELINE

- CS Job Generates quad list
  - For each screen tile
    - Conservatively read zeroTransmittanceDepth texture
    - Generate quad covering screen tile at depth
- Populate depth buffer
  - Render all quads with DrawIndirect
  - Much faster than fullscreen SV_DEPTH

Pass	Quads	SV_DEPTH
Spawn	0.004	N/A
Depth Draw	0.01	0.2
Total	0.014	0.2

4k depth buffer from 480x270 zeroTransmittance | PS5



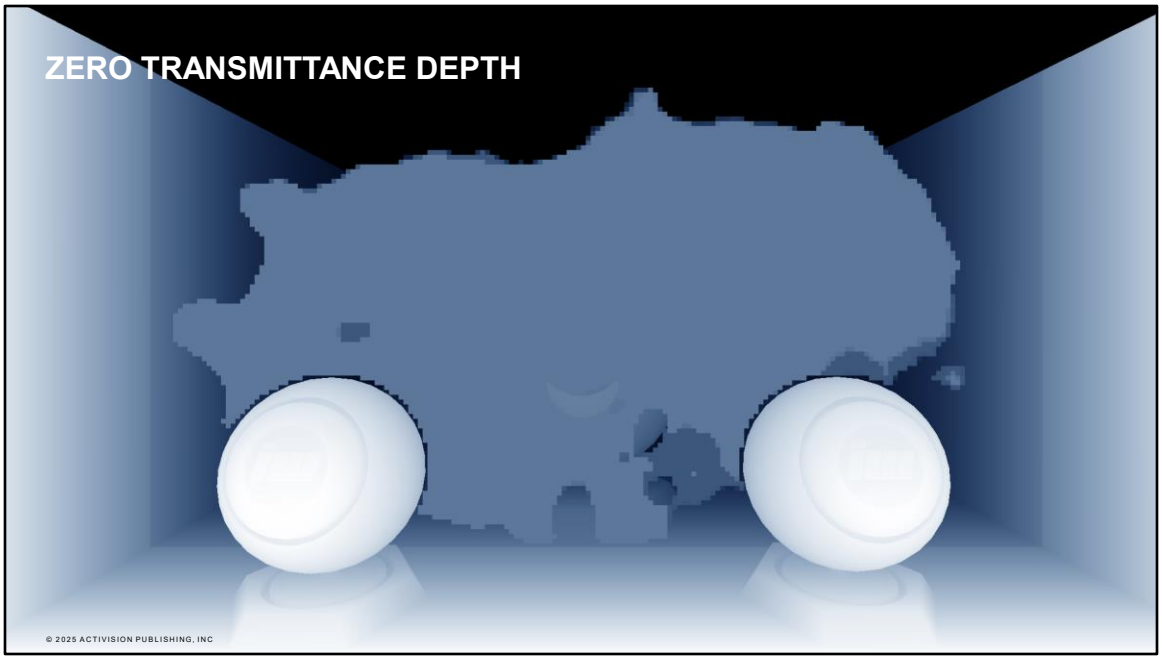
Tile size at least 16x16

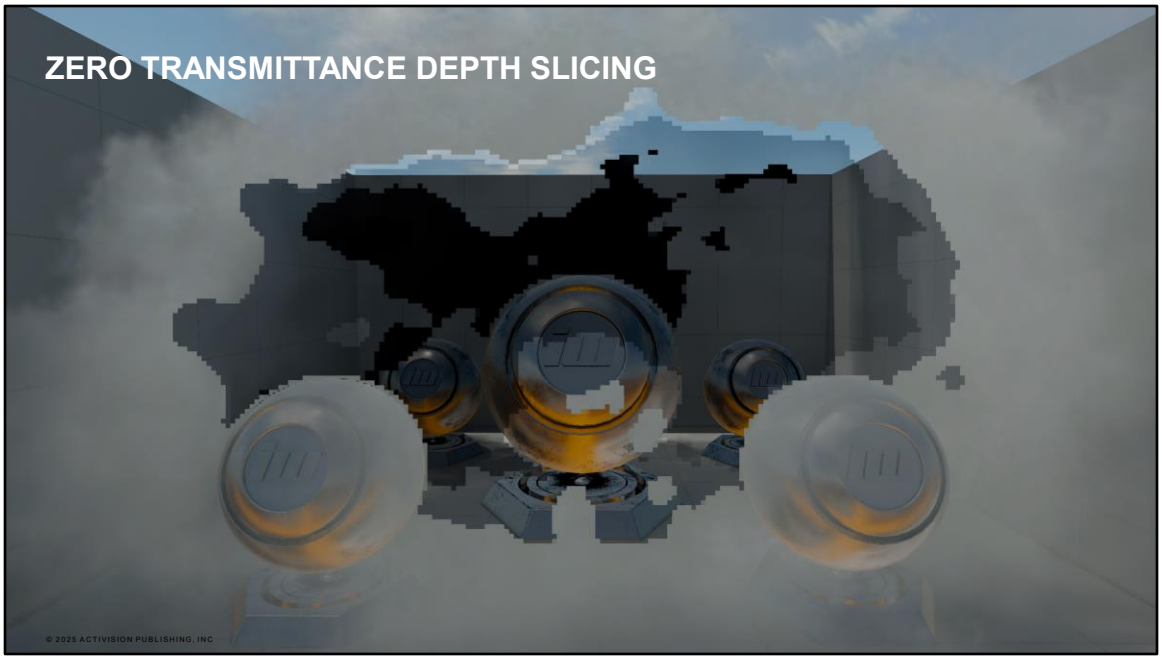
Guarantees compressed depth

HiZ benefits

Much faster than doing a single quad with SV_Depth output

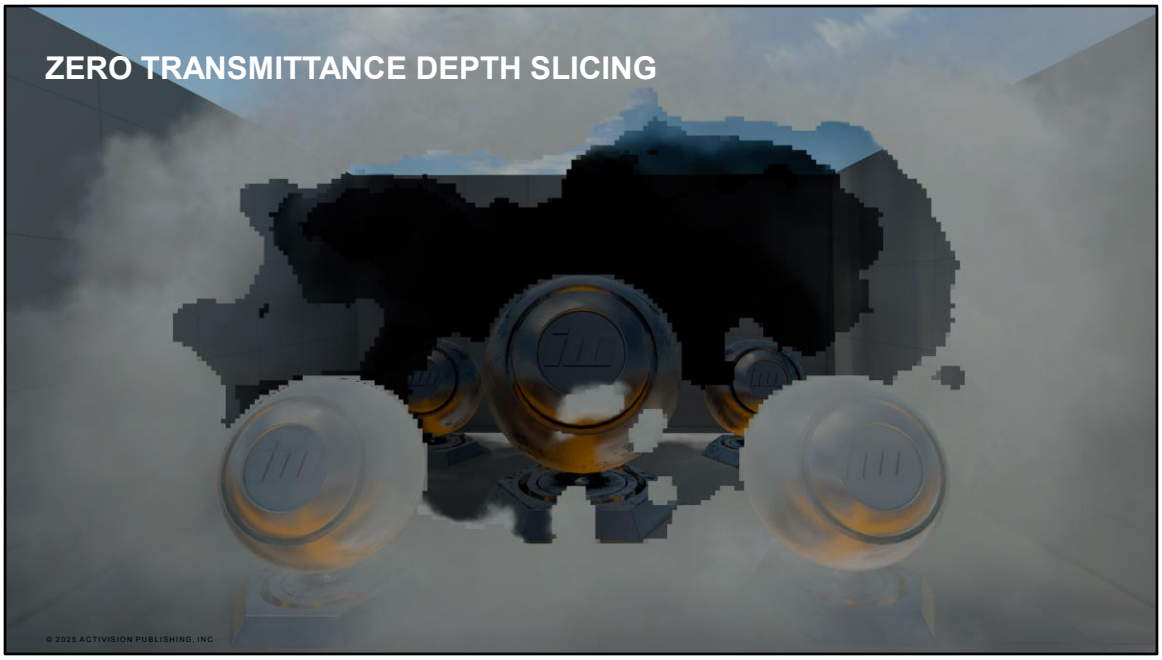

VBOIT VFX WITH ZERO TRANSMITTANCE EARLY OUT
© 2025 ACTIVISION PUBLISHING, INC


ZERO TRANSMITTANCE DEPTH
© 2025 ACTIVISION PUBLISHING, INC



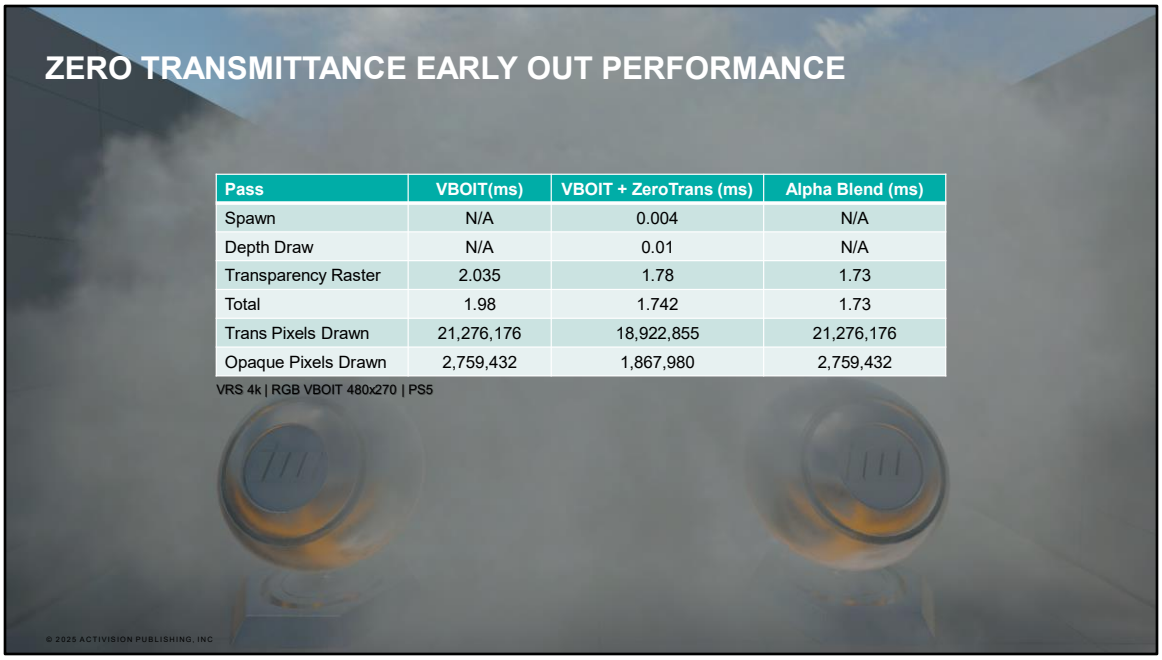
Slicing through the accumulated integral we can see how our zero-transmittance depth effectively culls out draws in 3D

ZERO TRANSMITTANCE DEPTH SLICING

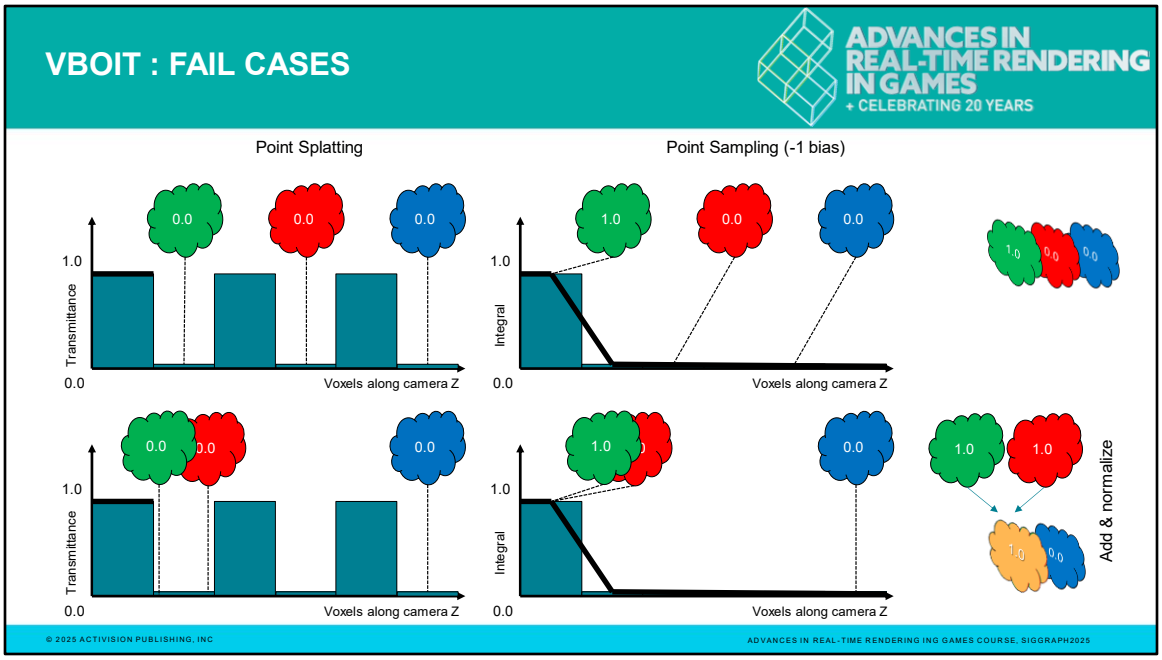



ZERO TRANSMITTANCE DEPTH SLICING
© 2025 ACTIVISION PUBLISHING, INC


ZERO TRANSMITTANCE DEPTH SLICING
© 2025 ACTIVISION PUBLISHING, INC

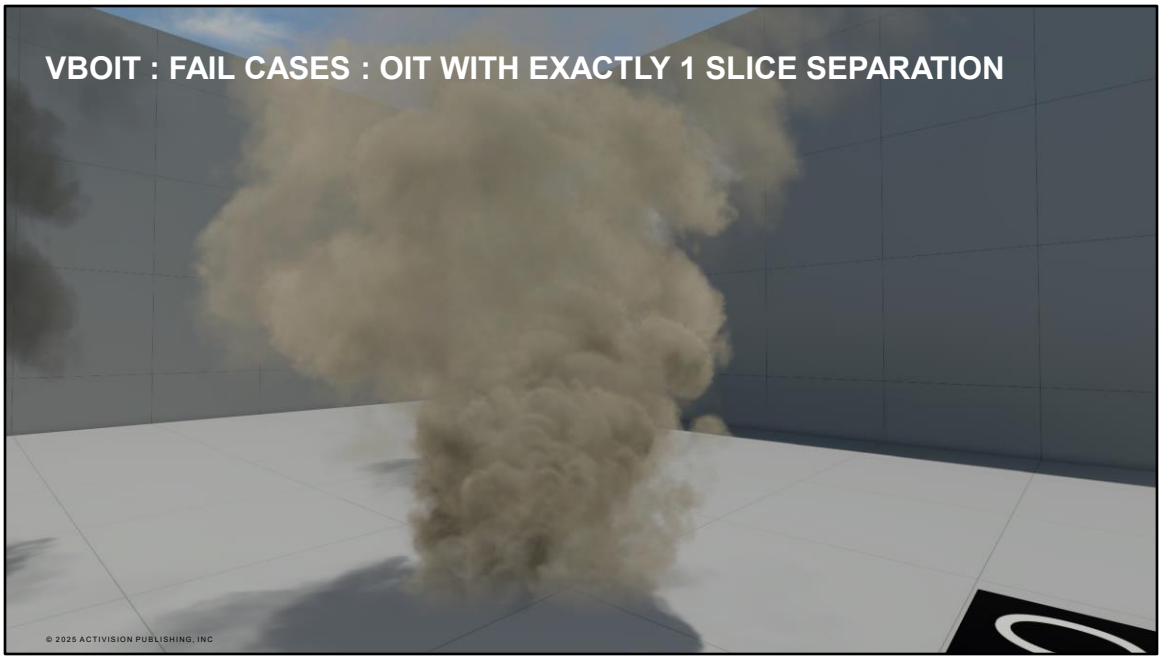


Effectively leaving no trace of visual artifacts (as those should not be possible if we guarantee that past that point nothing renders).
Using zero-transmittance depth comes as a significant optimization to all transparent draws.
Furthermore, if your engine order allows rendering transparencies first, you might use zero-transmittance depth to further cull any opaque draws.
For sake of this presentation and presented performance numbers we are NOT using zero-transmittance depth to cull opaque geometry.

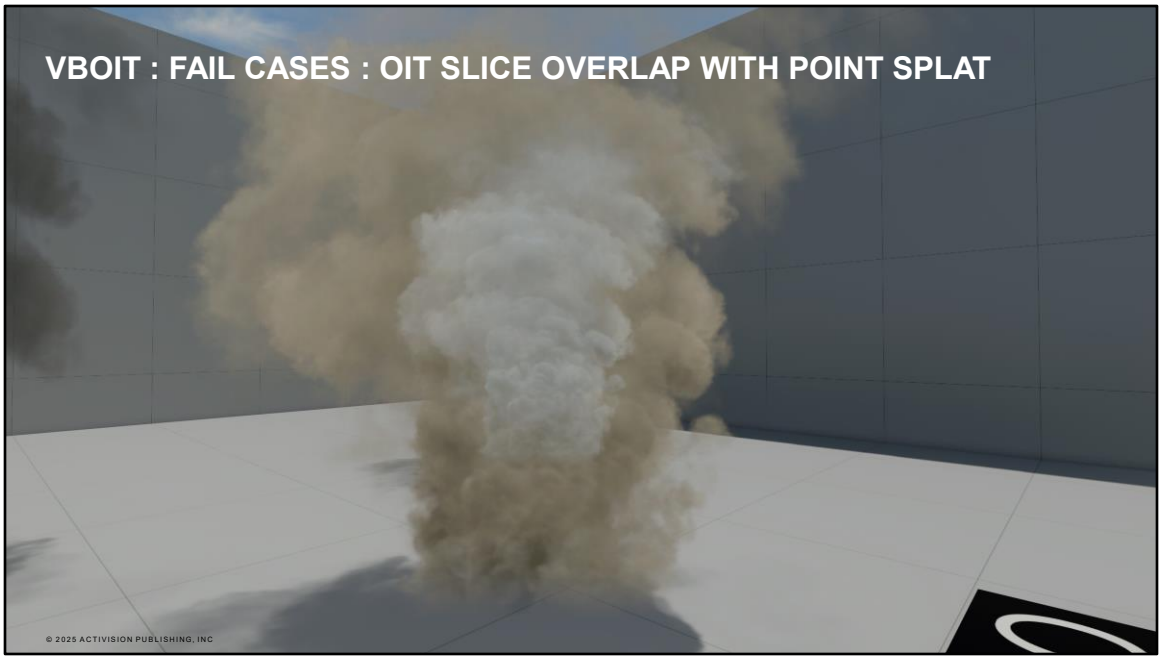


If transmittance events overlap inside same slice
Surfaces will get over / under occluded
VBOIT inside single slice degenerates to Weight Blending with linear splatting implicit weight.


VBOIT : FAIL CASES : NO OIT
© 2025 ACTIVISION PUBLISHING, INC

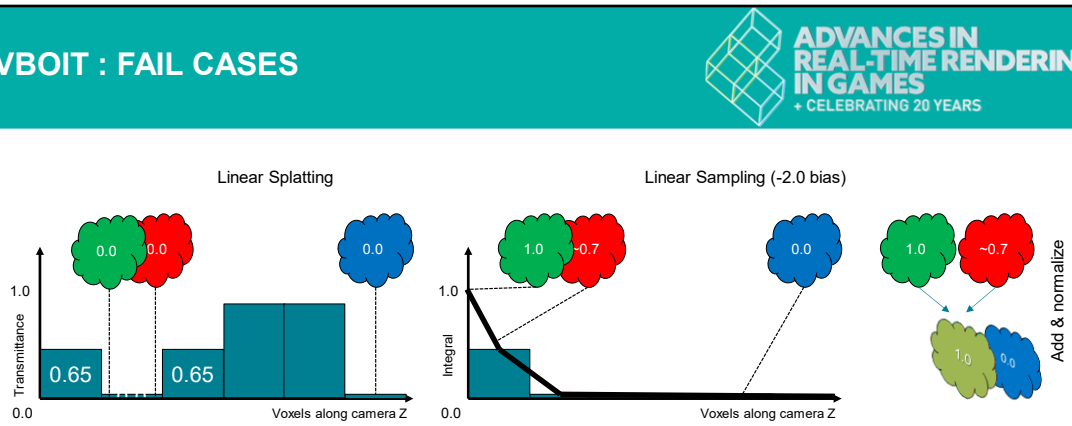


When there is enough depth separation resulting in at least a single slice separation, rendering is artifact free.



However, when events overlap same slice, we are starting to deal with frequency soup and thus a mix of all events landing over same depth slice.

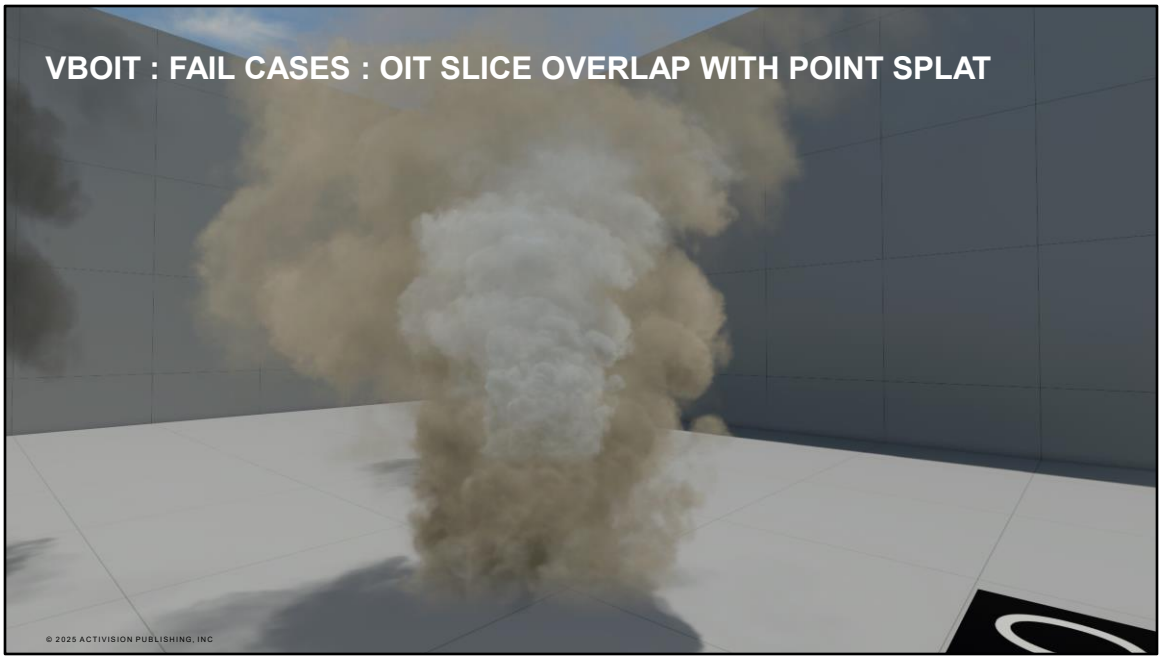
VBOIT : FAIL CASES

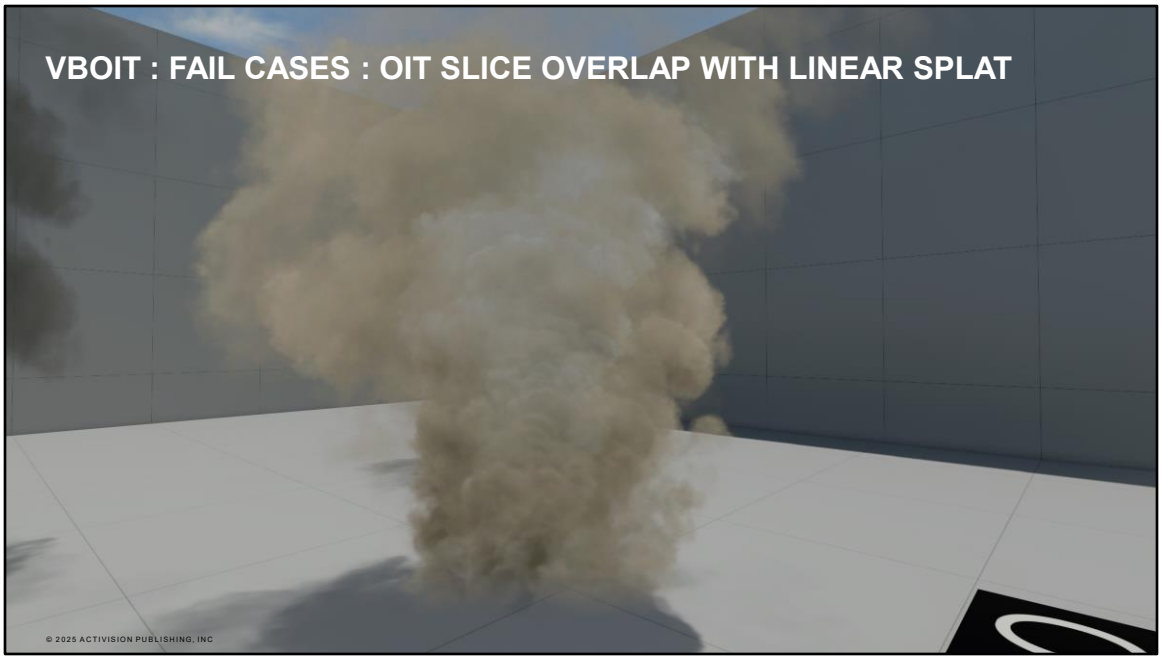


- Linear Splatting & Sampling improves Color mix via spatial information

Linear splat interpolation helps with over / under occlusion separation but extends self occlusion range

For precise results it is crucial to improve depth separation – thus our goal should be to provide enough slices where it matters


VBOIT : FAIL CASES : OIT SLICE OVERLAP WITH POINT SPLAT
© 2025 ACTIVISION PUBLISHING, INC



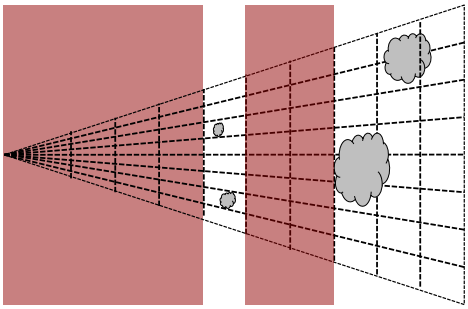
Linear splatting alone can help making the artifacts less visible as it does provide some depth separation via weighting.
That said for precise results it is crucial to improve depth separation – thus our goal should be to provide enough slices where it matters

To address slice precision we would like to introduce extension to VBOIT called Adaptive VBOIT

ADAPTIVE VBOIT

SPARSE USAGE OVER DEPTH

- Already have coverage information prior
- Skip empty spaces
- VBOIT is invariant (between slices)
  - Shift
  - Data
- Can create compressed depth curve to pack slices



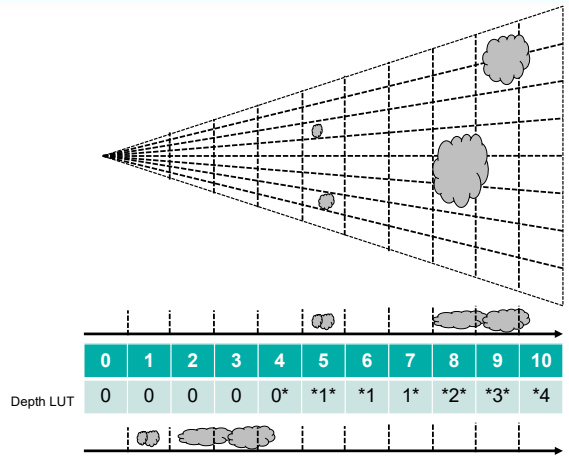
We already know that empty space can be skipped. Furthermore, due to sparse buffer population necessary for clear / integration optimizations, we already know where empty space is.

ADAPTIVE VBOIT

ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS

DEPTH WARP LUT

- Z binning for depth slices [DRO17.2]
- Before splatting
  - Calculate prefix sum of Z occupancy
  - Compact
  - Mark range begin / end as 'filterable' (*)
    - Disable filtering in empty spaces
    - Snap filtering to begin / end points
      - Recalculate fractional position
  - LUT needs to be filterable
- Sample 1D Depth Warp LUT in
  - Splatting
  - Sampling



	0	1	2	3	4	5	6	7	8	9	10
Depth LUT	0	0	0	0	0*	*1*	*1	1*	*2*	*3*	*4

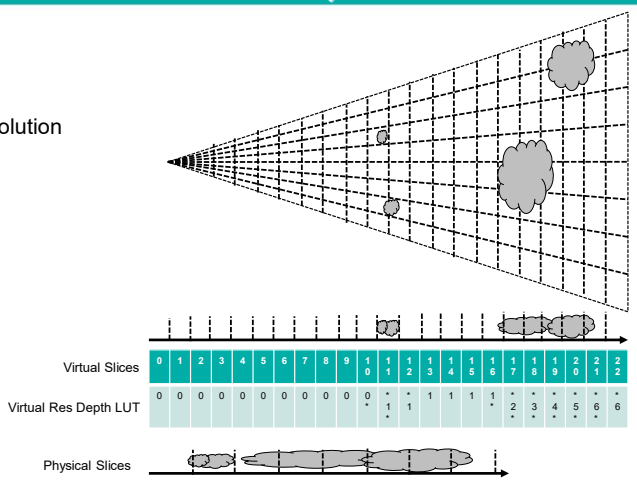
A simple scheme that can trivially skip empty space in coherent manner would be to warp depth over certain ranges.
Working only in single dimension (as opposed to going fully sparse in XY and Z) significantly simplifies all the steps necessary to construct this data structure as well as to sample it.
Using prefix sum of Z occupancy of slices, we generate a 1D Depth Warp LUT that on splat and read – for a given depth – returns a slice skipping all empty space.
For linear interpolation it also needs to reserve extra bits marking if this warped depth is beginning, or end or middle of a non-empty region – so linear sampling can be adjusted in order to simulate a continuous unchanged space (by snapping to end of previous slice).
This scheme provides an attractive alternative to partially skipping space in clear and integration passes – while alone doesn't yet improve visual precision.
It does however provide an effective packing scheme.

ADAPTIVE VBOIT

ADVANCES IN REAL-TIME RENDERING IN GAMES + CELEBRATING 20 YEARS

REDISTRIBUTE RESOLUTION

- Calculate 1D Z coverage at virtual slice resolution
  - Sum occupied slices
  - If does not fit physical slice resolution
    - Lower virtual slice resolution
    - Re-adjust log depth curve
- Fits best resolution into physical scratch



Our depth distribution curve is logarithmic and parametrized in such a way that we can improve precision over visible range by consuming more slices (visually flattening the curve).
Before even splatting we specify the requested minimal slice thickness over visible depth range, set slice count to large number of slices that would allow that slice thickness and generate occupancy buffer with that parametrization.
This will most likely result in oversubscription of slice count.
We generate slice occupancy at this ‘high slice resolution’ and check if count of occupied slices would fit our expected physical budget.
If not we reparametrize the curve again by halving slice count and repeat packing process.
Fwiw coverage data doesn’t need to be regenerated, just conservatively rewritten into ‘lower resolution’.
The process is repeated until we can fit all slices in memory – which is bound by parametrization of our physical slice buffer.

Once process is completed, we can splat, integrate and sample a parametrization that can potentially have a higher count of ‘virtual slices’ than physical slices, using our 1D Depth Warp LUT indirection.

ADAPTIVE VBOIT

- Depth LUT
  - ~0 performance cost at sample
  - ~0 performance cost to generate
- Integration cost scales with slice occupancy
  - Tune physical memory pool
  - Performance
  - Precision
  - Memory consumption


SIGGRAPH 2025
Vancouver+ 10-14 August
RESULTS
ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS
CALL OF DUTY
WARZONE

TESTING SETUP

SHARED FRAMEWORK

- All methods
  - Same Resolve & Transparency Rendering Pass
  - Transmittance Prepass at 1/8th resolution
  - Same depth bounds
- Prepass Output
  - MBOIT (rank 2) : 5xFP16 (x3)
  - WBOIT (rank 2) : 8xFP16 (x3)
  - AVBOIT : 8bit x 128 slices (+ 32bit x 128)
    - 8k Depth Warp LUT
- Transparency Output (RGB)
  - Accumulated Color : 11R11G10B_Float
  - Accumulated Extinction : 16F(11R11G10B_Float)
  - Accumulated Norm Denominator : 16F(11R11G10B_Float)

Method	Transmittance Function Memory 1/8th 4k				
	Write(bpp)	Read(bpp)	Sampling	Size (mb)	Mem Access
MBOIT	$[1+(2^r)]$ (10)	10	2 x 2D	1.3	dense
WBOIT	$[1+r+1]$ (8)	8	3 x 2D	2	dense
AVBOIT	1	1	1 x 3D	16.8	sparse

Method	Transmittance Function Memory 1/8th 4k				
	Write(bpp)	Read(bpp)	Sampling	Size (mb)	Mem Access
MBOIT	30	30	4 x 2D	4	dense
WBOIT	24	24	3 x 2D	6	dense
AVBOIT	1-4	1-4	1 x 3D	66.3	sparse

RGB PERFORMANCE

SPLIT CHROMATIC SOURCES

- AVBOIT
  - Split chromatic transmittance sources
    - Pay extra prepass cost only for RGB sources
  - Combine in transmittance integration
- MBOIT/WBOIT
  - Not obvious how to split
  - Not obvious how to combine for sampling

Pass	RGB Extra Cost (x)		
	MBOIT	WBOIT	AVBOIT
Clear	3x	3x	3x
Prepass	1.9x	1.7x	1.04x
Integration	N/A	N/A	3x
Draws	1.4x	1.3x	1.05x
Resolve	1.1x	1.1x	1.1x
Avg Cost	>1.5x	>1.3x	~1.05x

In case of AVBOIT, we store RGB extinction buffers separate at extra memory cost.
This however allows us to significantly cut down on bandwidth during clears, splatting and integration – as those expensive operations only need to happen on slices used by RGB surfaces.
Integration reads monochromatic extinction and RGB extinction – merging them together into RGB integral (at which point both extinction types buffers can be thrown away).
In general, in our content – it is way more likely to have a monochromatic event coming from particles.
Obviously, this choice is very project and content dependent and, in a case, where vfx use RGB transmittance – it becomes more efficient to just keep one RGB representation around.


NO OIT
© 2025 ACTIVISION PUBLISHING, INC

Transparencies show through.

Transparencies show through.

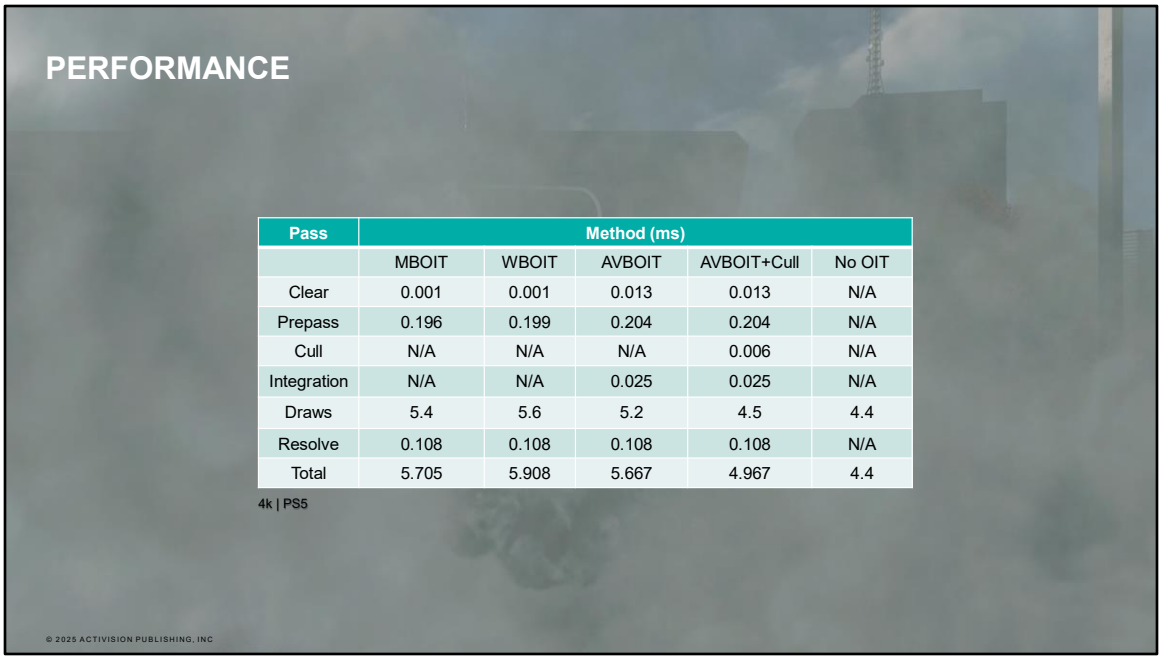
Note: puff of smoke in the middle is in original asset. AVBOIT makes it stand our due to improved depth separation.

COVERAGE AT DISTANCE
150M (LONG DISTANCE ENGAGEMENT)

ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS

No OIT | MBOIT | WBOIT | AVBOIT

No OIT exhibits correct occlusion of car in the background. Unfortunately, it doesn't correctly resolve glass panes on the building on right side.
MBOIT and WBOIT resolve glass and smoke to a degree but fail to occlude glass in the background.
AVBOIT resolves bot problematic events correctly.

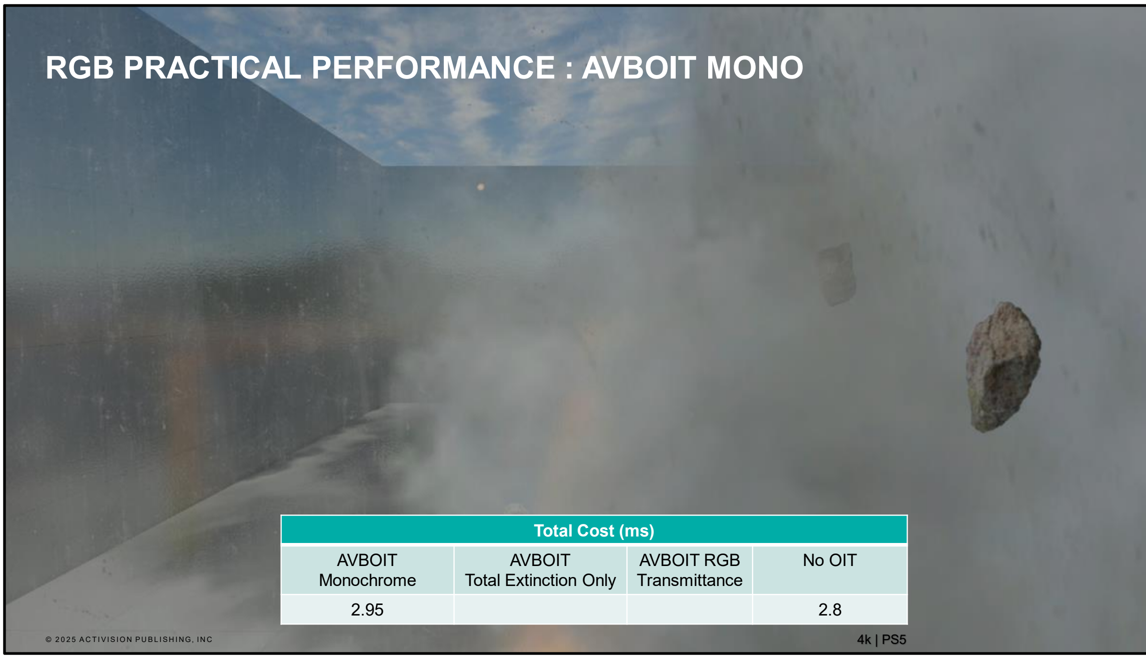


For performance considerations we are assuming that MBOIT / WBOIT is running on a platform configured for maximum bandwidth throughput (i.e. accelerated clears, accelerated ‘zero’ render target output etc.)
MBOIT / WBOIT clears should be HW accelerated on all platforms and PC. ‘zero’ output might vary.
AVBOIT shows as a compelling alternative to all other OIT methods – getting close into no OIT territory.

RGB PRACTICAL PERFORMANCE : NO OIT

Total Cost (ms)			
AVBOIT Monochrome	AVBOIT Total Extinction Only	AVBOIT RGB Transmittance	No OIT
			2.8

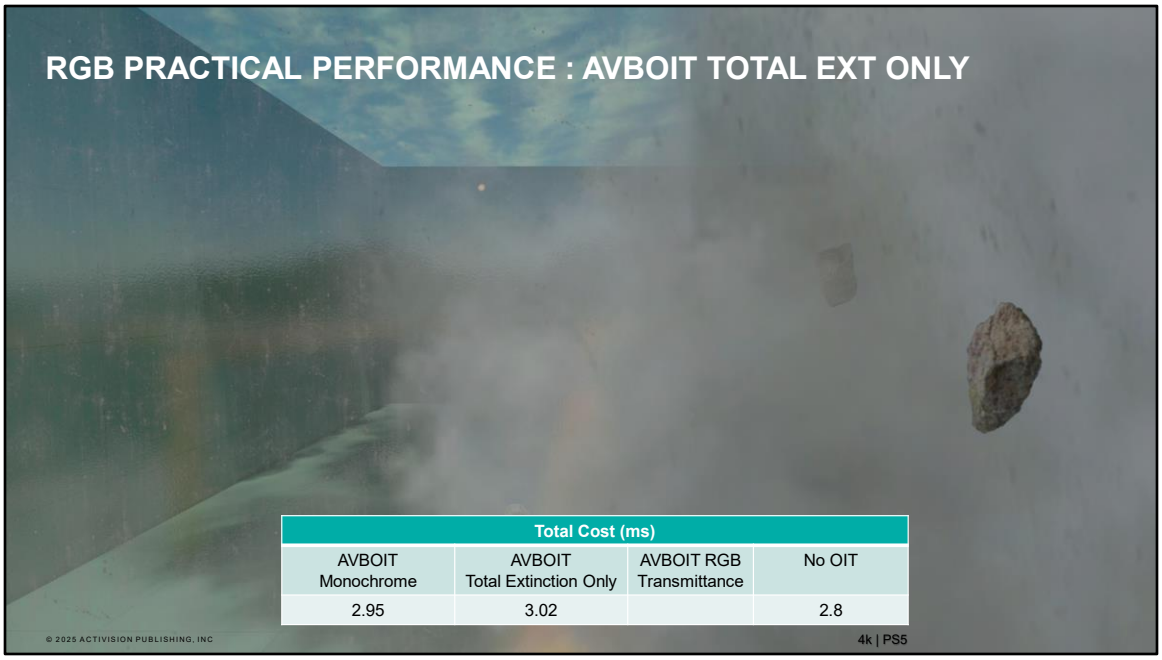
4k | PS5

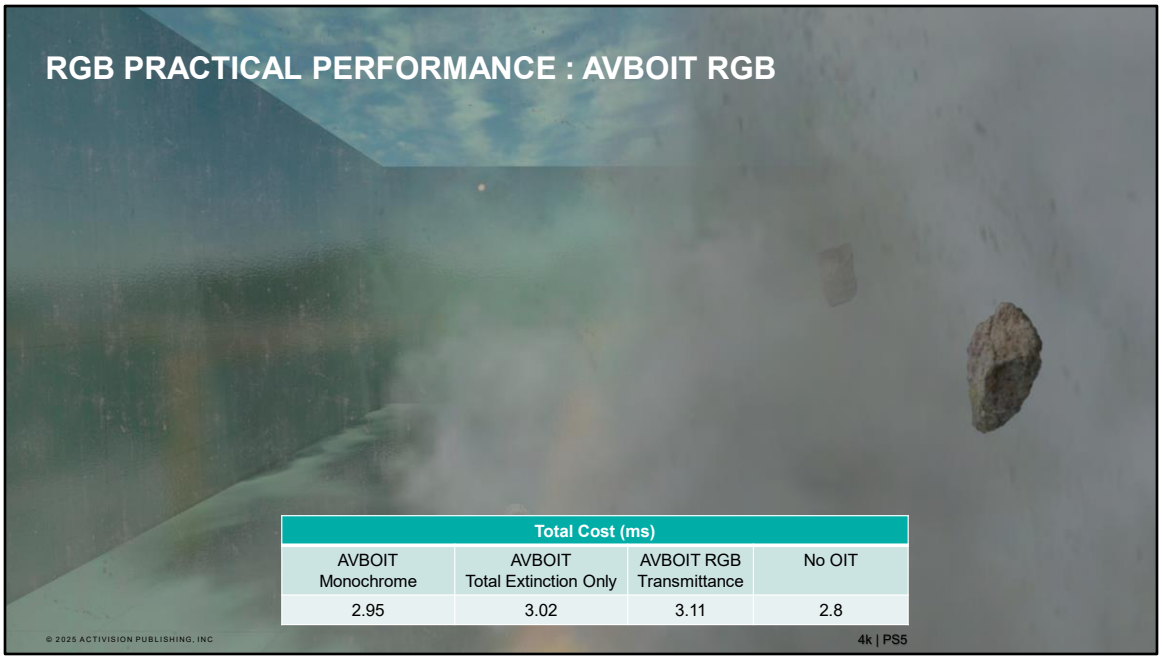


RGB PRACTICAL PERFORMANCE : AVBOIT MONO

Total Cost (ms)			
AVBOIT Monochrome	AVBOIT Total Extinction Only	AVBOIT RGB Transmittance	No OIT
2.95			2.8

4k | PS5

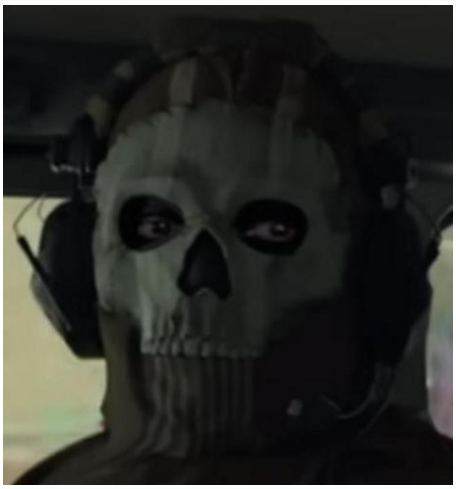

RGB PRACTICAL PERFORMANCE : AVBOIT TOTAL EXT ONLY
Total Cost (ms)
AVBOIT Monochrome
AVBOIT Total Extinction Only
AVBOIT RGB Transmittance
No OIT
2.95
3.02
2.8
© 2025 ACTIVISION PUBLISHING, INC
4k | PS5


RGB PRACTICAL PERFORMANCE : AVBOIT RGB
Total Cost (ms)
AVBOIT Monochrome
AVBOIT Total Extinction Only
AVBOIT RGB Transmittance
No OIT
2.95
3.02
3.11
2.8
© 2025 ACTIVISION PUBLISHING, INC
4k | PS5

AVBOIT : PRODUCTION IMPACT

LESS WORK & LESS BUGS

- Environmental art team
  - No impact, no bugs, problems fixed
  - No need for manual sorting
    - No custom sort order materials
    - No thinking about ordering during design
- VFX team
  - Many problems fixed
  - New recurring “bug”
    - “VFX emitters don’t sort according to manual sort”
    - “UI doesn’t sort correctly”

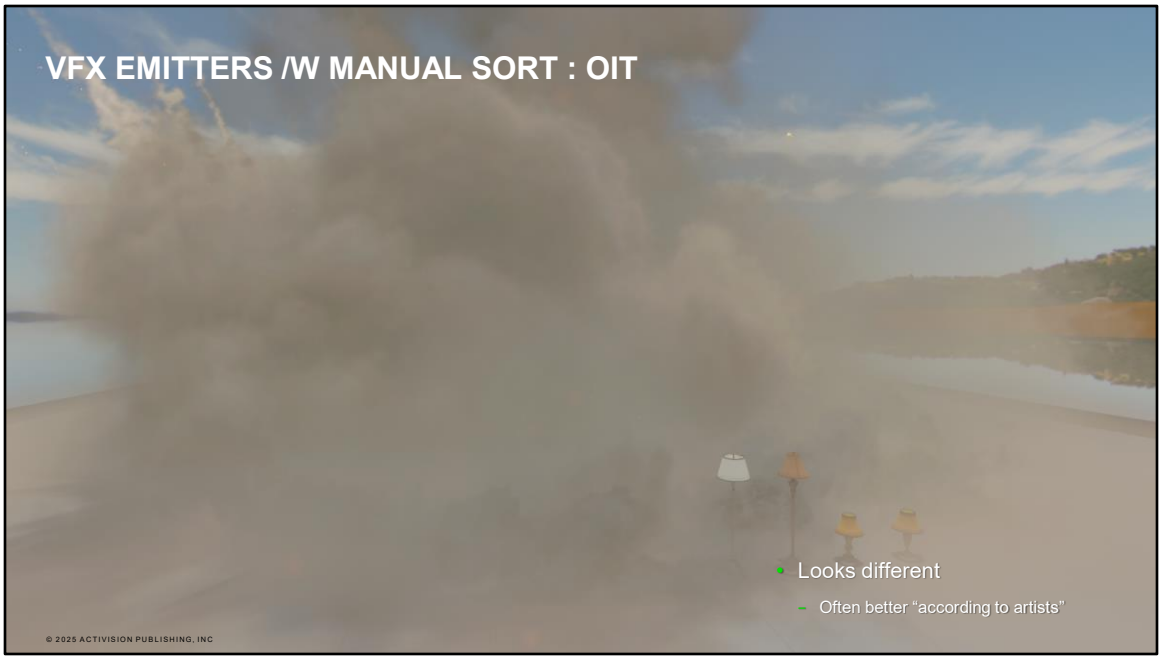


Environments are usually well behaved and do not abuse physical logic of sorting. Same can’t be said about VFX where the final effect is often achieved by smoke and mirrors.

Some of our VFX had emitters with manual sort order – that was used to exaggerate certain elements. In this case fireball showing strongly through smoke.

Resolving this scene with OIT results in physically correct image – which unfortunately was not the artists intention.
Recovering this look in exact way is impossible, but we can potentially get closer to it by emulating sorting with placement of effects over depth.
Artists already have access to ‘camera offset’ that moves VFX depth along camera ray. This can be used to place effects at certain ranges or even go as far as presetting fixed amount of ‘biases’ that can be used for layering.

That said, manual sorting can result in a lot of side effects as we can see later during explosion frames where smoke tendrils look buggy and are unintended at that point.



Which can be naturally fixed by OIT.
Of course, this depends on what was the intention – but the important piece of information is that with depth offsetting tools some of this look can be achieved even with OIT.

UI DOESN'T SORT CORRECTLY

3D UI IN THE WORLD

- 2D UI classic composition mode
- 3D UI uses 3D VFX
  - Sorts with the scene
  - Needs similar 2D composition
- Camera offset /w perspective adjust
  - 'sorting' burns slices but workable
- Offscreen 2D UI compose
  - Use in 3D as composed texture

Some content teams were using VFX to compose 2D UI in the 3D world.
UI composition layering has its own layering and blend logic that assumes specific sorting orders.
Unfortunately, that is not supported by OIT as this is exactly the close depth overlap fail case scenario.
There are some way to support it with proposed AVBOIT.
A certain number of slices can be dedicated to layering – which would emulate strict ordering of those VFX.
Similar effect can be achieved by manually altering vertex shaders for those draws, that would manually alter depth at which those draws render to simulate sorted layering.
Finally, the recommended way is to render UI as 2D composite offscreen, and once composed, use it as texture in 3D world – which avoids all those problems.

MIN SPEC PERFORMANCE / GEN8

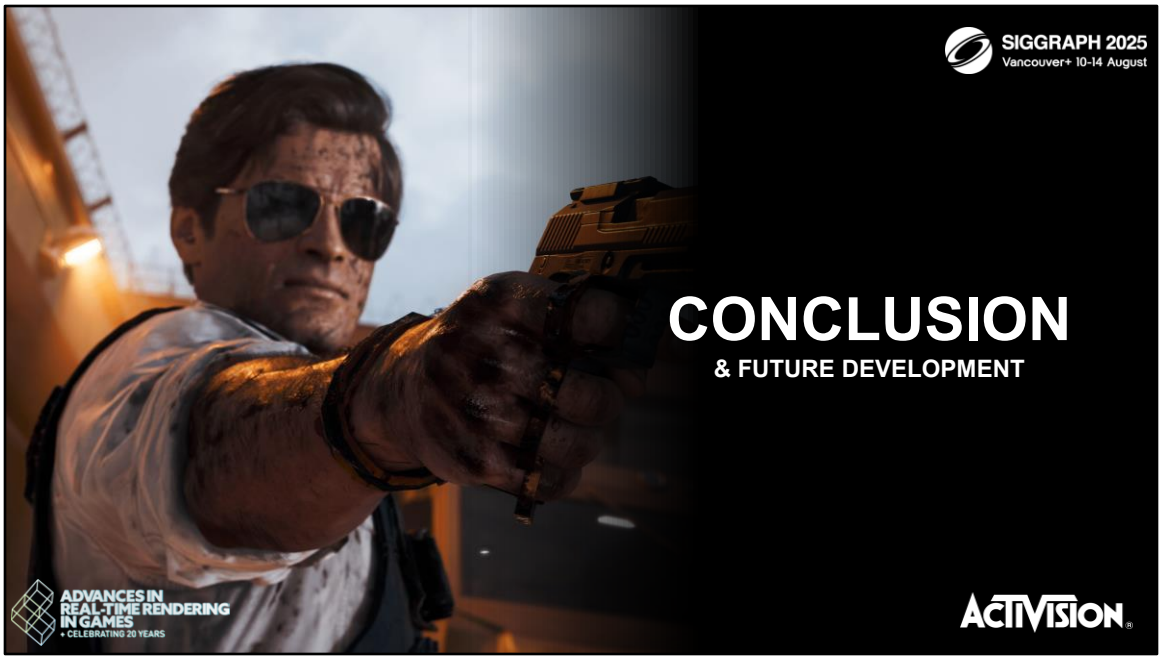
SIMILAR SCALING ACROSS GENERATIONS

- Gen8 similar scaling to Gen9
- Same for PC low specs
  - AMD RX470
  - Nvidia Gtx 1060

Nvidia Gtx 1060 Cinematic test 1080p (ms)	
AVBOIT RGB Transmittance	No OIT
12.2	11.2

Pass	PS4 1080p (ms)			
	MBOIT	WBOIT	AVBOIT	No OIT
Transparency	3.31	3.04	2.11	1.8
VFX	3.34	3.12	2.91	2.72
Total	6.65	6.16	5.03	4.5

ABVOIT provides a reasonable performance alternative on all paltforms.


SIGGRAPH 2025
Vancouver+ 10-14 August
CONCLUSION
& FUTURE DEVELOPMENT
ADVANCES IN
REAL-TIME RENDERING
IN GAMES
+ CELEBRATING 20 YEARS
ACTIVISION

AVBOIT : CONCLUSION

GOOD ENOUGH

- **Fast(er) as no OIT**
  - Early cull
  - Multi resolution rendering
- Stable
- Minimal content rework
- RGB transparency OIT becomes practical
- Scalable
- Opens door for new features
  - OIT refraction & scattering
  - Transparency depth of field
  - Transparency motion blur

EXTENSIONS : DISTORTION

EXTENSIONS : DISTORTION

ACCUMULATE DISTORTION AND WEIGHT BY TRANSMITTANCE INTEGRAL

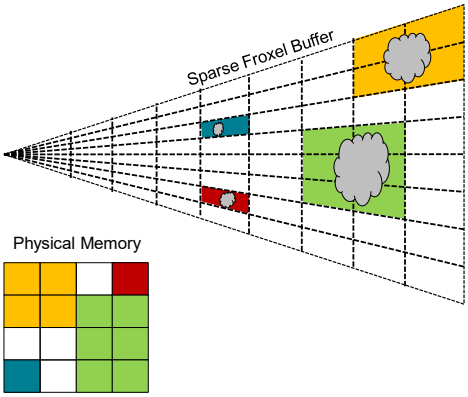
ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS

- Weight more things by transmittance integral
  - Motion vectors
  - Optical depth for DOF
  - Refraction and scattering

AVBOIT : FUTURE WORK

SPARSE VOXEL TEXTURE

- High precision needs a lot of memory
  - But usage is very sparse
- Software bounds depth pre-pass
  - occupancy data at target resolution
- Use occupancy to allocate chunks of virtual buffer
- On physical memory overflow
  - Reallocate to lower target resolution
    - Repat on fail



REFERENCES

- [KOH16] Practical Order Independent Transparency, Johan Köhler, Treyarch 2016
- [VAS20] A Survey of Multifragment Rendering, A. A. Vasilakis et al., Eurographics 2020
- [WYM16] Exploring and Expanding the Continuum of OIT Algorithms, Wyman, C., HPG16
- [MB13] Weighted Blended Order-Independent Transparency, Morgan McGuire, Louis Bavoil, Siggraph 2014
- [KIR18] Rendering Technology in 'Agents of Mayhem, Scott Kircher, GDC 2018
- [ASM21] Wavelet Transparency, Maksim Aizenshtein, Niklas Smal, Morgan McGuire, 2021
- [PMWK17] Improved moment shadow maps for translucent occluders, soft shadows and single scattering, Peters, Münstermann, Wetzstein, Klein, JCGT 6, 1 March 2017
- [BAV13] Particle Shadows & Cache-Efficient Post-Processing, Louis Bavoil, GDC 2013
- [WRO14] Volumetric fog: Unified, compute shader based solution to atmospheric scattering, Bart Wronski, Siggraph 2014
- [DRO17] Rendering of Call of Duty : Infinite Warfare, Michal Drobot, Digital Dragons 2017
- [DRO17.2] Improved Culling for Tiled and Clustered Rendering in Call of Duty: Infinite Warfare, Michal Drobot, Siggraph 2017
- [SAL11] Adaptive Transparency, Marco Salvi et al., Siggraph 2011
- [SHA18] Moment Transparency, Brian Sharpe, Hpg 18
- [ESSL10] Stochastic transparency, Enderton E., Sintorn E., Shirley P., Luebke D.: I3DG 2010
- [WYM17] Hashed Alpha Testing, Wyman Chris, Morgan McGuire, I3DG 2017
- [DEL11] Transmittance Function Mapping , Cyril Delalandre, Pascal Gautron, Jean-Eudes Marvie, Guillaume Francois, I3D 2011
- [LOK00] Deep Shadow Maps, Tom Lokovic, Eric Veach, Siggraph 2000
- [NGU05] Real-time rendering and animation of realistic hair in 'nalu', Hubert Nguyen, William Donnelly, GPU Gems 2
- [SIN08] Real-time approximate sorting for self shadowing and transparency in hair rendering, Erik Sintron, Ulf Assarsson, Siggraph 2008


JOIN US
www.activisionblizzard.com/careers
BLACK OPS 6
© 2025 Activision Publishing, Inc.


research.activision.com
Q&A
michal@infinityward.com
@MichalDrobot
© 2025 SIGGRAPH. ALL RIGHTS RESERVED.
© 2025 Activision Publishing, Inc.


BONUS
CALL OF DUTY
BLACK OPS 6

Using monochrome transmittance OIT results in visually correct results – but clearly lacking color.

Accumulating color transmittance during transparency draws will result in correct color of all opaque pixels, but will create odd looking color misses on any transparency.
I.e. here opaque pixels behind glass show green tint, while smoke, that is behind glass (and looked convincing in monochrome) suddenly sticks out missing color hint.

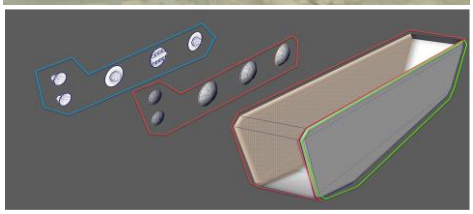
Applying chromaticity skew in resolve partially solves the problem with incorrectly colored transparency but is unable to resolve depth dependent complexities with partial incorrect color occlusion.

Using RGB OIT extinction integral correctly resolves the image.

VBOIT : COLOR APPROXIMATION : FAIL CASE

HARD PROBLEM

- VBOIT fails due to slice overlap
  - Color skew will get mixed in
  - Low precision of normalization and extinction does not help
- Car fixture is made of 2 glass surfaces
  - Back to back
  - 0.125 inch depth delta
  - Front is clear refractive plastic
  - Back is red close to opaque plastic
- Slices with overlap can be detected
  - Remove color skew
  - Overly conservative but shippable
- Ask artists to avoid this setup
- **RGB Transmittance fixes the problem**
- **More slices fixes the problem**



VBOIT : COLOR

FOR MEMORY CONSTRAINED PLATFORMS

- Skew final color by accumulated RGB extinction
  - Use monochrome transmittance integral
  - Accumulate color extinction in transparency pass
    - Run only for color transparencies
  - During resolve
    - Accumulated RGB extinction integral from transparency pass for background
    - Skew normalization factor by transmittance chromaticity

```
// Transparency draw
float3 extinction = GetExtinctionFromTransmittance( 1.0 - transparency );

if( OitEnableChromaTransSkew() )
{
 extinctionRGB = lerp( extinction, extinctionRGB, transmittanceIntegral *   FogTransmittanceIntegral );
}
```

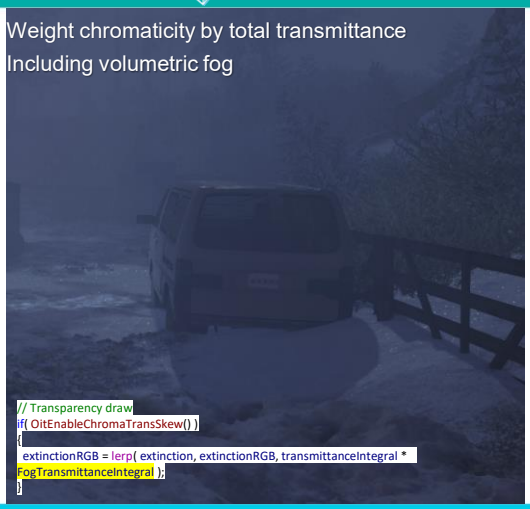
```
// Resolve
if( OitEnableChromaTransSkew() )
{
 float transmittanceLuma = GetLuma( transmittance );
 chromaSkew = transmittanceRGB * rcp( transmittanceLuma );
}

float3 cf = opaqueColor * transmittance
 + transColor * normFactor * chromaSkew;
```

VBOIT : COLOR APPROXIMATION : FAIL CASE

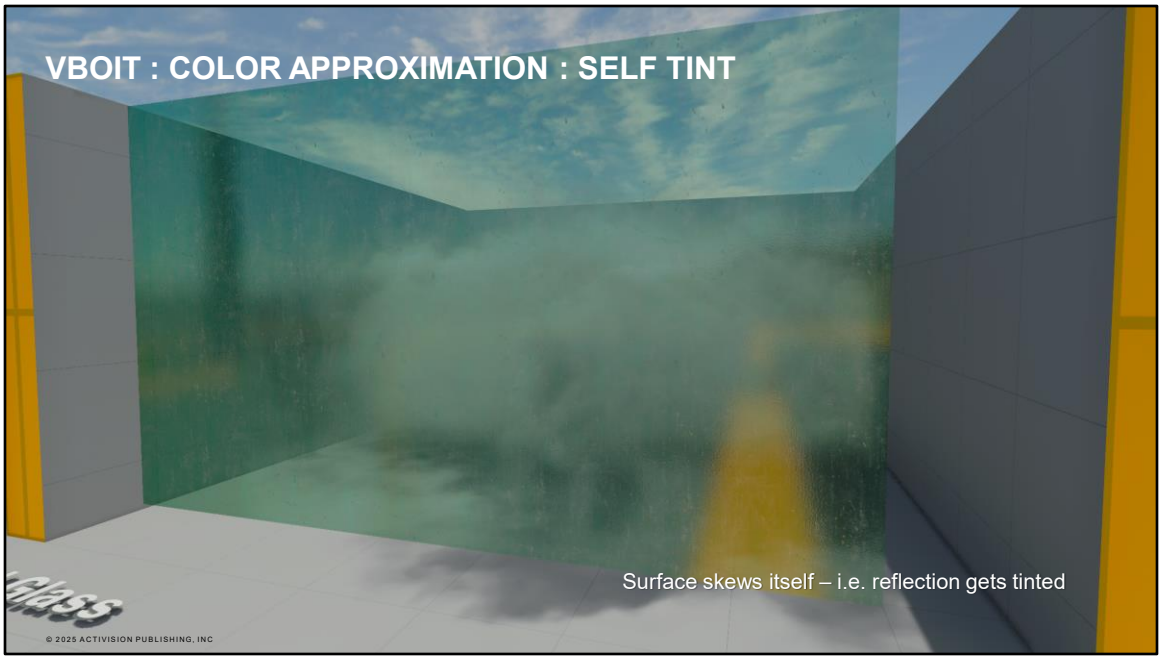
ADVANCES IN REAL-TIME RENDERING IN GAMES
+ CELEBRATING 20 YEARS

Fog scattering gets color skew

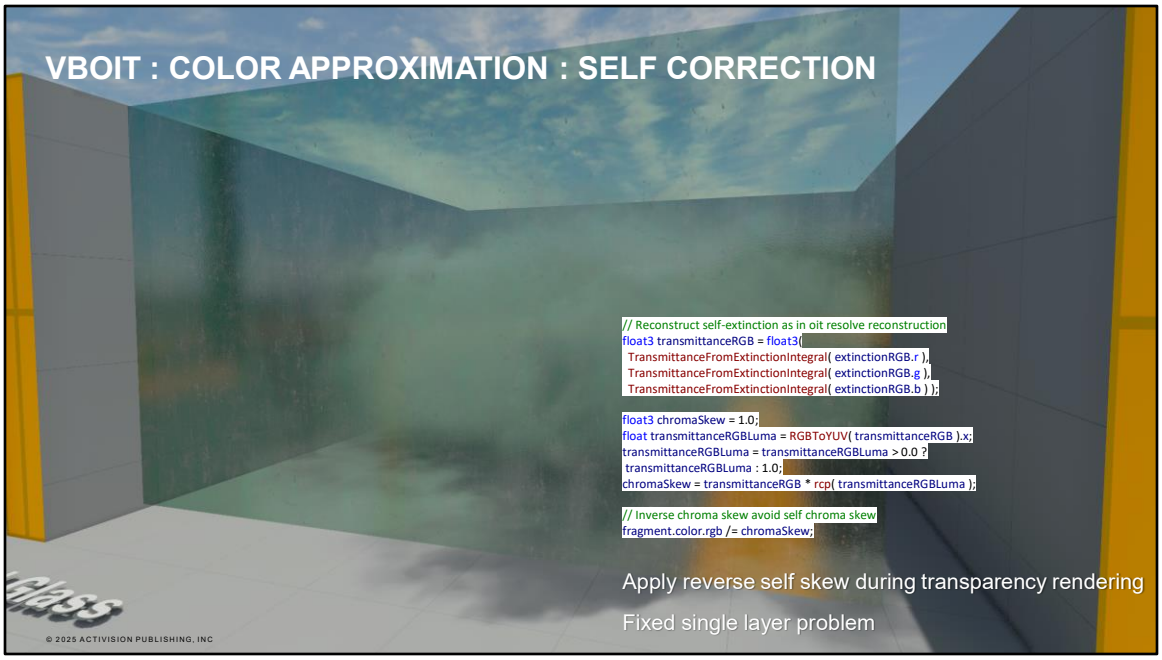
Weight chromaticity by total transmittance
Including volumetric fog



```
// Transparency draw
if( OitEnableChromaTransSkew() )
{
  extinctionRGB = lerp( extinction, extinctionRGB, transmittanceIntegral *
FogTransmittanceIntegral );
}
```



During resolve accumulated transmittance chromaticity will be applied to pixels – thus all surfaces rendered without any depth / surface separation.
While this is unavoidable, there are some things we can do to improve the first event.



During transparency rendering we can calculate future skew, and apply its reverse to pixels being rendered. This will effectively fix the event and partially correct others – although it will become a frequency soup.


SIGGRAPH
2025