30 JULY – 3 AUGUST Los Angeles
SIGGRAPH 2017

Improved Culling for Tiled and Clustered Rendering

CALL OF DUTY INFINITE WARFARE

Michal Drobot

Principal Rendering Engineer

infinity ward ACTIVISION


CALL OF DUTY
INFINITE WARFARE

Plus(+) Methods: Introduction

Plus(+) Methods: Algorithm Steps

- List of rendering entities
- Spatial acceleration structure with culled entity lists
- Execution algorithm per sampling point
  - Traverse acceleration structure
  - Iterate over existing entities

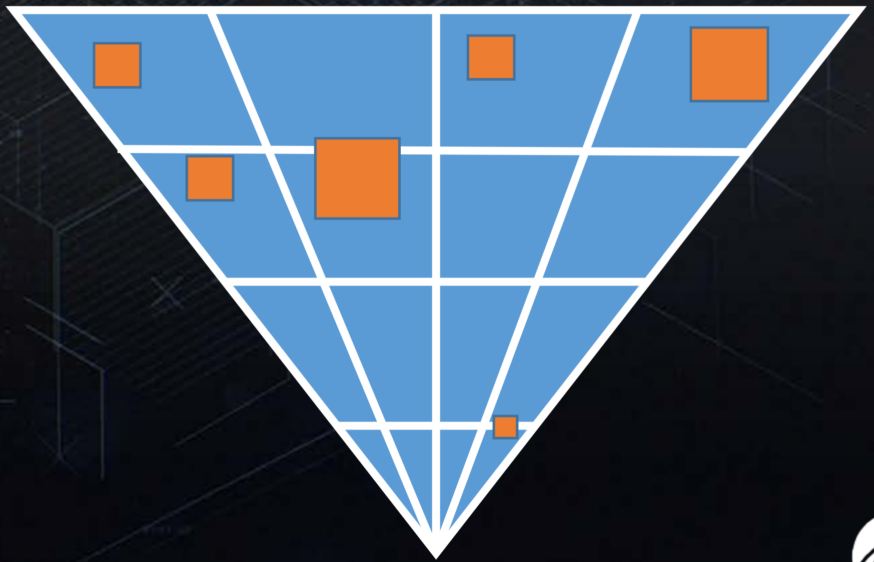
- Also known as Tiled / Clustered Forward+/Deferred+

Spatial acceleration structure : Frustum

- Tile
  - Depth pre-pass
  - Cull entities against tile boundaries and depth min / max
  - 2D dense grid
    - i.e. 8x8 pixel tiles
  - Suffers from depth discontinuities

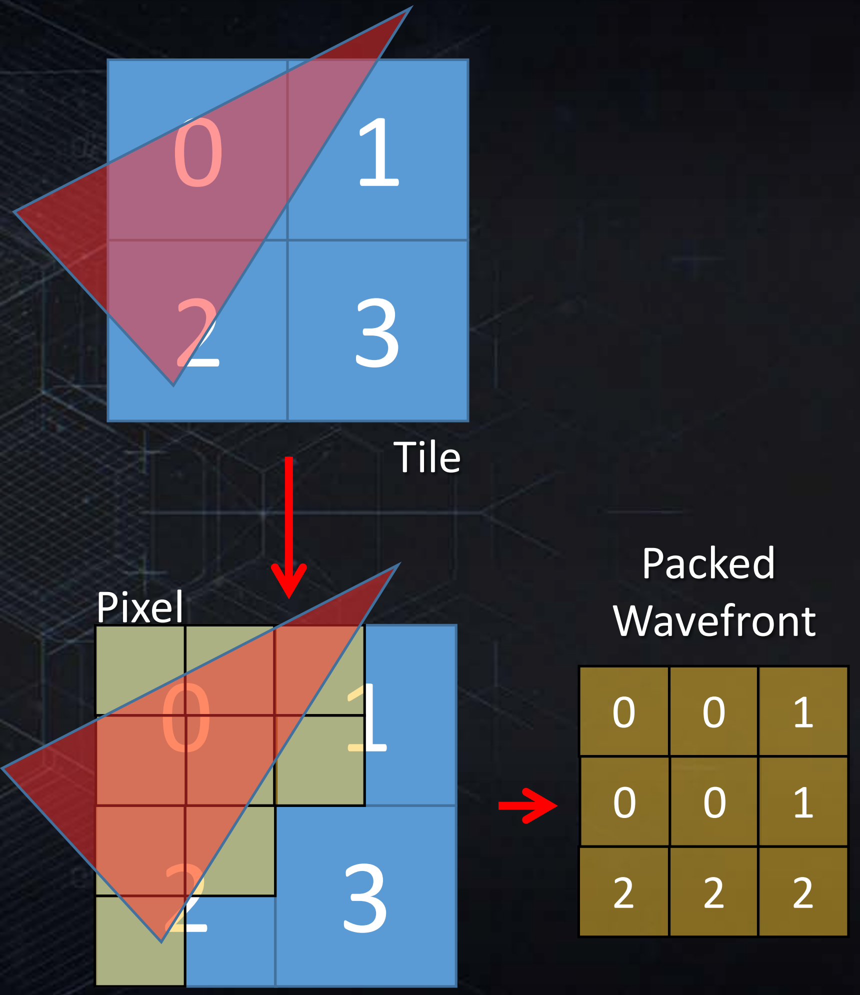
Spatial acceleration structure : Frustum

- Cluster
  - No need for depth pre-pass - cull entities against 3D clusters
  - Allows arbitrary 3D (within frustum) lookup
  - Depth discontinuities mitigated as much as memory consumption allows
    - Z – slice distribution
  - Likely sparse in 2D due to memory consumption



Data Structure Divergence performance issues

- Tile
  - In Forward a triangle can cross multiple tiles
  - In Deferred wavefront can be matched to tile size
- Cluster
  - In Forward a triangle can cross multiple clusters
  - In Deferred a wavefront can cross multiple Z slices
- Voxel Tree
  - Wavefront can cross multiple voxels
- Triangle / Texel
  - Wavefront can cross multiple triangles / texels



Divergence performance issues

- High VMEM and VALU cost
  - All calculations past divergence point (i.e. within tile) happen per vector
  - All memory loads, even if coherent, will happen per vector thus spamming TCC units
  - Memory arithmetic will happen per vector adding to VALU cost
- High VGPR cost
  - Because all operations are vector based, all constant data (such as entity descriptors) will have to be loaded into VGPRs

Data Structures & Scalarization

Scalarization

- Execute wavefront only on one divergent item at a time
- Loop over all items sampled by wavefront
  - Mask all wavefront lanes to work only on selected item
  - Move to next
- Can be used in Forward as well as in Deferred

Data Container : Hierarchical

- Pointer to leaf cluster
  - Leaf cluster stores all pointers to visible entities inside each entry
  - Not bound by entity number
  - Expensive traversal due to indirection
  - Variable memory storage cost

Global	0	1	2	3	4	5	6	7	…	N

Leaf payload
(same leaves should be merged)

Entity Idx	0	1
Pointer	5	7

2D/3D Lookup → **Spatial Acceleration Structure with leaf pointers** →

Leaf Header	
Head Ptr	Entity # : 2

Execute on entities 5 & 7

```
// Typical Hierarchical Container iterator
address = containerAddressFromSamplePosition( samplePosition ); // Find address of container for 2D / 3D position

// First entry is the header. Read and advance pointer
header = g_EntityClusterData[address++];
entityCount = GetEntityCountFromHeader( header );

// Iterate over entities inside the container
for ( entityItr = 0; entityItr < entityCount; entityItr++ )
      ProcessEntity( g_EntityClusterData[address++] );
```

```
// Hierarchical Container iterator scalarized over containers
address = containerAddressFromSamplePosition( samplePosition ); // Find address of container for 2D / 3D position
uniformAddress = address;
currentLaneID = WaveGetLaneIndex();
execMask = ~ulong( 0 ); // init mask to 111…111 - open for all lanes
ulong currentLaneMask = ulong( ulong( 1 ) << ulong( currentLaneID ) );
while ( ( execMask & currentLaneMask ) != 0 ) // set EXEC to remaining lanes
{
     uniformAddress = WaveReadFirstLane( address );
     laneMask     = WaveBallot( uniformAddress == address ); // mask of lanes to be processed in current iteration
     execMask    &= ~laneMask; // remove currently alive lanes from mask
     if ( uniformAddress == address ) // execute for lanes with matching coherent address
     {
               header = g_EntityClusterData[uniformAddress++]; // First entry is the header. Read and advance pointer
               entityCount = GetEntityCountFromHeader( header );
               // Iterate over entities inside the container
               for ( entityItr = 0; entityItr < entityCount; entityItr++ )
                         ProcessEntity( g_EntityClusterData[uniformAddress++] );
     }
}
```

Scalarization : Hierarchical Container

- Shader is fully scalarized
  - Better VMEM/SMEM and VALU/SALU balancing
  - Low VGPR usage (similar to Forward constant loading)

- Performance highly variable
  - Wavefront can process entities multiple times if same entities are in different containers
  - Depending on data coherency / redundancy this can result in slowdown

- Ideally scalarize on entity level
  - This requires ordered containers - Flat Bit Arrays

Data Container: Flat

- Flat Bit Array
  - Collection of bits – Nth bit representing visibility of Nth entity from global list
  - Simple traversal – iterate through bits
  - Memory / Entity bound – mostly used in per-frustum context

Global	0	1	2	3	4	5	6	7	…	N

Per-frustum culled list of M entities

Store	3	5	7	12
Idx	0	1	2	3

2D/3D Lookup → **Spatial Acceleration Structure with M bit entries** →

0	1	1	0

Execute on entities 5 & 7

```
// Typical Flat Bit Array iterator
wordMin = 0;
wordMax = max( MAX_WORDS - 1, 0 );
address = containerAddressFromSamplePosition( samplePosition );
// Read range of words of visibility bits
for ( uint wordIndex = wordMin; wordIndex <= wordMax; wordIndex++ )
{
    // Load bit mask data per lane
    mask = entityMasksTile[address + wordIndex];
    while ( mask != 0 ) // processed per lane
    {
        bitIndex = firstbitlow( mask );
        entityIndex = 32 * wordIndex + bitIndex;
        mask ^= ( 1 << bitIndex );
        ProcessEntity( entityIndex );
    }
}
```

```
// Flat Bit Array iterator scalarized on entity
wordMin = 0;
wordMax = max( MAX_WORDS - 1, 0 );
address = containerAddressFromSamplePosition( samplePosition );
// Read range of words of visibility bits
for ( uint wordIndex = wordMin; wordIndex <= wordMax; wordIndex++ )
{
     // Load bit mask data per lane
     mask = entityMasksTile[address + wordIndex];
     // Compact word bitmask over all lanes in wavefront
     mergedMask = WaveReadFirstLane( WaveAllBitOr( mask ) );
     while ( mergedMask != 0 ) // processed scalar over merged bitmask
     {
               bitIndex = firstbitlow( mergedMask );
               entityIndex = 32 * wordIndex + bitIndex;
               mergedMask ^= ( 1 << bitIndex );
               ProcessEntity( entityIndex );
     }
}
```

Scalarization : Flat Container

- Bitmask Scalarization
  - Shader is fully scalarized and executes once per each entity
  - Low VGPR usage
  - Significantly more efficient than baseline
  - Arguably more elegant code
- Results of scalarization on synthetic test shader
  - Scalarization applied to Light lookups in densely lit environment

	VGPR Count	Occupancy	Cost
Base F+	56	5	100%
Hierarchical F+ Scalarized	32	8	98%
Flat F+ Scalarized	32	8	72%

Scalarization

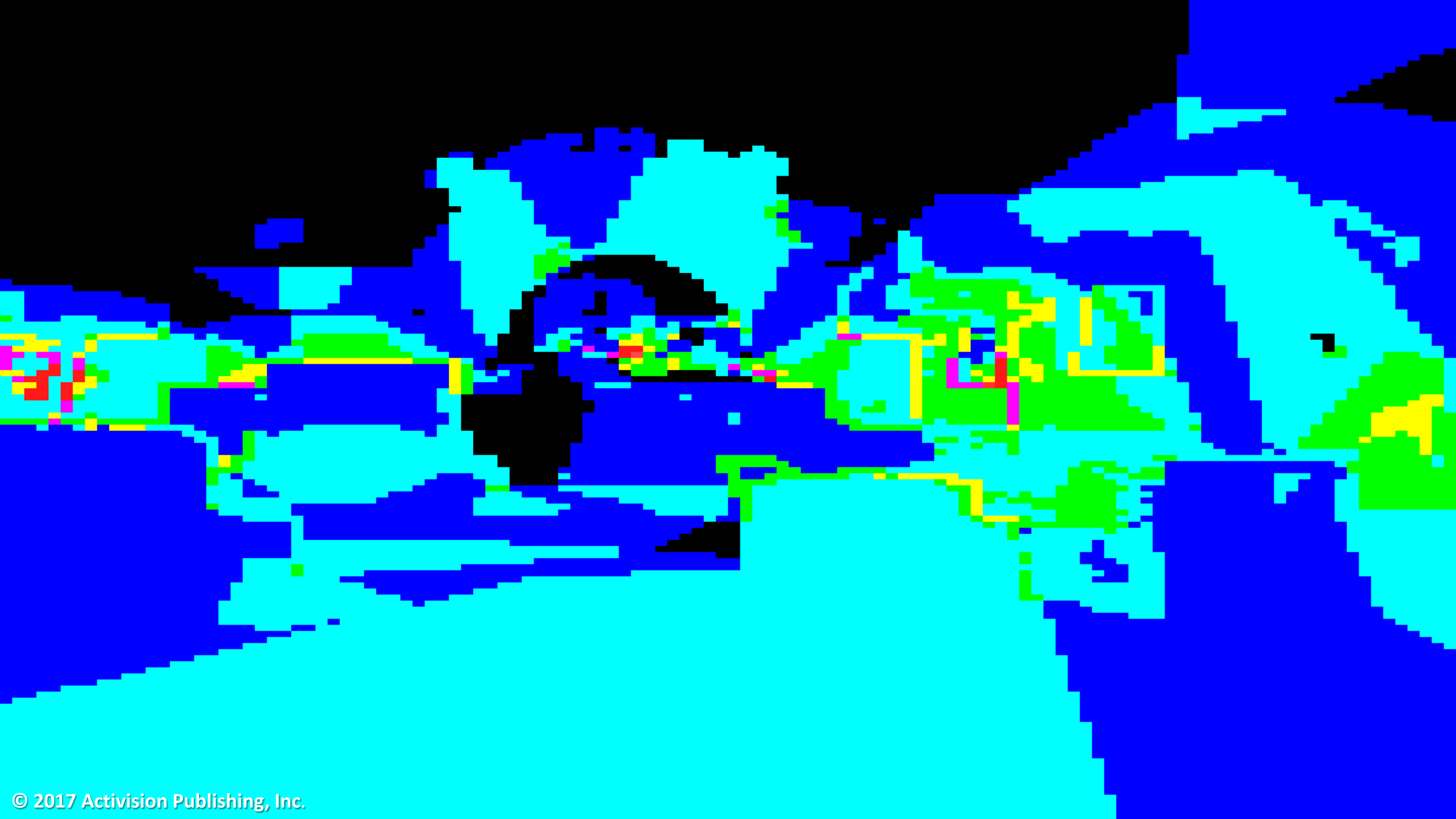
- Hierarchical Data Containers can be scalarized on container level
  - Tile / Cluster / Voxel address
- Flat Data Containers can be scalarized on stored entity level
  - Light / Probe / Decal idx

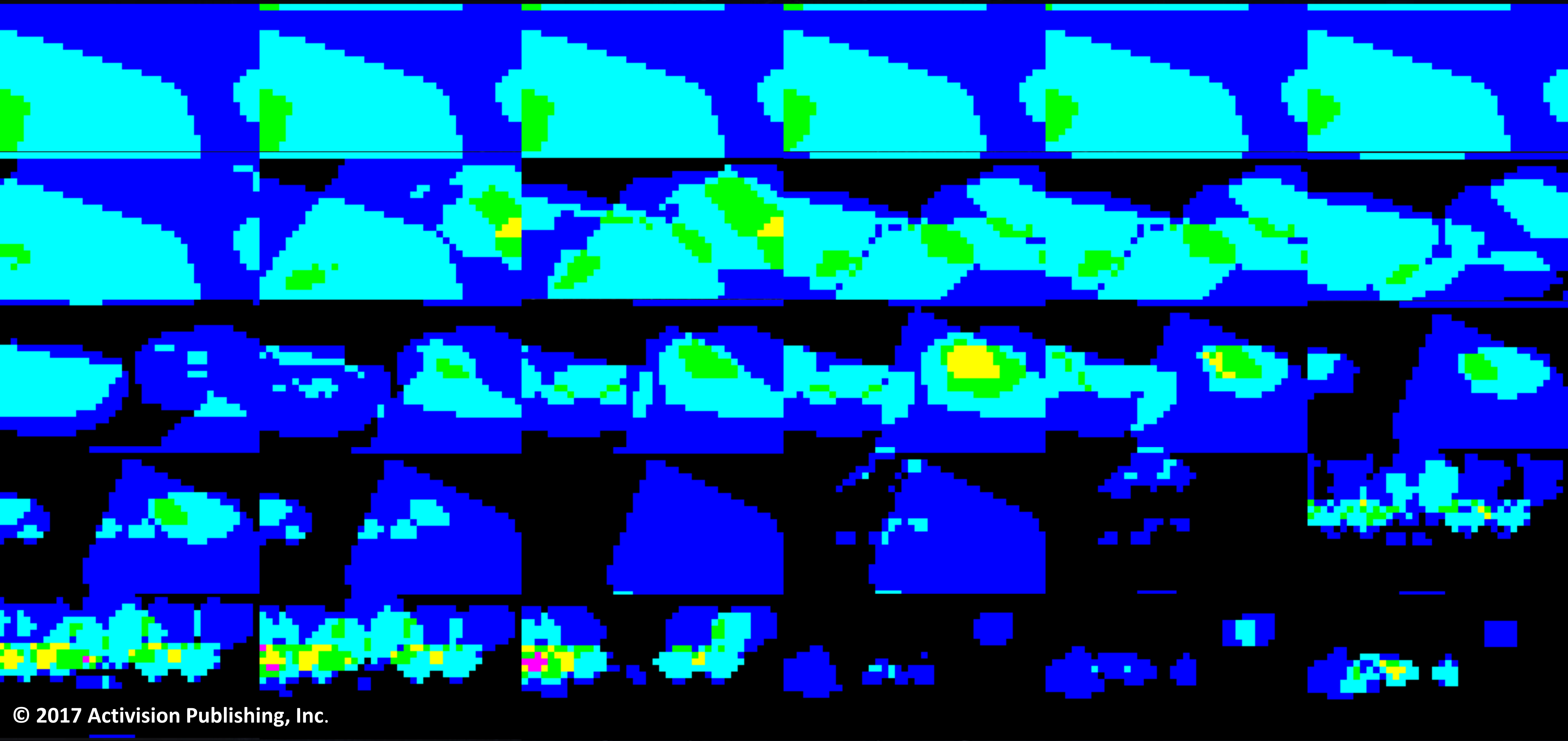
Z-Binning

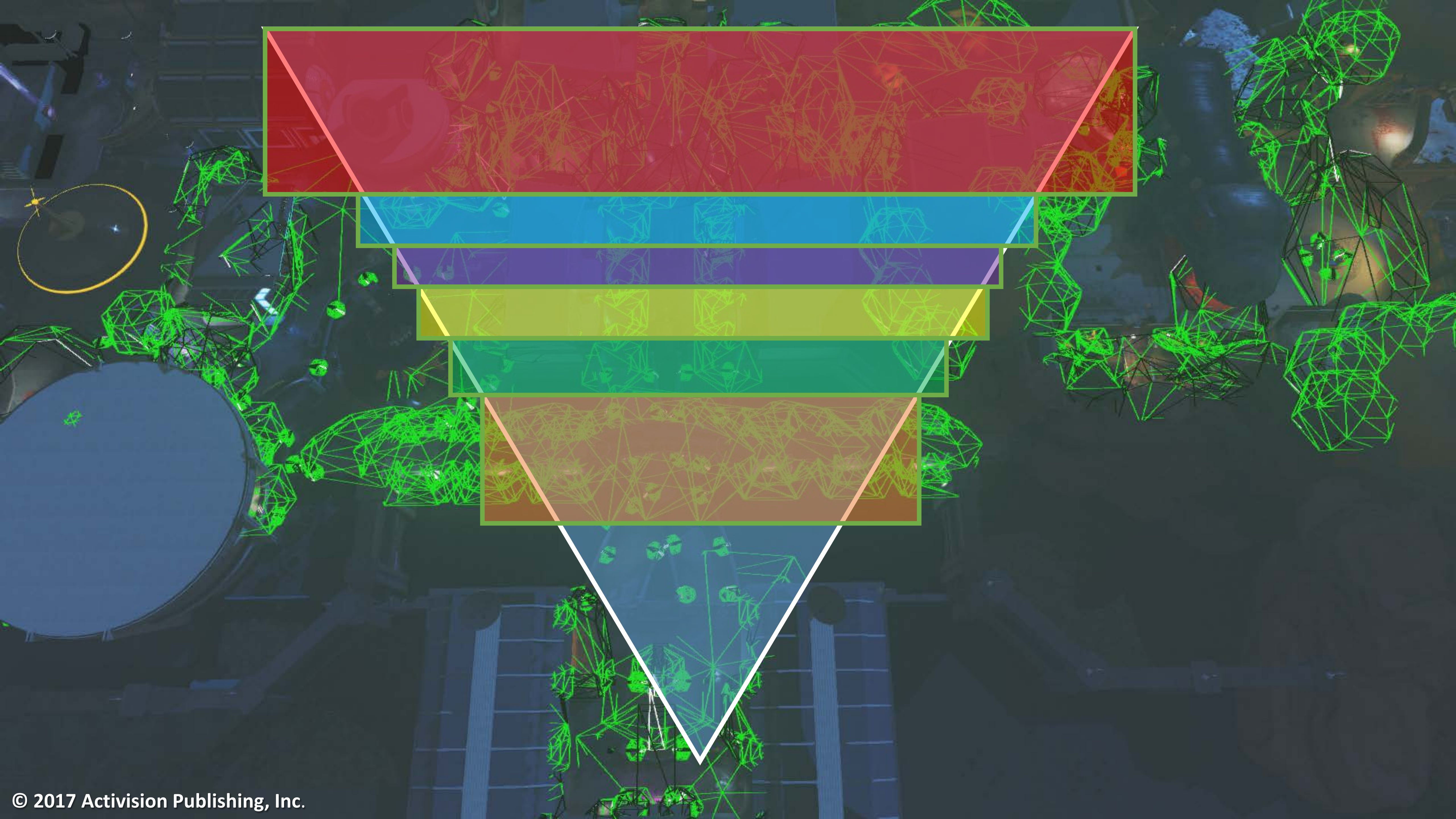
Z-Binning

	Depth Discontinuities	Spatial Resolution	Memory Scaling with Z
Tiles	-	+	+
Clusters	+	-	-

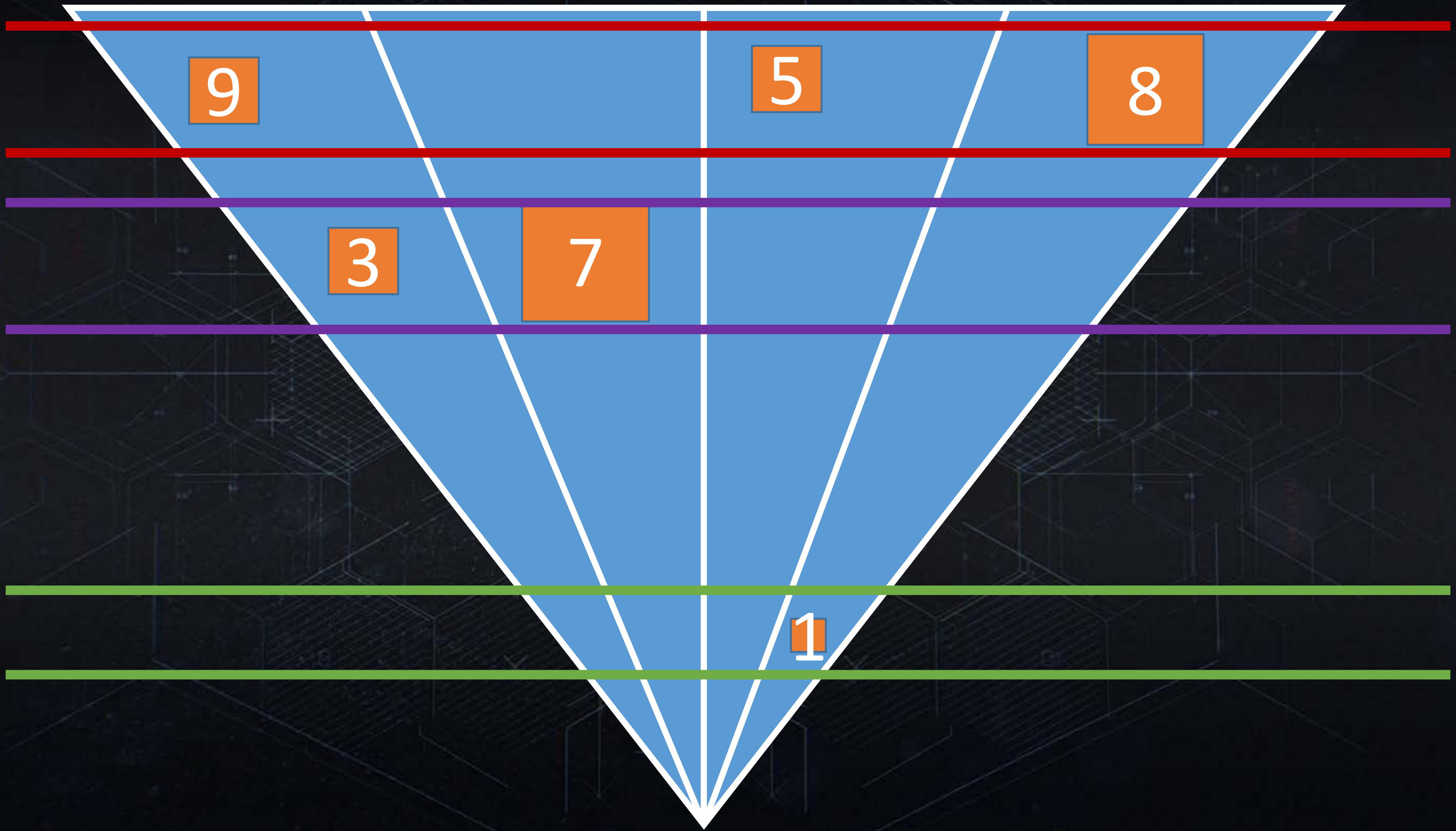
- Open world depth ranges
  - Clustering performance / memory impractical for high efficiency at high depth complexity



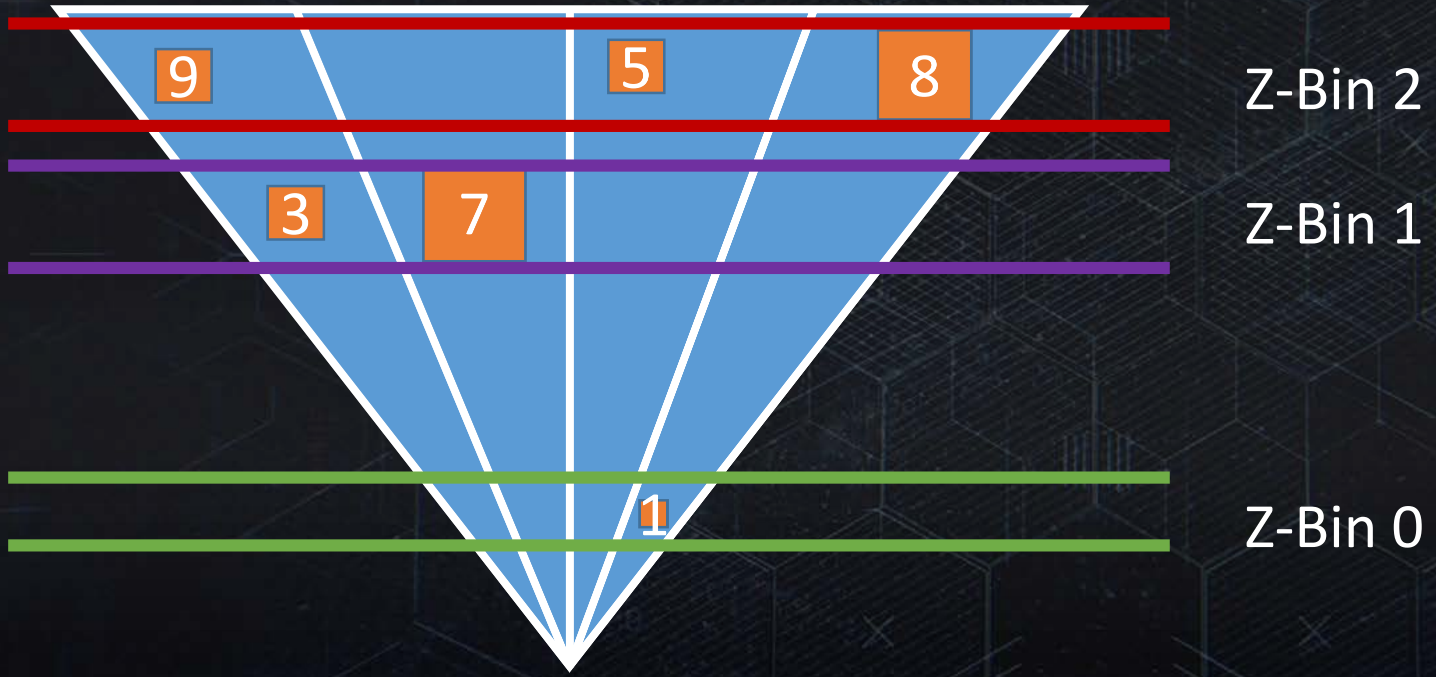




Z-Binning

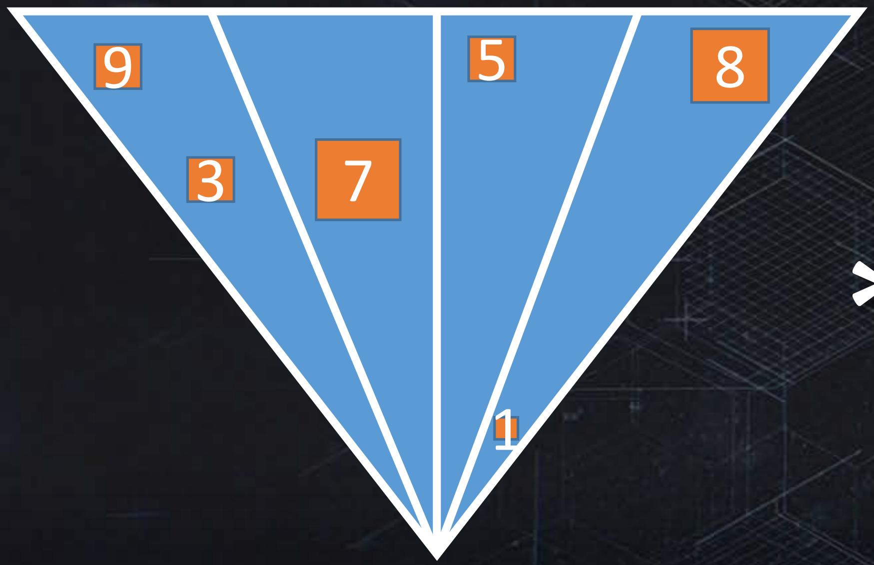


Z-Binning



Z-Bin Idx	Content
2	9 | 5 | 8
1	3 | 7
0	1

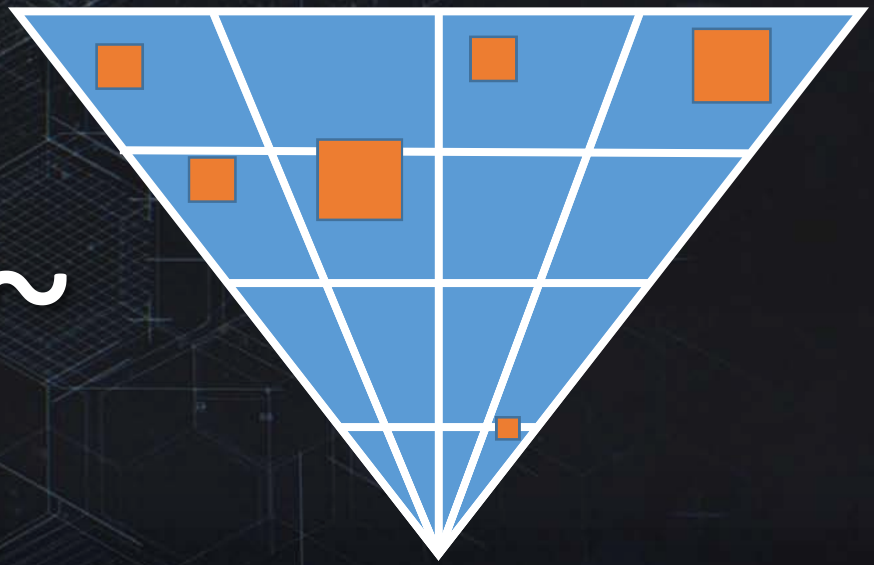
Z-Binning



*

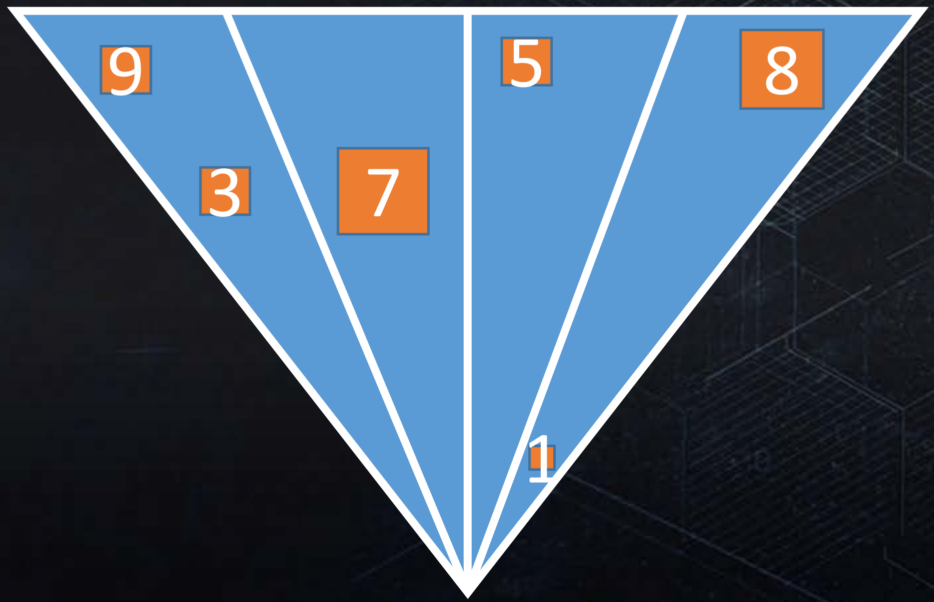
Z-Bin Idx	Content
2	9 | 5 | 8
1	3 | 7
0	1

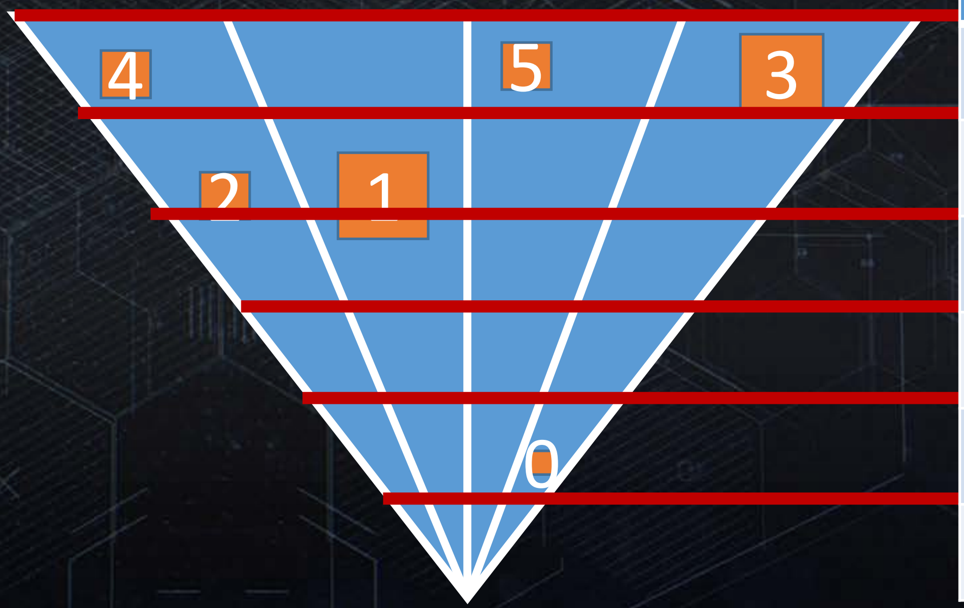
~



Z-Binning : Efficient LUT

```
ZBIN[ int( LinearZ / BIN_WIDTH ) ] = MIN_LIGHT_ID_IN_BIN | MAX_LIGHT_ID_IN_BIN
```





Idx	Data
5	3|5
4	1|2
3	1|1
2	MAX|0
1	0|0
0	MAX|0

F+ Renderer : Z-binning algorithm

- CPU:
  - Sort lights by Z
  - Set uniformly distributed bins over total possible depth range
  - Generate 2 x 16bit LUT with MIN / MAX light ID within each bin boundaries
- GPU (PS / CS):
  - Vector load ZBIN
  - Wave uniform LIGHT MIN / MAX ID
  - Wave uniform LOAD of light bit WORDS from MIN / MAX range
  - Create Vector bit mask from LIGHT MIN / MAX ID
  - Mask uniform lights by vector Z-Bin mask

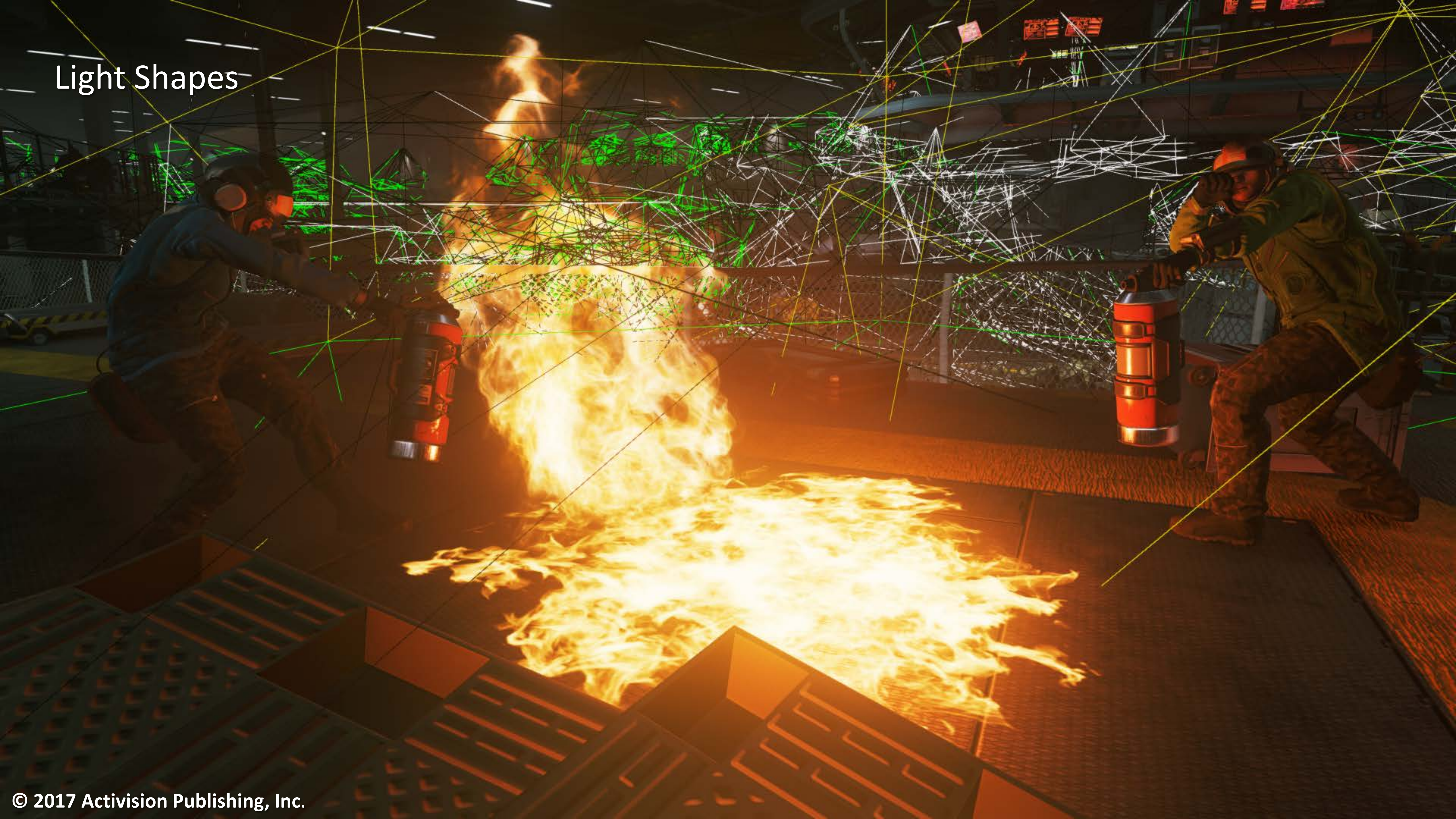
```
// Flat Bit Array iterator scalarized on entity with Z-Bin masked words
wordMin = 0;
wordMax = max( MAX_WORDS - 1, 0 );
address = containerAddressFromScreenPosition( screenCoords.xy );

zbinAddr = ContainerZBinScreenPosition( screenCoords.z );
zbinData = maskZBin.TypedLoad( zbinAddr, TYPEMASK_NUM_DATA( FORMAT_NUMERICAL_UINT, FORMAT_DATA_16_16 ) );
minIdx = zbinData.x;
maxIdx = zbinData.y;
mergedMin = WaveReadFirstLane( WaveAllMin( minIdx ) ); // mergedMin scalar from this point
mergedMax = WaveReadFirstLane( WaveAllMax( maxIdx ) ); // mergedMax scalar from this point
wordMin = max( mergedLightMin / 32, wordMin );
wordMax = min( mergedLightMax / 32, wordMax );

// Read range of words of visibility bits
for ( uint wordIndex = wordMin; wordIndex <= wordMax; wordIndex++ )
{
     // … //
}
```

```
// Read range of words of visibility bits
for ( uint wordIndex = wordMin; wordIndex <= wordMax; wordIndex++ )
{
    // Load bit mask data per lane
    mask = entityMasksTile[address + wordIndex];
    // Mask by ZBin mask
    uint localMin  = clamp( (int)minIdx - (int)( wordIndex * 32 ), 0, 31 );
    uint maskWidth = clamp( (int)maxIdx - (int)minIdx + 1, 0, 32 );
    // BitFieldMask op needs manual 32 size wrap support
    uint zbinMask  = maskWidth == 32 ? (uint)(0xFFFFFFFF)  : BitFieldMask( maskWidth, localMin );
    mask &= zbinMask;
    // Compact word bitmask over all lanes in wavefront
    mergedMask = WaveReadFirstLane( WaveAllBitOr ( mask ) );
    while ( mergedMask != 0 ) // processed scalar over merged bitmask
    {
        bitIndex = firstbitlow( mergedMask );
        entityIndex = 32 * wordIndex + bitIndex;
        mergedMask ^= ( 1 << bitIndex );
        ProcessEntity( entityIndex );
    }
}
```

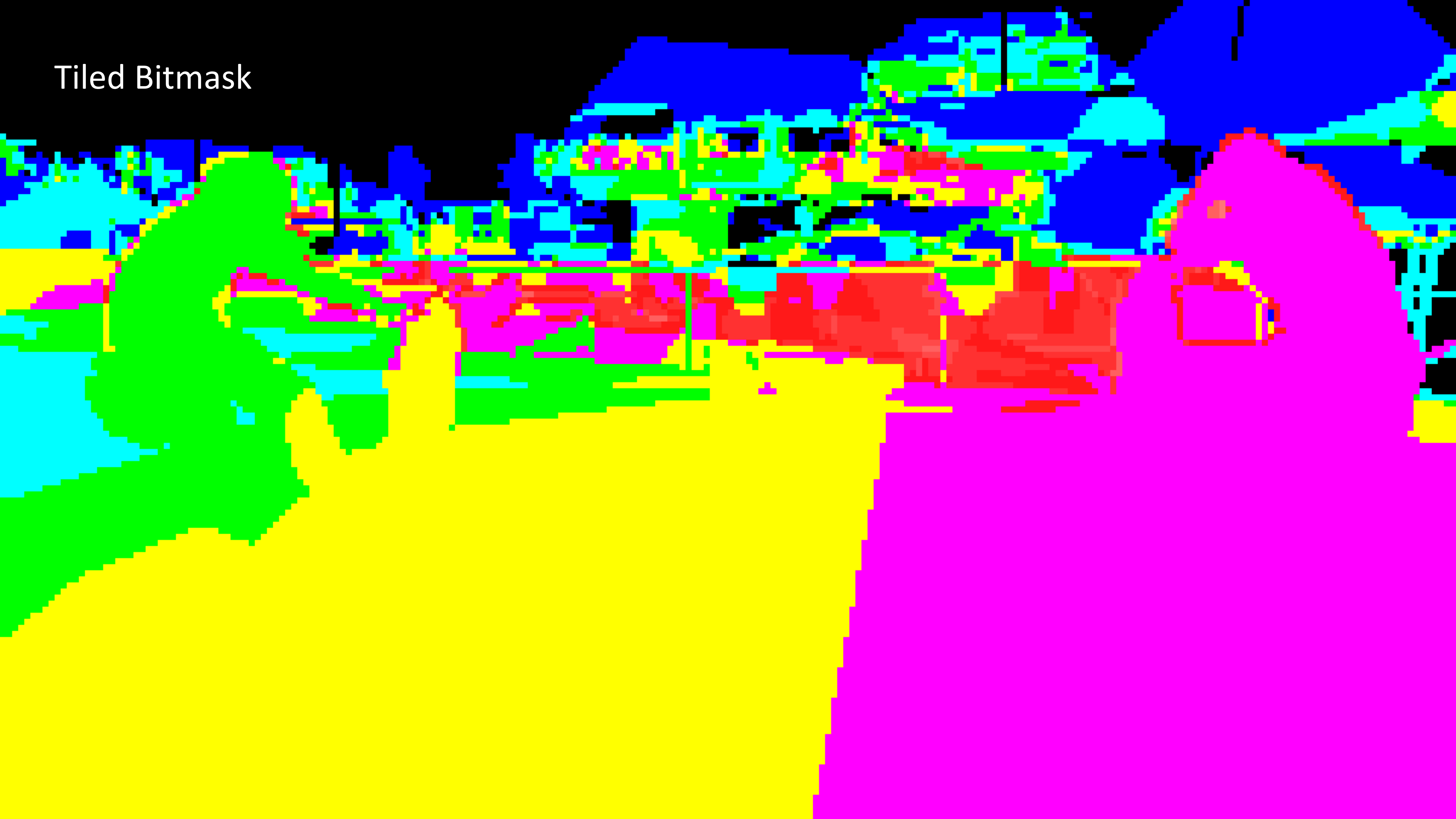
F+ Renderer : Memory Performance

Structure (256 bit array)	XY Resolution	Z Resolution	Complexity	Storage	Cull Operations Per Entity
Tiled Buffer	240 x 135	1	O(X * Y)	1,036 KB	32400
Tiled + ZBin	240 x 135	8096	O(X * Y + Z)	1,036 KB + 32 KB	32400 + 8096 (simple)
Clustered	60 x 32	18	O(X * Y * Z)	1,106 KB	34560

Light Shapes



Tiled Bitmask



Effective Light Evaluations


Z-Binning Enabled

F+ Renderer : Z-Bin Performance

Hangar Fire Scene (PS4 1080p)	
Structure (256 bit array)	**Opaque render time**
Tiled Buffer	9.00 ms
Tiled + Zbin	7.65 ms (15%)

F+ Renderer : Optimization Performance

Zombies opening scene (PS4 1080p)		
Structure (256 bit array)	**PS Opaque pass time**	**PS Opaque pass average occupancy**
Base Tile	5.7 ms (100%)	~3
Based Tile + ZBin	5.2 ms (91%)	~3
Scalarized Tile	5.1 ms (88%)	~4.3
Scalarized Tile + ZBin	4.6 ms (80%)	~4.3

Rasterization based culling

Classic compute culling issues

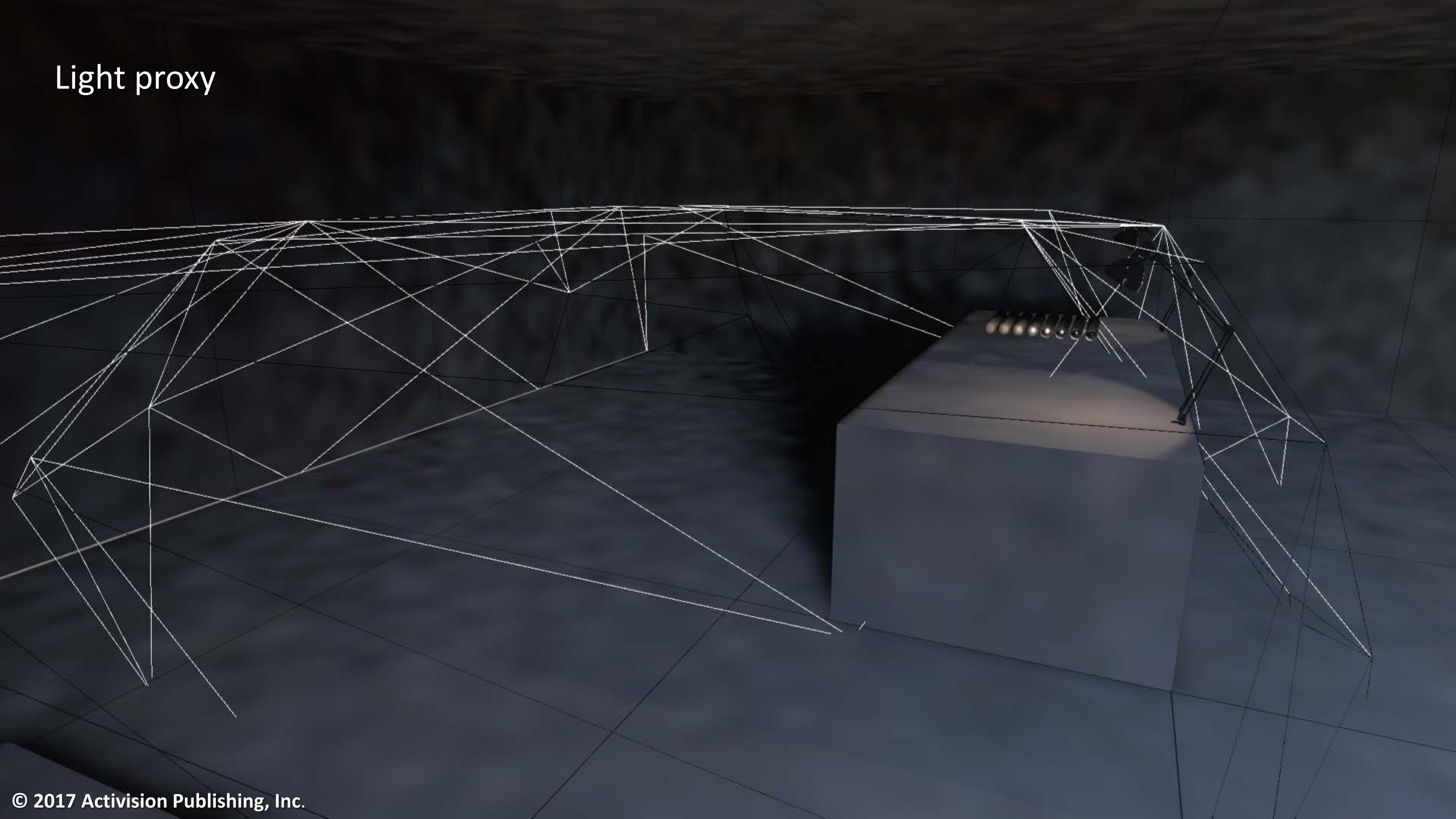
- Accuracy [WRO17]
- Occlusion [KAS11]
- Complex shapes [HEI16]

Solution?

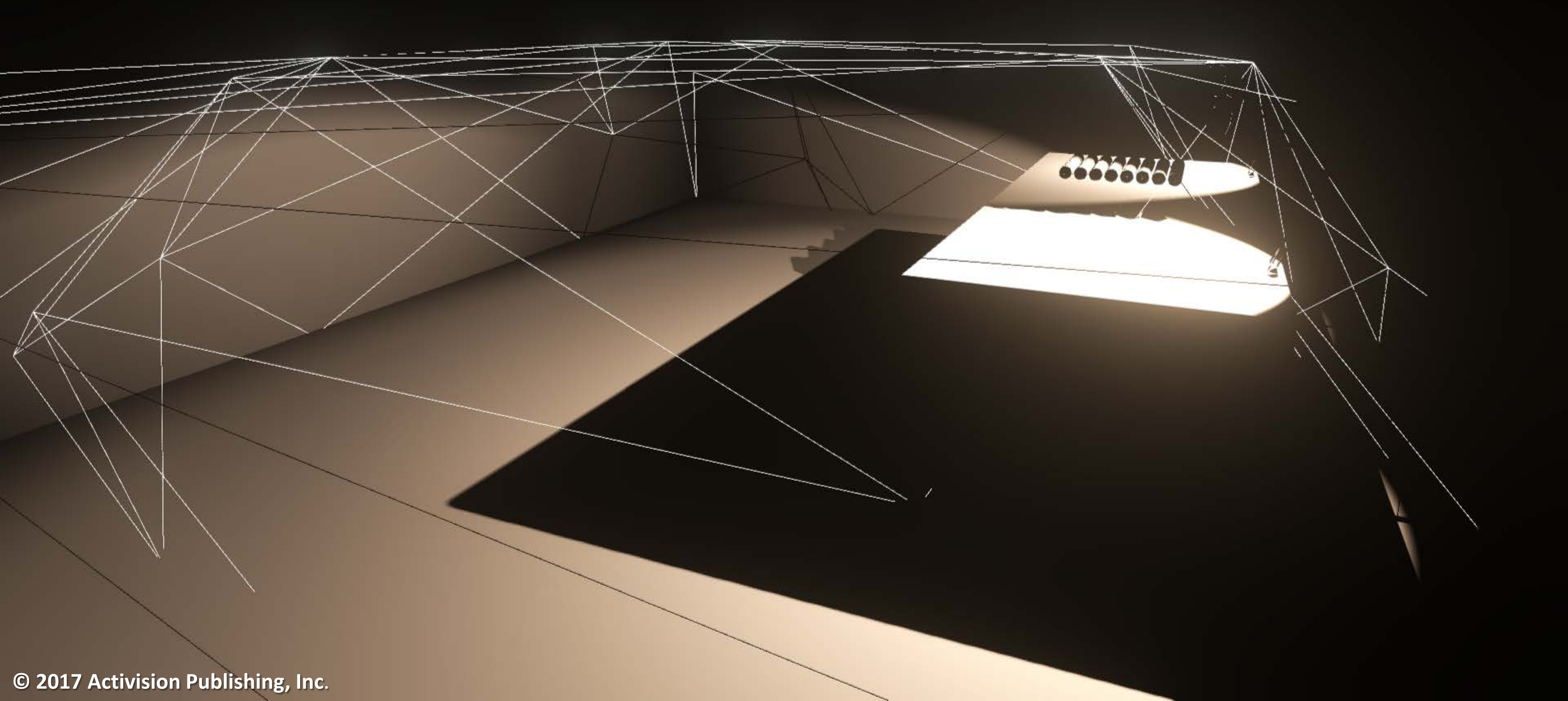
Image courtesy of Eric Heiz

Classic Rasterized Deferred Rendering [HAR04]

Light proxy



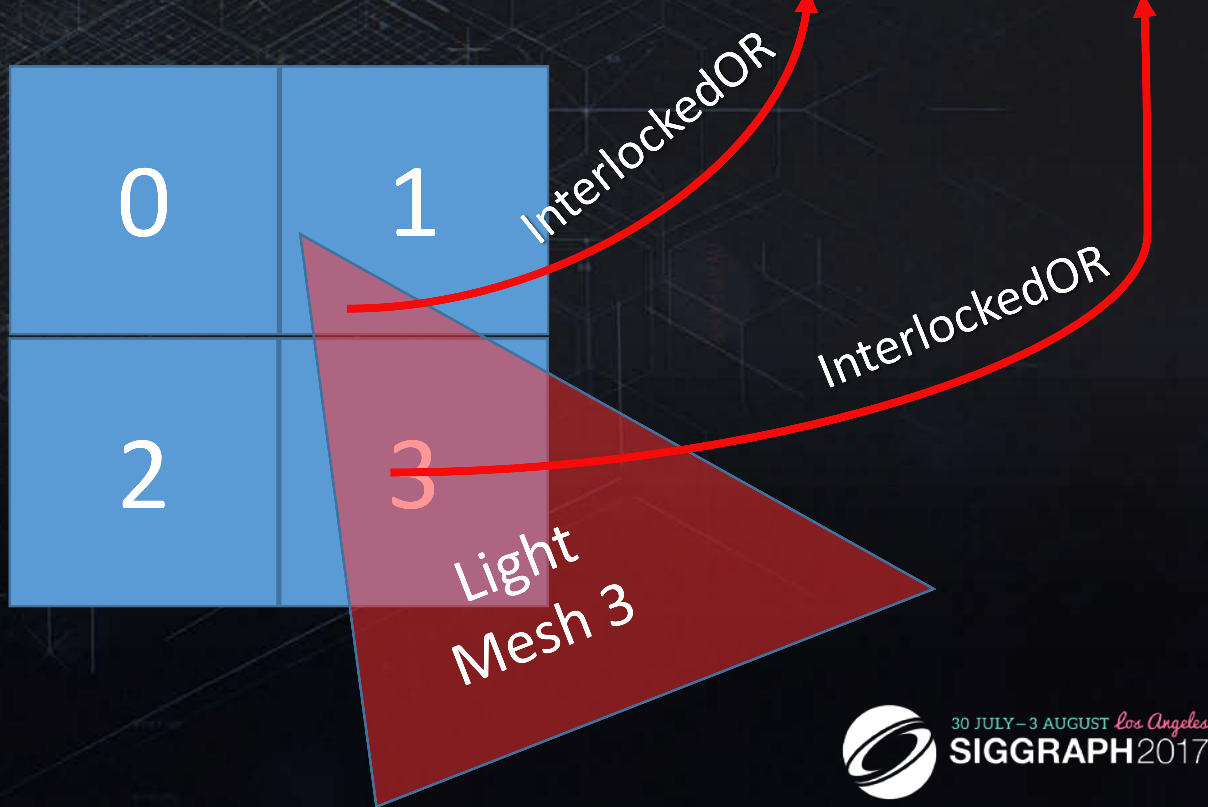
Light proxy
Direct light only



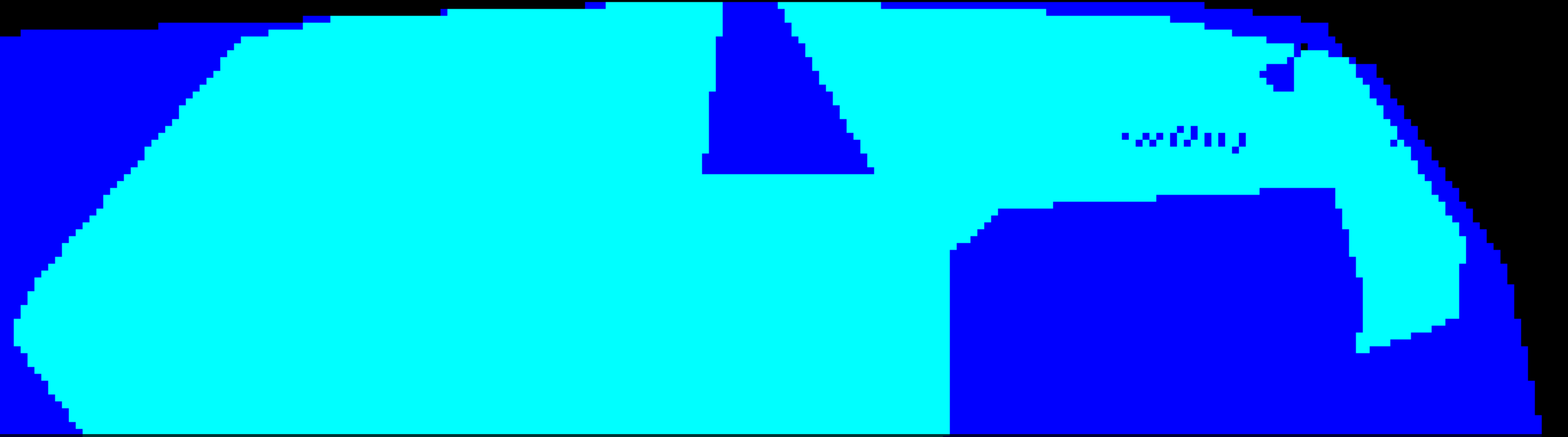
Conservative Rasterized Culling

- Light is a mesh (Light Proxy)
- Atomic writes to tiles / clusters
  - InterlockedOR lightBit bit for Flat Bit Array
- Conservative Rasterization w/o Hw support
  - Rasterize at full resolution
  - 4xMSAA for speedup [DRO17]
- Fixed pipeline HW optimizations
  - EarlyZ (HiZ)
  - Depth Bounds Test (HiZ)
  - Stencil (HiS)

Tile	0	1	2	3
Lights	0000	0010	0000	0010

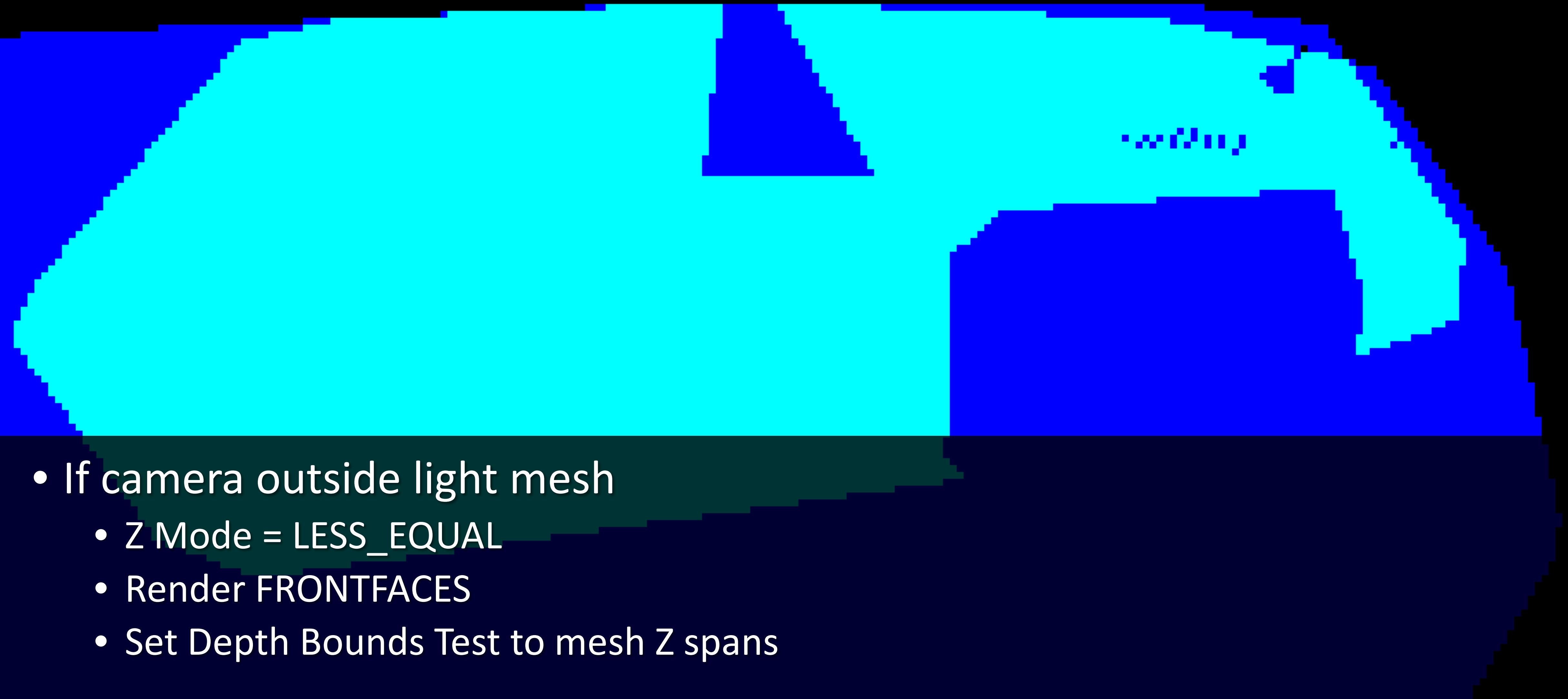


Tile Rasterization in 1 pass
EarlyZ + Depth Bounds Test
8x8 pixel tiles



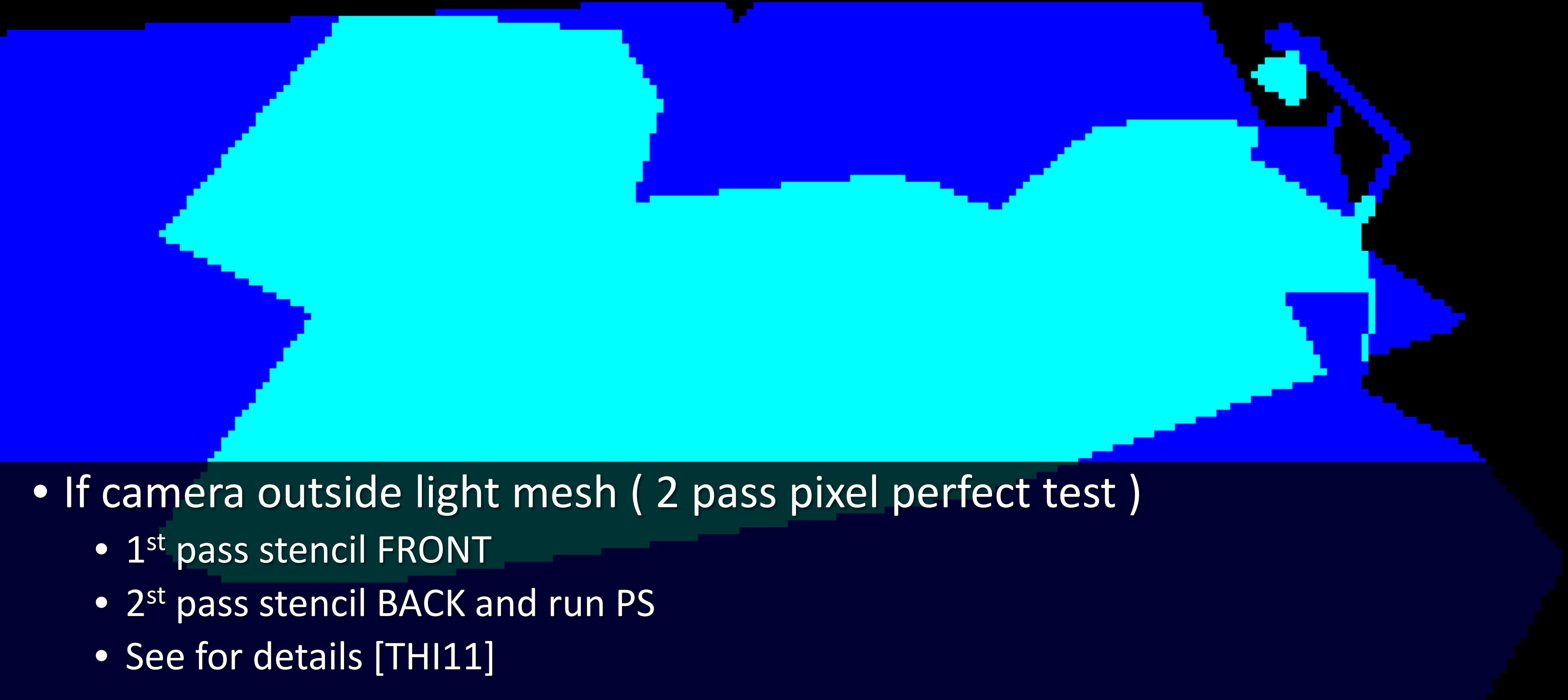
- Pixels inside light mesh volume run PS and InterlockOR lighBit to tiles
- If camera inside light mesh
  - Z Mode = Greater
  - Render BACKFACES

Tile Rasterization in 1 pass
EarlyZ + Depth Bounds Test
8x8 pixel tiles



- If camera outside light mesh
  - Z Mode = LESS_EQUAL
  - Render FRONTFACES
  - Set Depth Bounds Test to mesh Z spans

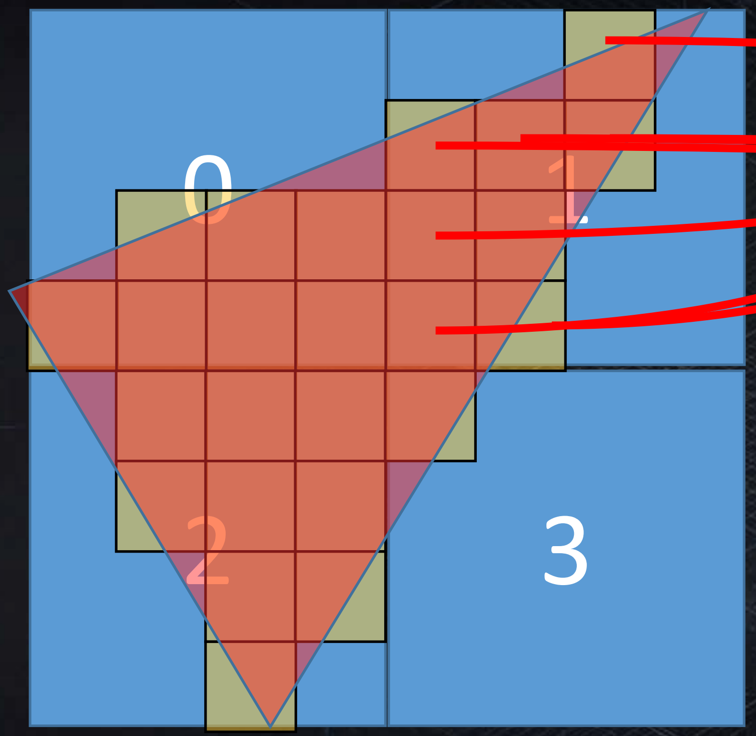
Tile Rasterization in 2 passes
EarlyZ + Depth Bounds Test + Stencil
8x8 pixel tiles



- If camera outside light mesh (2 pass pixel perfect test)
  - 1st pass stencil FRONT
  - 2st pass stencil BACK and run PS
  - See for details [THI11]

```
// Used on Flat Bit Array
[earlydepthstencil] // make sure to enable EarlyZ with UAV
void ps_main( const PixelInput pixel )
{
    uint tileIndex = FrustumGrid_TileFromScreenPos( rasterizerScale * pixel.position.xy ); // scale to MSAA raster
    uint lightIndex = uint( entityID.x );
    const uint lightBit = 1 << ( lightIndex % 32 ); // find correct light bit
    const uint word = lightIndex / 32;
    const uint wordIndex = ( tileIndex * FRUSTUM_GRID_FRAME_WORDS_LIGHTS ) + word; // find correct word

    {
            InterlockedOr( lightMasksTile[wordIndex], lightBit ); // update light bit in correct word
    }
}
```

Atomic Contention



Tile	0	1	2	3	Total
AtomicOR Operations	7	8	9	1	25

Only need 4 operations to tag all tiles

Need to prune Atomics on Tile Id

```
uint WaveCompactValue( uint checkValue )
{
    ulong mask; // lane unique compaction mask
    for ( ; ; ) // Loop until all active lanes removed
    {
        uint firstValue = WaveReadFirstLane( checkValue );
        mask = WaveBallot( firstValue == checkValue ); // mask is only updated for remaining active lanes
        if ( firstValue == checkValue ) break; // exclude all lanes with firstValue from next iteration
    }
    // At this point, each lane of mask should contain a bit mask of all other lanes with the same value.
    uint index = WavePrefixCountBits( mask ); // Note this is performed independently on a different mask for each
lane.
    return index;
}
```

Lane	0	1	2	3	4	5	6	7
Active	y	y	y	y	y	y	y	y
index	0	0	1	1	2	0	3	1
mask	10011010	01100000	01100000	10011010	10011010	00000101	10011010	00000101

```
// Used on Flat Bit Array
[earlydepthstencil] // make sure to enable EarlyZ with UAV
void ps_main( const PixelInput pixel )
{
	uint tileIndex = FrustumGrid_TileFromScreenPos( rasterizerScale * pixel.position.xy ); // scale to MSAA raster
	uint lightIndex = uint( entityID.x );
	const uint lightBit = 1 << ( lightIndex % 32 ); // find correct light bit
	const uint word = lightIndex / 32;
	const uint wordIndex = ( tileIndex * FRUSTUM_GRID_FRAME_WORDS_LIGHTS ) + word; // find correct word

	const uint key  = ( wordIndex << FRUSTUM_GRID_MAX_LIGHTS_LOG2 );
	const uint hash = WaveCompactValue( key );

	[branch]
	if ( hash == 0 ) // Branch only for first occurrence of unique key within wavefront
	{
		InterlockedOr( lightMasksTile[wordIndex], lightBit ); // update light bit in correct word
	}
}
```

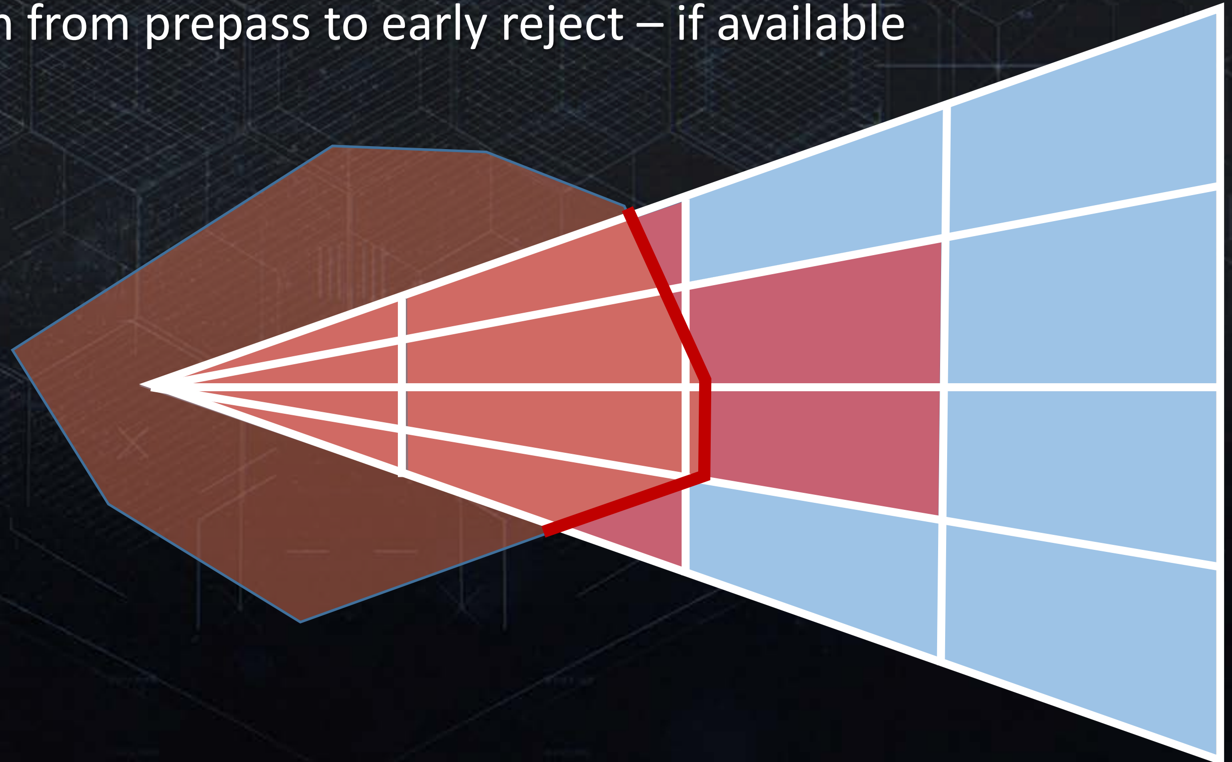
Tile Rasterizer Performance

3 x full screen lights - 240x135 - (PS4)	
Algorithm	**Culling time**
Atomic Raster	1.44 ms (100%)
Atomic Raster + 4xMSAA	0.27 ms (18%)
Atomic Raster + Compaction	0.36 ms (25%)
Atomic Raster + Compaction + 4xMSAA	0.10 ms (7%)

Most optimial variant starts to be setup bound, due to underutilization on per draw basis.
Needs batching.
Good candidate to use in parallel with Async Compute.

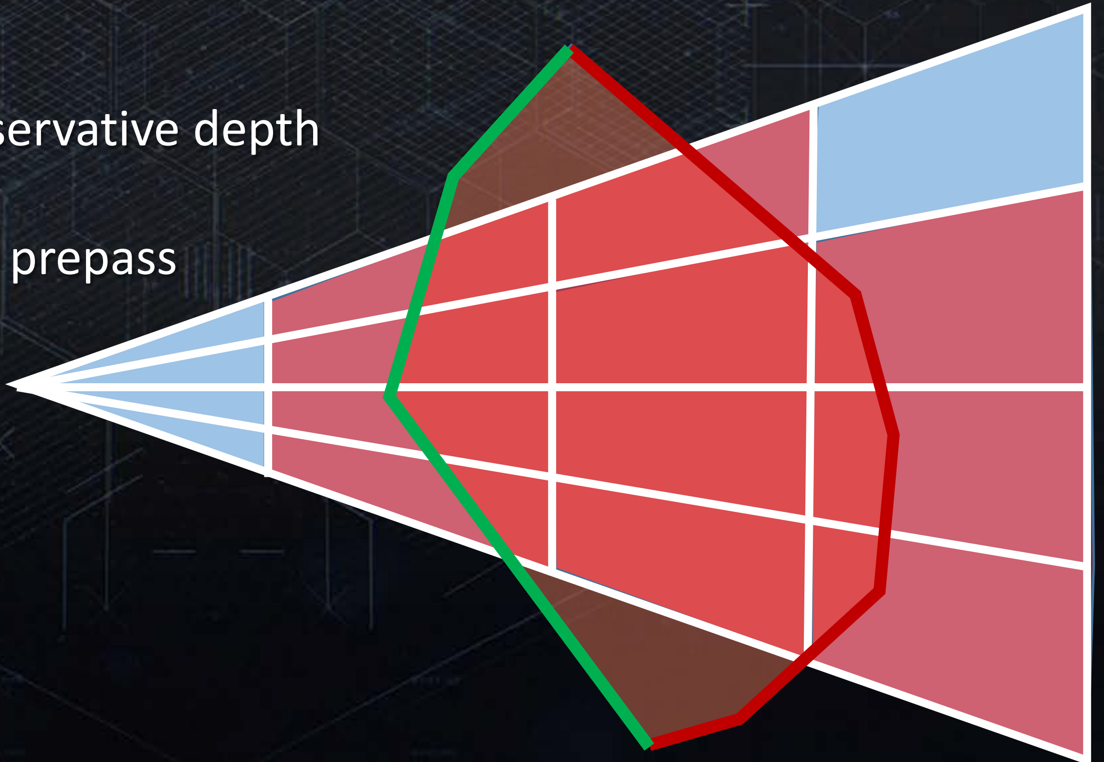
Cluster Rasterizer

- Conservatively mark lightBit in clusters within light mesh volume
- If camera inside light mesh (convex)
  - Render BACKFACES
  - Walk clusters in Z from near plane to conservative backface depth
    - Use conservative depth from prepass to early reject – if available



Cluster Rasterizer

- If camera outside light mesh (or inside non-convex)
  - 1s pass : MARK
    - Render BACKFACES
    - Mark clusters conservatively intersecting with BACKFACES
  - 2nd pass : WALK
    - Render FRONTFACE
    - Walk clusters from front conservative depth until hitting marked cluster
    - Use conservative depth from prepass to early reject – if available



Triangle Conservative Depth

- MARK/WALK requires conservative depth estimate for triangle

```
// Estimate triangle depth bounds from derivatives
// Derivatives will most likely trigger WQM! Make sure your lane wide operations behave correctly
float w = pixel.position.w;
float wDX = ddx_fine( w );
float wDY = ddy_fine( w );
tileMinW = max( tileMinW, w - abs( wDX ) - abs( wDY ) );
tileMaxW = min( tileMaxW, w + abs( wDX ) + abs( wDY ) );
// Estimate triangle depth bounds from triangle vertex depth check [DRO14]
// Conservatively cull in case of derivatives outside of triangle
float w0 = GetVertexParameterP0( pixel.posW );
float w1 = GetVertexParameterP1( pixel.posW );
float w2 = GetVertexParameterP2( pixel.posW );
tileMinW = max( tileMinW, min3( w0, w1, w2 ) );
tileMaxW = min( tileMaxW, max3( w0, w1, w2 ) );
```

Cluster Rasterizer Performance

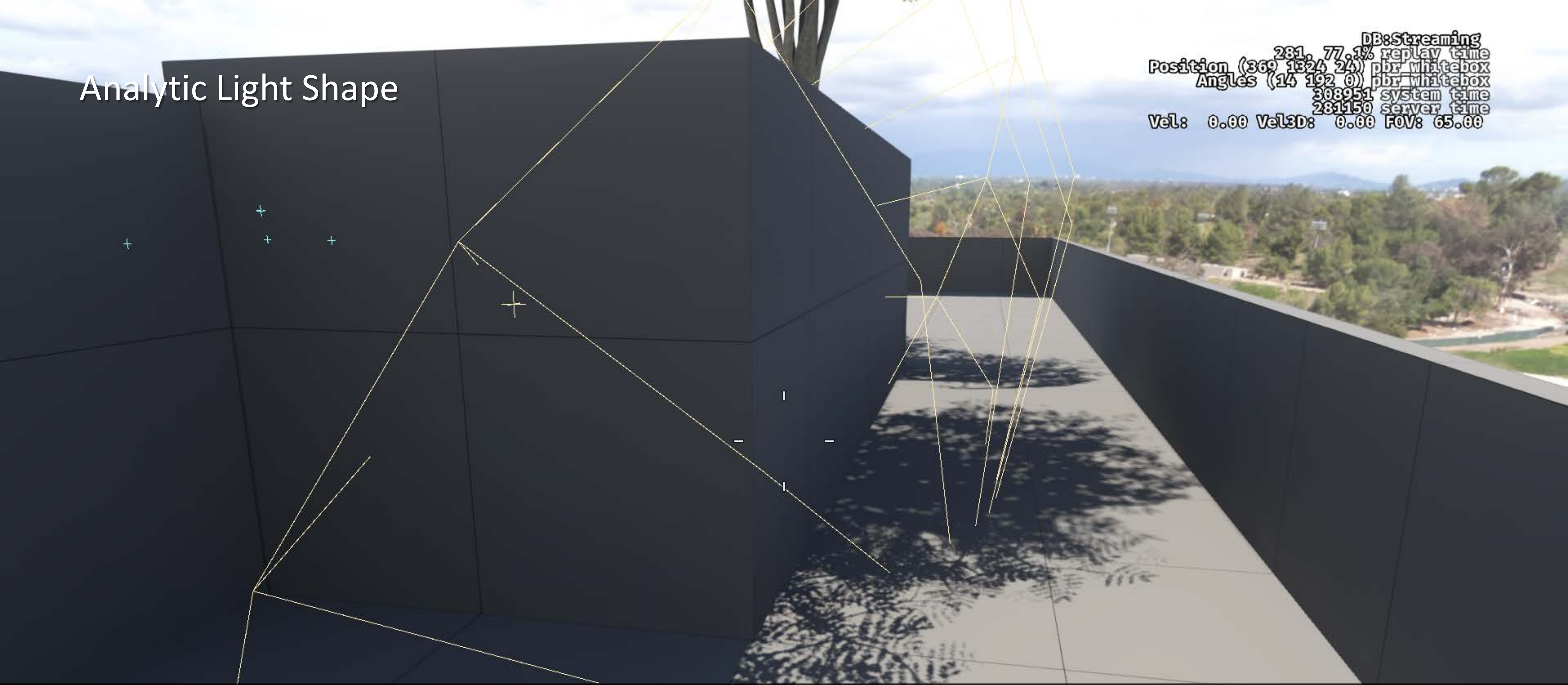
256 lights in Zombies opening scene - 60x40x32 - (PS4)	
Algorithm	**Culling time**
Atomic Raster	0.90 ms (100%)
Atomic Raster + 4xMSAA	0.51 ms (55%)
Atomic Raster + Compaction	0.66 ms (73%)
Atomic Raster + Compaction + 4xMSAA	0.32 ms (35%)

Worse scaling than tile, due to higher underutilization, due to low XY resolution.
Needs batching.
High latency due to WALK mode.
Good candidate to use in parallel with Async Compute.

Light Proxy

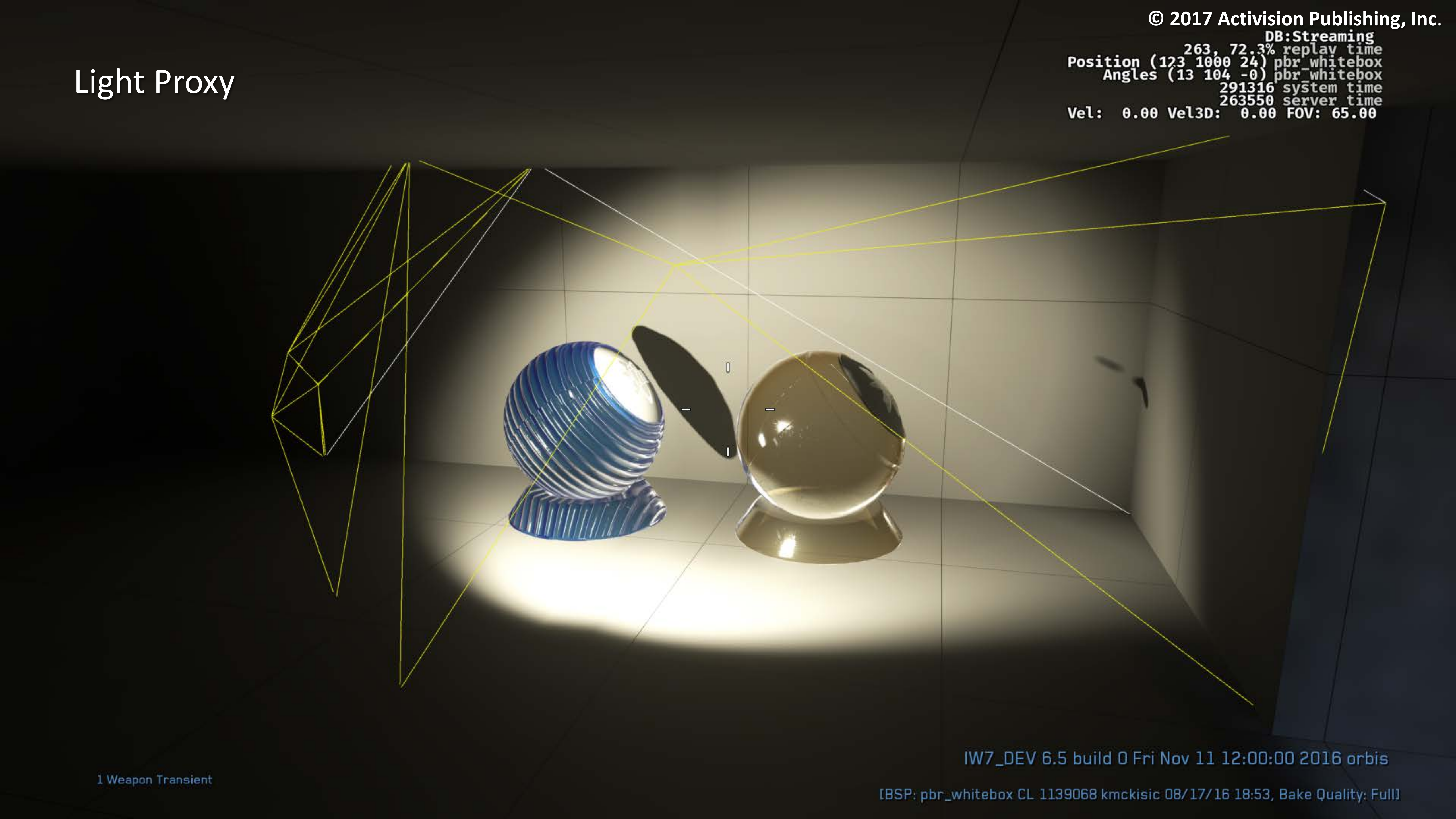
Analytic Light Shape

Analytic Light Shape

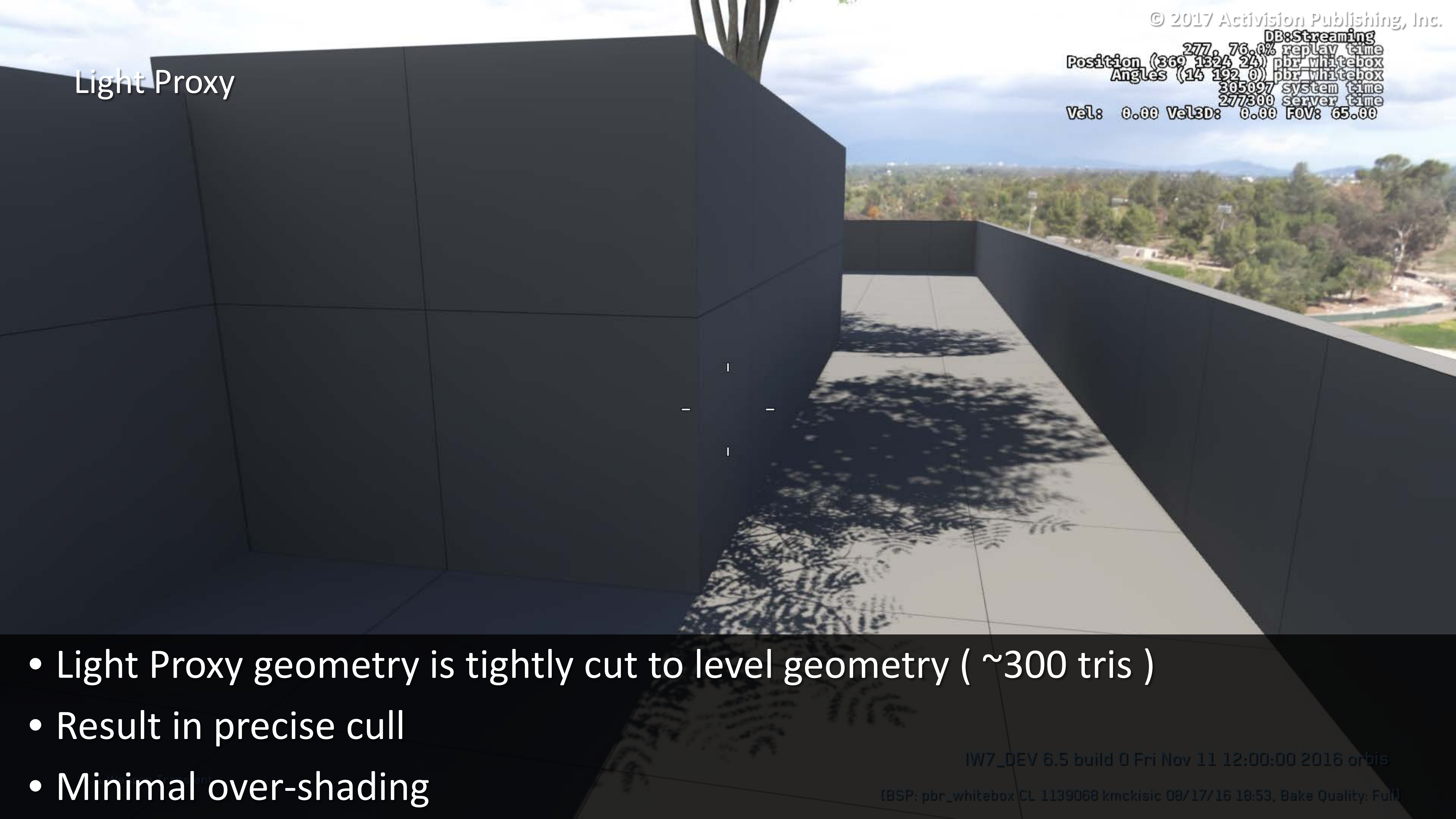


- Analytic light shapes do not respect occlusion
- Result in expensive over-shading
  - Visually corrected by shadow maps

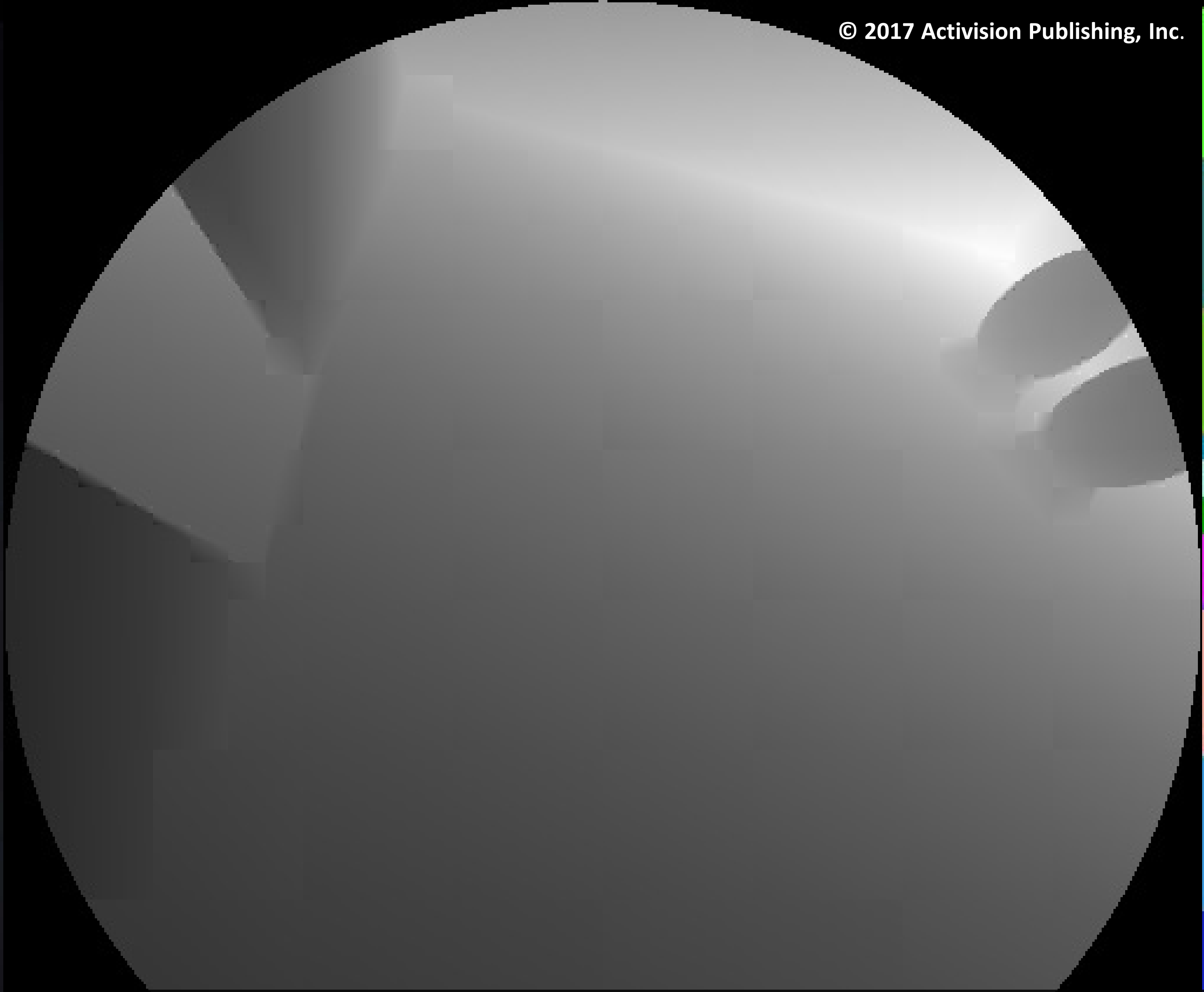
Light Proxy

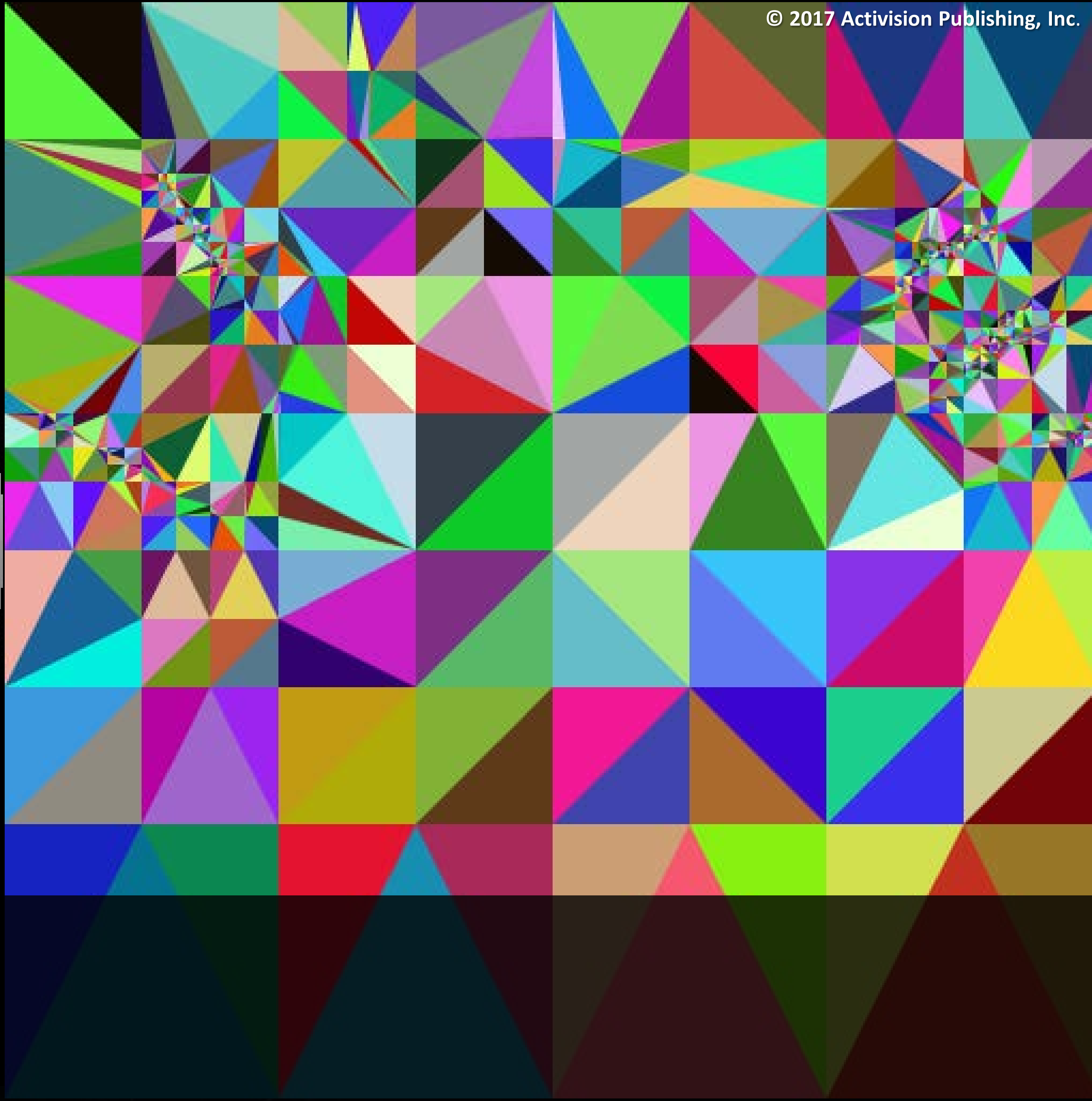

DB:Streaming
263, 72.3% replay time
Position (123 1000 24) pbr_whitebox
Angles (13 104 -0) pbr_whitebox
291316 system time
263550 server time
Vel: 0.00 Vel3D: 0.00 FOV: 65.00
1 Weapon Transient
IW7_DEV 6.5 build 0 Fri Nov 11 12:00:00 2016 orbis
[BSP: pbr_whitebox CL 1139068 kmckisic 08/17/16 18:53, Bake Quality: Full]

Light Proxy



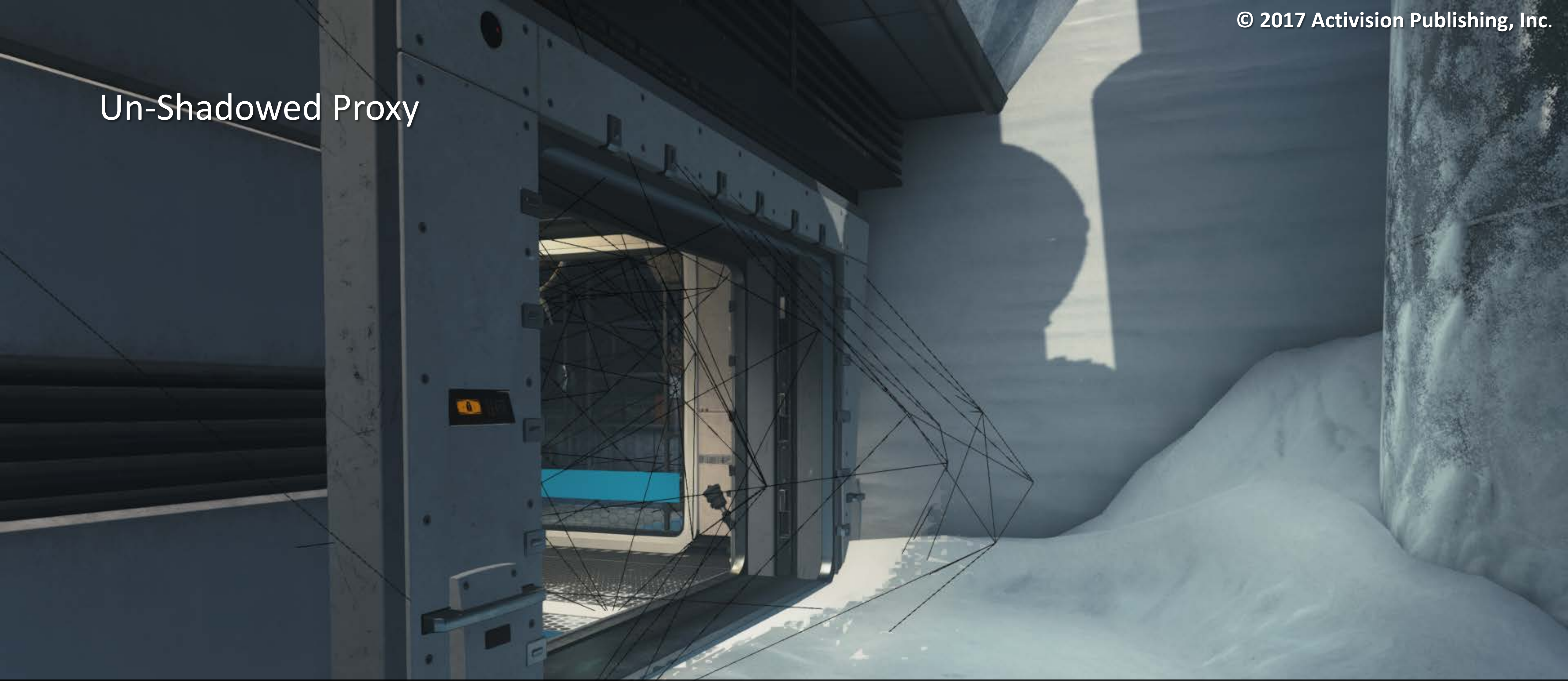
- Light Proxy geometry is tightly cut to level geometry (~300 tris)
- Result in precise cull
- Minimal over-shading





- Proxies generated for all stationary lights
  - Conservative Shadow Map rendering
  - Triangulation of CSM
    - Triangulation optimization
  - Greedy plane fitting to CSM depth for Low Tri Proxies


Shadowed Proxy



- Used to generate 8x8 tile buffer
  - Proxy intersection will be 8x8 tile ‘harsh’ visually if not masked
  - Static Shadow Map required for all lights
  - Use shadow map caching [DRO17]

Shadowed Proxy

Un-Shadowed Proxy

Light Proxy Performance

Open vista in Zombies (minimal light occlusion) - (PS4)		
Light Shapes	**Frame Time**	
Default	16.5 ms	
Light Proxies	15.6 ms	0.9ms

Space ship corridor (good light occlusion) - (PS4)		
Light Shapes	**Frame Time**	
Default	17.5 ms	
Light Proxies	14.4 ms	3.1ms

- Good candidate for async overlap
  - Majority of work is utilizing GPU fixed pipeline
  - Long Atomic queues
  - Long WALK loops
- Good balancing with CS based culling jobs for 'simple' cases

Improvements

- Extend proxies to more rendering entities
  - Complex shape decals
  - Complex shape reflection probes
- Improve rasterization batching
  - Sorting by Z allows efficient batching
  - Merge meshes in groups of lightBit % 32
    - `WaveAllBitOr(lightBit)` before `InterlockedOR`
  - Batch stencil passes in groups of lightBit % 32
    - Manual stencil write / read (to R32_UINT texture) <no HiS>
    - `WaveAllBitOr(lightBit)` before `manualStencil.Store`
    - Try only if base stencil optimization helps your content
- 8xMSAA

Rendering presentations 2017

- EGSR
  - Ambient Dice — Michal Iwanicki
- Siggraph
  - Indirect Lighting in COD: Infinite Warfare — Michal Iwanicki
  - Dynamic Temporal Supersampling and Anti-Aliasing — Jorge Jimenez
  - Improved Culling for Tiled and Clustered Rendering — Michal Drobot
  - Practical Multilayered Materials in COD: Infinite Warfare — Michal Drobot
- Microsoft XFest 2017
  - Optimizing the Renderer of Call of Duty: Infinite Warfare — Michal Drobot

research.activision.com


JOIN US
www.activisionblizzard.com/careers
CALL OF DUTY
INFINITE WARFARE

Special Thanks

- Rasterized Culling
  - Paul Edelstein
  - Johan Kohler
  - Michael Vance

- Proxy Generation
  - Peter-Pike Sloan
  - Peter Pon

Course Organizer:

Natalya Tatarchuk

Additional Thanks

- IW Rendering Team
  - Anthony Carotenuto, Rulon Raymond, Peter Pon, Mike Esposito, Vishal Kashyap, Felipe Gomez
- Activision Central Tech
- Infinity Ward
- Sledgehammer Games
- Treyarch
- Raven


research.activision.com
Q&A
michal@infinityward.com
@MichalDrobot
CALL OF DUTY
INFINITE WARFARE

References

[DRO14] "Low Level Optimizations for GCN", Michal Drobot, *Digital Dragons 2014*

[DRO17] "Rendering of Call of Duty: Infinite Warfare", Michal Drobot, *Digital Dragons 2017*

[HAR04] "Deferred Shading", Shawn Hargreaves, Mark Harris, *2014*

[HEI16] "Real-Time Polygonal-Light Shading with Linearly Transformed Cosines", Eric Heitz, Jonathan Dupuy, Stephen Hill and David Neubelt, *Siggraph 2016*

[JOH09] "Parallel Graphics in Frostbite – Current & Future"m Johan Andersson, *Siggraph 2009*

[KAS11] "Secrets of CryENGINE 3 Graphics Technology", Nickolay Kasyan, Nicolas Schulz and Tiago Sousa, *Siggraph 2011*

[SOU16] "The Devil is in the details : idTech 666", Tiago Sousa, *Siggraph 2016*

[THI11] "Deferred Shading Optimizations", Nicolas Thibieroz, *GDC 2011*

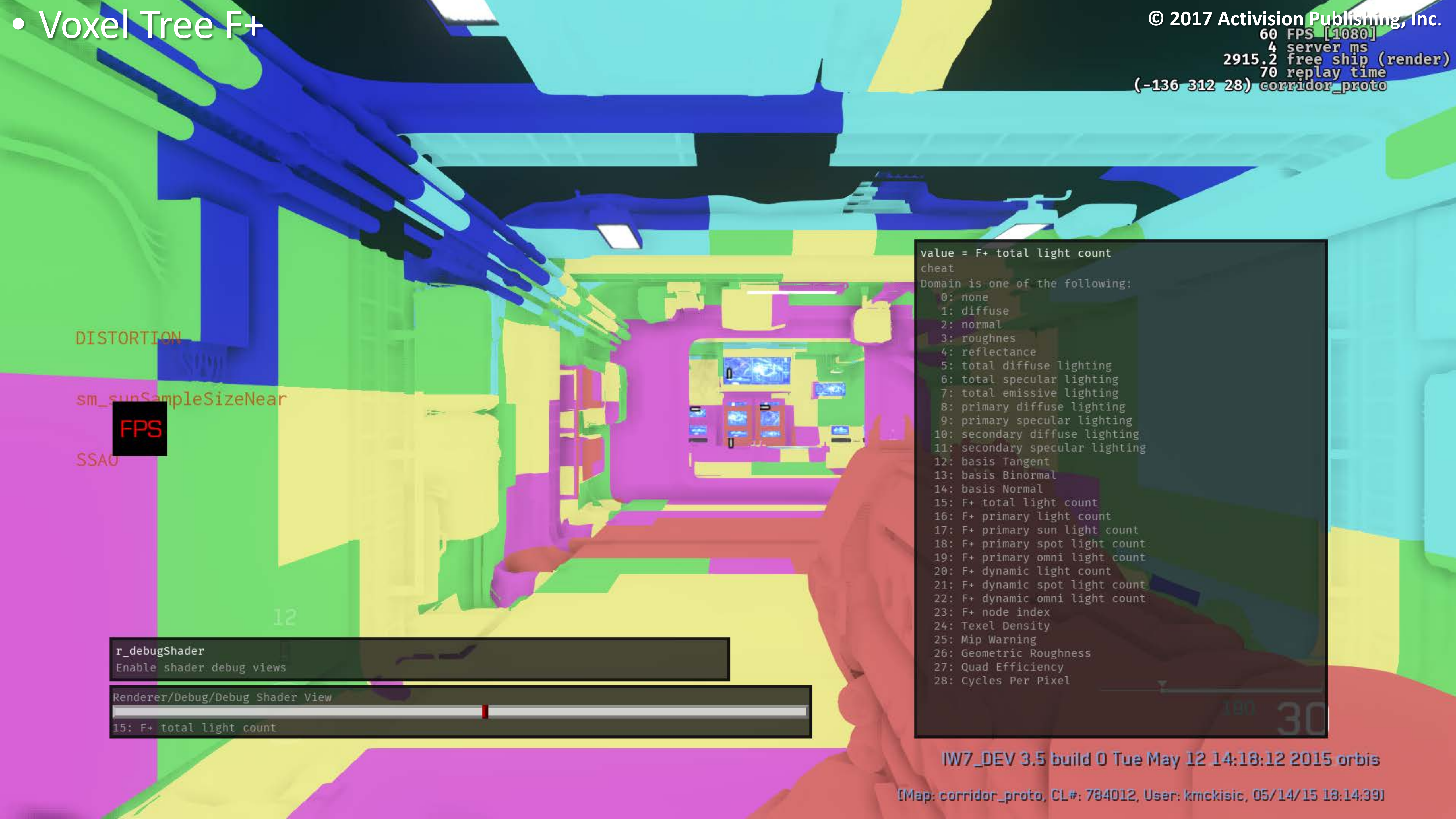
[WRO17] "Cull That Cone", Bart Wronski, https://bartwronski.com/2017/04/13/cull-that-cone/

Bonus Slides

Spatial acceleration structure : World

- Voxel Tree
  - World space Oct-tree
    - Each leaf is a ‘cube’ / voxel
  - Precomputed with occlusion
    - i.e. lights would be shadowed and contained to their volume of influence only
  - Allows easy precomputed / cached out-of-frustum 3D lookups
  - Expensive traversal
    - Need to traverse hierarchy, multiple $ misses, indirect reads
  - High memory consumption (resolution dependent)
  - Significant pre-computation time

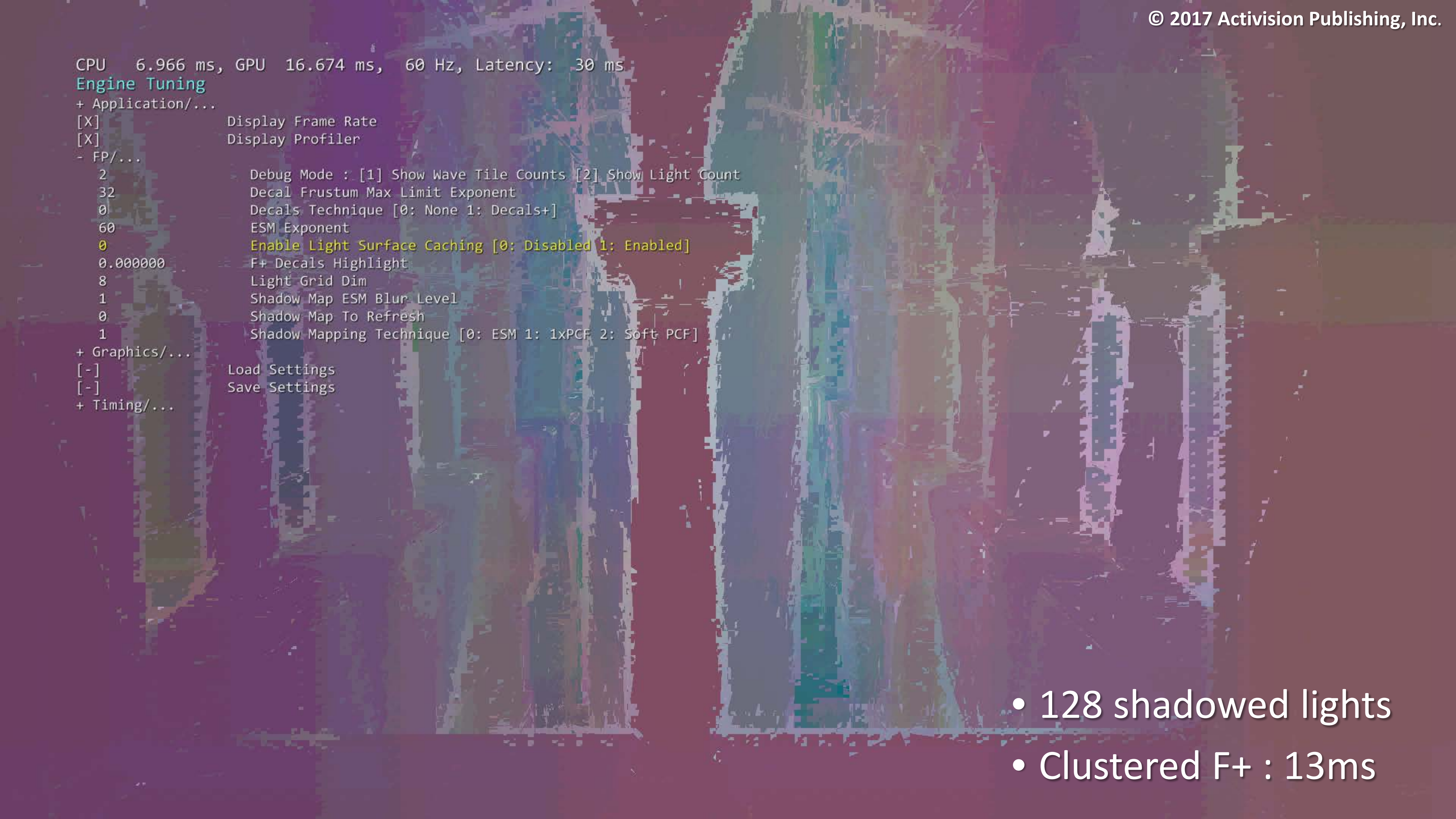
- Voxel Tree F+


60 FPS [1080]
4 server ms
2915.2 free ship (render)
70 replay time
(-136 312 28) corridor_proto
DISTORTION
sm_sunSampleSizeNear
FPS
SSAO
value = F+ total light count
cheat
Domain is one of the following:
0: none
1: diffuse
2: normal
3: roughnes
4: reflectance
5: total diffuse lighting
6: total specular lighting
7: total emissive lighting
8: primary diffuse lighting
9: primary specular lighting
10: secondary diffuse lighting
11: secondary specular lighting
12: basis Tangent
13: basis Binormal
14: basis Normal
15: F+ total light count
16: F+ primary light count
17: F+ primary sun light count
18: F+ primary spot light count
19: F+ primary omni light count
20: F+ dynamic light count
21: F+ dynamic spot light count
22: F+ dynamic omni light count
23: F+ node index
24: Texel Density
25: Mip Warning
26: Geometric Roughness
27: Quad Efficiency
28: Cycles Per Pixel
r_debugShader
Enable shader debug views
Renderer/Debug/Debug Shader View
15: F+ total light count
IW7_DEV 3.5 build 0 Tue May 12 14:18:12 2015 orbis
[Map: corridor_proto, CL#: 784012, User: kmckisic, 05/14/15 18:14:39]

Spatial acceleration structure : World

- Per-mesh Triangle / Texel
  - Precomputed with occlusion
  - Allows per-mesh lookup
  - Stored per-triangle (triID) or per-texel (like a lightmap)
  - Medium memory consumption (resolution dependent)
  - Arguably most efficient in Forward once cached
  - Complex pipeline – high caching time

- An interesting experiment that did not ship
  - Looking forward to revisit in next projects



- 128 shadowed lights
- Clustered F+ : 13ms

- 128 Shadowed lights
- Surface Cached F+ : 7ms
  - Shadow / back-face culling