NEXT GENERATION POST PROCESSING IN CALL OF DUTY ADVANCED WARFARE

Jorge Jimenez
Graphics R&D Technical Director - Activision Blizzard

SLEDGEHAMMER GAMES

BACKGROUND

- I’ve been working inside of Sledgehammer Games for a year
- I’m presenting this work on their behalf
- I’d like to highlight the tremendous work done by Sledgehammer Games and all the credit they deserve
  - Regarding the game technology, this presentation is just the tip of the iceberg

BACKGROUND

- My Sledgehammer Games co-presenter Danny Chan unfortunately could not join because of game finaling needs
- All images/footage are from development builds and not from the completed game
- I feel really lucky to be here presenting this work, and wish you will enjoy the presentation as much as we did developing the technology we will explain

INTRODUCTION

- **Call of Duty: Advanced Warfare** art direction aims for photorealism
- **Post effects** play a huge role in **photorealism**
  - They received extensive attention for this game
- Post effects **quality** and **image stability** is probably one of the most striking differences between games and films

INTRODUCTION

- Improve image quality: better sampling approaches
- Improve accuracy: better approximate ground truth
- Pay attention to the details
- Do not sacrifice performance for quality (60 fps constraint)

POST EFFECT PIPELINE

- Motion Blur
- Bokeh Depth of Field
- Subsurface Scattering
- Bloom/Veil

- Shadow Sampling

MOTION BLUR

BASIC SCATTER IDEA

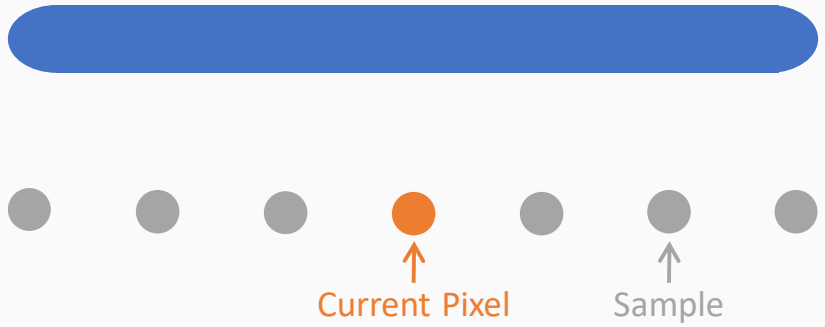
- Naturally implemented with scatter techniques

BASIC SCATTER IDEA

- Naturally implemented with scatter techniques

SCATTER-AS-YOU-GATHER

- Scatter as you gather allows for a filter-based approach

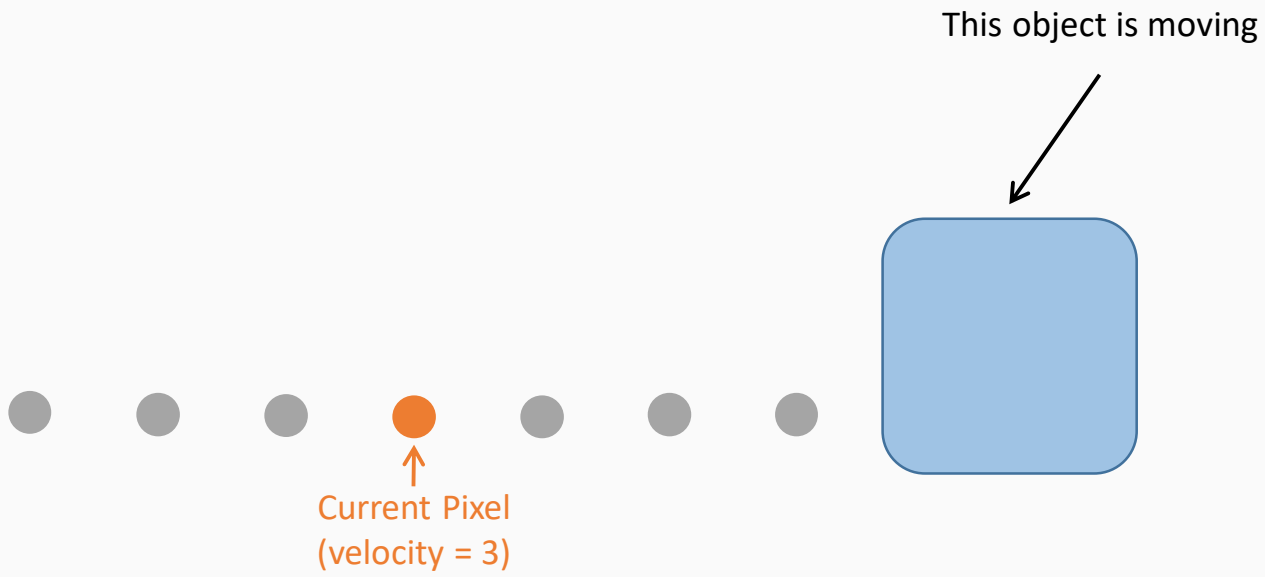


SCATTER-AS-YOU-GATHER PROBLEMS

- **Main scatter-as-you-gather problems:**
  - How to determine the sampling area
  - Detecting which samples actually contribute
  - How to recover the background

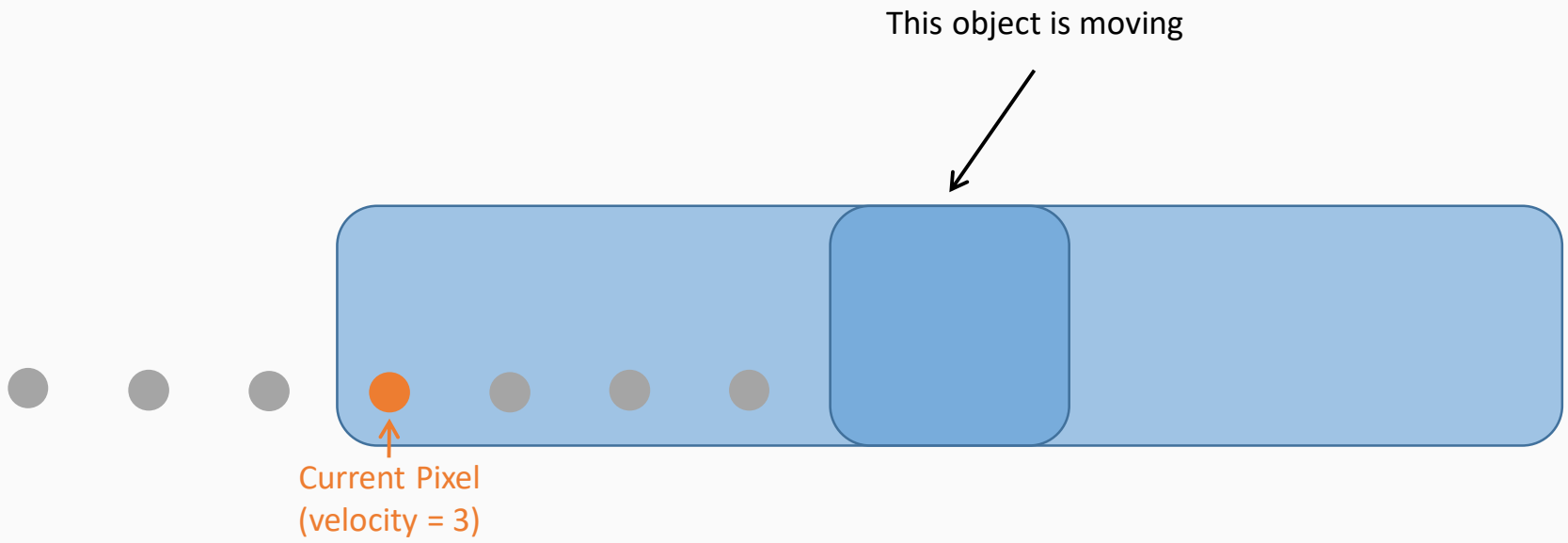
SAMPLING AREA

- Simple approaches use current pixel velocity



SAMPLING AREA

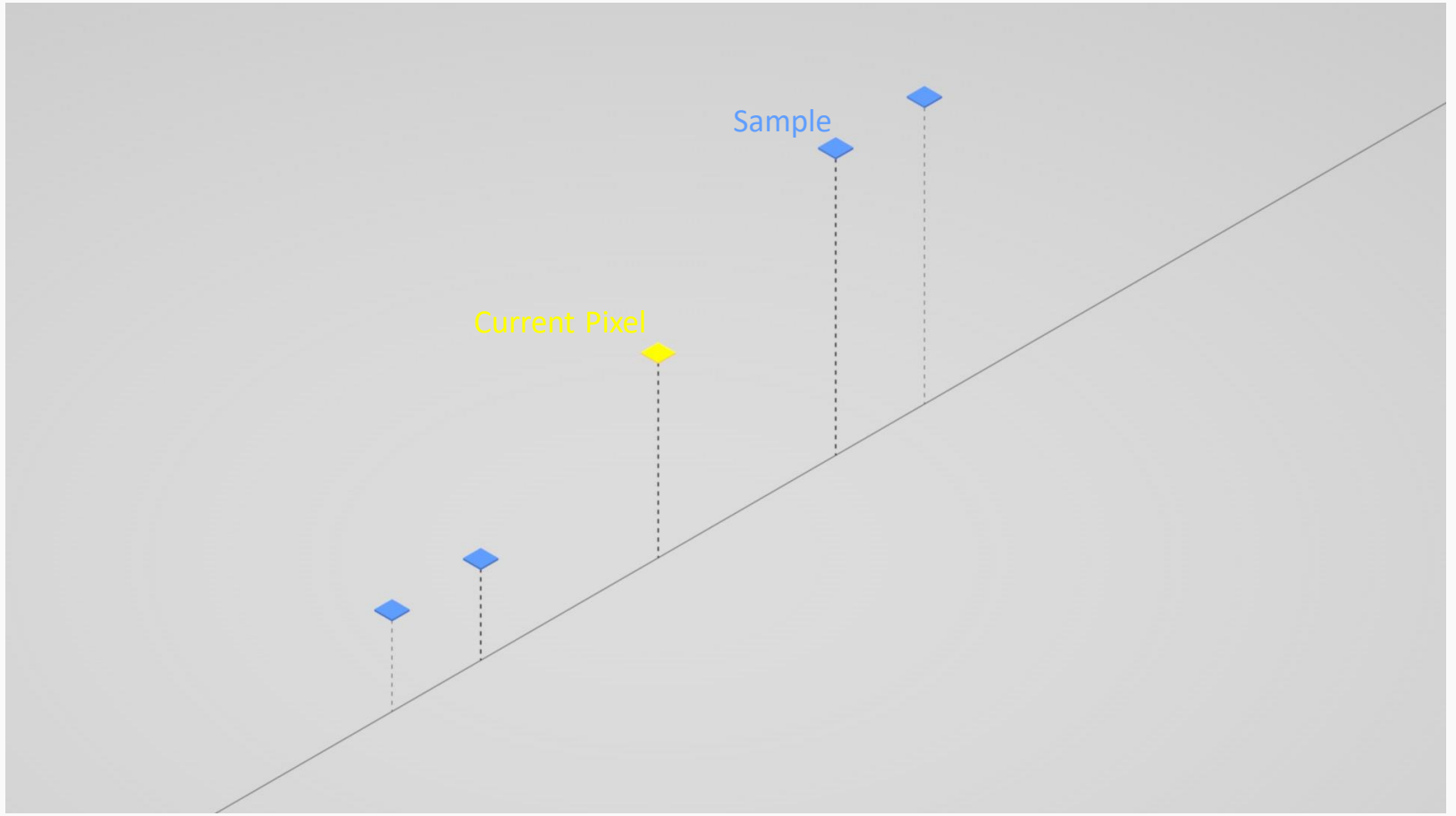
- But this fails in some cases



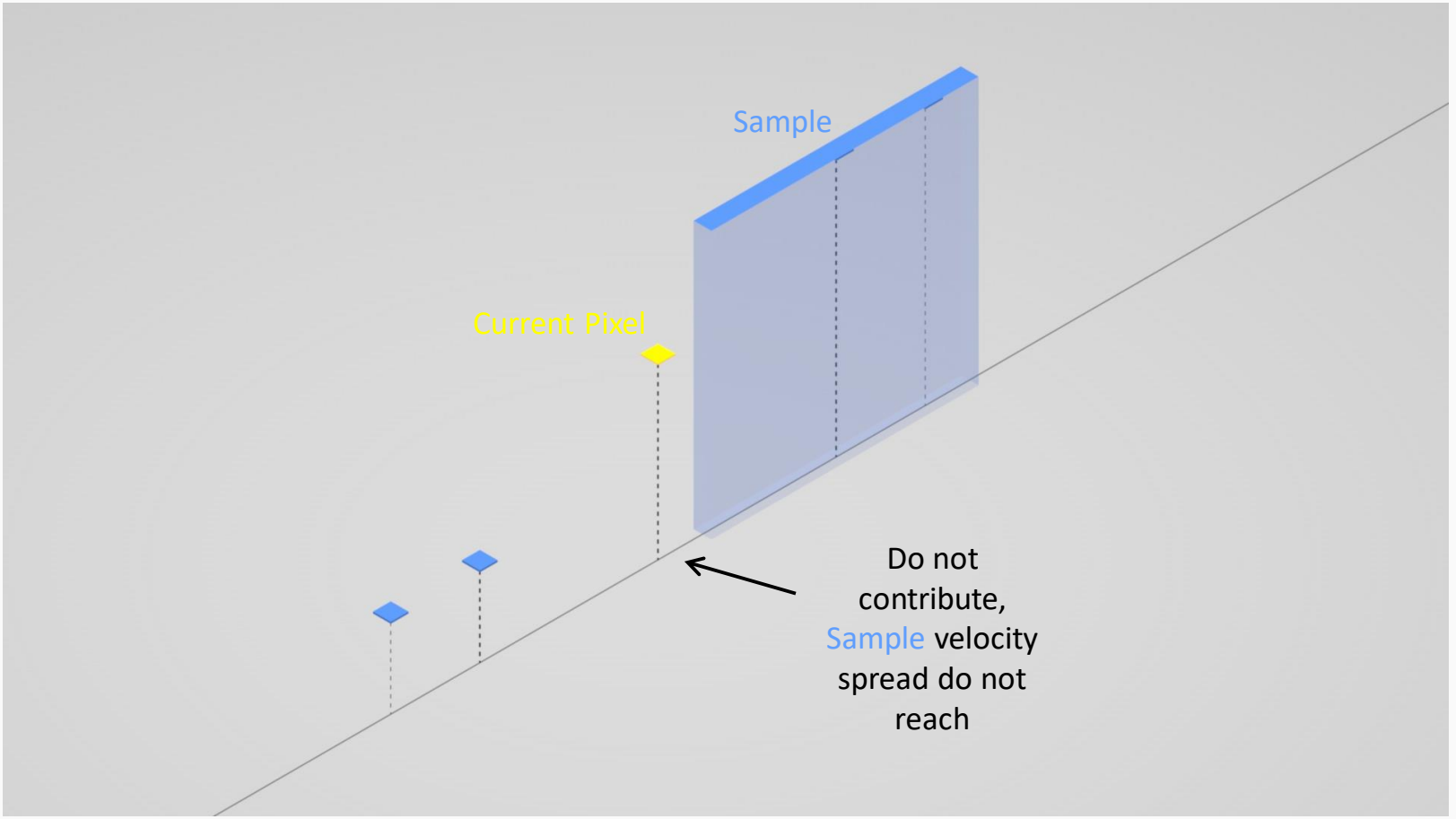
SCATTER-AS-YOU-GATHER PROBLEMS

- **Main scatter-as-you-gather problems:**
  - ~~How to determine the sampling area~~
  - Detecting which samples actually contribute
  - How to recover the background

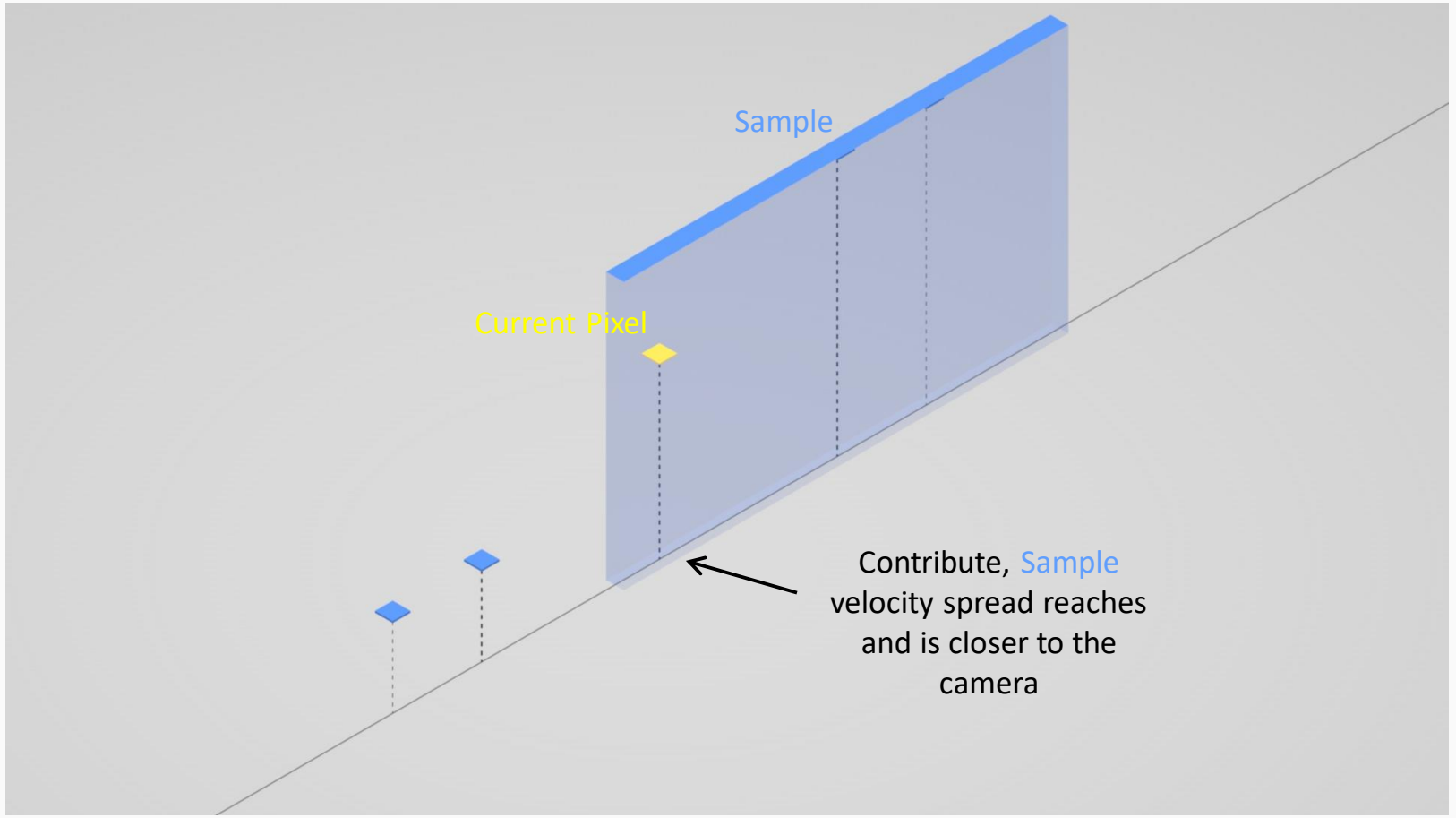
SAMPLE CONTRIBUTION



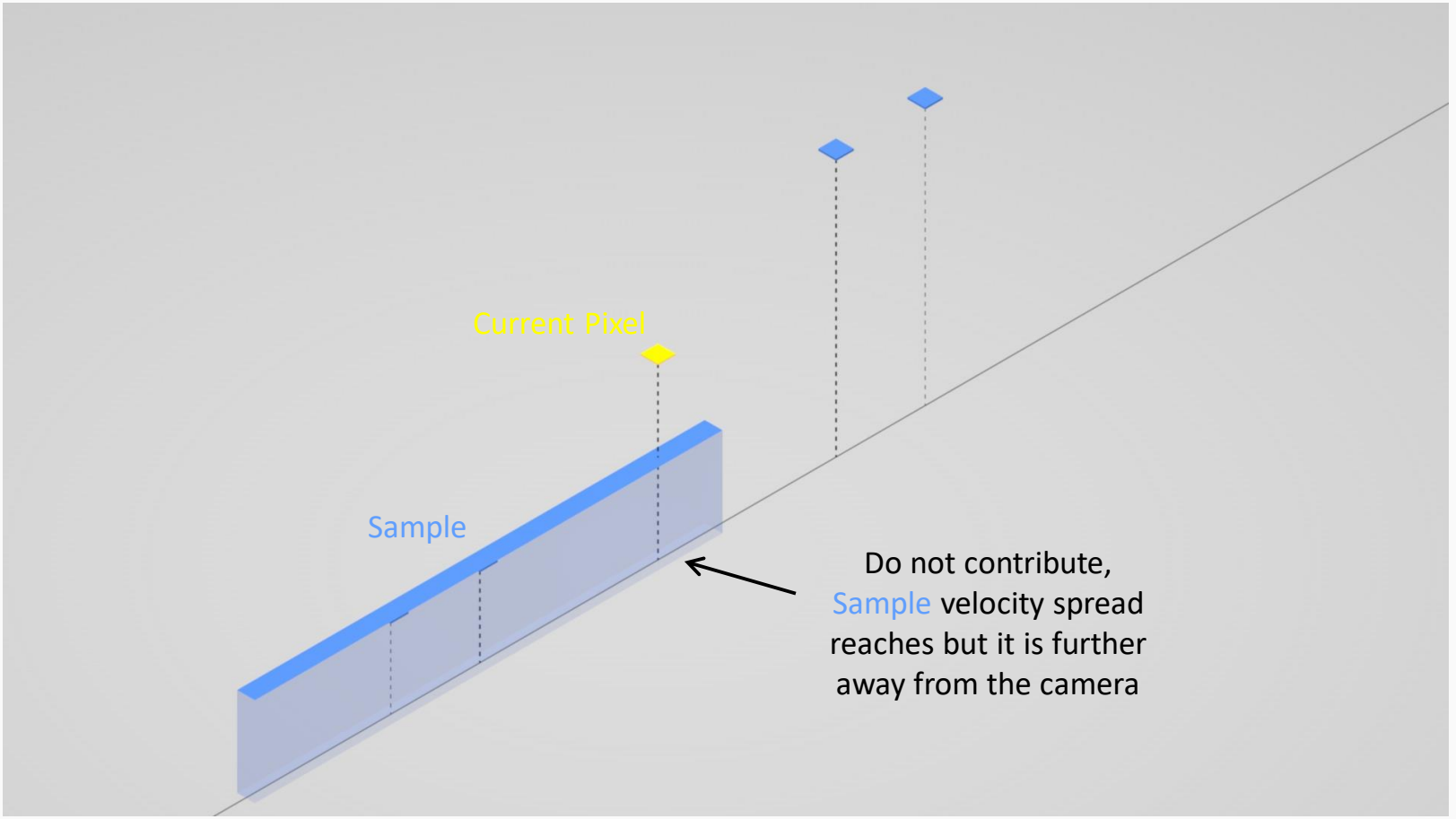
SAMPLE CONTRIBUTION



SAMPLE CONTRIBUTION



SAMPLE CONTRIBUTION



SCATTER-AS-YOU-GATHER PROBLEMS

- **Main scatter-as-you-gather problems:**
  - ~~How to determine the sampling area~~
  - ~~Detecting which samples actually contribute~~
  - How to recover the background

BACKGROUND RECOVERY

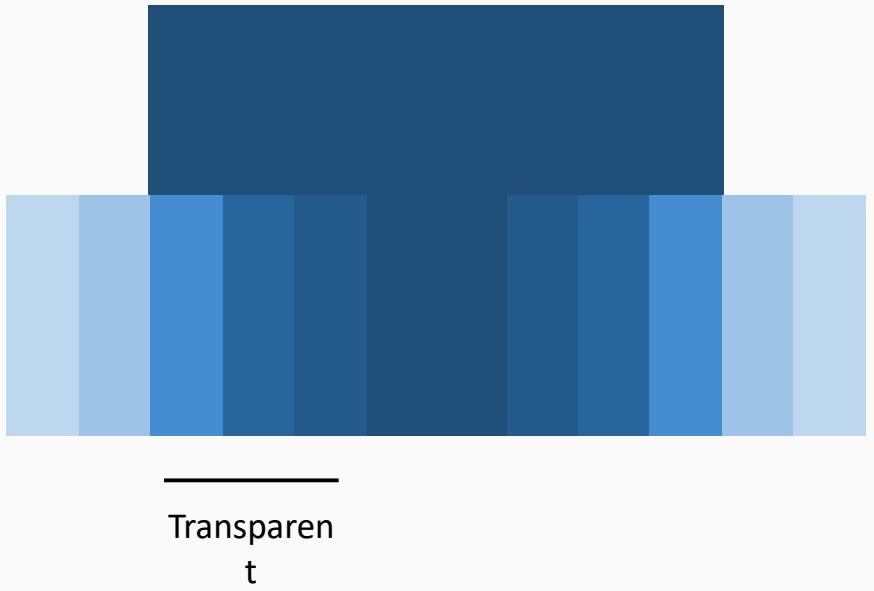
BACKGROUND RECOVERY

BACKGROUND RECOVERY

BACKGROUND RECOVERY

Input for filter approaches

Motion Blurred Object



BACKGROUND RECOVERY

Input Image

BACKGROUND RECOVERY

- **The blur should be centered in the object silhouette**
  - This means the object will become transparent on the silhouette
  - We do not know the background
    - Need to reconstruct it

Motion Blur

PREVIOUS WORK

- Our motion blur technique based on
  - [McGuire2012] [McGuire2013]
- And its code vectorization
  - [Sousa2013]

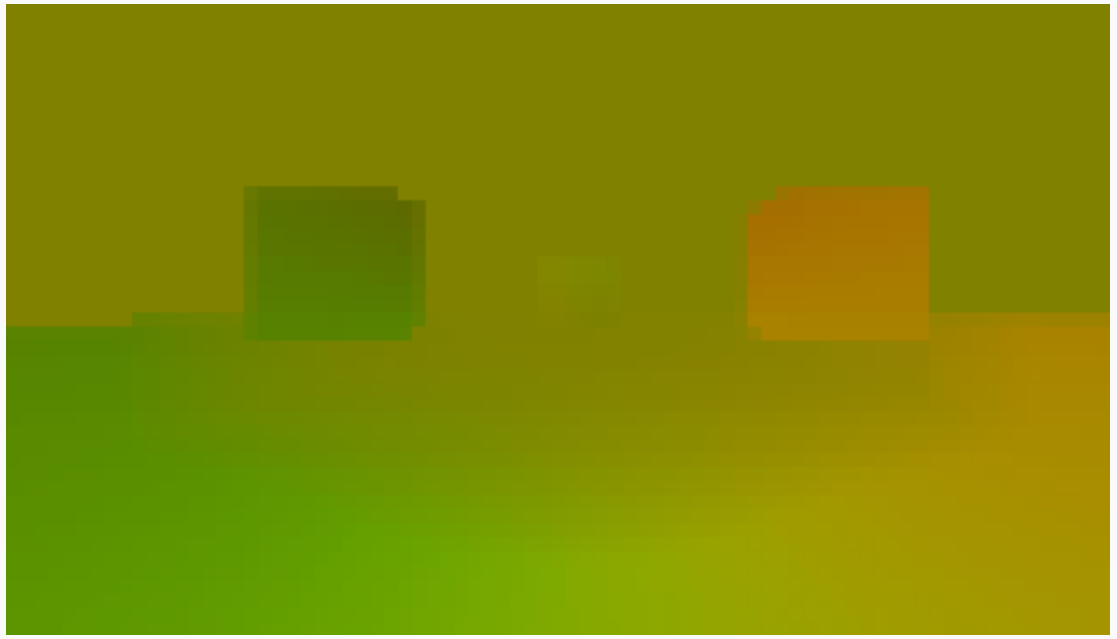
- Our Improvements
  - Accuracy
    - Using more accurate sample contributions and background recovery
  - Image quality
    - Using better sampling patterns to mask undersampling – See online slides
  - Optimization
    - Running simplified versions of the shader were possible

SCATTER-AS-YOU-GATHER PROBLEMS

- **Main scatter-as-you-gather problems:**
  - How to determine the sampling area
  - Detecting which samples actually contribute
  - How to recover the background

SAMPLING AREA

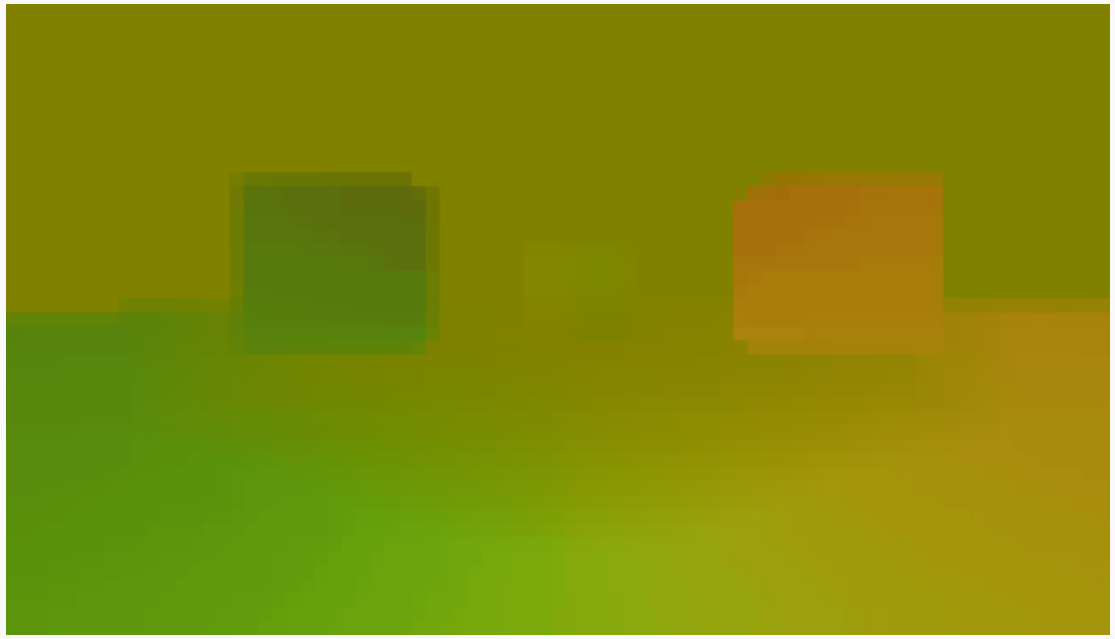
- **First Pass [Tile Max]:**
  - Calculate maximum velocities in tiles (20x20 pixels)



Tile Max

SAMPLING AREA

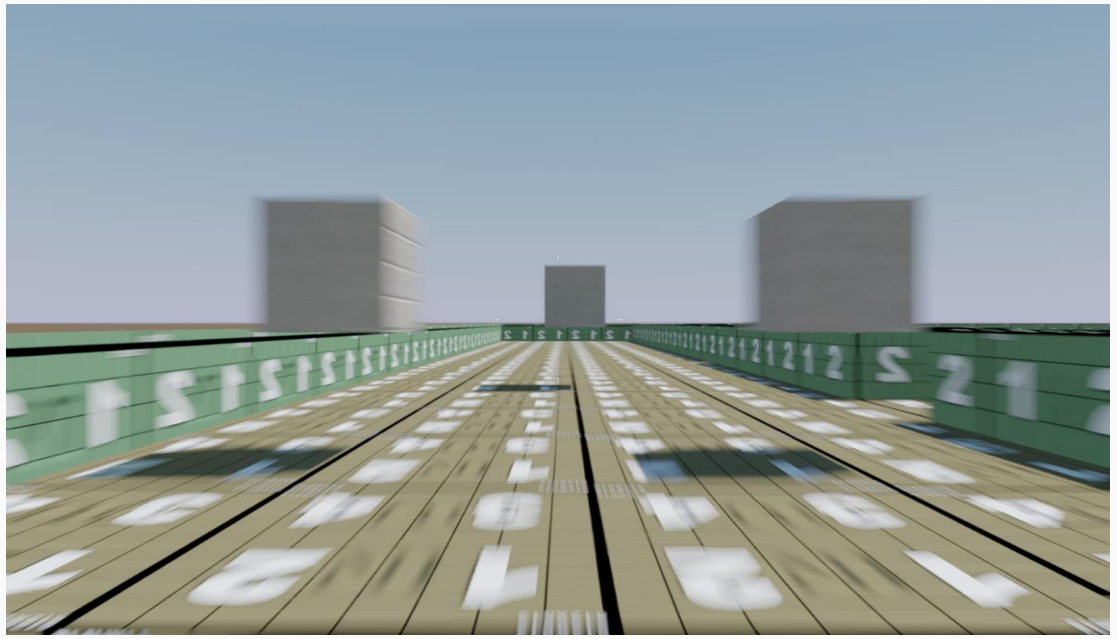
- **First Pass [Tile Max]:**
  - Calculate maximum velocities in tiles (20x20 pixels)
- **Second Pass [Tile Neighborhood]:**
  - Calculate maximum velocity in a 3x3 kernel for each tile



Tile Neighborhood

SAMPLING AREA

- **First Pass [Tile Max]:**
  - Calculate maximum velocities in tiles (20x20 pixels)
- **Second Pass [Tile Neighborhood]:**
  - Calculate maximum velocity in a 3x3 kernel for each tile
- **Third Pass [Motion Blur]:**
  - Apply a fullres variable-width blur in the tile velocity direction



Motion Blur

SAMPLING AREA

Input Image

SAMPLING AREA

Motion Blur using Pixel Velocity

SAMPLING AREA

Motion Blur using Max Tile Velocity

SCATTER-AS-YOU-GATHER PROBLEMS

- **Main scatter-as-you-gather problems:**
  - ~~How to determine the sampling area~~
  - Detecting which samples actually contribute
  - How to recover the background

IMPROVED SAMPLE CONTRIBUTION

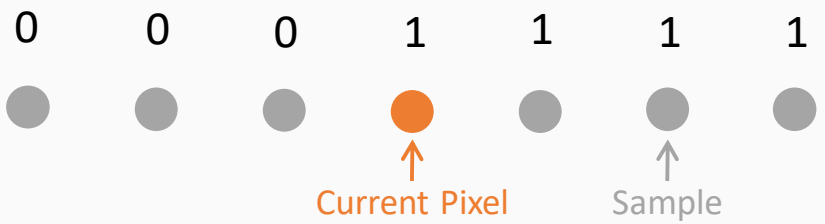
- The first improvement that we have introduced is to more accurately calculate the contribution from each sample

[McGuire2013]

Improved Sample Contribution

IMPROVED SAMPLE CONTRIBUTION

- Each sample receives a weight depending on its contribution
- The total contribution is tracked
- The average is then renormalized



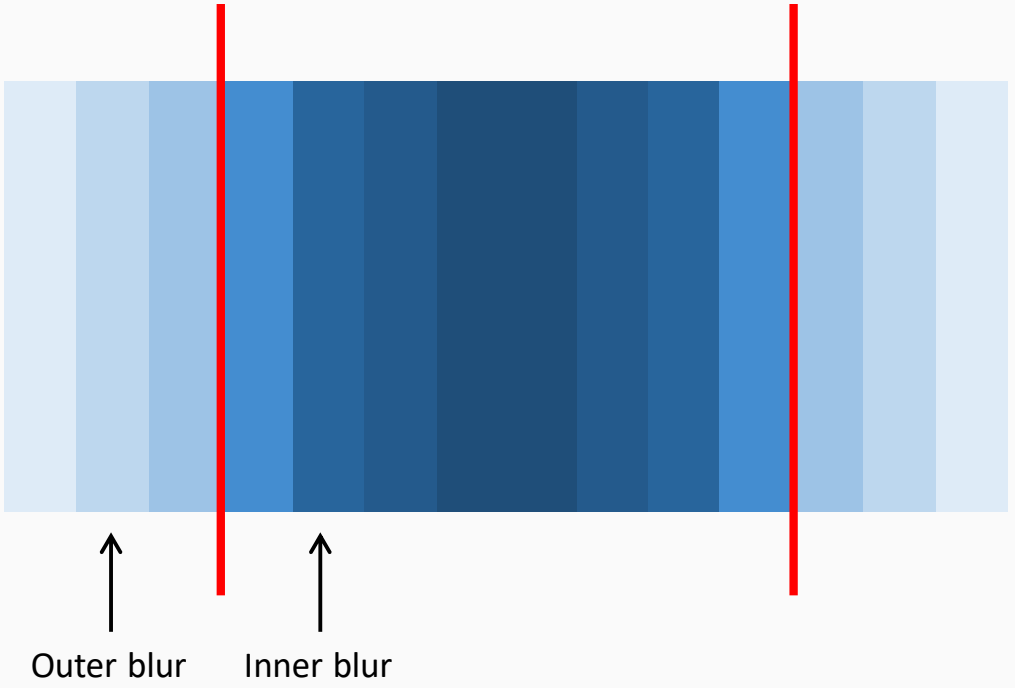
Color = (0 * c1 + 0 * c2 + 0 * c3 + 1 * c4 + 1 * c5 + 1 * c6 + 1 * c7) / (1 + 1 + 1 + 1)

IMPROVED SAMPLE CONTRIBUTION

- Consider this moving object

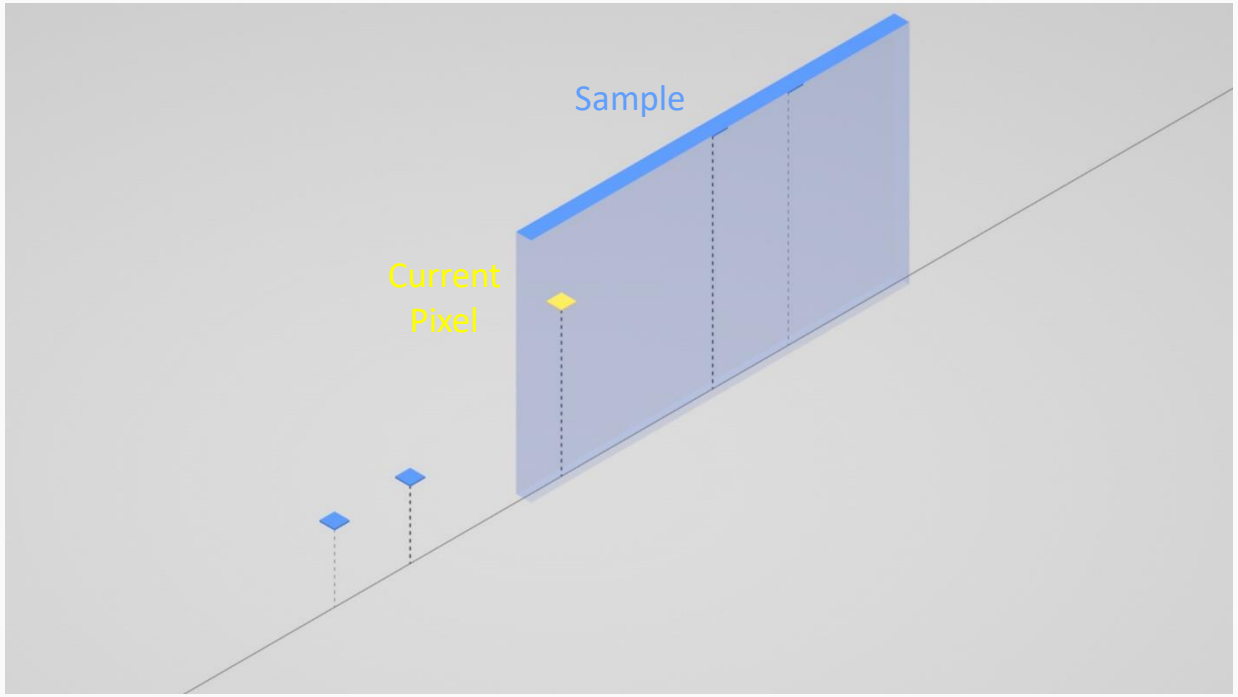
IMPROVED SAMPLE CONTRIBUTION

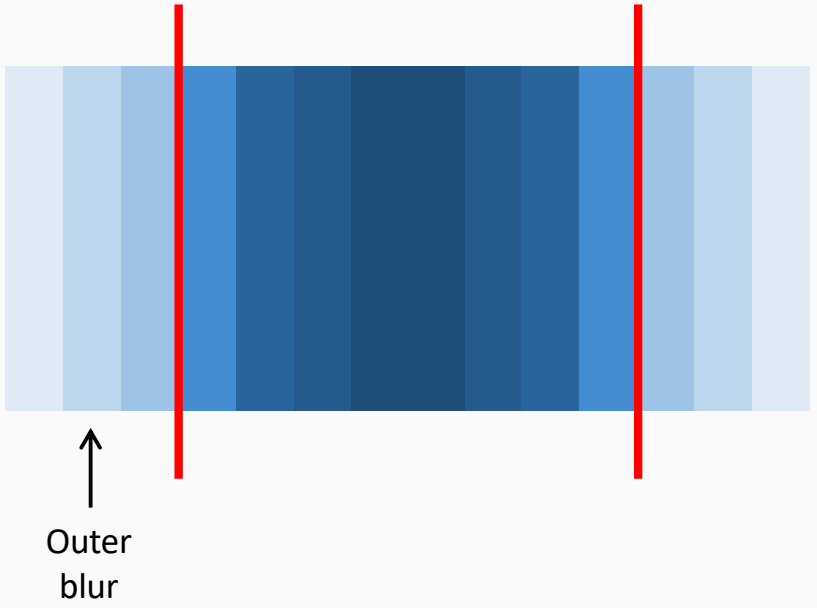
- It creates two blurred areas
  - Outer blur
  - Inner blur
- Inner blur reveals a background that we do not have



IMPROVED SAMPLE CONTRIBUTION

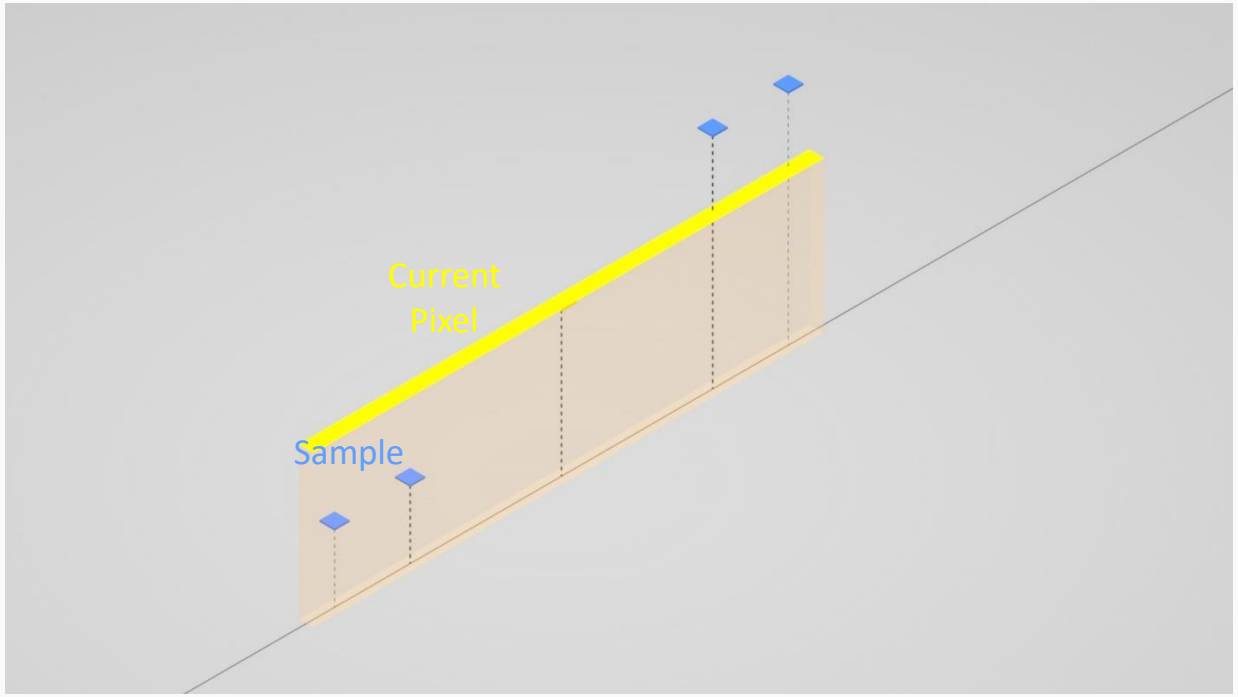
- **Outer blur:** sample contributes if is foreground and it spreads over current pixel
  - The only real case that happens in real world

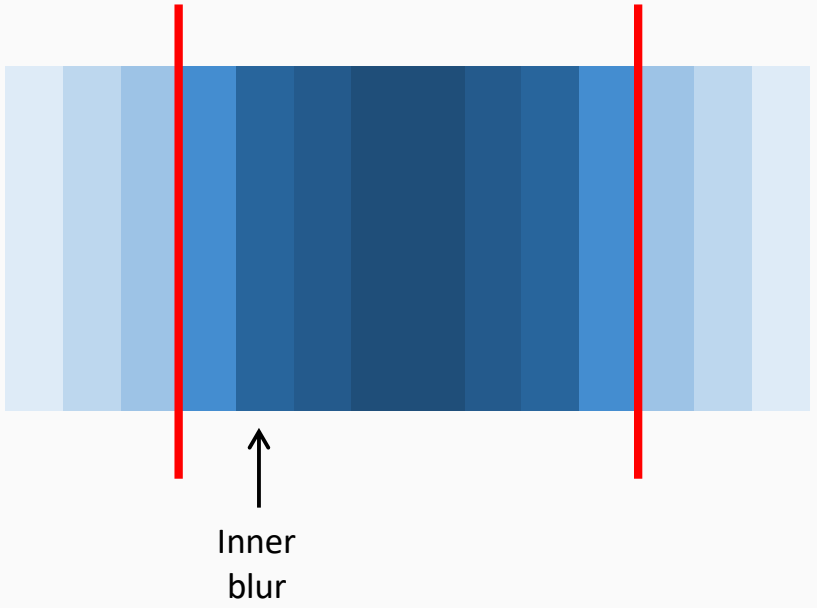




IMPROVED SAMPLE CONTRIBUTION

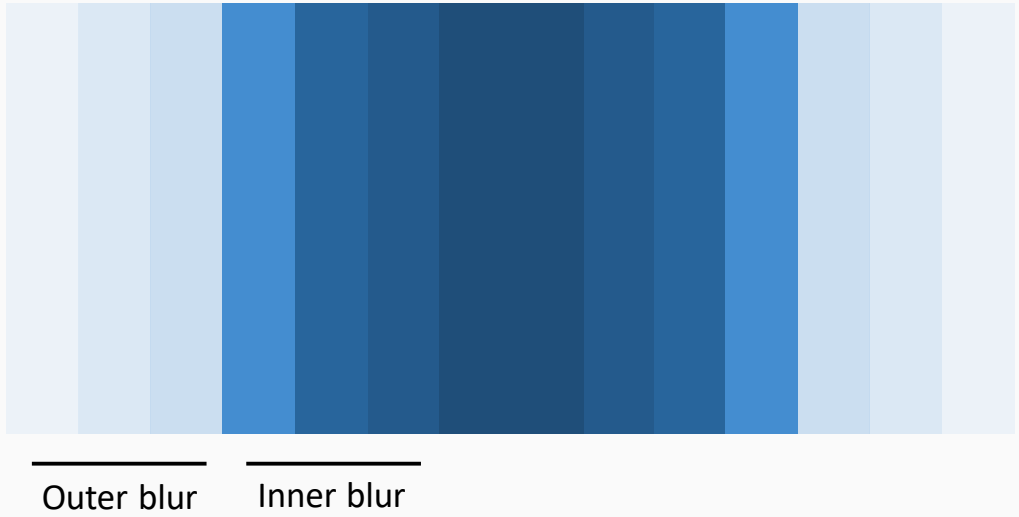
- **Inner blur:** sample contributes if is background but current pixel velocity is bigger than the offset to the sample
  - Allows to reconstruct the background by blurring background pixels [McGuire2012]





IMPROVED SAMPLE CONTRIBUTION

- There is a discontinuity between the outer and inner blurs



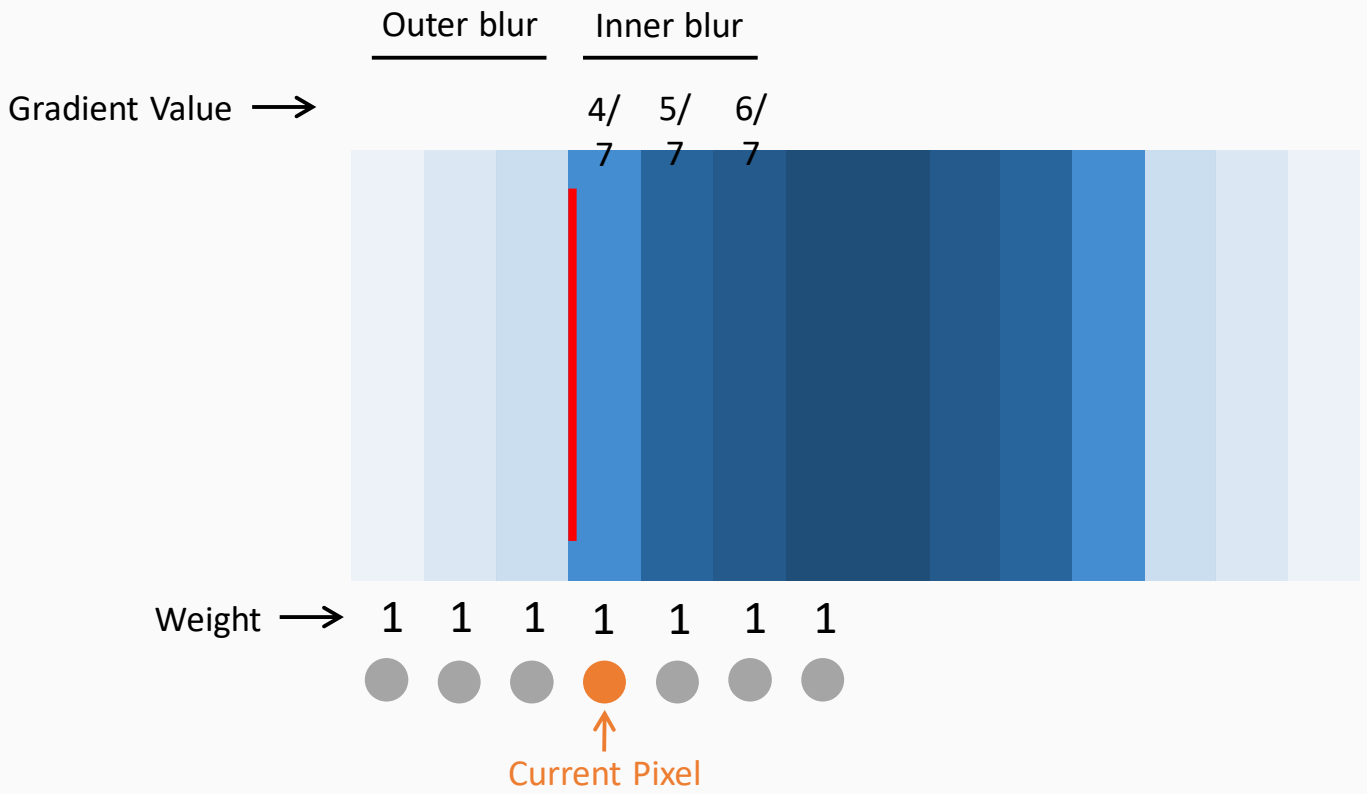
[McGuire2013]

IMPROVED SAMPLE CONTRIBUTION

- Consider this moving object

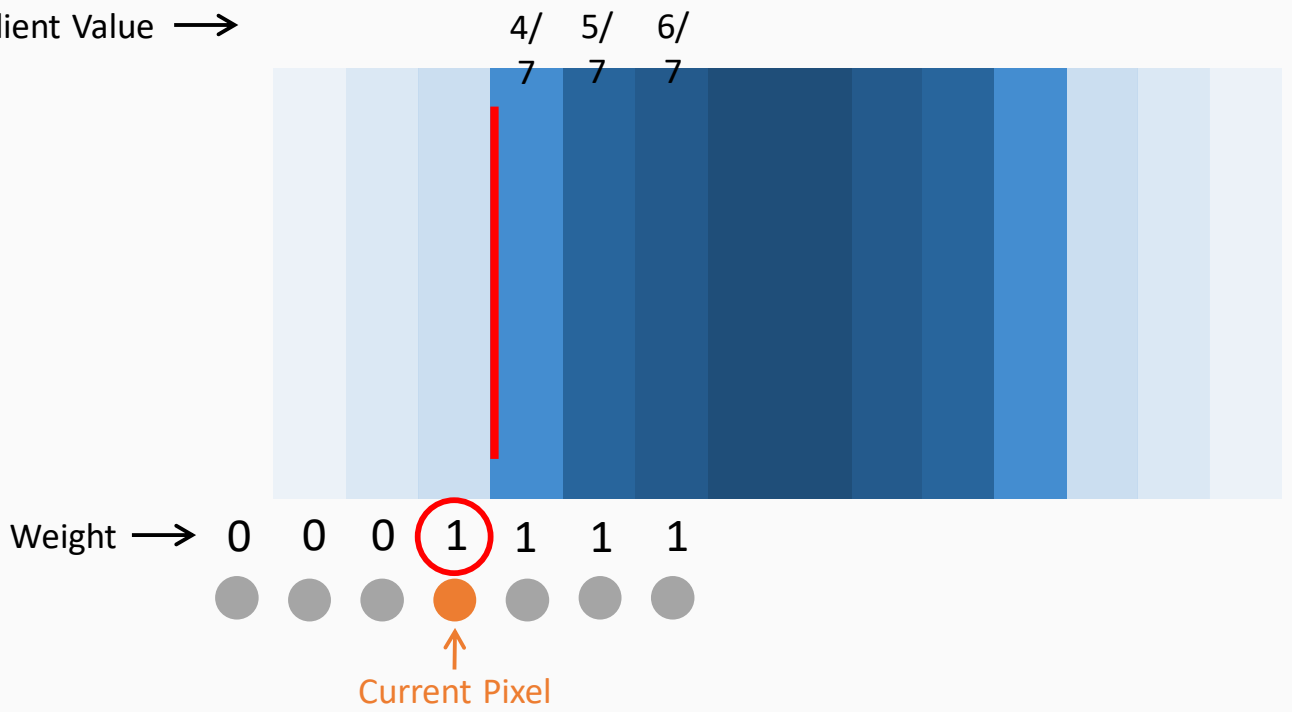
IMPROVED SAMPLE CONTRIBUTION

- Consider this moving object



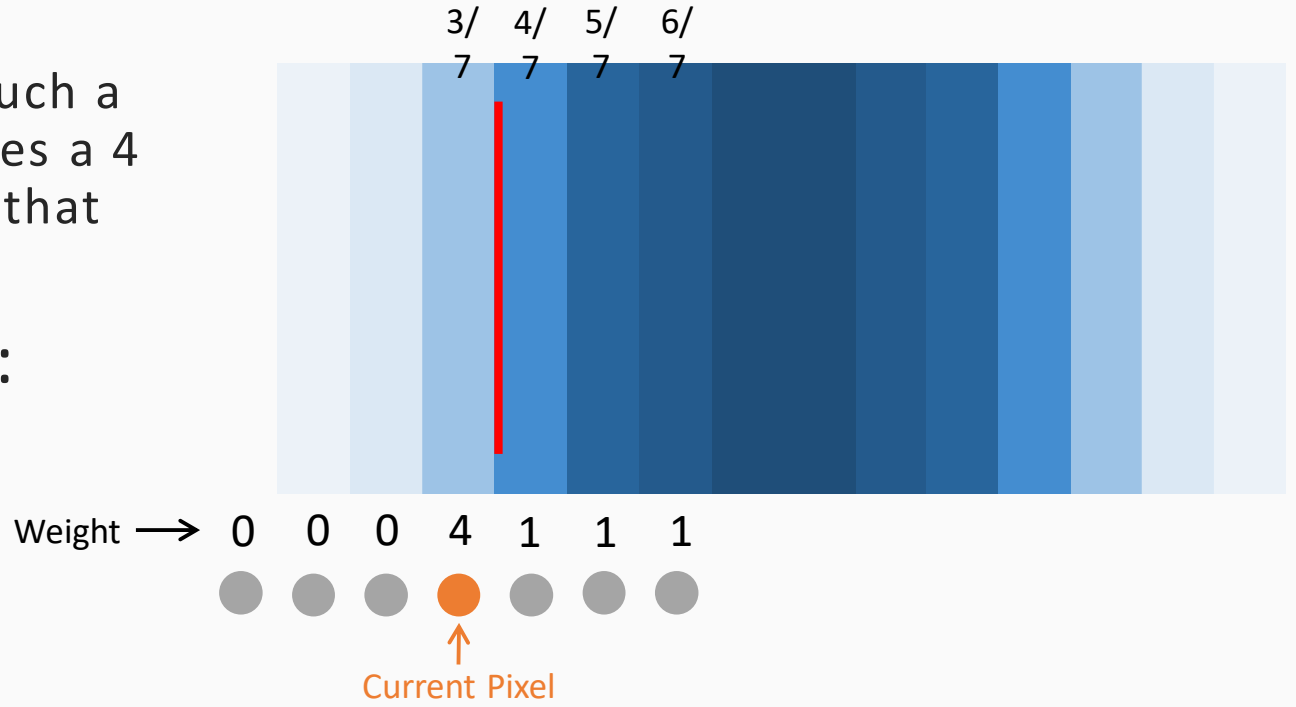
IMPROVED SAMPLE CONTRIBUTION

- For the first outer blur pixel there is an abrupt change in the weights



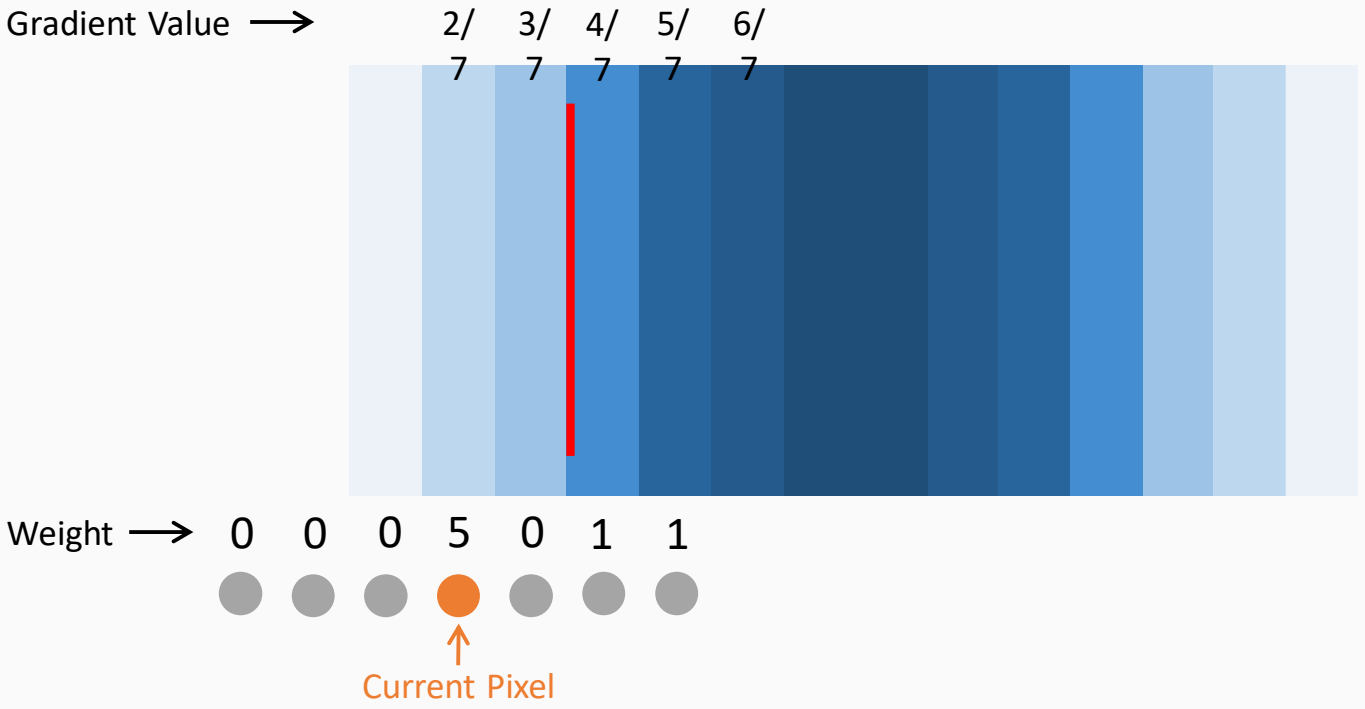
IMPROVED SAMPLE CONTRIBUTION

- For the first outer blur pixel there is an abrupt change in the weights
  - We want renormalize them in such a way that the background receives a 4 to compensate for the samples that are failing (with zeroes)
- We assign to current pixel:
  - 7-3=4



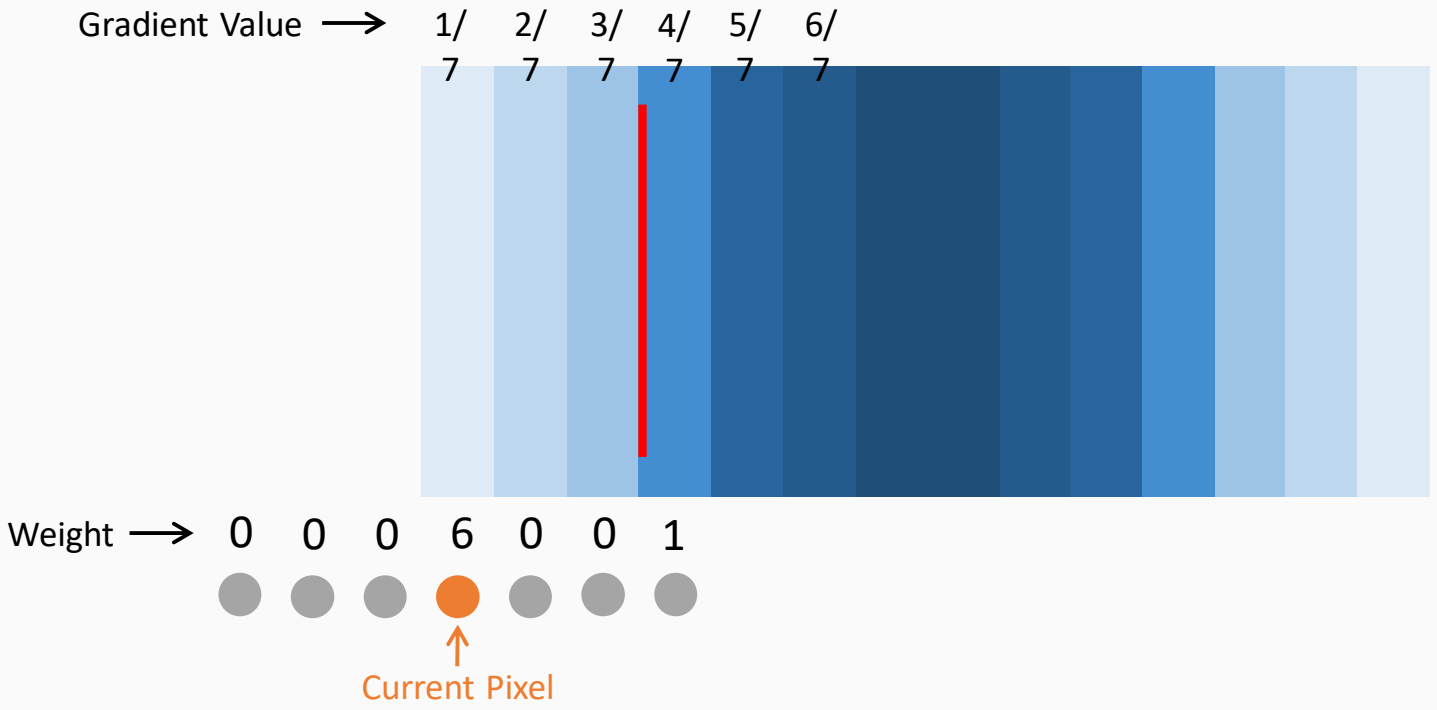
IMPROVED SAMPLE CONTRIBUTION

- We assign to current pixel:
  - 7-2=5



IMPROVED SAMPLE CONTRIBUTION

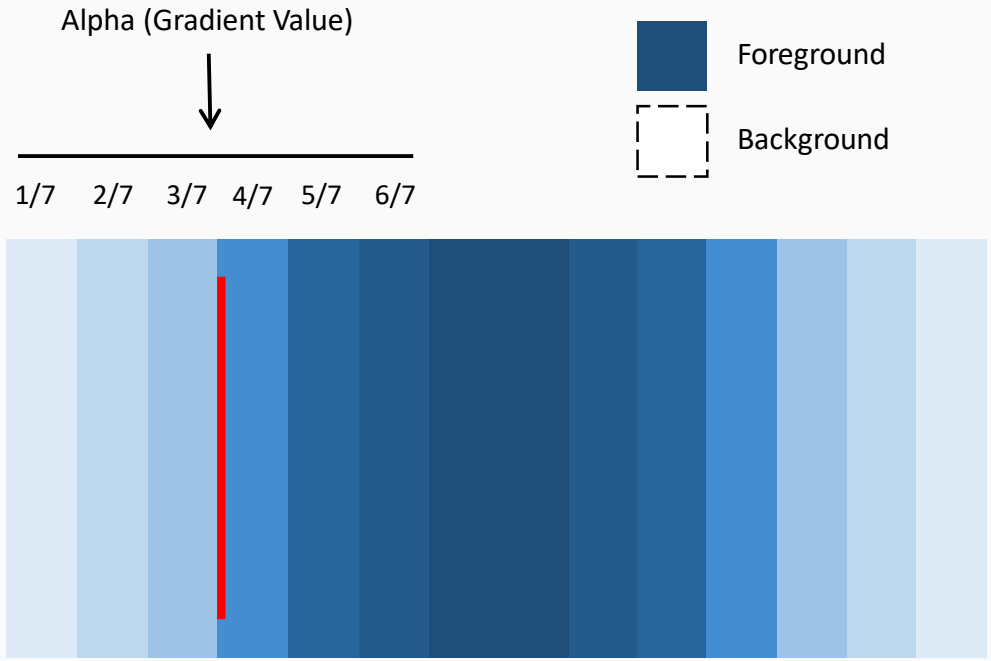
- We assign to current pixel:
  - 7-1=6



IMPROVED SAMPLE CONTRIBUTION

- From a logic stand point, for each pixel we have:
  - Background
  - Foreground
  - Alpha (Gradient Value)

- The actual implementation is a bit different
  - Accumulate background and foreground together as in [McGuire2012] [McGuire2013]
  - Correct the gradient in the end of the shader



IMPROVED SAMPLE CONTRIBUTION

- Works great for constant or low-frequency backgrounds, but it create artifacts on detailed ones

[McGuire2013]

Our approach

MIRRORED BACKGROUND RECONSTRUCTION

- **The issue is that the background comes from two different sources**
  - In the outer blur, the background is readily available
    - The center pixel, which is not blurred
  - In the inner blur, the real background is not available
    - It is reconstructed and approximated by blurring nearby background pixels

Background Reconstruction Artifacts

Notes

MIRRORED BACKGROUND RECONSTRUCTION

- The solution is to blur the background even on areas where we know the exact non-blurred background
- We mirror the blurriness along the silhouette edge

Continue to use the background reconstruction in this area

Notes **Without Blur Mirroring** **With Blur Mirroring**

MIRRORED BACKGROUND RECONSTRUCTION

```
float weight1 = SampleWeight( centerDepth, sampleDepth1, offsetLen, centerVelocityLen,
sampleVelocityLen1, pixelToSampleUnitsScale, MB_SOFTZ_INCHES ); // to the left

float weight2 = SampleWeight( centerDepth, sampleDepth2, offsetLen, centerVelocityLen,
sampleVelocityLen2, pixelToSampleUnitsScale, MB_SOFTZ_INCHES ); // to the right


#if MB_MIRROR_FILTER

bool2 mirror = bool2( rawVelocityDepth1.b > rawVelocityDepth2.b, sampleVelocityLen2 >
sampleVelocityLen1 );

weight1 = all( mirror ) ? weight2 : weight1;

weight2 = any( mirror ) ? weight2 : weight1;

#endif // #if MB_MIRROR_FILTER


sum += weight1 * float4( tex2D( colorMapSampler, float4( sampleTexCoords.xy, 0.0, 1.0 ) ).rgb, 1.0 );

sum += weight2 * float4( tex2D( colorMapSampler, float4( sampleTexCoords.zw, 0.0, 1.0 ) ).rgb, 1.0 );
```

MIRRORED BACKGROUND RECONSTRUCTION

[McGuire2013]

MIRROR BACKGROUND RECONSTRUCTION

Accurate Sample Contribution

MIRRORED BACKGROUND RECONSTRUCTION

Accurate Sample Contribution + Mirrored Background Reconstruction

MIRRORED BACKGROUND RECONSTRUCTION

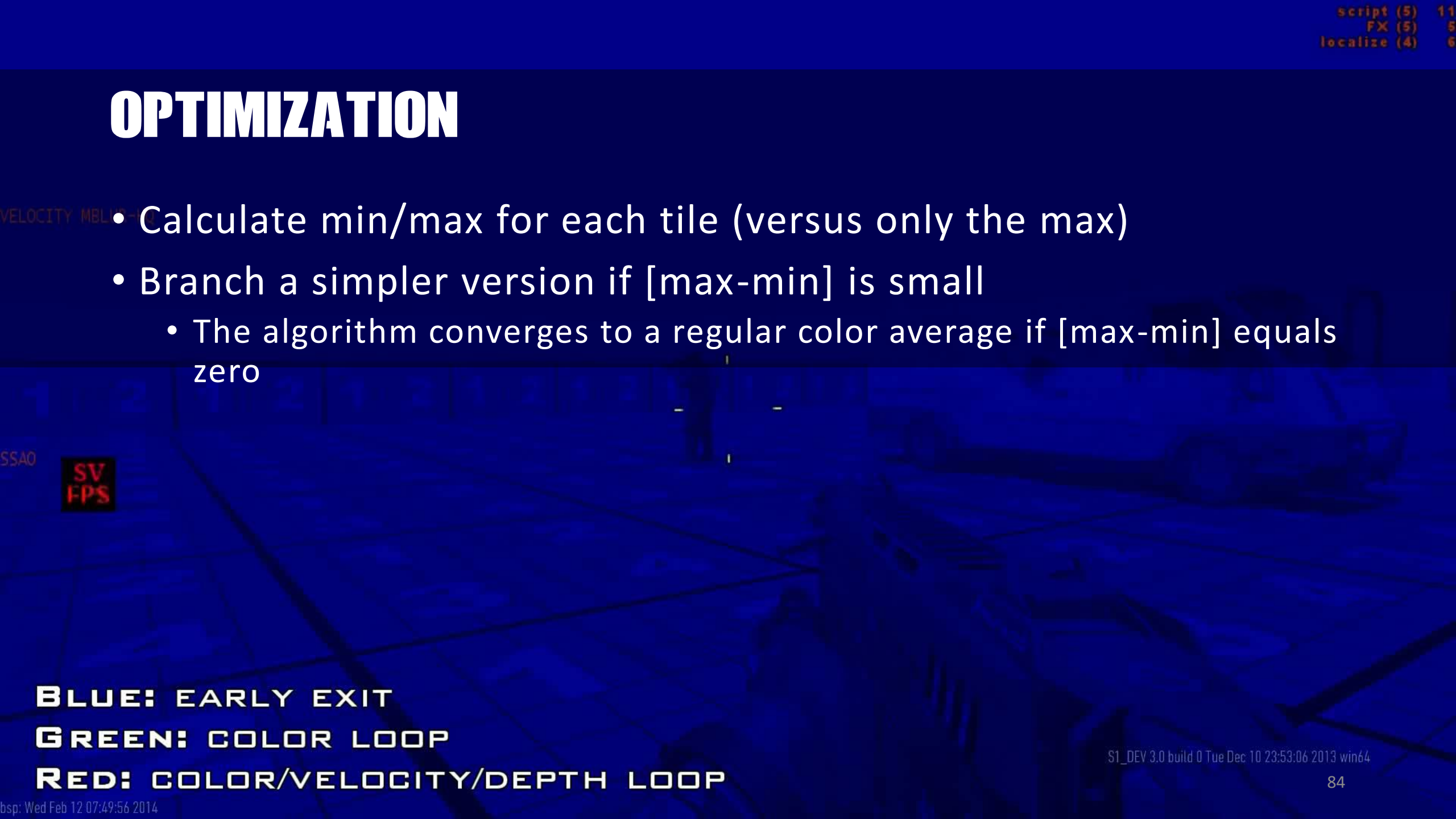
[McGuire2013]

FOCUS – ACCURACY, IQ, PERFORMANCE

- Accuracy
  - Using more accurate sample contributions and background recovery
- Image quality
  - Using better sampling patterns to mask undersampling
- Optimization
  - Running simplified versions of the shader were possible

OPTIMIZATION

- Calculate min/max for each tile (versus only the max)
- Branch a simpler version if [max-min] is small
  - The algorithm converges to a regular color average if [max-min] equals zero



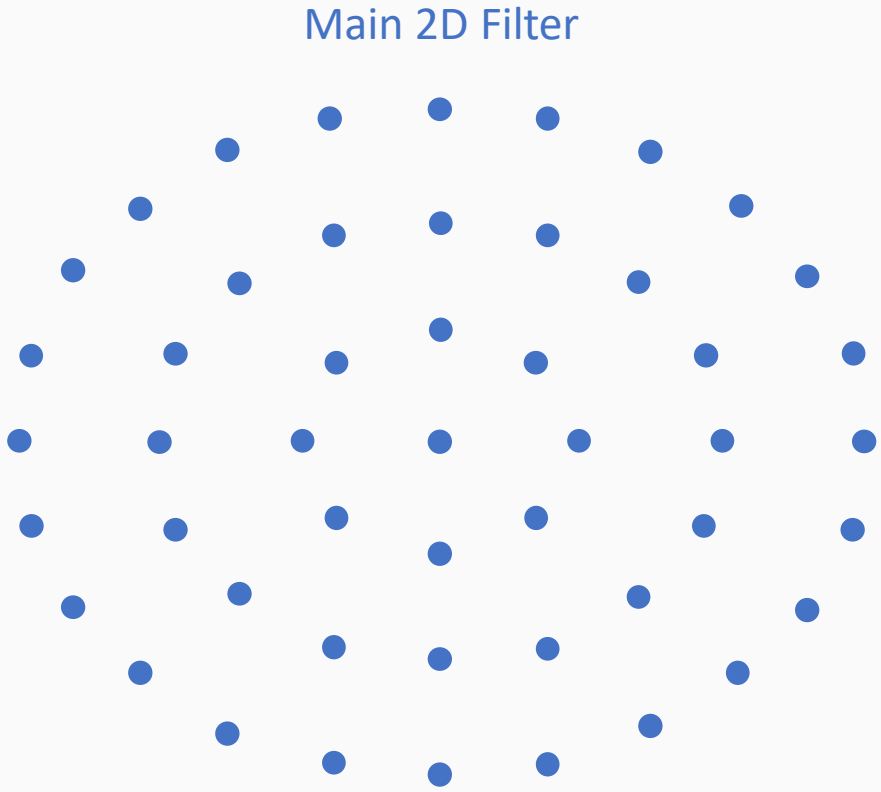
DEPTH OF FIELD

OVERVIEW

- Many ideas from motion blur apply to DOF
  - Scatter as you gather ideas useful for both
- The presented motion blur scatter-as-you-gather approach only covers a single layer (single direction)
- Many can potentially happen and overlap
  - Not much of an issue in our case for motion blur
  - In DOF they are easier to see
  - Moving objects hide issues
- So we developed a 2-layer approach

OVERVIEW

- Experimented with separable hexagonal [White2011] and octagonal [McIntosh2012] shapes
  - Great filter quality, but hard to make them to work with scatter-as-you-gather (because of their separable nature)
- Preferred circular Bokeh because that is the aperture found in our eyes
  - 49-tap 2d filter ran in halfres
- Heavily inspired by:
  - [Sousa2013] for filtering
  - [Lee2008] for sorting



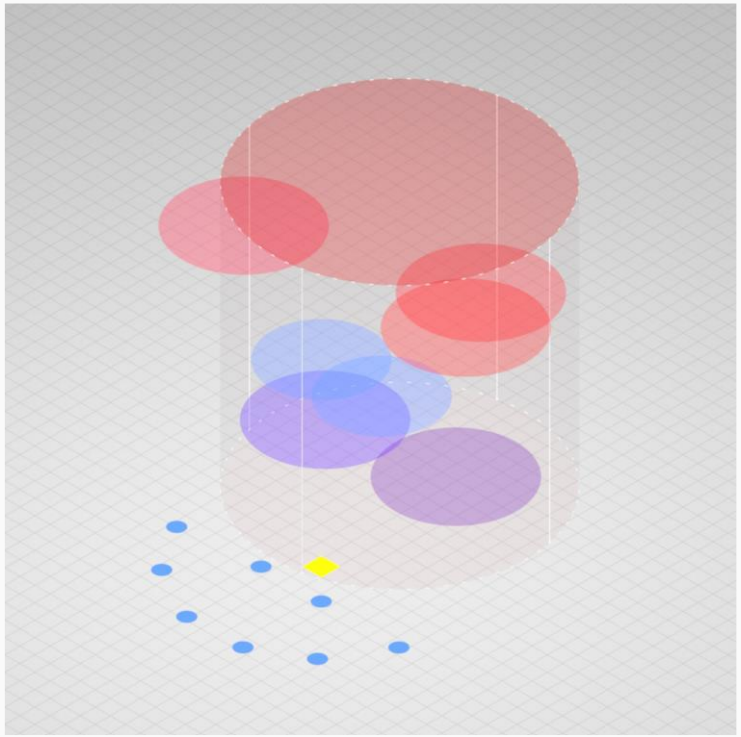
TYPICAL DOF ISSUES

- **Scatter-As-You-Gather Issues**
  - As in motion blur, neighborhood pixels can bleed into current pixel, similar solution
    - Calculate maximum COC in tiles
    - Main 2D Filter radius scaled wit it
  - Calculating each sample contribution
    - Sorting samples and blending them
  - Background reconstruction – See online slides
- **Filter Quality Issues**
  - Undersampling
  - How to use bilinear filtering? – See online slides
- **Performance Issues**
  - Lots of samples
  - Half-res rendering – See online slides

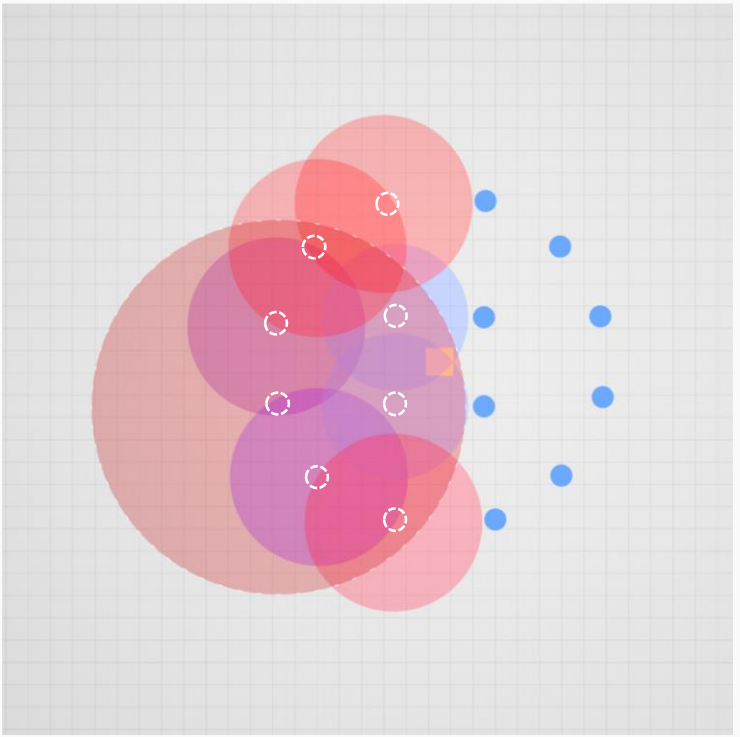
SCATTER-AS-YOU-GATHER – BIG PICTURE

Ideal algorithm

- Check if a sample circle of confusion overlaps current pixel
- Check if it is in the foreground
- Sort overlapping and foreground pixels from background to foreground
- Assign an alpha to each sample
  - $a = \frac{1}{\pi c^2}$
  - Where c is the COC radius
- Blend them
- **Sorting is extremely expensive**



Isometric View

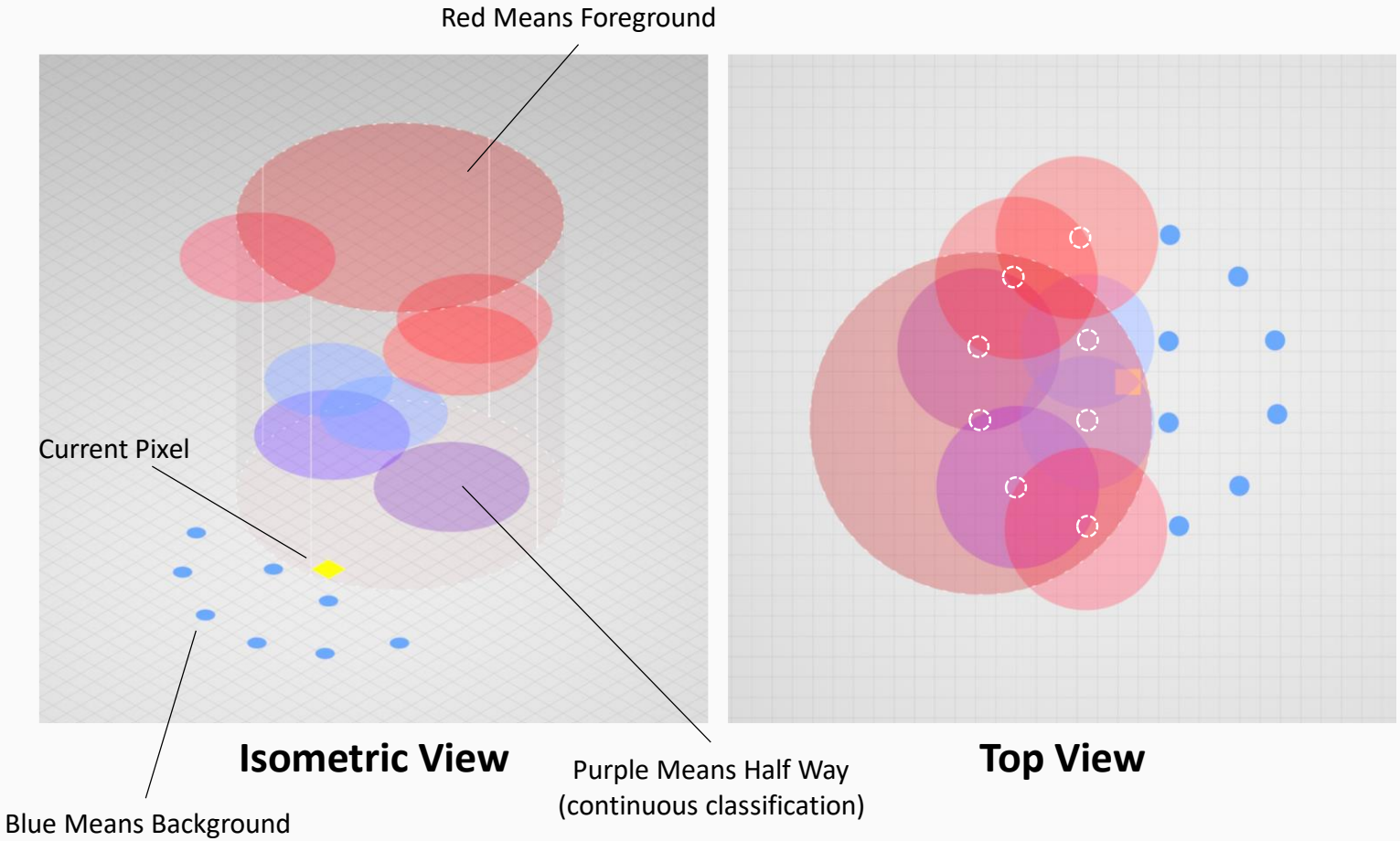


Top View

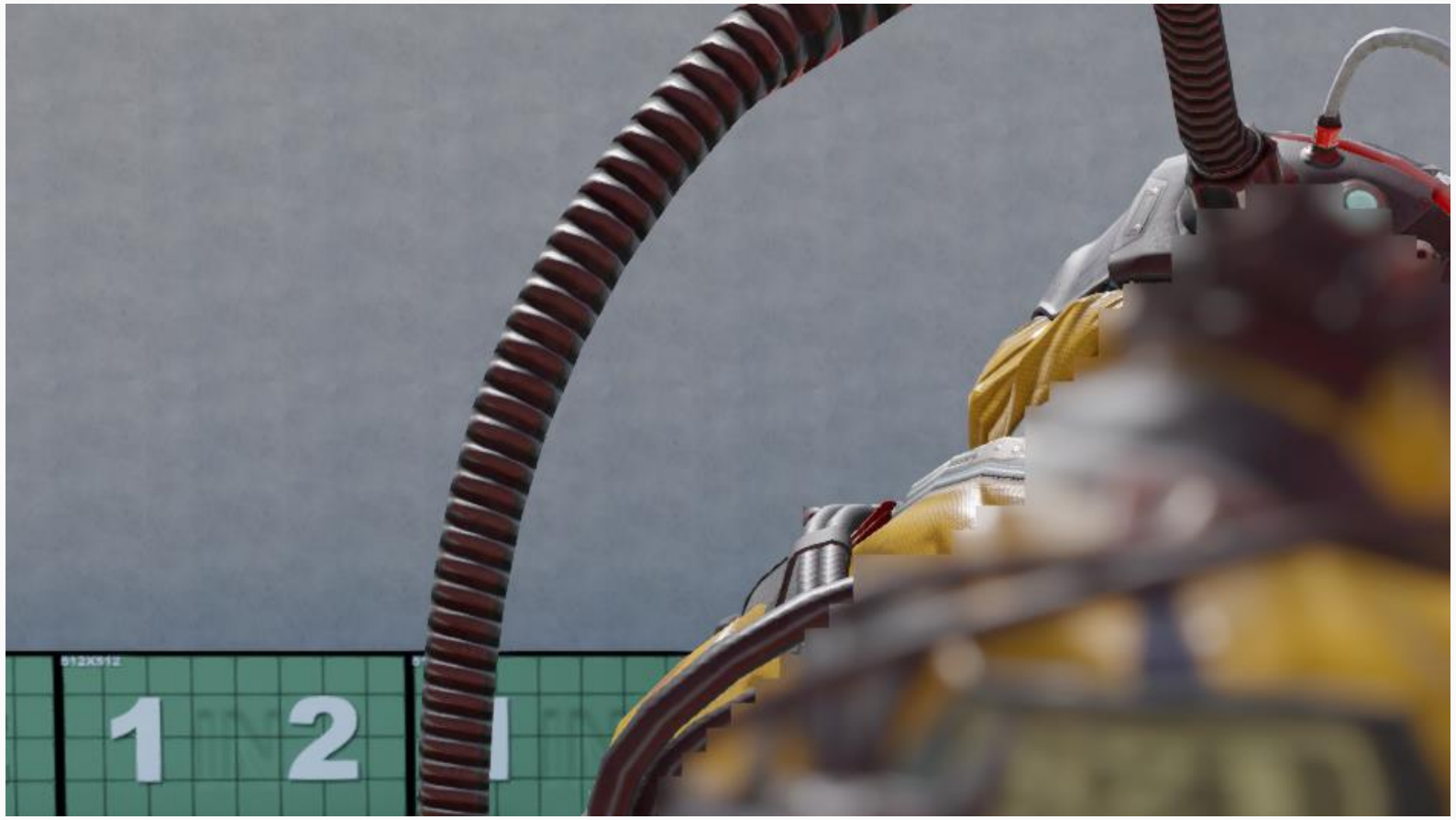
SCATTER-AS-YOU-GATHER – BIG PICTURE

Our sort approach

- Similar in spirit to [Lee2008]
  - Simplified for performance
- Divide samples in two layers
  - Background
  - Foreground
- For each layer
  - Additive alpha blending average
- Calculate foreground alpha
- Alpha blend foreground/background
- Loose foreground/background categorization ramp
  - Distance 0 in: foreground
  - Distance 100 in: background



SCATTER-AS-YOU-GATHER – BIG PICTURE

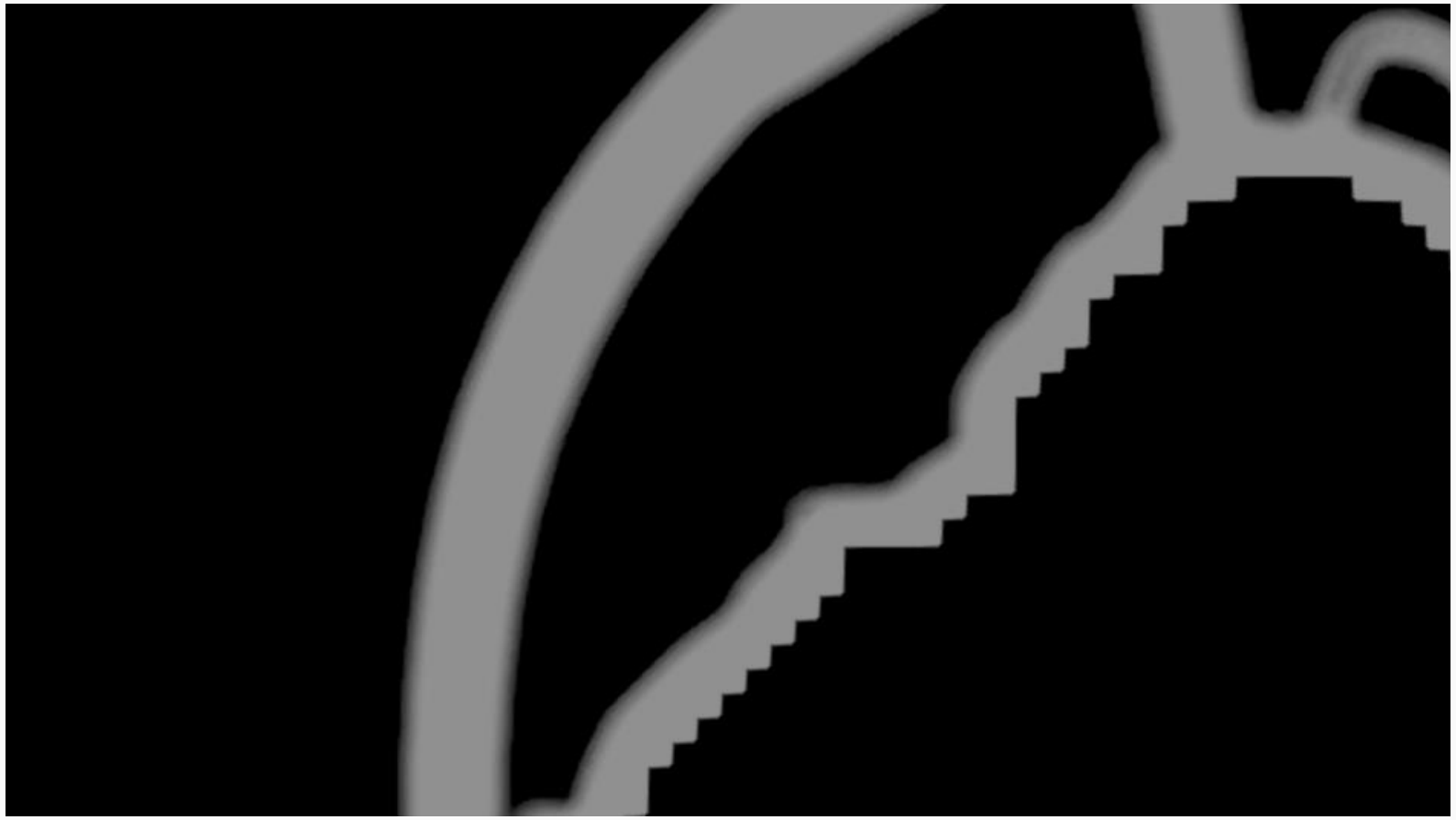


Background

SCATTER-AS-YOU-GATHER – BIG PICTURE

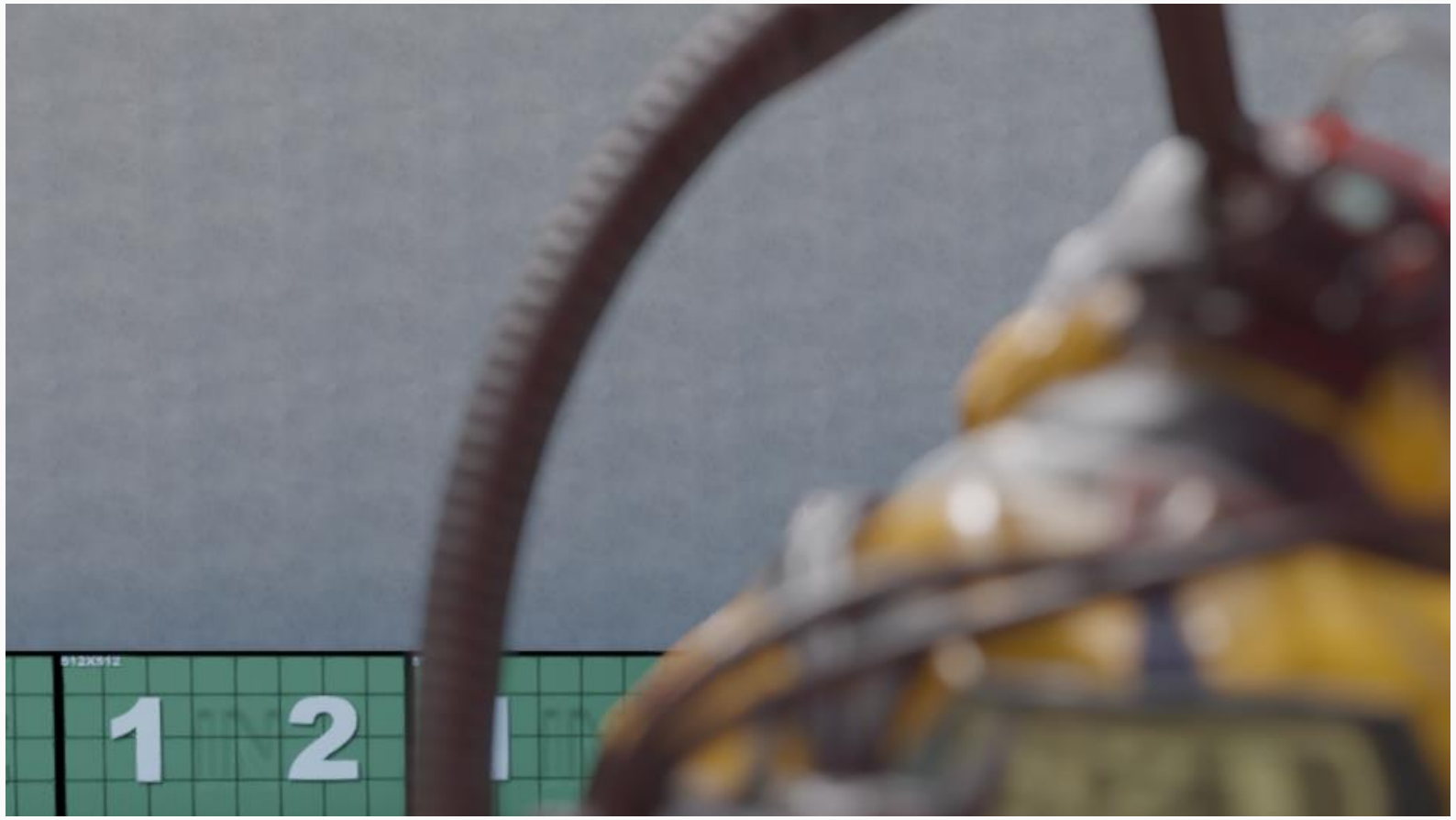
Foreground

SCATTER-AS-YOU-GATHER – BIG PICTURE



Alpha

SCATTER-AS-YOU-GATHER – BIG PICTURE



lerp(Background, Foreground, Alpha)

SCATTER-AS-YOU-GATHER – TILING

- For each tile
  - Compute closest depth and maximum COC
- Gather information from neighbors
  - 3x3 filter, as in Motion Blur

SCATTER-AS-YOU-GATHER – PRESORT

- We run a pass that we call presort
  - For each pixel
    - Calculates its COC
    - Classifies it as background or foreground
    - Weights with the alpha for the additive alpha blending average in the main filter pass
  - Output to a R11G11B10
    - **r:** coc
    - **g:** SampleAlpha() * DepthCmp2().x
    - **b:** SampleAlpha() * DepthCmp2().y
- To determine if a pixel is background or foreground we compare with the closest depth in each tile
- Presorting before filtering reduces ALU and VGPR pressure on the main filter pass

```
float2 DepthCmp2( float depth, float tileMaxDepth )
{
	float d = DOF_DEPTH_SCALE_FOREGROUND * ( depth -
	tileMaxDepth );
	float2 depthCmp;
	depthCmp.x = smoothstep( 0.0, 1.0, d );
	depthCmp.y = 1.0 - depthCmp.x;
	return depthCmp;
}
```

Radius in pixels

```
float SampleAlpha( float sampleCoc )
{
	return min( rcp( PI * sampleCoc * sampleCoc ), PI *
		DOF_SINGLE_PIXEL_RADIUS *
	DOF_SINGLE_PIXEL_RADIUS );
}
```

ALPHA CALCULATION

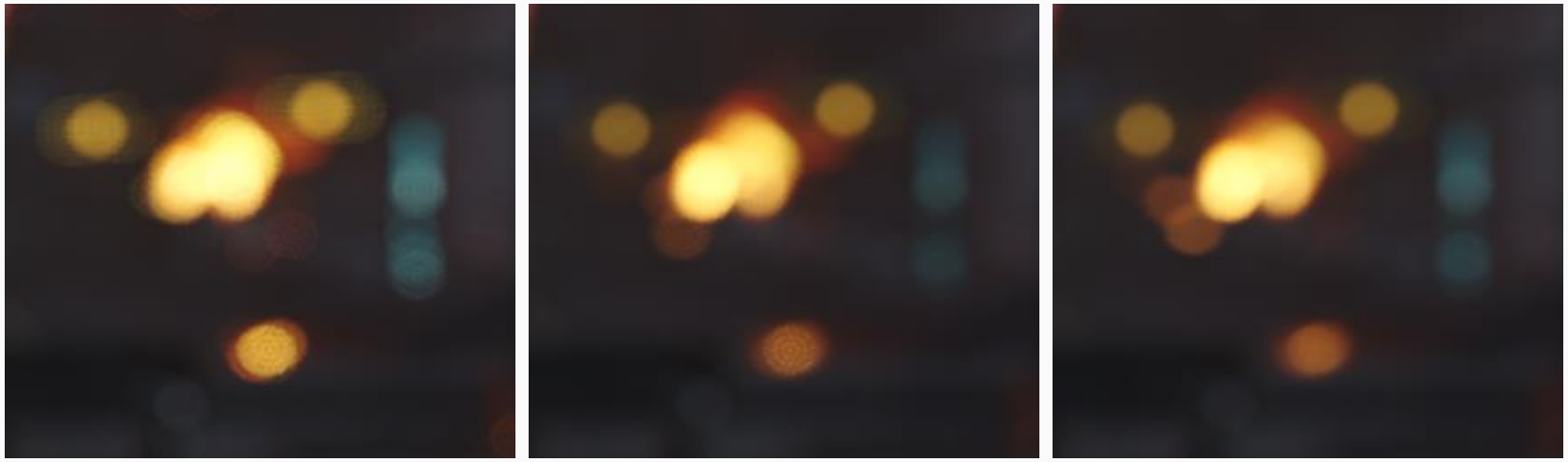
- We have accumulated the samples into two float4 registers, background and foreground
  - **rgb:** accumulated color
  - **a:** accumulated weight
- The alpha comes from additive alpha blending over all samples [Lee2008]
  - foreground.a
- We scatter-as-you-gather (vs. scatter), so it needs normalization

Sampling Area

```
float alpha = saturate( ( 1.0 / SAMPLE_COUNT ) * ( 1.0 / SampleAlpha( tileMaxCoc ) ) *
foreground.a);
```

FIGHTING UNDERSAMPLING

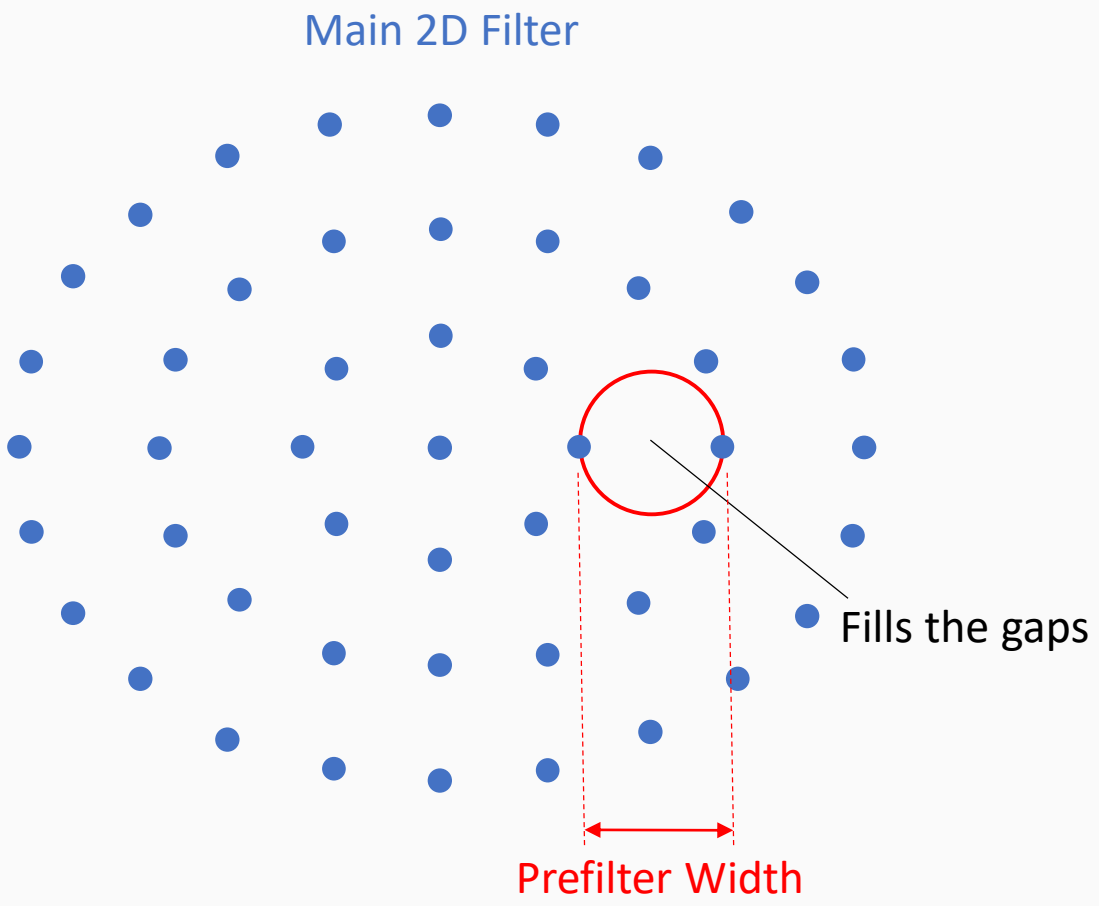
- Two measures
  - Before filtering: **Prefilter**
  - After filtering: **Median**



No Undersampling Measures **Prefilter** **Prefilter + Median**

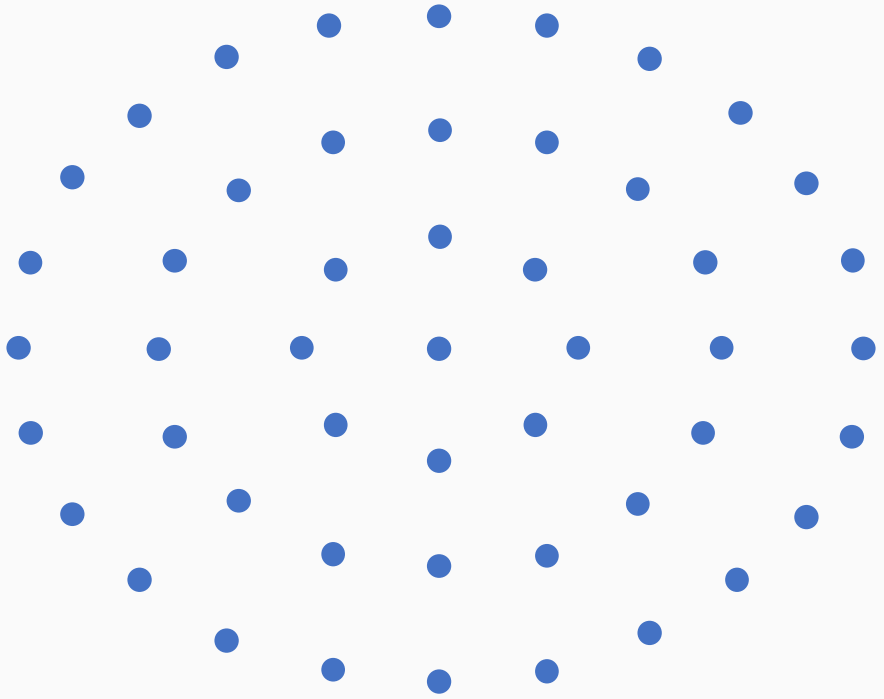
PREFILTER

- Blurring in multiple passes increases quality
  - More effective samples
  - Can't be used with scatter-as-gather
    - Data (Depth & COC) information meaning lost after first pass
  - But It is possible to prefilter color values if the blur follows the surface
    - Depth and COC meaning preserved
- We apply a circular 9-tap bilateral prefilter
  - Radius scaled by the center COC
  - Depth bilateral weighting
  - Width close the main filter sample spacing



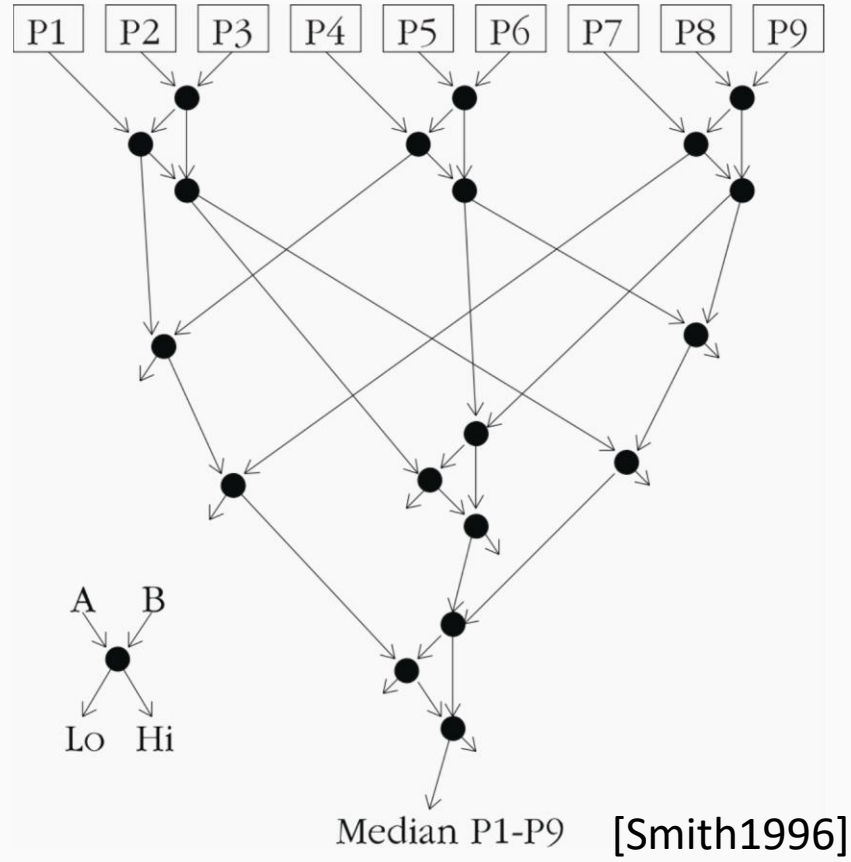
MAIN FILTER PASS

Main 2D Filter



MEDIAN

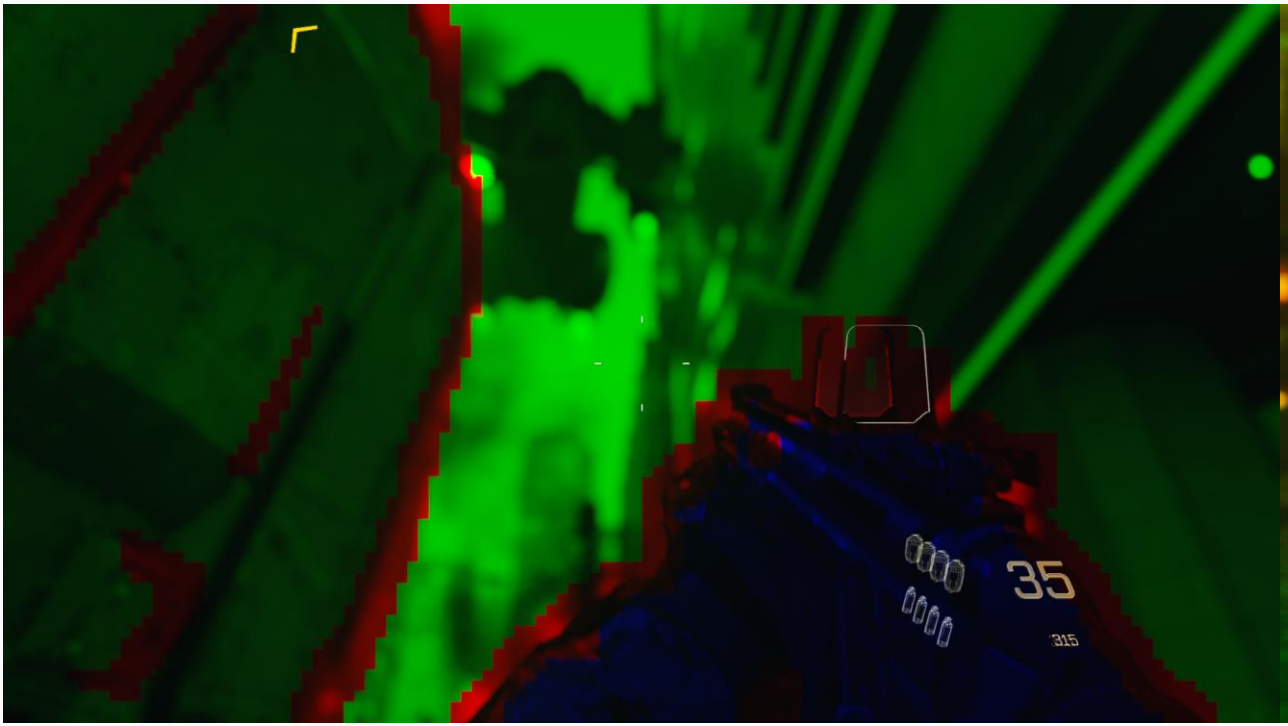
- Given the very high dynamic range, there might still be signs of undersampling
  - And signs of noise (see next slides)
- Similar in spirit to [Sousa2013] we apply a postfilter
  - Median instead of Max
- 3x3 Median on halfres buffer
  - Using [Smith1996]
  - Hardware min3/max3/med3 friendly
- Median vs. Max
  - Median more effective in our case
  - Median do not grow highlights



[Smith1996]

OPTIMIZATIONS

Color Only Loop – Color/Data Loop – Early Exit



Tiling Optimization

Similar to Motion Blur

1 Ring – 2 Rings – 3 Rings

Ring optimization

Reduce sample counts according to the COC [Valient2013]

SHADOW SAMPLING

OVERVIEW

- Being a 60-fps title, the constraints on the shadow filtering are extremely challenging, as there is not budget for many taps
- We experimented with randomly per-pixel rotated Poisson disks
- We found them to produce low-quality results, with moderate sample counts (eight)
- Very unstable under motion

OVERVIEW

```
values.xy = sv_position;
values.xy = frac( dot( values.xy, magic.xy ) );
values.xy = frac( dot( values.xy, magic.xy ) );
…
values.xy = frac( dot( values.xy, magic.xy ) );
```

OVERVIEW

- Experimenting and optimizing a noise generator, we found a noise function that we could classify as being half way between dithered and random, and that we called **Interleaved Gradient Noise**:

```
float3 magic = float3( 0.06711056, 0.00583715, 52.9829189 );
return -scale + 2.0 * scale * frac( magic.z * frac( dot( sv_position, magic.xy ) ) );
```

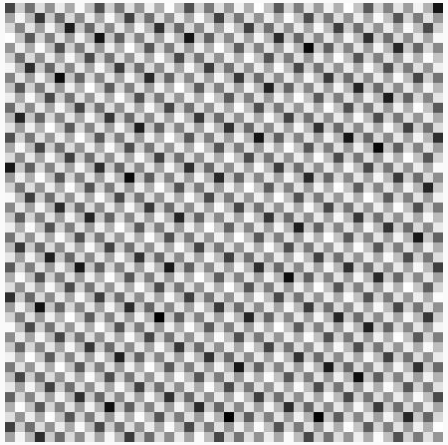
- Used to rotate the samples:

```
sincos( 2.0 * PI * InterleavedGradientNoise( sv_position ), rotation.y, rotation.x );
float2x2 rotationMatrix = { rotation.x, rotation.y, -rotation.y, rotation.x };
...
float2 sampleOffset = mul( offsets[i], rotationMatrix );
```

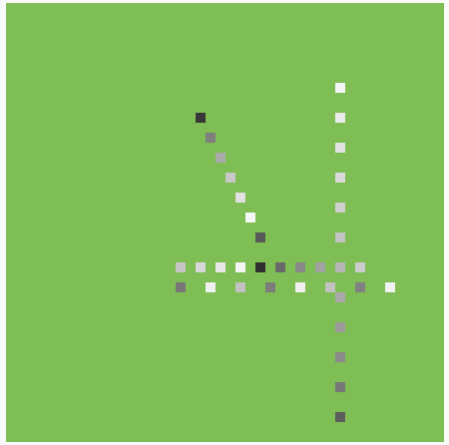
- **Best of both worlds:**
  - Produces a rich range of values, similar to random noise
  - Produces temporally coherent results, similar to dithered schemes

INTERLEAVED GRADIENT NOISE

- This noise produces interleaved gradients
- This means that objects moving at a constant speed will rotate the samples smoothly
- To make it work for static images, scroll the horizontal position
  - sv_position.x += scale * time



Interleaved Gradient Noise



Interleaved Gradients

PERFORMANCE ON XBOX ONE

- Veil (Fixed): 0.35ms
- Motion Blur (Running): 0.52ms
- Depth of Field (Portrait): 1.27ms
- Subsurface Scattering (Portrait): 0.46ms

QUESTIONS AND ACKS – IN ALPHABETICAL ORDER

Michael Condrey and Glen Schofield for kindly allowing me to present this work
Natalya Tatarchuk for inviting us to the course

- Angelo Pesce
- Adam Micciulla
- Christer Ericson
- Danny Chan
- Dave Blizzard
- Dave Cowling
- Javier Pahlen
- Jian Zhang
- Jose Checa
- Maria Lopez
- Martin Kraus
- Michael Vance
- Morgan McGuire
- Padraic Hennessy
- Paul Edelstein
- Peter-Pike Sloan
- Stephanus
- Wade Brainerd
- Xian-Chun Wu

REFERENCES

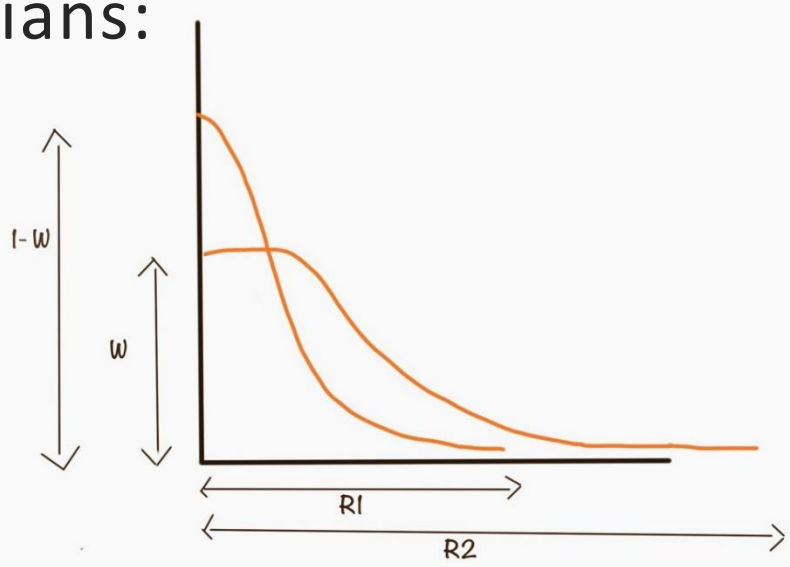
- [Cook1984] Distributed ray tracing
- [Cook1987] The Reyes Image Rendering Architecture
- [Haeberli1990] The Accumulation Buffer: Hardware Support for High-Quality Rendering
- [Jimenez2013] Next Generation Character Rendering
- [Karis2013] http://graphicrants.blogspot.com.es/2013/12/tone-mapping.html
- [Kawase2011] Anti-Downsized Buffer Artifacts
- [Kraus2007] Pyramid Filters Based On Bilinear Interpolation
- [McGuire2010] Real-Time Stochastic Rasterization on Conventional GPU Architectures
- [McGuire2012] A Reconstruction Filter for Plausible Motion Blur
- [McGuire2013] A Fast and Stable Feature-Aware Motion Blur Filter
- [McIntosh2012] Efficiently Simulating the Bokeh of Polygonal Apertures in a Post-Process Depth of Field Shader
- [Mittring2012] The Technology Behind the Elemental Demo
- [Smith1996] Implementing median filters in XC4000E FPGAs
- [Sousa2013] CryENGINE 3 Graphics Gems
- [Strengert2006] Pyramid Methods in GPU-Based Image Processing
- [Valient2013] Killzone Shadow Fall Demo Postmortem
- [Ward1997] A Visibility Matching Tone Reproduction Operator for High Dynamic Range Scenes
- [White2011] More Performance! Five Rendering Ideas from Battlefield 3 and Need for Speed: The Run

BONUS SLIDES

SUBSURFACE SCATTERING

SUBSURFACE SCATTERING

- We are using the Separable Subsurface Scattering approach [Jimenez2013]
- 2-pass screen space blur (vertical + horizontal)
- Using noise to hide undersampling artifacts (random rotation)
- Shape for the filter is a sum of two Gaussians:
  - R1: near scattering (float3)
  - R2: far scattering (float3)
  - W: blend factor (float)



SUBSURFACE SCATTERING

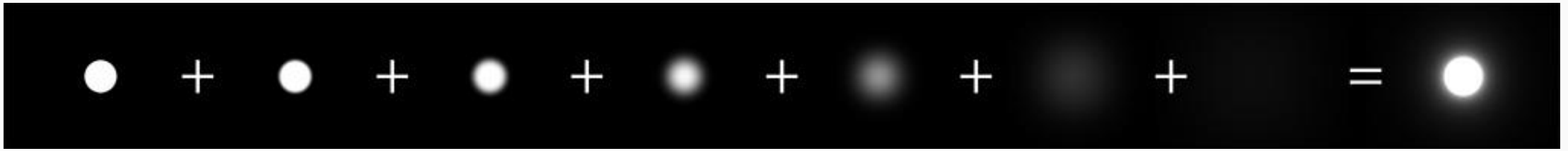
- **Following [Mikklesen2010] Skin Rendering by Pseudo-Separable Cross Bilateral Filtering, we have done various improvements:**
  - Using Importance Sampling
    - Only 9 samples per pass (but 7 is plausible in most of the cases)
  - No longer using hacks for depth differences
    - From a high level perspective, we convert the positions to world space, and apply the profile on the fly. It enables to better approximate ground truth (except where information is missing in screen space)
    - From a low level perspective, it has been heavily optimized (at assembly level), with the XY component of the profile being baked in the CPU and the Z component being evaluated on the fly (see derivation on the right)
    - 16 instruction slots per sample
- **Special thanks to our mathematician Xian-Chun Wu!**

$$
\begin{aligned}
s[\sqrt{d_{xy}^2+d_z^2}] &= wG(\sqrt{d_{xy}^2+d_z^2},\sigma_{near})+ \\
&\quad (1-w)G(\sqrt{d_{xy}^2+d_z^2},\sigma_{far}) \\
&= we^{\frac{-d_z^2}{2\sigma_{near}}}G(d_{xy},\sigma_{near})+ \\
&\quad (1-w)e^{\frac{-d_z^2}{2\sigma_{far}}}G(d_{xy},\sigma_{far}) \\
&\approx e^{\frac{-d_z^2}{2\sigma_{max}}}(wG(d_{xy},\sigma_{near})+ \\
&\quad (1-w)G(d_{xy},\sigma_{far})) \\
&= e^{\frac{-d_z^2}{2\sigma_{max}}}s'[d_{xy}],
\end{aligned}
$$

BLOOM/VEIL

TYPICAL IMPLEMENTATION

- Thresholded color used as input
- Downscale input image into mipmap chain
- Upscales each mip to the original image resolution
- Adds mips to original image
- Gaussian blurred somewhere in bloom/veil pipeline



OUR BLOOM/VEIL CORE FEATURES

- **Aims to improve the perception of a wider dynamic range**
  - In a non-intrusive way
- **Non-thresholded color used as input for bloom**
  - As suggested by [Ward1997]
  - Our dynamic range (physically based rendering) is very wide
  - Aims to yield natural results, as if we were looking with our eyes
  - Scattering in our eyes, like SSS, is not a thresholded phenomenon
- **Temporally stable**
  - Wide dynamic range makes it harder

TEMPORAL STABILITY

Sources of temporal instability

- Filtering
  - Downsampling: information must fit in the lower mips
  - Upsampling: information must be correctly synthesized
- Fireflies (very bright subpixels)

FILTERING POSSIBILITIES

- #1
  - Downsample
  - Filter each MIP
  - Upscale each MIP
- #2
  - Downsample and filter
  - Upscale
- #3
  - Downsample
  - Upscale and filter
- **#4**
  - **Downsample and filter**
  - **Upscale and filter**

- Filtering both when downsampling AND Upsampling is key [Mittring2012]
  - **Downsampling:** for preventing aliasing artifacts
  - **Upscaling:** for image quality and smooth results
- You can use the filter to blur the image and to analyze (downsample) or synthesize (upsample) at the same time

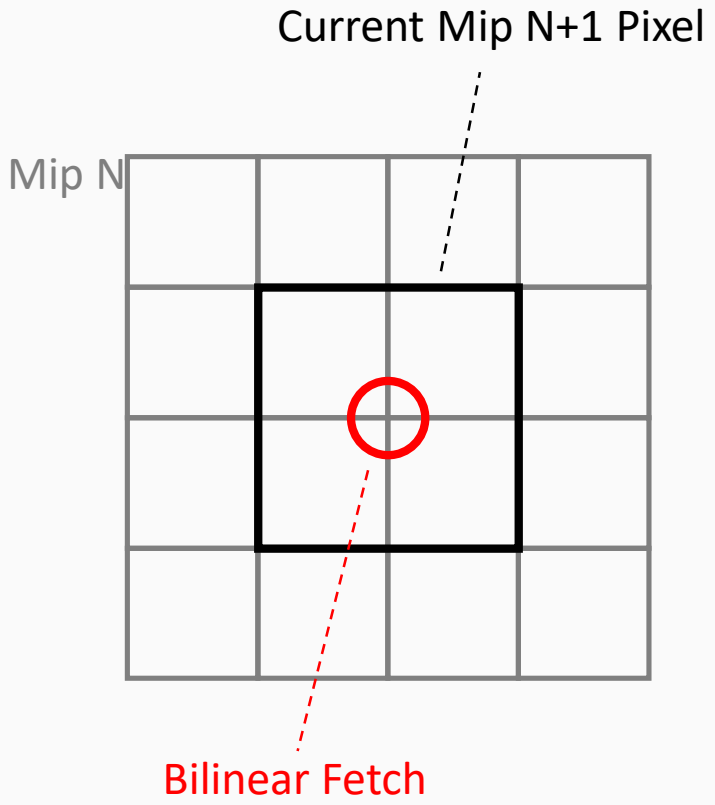
RESAMLPING

- Bilinear downsampling/upsampling generates low-quality results
- Many filtering possibilities [Kraus2007]
- Bicubic do not work well due to the negative lobes [Kawase2011]
- Gaussian possibly the best option for bloom [Kawase2011]

DOWNSAMPLING – 1 FETCH

- Average 4-texel (1 bilinear fetch)
- Typical way to create mip chains
- Creates pulsating artifacts and temporal stability issues

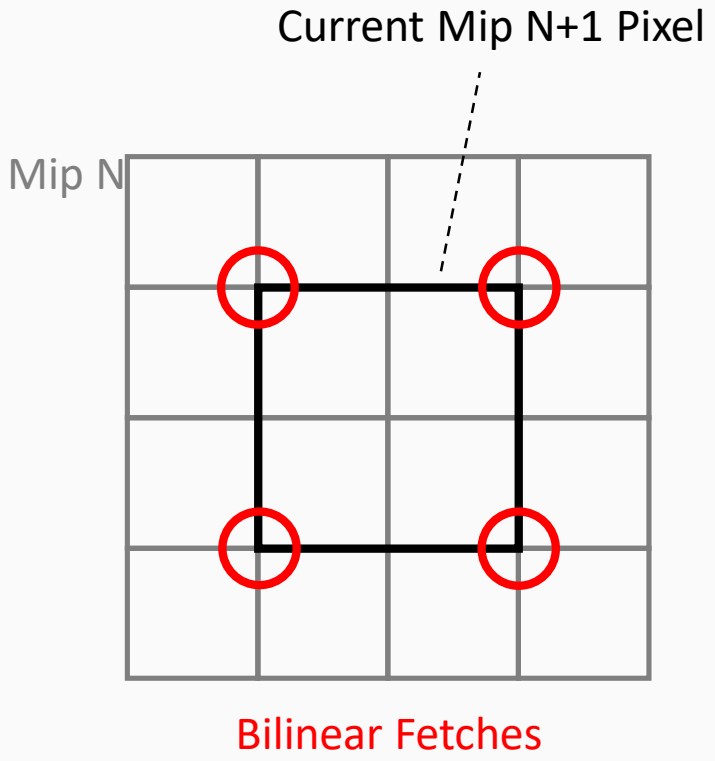
- Called $h_{box}^{2\times2}$ in [Kraus2007] notation
- Infinite $h_{box}^{2\times2}$ downsamples would be equivalent to a box filter (discontinuous)



DOWNSAMPLING – 4 FETCHES

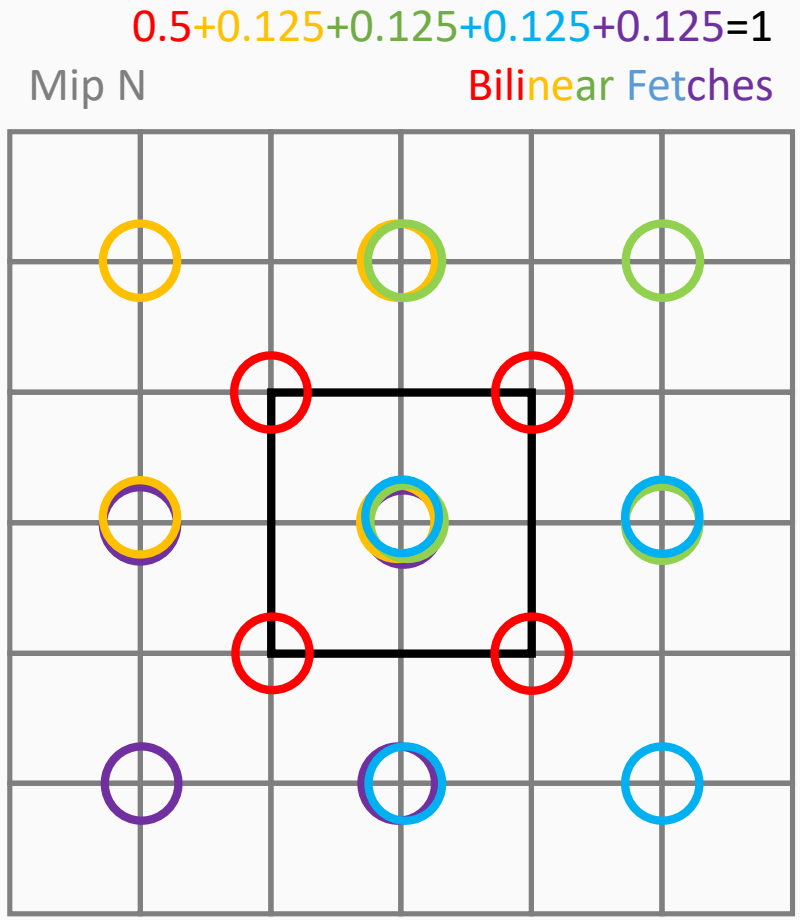
- Average 16-texel (4 bilinear fetches)
- Mitigates the issues – but does not completely fix them (unless wide filter radii is used)

- Called $h_{box}^{4\times4}$ in [Kraus2007] notation
- Infinite $h_{box}^{4\times4}$ downsamples would be equivalent to a continuous filter (similar to a triangle filter, but with a flat constant part)
- Used in Unreal Engine 4 [Mittring2012]



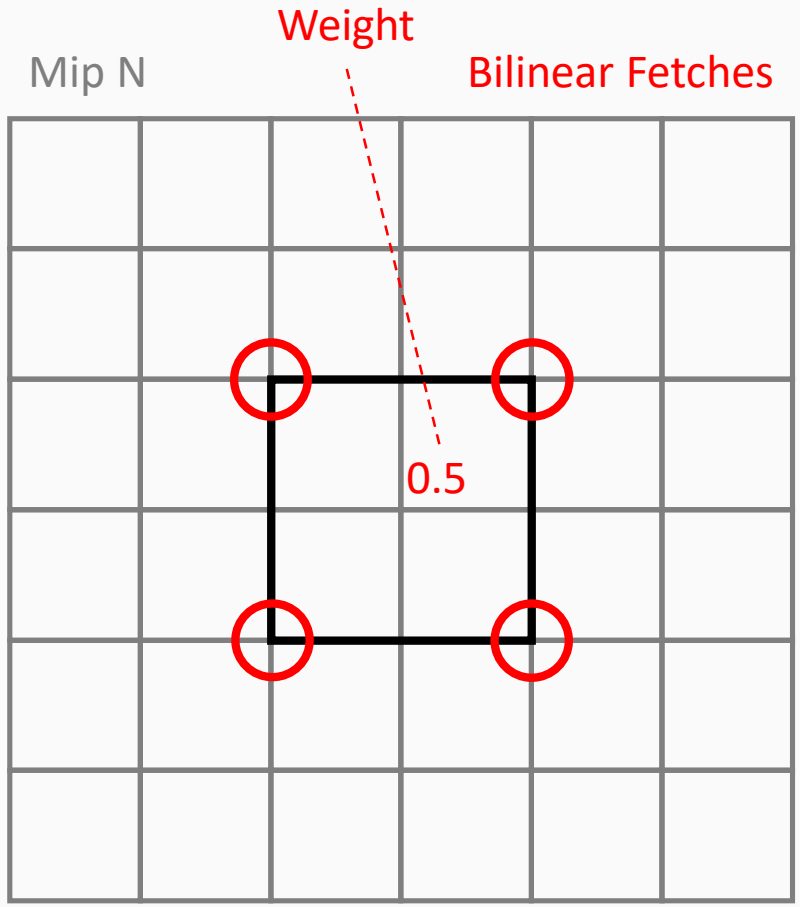
DOWNSAMPLING – OUR SOLUTION

- Custom hand-crafted 36-texel downsample (13 bilinear fetches)
- Created by a weighted average of one-pixel shifts of $h_{box}^{4\times4}$
- Intuitively, the overlapping further breaks the grid nature of pyramid approaches
- Works better than comparable-sized filters
- **Eliminates the pulsating artifacts and temporal stability issues**



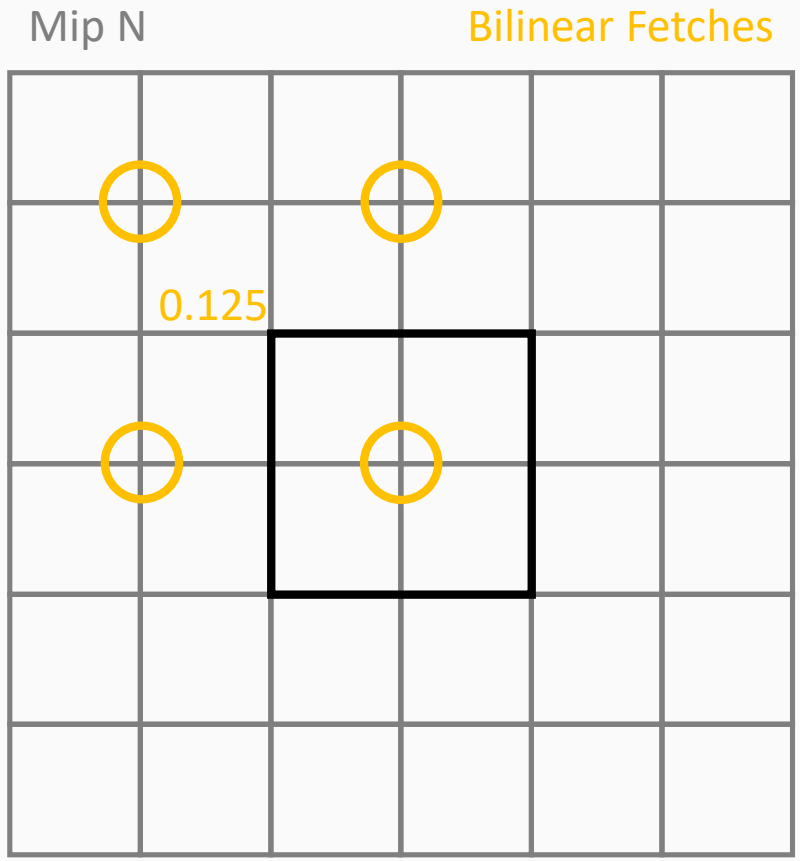
DOWNSAMPLING – OUR SOLUTION

- Custom hand-crafted 36-texel downsample (13 bilinear fetches)
- Created by a weighted average of one-pixel shifts of $h_{box}^{4\times4}$
- Intuitively, the overlapping further breaks the grid nature of pyramid approaches
- Works better than comparable-sized filters
- **Eliminates the pulsating artifacts and temporal stability issues**



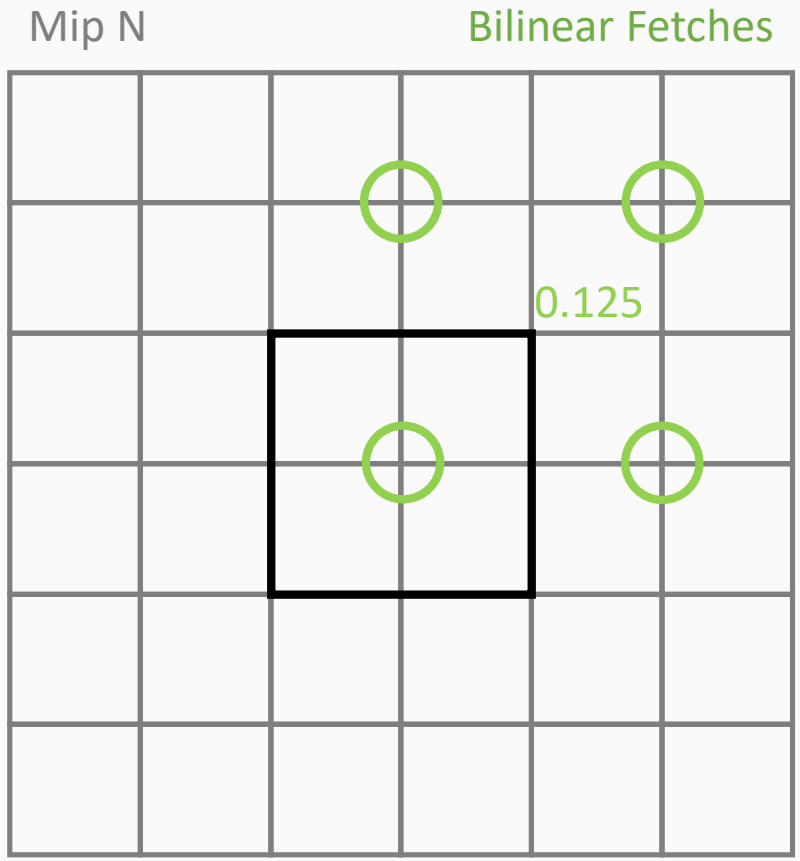
DOWNSAMPLING – OUR SOLUTION

- Custom hand-crafted 36-texel downsample (13 bilinear fetches)
- Created by a weighted average of one-pixel shifts of $h_{box}^{4\times4}$
- Intuitively, the overlapping further breaks the grid nature of pyramid approaches
- Works better than comparable-sized filters
- **Eliminates the pulsating artifacts and temporal stability issues**



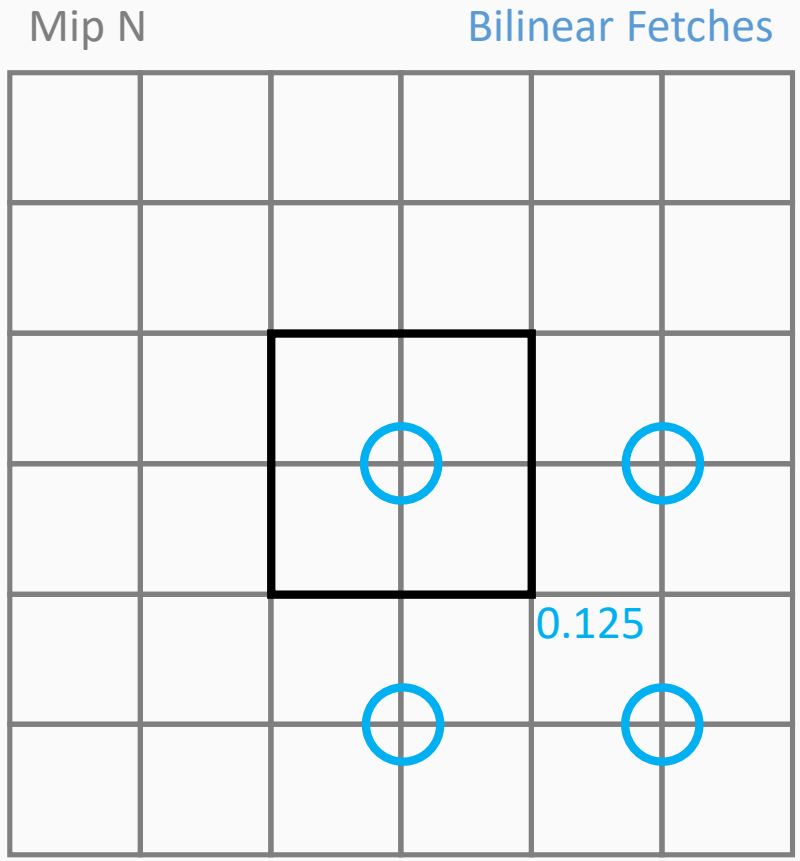
DOWNSAMPLING – OUR SOLUTION

- Custom hand-crafted 36-texel downsample (13 bilinear fetches)
- Created by a weighted average of one-pixel shifts of $h_{box}^{4\times4}$
- Intuitively, the overlapping further breaks the grid nature of pyramid approaches
- Works better than comparable-sized filters
- **Eliminates the pulsating artifacts and temporal stability issues**



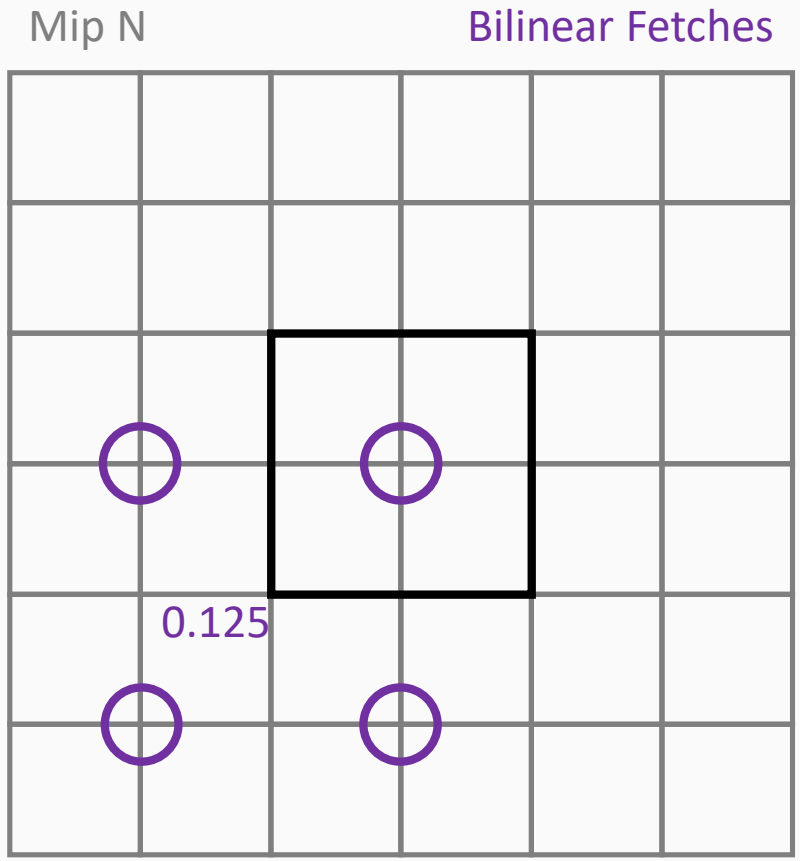
DOWNSAMPLING – OUR SOLUTION

- Custom hand-crafted 36-texel downsample (13 bilinear fetches)
- Created by a weighted average of one-pixel shifts of $h_{box}^{4\times4}$
- Intuitively, the overlapping further breaks the grid nature of pyramid approaches
- Works better than comparable-sized filters
- **Eliminates the pulsating artifacts and temporal stability issues**



DOWNSAMPLING – OUR SOLUTION

- Custom hand-crafted 36-texel downsample (13 bilinear fetches)
- Created by a weighted average of one-pixel shifts of $h_{box}^{4\times4}$
- Intuitively, the overlapping further breaks the grid nature of pyramid approaches
- Works better than comparable-sized filters
- **Eliminates the pulsating artifacts and temporal stability issues**



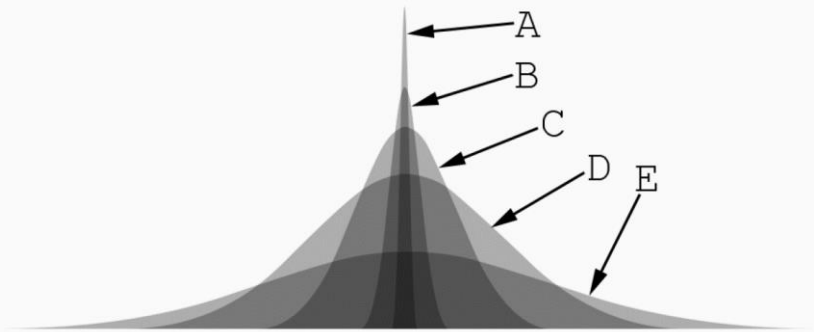
UPSAMPLING

- **Upsampling progressively yields the best results [Kraus2007] [Kawase2011] [Mittring2012]**
  - Never bilinearly upsample mip6 to mip0 resolution directly
  - Upsample from mip6 to mip5, from mip5 to mip4 and so on
- **Why this works? [Kraus2007]**
  - Progressively natural bilinearly upsampling is equivalent to bi-quadratic b-spline filtering
    - "Natural" means to sample in finer MIP pixel centers
  - If you box blur at each upscale step (with a $h_{box}^{2\times 2}$, for example), you get a higher order b-spline upscale

UPSAMPLING

- **Added bonus of upscaling progressively: you can also do the sum with previous mip as you upscale [Mittring2012]**
  - Reduces memory fetches required at higher resolutions

```
E’= blur(E,b5)
D’= blur(D,b4)+E’
C’= blur(C,b3)+D’
B’= blur(B,b2)+C’
A’= blur(A,b1)+B’
```



UPSAMPLING – OUR SOLUTION

- Progressive upsample (and sum the chain as you upscale) as found in [Mittring2012]
- We use a 3x3 tent filter $\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$
- Scaled by a radius parameter
  - The filter do not map to pixels, has “holes” in it
- Repeated convolutions converge to a Gaussian [Kraus2007]
- Simple and fast

TEMPORAL STABILITY

Typically hampered by

- Filtering artifacts
  - Downsampling: information must fit in the lower mips
  - Upsampling: information must be correctly synthesized
- Fireflies (very bright subpixels)

FIREFLIES

- HDR physically-based rendering often produces flickering bright pixels as result of image undersampling
- [Karis2013] proposes reducing the dynamic range before averaging

$$weight = \frac{1}{1 + luma}$$

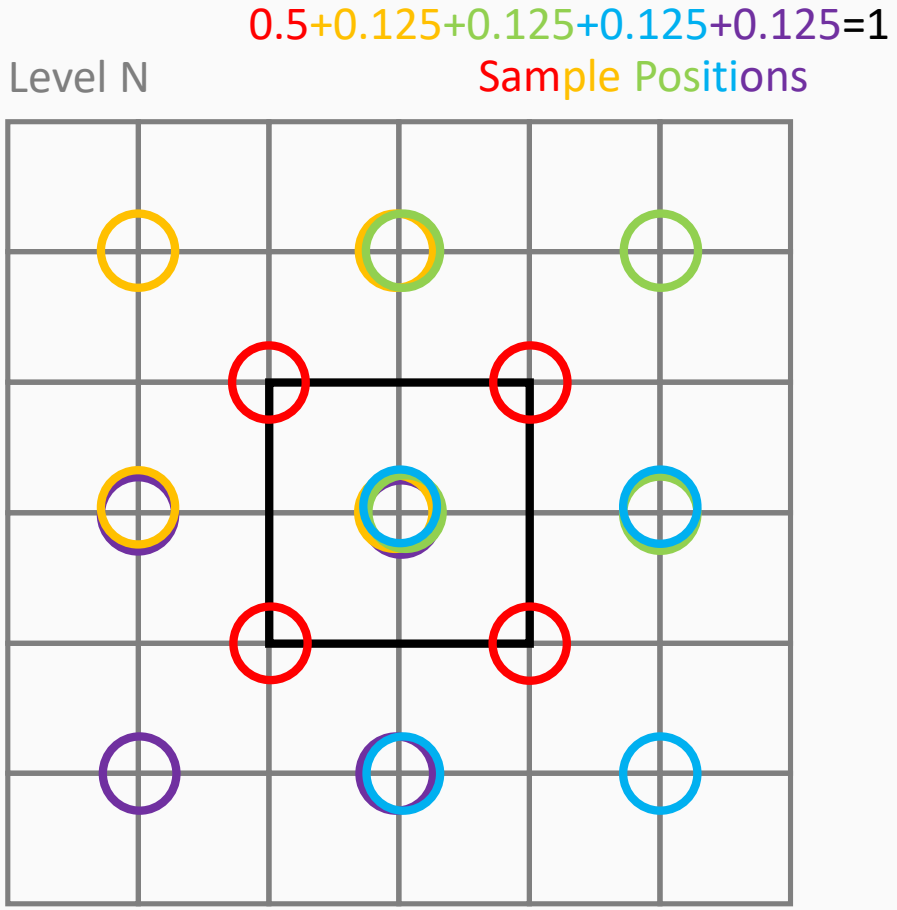
- **We apply a similar technique for the mip0 to mip1 downsample**

FIREFLIES

- We found that any non-linear intensity mapping before averaging yields produces pulsating artifacts
  - On small objects that move in steps similar to their size
- Subpixel motion would solve this issue (multi/supersampling)
  - But that is not universally available

FIREFLIES – PARTIAL KARIS AVERAGE

- We found that doing the Karis average on each $h_{box}^{4\times4}$ separately mitigates the issue
- Intuitively, it allows to recover linear motion of small objects after the fireflies are removed by each separate $h_{box}^{4\times4}$ filter
  - Similar to supersampling in practice



FIREFLIES – PARTIAL KARIS AVERAGE

- **Full Karis Average:** apply the Karis average to all 13-tap together
- **Partial Karis Average:** apply the Karis average in blocks of 4 samples
  - Our approach

- See the comparison in the next movie
  - In both cases we only apply the Karis average on the mip0 to mip1 downsample

BLOOM–VEIL WRAP UP

- **Karis average for MIP0 to MIP1 downsample**
- **Filtering using small 2D kernels**
  - 13-tap custom convolution for downsampling
  - 9-tap convolution for upsampling (3x3 Bartlett)
- **6 R11G11B10 mips**