Stable SSAO in Battlefield 3 with Selective Temporal Filtering

Louis Bavoil
Developer Technology Engineer
NVIDIA

Johan Andersson
Rendering Architect
DICE / EA

GAME DEVELOPERS CONFERENCE
SAN FRANCISCO, CA
MARCH 5-9, 2012
EXPO DATES: MARCH 7-9
2012

SSAO

- Screen Space Ambient Occlusion (SSAO)
  - Has become de-facto approach for rendering AO in games with no precomputation
  - Key Idea: use depth buffer as approximation of the opaque geometry of the scene
  - Large variety of SSAO algorithms, all taking as input the scene depth buffer

[Kajalin 09]
[Loos and Sloan 10]
[McGuire et al. 11]

HBAO

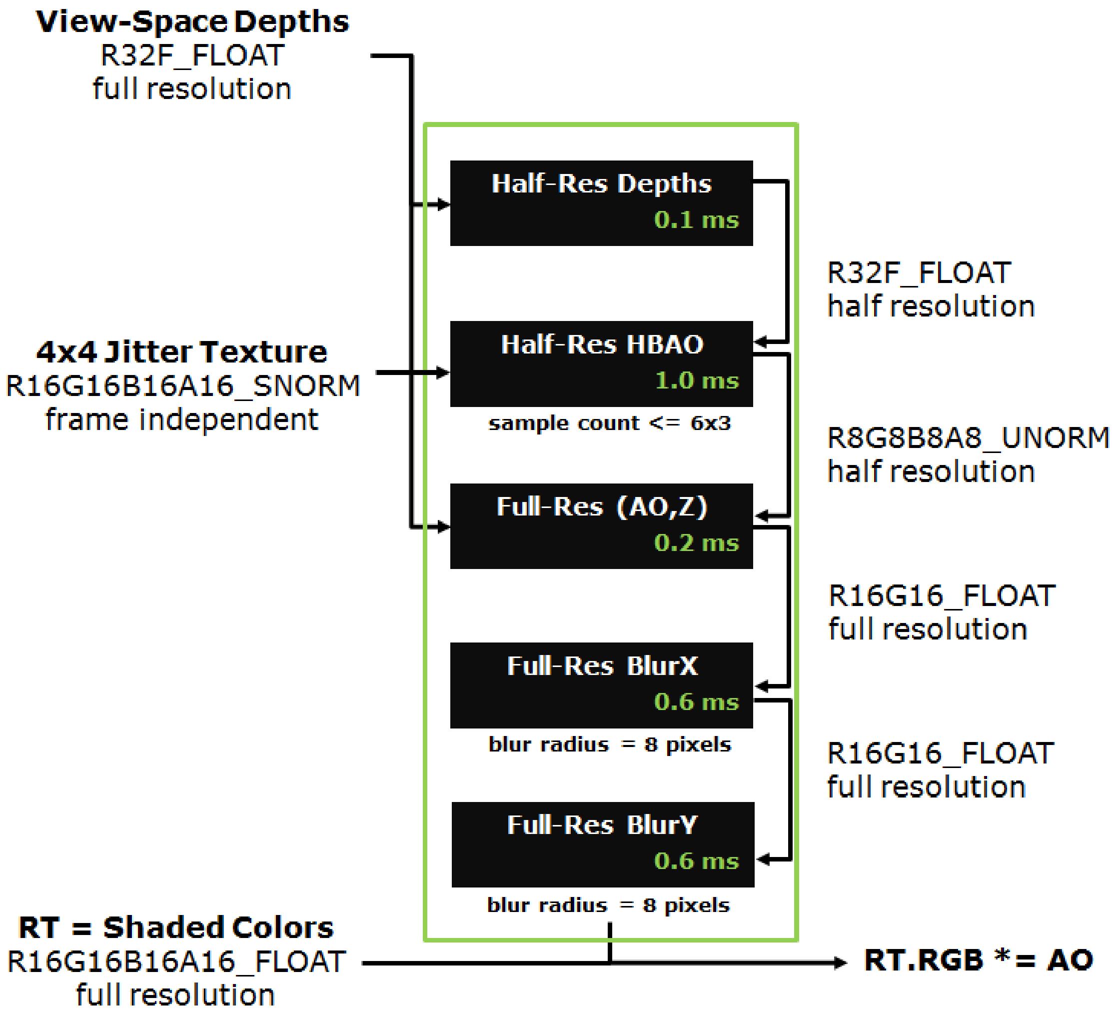
- Horizon-Based Ambient Occlusion (HBAO)
  - Considers the depth buffer as a heightfield, and approximates **ray-tracing** this heightfield
  - Improved implementation available in NVIDIA's SDK11 (SSAO11.zip / HBAO_PS.hlsl)
  - Used for rendering SSAO in Battlefield 3 / PC for its "High" and "Ultra" Graphics Quality presets

[Bavoil and Sainz 09a]
[Andersson 10]
[White and Barré-Brisebois 11]

HBAO

[Andersson 10]

Original HBAO Implementation in Frostbite 2



Frame time (GPU):
25.2 ms

Total HBAO time:
2.5 ms (10% of frame)

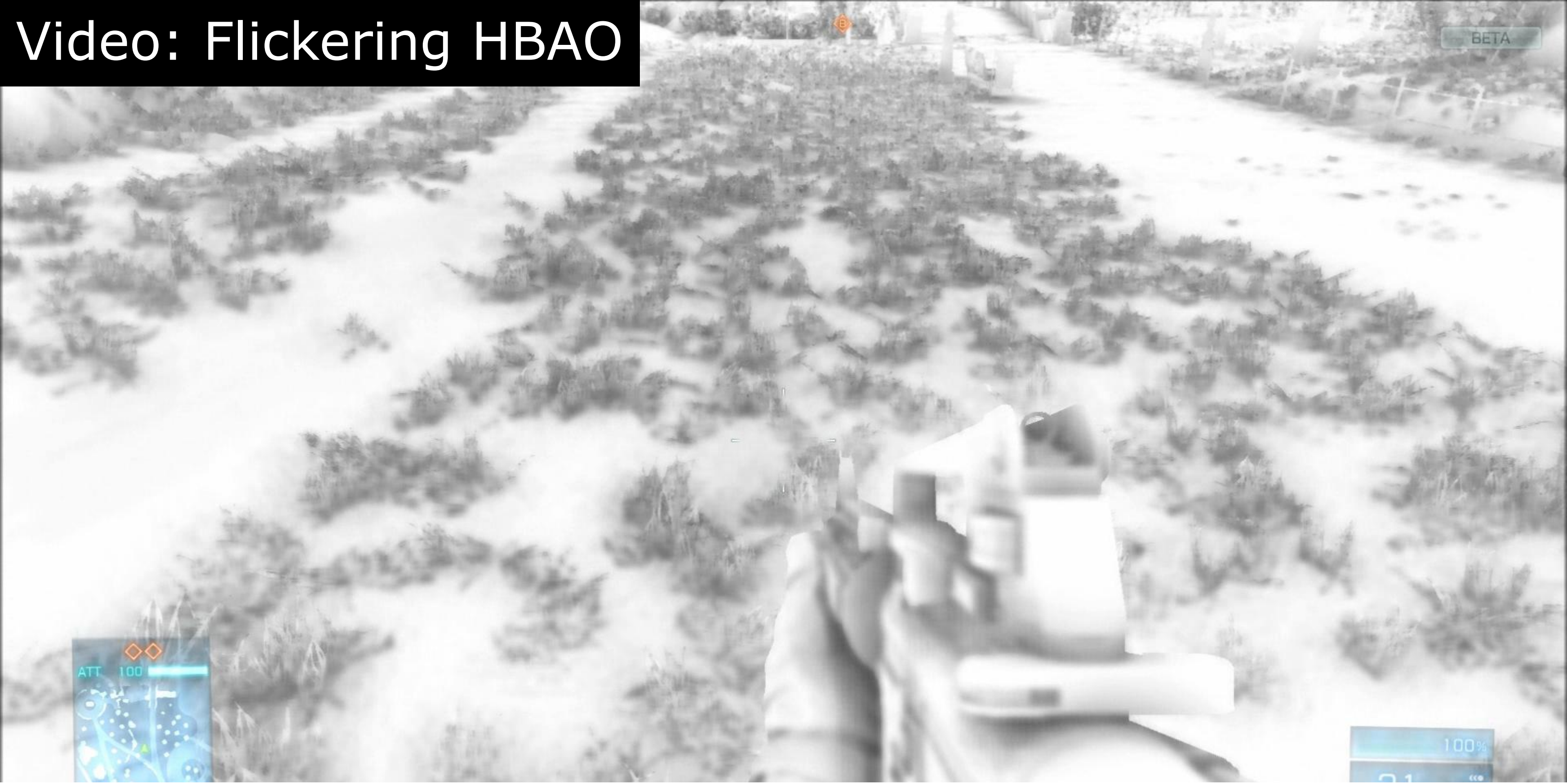
[1920x1200 “High” DX11
GeForce GTX 560 Ti]

Screenshot: With HBAO

Screenshot: HBAO Only

The HBAO looked good-enough on screenshots…

Video: Flickering HBAO



…but produced objectionable flickering in motion on thin objects such as alpha-tested foliage (grass and trees in particular)

Our Constraints

- PC-only
  - In Frostbite 2, HBAO implemented only for DX10 & DX11
- Low Perf Hit, High Quality
  - Whole HBAO was already 2.5 ms (1920x1200 / GTX 560 Ti)
  - HBAO used in High and Ultra presets, had to look great

Considered Workarounds

- Full-resolution SSAO or dual-resolution SSAO (*)

  …but that more-than-doubled the cost of the SSAO, and some flickering could remain

- Brighten SSAO on the problematic objects

  …but we wanted a way to keep full-strength SSAO on everything (in particular on foliage)

(*) [Bavoil and Sainz 09b]

Temporal Filtering Approach

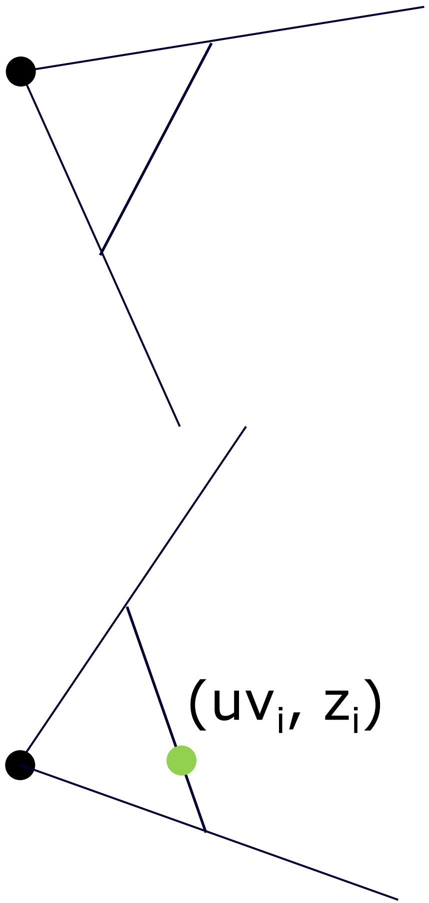
- By definition, AO depends only on the scene geometry, not on the camera
  - For static (or nearly-static geometry), can re-project AO from previous frame
  - Reduce AO variance between frames by using a temporal filter: newAO = lerp(newAO,previousAO,x)
- Known approach used in Gears of War 2

[Nehab et al. 07]
[Smedberg and Wright 09]

Reverse Reprojection

[Nehab et al. 07]

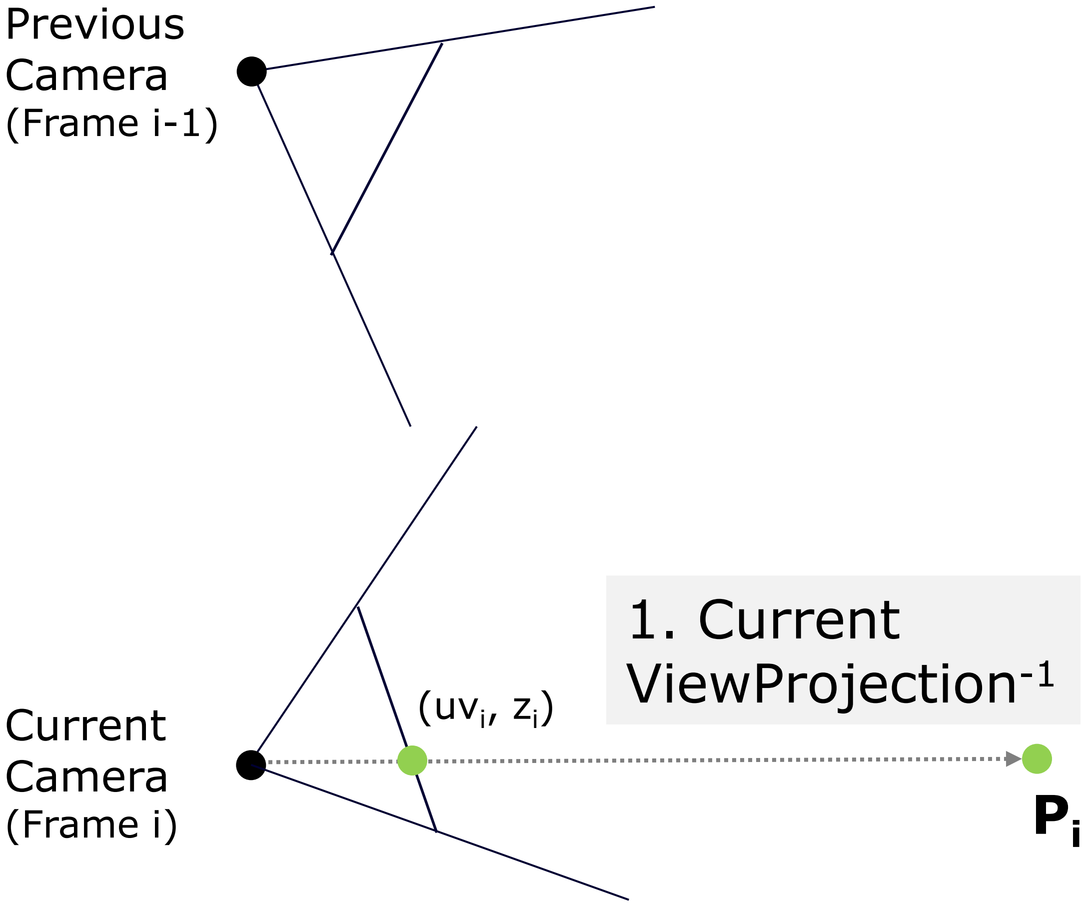
Previous Camera (Frame i-1)



Current Camera (Frame i)

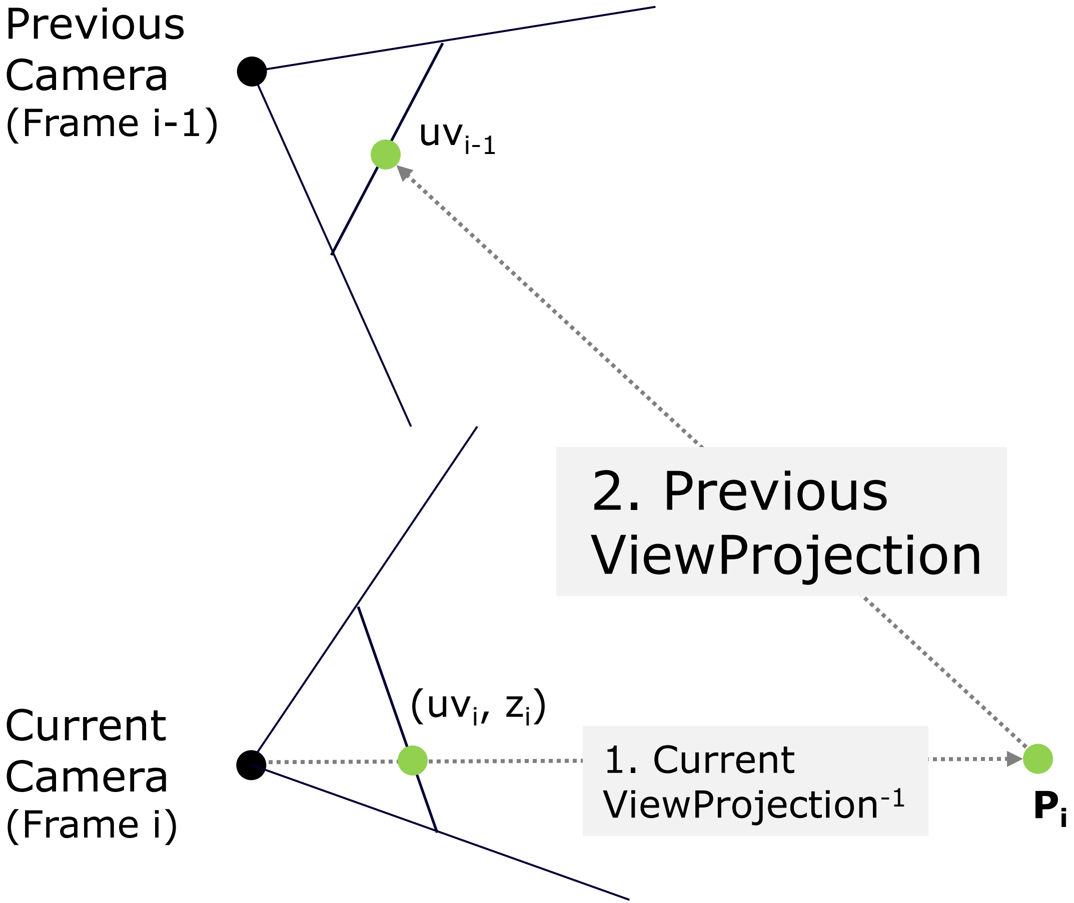
Reverse Reprojection

[Nehab et al. 07]



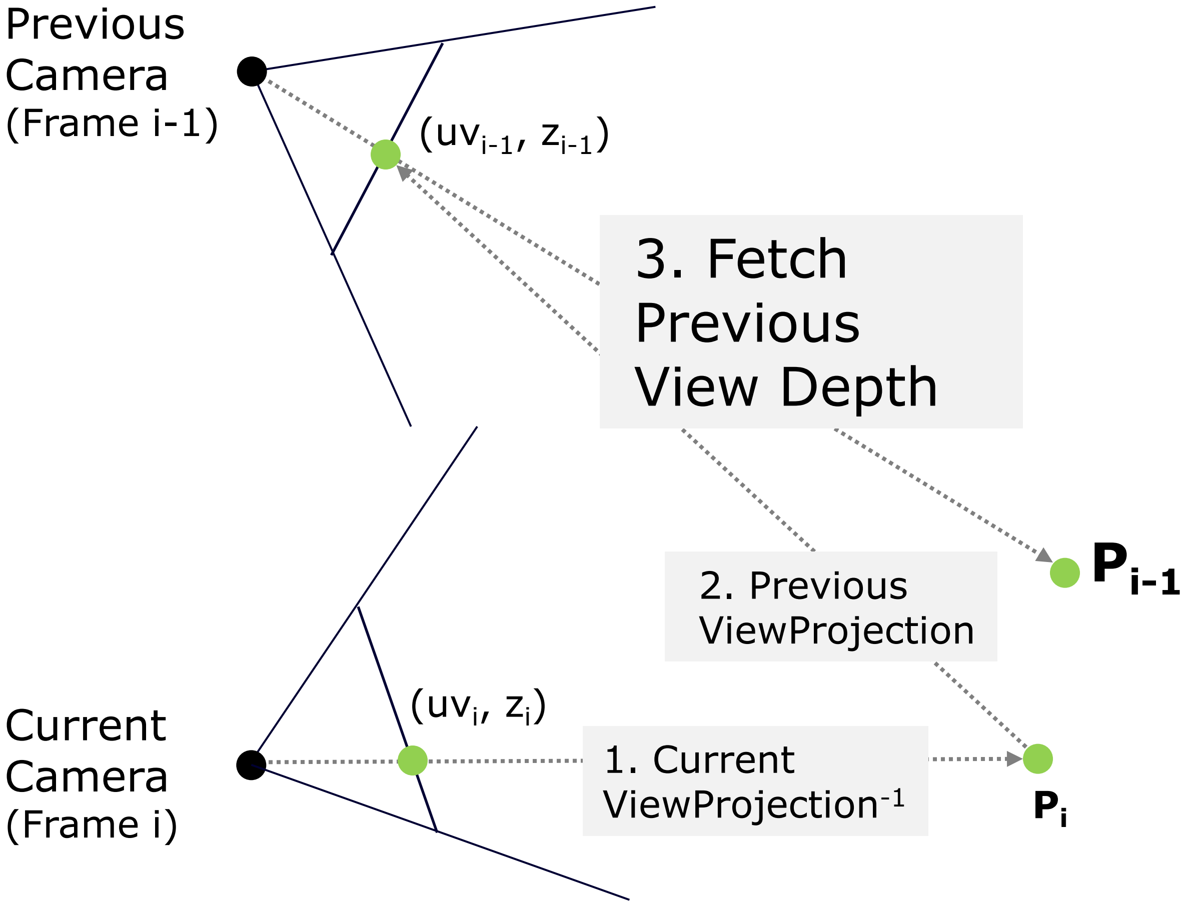
Reverse Reprojection

[Nehab et al. 07]



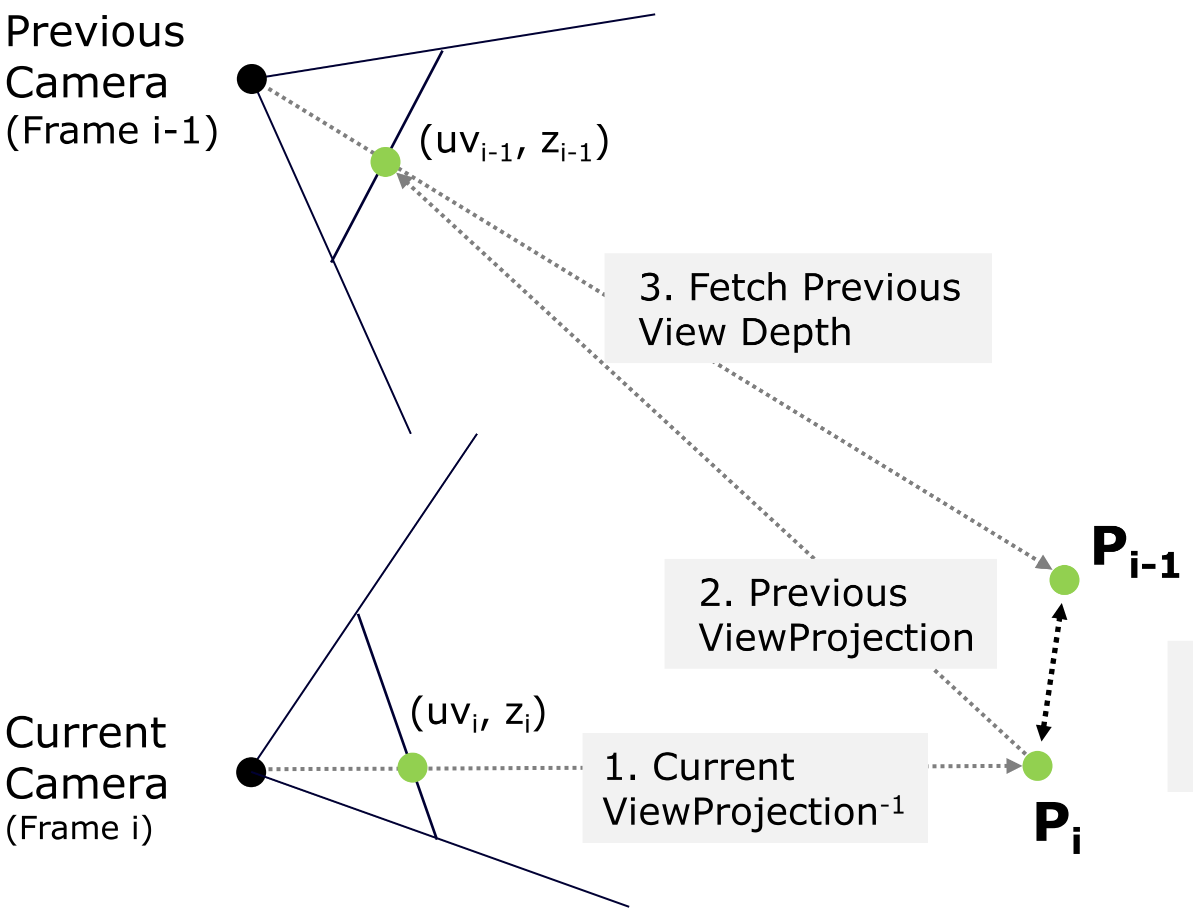
Reverse Reprojection

[Nehab et al. 07]



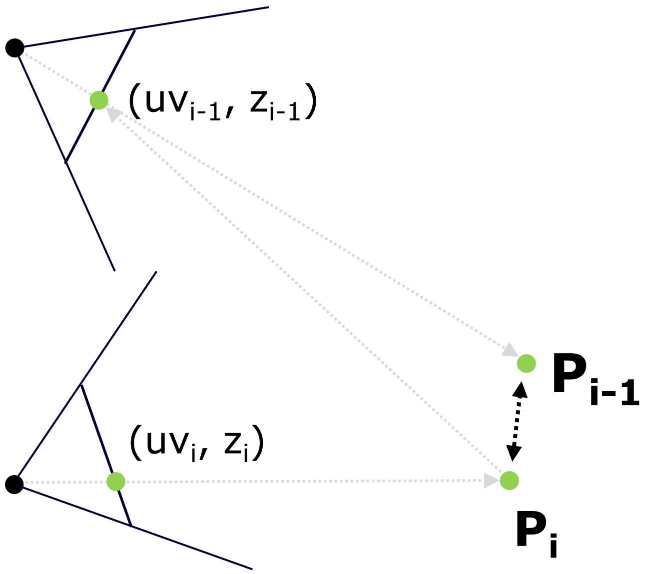
Reverse Reprojection

[Nehab et al. 07]



4. If $P_i \sim= P_{i-1}$ re-use $AO(P_{i-1})$

Temporal Refinement [Mattausch et al. 11]



If $P_{i-1} \sim= P_i$

$AO_i = (\mathbf{N_{i-1}}\ \mathbf{AO_{i-1}} + AO_i) / (\mathbf{N_{i-1}} + 1)$

$N_i = \min(\mathbf{N_{i-1}} + 1, N_{max})$

Else

$AO_i = AO_i$

$N_i = 1$

N_i = num. frames that have been accumulated in current solution at P_i

N_{max} = max num. frames (~8), to keep AO_{i-1} contributing to AO_i

Disocclusion Test [Mattausch et al. 11]

$$P_i \sim= P_{i-1} \Leftrightarrow \left|1 - \frac{w_i}{w_{i-1}}\right| < \varepsilon$$

w_i = ViewDepth($View_{i-1}$, $\mathbf{P_i}$)

w_{i-1} = ViewDepth($View_{i-1}$, $\mathbf{P_{i-1}}$)

Relaxed Disocclusion Test

$$P_i \sim= P_{i-1} \Leftrightarrow \left|1 - \frac{w_i}{w_{i-1}}\right| < \varepsilon$$

To support nearly-static objects

- Such as foliage waving in the wind (grass, trees, …)
- We simply relaxed the threshold (used $\varepsilon = 10\%$)

Video: Temporal Filtering with **ε=+Inf**

Flickering is fixed, but there are ghosting artifacts due to disocclusions

Video: Temporal Filtering with **ε=0.1%**

Flickering on the grass (nearly static), but no ghosting artifacts

Video: Temporal Filtering with **ε=10%**

No flickering, no ghosting, 1% perf hit (25.2 -> 25.5 ms)

Video: Trailing Artifacts

New issue: trailing artifacts on static objects receiving AO from dynamic objects

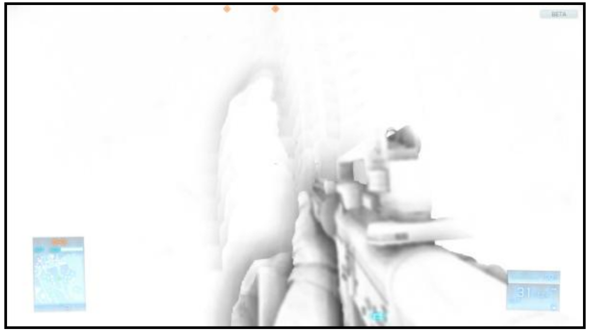
Observations

1. With temporal filtering OFF

   The **flickering pixels** are mostly on foliage. The other pixels do not have any objectionable flickering.

2. With temporal filtering ON

   The **trailing artifacts** (near the character's feet) are not an issue on foliage pixels.



Selective Temporal Filtering

- Assumption

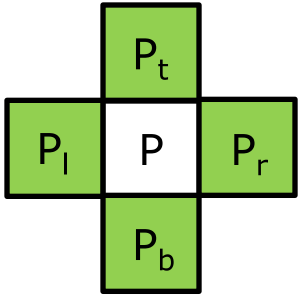
  The set of **flickering pixels** and the set of **trailing pixels** are **mutually exclusive**

- Our Approach:

  1. Classify the pixels as **stable** (potential trailing) or **unstable** (potential flickering)
  2. Disable the temporal filter for the stable pixels

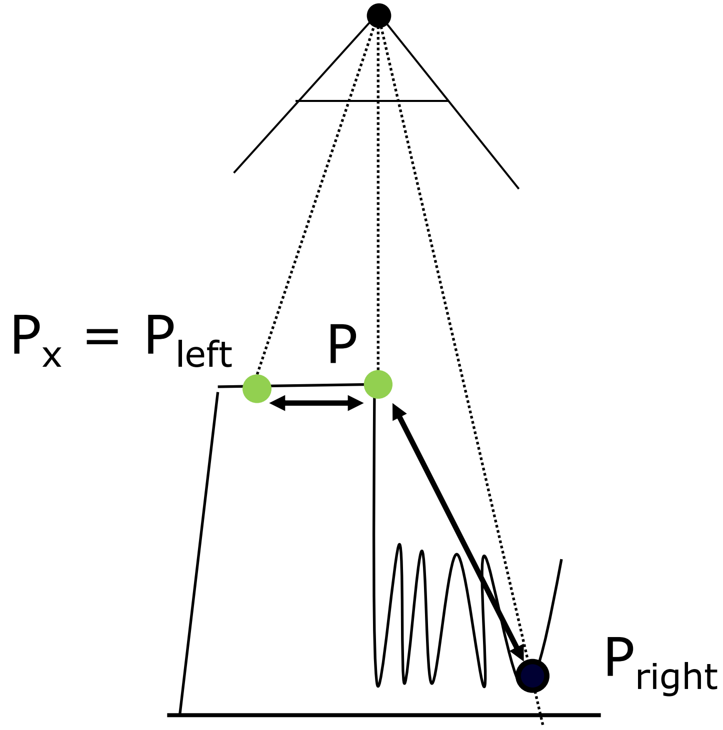
Pixel Classification Approach

- Normal reconstruction in SSAO shader
  - $\mathbf{P_x} = ||P - P_{left}|| < ||P - P_{right}|| \ ? \ P_{left} : P_{right}$
  - $\mathbf{P_y} = ||P - P_{top}|| < ||P - P_{bottom}|| \ ? \ P_{top} : P_{bottom}$
  - $N = \pm \text{normalize}(\text{cross}(P - P_x, P - P_y))$



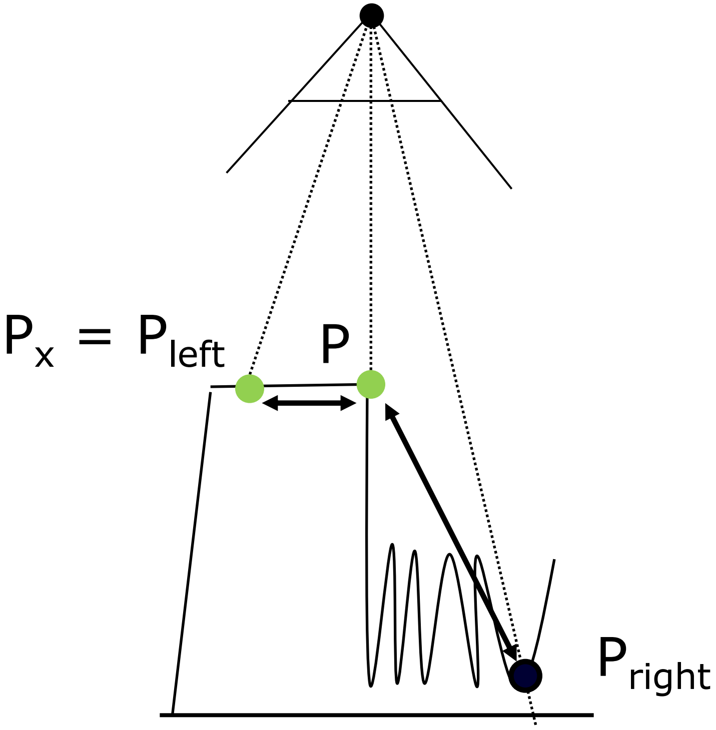
- Idea: If reconstructed normal is noisy, the SSAO will be noisy

Piecewise Continuity Test



1. Select nearest neighbor P_x between P_{left} and P_{right}

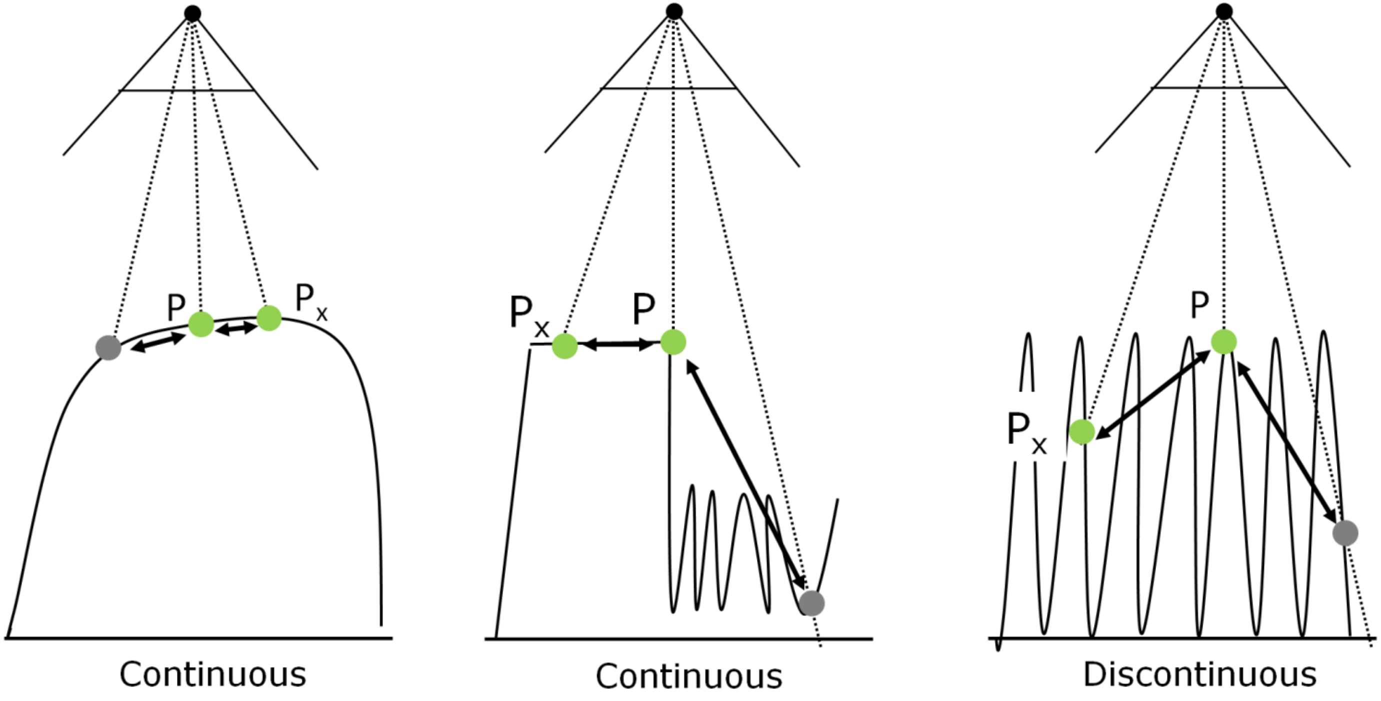
Piecewise Continuity Test



1. Select nearest neighbor P_x between P_{left} and P_{right}
2. Continuous pixels ⇔ $|| P_x - P || < L$

   where L = distance threshold (in view-space units)

Pixel Classification Examples



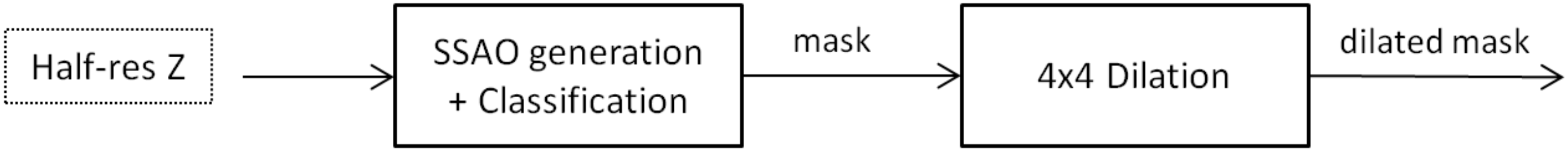
Two Half-Res Passes

Pass 1: Output SSAO and continuity mask

$$\text{continuityMask} = (\| P_x - P \| < L \;\&\&\; \| P_y - P \| < L)$$

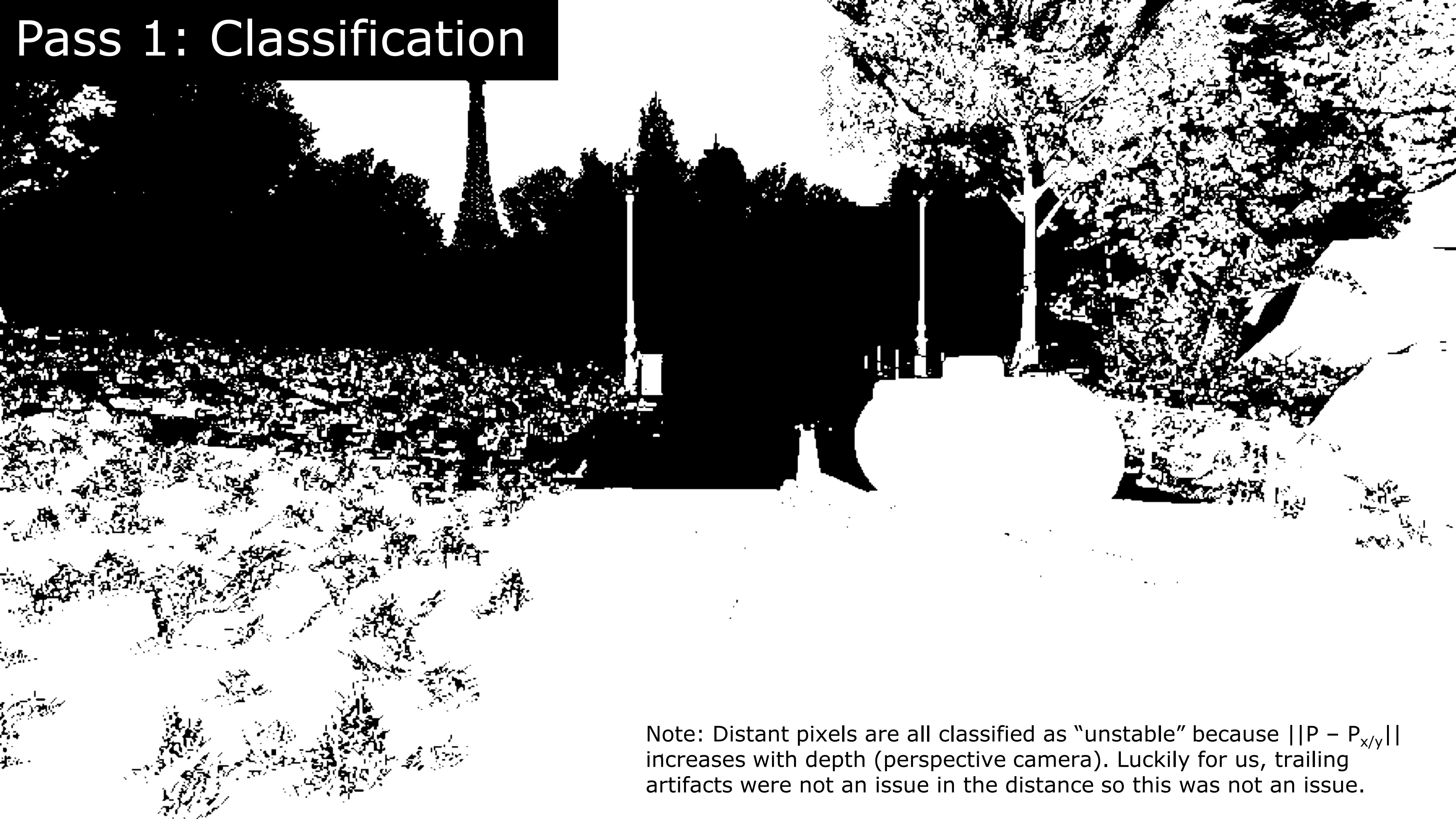
Pass 2: Dilate the discontinuities

$$\text{dilatedMask} = \text{all}_{4x4}(\text{continuityMask})$$



Example Scene

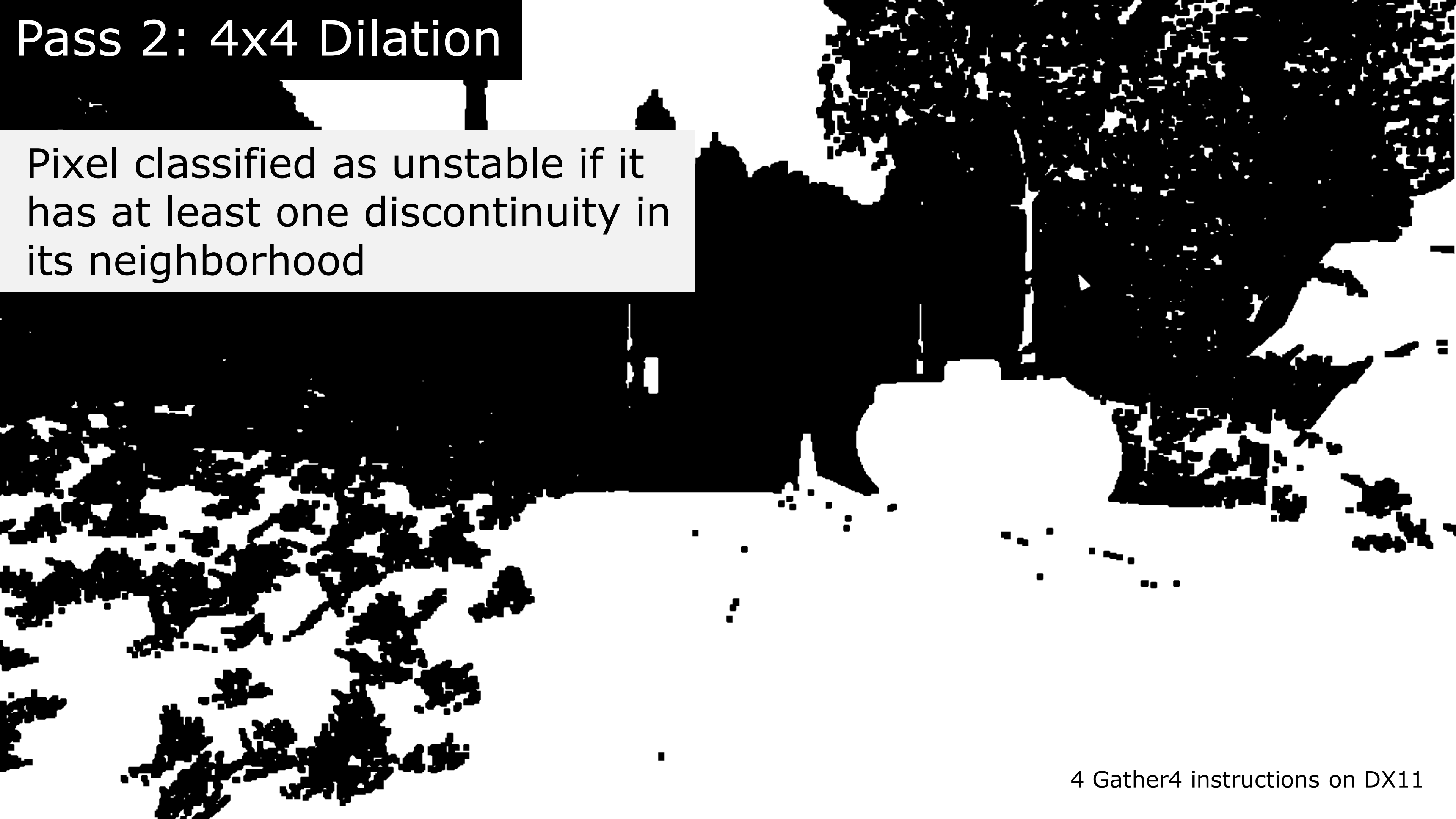
Pass 1: Classification



Note: Distant pixels are all classified as "unstable" because $||P - P_{x/y}||$ increases with depth (perspective camera). Luckily for us, trailing artifacts were not an issue in the distance so this was not an issue.

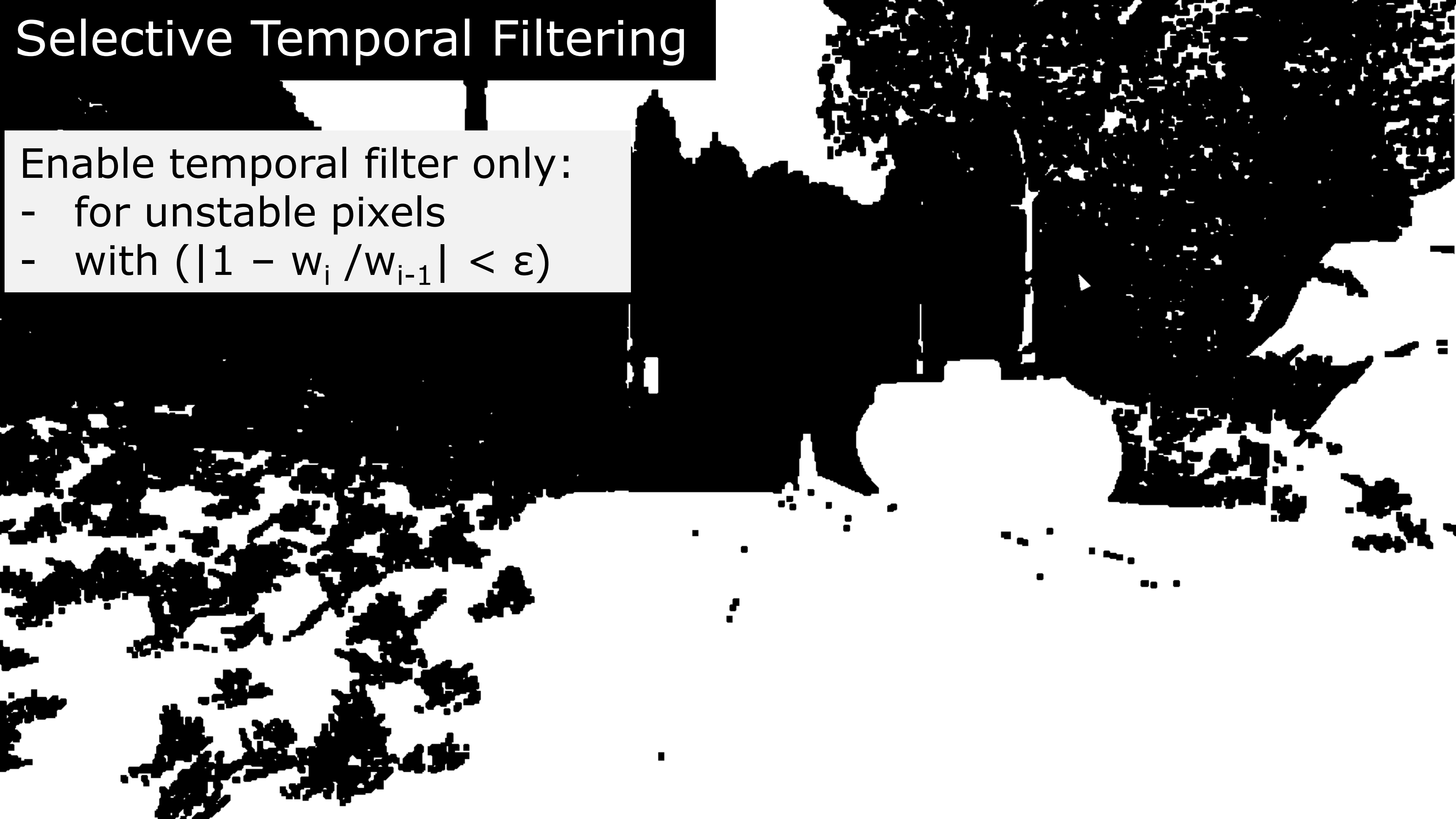
Pass 2: 4x4 Dilation

Pixel classified as unstable if it has at least one discontinuity in its neighborhood



4 Gather4 instructions on DX11

Selective Temporal Filtering

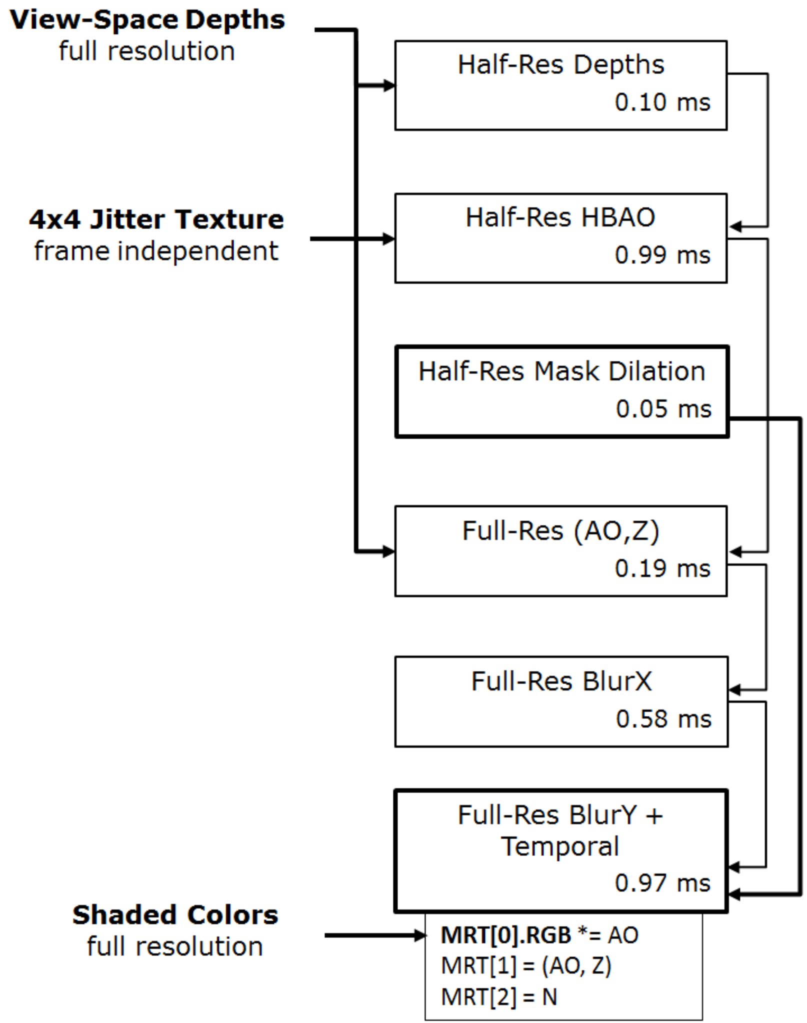


Enable temporal filter only:

- for unstable pixels
- with ($|1 - w_i / w_{i-1}| < \varepsilon$)

Video: HBAO + Selective Temporal Filtering

Video: Final Result



Final Pipeline with Selective Temporal Filtering (STF)

STF Performance Hit

[1920x1200 “High”, GTX 560 Ti]

- HBAO total: 2.5 ms -> 2.9 ms
- Frame time (GPU): 25.2 -> 25.6 ms (1.6% performance hit)

STF Parameters

- Reprojection Threshold: For detecting disocclusions (ε=10%)
- Distance Threshold: For detecting discontinuities (L=0.1 meter)
- Dilation Kernel Size (4x4 texels)

History Buffers

- Additional GPU memory required for the history buffers
  - For (AO_i, Z_i, N_i) and (AO_{i-1}, Z_{i-1}, N_i)
  - Full-res, 1xMSAA
- For Multi-GPU Alternate Frame Rendering (AFR) rendering
  - Create one set of buffers per GPU and alternate between them
  - Each AFR GPU has its own buffers & reprojection state
- The history buffers are cleared on first use
  - Clear values: (AO,Z)=(1.f, 0.f) and N=0

SelectiveTemporalFilter(uv_i, AO_i)

```
z_i = Fetch(ZBuffer_i, uv_i)
P_i = UnprojectToWorld(uv_i, z_i)
uv_i-1 = ProjectToUV(View_i-1, P_i)
z_i-1 = Fetch(ZBuffer_i-1, uv_i-1)
P_i-1 = UnprojectToWorld(uv_i-1, z_i-1)
w_i   = ViewDepth(View_i-1 , P_i)
w_i-1 = ViewDepth(View_i-1 , P_i-1)
isStablePixel = Fetch(StabilityMask, uv_i)
if ( |1 – w_i/w_i-1 | < ε && !isStablePixel )
   AO_i-1 = Fetch(AOTexture_i-1, uv_i-1)
   N_i-1 = Fetch(NTexture_i-1, uv_i-1)
   AO_i = (N_i-1 AO_i-1 + AO_i) / (N_i-1 + 1)
   N_i = min(N_i-1 + 1, N_max)
else
   N_i = 1
return(AO_i, w_i, N_i)
```

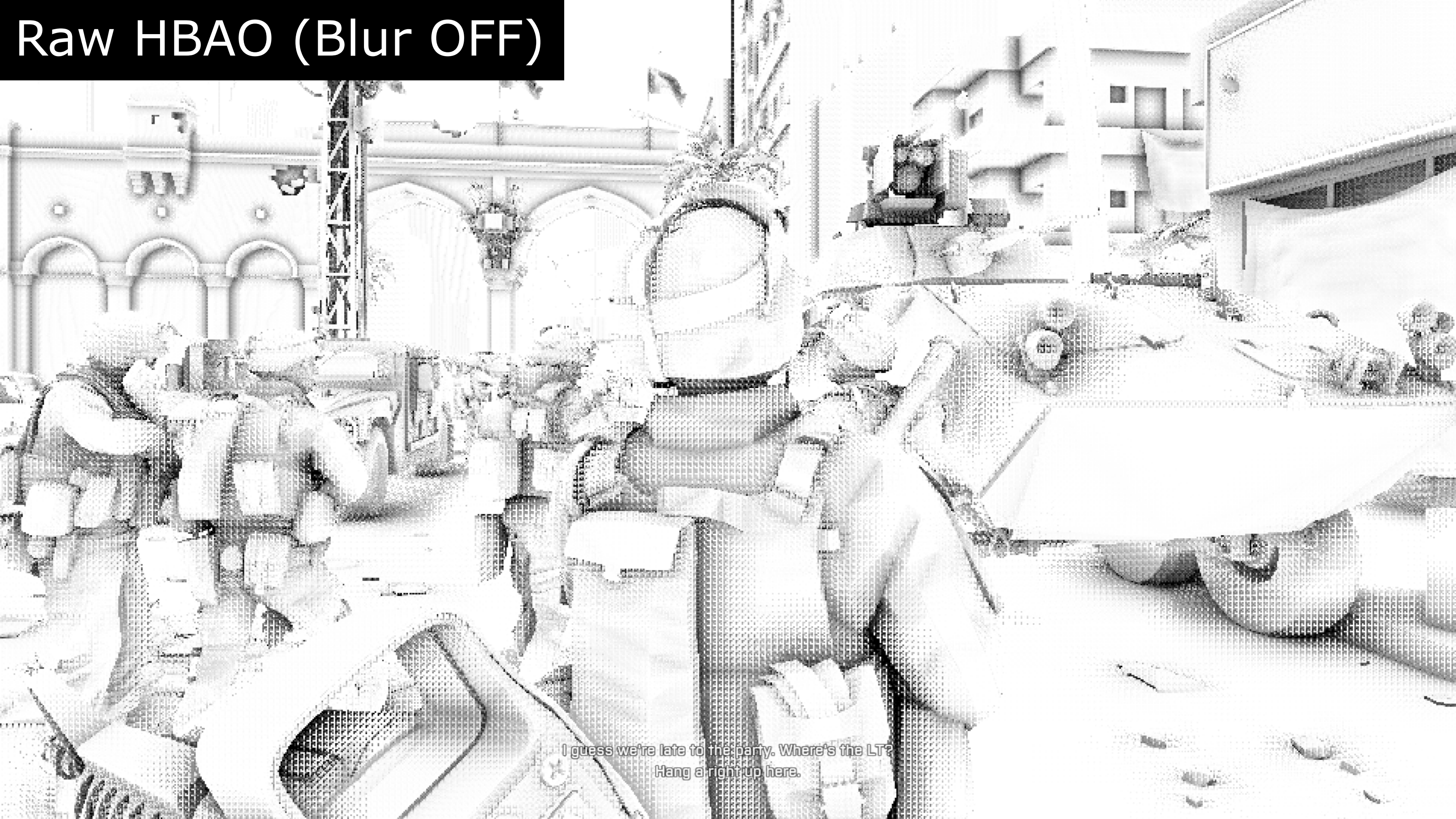
Unoptimized pseudo-code

For fetching z_{i-1}, use **clamp-to-border** to discard out-of-frame data, with borderZ=0.f

For fetching AO_{i-1}, use **bilinear filtering** like in [Nehab et al. 07]

Blur Optimization

Raw HBAO (Blur OFF)



Final HBAO (Blur ON)

Blur Overview

- Full-screen pixel-shader passes
  - BlurX (horizontal)
  - BlurY (vertical)

- BlurX takes as input
  - Half-res AO
  - Full-res linear depth (non-MSAA)

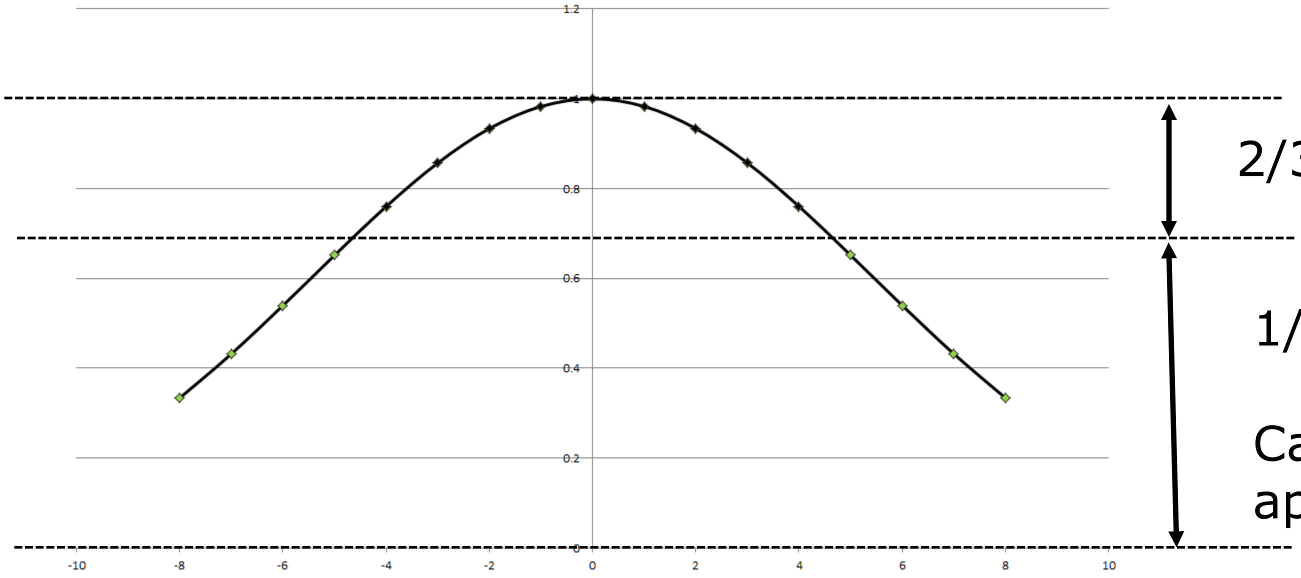
Blur Kernel

- We use 1D Cross-Bilateral Filters (CBF)
- Gaussian blur with depth-dependent weights

$$\text{Output} = \frac{\text{Sum}[\ \mathbf{AO_i}\ w(i,\mathbf{Z_i},\mathbf{Z_0}),\ i=-R..R\ ]}{\text{Sum}[\ w(i,\mathbf{Z}_i,\mathbf{Z_0}),\ i=-R..R\ ]}$$

[Petschnigg et al. 04]
[Eisemann and Durand 04]
[Kopf et al. 07]
[Bavoil et al. 08]
[McGuire et al. 11]

Blur Opt: Adaptive Sampling

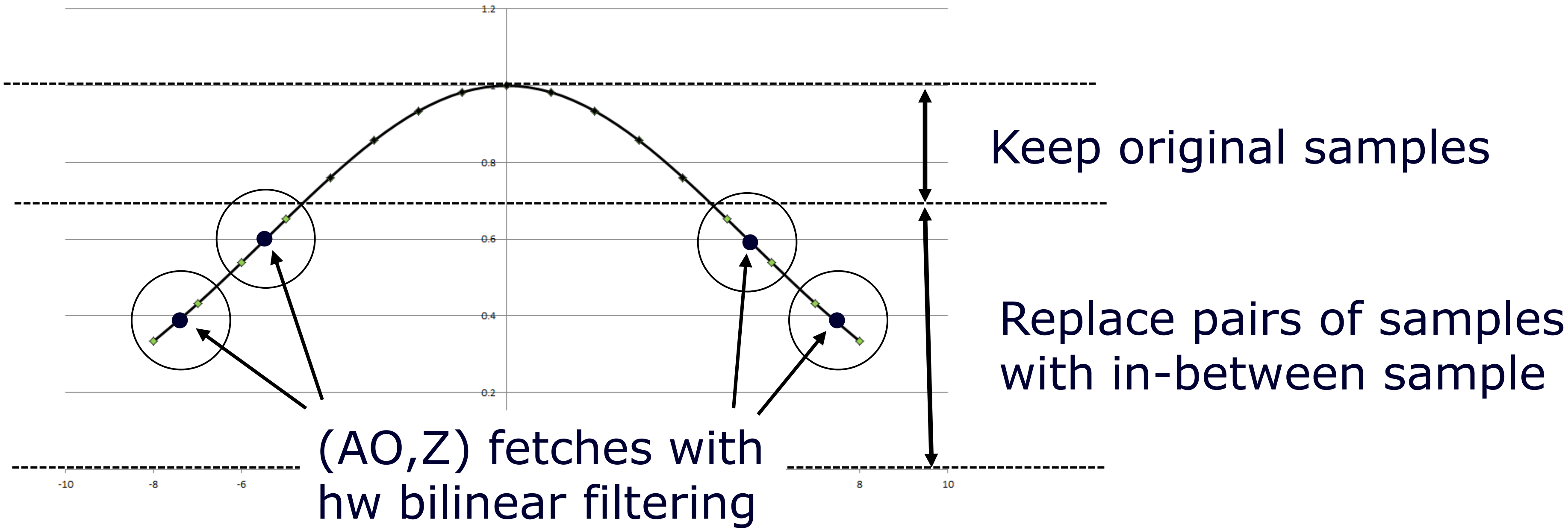


2/3 of total weights

1/3 of total weights

Can these samples be approximated?

Blur Opt: Adaptive Sampling



Blur Opt: Adaptive Sampling

Kernel to be sampled (radius = 8 pixels):

Before: Sample the inner-half of the kernel with POINT filtering and step size = 1:

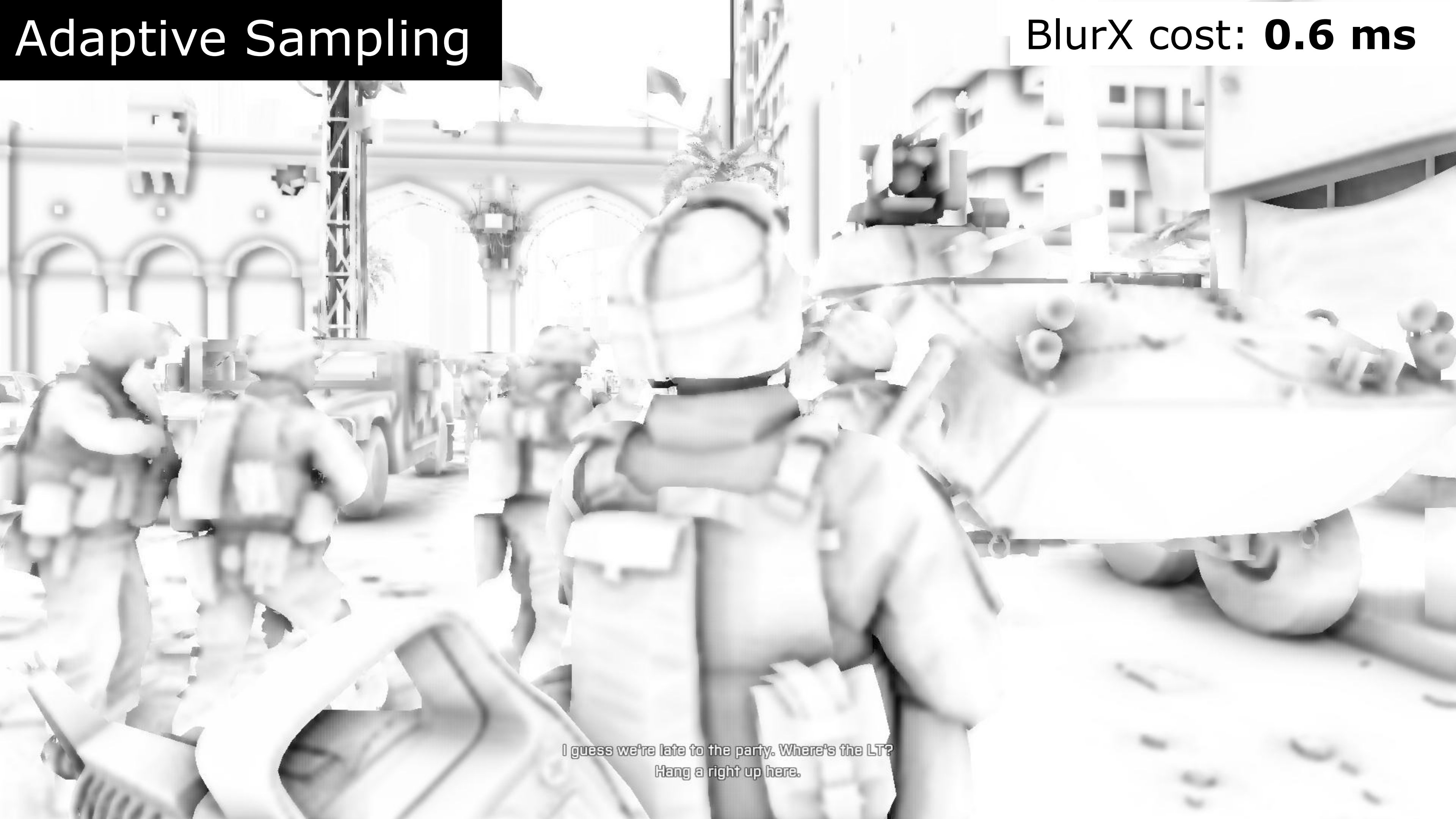
After: Sample the outer-half of the kernel with LINEAR filtering and step size = 2:

Brute-Force Sampling

BlurX cost: **0.8 ms**

Adaptive Sampling

BlurX cost: **0.6 ms**



Blur Opt: Speedup

GPU Time	Before	After	Speedup
Pack (AO,Z)	0.18 ms	0.18 ms	0%
BlurX	0.75 ms	0.58 ms	**29%**
BlurY+STF	1.00 ms	0.95 ms	**5% (*)**
Blur Total	1.93 ms	1.71 ms	13%

(*) Lower speedup due to the math overhead of the Selective Temporal Filter (STF)

Blur Radius: 8
Resolution: 1920x1200
GeForce GTX 560 Ti

Summary

Two techniques used in Battlefield 3 / PC

1. A generic solution to fix SSAO flickering with a low perf hit (*) on DX10/11 GPUs
2. An approximate cross-bilateral filter, using a mix of point and bilinear taps

(*) **0.4 ms** in 1920x1200 on GeForce GTX 560 Ti

Questions?

Louis Bavoil
lbavoil@nvidia.com

Johan Andersson
johan.andersson@dice.se

References

[McGuire et al. 11] McGuire, Osman, Bukowski, Hennessy. The Alchemy Screen-Space Ambient Obscurance Algorithm. Proceedings of ACM SIGGRAPH / Eurographics High-Performance Graphics 2011 (HPG '11).

[White and Barré-Brisebois 11] White, Barré-Brisebois. More Performance! Five Rendering Ideas from Battlefield 3 and Need For Speed: The Run. Advances in Real-Time Rendering in Games. SIGGRAPH 2011.

[Mattausch et al. 11] Mattausch, Scherzer, Wimmer. Temporal Screen-Space Ambient Occlusion. In GPU Pro 2. 2011.

[Loos and Sloan 10] Loos, Sloan. Volumetric Obscurance. ACM Symposium on Interactive 3D Graphics and Games 2010.

References

[Andersson 10] Andersson. Bending the Graphics Pipeline. Beyond Programmable Shading course, SIGGRAPH 2010.

[Bavoil and Sainz 09a] Bavoil, Sainz. Image-Space Horizon-Based Ambient Occlusion. In ShaderX7. 2009.

[Bavoil and Sainz 09b] Bavoil, Sainz. Multi-Layer Dual-Resolution Screen-Space Ambient Occlusion. SIGGRAPH Talk. 2009.

[Smedberg and Wright 09] Smedberg, Wright. Rendering Techniques in Gears of War 2. GDC 2009.

[Kajalin 09] Kajalin. Screen Space Ambient Occlusion. In ShaderX7. 2009.

[Bavoil et al. 08] Bavoil, Sainz, Dimitrov. Image-Space Horizon-Based Ambient Occlusion. SIGGRAPH Talk. 2008.

References

[Nehab et al. 07] Nehab, Sander, Lawrence, Tatarchuk, Isidoro. Accelerating Real-Time Shading with Reverse Reprojection Caching. In ACM SIGGRAPH/Eurographics Symposium on Graphics Hardware 2007.

[Kopf et al. 07] Kopf, Cohen, Lischinski, Uyttendaele. 2007. Joint Bilateral Upsampling. In Proceedings of SIGGRAPH 2007.

[Petschnigg et al. 04] Petschnigg, Szeliski, Agrawala, Cohen, Hoppe. Toyama: Digital photography with flash and no-flash image pairs. In Proceedings of SIGGRAPH 2004.

[Eisemann and Durand 04] Eisemann, Durand. “Flash Photography Enhancement via Intrinsic Relighting”. In Proceedings of SIGGRAPH 2004.

Bonus Slides

HLSL: Adaptive Sampling

```
float r = 1;
// Inner half of the kernel: step size = 1 and POINT filtering
[unroll] for (; r <= KERNEL_RADIUS/2; r += 1)
{
        float2 uv = r * deltaUV + uv0;
        float2 AOZ = mainTexture.Sample(pointClampSampler, uv).xy;
        processSample(AOZ, r, centerDepth, totalAO, totalW);
}
//  Outer half of the kernel: step size = 2 and LINEAR filtering
[unroll] for (; r <= KERNEL_RADIUS; r += 2)
{
        float2 uv = (r + 0.5) * deltaUV + uv0;
        float2 AOZ = mainTexture.Sample(linearClampSampler, uv).xy;
        processSample(AOZ, r, centerDepth, totalAO, totalW);
}
```

HLSL: Cross-Bilateral Weights

```
// d and d0 = linear depths
float crossBilateralWeight(float r, float d, float d0)
{
    // precompiled by fxc
    const float BlurSigma = ((float)KERNEL_RADIUS+1.0f) * 0.5f;
    const float BlurFalloff = 1.f / (2.0f*BlurSigma*BlurSigma);

    // assuming that d and d0 are pre-scaled linear depths
    float dz = d0 - d;
    return exp2(-r*r*BlurFalloff - dz*dz);
}
```