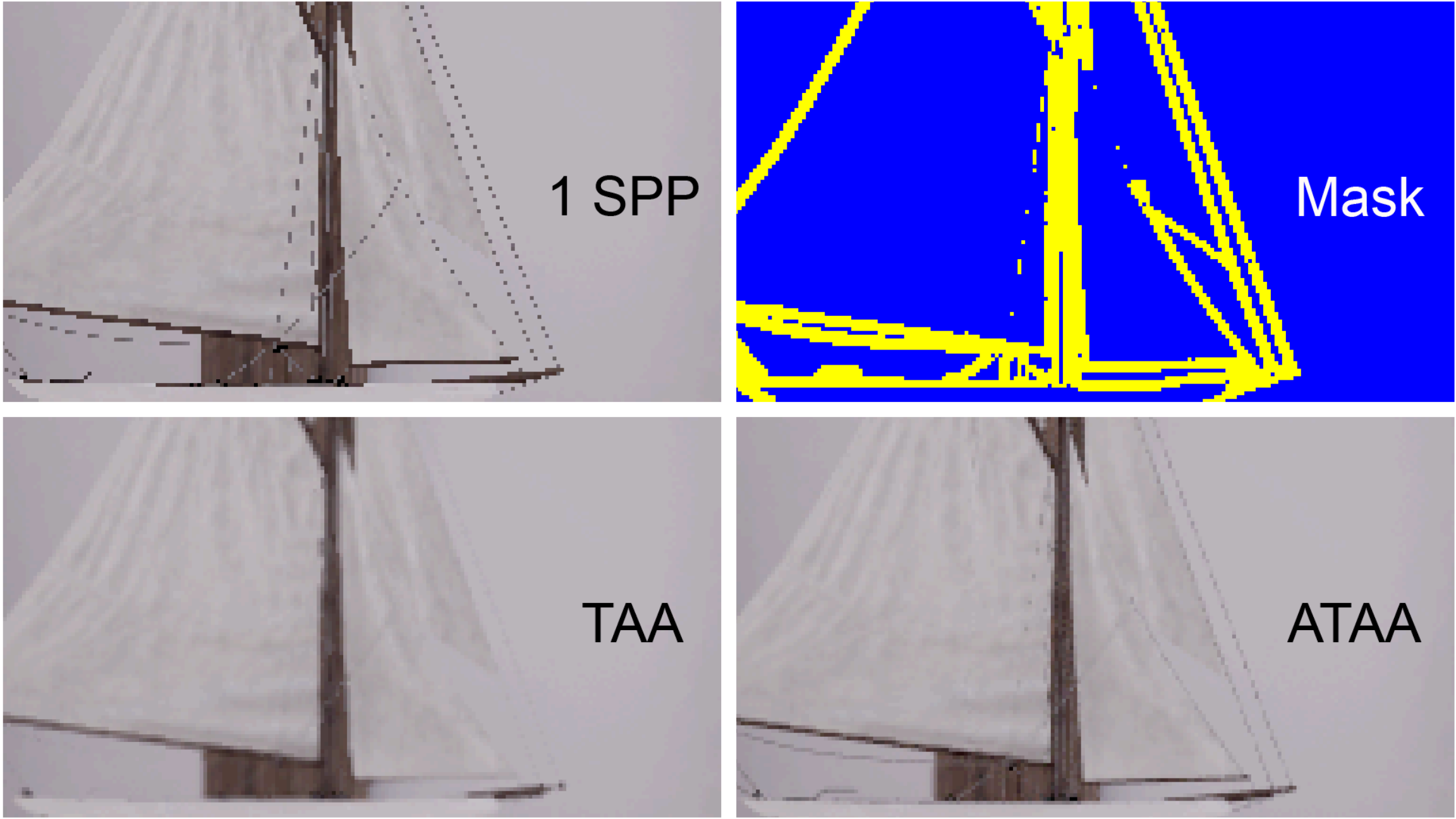
ADAPTIVE TEMPORAL ANTIALIASING

Adam Marrs, August 10th 2018

“…the hardest problem in computer graphics is finding something that hasn’t already been done. That is your challenge.”

Jim Blinn, 1987


1 SPP
Mask
TAA
ATAA

The State of (Anti)aliasing in Real Time

Problem Space

Games limited to a low ***fixed* sample counts** (~1spp) at modest resolutions (< 4K)

Result: primary surfaces are **undersampled** and have **unbounded error** when material, geometric, or shading features exist between samples

Aliasing due to undersampling manifests as **jagged edges**, **spatial noise**, and **flickering**

Constraints: 8x supersampled image quality without ghosting/blurring on a real-time budget

The State of (Anti)aliasing in Real Time

TAA All Day

Supersampling (SSAA): cost linearly proportional to the number of samples while only improving quality with the square root

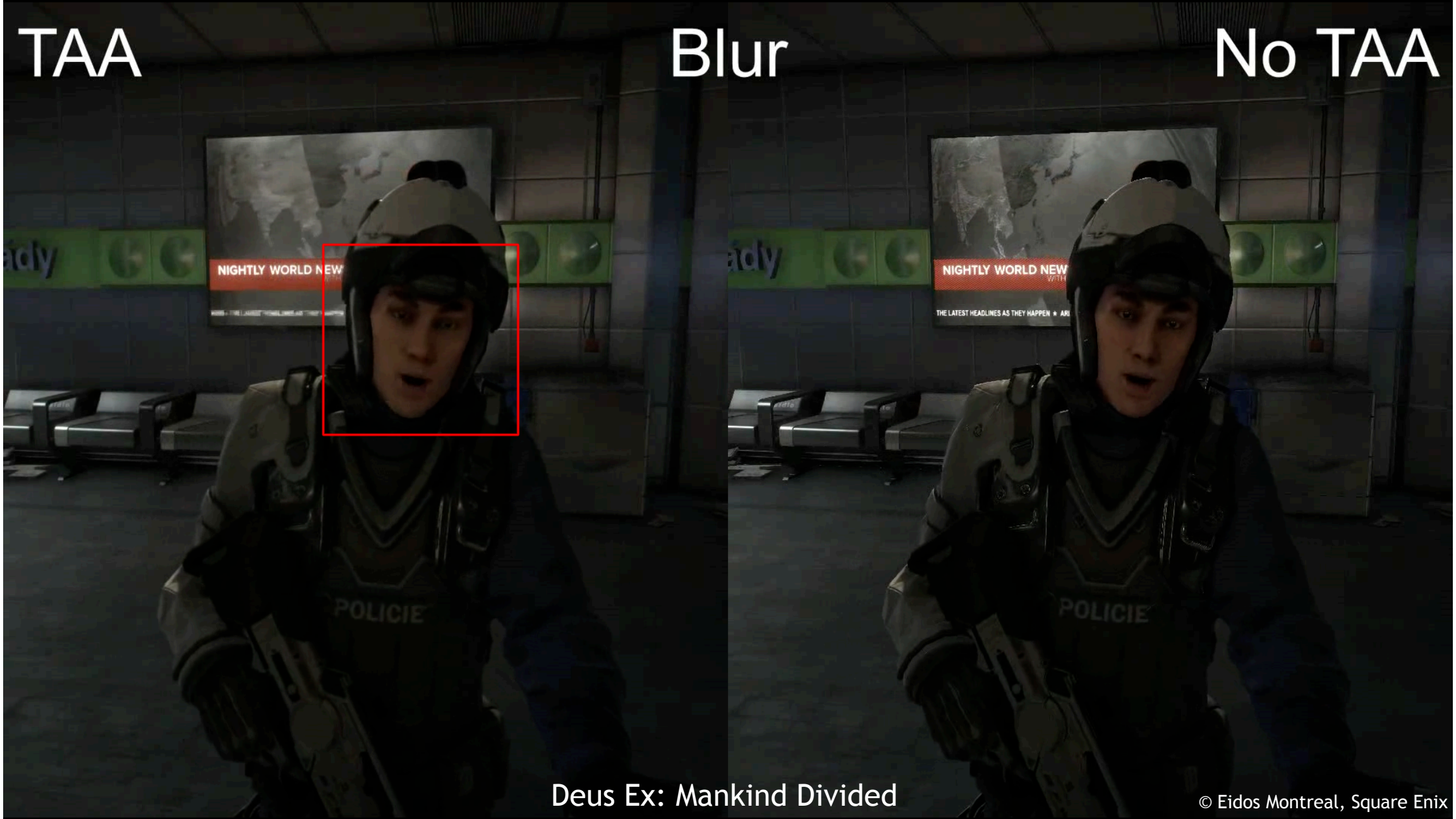
Multisampling: MSAA, CSAA, SBAA [Salvi and Vidimce 2012], SRAA [Chajdas et al. 2011]

Aggregation: DCAA [Wang et al. 2015], AGAA [Crassin et al. 2016]

Spatial: MLAA [Reshetov 2009], FXAA [Lottes 2009]

Time: SMAA [Jimenez et al. 2012], TAA [Yang el al] [Karis 2014]

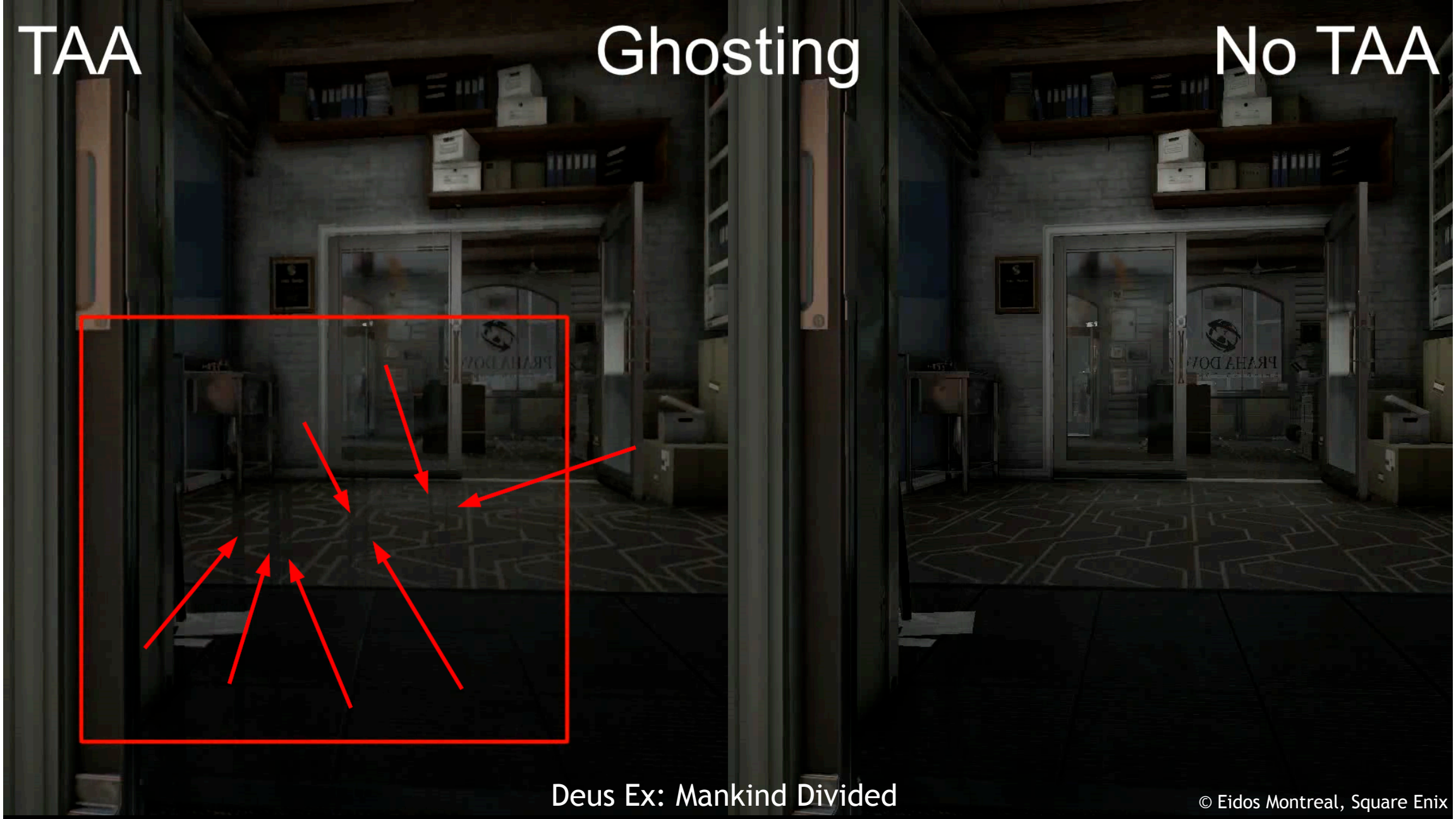
Current Best Practice: employ many strategies simultaneously, hand tune by artists [Pettineo 2015, Pedersen 2016], rely on TAA for the best !/$ solution



Deus Ex: Mankind Divided

TAA

No TAA


TAA
Ghosting
No TAA
Deus Ex: Mankind Divided
© Eidos Montreal, Square Enix

Finding A Practical Hybrid Algorithm

Redefining AA

Offline ray tracers use highly ***adaptive* sample counts** to resolve aliasing and bound error

Some real-time work in this area [Hollander et al. 2013], but still must rasterize all geometry

Previous hybrid ray and raster algorithms were **impractical** due to HW architectures & APIs

DXR and RTX enable **full interoperability** between ray and raster rendering on the GPU

Finding A Practical Hybrid Algorithm

Redefining AA

Naively ray tracing every pixel is **too expensive**

Find the pixels that will benefit most from supersampling, using the existing 1spp input

Goal: harness strengths of TAA while addressing its failures simply and unequivocally

Goal: work within the constraints of conventional game engines

Adaptive Temporal Antialiasing

Core Idea

We ***efficiently*** combine ray and raster, leverage ***adaptive*** sampling in the context of TAA

Step 1: Run TAA

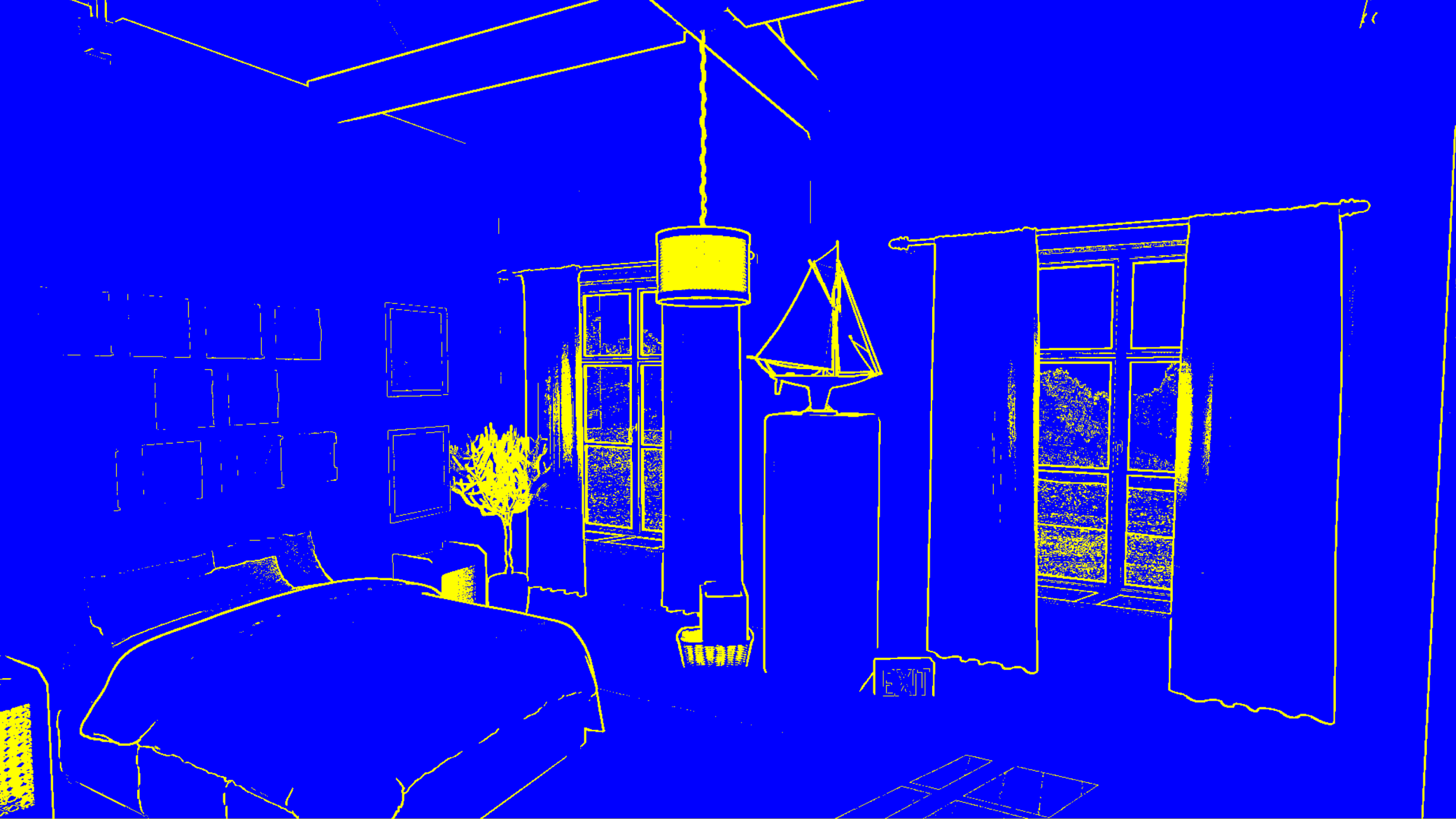
Step 2: Compute a segmentation mask of where TAA fails, and why

Step 3: Replace complex post-failure TAA heuristics with robust alternatives: **ray tracing**

Step 4: Use segmentation mask to guide ray tracing adaptivity

Step 5: Enjoy!


ROAD
TO
HOME
EXIT
No AA


EXIT


Sparse RT
EXIT
Post ATAA
Final Frame


ATAA
FXAA
TAA

Adaptive Temporal Antialiasing

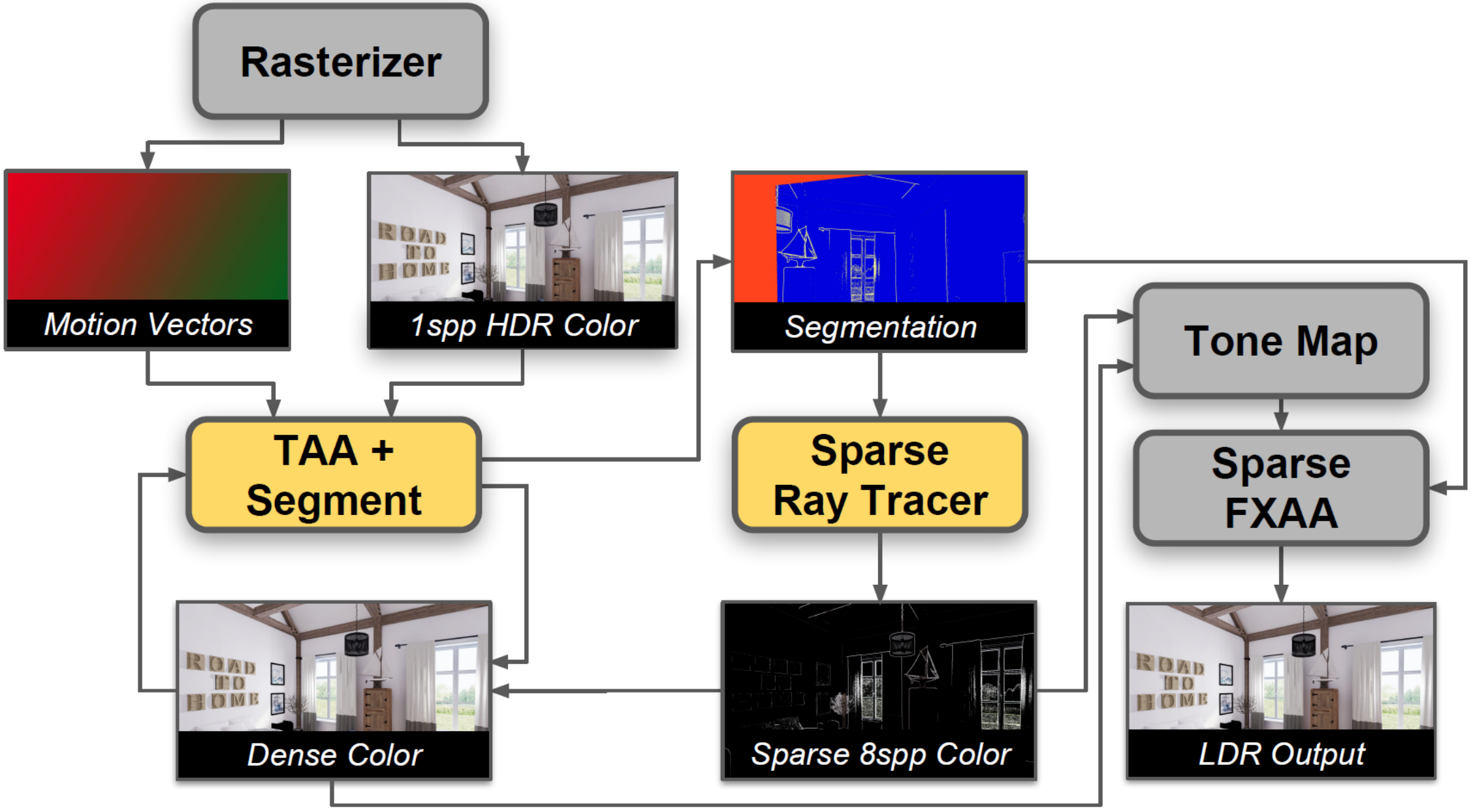
Implementation Details

Unreal Engine 4 extended with DXR API support, running on NVIDIA RTX

TAA fullscreen post-process **extended** to compute and output segmentation mask

Sparse ray tracing in DXR Ray Generation shaders, dispatched as separate fullscreen post-process pass ***before*** tonemapping

Each primary ray casts a single shadow ray to the sun's directional light source (hard shadows)


Rasterizer
Motion Vectors
1spp HDR Color
Segmentation
Tone Map
TAA + Segment
Sparse Ray Tracer
Sparse FXAA
Dense Color
Sparse 8spp Color
LDR Output

Adaptive Temporal Antialiasing

Implementation Details

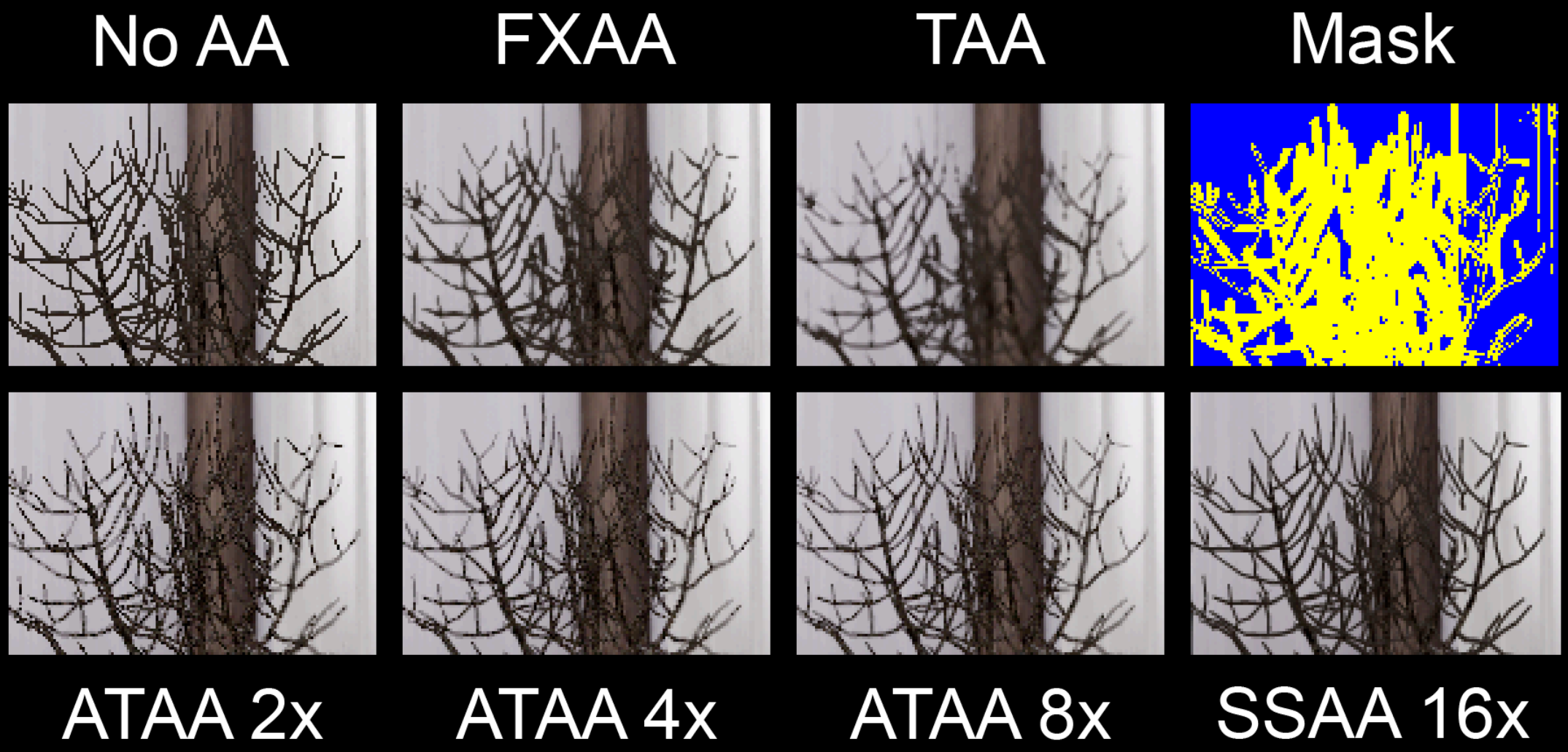
TAA failure detection (segmentation) is a combination of criteria:

- Motion Vectors
- Segmentation History, single frame look-back (was this pixel marked as ATAA?)
- Luminance, temporal change within a pixel neighborhood
- Depth, 3x3 edge-detecting Sobel filter

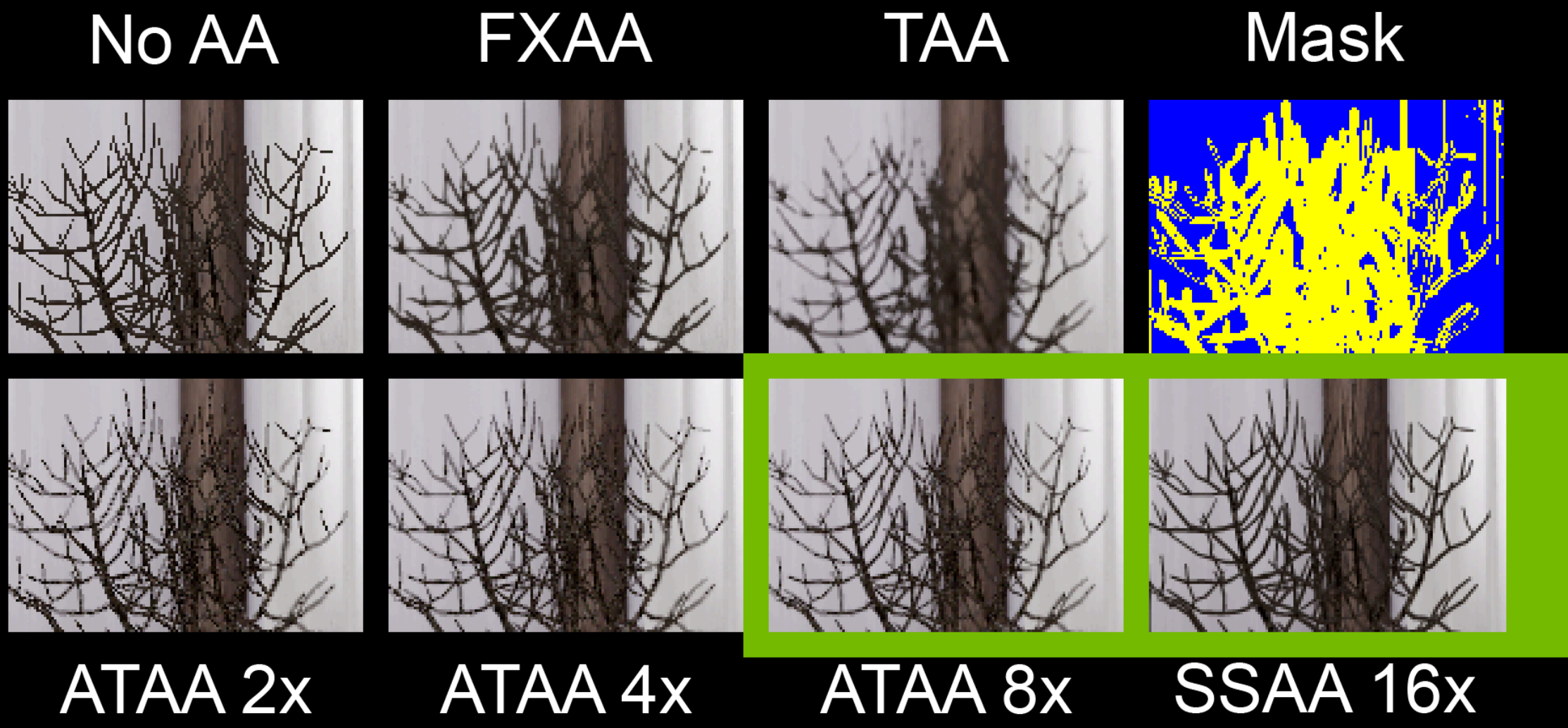
Frames are almost always dominated by TAA-classified (blue) pixels

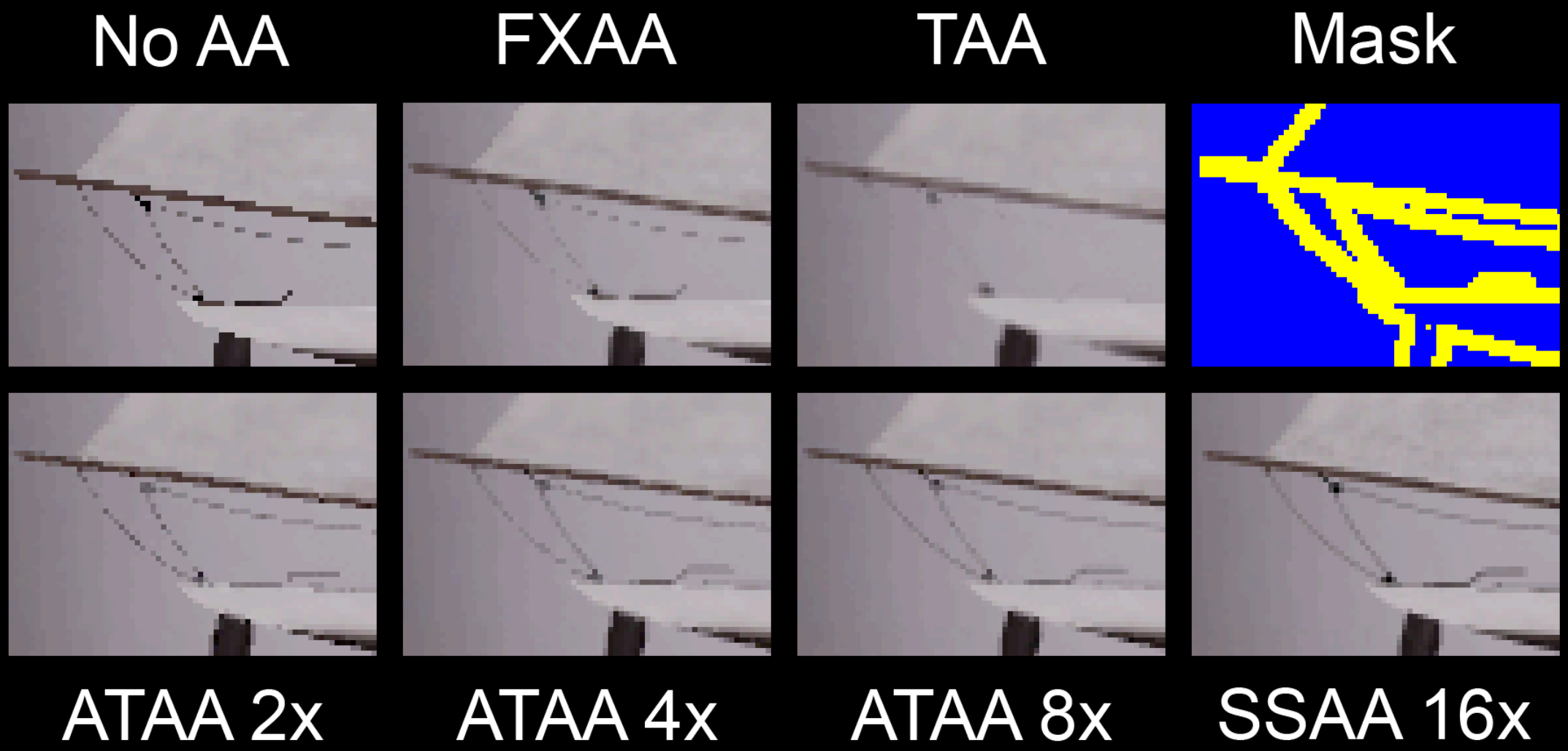

ROAD
TO
HOME
EXIT
TAA

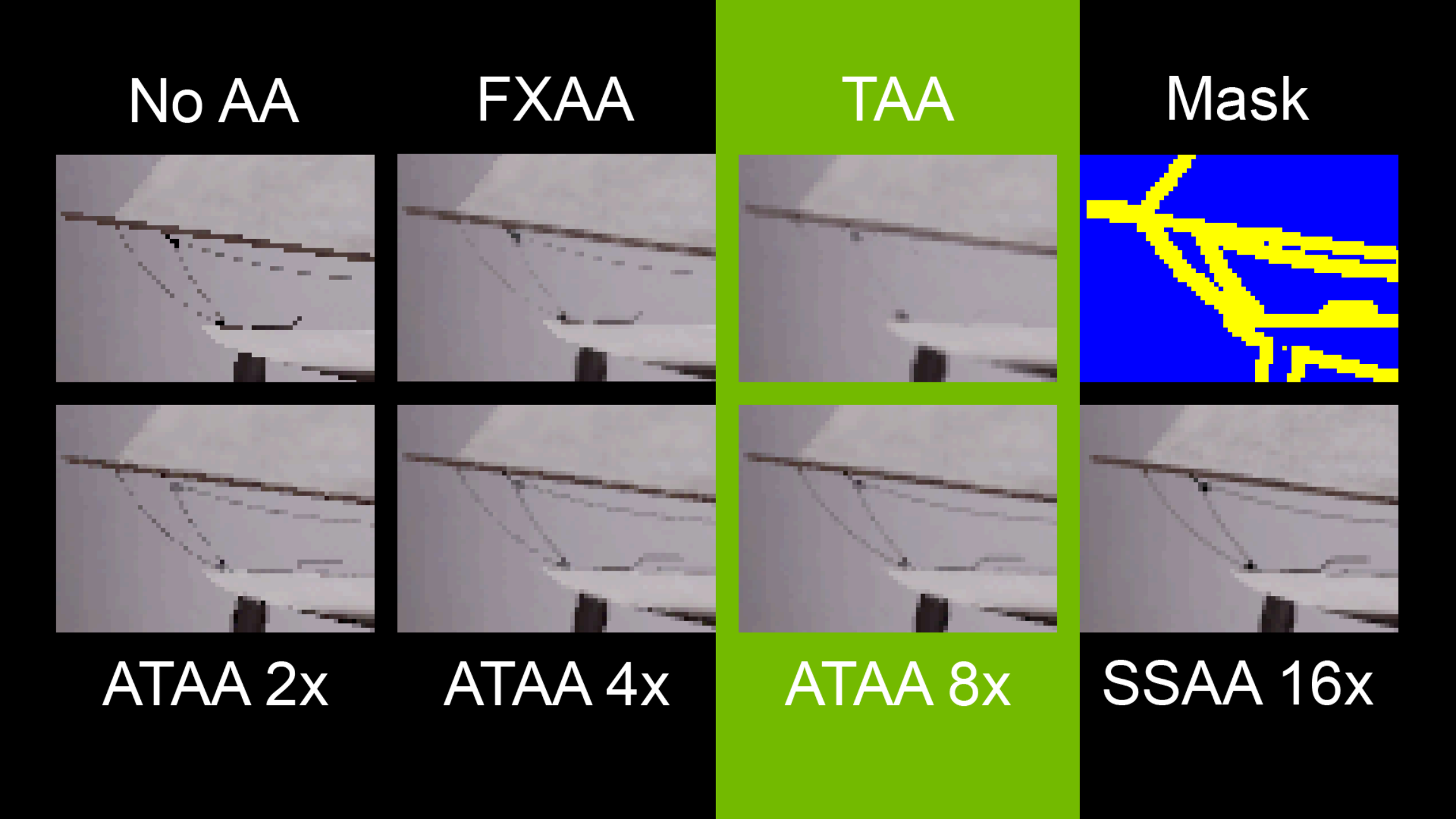

ROAD
TO
HOME
EXIT
ATAA

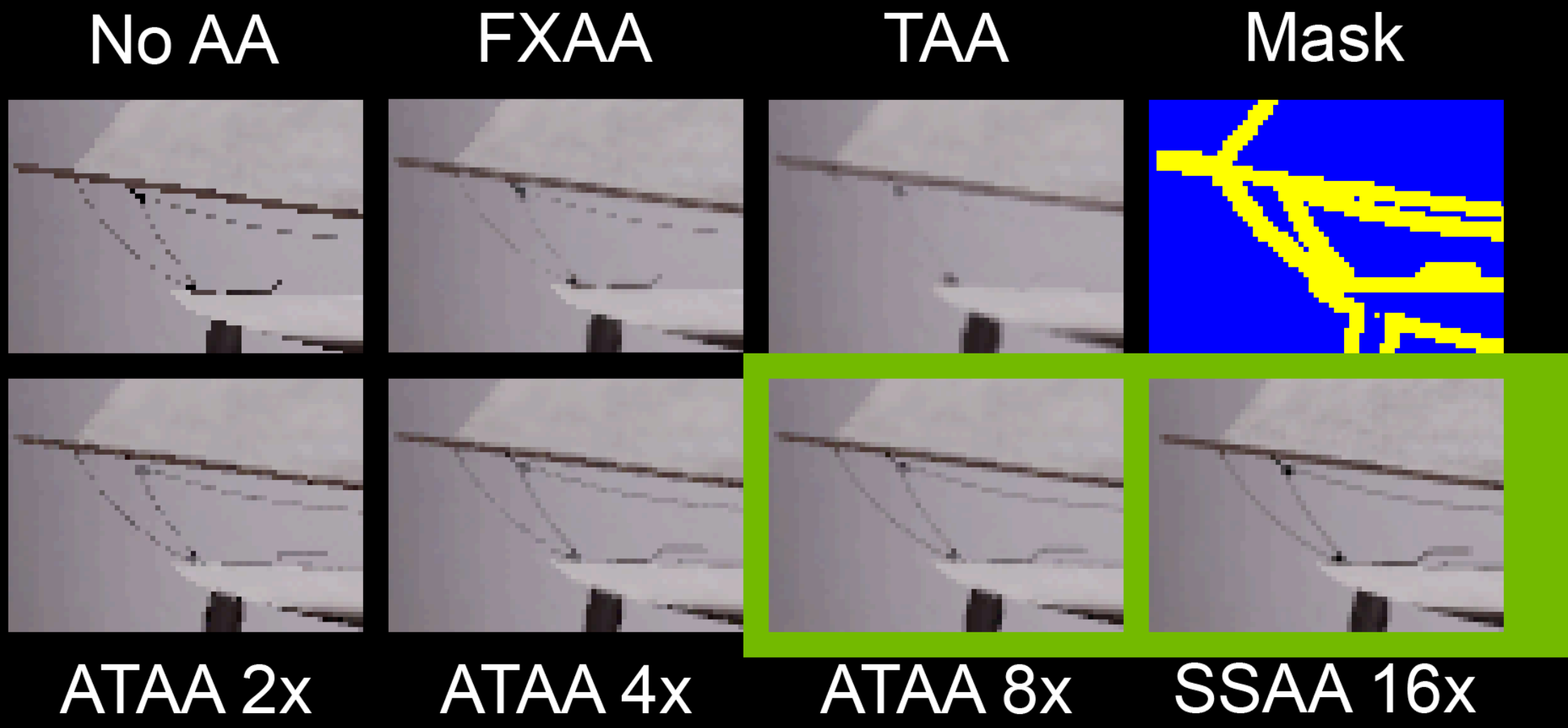

No AA
FXAA
TAA
Mask
ATAA 2x
ATAA 4x
ATAA 8x
SSAA 16x

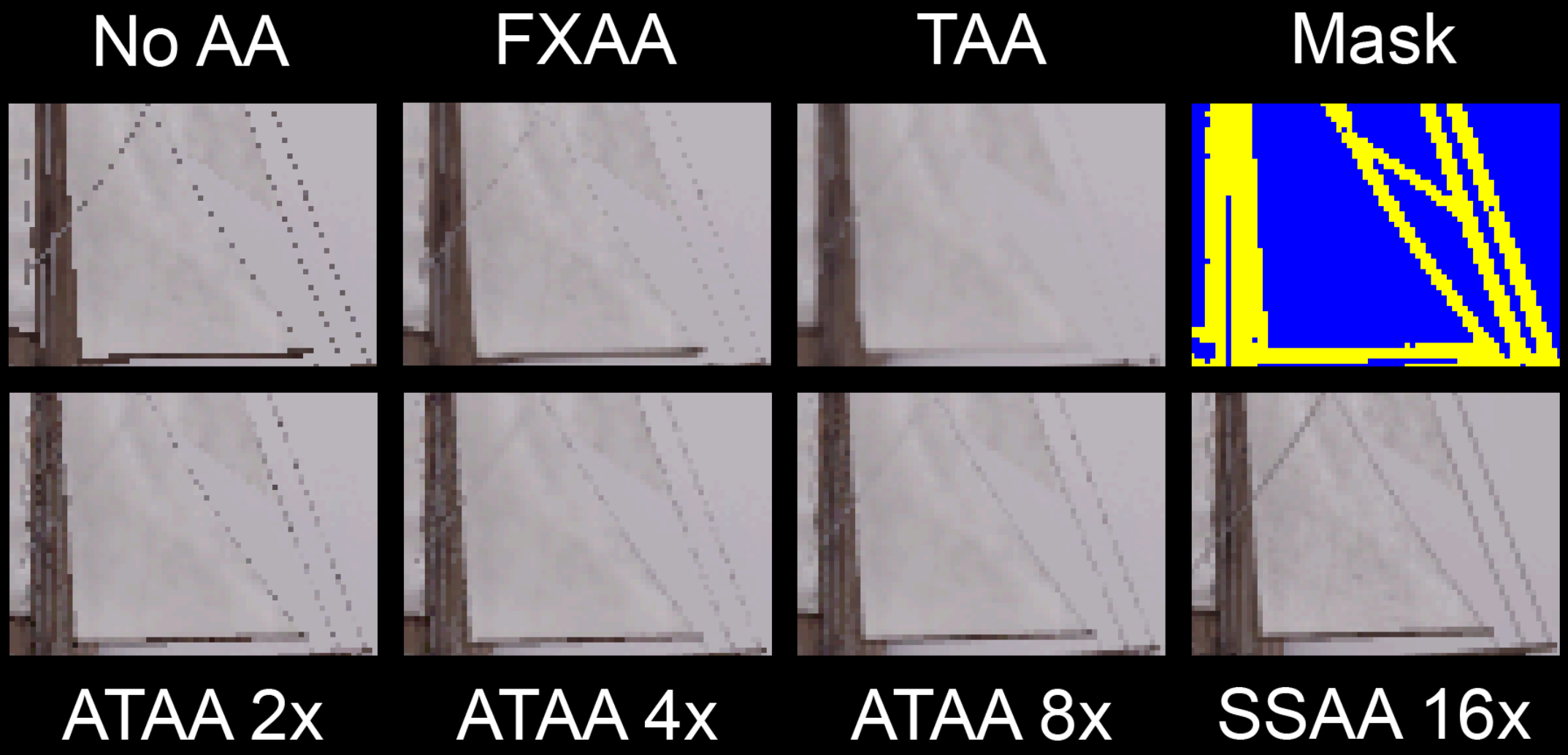

No AA
FXAA
TAA
Mask
ATAA 2x
ATAA 4x
ATAA 8x
SSAA 16x

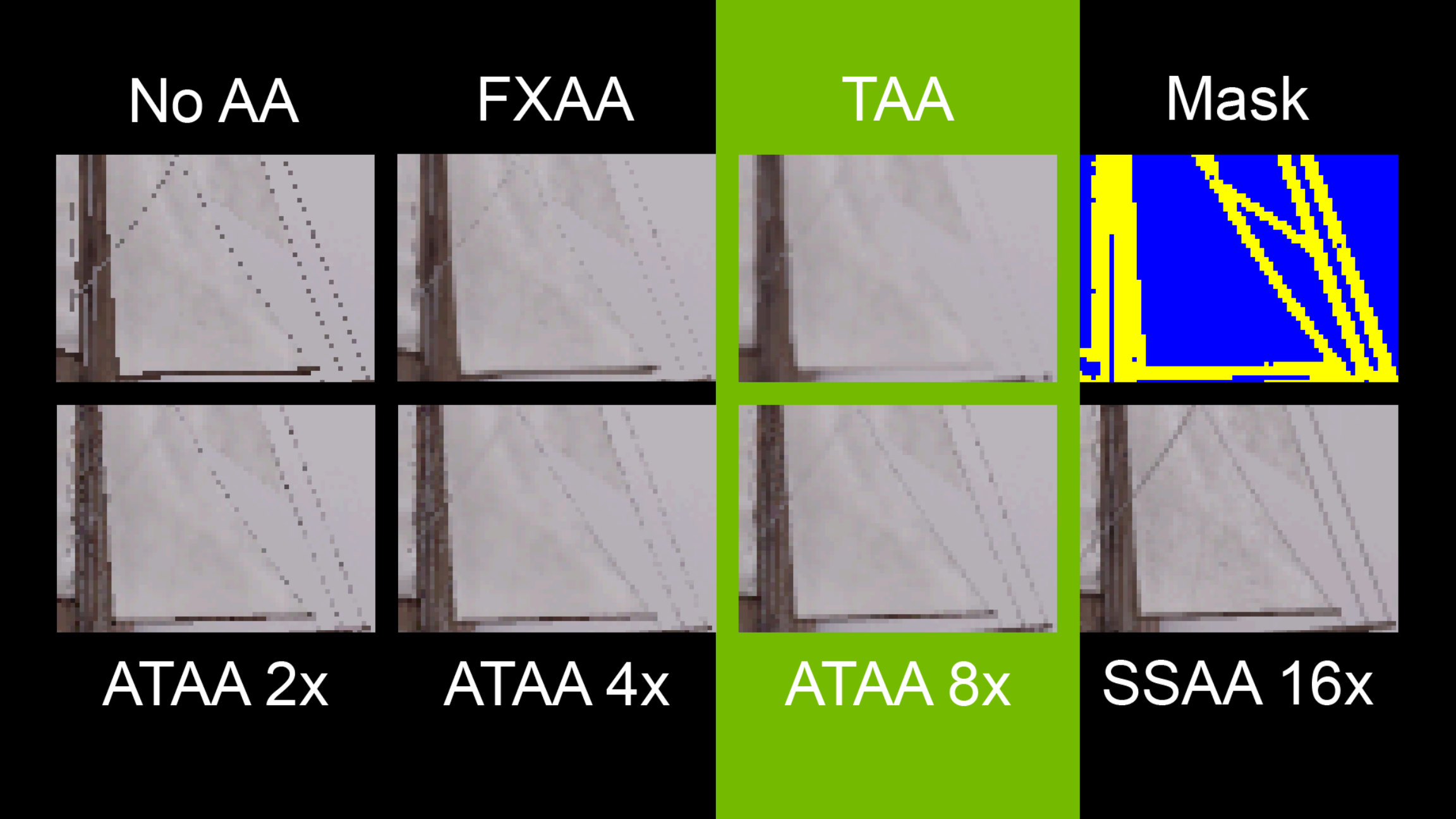

No AA
FXAA
TAA
Mask
ATAA 2x
ATAA 4x
ATAA 8x
SSAA 16x

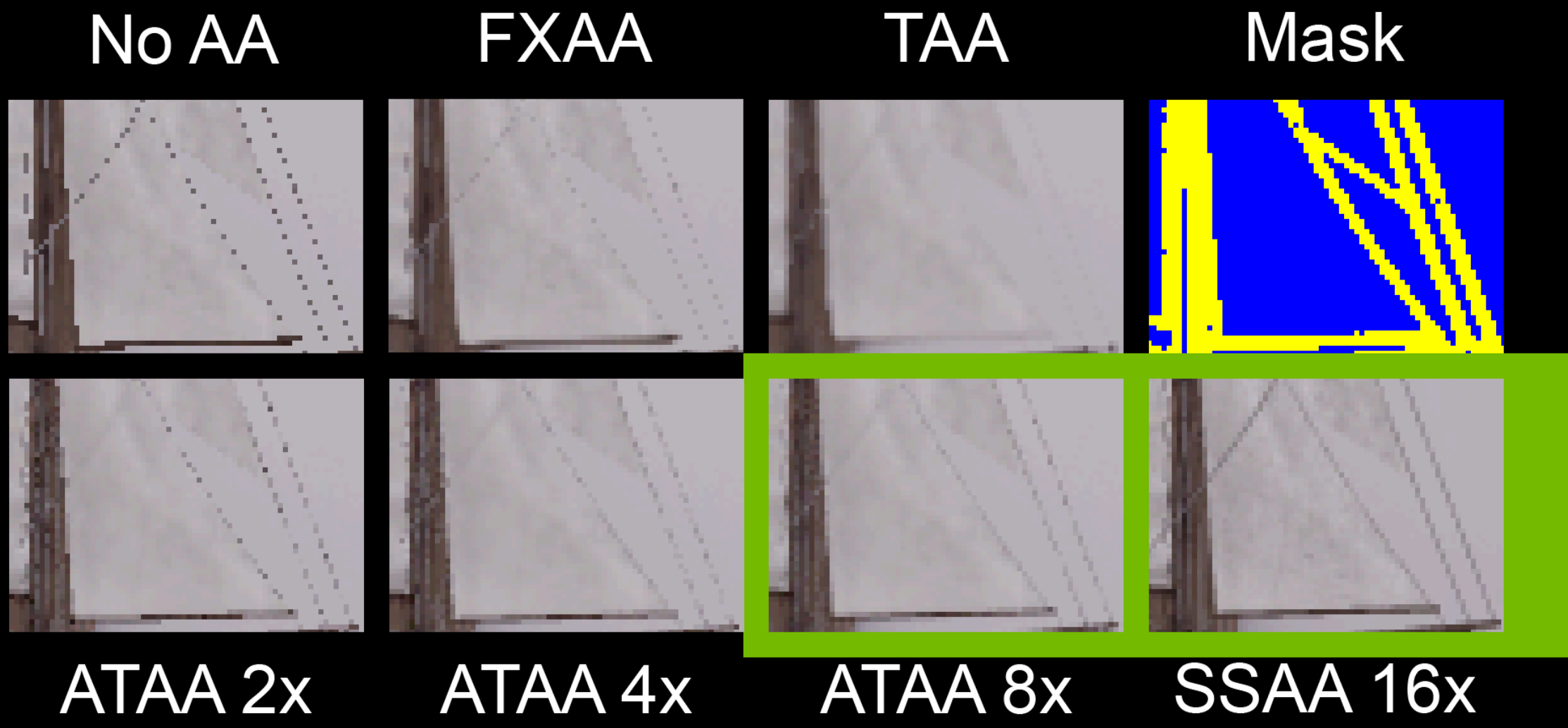

No AA
FXAA
TAA
Mask
ATAA 2x
ATAA 4x
ATAA 8x
SSAA 16x

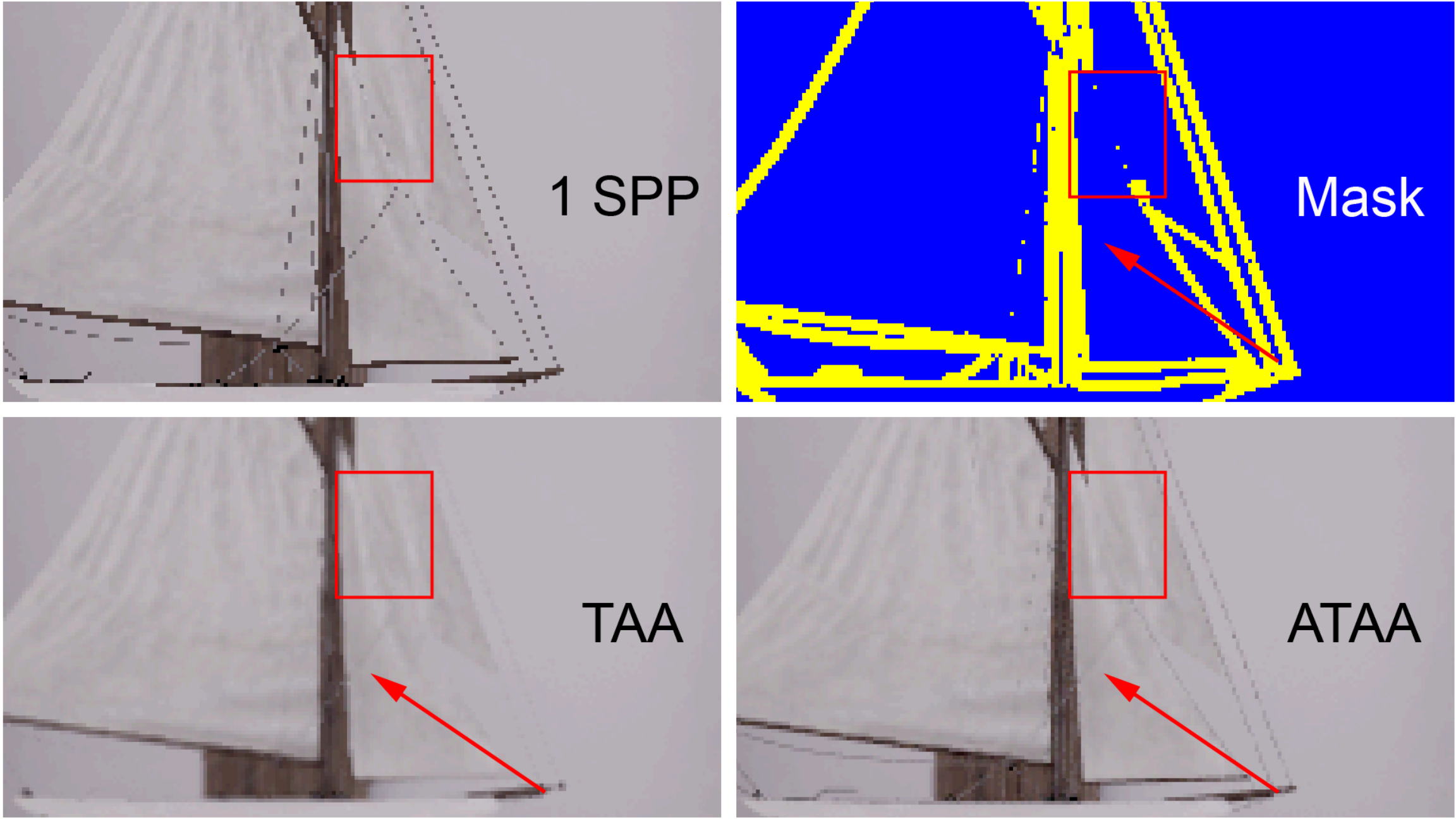

No AA
FXAA
TAA
Mask
ATAA 2x
ATAA 4x
ATAA 8x
SSAA 16x


No AA
FXAA
TAA
Mask
ATAA 2x
ATAA 4x
ATAA 8x
SSAA 16x


No AA
FXAA
TAA
Mask
ATAA 2x
ATAA 4x
ATAA 8x
SSAA 16x


No AA
FXAA
TAA
Mask
ATAA 2x
ATAA 4x
ATAA 8x
SSAA 16x


No AA
FXAA
TAA
Mask
ATAA 2x
ATAA 4x
ATAA 8x
SSAA 16x


1 SPP
Mask
TAA
ATAA

Adaptive Temporal Antialiasing

Nota Bene!

Lighting and shading methods between ray and raster must match!

Shadow Rays vs. Shadow Maps, Reflection Cubemaps vs. Reflection Rays

Be aware of pre-AA screen-space algorithms (DoF, Lens Flare, SSAO)

Denoising of area light contributions can make your life tricky

Performance
Titan V

1920x1080 resolution, **107,881** pixels selected for RT, **5.2%** of total image resolution

Trace, Material Evaluation, Dynamic Lighting, Reflection Probe, 1 Shadow Ray

Variant	Rays	GPU Time (ms)	Optimized GPU Time (ms)
ATAA 8x	1,693,280	18.40	16.85
ATAA 4x	846,640	9.30	8.55
ATAA 2x	423,320	4.60	4.28

Performance figures reported in milliseconds (ms)

Conclusions

Demonstrated a **practical hybrid AA solution** in a production game engine **for the first time**

The adaptive hybrid strategy injects the advantages of best-quality offline AA strategies while avoiding the limitations of existing best performance real-time methods

Real-time even on first-generation ray tracing commodity hardware and software

Just the beginning: of real-time hybrid raster + adaptive ray tracing

Adaptive Temporal Antialiasing

Future Work

Performance: dominated by the ray trace. How to handle about denoising approaches?

DXR is still an “experimental” feature of Windows 10!

Mainstream GPUs do not yet exist that support the DXR API

RT ecosystem is still evolving: wide-spread deployment in games not yet recommended

Adaptive Temporal Antialiasing

Future Work

Texture LoD: no forward-difference derivatives in DXR, how to evaluate texture mipmap levels in arbitrary material graphs is an open problem

Improve Sampling & Filtering: casting rays in static 8x, 4x, or 2x MSAA *n*-rooks patterns

Segmentation: limited by the 1spp raster input. Conservative raster to the rescue?

Improved Adaptivity: enforcing fixed per frame ray budgets and per-pixel ray adaptivity

Acknowledgements

Coauthors: Josef Spjut, Holger Gruen, Rahul Sathe, and Morgan McGuire

Ignacio llamas, Edward Liu, and the entire NVIDIA ray tracing team

Epic Games

@acmarrs


NVIDIA