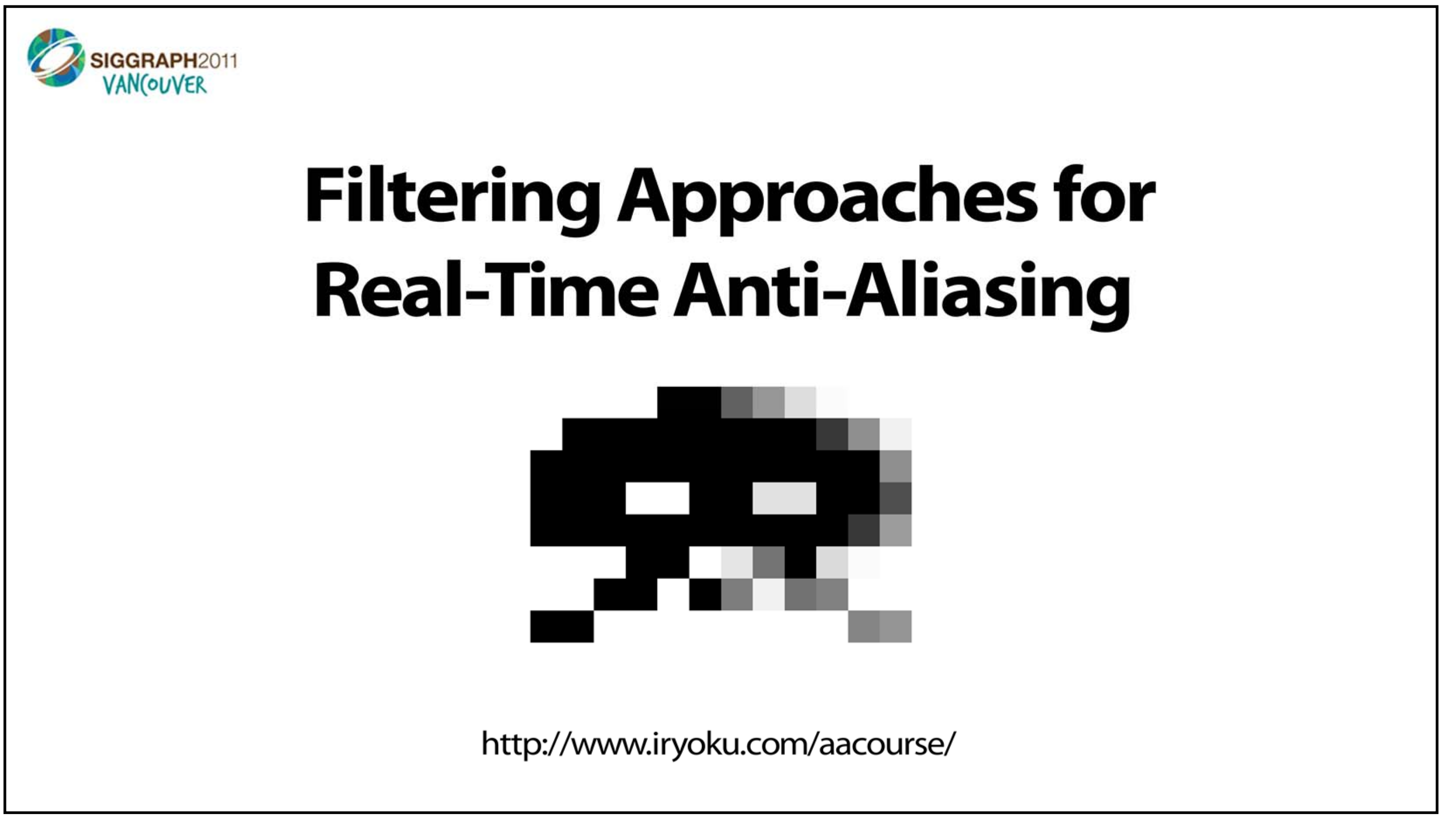
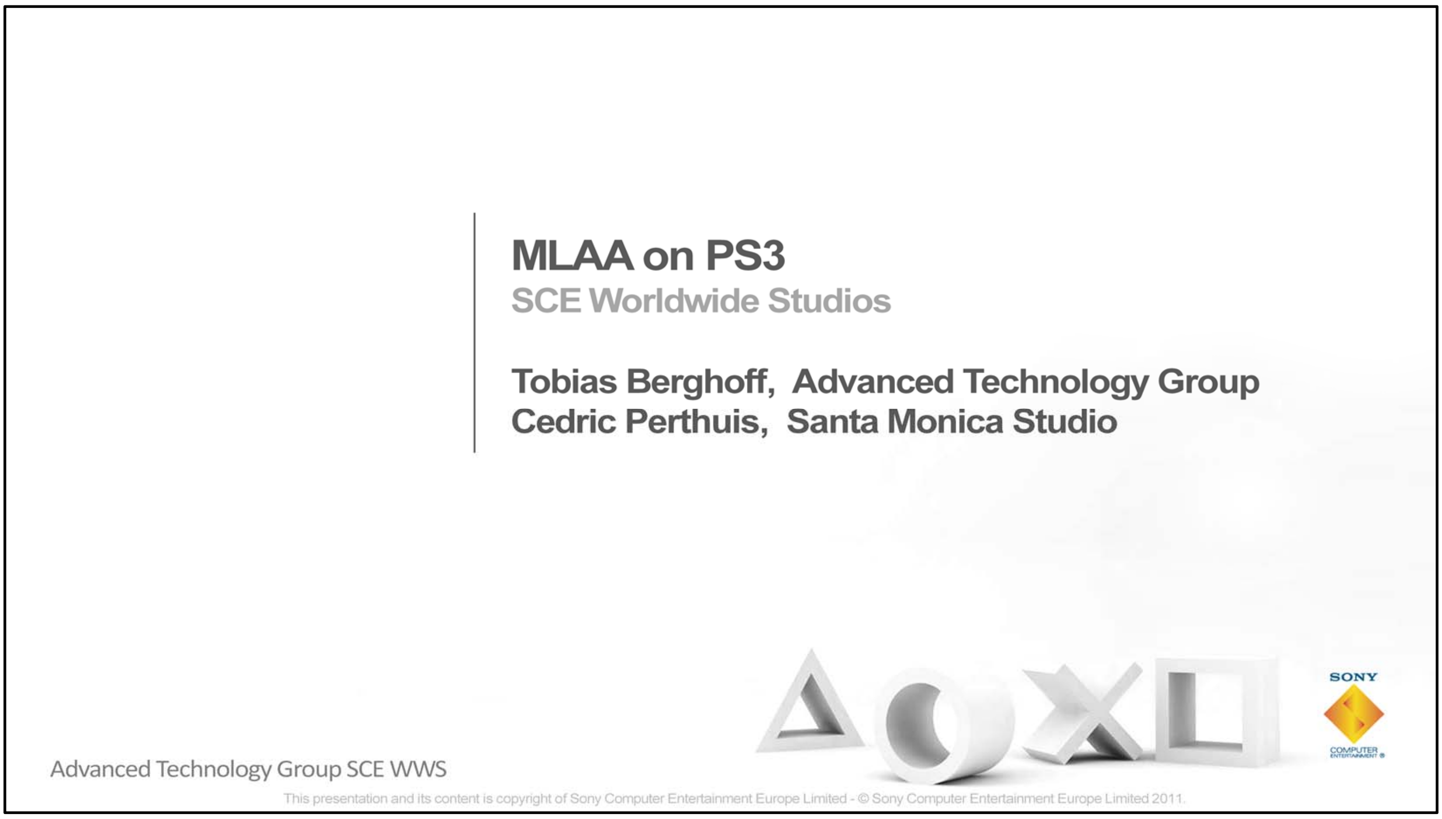


SIGGRAPH2011 VANCOUVER

Filtering Approaches for Real-Time Anti-Aliasing

http://www.iryoku.com/aacourse/



MLAA on PS3

SCE Worldwide Studios

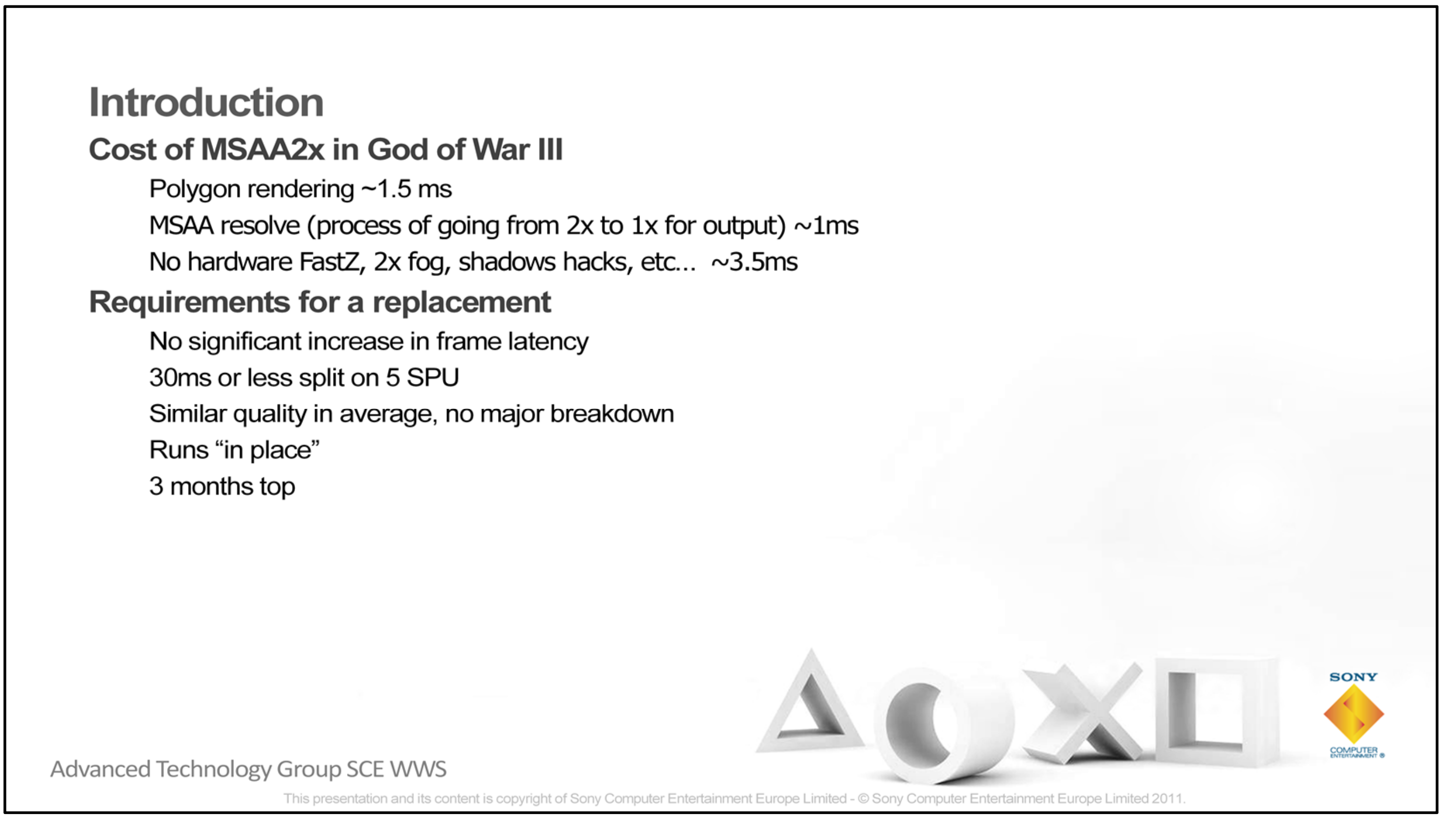
Tobias Berghoff, Advanced Technology Group
Cedric Perthuis, Santa Monica Studio

Advanced Technology Group SCE WWS

Hello,

I am Cedric Perthuis, I am from Sony Santa Monica and I am here with Tobias Berghoff from the SCE advance technology group to talk to you about MLAA on PS3. Tobias will first present the Playstation edge MLAA library and I will then show you how we used it and the benefits we got from it in God of War III.

About 2 years ago, Alexander' Siggraph paper about MLAA started circulating on our various email threads within the Sony game studios. We were starting the final push with about 6 months to go before our gold master. We still had many graphics features to add and the artists hadn't started polishing the game. There were many unknowns and only a few certainties in term of what we would make it into the final game. MSAA was for us at that point one of a certainties. Still we somehow decided to measure the true cost of MLAA in the game.



Introduction

Cost of MSAA2x in God of War III

- Polygon rendering ~1.5 ms
- MSAA resolve (process of going from 2x to 1x for output) ~1ms
- No hardware FastZ, 2x fog, shadows hacks, etc… ~3.5ms

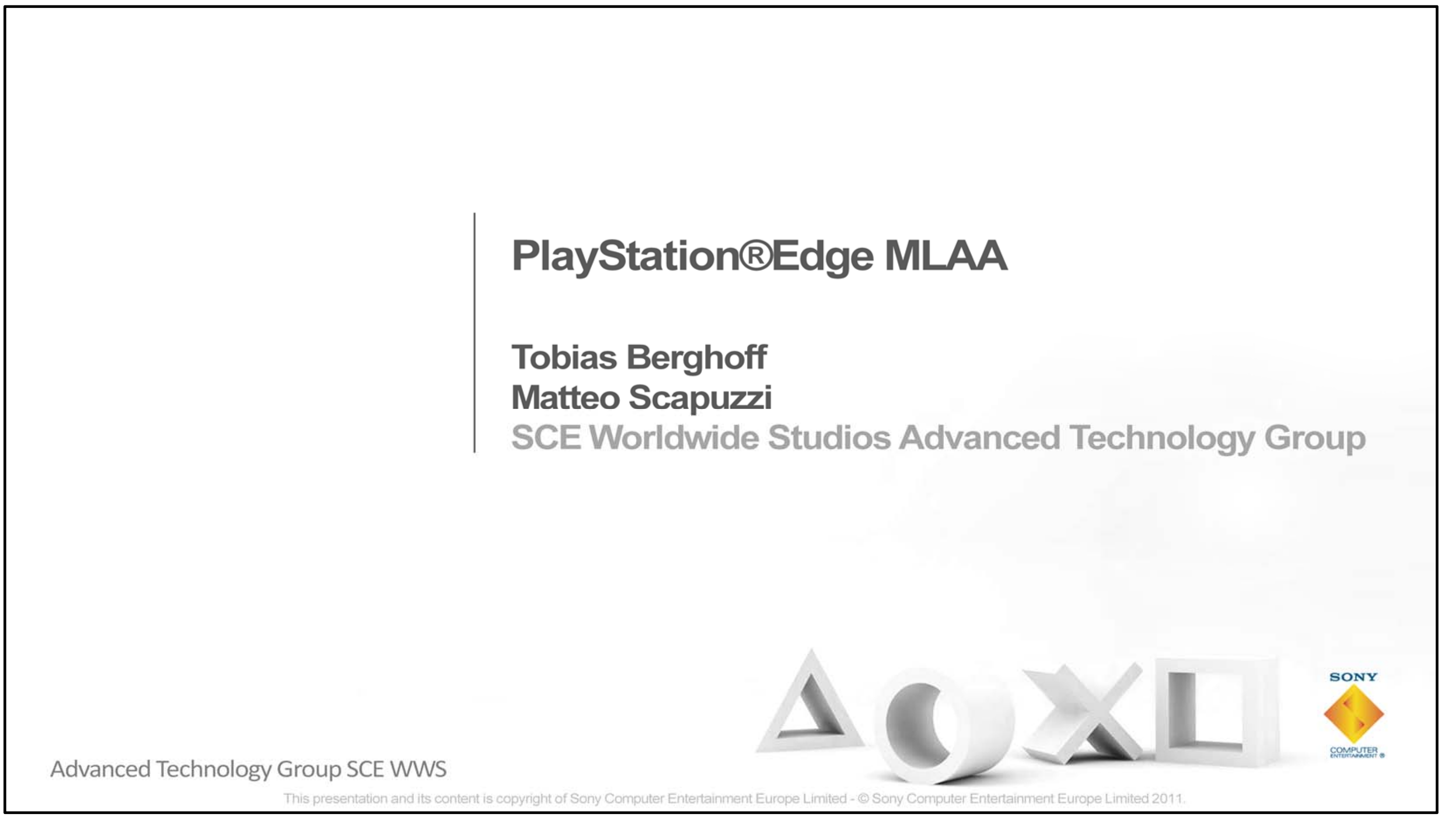
Requirements for a replacement

- No significant increase in frame latency
- 30ms or less split on 5 SPU
- Similar quality in average, no major breakdown
- Runs “in place”
- 3 months top

Advanced Technology Group SCE WWS

We were a bit taken by surprise, the cost was more than what we were expecting, total it was about 6ms. It was not because of the hardware MSAA itself, but rather because of everything else we had to do around it.

Few weeks later we heard about our technology team in Europe investigating MLAA. We seized the opportunity and sent them our list of requirements.



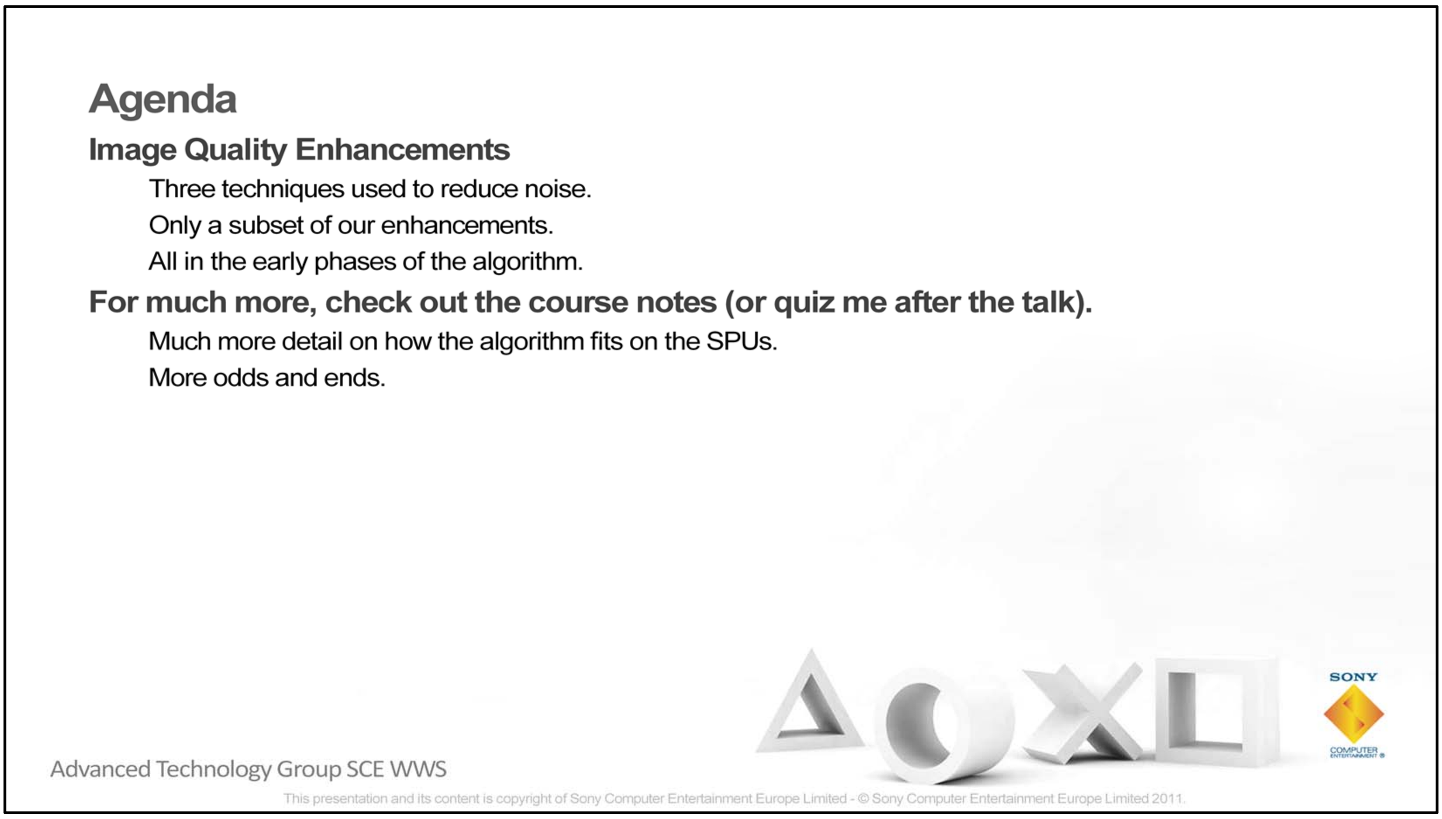
PlayStation®Edge MLAA

Tobias Berghoff
Matteo Scapuzzi
SCE Worldwide Studios Advanced Technology Group

Advanced Technology Group SCE WWS

Thank you, Cedric.

Today I'll be talking about Playstation Edge MLAA, which has been available to Playstation 3 developers for about a year. Most of the things I'll talk about today are new however, and have not yet been released. This work was done in collaboration with Matteo Scapuzzi, who can't be here today.



Agenda

Image Quality Enhancements

- Three techniques used to reduce noise.
- Only a subset of our enhancements.
- All in the early phases of the algorithm.

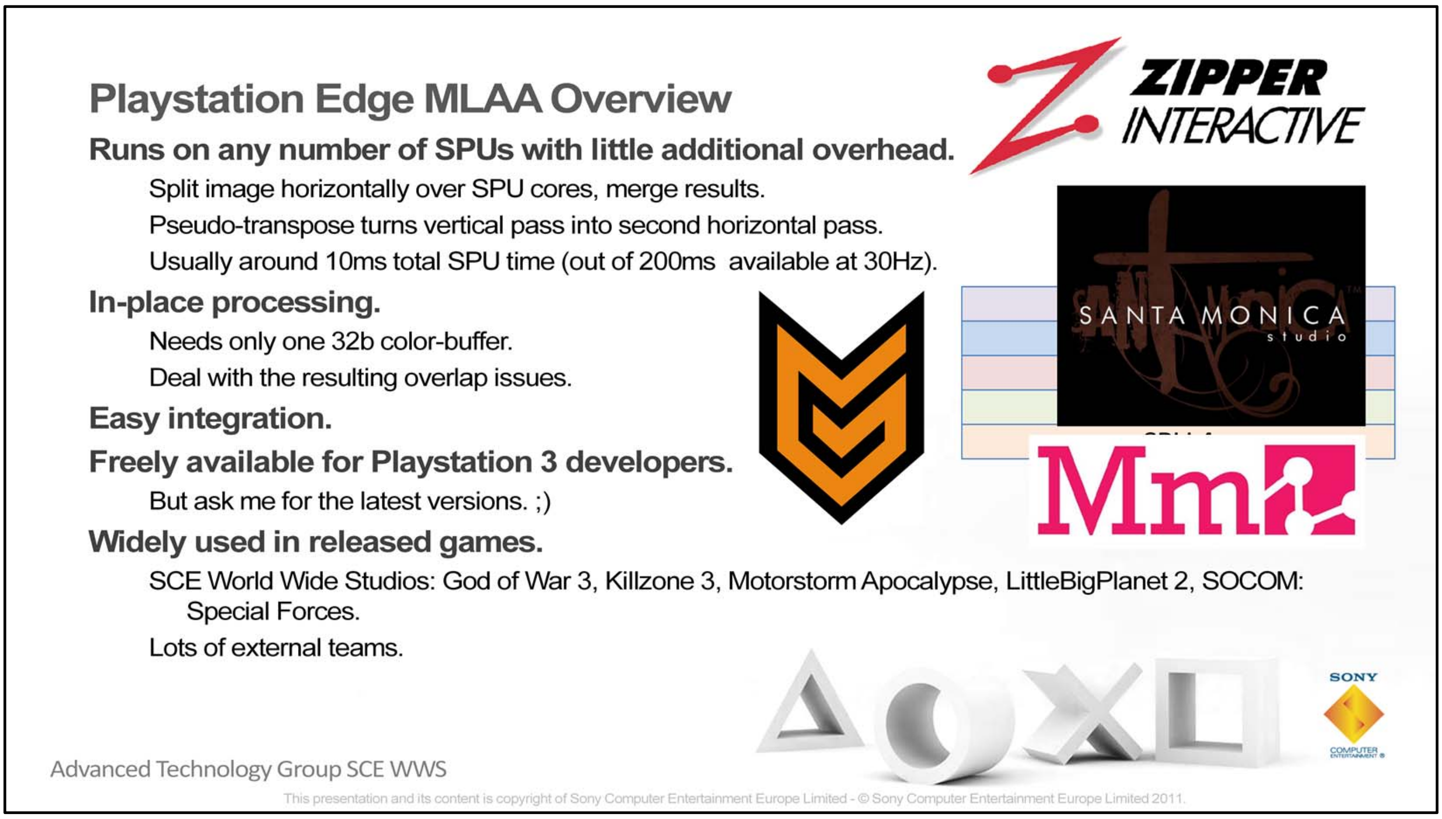
For much more, check out the course notes (or quiz me after the talk).

- Much more detail on how the algorithm fits on the SPUs.
- More odds and ends.

Advanced Technology Group SCE WWS

Due to the extremely tight time limit, I’ll focus on the improvements we made to early phases of MLAA, and how they affect image quality. This is sort of a “greatest hits” selection of techniques that I think are helpful for most post-process AA systems. There are more nice features in the code, but this is all I could fit in.

If you are interested in more details, please have a look at the course notes, or if you are a PS3 developer, go and grab the actual docs.



Playstation Edge MLAA Overview

Runs on any number of SPUs with little additional overhead.

- Split image horizontally over SPU cores, merge results.
- Pseudo-transpose turns vertical pass into second horizontal pass.
- Usually around 10ms total SPU time (out of 200ms available at 30Hz).

In-place processing.

- Needs only one 32b color-buffer.
- Deal with the resulting overlap issues.

Easy integration.

Freely available for Playstation 3 developers.

- But ask me for the latest versions. ;)

Widely used in released games.

- SCE World Wide Studios: God of War 3, Killzone 3, Motorstorm Apocalypse, LittleBigPlanet 2, SOCOM: Special Forces.
- Lots of external teams.

I'll start with some general bulletpoints about the system.

[click]

Naturally, we run on the PS3's SPU cores. Both our MLAA passes are actually horizontal, with a pseudo-transpose in between to make that work. For each pass, we horizontally split the framebuffer into equal-sized chunks and let one SPU loose on each of them.

The entire operation usually takes about 10ms of total SPU time for a 720p buffer, depending on image complexity. At 30Hz, that 5% of your total SPU budget.

[click]

In-place processing was mandated by our game teams, to minimize memory usage,

[click]

and the fastest integration into a game I know of took about 2 hours, so I suppose our integration package can't be that bad.

[click]

Finally, I should mention that this code has shipped in a whole lot of tiles, starting with God of War 3 last year. Many titles of our Worldwide Studios have shipped with it and many external teams are using it as well.

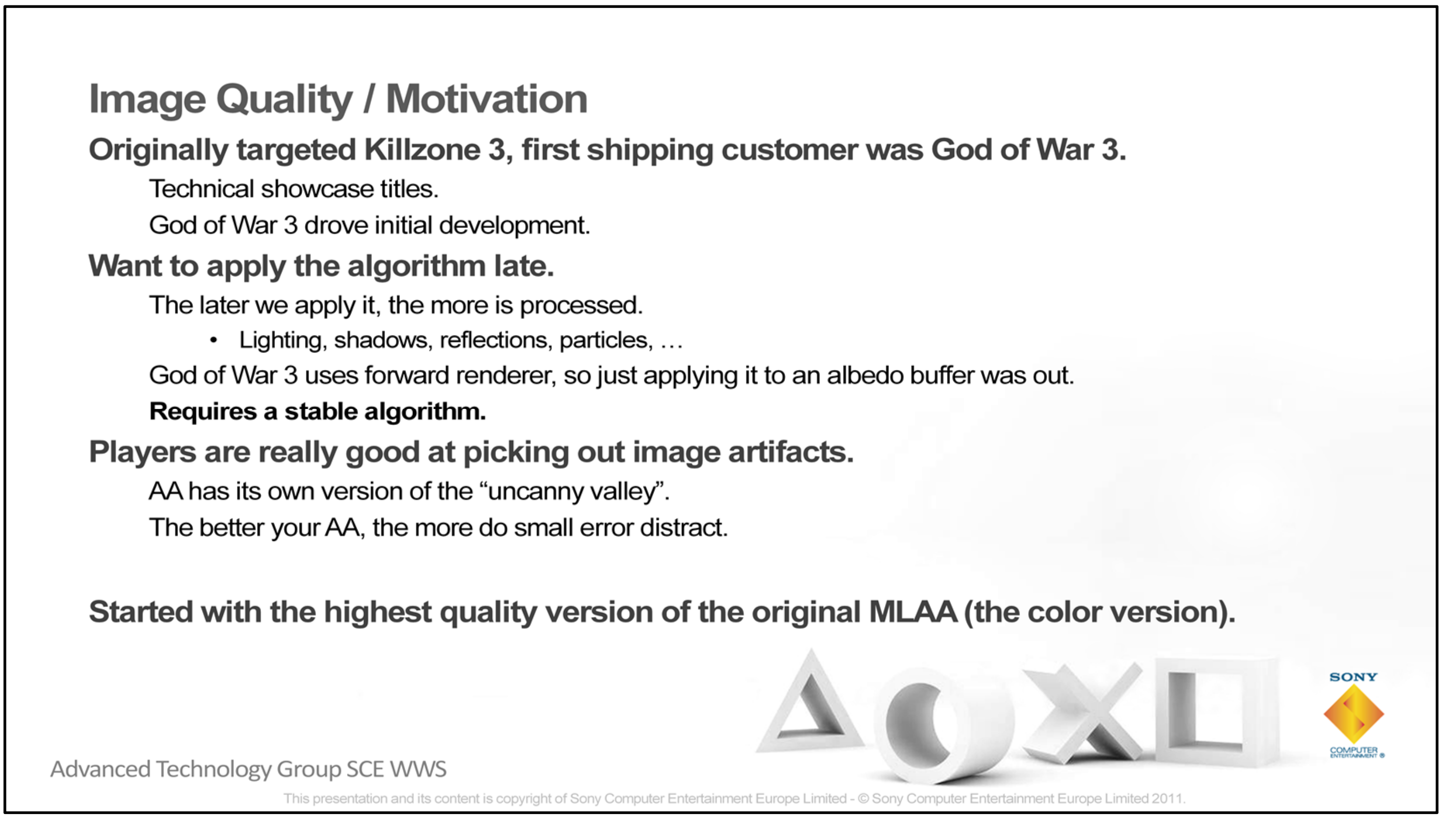


Image Quality / Motivation

Originally targeted Killzone 3, first shipping customer was God of War 3.

Technical showcase titles.

God of War 3 drove initial development.

Want to apply the algorithm late.

The later we apply it, the more is processed.

- Lighting, shadows, reflections, particles, …

God of War 3 uses forward renderer, so just applying it to an albedo buffer was out.

Requires a stable algorithm.

Players are really good at picking out image artifacts.

AA has its own version of the “uncanny valley”.

The better your AA, the more do small error distract.

Started with the highest quality version of the original MLAA (the color version).

Advanced Technology Group SCE WWS

After that short message from our sponsor, let’s talk about image quality.

The first title shipping with MLAA was God of War 3, but originally, we looked into real-time MLAA for Killzone 3. Both of these titles are big technology showcases for us, so apart from them having to be really good games, we also need them to look great. Thus, any AA system we use needs to be a significant improvement over what was used before, which is 2xMSAA in both cases.

[click]

To get this quality, we not only want to smooth features that MSAA misses, such as alpha tested geometry, but the goal is to filter every bit of opaque and transparent geometry, all reflections, refractions, shadows, etc. And we want to do this after tonemapping. This all translates to MLAA being applied quite late in the frame, and to the inputs being quite varied in terms of luminance, which requires high temporal stability. And temporal stability is not MLAA’s greatest strength.

[click]

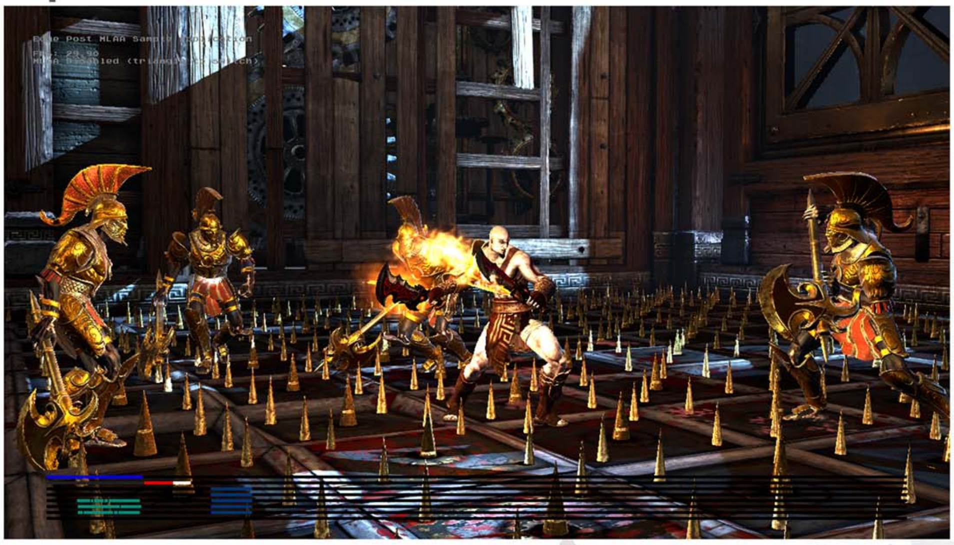
Furthermore, jaggies and artefacts in an otherwise clean image are very noticeable, leading to what I call the “AA uncanny valley”. This means that the perceived loss of quality due to an artifact gets worse as the image gets better, making this very much an uphill battle.

[click]

As a starting point, we used the “color” version of MLAA that Alex proposed in his original paper, which doesn’t seem to be what most people are using. It gave us better quality, so we went with this more expensive approach. This is interesting by itself, but I don’t have the time to go into this here.

So without further rambling, let’s have a look at the improvements.

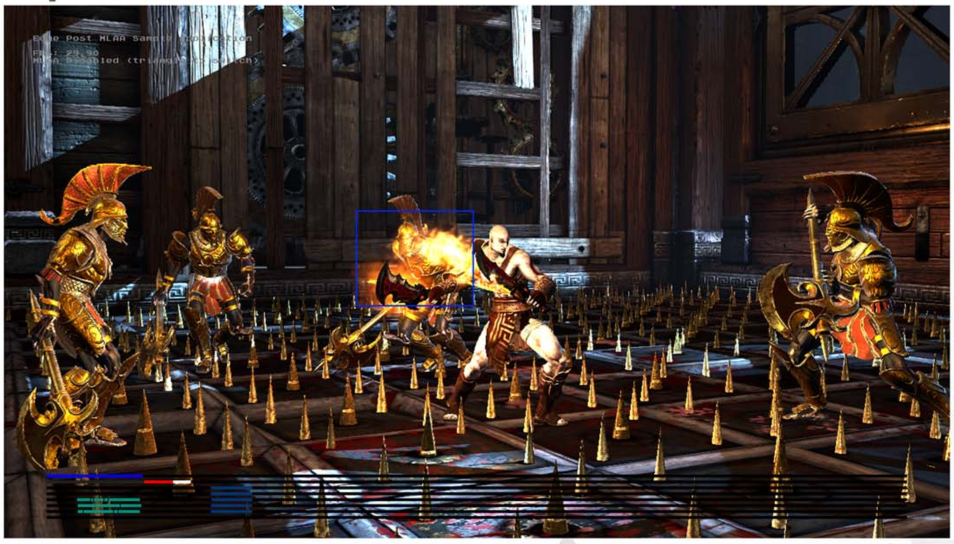
Meet the Spike Room


Post MLAA Sample Application
FPS: 29.96
MLAA Disabled (triangle to switch)

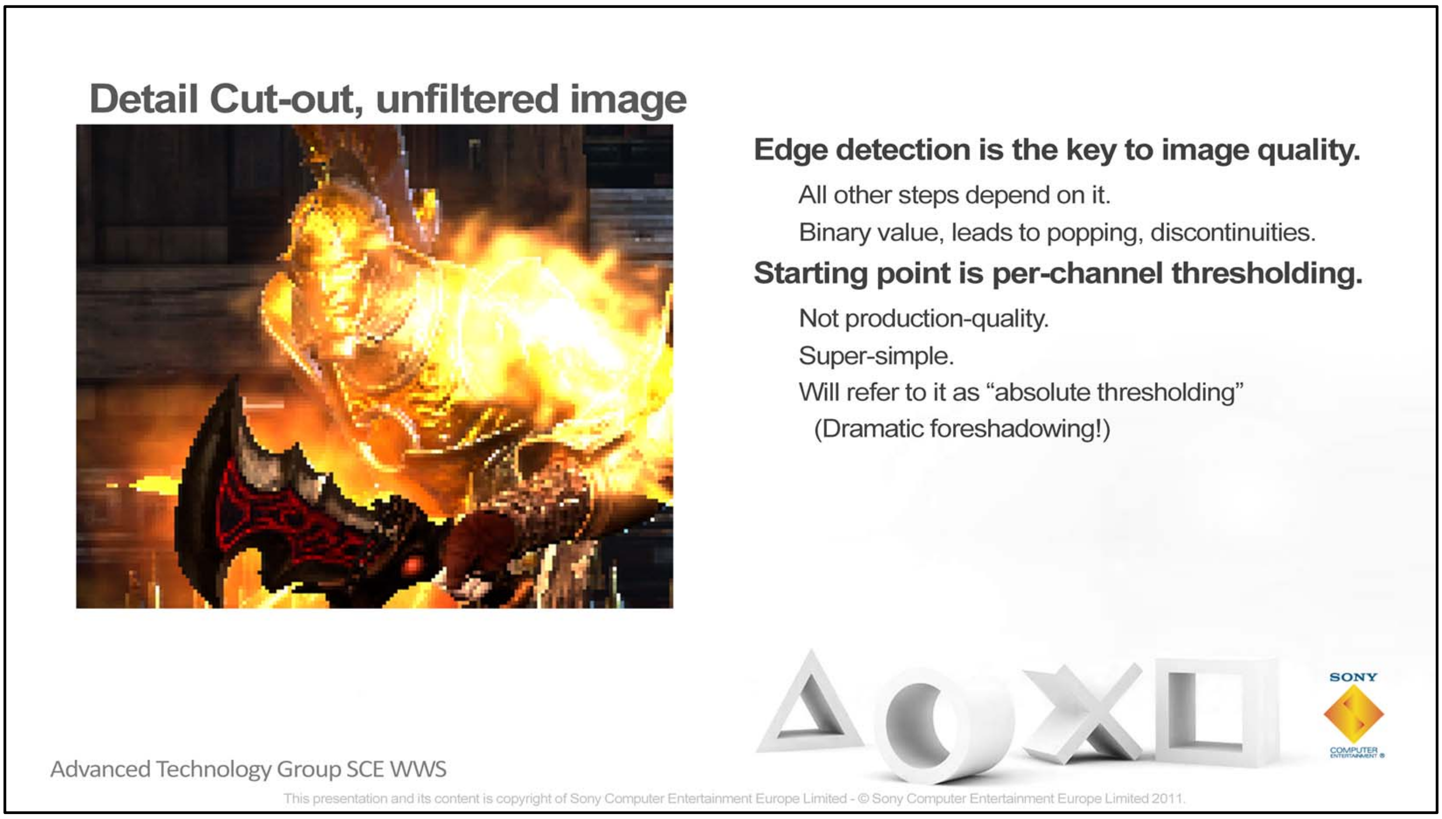
Advanced Technology Group SCE WWS

This is the God of War 3 spike room, which has been our go-to image since the initial development of the system. This is not actually what this room looks like in the final game, but I'll be using that as the source image.

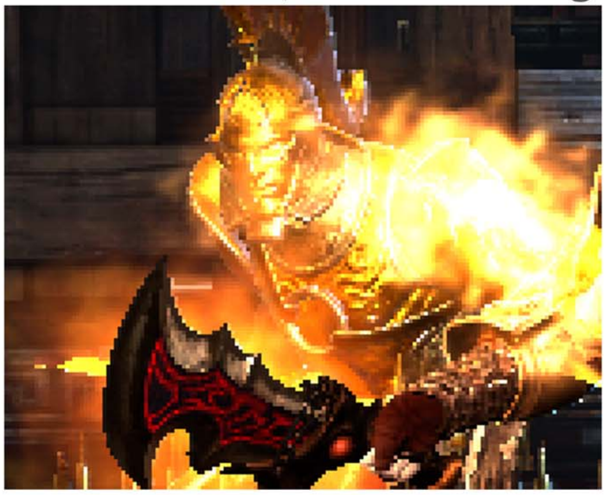
Meet the Spike Room



We'll start by looking at this cut-out.



Detail Cut-out, unfiltered image



Edge detection is the key to image quality.

- All other steps depend on it.
- Binary value, leads to popping, discontinuities.

Starting point is per-channel thresholding.

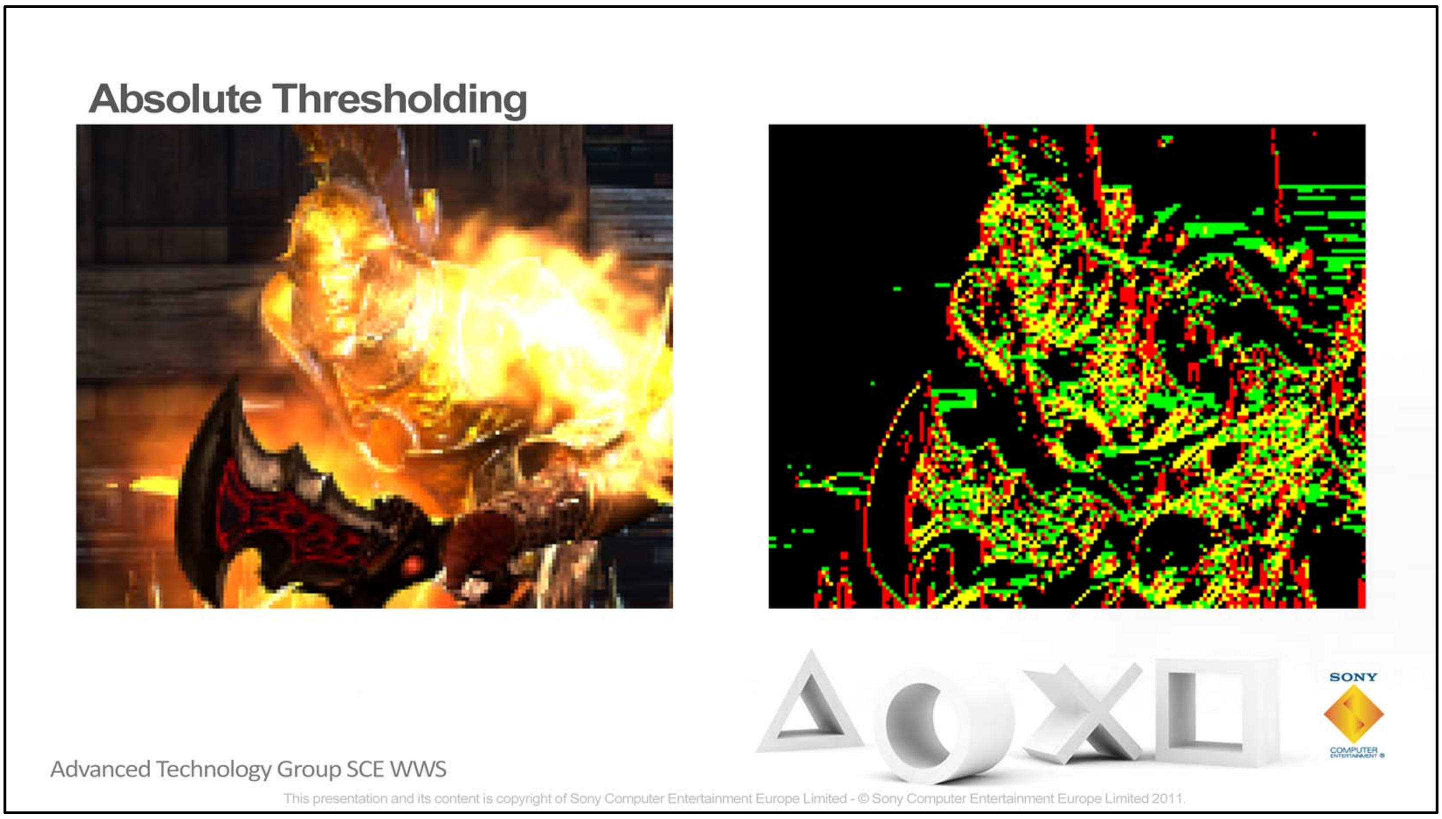
- Not production-quality.
- Super-simple.
- Will refer to it as “absolute thresholding”
  (Dramatic foreshadowing!)

Advanced Technology Group SCE WWS

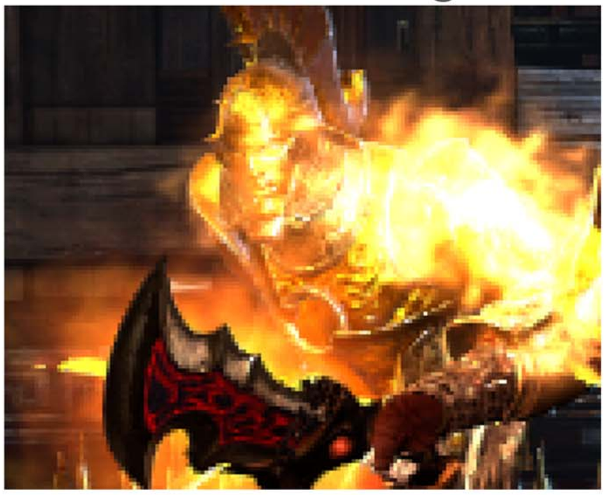
Let’s start with the edge detection, which is the first step of the algorithm, and all other operations depend on it. Since it’s used as an on/off switch for the rest of the operations, changes in the edge values are great sources of noise that you’ll never get rid of again. In my experience, the worst anti-aliasing artefacts are those that have no obvious causes, as they will attract the viewer’s attention without allowing them to easily discard them as artefacts. This is exactly what a noisy edge-detection will give you.

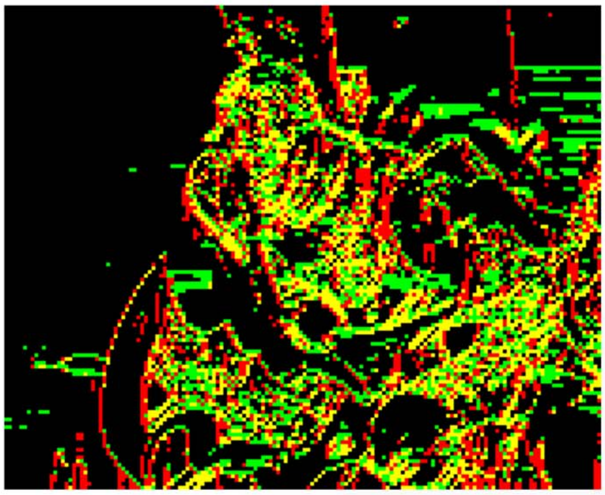
[click]

As a reference point, I’d like to start with the default edge detection algorithm often used, which is per-channel thresholding. The idea is to compute the absolute differences between the color channels of two pixels and if any of them are larger than some global threshold, that’s an edge. It’s very simple, it’s very fast, but not really production quality. We call it “absolute thresholding” for soon-to-be obvious reasons.



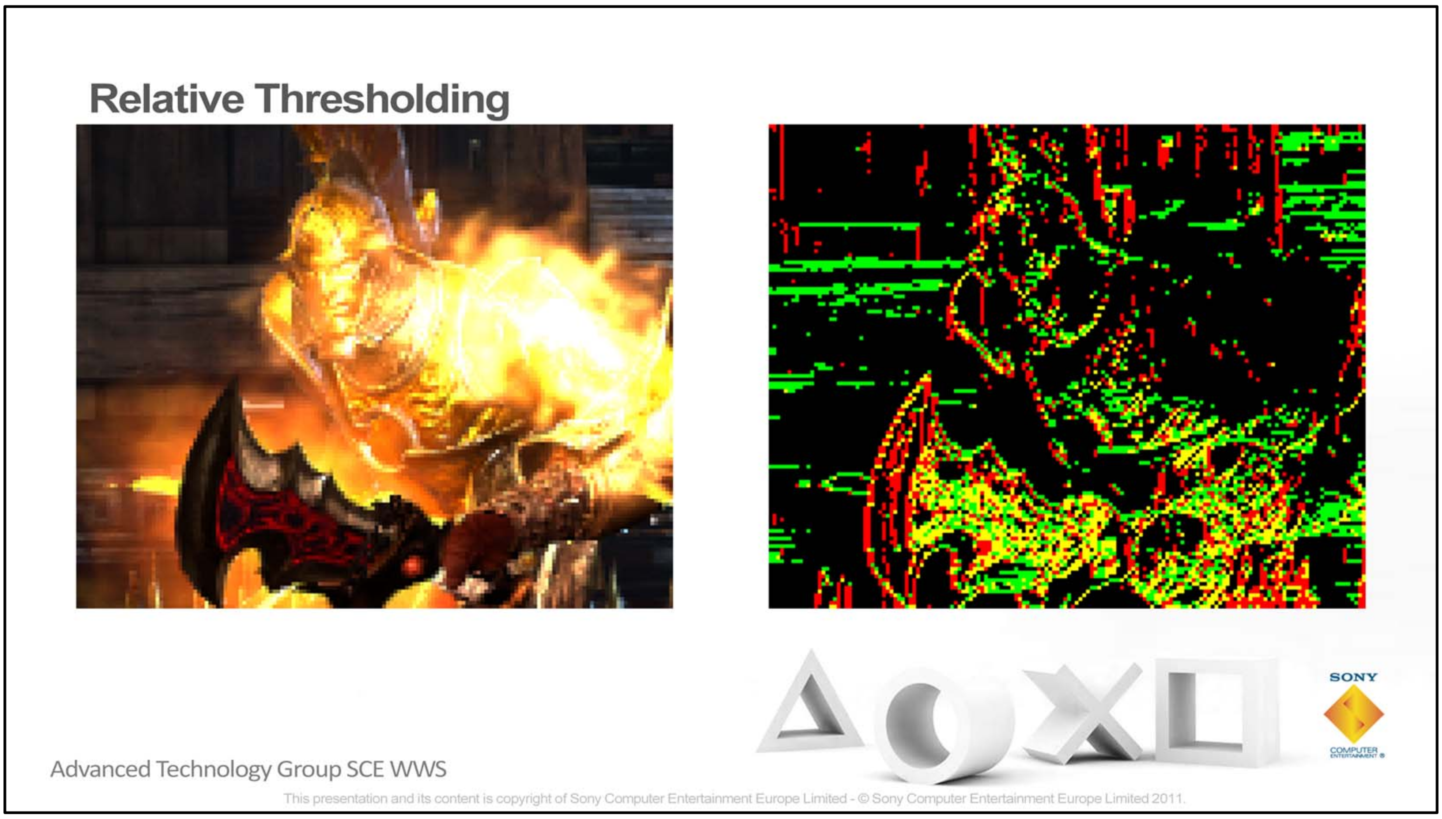
Absolute Thresholding



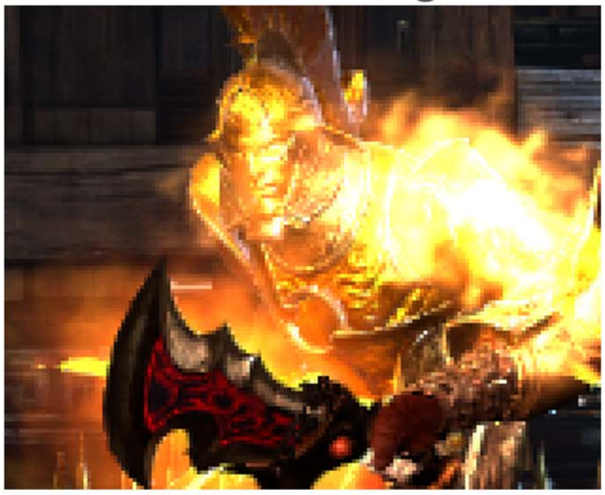


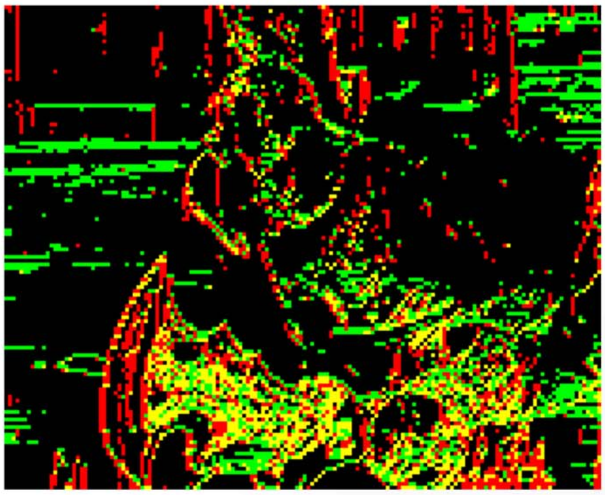
Advanced Technology Group SCE WWS

This is the result with absolute thresholding and a threshold of 31. As you can tell, the jaggies are mostly gone, but so is a lot of the detail.
On the right side you're seeing the edge buffer, where horizontal edges are green, vertical edges are red. Notice that the dark background regions are mostly edge free and the bright regions are pretty much all edges, except for the center of the flame, where the color is clamped. Only in the middle of our luma range do we get reliable detection.



Relative Thresholding

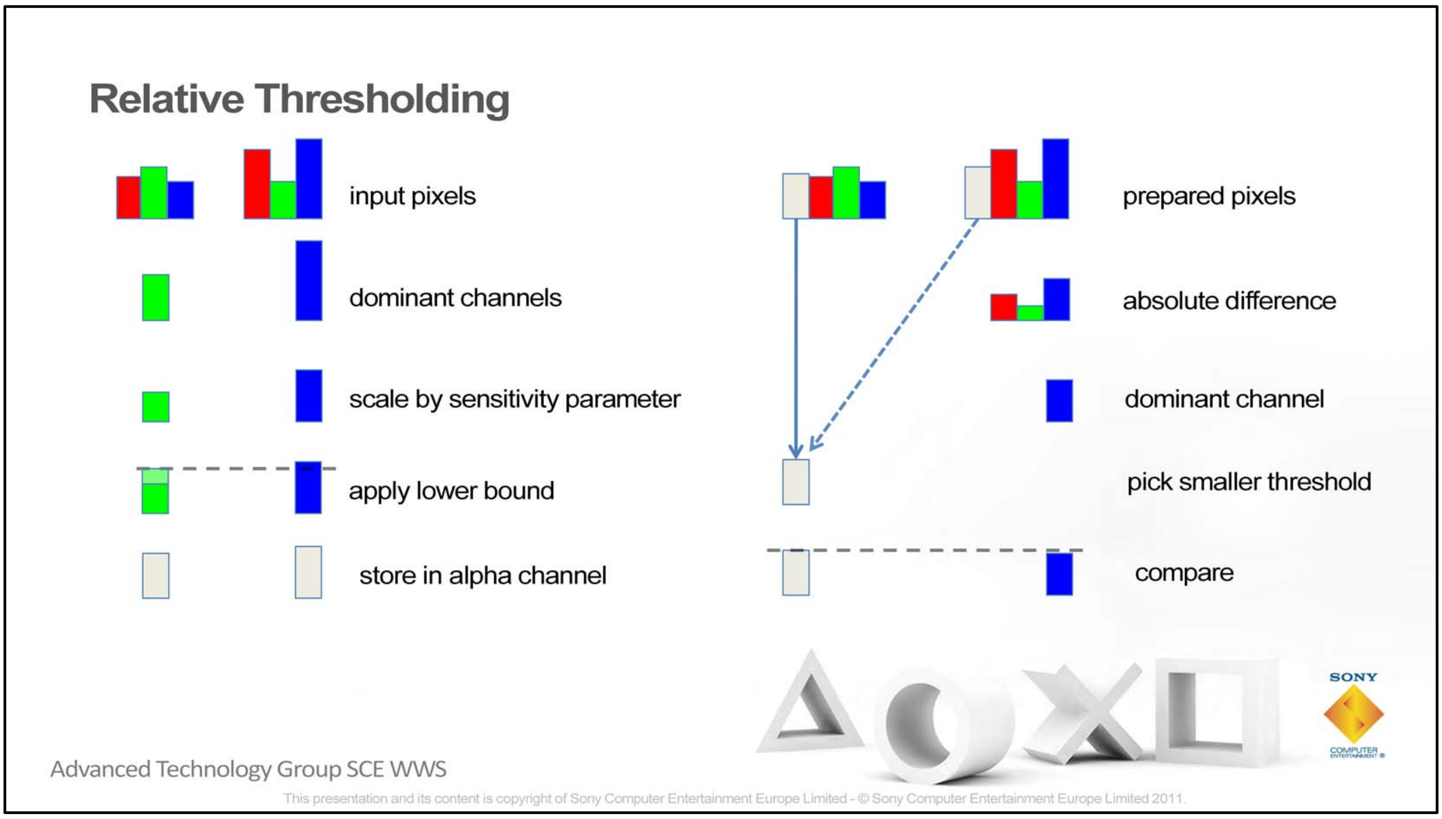




Advanced Technology Group SCE WWS

Here we have the result we get from what I call "relative thresholding". I'm sure I'm not the first one to use it, but my google-fu didn't get me anywhere. It's pretty simple and I'll explain it in a second. The important part to notice is that we now have good detection over the entire luma range, which also translates into a noticeably sharper image on the left. The edge detection response is almost inverted compared to the previous slide.

I have to point out that the edge of the blade is a bit less smooth, because there is actually a double edge in the source image here which absolute thresholding doesn't pick up, but when it is picked up causes blending to be less smooth.



Relative Thresholding

Advanced Technology Group SCE WWS

So, how does relative thresholding work? It's a two part process, the first of which prepares each pixel and the second performs the actual comparison. The overall idea is to compute the threshold value per pixel-pair, instead of having one global value for the entire image.

[click]
We start with two input pixels, but in reality you'd do this for all pixels before moving on to the second phase.

[click]
For each pixel, we first pick the numerically largest channel.

[click]
Following that, we scale those channels by a user defined value, which determines the sensitivity of the edge detection. We then apply a lower bound to the result, which is used to deal with very dark regions. The green channel here gets adjusted by this. If we didn't do this, we'd get lots of edges as colors and thus thresholds go towards zero.

[click]
The results are then stored back into the alpha channels of the source pixels.

[click]
The test itself starts with two prepare pixels.

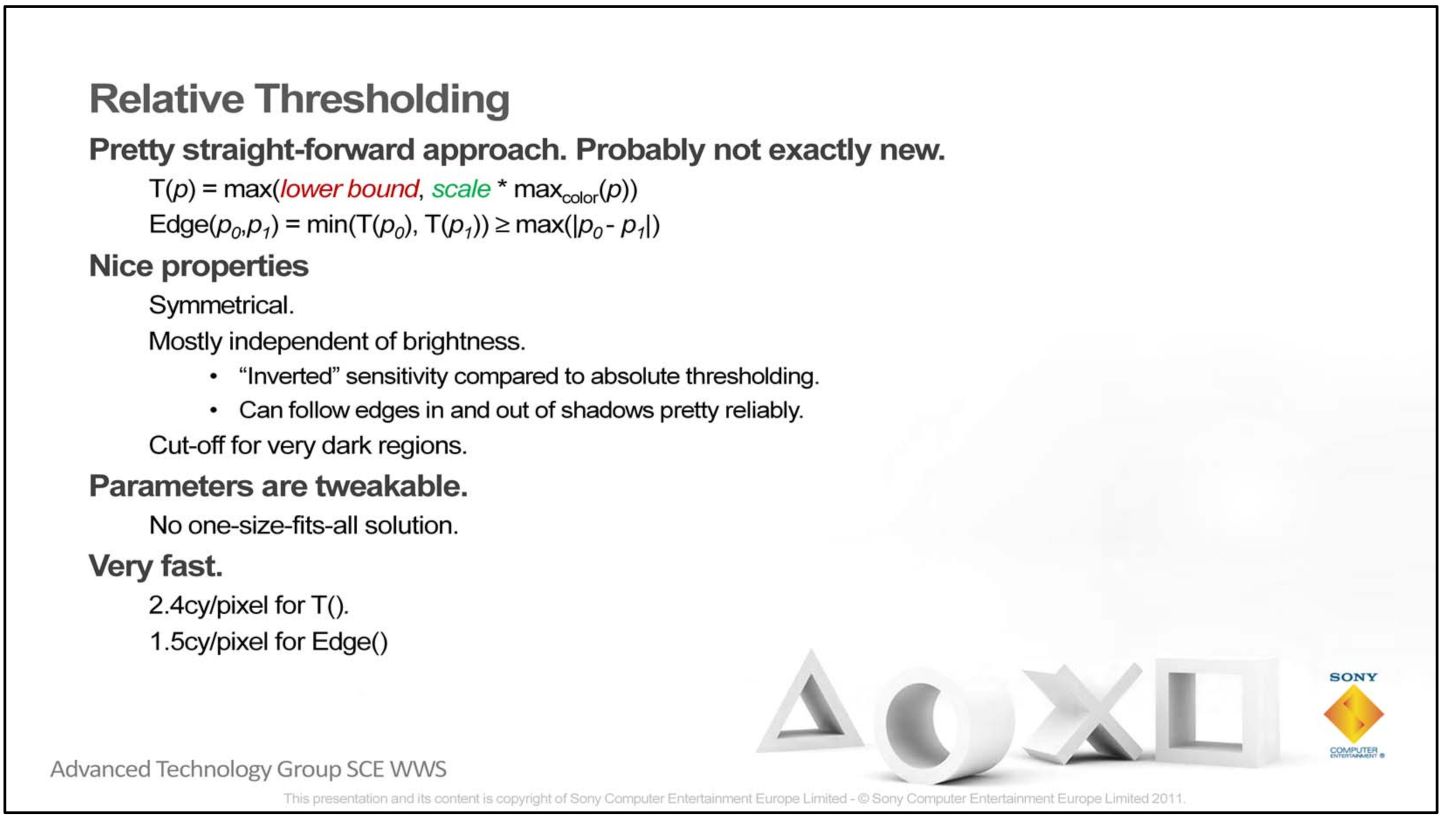
[click]
We compute the absolute difference of the color channels, just like in the absolute thresholding case.

[click]
Again pick the numerically largest channel of that,

[click]
but now we also pick the smaller of the two thresholds.

[click]
Finally, we compare the two and if the threshold is smaller, we have an edge.

We use the color channels and not luma to be able to detect chroma edges. The single threshold, as opposed to having per-channel thresholds, comes mostly from storage requirements, as we only have 8 bits, but we also didn't want to get an edge if the relative difference is only large in "unimportant" channels. The reason we pick the smaller threshold is to make the test symmetrical. Averaging the thresholds or taking the max works as well.
As I said: Pretty simple. And actually pretty fast.



Relative Thresholding

Pretty straight-forward approach. Probably not exactly new.

T(p) = max(*lower bound*, *scale* * $\max_{color}(p)$)

Edge(p_0,p_1) = min(T(p_0), T(p_1)) ≥ max(|p_0 - p_1|)

Nice properties

Symmetrical.

Mostly independent of brightness.

- “Inverted” sensitivity compared to absolute thresholding.
- Can follow edges in and out of shadows pretty reliably.

Cut-off for very dark regions.

Parameters are tweakable.

No one-size-fits-all solution.

Very fast.

2.4cy/pixel for T().

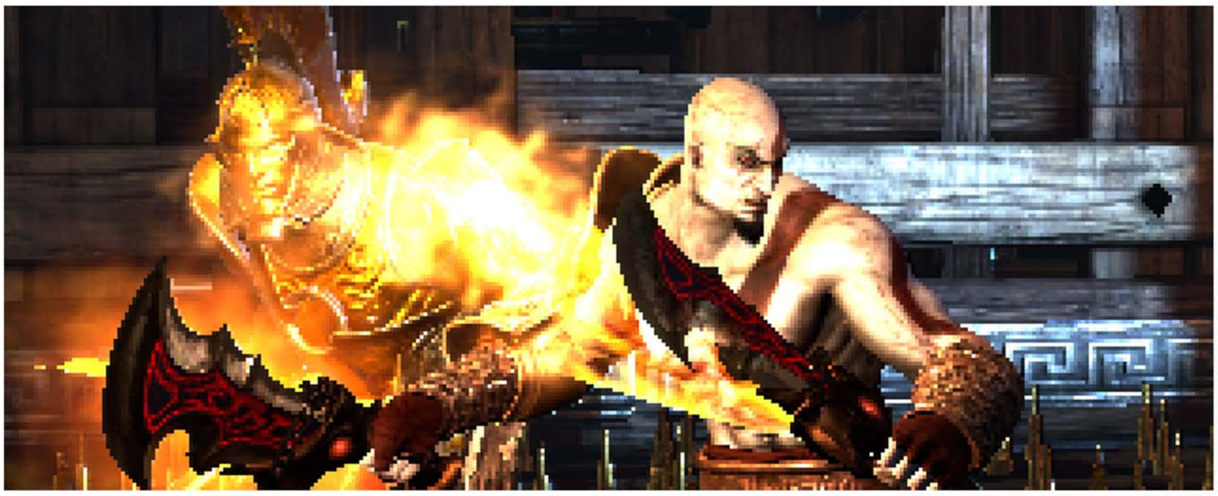
1.5cy/pixel for Edge()

Advanced Technology Group SCE WWS

With the scale and lower bound – which we call the base -, we have two parameters to tweak, which is quite important for us. No two games are the same, and for some titles, even different levels benefit greatly from using different settings.

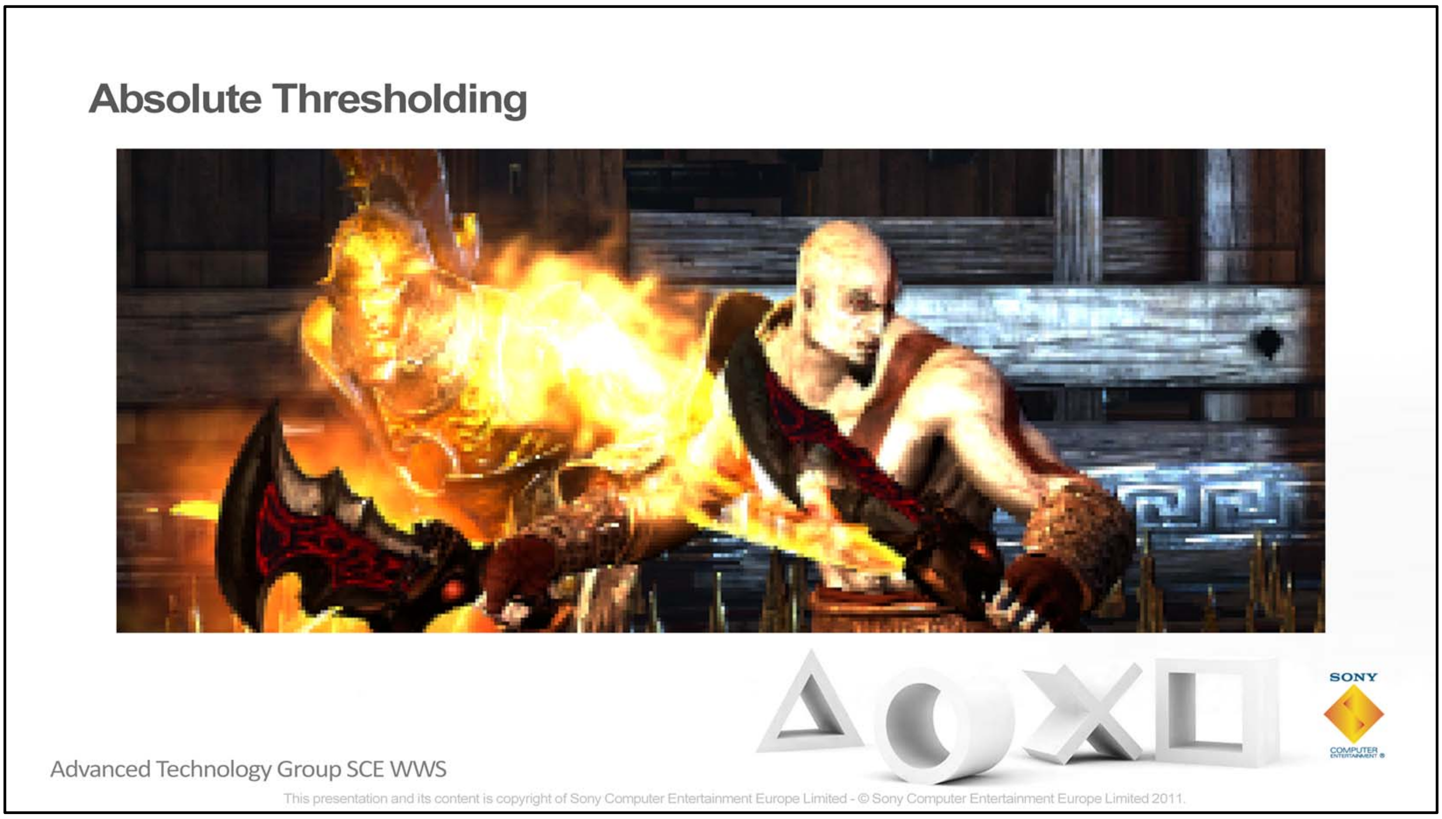
This was the only real change we made for God of War 3 and the Edge MLAA code we released, so let’s move on to new things.

Source

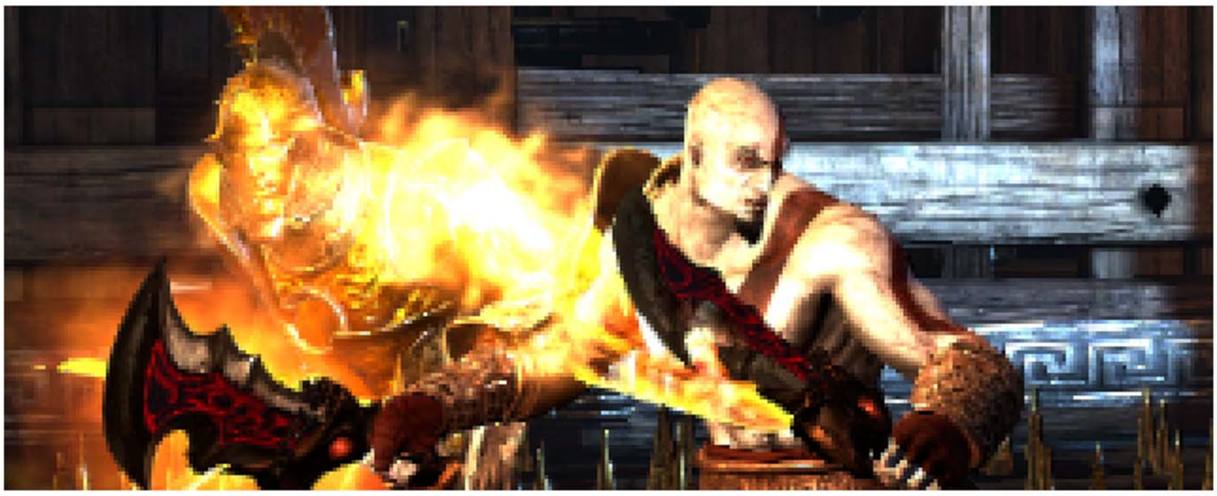


Advanced Technology Group SCE WWS

Here’s a slightly larger cutout from the spike room. Please keep an eye on Kratos’s face.

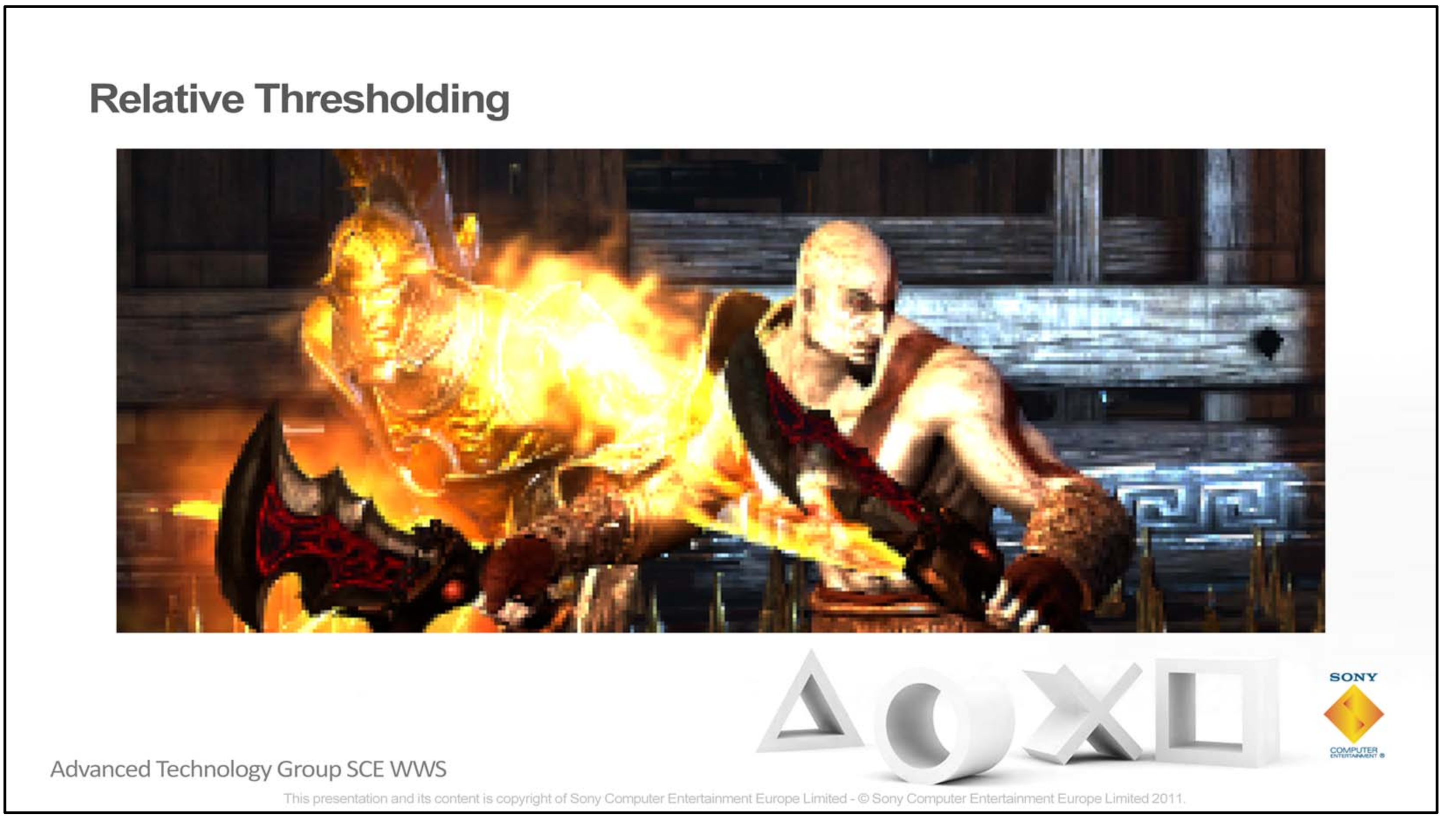


Absolute Thresholding

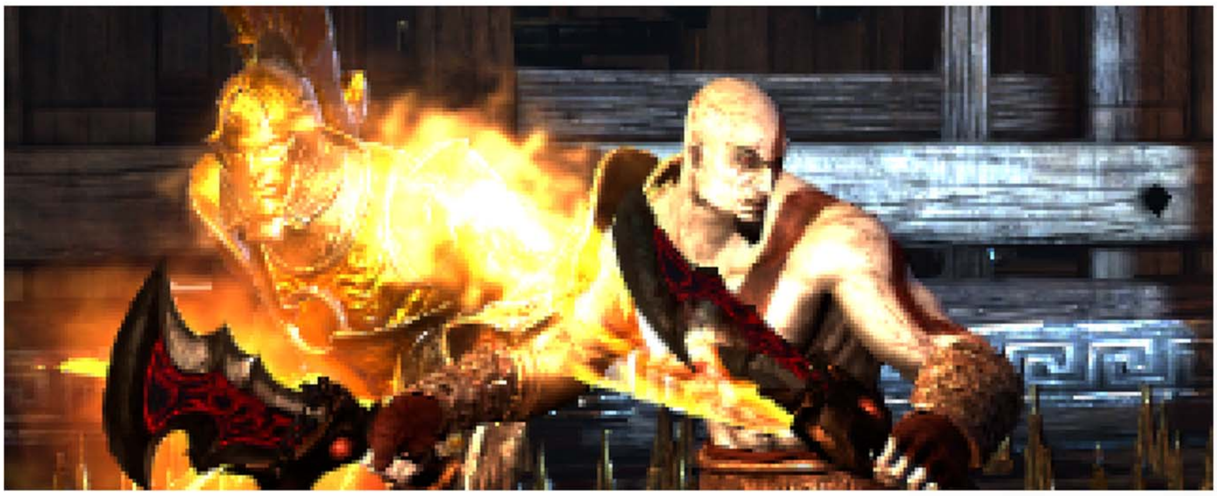


Advanced Technology Group SCE WWS

This is absolute thesholding again. You'll notice it's quite blurry.

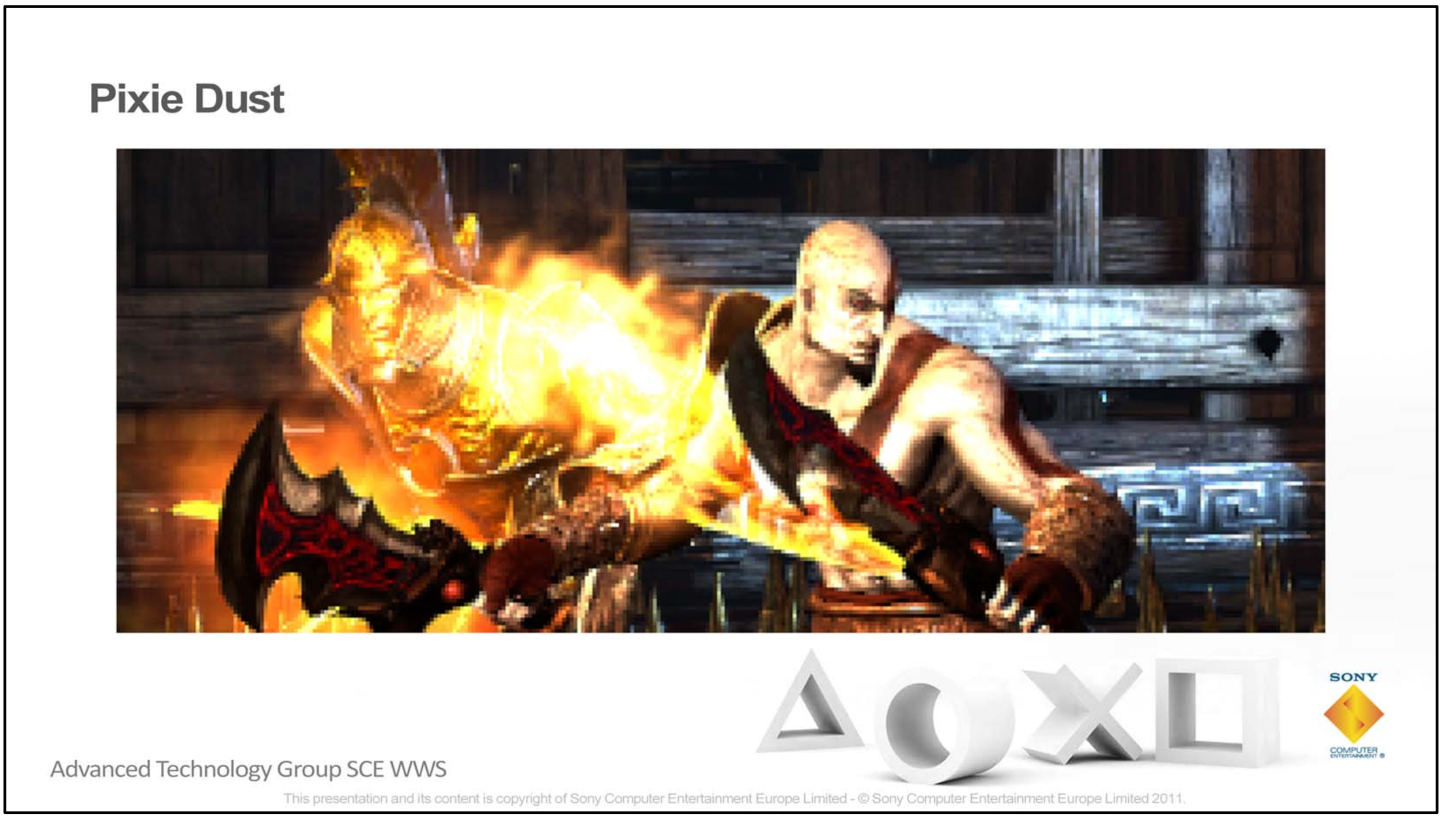


Relative Thresholding

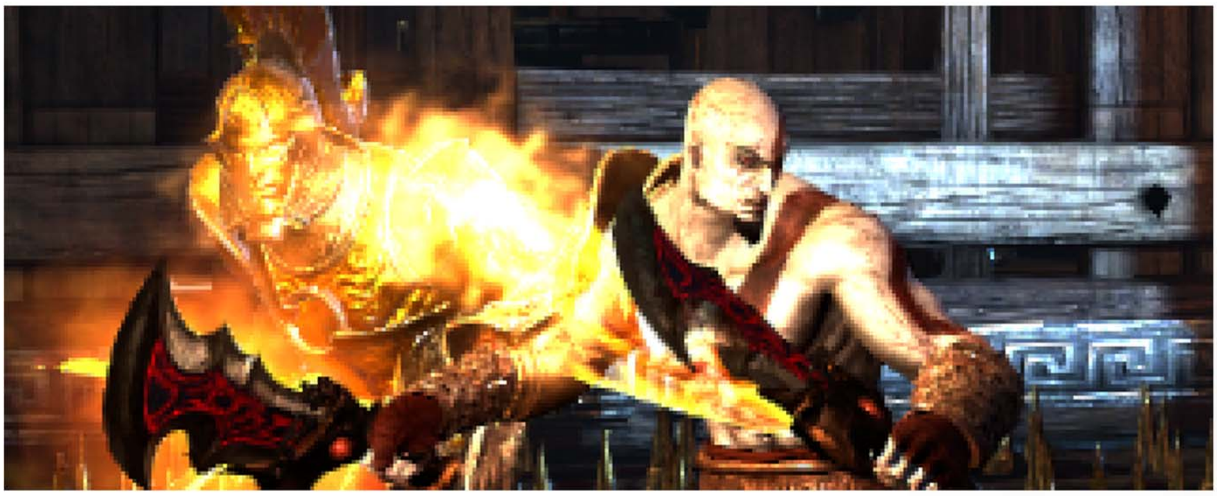


Advanced Technology Group SCE WWS

Relative Thresholding. Better, but with it's own artifacts.

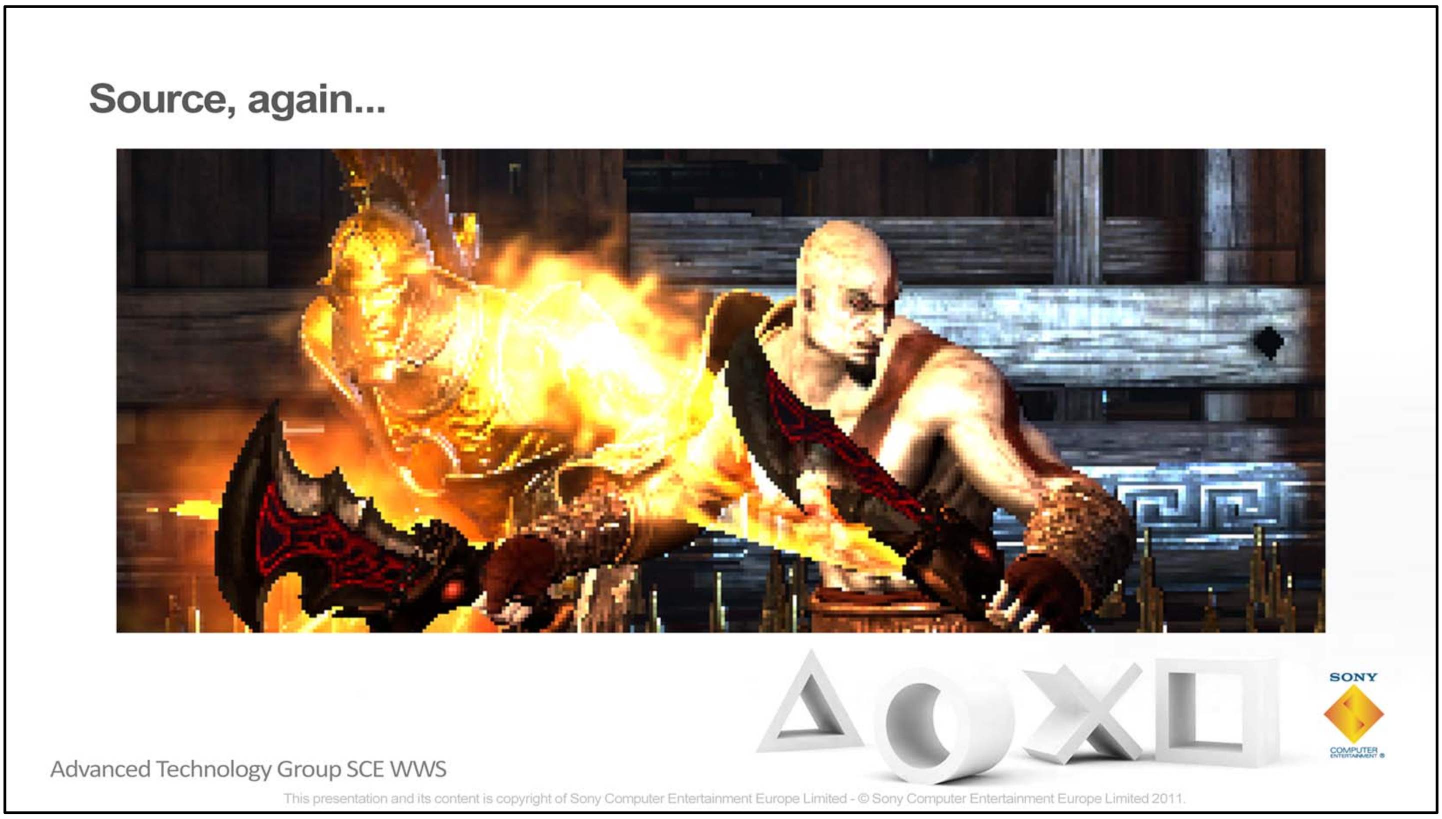


Pixie Dust

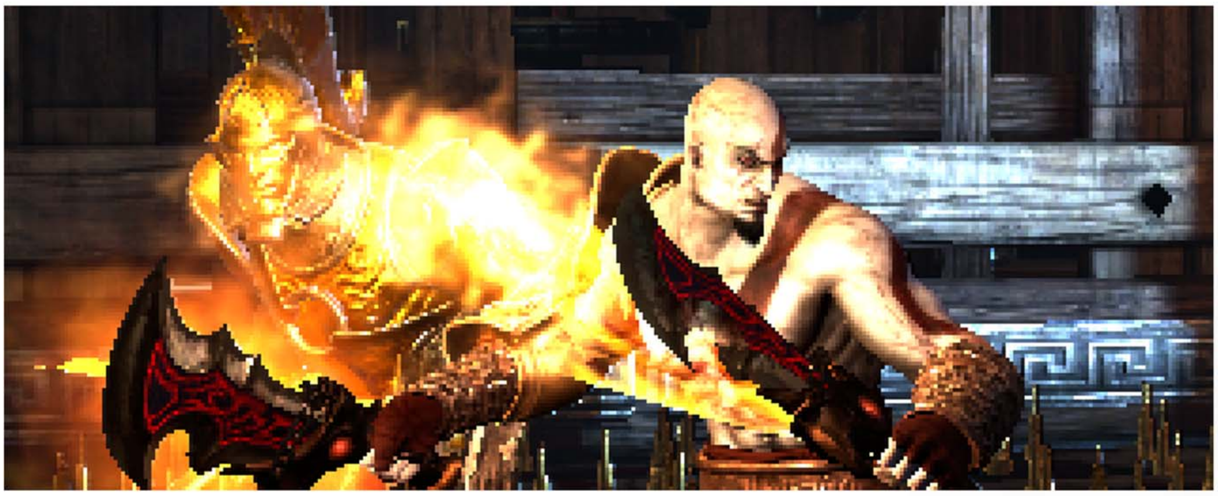


Advanced Technology Group SCE WWS

And our next improvement. The blade in front of Kratos's chest probably benefits the most, but the red detail textures gain some sharpness and we can again see Kratos's fashinable eye-liner.

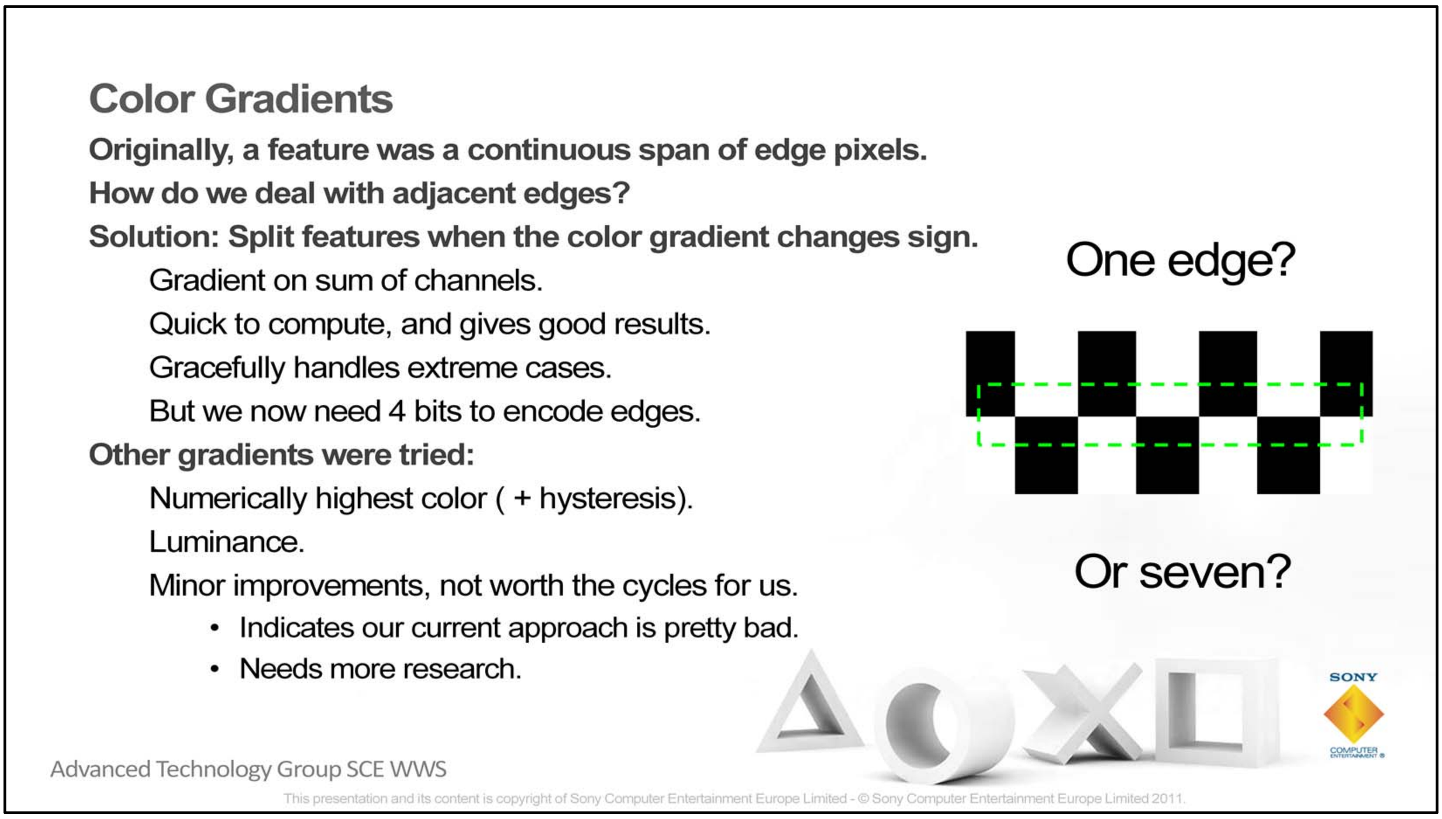


Source, again...



Advanced Technology Group SCE WWS

Again, the source for comparison.
So. How did we do that?



Color Gradients

Originally, a feature was a continuous span of edge pixels.

How do we deal with adjacent edges?

Solution: Split features when the color gradient changes sign.

- Gradient on sum of channels.
- Quick to compute, and gives good results.
- Gracefully handles extreme cases.
- But we now need 4 bits to encode edges.

Other gradients were tried:

- Numerically highest color (+ hysteresis).
- Luminance.
- Minor improvements, not worth the cycles for us.
  - Indicates our current approach is pretty bad.
  - Needs more research.

The way it is usually implemented, MLAA has no concept of adjacent edges.

[click]

If one edge begins right when another one ends, those two are considered to be one edge.

[click]

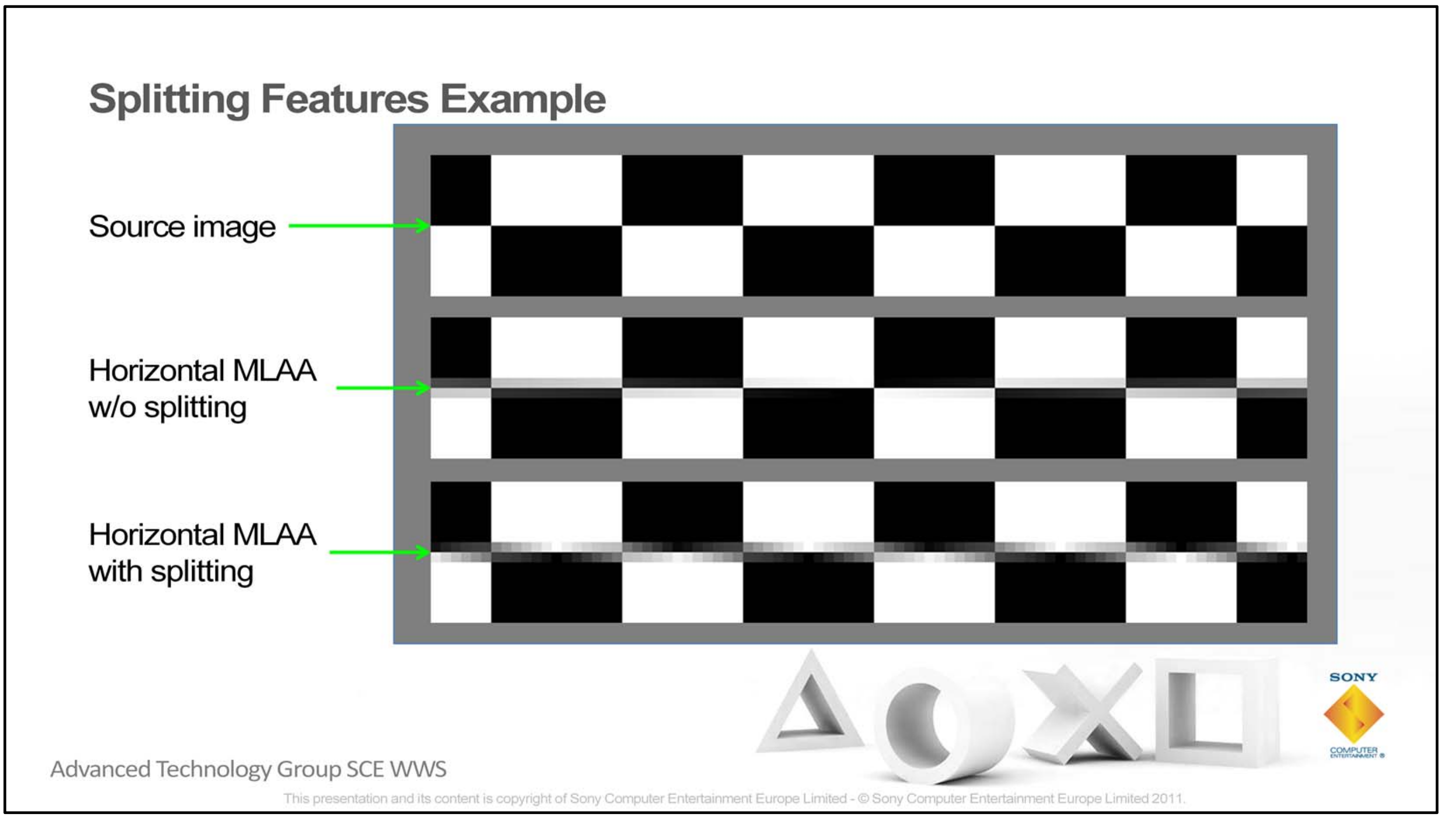
A start and an end-point are then determined for this fused edge and blending is performed over the entire structure. Clearly, this is not ideal.

[click]

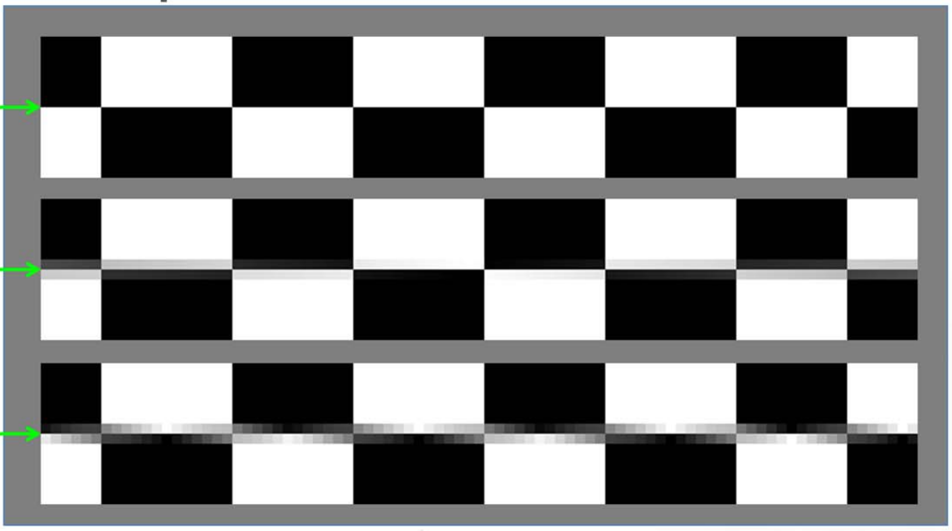
Our solution is to split an edge if the color gradient sign changes, which we approximate by looking at the sum of color channels.

[click]

I’ve tried a handful of other approaches to determine this sign change, but none of them really improved the result over this very simple and fast approach. I take that to mean they were all equally bad, so I’m sure some smart person will find a much better way to do this.



Splitting Features Example



Advanced Technology Group SCE WWS

To illustrate the effect, I'd like to show you the checkerboard pattern again.

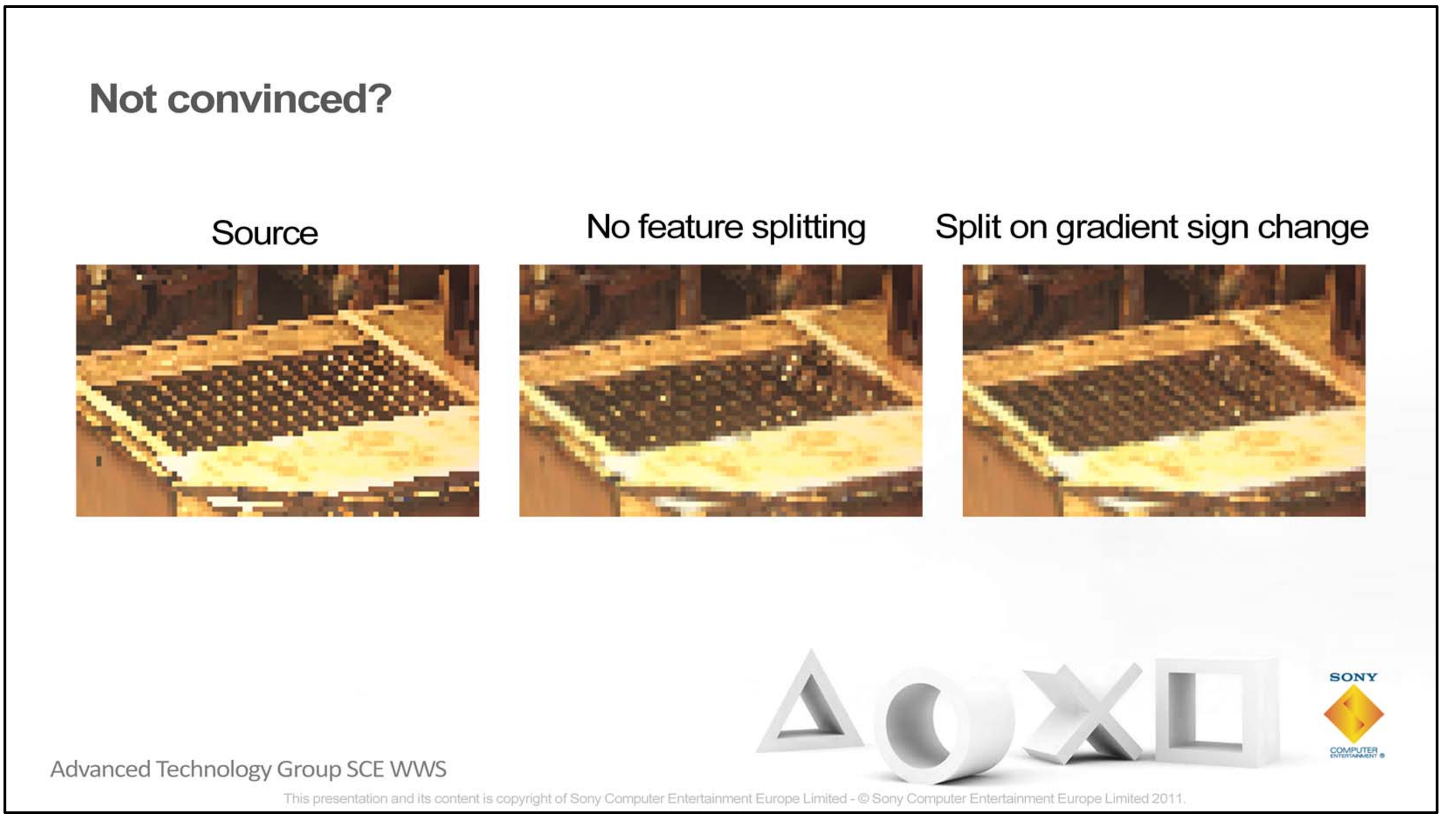
[click]

Performing just a normal horizontal MLAA doesn't really get us any anti-aliasing.

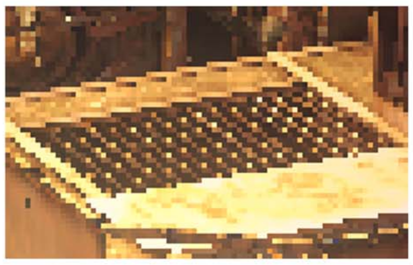
[click]

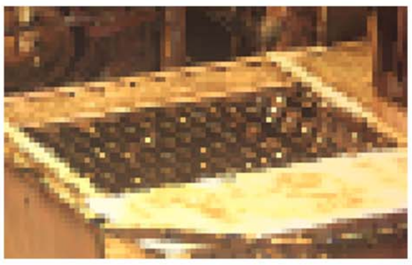
Only when we split the composite edge, do we get the desired result.

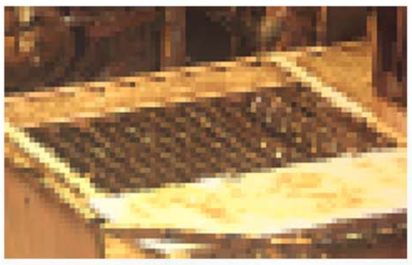
I've already shown you the improvement this made for the spike room, but maybe you found that a bit underwhelming, so let me show you why we did this.



Not convinced?


Source


No feature splitting


Split on gradient sign change

Advanced Technology Group SCE WWS

This is a seriously undersampled metal grill from Killzone 3.

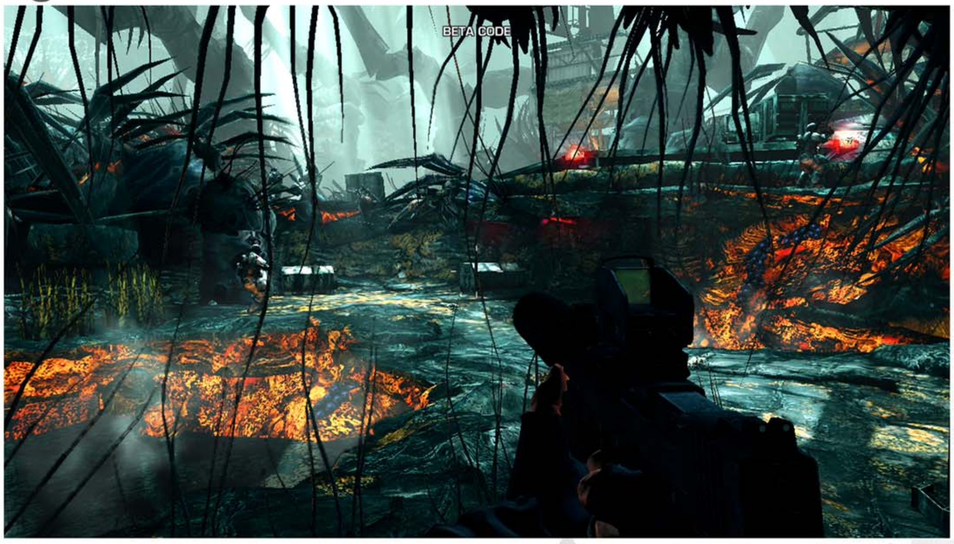
[click]

With the previous method, we'd lose most of the orientation and simply get, well, noisy stuff.

[click]

Once we split the features, we get a much nicer result. Notice that MLAA actually works pretty well against specular aliasing here, which is related. The reason is that for really short features, MLAA approximates a blur. As we split features aggressively, we get these short features here, which is a much more stable way to deal with hard to filter inputs.

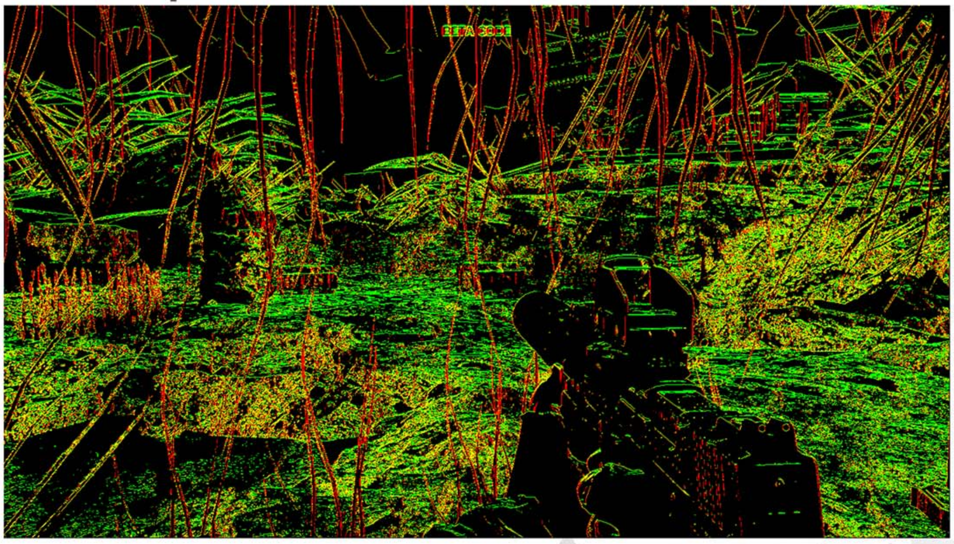
Preserving Texture Detail



OK, one more. Some time after God of War, Killzone 3 was getting ready to ship. They had been using our MLAA code for a while, but when I showed them the feature splitting I've just shown you, a lot of last minute activity followed.
The reason they were interested in getting some more work done is that the game has very high-frequency, high-contrast textures. Just look at the red/black details in those pits. God of War has a very different art style, this was much less of an issue.

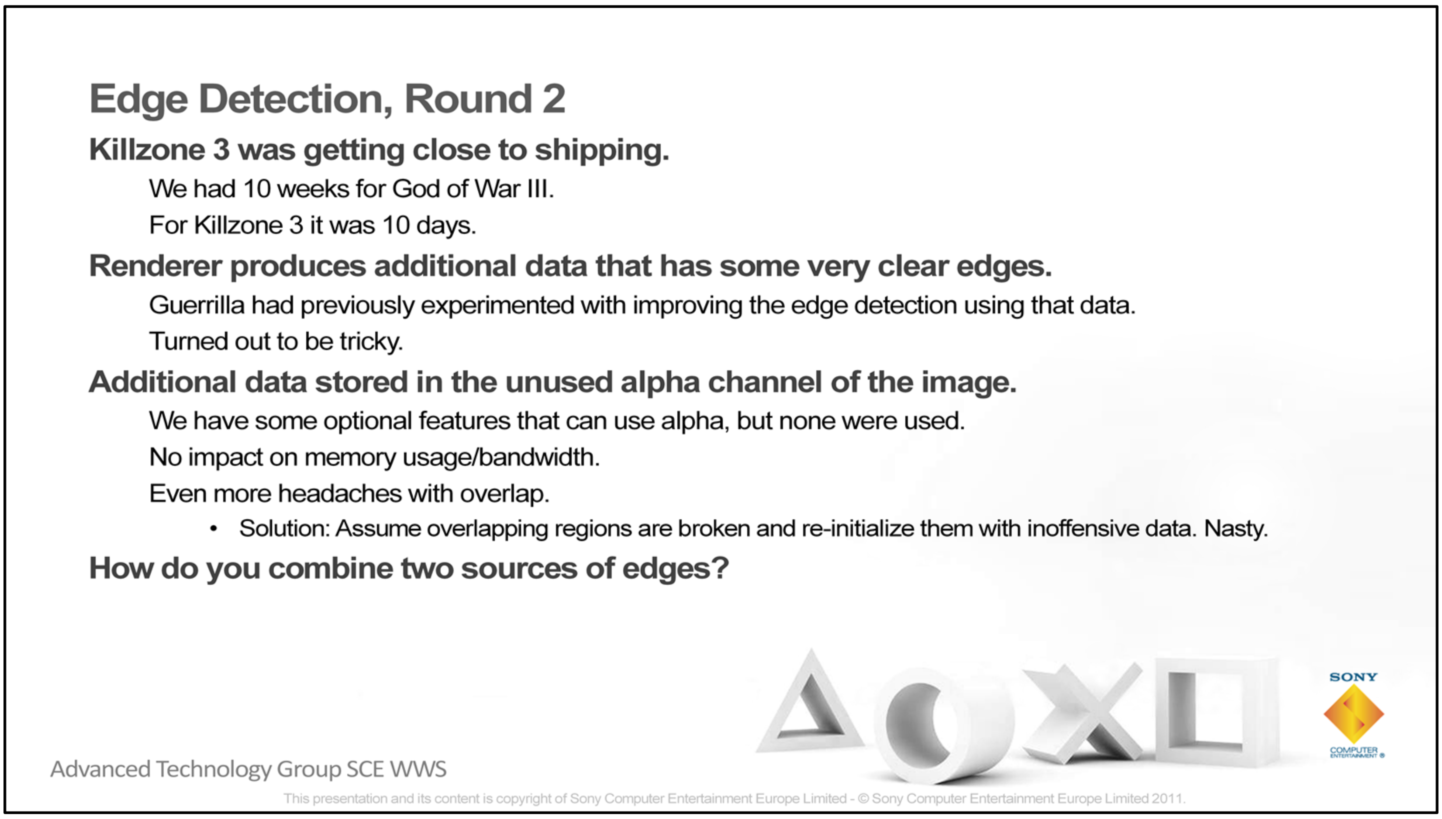
This frame, however, is not handled sufficiently well by relative thresholding.

Edges, fist attempt.



Advanced Technology Group SCE WWS

...as you can see here.



Edge Detection, Round 2

Killzone 3 was getting close to shipping.

- We had 10 weeks for God of War III.
- For Killzone 3 it was 10 days.

Renderer produces additional data that has some very clear edges.

- Guerrilla had previously experimented with improving the edge detection using that data.
- Turned out to be tricky.

Additional data stored in the unused alpha channel of the image.

- We have some optional features that can use alpha, but none were used.
- No impact on memory usage/bandwidth.
- Even more headaches with overlap.
  - Solution: Assume overlapping regions are broken and re-initialize them with inoffensive data. Nasty.

How do you combine two sources of edges?

Advanced Technology Group SCE WWS

So the current result wasn't really what we wanted and we had to revisit edge detection.

[click]

The Killzone guys had been experimenting with feeding additional non-color data into the edge detection before, but we decided to give it another shot literally 10 days before going gold.

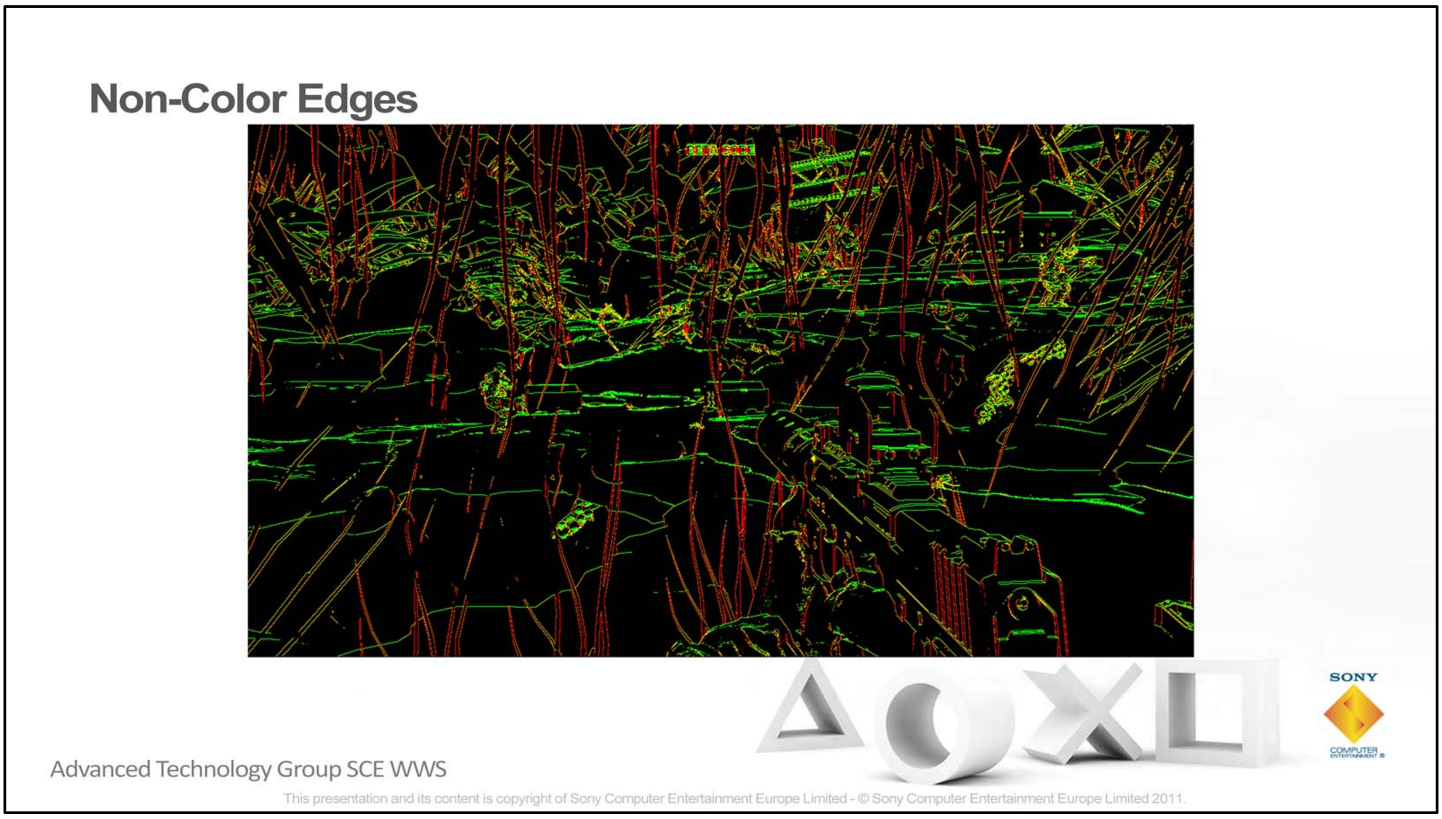
As they are using a deferred renderer, there are all kinds of buffers lying around that could be used for edge detection.

[click]

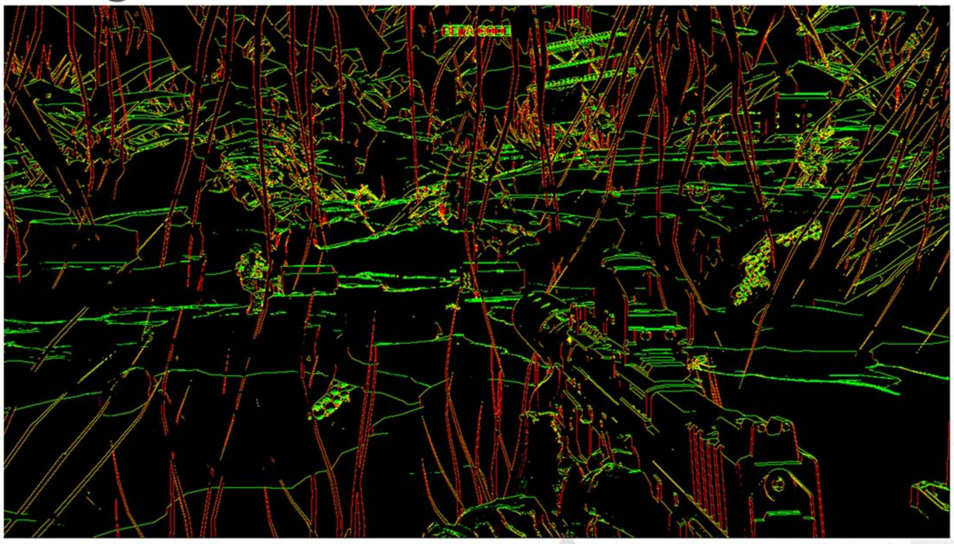
They ended up feeding part of the lighting computation into the alpha channel, where MLAA would pick it up. That way no additional storage or bandwidth was required.

[click]

The real question, however, is how to combine the two edge sources.



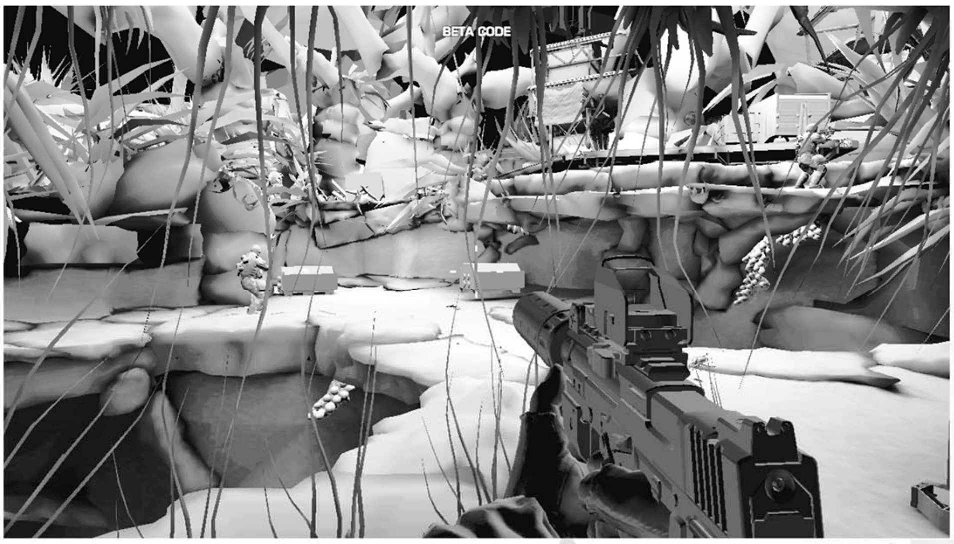
Non-Color Edges



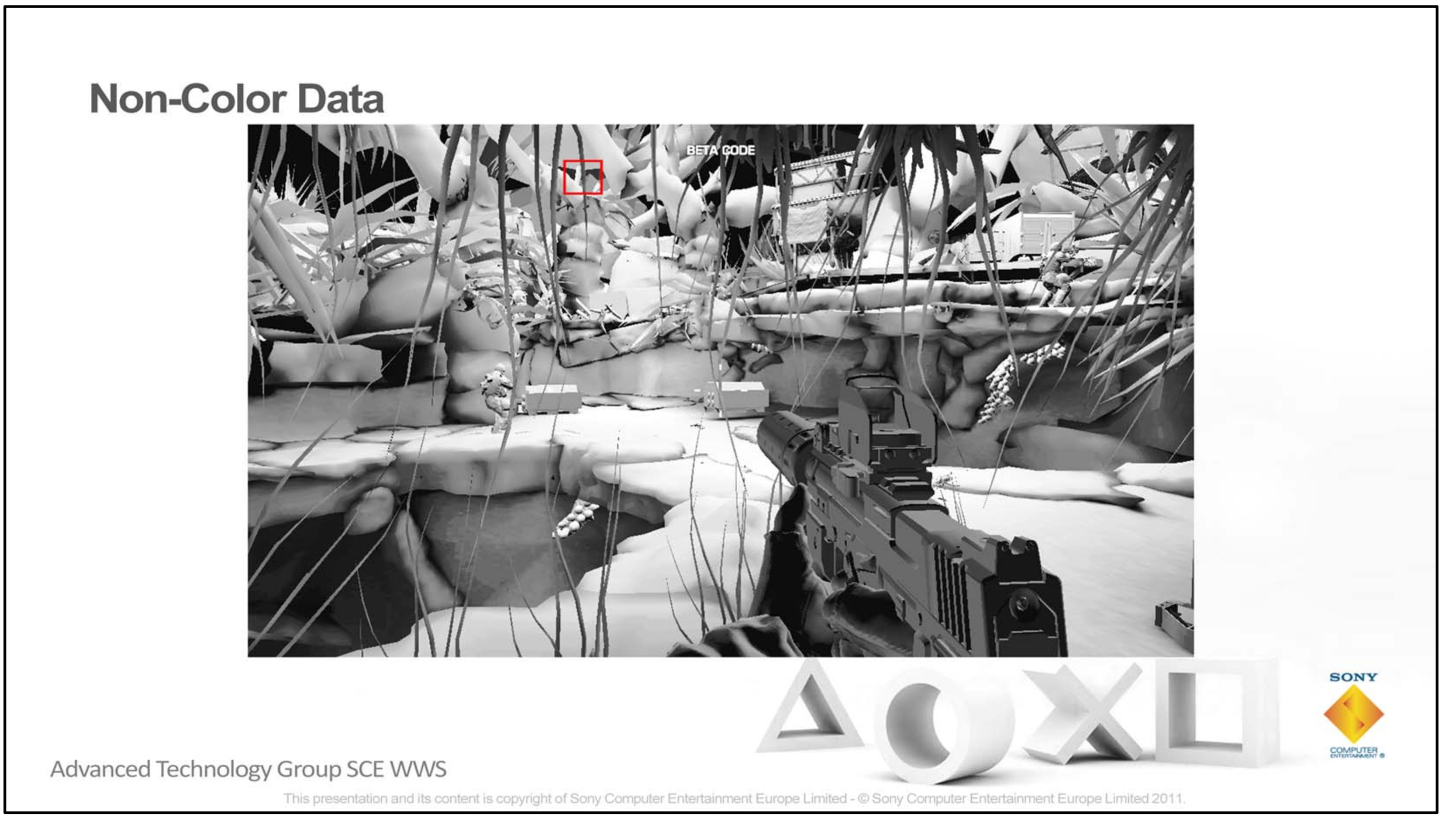
Advanced Technology Group SCE WWS

Here’s the edge buffer for just the lighting data, using relative thresholding to find the edges.

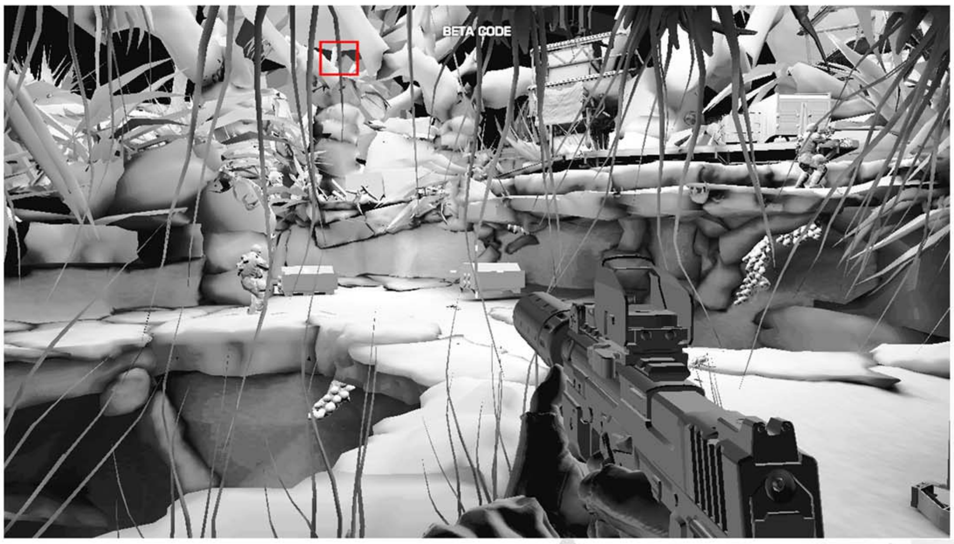
Non-Color Data



This is the actual input, somewhat contrast enhanced to improve visibility on the projector here.

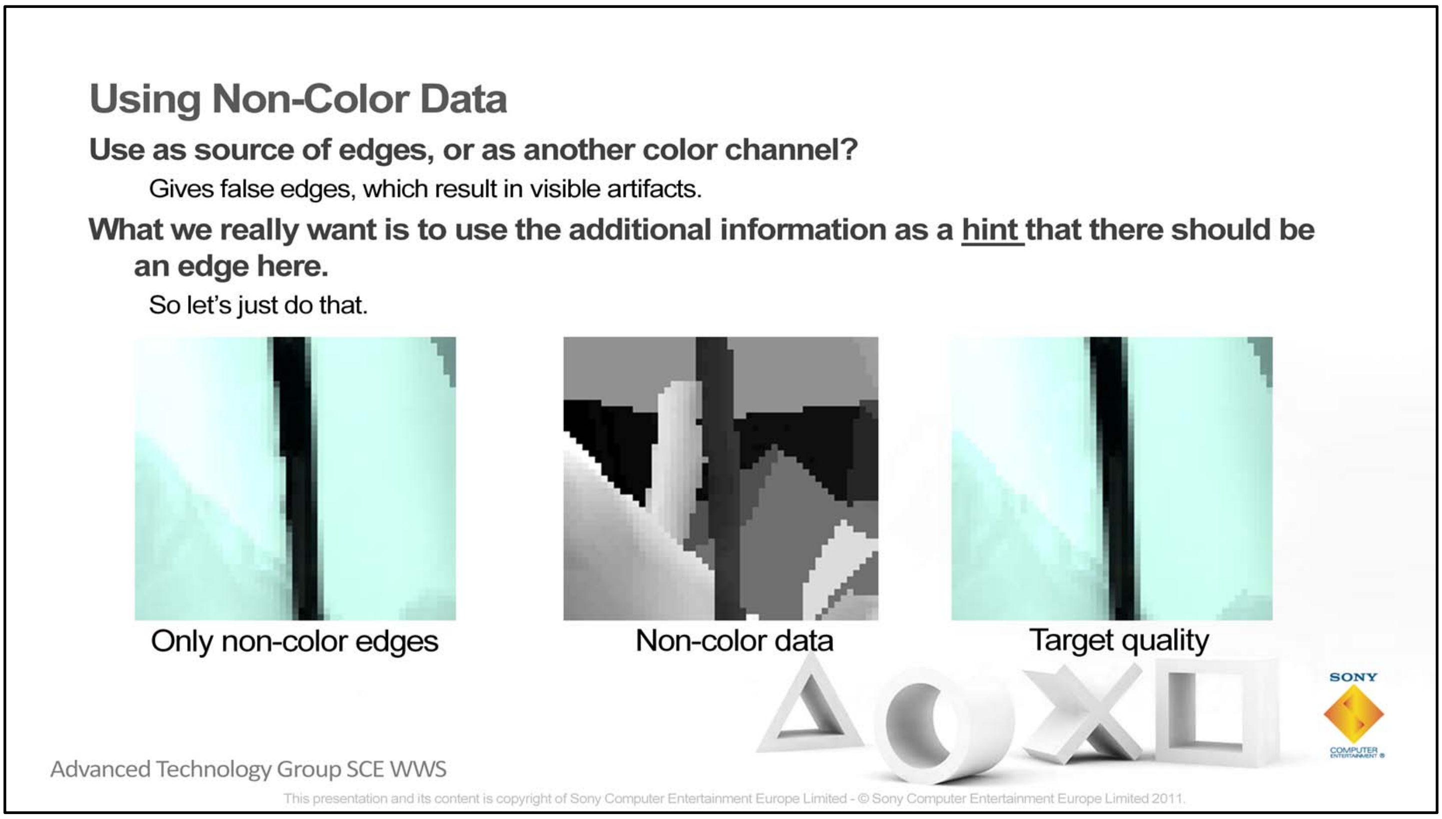


Non-Color Data



Advanced Technology Group SCE WWS

Up there is a cut-out I want to show.



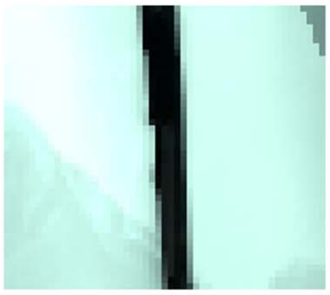
Using Non-Color Data

Use as source of edges, or as another color channel?

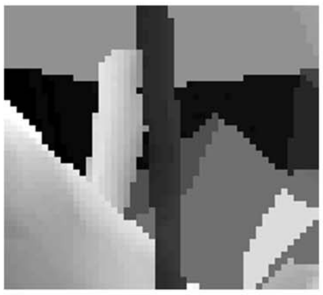
Gives false edges, which result in visible artifacts.

What we really want is to use the additional information as a hint that there should be an edge here.

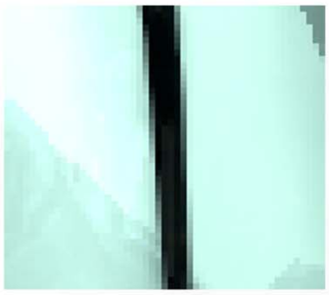
So let's just do that.



Only non-color edges



Non-color data



Target quality

Advanced Technology Group SCE WWS

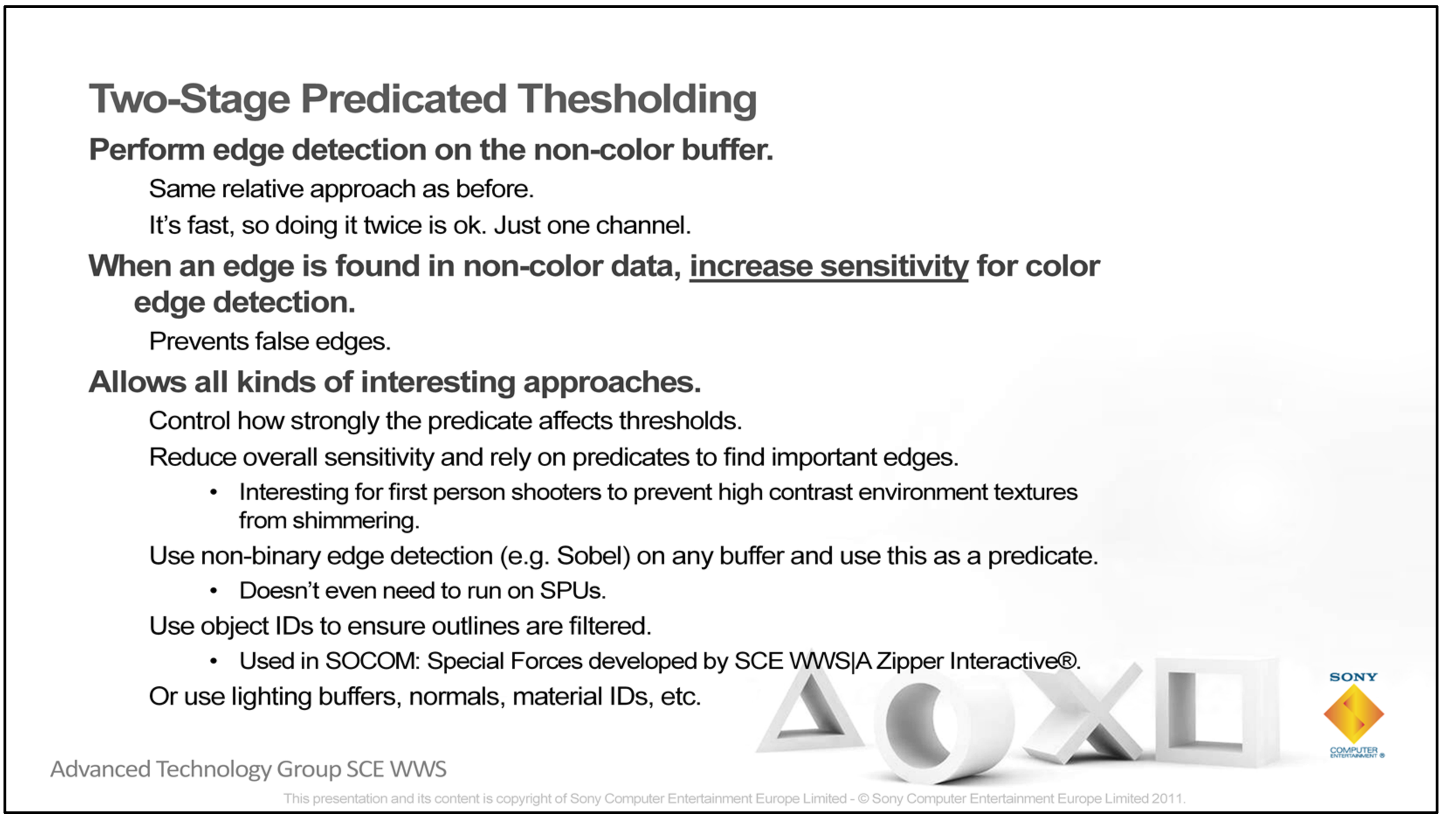
The first ideas centered around either dropping color edges altogether, or using the lighting as another "color" channel.

[click]

The problem with these approaches is demonstrated on the left. If an edge that does not exist in the color data intersects an edge that does exist in the color data, the algorithm will try to create a smooth transition between the two, effectively damaging the visible edge.

[click]

So we need to base the final decision if something is an edge on the actual color data. Which is what we did.



Two-Stage Predicated Thesholding

Perform edge detection on the non-color buffer.

- Same relative approach as before.
- It's fast, so doing it twice is ok. Just one channel.

When an edge is found in non-color data, increase sensitivity for color edge detection.

- Prevents false edges.

Allows all kinds of interesting approaches.

- Control how strongly the predicate affects thresholds.
- Reduce overall sensitivity and rely on predicates to find important edges.
  - Interesting for first person shooters to prevent high contrast environment textures from shimmering.
- Use non-binary edge detection (e.g. Sobel) on any buffer and use this as a predicate.
  - Doesn't even need to run on SPUs.
- Use object IDs to ensure outlines are filtered.
  - Used in SOCOM: Special Forces developed by SCE WWS|A Zipper Interactive®.
- Or use lighting buffers, normals, material IDs, etc.

Advanced Technology Group SCE WWS

Our solution was to perform two edge detection stages.

[click]

First, we process the non-color data. This gives us an edge mask which we feed into the second stage.

[click]

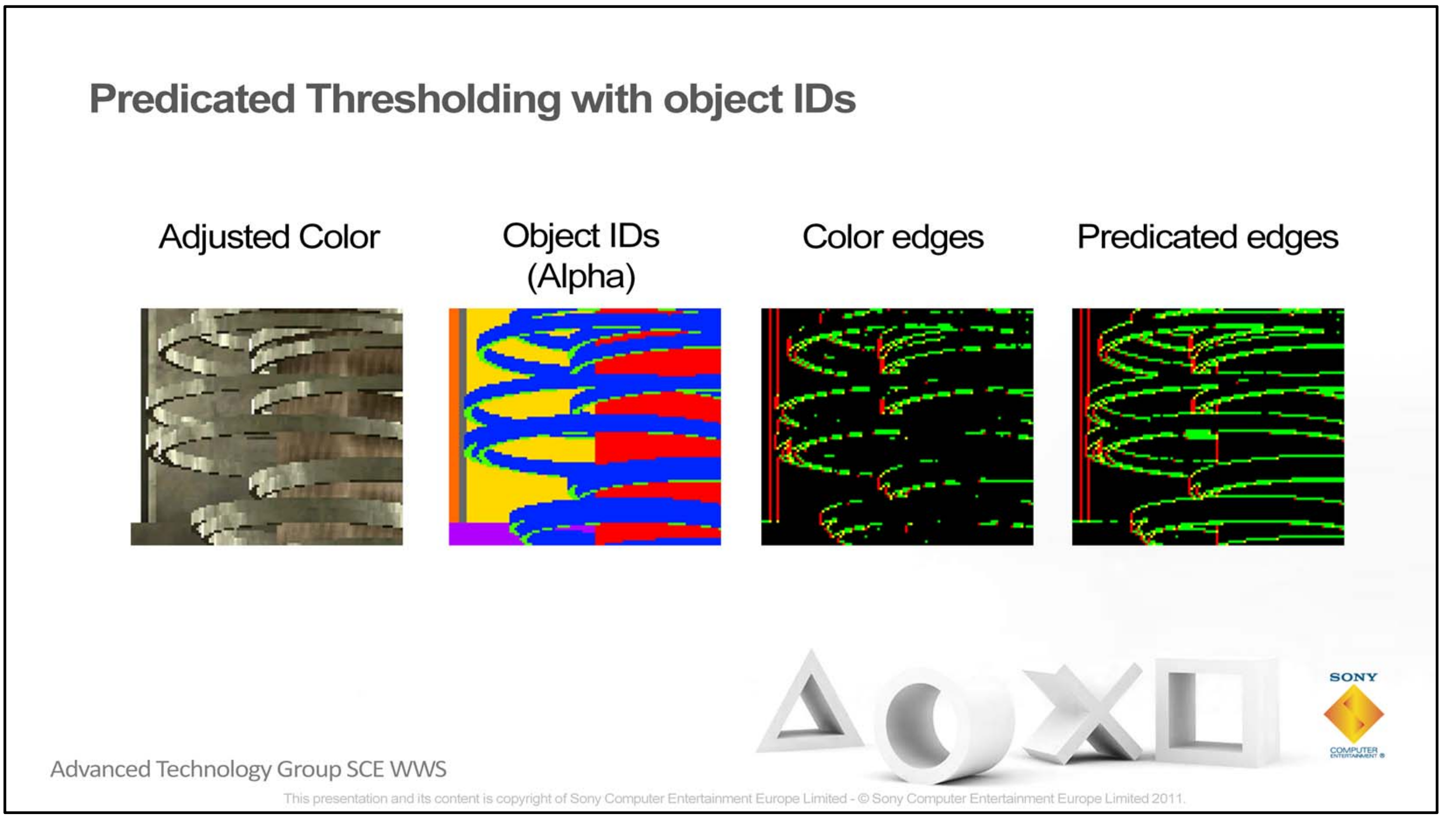
Here we look at the color data, but if we previously found a non-color edge, we increase the sensitivity. We know something's there so we look extra closely.

[click]

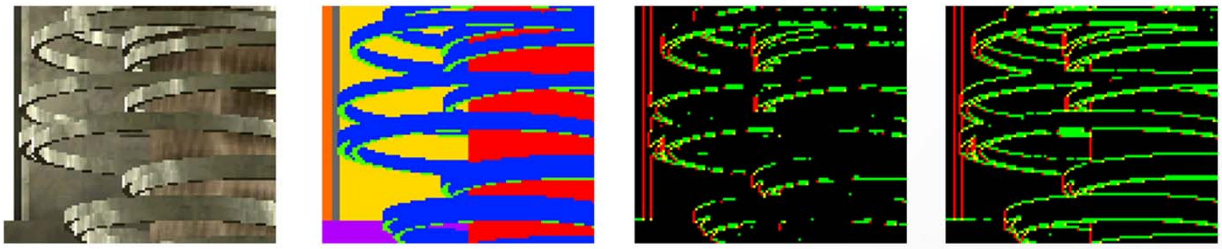
We could have gone for only looking for color edges where there are non-color edges, but that would have missed too many important features. Still, we can now tone down the general sensitivity and still get pretty much all edges we want.

[click]

The really cool thing about this, however, is that this first pass can be pretty much anything you want. You could run a Sobel, which doesn't even give the binary edges we need later. Or work on normals, or object Ids, which we used in SOCOM: Special Forces.



Predicated Thresholding with object IDs



Here’s an example from SOCOM, again contrast enhanced,

[click]

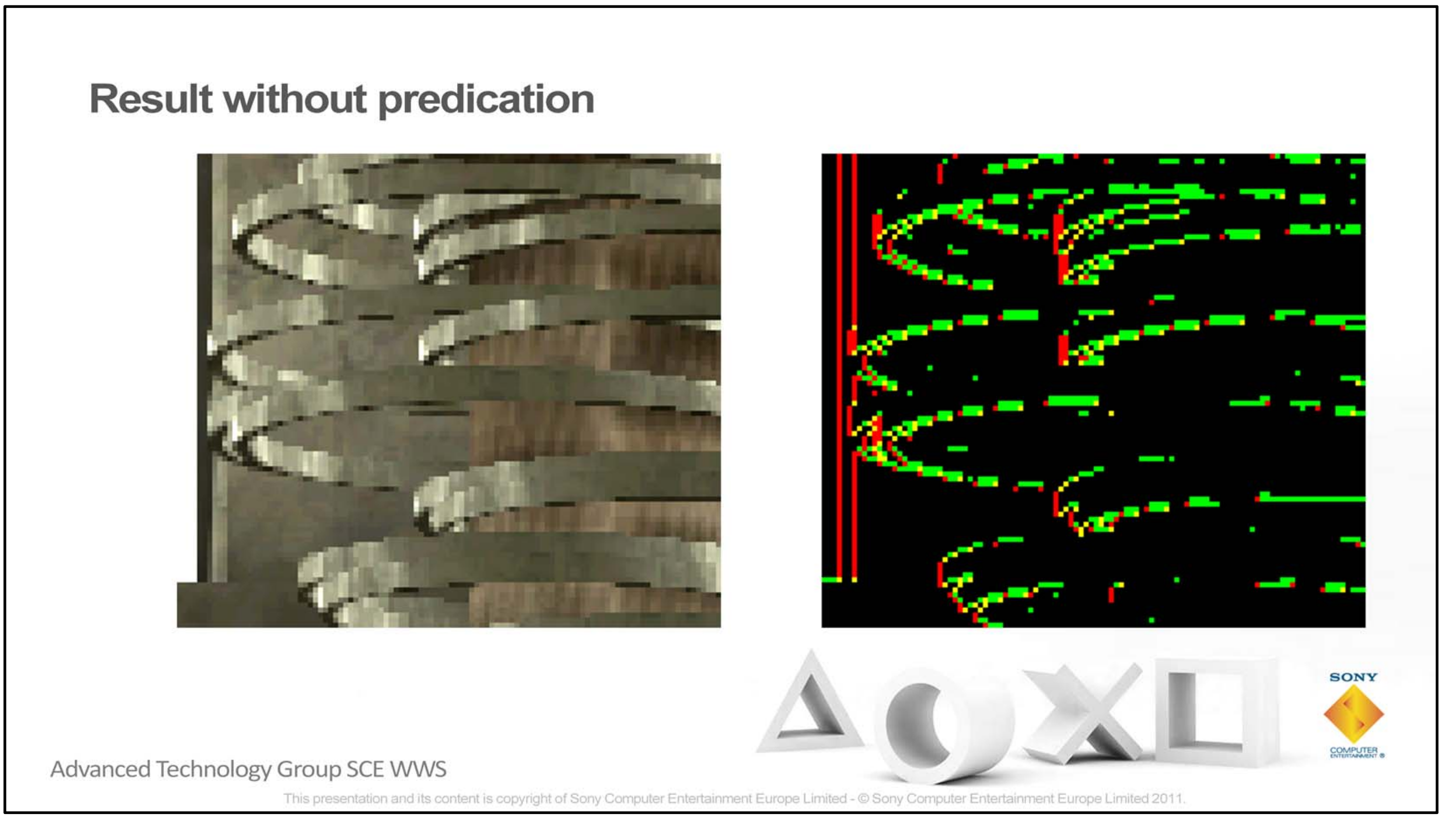
then the material Ids,

[click]

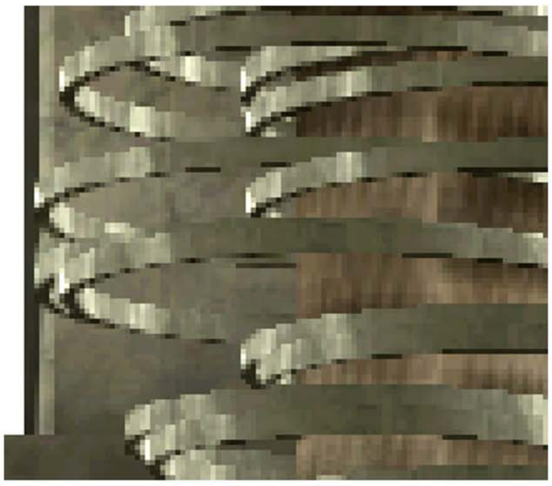
the color edges

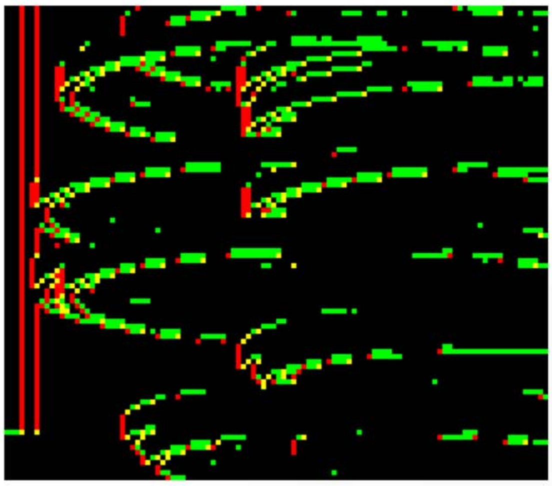
[click]

and the combined edges. Note that there are still holes in the combined edges, since the color values are extremely close or equal in the source image. Maybe we could patch these these with the non-color data.

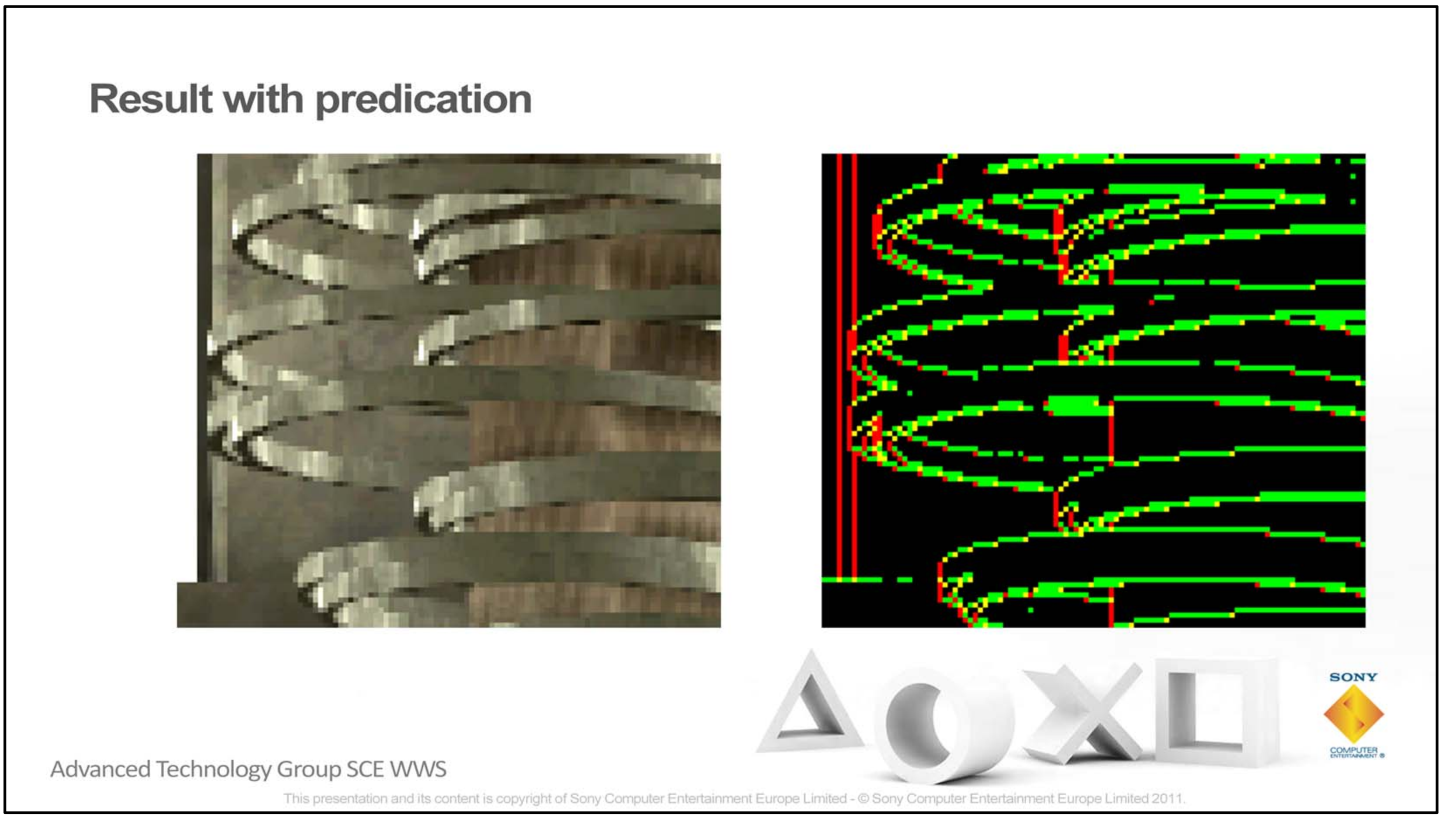


Result without predication

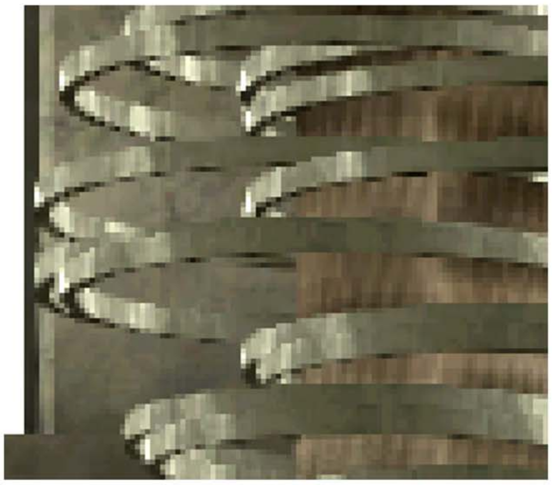


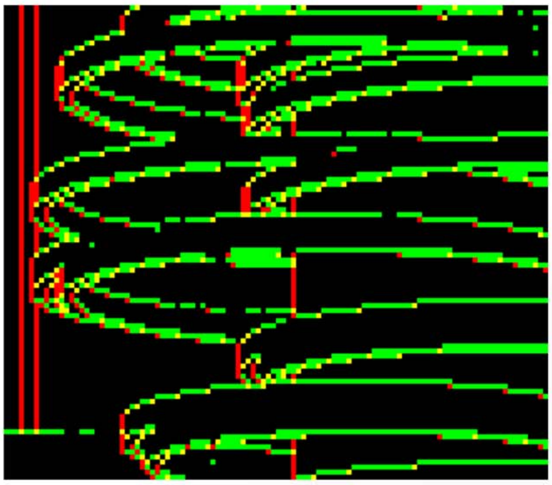


Here's the non-predicated result. Notice the unfiltered black lines.



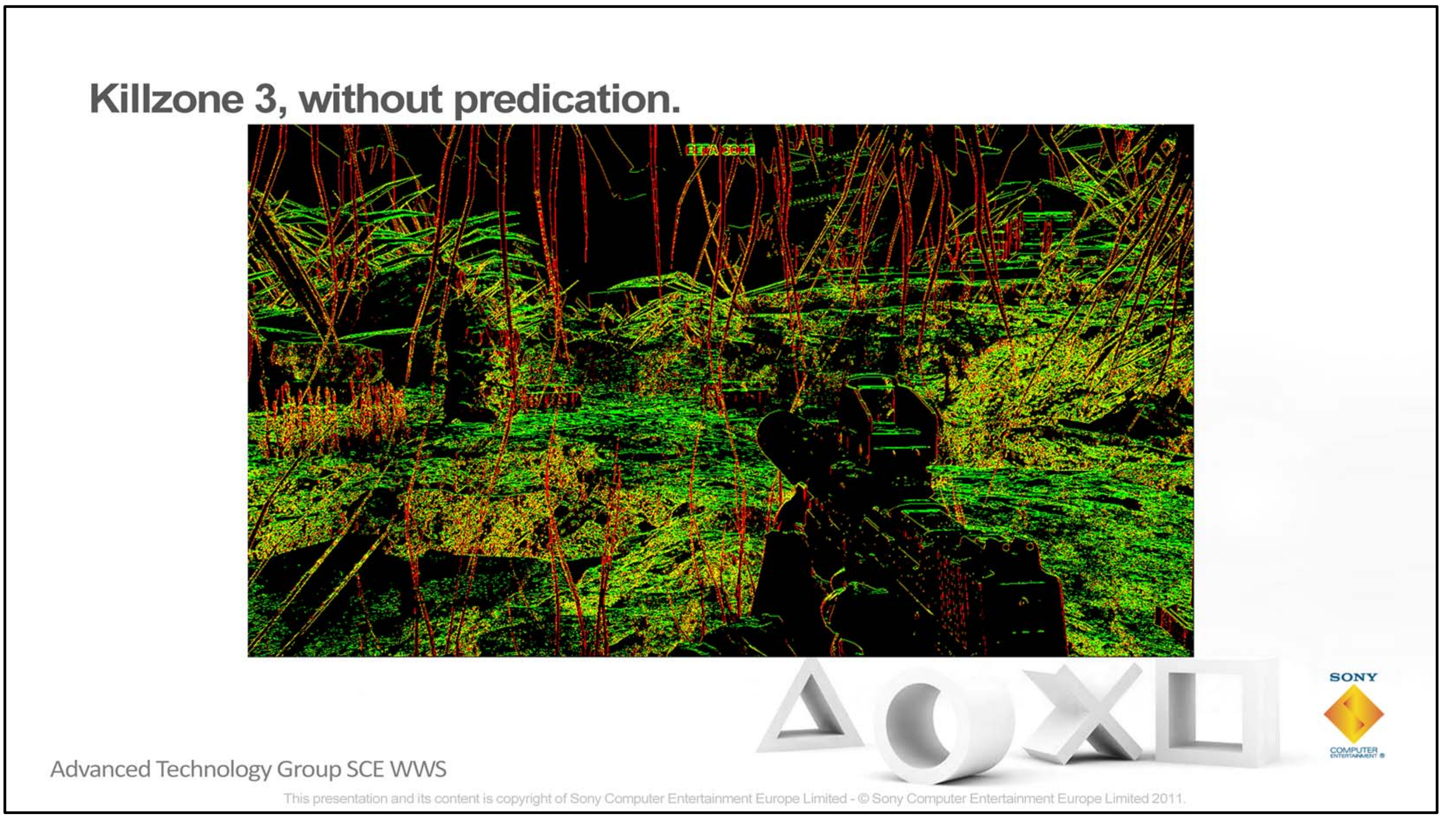
Result with predication



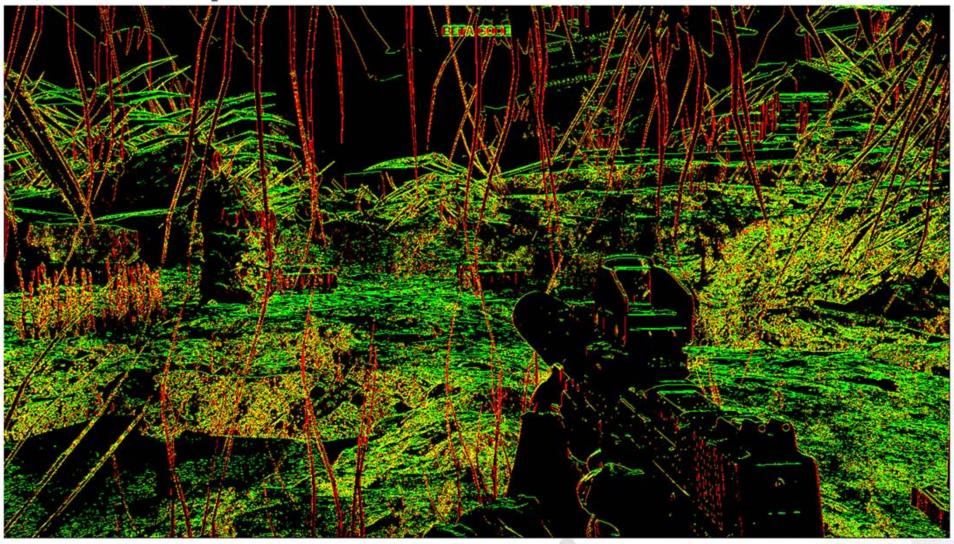


Advanced Technology Group SCE WWS

And the predicated result.

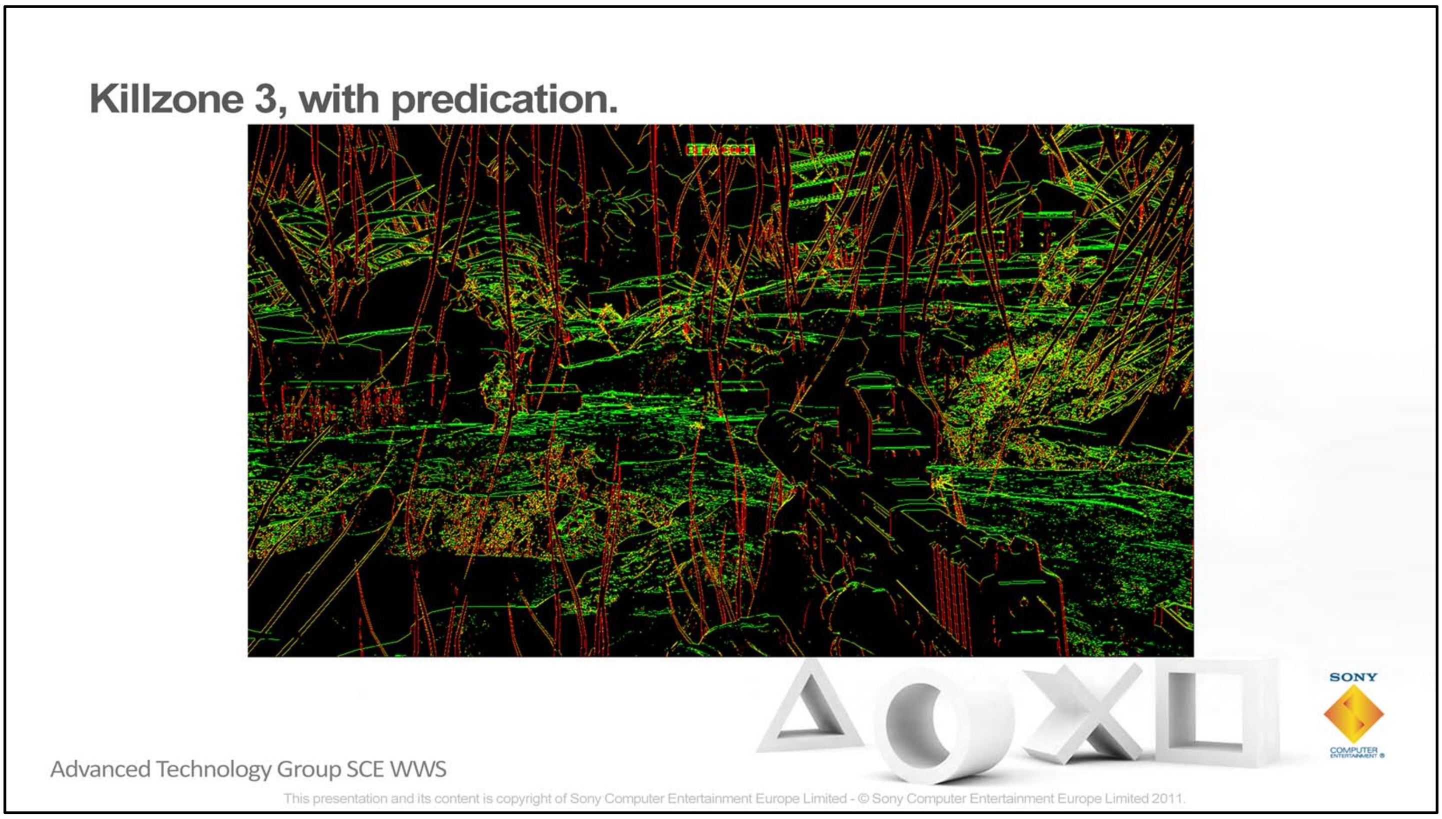


Killzone 3, without predication.

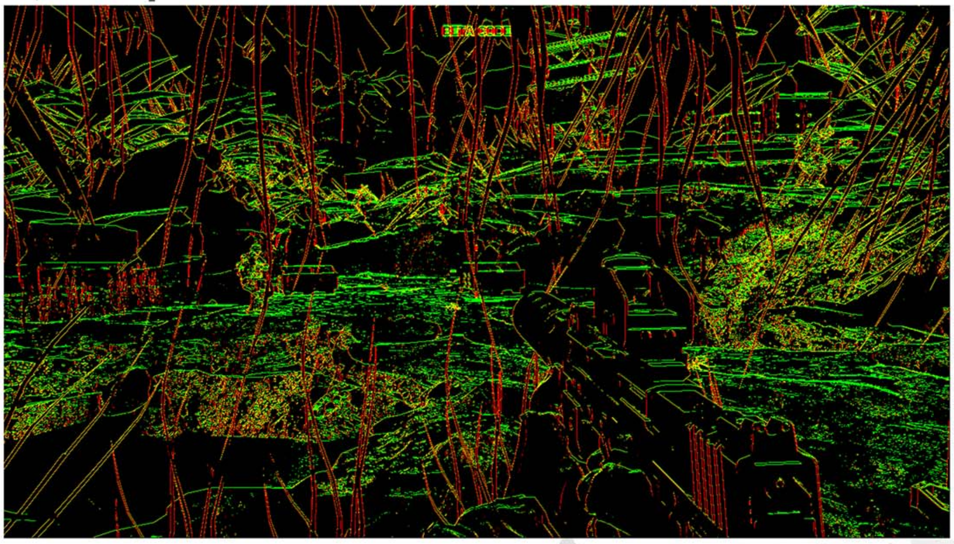


Advanced Technology Group SCE WWS

This is again Killzone 3 without predication.

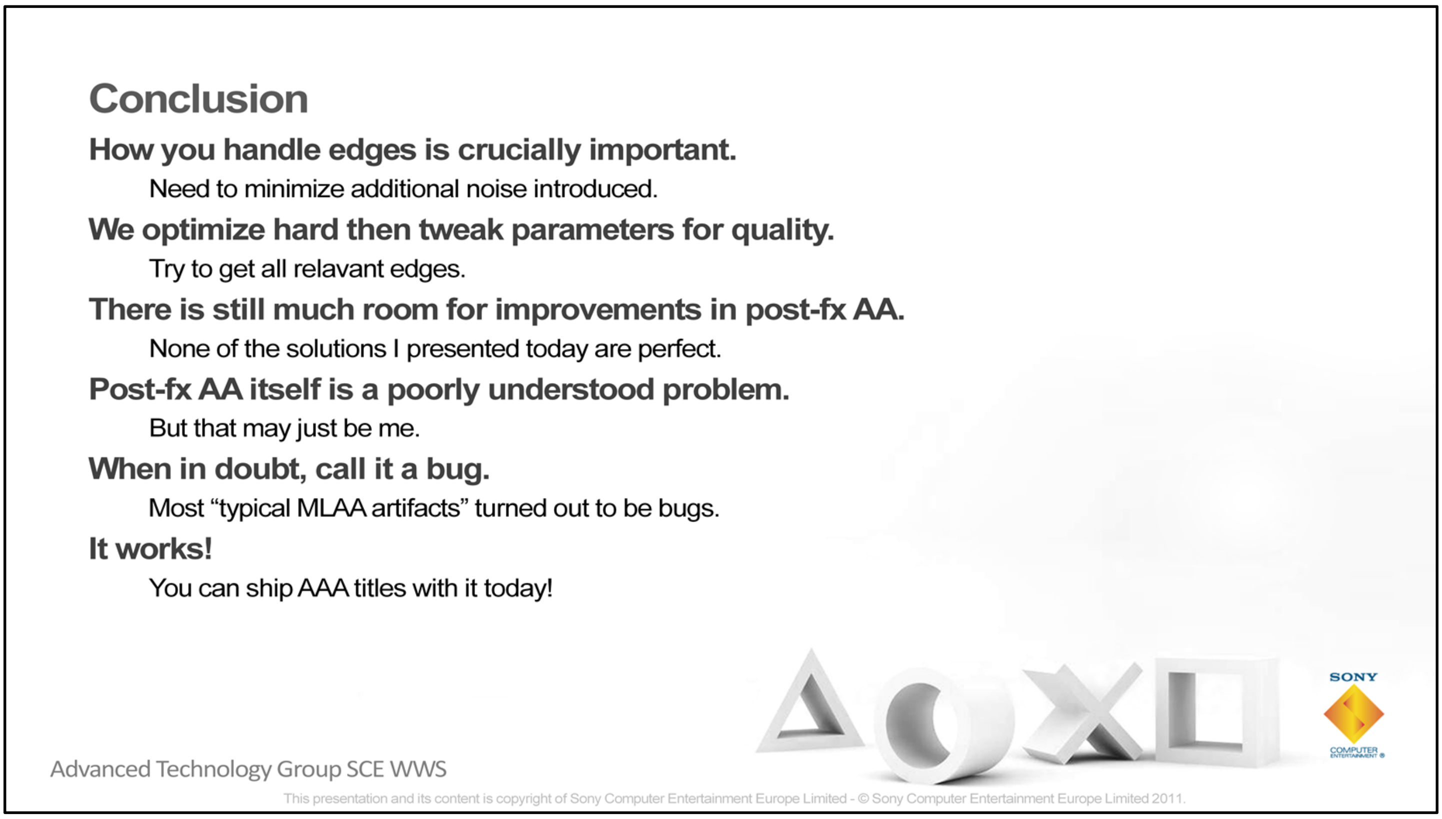


Killzone 3, with predication.



Advanced Technology Group SCE WWS

...and with predication. I didn’t perfectly tweak the parameters for the image here, but I think the improvement is pretty clear.



Conclusion

How you handle edges is crucially important.

Need to minimize additional noise introduced.

We optimize hard then tweak parameters for quality.

Try to get all relavant edges.

There is still much room for improvements in post-fx AA.

None of the solutions I presented today are perfect.

Post-fx AA itself is a poorly understood problem.

But that may just be me.

When in doubt, call it a bug.

Most "typical MLAA artifacts" turned out to be bugs.

It works!

You can ship AAA titles with it today!

Advanced Technology Group SCE WWS

Alright, let's wrap this up.

[click]
The message I wanted to bring across today is that how you find your edges and what you do with them is extremely important for the final result. AA systems are in the business of reducing high-frequency noise, so you want to add as little to the problem as possible.

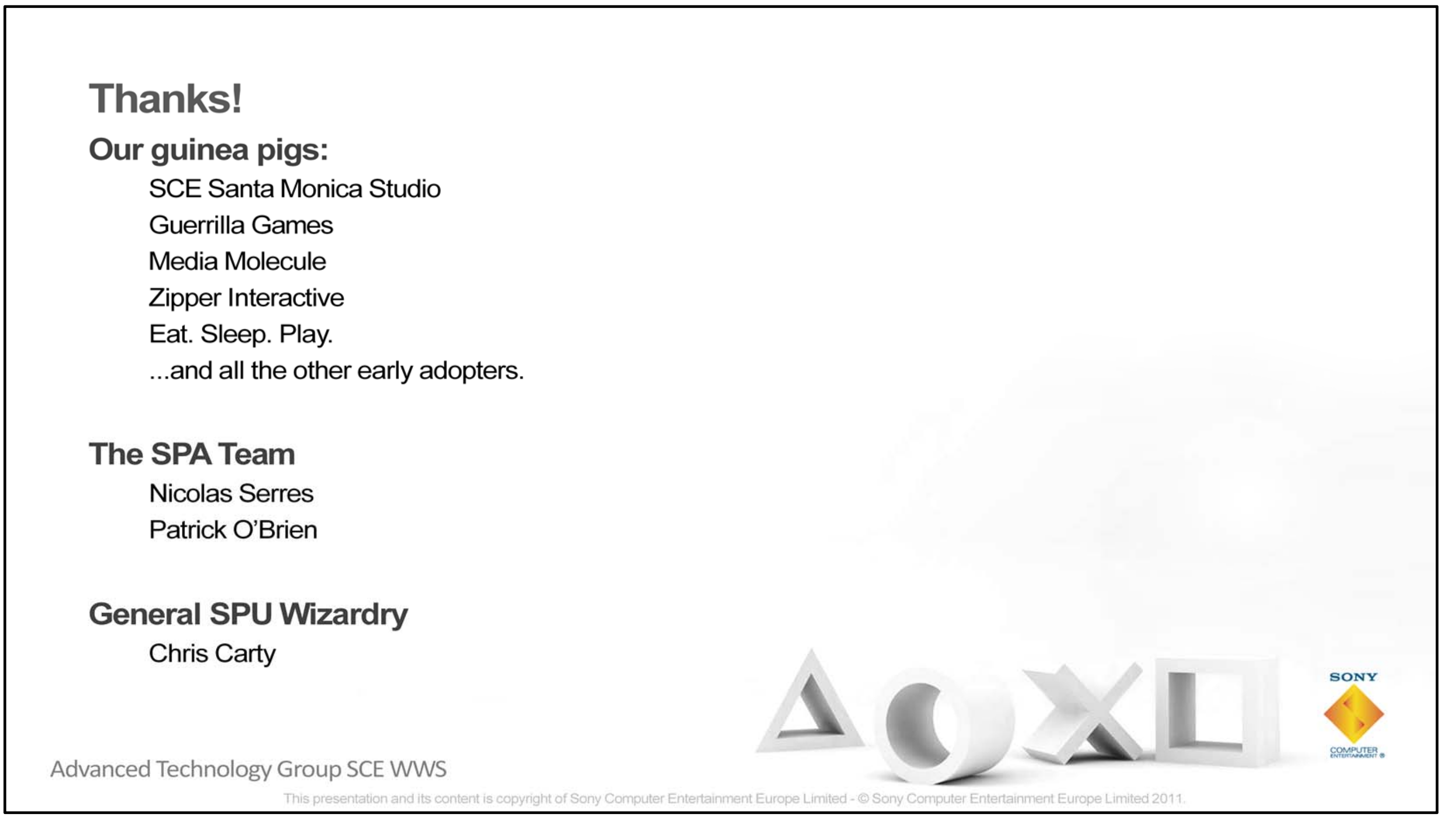
[click]
For us that means that we usually try to find the best quality solution within reason, and then optimize that till we meet the performance goals. We never changed parameters to compromise quality for speed.

[click]
So what I presented today is basically a small selection of hacks to make MLAA produce better results. None of these are without their own issues and I pointed out a few of them, so in my opinion, post-fx AA remains a very interesting field of research.

[click]
One reason for that is that it's not really a well understood problem, at this point. Other people may disagree with that, but I certainly don't know how to make the perfect post-fx AA system.

[click]
This also means that we tend to classify things as "typical MLAA artifacts" that are actually solvable problems or just bugs.

[click]
Still, at the end of the day, we can and do ship big AAA titles with it successfully, because it's a really smart idea, and I'm looking forward to seeing where it goes.



Thanks!

Our guinea pigs:

SCE Santa Monica Studio
Guerrilla Games
Media Molecule
Zipper Interactive
Eat. Sleep. Play.
...and all the other early adopters.

The SPA Team

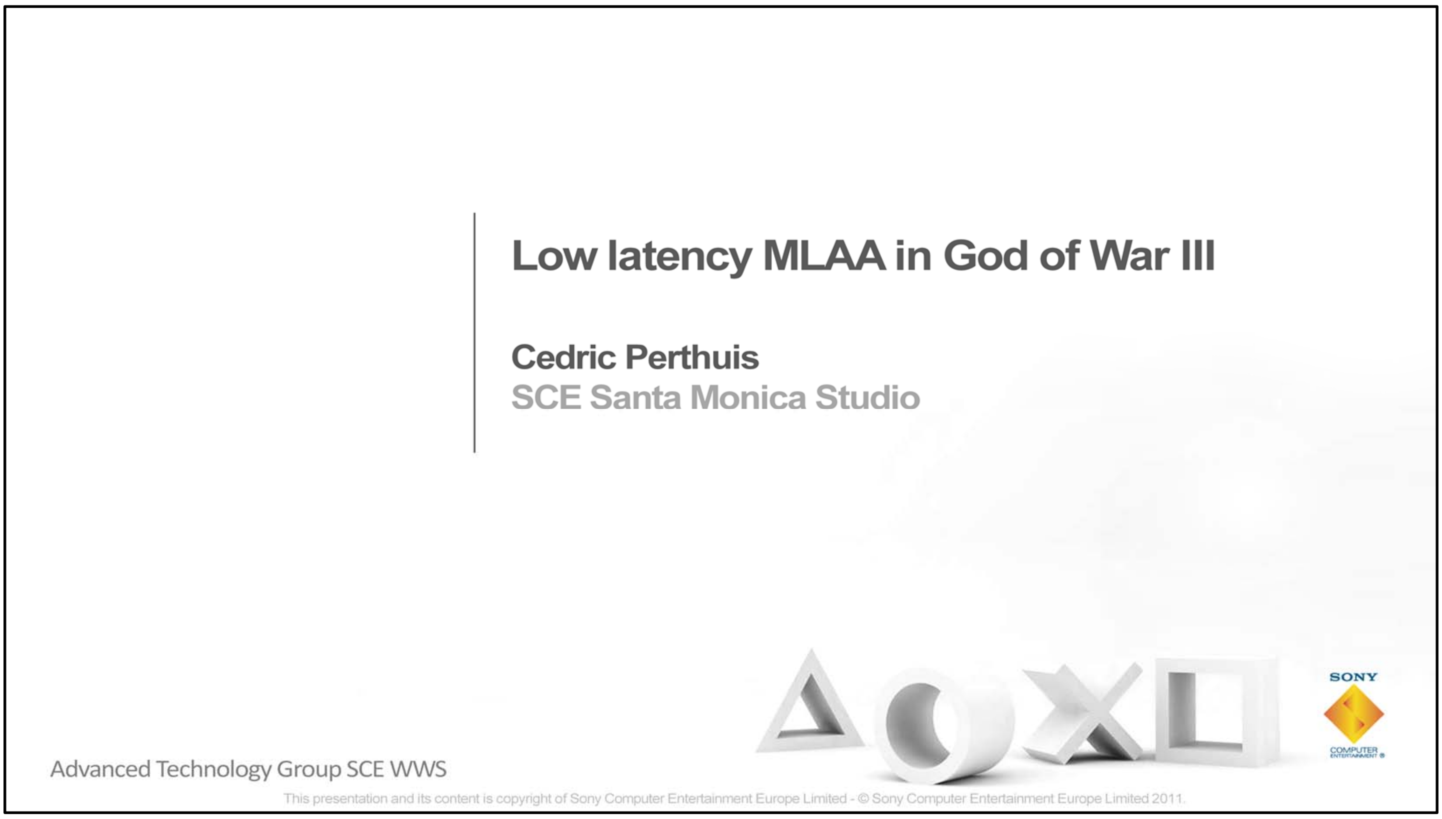
Nicolas Serres
Patrick O'Brien

General SPU Wizardry

Chris Carty

Advanced Technology Group SCE WWS

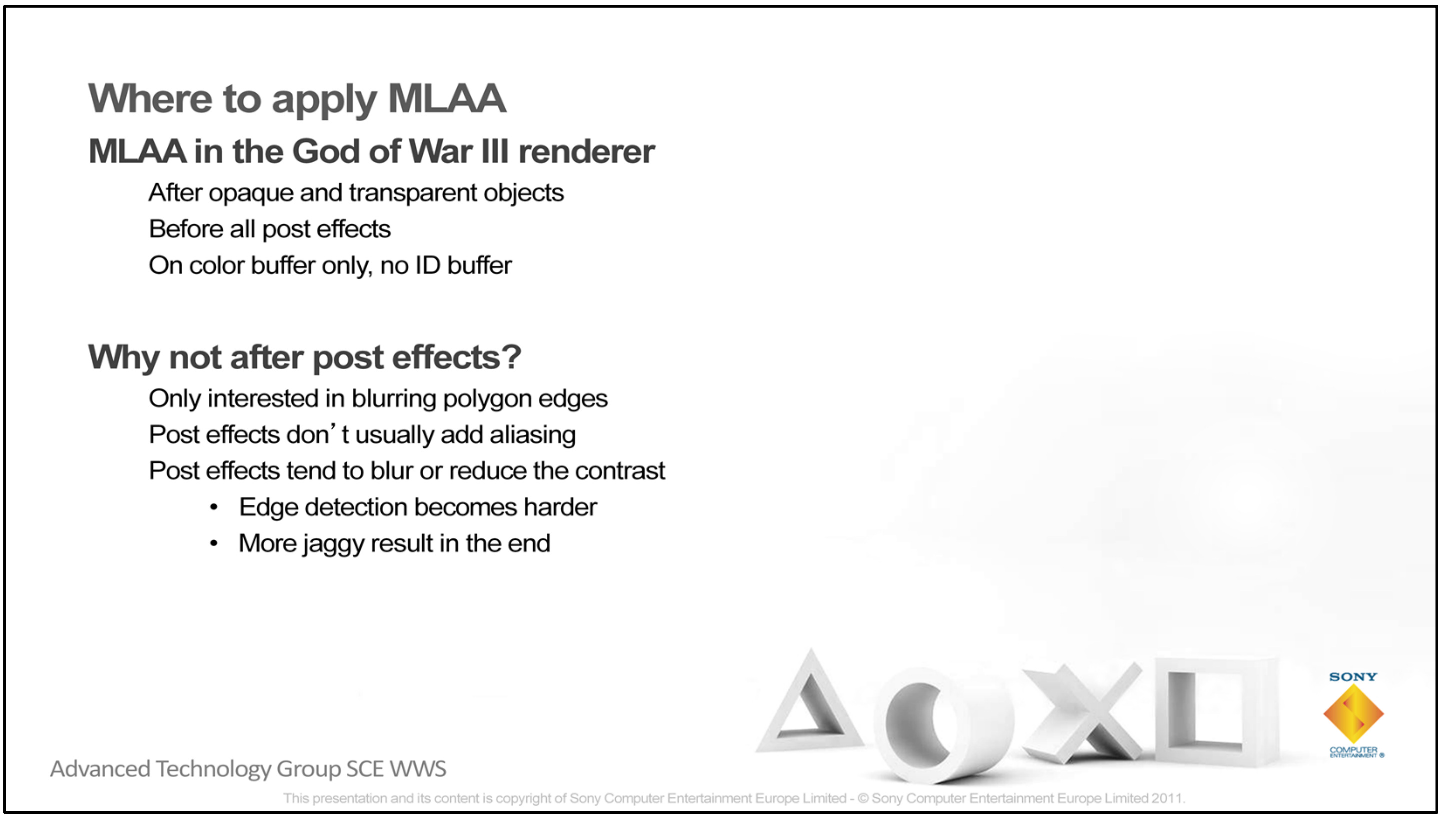
And that's all from me. I'd like to then these teams and individuals and you for listening. Cedric will now be talking about the God of War 3 integration and demo it to you, so you can get some intuition about how this looks in a real game.



Low latency MLAA in God of War III

Cedric Perthuis
SCE Santa Monica Studio

Advanced Technology Group SCE WWS



Where to apply MLAA

MLAA in the God of War III renderer

After opaque and transparent objects

Before all post effects

On color buffer only, no ID buffer

Why not after post effects?

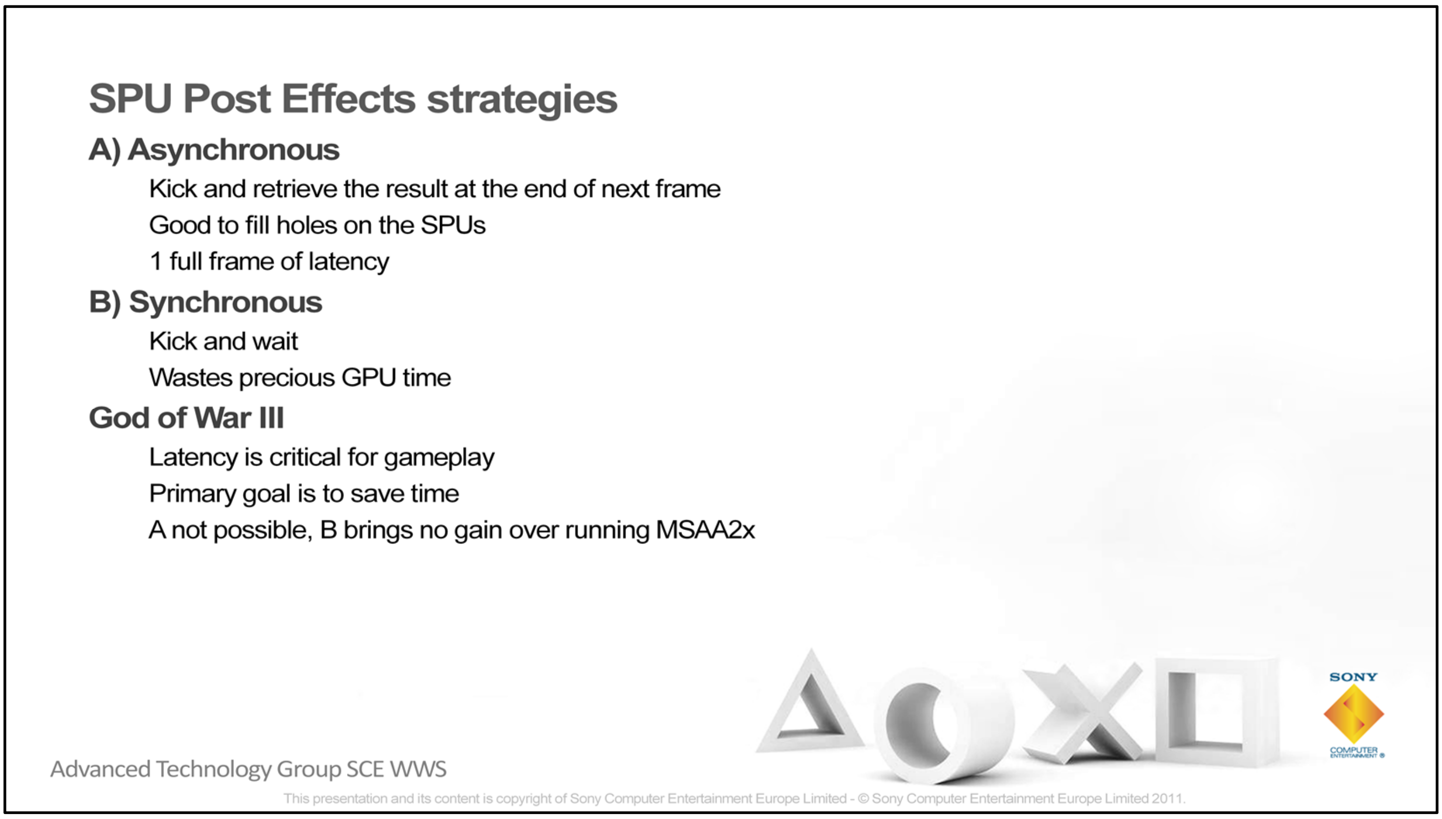
Only interested in blurring polygon edges

Post effects don't usually add aliasing

Post effects tend to blur or reduce the contrast

- Edge detection becomes harder
- More jaggy result in the end

Advanced Technology Group SCE WWS



SPU Post Effects strategies

A) Asynchronous

- Kick and retrieve the result at the end of next frame
- Good to fill holes on the SPUs
- 1 full frame of latency

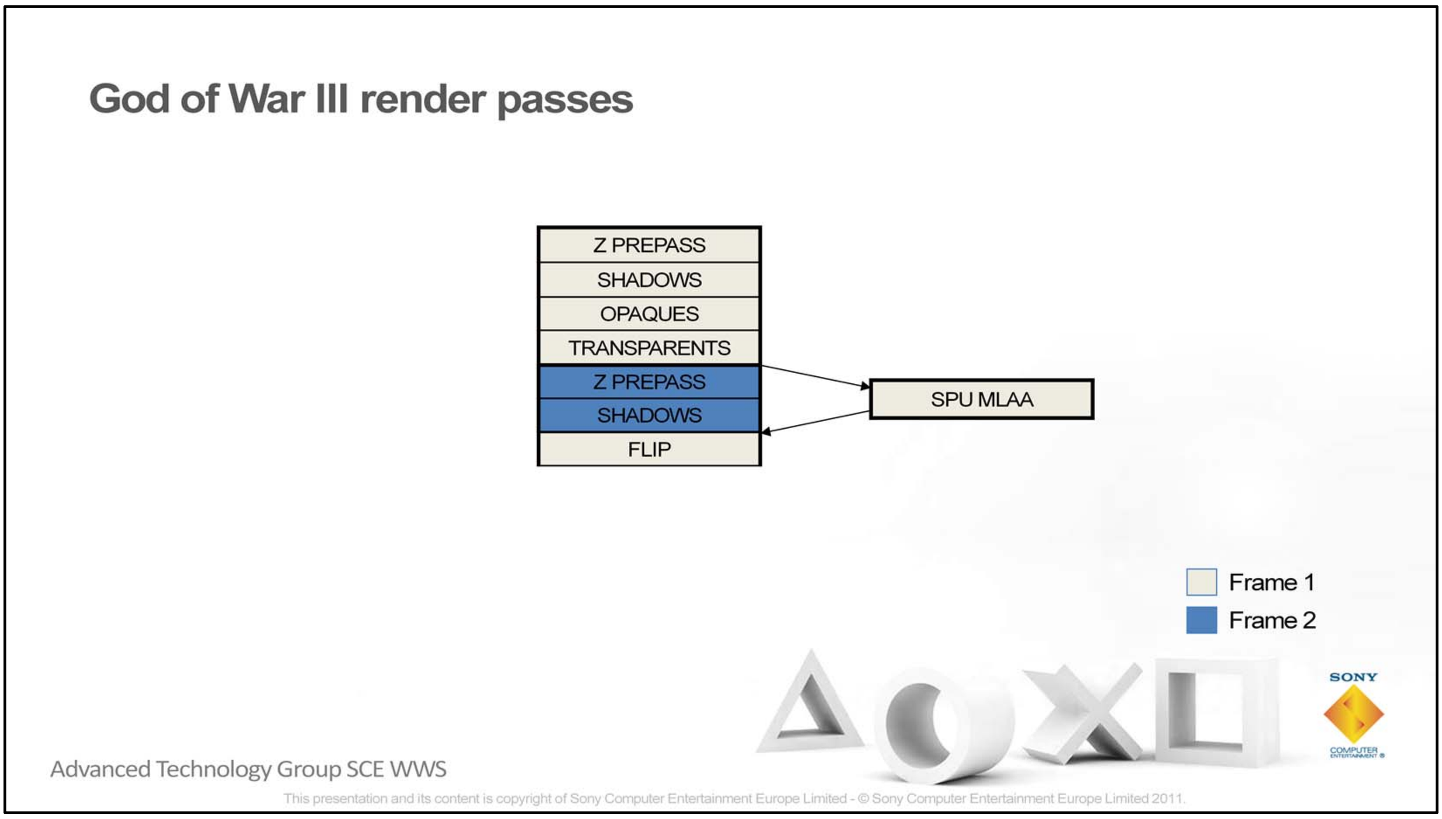
B) Synchronous

- Kick and wait
- Wastes precious GPU time

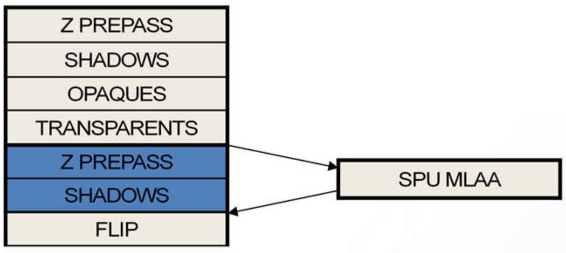
God of War III

- Latency is critical for gameplay
- Primary goal is to save time
- A not possible, B brings no gain over running MSAA2x

Advanced Technology Group SCE WWS

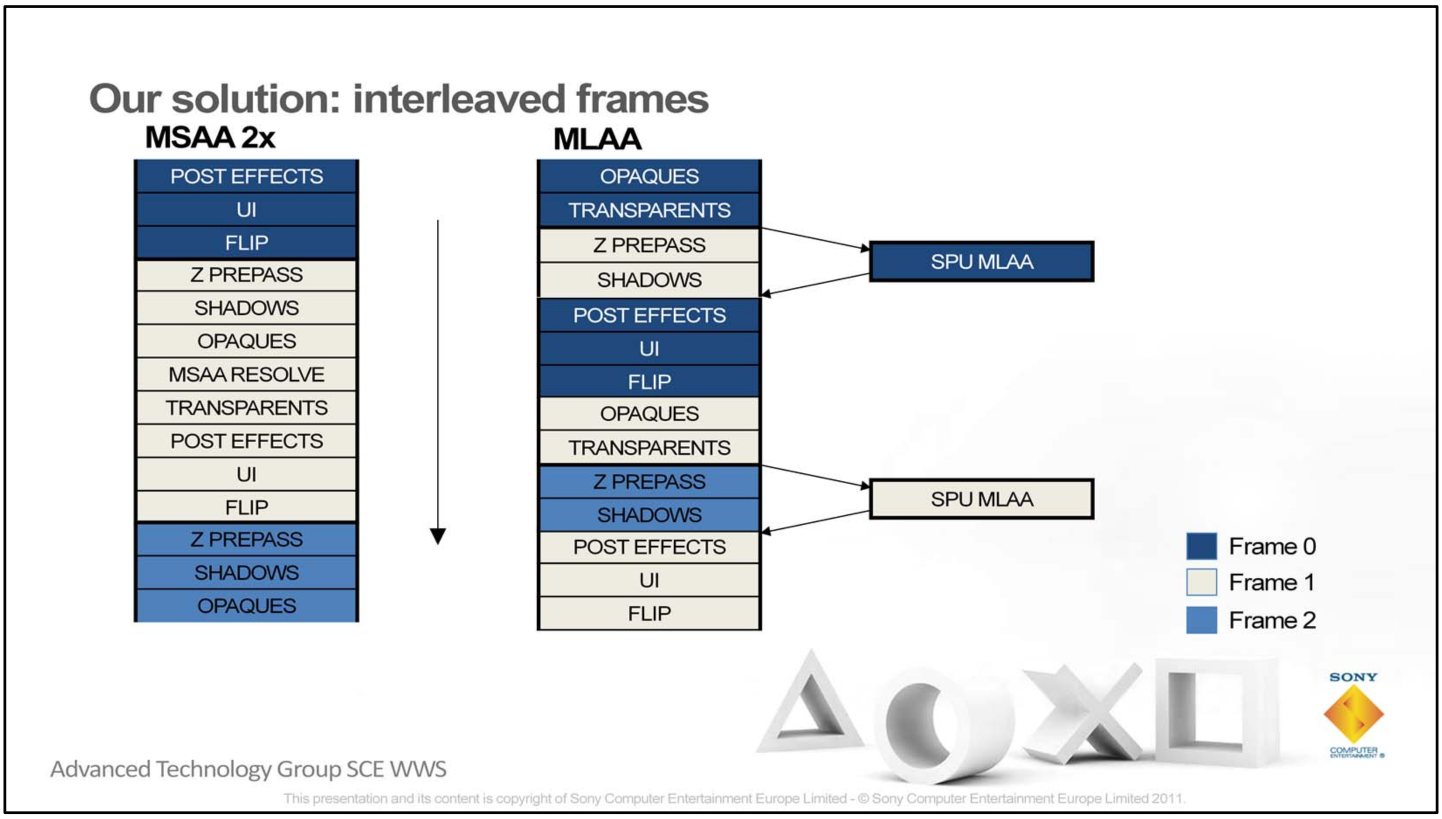


God of War III render passes



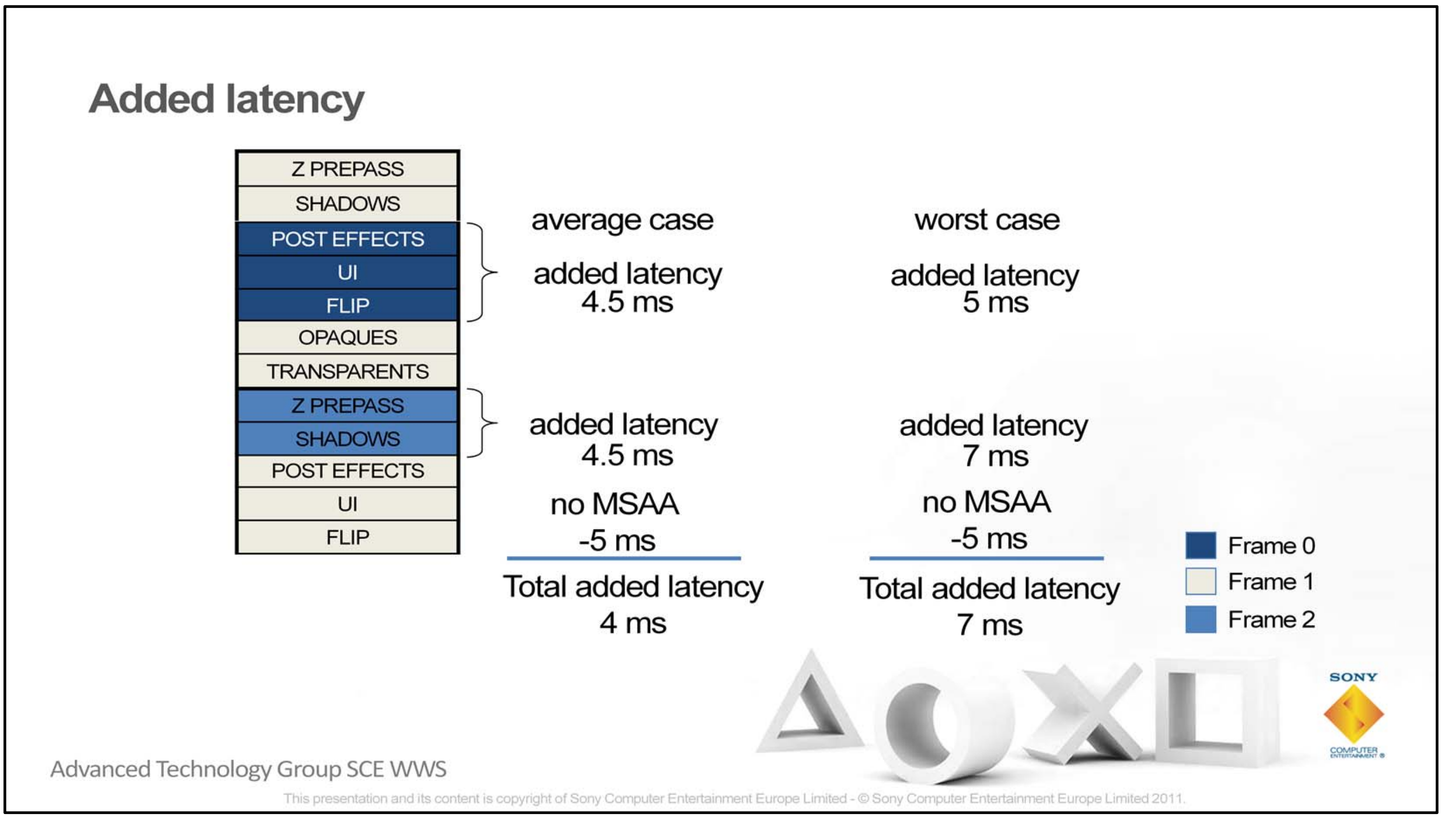
⬜ Frame 1
🟦 Frame 2

Advanced Technology Group SCE WWS



Our solution: interleaved frames

Advanced Technology Group SCE WWS



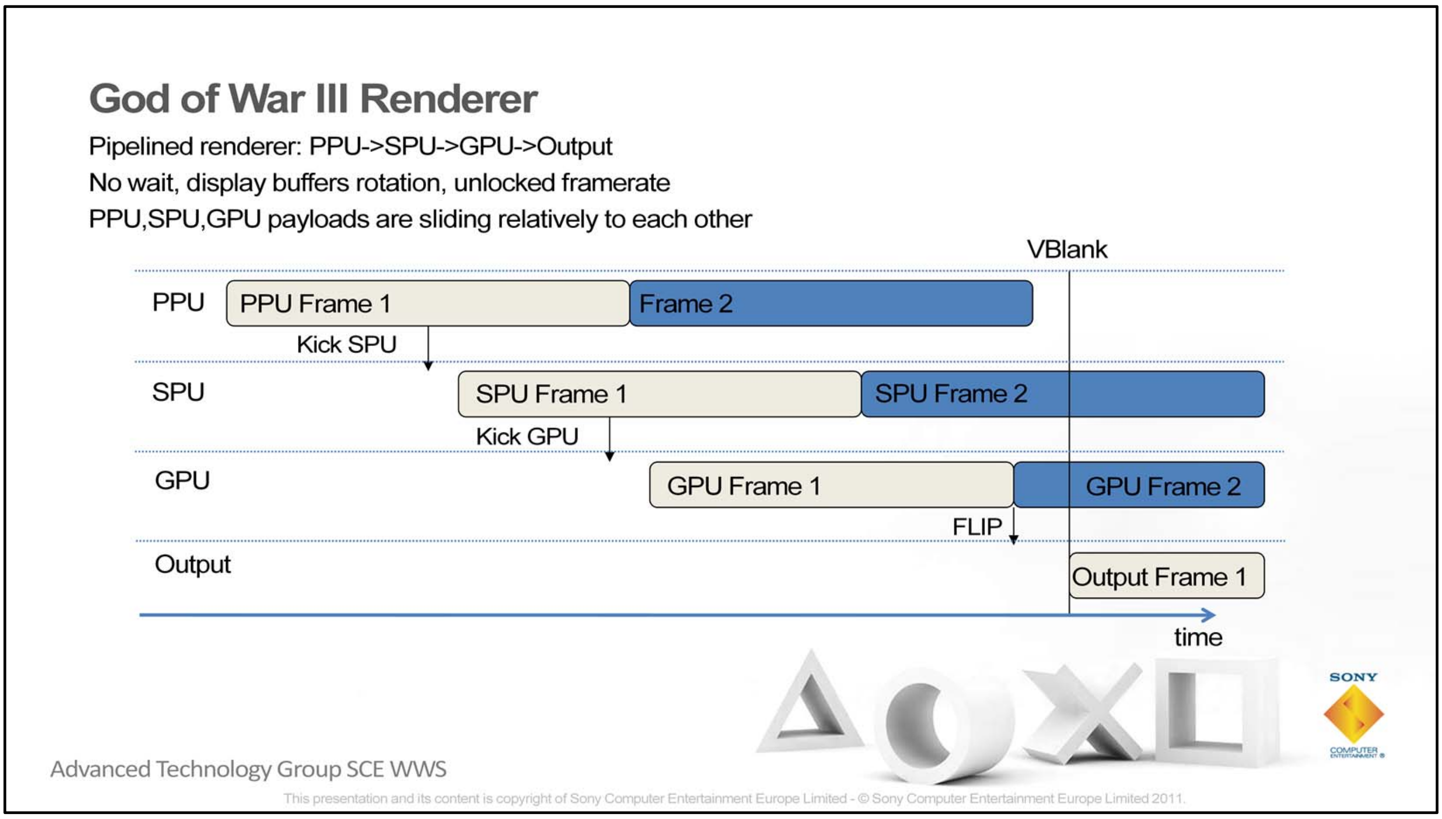
Added latency

- Z PREPASS
- SHADOWS
- POST EFFECTS
- UI
- FLIP
- OPAQUES
- TRANSPARENTS
- Z PREPASS
- SHADOWS
- POST EFFECTS
- UI
- FLIP

average case	worst case
added latency 4.5 ms	added latency 5 ms
added latency 4.5 ms	added latency 7 ms
no MSAA -5 ms	no MSAA -5 ms
Total added latency 4 ms	Total added latency 7 ms

🟦 Frame 0
⬜ Frame 1
🟦 Frame 2

Advanced Technology Group SCE WWS

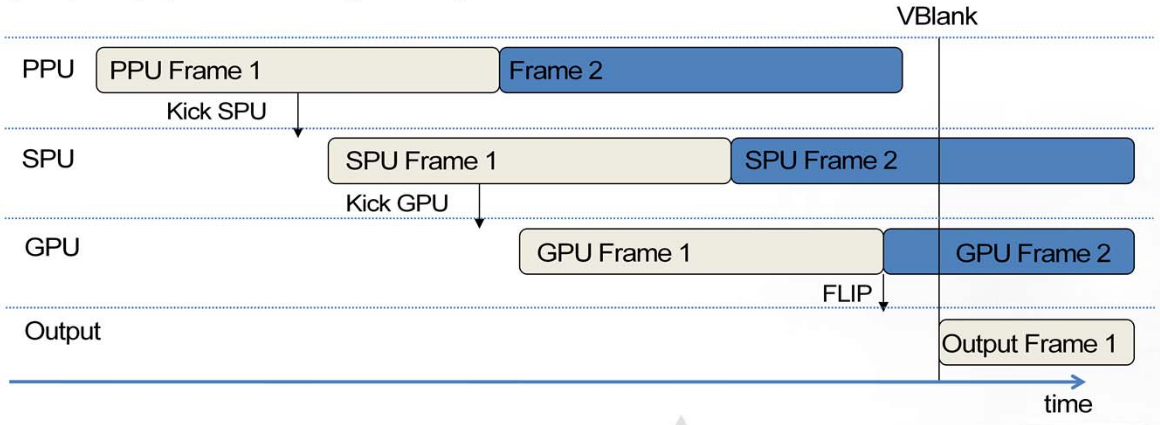


God of War III Renderer

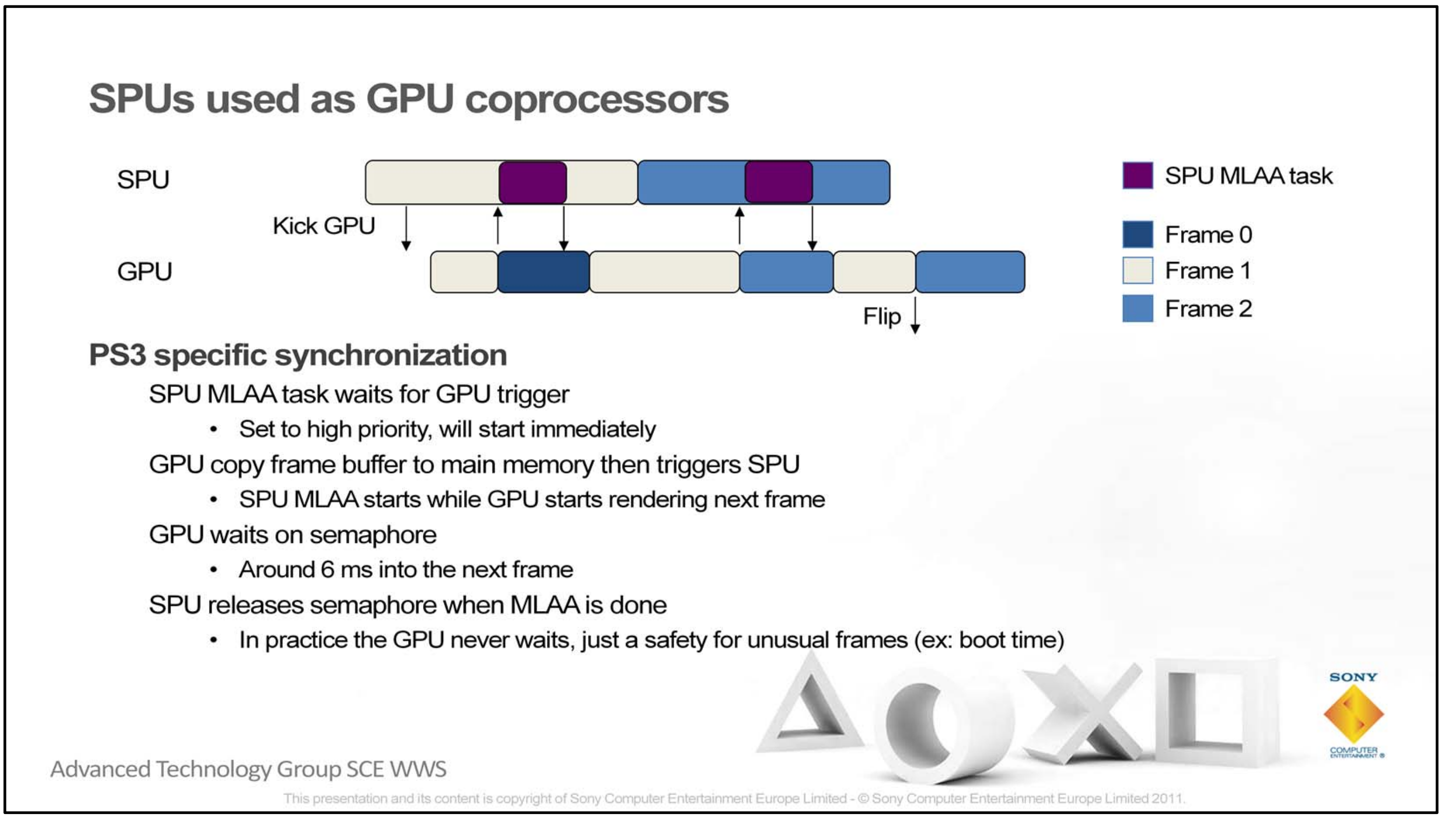
Pipelined renderer: PPU->SPU->GPU->Output

No wait, display buffers rotation, unlocked framerate

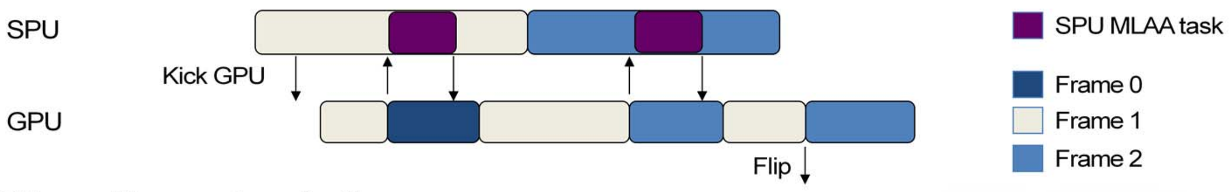
PPU,SPU,GPU payloads are sliding relatively to each other



Advanced Technology Group SCE WWS



SPUs used as GPU coprocessors



PS3 specific synchronization

SPU MLAA task waits for GPU trigger
- Set to high priority, will start immediately

GPU copy frame buffer to main memory then triggers SPU
- SPU MLAA starts while GPU starts rendering next frame

GPU waits on semaphore
- Around 6 ms into the next frame

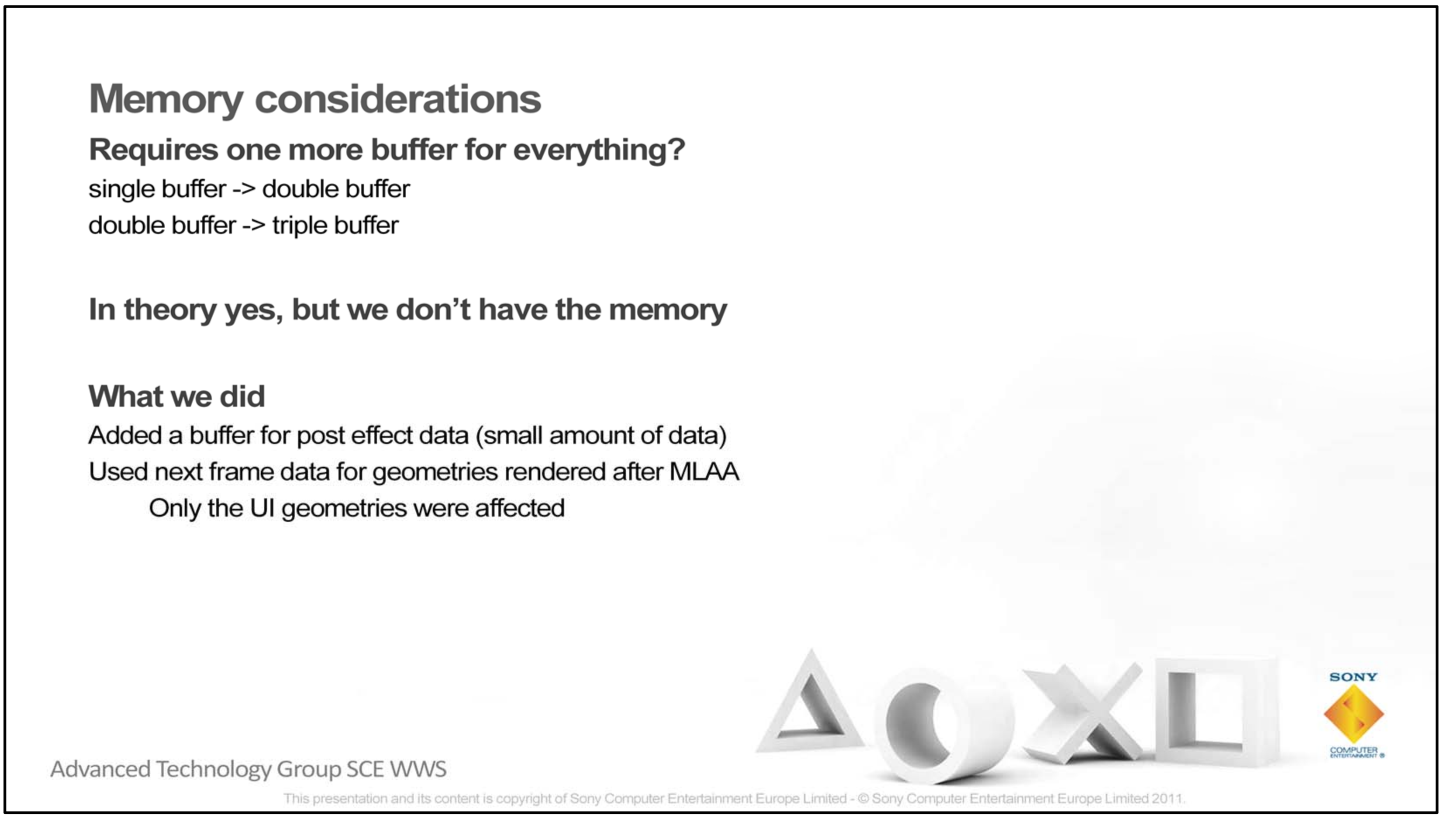
SPU releases semaphore when MLAA is done
- In practice the GPU never waits, just a safety for unusual frames (ex: boot time)

Advanced Technology Group SCE WWS

Uses relatively advanced PS3 features

Rarely used monitoring feature of the SPU kernel, and GPU back-end report command

Allows perfect coupling between GPU and SPU MLAA task



Memory considerations

Requires one more buffer for everything?

single buffer -> double buffer

double buffer -> triple buffer

In theory yes, but we don't have the memory

What we did

Added a buffer for post effect data (small amount of data)

Used next frame data for geometries rendered after MLAA

- Only the UI geometries were affected

Advanced Technology Group SCE WWS

Should require to keep all draw data for 1 more frame

- Ok for post effects, very little data
- Too difficult for geometries due to the deep renderer pipeline and the memory requirements

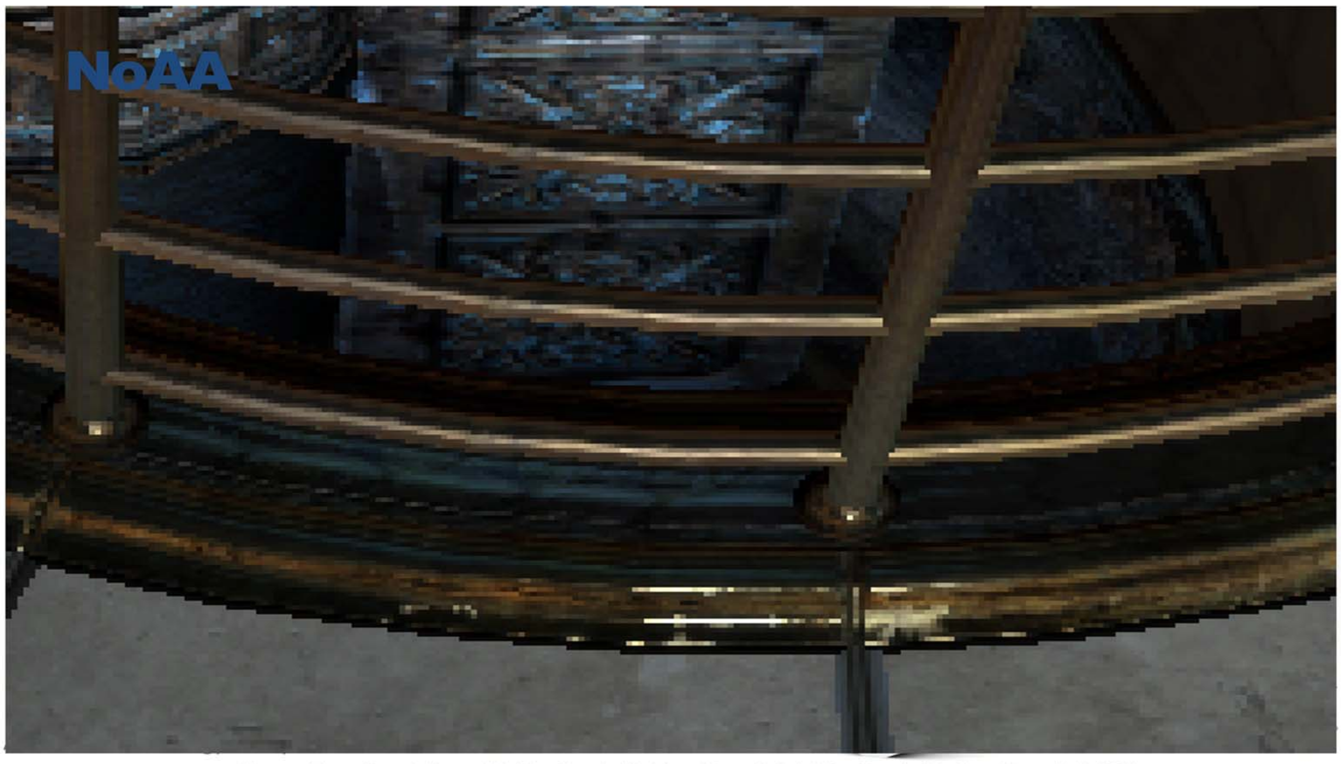
God of War III trick:

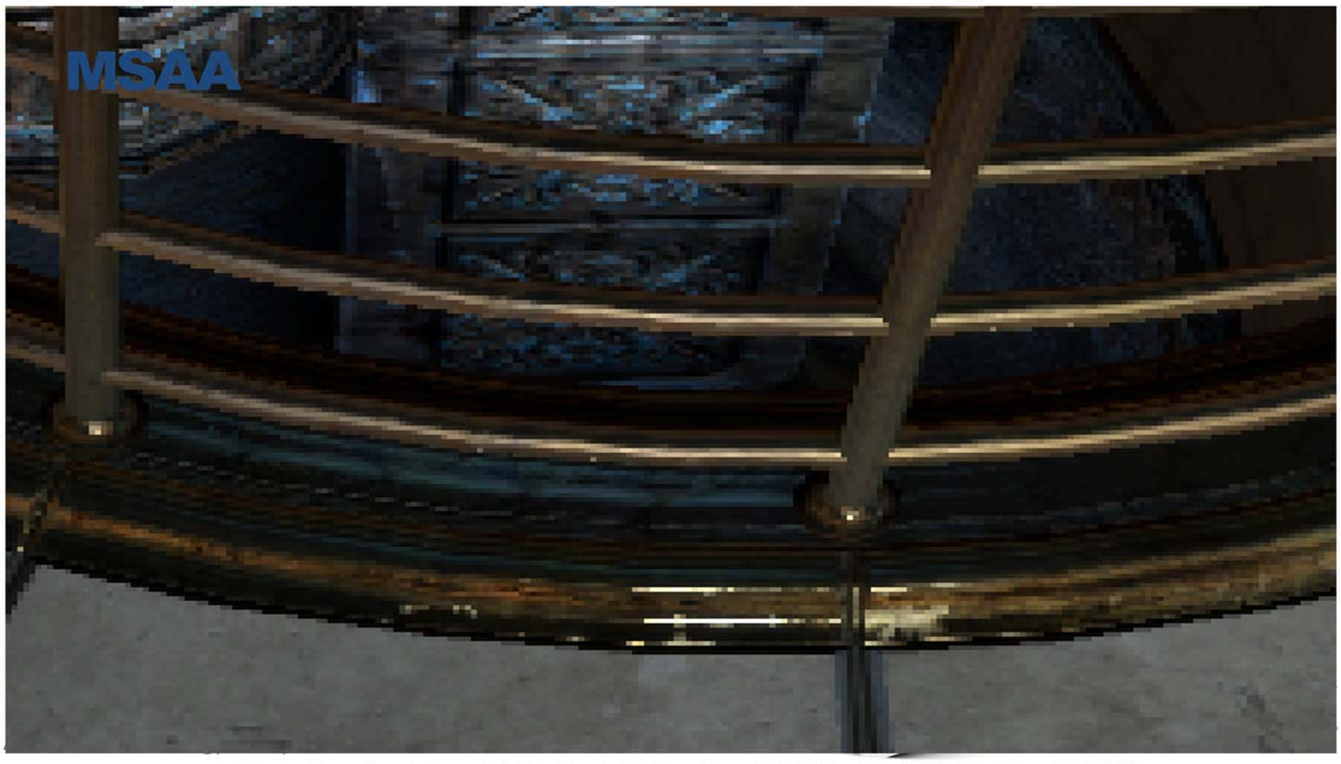
- For post MLAA geometries, use next frame geometry

Results

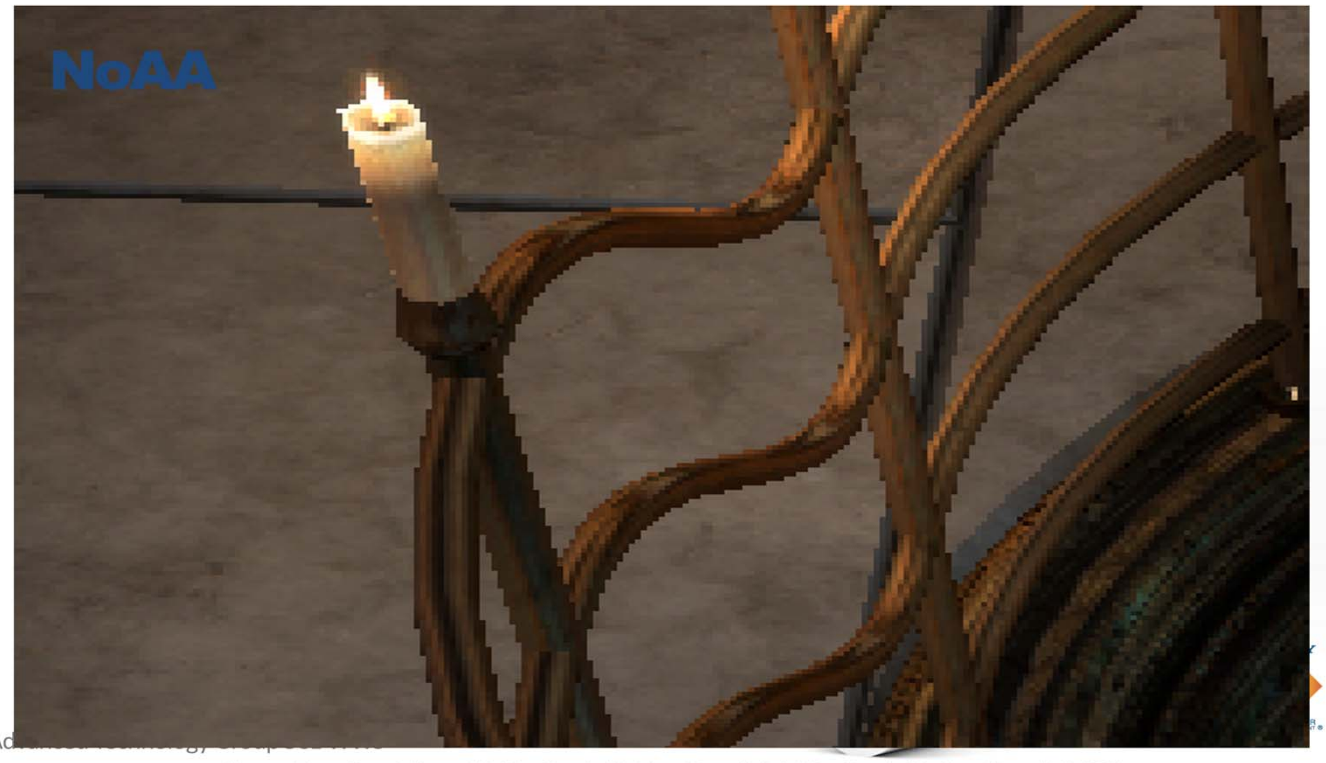
Live PS3 demo

Advanced Technology Group SCE WWS

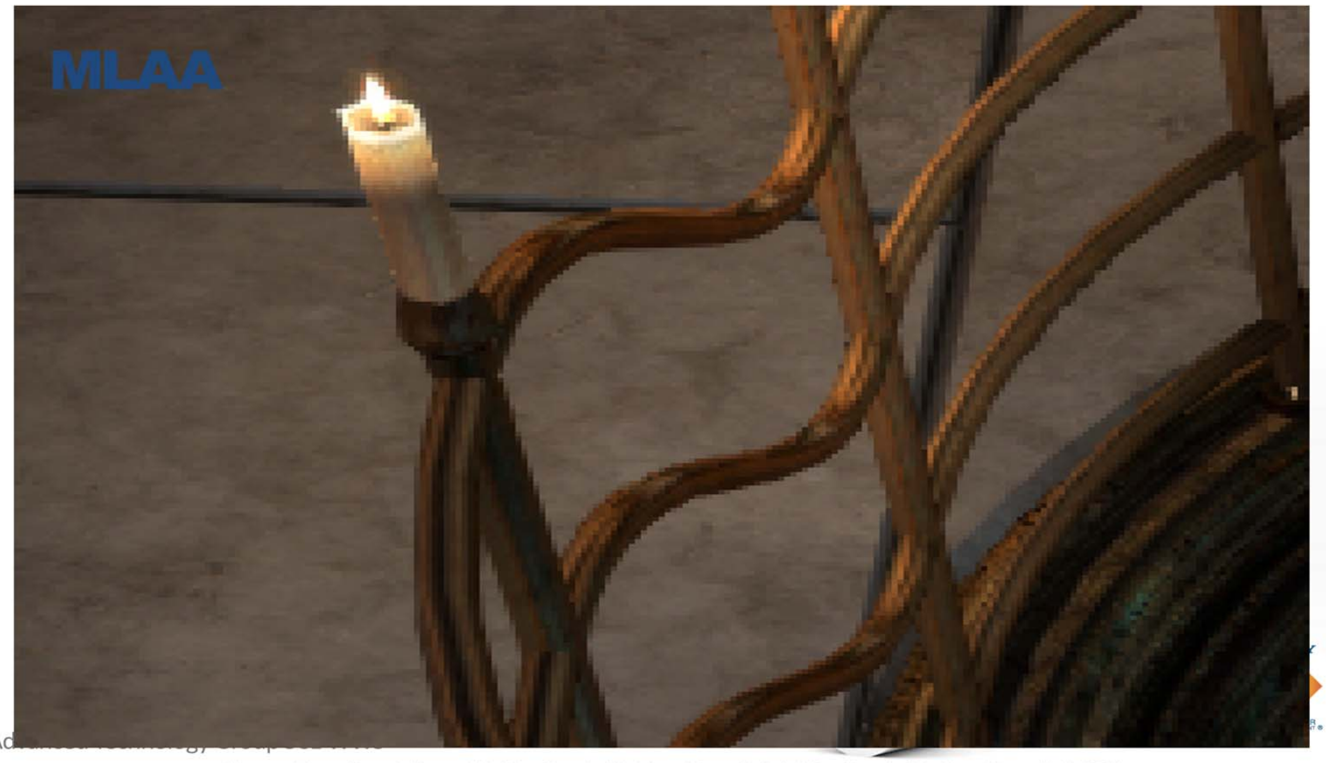

NoAA


MSAA

MLAA

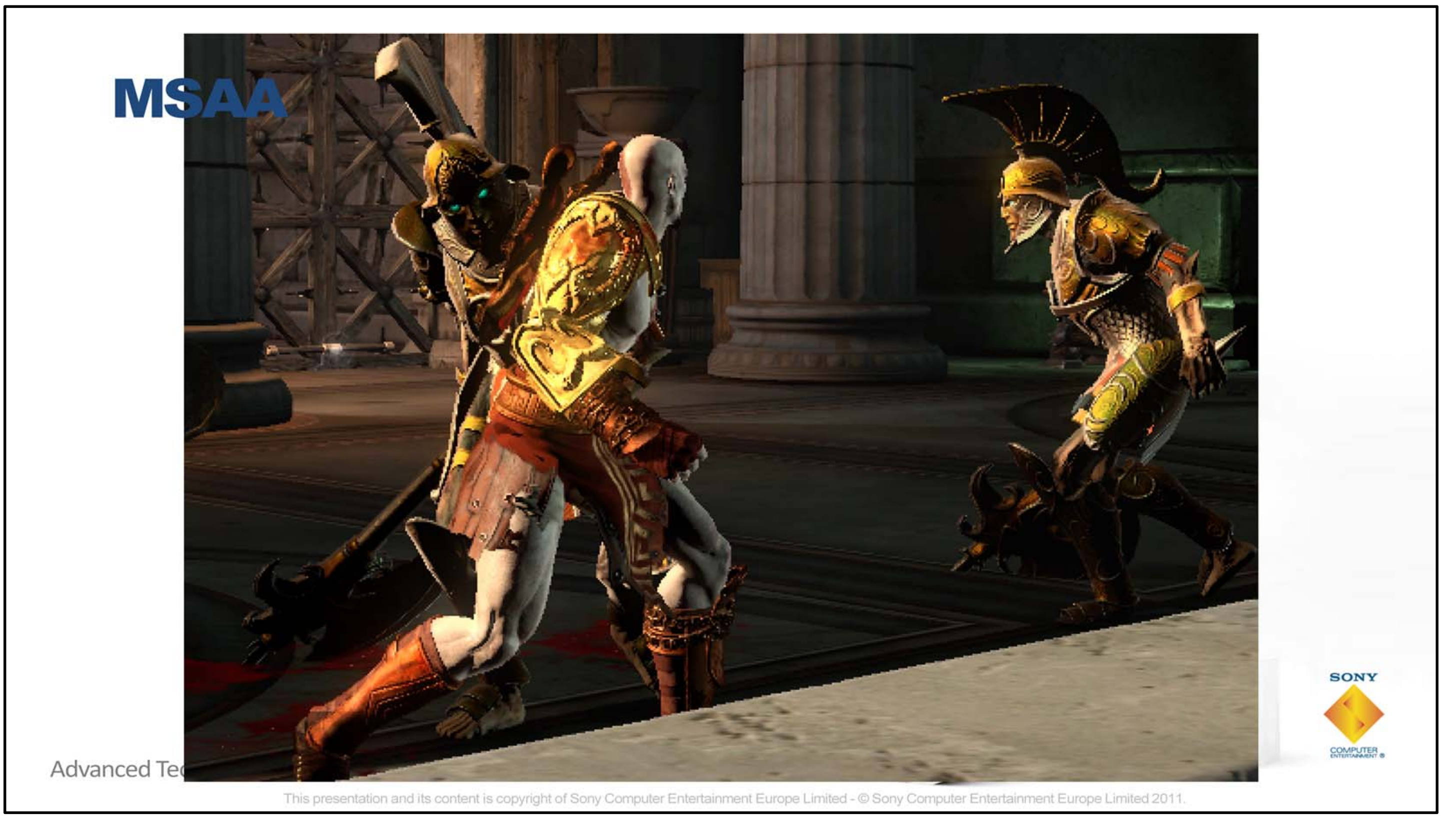

NoAA

MSAA


MLAA

NoAA

Advanced Te



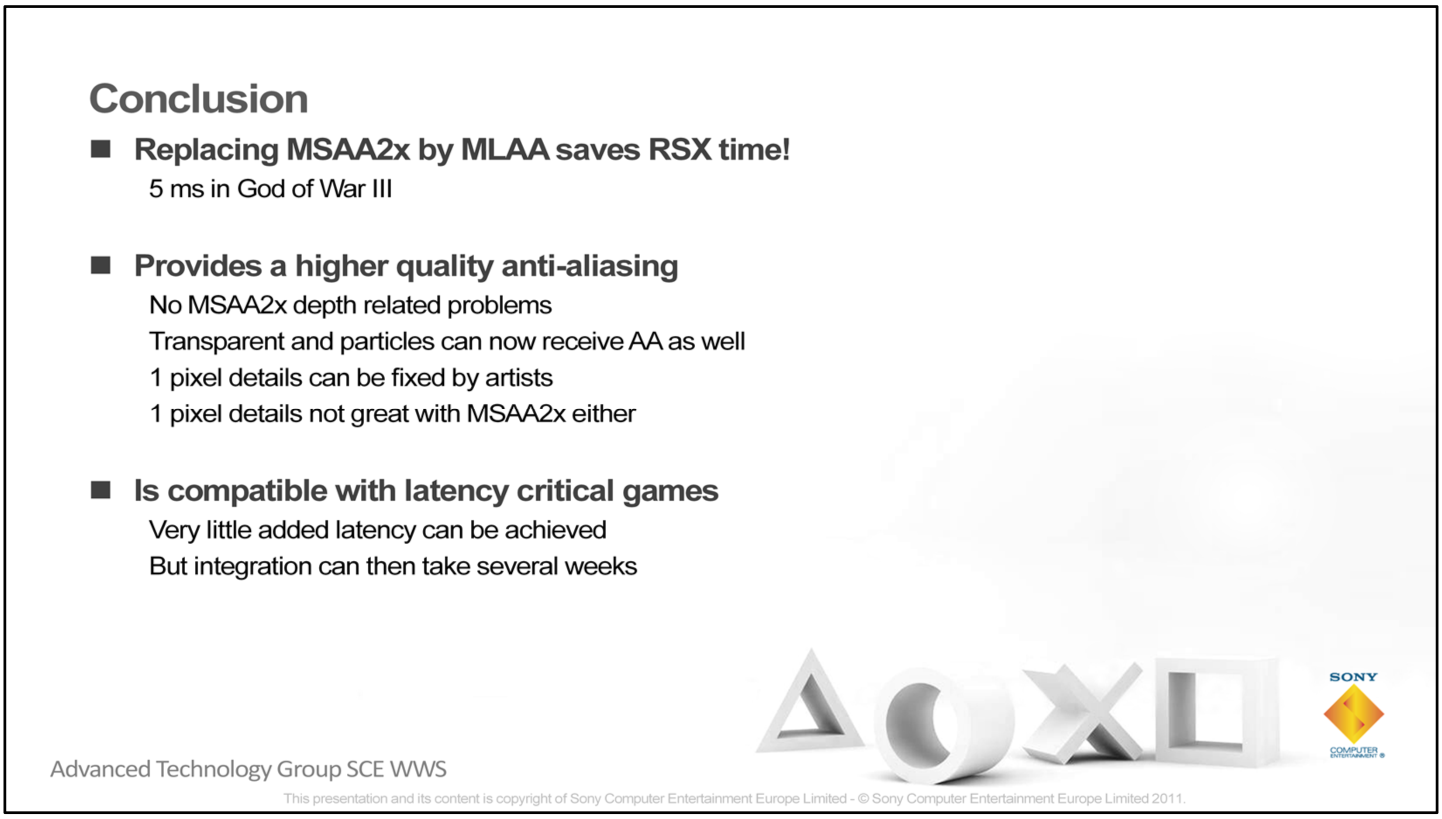
MSAA

Advanced Te

SONY COMPUTER ENTERTAINMENT

MLAA

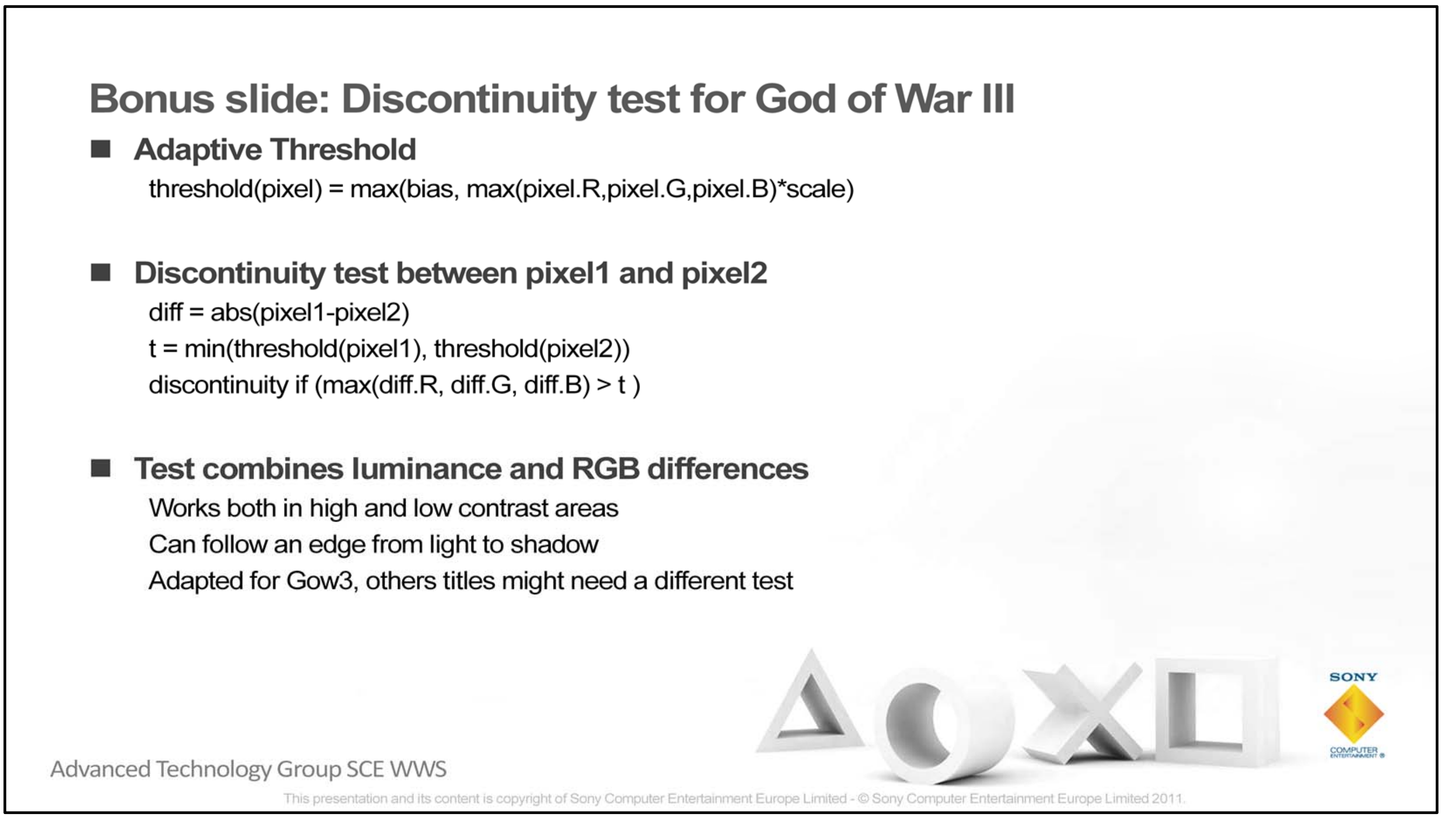
Advanced Te



Conclusion

- **Replacing MSAA2x by MLAA saves RSX time!**
  - 5 ms in God of War III
- **Provides a higher quality anti-aliasing**
  - No MSAA2x depth related problems
  - Transparent and particles can now receive AA as well
  - 1 pixel details can be fixed by artists
  - 1 pixel details not great with MSAA2x either
- **Is compatible with latency critical games**
  - Very little added latency can be achieved
  - But integration can then take several weeks

Advanced Technology Group SCE WWS



Bonus slide: Discontinuity test for God of War III

- **Adaptive Threshold**

```
threshold(pixel) = max(bias, max(pixel.R,pixel.G,pixel.B)*scale)
```

- **Discontinuity test between pixel1 and pixel2**

```
diff = abs(pixel1-pixel2)
t = min(threshold(pixel1), threshold(pixel2))
discontinuity if (max(diff.R, diff.G, diff.B) > t )
```

- **Test combines luminance and RGB differences**
  - Works both in high and low contrast areas
  - Can follow an edge from light to shadow
  - Adapted for Gow3, others titles might need a different test

Advanced Technology Group SCE WWS

MLAA on PS3

Tobias Berghoff
Matteo Scapuzzi
SCE Worldwide Studios Advanced Technology Group

Cedric Perthuis
SCE Worldwide Studios Santa Monica Studio

Thanks!