

# The Danger Zone

## 99.9% genuine graphics programming

WRITTEN BY MJPAUGUST 16, 2010NOVEMBER 6, 2016

## Deferred MSAA

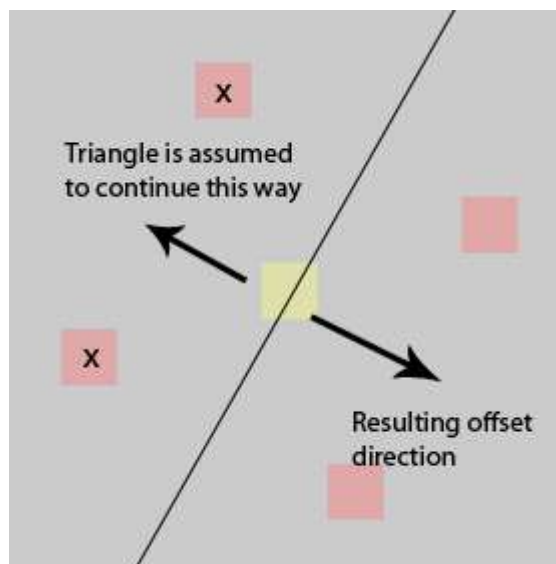
A long while ago I was looking into morphological antialiasing (<http://visual-computing.intel-research.net/publications/papers/2009/mlaa/mlaa.pdf>) (MLAA) to see if I could somehow make it practical for a GPU that isn't the latest monster from Nvidia or ATI. With MLAA most people talk about how nicely it cleans up edges (which it certainly does), but for me the really cool part is how it's completely orthogonal to the technique used to render the image. It could have been rasterized and forward rendered, it could be the product of a deferred rendering, or it could even be ray-traced: in all cases the algorithm works the same. This seems pretty important to me, since in real-time rendering we're getting further and further away from the old days of forward rendering to the backbuffer (AKA, the case where MSAA actually works properly).

Anyway I never really got anywhere (although someone took a stab at it (<http://igm.univ-mlv.fr/~biri/mlaa-gpu/MLAAGPU.pdf>) in the meantime), but what *DID* happen is that I spent some time thinking about the downsides of MLAA and how they could be avoided. In particular, I'm talking about the inability of MLAA to handle temporal changes to polygon edges due to no sub-pixel information being available. MSAA doesn't have the same problem, since it at handles depth testing and triangle coverage at a sub-pixel level. So I thought to myself: "wouldn't it be nice to use MSAA to get coverage info, and then use that to some smart filtering in a post process?" Turns out it's not too difficult to pull it off, and the results aren't too bad. I'm calling my sample "Deferred MSAA" even though that's a terrible name for it, but I can't really think of anything better. Here's the basic premise:

1. Render all scene geometry in a depth-only prepass, with 4xMSAA enabled
2. Render the scene normally with no MSAA to an HDR render target
3. Apply tone mapping and other post processes
4. Apply a single "deferred AA" pass that samples the MSAA depth buffer from step #1, uses it to calculate pixel coverage, and filters based on the coverage

To "calculate pixel coverage", all I do is read all 4 depth subsamples and compare them (with a small tolerance) with the depth value from the non-MSAA pass. The idea is to try to figure out which depth subsamples came from rasterizing the same triangle that was rasterized in the non-MSAA pass. Then, using knowledge of the points in the sample pattern, I come up with a sample location where it's likely that we'll a color on the other side of the triangle edge. The method I use to do this is really simple: if a

subsample fails the equivalence test, add a vector from the pixel center to the sample location to a summed offset vector. Then after all subsamples are compared, average the offsets and take a single bilinear sample at that offset. Check out the diagram below for a visual representation:



(<https://mynameismjp.files.wordpress.com/2010/08/offset.png>).

As I see it, this approach has these things going for it:

1. Like with MLAA, AA no longer has anything to do with how you shade a pixel. So if your going deferred, there's no "lighting the subsamples" nonsense. It also means you don't necessarily need to render your G-Buffer with MSAA, which can save a lot in terms of memory usage and bandwidth.
2. Also like MLAA, AA is applied *after* tone mapping. So you don't have to tone map each subsample (<http://www.humus.name/index.php?page=3D&ID=77>) to avoid aliasing where there's high contrast.
3. Unlike MLAA, it's simple and straightforward to implement on a GPU in a pixel shader.
4. Unlike MLAA, it uses sub-pixel information which helps it handle temporal changes a lot better.

For comparison, here's a screenshot from my sample app (click for full size):



(<https://mynameismjp.files.wordpress.com/2010/08/comparison.png>).

That particular shot shows off the weakness of both standard MSAA and my approach. If you look at the high-contrast area whether the tank overlaps with the bright sky, you can see how standard MSAA totally fails to do anything about the edge aliasing. This is because the MSAA resolve is done on the HDR pixel values, and tone mapping is a non-linear operation. However in the darker areas towards the bottom MSAA does just fine. In contrast (pun not intended), deferred MSAA holds up really well in the silhouettes. The AA quality is surprisingly pretty decent for an edge filtering-based approach, and since it's done after tone mapping there's no non-linearity to worry about. However if you look carefully down in the tank, you'll spot edges that don't get the AA treatment at all. This is because the depth values are similar enough to pass the comparison test, despite coming from different triangles. You can see it most clearly in the grate on the front of the tank. The sample app has a tweakable threshold value

for the depth comparison, and setting it lower causes more edges to be properly distinguished. However it also starts to cause false positives in triangles nearly parallel to the camera. The image below shows the same scene with a default threshold, and with a low threshold:



(<https://mynameismjp.files.wordpress.com/2010/08/threshold.png>).

Even with a low threshold, some edges are still missed since only the normal or material properties will change between two triangles and not the depth. For those cases more information would be required, such as the scene normals or an ID buffer. But you could probably make the case that edges with depth discontinuities are more likely to have high contrast, and the high-contrast areas are where AA makes the biggest difference.

Aside from the quality issues, the other obvious concern with this approach is performance. Even with depth only, an MSAA prepass isn't going to be cheap on a lot of hardware. It makes more sense if the depth prepass isn't wasted and only used for the post process...for instance it's certainly possible to do downscale the MSAA depth buffer with a MAX operation and use it to pre-populate the non-MSAA depth buffer. However it may be hard to get this to play nice with early z-cull, depending on the hardware. You could also possibly lay out your G-Buffer for light prepass with MSAA enabled, but then you might have trouble using depth values to reconstruct position.

You can get the sample code and binaries here:

<https://mynameismjp.files.wordpress.com/2016/11/dmsaa1.zip>

(<https://mynameismjp.files.wordpress.com/2016/11/dmsaa1.zip>). This one requires a D3D\_FEATURE\_LEVEL\_11\_0-capable device.

POSTED IN DIRECTX, DIRECTX 11, GRAPHICS, PROGRAMMING.

## 4 thoughts on “Deferred MSAA”

1.  **DIRECTTOVIDEO SAYS:**  
AUGUST 16, 2010 AT 2:47 AM

This is actually quite similar actually to what I did here:


<http://directtovideo.wordpress.com/2009/11/13/deferred-rendering-in-frameranger/>  
(see the end of the post)

I do this:

- render the gbuffers to non-MSAA, light them and composite them and so on.
- render a pass to a MSAA RGBA8 colour target. pass up depth, normal, object ID and material ID from the vertex shader. In the fragment shader, read the GBuffer depth+normals and calculate weights, and use these weights to perform a bilateral upsample of the colour buffer.
- finally resolve that MSAA colour buffer with a stretchrect.

As an optimisation – perform an edge detect pass on the gbuffers to lay down stencil to mask out pixels in the MSAA pass.

The result is pretty nice actually, and it works on DX9. As you said, it's a lot nicer if you use normal and ID information too – depth doesn't give you great results. Normal is probably the most important feature.

2.  **MAXEST SAYS:**  
AUGUST 3, 2011 AT 11:16 AM

One question: what does 4xMSAA mean? In DX9 you don't have access to subsamples, right? So by 4xMSAA you simply mean creating a 4x bigger 1-sample render target?

3.  **ANONYMOUS SAYS:**  
AUGUST 30, 2016 AT 3:55 AM

links dead 🙄

4.  **MJP SAYS:**  
AUGUST 31, 2016 AT 10:04 PM

Sorry about that, I re-uploaded the file and updated the link.

*Blog at WordPress.com. Do Not Sell or Share My Personal Information*