



Resolve your Resolves

Jon Story

Holger Gruen

AMD Graphics Products Group

jon.story@amd.com

holger.gruen@amd.com

Introduction

Over the last few years it has become common place for PC games to make use of Multi-Sample Anti-Aliasing (MSAA) to achieve higher quality rendering. MSAA is a very effective and efficient method for reducing the unsightly “jaggies” that result from the triangle rasterization process. At the same time most game engines also employ post processing techniques such as depth-of-field, motion blur, colour correction and refraction. Post-processing has become increasingly popular, as it provides a way to carry out complex computations, but only pay the cost for visible pixels. It is not unheard of for an engine to contain up to 20 passes, and these techniques usually require a copy of the main render target as a texture input. If the engine is making use of MSAA, then the render target will need to be resolved before it can be used in the next pass. This is accomplished through calls to *IDirect3DDevice9::StretchRect* or *ID3D10Device::ResolveSubresource*, depending on which version of D3D is being used. As modern game engines tend to apply multiple post-processing techniques, it is easy to understand how the application could trigger a loop of resolves, see Figure 1 below.

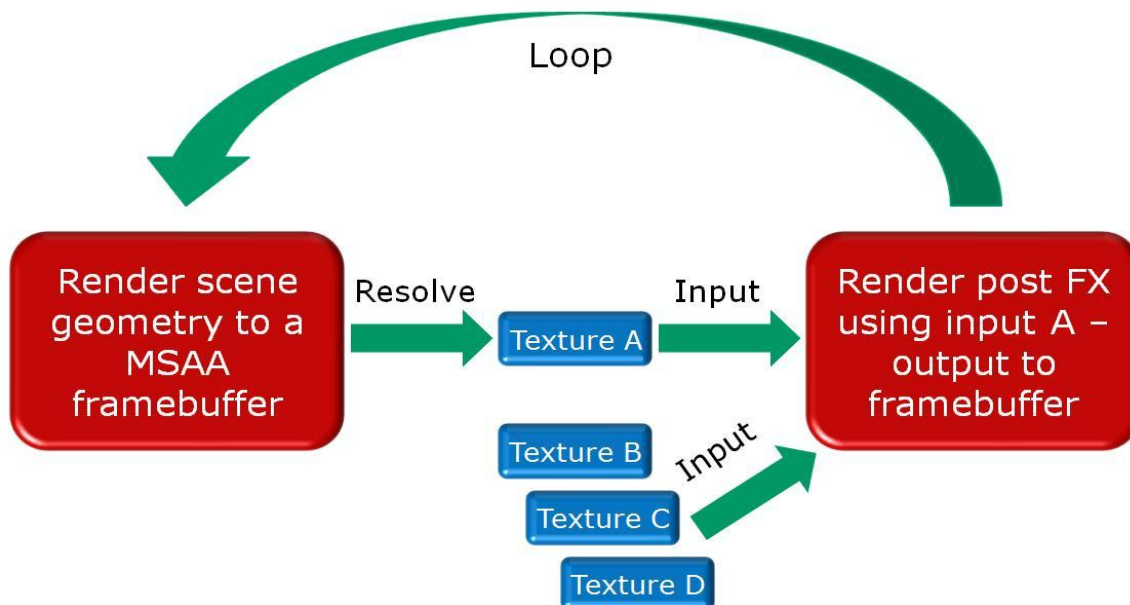


Figure 1 (Resolve Loop)

It is critically important to understand that a resolve is not a free operation, and that performing multiple resolves per frame can have a very serious impact on performance. This statement is true for all graphics hardware. To take a real world example, the developers of a recently released PC title managed to reduce their resolve count from a staggering 22 to just 12. This generated a saving of around 12 ms per frame, at a resolution of 1280x1024@4xAA.

The goal of this paper is to describe how to minimize the resolve count in the rendering pipeline without compromising the quality of post-processing effects or deferred shading techniques. The resolves that should be removed fall into two categories, redundant resolves and harmful resolves, and these will be described in detail later in this paper. But first let's consider the resolves that are necessary for good image quality.

Useful Resolves

We know that the use of MSAA render targets is only helpful when draw calls produce visible "jaggies". In an ideal world the main geometry pass would be rendered in MSAA mode, and then resolved to a non-MSAA render target. Any subsequent post processing passes would all be completed in non-MSAA mode. This would therefore give rise to just a single resolve per frame.

However there are two reasons why a post processing technique may need to be performed in MSAA mode:

- 1) If a post processing technique enables subsample based depth testing, it can result in an update to some of the subsamples of a pixel.
- 2) In a similar way if alpha blending is enabled, then subsample data is preserved through the blend operation.

In these two cases it does indeed make sense to resolve the render target for further passes. However these two examples are the exception and it should be noted that for full screen passes that do not enable depth testing or alpha blending, there is precious little point in using MSAA mode.

Redundant Resolves

A technique that does not actually draw any geometry, other than a full screen quad, will usually write the same color to all subsamples in a MSAA render target as depicted in Figure 2 below. The reason for this is that the pixel shader is only run once per pixel and the whole pixel is covered. Effectively the MSAA buffer has been turned into a non-MSAA buffer, and every further resolve operation on this surface is redundant. Aside from the obvious redundancy, once the same color has been written to all subsamples of the corresponding pixels, it should be noted that the MSAA depth buffer does not actually match the silhouettes of the objects anymore.

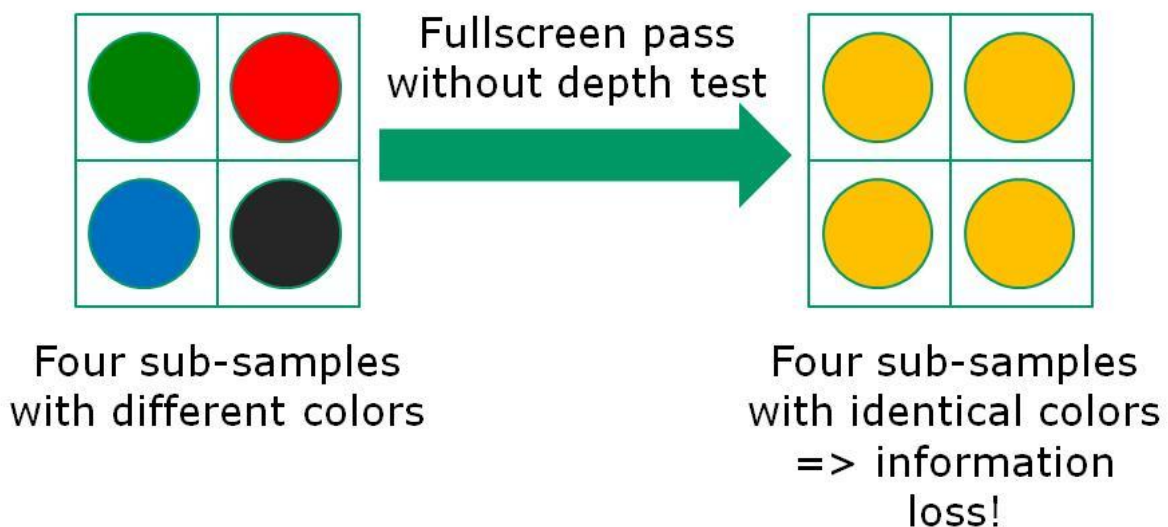


Figure 2 (Full screen Pass)

Clearly the solution is to render these passes in non-MSAA mode, thus completely avoiding the need to perform resolves. The recommended way to avoid these unnecessary resolves is as follows:

- 1) Create the main frame buffer (swap chain) in non-MSAA mode.
- 2) Create an intermediate MSAA render target where the main scene geometry is rendered, and anything else that would result in “jaggies”.
- 3) Perform a resolve of the intermediate MSAA render target to a non-MSAA surface.
- 4) Ping pong between non-MSAA render targets for the remaining passes as shown in Figure 3 on the next page.

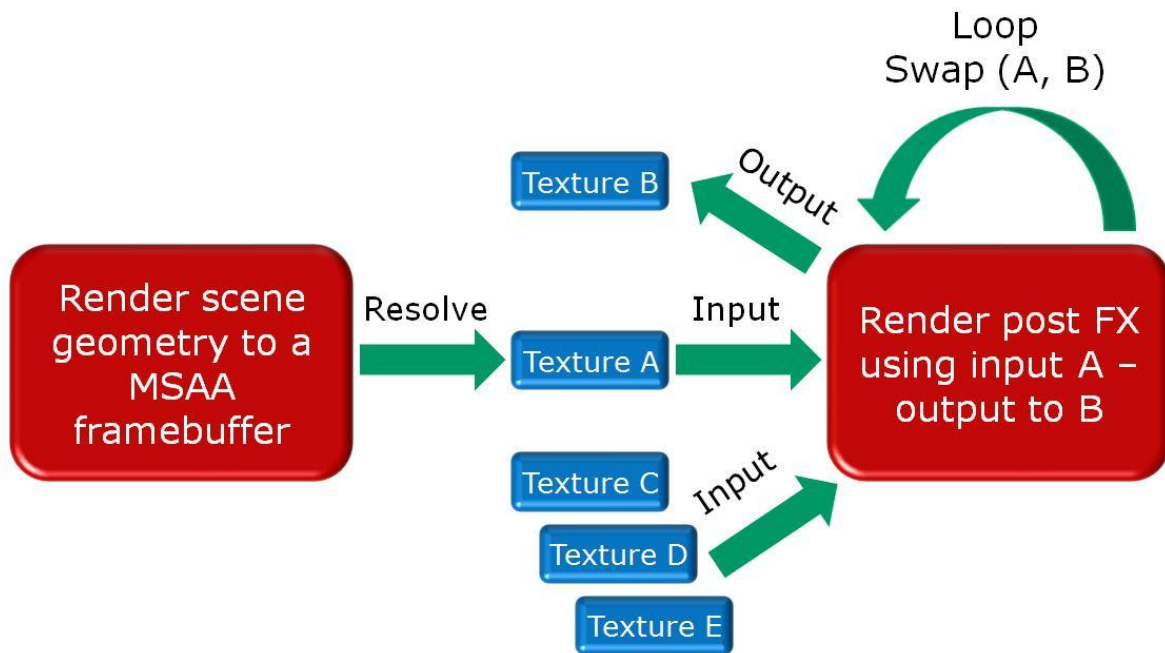


Figure 3 (Fixed Render Loop)

To add a real world example to this discussion, the following sequence of passes was uncovered during the analysis of a recently released PC title:

- 1) Render the geometry pass into the main MSAA render target M
- 2) Resolve M into a non-MSAA render target A
- 3) Render A on to M using a full-screen quad
- 4) Resolve M into A
- 5) Render water to M
- 6) Resolve M into A for further post-processing

It is fairly obvious from an initial glance at this sequence, that steps 2 through 4 are totally redundant. In fact step 3 is actually harmful from a quality stand point, as it destroys the subsample color information. Clearly it is possible to jump directly from step 1 to 5, having removed no less than two resolve operations, while maintaining the subsample color information.

So why would the developer fail to spot this? The answer lies in the fact that modern engines are highly object oriented, and that several developers are making changes to the rendering code over time. Apparently step 3 was originally a valid post processing effect, and when it was changed, it effectively became just a copy operation which made steps 2 and 3 redundant. The resolve in step 4 was triggered because M was accidentally added as an input to step 5.

As you can see it is very easy to introduce redundant resolves into the rendering pipeline. It always pays to be on top of the various passes carried out during a frame, and is generally good practice to regularly inspect PIX dumps for unexpected behavior.

Harmful Resolves

It is common for deferred rendering techniques to store information such as depth, position, normal, velocity and material ID to an intermediate render target. If this is carried out in MSAA mode, then the data would need to be resolved before being put to use later in the frame. The problem here is that the fixed function resolve operation will simply perform an average of the subsamples. This is very unlikely to yield the developer's intended result, and will most probably result in graphical artifacts.

Let us consider the case where material ID's are to be resolved. I think we would all have to agree that averaging material ID's is never going to make any sense, and that performing such an operation would, in a worst case scenario, produce invalid ID's. So how should we deal with this kind of data, when a standard fixed function resolve, is clearly not the way to go?

In DX10 it is possible to write a pixel shader that can read the subsamples of an input texture. In the case of a deferred lighting technique, it would then be possible to accumulate all lighting calculations on each subsample, and then finally average the results. In this way the shader has effectively performed a custom resolve. DX10.1 capable hardware removes a further limitation by allowing access to the subsamples of the depth buffer, which can eliminate the need for a separate depth pass.

Another prominent example of a technique that suffers from using fixed function resolved data is non-linear tone mapping. The only correct way to perform tone mapping in a multi-sampling context is to tone map every subsample using a shader based custom resolve. Figures 4 to 7 clearly show the quality difference between a fixed function and custom resolve operation, especially when edges of high contrast are considered.



Figure 4 (Fixed Function Resolve)



Figure 5 (Custom Resolve)



Figure 6 (Fixed Function Resolve - Zoom In)



Figure 7 (Custom Resolve - Zoom In)

In DX9 it is not possible to do this, so it may be that the resulting artifacts have to be tolerated, although it should be said that the implementation of explicit super sampling could achieve similar results. For performance reasons it may be necessary to carry out post-processing with data produced by a harmful resolve, though this should be kept to a minimum.

Call to Action

It is very important to appreciate that a resolve is not a free operation, in fact it is a decidedly expensive procedure, and should therefore be kept to a minimum. Keep in mind that most resolves are either redundant or harmful. To avoid redundancy, remember to resolve the main MSAA render target as early as possible, and then work in non-MSAA mode for post processing effects. Write shader based custom resolves, to properly deal with high quality post processing and deferred rendering techniques. Remember that it's easy to over look what is really happening among the various rendering passes, so regular analysis is essential to resolving your resolves!

Feedback

We would welcome your feedback on any aspects of this paper, as well as any recommendations you may have for how we can better support developers with regard to this topic.

Please send your feedback to: jon.story@amd.com or holger.gruen@amd.com



Advanced Micro Devices
One AMD Place
P.O. Box 3453
Sunnyvale, CA 94088-3453

www.amd.com
<http://ati.amd.com/developer>