[**GDC follow-up: Screenspace reflections filtering and up-sampling**](https://bartwronski.com/2014/03/23/gdc-follow-up-screenspace-reflections-filtering-and-up-sampling/)

Posted on [March 23, 2014](https://bartwronski.com/2014/03/23/gdc-follow-up-screenspace-reflections-filtering-and-up-sampling/) by [bartwronski](https://bartwronski.com/author/bartwronski/)

After GDC I’ve had some great questions and discussions about techniques we’ve used to filter and upsample the screenspace reflections to avoid flickering and edge artifacts. Special thanks here go to Angelo Pesce, who convinced me that our variation of weighting the up-sampling and filtering technique is not obvious and worth describing.

Reasons for filtering

As I mentioned in my presentation, there were four reasons to blur the screenspace reflections:

* Simulating different BRDF specular lobe for surfaces of different roughness – if they are rougher, reflections should appear very blurry (wide BRDF lobe).
* Filling holes from missed rays. Screenspace reflections are very approximate technique that relies on screenspace depth and colour information that very rarely represents properly the scene geometric complexity. Therefore some rays will miss objects and you will have some holes in your reflection buffer.
* Fighting aliasing and flickering. Quite obvious one – and lowpass filter will help a bit.
* Upsampling half-resolution information. When raytracing in half-resolution, all the previous problem become even more exaggerated, especially on edges of geometry. We had to do something to fight them.

[](https://bartwronski.files.wordpress.com/2014/03/ss_reflection_blur_radius.jpg)

Filtering radius difference on rough and smooth surfaces

Up-sampling technique

First I’m going to describe our up-sampling technique, as it is very simple.

For up-sampling we tried first industry standard depth-edge aware bilateral up-sampling. It worked just fine for geometric and normal edges, but we faced different problem. Due to different gloss of various areas of same surface, blur kernel was also different (blur was also in half resolution).

We observed a quite serious problem on important part of our environments – water puddles that stayed after the rain. We have seen typical jaggy edge and low-res artifacts on a border of very glossy and reflective water puddle surface (around it there was quite rough ground / dirt).

As roughness affects also reflection / indirect specular visibility / intensity, effect was even more pronounced. Therefore I have tried adding a second up-sample weight based on comparison of surface reflectivity (combination of gloss based specular response and Fresnel) and it worked just perfectly!

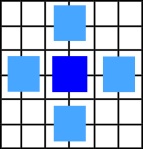
It could be used even on its own in our case – but it may not be true in case of other games – we used it to save some ALU / BW. For us it discriminated very well general geometric edges (characters / buildings had very different gloss values than the ground), but probably not every game or scene could do it.

Filtering technique

We spend really lots of time on getting filtering of the reflections buffer right – probably more than on the actual raytracing code or optimizations.

As kind of pre-pass and help for it, we did cross-style slight blur during downsampling of our color buffer for the screenspace reflections.

Similar technique was suggested by[Mittring for bloom [1]](http://advances.realtimerendering.com/s2012/Epic/The%20Technology%20Behind%20the%20Elemental%20Demo%2016x9.pptx) and in general is very useful to fight various aliasing problems when using half-res colour buffers and I recommend it to anyone trying to use half-res color buffer for anything.

[](https://bartwronski.files.wordpress.com/2014/03/downsampling.jpg)

Downsampling filtering / blur pattern

Then later we performed a weighted separable blur for performance / quality reasons – to get properly blurred screenspace reflections for very rough surfaces blurring radius must be huge! Using separable blur with varying radius is in general improper (special thanks to Stephen Hill for reminding it to me) as the second pass could catch some wrong blurred samples (with a different blur radius in orthogonal direction), but worked in our case – as surface glossiness was quite coherent on screen, we didn’t have any mixed patterns that would break it.

Also screen-space blur is in general improper approximation of convolution of multiple rays agains the BRDF kernel, but as also both Crytek and Guerrilla Games mentioned in their GDC presentations [2] [3], it looks quite convincing.

Filtering radius

Filtering radius depended just on two factors. Quite obvious one is surface roughness. We ignored the effect of cone widening with distance – I knew it would be “physically wrong” but from my experiments and comparing against real multiple ray traced reference convolved with BRDF – visual difference was significant only on rough, but flat surfaces (like polished floors) and vert close to the reflected surface – with normal maps and on organic and natural surfaces or bigger distances it wasn’t noticeable as something “wrong”. Therefore for performance / simplicity reasons we have ignored it.

At first I have tried basing the blur radius on some approximation of a fixed-distance cone and surface glossiness (similar to the way of biasing mips of pre filtered cubemaps). However, artists complained about the lack of control and as our rendering was not physically based, I just gave them blur bias and scale control based on the gloss.

There was a second filtering factor – when there was a “hole” in our reflections buffer, we artificially increased the blurring radius, even for shiny surfaces. Therefore we applied form of push-pull filter.

* Push – we tried to “push” further away proper ray-tracing information by weighting it higher
* Pull – pixels that lacked proper information looked for it in larger neighbourhood.

It was better to fill the holes and look for proper samples in the neighbourhood than have ugly flickering image.

Filtering weight

Our filtering weight depended on just two factors:

* Alpha of sample being read – if it was a hole or properly ray traced sample.
* Gaussian function.

Reason for the first one was again to ignore missing samples and do pulling of proper information from pixel neighbourhood. We didn’t weight hole samples to 0.0f – AFAIR it was 0.3f. The reason here was to still get some proper fadeout of reflections and to have lower screen-space reflections weight in “problematic” areas to blend them out to fall-back cube-map information.

Finally, the Gaussian function isn’t 100% accurate approximation of Blinn-Phong BRDF shape, but smoothed out the result nicely. Furthermore and as I mentioned previously, whole no screen-space blur is a *proper* approximation of 3D multiple ray convolution with BRDF – but can look *properly* to human brain.

Thing worth noting here is that our filter didn’t use depth difference in weighting function – but on depth discontinuities there was already no reflection information, so we didn’t see any visible artifacts from reflection leaking. Guerilla Games presentation by Michal Valient [3] also mentioned doing regular full blur – without any depth or edge-aware logic.

References

[1] Mittring, [“The Technology behind the Unreal Engine 4 Elemental Demo”](http://advances.realtimerendering.com/s2012/Epic/The%20Technology%20Behind%20the%20Elemental%20Demo%2016x9.pptx)

[2] Schulz, [“Moving to the Next Generation: The Rendering Technology of Ryse”](http://schedule.gdconf.com/session-id/826333)

[3] Valient, [“Taking Killzone Shadow Fall Image Quality into the Next Generation”](http://schedule.gdconf.com/session-id/827807)

**Share this:**