see also
http://loopit.dk/publications/

Banding in Games: A Noisy Rant

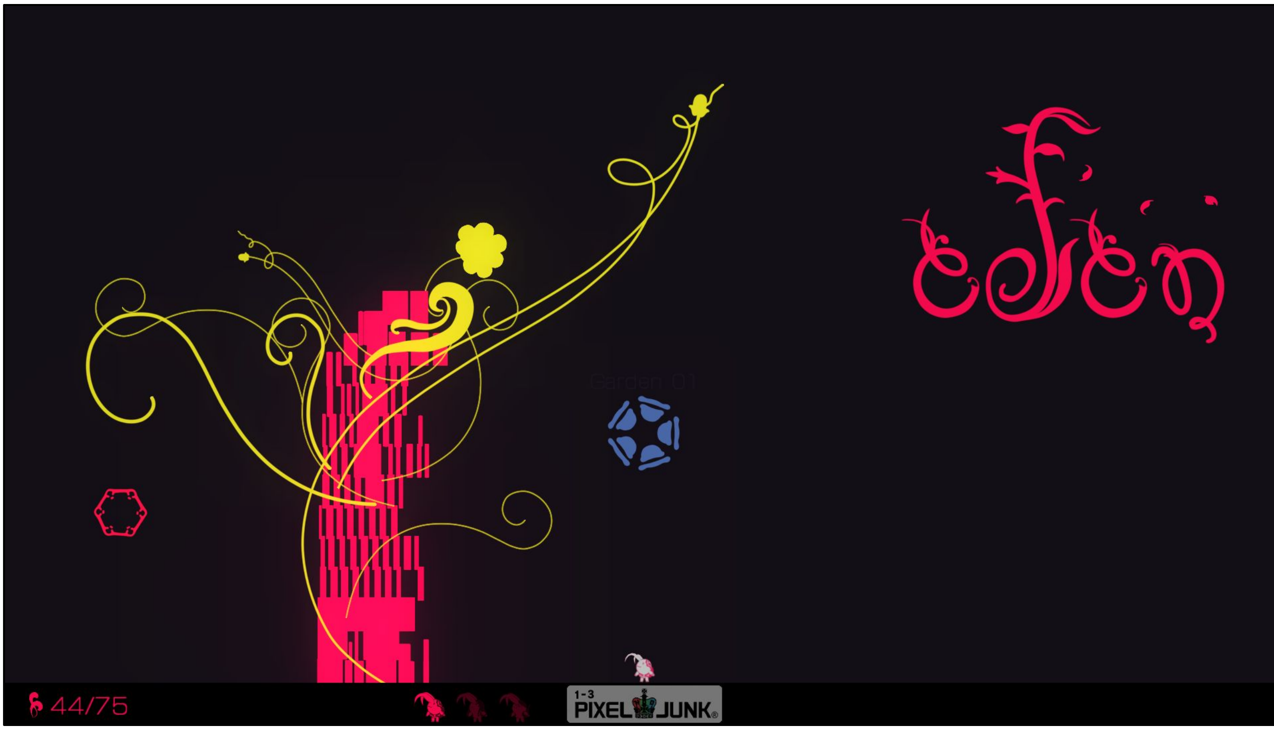
(revision 5)

Mikkel Gjøl, Playdead
@pixelmager

online link to this presentation
http://loopit.dk/banding_in_games.pdf

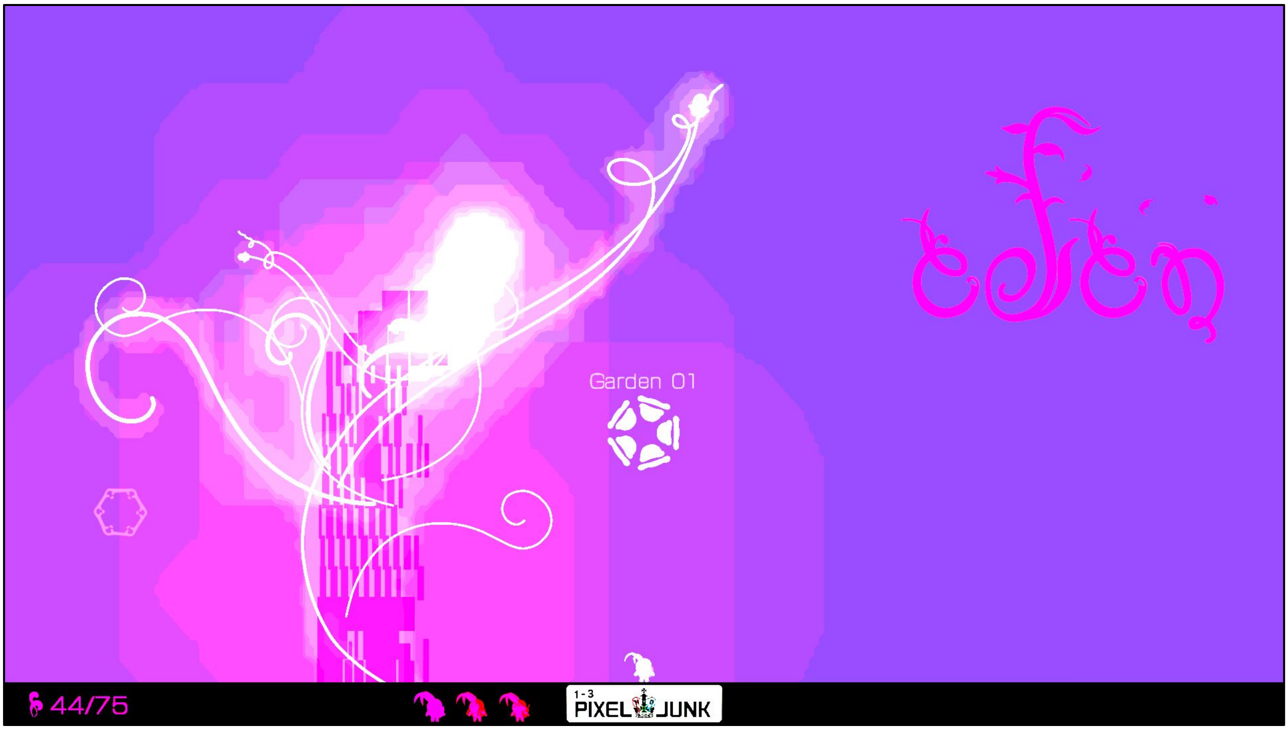
online link to this presentation http://loopit.dk/banding_in_games.pdf

- not going to go into too much detail about the theoretical background, try to give intuitive understanding and focus on issues



First a couple of example to motivate that this is actually still an issue in games.

amazing game, pixeljunk eden (if you haven’t played it, you should literally leave the room and do so now!)
http://pixeljunk.jp/library/Eden/



so what we’re looking at here are the “abrupt changes between shades of the same colour”

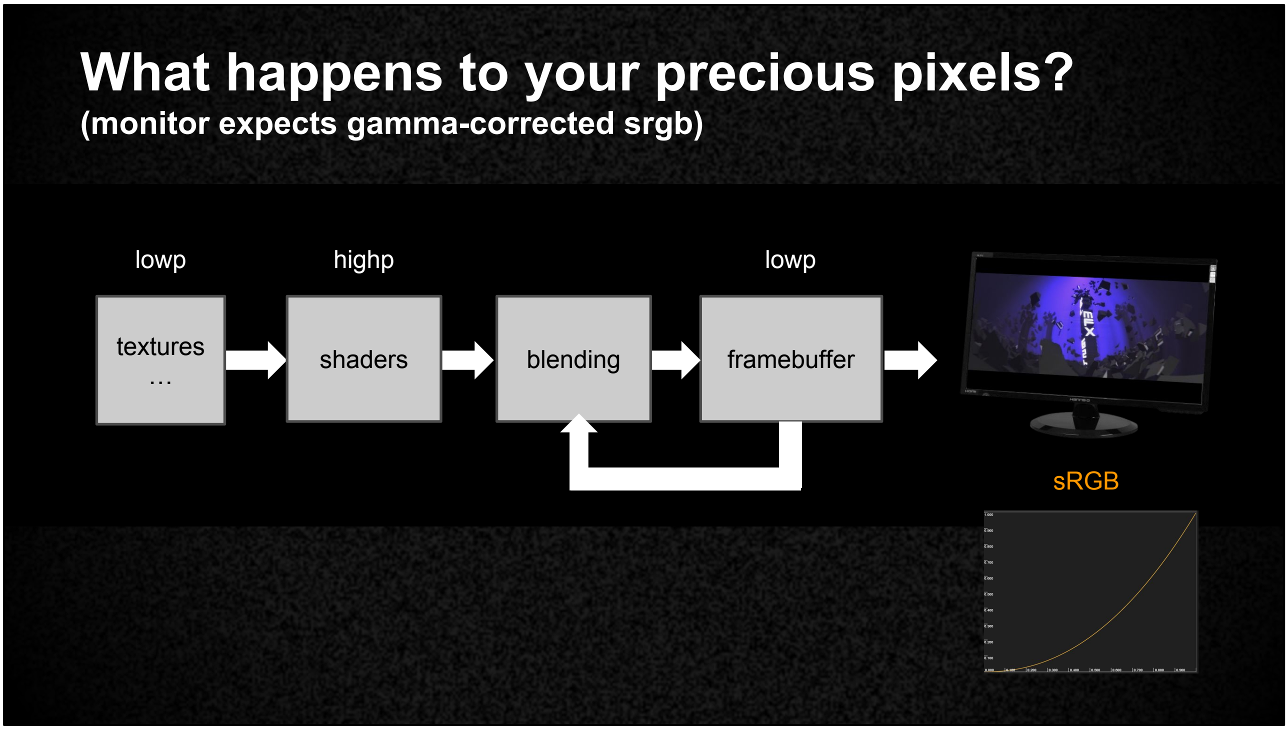
Skyrim, amazing do-whatever-you-want game… which means most will remember skyrim looking something like this
http://www.elderscrolls.com/skyrim

I was working on banding at the time I played it, so I remember the menus

...so I remember it looking mostly like this :)

...this is another beautiful indie-game, kentucky route zero
http://kentuckyroutezero.com/

which on most high-quality monitors will have quite noticeable banding (but almost get away with it due to their graphical style)



the roughest pipeline-overview you will ever see
things to note:
- monitor expects an input that is “roughly” srgb - in order to give the monitor srgb without terrible loss of precision, we either need >8bit precision, or need for all other stages to be aware that color-values end up in sRGB space… there are many interesting ways in which this can fail.
note also that blending reads from the framebuffer

see e.g. http://www.opengl.org/registry/specs/ARB/framebuffer_sRGB.txt

Gamma-incorrectness exposes banding

- so just render gamma-correct...

Might not be so easy...

- sRGB may not be available and functional
- ...under all circumstances, RTT, blending etc
- ...on all your target platforms
- ...including mobile and old PC-hardware

(doable on most platforms, but there may be good reasons why your particular engine is not fully gamma-correct)

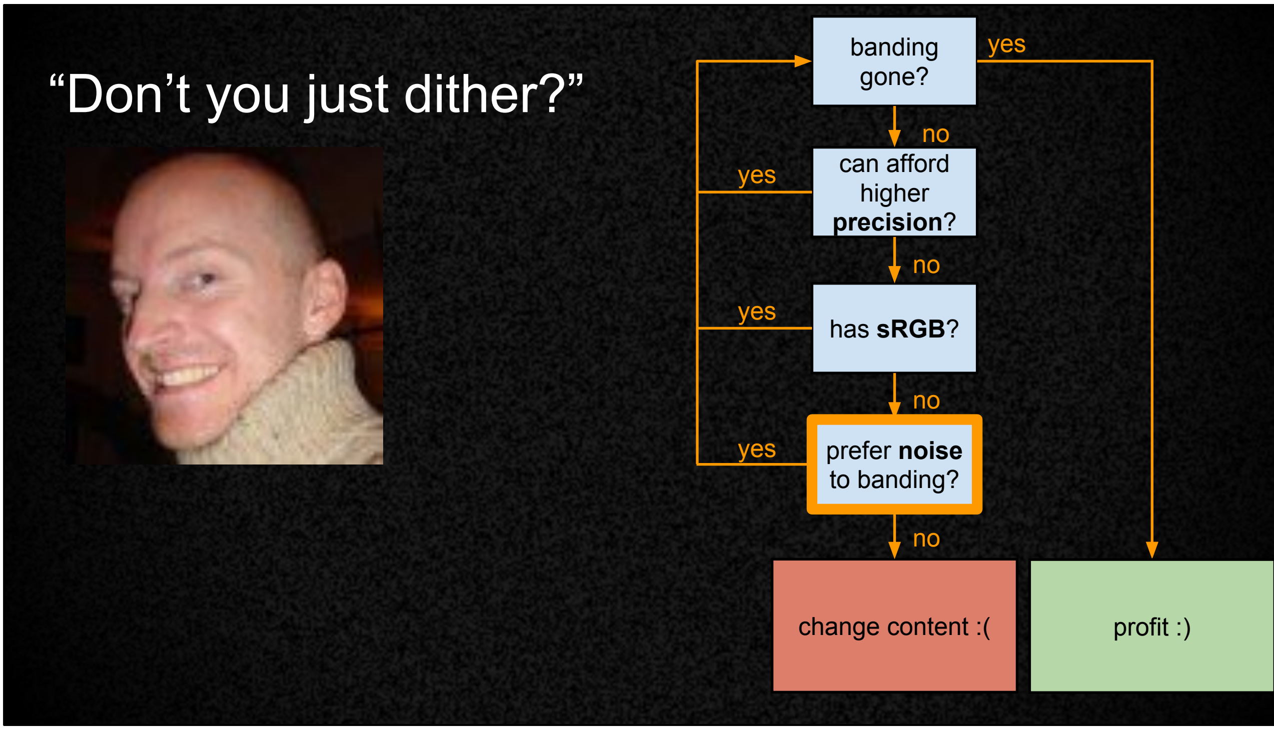
When incorrect the monitor will still correct the signal as if the image was in gamma-space, exposes all the horrible bands in the dark areas
If you use sRGB correctly, you're doing pretty well - you will generally hardly notice banding (though dark areas remain)
If you are not on a platform where it's readily available, or you want to get rid of the last issues, the rest of this presentation is for you

rtt => render to texture

...just wanted to go over this, so we know where we are in the pipeline, and there are good reasons a multiplatform engine could potentially have issues… mostly a last-gen / mobile issue though.
(thanks http://www.keepcalm-o-matic.co.uk/)

- someone else already did that way better:
http://download.microsoft.com/download/b/5/5/b55d67ff-f1cb-4174-836a-bbf8f84fb7e1/Picture%20Perfect%20-%20Gamma%20Through%20the%20Rendering%20Pipeline.zip
(mirror: http://loopit.dk/Gamma_Through_the_Rendering_Pipeline.zip)

also http://www.poynton.com/GammaFAQ.html



This is the happy face of Thomas Rued, a man of experience who knows his pixels!
...when all else fails, add noise

Areas affected by color-banding

(...anything that renders gradients on screen)

Lighting

Fog

Alpha Blending

- additive (multipass rendering)
- subtractive
- multiplicative (lerp)
- ...creative :p

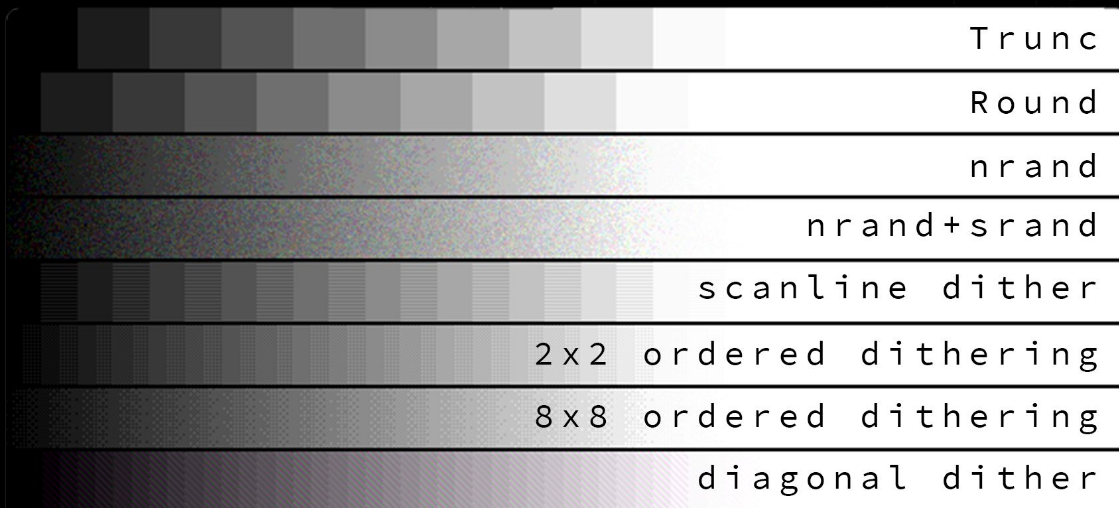
Particles

Posteffects, e.g. glow, aa, screen fades

(we will not go into undersampling and compression issues)

this talk is about the specific issues related to banding, where they occur and how to remedy them
..most noticeable on anything creating gradients: fog, falloffs, glow, particle splotches, from lights etc.
Noisy ‘signals’ (e.g. with a texture) “hides” most banding.

Dithering: The art of noise

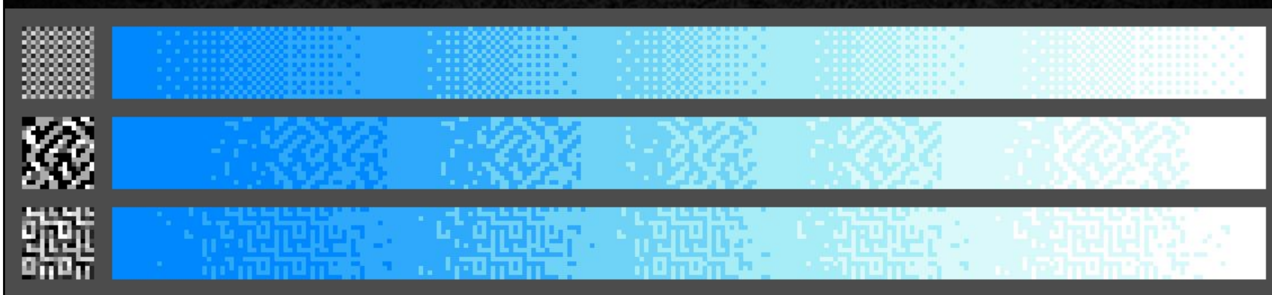


a variety of ways to do dithering, ...they all have to do with how the eye blurs out small changes (or integrates over an area). Squint a bit at the above image and you can barely tell the difference...
https://www.shadertoy.com/view/MslGR8
...you should of course dither r,g,b separately for improved luminance

Dither Spatially (take advantage of eye accumulating multiple pixels)
Dither Temporally (take advantage of eye accumulating over time) => change dither pattern per frame

Scanline dither: x2 steps
2x2 ordered: x4 steps
8x8 ordered: x8(?) steps

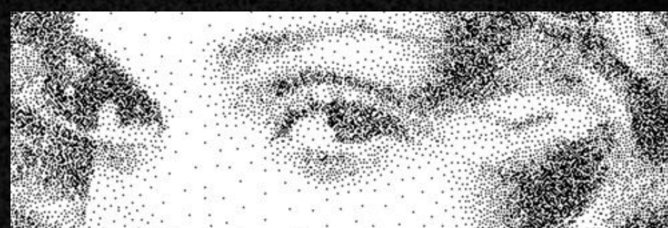
Ordered dithering on pixeldrugs
(gamedevs… sheesh!)



...or other weirdness if you want to be ‘creative’ ;)
(pixelartists)
- this is taken from a tutorial in drawing pixel-graphics with photoshop
http://danfessler.com/blog/hd-index-painting-in-photoshop

Much better dithering schemes exist

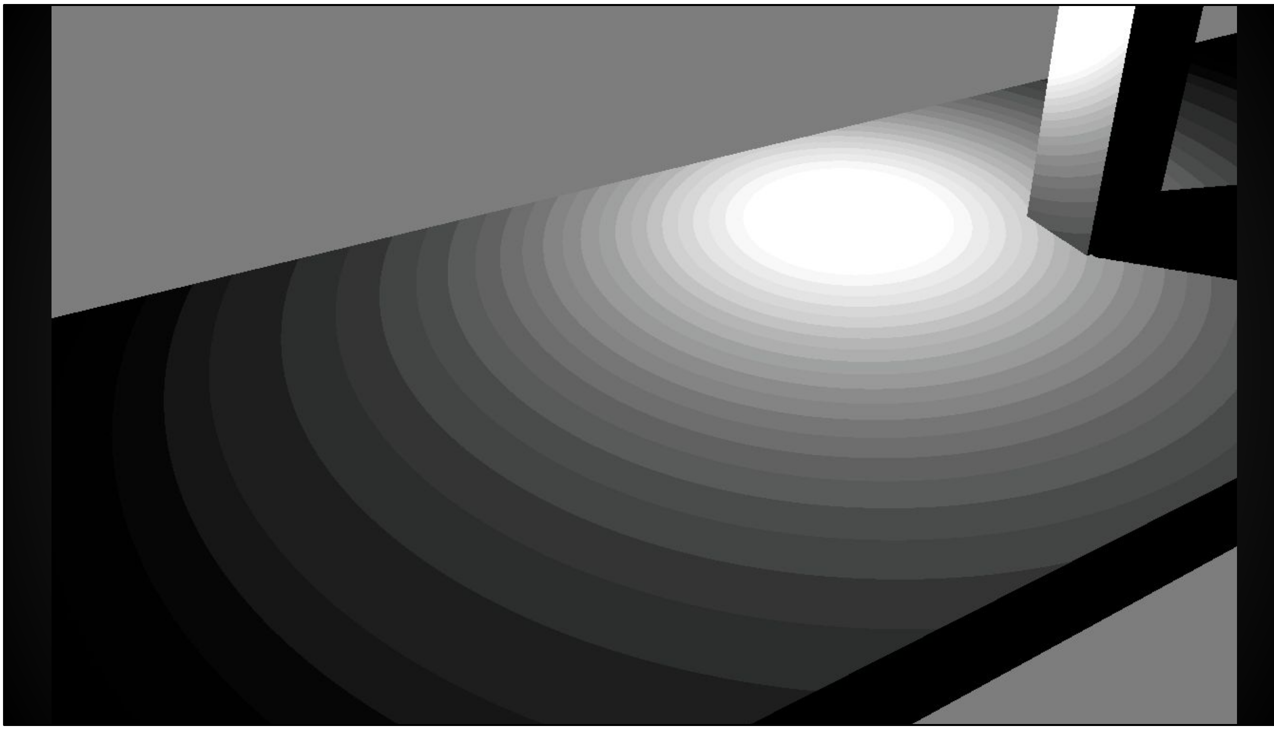
- Floyd Steinberg and related methods
- Most are error-diffusion based, don't map well to GPUs
- uniform noise is good enough for Pixar
  (and we don't mind a bit of grain in our camera to get less sterile images)



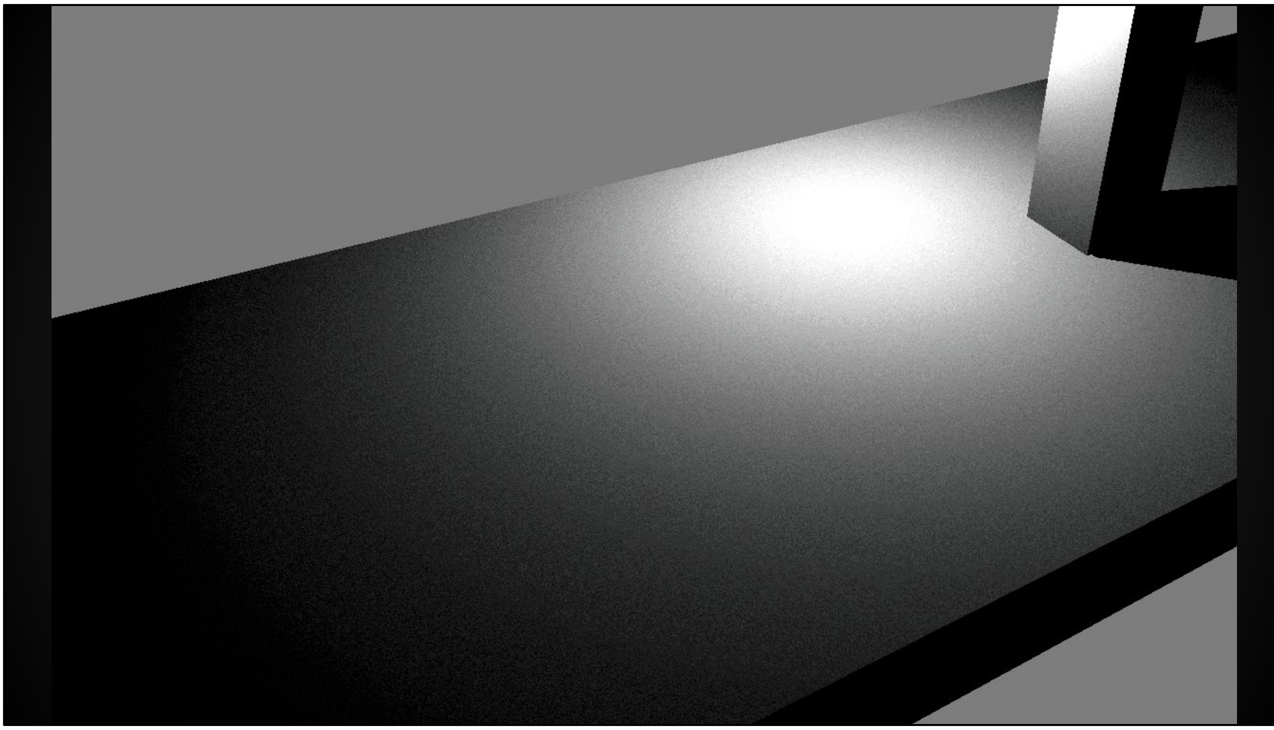
error diffusion based methods rely on knowledge of the surrounding area, something that doesn't map well to the GPU, which prefer doing a lot of unrelated things in parallel

http://www.realtimerendering.com/blog/2011-color-and-imaging-conference-part-ii-courses-a/

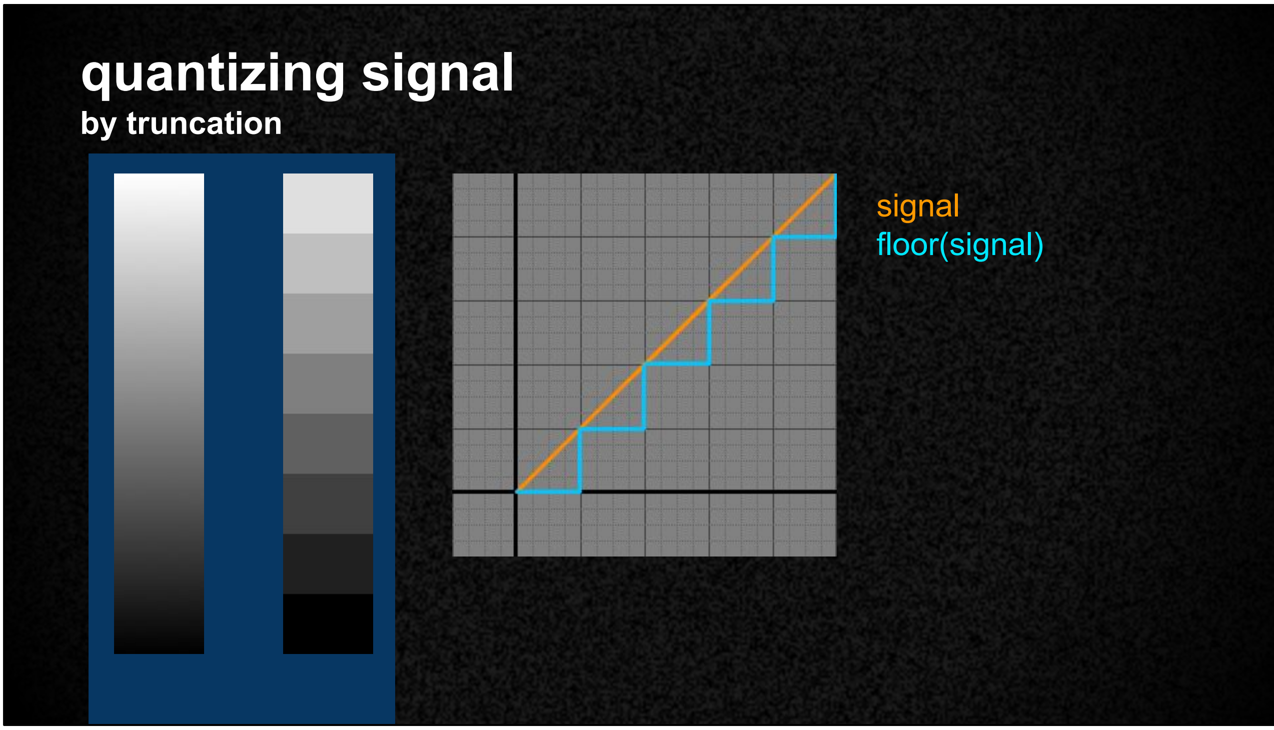
http://www.joesfer.com/?p=149
http://www.joesfer.com/?p=108



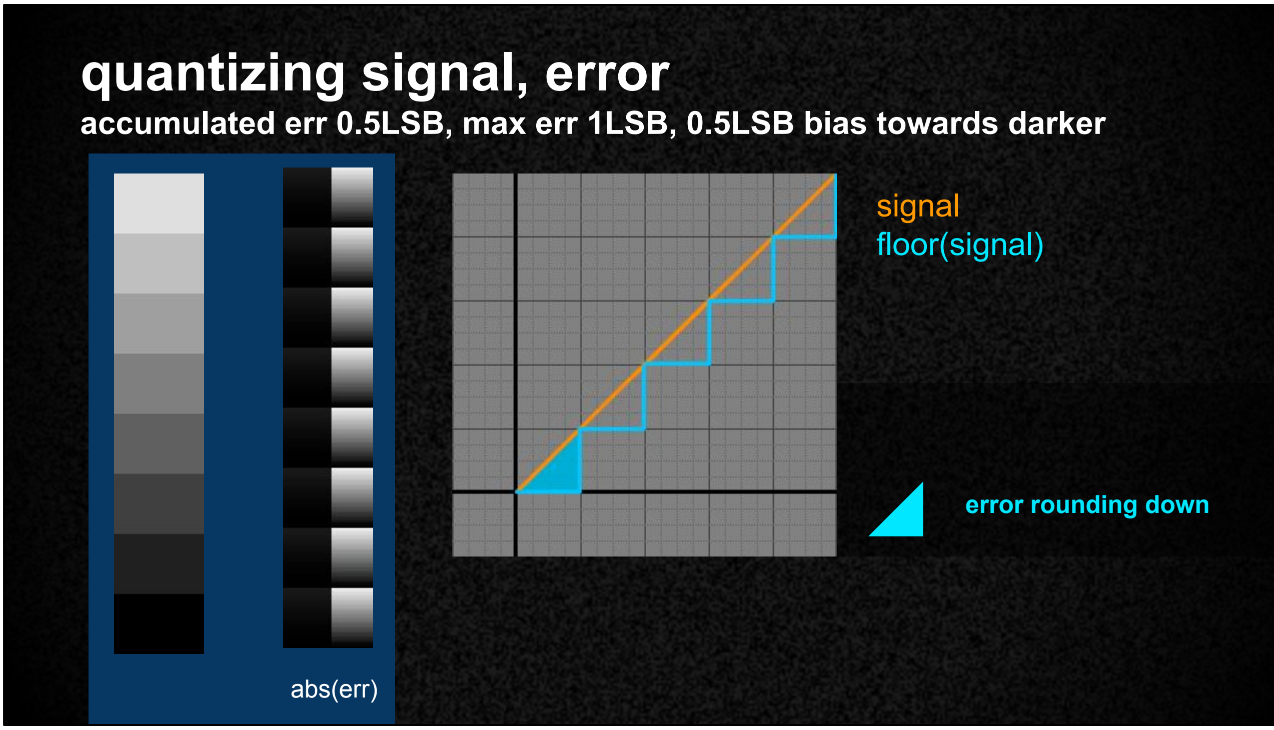
unity lightbuffer - already in a logarithmic colorspace to enable HDR lighting, so no sense in additionally using srgb



modified Internal-PrePassLighting.shader to do (monochrome) random dithering
...in order to not create discontinuities at deferred light-edge (light fades out to edge), we subtract from the intensity instead of adding…
..so slightly darker, but otherwise uniform (we vary noise over time as well, so with a bad monitor, and/or eyesight, the result is quite good)

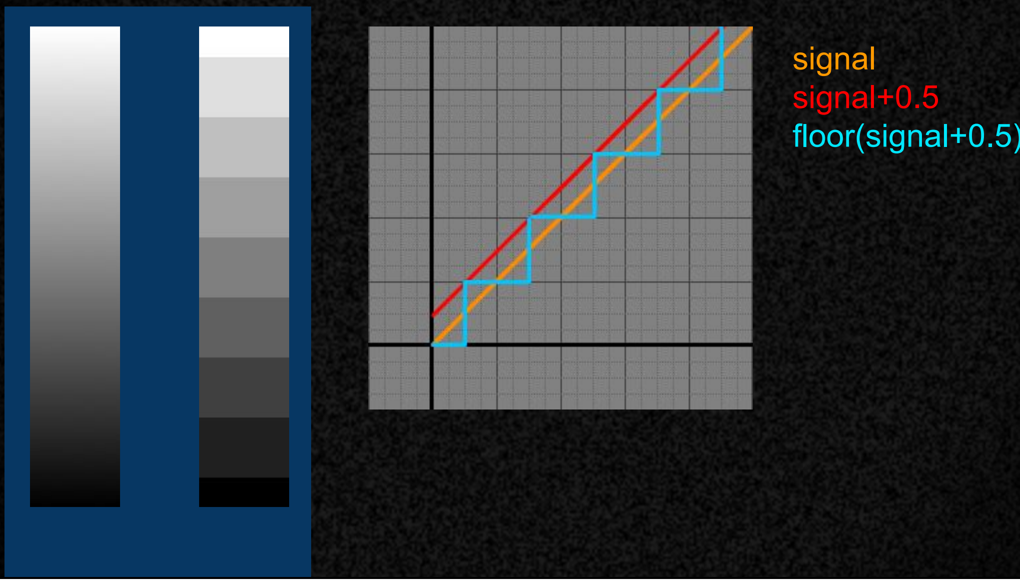


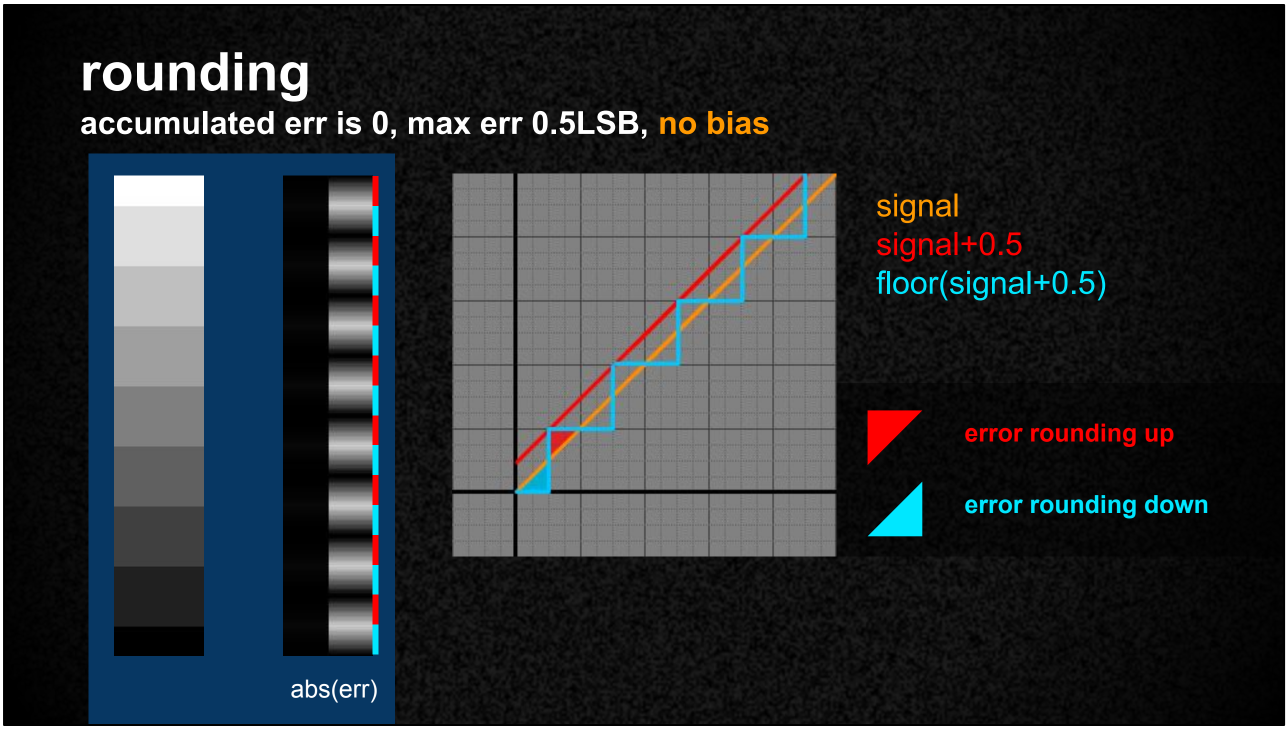
doing simple quantization (by truncating) is always darker than intended signal (by 0.5bit on avg)



max error of 1 a Least-Signicant-Bit
0.5 LSB error on avg
0.5 LSB bias towards darker result

rounding



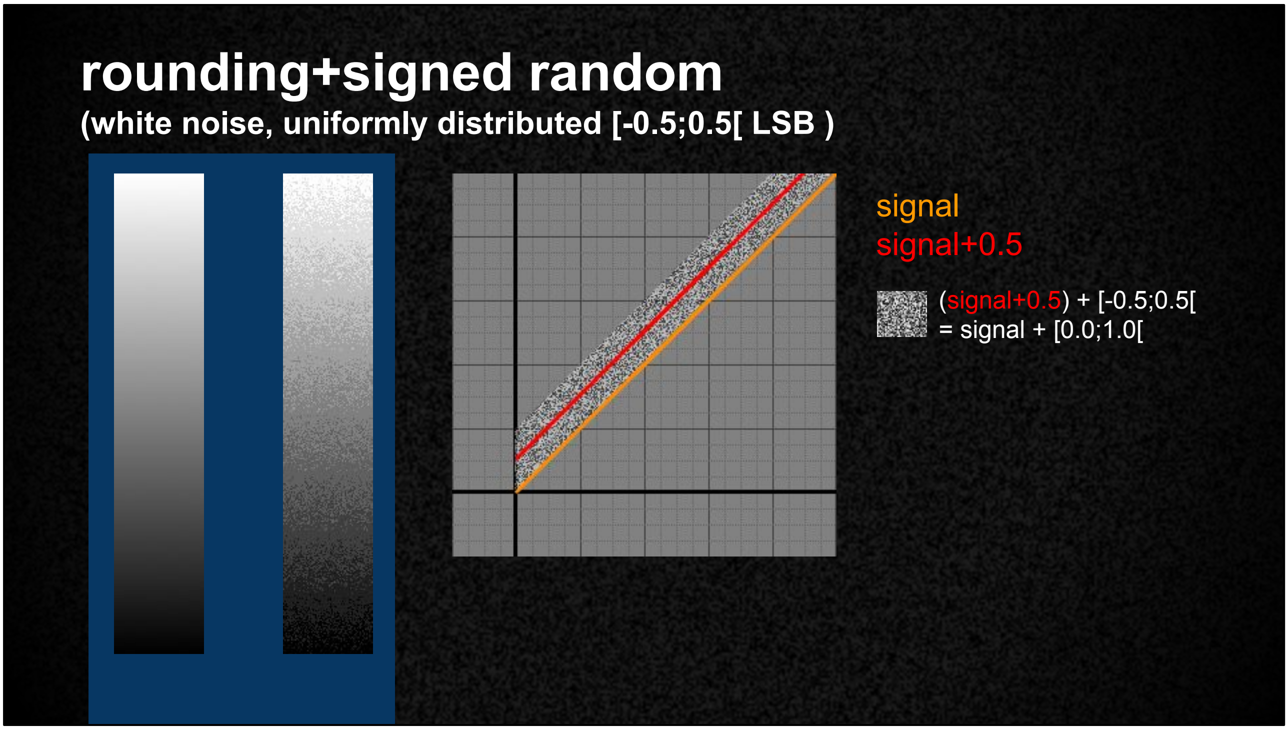


Though the average error is 0.25LSB, the accumulated error is 0 (intuitively as area rounding up == area rounding down), but absolute error at any value is as much as 0.5*LSB

Signal does not lose 'energi' as opposed to trunc

Smaller err, smaller maxerr (half before),

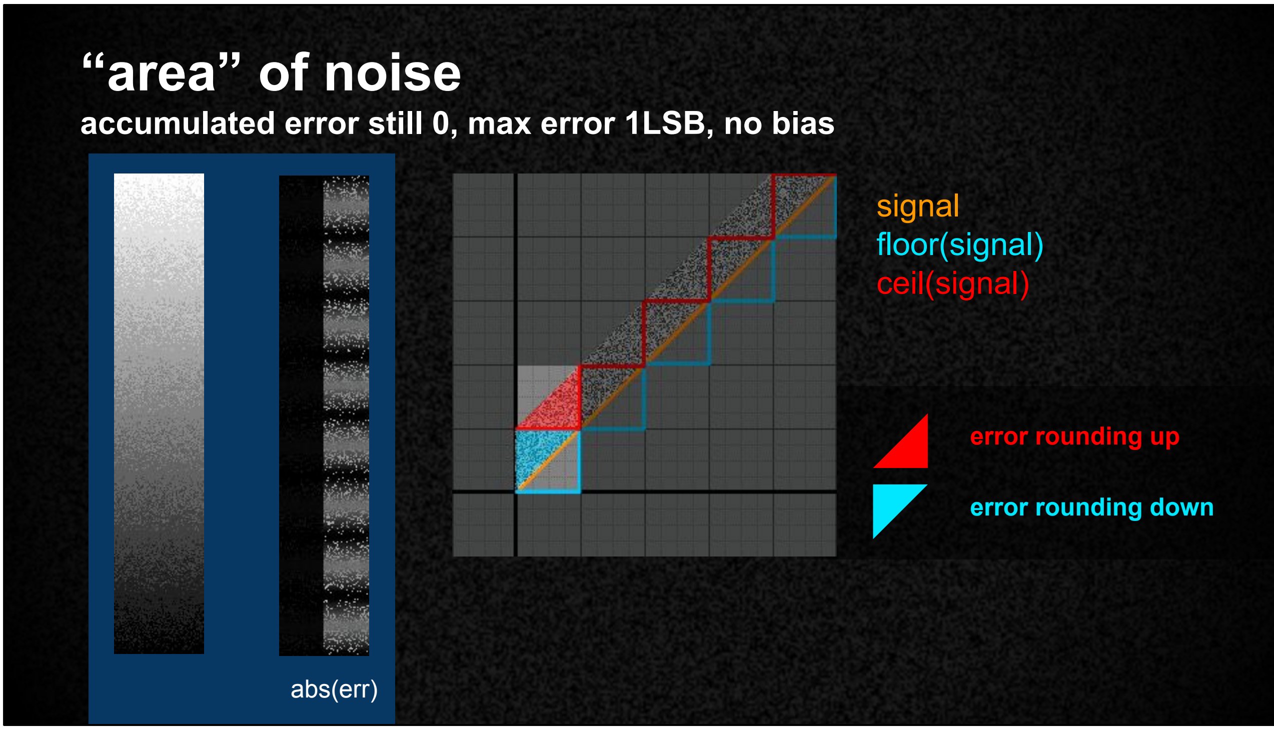
now at least on average, it's the same signal as the original. It is unbiased.



adding a uniform signed noise means we're still not biasing the signal - sometimes brighter, sometimes darker, so on average it does neither...

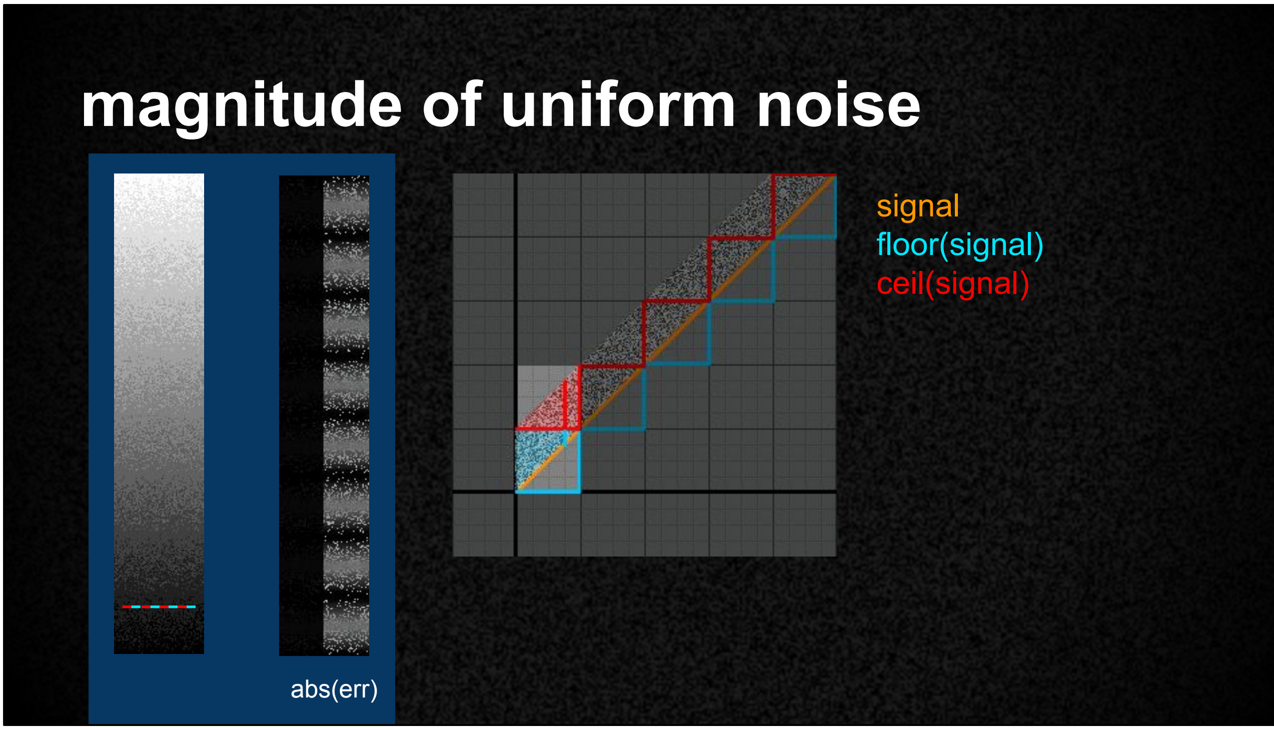
...property still valid that avg error is 0

This is exactly the same as adding a normalized [0;1[to the signal



max err of course now larger as we just added noise (but still: no bias)
intuitively: on average quantizing as much to value above as value below, so end up with the same signal… on average

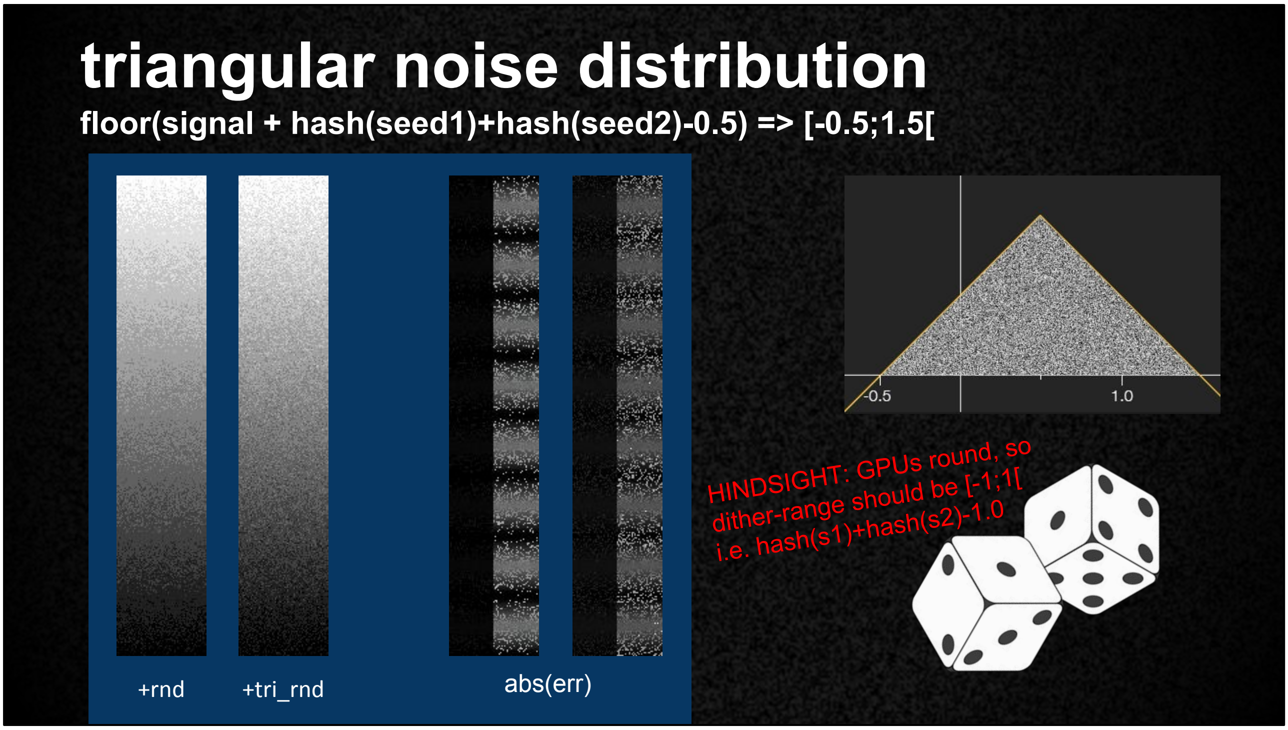
- same as before, on average the positive/negative noise will cancel out, and we get the original signal on average.
Error still varies (intuitively: When signal crosses the truncated value, the resulting error is 0)
...we get slightly worse max error than from just rounding.. but….



...if we look at the accumulated error for a single value (integrating over the red/blue stippled line), the error will now cancel out as well as for entire signal, resulting in the original signal for any arbitrary single value
This is a property of the noise being uniformly distributed

intuitively, since the noise-distribution is uniform, when integrating across the line shown the length of the line corresponds to the probability that the value will either round up or down
...integrating floor(f) / ceil(f) across this line, we'll end up at the signal

...noise is uniformly distributed, but the resulting visual noise is NOT uniformly distributed - almost no noise near “correct” values
Really no reason to add noise where signal -is- it's truncated value



Supposedly a PDF with better characteristics, commonly used for audio-dithering. Effectively adds noise in low-noise-areas, giving a more uniform noise-appearance.

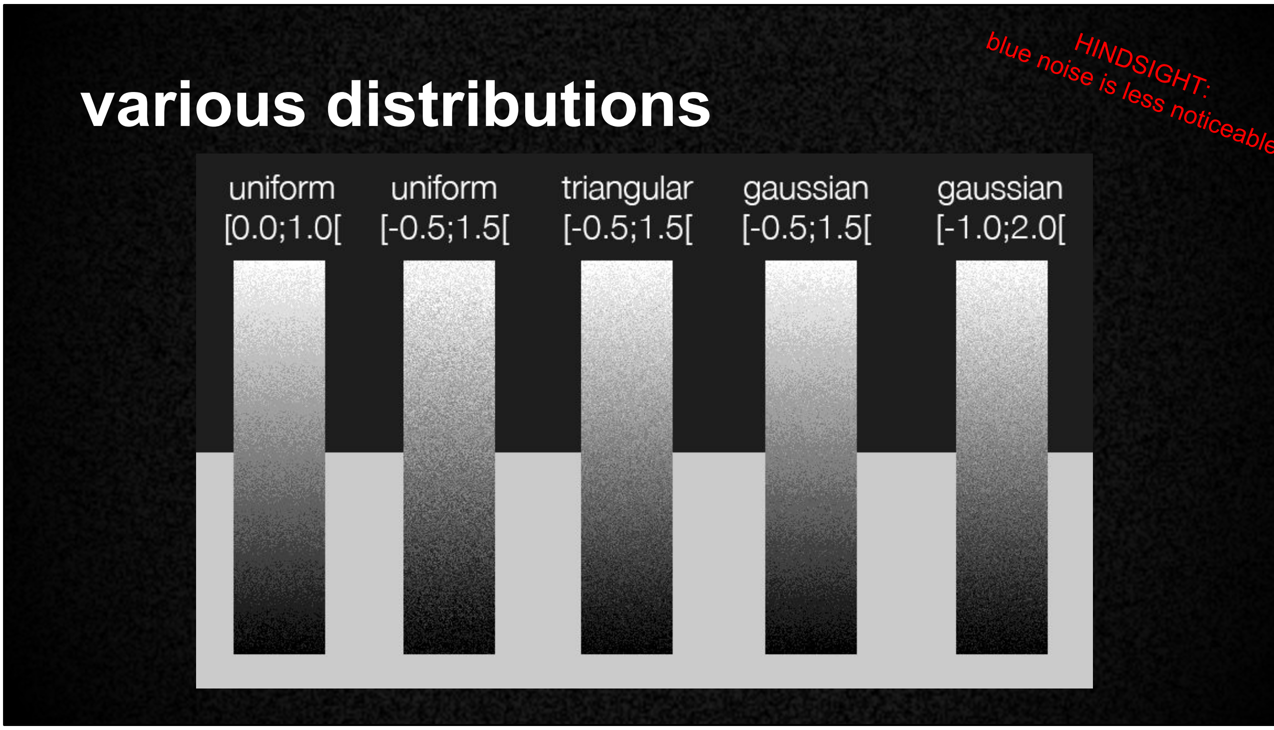
We're not currently using this, wikipedia suggests using it if the result is to be "worked on further"...

Visually uniform noise in gradient (pretty much just adds noise to low-error areas)

HINDSIGHT: The phenomenon is called "noise modulation". This turns out to be very important to achieve a banding-free look, and we relied heavily on it in the shipped product.

HINDSIGHT: instead of calculating two hashes from scratch, they can be combined to make them a lot faster. Also, remapping a single hash to have a triangular PDF may be faster, see https://www.shadertoy.com/view/4t2SDh

references
http://en.wikipedia.org/wiki/Dither
http://www.ece.rochester.edu/courses/ECE472/resources/Papers/Lipshitz_1992.pdf
https://www.shadertoy.com/view/4ssXRX

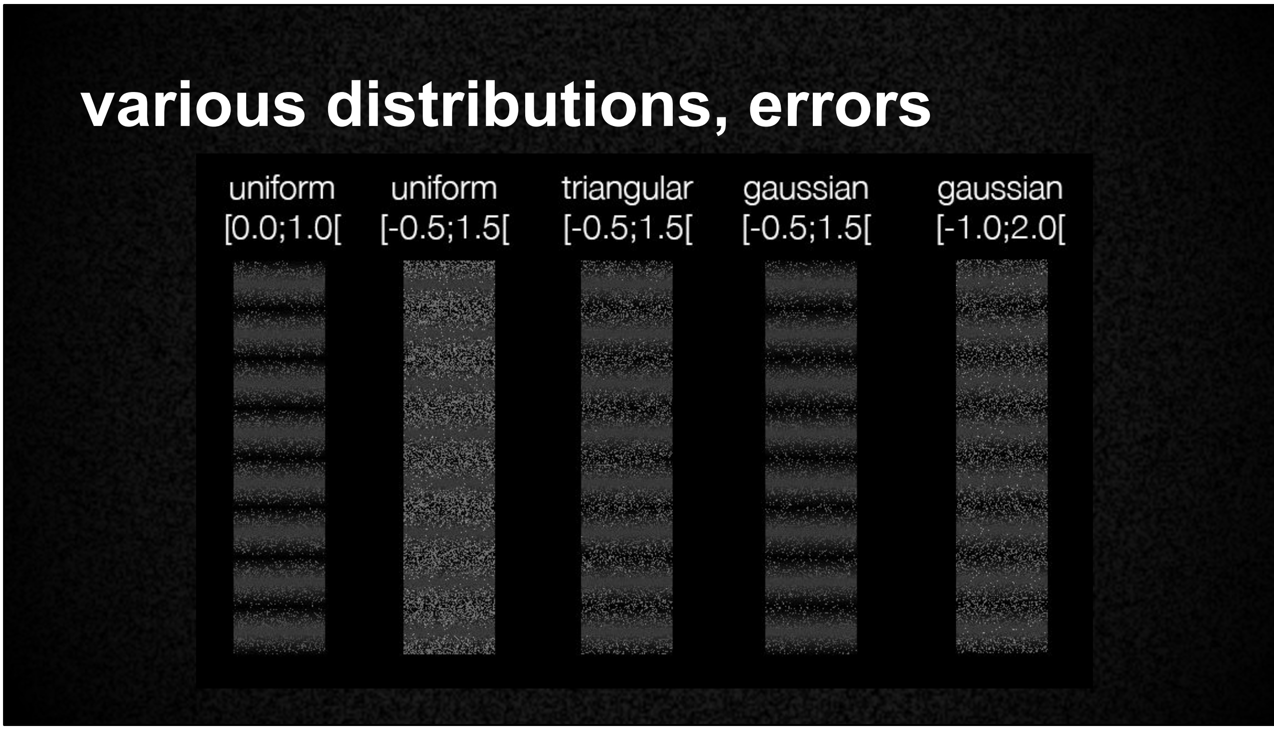


“Awesome, if triangular is good, gaussian must be even betterer!” (spoiler: don’t do it)

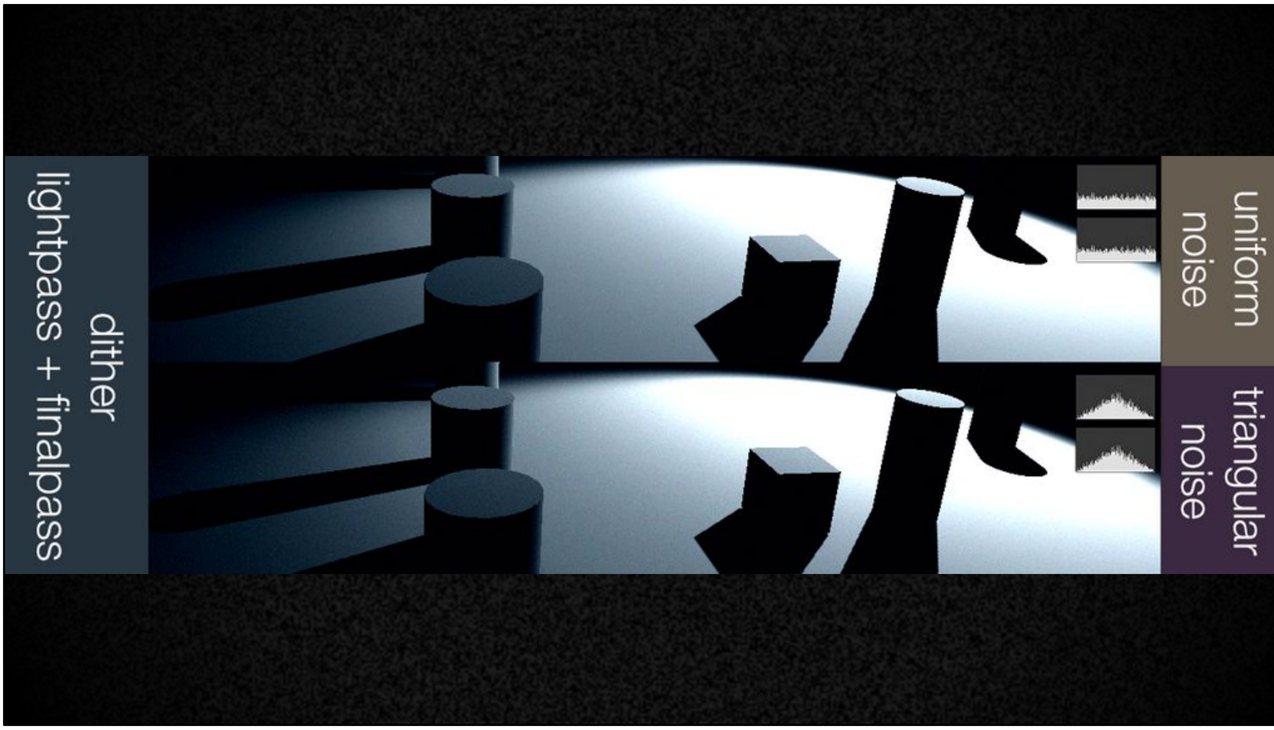
From left to right

- A uniform distribution in the interval [0;1[creates areas of little to no noise, giving the impression of a smooth signal, but non-uniform noise.
- expanding the noise to [-0.5;1.5[creates areas of too much noise (in the “overlapping” regions)
- Switching to a triangular distribution [-0.5;1.5[gives a nice uniform distribution (the “overlapping” noise-regions accumulate to 1)
- Using a gaussian distribution again creates areas with too little noise
- using a gaussian-pdf in range [-1.0;2.0[appears very similar to triangular, but with more noticeable noise

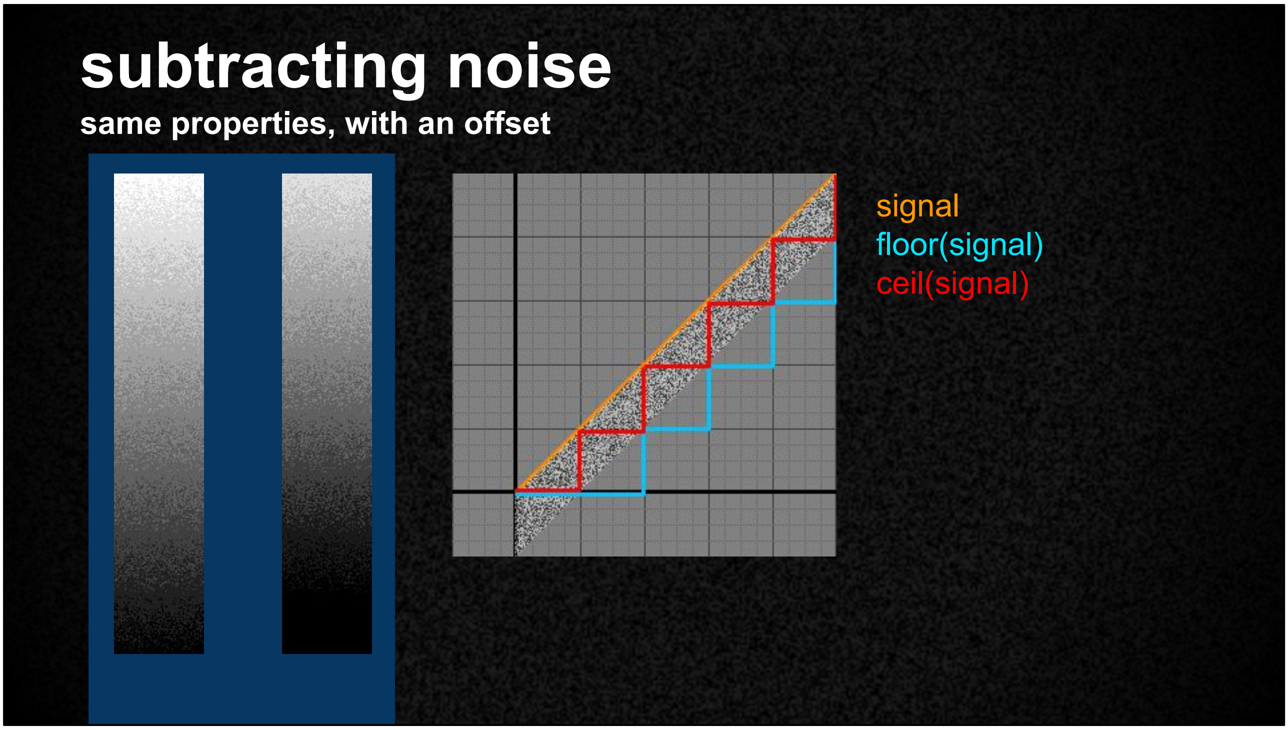
Empiric conclusion: Use a triangular distribution in the interval [-0.5;1.5[
HINDSIGHT: The reason a triangular distribution is “enough”, is that it does not exhibit “noise modulation”. Neither do gaussians, but this is the main benefit.
HINDSIGHT: GPUs round, so dither-range should be [-1;1[
HINDSIGHT: Using high-pass filtered noise can give even better results, as the noise is less noticeable. This is what is called “noise shaping” in audio-dithering.



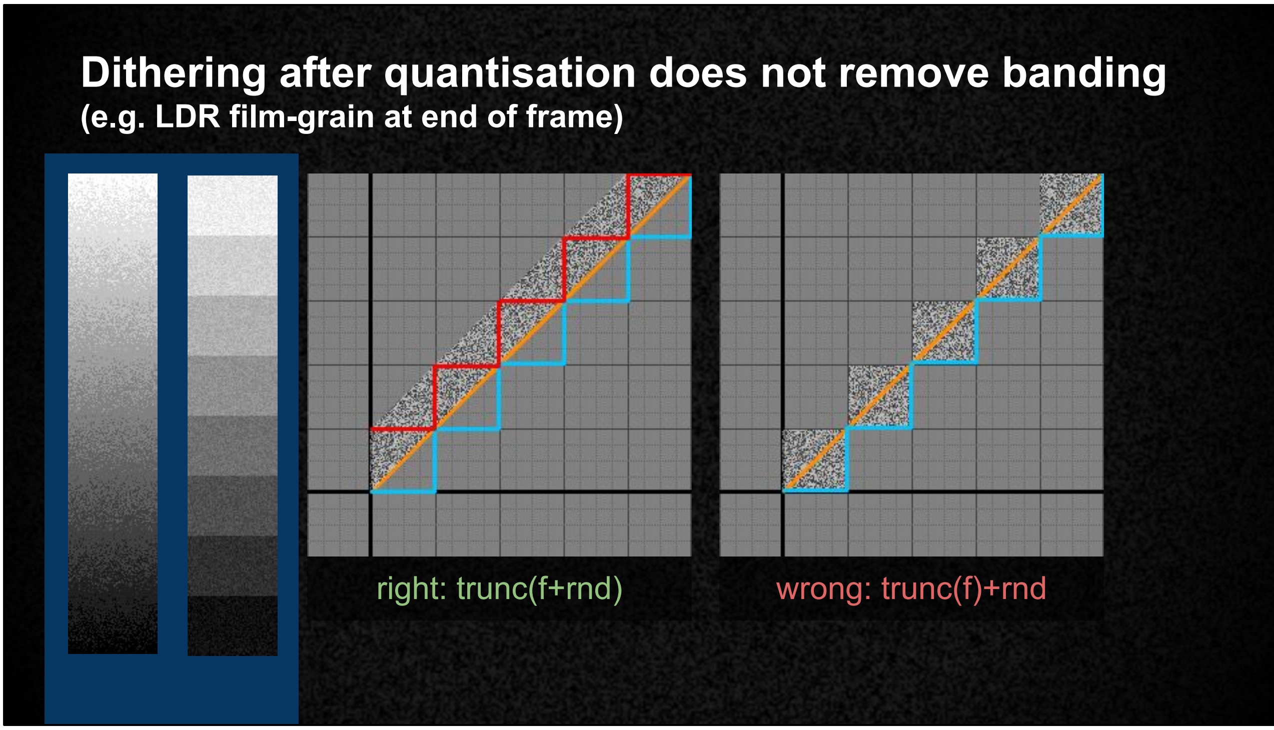
HINDSIGHT: this shows the absolute value of the error, abs(err), which makes it hard to see that positive/negative errors “cancel each out” in the noisy areas. Remapping the actual error to a [0;1] range gives a much more accurate depiction of the error.



Example of dithering using triangular noise [-0.5;1.0[vs. uniform noise [0;1] - triangular noise makes the noise more evident, but also more uniform - no areas with little noise.
...the effect is certainly subtle, but better is better :) (and also more noisy :p)



just a quick note that subtracting noise of course results in a signal with the same properties, but offset



“lets just add grain at end of frame to make banding go away”
...only works if input signal is kept at high precision, so e.g. HDR-rendering and adding grain at time of tonemapping is fine…
Otherwise you are just adding noise on top of an already banding image.

the right color space

```
vec4 fragmain(vec2 fragpos) {
  vec4 outcol;
  …
  return outcol;
}
```

...spend a bit on time on this, because it’s important
colors are in linear-space in the pixelshader (or should be unless you’re messing something up)...

In essense: Dither just before quantizing.

the right color space
(linear target, dithered)

```
vec4 hash42n( vec2 seed ) { .. }

vec4 ditherRGBA( vec4 c, vec2 seed ) {
  return c + hash42n( seed ) / 255.0; //8bit
}
vec4 fragmain(vec2 fragpos) {
  vec4 outcol;
  …
  return ditherRGBA( outcol, pos );
}
```

...hash42n is a function returning a vec4 of uniformly distributed [0;1[random numbers given a vec2 as seed.
It can be ALU or a texture-lookup depending on where you have most to spare...

the right color space

(manual srgb rendertarget)

```
vec4 ditherRGBA( vec4 c, vec2 seed ) {
  return c + hash42n( seed ) / 255.0; //8bit
}
vec4 fragmain(vec2 fragpos) {
  vec4 outcol;
  …
  return ditherRGBA(lin2srgb(outcol), fragpos);
}
```

convert first, dither afterwards… dither in whatever space the color will be quantized in.

the right color space

(manual logarithmic buffer, e.g. unity lightbuffer)

```
vec4 ditherRGBA( vec4 c, vec2 seed ) {
  return c + hash42n( seed ) / 255.0; //8bit
}
vec4 fragmain(vec2 fragpos) {
  vec4 outcol;
  …
  return ditherRGBA(exp2(-outcol), fragpos);
}
```

again, do it as close to quantization as possible - jittering the light-value does not help...

the right color space
(autoconverting srgb rendertarget)

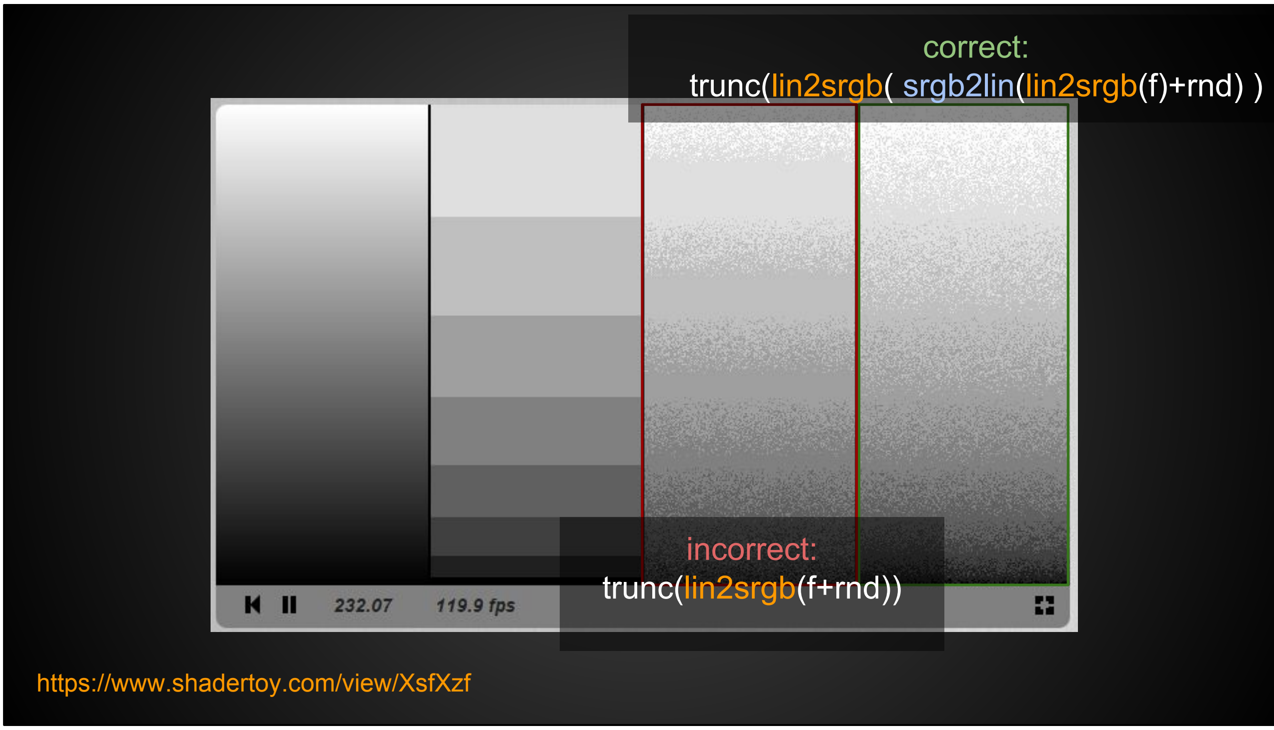
```
vec4 ditherRGBA(vec4 c,vec2 s){return c+hash42n(s)/255.0;}
vec4 fragmain(vec2 fragpos) {
  vec4 outcol;
  …
  return srgb2lin(ditherRGBA(lin2srgb(outcol), fragpos));
}
```

...need to dither in whatever space the color will be quantized in.

the right color space
(autoconverting srgb rendertarget)

```
vec4 ditherRGBA(vec4 c,vec2 s){return c+hash42n(s)/255.0;}
vec4 fragmain(vec2 fragpos) {
  vec4 outcol;
  …
  return srgb2lin(ditherRGBA(lin2srgb(outcol),fragpos));
}
//doesn’t strictly need to be all that accurate for
this...
vec4 srgb2lin(vec4 c) { return vec4(c.rgb*c.rgb, c.a); }
vec4 lin2srgb(vec4 c) { return vec4(sqrt(c.rgb), c.a); }
```

(sqrt(c)+n)^2 == c + 2*n*sqrt(c) + n^2
...but pow(sqrt(c) + rnd/255.0, 2.0); is probably faster :p
… see also Timothy Lottes’: https://www.shadertoy.com/view/Md2XWw



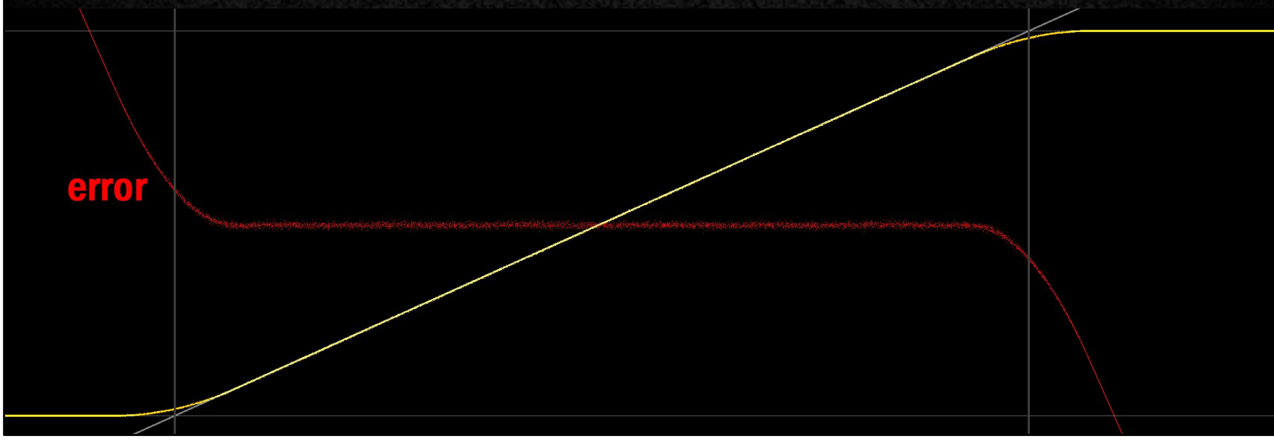
correct:
trunc(lin2srgb(srgb2lin(lin2srgb(f)+rnd))
232.07
119.9 fps
incorrect:
trunc(lin2srgb(f+rnd))
https://www.shadertoy.com/view/XsfXzf

menu of steam big picture (best thing that has happened to pc-gaming for a long time)
- has noise added, but interestingly in what appears to be the wrong color-space.

Triangular dithering at boundaries

HINDSIGHT

It turns out, that due to clamping, the boundaries around black/white (0/1) become wrong when using 2LSB triangular dithering.



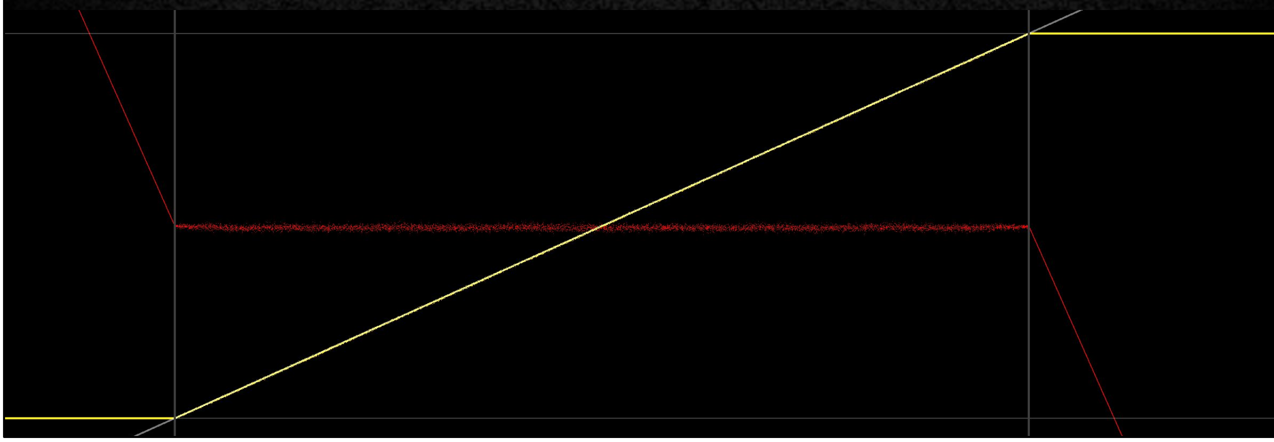
https://computergraphics.stackexchange.com/questions/5904/whats-a-proper-way-to-clamp-dither-noise/5952#5952

intuition: As the noise gets clamped, clamping destroys the symmetry in the noise - and for that reason the average does not tend towards the actual signal

Triangular dithering at boundaries

HINDSIGHT

A solution is to lerp to a 1LSB uniform-PDF noise at boundaries



https://computergraphics.stackexchange.com/questions/5904/whats-a-proper-way-to-clamp-dither-noise/5952#5952

HINDSIGHT

Triangular dithering at boundaries

```
vec2 rnd = hash22(seed);

float dithertri = (rnd.x + rnd.y - 1.0); //note: symmetric, triangular dither, [-1;1[

float dithernorm = rnd.x - 0.5; //note: symmetric, uniform dither [-0.5;0.5[

float sizt_lo = clamp( v/(0.5/7.0), 0.0, 1.0 );

float sizt_hi = 1.0 - clamp( (v-6.5/7.0)/(1.0-6.5/7.0), 0.0, 1.0 );

dither = lerp( dithernorm, dithertri, min(sizt_lo, sizt_hi) );
```

https://computergraphics.stackexchange.com/questions/5904/whats-a-proper-way-to-clamp-dither-noise/5952#5952

alpha blending

blend-modes

- additive, `c = c0+c1; //dither c0,c1`
- subtractive, `c = c0-c1; //dither c0,c1`
- multiplicative, `c = c0*c1; //… :(`
- creative :p

adding / subtracting does not change the magnitude of LSB, so the dithering is fine. You still end up adding 1LSB of noise for each input, which adds up...
multiplying ruins the whole thing - no way to determine the magnitude of a LSB is after multiplication of “some value [0;1]” with a constant (accessing the destination-buffer of course is even worse)

alphablending, non-constructive rnd

(blending multiple shaders to same pixel)

```
vec4 ditherRGBA( vec4 c, vec2 seed ) {
  return c + hash42n( seed ) / 255.0; //8bit
}
vec4 fragmain(vec2 fragpos) {
  vec4 outcol;
  …
  vec2 UNIQUE_SEED = vec2(0.6849);
  return ditherRGBA( outcol, fragpos+UNIQUE_SEED );
}
```

...blending tends to deal with translucency, so
...if blending multiple objects with to the same pixel, using a different noise-value for each layer helps reduce noise.

alphablending, non-constructive rnd

(blending same shader to same pixel, particle systems)

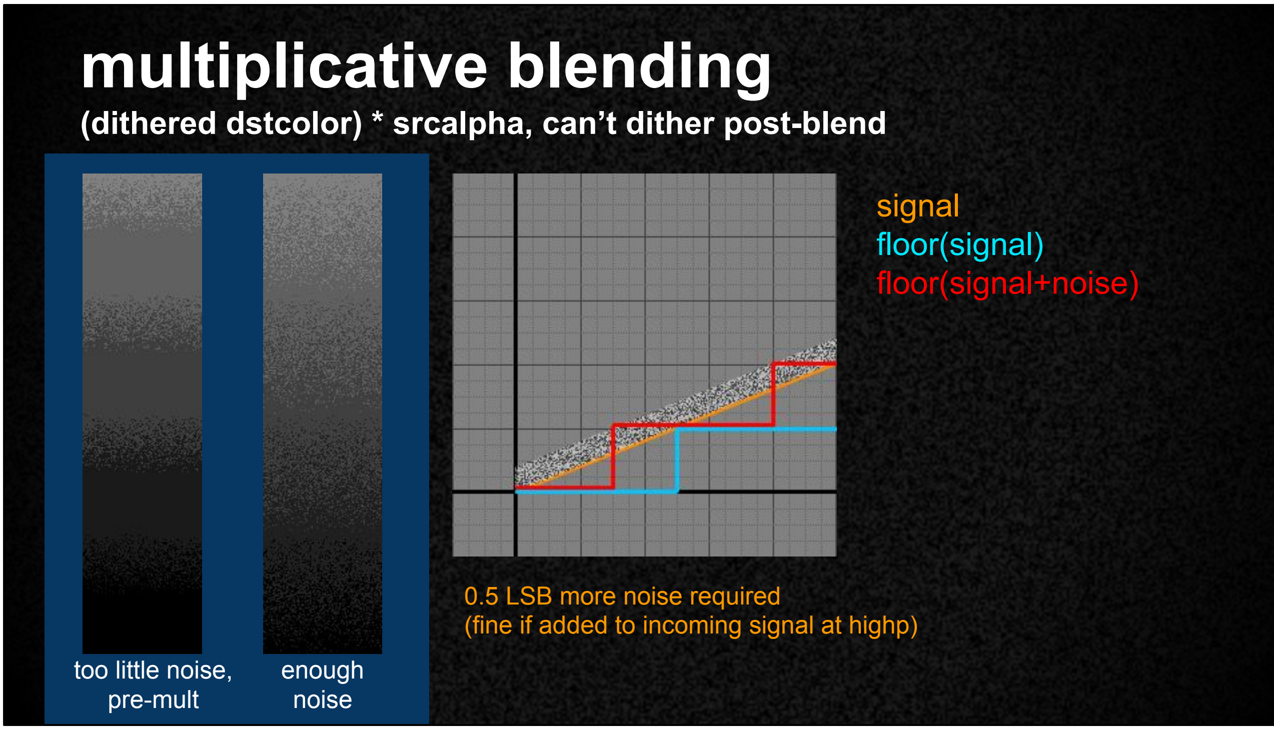
```
vec4 ditherRGBA( vec4 c, vec2 seed ) {
  return c + hash42n( seed ) / 255.0; //8bit
}
vec4 fragmain(vec2 fragpos) {
  vec4 outcol;
  …
  vec2 UNIQUE_SEED = vec2(0.6849 + vspos.z);
  return ditherRGBA( outcol, fragpos+UNIQUE_SEED );
}
```

...if blending multiple objects with the same **shader**, using a different noise-value for each layer helps reduce noise.
E.g. a particle-system with many layers could use the viewspace-z as a seed on top of screenposition.

further blending issues

Anything that multiplies something onto dstcol

- do as much in shader as possible
  - don't use blendunit for srcAlpha if you can do it
- 2pass: subtract from dst, then add color
  (doesn't band... but also doesn't look the same)
- add moar noise to dst-alpha :(
  ...helps mask problem, but not a "solution".



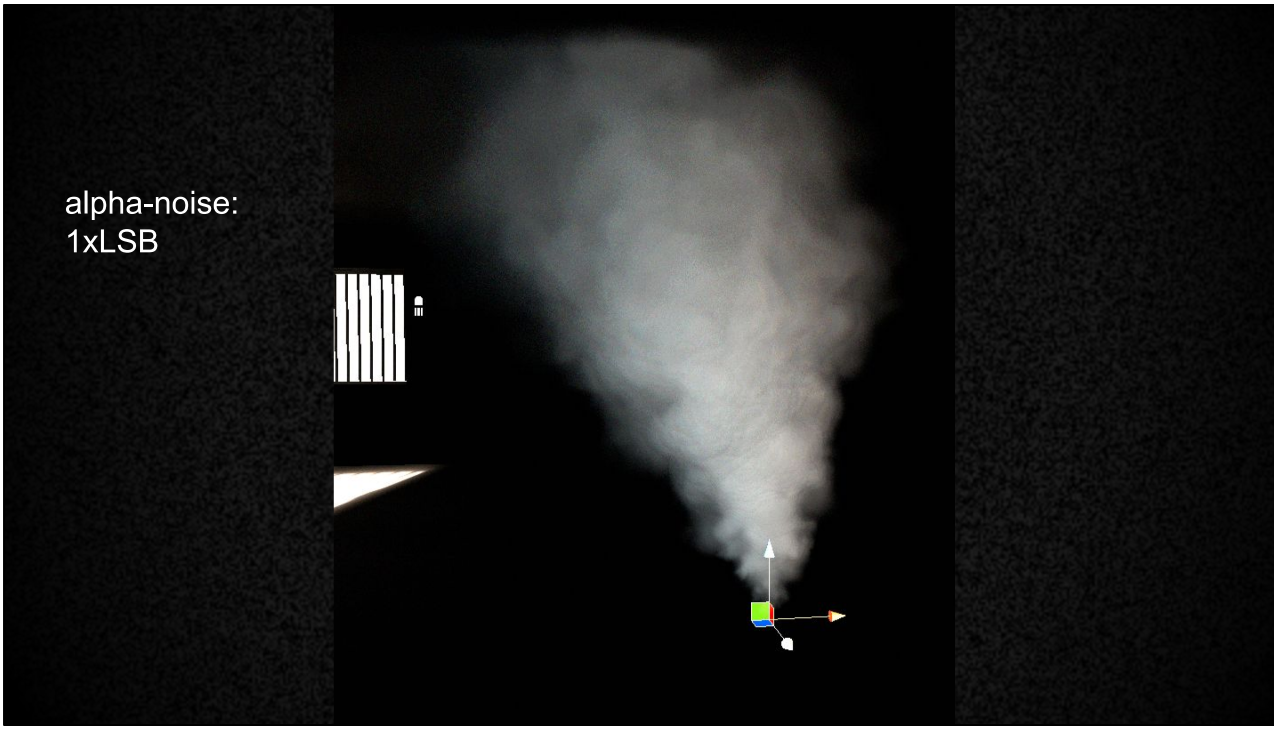
scales down noise, need to add noise again before quantization
...really, we would be fine if the noise added to src-color would just remain at high precision until after blending (more on that in a second...)

alphablending
(prefer premultiplied alpha)

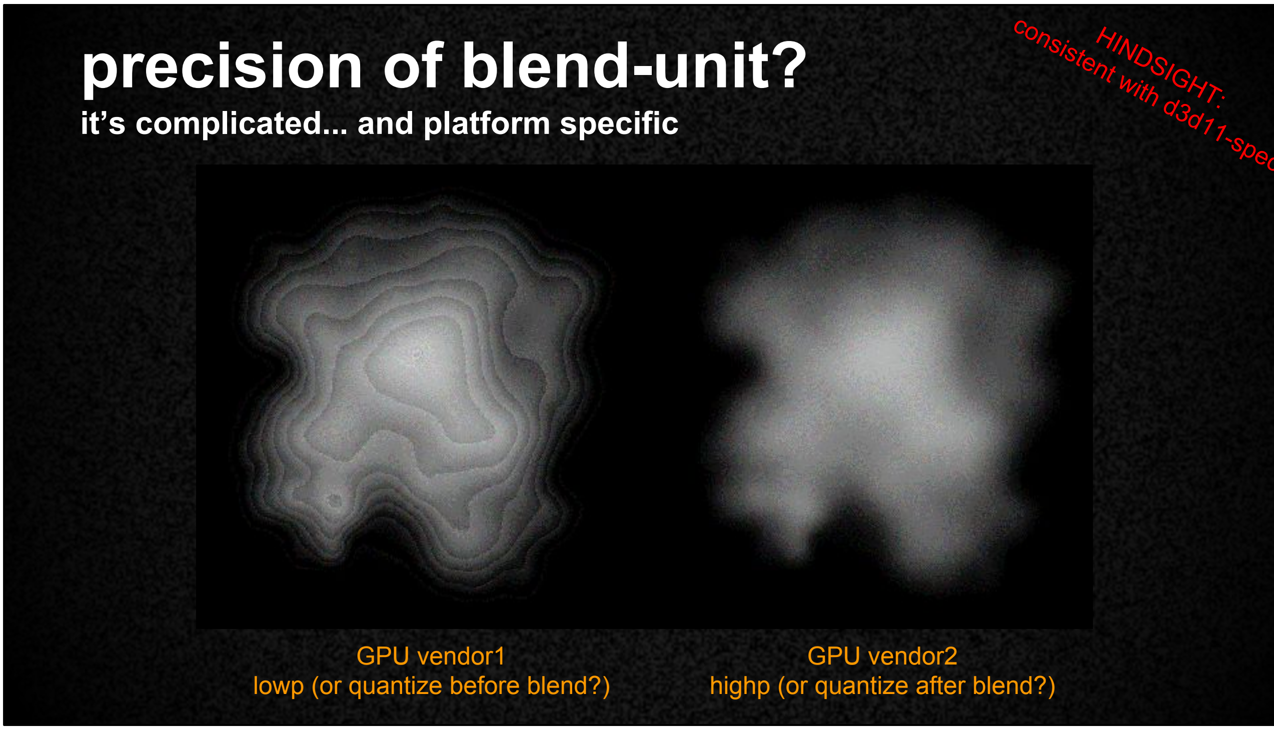
```
vec4 ditherRGBA( vec4 c, vec2 seed ) {
  return c + hash42n( seed ) / 255.0; //8bit
}
vec4 fragmain(vec2 fragpos) {
  vec4 outcol;
  …
  outcol.rgb *= outcol.a;
  return ditherRGBA( outcol, fragpos );
}
```

additive blending with SRC_ALPHA means adding noise before multiplying!

premult: then at least the incoming signal is ok...
dithering alpha depends on how it's used in the blend-function… prefer pre-multiplied alpha to SRCALPHA, ONE_MINUS_SRCALPHA
http://home.comcast.net/~tom_forsyth/blog.wiki.html#[[Premultiplied%20alpha]
]
...allows to blend colors smaller than 1/255 (as they are dithered afterwards)


alpha-noise:
1xLSB

...empirical values \o/ Set manually by artists

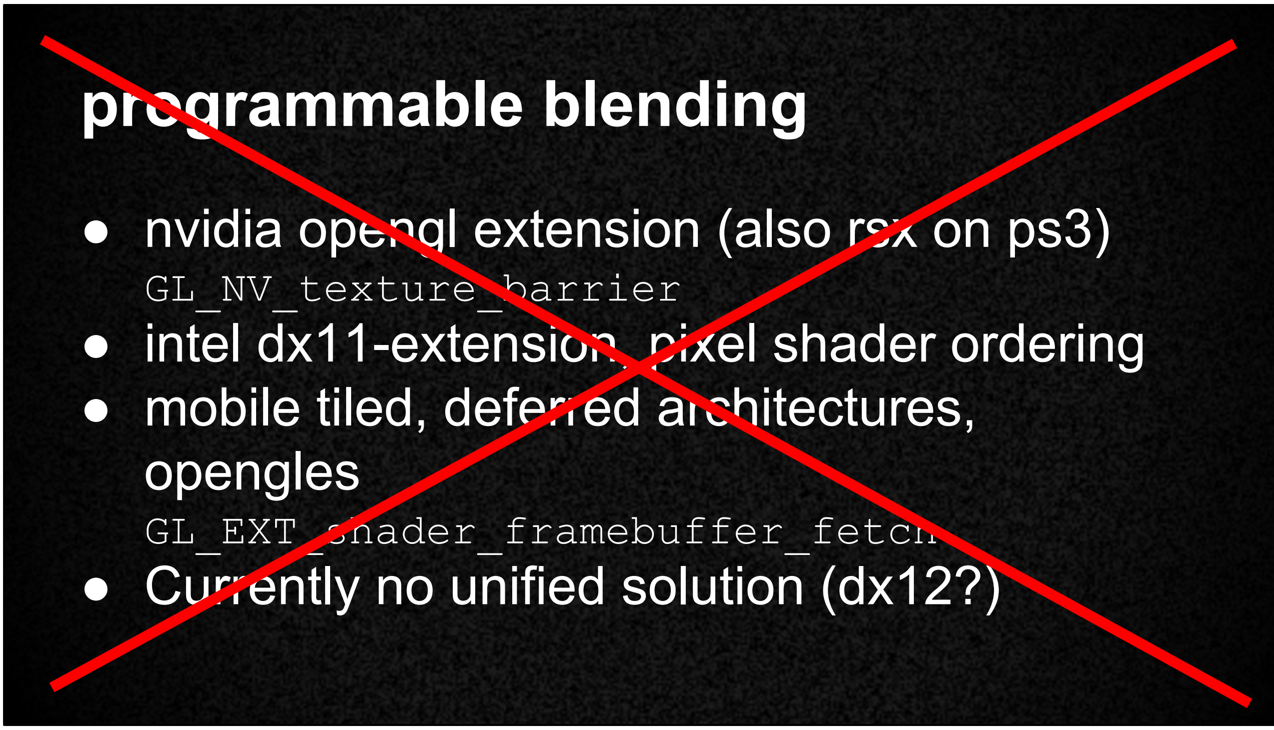


honestly not entirely sure what is going on here… but seems vendor2 runs blending at higher precision (or quantizes after blending). Fortunately, vendor2 can be found in all major new consoles.

a * srccol + b * dstcol

trunc(a)*trunc(c0) + trunc(b)*c1t //… :(
trunc(c0d + b*c1t) // dithering-noise gets added “correctly”

-
Hindsight: This appears to be consistent with the d3d11 spec, which states that blending only has to be performed at rendertarget-precision (e.g. rgba8 => 8bit blending precision):
“When a RenderTarget has a fixed point format <snip> blending operations may be performed at equal or more (e.g up to float32) precision/range than the output format.”



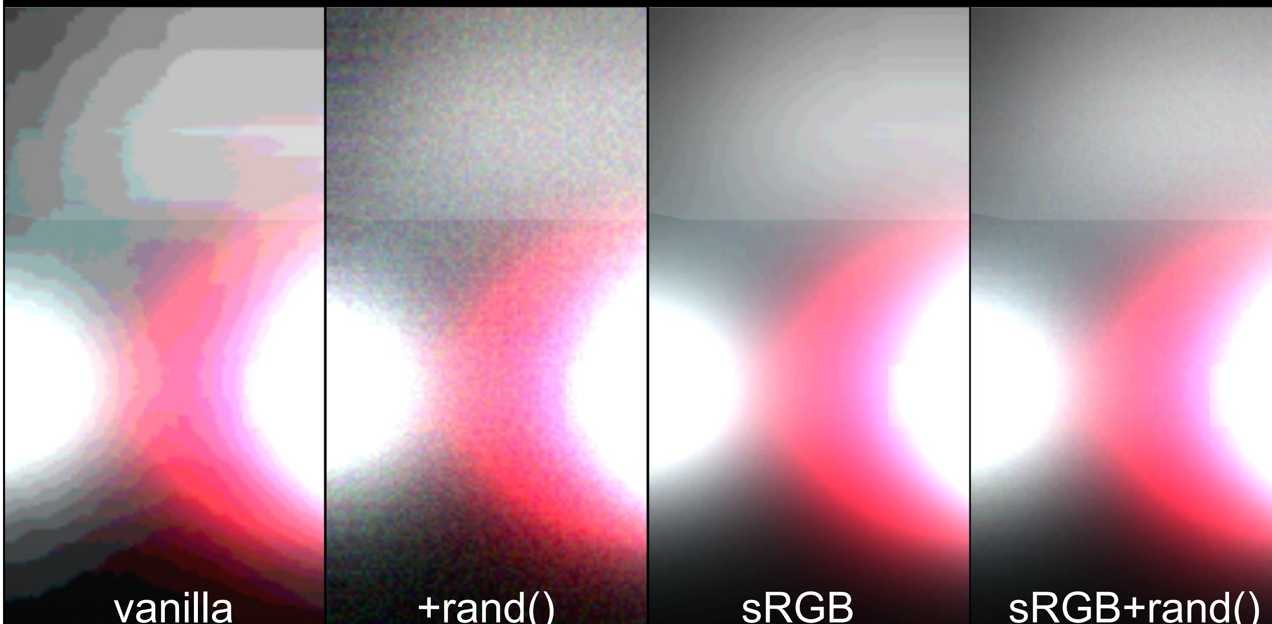
way overkill - on platforms that support this, there are likely better alternatives: sRGB, higher precision buffers etc.

nvidia / ps3 texture barrier http://www.opengl.org/registry/specs/NV/texture_barrier.txt
Intel pixelsync
https://software.intel.com/en-us/blogs/2013/03/27/programmable-blend-with-pixel-shader-ordering
moar pixelsync
http://advances.realtimerendering.com/s2013/2013-07-23-SIGGRAPH-PixelSync.pdf
Nvidia Raster-ordered view
http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_980_Whitepaper_FINAL.PDF

post-processing

- Glow, screen-fades, vignettes...
- Typically happens on 8bit textures
  (for very good bandwidth reasons)
- Writes images in multiple steps
  - ...so you should dither each step!
    (or just the “important” ones, ymmv)
  - if need to match former incorrect, truncated, non-dithered color, add (signed) rand[-0.5;0.5] instead of (normalized) rand[0;1]

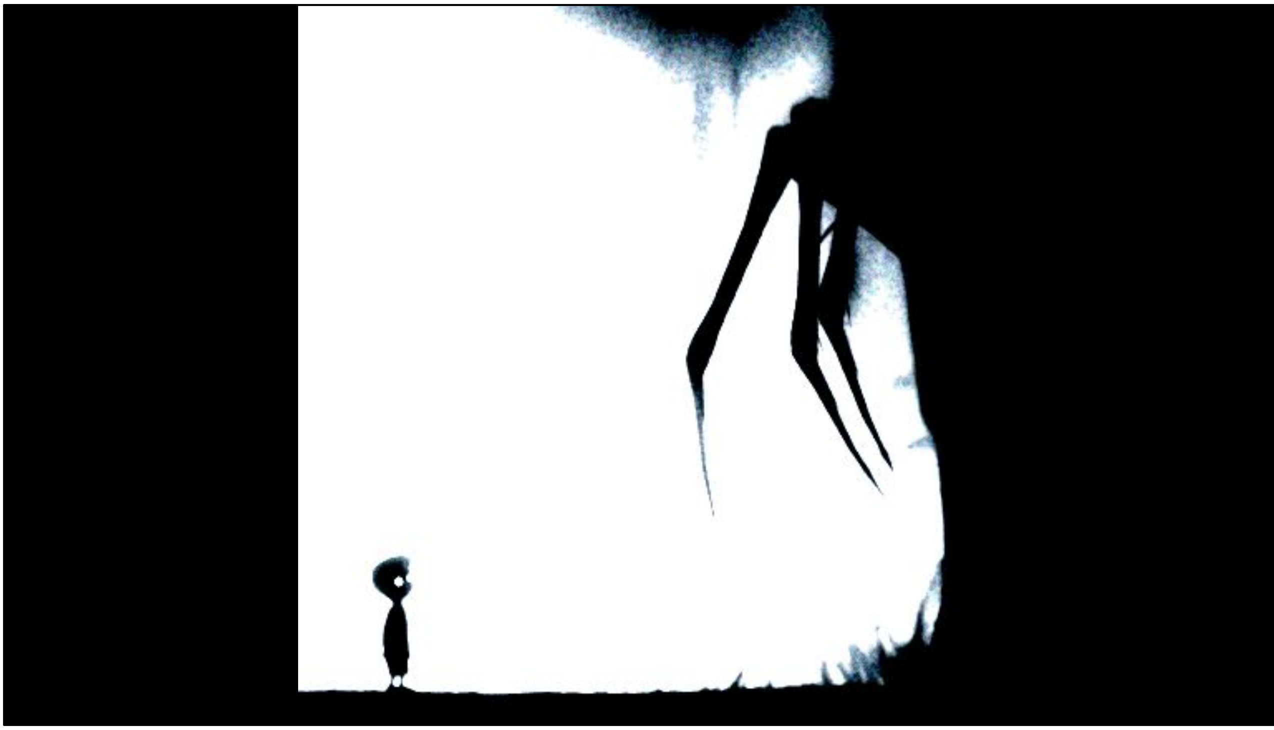
post-processing: Glow



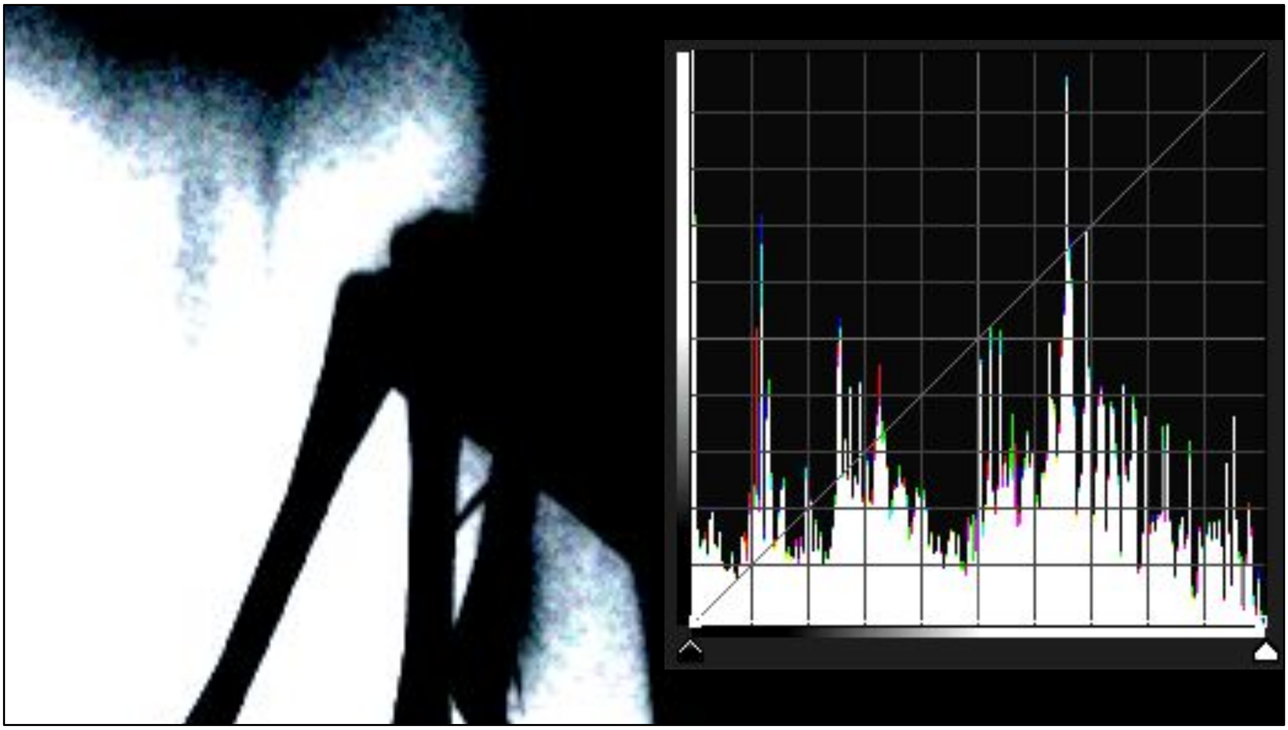
our solution dithers after every pass
manually converts all intermediate rendertargets to srgb (pow2, nothing fancy)
results in larger than 1pixel noise, but still not visible
sRGB goes a long way! adding noise was necessary for us though

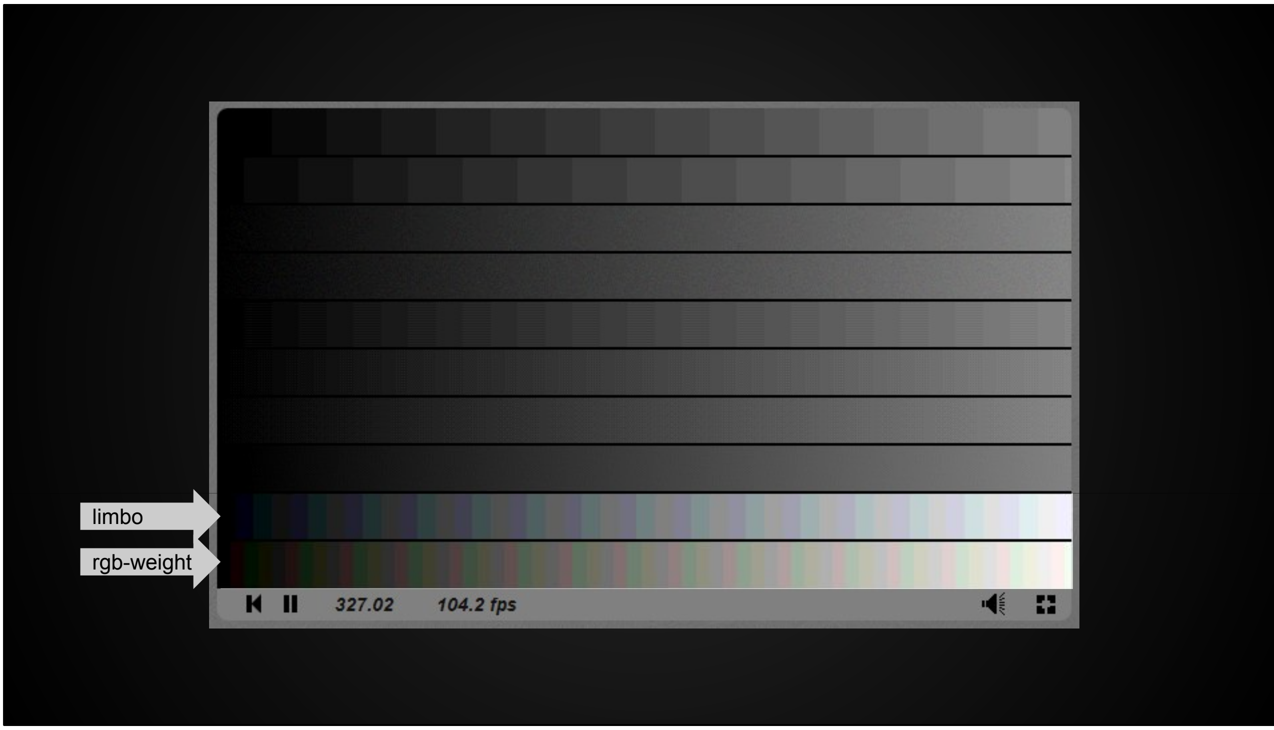
limbo was entirely grayscale, also through the rendering pipeline

rendered in 10:10:10:2 on pc/xbox360, rgba16 on ps3 - color “offset-dithering” on tonemapping (plus tons of film-grain in post!)
- please keep this “entirely gray-scale” image in mind, for when you during the next slide think “oh this can’t possibly work!”



color-offset dithering
idea by Kim Steen Riber, now at Unity





Very fast. Works for grayscale ONLY
“blueish” - color offset used in Limbo, adds x3 luminance steps
“red/green-ish” - weighted by rgb individual luminances, still only x3 steps
(even better using a custom LUT for luminances:
https://www.shadertoy.com/view/4sjXWw , adds ~x16 steps)

dithering overview: https://www.shadertoy.com/view/MslGR8

concluding recommendations

- Use highest affordable precision available (duh)
- Use sRGB if feasible across target platforms
- Dither using triangular noise [-0.5;1.0[during quantisation
  (e.g. at tonemapping, or any write to an 8bit rt)
- Add noise in the right color-space
  (the color-space your quantised values are stored in)

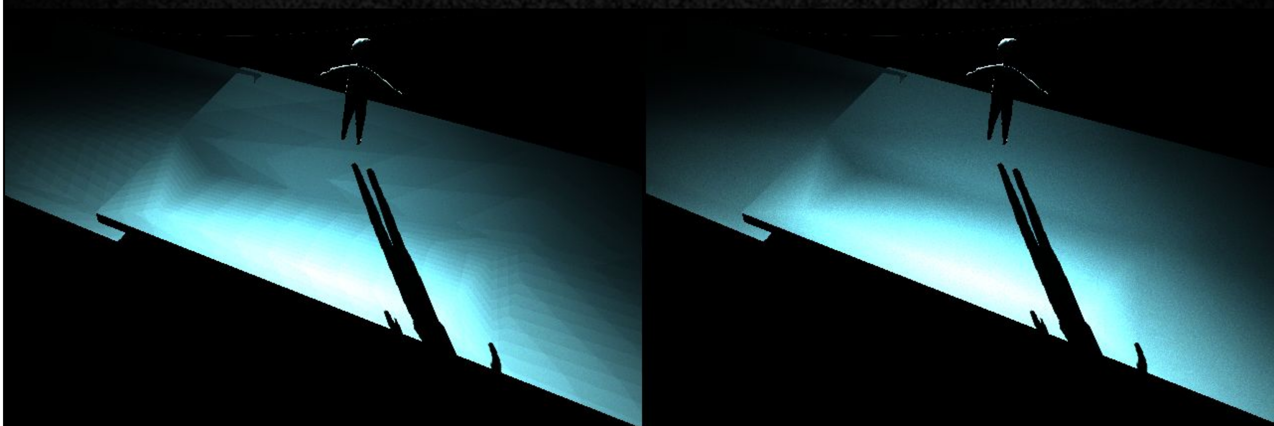
HINDSIGHT: GPUs round, so dither-range should be [-1;1[

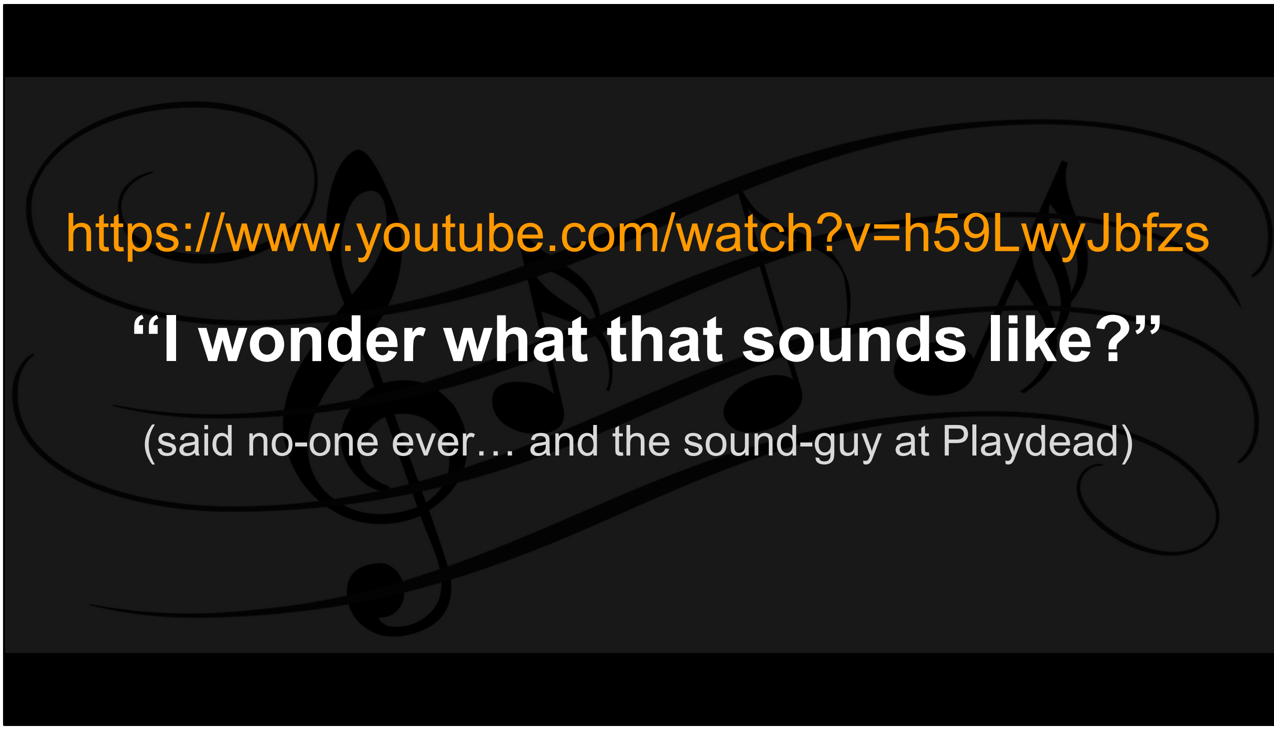
welcome side-effects from dithering

- Can now use and blend colors < 1/255
- No color-shifts in gradients (rgb truncate differently)
- Tons of additive layers? Now adds noise instead of banding
- More colors perceived than the monitor is capable of
- smoooooth screen-fades!

more or less intended side-effects

ooh, this stuff works for gbuffer-normals too!





https://www.youtube.com/watch?v=h59LwyJbfzs
not something you hear about graphics algorithms every day :)

Thanks

Daniel Povlsen, @danielpovlsen
Mikkel Svendsen, @IkarosAV
Jakob Schmid, @jakobschmid

All images are copyright of their respective authors

references

http://en.wikipedia.org/wiki/Dither

http://caca.zoy.org/wiki/libcaca/study/introduction

http://sandervanrossen.blogspot.dk/2012/02/hdr-dithering.html

http://www.realtimerendering.com/blog/2011-color-and-imaging-conference-part-ii-courses-a/

https://bartwronski.com/2016/10/30/dithering-part-one-simple-quantization/

https://www.shadertoy.com/view/MslGR8 - various dithering methods
https://www.shadertoy.com/view/4dfXW8 - example-workspace used in this presentation

http://xiph.org/video/vid2.shtml - on audio dithering

http://www.ece.rochester.edu/courses/ECE472/Site/Assignments/Entries/2009/1/15_Week_1_files/Lipshitz_1992.pdf - dithering theory

http://download.microsoft.com/download/b/5/5/b55d67ff-f1cb-4174-836a-bbf8f84fb7e1/Picture%20Perfect%20-%20Gamma%20Through%20the%20Rendering%20Pipeline.zip


PLAYDEAD IS HIRING
job@playdead.com

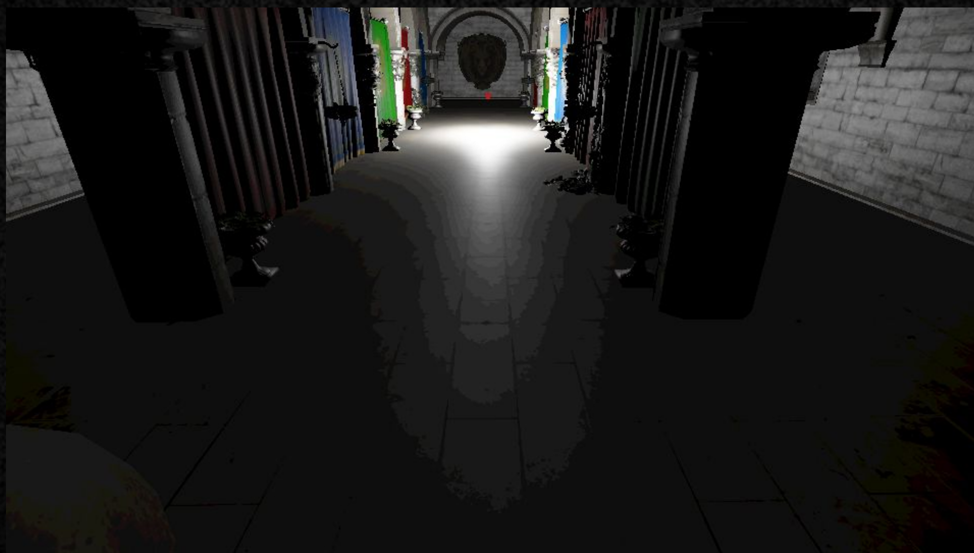
~~bad~~ bonus slides!

ARB_framebuffer_sRGB (2008)

"…render into a framebuffer that is scanned to a monitor configured to assume framebuffer color values are sRGB encoded. This assumption is roughly true of most PC monitors with default gamma correction. This allows applications to achieve faithful color reproduction for OpenGL rendering without adjusting the monitor's gamma correction."

http://www.opengl.org/registry/specs/ARB/framebuffer_sRGB.txt

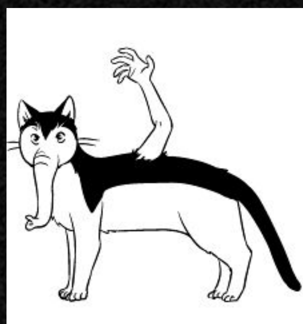
HDR resolve



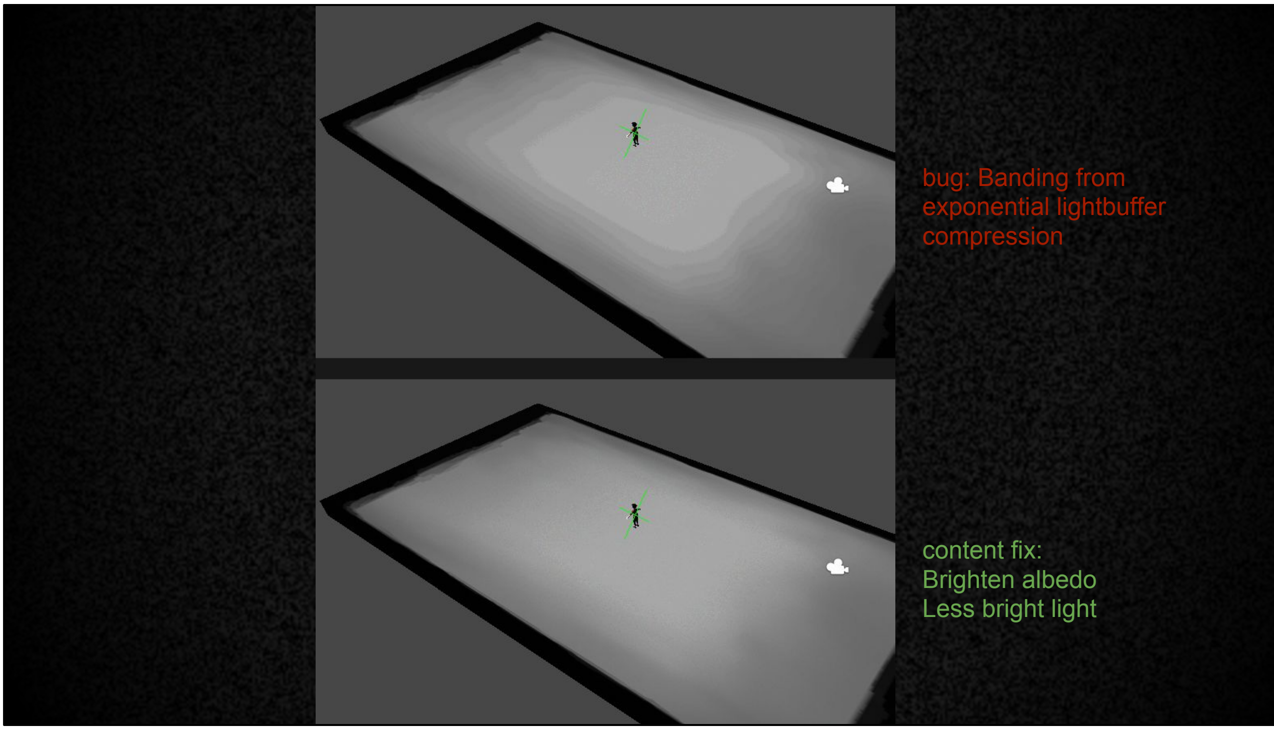
http://sandervanrossen.blogspot.dk/2012/02/hdr-dithering.html

Real World Assets

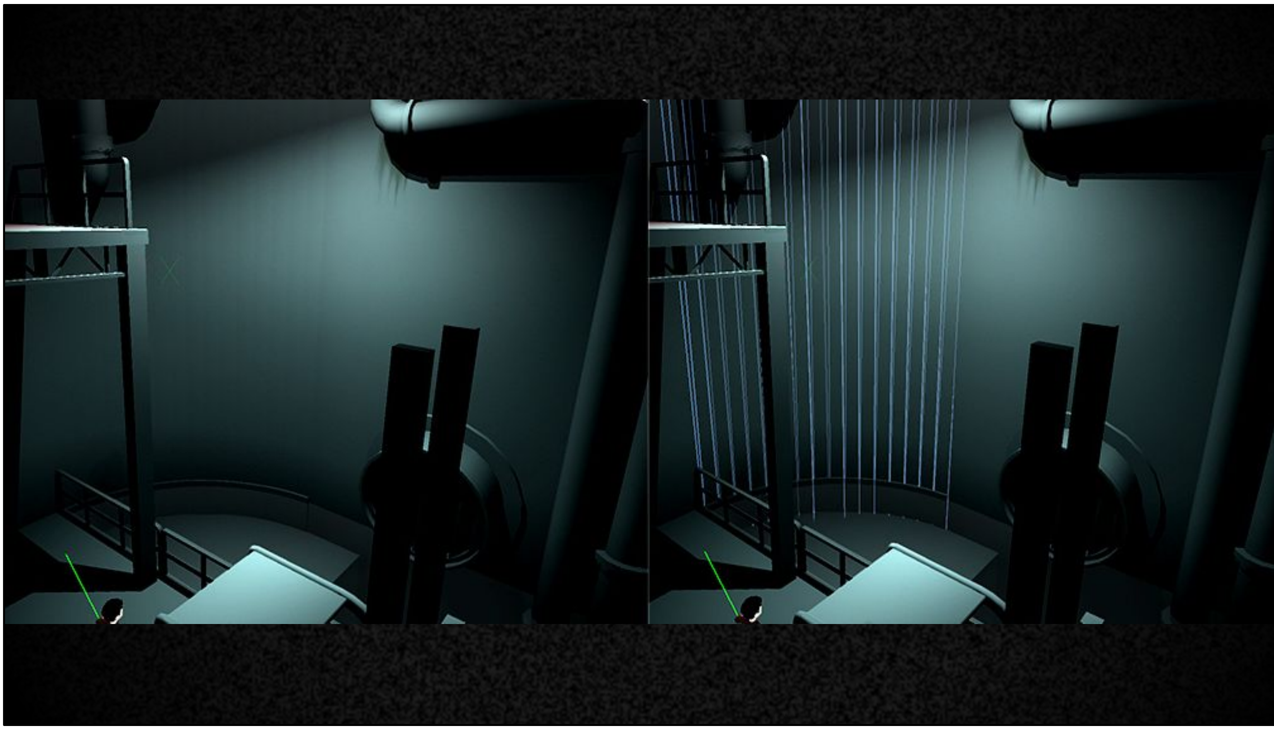
(it's just noise, not magic)



http://www.sandraandwoo.com/comics/2012-11-19-0430-software-engineering-now-with-cats.png



bugs from the real world
- a lightsource brighter than the sun, illuminating dark, black granite stone.



bugs from the real world (real-world assets)
“what the… oh”....not banding, just faceted geometry...