I'm doing it wrong

more often than not.

Why Reinhard desaturates my blacks

August 19, 2010

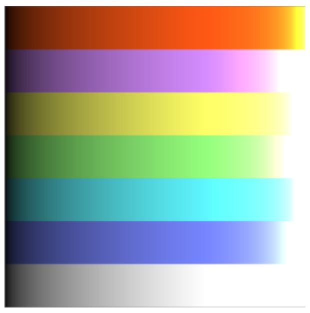
I'm doing it wrong.

Now that TFU2 (http://www.unleashed2010.com) is *almost* out of the door, I've been catching up on this year's GDC presentations. One that was of particular interest to me was John Hable's (http://filmicgames.com/) Uncharted 2 HDR Lighting talk because I think we're all in agreement about how awesome the game looks. That led me to checking out his blog and his discussions of various tone mapping operators.

I agree with him on most of his points and I really like the results of his operator, but I was a bit disappointed by the treatment of Erik Reinhard's (http://www.cs.ucf.edu/~reinhard/cdrom/) tone mapping operator.

In order to explain why, I've pinched the HDR photo from John's blog and it's accompanied by some colour ramps to illustrate the results of applying various operations. The ramps go from a luminance of 0 up to a luminance of er… *very large* and are in linear RGB space.

Shown below are the source images (the photo is exposed at +4 stops). Click through for less tiny versions:

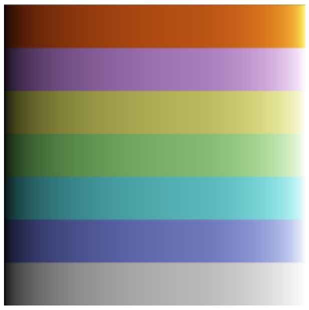

(https://imdoingitwrong.files.wordpress.com/2010/08/linear_ramps.png)

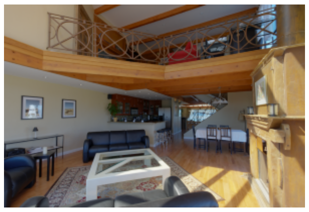
(https://imdoingitwrong.files.wordpress.com/2010/08/linear_house.png)

In both his blog and GDC presentation, John describes a simplified version of Reinhard's operator as applying the following function to each colour channel:

$$F(x) = \frac{x}{x+1}$$

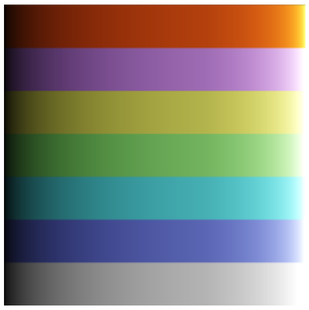
Let's do that to our test image and see what happens:

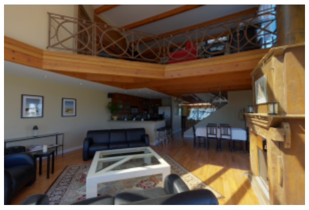

(https://imdoingitwrong.files.wordpress.com/2010/08/simple_reinhard_rgb_ramps.png)


(https://imdoingitwrong.files.wordpress.com/2010/08/simple_reinhard_rgb_house1.png)

The top end isn't nearly so blown out, but where did all my colour go?! That's no good at all!

Let's check out how John's operator (http://filmicgames.com/archives/75) does:


(https://imdoingitwrong.files.wordpress.com/2010/08/hable_rgb_ramps.png)


(https://imdoingitwrong.files.wordpress.com/2010/08/hable_rgb_house.png)

It's *much* better, especially in the blacks, but it's still rather desaturated towards the top end. Perhaps that's the price one pays for compressing the dynamic range so heavily.

Actually it doesn't have to be.

The problem with the tone mapping operators that John describes is that they all operate on the RGB channels independently. Applying *any* non-linear transform in this way will result in both hue and saturation shifts, which is something that should be performed during final colour grading, not tone mapping. Instead, Reinhard's tone mapping operator should be applied on each pixel's *luminance*, which will preserve both the hue and saturation of the original image.

There's some confusion on the internet about the correct way to apply Reinhard's operator. Some (http://wiki.gamedev.net/index.php/D3DBook:High-Dynamic_Range_Rendering#Luminance_Transform) sources (http://www.gamedev.net/community/forums/topic.asp?topic_id=407348) recommend converting from linear RGB into CIE xyY, a colour space derived from CIE XYZ (http://en.wikipedia.org/wiki/CIE_1931_color_space). The advantage of this colour space is that luminance is stored in the Y channel, independently of chromacity in xy. The idea is that you convert your image from RGB to xyY, perform the tone mapping on the Y channel only and then convert back to RGB.

While not complicated, the transform between linear RGB and xyY isn't exactly trivial either. Here's a simple implementation in C#, for a reference white of D65 (http://www.brucelindbloom.com/index.html?Eqn_RGB_XYZ_Matrix.html):

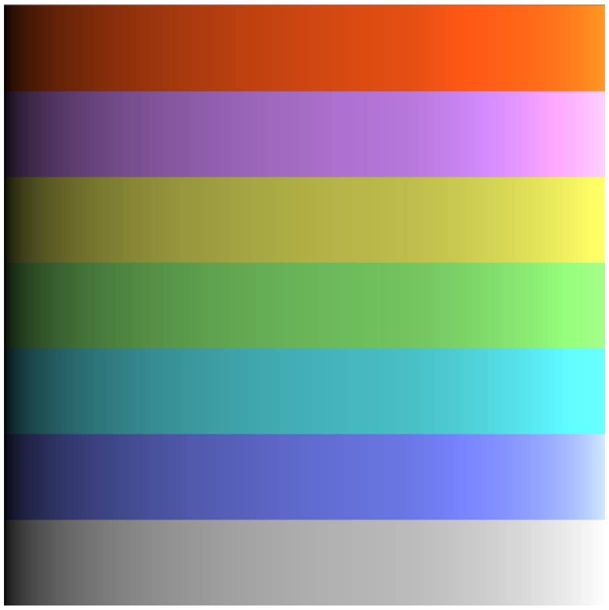
```csharp
void RGBtoxyY(double R, double G, double B,
              out double x, out double y, out double Y)
{
    // Convert from RGB to XYZ
    double X = R * 0.4124 + G * 0.3576 + B * 0.1805;
    double Y = R * 0.2126 + G * 0.7152 + B * 0.0722;
    double Z = R * 0.0193 + G * 0.1192 + B * 0.9505;

    // Convert from XYZ to xyY
    double L = (X + Y + Z);
    x = X / L;
    y = Y / L;
}

void xyYtoRGB(double x, double y, double Y,
              out double R, out double G, out double B)
{
    // Convert from xyY to XYZ
    double X = x * (Y / y);
    double Z = (1 - x - y) * (Y / y);

    // Convert from XYZ to RGB
    R = X *  3.2406 + Y * -1.5372 + Z * -0.4986;
    G = X * -0.9689 + Y *  1.8758 + Z *  0.0415;
    B = X *  0.0557 + Y * -0.2040 + Z *  1.0570;
}
```

I was using this colour space transform for a couple of days, until my esteemed colleague Miles (http://mmack.wordpress.com) pointed out that I was doing it wrong. A much simpler approach is to calculate your luminance directly from the RGB values, perform the tone mapping on this value and then scale the original RGB values appropriately:

```
double L = 0.2126 * R + 0.7152 * G + 0.0722 * B;
double nL = ToneMap(L);
double scale = nL / L;
R *= scale;
G *= scale;
B *= scale;
```

This yields the same results as the conversion to and from xyY and has fewer magic numbers, which is always a win.

Now lets see what happens when we apply the same *x / (x+1)* to each pixel's *luminance*:


(https://imdoingitwrong.files.wordpress.com/2010/08/simple_reinhard_luminance_ramps.png)

(https://imdoingitwrong.files.wordpress.com/2010/08/simple_reinhard_luminance_house.png)

Balls. This has preserved the colours, but at a terrible price; now all the whites are gone. The reason is that by preserving the hue and saturation, the operator prevents any colours being blown out to a full white. Luckily, Reinhard's got our back. In his paper, the preceding operation is written as:

$$L_d(x,y) = \frac{L(x,y)}{1+L(x,y)}$$

Almost immediately after this equation, Reinhard goes on to say:

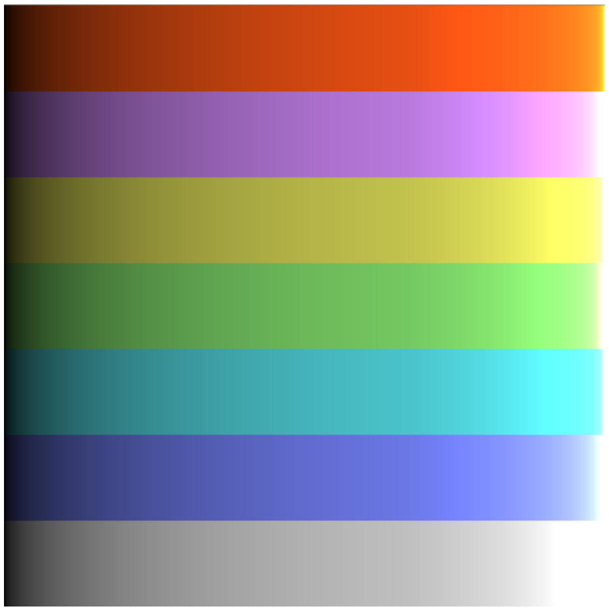
> "This formulation is guaranteed to bring all luminances within displayable range. However, as mentioned in the previous section, this is not always desirable"

He then presents the following:

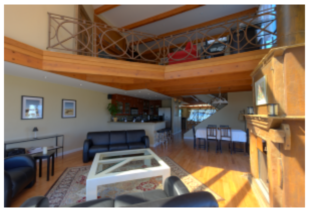
$$L_d(x,y) = \frac{L(x,y)\left(1+\frac{L(x,y)}{L^2_{white}}\right)}{1+L(x,y)}$$

Here, L_{white} is the smallest luminance that will be mapped to 1.0.

Let's give that a whirl, with an L_{white} of 4 for the colour ramps and an L_{white} of 2.4 for the condo photo:



(https://imdoingitwrong.files.wordpress.com/2010/08/full_reinhard_luminance_ramps.png)



(https://imdoingitwrong.files.wordpress.com/2010/08/full_reinhard_luminance_house.png)

Well that's much better than the previous effort; whether its better than John's results is up for debate. What I like about it is that because it mostly preserves the hues, so you haven't lost any information; if you want to crisp the blacks or desaturate the whites, you can do that in your final colour grade. If you're colour grading via a volume texture lookup, this is for free!

Having said all that, the TFU2 environment artists were most happy with a simple brightness & contrast adjustment with a small 0 – 0.05 toe, which in the end only costs one extra madd instruction at the end of our shaders. Whatever works for your game, makes the artists happy and means you get to go home at 6pm, right?

A full implementation of Reinhard's operator adapts to the local changes in luminance around each pixel and is a little more involved that what I've described here, but hopefully I've managed to contribute something useful to the tone mapping in games discussion and not just reduced the signal to noise ratio on the subject.

Posted by Tom Madams
Filed in wrongness
Tags: graphics, hdr, tone mapping
12 Comments »

12 Responses to “Why Reinhard desaturates my blacks”

Ben Diamand Says:

August 26, 2010 at 9:47 pm
Very good article! In fact, God of War III shipped with almost the exact same tone mapping operator as what you have presented here. The only real interesting extension came about because we found that Reinhard did not allow us to drop the low end of the curve enough. So in a similar way to what you suggest, we had what amounted to a madd to drop the low end contrast (well, really, two madds in order to smoothly ramp on a low end contrast shift without affecting values closer to the white point).

I do have a couple of nits to pick though!

You said, “if you want to crisp the blacks …you can do that in your final colour grade. If you’re colour grading via a volume texture lookup, this is for free!”

My major nit is that while you can certainly apply color correction via a texture lookup at the end (and indeed, GOW did just this to get many cool effects), I found it very difficult to get the kind of complex function required to do decent quality color grading (the kind that emulates the ‘filmic’ look) without a very large 3D map. And to get, for example, the same effect as John got by doing tone mapping per channel requires quite a bit of precision, at least for us, due to how the various operators modify intensities in such non-linear ways.

It was even worse on GOW, since we actually weren’t able to maintain a full precision buffer throughout the pipeline, meaning that all we had was a 32-bit RGBA buffer to color correct. This wass fine for simple stuff like desaturation or brightness/contrast changes, but was not nearly enough precision for high precision color grading.

And then there’s the problem of map creation. If all you are doing is mapping a 32-bit value to another 32-bit value, using artist driven tools like Photoshop is great, and the artists get to come up with all kinds of nice mapping textures! But as soon as you want to allow either larger range or larger precision, we’ve found a woeful lack of support out there (Photoshop’s plain old 16-bit support is better than it used to be, but still crappy). So even if we had a post tone mapped FP16 buffer to work with, at the end of the day, getting film-style color correction in there (for which we didn’t find any natural mapping with any of the built in Photoshop knobs) would have been both painful, and I think would have required lots of texels.

Interestingly, GOW did not do per channel Reinhard (I simply didn’t think to do it for one thing – John had much better theoretical handle on the problem than I did!), but post shipping, as we’ve ramped up to using a larger dynamic range (we added HDR to GOW alarmingly late in the dev cycle), the particular look of over saturated darks has grown on many of our artists, and we’ve experimented with per channel Reinhard (as well as a full implementation of John operator).

What we’ve found is that while the per channel idea has a couple of downsides (it’s more expensive for one, and as you rightly pointed out it is completely fixed in nature and so not easily amenable to artist modification), the look it gives is unique, and often quite nice. And it’s also a look that we’ve found very difficult to replicate with post effects (though it is theoretically possible). From a theoretical standpoint, I've become convinced that John was on to the right idea – film really does respond to different colors independently, and so in effect, the tone mapping 'exposure' really should be done per color, at least if you want to emulate film. So incorporating that idea within tone mapping instead of after does have a good basis in (film) reality.

As for John's specific operator, it obviously worked very well for Naughty Dog, but we have (at least for now) chosen to stick with Reinhard. As it turns out, Reinhard is a sub-function of John 'filmic' operator. In other words, a little algebra shows that you can implement all of Reinhard with proper constants in John's operator. But we've found the added functionality to be a lot harder to control, and with not too many benefits over Reinhard for the added complexity, so while there are some curve shapes that we cannot get, the trade-off in ease of use will likely keep us in the Reinhard camp (albeit with 'filmic' style per channel mapping).

Wow – that was a larger reply than I was intending – I hope not too large! ☺

Ben Diamand

Reply
Naty Hoffman Says:

August 30, 2010 at 12:48 pm
Reinhard was on the wrong track in applying his tone mapping curve to luminance; his curve was inspired by film, but film curves are applied to each color channel separately. And that is a good thing – here is a salient quote from Georgianni & Madden's "Digital Color Management, 2nd ed." (a book I highly recommend to anyone who wants to understand color reproduction): "…adjusting RGB, rather than luminance… results in higher overall reproduced color saturation, which is desirable because it helps compensate for color-saturation decreases associated with viewing flare and with the relatively low absolute luminance levels of the… (reproduced) images.".

The "hue and saturation shifts" (really mostly saturation shifts) resulting from applying nonlinear curves per channel are an important feature, not a bug. When the nonlinear curve has the proper filmic S-shape these shifts are highly desirable. For more information on why an S-curve is optimal (beyond the fact that it is the result of 200 years of careful engineering), please see Josh Pine's slides in my recent color course (http://renderwonk.com/publications/s2010-color-course/) – Josh's course notes will go into more detail than the slides, but they are not ready yet.

I agree with Ben that while in principle all effects can be achieved with a LUT post pass, in practice there are precision and size issues which mean that tone mapping curves like this are best applied using either math or dense 1D textures, onto the original HDR data (inside the pixel shader, or on an HDR buffer). The color LUT should be reserved for color enhancement effects rather than basic color rendering.

The main problem with the Reinhard curve is that it lacks a "toe". From Ben Diamand's description ("two madds in order to smoothly ramp on a low end contrast shift without affecting values closer to the white point") it sounds like they have added a toe to Reinhard, giving it similar visual properties to the Hable curve. This "toe" when applied per channel, gives an increase in shadow saturation that has very good perceptual effects.

Reply

Michael Says:

September 25, 2010 at 10:17 am
Very useful post, thanks a bunch!

Reply
Tone mapping Says:

November 5, 2010 at 5:57 am
[…] I have to see how this all relates now to new exposure values (and the exposure_factor). The link https://imdoingitwrong.wordpress.com/…s-my-blacks-3/ also tries to undo John Hable's filmic tone operator where Reinhard is improved a bit; I tried that […]

Reply
ZACK Says:

December 26, 2011 at 9:23 am
Great post!!!
It's what I am looking for!

Reply
Nirans Viewer (fixed) - Page 44 - SLUniverse Forums Says:

July 27, 2012 at 5:48 pm
[…] range than computer monitors are capable of. I was mostly looking at Wikipedia and articles like this. Basically, making an image which would normally look like this: Look like this: Basically by […]

Reply
Readings on color management | Light is beautiful Says:

September 10, 2012 at 10:55 pm
[…] Why Reinhard desaturates my blacks: a follow up on John Habble's article (with insightful comments to read too). […]

Reply
Ruud van Gaal Says:

October 19, 2012 at 7:45 am
Modified Reinhard is a lot softer than John Hable's UC2 curve; that one sometimes gives a bit too filmic contrast IMHO. I'll give this a go for some time, thanks for the article!

Reply
Constantine Says:

February 20, 2017 at 7:36 am
Hi!

I was experimenting with tonemapping for my Unity game and came up with formula below. It`s damn simple but I found the result very satisfying. Could you please give it a look?

```
vec3 logToneMap(vec3 c, float limit){
float luma = dot(c, vec3(0.2126, 0.7152, 0.0722));
float luma2 = log(luma + 1.0) / log(limit + 1.0);
c *= luma2 / luma;
return c;
}
```

In my scene it lacked contrast in middle range so i added this dirty hack. though i suppose it should not be needed.

```
float hackyContrast(float x, float coef){
float s_curve = x * x * (3 – 2 * x);
return mix(x, s_curve, coef);
}
```

I also included my tonemapping formula into your tonemapping demo on shadertoy.com. its number three from the botton

https://www.shadertoy.com/view/4sXcWn

Reply

Constantine Says:

February 20, 2017 at 9:30 am

I made a mistake above – should have applied logToneMap separately to every channel instead of luma.
The code in shadertoy is the one i actually use.

Reply

netocg Says:

December 11, 2018 at 3:16 pm

Hi,

Interesting discussion.
I've been recently reading more and more about tone mapping.
I work with VFX, and I was playing recently with some Nuke script to tone map some hdr images that I've done with my camera. I've also tested photometric and other software.

Going back to the formula I remember there's values you should use for the white point, as well toe, shoulder etc…

My idea is get LDR images that have a high perceptual fidelity of the light I was seeing it when I captured the images. And I wonder if you have any tips in regards to what values should I use for each "input"?

I am currently capturing hdr doing 7 backetings exposures with 1ev internal between each. My camera is capable of register 11.6stops of dynamic range in it's raw file.

Best,
Antonio.

Reply

Tom Madams Says:

December 16, 2018 at 12:52 pm

Not really, I’m afraid. I’m just a humble programmer and the only experience I have with tone mapping is from video games, where configuring those options was left up to much more talented artists. Plus, I haven’t worked on this stuff for more than seven years 😉

Reply