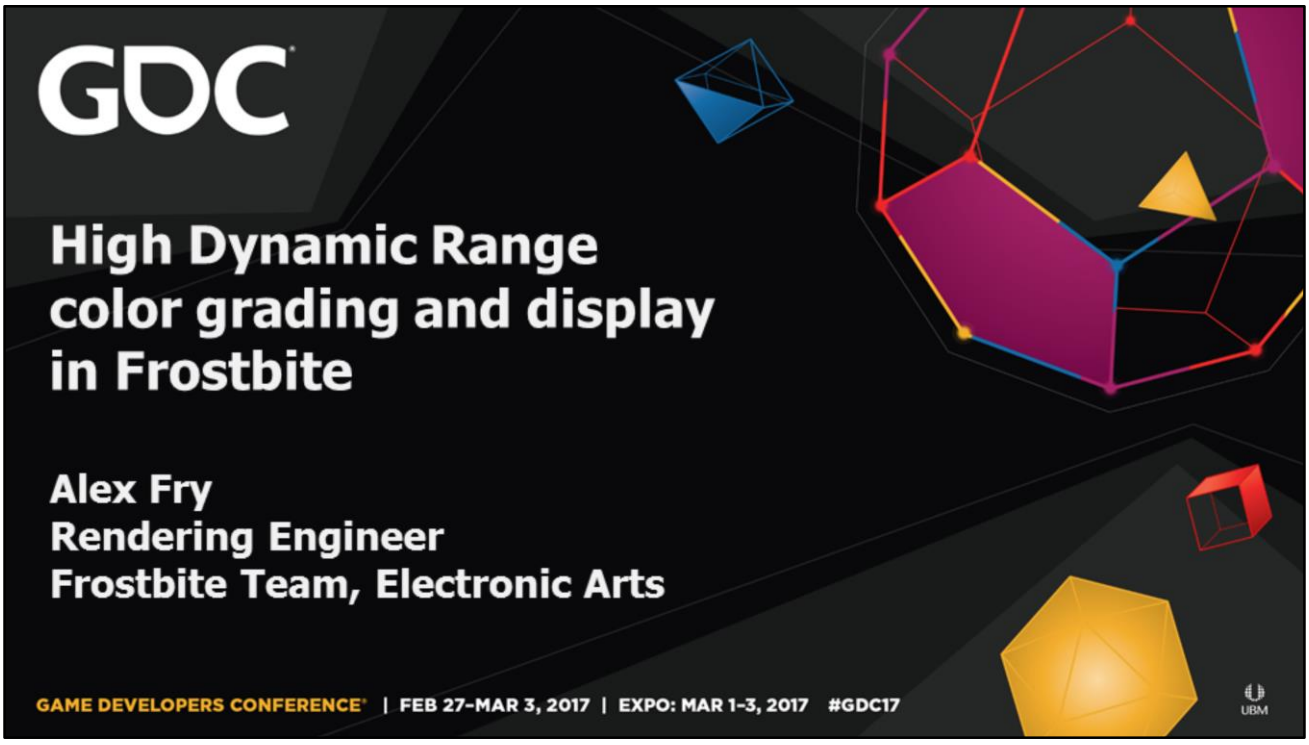
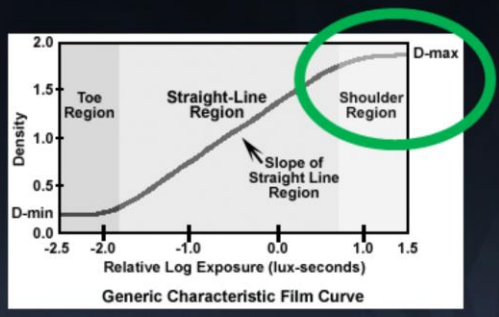

GDC
High Dynamic Range color grading and display in Frostbite
Alex Fry
Rendering Engineer
Frostbite Team, Electronic Arts
GAME DEVELOPERS CONFERENCE® | FEB 27–MAR 3, 2017 | EXPO: MAR 1–3, 2017 #GDC17
UBM

Contents

- History, terminology & tonemapping
- Frostbite legacy post process pipeline & issues
- Considering how to support HDR TVs & our new pipeline
- "Display mapping" & issues uncovered
- LUT precision & decorrelation
- Performance
- HDR standards and platform support
- Next steps

Brief history of film & terminology

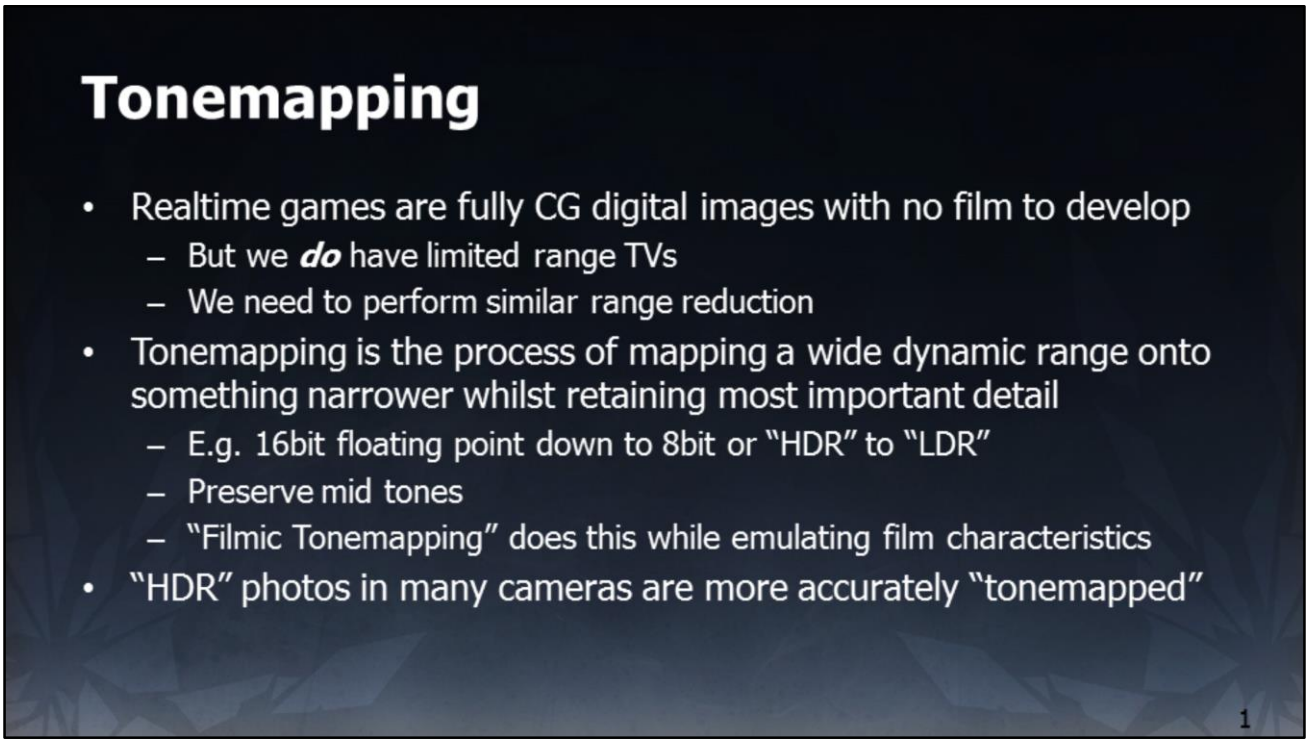
- Film density (of the negative) responds to light at different rates
- 3 main sections
  - Mid section ('linear')
  - Toe (slower response)
  - Shoulder (slower response)
- Captures wide range of light values
  - Toe/Shoulder are main range reduction
  - Expose scene for mid section
- Characteristic, familiar and pleasant look



1

Basic explanation of film response to light.
Non-linear, captures wide range.
Looks nice.

Image credit: http://www.naturephotographers.net/articles0303/tw0303-4.gif
From http://www.naturephotographers.net/articles0303/tw0303-1.html



Games are realtime CG with no film to develop
But we do need to broadcast our games on TVs
These TVs have a limited range not dissimilar to film/slide projectors
So we have similar requirements to perform range reduction

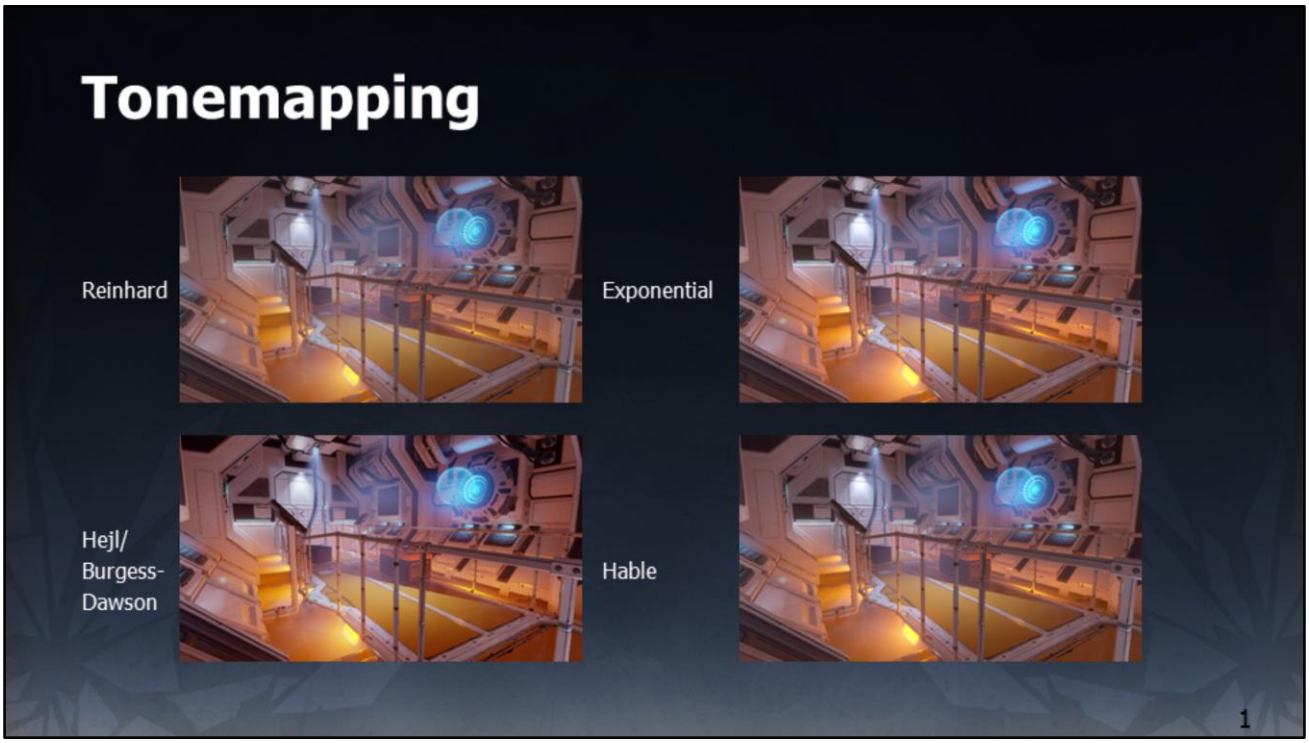
Tonemapping is just range reduction, done in such a way to preserve detail outside the display range.
Follow a similar process.
First we would expose the scene such that the most interesting parts are in the mid tones.
Then tonemap the rest of the range to best fit into the upper/lower regions of the SDR.

Tonemapping

- Example image from Frostbite internal demo level

1

Going to use this example image for our tests, screenshot from internal test level built by Joacim Lunde
-Good contrast
-Wide dynamic range
-Some saturated colours
-Clearly suffering from lack of tonemapping in several parts (lights, hologram)



Example of four tonemappers:
Two simple (Reinhard & Exponential). Primarily a shoulder (affect brightest parts the most, darkest parts the least).
Two filmic (Hejl/Burgess-Dawson & Hable). Toe and shoulder as well as contrast changes.

Color grading

- The act of applying a "look"
- Background in film developing. E.g.
  - Choose different film stock
  - Apply or skip different processing steps (e.g. bleach bypass)
- Digitally we can do a lot more
  - White balancing
  - Color replacement
  - Orange & Teal …

1

Color grading is the act of applying a characteristic look to the scene
Still has its background in film, for example one could choose different film stock to achieve a look
Or during film processing (developing the negative) one could perhaps change the processing
e.g. bleach bypass is skipping the bleaching process which produces a higher contrast 'silvery' look.

With CG, we can do a lot more than this
-Change white balance or exposure
-Arbitrary color replacement or highlighting (Schindler's List)
-Orange and Teal (Michael Bay)
-"Fix it in post"

Image credit: http://www.drodd.com/images14/schindlers-list3.jpg
Image credit: https://pix-media.priceonomics-media.com/blog/892/trans.jpg

Brief history of TVs & standards

- Older (CRT) TV/monitors had limited dynamic range
  - 0.1 – 100 nits (cd/m2)
- Non-linear response to electrical input
  - Electro Optical Transfer Function (EOTF)
  - Also known as 'gamma curve'
- sRGB/BT.1886 standard introduced
  - Standard and display capability are similar
- We still use this standard today

1

Image credit: http://zerowaste.co.in/img/products/tv-crt-curve.png

Brief history of TVs & standards

- Modern TVs (0.01-300+ nits … HDR ***way*** more)
  - Hardware ***far*** more capable than the standard
  - LCD response different to CRT response
- We still use sRGB/709
  - TVs modify our 0.1-100 nit signal to best "show off" the TV
  - We have no control over this, other than asking for calibration
  - Tends to end up with too much contrast
- And that's just luminance
  - Too much over-saturation as well (e.g. store demo mode)

1

Now to cover our legacy grading pipeline

Frostbite legacy pipeline

- Scene is linear floating point up to and including post FX
  - Bloom, motion blur, vignette, depth of field etc
- Apply tonemap
- Linear to sRGB conversion
- Apply color grade
- Draw UI on top
- Scanout to TV

1

High level summary of the legacy pipeline.

Frostbite legacy pipeline

- Tonemap chosen by artist from a fixed set of built-in algorithms
  - Usually a filmic tonemap
  - Tonemaps are 1D curves, applied independently to RGB values
- Linear to sRGB conversion
  - Constrains results to 0-1

Frostbite legacy pipeline

- Color grading is a 3D LUT generated offline
  - Load (tonemapped, sRGB) screenshot in Photoshop
  - Load 32x32x32 'identity' color cube layer
  - Apply look transformations
  - Save (now non-identity) colour cube LUT
  - Index this LUT by [s]RGB at runtime
  - Use LUT RGB value as color

1

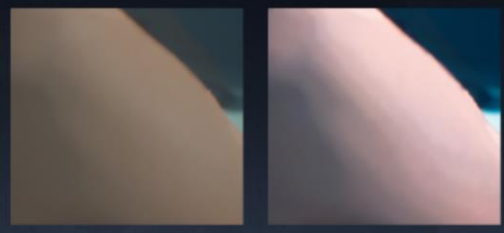
Frostbite legacy pipeline troubles

- 1D curve applied separately RGB causes hue shifts
  - Not completely linear in mid section (shots later)
  - Highlights shift as channels clip
- Usually **filmic** so this applies a "look"
  - This "look" choice is a different workflow to grading
- It's not easy for content to author new tonemaps
  - Tonemaps are not data driven (only selectable)

1

Frostbite legacy pipeline troubles

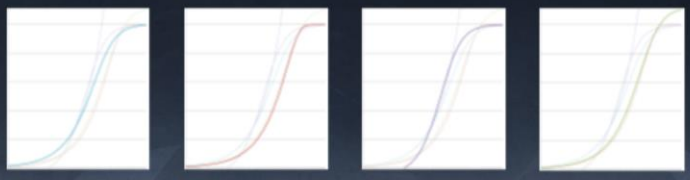
- Photoshop is not really a grading package
  - Non-optimal workflow
- LUT is often 8bit
  - Irrecoverable quantisation
  - Mach banding & hue shifts
- No access to values >1!
  - We clamped it earlier



1

Frostbite legacy pipeline troubles

- Tonemap + sRGB = LUT distribution function
  - Distribution function is not optimal for grading
  - Different distribution depending on tonemap
  - Cannot share LUTs with other EA teams if they use a different tonemap



1

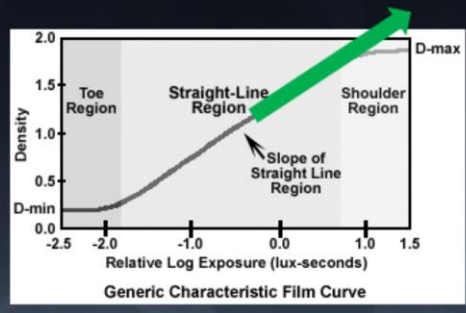
Frostbite legacy pipeline troubles

- Legacy pipeline is hardcoded to SDR
- Want to support HDR TVs
  - Higher bit depth
  - Higher dynamic range
  - Higher color range (wider gamut)
- Want to give content creators control over all of this
  - Take creative control of the TV, not rely on TV manufacturers

1

HDR TV support – simple approach

- Reverse tonemap curve to extract original HDR data
  - Cheapest possible option
- Recall that tonemaps have a shoulder
  - Reverse the shoulder
  - Recover the original HDR data
  - Profit?



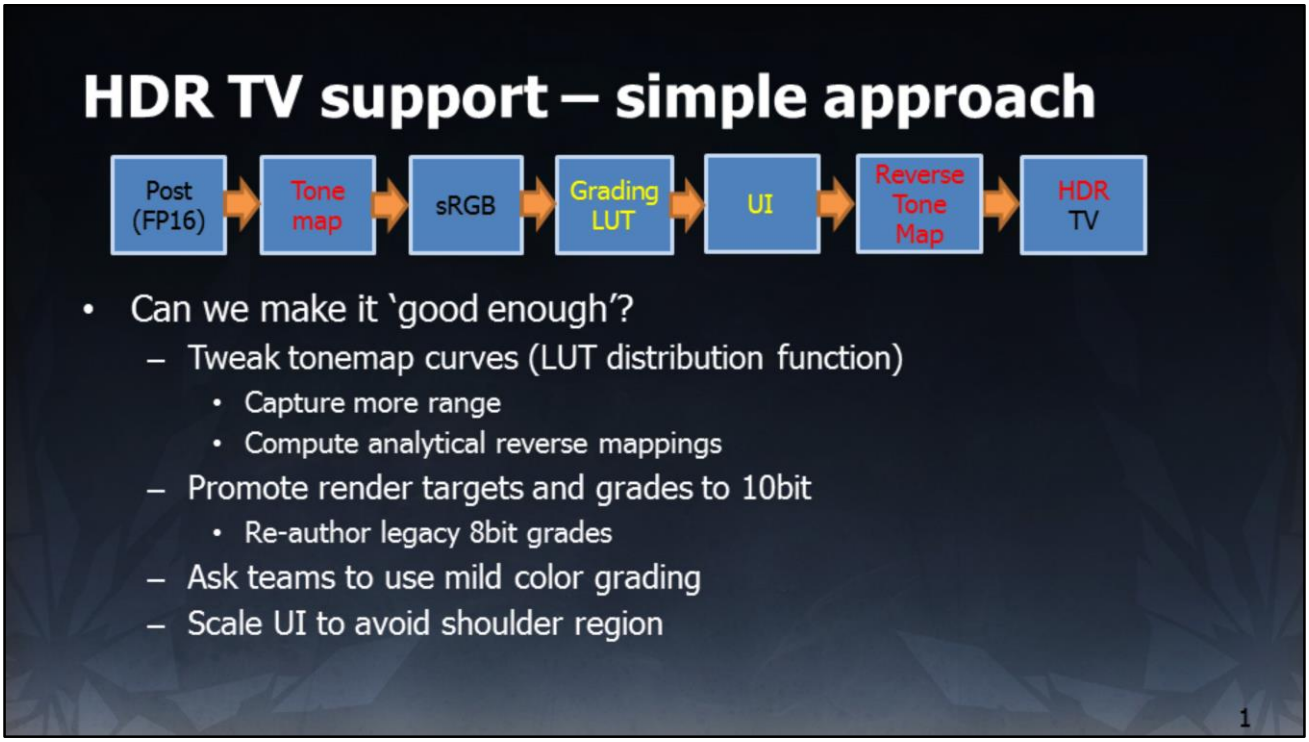
1

Image credit: http://www.naturephotographers.net/articles0303/tw0303-4.gif
From http://www.naturephotographers.net/articles0303/tw0303-1.html

HDR TV support – simple approach

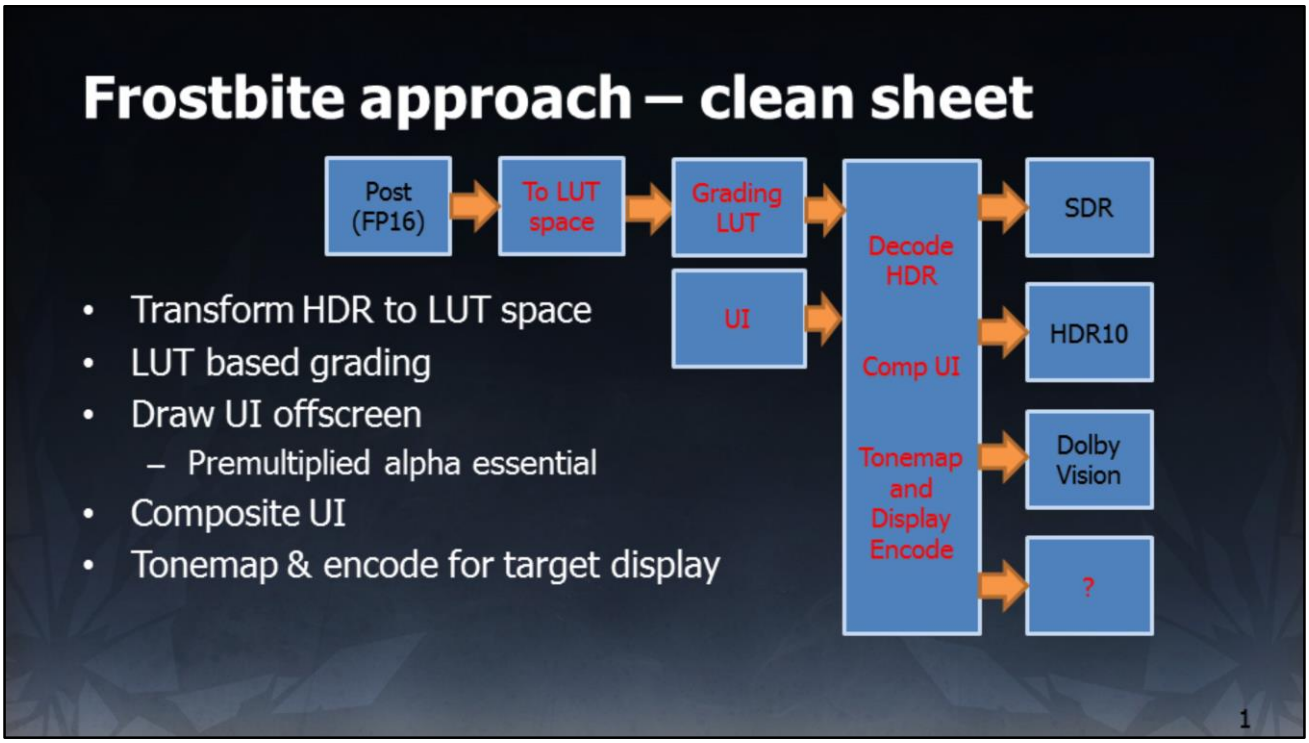
- Plenty of restrictions
  - 8bit not enough precision to reverse
  - Multiple different tonemap curves, all need to be reversible
  - A limited range can be recovered
  - Order of operations not correct
    - Grading and UI are drawn after tonemapping
    - Reversing tonemap later *incorrectly* reverses these

1



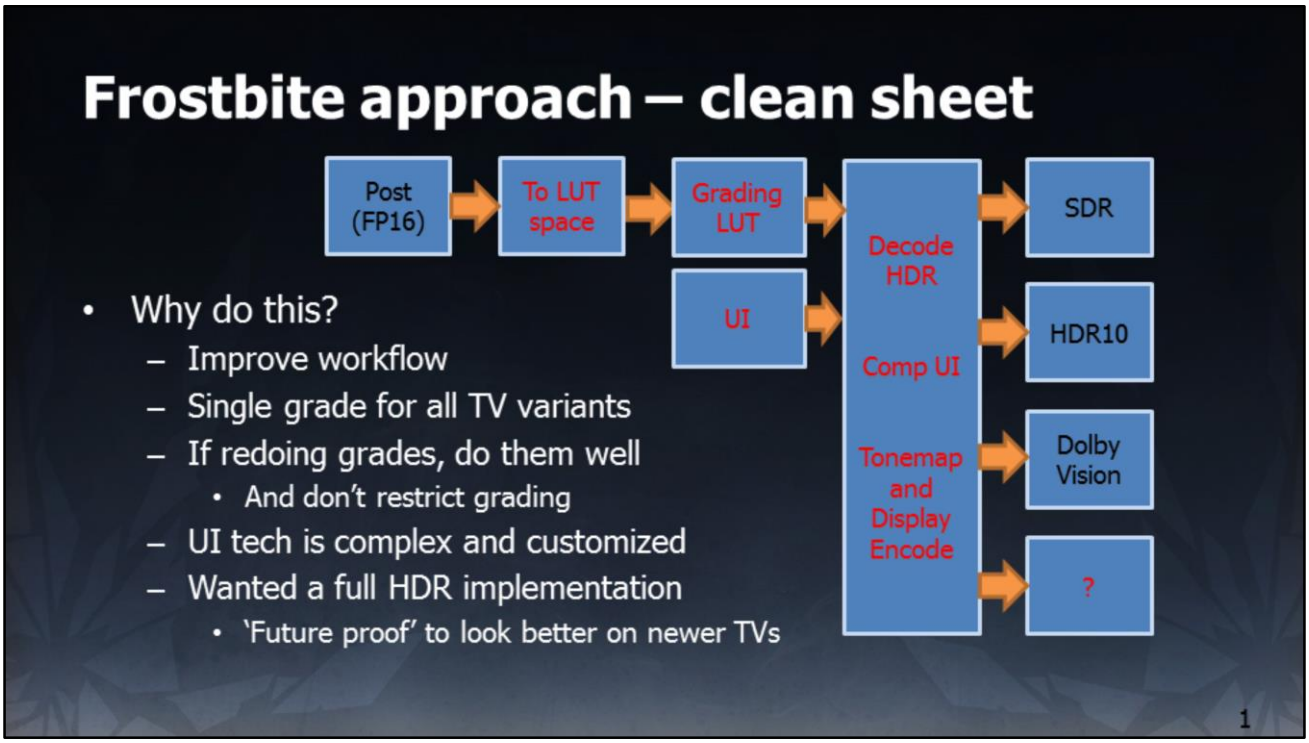
Question is – can this be made to work?
Yes, in many cases it can.
Tweak tonemap to capture 'enough' range to reverse, and compute reverse function.
Increase precision of render targets and grades to 10bit.
Use less extreme color grading, work within the limits.
Scale the UI during rendering to avoid the shoulder region that will be reversed.

Sucker Punch (team behind Infamous: Second Son) has done some nice blog posts which cover this approach. See
http://www.glowybits.com/blog/2016/12/21/ifl_iss_hdr_1/
http://www.glowybits.com/blog/2017/01/04/ifl_iss_hdr_2/



Our approach was to go for a new, clean sheet implementation.

LUT space = a single distribution function optimized for grading once in a master HDR space, regardless of connected TV or output tonemap.



Move all "look" workflow into a single place.
Grade once in a master space, regardless of connected TV/output dynamic range.
Remove UI from the equation, since lots of UI exists that would look different/incorrect if we changed the way that was drawn.
Regarding "future proof", the current HDR specification is bigger (in terms of luminance range and color gamut) than any display can reproduce today.
By targeting the full specification, our games will look better when played on better TVs in future, which reproduce more of the range we use.

HDR grading

- HDR friendly distribution function
  - Went through a few ideas (log, S-Log, LogC) etc
  - Ended up using ST.2084 or "PQ" (Perceptual Quantiser)
  - PQ ensures LUT entries are perceptually spaced, gives great control
- 33x33x33 LUT lookup (10bit UNORM runtime format)
  - Industry standard size allows use of 'standard' grading tools
- Single distribution function gives other gains
  - Anyone can make a "look" LUT and share it across EA
  - Build a look library

HDR grading

- Create grades using DaVinci Resolve
  - Industry standard grading tool
  - Build view LUTs to match Frostbite tonemapping (WYSIWYG workflow)
- Why not write our own in engine?
  - No need, time or desire to reinvent the wheel & maintain it
  - Reduces friction hiring experienced graders
  - Free version of Resolve does enough for many
- **"Resolve Live" is a great workflow improvement**
  - Must buy a capture/monitor card though … so not free

1

https://www.blackmagicdesign.com/uk/products/davinciresolve
Image credit:
http://is3.mzstatic.com/image/thumb/Purple122/v4/04/cc/37/04cc3729-b79f-37f4-c5e3-1b70a942a728/source/1200x630bf.jpg

~~Tone~~ Display mapping

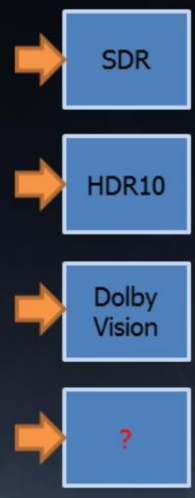
- The HDR version of the game is the reference version
- We tonemap at the last stage of the pipeline
- The tonemap is **different** depending on the attached display
  - Aggressive tonemap on SDR
  - Less aggressive in HDR but still varies depending on TV
  - No tonemap for Dolby Vision
  - Each TV gets a subtly different version tuned for each case
  - Not just curves, exposure too
- Decided to use the term **Display Mapping** instead

2

Will explain the HDR/Dolby Vision differences in later slides

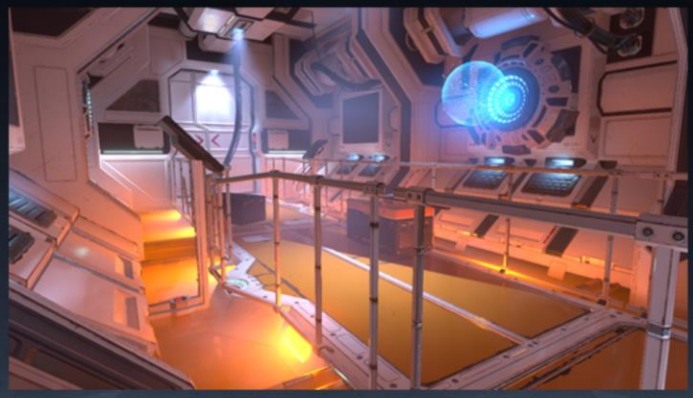
~~Tone~~ Display mapping

- Main challenges of display mapping
  - Scale across wide range of devices
  - Achieve a similar look regardless of TV
- Desirable properties
  - No toe
  - No contrast change
  - No hue shifts
  - No built in film look
- Build as neutral a display map as we could



Display mapping

- Recall our example image



1

Going to use this example image for our tests, screenshot from internal test level built by Joacim Lunde
-Good contrast
-Wide dynamic range
-Clearly suffering from lack of tonemapping in several parts (lights, hologram)
-This shot is the image naively converted to sRGB

Display mapping

- Our solution

1

Going to use this example image for our tests, screenshot from internal test level built by Joacim Lunde
-Good contrast
-Wide dynamic range
-Clearly suffering from lack of tonemapping in several parts (lights, hologram)
-This shot is the display mapped image

Display mapping

- Side by side

2

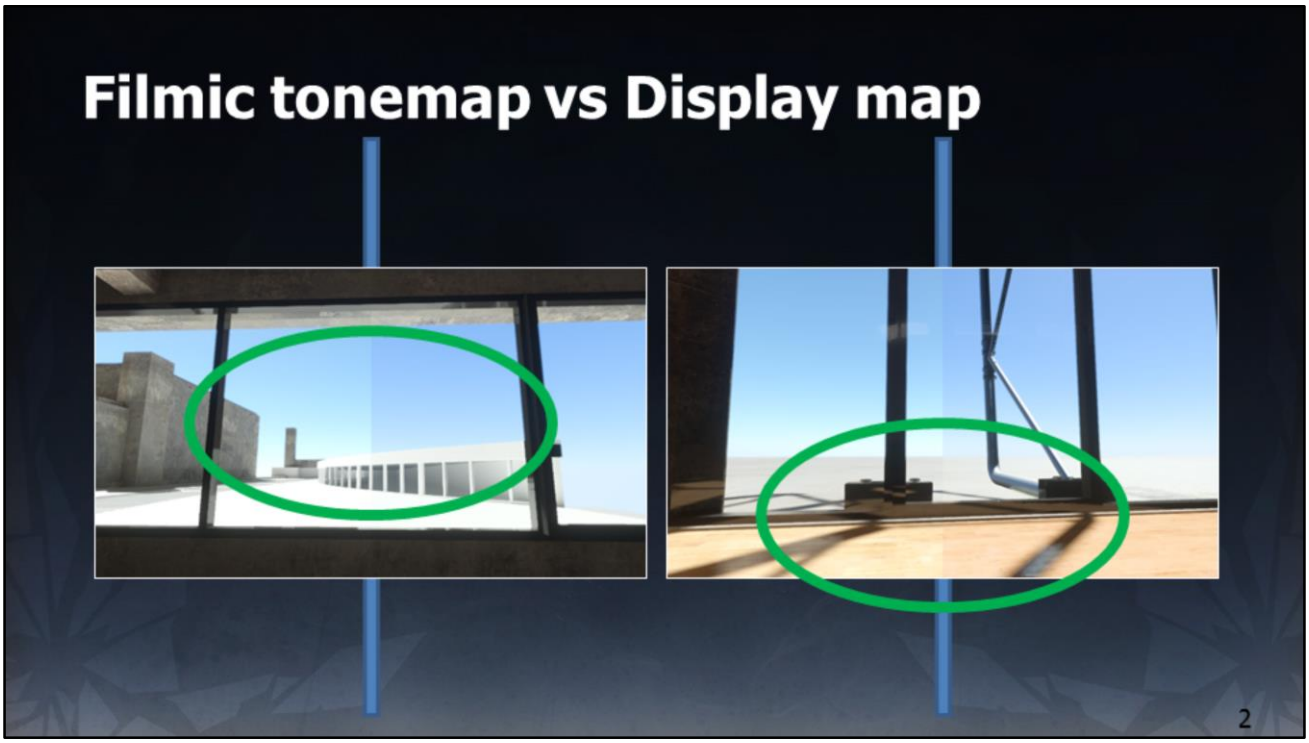
Note lack of hue shifts in the shoulder region.
No toe, no contrast or “look” just very neutral.
Rest of image is unchanged.

Display mapping

- How?
  - Work in luma/chroma rather than RGB
    - Apply shoulder to luma only
    - Progressively desaturate chroma depending on shoulder
  - Use a 'new' working space (ICtCp)
    - Developed by Dolby for HDR
    - Chroma follows lines of perceptually constant hue
  - Tuned by eye on a lot of different images/games
    - Not very "PBR" but this is about perception not maths

2

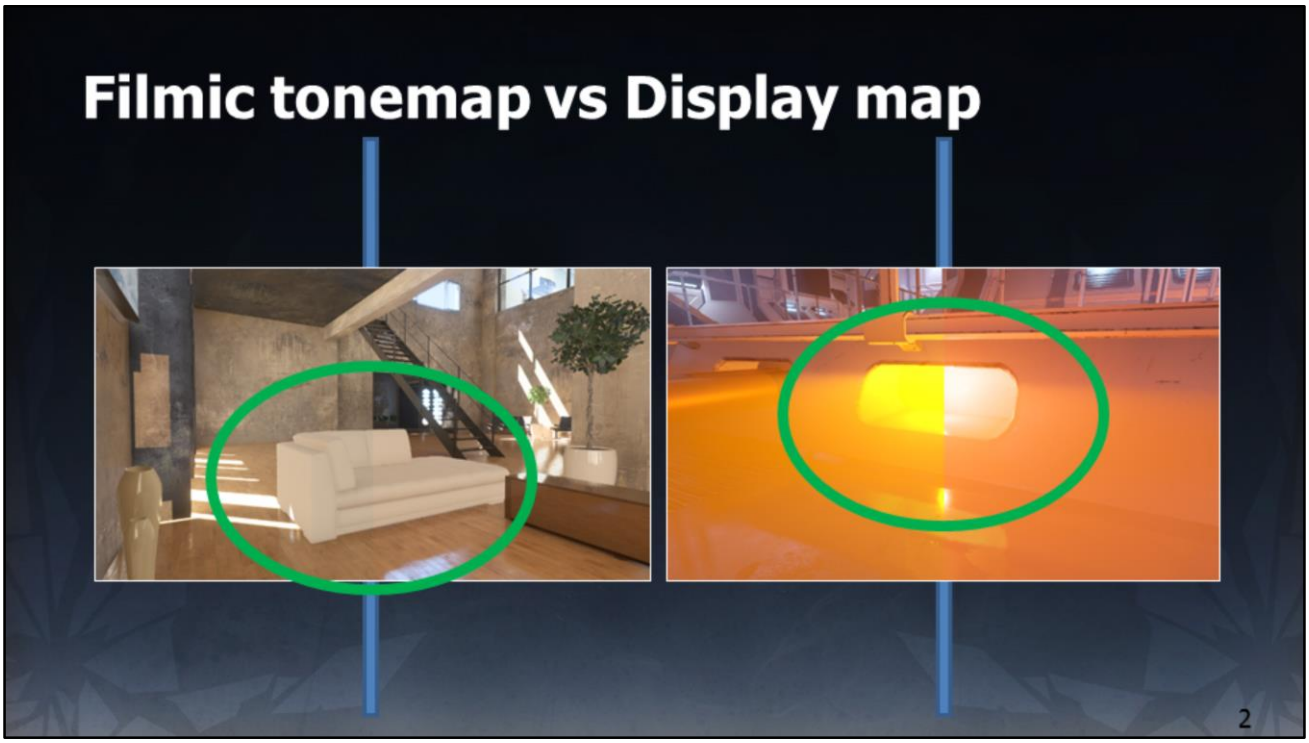
https://www.dolby.com/us/en/technologies/dolby-vision/ICtCp-white-paper.pdf



Comparison with our Filmic (modified Hable) and our Display Mapper

Filmic is 1D and applied to each channel independently, has hue shifts.
This can all be tuned, but ultimately any non-linear 1D operator applied to color channels will hue shift.
Want to make a bright blue sky? It will become cyan.

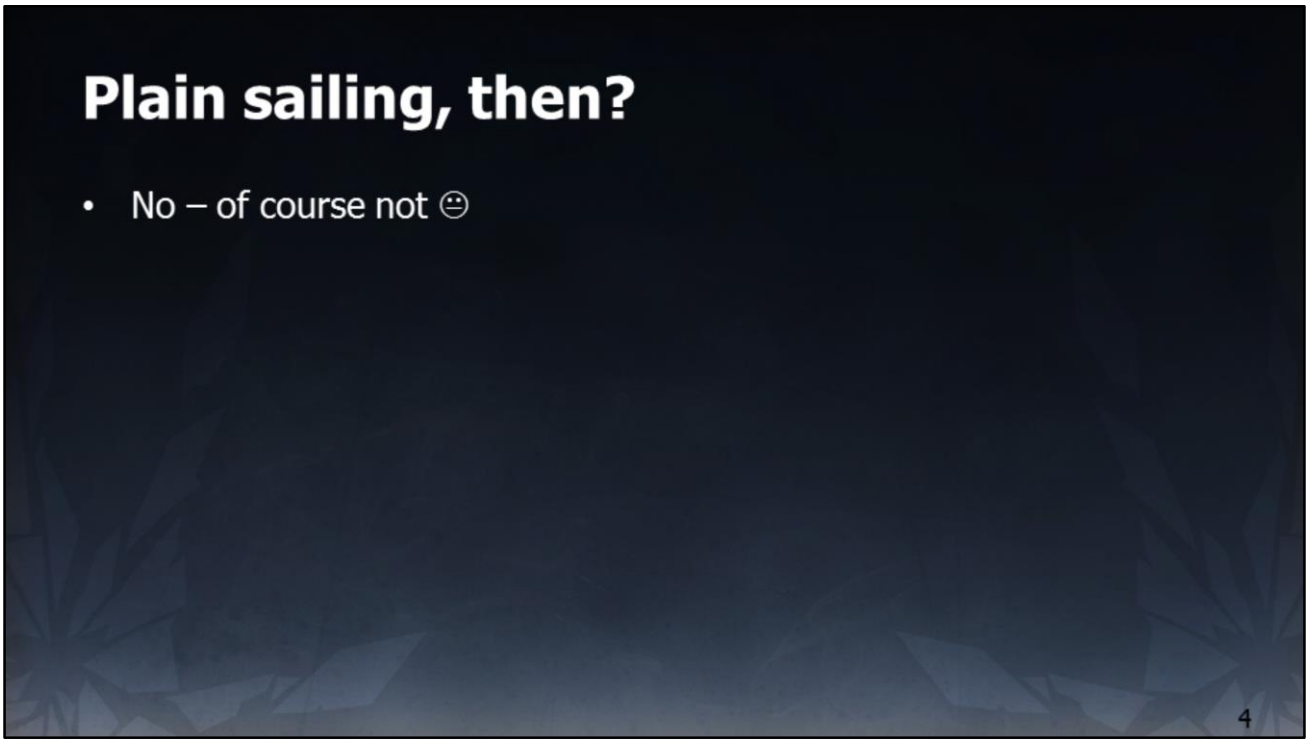
Display mapper is highly neutral, hue-preserving, does a good job of reproducing artistic intent.



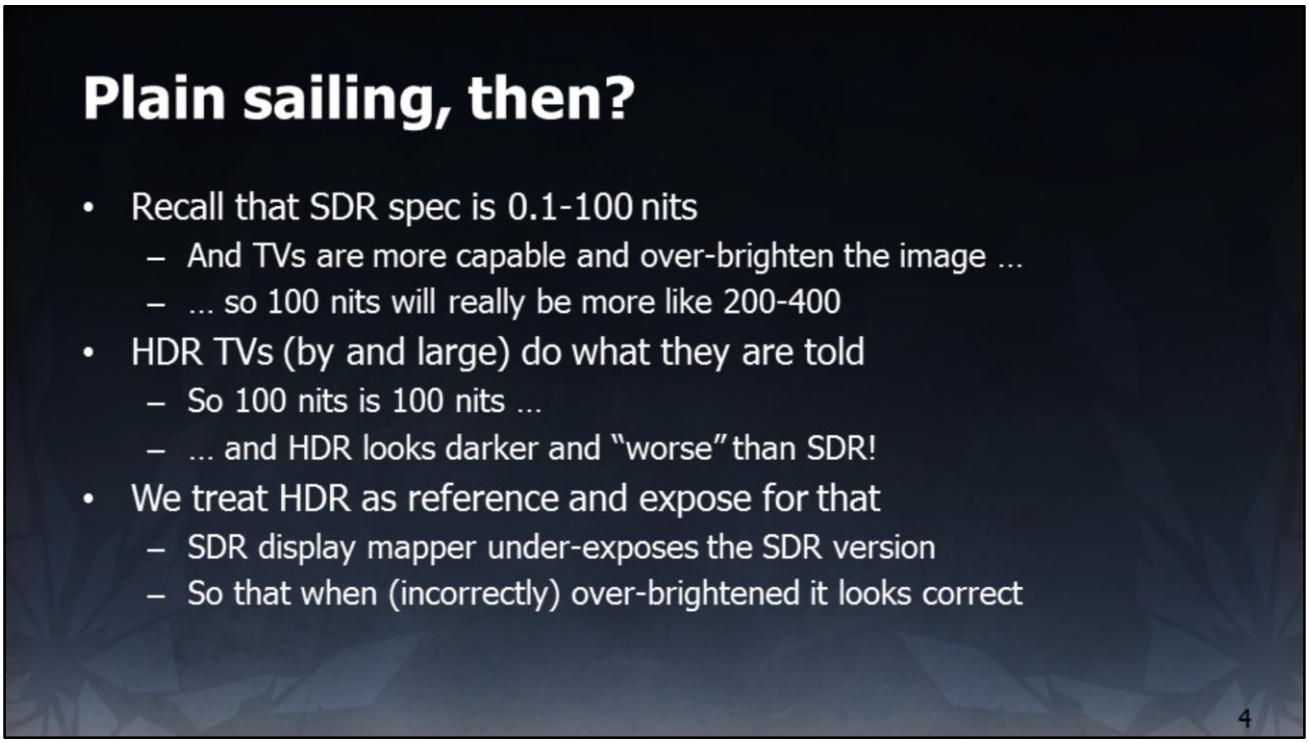
Comparison with our Filmic (modified Hable) and our Display Mapper

Filmic is 1D and applied to each channel independently, has hue shifts.
This can all be tuned, but ultimately any non-linear 1D operator applied to color channels will hue shift.
Watch our for desaturated midtones (in our case, at least).
Want to make a bright orange sunset? It will become yellow.

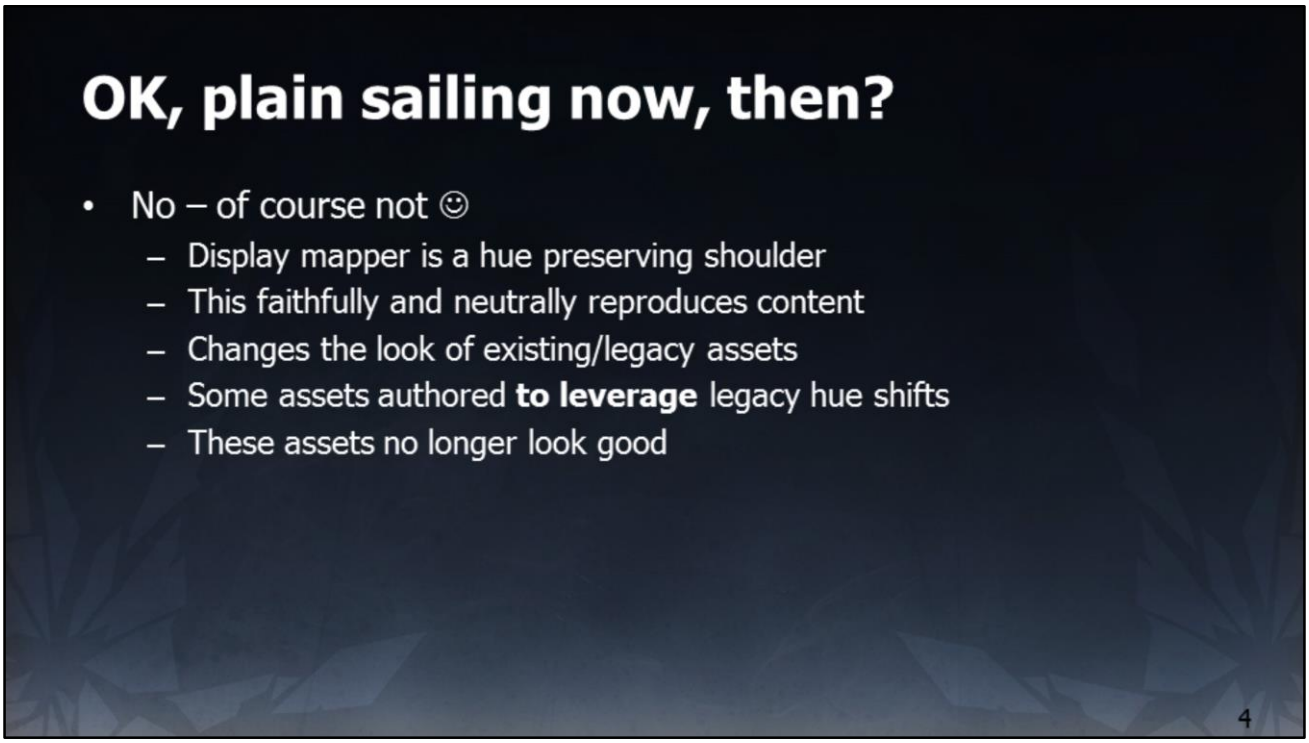
Display mapper is highly neutral, hue-preserving, does a good job of reproducing artistic intent.



Even though this was planned & we worked with several game teams to figure this out and roll it out, there were a few gotchas along the way.



As mentioned at the start the SDR version gets brightened up by the TV, sometimes massively.
We can play to this strength and under-expose the SDR version so, when brightened by the TV, it looks OK again.
No this is not correct, and it does require some guesswork as to how bright the TV will be.
Artists can configure this SDR Peak value so when working on calibrated monitors it is accurate.
Assume SDR TV is 200 nits as a default, considering letting people tune this value at home as well.



Many assets were authored to leverage the hue shifts that come from clipping color channels.
Any color that contains multiple primaries will hue shift if it's brightened and one channel clips but the other doesn't.
The ratios between the color channels fundamentally change at this point, changing the color itself.

If you leverage this hue shift during asset authoring, then these effects long longer look correct when you remove the hue shift.

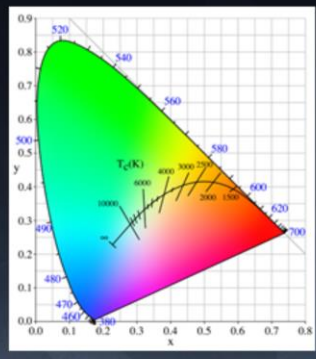
This is a contrived example showing a fireball.
It's an effect kindly shared by DICE but modified by me to highlight the issue in question.

This fireball is authored using a single hue … lets call it 'burnt orange'.
Burnt orange has little to no blue component, but does have a lot of red and some green.
When over-exposed and fed into a tonemap that has a strong shoulder and/or simply hitting the color channel clipping point, the rate of change of red slows quickly and then clips but green does not.
As one keeps over-exposing, red moves slowly (or is stationary, if clipped) but green keeps increasing, fundamentally changing the color from orange to yellow creating a multi-hue effect.

When using a hue preserving shoulder, the authored hue is preserved and it looks 'correct' but that is now completely wrong.
This isn't just the case for hue preserving shoulders though. If the original image were displayed via HDR broadcast standards which have much higher color channel clipping points, it would look wrong in HDR as well because the red channel simply wouldn't clip as quickly. So the effect would appear bright orange.
So the hue preserving shoulder in SDR actually highlights trouble in the content that would be wrong in HDR.
This is quite a nice way to author HDR content (and find issues with existing content) without needing an HDR display.

Re-author VFX

- Fire effects changing was most common complaint
  - VFX authored to leverage hue shifts in legacy tonemapper
  - Looked OK in SDR but not in HDR
- Use blackbody simulation
  - Author effect as temperature
  - Build colour ramp from blackbody radiation
  - Index ramp by temperature
  - Tweak ramp to suit

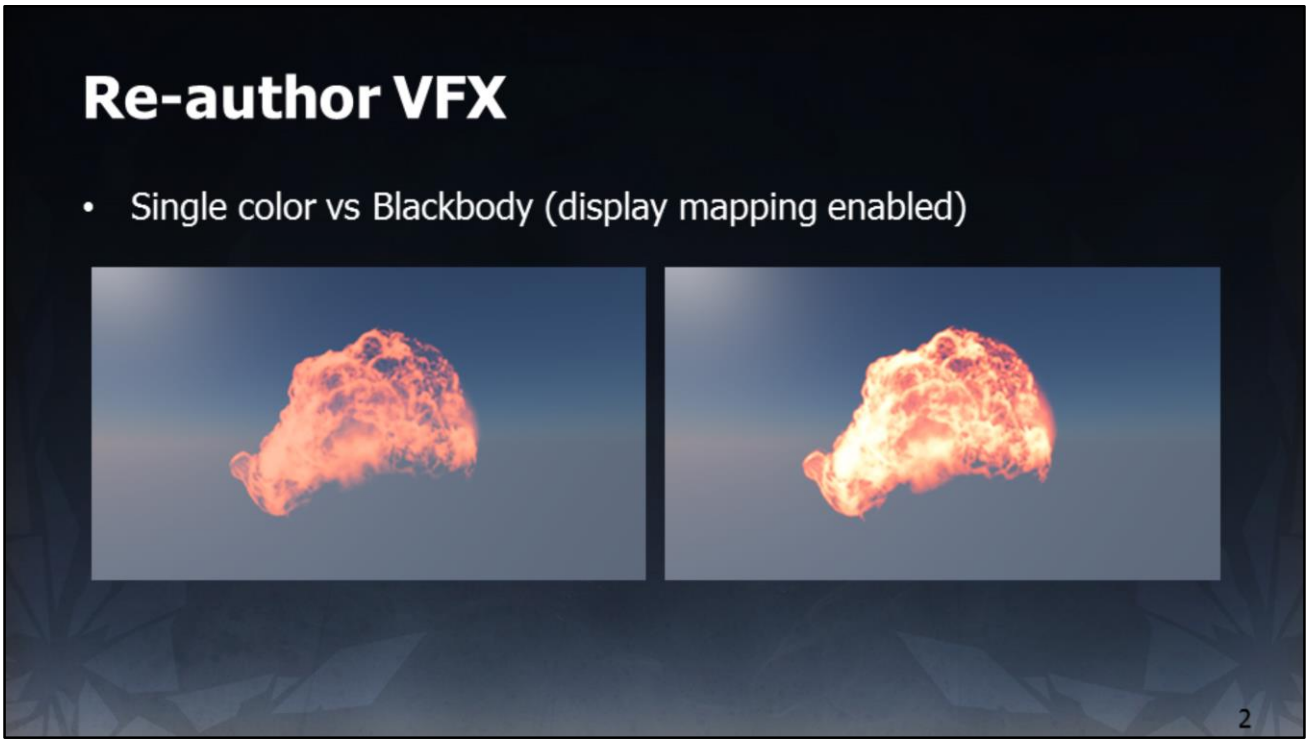


4

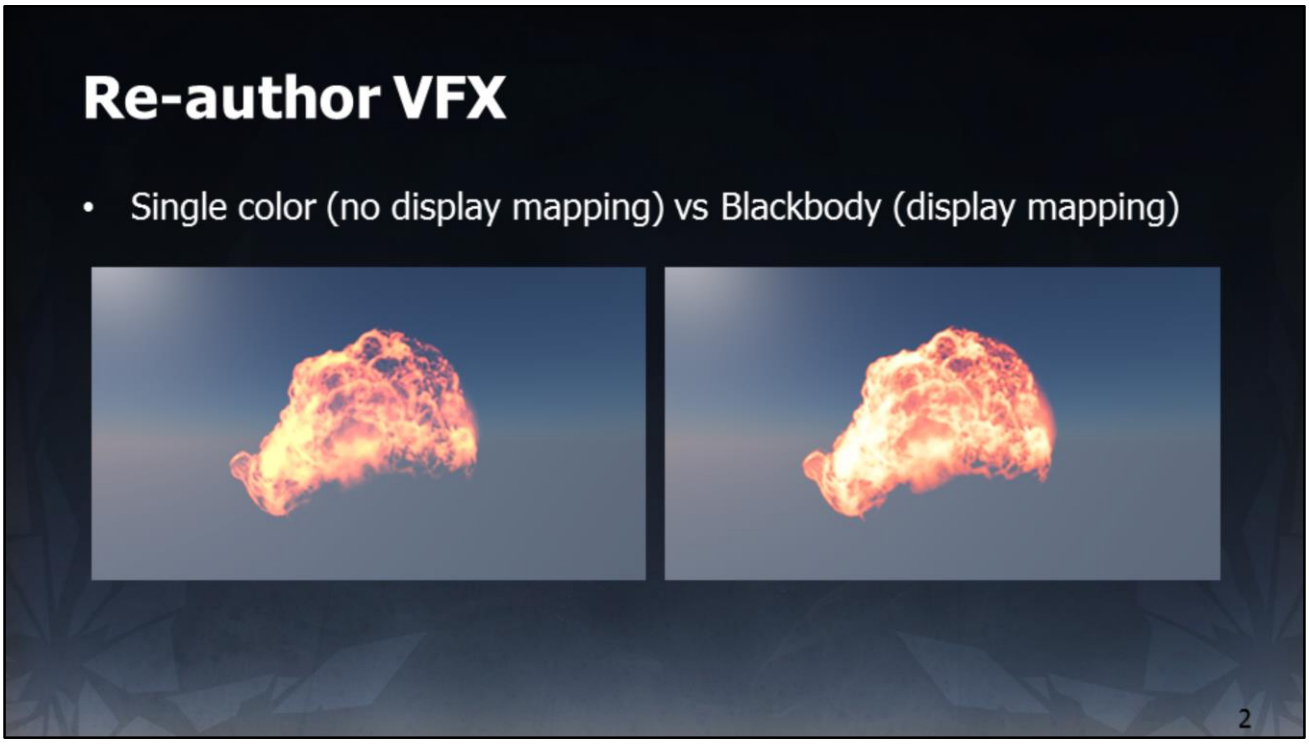
Main point of this – any bright effects will suffer.
Many are authored to leverage the hue shifts that come from clipping color channels (red clips but green doesn't -> effect increasingly becomes yellow as green increases).
These effects do not work well in HDR, as the clip point differs per TV (see notes on previous slide).

Intent must be to author to HDR as the reference, which means the hue shifts must be present in the artwork, not an artefact of the mapper.
Blackbody 'simulation' (temp based hue color lookup) to the rescue.
Iterate the display mapper to preserve saturation of medium/bright effects on SDR devices.

Image credit: https://en.wikipedia.org/wiki/Black-body_radiation



Old asset with display mapping ON (no hue shifts) vs Display mapping ON and hue shifts are correctly present in source asset



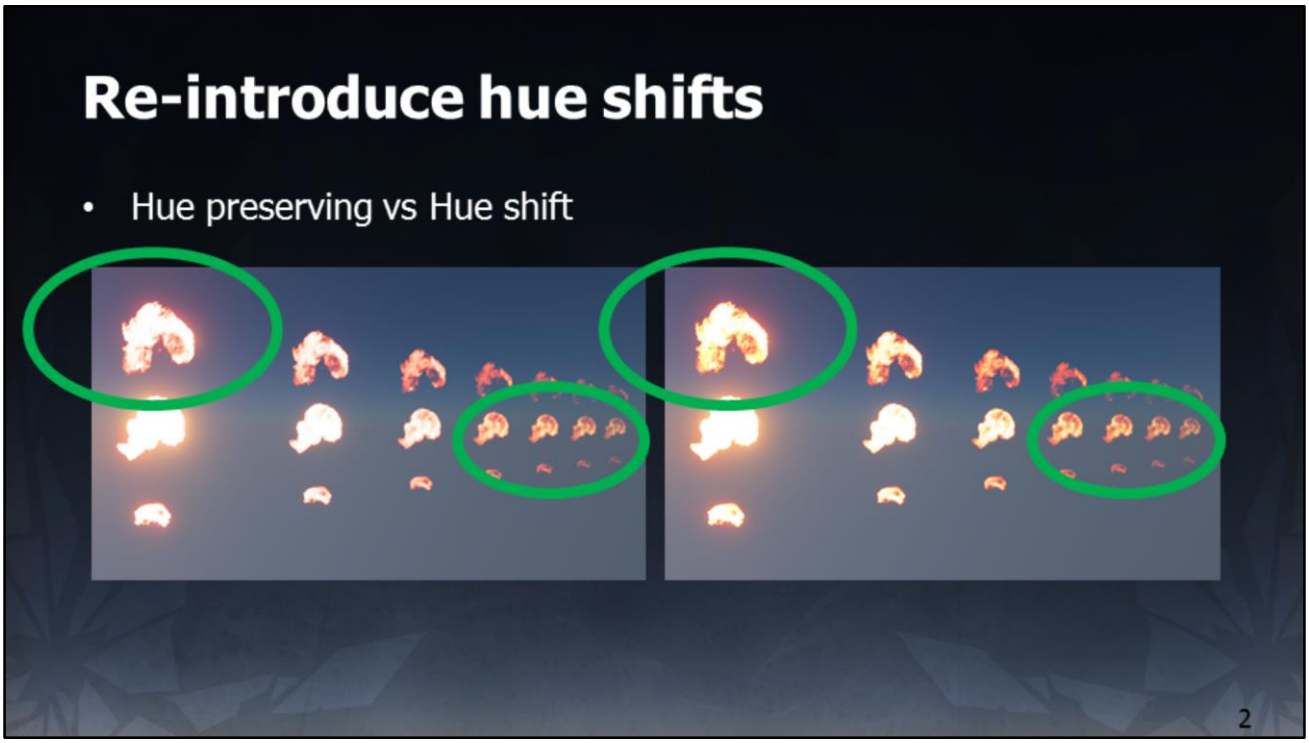
Old asset with display mapping OFF (hue shifts come from channels clipping) vs Display mapping ON and hue shifts are correctly present in source asset

So … plain sailing *now*?

- Er, no …
  - Hue preserving shoulder not always desirable
  - Prevents highly saturated very bright visuals … effects (again)
  - Prevents matching the look of certain films
- Working with DICE effects team
  - Still iterating on display mapper
- Re-introduce hue shifts

4

Hue preserving operator is also troublesome if you are trying to match a “look” that is close to a certain film stock.
Must be able to do that.
Have started to work on re-adding hue shifts.
We’re not done yet, are still actively working on this.



Blackbody fire contained all the correct hues …
-See smaller circle

…were getting desaturated when very bright.
-See larger circle

Original display mapper on the left, small hue shifts (but massive improvement) on the right.

Re-introduce hue shifts

- Hue preserving (left) vs Hue shift (right)

2

Hue preserving on the left.
Re-introduction of some hue shifts on the right.

Re-introduce hue shifts

- And against the original with no display mapping (left)

2

No display mapping (tonemap disabled) on the left, hue-shifting display mapper on the right.
Even with some hue shifts, still dramatically better than no display map.

Re-introduce hue shifts

- We're not done
  - But getting there
- Games shipping this year will have different implementations
  - Some prefer hue preserving
  - Some prefer hue shifts
  - Likely to offer content-driven mixing
- Easy to do
  - Have graded an [HDR] reference implementation
  - Simply tweak to suit different displays

2

As mentioned, still not done.
Likely to offer a configurable implementation, that each game can dial in the amount of hue shifting.
Really easy to do though (no re-grading etc) due to the wide HDR working space and display mapping right at the end of the frame.

But what about ACES?

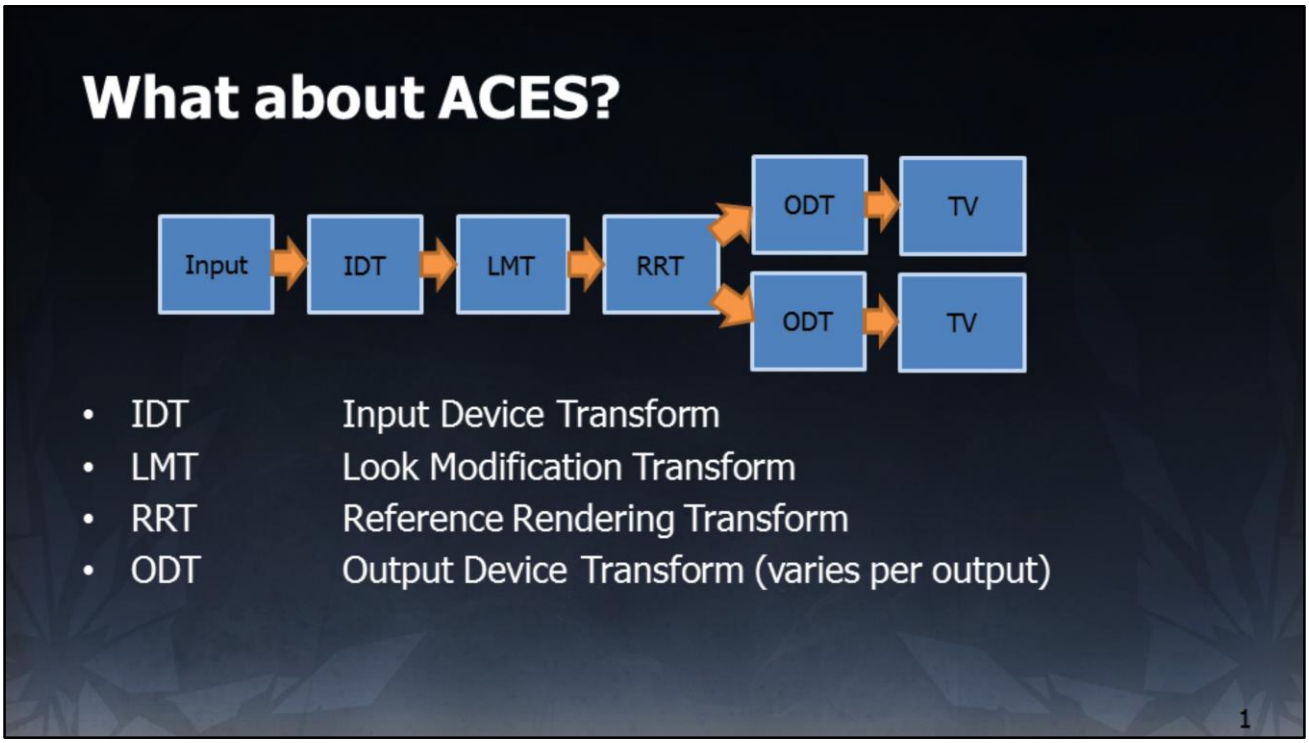
What about ACES?

- Academy Color Encoding System

- Standardised color management for film CGI
- Defines a processing pipeline
- Includes "look" and display mapping

1

http://www.oscars.org/science-technology/sci-tech-projects/aces
Image credit:
http://www.oscars.org/sites/oscars/files/styles/hero_image_wide_default/public/aces_main3.png?itok=inY3Kf2j



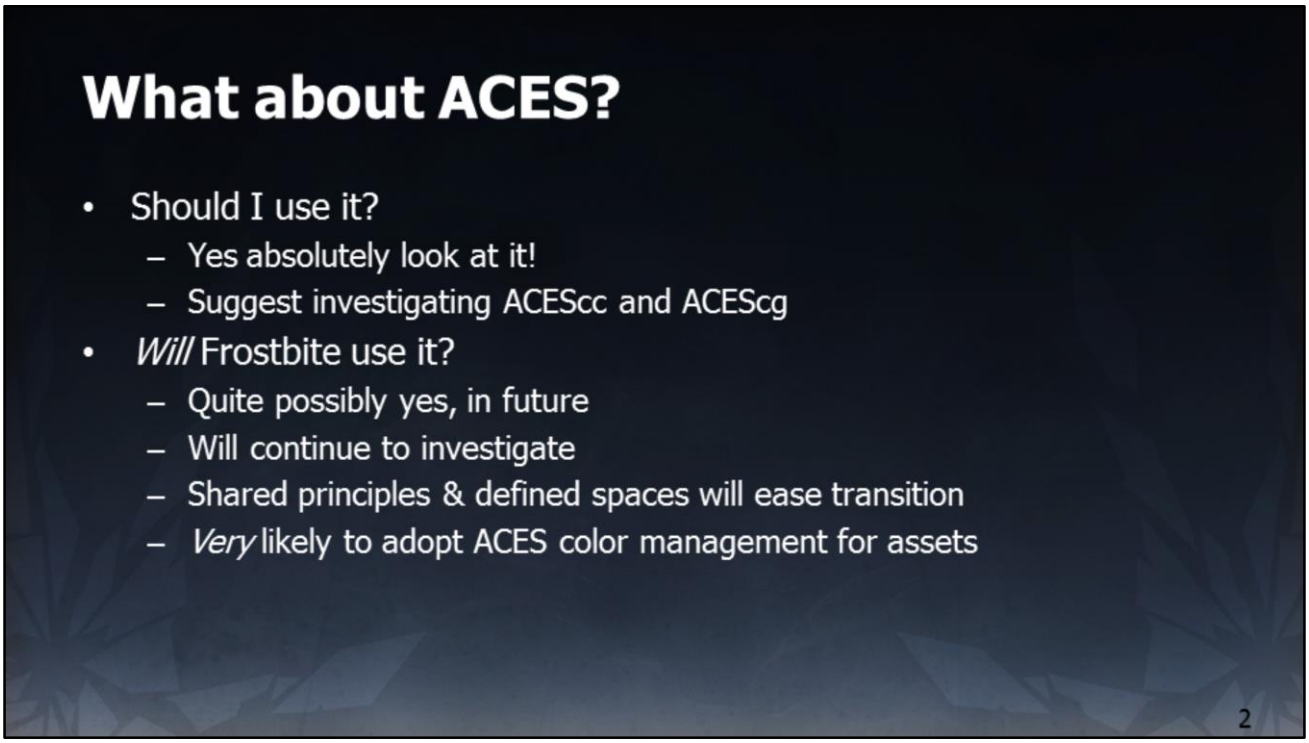
High level summary of the ACES pipeline.

IDT (Input Device Transform) transforms a known input space to the ACES working space.
LMT (Look Modification Transform) is where one applies the grade & “look”
RRT (Reference Rendering Transform) is essentially a filmic tonemap (an S curve applying a toe and shoulder)
ODT (Output Device Transform) is a per-display output transform to ensure consistent results on that display.

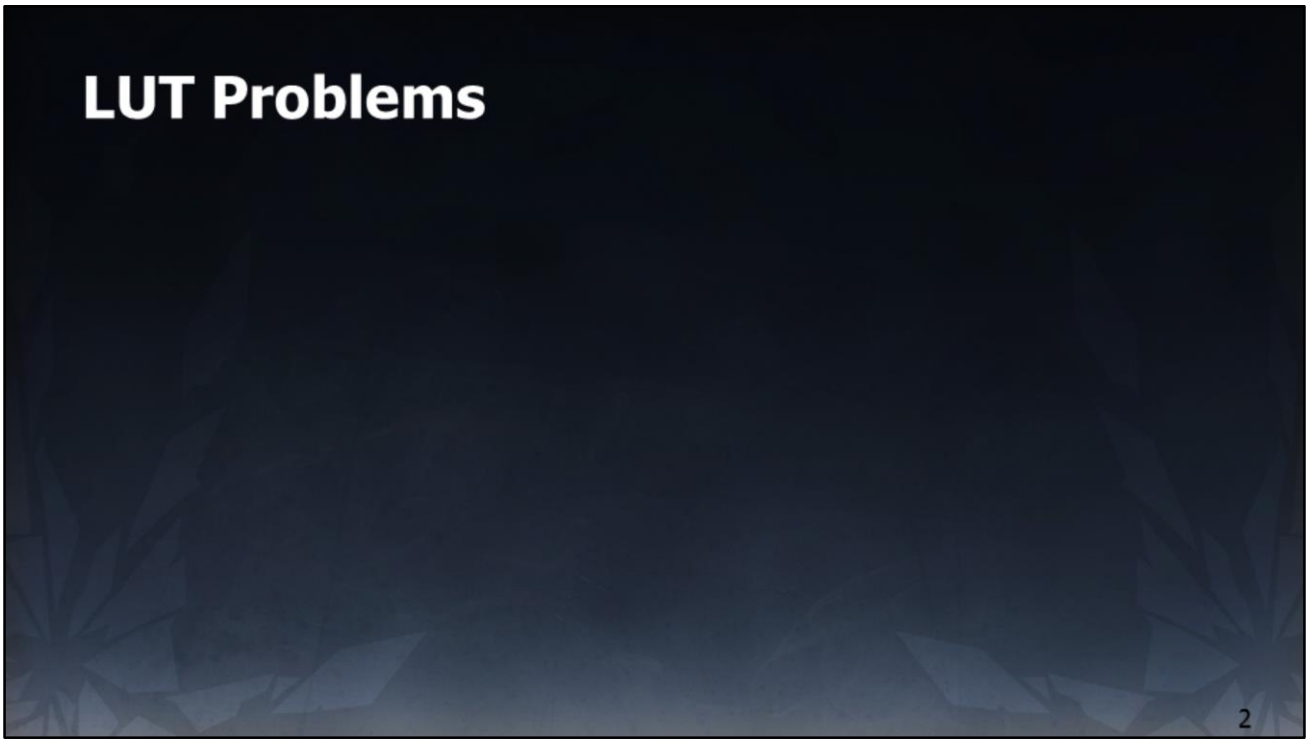
What about ACES?

- Why didn't we use it as-is?
  - We started work in 2014, ACES was only just getting going
  - Early versions required FP16, not suited to some texture formats
  - Was not convinced by the "filmic" RRT
- But we agreed with the principles so we used it in *concept*
  - Order of operations
  - Used a wide 'master' working space
  - LMT = grading
  - ODT = display mapping

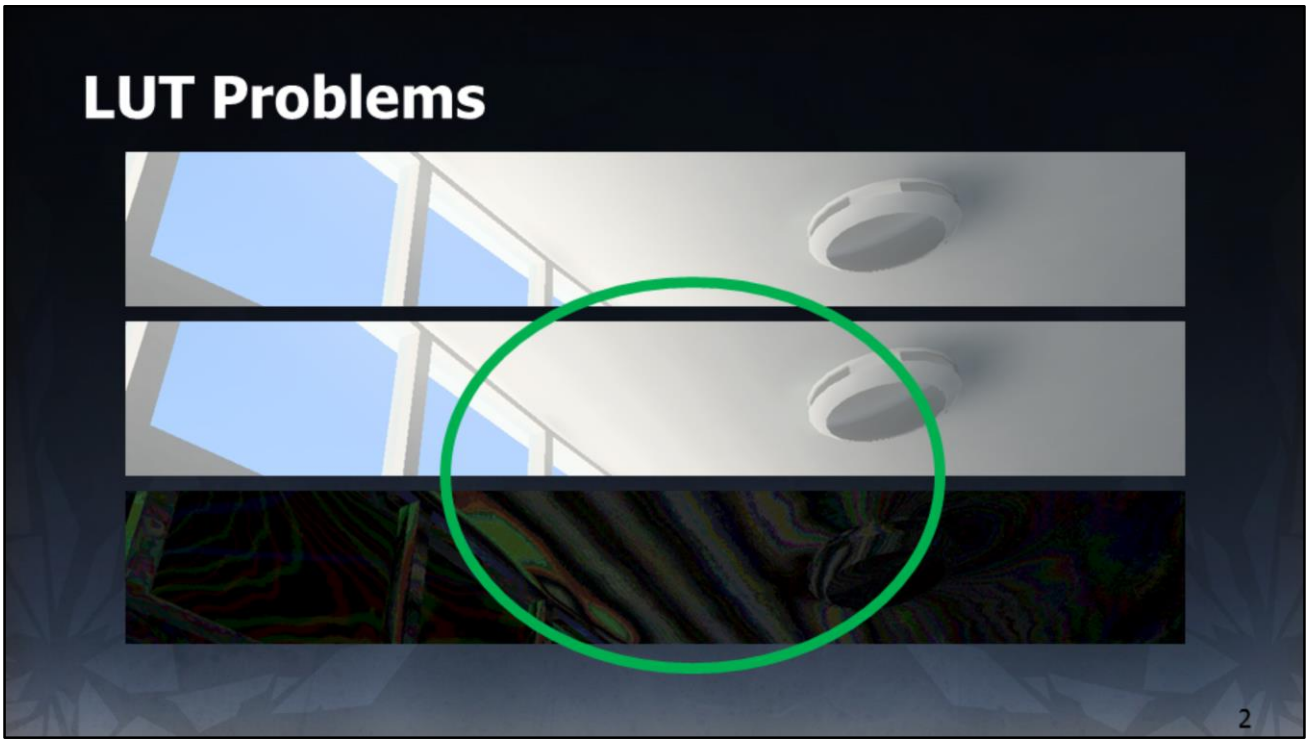
2



Due to the fundamentally aligned approaches, there is nothing stopping us from changing from our custom approach to ACES (ACEScc/cg in particular) in future, and all grades can be automatically upgraded/converted since both spaces are known and published. We can and will re-evaluate this in due course.



Now we look at some performance/quality tradeoffs.
For performance reasons we wanted to dynamically inject the display mapping into the same LUT as the grading.
This has some implications that we investigate here.



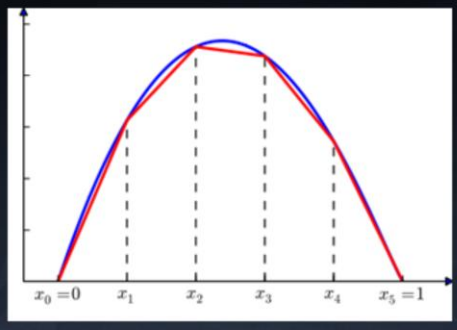
For performance reasons we wanted to dynamically inject the display mapping into the same LUT as the grading.
The order of operations of combining the display mapping at the end of the grading, is the same as doing it at the start of the composition shader. It's just "free" (aside from the cost of injecting it into the grade, which is very cheap).

However, even with PQ distribution we have precision issues which masquerade as mach banding.
Top image is analytical display mapper, middle is baked into the RGB LUT, bottom has levels adjusted to highlight the differences.

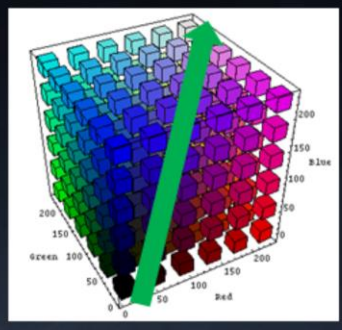
Note: screenshot is from a level purchased from Evermotion, not authored by Frostbite.

LUT problems

Piecewise Linear Approximation



3D LUT: 8 neighbors contribute



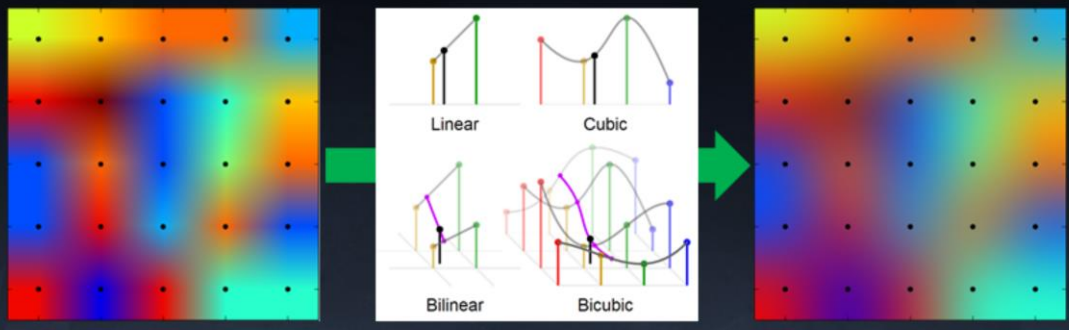
2

Show linear filtering turns curves into piecewise linear approximations of curves.
Looks OK in 1D, where only pairs of values contribute.
But our LUTs are volumes (3D).
The luma axis (greyscale from black to white) is a diagonal, so exactly half way between texels 8 neighbors contribute equally.

Image credit: https://en.wikipedia.org/wiki/Piecewise_linear_function
Image credit: https://engineering.purdue.edu/~abe305/HTMLS/rgbspace.htm

LUT filtering

- Just use higher order filtering?



- Costs too much ☹

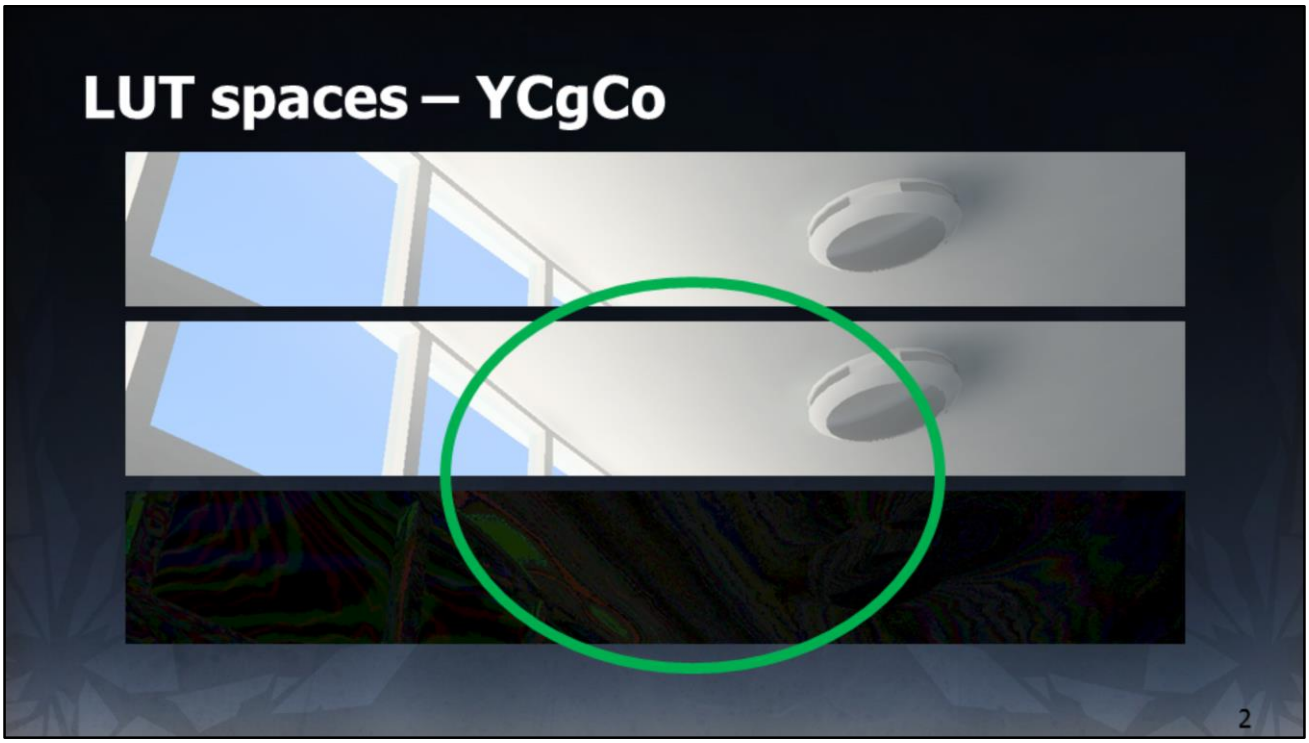
2

Using a higher order filter should improve things.
E.g. move from trilinear to tricubic.
But the cost is expensive (additional dimension over 2D).
Early tests of high order filtering doubled the costs of our main post process pass (which does a lot more than just grading) so it was immediately prohibitive.

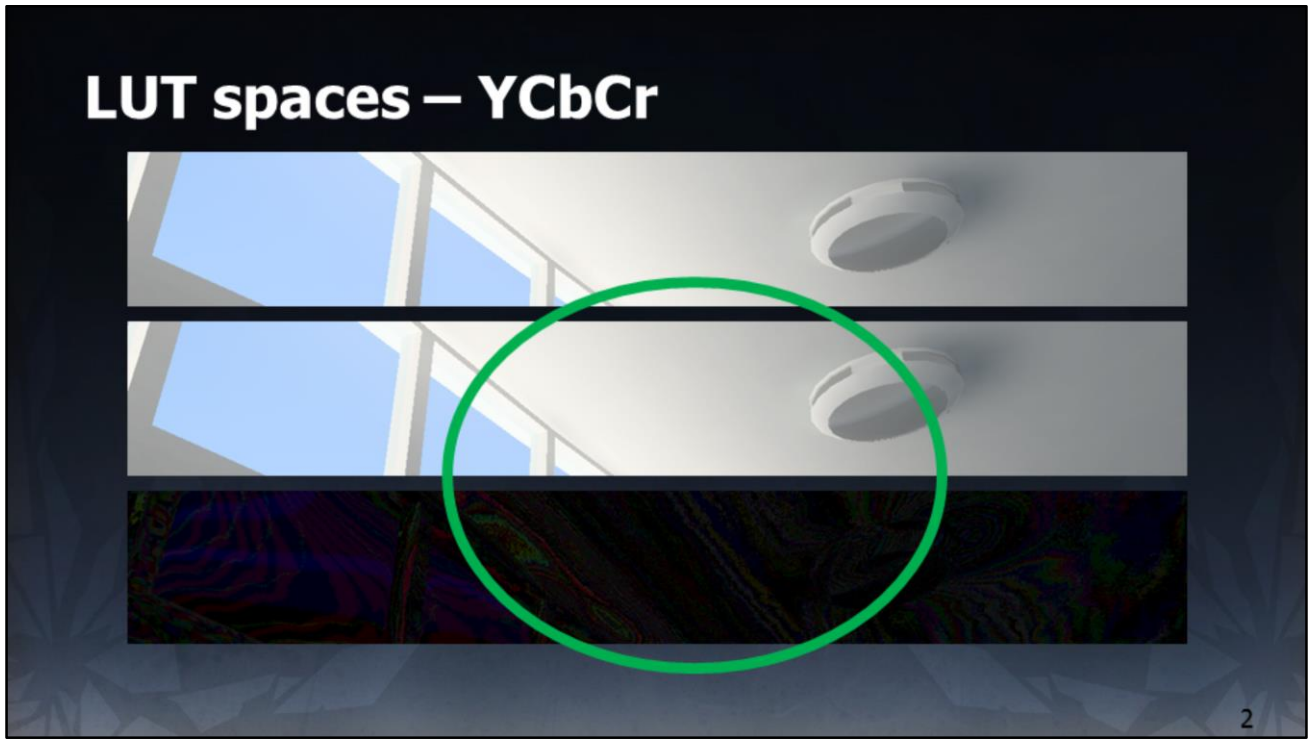
Image credit: https://en.wikipedia.org/wiki/Bicubic_interpolation

But can use different spaces for the LUT. We don't *have* to index by RGB …
In fact the display mapper is working in luma/chroma, so let's try that.

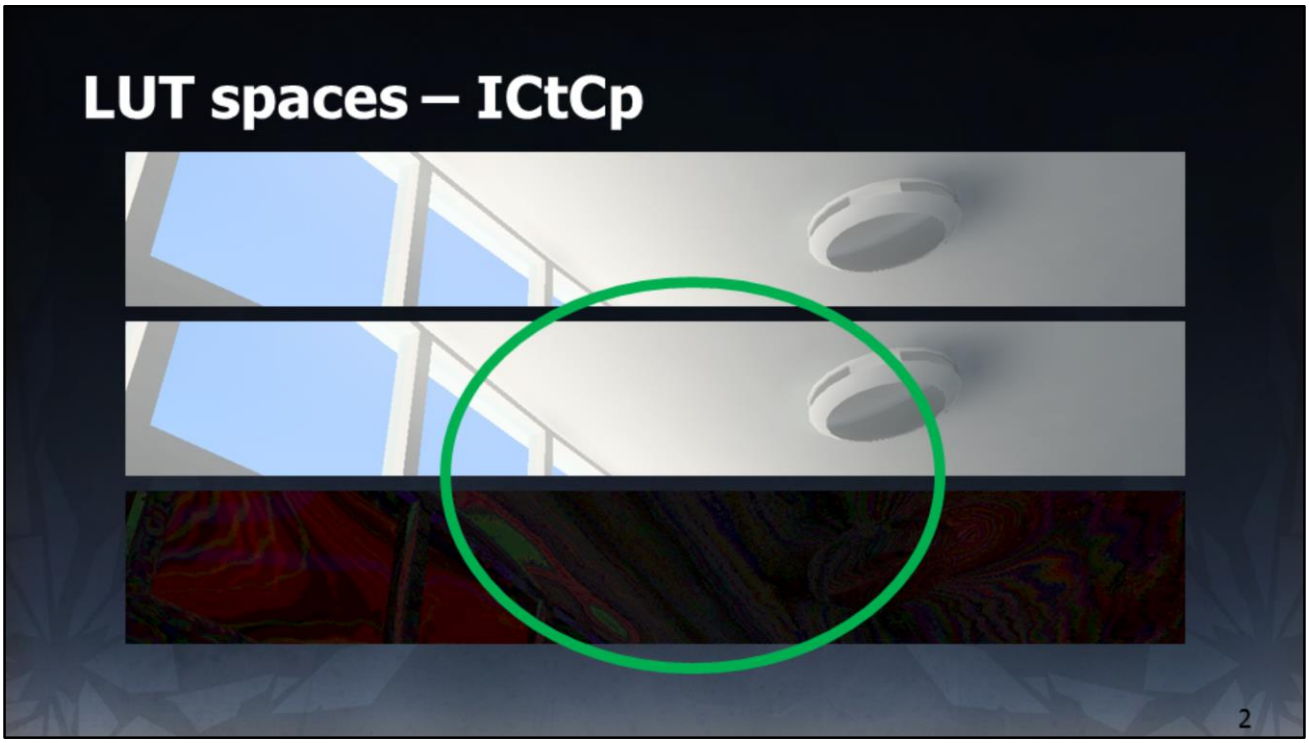
Here's RGB again as a reminder.



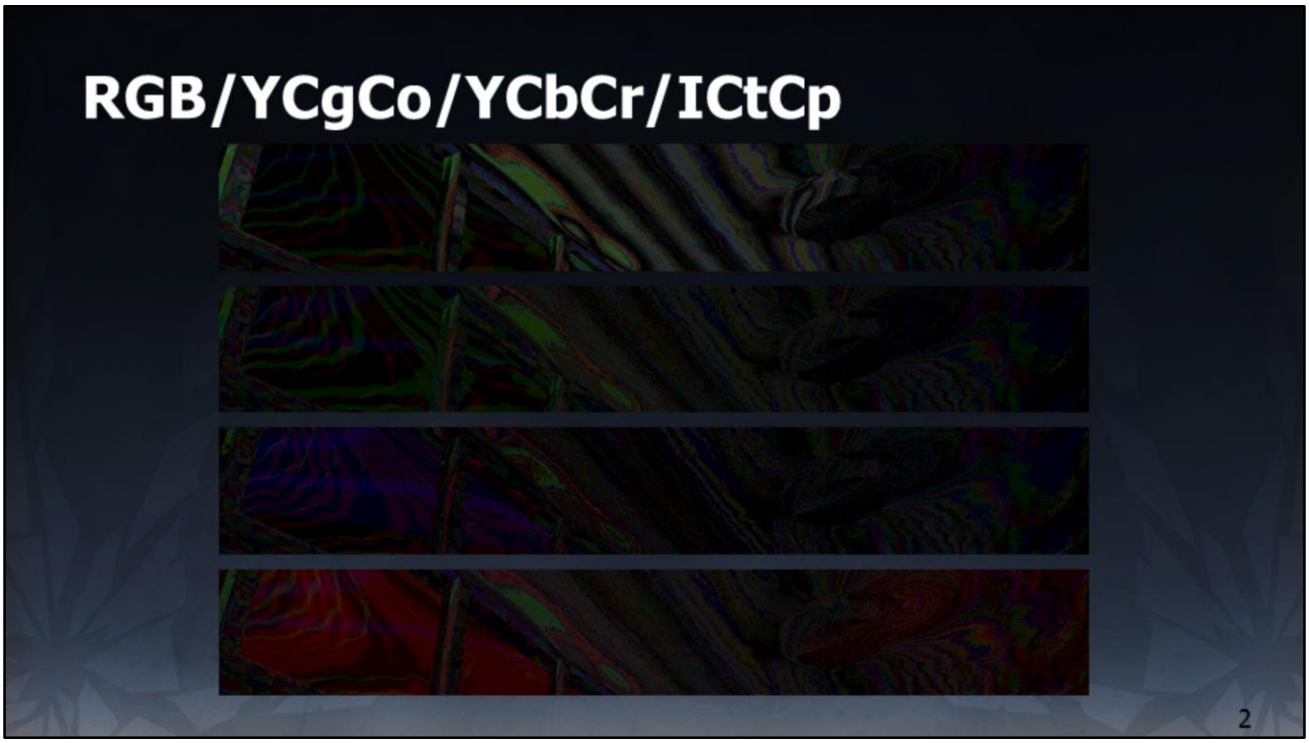
YCgCo – fastest decorrelated space (luma and chroma are separate).
Major improvement on RGB.



YCbCr – better than YCgCo

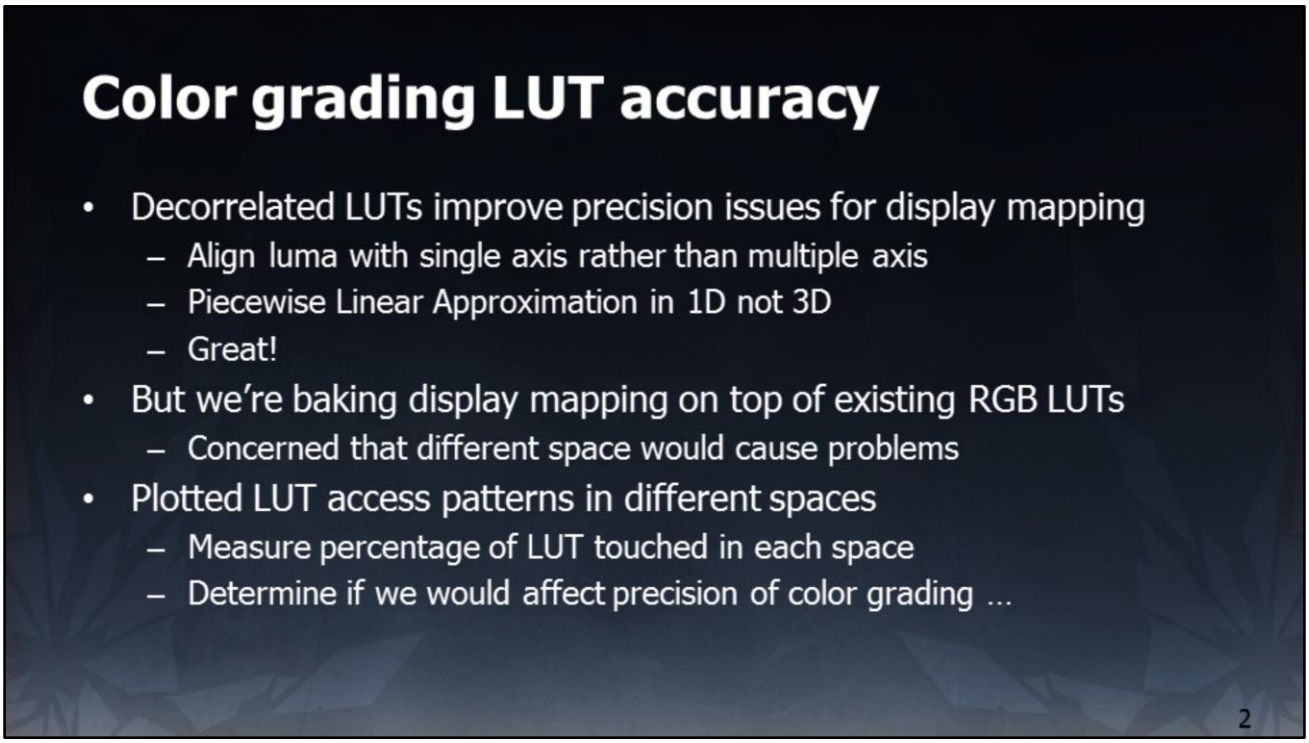


ICtCp – not as good as YCbCr but comparable to YCgCo.
It should be best since it natively matches our display mapping space, but reasons for this will be explained later. Relates to color gamut.



Compare RGB to the three decorrelated spaces.

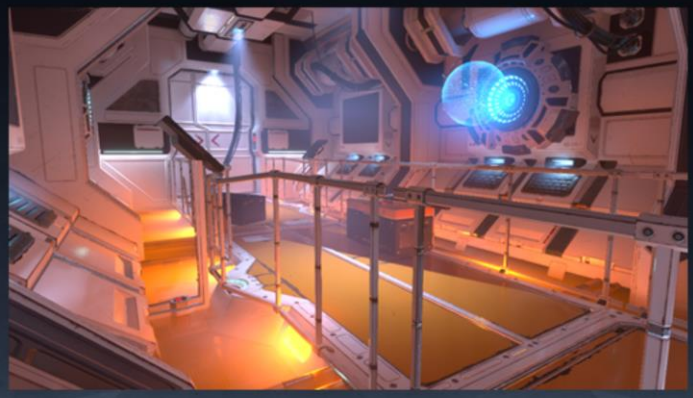
Decorrelated are all better in terms of luma than RGB.
Still using linear filtering but luma and chroma axis are aligned with cubemap axis now.
Luma-only ramps become 1D and touch fewer neighbours so the PLA artefacts are reduced.



So this is a major improvement and allows use of linear filtering without obvious artefacts.
But, will it impact the grades themselves, which are authored in RGB?

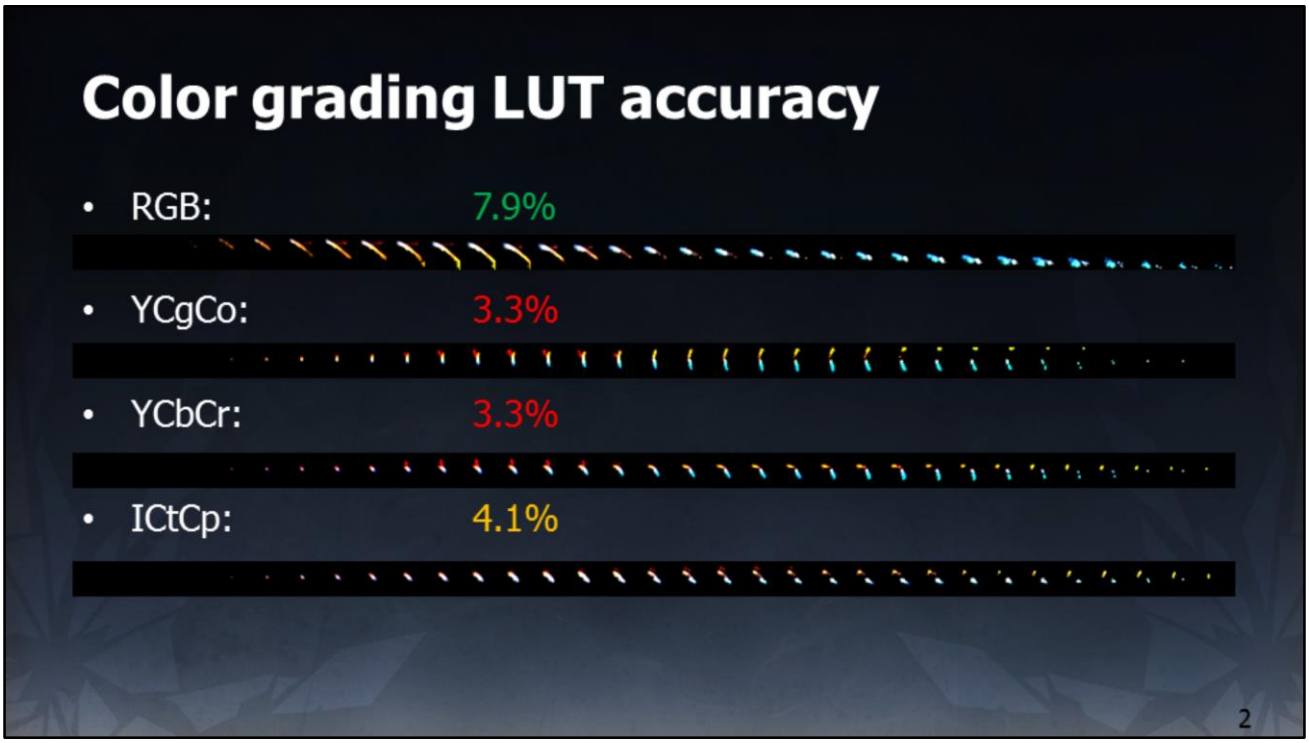
Color grading LUT accuracy

- Reminder



1

Again, we look at this test image.
By plotting each pixel into each LUT and tracking the coverage, we can easily compute the LUT volume used, as a percentage of the total number of texels that exist in the LUT.
Basically, more texels used = more precision for grading.

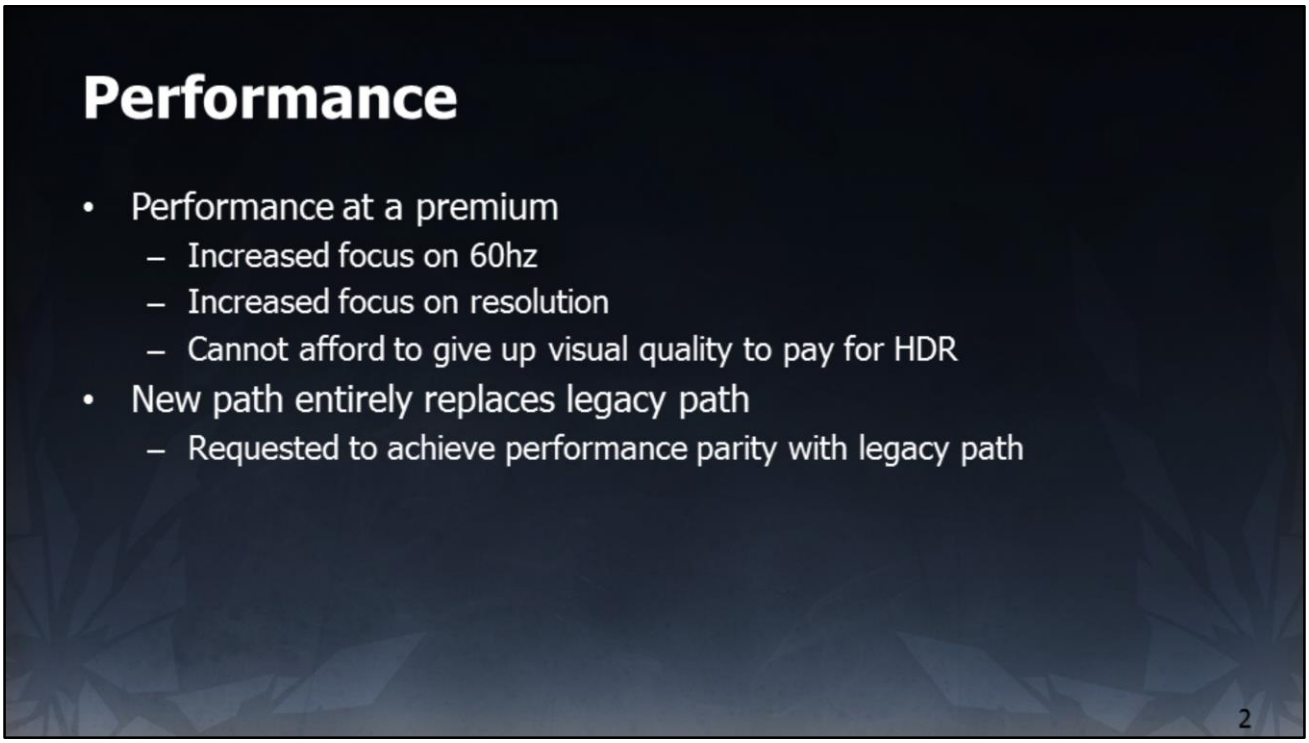


Ah.
Decorrelated spaces fundamentally touch fewer texels, which is likely to have an impact on color grading accuracy ☹
ICtCp is better than either YCC format, but still not as accurate as RGB.

So today we have stuck with RGB (the visual artefacts are minimal, after all) but we are continuing to investigate decorrelated spaces for the future.
Perhaps we will transform RGB into a decorrelated space in the offline pipeline, using most appropriate high order filter.

Performance

2



Performance parity was needed with the legacy path in order to achieve quick adoption. Performance timings from Xbox One as that was the slowest platform.

Legacy end-of-frame path:
Separable Resample from sRGB render target to 1080p backbuffer, via ESRAM transient on XBox: 0.43ms (Xbox is slowest platform, so is the one we use for timings).
UI draw on top of backbuffer: Arbitrary, typically 0.2-0.3ms.

Performance

- Performance of legacy path
  - At the end of the frame we already do a high order resample
    - Dual pass (V/H)
    - Resample vertically to intermediate (ESRAM): 0.17ms (720 -> 1080)
    - Resample horizontally to backbuffer (DRAM): 0.26ms (1280 –> 1920)
  - Draw UI on top
- Total: 0.43ms + UI
  - But UI costs a lot since it's drawn to DRAM not ESRAM
  - NOTE: Numbers all from XBox One

2

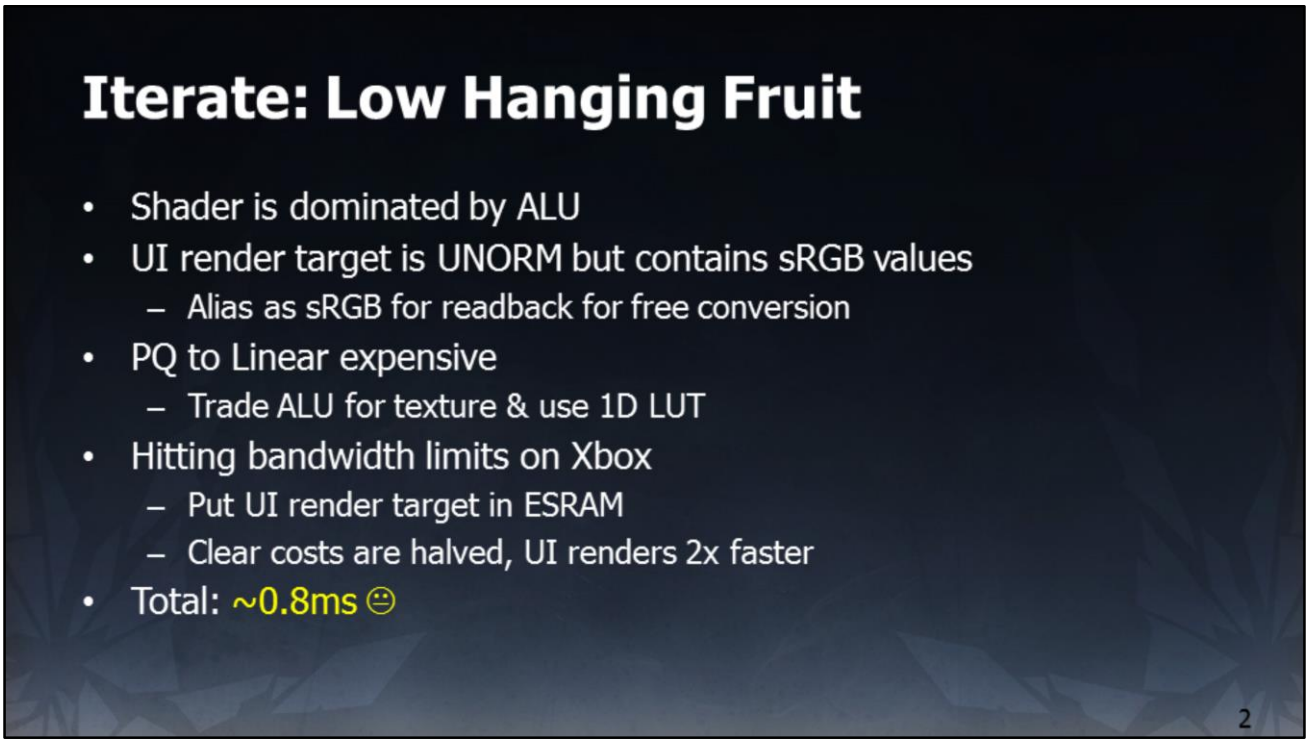
Version 1: Naïve implementation

- Render UI to offscreen RGBA8 target at backbuffer resolution
  - Must clear UI target first

  +0.25ms
- Modify *second pass* of dual-pass resample
  - Resample HDR & convert PQ to linear
  - Load UI target & convert sRGB to linear
  - Composite HDR & UI
  - Encode to sRGB

  +0.45ms
- Total: ~1.1ms ☹

2



ESRAM used for both planes (intermediate & UI). Manual ESRAM management *
Also roughly doubles speed of UI rendering, reaping benefits not shown here.

* See “FrameGraph: Extensible Rendering Architecture” talk for how we are moving to automatic ESRAM management

Iterate: Compute resample

- Use single-pass CS for resample
  - Resample vertically to groupshared
  - Resample horizontally from groupshared
- Optimisations (some GCN specific)
  - 64x1 thread layout (linear tile mode backbuffers)
  - Precomputed kernel weights in StructuredBuffers
  - SMEM/SALU to load & extract vertical weights
- Total ~0.7ms 😐

2

Not super relevant to HDR, this was simply an enabling change.
Moving from two to one pass reduced work and enabled significantly better scheduling.
We use a thread layout optimized for the consoles and also for the linear tile modes used for swapchain/scanout targets.
Use of the 1D vertical thread layout allowed us to use the scalar pipeline present in GCN to obtain some of the filter weights for free.

Iterate: CMASK-aware composite

- GCN hardware maintains "CMASK" metadata for written pixels
- Stores whether any block of 4x4 pixels was rendered to
- Used in "Fast Clear Eliminate" to write clear color to unwritten pixels

2

Actually GCN doesn't always clear all pixels
Maintains metadata for which pixels were written.
Before readback, performs a "Fast Clear Eliminate" (FCE) pass to write back the clear color only to unwritten pixels.

Example CMASK screenshot shows typical in-game UI coverage. Red = pixels written. Black = unwritten pixels that will need clearing.

Iterate: CMASK-aware composite

- We can read this metadata
  - Remove the Fast Clear Eliminate (FCE)
  - Read CMASK in composite pass
  - Skip reading and compositing of unwritten UI pixels
- Not quite that simple
  - CMASK tiles are tiled & packed
  - Software de-tiling and unpacking needed
  - Not cheap ☹

2

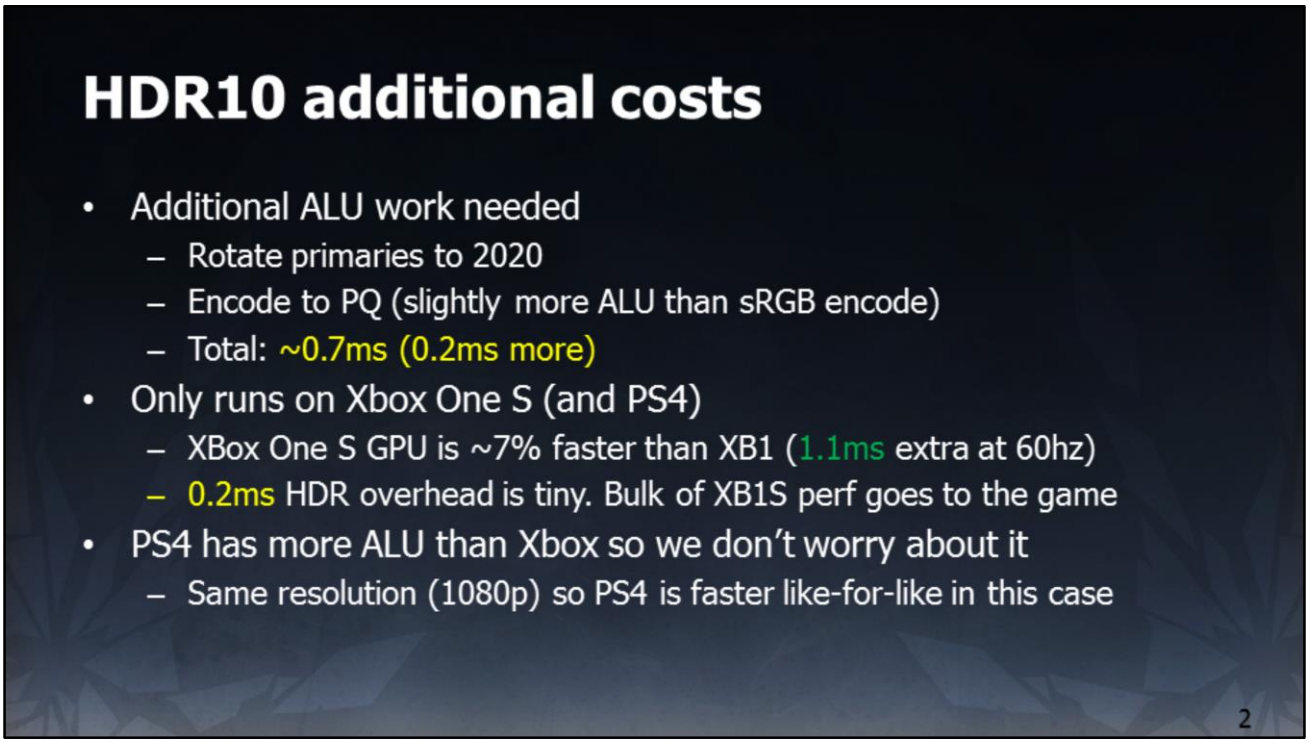
No FCE, instead use a custom CMASK transcode from sub-tiles to a 32x1 bitmask designed to be loaded via the scalar pipeline ‘for free’ on the final resample/merge/displaymap pass.

Iterate: CMASK-aware composite

- Transcode CMASK via compute to composite-friendly format
  - 1 bit per 4x4 pixels
  - 32 bits per 128x4 pixels
  - Adds a cost of 0.02ms but *much* cheaper than FCE
- Composite pass
  - Load 32bit bitmask via SMEM (free)
  - Unpack bit corresponding to current pixel (cheap)
  - If bit is zero, skip entire UI load & composite operation
- Total: 0.5ms ☺

2

No FCE, instead use a custom CMASK transcode from sub-tiles to a 32x1 bitmask designed to be loaded via the scalar pipeline ‘for free’ on the final resample/merge/displaymap pass.



All timings so far have been from the SDR version on base Xbox One, since base Xbox One only supports SDR and SDR output will be the most common path for a while. However, Xbox One S supports HDR10 and this incurs some additional costs to encode. Xbox One S is faster though, so these overheads are well within the extra performance.

HDR standards & platforms

HDR standards

UHD ALLIANCE ULTRA HD PREMIUM™

- Two primary standards
  - Dolby Vision
  - HDR10
- Frostbite can support both * & supporting more is relatively easy

2

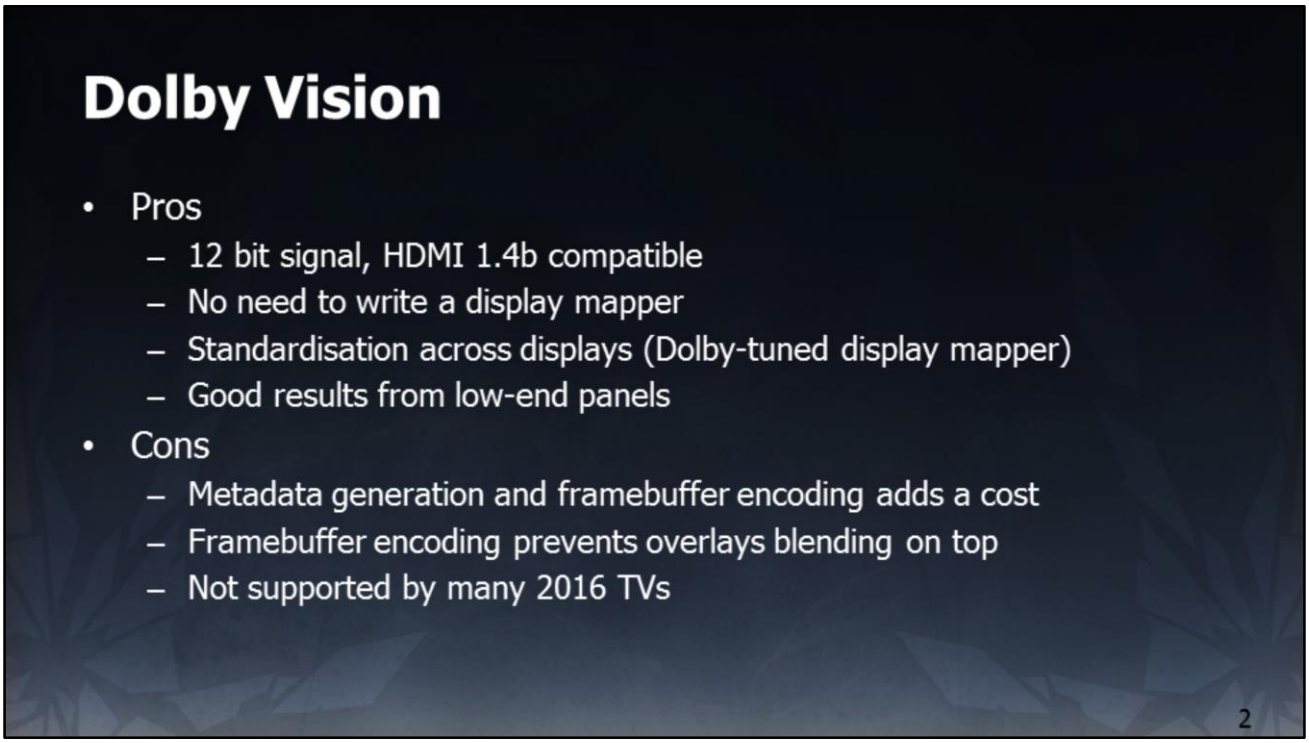
Two primary (but very similar) HDR standards; likely more are coming

Frostbite can and will target any standard that makes sense

* It is down to each game to negotiate licensing though so I can't say anything about specific games

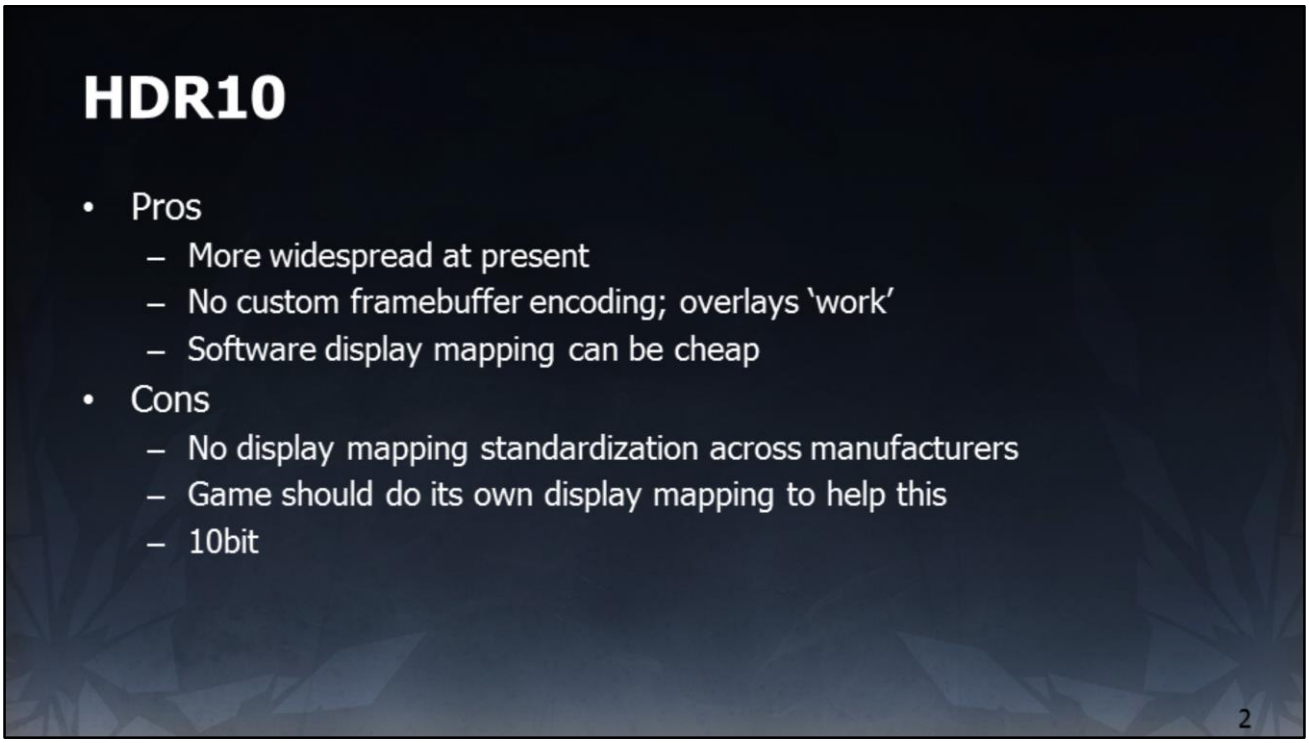
Our display mapping "scanout shader" allows easy plugin of any format, it's just another encoding/mapping.

Image credit:
https://s.aolcdn.com/hss/storage/midas/e94dc10a6894b46ee659c6136fd7040c/203213453/dolbyvision.jpg
Image credit: http://edge.alluremedia.com.au/m/g/2016/01/ultra_hd_premium_2.jpg



Essentially:
Dolby Vision uses a custom encoded framebuffer.
One must generate dynamic metadata (e.g. Min/max/avg luminance) & send it up to every frame to the TV.
Dolby build a custom display mapping for each panel to get the best from it, and achieve standardization of look across displays.

Can't go into any more details here, suggest contacting Dolby if you are interested in supporting Dolby Vision.
https://www.dolby.com/us/en/technologies/dolby-vision/dolby-vision-white-paper.pdf



Game can look quite different on each HDR10 TV, due to lack of standardization across manufacturers.

Commonalities

- Share an EOTF (or 'gamma curve')
- Share min, max & mastering luma
- Share color gamut
- Same 'master' content works on both
- One display map target for each

2

Lots of high level commonalities … try to support both.

Platform support

- PS4 & PS4 Pro: HDR10
- XBox One **S**: HDR10
- PC: HDR10, Dolby Vision
  - Requires GPU vendor API to handle HDMI metadata
  - DX11: Exclusive fullscreen "just works"
  - DX12: *Non-exclusive* fullscreen
    - Desktop compositor can scale or add overlays
    - Can cause issues with Dolby Vision
    - Working with various parties to improve this

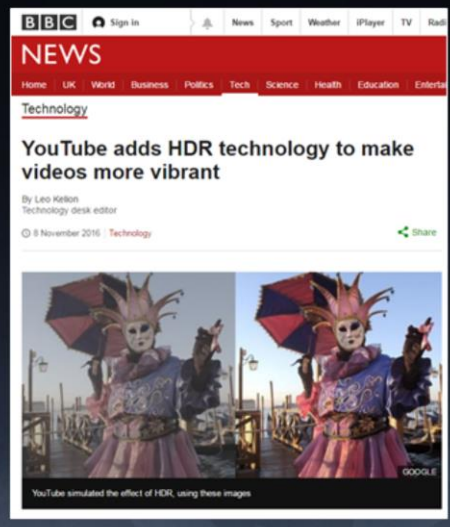
2

Huge number of SDR devices out there.
HDR must not be worse than SDR!
HDR is the reference in Frostbite; SDR is just an artefact of the display mapper.
Recall that SDR TVs scale the image up in terms of gamut and luminance.
No real control over this, though having HDR in the engine and being able to map to SDR helps get the best from each display.
This includes re-exposing the image as part of display mapping (under-expose it so that the SDR TV can re-brighten it again).

Next steps

HDR video

- High bit depth video
  - Decode performance overheads
  - File size and streaming overheads
- Marketing materials
  - Multiple versions of same video
- Wide Color Gamut support needed



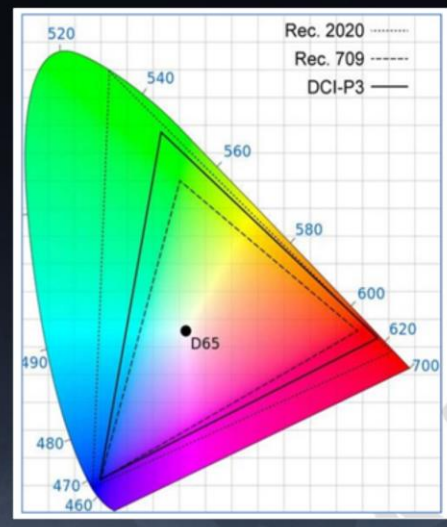
2

HDR movies need work.
Right now we use SDR movies but have a few tricks to extract pseudo-HDR data from them.
True HDR movies need increased storage, streaming, runtime playback costs of high bit depth video.
But the main issue is one of needing wide color gamut support.

Image credit: http://www.bbc.co.uk/news/technology-37908975

Wide gamut rendering

- Lots to do
  - Expand runtime gamut
  - Add gamut metadata on every asset
  - Add color management to editors
  - Preserve metadata between DCCs
- Where to start
  - Work from the TV backwards
  - Convert color grading to wide gamut

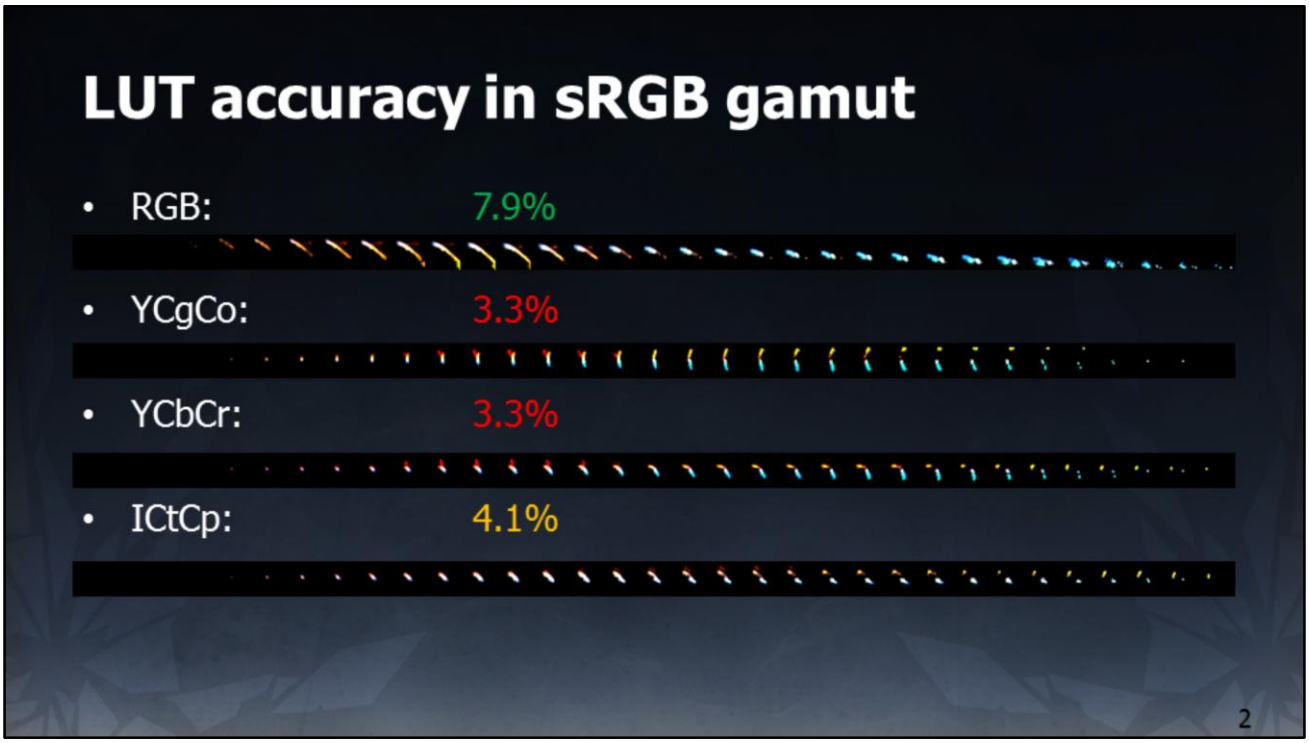


2

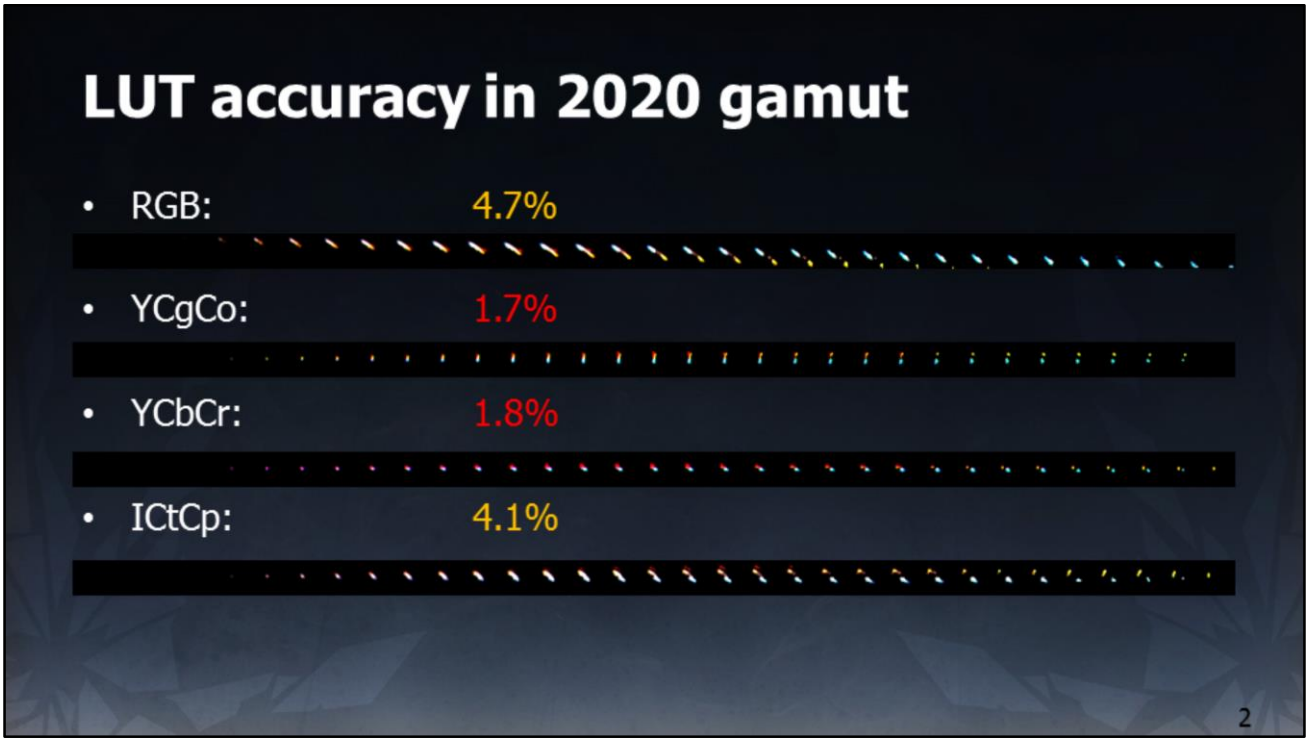
Not “just” rendering – requires a collaboration with multiple parts of Frostbite (data pipelines, import/export, UI, movies, textures, shaders, timeline editors etc – anything with color data needs color management).
Specifically, challenge is related to maintaining and respecting the necessary gamut metadata.
First need to assign and manage gamut on every ‘colour’ asset (textures, colours in shader graphs or timelines, movies etc).
DCC packages may or may not support this; different approaches may be necessary to manage import/edit/export.

Likely to start by upgrading the engine from the “TV back” – first step will be to upgrade the color grading to wide gamut (likely 2020). ICtCp likely necessary for LUTs at this point (see next slides).
Runtime gamut reduction necessary; again we expect to use ICtCp for hue linear desaturation and fold it into the display mapper for free.

Image credit: http://www.acousticfrontiers.com/wp-content/uploads/2016/03/Color_Gamut.png



And we're back here again.
This is a reminder about the LUT accuracy of different spaces, but specifically calling out that we're currently working in the sRGB gamut.



In the future we want to support wider gamuts, so let's test in 2020.
Aha. Not so good now.

RGB, YCgCo and YCbCr all nearly halve in accuracy/volume.
But ICtCp is natively wide gamut so stays the same, becoming more of a sensible choice now.

A note on gamut reduction

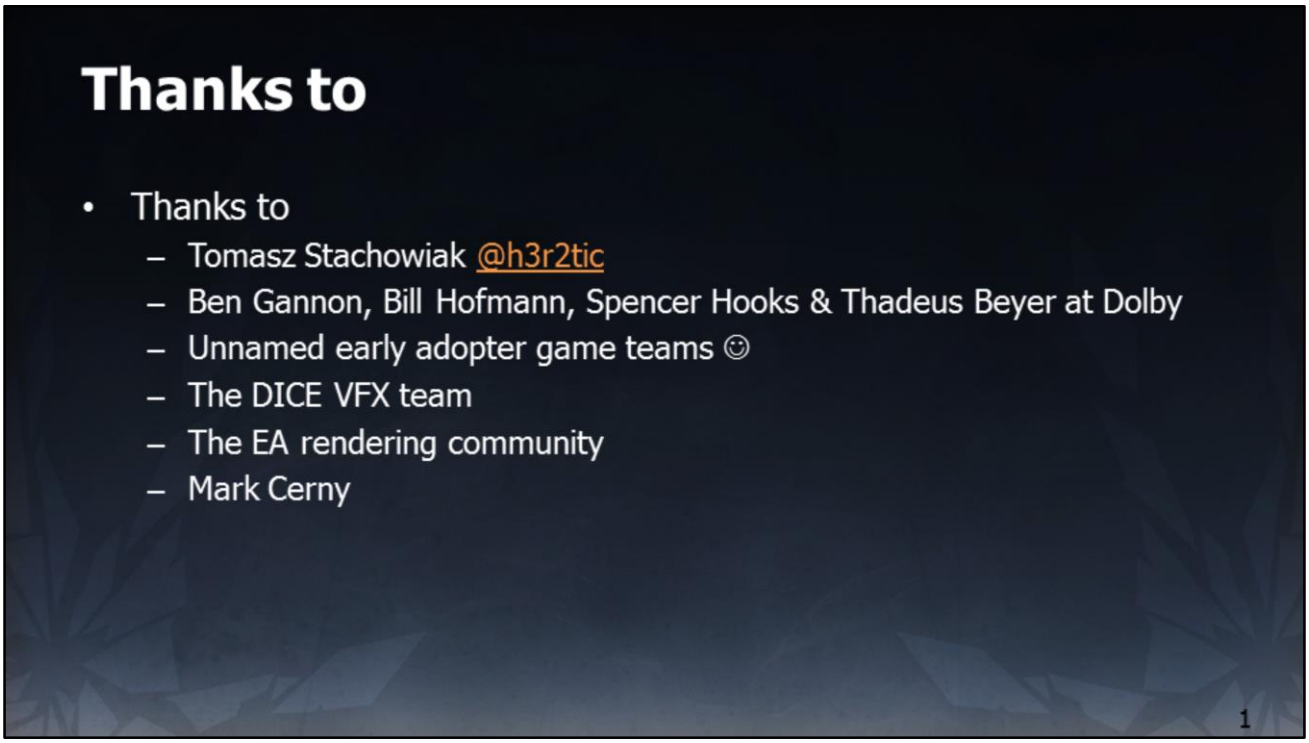
- Gamut expansion is trivial
  - 3x3 matrix multiply in linear space
- Gamut reduction can produce out-of-gamut colors
  - Negative numbers when clipped to 0 cause hue shifts
  - Must map colors to target gamut before 3x3 matrix multiply
- Suggest investigating ICtCp as a working space
  - Chroma scaling is perceptually hue linear
  - Adjust saturation to fit target gamut

Wrap up

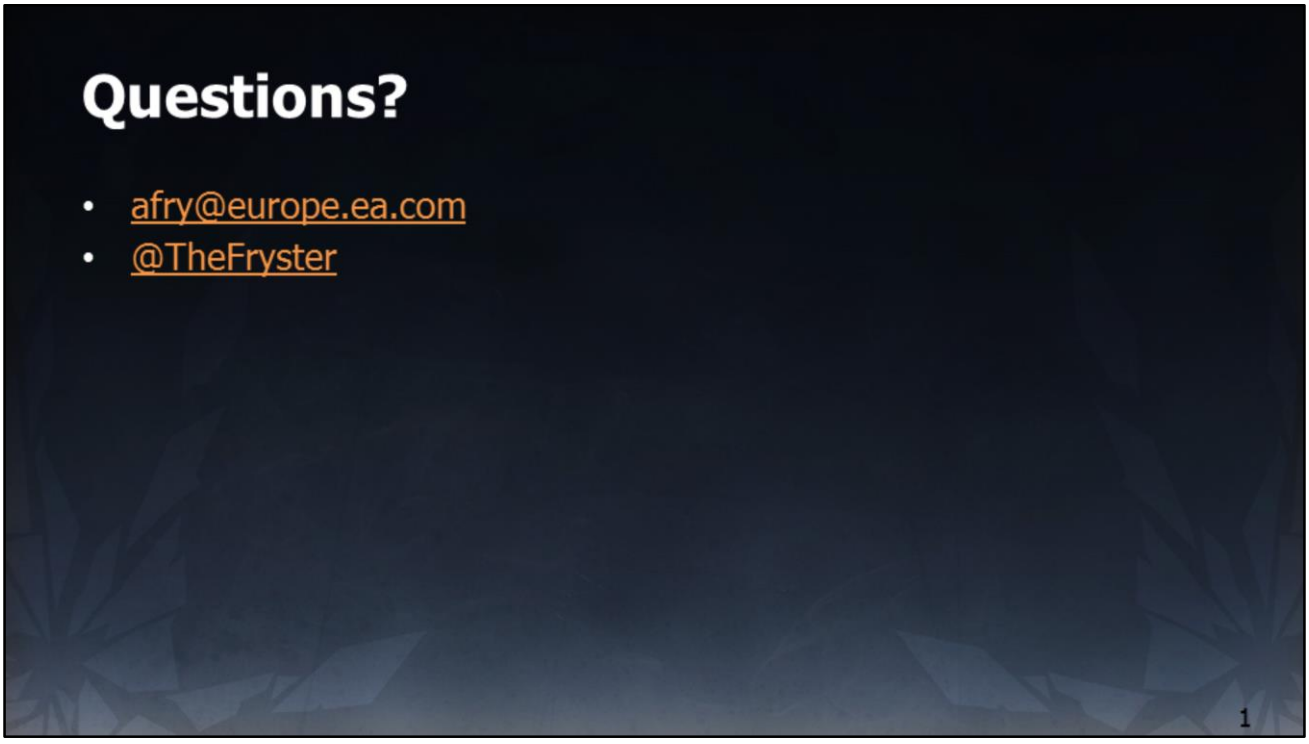
- Ensure your colors are in the assets
  - Don't rely on tonemaps or clip points to change hue
- Master your game in HDR
  - Move your tonemap as late in the pipeline as possible
  - Vary the tonemap for each display
- Consider decorrelated spaces
  - RGB isn't the only way to do things
- Aim to support all standards
  - Please don't forget about SDR ☺

1

It has been an interesting journey, and we're not done yet.
Hopefully our experiences learnt, and this talk, can be of use to someone.

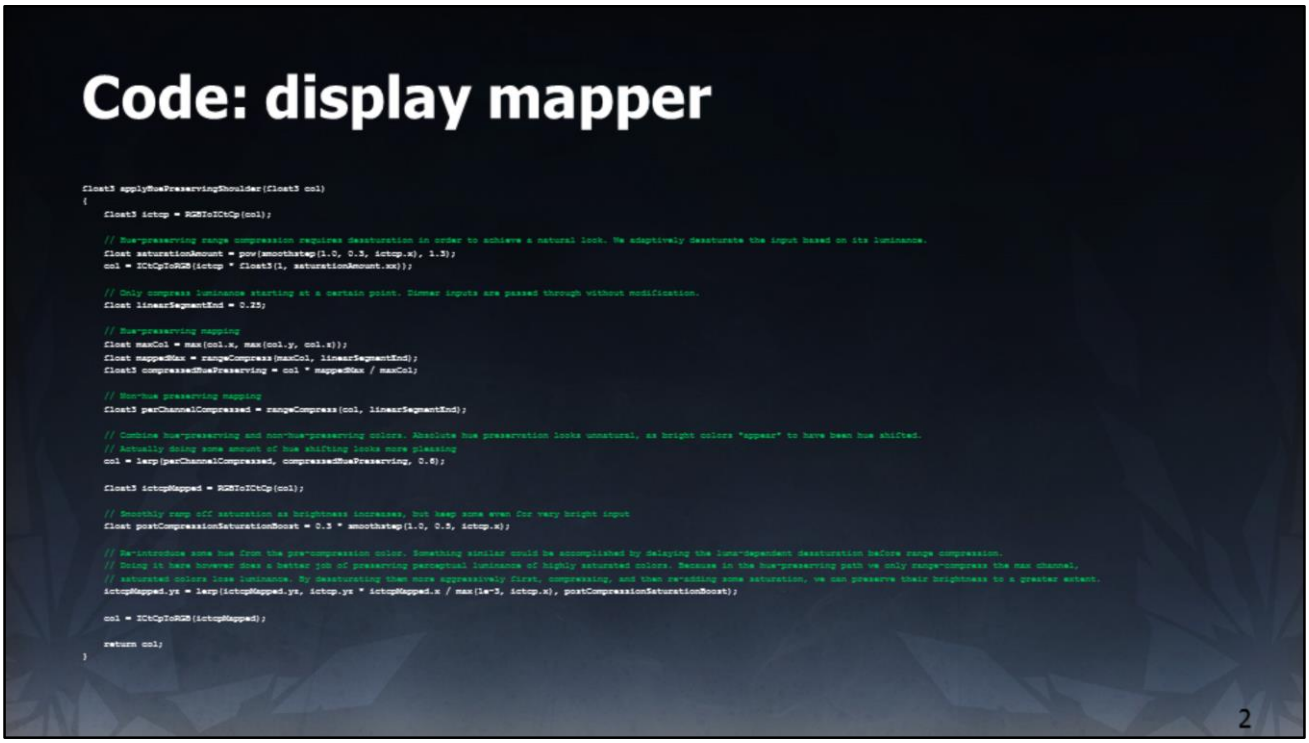


Special thanks and credit especially to Tomasz who is the author of our display mapper.



Contact me if you need:

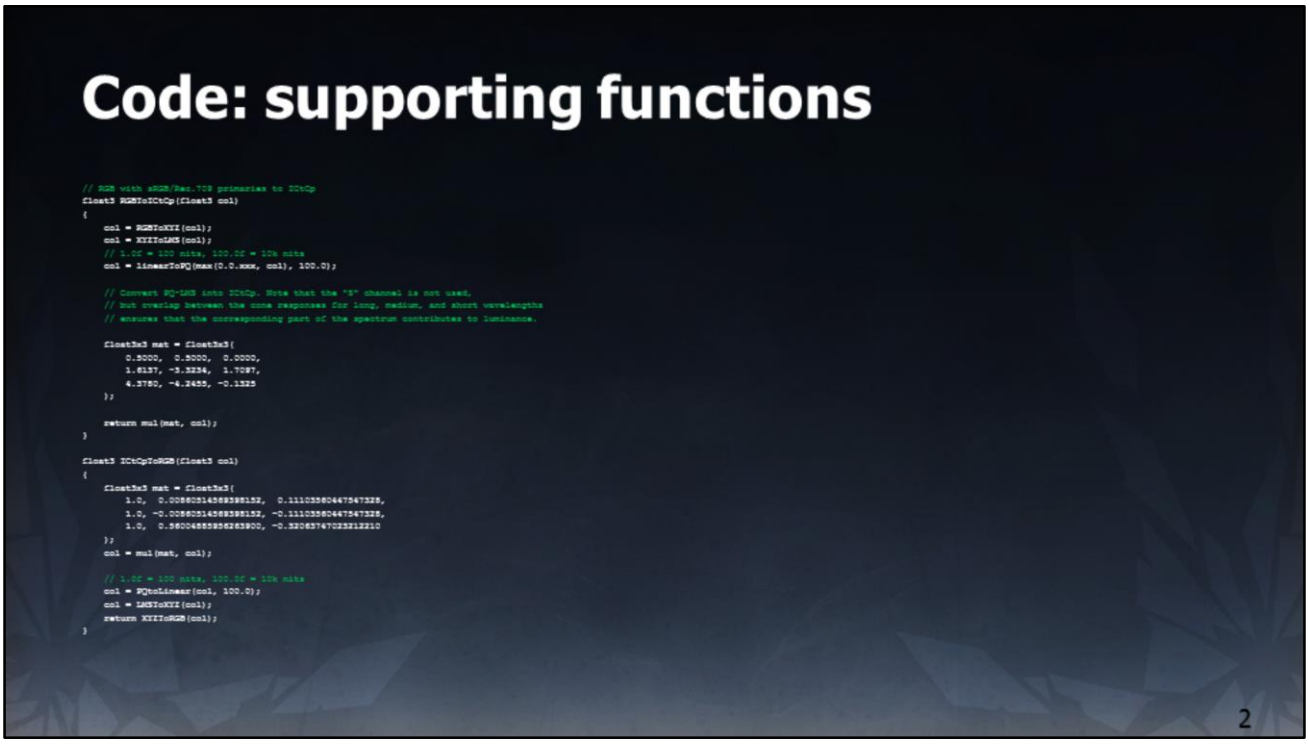
afry@europe.ea.com
@TheFryster



Note: this function is expensive.
This is why we bake it down into a LUT (ideally the same LUT we use for grading, to make it free).
This is the hue-preserving version used for screenshots in this article, it's very ad-hoc but hopefully interesting to play with.

```
float3 applyHuePreservingShoulder(float3 col)
{
    float3 ictcp = RGBToICtCp(col);

    // Hue-preserving range compression requires desaturation in order to achieve a natural look. We adaptively
desaturate the input based on its luminance.
    float saturationAmount = pow(smoothstep(1.0, 0.3, ictcp.x), 1.3);
    col = ICtCpToRGB(ictcp * float3(1, saturationAmount.xx));

    // Only compress luminance starting at a certain point. Dimmer inputs are passed through without modification.
    float linearSegmentEnd = 0.25;

    // Hue-preserving mapping
    float maxCol = max(col.x, max(col.y, col.z));
    float mappedMax = rangeCompress(maxCol, linearSegmentEnd);
    float3 compressedHuePreserving = col * mappedMax / maxCol;

    // Non-hue preserving mapping
    float3 perChannelCompressed = rangeCompress(col, linearSegmentEnd);

    // Combine hue-preserving and non-hue-preserving colors. Absolute hue preservation looks unnatural, as bright
colors *appear* to have been hue shifted.
    // Actually doing some amount of hue shifting looks more pleasing
    col = lerp(perChannelCompressed, compressedHuePreserving, 0.6);

    float3 ictcpMapped = RGBToICtCp(col);

    // Smoothly ramp off saturation as brightness increases, but keep some even for very bright input
    float postCompressionSaturationBoost = 0.3 * smoothstep(1.0, 0.5, ictcp.x);

    // Re-introduce some hue from the pre-compression color. Something similar could be accomplished by delaying the
luma-dependent desaturation before range compression.
    // Doing it here however does a better job of preserving perceptual luminance of highly saturated colors. Because
in the hue-preserving path we only range-compress the max channel,
    // saturated colors lose luminance. By desaturating them more aggressively first, compressing, and then re-adding
some saturation, we can preserve their brightness to a greater extent.
    ictcpMapped.yz = lerp(ictcpMapped.yz, ictcp.yz * ictcpMapped.x / max(1e-3, ictcp.x),
postCompressionSaturationBoost);

    col = ICtCpToRGB(ictcpMapped);

    return col;
}
```

Code: supporting functions




```
// RGB with sRGB/Rec.709 primaries to ICtCp
float3 RGBToICtCp(float3 col)
{
    col = RGBToXYZ(col);
    col = XYZToLMS(col);
    // 1.0f = 100 nits, 100.0f = 10k nits
    col = linearToPQ(max(0.0.xxx, col), 100.0);

    // Convert PQ-LMS into ICtCp. Note that the "S" channel is not used,
    // but overlap between the cone responses for long, medium, and short wavelengths
    // ensures that the corresponding part of the spectrum contributes to luminance.

    float3x3 mat = float3x3(
        0.5000,  0.5000,  0.0000,
        1.6137, -3.3234,  1.7097,
        4.3780, -4.2455, -0.1325
    );

    return mul(mat, col);
}

float3 ICtCpToRGB(float3 col)
{
    float3x3 mat = float3x3(
        1.0,  0.00860514569398152,  0.11103560447547328,
        1.0, -0.00860514569398152, -0.11103560447547328,
        1.0,  0.56004885956263900, -0.32063747023212210
    );
    col = mul(mat, col);

    // 1.0f = 100 nits, 100.0f = 10k nits
    col = PQtoLinear(col, 100.0);
    col = LMSToXYZ(col);
    return XYZToRGB(col);
}
```

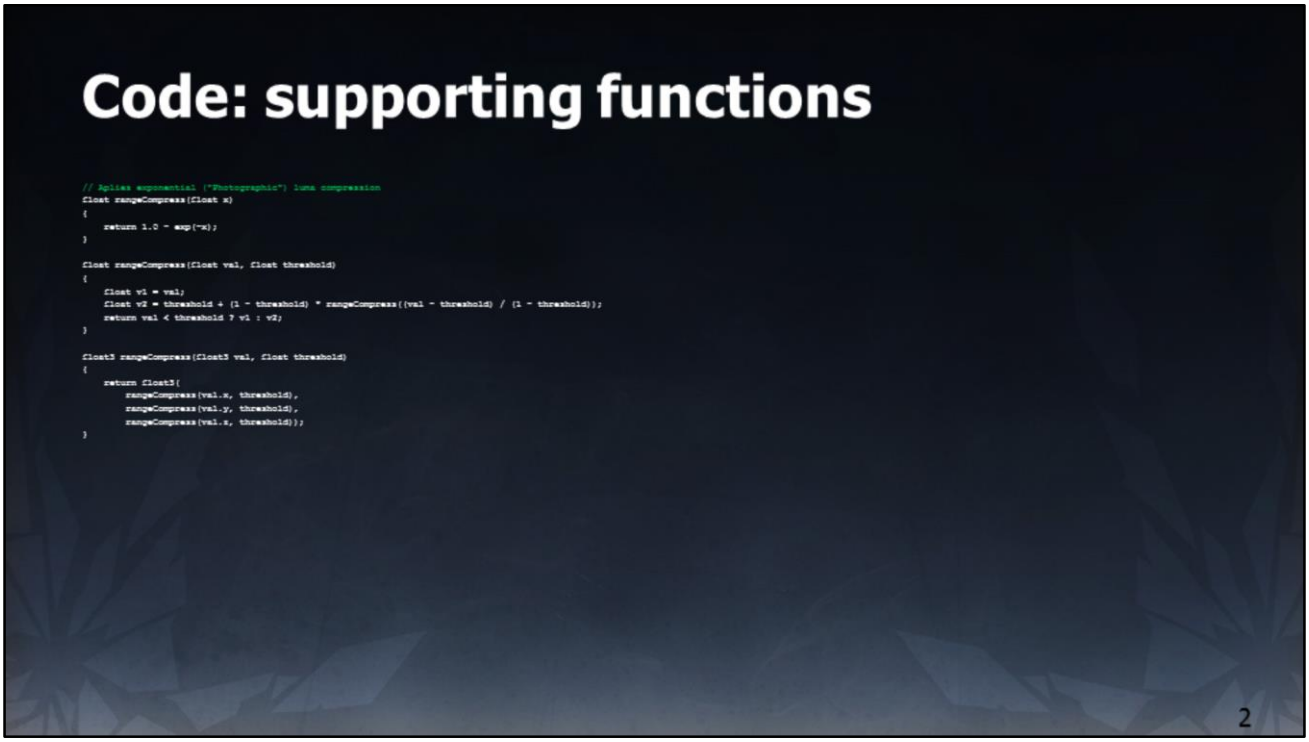
Code: supporting functions

```
// RGB with sRGB/Rec.709 primaries to CIE XYZ
float3 RGBToXYZ(float3 c)
{
    float3x3 mat = float3x3(
        0.4124564,  0.3575761,  0.1804375,
        0.2126729,  0.7151522,  0.0721750,
        0.0193339,  0.1191920,  0.9503041
    );
    return mul(mat, c);
}
float3 XYZToRGB(float3 c)
{
    float3x3 mat = float3x3(
         3.24045483602140870, -1.53713885010257510, -0.49853154686848090,
        -0.96926638987565370,  1.87601092884249100,  0.04155608234667354,
         0.05564341960421366, -0.20402585426769815,  1.05722516245792870
    );
    return mul(mat, c);
}
// Converts XYZ tristimulus values into cone responses for the three types of cones in the human visual system, matching long, medium, and short wavelengths.
// Note that there are many LMS color spaces; this one follows the ICtCp color space specification.
float3 XYZToLMS(float3 c)
{
    float3x3 mat = float3x3(
        0.3592,  0.6976, -0.0358,
        -0.1922, 1.1004,  0.0755,
        0.0070,  0.0749,  0.8434
    );
    return mul(mat, c);
}
float3 LMSToXYZ(float3 c)
{
    float3x3 mat = float3x3(
         2.07018005669561320, -1.32645687610302100,  0.206616006847855170,
         0.36498825003265756,  0.68046736285223520, -0.045421753075853236,
        -0.04959554223893212, -0.04942116118675749,  1.187995941732803400
    );
    return mul(mat, c);
}
```

Code: supporting functions


```
static const float PQ_constant_N = (2610.0 / 4096.0 / 4.0);
static const float PQ_constant_M = (2523.0 / 4096.0 * 128.0);
static const float PQ_constant_C1 = (3424.0 / 4096.0);
static const float PQ_constant_C2 = (2413.0 / 4096.0 * 32.0);
static const float PQ_constant_C3 = (2392.0 / 4096.0 * 32.0);

// PQ (Perceptual Quantiser; ST.2084) encode/decode used for HDR TV and grading
float3 linearToPQ(float3 linearCol, const float maxPqValue)
{
    linearCol /= maxPqValue;

    float3 colToPow = pow(linearCol, PQ_constant_N);
    float3 numerator = PQ_constant_C1 + PQ_constant_C2*colToPow;
    float3 denominator = 1.0 + PQ_constant_C3*colToPow;
    float3 pq = pow(numerator / denominator, PQ_constant_M);

    return pq;
}

float3 PQtoLinear(float3 linearCol, const float maxPqValue)
{
    float3 colToPow = pow(linearCol, 1.0 / PQ_constant_M);
    float3 numerator = max(colToPow - PQ_constant_C1, 0.0);
    float3 denominator = PQ_constant_C2 - (PQ_constant_C3 * colToPow);
    float3 linearColor = pow(numerator / denominator, 1.0 / PQ_constant_N);

    linearColor *= maxPqValue;

    return linearColor;
}
```

Code: supporting functions




```
// Aplies exponential ("Photographic") luma compression
float rangeCompress(float x)
{
    return 1.0 - exp(-x);
}

float rangeCompress(float val, float threshold)
{
    float v1 = val;
    float v2 = threshold + (1 - threshold) * rangeCompress((val - threshold) / (1 -
threshold));
    return val < threshold ? v1 : v2;
}

float3 rangeCompress(float3 val, float threshold)
{
    return float3(
        rangeCompress(val.x, threshold),
        rangeCompress(val.y, threshold),
        rangeCompress(val.z, threshold));
}
```