Advanced Techniques and Optimization of ~~HDR~~ VDR Color Pipelines

Timothy Lottes [AMD]

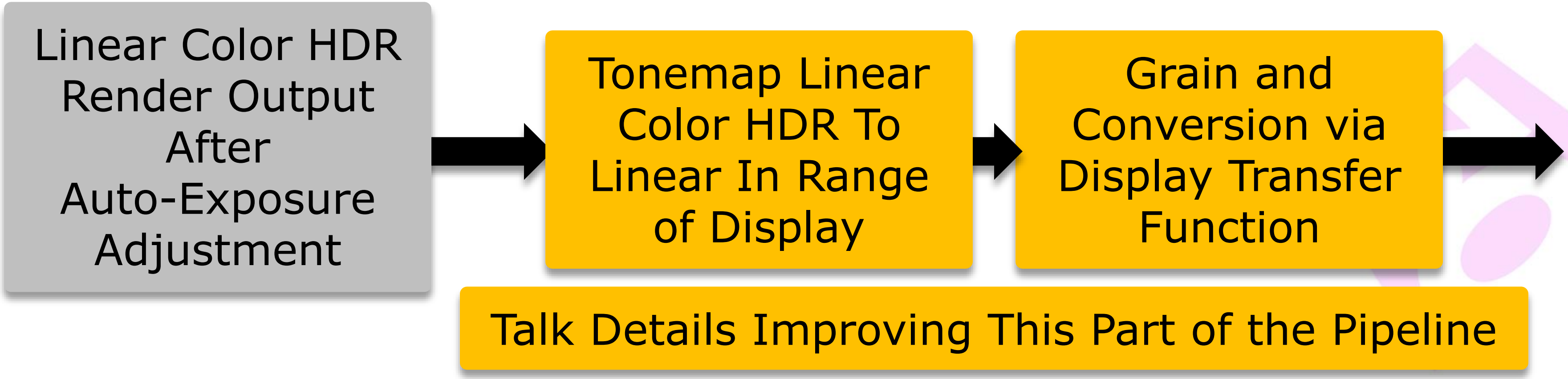
Contents

- Part 1: Variable Dynamic Range (VDR)
- Part 2: Tonemapping for VDR
- Part 3: Transfer Functions and Dithering

- Goal: Better quality for LDR through HDR

Graphics Pipeline Application

- Also applicable to LDR (reduced banding)
- Ideal for HDR on OLED VR HMDs


Linear Color HDR Render Output After Auto-Exposure Adjustment
Tonemap Linear Color HDR To Linear In Range of Display
Grain and Conversion via Display Transfer Function
Talk Details Improving This Part of the Pipeline

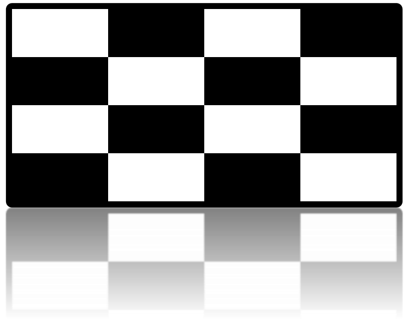
Part 1 – VDR

Variable Dynamic Range

Dynamic Range is a function of display capability and viewing environment

Terms

- ANSI Contrast = Measure this
  - Contrast Ratio = Dynamic Range



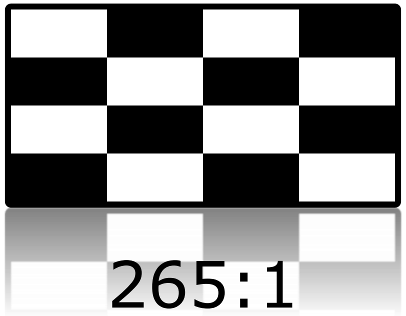
- APL = Average Picture Level
- Nits = Measure of Luminance (cd/m^2)
- Stop = A doubling of Luminance

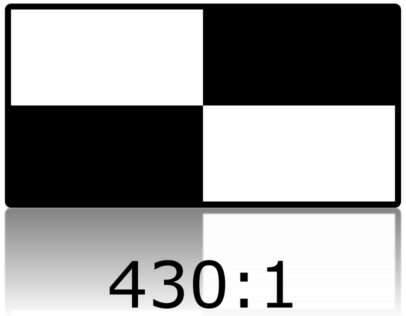
Display Capability

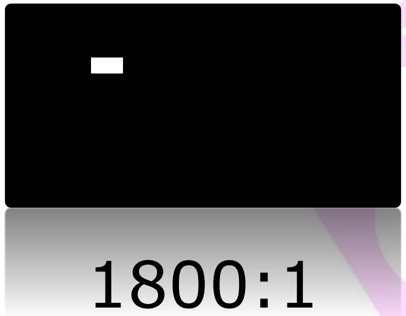
- Understanding display capability through measured examples
  - From past through current display

Desktop CRT Example

- 18 year old CRT
- 151 nits @ 50% APL
- CRT contrast varies based on content

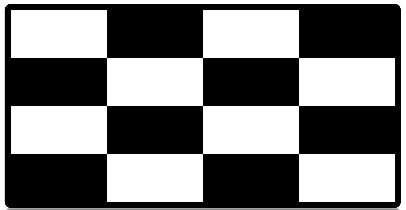

265:1

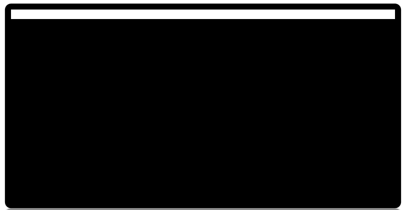

430:1

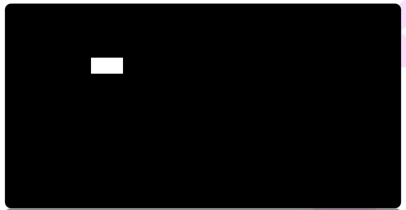

1800:1

HD CRT TV Example

- 720p/1080i “HD-ready” TV
- 302 nits @ 50% APL
- CRT contrast varies based on content


158:1


2500:1


30000:1

Average PC LCD

- Post CRTs with LCDs
- ~300 nits (with white ~6500 K)
- sRGB/Rec.709 gamut
- ~1000:1 ANSI contrast (~10 stops)
  - More uniform contrast (no CRT blooming)
  - But often higher black levels

Plasma Example

- Plasma TV from 2007
- Large HDTV
- 172 nits
- Around sRGB/Rec.709 gamut
- 3542:1 ANSI contrast (11.8 stops)

High-End Gaming LCD Example

- LCD from 2013
- 386 nits (white at 6364 K)
  - 257 nits in low-persistence mode
- sRGB/Rec.709 gamut
- 5214:1 ANSI contrast (12.3 stops)
  - 4602:1 in low-persistence mode

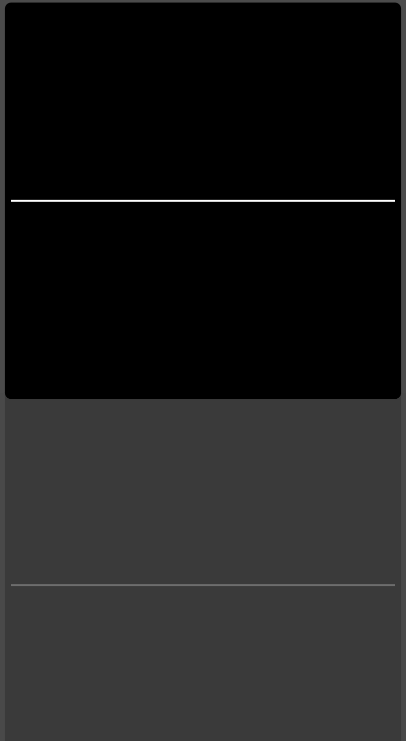
LED LCD HDR TV Example

- 419 nits (with 14473 K white)
  - No global dimming at 100% APL
- 104% DCI-P3 gamut
- 2.2% ambient, 3.1% specular reflection
- 3844:1 ANSI contrast ratio (11.9 stops)

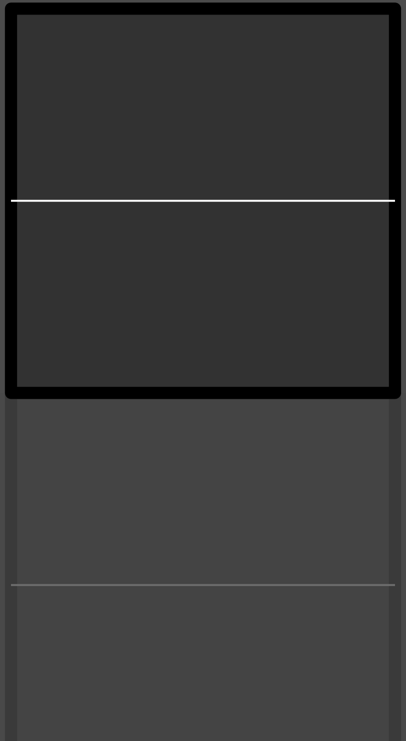
LED LCD and Local Dimming

- Backlight = array of LEDs (called zones)
- Joint scaling of {black level, peak brightness} non-uniformly across screen
    - +1 stop peak results in +1 stop in black level
- Spatially limited to gradual changes
    - Otherwise small bright pixels cause halos

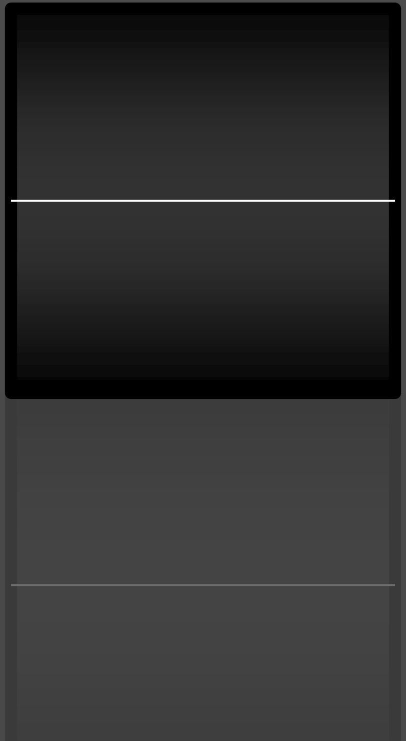
OLED | CRT



LCD



LCD With LED
Local Dimming



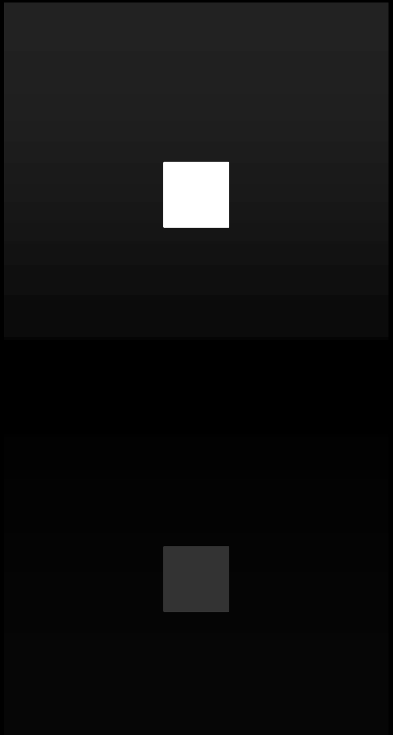
OLED HDR TV Example

- 433 nits @ 25% APL (11695 K)
  - 232 nits @ 50% APL, 141 nits @ 100% APL
- 93% DCI-P3 gamut
- 1.2% ambient, 2.1% specular reflection
- ANSI contrast limited by reflection
  - And panel uniformity

APL and Global Dimming

- Large TVs are often power limited
  - Screen dimmed to stay in power limit
- Example OLED
  - 452 nits when not dimming [X]
  - 151 nits on full white screen [Y]
  - LG dimming starts between 25-50% APL
  - Want in-game auto-exposure under 25% APL

X



Y

Takeaways

- Consumer space filled with displays of varying amounts of Dynamic Range
- Displays with High Dynamic Range have existed for some time
- Going brighter limited by power
- Most advantages from better darks

Viewing Environment

- Viewing environment has a larger magnitude of effects on the ability to display HDR, than the range of display capabilities

Actual Black Level

Sum of light contribution from

1. Display's native black
2. Screen reflection

Screen Reflection

- Room lighting reflecting off screen
  - Places practical limit on contrast ratio
- Example display screen reflectance
  - OLED: 1.2% ambient, 2.1% specular
  - LCD: 2.2% ambient, 3.1% specular
  - Older LCD: 6.5% ambient, 8.5% specular

Example Measured Rooms

- Different room ambient levels
  - Easily 15 stop difference in contrast limit

- Sampled 18% reflector (grey card)
- Contrast limit as limited by reflection

Display Facing Covered Patio

- 1000 nit ambient
- 400 nit screen contrast limit
  - 1% reflectance = 5.3 stops
  - 2% reflectance = 4.3 stops
  - 4% reflectance = 3.3 stops
  - 8% reflectance = 2.3 stops

Display Facing Closed Blinds

- 200 nit ambient
- 400 nit screen contrast limit
  - 1% reflectance = 7.6 stops
  - 2% reflectance = 6.6 stops
  - 4% reflectance = 5.6 stops
  - 8% reflectance = 4.6 stops

Display Facing Closed Curtain

- 5 nit ambient
- 400 nit screen contrast limit
  - 1% reflectance = 13.0 stops
  - 2% reflectance = 12.0 stops
  - 4% reflectance = 11.0 stops
  - 8% reflectance = 10.0 stops

In Next Room With No Windows

- 0.25 nit ambient
- 400 nit screen contrast limit
  - 1% reflectance = 17.3 stops
  - 2% reflectance = 16.3 stops
  - 4% reflectance = 15.3 stops
  - 8% reflectance = 14.3 stops

100 nit Display Lighting Room

- 0.05 nit ambient
- 400 nit contrast limit
  - 1% reflectance = 19.6 stops
  - 2% reflectance = 18.6 stops
  - 4% reflectance = 17.6 stops
  - 8% reflectance = 16.6 stops

Takeaways

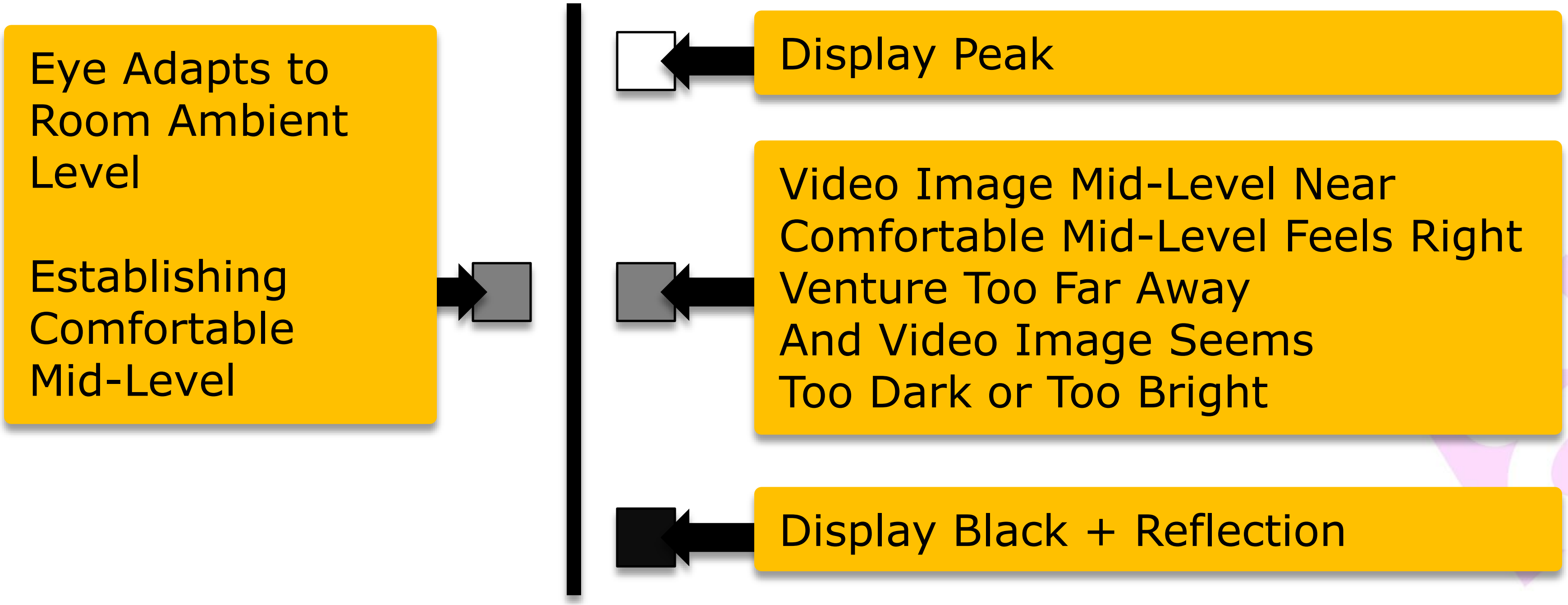
- Player views game in a Variable Dynamic Range environment
  - Bright room / day = low dynamic range
  - Dark room / night = higher dynamic range

- HDR ready =
  good ANSI contrast display + dark room

Part 2 – Tonemapping

Adjusting game display settings and optimizing tonemapping for VDR

Eye Adaption to Room Ambient



Eye Adapts to Room Ambient Level

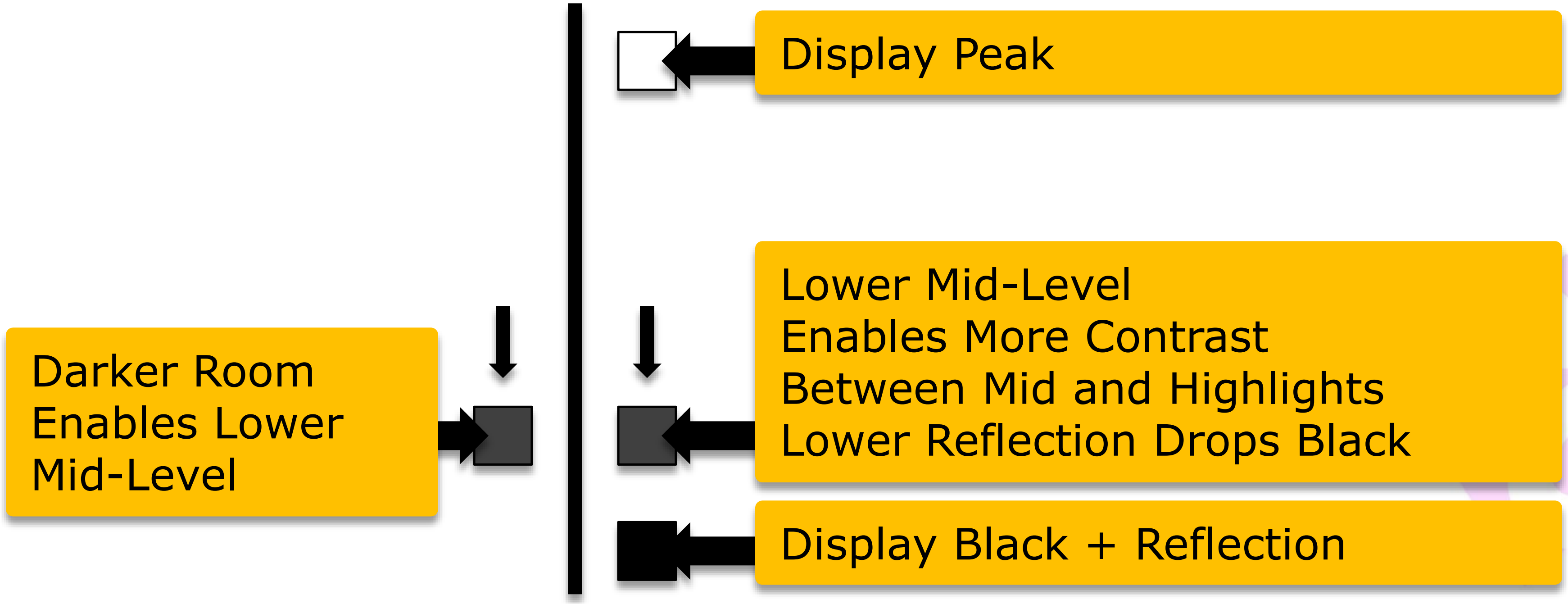
Establishing Comfortable Mid-Level

Display Peak

Video Image Mid-Level Near Comfortable Mid-Level Feels Right Venture Too Far Away And Video Image Seems Too Dark or Too Bright

Display Black + Reflection

Dark Ambient Enables HDR


Display Peak
Darker Room Enables Lower Mid-Level
Lower Mid-Level Enables More Contrast Between Mid and Highlights Lower Reflection Drops Black
Display Black + Reflection

Tonemapping Goals

- Adaption viewing condition for VDR
- Want believably real images
  - Effects can be subtle (little details)
- Want {saturation, contrast, hue} changes fully decoupled from changes in tonality
  - Same content with different mid-level mapping should look consistent

Tonemapping Goals Part 2

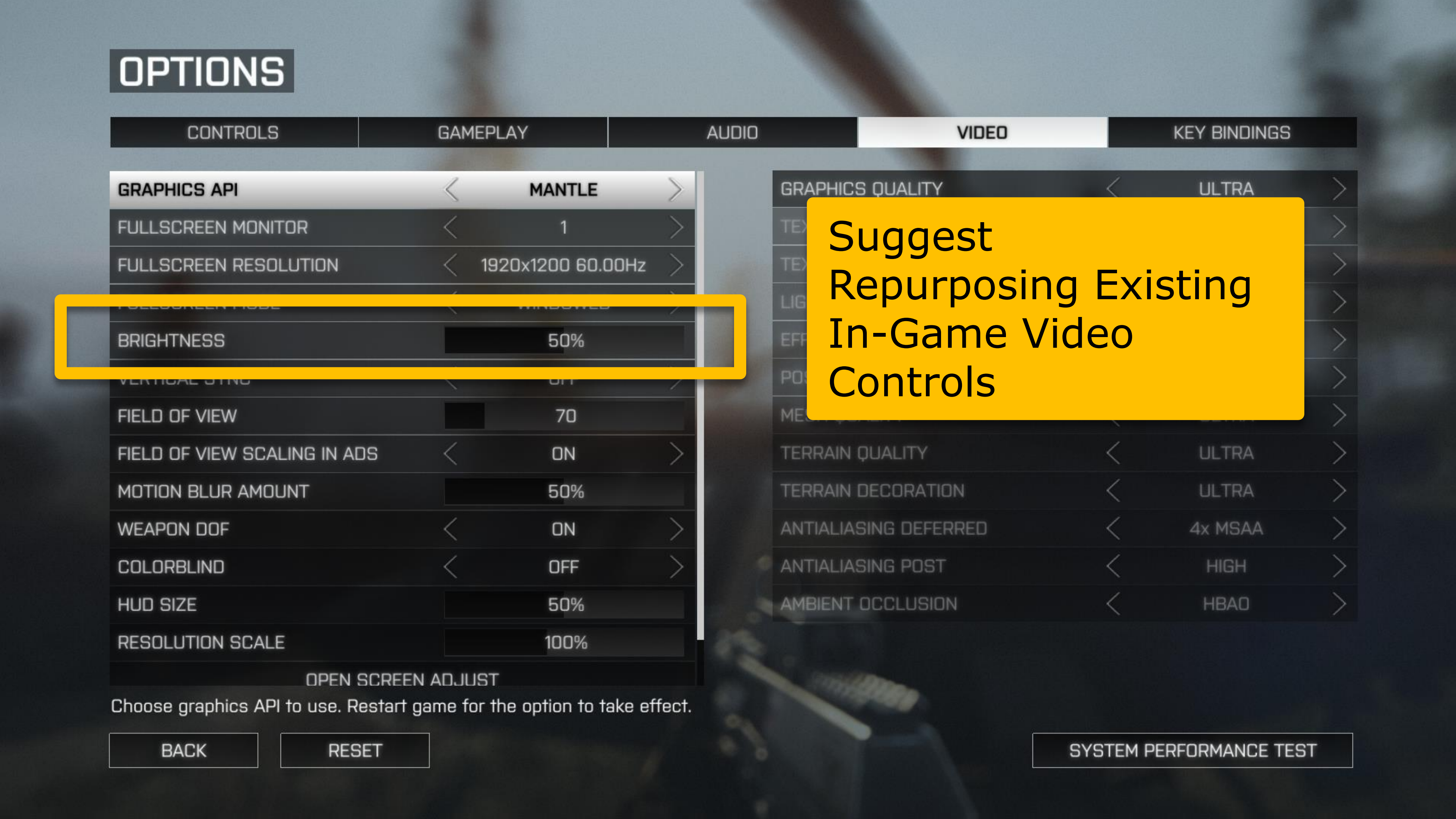
- Built in contrast & saturation control
- Existing in-game auto-exposure
    - Driving input mid-level
- Stable settings
    - No retune as output mid-level changes

Conventions for VDR

- Traditional display calibration/viewing convention is to drop peak brightness to comfortable level

- Instead keep peak for HDR and drop in-game mid-level to comfortable level
  - Also drop in-game GUI white level

Suggest Repurposing Existing In-Game Video Controls



OPTIONS

CONTROLS | GAMEPLAY | AUDIO | VIDEO | KEY BINDINGS

GRAPHICS API	MANTLE
FULLSCREEN MONITOR	1
FULLSCREEN RESOLUTION	1920x1200 60.00Hz
FULLSCREEN MODE	WINDOWED
BRIGHTNESS	50%
VERTICAL SYNC	OFF
FIELD OF VIEW	70
FIELD OF VIEW SCALING IN ADS	ON
MOTION BLUR AMOUNT	50%
WEAPON DOF	ON
COLORBLIND	OFF
HUD SIZE	50%
RESOLUTION SCALE	100%

OPEN SCREEN ADJUST

GRAPHICS QUALITY	ULTRA
TEX	
TEX	
LIG	
EFF	
PO	
ME	
TERRAIN QUALITY	ULTRA
TERRAIN DECORATION	ULTRA
ANTIALIASING DEFERRED	4x MSAA
ANTIALIASING POST	HIGH
AMBIENT OCCLUSION	HBAO

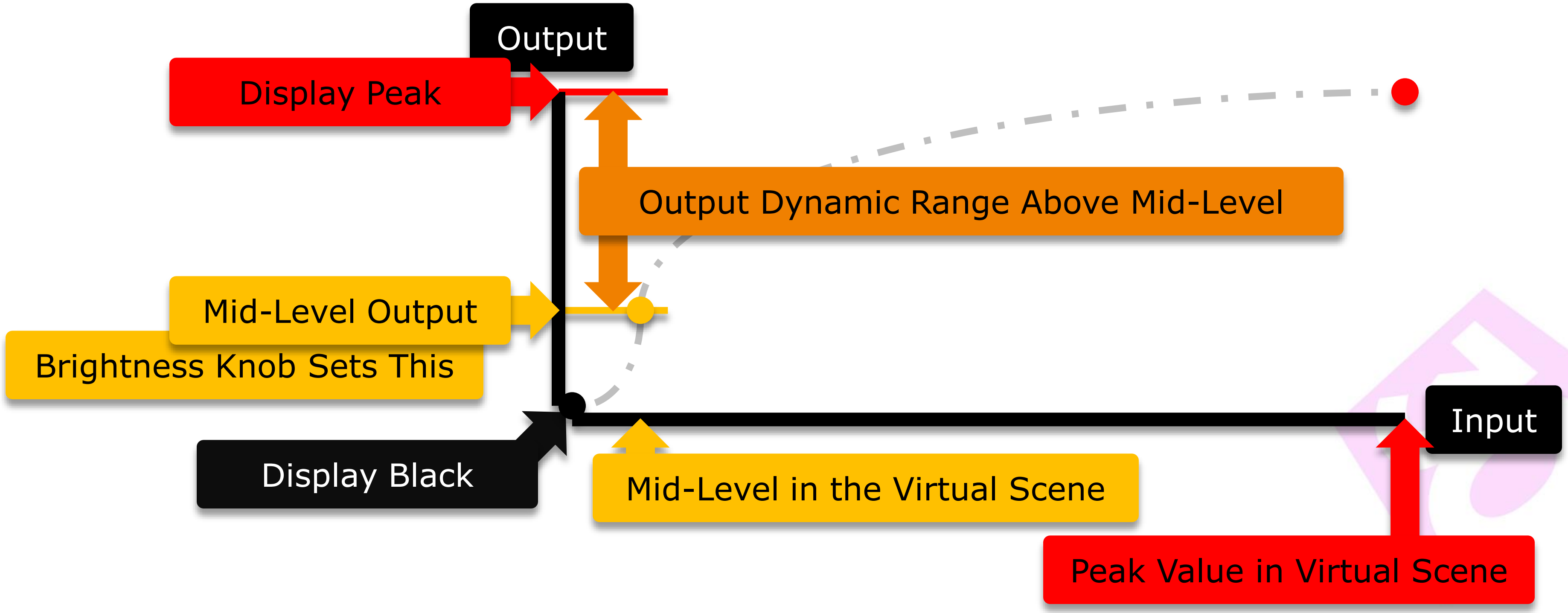
Choose graphics API to use. Restart game for the option to take effect.

BACK | RESET | SYSTEM PERFORMANCE TEST

Brightness or Contrast Knob

- Sometimes traditionally a power function
  - Often done after tonemapping, simultaneously changing {mid-level mapping, constrast, and saturation}
- Suggest using Knob to drive mid-level mapping for tonemapping
  - Adjusting visible dynamic range

Tonemapper Mapping


Output
Display Peak
Output Dynamic Range Above Mid-Level
Mid-Level Output
Brightness Knob Sets This
Display Black
Mid-Level in the Virtual Scene
Input
Peak Value in Virtual Scene

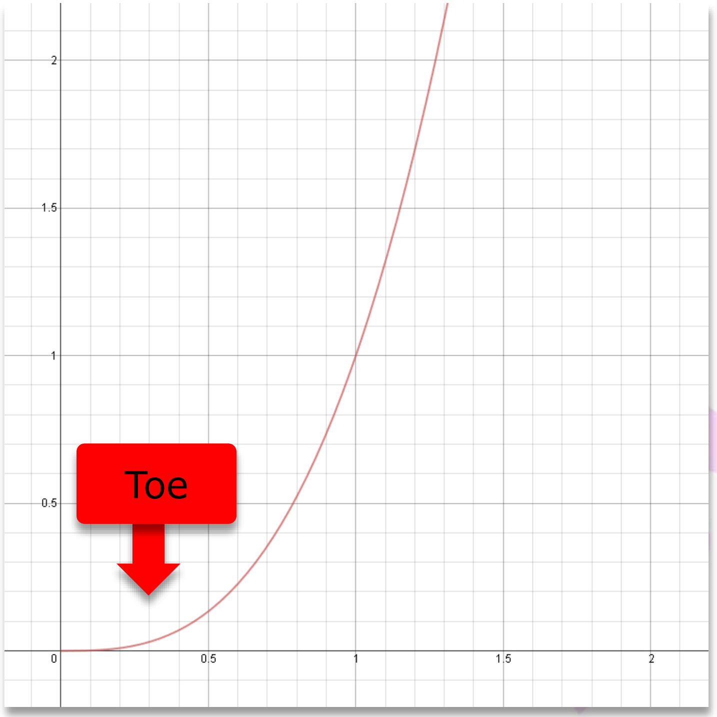
Building a Tonemapper

- Showing how to construct and control a high quality tonemapper working from base mathematical constructs

Building a Tonemapper: Part 1

- y=pow(x,a);

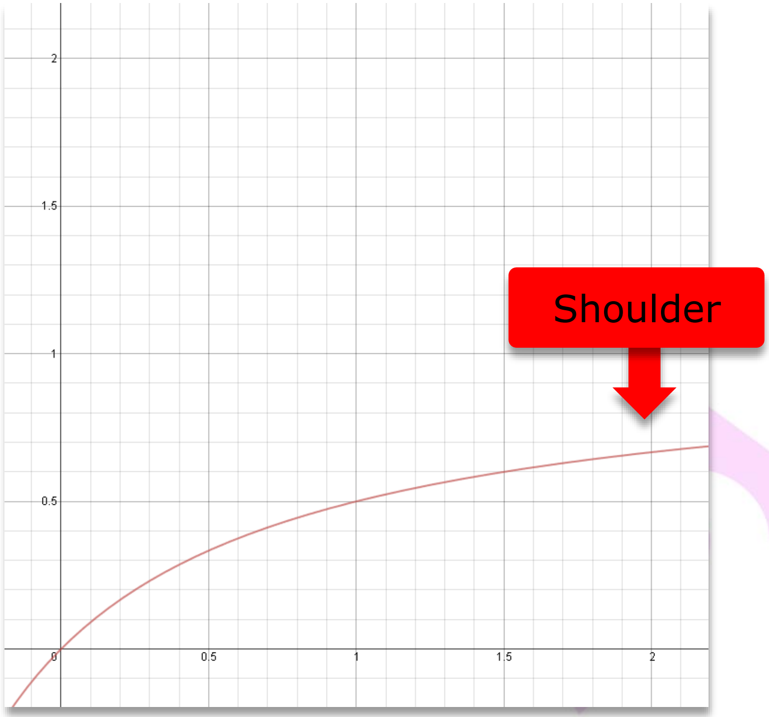
- Contrast
- Sets toe of curve



Building a Tonemapper: Part 2.0

- y=x/(x+1);

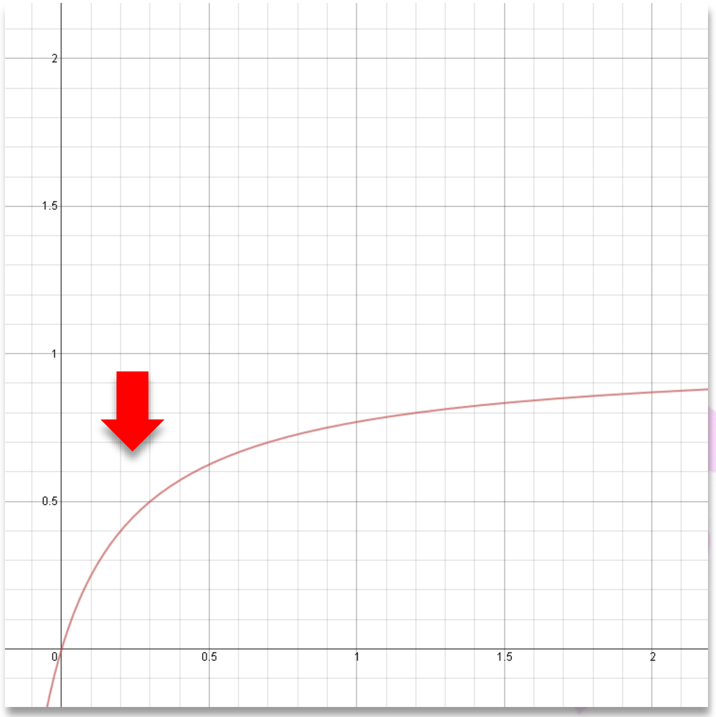
- Highlight compression
- Sets shoulder of curve
- Smoothly limits output



Building a Tonemapper: Part 2.1

- y=x/(x+**c**);

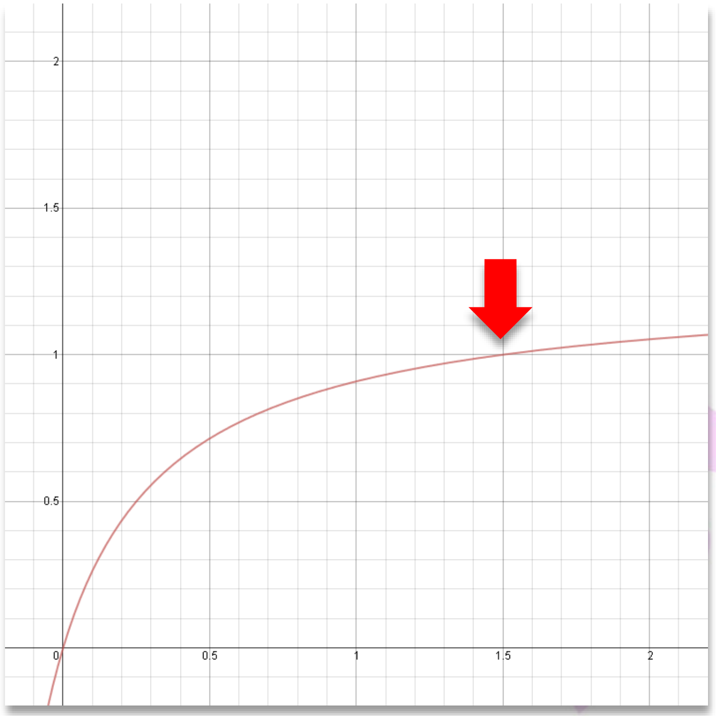
- Add ‘**c**’ term
- Speed of compression



Building a Tonemapper: Part 2.2

- y=x/(x***b**+c);

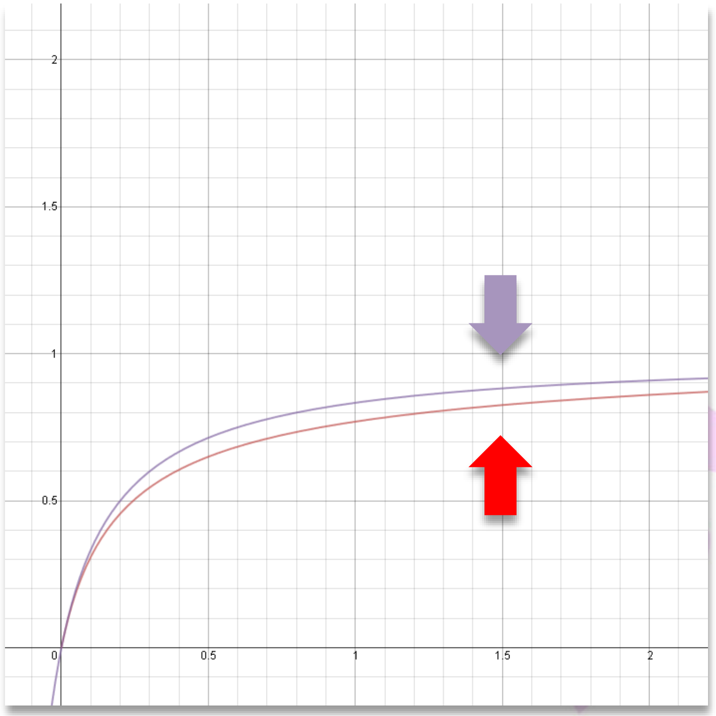
- Add '**b**' term
- Sets clipping point
- Sets peak of input dynamic range



Building a Tonemapper: Part 2.3

- y=x/(**pow(**x,**d)***b+c);

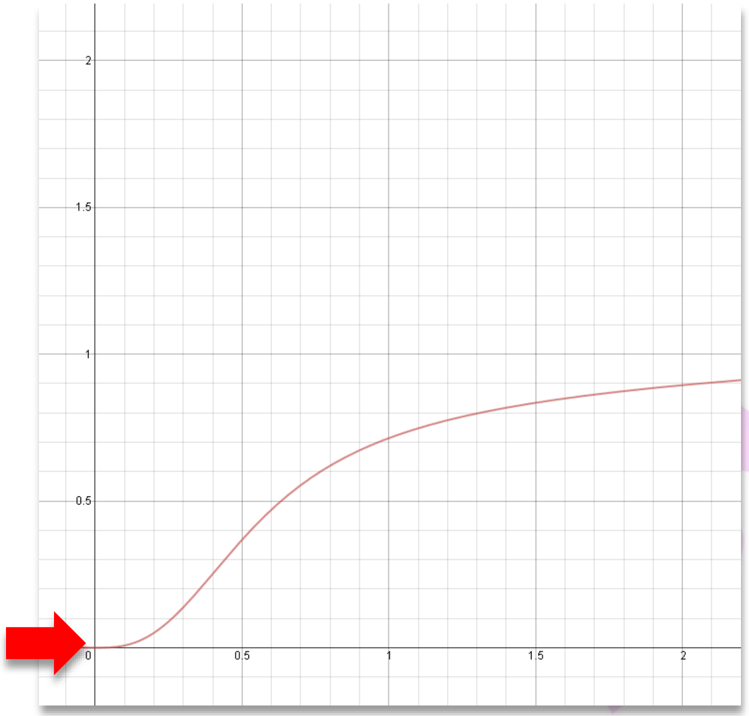
- Add '**d**' power
- Adjusts compression speed in the shoulder region of the curve



Building a Tonemapper: Part 3.0

- z=pow(x,a);
- y=z/(pow(z,d)*b+c);

- Combine both parts to build tonemapper



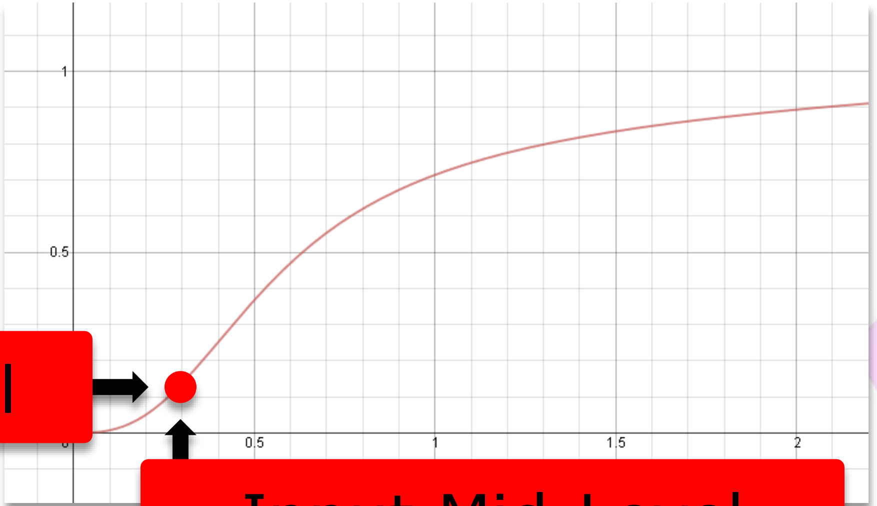
Controlling The Tonemapper

- z=pow(x,**contrast**);
- y=z/(pow(z,**shoulder**)***b**+**c**);

- {**contrast**,**shoulder**} shapes curve
- {**b**,**c**} anchors curve

Mid-Level Anchor to Set {b,c}

- Use 18% grey mid-level as anchor point



Output Mid-Level

Input Mid-Level

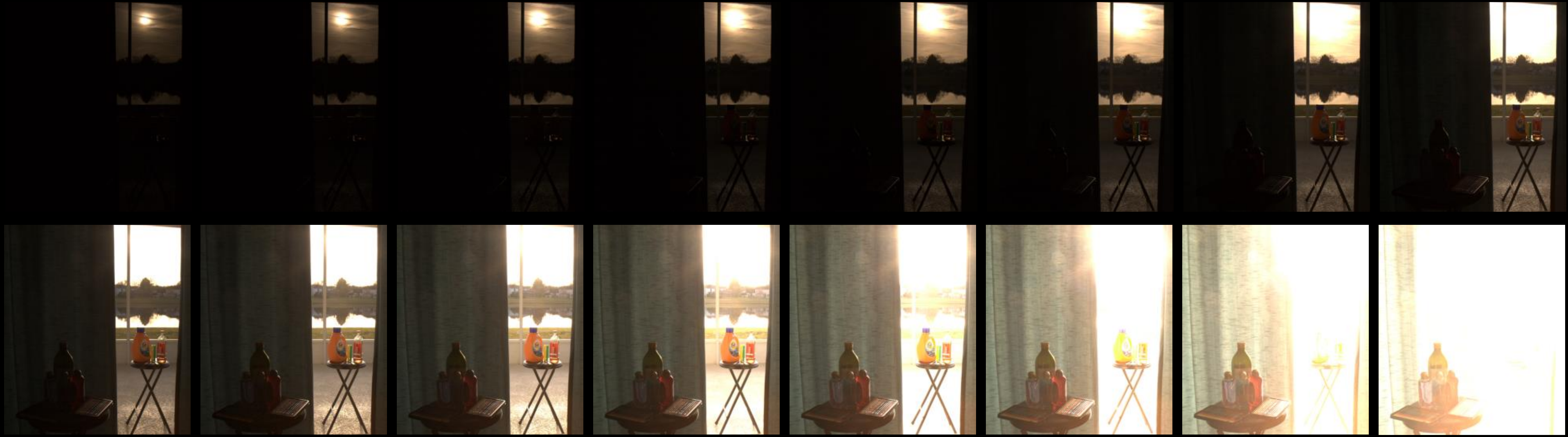
```
(* Mathematica source for solving for {b,c} *)

tonemap[x_]:=(x^a) / (((x^a)^d) * b + c)

Solve[{tonemap[midIn] == midOut,
       tonemap[hdrMax] == 1}, {b,c}, Reals]

(* output *)

b -> (-midIn^a + hdrMax^a * midOut) /
       (((hdrMax^a)^d - (midIn^a)^d) * midOut)

c -> ((hdrMax^a)^d * midIn^a - hdrMax^a * (midIn^a)^d * midOut) /
       (((hdrMax^a)^d - (midIn^a)^d) * midOut)
```

Apply to a Real HDR Scene

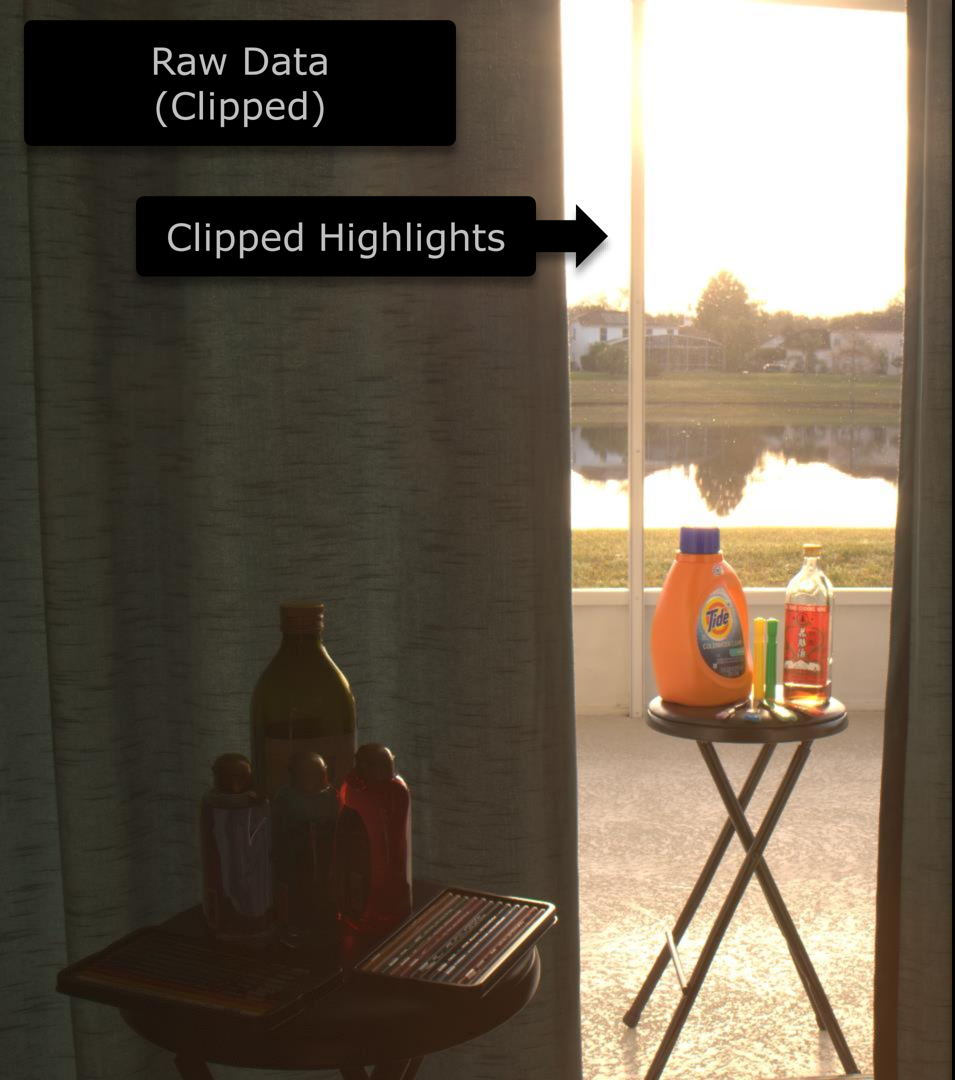
- HDR image built from 16 exposures

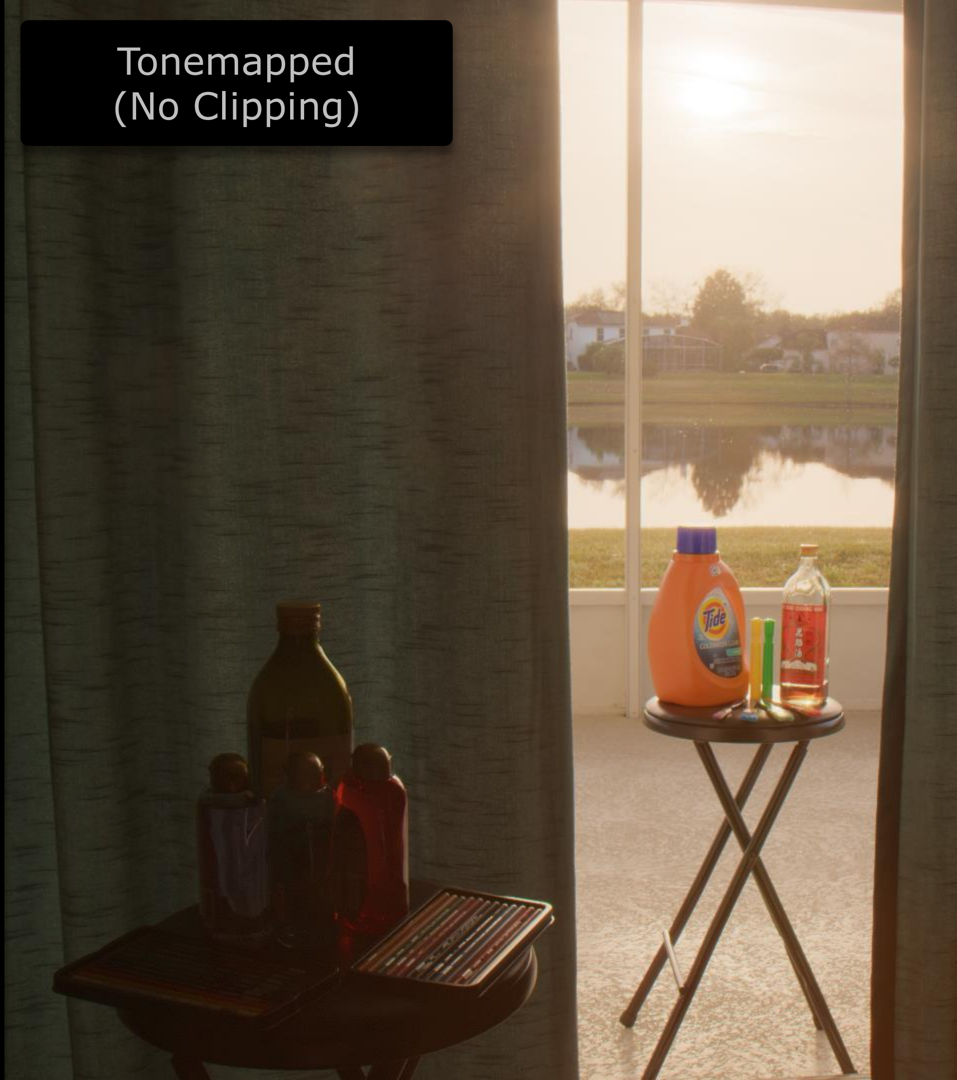
- Challenging content for tonemapper
  - Shot includes the sun
  - Using 10 stops of data above 18% mid-level for base exposure in comparison shots

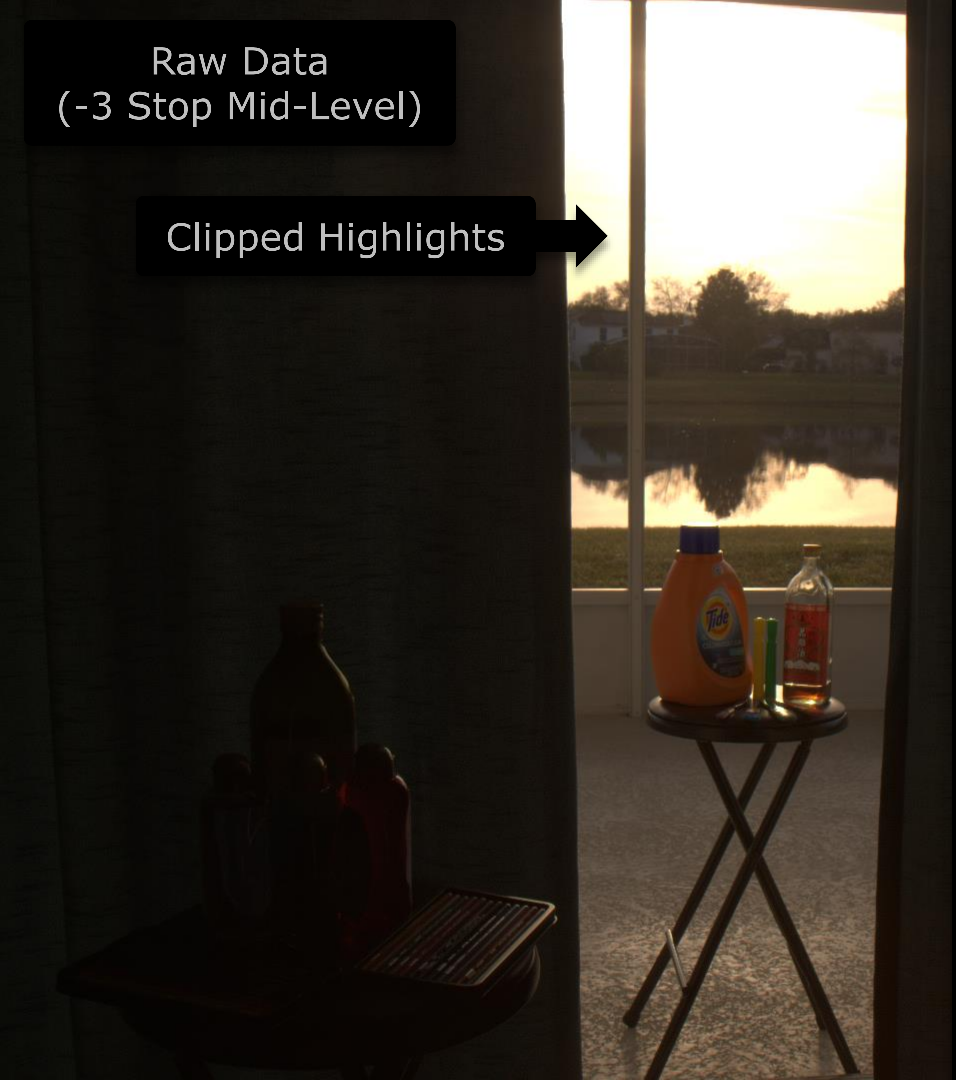
Raw Photos, No Tonemapping

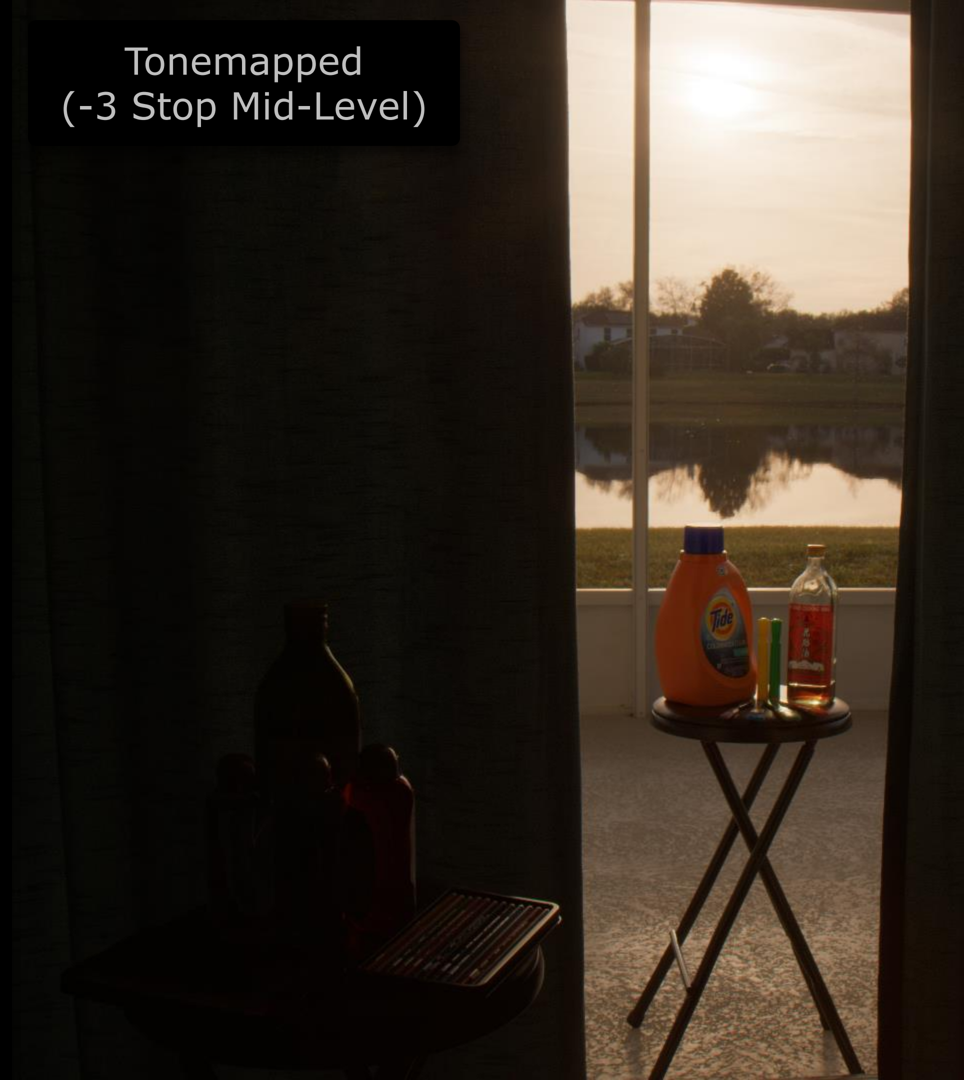


16 Stops of a Real HDR Scene


Raw Data
(Clipped)
Clipped Highlights

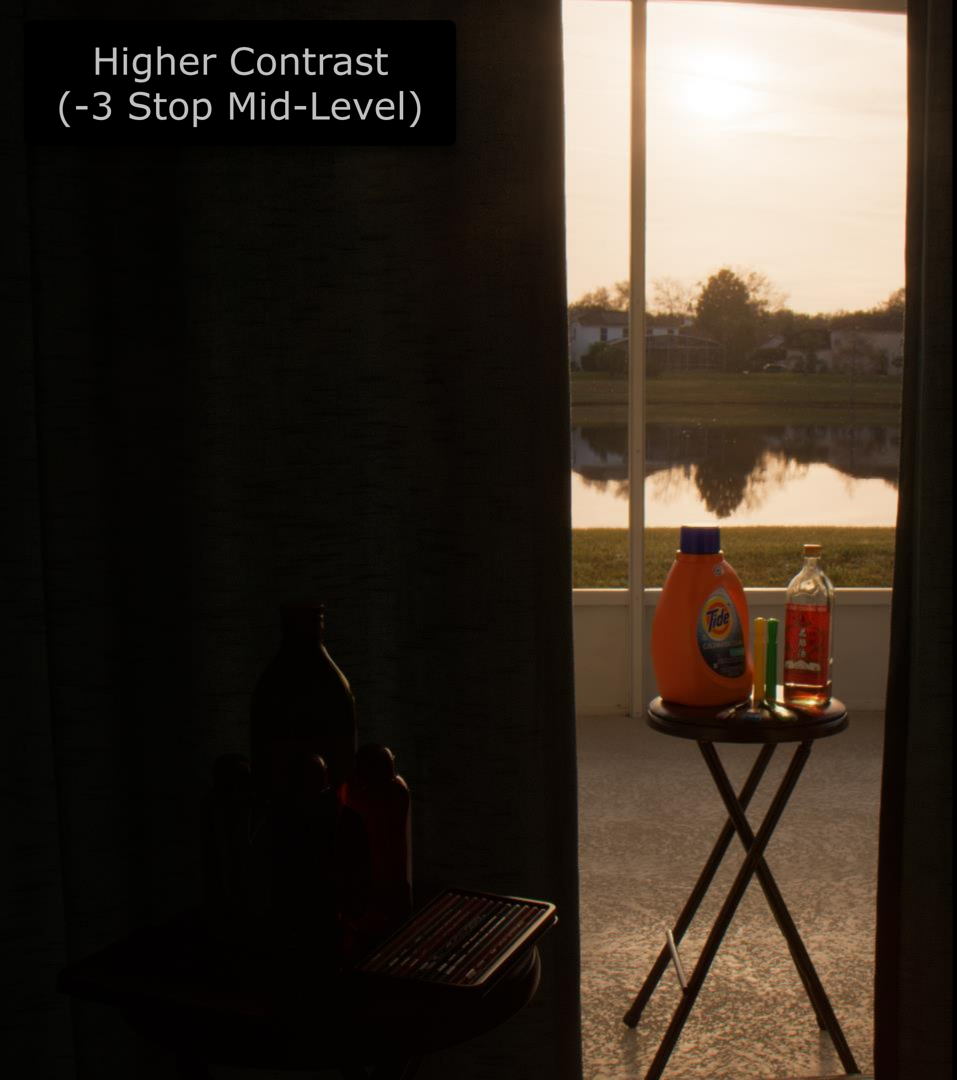

Tonemapped
(No Clipping)

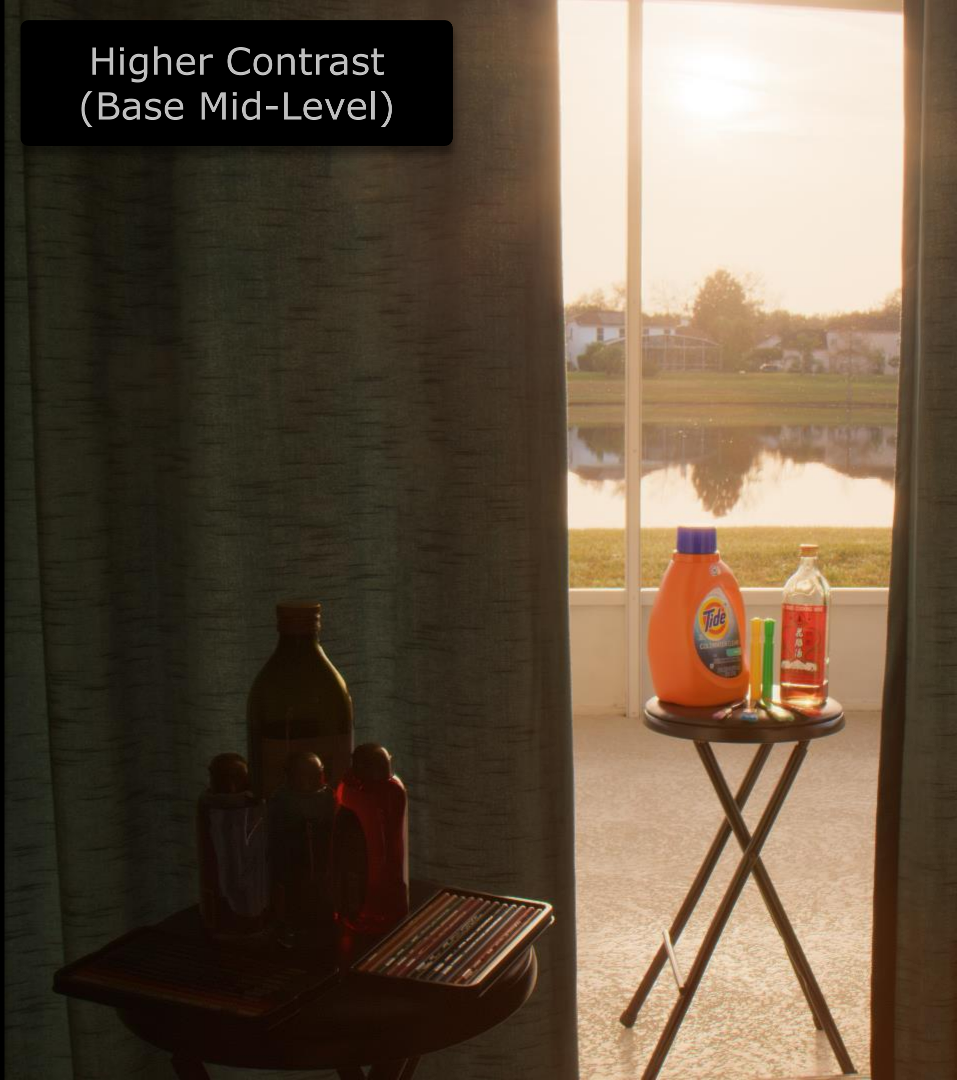

Raw Data
(-3 Stop Mid-Level)
Clipped Highlights


Tonemapped
(-3 Stop Mid-Level)

Eye Adaption

- 1st image and 2nd image of prior series
  - Represents a practical difference between "lights off" and "lights on" viewing conditions for VDR when display has good ANSI Contrast
- Next side by side but with larger contrast setting in tonemapper

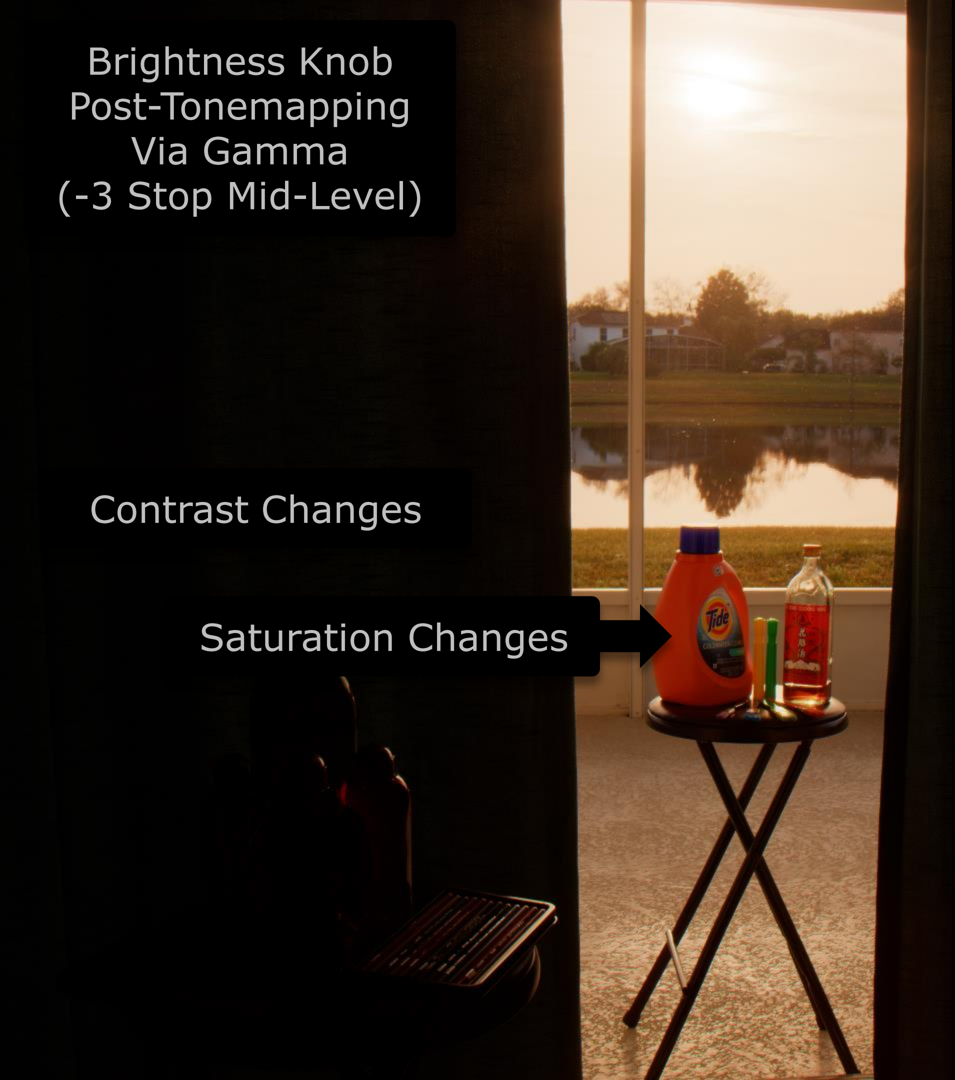

Higher Contrast
(-3 Stop Mid-Level)


Higher Contrast
(Base Mid-Level)

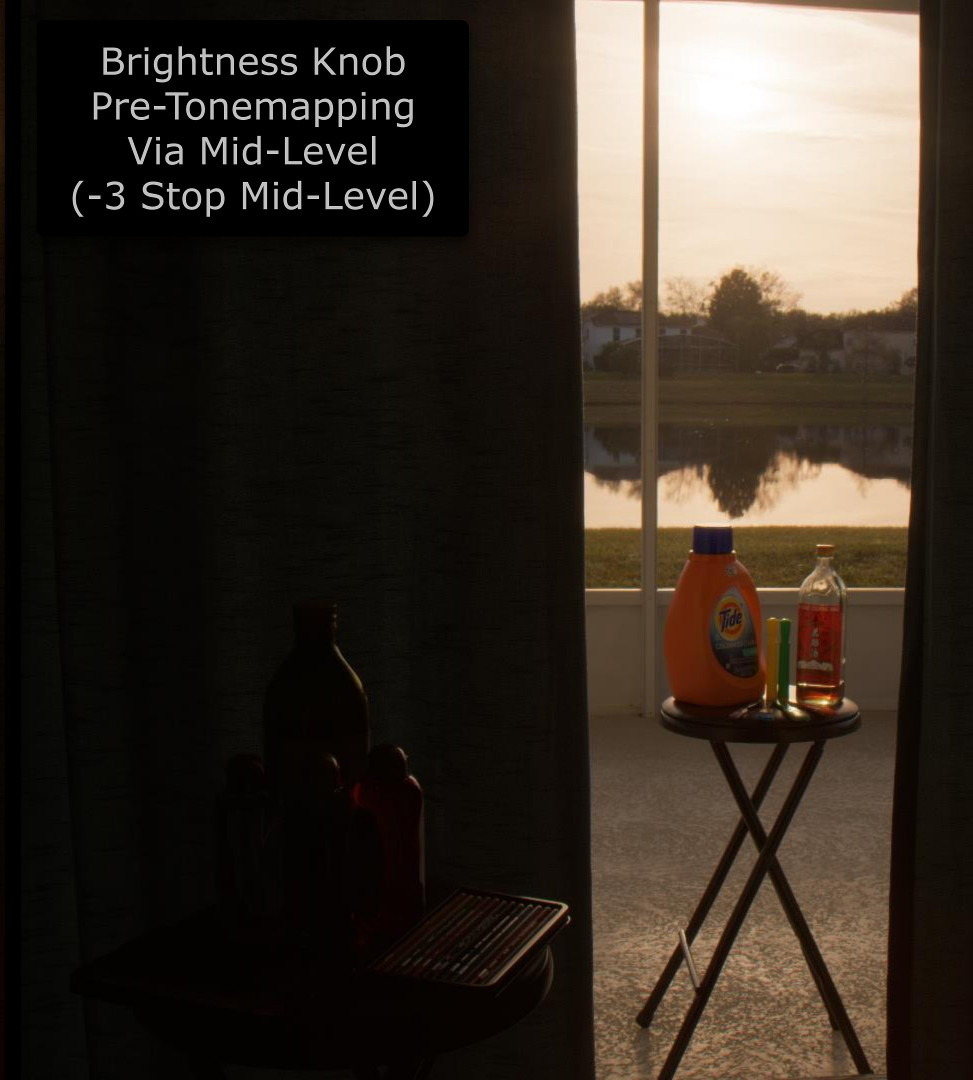
Knob Differences

- Using mid-level mapping to adjust brightness pre-tonemapping keeps contrast and saturation consistent
  - Post-tonemapping gamma adjustment increases contrast+saturation when darkening, and decreases contrast+saturation when lightening mid-level (want to avoid this)

Brightness Knob
Post-Tonemapping
Via Gamma
(-3 Stop Mid-Level)

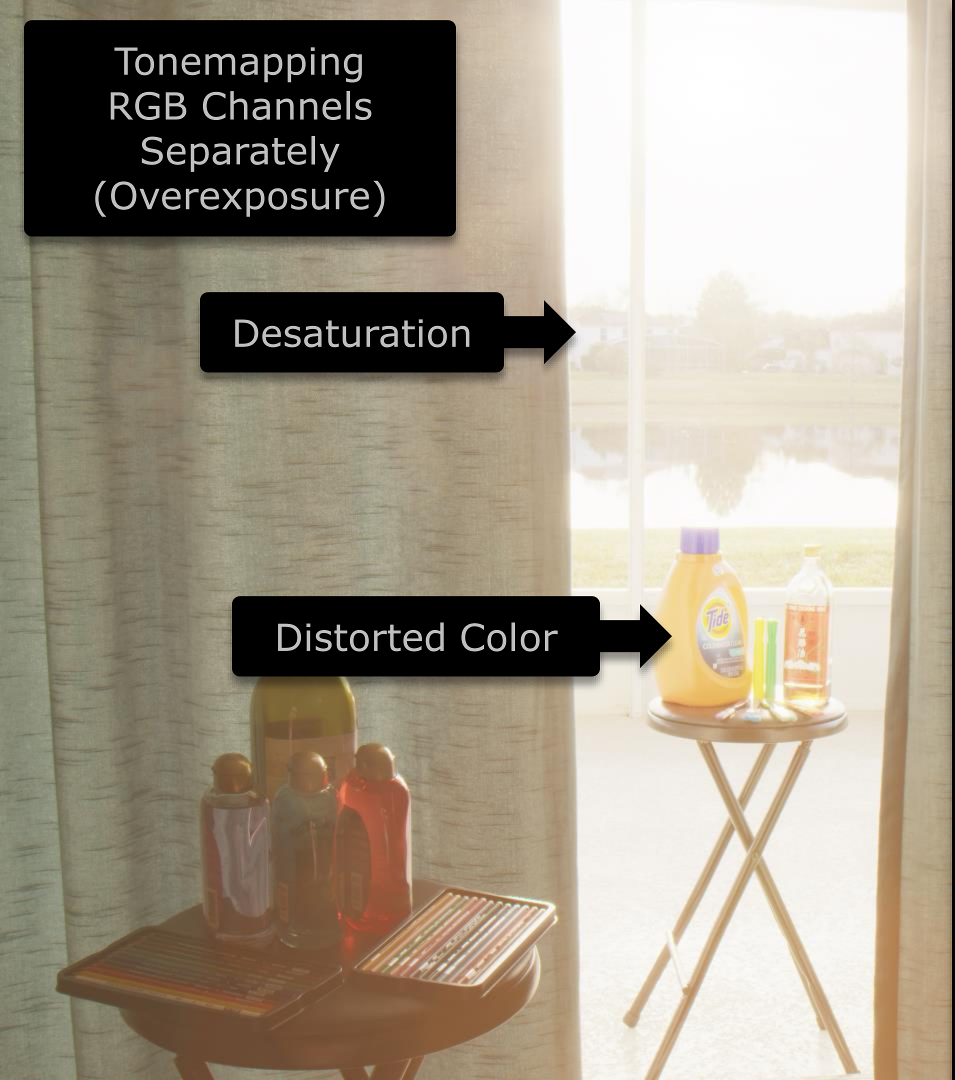


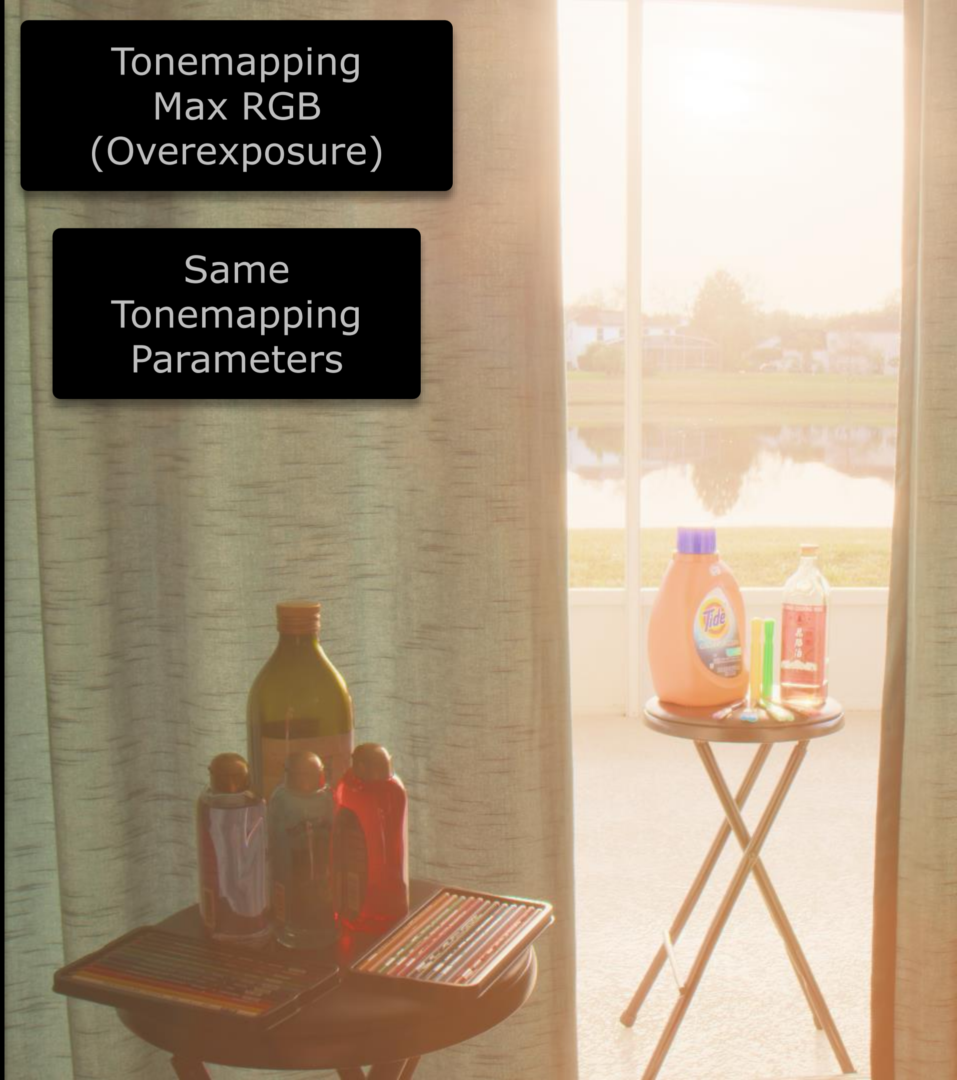
Brightness Knob
Pre-Tonemapping
Via Mid-Level
(-3 Stop Mid-Level)



Separation of Max and RGB Ratio

- Tonemapping is applied to max of {r,g,b}
  - peak = max3(rgb); ratio = rgb/peak;
  - peak = tonemap(peak); output = peak*ratio;
- Color processing is done separately
  - (adjusting ratio done separately)
- Has advantages in over-exposure cases


Tonemapping RGB Channels Separately (Overexposure)

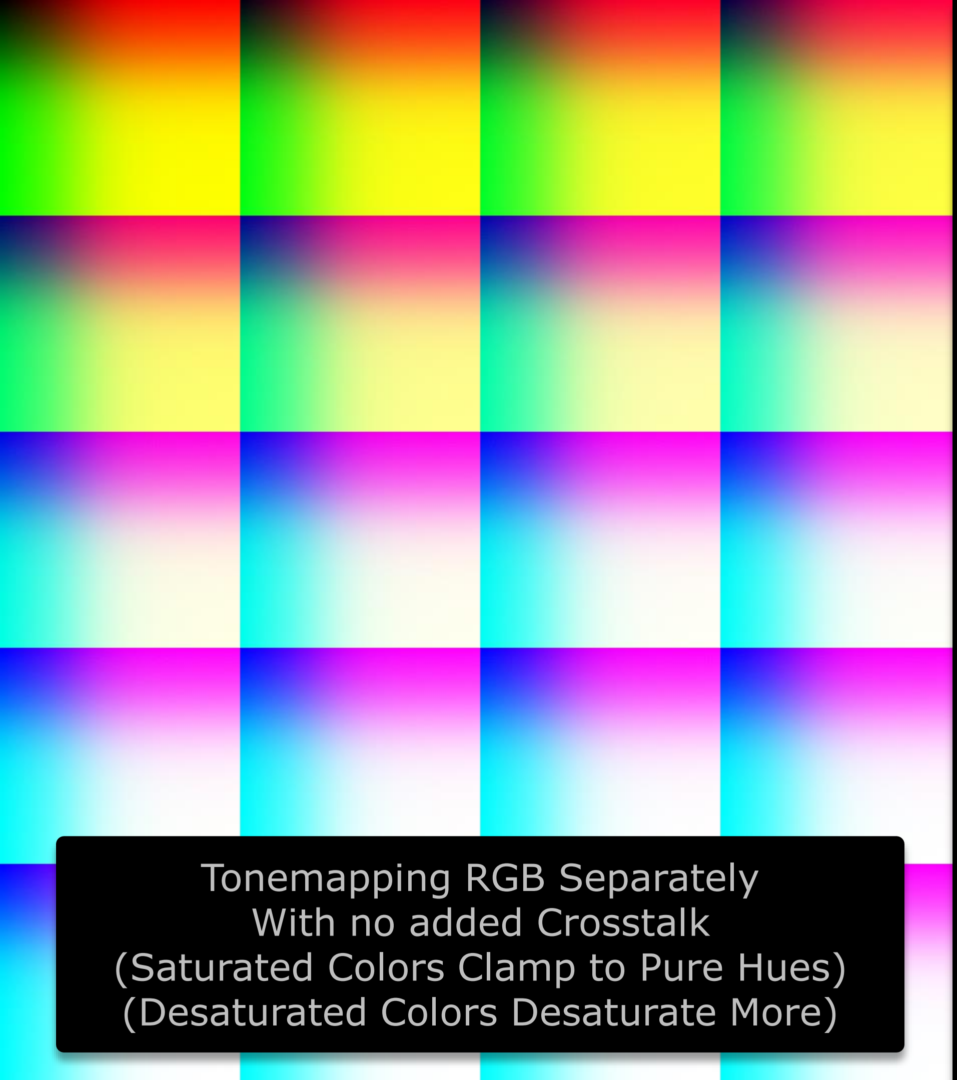

Tonemapping Max RGB (Overexposure)

Processing of Color Ratio

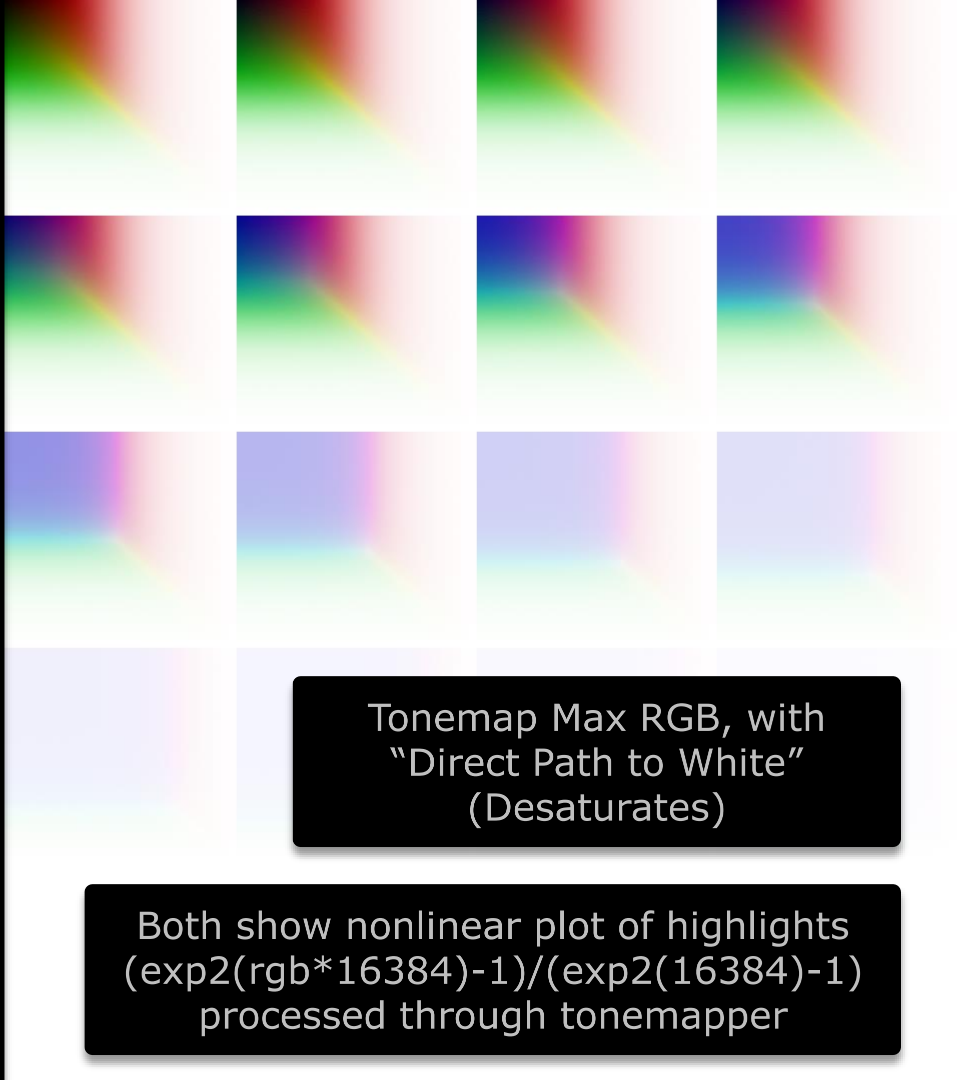
- Channel Crosstalk
  - Works similar in concept to real-world film
  - Move colors towards white as they overexpose maintaining perception of brightness

Path to White

- Path colors take to white is important
  - Direct path desaturates fast
- Indirect path which moves brighter channels faster maintains more saturation
  - Apply crosstalk non-linearly

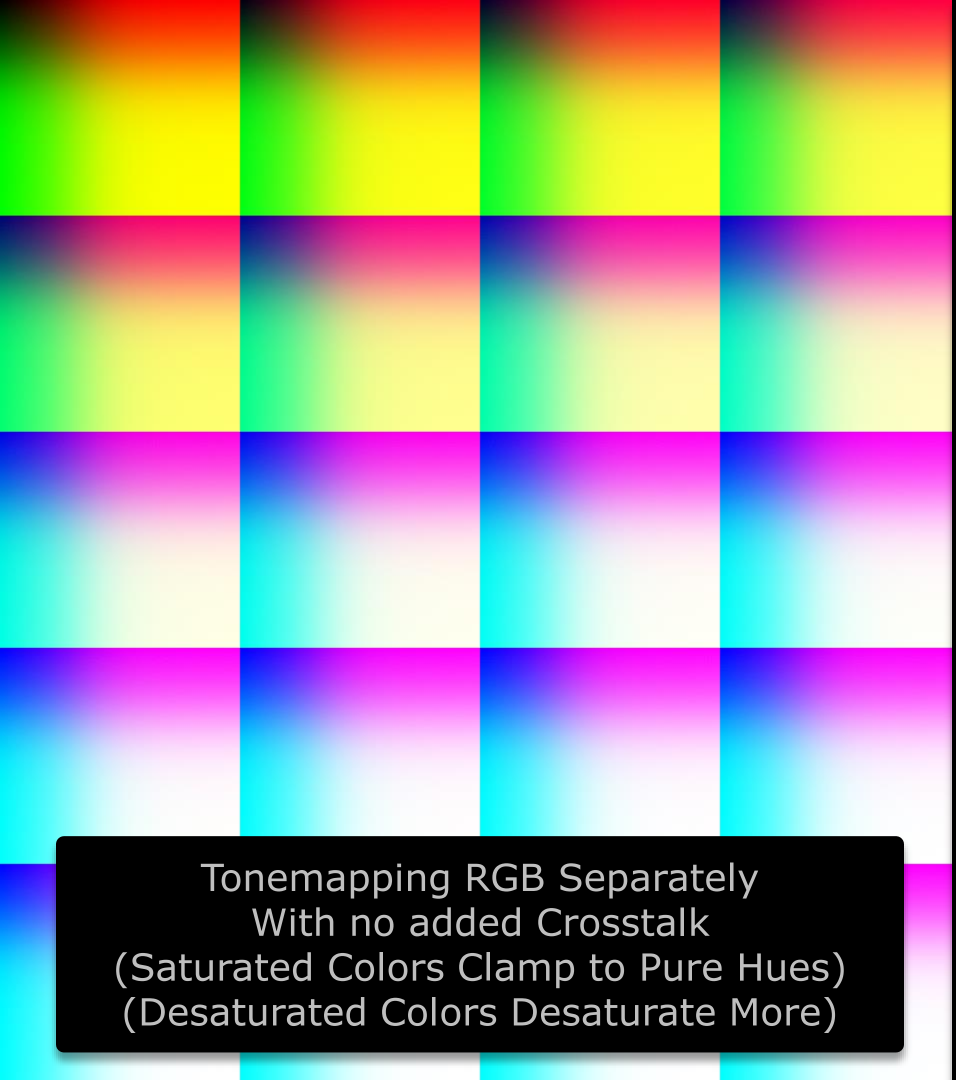


Tonemapping RGB Separately
With no added Crosstalk
(Saturated Colors Clamp to Pure Hues)
(Desaturated Colors Desaturate More)

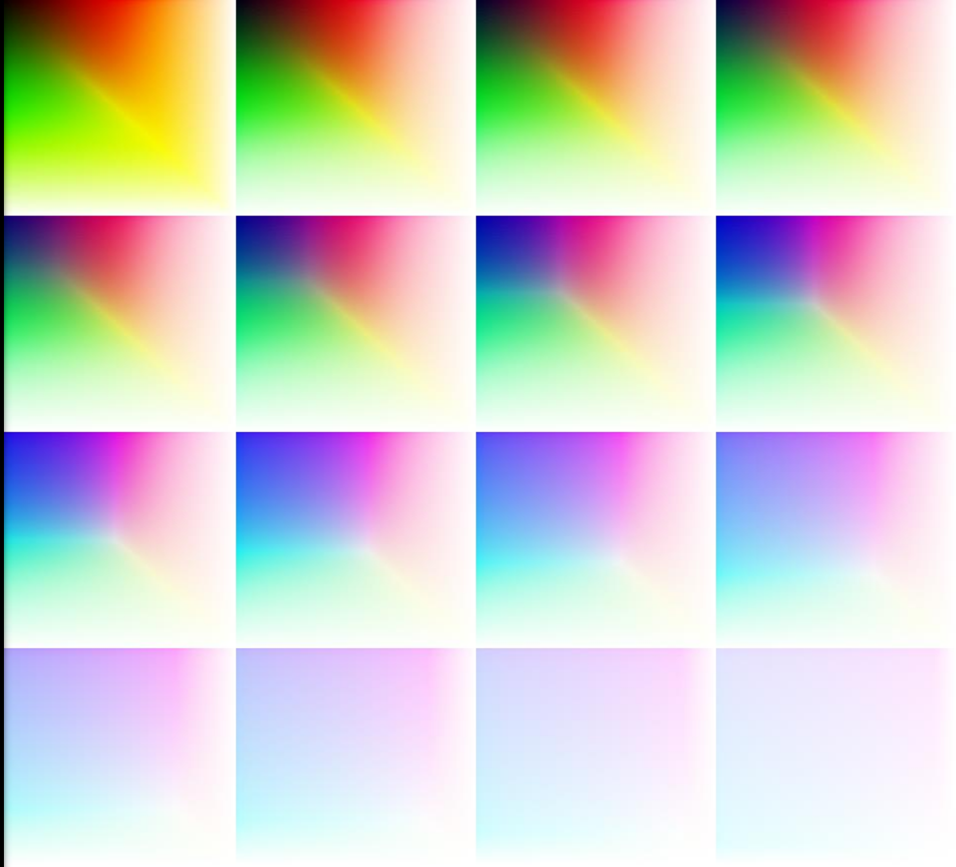


Tonemap Max RGB, with
“Direct Path to White”
(Desaturates)

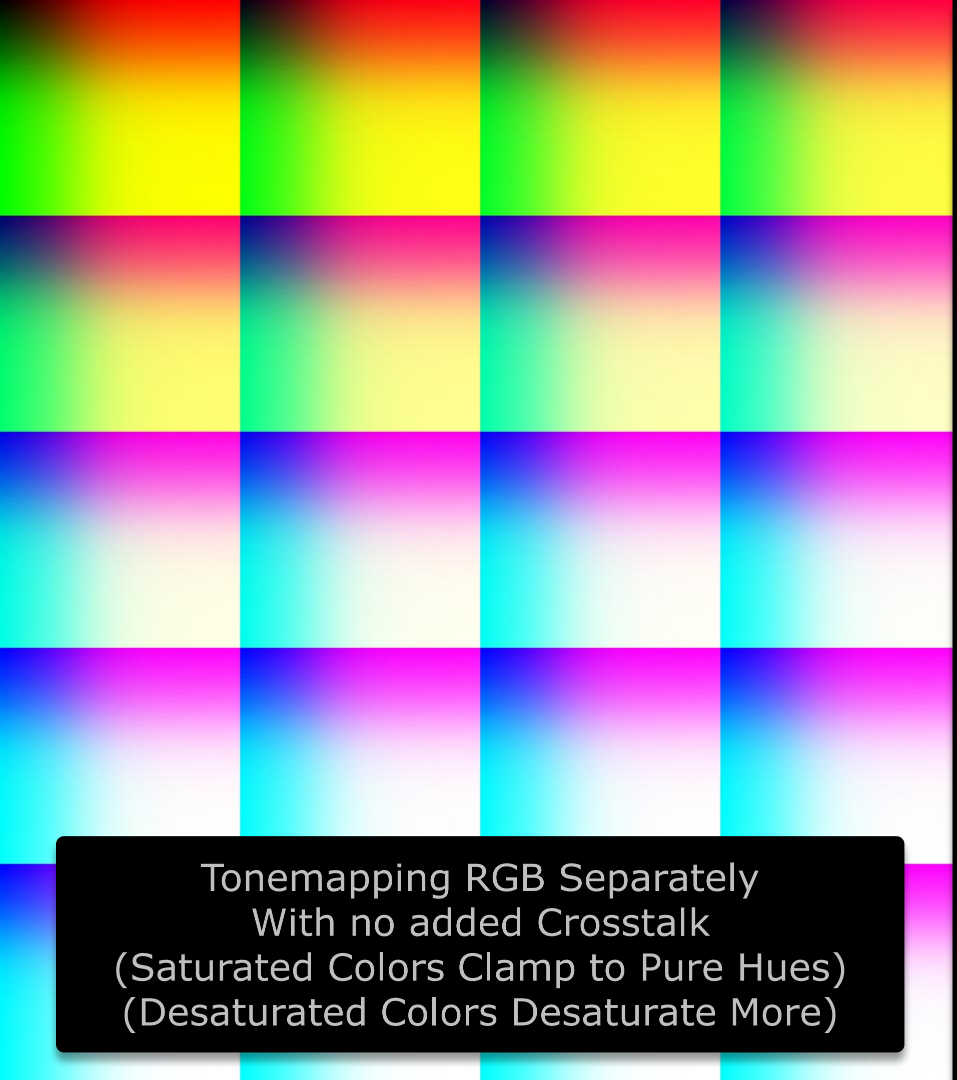
Both show nonlinear plot of highlights
(exp2(rgb*16384)-1)/(exp2(16384)-1)
processed through tonemapper



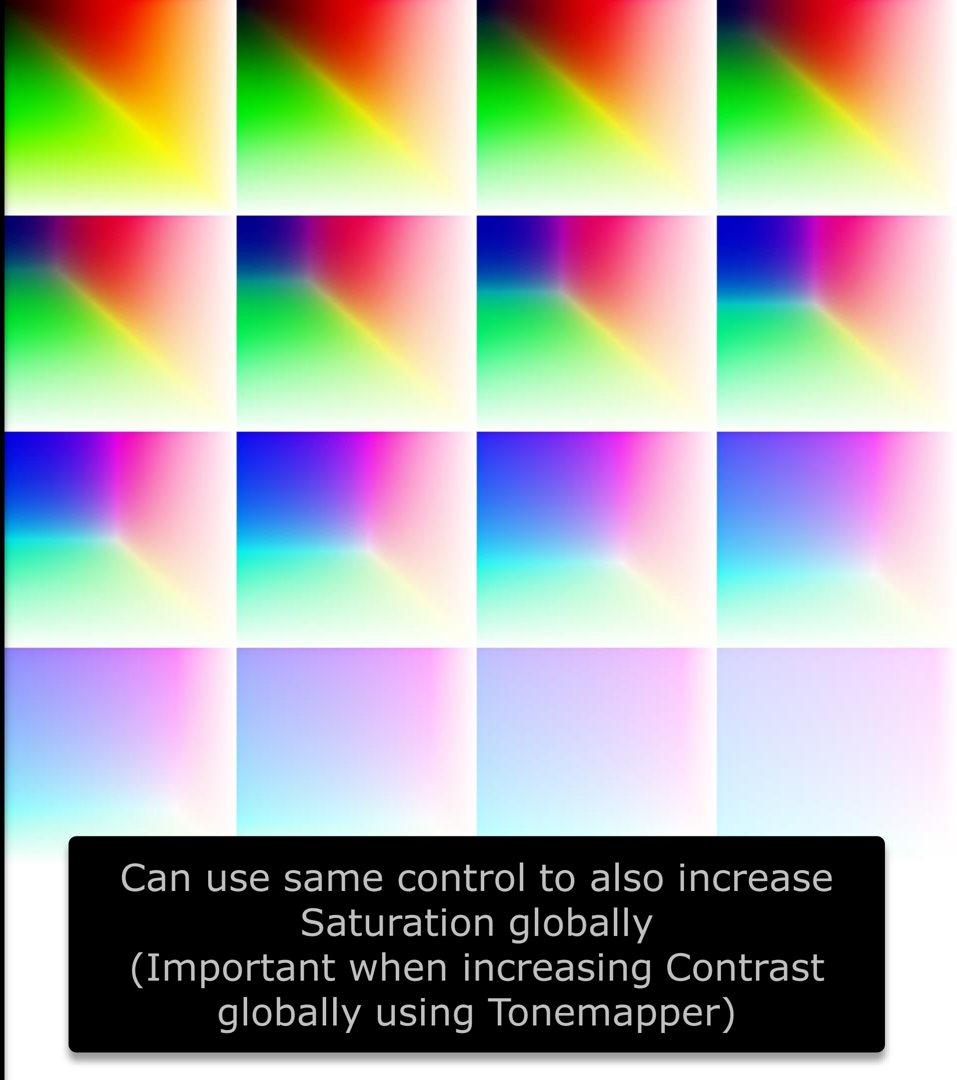
Tonemapping RGB Separately
With no added Crosstalk
(Saturated Colors Clamp to Pure Hues)
(Desaturated Colors Desaturate More)



Tonemap Max RGB, with
“Indirect Path to White”
(Maintains Saturation)



Tonemapping RGB Separately
With no added Crosstalk
(Saturated Colors Clamp to Pure Hues)
(Desaturated Colors Desaturate More)



Can use same control to also increase
Saturation globally
(Important when increasing Contrast
globally using Tonemapper)

```
// improved crosstalk – maintaining saturation

float tonemappedMaximum; // max(color.r, color.g, color.b)
float3 ratio; // color / tonemappedMaximum
float crosstalk; // controls amount of channel crosstalk
float saturation; // full tonal range saturation control
float crossSaturation; // crosstalk saturation

// wrap crosstalk in transform
ratio = pow(ratio, saturation / crossSaturation);
ratio = lerp(ratio, white, pow(tonemappedMaximum, crosstalk));
ratio = pow(ratio, crossSaturation);

// final color
color = ratio * tonemappedMaximum;
```

Tonemapping Optimization

- If tonemapping is applied in texture or bandwidth bound situation, or used to build LUTs dynamically at run-time
  - Will be using shader based implementation
  - Most parameters can be factored out to constants changed per frame

Optimizing for LUT (3D Lookup)

- If tonemapping applied in an ALU limited situation or if doing {color grading, tonemapping, final transfer function} with LUT simultaneously
  - Often a 3D texture will be used
  - Large dynamic range is a challenge

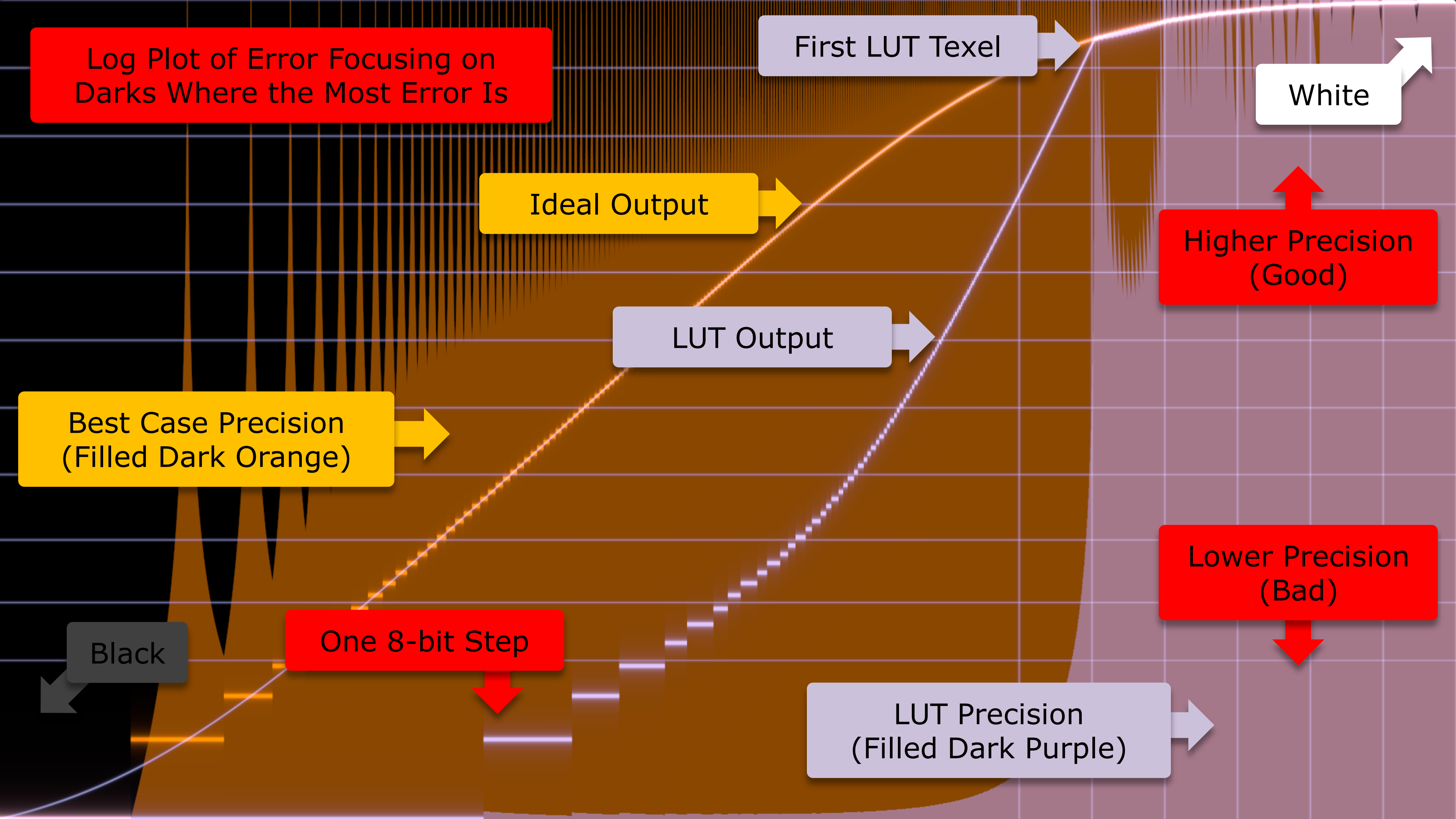
Looking at LUT Error Plots

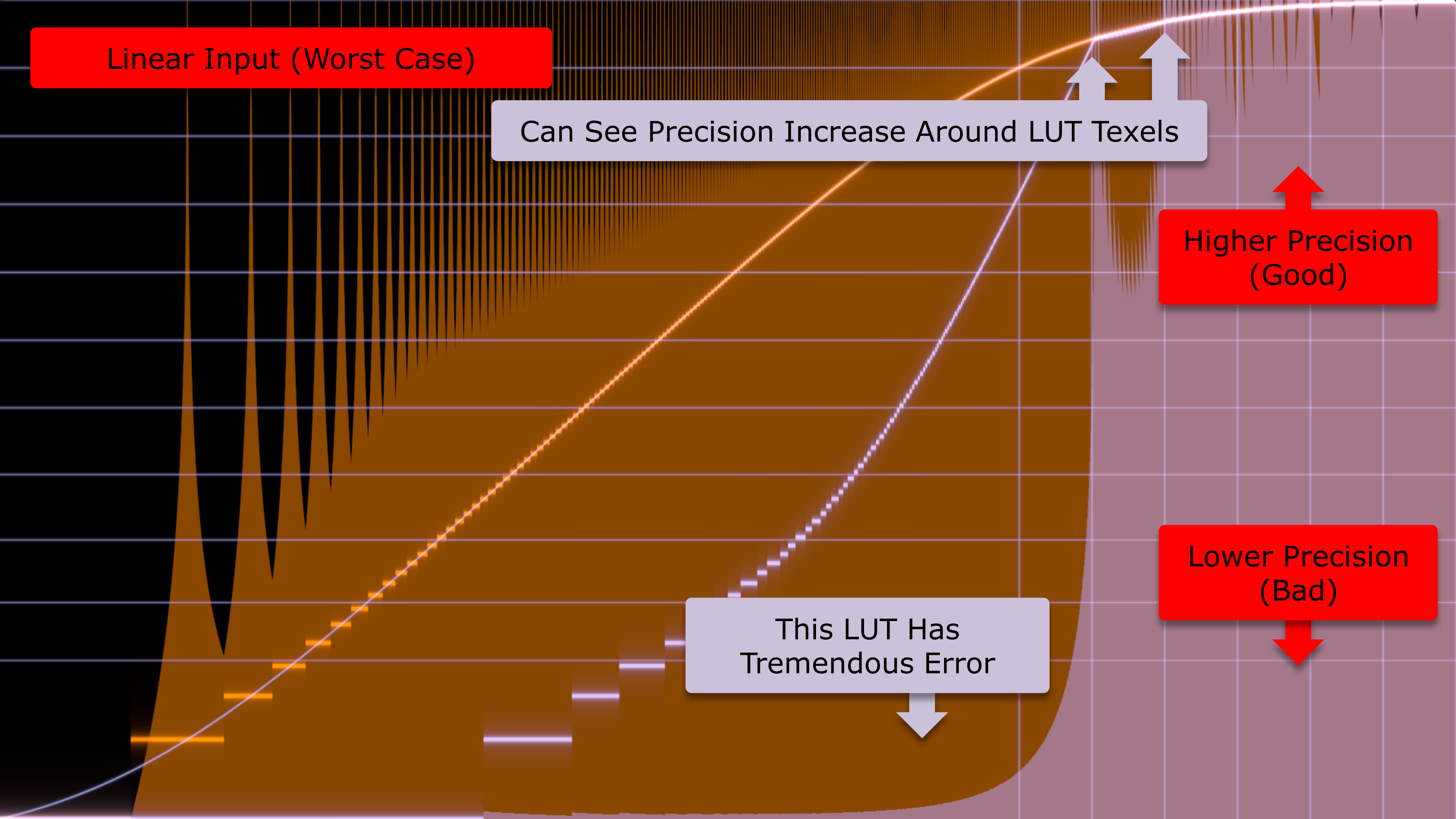
- For the first plots
  - Using x/(x+1) as proxy for tonemapper
  - 8-bit output for sRGB
- Using 6 stops of range above 1.0
- 32x32x32 LUT
  - But showing one dimension in the plot

First Plot = Worst Case

- Using linear input directly
  - output = LUT.Sample(s, linearColor/64.0);

- Providing orientation for understanding the error plots


Log Plot of Error Focusing on Darks Where the Most Error Is
First LUT Texel
White
Ideal Output
Higher Precision (Good)
LUT Output
Best Case Precision (Filled Dark Orange)
Lower Precision (Bad)
Black
One 8-bit Step
LUT Precision (Filled Dark Purple)

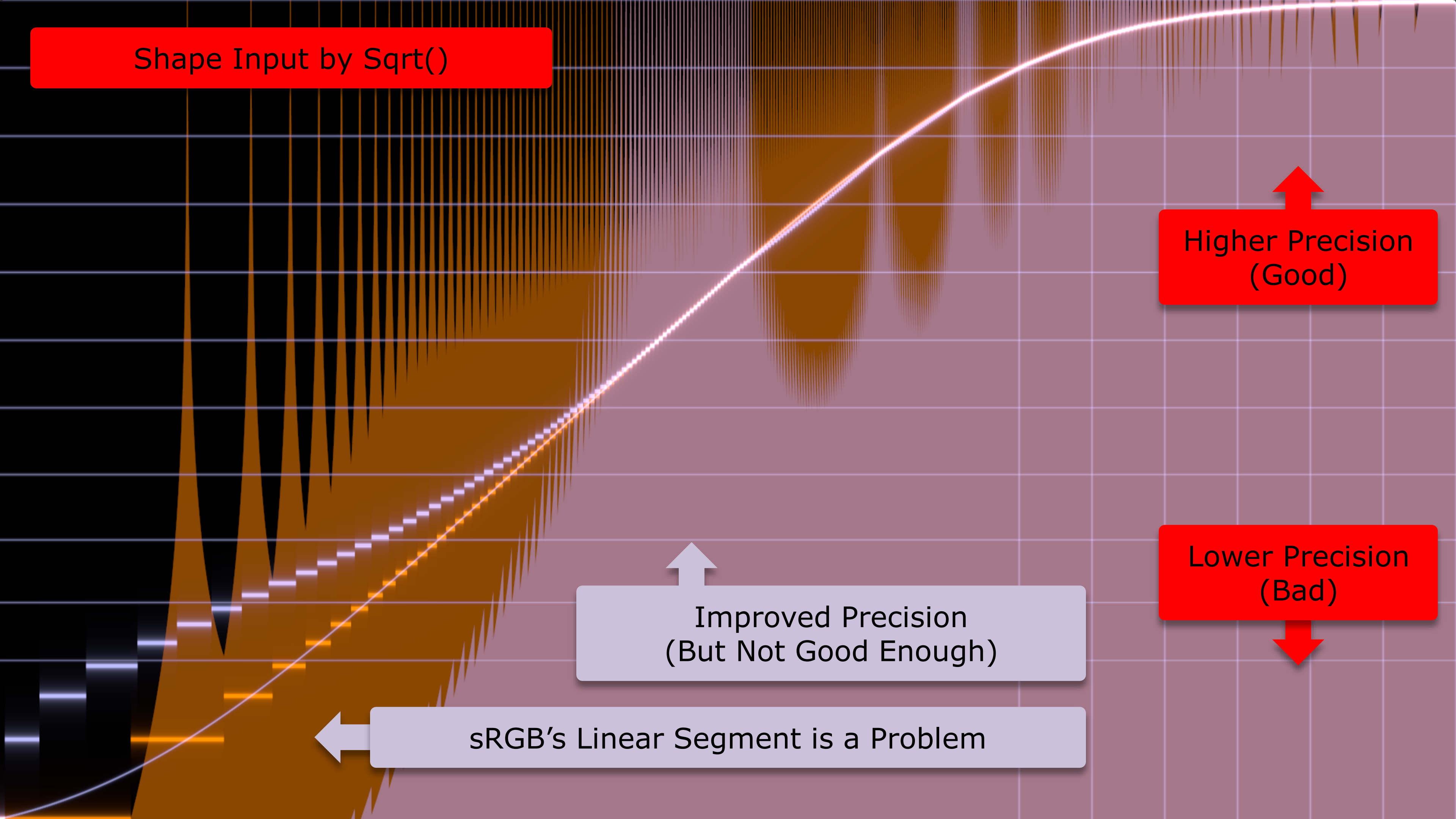

Linear Input (Worst Case)
Can See Precision Increase Around LUT Texels
Higher Precision (Good)
Lower Precision (Bad)
This LUT Has Tremendous Error

2nd Plot = Wrap in Sqrt()

- output = LUT.Sample(s, sqrt(linearColor/64.0)).rgb;

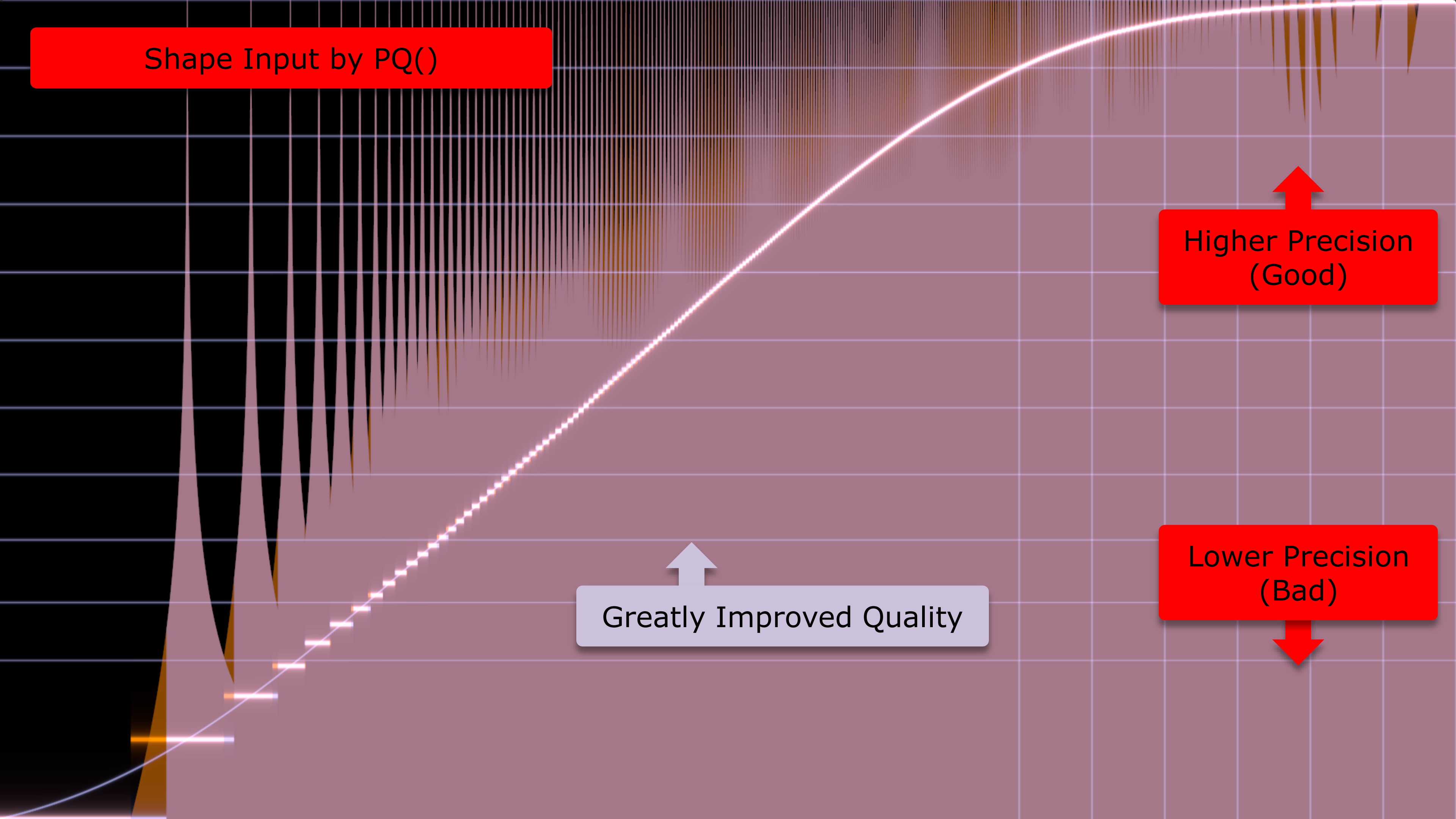
- Pre-shaping input for better LUT texel distribution to the signal

Shape Input by Sqrt()



3rd Plot = Wrap by PQ()

- output = LUT.Sample(s, PQ(linearColor/64.0)).rgb;
- Using SMPTE ST 2084-2014’s Perceptual Quantizer (used for new HDR Video standard)

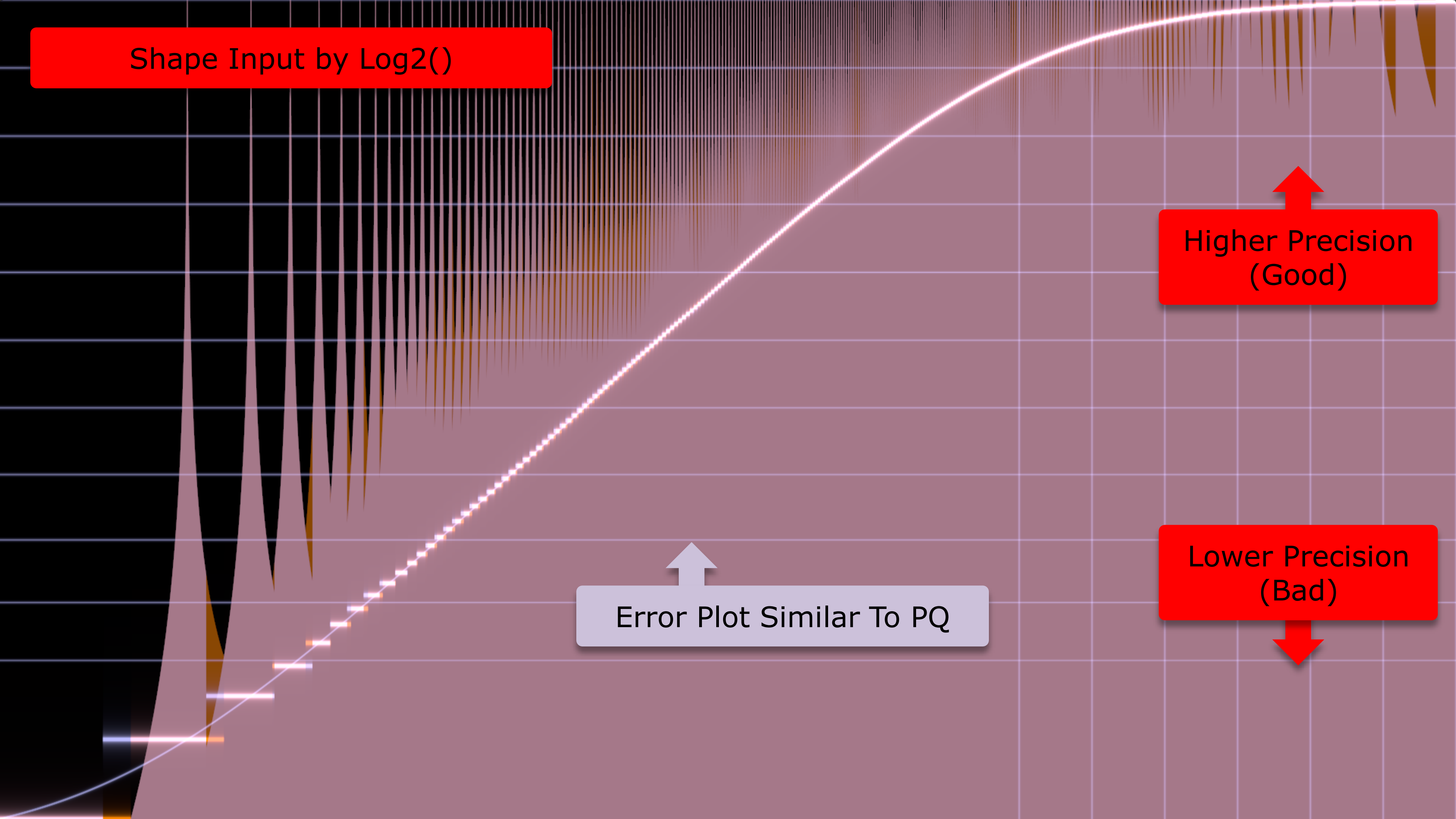

Shape Input by PQ()
Higher Precision (Good)
Lower Precision (Bad)
Greatly Improved Quality

```
// pq from linear
// based on implementation in aces 1.0 (see url)

// https://github.com/ampas/aces-
dev/blob/master/transforms/ctl/utilities/ACESlib.Utilities_Color.a1
.0.1.ctl

float pq(float x) {
    float m1 = 0.1593017578125;
    float m2 = 78.84375;
    float c1 = 0.8359375;
    float c2 = 18.8515625;
    float c3 = 18.6875;
    float p = pow(x, m1);
    return pow((c1 + c2 * p) / (1.0 + c3 * p), m2); }
```

4rd Plot = Wrap by Log2()

- PQ(rgb) requires 15 transcendental ops
  - More than the tonemapper uses

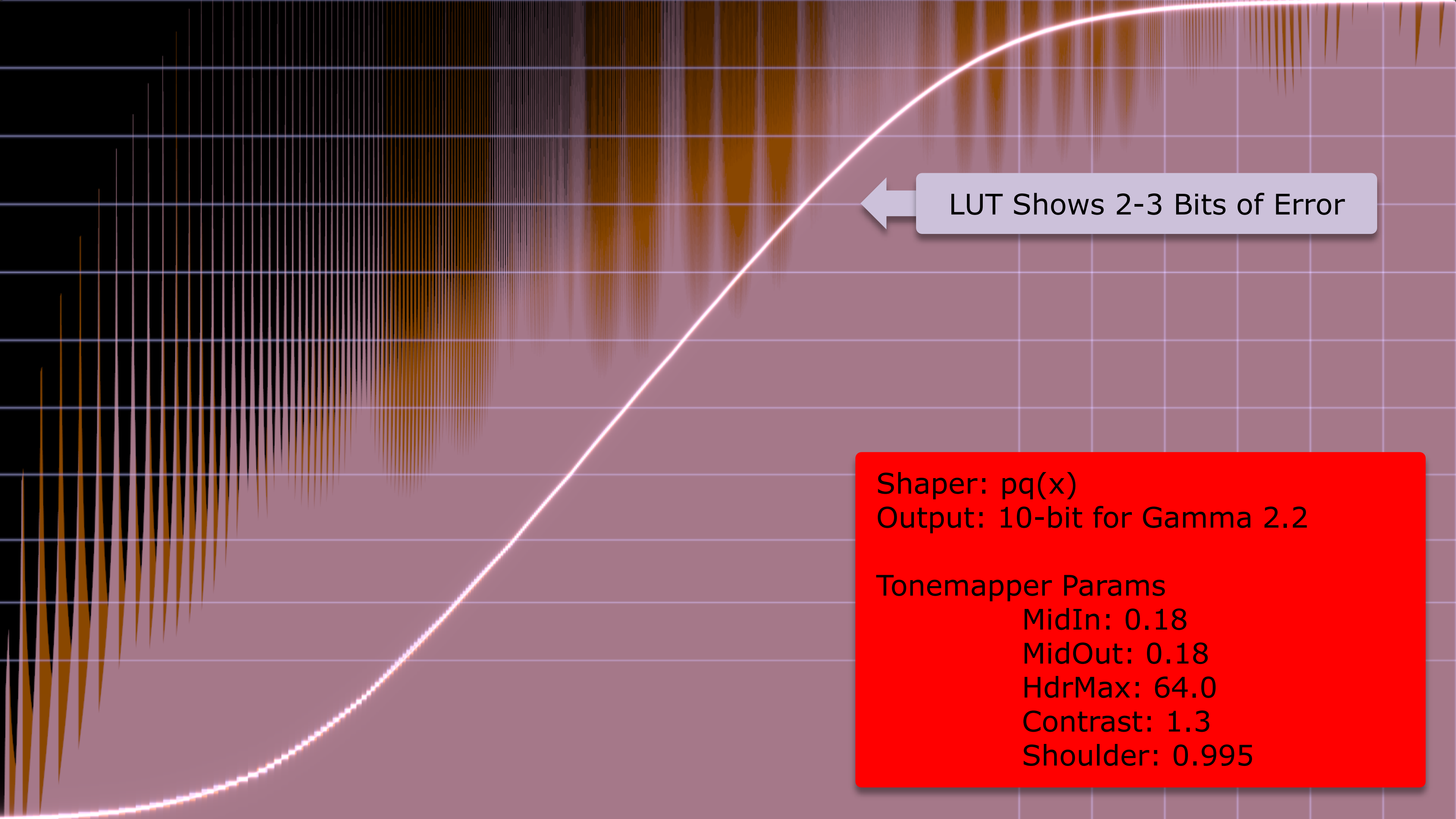
- Want something faster, wrap by
  - log2(rgb * **c** + 1.0) * (1.0 / log2(**c** + 1.0))
  - Choose '**c**' constant to reduce error


Shape Input by Log2()
Higher Precision (Good)
Error Plot Similar To PQ
Lower Precision (Bad)

More Complex Cases

- Showing a mix of different options
  - Using tonemapper from this presentation
  - 10-bit output instead of 8-bit
  - Gamma 2.2 output instead of sRGB

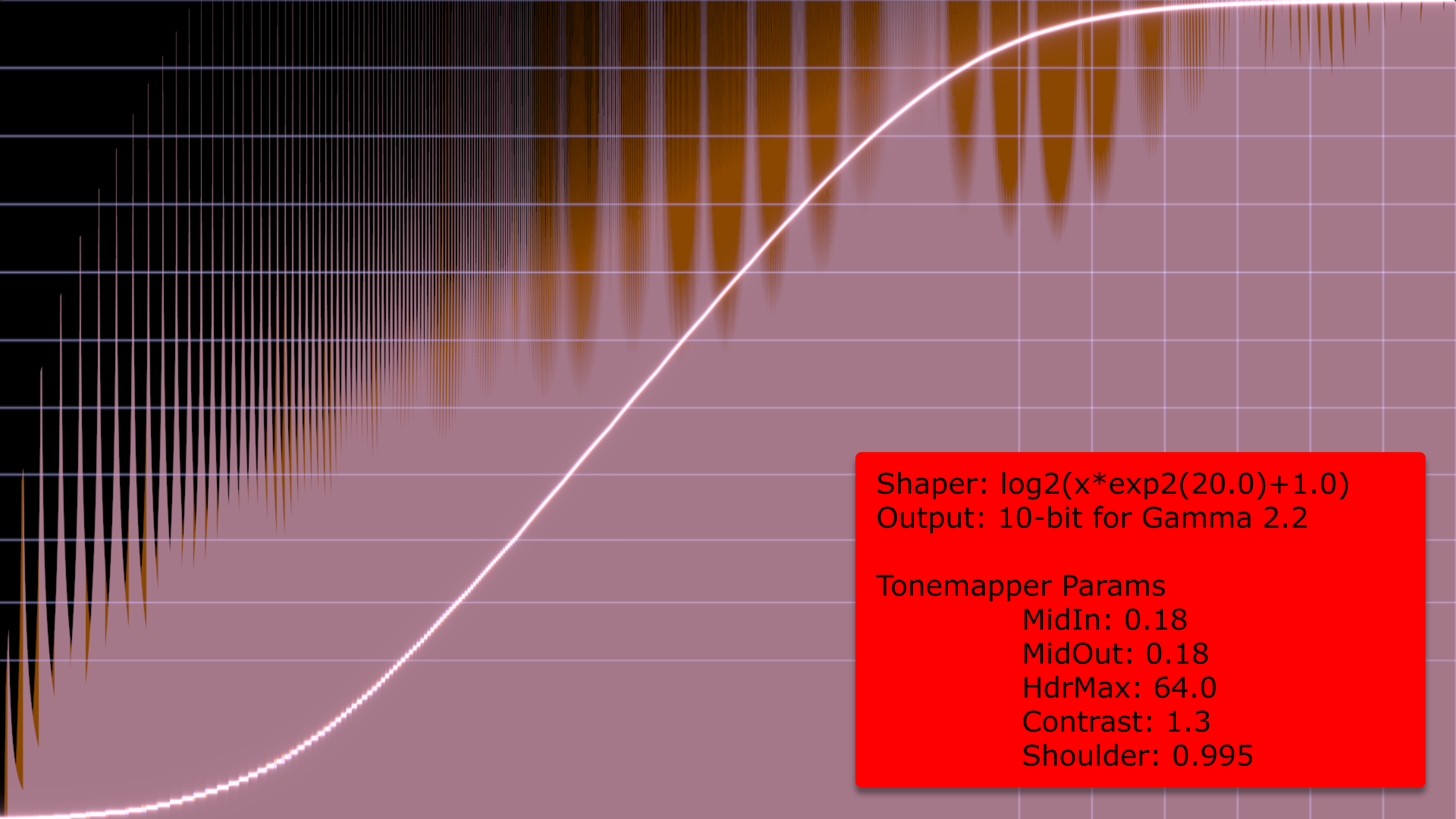
- Unable to get 10-bit precision with 32^3



LUT Shows 2-3 Bits of Error

Shaper: pq(x)
Output: 10-bit for Gamma 2.2

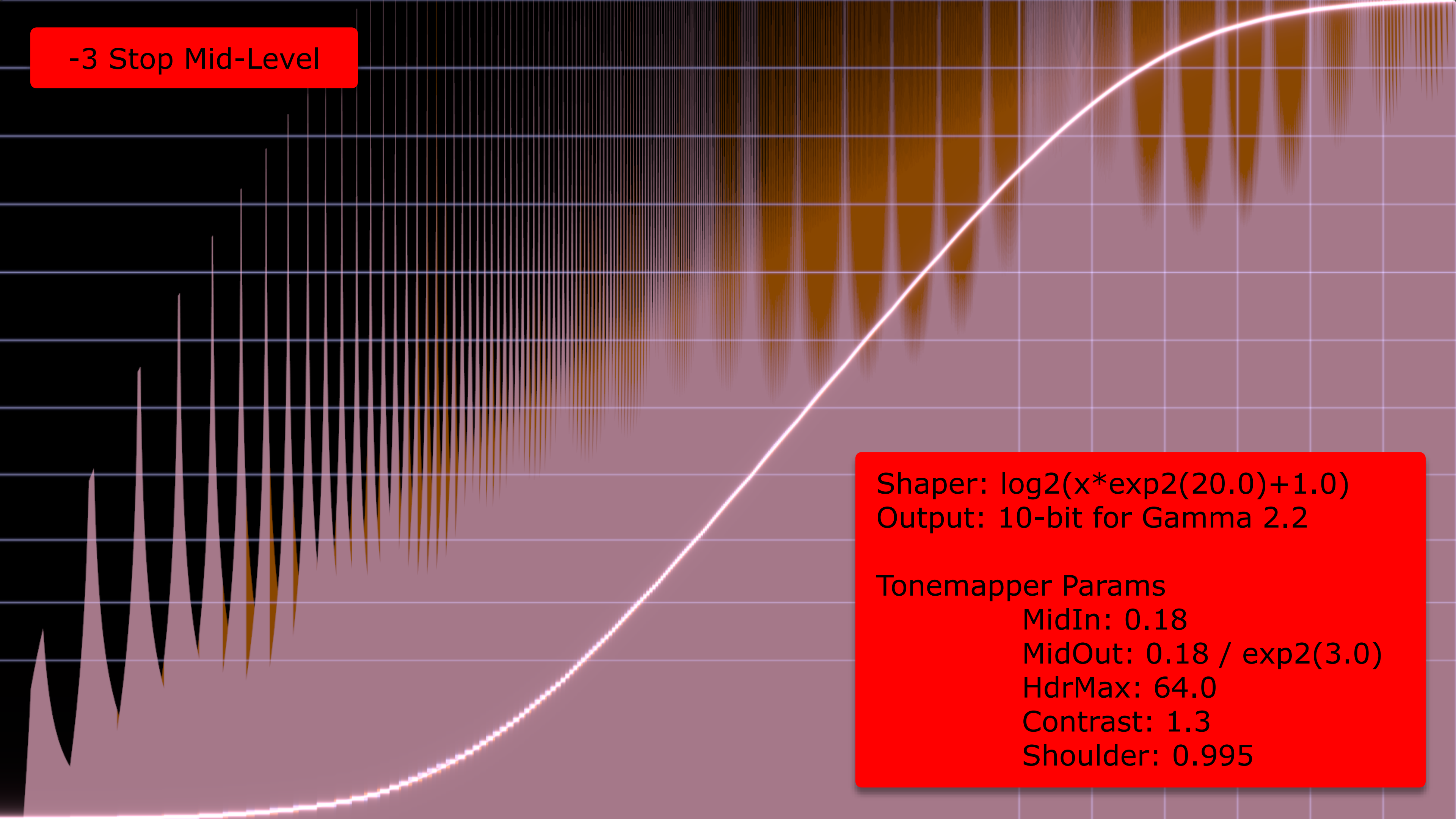
Tonemapper Params
MidIn: 0.18
MidOut: 0.18
HdrMax: 64.0
Contrast: 1.3
Shoulder: 0.995



Shaper: log2(x*exp2(20.0)+1.0)
Output: 10-bit for Gamma 2.2

Tonemapper Params
MidIn: 0.18
MidOut: 0.18
HdrMax: 64.0
Contrast: 1.3
Shoulder: 0.995

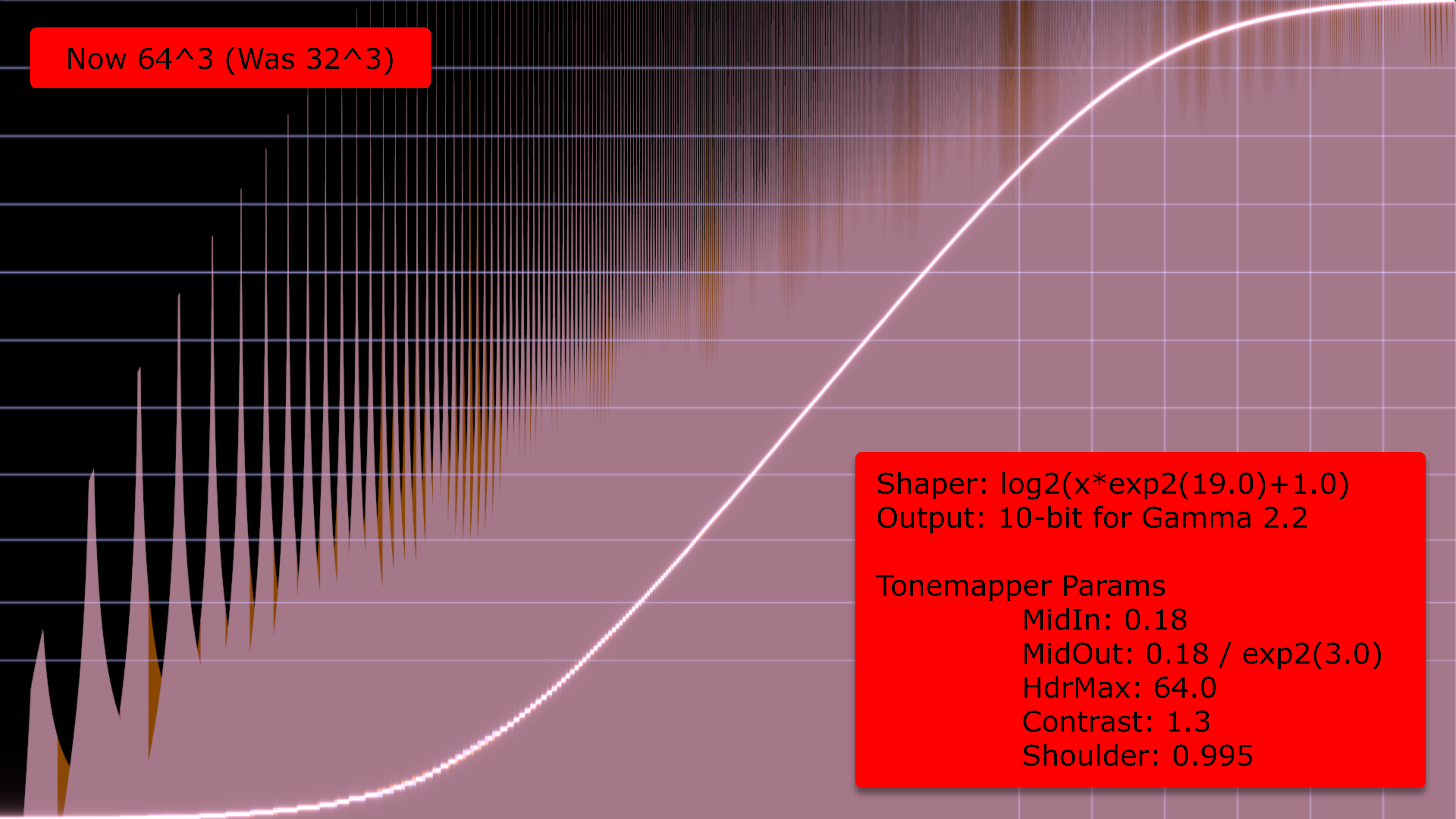
-3 Stop Mid-Level



Shaper: log2(x*exp2(20.0)+1.0)
Output: 10-bit for Gamma 2.2

Tonemapper Params

MidIn: 0.18
MidOut: 0.18 / exp2(3.0)
HdrMax: 64.0
Contrast: 1.3
Shoulder: 0.995



Now 64^3 (Was 32^3)

Shaper: log2(x*exp2(19.0)+1.0)
Output: 10-bit for Gamma 2.2

Tonemapper Params
MidIn: 0.18
MidOut: 0.18 / exp2(3.0)
HdrMax: 64.0
Contrast: 1.3
Shoulder: 0.995

LUT Error Plot Takeaways

- Various options to customize to goals
  - Log2() input shaper
  - Can run LUT after ALU based tonemapping and after application of transfer function
- Could also try 32x64x32 (RxGxB) LUT
  - Green more perceptually noticeable

Part 3 – Transfer Functions & Quantization

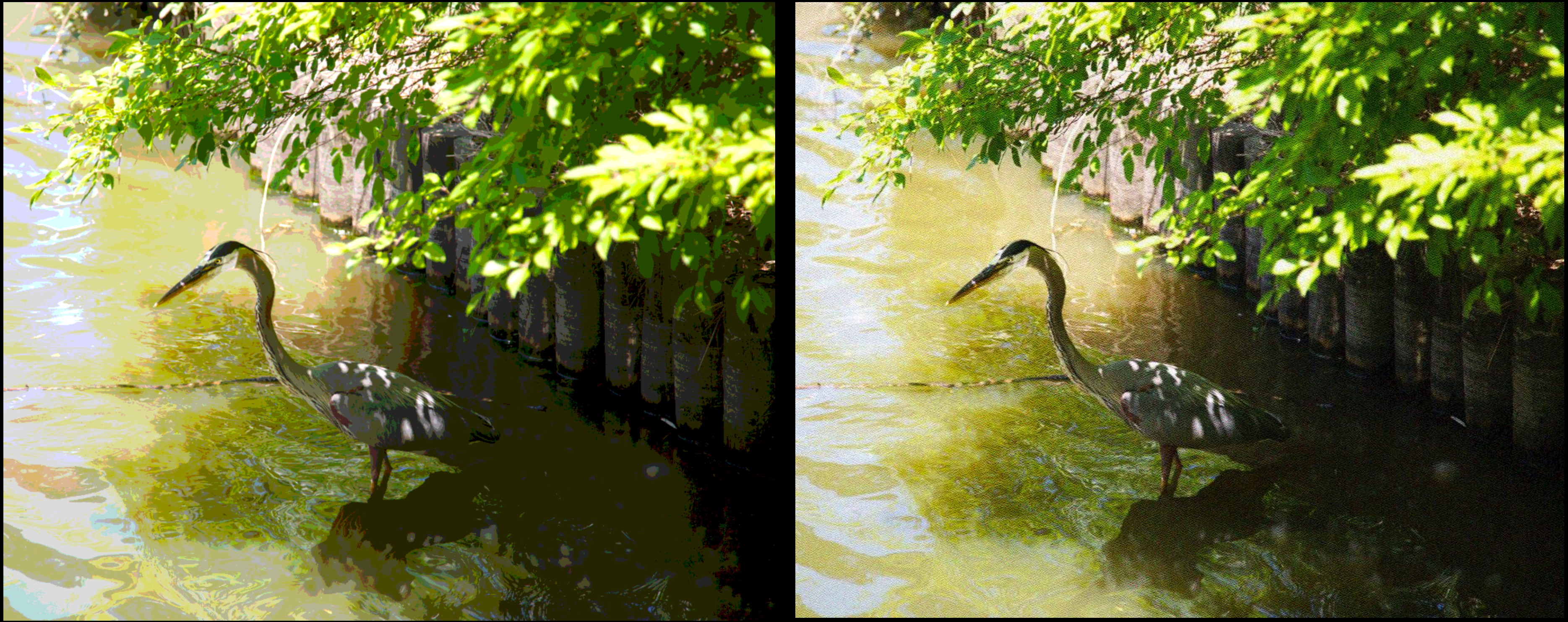
Techniques to get high quality output on any display

Quantization Goals

- **Enable high quality blacks at any bit depth!**
- Energy-preserving (won’t change tone/color)
- Can be applied and work for different transfer functions even if app can not query transfer function OS is using
- Can be artistically controlled to serve same purposes as film grain

Quantization Example

- Next slide shows an example of what is possible with static high quality quantization techniques described in this presentation
  - Uses a high quality grain texture
  - Looks even better temporally



Both Shots Are 3-Bits/Channel

Transfer Function?

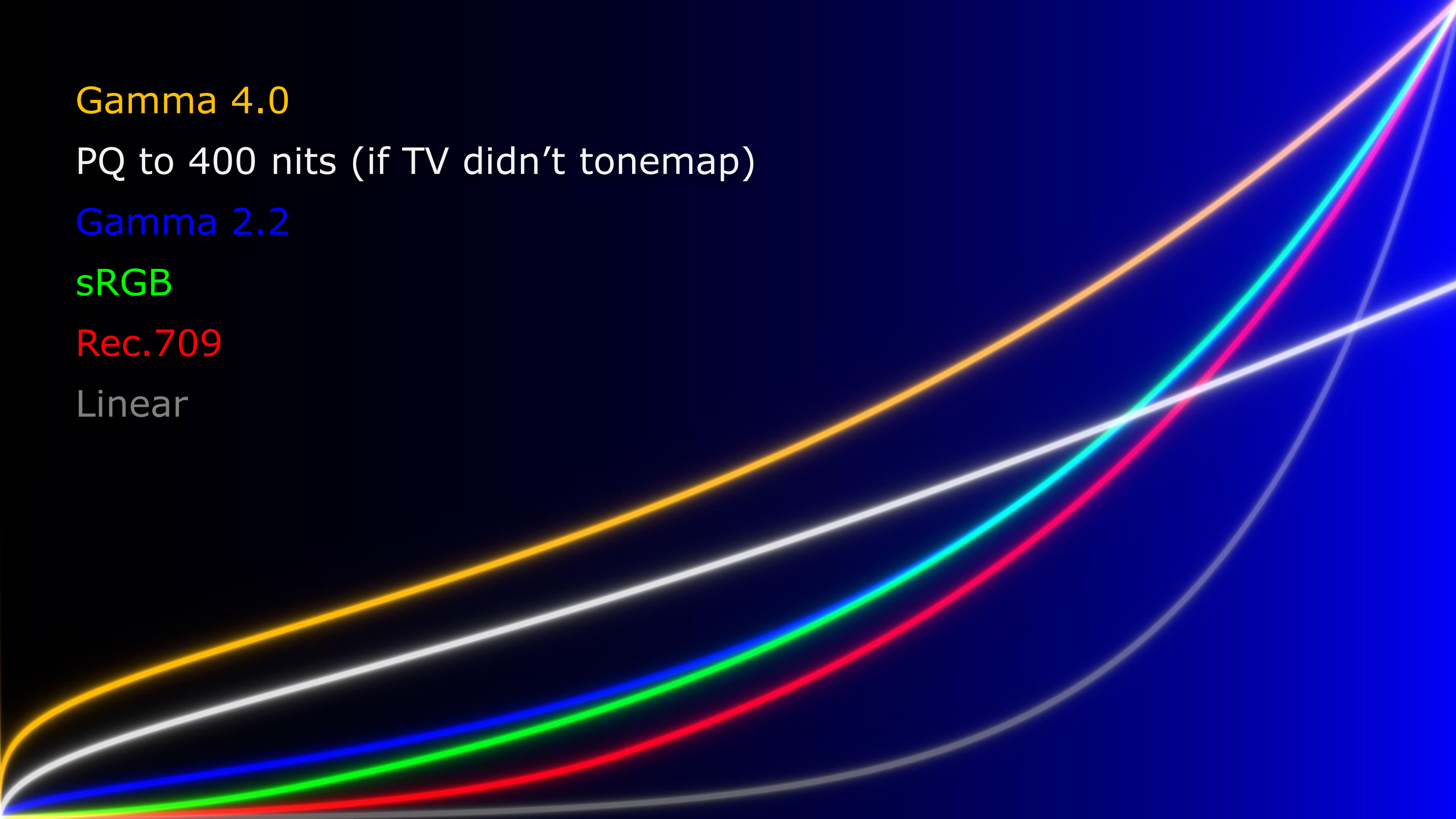
- Often impossible to query or know on PC
- Useful to have user selection
  - Via in-game video settings

PQ Transfer Function

- HDR TVs add Perceptual Quantizer [PQ]
  - SMPTE ST 2084-2014
  - Peak = absolute 10,000 nits
  - Not display relative like other standards
  - Displays variably tonemap content
  - PQ allocates more bits to darks

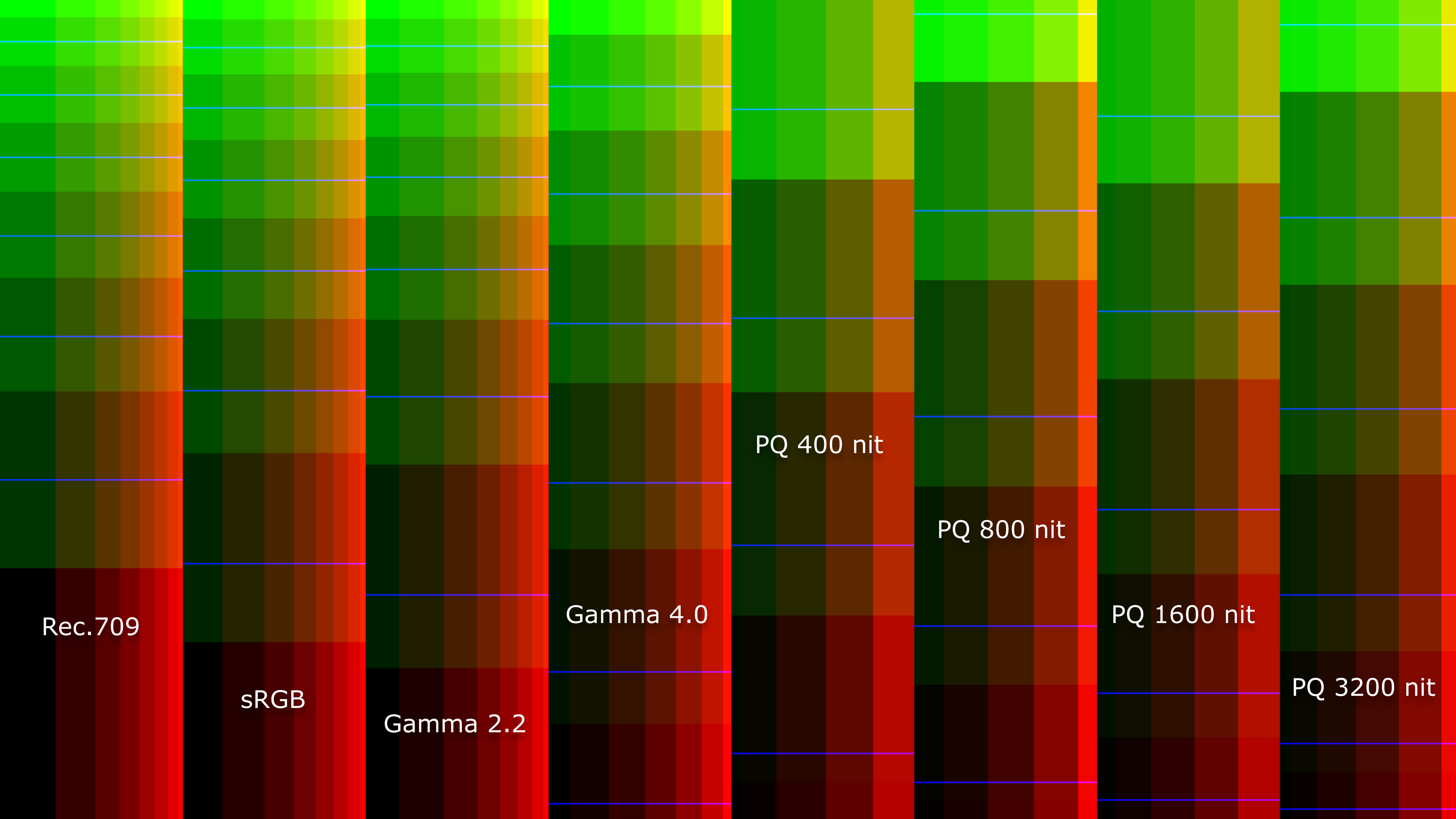
HD+HDR TV Transfer Functions

- TVs set to Game Mode and Input Renamed "PC": unofficial convention for
  - Reducing input latency
  - Getting less random color transforms
  - Drive HDR TVs like classic monitors (use gamma transfer function without PQ's bundled tonemapping)


Gamma 4.0
PQ to 400 nits (if TV didn’t tonemap)
Gamma 2.2
sRGB
Rec.709
Linear

Transfer Functions At 3-Bits

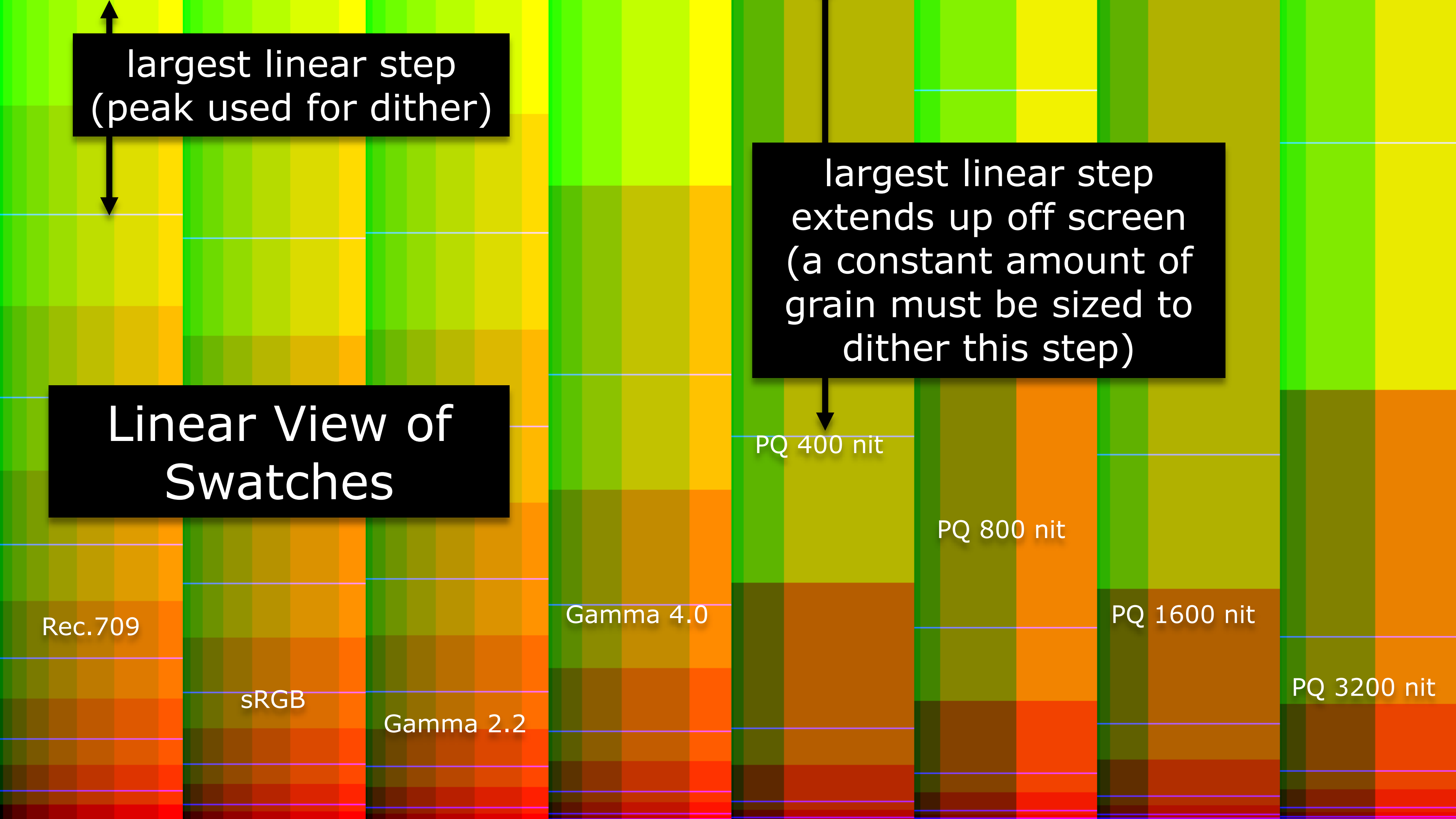
- Showing linear nearest quantization
  - At 3-bits/channel to visualize differences
  - Blue line provides quantization center
  - Log display (like prior slide)
  - PQ shown with different display peaks (values near or above peak get tonemapped)


Rec.709
sRGB
Gamma 2.2
Gamma 4.0
PQ 400 nit
PQ 800 nit
PQ 1600 nit
PQ 3200 nit

Dithering At 3-Bits

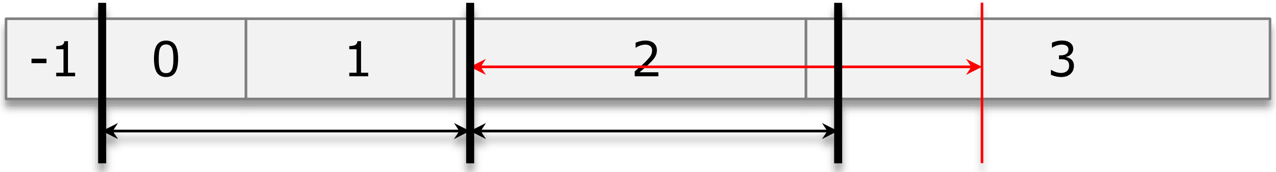
- Simple energy-preserving dither
  - Add constant amount of grain in linear
  - Done before transform and quantization
  - Blacks corrected, peaks not (error at top)
- HDR Transfer Functions show “banding”
  - Because of symmetrical (-/+) grain

Linear View of Swatches



Peaks Not Energy Preserving

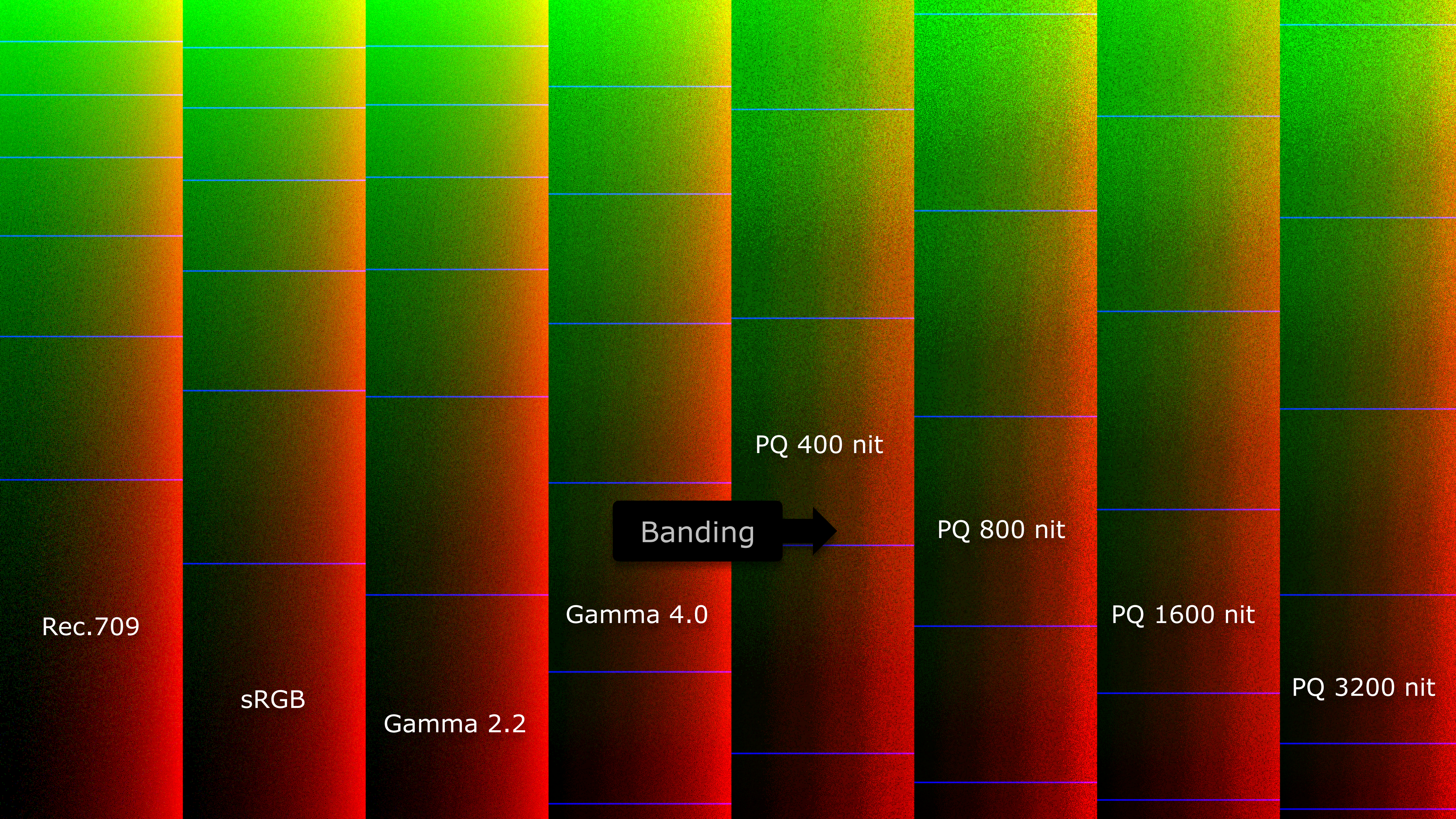
- Not correcting for lack of values above 1.0 in this example
  - Not important as bit/pixel increases
- Correcting blacks
  - Important as bit/pixel increases



```
// simple energy-preserving dither

float3 color; // input color in linear
float3 grain; // input {-1 to 1} grain value

// constant, step size in non-linear space
float rcpStep = 1.0 / (steps - 1.0);

// constant, amount negative which still quantizes to zero
float black = 0.5 * OutputToLinear(rcpStep);

// constant, biggest linear step size * overlap
float biggest = 0.75 * (OutputToLinear(1.0 + rcpStep) - 1.0);

// add grain (3 adds, 3 mins, 3 mads)
return color + grain * min(color + black, biggest);
```

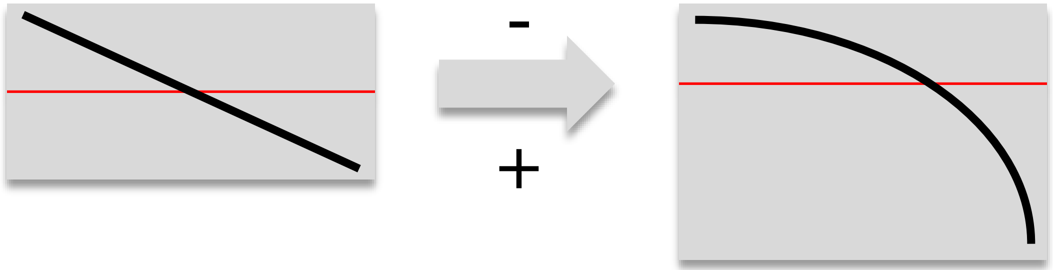
Grain Texture Note

- Using low-quality grain
  - Simple ALU based grain in Shadertoy
  - Easier to see banding problems

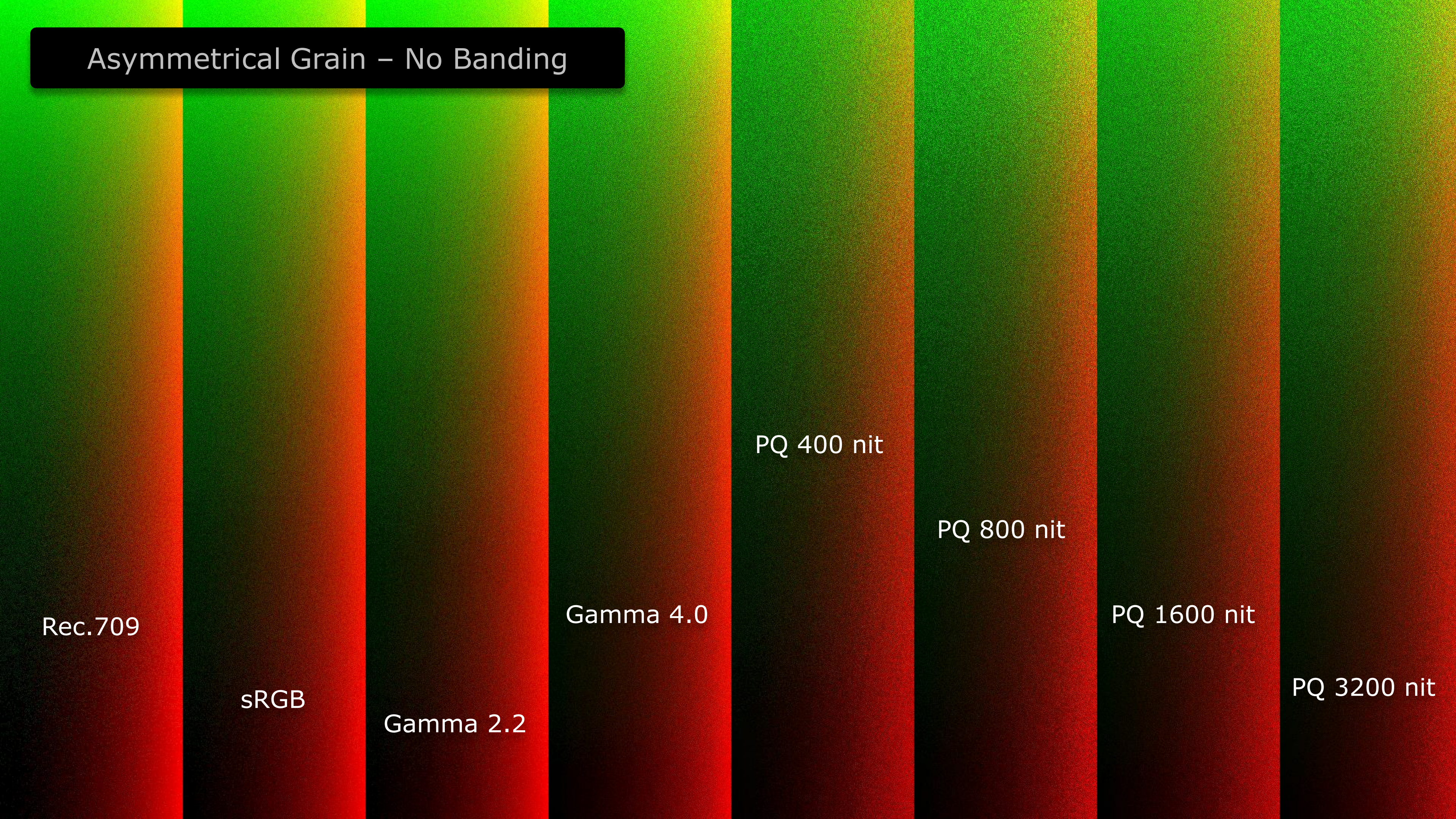

Rec.709
sRGB
Gamma 2.2
Gamma 4.0
Banding
PQ 400 nit
PQ 800 nit
PQ 1600 nit
PQ 3200 nit

Asymmetrical Grain Distribution

- Required with HDR Transfer Functions
  - Shift so negative side gets more grains
  - Shift so positive side gets higher values
  - Offset and scale to maintain equal area (-/+)



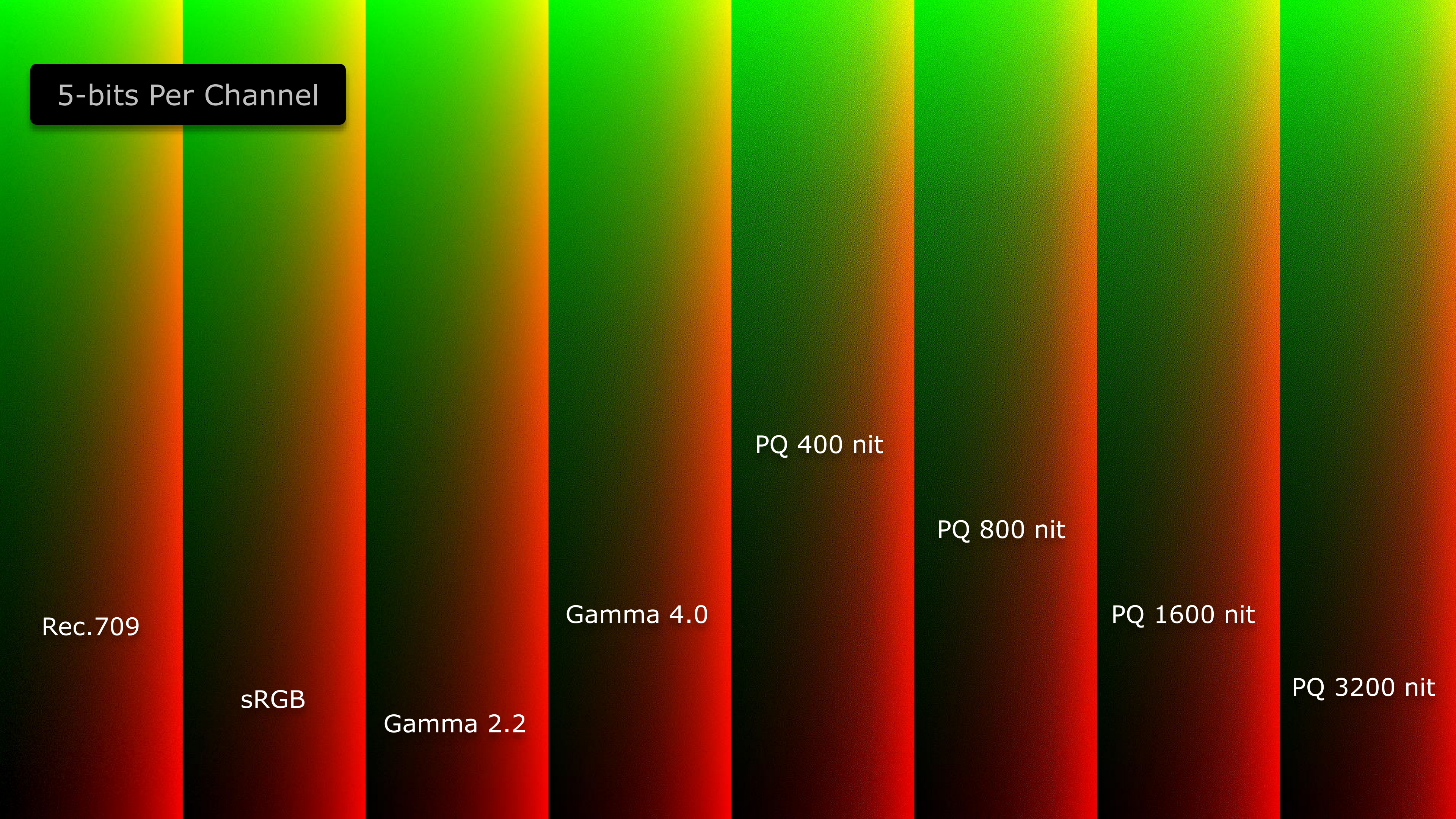
Asymmetrical Grain – No Banding



Now at 5-Bits

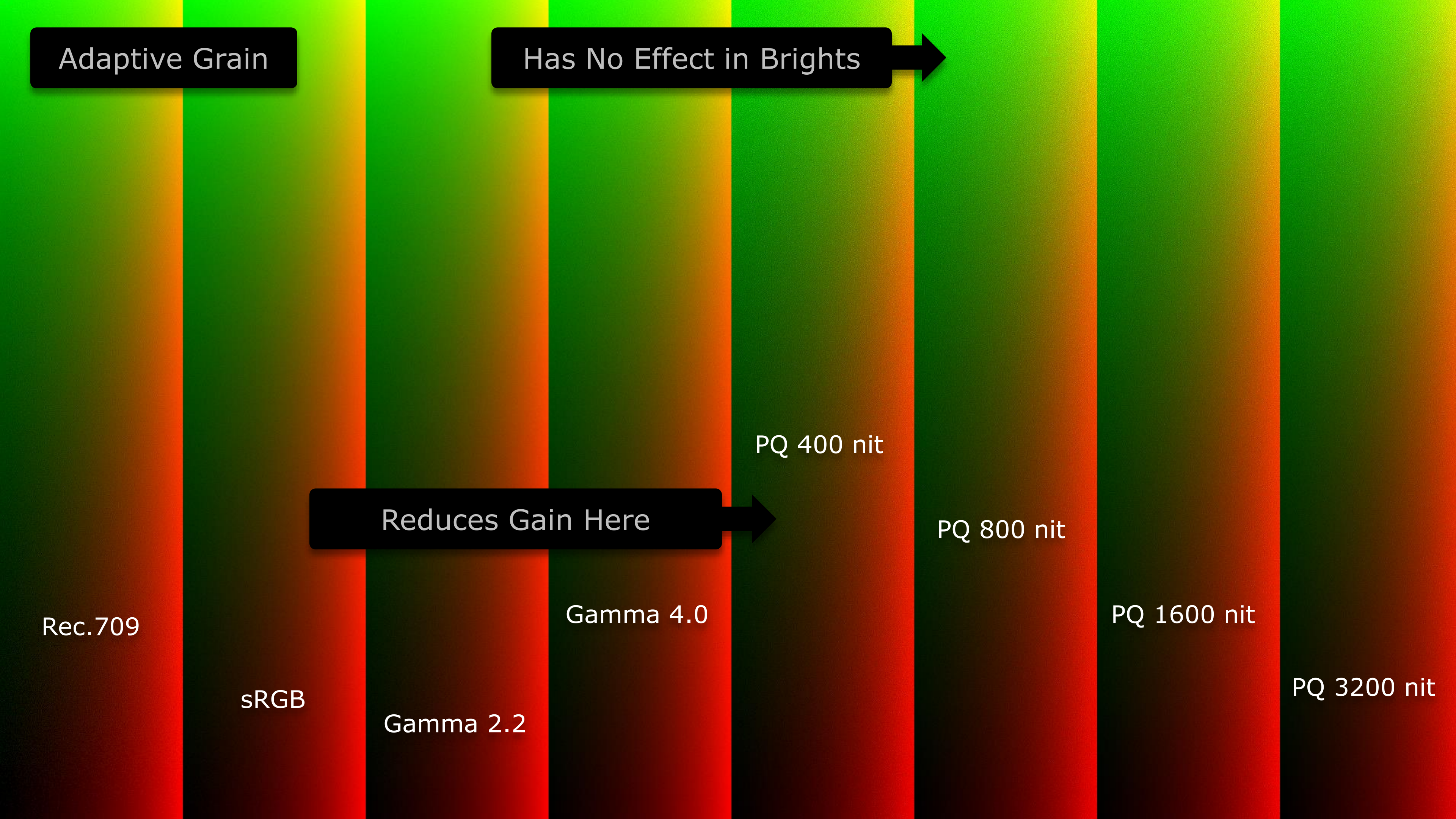
- HDR Transfer Functions
  - Need higher amounts of constant grain because they have a larger max linear step size (located in towards the peak)
  - PQ examples take peak step at display peak nits (instead of at 10,000 nit encoding peak)

5-bits Per Channel



Adaptive Grain at 5-Bits

- Adapting the grain to the step size
  - Requires computing step size per channel
  - More GPU work
  - Workaround for constant grain with PQ
  - Might not be required at high bit depths


Adaptive Grain
Has No Effect in Brights
PQ 400 nit
Reduces Gain Here
PQ 800 nit
Gamma 4.0
PQ 1600 nit
Rec.709
sRGB
Gamma 2.2
PQ 3200 nit

Takeaway

- Use high quality quantization to get high quality output for any bit depth and transfer function
  - Great solution to remove banding for traditional LDR games as well

Part 4 – Choose Your Own Adventure

Thanks for watching

For questions or discussion, Timothy.Lottes@amd.com