UHD Color for Games

Author: Evan Hart
Date: June 14, 2016
Version: 1.0

Introduction

With the advent of Ultra High Definition television and the UHD Alliance[1] specification for televisions, displays are taking a substantial step forward compared to the standards developers have been used to for the past two decades. While this is a specification for televisions, the technologies involved will ultimately impact many classes of displays. This shift in display standards presents substantial challenges for anyone concerned about the quality of images they display. The audience for this paper is any graphics programmer interested in being ready for this transition, but in particular it focuses on challenges tied to game development.

Microsoft and Hewlett-Packard created the sRGB standard in 1996. This standard allowed reasonably faithful representation of images on most computers. In many ways, it was standardizing what was already in use, as the standard conforms to what was implemented in many CRT displays at the time. The standard not only specifies the 'gamma' function most graphics programmers think of, but it also specifies the chromaticities of the red, green, and blue color primaries as well as the white point and the maximum luminance. Similarly, the rec.709 specification describes these properties for HDTVs. It differs somewhat in that it has a dimmer expected viewing environment. (Viewing environments impact the perception of properties of images, such as contrast, resulting in modified gamma correction curve, etc)

The new UHD Alliance specification brings about new versions of components that defined the sRGB space graphics programmers have been living in for two decades. The specification has a new set of color primaries known as BT.2020 or Rec.2020. It has a new maximum luminance that is as much as 125x (10000 nits) the one specified for sRGB, and finally, it also has a new non-linear encoding function known as the Perceptual Quantizer (PQ) or SMPTE 2084.

How is it better?

The single most important question with any new technology is always, "Why?" The changes with the new display standards allow for the creation of much more realistic and engaging images. The two dimensions of improvement are often thought of as wider gamut and higher dynamic range. Higher dynamic range is easy to understand as

[1] UHD Alliance is an industry group setting standards for the quality of Ultra HD televisions. (http://www.uhdalliance.org/) The standards for encoding and transmission are done by other bodies, such as the ITU.

games have been rendering to a higher dynamic range and tone mapping down to standard dynamic range (SDR) for many years. Wide gamut is a bit more nuanced. It refers to the intensity of the hues that can be generated. The gamut supported by sRGB cannot represent many of the colors experienced in daily life.

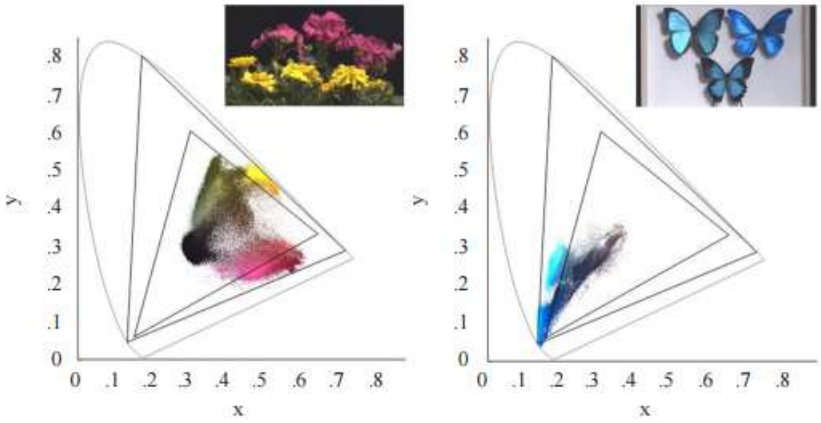


Figure 1 - Gamuts of items found in nature (From ITU-R BT.2246-5)

Figure 1 visualizes real examples of how objects we see in real life need a wider gamut than what prior specifications for displays supported. The inner triangle represents the limit of colors reproducible with prior display standards, and the larger triangle is the extent of what can be handled by the color space used in the new UHD standards.

Display Advancements

A fair question to ask might be why displays are taking this sudden jump forward. The simplest answer might be a convergence of technology. For the past two decades, displays have mostly improved in size and weight (CRT to LCD) with some improvements in brightness and resolution. Now, displays are improving in other dimensions such as purity of colors, contrast, and brightness. These enhanced capabilities go well beyond the present standards for displays.

Background

Colors are such a central part of computer graphics that we often forget how complex they really are. This section serves as a reminder of many things we forget, and it attempts to provide a somewhat more rigorous definition of terms and concepts we work with daily. (What is a color space? What is sRGB?) You don't need to know and understand all this information to take advantage of new display technology, but it is intended to help understand the directions recommended later. As entire books and theses can and have been written on the topics here, this is not intended as a thorough discussion, but as a practical overview and departure point for future study if desired.

Human Visual System

Given that the central goal of the creation of any image in computer graphics is for it to be seen by humans, a natural place to start describing information related to color is the human visual system. As we all know, real-time computer graphics has a giant component of approximation to it. Understanding a bit about vision allows us to make the right tradeoffs when attempting to generate images to take advantage of the new displays.

As most everyone involved in computer graphics knows, humans see in what is known as trichromatic vision, unless they are colorblind. This is why displays using the three color framework of RGB can produce what appear to be convincing color images. In reality, the cells responsible for color vision (cones) are fairly broadband in their sensitivities. Colors that roughly equate to what we'd call red, green, and blue occupy the peak sensitivities of the three types of cones which are typically referred to as long, medium, and short (LMS) corresponding to the wavelengths of light they sense. Figure 2 shows how the sensitivities of the cones vary across wavelengths for the average human. Because color perception is based on a linear combination of these signals, colors can be mathematically represented as a linear combination of (at least) three basis vectors. Every such combination doesn't work for stimulating the eye, as the colors chosen need to offer isolation with respect to the three cones, but any reasonable combination can be used as a representation.

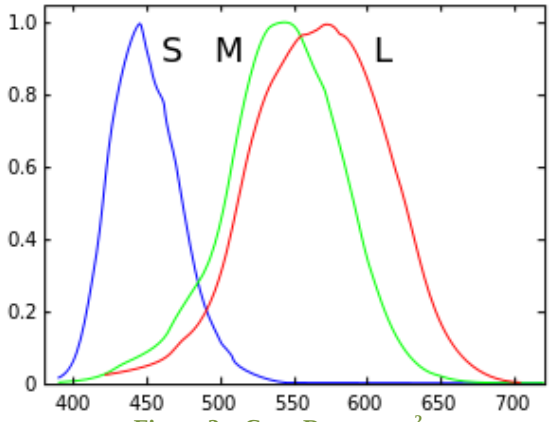


Figure 2 – Cone Responses [2]

The range of sensitivities of the cones leads to the set of colors which humans can perceive. This is typically referred to as the visual locus. The visual locus is often displayed in a chromaticity diagram where the colors are displayed with a constant level of brightness but varying hue and saturation. Figure 3 shows such a diagram in the most common representation. These chromaticity diagrams generally take on a horseshoe-like shape where the curved boundary of the horseshoe equates to spectrally pure colors (a single wavelength of light, like a laser). Anything outside this curve is termed an imaginary color, as it can't be physically created. The straight region at the end of the horseshoe is often called the "line of purples". These are colors

[2] "Cones SMJ2 E" by Vanessaezekowitz at en.wikipedia. Licensed under CC BY-SA 3.0 via Commons - https://commons.wikimedia.org/wiki/File:Cones_SMJ2_E.svg#/media/File:Cones_SMJ2_E.svg

perceived primarily as a combination of L and S cones. Anything outside this line may be physically realizable, but it can't be perceived by humans.

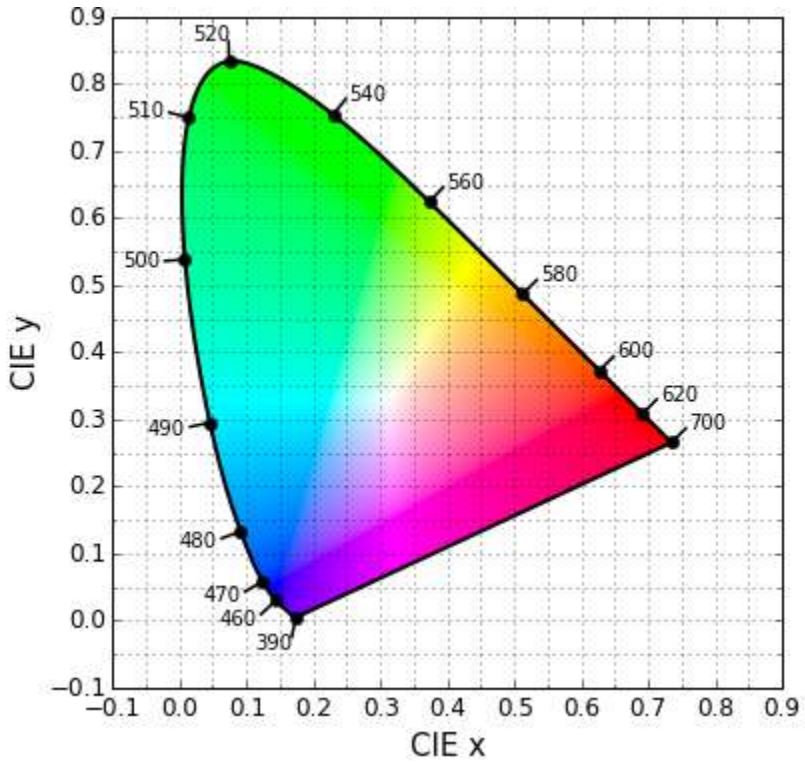


Figure 3 – Chromaticity diagram in xyY color space

While absolute responses, like the spectral sensitivities of the cones are important to describing human vision, vision is heavily impacted by processing and perceptual factors. As it turns out, the brain doesn't really receive a bunch of LMS signals. Instead, filtering happens in cells at the back of the eye that produces a set of signals that roughly correspond to luminance, red-green opponency[3], and yellow-blue opponency. Because color information is broken into opponent signals, we don't perceive reddish greens or yellowish blues. Also, these color opponency signals are perceived at a lower resolution than the luminance signal, allowing common types of lossy image compression, like JPEG (YCbCr with chroma subsampling).

The luminance signal perceived by the human visual system is capable of adapting to conditions that exceed 1,000,000,000:1 range. (where sunlight can be over one million times brighter than moonlight, and starlight can be over one thousand times dimmer than moonlight) While we are capable of experiencing this enormous range, it isn't possible to experience it simultaneously. It requires adaptation that cannot occur instantaneously. This is why headlights are blindingly bright at night, but not during the day. Within a set level of adaptation humans can discern roughly six orders of

[3] Opponency is value describing some weighted difference between two signals. Red-green opponency is simply a differential signal between the L and M cones giving the impression of redness or greenness.

magnitude difference in luminance. (the monochromatic rods active at low luminance are being ignored here) Anything outside these levels will saturate. Further, the response curves are highly non-linear. They take on a sigmoidal (S-curve) shape in the log domain, which means that our perception of brightness is non-linear, and that we have a mid-tone range of roughly two to four orders of magnitude where we are most sensitive.

As those familiar with photography may know, the non-linearity of this response curve leads to the definition of what is commonly referred to as middle-gray. Middle gray is generally taken to be an 18% reflective gray. As the word middle implies, it is generally perceived as roughly the 50% level of perceived brightness in a scene. This value is commonly used for setting a base level for exposure on a scene, and it is no different in the recommendations later in this paper.

The non-linear fashion in which we perceive the intensity of luminance also leads to interesting implications in the representation of color data. Steps in luminance are often correlated against a metric known as Just Notable Difference (JND). A step in luminance that matches the JND for a given level of luminance will be the smallest amount where a viewer can discriminate a change in brightness. To avoid banding in an image, the steps between representable values need to be at or below the JND threshold. There is a function known as the “Barten Ramp” that describes the location of JND across a range of luminance. This is used to determine the number of bits necessary to represent images with different encodings across different ranges of luminance. It is the reason that the sRGB encoding curve is enough with 8 bits at a range of 100-200 nits, but that it isn’t enough when pushing to something like the 1000 nits of an HDR display.

Finally, perception plays another very important role in how we perceive color. In computer graphics, we generally treat white as being the color where red, green, and blue are at full intensity. In real life, the color perceived as white is tied to the environment. This is because we adapt to the present illumination environment, and it is where the concepts of cool and warm whites come from as well as what is termed the white point. (White point is the color that is presently perceived as white, often given in chromaticity coordinates or correlated color temperature.) Taking this into account can lead to producing more realistic images, as the virtual environment being portrayed typically has different lighting than the one in which the user is sitting. For those familiar with photography, this simply corresponds to the concept of white balance. The internet meme of white and gold versus black and blue dress is an excellent example of shifted perception.

Figure 4 - Well known example of color perception being tied to the viewing environment.

The wide range of visible colors and luminance levels should provide a good motivation for why advances in display are important. While it isn't necessary to apply a complete model of the visual system to produce a good image for an HDR display, it is useful to have some understanding of the concepts when setting up the environment and display pipeline.

Color Spaces

Color spaces are one of those things that most graphics programmers understand, but not in a fully rigorous manner. It simply hasn't been all that necessary, as all day-to-day work for the past couple decades has implicitly been tied to the sRGB color space. (or the highly similar rec.709) Sure, we've thought about sRGB, but mostly as this funky gamma curve that prevents banding. In reality, a color space generally consists of three things: a set of primaries, an encoding, and a white point. Color primaries for use in computer graphics are almost always red, green, and blue. However, there is still an important question of which shade of each. The encoding covers things like floating point versus normalized integer, but also items like non-linear transforms such as the sRGB function or similar gamma correction functions. Finally, a color space generally specifies an assumed white point. This allows the correction of the color balance under different viewing conditions. (as you might do by changing the color temperature setting on your monitor)

XYZ color space and derivatives

Since not all color spaces can represent all visible colors, it is helpful to use a reference space that can to describe all the others. The International Commission on Illumination (CIE from the French Commission internationale de l'eclairage) did this in 1931 by defining the XYZ color space. The color primaries X, Y, and Z are imaginary, and therefore much of the space represents colors that don't exist. The primaries are defined so that all real colors are within the positive quadrant of the space, and Y corresponds to a signal that humans interpret as luminance. The XYZ space has a linear encoding, and it is often represented with floating point values. These properties make XYZ a great space to use as a central reference when doing conversions and diagrams, but a relatively poor one for rendering and storage. Derived from XYZ is a

projected space xyY used to describe the chromaticity (hue, saturation, etc) decoupled from the luminance. The chromaticity coordinates in this space are simply the X and Y values divided by the sum of the X, Y, and Z channels. This yields the space used by the horseshoe shaped diagram shown in Figure 3.

Color Gamut and Volume

Every color space implicitly defines a gamut or volume containing the represented colors. When the color space encoding allows for negative values and essentially unlimited ranges, like floating point, the volume is effectively infinite. However, only the region where all components are >=0 is really useful for rendering operations. While we often think of the volume as an RGB color cube, it is often more useful to think of it as polyhedron in a space such as XYZ.

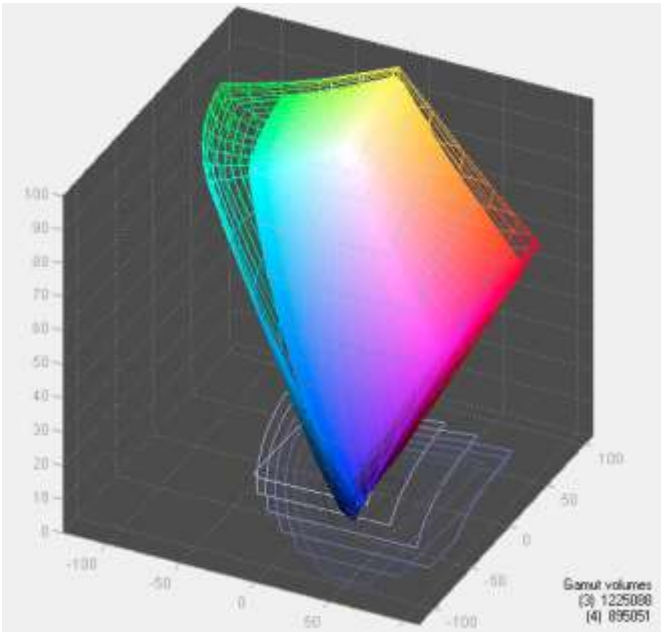


Figure 5 - Visualization of color volume supported by a particular monitor. (Image from AnandTech http://www.anandtech.com/show/3728/sceptre-x270w-1080p-review-value-27-that-delivers/3)

Visualizing color volumes in this way demonstrates how peak luminance is only available at the white point, and saturated colors offer a much lower maximum luminance. A display with 100 nit (cd/m^2) luminance will only have a pure blue with 15 nit luminance.

When considering real-time graphics, I personally find it more useful to segment the chromaticity and luminance. When I think of wider gamuts, I primarily think of more saturated hues from richer chromaticities. When I think of a higher range of luminance, I think of higher dynamic range. As a result, I will typically use that short-hand, as for practical purposes the two are very disconnected in game development and real-time graphics in general.

It is important to understand that there are many ways of measuring the color volume or gamut of chromaticities. Most of the time, this document uses the fairly common CIE 1931 xyY diagram presented previously. There has been concern that it doesn't correspond that well to the perceptual sensitivities in human vision, so there is another diagram based on the CIELUV color space that was created with the intent of being more perceptually linear[4]. It greatly reduces the portion of the visual locus dedicated to green and cyan. One often quoted metric for how well a color space represents our experiences is via Pointer's Gamut[5]. Pointer's Gamut is a collection of measured chromaticities from diffuse reflections in the real world. It is intended to cover the space of colors we see on a daily basis. It is important to remember that it only covers diffuse interactions, so representing all of Pointer's Gamut is not a guarantee that every color can be represented. Pointer's Gamut is visualized in comparison to some well-known color spaces in Figure 6.

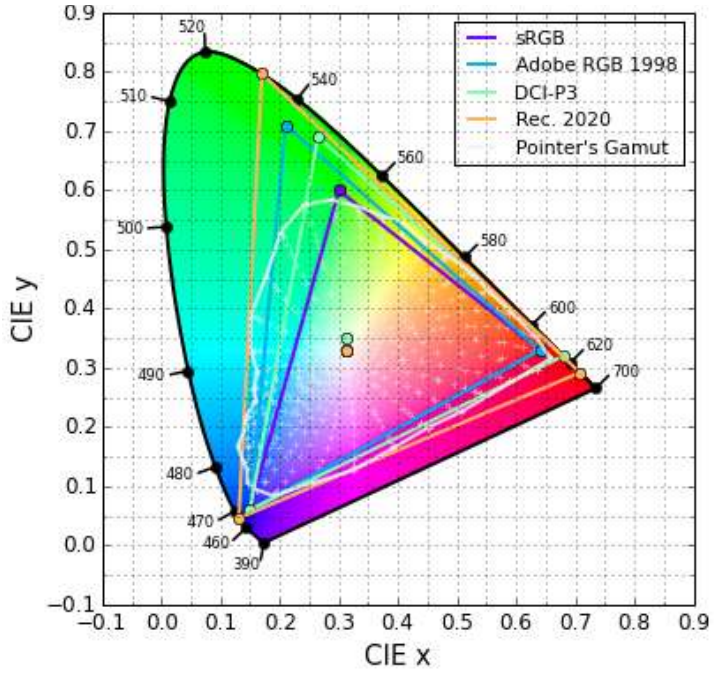


Figure 6 – Gamuts of some common color spaces along with Pointer's Gamut.

RGB Color Spaces

While all RGB color spaces have primaries that can generally be described as red, green, and blue, they differ in the exact definition of these color primaries. They also may specify different encodings (like sRGB), different white points, and different max intensities. The following sections list color spaces you may already be aware of, and a few that you should soon familiarize yourself with.

---

[4] Please see the discussion in the section on CIELUV for some details on how there is still disagreement over what the most perceptually uniform space is.

[5] Pointer's Gamut comes from "The Gamut of Real Surface Colors" by M. R. Pointer

sRGB

This is the color space anyone doing real-time rendering, such as game development, is likely intimately familiar with even if they aren't aware of it. sRGB is the space our content is authored and displayed in today, because it is the standard that monitors are generally built toward. As can be seen in the diagram below, it occupies a relatively small portion of the visual locus. This limitation is governed by the purity of the primaries. Since sRGB doesn't allow negative values to be represented, nothing can be more saturated than the color primaries, restricting the colors that can be represented.

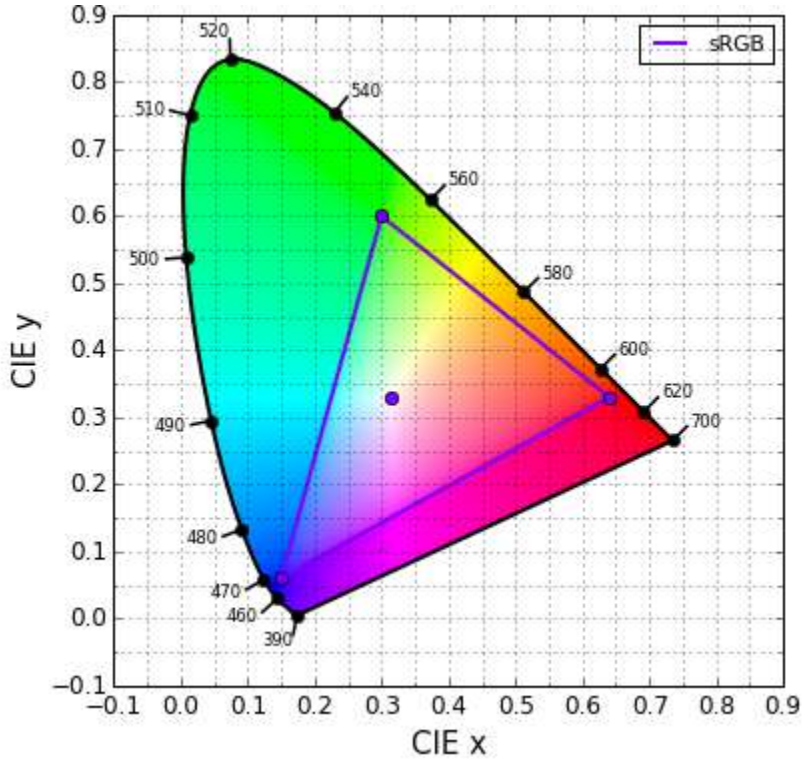


Figure 7 - Chromaticity diagram of sRGB color space.

sRGB covers only 33% of chromaticities in the visual locus, and only 69% of Pointer's gamut. In addition, the standard limits maximum luminance to only 80 nits. This greatly restricts the portion of the real world that can be faithfully represented in this color space.

sRGB's white point is a standardized one known as D65. (As with many things color it was a standard set by CIE) D65 corresponds to the white perceived at noon under an overcast sky. This also equates to the correlated color temperature of roughly 6500K. (Color emitted by blackbody radiation at 6500K[6])

[6] Actually not quite 6500K, there was a small alteration to the constants in Plank's Law after things were defined, such that it is actually now 6504K.

As a historical note, sRGB was developed by Microsoft and HP back in the mid 1990's to standardize the display and printing of color for PCs. It has evolved into a ubiquitous standard on the internet.

In practice, many monitors that support sRGB as their standard actually do exceed the standard color space. Typically, this is in the realm of luminance, with 200 or even 300 nits being common. In practice, the higher luminance gets used to adjust for use in brighter environments, with most users having their monitor set nowhere near maximum brightness. The monitors still only deliver at best 1000:1 static contrast ratios as backlight leaking and reflections of ambient light lead to black pixels really being brighter than 0.1 nit.

Rec.709

As mentioned previously, Rec.709 is the specification for HDTV color. For the purposes of game development, it is very similar to sRGB. It shares the same color primaries and thus it's gamut of chromaticities is identical. It does have a somewhat different encoding, partially due to the expectation of a different viewing environment. TV's are expected to be viewed in a dimmer environment than computer monitors, altering the perception of contrast. Because of this, it specifies a different gamma function.

Adobe RGB 1998

Another RGB color space many game developers may be familiar with is Adobe RGB. (It is important to note that I am referring to the one standardized in 1998 as there is a newer wide-gamut Adobe RGB also) It is something that you may have encountered via a high-end monitor, photography, or Photoshop. This color space has a substantially more saturated green, allowing the representation of many additional printable colors in the green and cyan regions. As this space has a somewhat murky history, you can sometimes find varying definitions. (D50 or D65 white point) In general, it matches sRGB's red and blue primaries, but has a richer green.

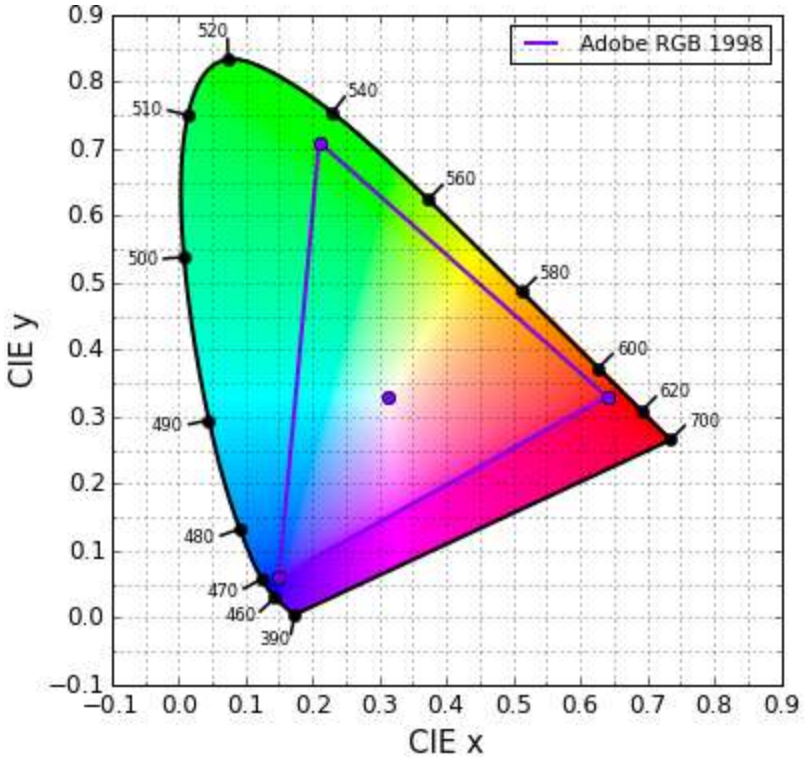


Figure 8 – Chromaticity diagram of Adobe RGB 1998

The wider gamut supported by Adobe RGB allows for 45% coverage of the visual locus and 86% coverage of Pointer's gamut. Interestingly, Adobe RGB appears to be a bit of a historical accident[7] resulting in some conflicting definitions of the space.

DCI-p3

DCI-p3 is a color space standardized for digital film, with DCI standing for Digital Cinema Initiative. It shares the same blue as sRGB and Adobe RGB, but it has a much richer (monochromatic even) red. The green is also richer than sRGB, and while it is more saturated than the one found in Adobe RGB, it is yellower. It is also notable that the white point in this color space isn't the D65 seen in many others, but instead D60. (Often referred to as a "greenish" white) The result is a gamut that covers roughly 45% of visible colors and 87% of Pointer's gamut.

[7] http://www.realtimerendering.com/blog/2011-color-and-imaging-conference-part-vi-special-session/

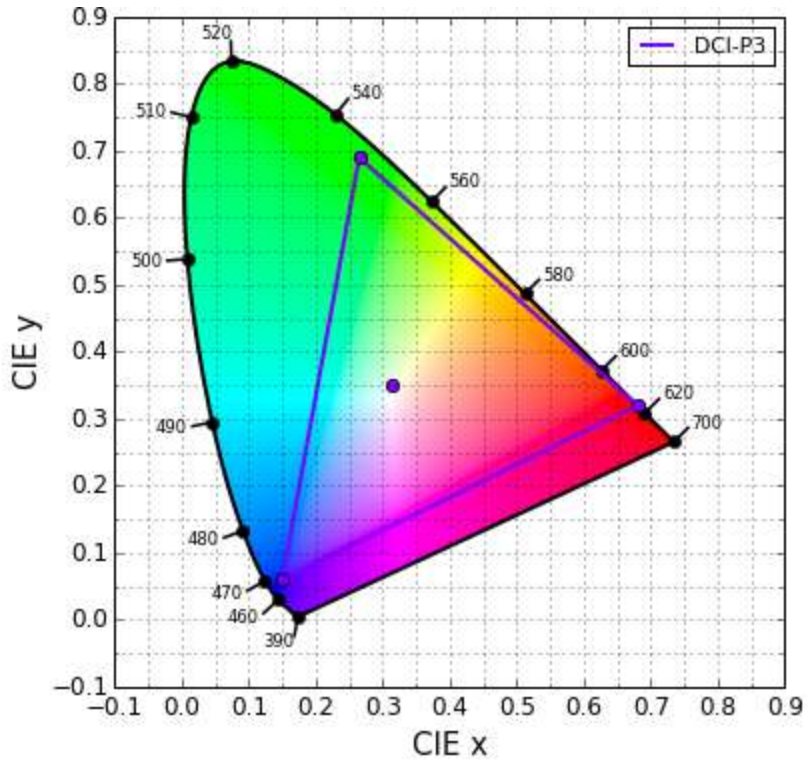


Figure 9 – Chromaticity diagram of DCI-P3 color space.

DCI-p3 is of particular interest in this document, due to challenges of implementability. While wide-gamut color specifications for UHD specify an even wider color gamut (Rec.2020), DCI-p3 is closer to what we as developers can expect as the supported gamut in first-generation displays. As we'll discuss later, targeting realizable color gamuts can help ensure that your content displays as you desire it to.

BT.2020 / Rec.2020

BT.2020 is the color space defined to be used for UHD displays. As can be seen in the diagram below it is based on completely monochromatic red, green, and blue primaries. This produces a very large color gamut which covers 63% of visible chromaticities and 99% of Pointer's gamut.

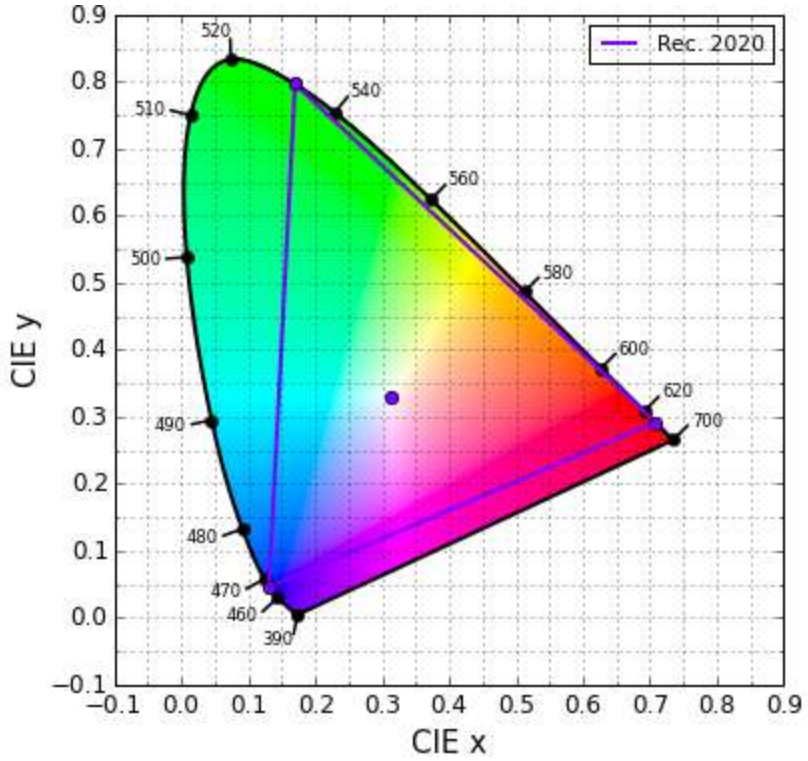


Figure 10 – Chromaticity diagram for BT. 2020 color space.

scRGB

scRGB is a color space that was introduced by Microsoft with Windows Vista. It offers an expanded gamut while maintaining compatibility with sRGB primaries. The color space uses a linear encoding with fixed point that permits a range of [-0.5,7.5]. The key bit is that all colors in the [0,1] range match sRGB colors exactly.

While scRGB is technically only defined for fixed point encodings, one can easily imagine that the same concept extends to floating point encodings as well. Naturally, the range restrictions are lifted under such a scenario allowing the representation of a very wide gamut and very high dynamic range. As with true scRGB, an fp16 floating point representation has been available since Windows Vista. This color space is extremely useful, because it is defined such that it can be easily composited with other desktop elements that are authored in straight sRGB.

As floating point isn't officially listed in the scRGB standard, it isn't technically scRGB. However, it appears likely that there will be an amendment, extension, etc to cover this in the future, so for the purposes of this document we'll just use the scRGB term for floating point data encoded in this manner for simplicity.

Converting Between Spaces

If you weren't already aware, converting between these different RGB color spaces is predominantly done via a simple linear transform. (once colors are in a linear representation) The standard way of describing the transforms is via the XYZ space as an intermediary. (sRGB -> XYZ -> BT. 2020) Naturally, the 3x3 matrices representing transforms to and from XYZ space get concatenated in any optimized scenario.

The reason you want to transform from space to space rather than simply mapping (1,0,0) in sRGB to (1,0,0) in BT. 2020 is that your colors will shift, and artists may wonder what has happened to their work. ("Why does the character look like he has a sunburn?") The gamut mapping section later in this paper offers some perspective on pros and cons of remapping rather than transforming colors.

In addition to simply changing the color primaries, it may be necessary to adjust the white point of the image as well. The standard way of doing this is through a Chromatic Adaptation Transform (CAT). The common CAT solutions are the Bradford or Von Kries transforms. These convert the colors to an LMS color space, then scale by destination white / source white (also in LMS space), and finally convert back from LMS space. As the transform to LMS space in both Bradford and Von Kries is simply a 3x3 matrix multiply, the entire CAT can be collapsed into a single 3x3 multiply, just like the transform of primaries.

The final complete color transform generally looks like the pseudocode below:

```
RGB = ToLinear( inRGB) // remove non-linear encodings, like
gamma
XYZ = mult( RGB_2_XYZ_mat, RGB)
LMS = mult( XYZ_2_LMS_mat, XYZ)
LMS = LMS * ( outWhite_LMS / inWhite_LMS)
XYZ = mult( LMS_2_XYZ_mat, LMS)
RGB = mult( XYZ_2_RGB_mat, XYZ)
outRGB = FromLinear( RGB)
```

It should be noted that the only steps not representable as 3x3 matrix multiplies are the decoding/encoding from/to non-linear representations. As a result, the optimized solution will typically be collapse the transformations into a single matrix multiply.

Other Color Spaces

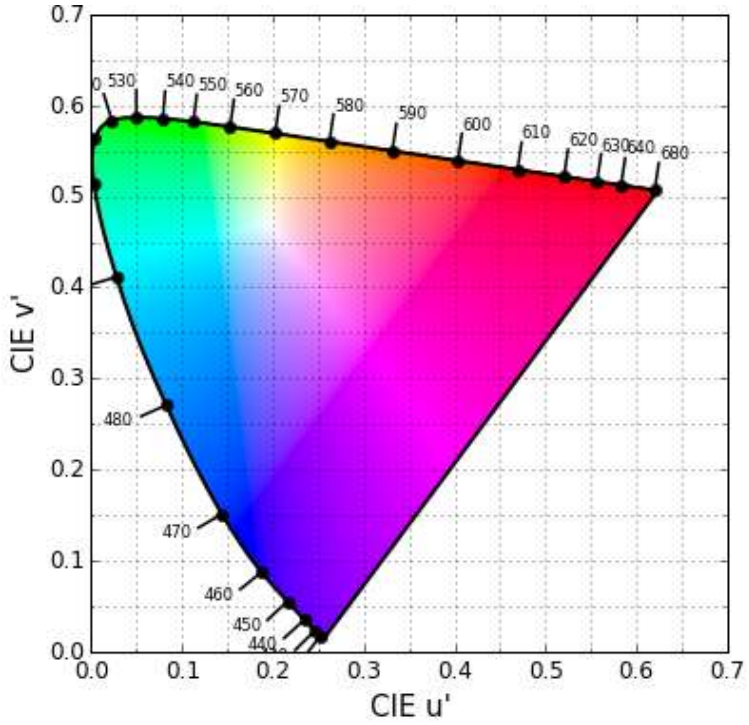
While RGB color spaces are great for rendering and generally representing image data, other color spaces are likely familiar for other purposes as well.

CIELUV and CIELAB

To deal with a perceived shortcoming of the xyY color space used for diagramming chromaticities, CIE standardized a pair of additional color spaces in the 1970's. They both attempt to map colors more linearly with respect to human perception, so that a

measure of distance in the space more closely matches the difference perceived by a viewer. The fact that two different ones exist is largely a historical accident of different industries addressing the same issue. Some controversy exists about whether CIELUV is truly a better perceptual space as pointed out by a paper by Goldstien in 2012[8] and a paper by Masaoka in 2015[9] both suggest that while not ideal the original xy diagrams are better particularly when wide gamuts are considered.

These spaces are noted here primarily to familiarize you with the alternate form of the chromaticity diagram, and that they may be useful when comparing colors or performing a transform that tries to minimize color shift. It should be noted that these spaces do not show hue linearity. As a result, interpolating colors in a straight line toward white will result in a perceived shift in hues.



HSV / HSL

HSV (Hue, Saturation, and Value) or HSL (Hue, Saturation, and Lightness) are color spaces that artists might use in the production of some assets. They can be highly intuitive to use. It is expected that if you need to use something like HSV with HDR images that you will need to adjust/expand the representable range of the V parameter.

IPT

[8] http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=1385765
[9] http://www.ncbi.nlm.nih.gov/pubmed/25837119

IPT is yet another color space invented to improve the description of differences in color. Its primary claim to fame is that it is designed to be linear with respect to perceived hues. This means that interpolating a set color toward gray should give the impression of a less saturated version of the same hue.

IPT is primarily mentioned in the paper because it can provide a convenient logical space for defining or performing some number of color grading or gamut mapping operations.

Properties of colors

Many terms are used in the everyday description of colors. In the literature related to color science, they often have very specific meanings. It is useful to have a quick definition of these terms, as this document tries to follow the meanings used in the literature. It is important to note that these terms are referring to the perceived appearance of a color rather than an absolute measurement like RGB, LMS, and XYZ. As a result, these are all relative to the viewing conditions.

Hue – Closeness of a stimulus to the stimuli that are described as red, green, yellow, and blue.

Lightness – Brightness of a stimulus compared to the brightness of a stimulus that appears white.

Brightness – Perceived quantity of light. (non-linear with respect to luminous energy)

Chroma – Colorfulness of a stimulus relative to the brightness of the source.

Colorfulness – Perceived quantity of hue in a stimulus, distance from gray

Saturation – Colorfulness of a stimulus relative to its brightness.

These properties are often used in color appearance models (CAMs) to describe color stimuli and attempt to replicate the same perceived color under different environments. The definitions used here pulled from the work of Fairchild in the development of the CIECAM02[10] (a particular standardized CAM).

Scene Referred vs Output Referred

One important concept that a developer venturing into the realm of creating images for HDR displays needs to internalize is that of output referred versus scene referred. Scene referred simply means that the image is representing light as it was captured by the virtual camera. Generally, this would represent the HDR buffer rendered in a game prior to tone mapping. In that case, the data typically has a linear encoding. Scene

[10] http://rit-mcsl.org/fairchild/PDFs/AppearanceLec.pdf

referred images intended for saving to disk (such as captured by a camera) may have some sort of non-linear encoding to save bits. Output referred images are images representing the luminance values intended to be output by a display. In a game, these will be anything after tone mapping. It is important to realize that many 2D graphics, like UIs are actually created as output referred as they are crafted in a space specifically for the display. The importance of the distinction is that different processing needs to occur to generate an output referred image for an SDR display versus an HDR display. Finally, the concept of scene referred and output referred impacts operations like blending. Just as with sRGB blending the operation should convert to linear first, but it also needs to ensure both inputs have the same reference frame (scene or output),

OETF and EOTF

Optical Electric Transfer Function (OETF) or the inverse Electrical Optical Transfer Function (EOTF) are generalizations of concepts many graphics programmers already know fairly well today. You probably simply call it gamma, because that is roughly what it is in an SDR world. The non-linear conversion of bits to luminance values by the display is an EOTF. Gamma functions technically only refer to power functions, so technically, sRGB isn't really a 'gamma' function, since it has a linear portion. These distinctions become important in HDR, because it turns out the gamma-like functions are typically inadequate for encoding the higher dynamic range efficiently. (They need extra bits to stay below the JND curve) As a result, HDR displays often use an EOTF known as the Perceptual Quantizer (PQ) that does a more effective job of representing the space. There are other possibilities as well, with one being "Hybrid Log-Gamma". An important corollary of this is that in the past, the gamma function might be tweaked as a way of adjusting the contrast on top of what a straight reproduction might be. Such a viewing conditions adaptation now needs to be done explicitly further up the pipeline prior to the encoding for EOTF.

Where UHD Fits with Gaming

It should be clear to most game developers that richer colors and brighter highlights can offer the promise of a more engaging image. Historically, we'd just take the image we're already generating in that [0,1] RGB space and slap it onto a new monitor with extended capabilities. As I hope you understand by now, this is not in general a good solution for the capabilities UHD brings. First, moving the brightness linearly to an HDR monitor will generally lead to an uncomfortable experience. That white UI dialog is going to be like shining a flashlight at your user. Second, we have moved forward a lot on image quality as an industry. Expectations of quality and reproducibility are much higher. No one wants the (virtual) human protagonist spending the whole game with a greenish tinge as though he is about to vomit. Finally, the delivered impact of HDR in particular has some physical constraints to live within. Making a display brighter takes more power. As a result, attempting to constantly drive much or all of a display to full brightness may result in overall brightness being restricted by the display. Restricting the extreme brightness to the highlights that are truly supposed to be bright will ensure a better use of the available range.

What your game can gain

The first thing to consider when asking “What will I gain?” is how you are compromising today. If you are like many developers, you are already generating fairly high-quality HDR data, likely through a physically-based rendering pipeline. This wide range of data then needs to be compressed via a tone mapper to be displayed on monitors that can only handle 2 orders of magnitude in brightness. (3 if all your customers have really good displays) The compression here limits how good of an experience you can offer to your users.

The restricted range limits highlights to looking much flatter than they would appear in real life. Extra dynamic range is required to allow good discrimination between diffuse white and true highlights. Additionally, as noted previously, displays can only produce their maximum brightness at their white point. For highlights that are distinctly not white, the choices on an SDR display are compromising on the saturation or compromising further on the brightness. A display with an HDR range allows for highlights that are substantially brighter than diffuse white, and it allows fully saturated colors at much higher luminance levels.

Further, the need for compression restricts the brightness of mid-tones, as the brightest colors get used for highlights. Removing the compression allows for overall brighter mid-tones. Brighter images have important perceptual effects that can make the image appear better in many ways beyond naive expectations. First, the Hunt Effect states that images will appear more colorful as the brightness increases[11]. This is simulated in the image below. The colors all have identical chromaticities, but the luminance level is increased. The brighter images are perceived as substantially more saturated. A result of this effect is that a HDR image with sRGB chromaticities can appear more richly colored than an SDR image with DCI-P3 primaries. Additionally, the Stevens Effect states that a brighter image will have more contrast. In addition to the image below, you can easily test this effect by looking at the test of a book under bright sun versus looking at that same book under the illumination of a dim room.

[11] The presentation arguing for the creation of the CIECAM2002 color appearance model discusses the Hunt and Stevens effects, as well as many other aspects of color perception: http://rit-mcsl.org/fairchild/PDFs/AppearanceLec.pdf

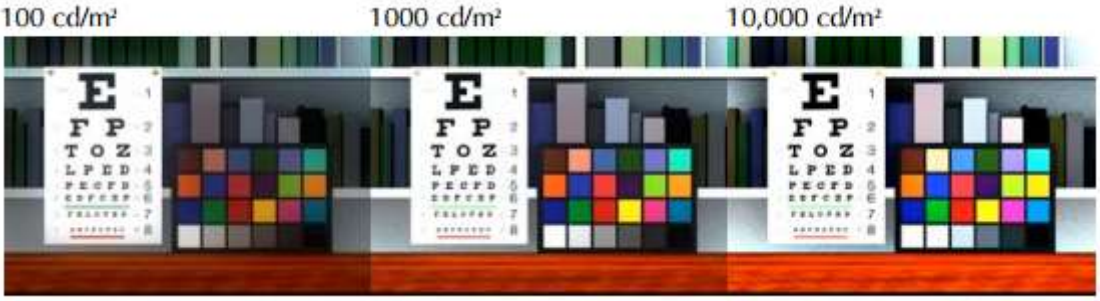


Figure 11 - Hunt Effect, theperceived intensity of colors increase at higher luminance, even though the hues and saturation are the same.

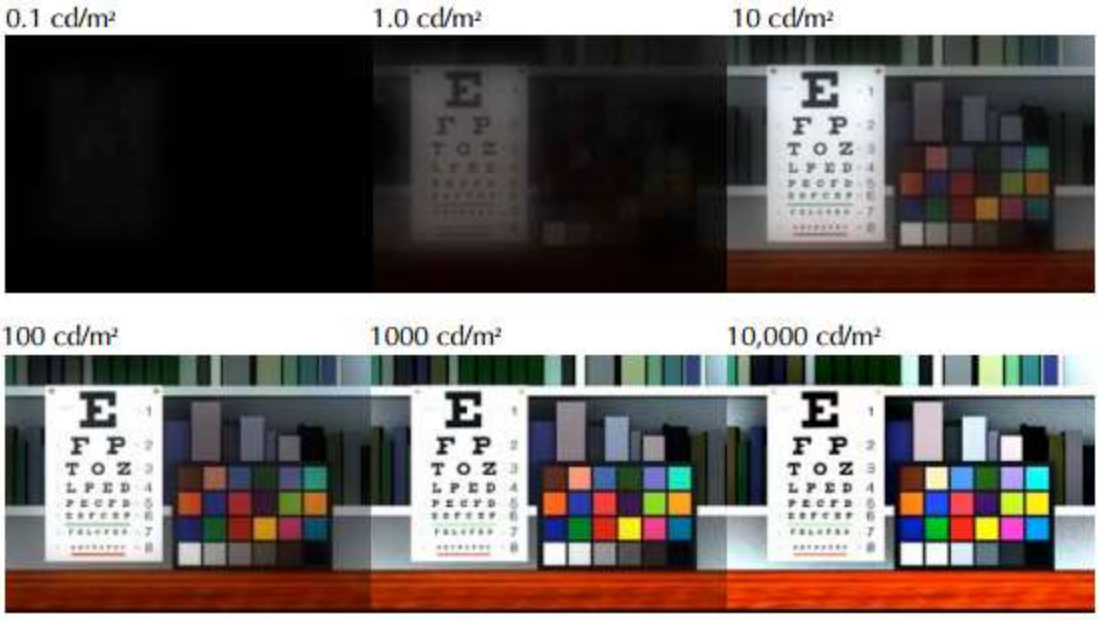


Figure 12 - Stevens Effect, perception of contrast increases at greater luminance, and the image looks sharper.

Finally on the dynamic range front, the need in SDR to devote as much range to the important mid-tones as possible means that shadows will get crushed toward black. The HDR made available in UHD displays allows for maintaining details in the shadowed portions of the image. Further, UHD displays offer truer blacks via local dimming. This means that in addition to not needing to steal part of the dynamic range from shadows to better represent mid-tones, you also gain from deeper black.

On the chromaticity side, we've already discussed how wider color gamuts allow the display of colors you'd otherwise not be able to create, potentially leading to richer, more vibrant, more realistic images. As mentioned in the next section, if replacing your content with higher color gamut content is impractical in the short term, there are still opportunities to take great advantage of UHD displays, even within the sRGB color space.

A practical path to utilizing UHD in the near term

As with all engineering, game development is an exercise in compromise. The goal in supporting UHD class displays isn't to simply generate content targeted directly to the native BT. 2020 color space with a dynamic range that tops out at 1000+ nits. The goal is to create a great game that looks fantastic on the majority of displays that your customers have (sRGB-based) and to enhance your pipeline to also produce images that make good use of these new, better displays. Given the complexity and cost of developing content for a game, it is generally impractical to expect UHD and sRGB assets, or convert over the art flow to generate most art in these new wider color spaces.

In support of these constraints, NVIDIA has developed what we feel is a good starting point for the practical adoption of UHD output support for games. It certainly isn't the only possible solution, but we feel it offers a good compromise on the challenges faced. Here are the core principles:

- Author content with sRGB primaries as today
- Render high quality HDR data using physically-based shading
  - Analyze content for dynamic range produced
  - Watch for hacks with luminance levels
- Perform all rendering operations with respect to the sRGB primaries you use today
- Postprocess in the scene referred space
  - Motion blur, DoF, etc
- Apply any color grading to the scene-referred image
- Tone map with filmic ACES[12]-derived tonemapper
- Keep backbuffer in fp16 scRGB
- Composite sRGB referenced UI as normal

This pipeline will allow you to keep content authoring as you likely have it today, with the exception that you will need to pay additional attention to certain aspects of the quality of your lighting and effects. Additionally, if you rely heavily on color grading today you may have some additional work as scene referred color grading can offer better portability than the output referred color grading commonly used today.

[12] Academy Color Encoding Standard, explained in more depth in a future section.

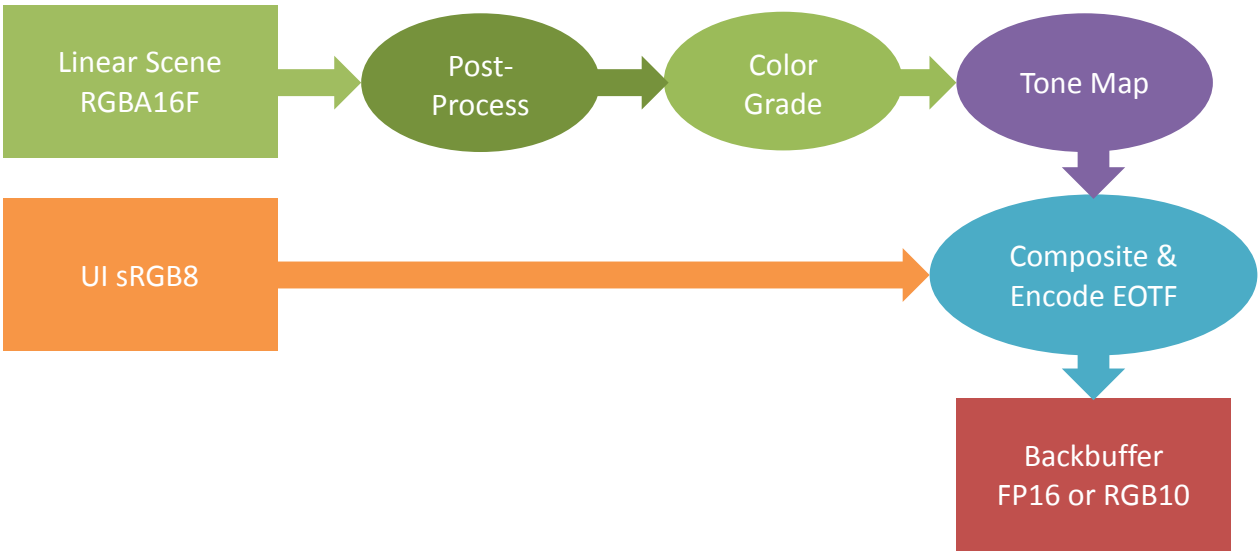


Figure 13 - Logocal game pipeline for HDR output

sRGB Primaries

Keeping your content authoring in sRGB is probably the biggest item to simplify life for games in development today. Artists can continue to use the workflow they already have and, studios don't need to re-equip the entire art staff with new monitors. Rendering using the sRGB primaries is directly related to this. sRGB primaries are what you render with today. If you were to take your sRGB content you create today and convert it into colors described by the BT. 2020 primaries before performing lighting, you'd see much different results. This happens because material interactions are relative to the color space in which they are defined. The modulation operator we all use to compute the interaction between a material and a light source will produce different results if you change the primaries. As a game developer, you don't want your artists authoring content to see an object looking brown on their monitor, but the same object being green or purple when you run the game on a UHD class display.

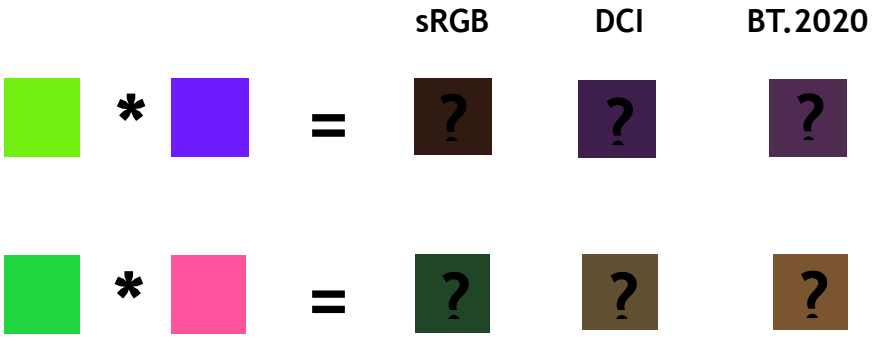


Figure 14 - Modulating colors has different results when the math is done in different spaces.

The downside for UHD displays of performing all the asset authoring and rendering with sRGB primaries is that you may not be able to use the width of the gamut offered by

UHD displays. The positive side of this is that you won't have the problems of generating hues that your SDR customers cannot see and the challenges of mapping those colors sensibly. It is important to remember that even with the primary limitation you will be generating richer colors for UHD displays, because highlights that would normally need to desaturate to maintain brightness can maintain their saturation with the higher dynamic range. Additionally, the Hunt Effect applies to the mid-tones that can now be represented as notably brighter. Even though the chromaticities are the same, they may appear just as rich or richer than wide gamut colors displayed at SDR levels.

While this is a great practical step to getting UHD content working, the general expectation is that it is an evolutionary step. As displays with wider gamut become more widely deployed, we expect shifts to rendering and/or authoring primaries such as DCI. These will allow the production of the richer colors we just can't render today.

Physically-Based Rendering

Physically-based rendering is the general trend in game development today, because it helps in the generation of consistent, high-quality images. These properties are also useful in generating better HDR outputs. Using physically-based techniques means that you are generating plausible levels of illumination, and that the scene-referred data has realistic relative magnitudes. This should allow you to generate the strong highlights that will take advantage of HDR displays.

While being physically-based is a great start, it isn't a hard requirement, and it isn't enough to guarantee that you won't have artifacts. The most common challenge we've seen is when lighting values cheat. The common case is where proxy geometry exists in the scene representing a light source. The emissive value of the geometry can often be wildly different from the values used for the light injecting illumination into the scene. When you are just looking at this on SDR displays, the problem is often hidden. Once you reach a level of "bright enough" the light source is in the highlight region of the tone mapper, and it looks as bright as anything else in the scene. However, once you go to an HDR display, you now have a much greater capacity to differentiate highlights. It is now easy to have the specular reflection be dramatically brighter than the geometry representing the light source. This can be quite obvious and make the scene look less realistic. (The light source now looks like a white globe instead of an illuminant) Another example of this effect rises from skyboxes. In the false-color image below, the light cast by the sun is much brighter than the value assigned to the skybox. As a result, the sun is blinding in the reflection from the floor, but the sky box looks dim in comparison. Similar issues can apply to particle effects as well.

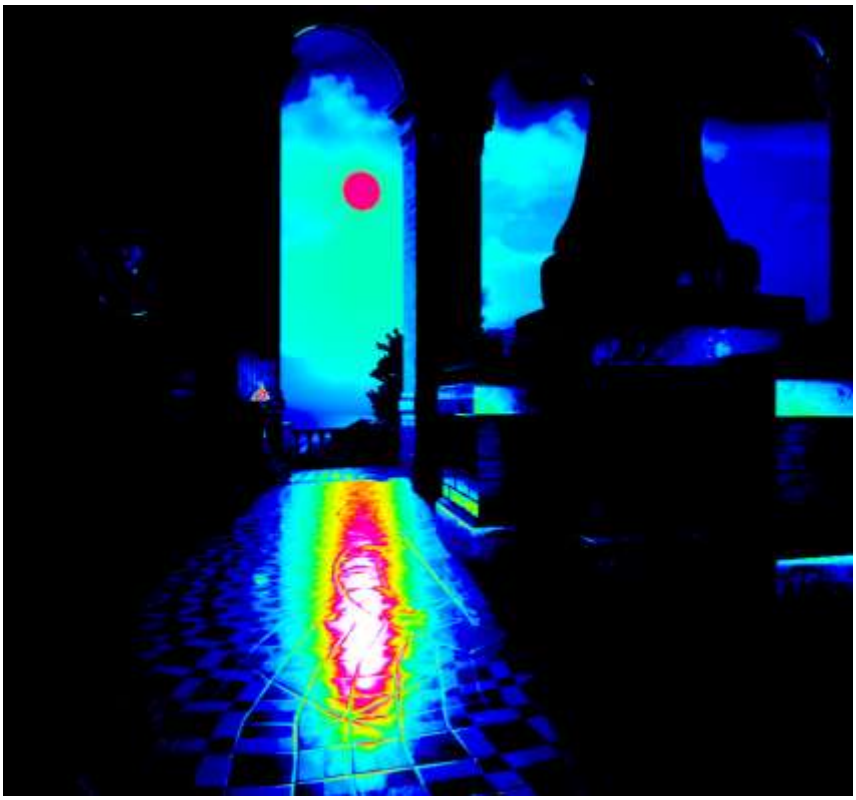


Figure 15- Specular highlight is two or more times more luminous than the sun that is supposed to be source of the light. (Visualization of scene referred luminance levels in Sun Temple sample from Unreal Engine 4)

Scenes that will utilize HDR well

Naturally, high dynamic range outputs really only look their best when the content that they are displaying is truly high dynamic range. A scene can look fantastic with it truly having a very large range, especially if you are doing physically based rendering correctly. It is important to remember that range isn't brightness or darkness, it is the distance between them. You can have bright or dim images that don't show a ton of range. Outdoor scenes are bright, but when they lack deep shadows or the sun or strong specular highlights within the view, they may not have an overly large dynamic range. Since HDR isn't about just turning up brightness, the exposure will still be bringing the range to a good viewing level. Similarly, a dimly lit indoor scene where everything is diffuse and no lights are directly visible will only show so much dynamic range. The best scenes for looking at what HDR provides will be ones that mix dim and bright components. (Looking outside from a dim room or cave, bright lights directly in the view) All this isn't to say that you won't see gains in most scenes, but that for testing or demonstration purposes you may want to look for these sorts of environments as they are the ones that you can expect to standout from across the room.

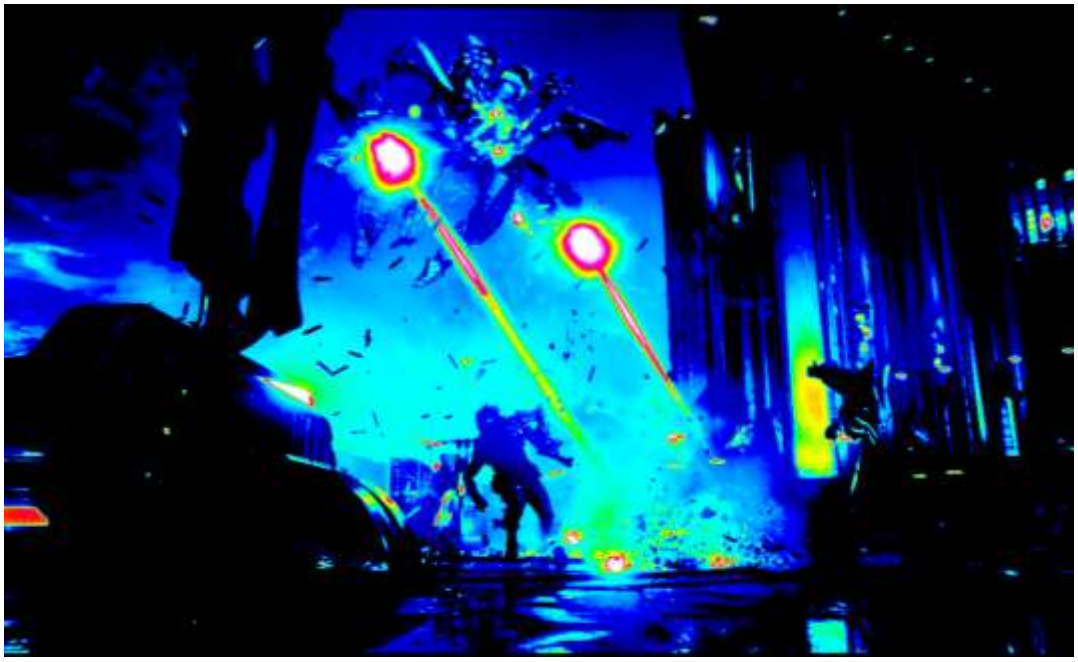


Figure 16 - Visualization of luminance of a scene from Infiltrator demo of Unreal Engine 4

These images visualizing the scene referred luminance levels of demos from Unreal Engine 4 show how fantastic rendering may or may not push the envelope on dynamic range. For all of these visualizations, each color step (blue, cyan, green, yellow, red, magenta, white, etc) represents a change of one stop (doubling) in luminance. In Figure 16, the Infiltrator demo is clearly showing a very large dynamic range, with deep shadows and very bright highlights. In Figure 17, the Kite demo is showing much brighter colors but a smaller range. As most people would say that Kite offers a better image, it is clear that dynamic range isn't directly related to the beauty of the image. What is notable is that both scenes actually do notably improve when displayed on an HDR display, but just for different reasons. Infiltrator gets highlights that pop and make you feel as though you are really looking at a light source. Kite gains in that its mid-tones appear brighter and as a result richer through the Hunt Effect. (Infiltrator has a lot more of a gray palette, so it doesn't improve as much here)

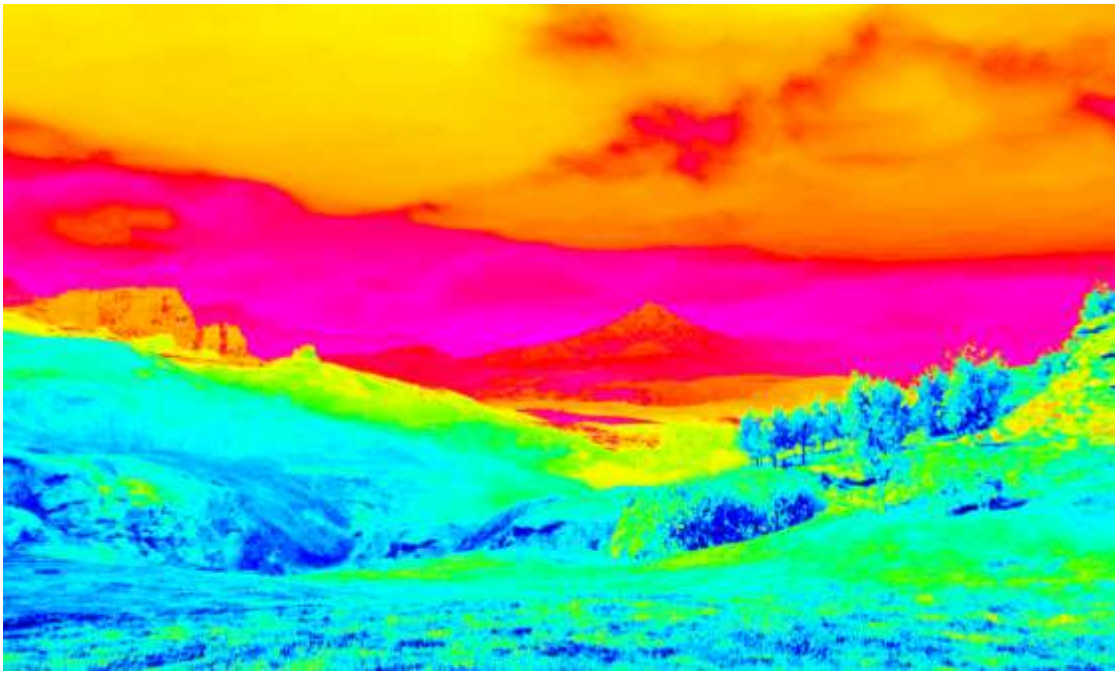


Figure 17 - Visualization of luminance in Kite demo for Unreal Engine 4

Validating your scene

As mentioned above, you need to pay attention to where and when you might be cheating to produce the best HDR output you can. A few good tools can go a long way here. First, dumping native HDR image data for offline analysis is very helpful. Even if you don't integrate a ton of tools into your engine directly, dumping HDR frames to run through an external tool like our HDR image viewer sample can offer lots of insight. Second, visualizing the dynamic range of the scene can be extremely useful to identify faults in luminance levels even while using SDR monitors. A simple solution is replacing the tone mapper with a shader the applies a false color ramp to the function log2(luminance / 0.18). This gives you steps in increments of stops away from middle gray with 0.0 being middle gray. A color step for every stop or two works great for identifying when highlights are brighter than the lights casting them. The colorized image also shows you what sort of range exists in the scene. Finally, you may find it useful to add a proper histogram tool so you can visualize the distribution of luminance values in the scene. The false color technique previously mentioned does provide this on an intuitive level, but the hard data can be enlightening as well. (Especially when the peak values in a scene occupy a relatively small spot)

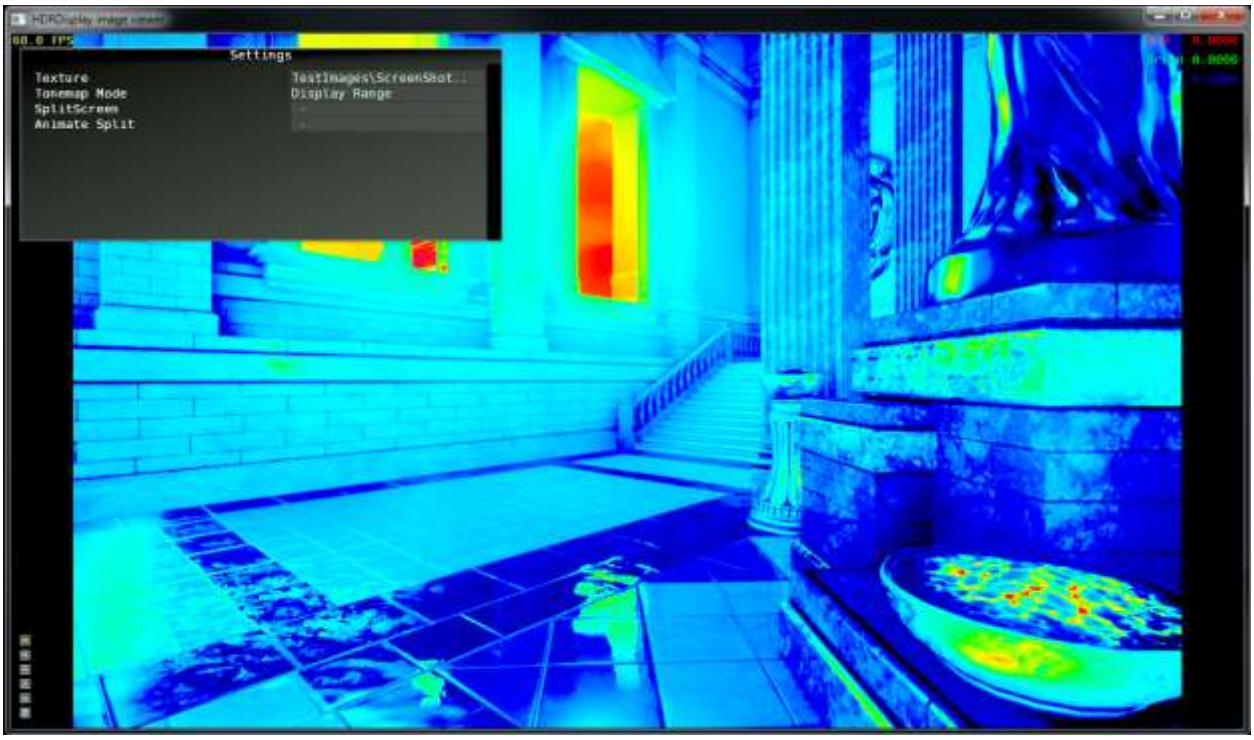


Figure 18 - Log luminance visualization of UE4 Sun Temple demo. Cyan is middle gray, blue is two stops darker, green is two stops brighter, etc.

Scene-Referred Post Processing

The general concept here is extremely simple, and most games already appear to function in this way; however, it bears noting. Our results so far show that performing post processing operations, like motion blur, depth of field, etc, are best done prior to tone mapping the scene referred image to an output referred image. Performing the operations in the scene referred space means that the operation is consistent no matter what display we're targeting. Performing these operations after tone mapping would mean that the result of the operation would very much depend on what display was being targeted. It is much the same reason you linearize an sRGB texture before filtering it.

An argument can be made that bloom effects may want to change somewhat when moving to HDR display outputs. It would appear that this may be highly dependent on what the bloom is attempting to simulate. We've actually had quite good results leaving bloom as is, and applying it prior to tone mapping. This case seems to argue for the fact that it is modeling a spatial bleeding of overly bright data on the image plane of the virtual camera. There is another interpretation where bloom is injecting a glow around objects too bright to represent in an effort to make them look bright. In that latter case, turning back the bloom may produce the more pleasing effect. As stated, we recommend leaving bloom as is for an initial pass, and then later reevaluating it if there appears to be an artifact.

ACES-derived Tone Mapper

The tone mapping operator in our recommended pipeline had to meet several constraints. Obviously, it needed to offer scalability for different output luminance ranges. It needed to work well with the looks game developers are pursuing, and it needed to have a fast implementation. We've found that the pipeline defined in the Academy Color Encoding System[13] (ACES) as created by the Academy of Motion Pictures is a good fit. As a bonus, it is a standard that appears to have traction in a related industry.

Tone mapping in ACES can (unsurprisingly) be described as 'filmic'. This means that it applies a sigmoid-style curve in a logarithmic space to produce the tone mapping. Further, it performs the operations mostly independently on the color channels rather than operating strictly on luminance like many other operators. This characteristic makes the operator naturally desaturate when it reaches the limits of the present adaptation level. Highlights gently roll toward white, and shadows gently roll toward black. Filmic operators have gotten a lot of traction with game developers over the past several years, because they offer a 'cinematic' look. Also, as they roughly model the behavior of film stock, they are taking over a century of refinement in how to capture and present an image that humans find pleasing or realistic.

The full pipeline defined by ACES is more than what a game developer typically needs to concern themselves with as it includes items related to cameras and archival storage. What we're most interested in here is the imaging pipeline that specifies how to convert a scene-referred image for display across a variety of reference devices. (from 48 nit projectors to 4000 nit HDR displays in the present reference implementation)

For us, ACES starts by taking a scene referred image and converting it into the wide-gamut working space used by ACES. At this point in the process, settings such as exposure are applied to get middle-gray to the expected 0.18 level. (Applying exposure doesn't change an image from being scene referred, since exposure is just a linear scale.) Prior to the tone mapping portion of the algorithm, ACES specifies that one or more "Look Modification Transforms" (LMT) may be applied to produce the intended look for the scene. (Please see the color grading section for more) Next, ACES applies what it refers to as the "Reference Rendering Transform" (RRT). The RRT contains a sigmoid-style curve in a very large range. It can be thought of as a base film operator that is independent of the intended display. Next, ACES applies what is called an "Output Device Transform" (ODT). The ODT can be thought of as containing the output device specific portion of tone mapping. The ODT will contain a stronger sigmoid-style mapping to bring the data into the limited range of the intended output. It also handles conversion into the output color space, and any encoding necessary. The powerful concept here is the segmentation of operations, particularly device independent versus device dependent. The goal of the system is to enable adapting to many different display environments.

[13] http://www.oscars.org/science-technology/sci-tech-projects/aces

To ease the application of the ACES-inspired system to games, we have created an ACES pipeline that is parameterized to be able to implement a wide range of the reference ACES ODTs as well as additional tweaks to handle cases beyond the reference ODTs.

Scene-Referred Color Grading

This section is out of place chronologically with the recommended pipeline, simply because our recommendation here is based heavily around the ACES work flow described in the last section. As described in the last section, LMTs are the prescribed way to apply a specific look to a scene within the ACES pipeline. As opposed to the color grading game developers have frequently used in recent years, it is defined to happen against scene referred data rather than against output referred data.

Today many game developers use a very simple but powerful pipeline to handle color grading. They have a small (often 16^3) 3D LUT, with is indexed by the color after tone mapping. Artists are often given a default identity LUT overlaid on an image from the game, and then given free rein to modify it with global operations in an editor like Photoshop. After many layers of color balancing, curve modification, and hue shifting, the transformed LUT is extracted out of the image and used in game. In the worst case, all this is done without properly accounting for the non-linear encoding of sRGB.

While the process described above is extremely powerful for an artist, it poses some challenges with a flow that needs to contend with HDR. The primary issue is that the grading is all being done relative to output referred colors. This means that the grading doesn't map well to the larger range. Obviously, it is possible to just author another color grade for a different output space. However, now a developer needs to worry about keeping things in sync, and possibly worry about keeping multiple levels of grading (500 nits for OLED HDR, 1000 nits LCD HDR, 2000 nit HDR, etc) Assuming that the tone map is invertible, it would be possible to map the changes back from the SDR space to the HDR space. Unfortunately, tone mappers are never quite fully invertible once you get to the final image. (quantization and saturation, particularly in the highlight ranges) So, mapping that operator backwards will lead to some degree of artifacts or limitations.

Additionally, grading in the output reference space can result in the artist being driven to 'fix' tone mapper short comings. A common potential issue is with luminance only tone mappers. Because they are restricted to luminance only, they maintain saturation at all levels of luminance. This means that a heavily saturated highlight will actually appear dimmer due to the natural restrictions of a display. However, an artist may now be tempted to throw some desaturation into those high ranges to make them look brighter. Now, you've desaturated colors that you could represent well on an HDR display in an unnatural manner.

To solve these sorts of issues, ACES specifies using a Look Modification Transform for most of what we'd term color grading in games today. The LMT operates on the full

range data. While it doesn't have to be, an LMT will typically be a 3D LUT just like we're used to. To compensate for the large range, the LUT needs a "shaper" function to more efficiently cover the space. Typically, a log function is a good enough solution for this. With log compression, a 32^3 LUT can cover a large range in the scene referred space with reasonably good precision.

For simplicity as you start out, a developer may wish to consider integrating a fairly restricted range of mathematic operators applied directly to the data. Gain, bias, contrast enhancement, and saturation may be enough for a lot of purposes. Further transforms, like processing in color opponent spaces like IPT may be useful to adjust balance between hues. We have implemented several of these in the HDR image viewer sample as an example of simple things that can be done. Additionally, these operations can all ultimately be baked out into a simple LUT as described above for final deployment.
Obviously, the solution proposed here cannot replicate everything that an artist might be doing today with the common color grading methods. Today, an artist can turn pure black to bright red if they really want to. In spite of this, we feel the path outlined is going to be a generally good one that will cover the needs and desires of most. It is something that seems to have some degree of support in the film industry, and we can do a lot worse than emulating a workflow that others with similar challenges utilize.

Practical Implementation

Some game developers might already be shaking their heads about the potential cost of the ACES-derived operators discussed above. In reality, even unoptimized versions aren't that expensive on the fairly high-end HW you typically expect to be connected to the UHD display of an early adopter. However, that cost isn't necessary. Instead, the same solution described previously for LMT application applies for tone mapping as well. The whole process can reasonably be mapped into a 3D LUT, even baking in the LMT. This turns the entire color grading and tone mapping pass into the execution of a shaper function and a 3D texture fetch. This is the sort of workflow that the movie industry has often used, so I think we can be fairly confident that it will generally be good enough for our purposes.

It is important to remember that the 3D LUT will need to hold high-precision data, preferably fp16. This is to ensure that the outputs have the proper precision. Recalling the Barten Ramp mentioned earlier in this paper, fp16 RGBA is the common format that GPUs support today which has the full necessary precision.

FP16 scRGB Back Buffer

As many developers might know, Microsoft Windows has supported the display of fp16 surfaces since Windows Vista. They have always been merely a floating point representation of scRGB. So far, this has meant that the desktop compositor or the display driver applies the sRGB transform then pushes the bits out to the monitor at

whatever precision the display link is running. (While the output to the display may be 30 or 36 bit RGB, the desktop compositor presently only operates with 24 bit RGB)

With the advent of the UHD display standard, NVIDIA has integrated support to send the signal necessary to properly drive these displays. This means signaling the display to operate in the expanded color mode, as well as encoding the data over the display link correctly. To do this, the display driver needs to be told that the application wishes to do so via NVAPI. Additionally, the application creates its display surface as RGBA fp16 to ensure enough precision. The application renders the frame out and leaves it in the scRGB. (sRGB primaries, linear encoding, with (1,1,1) corresponding to the sRGB white level) Finally, the swap chain must be set to fullscreen exclusive, as the desktop compositor does not yet understand UHD displays, so the display driver must be in a position to handle the data from end-to-end.

The display driver takes the scRGB back buffer, and converts it to the standard expected by the display presently connected. In general, this means converting the color space from sRGB primaries to BT. 2020 primaries, scaling to an appropriate level, and encoding with a mechanism like PQ. Also, possibly performing conversions like RGB to YCC if that display connection requires it.

Because the back buffer is floating point, it can still represent the full range of colors in the BT. 2020 gamut by using negative values. The brightest possible level in this output color space is expected to be 12.5, as this corresponds to 10,000 / 80. 10,000 nits is the maximum brightness encoding for the UHD standard, and 80 nits is the standard for sRGB brightness, which is mapped to (1,1,1).

It is important to note that the driver isn't performing tone mapping or gamut mapping. It is expecting the output colors to be encoded by the application in a space that is output referenced and represents a good approximation of what the display device desires. The application will have already tonemapped and applied an appropriate ODT. The display driver is just providing a uniform abstraction to multiple different display standards.

While this is all relying on the NVIDIA driver and NVAPI today, it is expected that this pipeline will port seamlessly to future updates of present operating systems with native functions for identifying streams as UHD compliant. As noted, the scRGB is actually a standard created by Microsoft.

UHD Metadata

As with any good standard, the standards for UHD signaling are setup for future growth. They specify an extremely large color volume of BT. 2020 primaries and maximum luminance of 10,000 nits. However practical devices in the immediate future would expected to be more along the lines of 1000 nits and approximately DCI primaries. As such, the displays are expected to tonemap these full range signals into their displayable range. This is obviously a compromise to allow variation in displays. To help

preserve the intent, the content can also pass along metadata that specifies what range it is mastered against. This is providing the display a hint as to how much tonemapping it potentially needs to apply. If the display only supports 1000 nits, and the signal can top out at 10,000 nits, it may need to reserve a lot of room to express that compressed data. However, if the content tells the display that it only wants to use up to 1000 nits, then the display can avoid that extra compression needed to support those extreme luminance values.

The great bit about games is that they are dynamic content, so they can actually choose to adjust a mastering level for their signal unlike pre-authored content like videos. In the ideal scenario, the game reads back the metadata, and then selects its mastering level to match the display (or provides the user a simple drop down). As it is still somewhat early in the lifecycle of UHD firmware, the feedback isn't necessarily always available. In general, generating a 1000 nit or 500 nit and DCI or sRGB primaries makes a good approximation that the display can come close to supporting. (The 500 nit level is what may likely be appropriate for OLED displays as they generally have lower peak luminance)

One important aspect of the metadata is that it is the way to inform the display that your signal has any high dynamic range or wide color gamut properties. The sending of this metadata is how the display knows to engage the extended capabilities. (Please note that with early firmware, it may be necessary to use a control on the display or its remote to engage this.)

NVAPI Functions

Since metadata is a critical portion of utilizing UHD display outputs, NVIDIA has created a set of NVAPI functions to handle the needed functionality. It is expected that native OS functionality will eventually supersede these functions, but until that time and on legacy operating systems this is the way to provide the data.

General Operation

As this isn't a tutorial on NVAPI, general information on the initialization and similar general topics should be obtained from the NVAPI documentation itself.

All functions work based on an NvDisplayId that must be obtained via NVAPI functions designed for enumerating connected displays. The SDK sample we're releasing in concert with this paper has some example code doing this.

Querying Capabilities

Once a display has been identified, it can be queried for whether it supports HDR via the NvAPI_Disp_GetHdrCapabilities function, which will fill out a structure describing the properties of the display.

```
typedef enum
```

```
{
    //!< The type of structure used to define the Static Metadata Descriptor block.
    NV_STATIC_METADATA_TYPE_1 = 0
}NV_STATIC_METADATA_DESCRIPTOR_ID;

typedef struct _NV_HDR_CAPABILITIES
{
    // Version of this structure
    NvU32 version;

    // HDMI2.0a UHDA HDR with ST2084 EOTF (CEA861.3).
    //  Boolean: 0 = not supported, 1 = supported;
    NvU32 isST2084EotfSupported                 :1;

    // HDMI2.0a traditional HDR gamma (CEA861.3).
    //  Boolean: 0 = not supported, 1 = supported;
    NvU32 isTraditionalHdrGammaSupported        :1;

    // Extended Dynamic Range on SDR displays.
    //  Boolean: 0 = not supported, 1 = supported;
    NvU32 isEdrSupported                        :1;

    // If set, driver will expand default (=zero) HDR capabilities parameters
    // contained in display's EDID.
    // Boolean: 0 = report actual HDR parameters, 1 = expand default HDR parameters;
    NvU32 driverExpandDefaultHdrParameters      :1;
    NvU32 reserved                              :28;

    // Static Metadata Descriptor Id (0 for static metadata type 1)
    NV_STATIC_METADATA_DESCRIPTOR_ID static_metadata_descriptor_id;

    //!< Static Metadata Descriptor Type 1, CEA-861.3, SMPTE ST2086
    struct
    {
        // All elements in this structure are encoded as:
        //  ([0x0000-0xC350] = [0.0 - 1.0])
        // unless noted otherwise

        // x coordinate of color primary 0 (e.g. Red) of the display
        NvU16    displayPrimary_x0;
        // y coordinate of color primary 0 (e.g. Red) of the display
        NvU16    displayPrimary_y0;

        // x coordinate of color primary 1 (e.g. Green) of the display
        NvU16    displayPrimary_x1;
        // y coordinate of color primary 1 (e.g. Green) of the display
        NvU16    displayPrimary_y1;

        // x coordinate of color primary 2 (e.g. Blue) of the display
        NvU16    displayPrimary_x2;
        // y coordinate of color primary 2 (e.g. Blue) of the display
        NvU16    displayPrimary_y2;
        // x coordinate of white point of the display
        NvU16    displayWhitePoint_x;
```

```
        // y coordinate of white point of the display
        NvU16    displayWhitePoint_y;

        // Maximum display luminance = desired max luminance of HDR content
        //  ([0x0001-0xFFFF] = [1.0 - 65535.0] cd/m^2)
        NvU16    desired_content_max_luminance;
        // Minimum display luminance = desired min luminance of HDR content
        //  ([0x0001-0xFFFF] = [1.0 - 6.55350] cd/m^2)
        NvU16    desired_content_min_luminance;
        // Desired maximum Frame-Average Light Level (MaxFALL) of HDR content
        //  ([0x0001-0xFFFF] = [1.0 - 65535.0] cd/m^2)
        NvU16    desired_content_max_frame_average_luminance;
    }display_data;
} NV_HDR_CAPABILITIES;

#define NV_HDR_CAPABILITIES_VER1  MAKE_NVAPI_VERSION(NV_HDR_CAPABILITIES, 1)
#define NV_HDR_CAPABILITIES_VER   NV_HDR_CAPABILITIES_VER1

NVAPI_INTERFACE NvAPI_Disp_GetHdrCapabilities(__in NvU32 displayId,
                    __inout NV_HDR_CAPABILITIES *pHdrCapabilities);
```

It is important to remember that the capabilities reporting by the display is an optional item. In this even, the capabilities will all return as zero. In this event, you should assume a maximum luminance of 1000 nits, a white point of D65, and color primaries of DCI. These are the standard mastering expectations for HDR10/UHDA. Any display supporting this interface should do a reasonably good job at adapting to that level.

Setting Metadata

To enable HDR output and specify the mastering data to the display use the function NvAPI_Disp_HdrColorControl.

```
typedef enum
{
    //!< Get current HDR output configuration
    NV_HDR_CMD_GET = 0,

    //!< Set HDR output configuration
    NV_HDR_CMD_SET = 1
} NV_HDR_CMD;

typedef enum
{
    //!< HDR off - standard Low Dynamic Range output
    NV_HDR_MODE_OFF    = 0,

    //!< UHD BD HDR baseline mandatory output: YCbCr4:2:0 10/12bpc ST2084(PQ) EOTF
    // Rec2020 color primaries. ST2086 static HDR metadata, 0..10000 nits luminance
    // range.
    NV_HDR_MODE_UHDBD  = 2,
} NV_HDR_MODE;

typedef struct _NV_HDR_COLOR_DATA
```

```
{
    //!< Version of this structure
    NvU32                               version;

    //!< Command get/set
    NV_HDR_CMD                          cmd;

    //!< HDR mode
    NV_HDR_MODE                         hdrMode;

    //!< Static Metadata Descriptor Id (0 for static metadata type 1)
    NV_STATIC_METADATA_DESCRIPTOR_ID  static_metadata_descriptor_id;

    //!< Static Metadata Descriptor Type 1, CEA-861.3, SMPTE ST2086
    struct
    {
        //!< x and y coordinates of color primary 0 (e.g. Red) of mastering display
        // ([0x0000-0xC350] = [0.0 - 1.0])
        NvU16    displayPrimary_x0;
        NvU16    displayPrimary_y0;

        //!< x and y coordinates of color primary 1 (e.g. Green) of mastering display
        // ([0x0000-0xC350] = [0.0 - 1.0])
        NvU16    displayPrimary_x1;
        NvU16    displayPrimary_y1;

        //!< x and y coordinates of color primary 2 (e.g. Blue) of mastering display
        // ([0x0000-0xC350] = [0.0 - 1.0])
        NvU16    displayPrimary_x2;
        NvU16    displayPrimary_y2;

        //!< x and y coordinates of white point of mastering display
        // ([0x0000-0xC350] = [0.0 - 1.0])
        NvU16    displayWhitePoint_x;
        NvU16    displayWhitePoint_y;

        //!< Maximum display mastering luminance ([0x0001-0xFFFF] = [1.0 - 65535.0]
        // cd/m^2)
        NvU16    max_display_mastering_luminance;

        //!< Minimum display mastering luminance ([0x0001-0xFFFF] = [1.0 - 65535.0]
        // cd/m^2)
        NvU16    min_display_mastering_luminance;

        //!< Maximum Content Light level (MaxCLL) ([0x0001-0xFFFF] = [1.0 - 65535.0]
        // cd/m^2)
        NvU16    max_content_light_level;

        //!< Maximum Frame-Average Light Level (MaxFALL) ([0x0001-0xFFFF] = [1.0 –
        // 65535.0] cd/m^2)
        NvU16    max_frame_average_light_level;
    } mastering_display_data;
} NV_HDR_COLOR_DATA;
```

```
#define NV_HDR_COLOR_DATA_VER1  MAKE_NVAPI_VERSION(NV_HDR_COLOR_DATA, 1)
#define NV_HDR_COLOR_DATA_VER   NV_HDR_COLOR_DATA_VER1

NVAPI_INTERFACE NvAPI_Disp_HdrColorControl(__in NvU32 displayId,
                    __inout NV_HDR_COLOR_DATA *pHdrColorData);
```

Setting the mastering data (primaries, white point, max luminance, etc) to 0 has the special meaning that the content is mastered for the default with 1000 nit max luminance and DCI primaries. For all gaming content today, DCI primaries or sRGB primaries and 1000 nits maximum specified should produce good results.

UI Compositing

At this point, we've reached the ability to generate and display a scene is HDR on a UHD display. However, no game is complete without user interface elements. This is where scRGB as the output color space really simplifies a developer's life.

Today, UI is authored pretty much directly in the output referred sRGB color space. This has been a standard 2D workflow since roughly the beginning of time. As a result, the simplest solution is to keep the workflow as is. Since the scRGB is set to the same primaries, and with a white level that is the prescribed white for sRGB, the developer can simply blend the UI straight over top as they do today. The colors should come out identical to what the UI artists have been expecting based on their work developing it on an SDR monitor.

If the back buffer was mapped with 1.0 being the max possible luminance for UHD, then the resulting UI would be quite uncomfortable for the user to look at for an extended period of time. Even if the user could tolerate the extreme brightness, they would likely lose a bit of the HDR fidelity, as the overly bright UI would stand as a reference to adapt the eyes making the scene look dimmer.

It is worth mentioning a couple enhancements that a game developer may wish to add to make things go as smooth as possible with the UI. First, it is worth adding a scale factor to the UI when in HDR mode. As users may be accustomed to viewing the game on an overly bright SDR monitor, it may be worth upping the luminance level of the UI by some amount. Additionally, this can counteract the viewer adapting to the brighter highlights in the HDR scene. Secondly, if your game relies on a lot of transparent UI, it may be worth compositing the entire UI plane offscreen, then blending it in one compositing operation. This is useful, because extremely bright highlights can glow through virtually any level of transparency set in the UI, overwhelming the transparent elements. You can counteract that by applying a very simple tone map operator, like Reinhard (x/(x+1)) to bring the max level of anything below transparent pixels to a reasonable level.

Tone Mapping in More Depth

As most experienced graphics programmers already know, tone mapping covers a wide variety of methods. This document makes no attempt to catalog and explain them all. However, it is useful to cover some basic classifications just to show how things compare and why the paper recommends what it does.

The most classic classification of tone mappers is local versus global. For those unaware, the differentiation is merely that global tone mappers perform the same operation on every pixel, while local tone mappers adjust what happens based on local conditions. Simplistically, you can turn the filmic operators recommended here into local operators by having an exposure that varies across the image. Because of the adaptation to different levels within one image, they can compress more data into a useful range. Generally, local tone mappers have a few of potential concerns as we see it today. First, they tend to have temporal stability issues making then less than ideal for moving images. (There was a recent paper by Disney research that offered solutions, but it required a window of several frames forward and backward.) Second, to get good results, local tone mappers often tend to be quite expensive. (Bilateral filters with a radius of more than 5% of the image size) Finally, they can at times look a bit hyper-real, just because of how much they can enhance the image. So, based on performance, quality, and aesthetics, we feel like global operators are still the best the best choice for game developers today.

Another dimension on which tone mappers can be classified is whether they are “constant color” or not. The issue here is whether the tone mapper operates on just the luminance value and maintains the same chromaticity no matter what, or whether it operates on the color channels independently. Obviously, most tone mapping algorithms can be adjusted to do either. Tone mapping the channels separately has a couple advantages. First, there is research tied to CAMs that the cones do adapt at least somewhat independently. Secondly, the curves working independently results in a natural desaturation toward white as you approach peak luminance. The advantage of a luminance only tone mapper is that it can provide something that looks more colorful, which might be a desirable art direction. On balance, we prefer the per-channel operators; however, given the possibility of art being tuned toward a luminance only tone mapper we added an extension to the standard ACES pipeline to allow a blending toward a luminance only mapping.

A really good question to ask is why a tone mapper for HDR should be different compared to the one that you use today on an SDR screen. To generate a good image for an HDR display, the tone mapper really needs to have an understanding of the output luminance range of the display. Simply taking a tone mapper that is intended to generate an SDR image, taking the output as [0,1], and interpreting that as minimum to maximum luminance has problems. First, the original tone mapper will have over-compressed much of the image, leaving it duller than you’d like. Secondly, even though it will pass through brighter values, it isn’t going to express any extra dynamic range from the scene, as this was already compressed out. Finally, the luminance levels produced for most of the scene will be overly bright. If the classic Reinhard operator (x / x + 1) is used and displayed on the full range of a 1000 nit display, the output luminance

for an input middle gray will be over 150 nits. This is twice as luminous as the standard max level for sRGB, and it is nearly as bright as the monitor you are likely reading this on. (Many common LCD monitors have a maximum luminance of 200-300 nits, and most do not have brightness set to the maximum) If the goal were to view it outside on a sunny day, that might be reasonable, but under standard expected viewing conditions this will look overly bright and washed out.

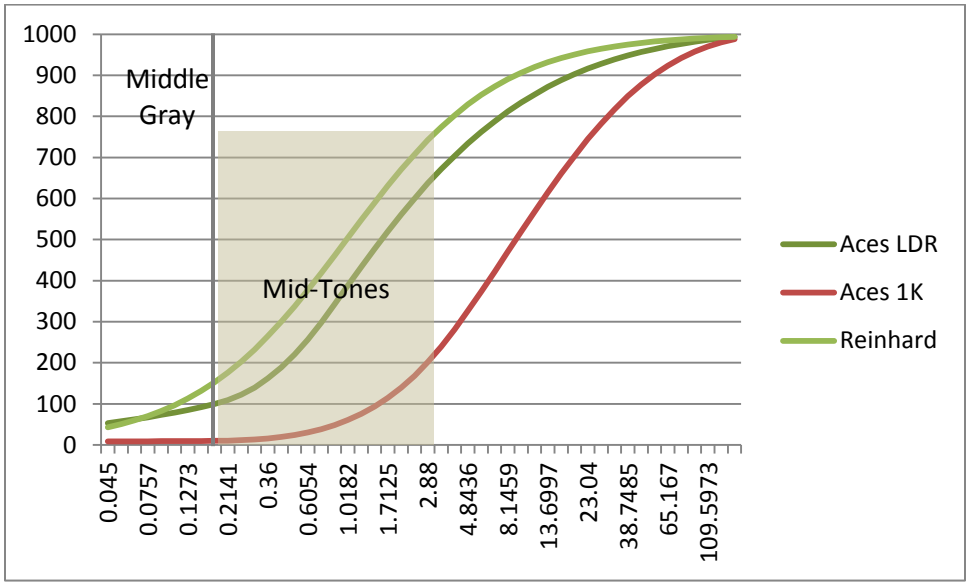


Figure 19 - Tone map functions scaled to max 1000 nit output

As you can see in the diagram above, even the SDR/LDR ACES curve pulls values much brighter than what the 1000 nit ACES curve does. The reasoning is that it is best for the HDR curve to take advantage of much of that dynamic range to try to represent a wider band of mid-tones without compression. It also performs less compression on the highlights, as it simply has more room to work. In practice, the luminance level of middle gray and the mid-tones are boosted only a small amount on an HDR monitor, as these are already at a comfortable viewing level. (ACES SDR transform places middle gray at 8 nits, and the HDR ACES curves place it at 10 nits) It is worth noting that the SDR curve above is already stretched to function over the wider range supported by the 1000 nit reference curve. Otherwise, the curve would have reached 1000 nits at an input level of 16.

Based on these pieces of data, and knowing that many games have recently picked up filmic tone mappers to get a more ‘cinematic’ look, ACES seems like a very logical choice. It is a global tone mapper that is modeled after the behavior of film. Second, it operates on the color channels independently and creates a natural desaturation of highlights. It has several reference curves that provide mappings for different display luminance levels. Finally, it is a standard with some traction.

Parameterized ACES

It is important to remember that ACES is a framework. It offers reference transforms that cover SDR, 1000 nit, 2000 nit, and 4000 nit display output levels. However, it is expected that additional ODTs will be created for additional devices. In that spirit, we have created a parameterized version of the reference ODTs. (Figure 20 shows the shapes and levels of the reference transforms) The parameterized implements the stages of the standard ODTs, but allows them to be modified to handle a wider range of devices. It also helps simplify the shader development and maintenance.

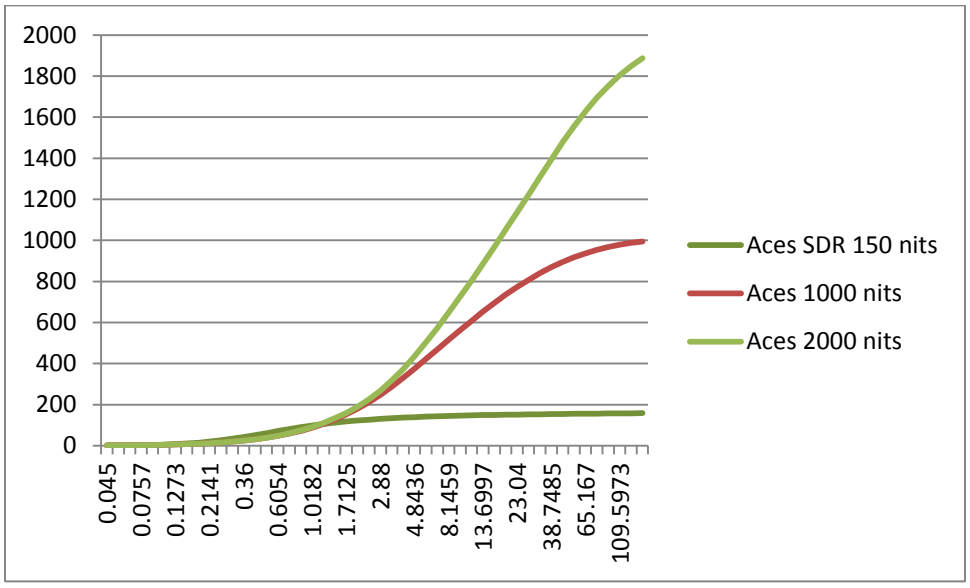


Figure 20 - Results produces by the reference ODTs, including the SDR ODT outputting toa 150 nit max display.

The core portion of the parameterization is in the adjustable ODT curve. The reference curves are all defined as quadratic splines with fixed middle gray, input max luminance, and output max luminance. We've extended on this by implementing rescaling capability where all of these properties can be tweaked to more accurately match the target. For instance, while the 1000 nit reference curve has a peak input luminance of 10 stops above middle gray (0.18 * 2^10 = 184.32) and peak output of 1000 nits. Figure 21 shows the how changes occur in the mid-tone region when the maximum input level is reduced by one stop to 9 stops above middle gray or the output value of middle gray has its luminance doubled. Extensions like this allow a more appropriate curve for a 500 nit class display with a peak output of 500 nits. A peak input of 9 stops above middle gray, and something close to the 1000 nit curve is a good start. Figure 22 shows a set of derived curves targeting 500 nits of output. Note that the SDR derived one likely makes the mid-tones too light. Additionally, it might be advantageous to reduce the maximum input level for content that doesn't get near the default maximum value. Finally, it is possible to interpolate between different reference curves. The code included in the sample demonstrates these sorts of manipulations.

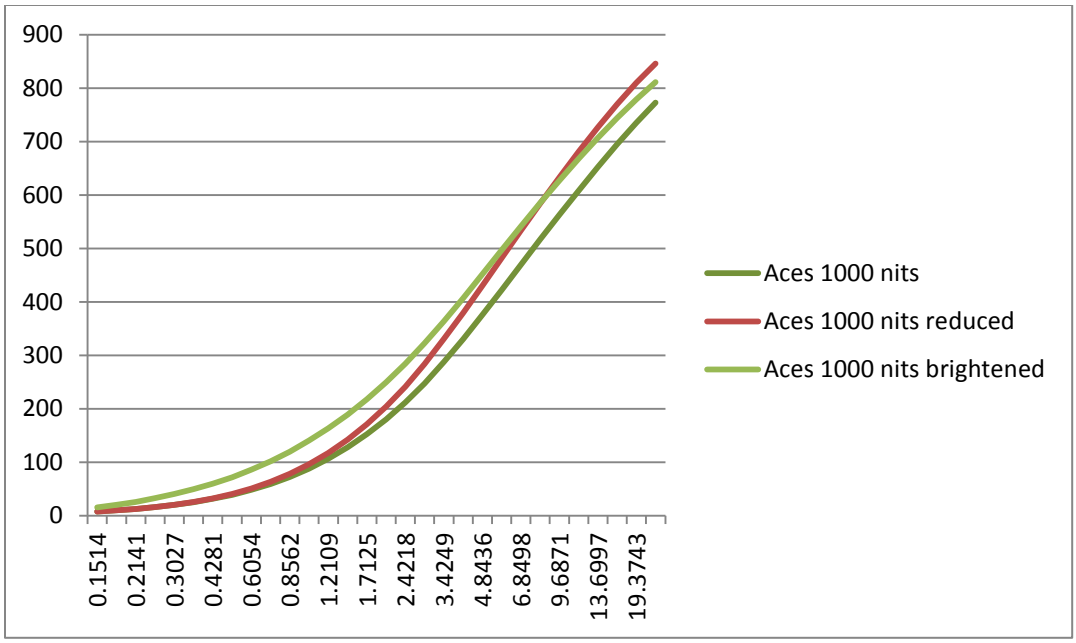


Figure 21 - Results from tweaked 1000 nit ACES ODTs

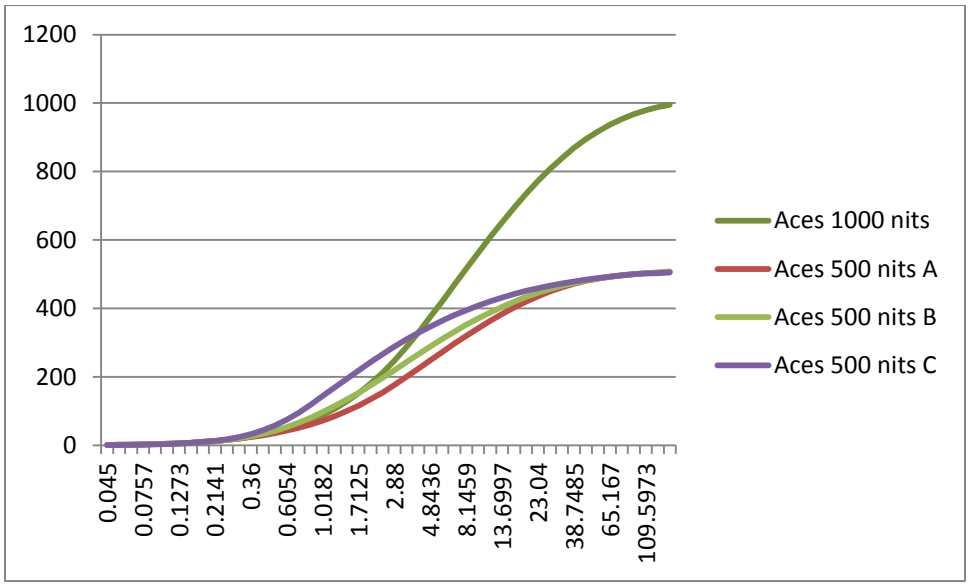


Figure 22 - Derived 500 nit curves compared against 1000 nit reference (A – 1000 nit reference just rescaled to 9 stops and 500 nits, B – 1000 nit curve blended with SDR curve at 50% strength, C – SDR curve scaled up to 500 nit level parameters)

Further, we have added the ability to apply the tone curve just to the luminance component of the input color. This is to emulate the behavior of constant-color tone mapping that some games might have been authored around. The functionality is actually set-up as a weight from fully RGB to fully luminance based allowing a complete continuum to the fully saturated look.

Some final options are tied to viewing environment, output gamut, and display white point. In general, the display white point won't need to be tweaked, as it is tied to the display standard. The output gamut is where colors will get clipped prior to conversion to scRGB. Since your inputs are likely based on sRGB primaries, you'll see little difference in selecting anything wider. Finally, the viewing environment transform is a power function to adjust somewhat for contrast sensitivity differences between dark, dim, and standard environments. In general, the ODTs are setup for the dark environments of movies, so enabling the dim adjustment is probably a good idea.

Gamut Mapping

Gamut mapping is the operation of stretching or compression one color space to fill or fit within another. What we typically think of as tone mapping handles this from the perspective of luminance, but it does not generally address the remapping of chromaticities. As mentioned previously, our advice is generally to render within the sRGB primaries and not attempt any gamut mapping. The primary reasoning here is that gamut mapping is still a very inexact science, and it is fairly likely that most solutions today will have some level of artifacts that upset the art team.

The simplest form of gamut mapping is merely clipping. This is essentially what happens in the pipeline described above. When you clip against the gamut, you can end up with hard cut offs where a hue gradient suddenly goes flat. Clipping isn't a problem for what we've described above, because the math of rendering with sRGB primaries means that you won't generate colors beyond the gamut of the sRGB chromacities. Neither the standard displays nor the new displays will have any issue with the sRGB gamut getting clipped. The tradeoff is that you cannot generate the highest saturation of colors for the UHD displays.

More complex gamut mapping using either linear interpolation to the boundaries or something called soft clipping, where some colors get pulled inside, but clipping still occurs. The choices here would be to move the authoring to a wider color space, which seems like an unrealistic proposition in the near term due to the large cost associated with generating art assets, or to continue in sRGB but to stretch the end result. The challenges with these sorts of remappings is that we as humans have a concept referred to as "memory colors". These are shades we implicitly recognize, such as skin tones. Any stretching or contracting of the gamut needs to be careful to not shift these unrealistically. Additionally, the perceived hues will often shift when linearly interpolating between white and the color extremes. This is because our perception of hues does not match the linear lines in most color spaces.

If a development team wants to pursue generating richer colors for UHD displays, the best advice we have today is to avoid being too aggressive. First, using a color space such as IPT which was created to ensure linear hue lines is a reasonable basis for defining the gamut mapping. Second, anchoring a set of memory colors is likely useful. Finally, practical displays aren't going to support the full BT. 2020 gamut. If you are going to gamut map, gamut map to something like the primaries of DCI-p3, as this is

roughly the space that displays in the near future will be capable of supporting. This way you can minimize them performing compression on the gamut you just attempted to stretch to.

Further Reading