|  |  |
| --- | --- |
| **Joint Collaborative Team on Video Coding (JCT-VC)**  **of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11**  25th Meeting: Chengdu, CN, 14–21 October 2016 | Document: JCTVC-Y0030 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **AHG on HDR and WCG: Multi-LUT Luma Adjustment Implementation** | | |
| *Status:* | Input Document to JCT-VC | | |
| *Purpose:* | Information | | |
| *Author(s) or Contact(s):* | Jacob Ström Färögatan 6 164 80 Stockholm Sweden | Tel: Email: | +46702672192 jacob.strom (at) ericsson.com |
| *Source:* | Ericsson | | |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Abstract

This contribution presents a way to implement luma adjustment using multiple look-up tables (LUTs). Trilinear interpolation is used, and in an attempt to reduce errors due to interpolation over discontinuities, different LUTs are employed for different cases, depending upon whether the output color will clip any of its color components. Seven 3D LUTs are used, indexed based on Cb, Cr and tfiYo, where the latter is the luminance of the color fed through the inverse transfer function. The different tables have different dimensions, such as values or values. In total 99200 values are used for the LUTs and other tables, which the contribution states makes it feasible for video-rate hardware implementations. In an estimate of the accuracy, on one frame of the Market sequence it is claimed that the RMSE error in luma is 0.53 code levels and a maximum error in luma of 1.61 code levels.

# Introduction

Prior to the MPEG Call for Evidence for HDR and WCG Video Coding [1] it was reported in several contributions [2], [3], [4] that there is a subjective quality problem with using the Y’CbCr non-constant luminance 4:2:0 color format for HDR.

In [5] an iterative solution to the problem is proposed that finds the luma sample that results in a linear luminance that is closest to the original luminance signal. We will refer to this solution as iterative luma adjustment. A more detailed description to this method is given in [7]. In [6] a non-iterative solution based on linearization of the transfer function is proposed.

The most similar prior art to our work is perhaps [8] and [9], where a LUT-based approach of obtaining the luma Y’ from the luminance Yo and Cb Cr is taken. The 3D space is divided into eight octants which are further subdivided until an error criterion has been reached. In the leaves of this tree coefficients of a second degree polynomial are stored that can be used to calculate Y’ from tfiYo, Cb and Cr. Here tfiYo = denotes the inverse of the transfer function applied to the original luminance Yo.

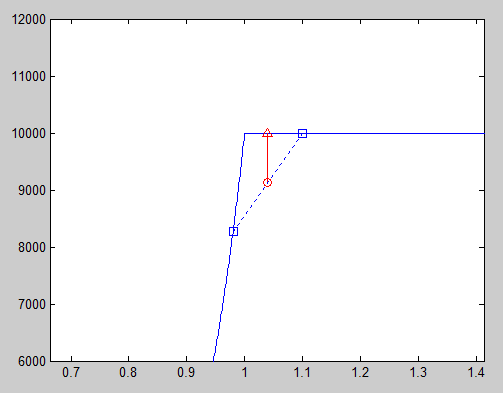
In this contribution an attempt is made to use LUTs without subdivision, employing trilinear interpolation between the eight values closest to the requested triplet (Cb, Cr, tfiYo).

# Interpolation over discontinuities

A brute-force method for creating a LUT would be to quantize Cb, Cr and tfiYo to ten bits and store the result. Assuming 1024 values per dimension this means values or 1.25 Gb if we store 10 bits per value. While this is certainly feasible storage wise it may be difficult to guarantee a random access every clock cycle with a LUT that size.

One possibility to lower the number of table entries while maintaining precision can be to fetch several neighboring values from the LUT and interpolate between them to get the result. In this contribution we use trilinear interpolation. This means that we find the closest values tfiYoLo and tfiYoHi surrounding the requested value tfiYo, and for both these cases we interpolate in Cb and Cr bilinearly. These two cases are then combined linearly to get the resulting luma Y’.

A problem with this solution is that the function Y’ = lumaAdjustment(tfiYo, Cb, Cr) that we are trying to approximate has discontinuities. Doing linear interpolation across a discontinuity typically gives bad results. This is intuitively easy to see from Figure 1 – the interpolated value (circle) becomes far from the real one (triangle) due to the discontinuity.

  
*Figure 1: Example showing the problem of linearly interpolating over a discontinuity. The function drawn with solid blue is sampled before and after the discontinuity (squares). The middle value (red circle) becomes far from the true value (red triangle).*

In the function we are trying to approximate, these discontinuities arise due to clipping of the color components. In detail, the luminance of a pixel is the weighted average of the linear RGB components

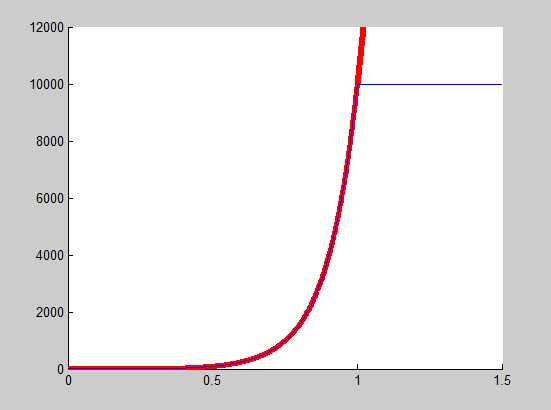
Since where is the transfer function (we have used SMPTE ST 2084), we get

or

where , , , and are positive constants that depend on the color space.

Assume that we want to obtain Y’ = luma\_adjustment(Yo, Cb, Cr) for a specific triplet (Yo, Cb, Cr) that we are interested in. Assume also that we for some reason already know that this particular triplet should not give a Y’ that will cause any component in Equation (5) to clip. When we use the look-up table, we have to interpolate between the eight closest values to (Yo, Cb, Cr), and even though (Yo, Cb, Cr) will not clip itself, some of these neighboring values clip. In that case we are interpolating across a discontinuity as in Figure 1 and the result will be poor.

Therefore we replace the function with a function which does not clip, but is just a linear extension of , as shown in Figure 2. The function can be defined as

  
*Figure 2: When we create a LUT for values that do not clip, we use the linearly extended function (red curve) instead of the typical transfer function (blue curve) that clips.*

For the case when we know that none of the components will clip, we therefore use

when we create the LUT. This means that there will be no interpolations across discontinuities, which will give a better interpolated value. However, it also means that we cannot use that LUT in case the solution actually does clip any of the components.

Therefore, the idea in this contribution is to first find out whether the resulting Y’ will clip. This can be done as presented in [10]. If no components clip, we use one LUT. If one component clips, we use one of six LUTs depending upon which component clips and to which value. In Appendix A we go into detail of the different cases and how the different LUTs are used.

# LUT size

The following table shows the different LUTs and the number of values in each one.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type** | **Number of values** | **Instantiations** | **Nestings** | **Number of values** |
|  | 2048 | 6 | 1 | 12288 |
|  | 2048 | 1 | 5 | 10240 |
|  | 2048 | 1 | 5 | 10240 |
| R0GfBf | 32x8x32 | 1 | 1 | 8192 |
| RfG0Bf | 16x16x32 | 1 | 1 | 8192 |
| RfGfB0 | 4x32x32 | 1 | 1 | 4096 |
| R1GfBf | 32x4x32 | 1 | 1 | 4096 |
| RfG1Bf | 16x16x16 | 1 | 1 | 4096 |
| RfGfB1 | 4x32x32 | 1 | 1 | 4096 |
| RfGfBf | 32x32x32 | 1 | 1 | 32768 |
| CbMinMax | 7x32x2 | 1 | 1 | 448 |
| CrMinMax | 7x32x2 | 1 | 1 | 448 |
| **sum** |  |  |  | **99200** |

Here, is a look-up table that calculates the forward transfer function. It is needed in the method in [10] that determines which variables clip, and it is multiplied by six since six parallel instantiations may be needed if one pixel per clock is needed. The next, , is used to calculate tfiYo, as well as in the case when two color components clip (see Appendix A). As described in [11] several tables (nestings) are needed to get good accuracy near 0. The power function is denoted and it is also nested. Among the LUTs storing the luminance, the one when no component clips is the biggest at 32 kValues. Finally we also need to store min and max values for Cb and Cr. In total 99 kValues are needed.

# Accuracy Evaluation

When we evaluate the accuracy, the error in the retrieved luma Y’ is not necessarily very important. The reason is that due to the non-linear relationship between luma and luminance, in some circumstances it is possible to have a rather large error in luma, whereas the error in luminance is very small.

Instead we can look at the error in luminance. However, just measuring the squared error of the luminance is also misleading, since it would say that an error between 5000 and 5005 is worse than an error between 0 and 4, which is at odds with how we perceive brightness.

A more relevant metric is the error in the luminance after it has been fed through the inverse of the transfer function . This is closer to the way tPSNR-Y is measured and we therefore use this error metric here.

We do two evaluations, for average error and for worst case error. We calculate the average of the squared error , where is the original luminance of the pixel and is the luminance we get when feeding into Equation (3), and is the value we get from the LUT technique presented here.

To find the average error, we do Monte Carlo sampling of the entire color space for all possible values of (Yo, Cb, Cr), and average the result. The root of that average equals the root mean square error of the system, and it is equal to 0.0003540. To set that into perspective, assume we have a white color and we change the luma value Y’ one code level. The luminance is defined as

Since the color is white, we have Cb = Cr = 0. We get

where the last equality is due to the fact that Hence

Assume we have the two code levels 364 and 365 where 64 represents 0.0 and 940 represents 1.0. This means that , and The difference in luminance after inverse PQ is therefore

.

A single code level should be more or less indistinguishable, otherwise we have designed the system with too few bits, 10-bit Y’CbCr would not be enough. In that sense we can use this number as a “just noticeable difference” (JND), and we can see that the average error of 0.0003540 is about 0.31 of such a JND.

To find the worst case error, we do Monte Carlo sampling of the entire color space for all possible values of (Yo, Cb, Cr), and find the worst result. The worst error in then equals 0.002735, or about 2.4 times the JND as defined above.

We have also tried the method on a frame of Market. The RMSE error in luma Y’ is here 0.53 code levels, and the RMSE error in is about 0.45 JNDs as defined above. The worst case error in luma Y’ is 1.61 code levels and about 1.13 JNDs as defined above.

In these accuracy tests , and was not implemented using LUTs.

# Conclusion

This contribution has presented a way to implement luma adjustment using multiple LUTs. First we find out whether the result will clip or not using the technique from [10]. Then we select a LUT based on how many components clip and to which value they clip. The number of values used in the LUTs can be varied according to what accuracy is desired. We present results for a combined size of the LUTs of about 100 kValues. This will generate an average error of about 0.31 times what should be just noticeable, and a worst case that is about 2.4 times what should be just noticeable. It is the understanding of the authors that it is feasible to have LUTs of this size provide one random access in each clock cycle. This means that it is should be possible to use this implementation for real time video rate processing.

# References

1. Call for Evidence (CfE) for HDR and WCG Video Coding, ISO/IEC (MPEG), http://mpeg.chiariglione.org/standards/exploration/high-dynamic-range-and-wide-colour-gamut-content-distribution/call-evidence.
2. E. François, P. Lopez, Y. Olivier, “About using a BT.2020 container for BT.709 content”, ISO/IEC (MPEG) document m35255, Sapporo, July 2014
3. J. Stessen, R. Nijland, R. Brondijk, R. Goris, “Chromaticity Based Color Signals for Wide Color Gamut and High Dynamic Range”, ISO/IEC (MPEG) document m35065, Strasbourg, France, Oct. 2014.
4. J. Ström, “Investigation of HDR color subsampling”, ISO/IEC (MPEG) document m35841, Geneva, Switzerland, Feb. 2015.
5. J. Ström, J. Samuelsson, M. Pettersson, K. Andersson, P. Wennersten, R. Sjöberg, “Ericsson's response to CfE for HDR and WCG”, ISO/IEC (MPEG) document m36184, Warsaw, Poland, June 2015.
6. A. Norkin “Closed form HDR 4:2:0 chroma subsampling (HDR CE1 and AHG5 related)”, JCT-VC JCTVC-W0107, Geneva, Feb 2016.
7. J. Ström, J. Samuelsson and K. Dovstam, ”Luma Adjustment for High Dynamic Range Video”, Proceedings of the IEEE Data Compression Conference (DCC), Snowbird, March 2016.
8. C. Rosewarne and V Kolesnikov, “CE1-related: LUT-based luma sample adjustment”, JCT-VC, JCTVC-W0056, San Diego 2016.
9. C. Rosewarne and V Kolesnikov, “AHG13: Further results for LUT-based luma sample adjustment”, JCT-VC, JCTVC-X0054, Geneva 2016.
10. J. Ström, J. Samuelsson, K. Andersson, P. Hermansson, “Modified Linearization of Luma Adjustment”, JCT-VC, JCTVC-X0036, Geneva 2016.
11. A.M. Tourapis, D. Singer HDRTools: Software status”, JCT-VC, JCTVC-X0053, Geneva 2016

# Appendix A

## No color component clips. RfGfBf

In this case, the optimal Y’ generates a red component R’ such that 0 < R’ < 1, and ditto for green and blue. We say that all three components are “free”, hence the name RfGfBf for this case. We use a LUT of size , i.e., the LUT is 32 elements wide in Cb, 32 elements wide in Cr and 32 elements wide in tfiYo. When creating the table, for every value we calculate the luma adjusted value Y’ = luma\_adjustment\_RfGfBf(Y’, Cb, Cr), but inside that function we do not use Equation (3) to determine the luminance Y in the interval halving procedure, instead we use the formula in Equation (4) to determine Y in order to avoid discontinuities.

## The red component clips.

Here it turns out that it is beneficial to differentiate between two cases, namely when R’ > 1 and when R’ < 0.

### Red clips to 1

In the first case, which we can call R1GfBf, we realize that since the luminance is defined as

the smallest value that can have is . Thus we do not need to tabulate from tfiYo = 0 to tfiYo = 1 but can instead tabulate from tfiYo = to 1, saving a lot of table entries. We can therefore calculate the position in the table as

tfiYoIndex = (LUTsizeYo-1)\*(tfiYo – tfiYoMin)/(tfiYoMax-tfiYoMin),

where tfiYoMin = 0.85708 and tfiYoMax = 1.0. This increases the precision for the same number of LUT elements. Also, since , it is not possible for to exceed 1 if is negative. We take advantage of this fact by not having entries for which Cr < 0 in this table.

To be able to avoid having entries for negative Cr values, we can use a CrMin value of 0 and a CrMax value of 0.5. For a particular Cr value, we can then calculate the position in the table as

CrIndex = (LUTSizeCr-1) \* (Cr – CrMin)/(CrMax-CrMin).

In fact, it turns out that for some values of we can do even better than CrMin = 0 and CrMax = 0.5. We can therefore make CrMin dependent on the tfYoIndex:

CrIndex = (LUTSizeCr-1) \* (Cr – CrMin[tfiYoIndex])/(CrMax[tfiYoIndex]-CrMin[tfiYoIndex]).

This also increases the precision, meaning that it becomes sufficient with a LUT of size . The fact that we can avoid a lot of values in Cr pays off – only a width of 4 is needed.

Since we know that red clips to 1 it does not make sense to use Equation (4) when calculating the values in the LUT, instead we use

since we know the red channel will always saturate.

### Red clips to 0

For the case when R’ < 0, which we can call R0GfBf, we cannot find a minimum value for tfiYo that is larger than 0, but we can find a maximum value. Since we know that R’ < 0, the largest we can possibly get is

so the largest possible value for tfiYo = The LUT we settled on for this case was . Since we know that the red component is always 0, instead of using Equation (4) when calculating the LUT we instead use

## Green or Blue component clips

These cases are done in a manner very similar to the case when the red component clips; the only difference is the values of tfiYoMin and tfiYoMax, and the size of the LUT dimensions. It is summarized in the following table, where we have also included R1GfBf and R0GfBf for completion.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case** | **which means** | **tfiYoMin** | **tfiYoMax** | **LUT dimensions**  () |
| R1GfBf | R’ > 1 | 0.85708 | 1.0 |  |
| R0GfBf | R’ < 0 | 0.0 | 0.96791 |  |
| RfG1Bf | G’ > 1 | 0.95902 | 1.0 |  |
| RfG0Bf | G’ < 0 | 0.0 | 0.87915 |  |
| RfGfB1 | B’ > 1 | 0.69502 | 1.0 |  |
| RfGfB0 | B’ < 0 | 0.0 | 0.99359 |  |

The equation used to calculate the luminance inside the iterative luma adjustment loop during the calculation of the LUT in these cases are

|  |  |
| --- | --- |
| **Case** | **Function used to calculate luminance during creation of LUT** |
| RfGfBf |  |
| R1GfBf |  |
| R0GfBf |  |
| RfG1Bf |  |
| RfG0Bf |  |
| RfGfB1 |  |
| RfGfB0 |  |

When calculating which color components clip according to [10], a by-product is that we get a range , in which the ideal luma Y’ must be included, from Step 5 in Section 4.2 in [10]. We can further refine this range by making sure we never venture outside [0, 1];

Hence we can clip the interpolated value Y’ we get from the LUT against the range

## Two color components clip.

In this situation we know the values of two of the color channels, and only one color channel is influenced by the luma Y’. It is then possible to analytically calculate the luma Y’, as we will see. Repeating Equation (3), we have

Assume we have a case where two of the color channels clip, for instance when the red clips against 0 and the blue channel clips against 1.

Equation (11) then becomes

which can be simplified to

but since Y’ only appears once, we can solve for Y’:

A similar approach can be taken for all the cases when two components clip. Therefore we do not need a LUT at all for this case. However, we do need to be able to calculate the inverse of the transfer function. This can be efficiently done using a nested lookup-table, as is done in the HDRTools software package [11].

## All color components clip

In this case, all three components will be either 0 or 1 when the optimal Y’ is selected. We can therefore choose any Y’ in the allowed range . The method in [10] calculates the midpoint of the interval for Y’ as YpMid. We can therefore use Y’ = YpMid without doing any further computation.

# Extragamma

The reason why the table uses tfiYo = to calculate the index instead of is that the latter would give far too poor results for low luminance values. However, it turns out that an even stronger nonlinearity than can be even more helpful. Therefore, instead of calculating the index in the tfiYo dimension as

tfiYoIndex = (LUTsizeYo-1)\*(tfiYo – tfiYoMin)/(tfiYoMax-tfiYoMin),

we use

tfiYo\_tab =(tfiYo – tfiYoMin)/(tfiYoMax-tfiYoMin),  
tifYoIndex = (LUTsizeYo-1)\* pow(tfiYo, 1/1.7)

This power function can be implemented as a LUT.

# Patent rights declaration(s)

**Ericsson may have current or pending patent rights relating to the technology described in this contribution and, conditioned on reciprocity, is prepared to grant licenses under reasonable and non-discriminatory terms as necessary for implementation of the resulting ITU-T Recommendation | ISO/IEC International Standard (per box 2 of the ITU-T/ITU-R/ISO/IEC patent statement and licensing declaration form).**