Adaptive Gain Curve Tag (ADGC) and adaptiveGainCurveType

The tag and type described in this document were added to the ICC.1 specification on 17 April 2025.

Normative References

ISO 21496-1:20XX Digital photography — Gain map metadata for image conversion — Part 1: Dynamic range conversion

ADGC Tag

Tag signature: 'ADGC' (41444743h)

Permitted tag type: adaptiveGainCurveType

This tag defines a gain-based global tone mapping curve for an SDR or HDR image.

It may be present when the data colour space in the profile header is RGB, and the profile class in the profile header is Input or Display. The tag shall not be present for other data colour spaces or profile classes indicated in the profile header. When the AGCT tag is image specific, the flags in the header indicating that the profile is embedded and cannot be used independently of the color data shall be set (bit position 0 = 1 and bit position 1 = 1).

adaptiveGainCurveType

The adaptiveGainCurveType specifies a gain-based global tone mapping curve, adaptive to varying headroom, for conversion between HDR and SDR representations of the image. It consists of a header and curve data. The adaptive gain curve function, which describes the application of the adaptiveGainCurveType, is defined in Annex 2.

adaptiveGainCurveType header

The byte assignment and encoding of the header shall be as given in Table 1.

Table 1 — adaptiveGainCurveType header encoding

Byte position	Field length (bytes)	Content	Encoded as
0 to 3	4	'adgc' (61646763h) type signature	
4 to 7	4	Reserved, shall be set to 0	
8 to 11	4	Adaptive Gain Curve Function Type ID (set to 1 in this version)	uInt32Number
12 to 27	16	GUID (all zeros when Adaptive Gain Curve is not image-specific)	
28 to 31	4	$H_{baseline}$ (baseline headroom in log base 2)	float32Number

32 to 35	4	$H_{alternate}$ (alternate headroom in log base 2)	float32Number
28 to 31	4	Red channel $Gain_{min}$ (in log base 2)	float32Number
32 to 35	4	Red channel $Gain_{max}$ (in log base 2)	float32Number
36 to 39	4	Red channel weight coefficient (k_{Red})	float32Number
40 to 43	4	Green channel $Gain_{min}$ (in log base 2)	float32Number
40 to 43	4	Green channel $Gain_{max}$ (in log base 2)	float32Number
44 to 47	4	Green channel weight coefficient (k_{Green})	float32Number
48 to 51	4	Blue channel $Gain_{max}$ (in log base 2)	float32Number
52 to 55	4	Blue channel $Gain_{min}$ (in log base 2)	float32Number
56 to 59	4	Blue channel weight coefficient (k_{Blue})	float32Number
60 to 63	4	MAX(R,G,B) weight coefficient (k_{Max})	float32Number
64 to 67	4	MIN(R,G,B) weight coefficient (k_{Min})	float32Number
68 to 71	4	Color component weight coefficient ($k_{Component}$)	float32Number
72 to 75	4	Pre-gain curve CICP (0 if no CICP specified)	uInt32Number
76 to 79	4	Post-gain curve CICP (0 if no CICP specified)	uInt32Number
88 to 91	4	Backward compatible A2B0 target headroom, 0.0: not created	float32Number
92 to 95	4	Backward compatible A2B1 target headroom, 0.0: not created	float32Number
96 to 99	4	Backward compatible A2B2 target headroom, 0.0: not created	float32Number
100 to 107	8	Red Adaptive Gain Curve data position	positionNumber
108 to 115	8	Green Adaptive Gain Curve data position	positionNumber
116 to 123	8	Blue Adaptive Gain Curve data position	positionNumber
124 to 127	4	Reserved for future use, shall be set to 0	

NOTE. Adaptive curve data offsets (byte positions 100-123) may be shared when the data is the same for the respective components.

Adaptive Gain Curve Data

The byte assignment and encoding of the Adaptive Gain Curve data shall be as given in Table 2.

Table 2 — Adaptive Gain Curve Data Type 1 encoding

Byte position	Field length (bytes)	Content	Encoded as
0 to 3	4	Count of {x, y, slope} triplets in the table	uInt32Number
4 to 7	4	x coordinate	float32Number
8 to 11	4	y coordinate	float32Number
12 to 15	4	slope value	float32Number
…	…	…	…
16+(N-1)*12	4	x coordinate	float32Number
20+(N-1)*12	4	y coordinate	float32Number
24+(N-1)*12	4	slope value	float32Number

Patent statement

ICC draws attention to the fact that it is claimed that compliance with this document can involve the use of one or more patents concerning the Adaptive Gain Curve Tag.

ICC takes no position concerning the evidence, validity and scope of these patent rights. The holders of these patent rights have assured the ICC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patent rights are registered with ICC.

ICC maintains a public database of patent declarations at https://www.color.org/iccip.xalter.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. ICC shall not be held responsible for identifying any or all such patent rights.

Annex 1 Adaptive Gain Curve Function (normative)

1.1 General

The metadata in an ADGC tag shall be used to calculate output values according to the Adaptive Gain Curve Function.

1.2 Adaptive Gain Curve Function

The Adaptive Gain Curve Function comprises three steps:

1. an input evaluator
2. a gain evaluator
3. an output evaluator

1.2.1 Adaptive Gain Curve Input Evaluator

The mathematical formulation of the Input Evaluator is expressed as follows:

$$\begin{aligned} \textit{Input value[component]} = {} & \textit{Red}*k_{Red} + \textit{Green}*k_{Green} + \textit{Blue}*k_{Blue} \\ & + \textit{MAX(Red, Green, Blue)}*k_{Max} \\ & + \textit{MIN(Red, Green, Blue)}*k_{Min} \\ & + \textit{component}*k_{Component} \end{aligned}$$

Where *component* is one of {*Red, Green, Blue*}.

1.2.2 Adaptive Gain Curve Gain Evaluator

The mathematical formulation of the Gain Evaluator is expressed as a piecewise cubic curve:

$$F(x) = C3*x^3 + C2*x^2 + C1*x + C0$$

where F(x) is normalized to be relative to the range of the gain values.

Each cubic is defined by a beginning triplet {x1, y1, slope1} and ending triplet {x2, y2, slope2}. The triplets are contained in Adaptive Gain Curve Data and encoded as described in Table 2.

Coefficients C3, C2, C1, C0 can be calculated using these triplets as follows:

$$C3 = (slope1 + slope2 - 2.0 * (y2 - y1) / (x2 - x1)) / ((x1 - x2) * (x1 - x2))$$
$$C2 = ((slope2 - slope1) / (2.0 * (x2 - x1))) - 1.5 * (x1 + x2) * C3$$
$$C1 = slope1 - 3.0 * x1^2 * C3 - 2.0 * x1 * C2$$
$$C0 = y1 - x1^3 * C3 - x1^2 * C2 - x1 * C1$$

1.2.3 Adaptive Gain Curve Output Evaluator

The mathematical formulation of the Output Evaluator comprises three steps:

1. Calculation of the target headroom weight coefficient:

$$W_{target} = sign(H_{alternate} - H_{baseline}) * clamp\left(\frac{H_{target} - H_{baseline}}{H_{alternate} - H_{baseline}}, 0,1\right)$$

2. Calculation of the gain:

$$Gain = 2^{((Gain_{min} + F(x) * (Gain_{max} - Gain_{min})) * W_{target})}$$

3. Calculation of the output:

$$Output\ value[component] = Gain[component] * component$$

Where *component* is one of {*Red, Green, Blue*}.

NOTE 1. When $k_{Component}$ equals zero, the *input value* becomes *luma A* and shall be the same for all components.
NOTE 2. It is recommended to perform the Adaptive Gain Curve calculations using double precision float and to create a lookup table with linear interpolation which can be adjusted by controlling the slew rate. Flat spots or saddles in the Adaptive Gain Curve could cause image quality issues.
NOTE 3. Annex 1 provides an example of calculating the Adaptive Gain Curve as a lookup table with 1024 grid points.

Annex 2 Implementation of adaptiveGainCurveType (Informative)

2.1 General

This annex provides guidance on the implementation of adaptiveGainCurveType.

This information can be also used for creating lutAtoBType approximating Adaptive Gain Curve.

A recommended hierarchy of tags depending on the processing criteria is discussed at the end.

2.2 Calculating a Gain Lookup Table

The pseudocode below is an example of calculating a gain look-up table from the Adaptive Gain Curve.

```
#define MAXNODES 32

typedef struct {
    float x;
    float y;
    float slope;
} node;

typedef struct {
         int
    nNodes;
    node nodes[MAXNODES];
} curve;

void computeCubicCoefficients( node *n1, node *n2, double &C3, double &C2, double &C1, double &C0 )
{
    double x1 = n1->x; double y1 = n1->y; double s1 = n1->slope;
    double x2 = n2->x; double y2 = n2->y; double s2 = n2->slope;

    double u = (x1 - x2) * (x1 - x2);
    double n = s1 + s2 - 2.0 * (y2 - y1)/(x2 - x1);

    C3 = n / u;
    n  = s2 - s1;
    u  = 2.0 * (x2 - x1);
    e  = 1.5 * (x1 + x2) * C3;
    C2 = (n / u) - e;
    C1 = s1 - 3.0 * x1 * x1 * C3 - 2.0 * x1 * C2;
    C0 = y1 - x1 * x1 * x1 * C3 - x1 * x1 * C2 - x1 * C1;
}

#define NTABLE 1024

void createLookupTable(curve *c, float table[NTABLE+1], float gmapMin, float gmapMax)
{
    node *n = c->nodes;
    node *end_n = c->nodes + nNodes - 1;
    float xlo = n->pt.x;
    float xhi = (n + 1)->pt.x;
    double C3, C2, C1, C0;

    computeCubicCoefficients(n, n+1, C3, C2, C1, C0);
    // compute an NTABLE-element table describing the gain factor
    float slew_transfer = -1.0;
    float minimum_slew = 0.00025f;
    for (int i = 0; i <= NTABLE; i++, slew_tranfer += minimum_slew) {
        // evaluate x at this index
        float x = (float)i / (float)NTABLE;
        // advance into the valid node interval enclosing x. this requires two nodes
        while (x > xhi && (n + 1) < end_n) {
            n++;
            xlo = n->x;
            xhi = (n + 1)->x;
            computeCubicCoefficients(n, n+1, C3, C2, C1, C0);
```

```
        }
        // if not enough nodes, we sample-and-hold the curve value at the last interval's end x
        x = (x > xhi) ? xhi : x;
        // evaluate normalized linear gain using doubles (result may be float, though)
        double dx = (double)x;
        float nlg = C3 * dx*dx*dx + C2 * dx*dx + C1 * dx + C0;
        // convert normalized linear gain to gain factor
        table[i] = powf(2.0f, gmapMin + nlg * (gmapMax - gmapMin));
        // compute y: the factor multiplied by luma_A
        // so the transfer table would actually be f(x) = y * x
        float y = table[i];

        // compute transfer table value(x*y)
        float transfer = x * y;
        if (transfer < slew_transfer)
        {
            transfer = slew_transfer
            y = transfer / x;
        }
        table[i] = y;
        slew_transfer = transfer;
    }
}
```

2.3 Applying a Gain Lookup Table

The pseudocode below is an example of applying a gain look-up table.

```
typedef struct
{
    float r; float g; float b;
} RGB_float;

float gain_lookup(float* table, size_t count, float y)
{
    size_t max_index = count-1;
    float fmax = (float)max_index;
    float fIndex = clampf_to_range(fmax * y, 0.0f, fmax);
    uint32_t index = (uint32_t) fIndex;
    float fract = fIndex - index;

    float y1 = table [index];
    float y2 = table [MIN(index + 1, max_index)];

     float gain = (y1 + (y2 - y1) * fract);

    return gain;
}

RGB_float apply_gain_preview(float R, float G, float B)
{
     float y = R*rCoefficient +
               G*gCoefficient +
               B*bCoefficient +
               MAX(R, MAX(G, B))*maxCoefficient +
                MIN(R, MIN(G, B))*minCoefficient;

     float gain = gain_lookup(lookup_table, gain_lookup_table_count, y);

     RGB_float out = {0.0};

     out.r = gain * R;
     out.g = gain * G;
     out.b = gain * B;

     return out;
}
```

2.4 Approximating Adaptive Gain Curve by luAToBType for HDR to SDR tone mapping

A lutAToBType tag can be created to approximate the application of Adaptive Gain Curve for tone mapping an HDR image to SDR. In this case it is recommended that the A curves of the lutAToBType are used to linearize the input data, and the 3D CLUT is calculated by applying the Adaptive Gain Curve to a uniformly distributed 3D grid of RGB components. The CLUT is followed by a matrix converting RGB primaries to CIEXYZ.

Representing the EOTF in a form of a lookup table sometimes leads to loss of precision due to an insufficient number of tone levels. In this case it is recommended to use an additional encoding step to preserve precision. For example, in the case of PQ EOTF, the additional encoding step could consist of taking the values of the PQ ETOF to the power of 1/5 (x^1/5). This would then need to be undone (i.e. x^5) before applying the preview curve to calculate 3D CLUT entries

It is possible to make additional adjustments to produce rendering intents A2B0, A2B1 and A2B2. The presence of lutAToBType tags in the profile to approximate the Adaptive Gain Curve for a specific rendering intent is indicated in the adaptiveGainCurveType header as described in Table 1 by specified target headroom which was used for calculations. It is recommended that the SDR target of 1.0 is used for the reason of backward compatibility with the systems which don't support the concept of headroom.

2.5 Approximating Adaptive Gain Curve by luAToBType for SDR to HDR tone mapping.

The Adaptive Gain Curve can be used for converting SDR to HDR, so potentially a lutAtoBType tag could be created for such a conversion, however no attempts were made by the authors of this proposal to confirm that.

2.6 HDR tone mapping method hierarchy

Adaptive Gain Map and multiple tags in ICC profile related to HDR tone mapping provide multiple choices of rendering method and the selection of a specific method could be based on different criteria like desired rendering quality, available processing power, available memory, rendered image size, etc. The hierarchy based on expected quality is as follows:

1. Adaptive Gain Map, image specific spatial conversion.
2. Adaptive Gain Curve, image specific global conversion.
3. v4 AToB0 created from Adaptive Gain Curve.
4. CICP tag, ITU Recommended Tone Mapping, non-image specific global conversion.
5. Non-image specific v4 AToB0 tag based on generic global conversion.