Implementation of CM for developers - v2 + BPC

Marti Maria, HP

Hi my name is Marti Maria, thanks for attending this small presentation. As the title says, I am going to introduce basic concepts when regarding to implement a color engine. I have to warn you that implementing a color engine may be a tough task. I know that for sure because I did it… twice!

Lately I got the opportunity of doing a full rewrite of my open source color engine, to accommodate new additions in the spec. Second chance means you can fix design errors found on first version, and that's good. I am putting together all that learning I got in this task, hope those recommendations would be useful to someone else.

First thing to do when creating a CMM from scratch is make sure to understand the scope of the project. I have seen many promising projects to fail just because the planned scope was too ambitious or –amazingly- too shy!

Examples of this are "I will create a completely new way to deal with color management that could read V2 and V4 profiles as legacy". Well, it can be done, any software project can be done, but ICC color management by itself is a world and a complete implementation of ICC color management may take years of developer time. In my case, it toke two years to write a fully V4 CMM engine. And it is still stabilizing.

For the opposite "I wish to create a minimal CMM to deal only with simplified V2 profiles". This is ok and perfectly legit, but you have to take into account what is the use you want to give to your CMM. If it is remotely intended to work in production code, please be aware of what kind of profiles are you going to find out there. Profile creators often do use obscure features on the spec. And many times, such "simplified" ICC profiles are all but simplified. End users are not required to understand what is going on, they will just check with Photoshop, which supports every single ICC feature under the sun, and then they will blame your CMM instead of the profile creator. So, at least, be ready to identify and nicely discard unsupported profiles.

A minimal V2 CMM

How to create a minimal V2 CMM is such an interesting topic that that's what I picked for the rest of this talk. Less is more, and for some applications that are already over 2 million lines of code a tiny library is a blessing. So, let's go for a minimal CMM that would be restricted to do v2 color management and as an extra feature would implement black point compensation.

Any capable CMM should be able to, at least, create color transforms by combining input and output profiles. There is no such thing as “apply a single profile”. Profiles are applied as pairs. A color manager converts from an input color space described by the input profile to a destination color space described by the output profile. Applying such transform is often referred as re-rendering.

To do that, profiles needs to be able to be “connected”. And they certainly have this capability. They have a live wire, a connection point called the “profile connection space”. This is the PCS. This PCS can be either a sort of XYZ or Lab. I say “a sort” because that’s not the conventional XYZ or Lab, but a scaled version of the original color spaces. ICC calls that “relative encoding”. Basically the device white is always D50 and black point is represented by XYZ or Lab zero, which should not interpreted as the physical value. Such perfect absorber is not likely to be very frequent in nature! Both spaces are colorimetric and therefore it is possible to convert between them without any additional information, but a word of warning: there are substantial differences on the profile interpretation depending on the selected PCS. More on that latter.

Profile contents

An ICC profile, V2 or V4 as well, behaves like a bag. A bag with an attached sign. In the sign, which is the profile header, you can see information on the profile as a whole, like which is the PCS, the name and creator of the profile, the copyright notice…etc. Inside the bag there are several building blocks, which the CMM have to use to build the color transform. Two or more building blocks which potentially do same thing can coexist in the profile. Did you identify any issue with this last sentence? Sure! Two different ways to do same thing may end in differences. And the main benefits of ICC color management are repeatability and consistence. If different CMM do choose different building blocks, we would end with a nice mess… To overcome that, ICC spec clearly defines a precedence rule. Any CMM should honor this precedence rule and of course all building blocks. Failure to do so will end in different color rendition and is when end user complains. Keep the idea that is NOT up to the CMM to choose which building block should be used for each situation; this is up to the ICC spec.

Now let’s examine which are those basic blocks any V2 compliant CMM should be able to handle.

A tone curve for gray profiles.

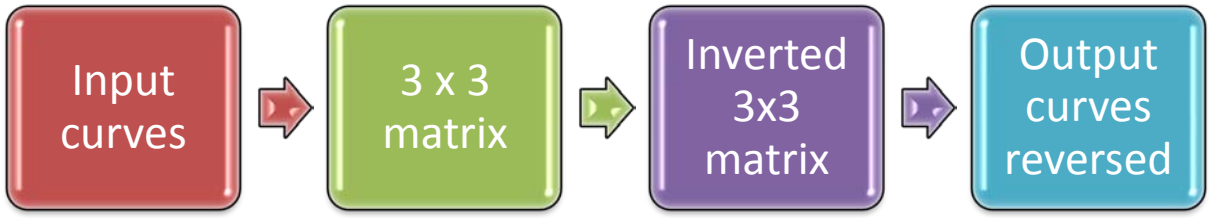
This is used by the simplest ICC profile you can build. A monochrome profile, describing a gray color space, which has only one channel. In theory those profiles only can work on XYZ. The curve represents a scaling on the white point, which is D50. Otherwise, I have seen profiles operating in Lab as PCS, where this curve is assumed to be a scaling on L*. Those profiles are rare. To connect two of those profiles, the CMM should invert the input tone curve. This is assumed to scale the white point, so the PCS is obtained by multiplying white point by the curve. The output can be obtained by taking the Y component of PCS and multiplying it by the output curve.

V4 makes things more complex with parametric curves, but on V2 you can only specify curves in two ways. Either the curve is a pure exponential, and the profile stores the exponent, as a fixed point number or the curve is given as a quantized table of 8 or 16 bit values. A zero entries table represents an identity which does no operation.

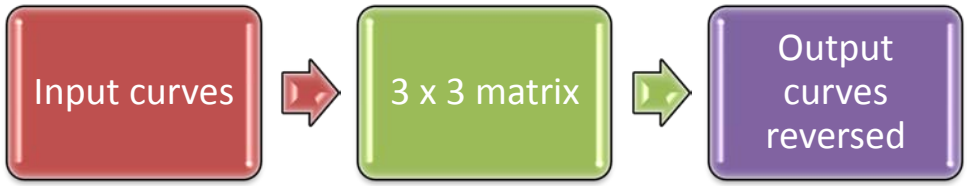
RGB Matrix-shaper.

A little bit more complex than the previous element. This structure implements a set of 3 curves and a 3x3 matrix. The structure is not stored in a single tag, but instead different tags are involved. Those tags have no meaning alone, only when used together. This is important because some people confuse primaries with matrix rows, and this is not really true. Matrix elements can be negative, for example. Please note there are only one set of curves and matrix for both input and output directions. That means matrix and curves should be inverted by the CMM when using the profile in the input direction. Inverting the matrix is not an issue at all, you can just use the determinant, but inverting the curves may be challenging. Because this inversion, curves are required to be monotonic. Some commercial CMM does impose a linear tram in the lower portion of the curve to avoid artifacts on dark zones. This is often known as “slope limiting”. Matrix shaper only works on XYZ profile connection space, so a profile using this building block should be marked as XYZ. This is a good check to do, if the PCS field in the header is not XYZ, something is going wrong.

If we want to create a color transform by connect two of those profiles, we need to invert curves on the input profile, invert matrix of the input profile and then concatenate both structures in this way:



Both middle matrices can be multiplied and this gives this simple model:



That can be nicely optimized by using SID instructions. I think you may be interested in how to optimize that.

C code example

If you, as me, prefer plain C you can use this trick with tables and fixed point: Shaper1 are 3 tables of 256 entries. Shaper2 are 3 tables of 16384 entries.

Shaper1 does include first tone curve plus conversion to 1.14 fixed point. Shaper2 includes second shaper and 1.14 fixed point to 8 bit conversion. Compilers like this code and parallelize it nicely.

```
#define DOUBLE_TO_1FIXED14(x) ((cmsS1Fixed14Number) floor((x) * 16384.0 + 0.5))
#define FIXED_TO_DOUBLE(x)    ((cmsFloat32Number) ((x) / 16384.0))


    // Across first shaper, which also converts to 1.14 fixed point

    r = Shaper1R[ri];
    g = Shaper1G[gi];
    b = Shaper1B[bi];

    // Evaluate the matrix in 1.14 fixed point
    l1 =  (Mat[0][0] * r + Mat[0][1] * g + Mat[0][2] * b + 0x2000) >> 14;
    l2 =  (Mat[1][0] * r + Mat[1][1] * g + Mat[1][2] * b + 0x2000) >> 14;
    l3 =  (Mat[2][0] * r + Mat[2][1] * g + Mat[2][2] * b + 0x2000) >> 14;

    // Now we have to clip to 0..1.0 range
    ri = (l1 < 0) ? 0 : ((l1 > 16384) ? 16384 : l1);
    gi = (l2 < 0) ? 0 : ((l2 > 16384) ? 16384 : l2);
    bi = (l3 < 0) ? 0 : ((l3 > 16384) ? 16384 : l3);

    // And across second shaper,
    ro = Shaper2R[ri];
    go = Shaper2G[gi];
    bo = Shaper2B[bi];
```

Now a word of warning.

CMM implementers may think that this simple approach is enough for their purpose, and create sort of “restricted” CMMs that use only the matrix shaper building block. Well, this is more frequent you may imagine. Unfortunately for those CMMs, a lot of profiles do include, aside matrix shaper, other building blocks. And those building blocks have more precedence that matrix shaper. So, using the matrix shaper means discarding the right resource. And this is of course is a bad idea. It uses the profile in a way that the profile creator didn’t intend, and therefore may not deliver the desired colors.

About precedence

Let's go into the details of this building block that have more precedence that matrix shaper: the Look up table. The CMM has not to reverse such structures, as the profile will specify different LUTs for input and output. The profile can also specify a different LUT for each intent, all but absolute colorimetric. We may have perceptual, relative colorimetric and saturation intents and the profile may be used as input and as output, so we end with 6 different building blocks stored in 6 different tags. ICC calls "A" to the device space and "B" to the PCS, and the intents are numbered from zero for perceptual to 2 for saturation. Here are the combinations

AtoB0, AtoB1, AtoB2, BToA0, BToA1, BToA2

It is possible for a profile to store only one tag for all intents, and in this case AtoB0 and BtoA0 are used.

ICC spec clearly states a priority rule: For those interested, priorities are defined in ICC spec point 8.10. If CLUT is present; CLUT should be used and takes precedence over matrix shaper. This is the complete rule for an input profile going on saturation intent:

- AtoB2
- AtoB0
- Matrix shaper

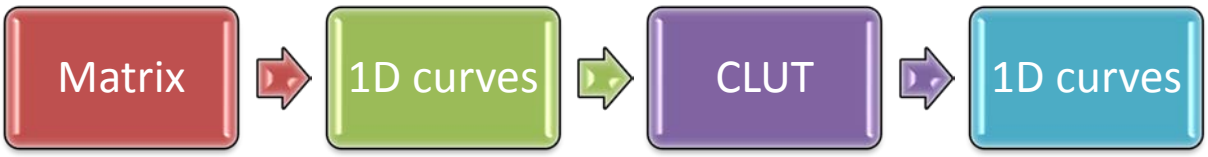
And here is the precedence rule for an output relative colorimetric:

- BToA1
- BtoA0
- Matrix Shaper

You may ask, why the spec allows two different building blocks to compete for same functionality. Wouldn't be easier just to forbid duplicate definitions? Fine, you are mostly right, but you should realize some of those blocks are wildcarding more than one different target. Ok, that sound complicated, so time for one example. Let's imagine a camera profile where we want precision on the input direction and a coarse approximation if used as output. Yes, a camera profile may be used as an output as well, for example to convert a yet-existing image to the camera space. Well, we want to mimic the camera has taken this image, maybe to do some emulations. This is not frequent and therefore a simple matrix shaper is enough. So the profile builder chooses to store an AtoB0 tag with the accurate input transformation and then a Matrix-Shaper for reverse transformation. This is ok, but wait, if we specify a matrix shaper, this is going to apply on both directions. Ok, no problem, since the precedence of AtoB0 prevents matrix shaper to be used in the input direction. But what happens when a simple minded CMM tries to use this profile? Yes, it founds the matrix; yes, it ignores the LUT, yes disaster strikes!

Look up tables

So far so good. Let's go on the details of this LUT structure. Features: It can be used on any color space. It is not limited to RGB, and on V2, it is the only way to deal with CMYK. Whilst is certainly harder to implement than matrix-shaper, the LUT has a simpler purpose. It can be used only in one direction, either input or output and there is no need of invert anything. Actually, LUTS, which was originally an acronym for "Look up table" involves more than a simple look up table.



Those are the stages that you can found in a XYZ LUT. And those are the stages you can find in a Lab LUT

The stages may be different depending on the PCS color space, and here you go with the promised example that the chosen PCS may change the way the CMM should interpret some building blocks. The spec says that the matrix "shall be the identity matrix unless the input color space is XYZ", but trust me, some well known profiles do store a non identity matrix assuming the CMM will ignore it.

You already know how to deal with the matrix and how to interpolate across 1D curve, since you did it for matrix shaper and now there is a new element which is the CLUT. A multidimensional interpolation algorithm should be used to get the output values. In general, a 3 to 4 interpolation, for example is same as 4 3 to 1 interpolations, so you have to repeat the process for any outputs the color space has. There is abundant literature on how to deal with that, and this topic is worth of another ½ hour, so I will not discuss how to interpolate. But here is a recommendation on which kind of interpolation to use. If the input space has the luma component centered, like RGB or CMYK then you can use tetrahedral interpolation. This is fast and can get rid of some artifacts like oscillations. If the input space has luma uncentered, like Lab, you should use trilinear or higher order interpolation. Well, if you don't trust me on this, please check profiles on different CMMs, both methods can differ 3 and more digital counts.

Building the color transform

To create a color transform, we need to concatenate the building blocks found in both profiles and some conversion stages. Now we know all possible building blocks you can find inside the profile, it is time to complete our list with some building blocks that should be provided by the CMM itself. Remember that PCS can be either XYZ or Lab, then, to connect the building blocks, we need at least a Lab to XYZ block and a XYZ to Lab. The conversion is straightforward, but may be challenging to implement conversions that go really fast. Aside those, we could add also two features: absolute colorimetric intent and black point compensation.

Absolute colorimetric intent

If our small CMM wants to support absolute colorimetric intent, we need a conversion stage between profiles. This intent can be accomplished by using the relative colorimetric intent on both sides of transform, and then doing a scaling on the XYZ PCS. Basically we want to get absolute colors from input to output. To do that, and since the PCS comes encoded relative to the white point, first thing to do is recover original color. It is encoded as:

$$PCS = \frac{XYZ \cdot D50}{W_{in}}$$

So, to get the original, absolute color, we need to undo the encoding:

$$XYZ = \frac{PCS \cdot W_{in}}{D50}$$

Then you want to pass this original color to output profile, but the output profile needs colors encoded in a relative way, so we need to apply our first equation:

$$\frac{\left({}^{PCS \cdot W_{in}}/_{D50}\right) \cdot D50}{W_{out}} = PCS \cdot \frac{W_{in}}{W_{out}}$$

Looks weird because the output white point comes as the dividend, but that is how it works. This scaling happens in the XYZ color space, so if input and output are operating in Lab color space, you need to convert back and forth Lab → XYZ → Lab

Black point compensation

Black point compensation works in a similar way. We want to map source black point over destination black point and source white point over destination white point. White points are both D50 because the relative encoding and for the black points we have a nice challenge. There is a tag named “black point”, but you should not trust it: ICC found such quantity of broken profiles on regarding this tag that has deprecated it. It’s a chicken-egg issue. CMM does not use black point tag because it is unreliable on many profiles. Profile vendors do not care about black point tag contents because CMM does not use it.

No need to worry. Here is a trick that works most of the cases. Create a transform by using the profile you want to analyze both as input and as output, by using following mixed intents:

Lab → BToA0 → Device Space → AToB1 → Lab

Then force a*/b* to zero and here you go with the profile black point. Then just apply this formula:

$$a=\frac{BPout - D50}{BPin - D50} \qquad b=\frac{D50\cdot(BPin - BPout)}{BPin - D50}$$

XYZo = a · XYZi + b

Optimizing

Ok, performance is one of the main drivers when one is trying to build a color manager. On my first try I went in many problems trying to optimize performance. My wrong assumption was to optimize each different building block combination. And since there is such amount of different combinations, it is likely some combination will get better performance. Then the wrong profile will strike on the least favorable situation. For our small CMM, I would recommend a different approach.

Don’t optimize! Seriously, do all calculations by using floating point. This would simplify a lot the CMM development, and will help a lot to catch bugs. Still you have normalization issues, but if you convert all to the 0...1.0 range, this helps a lot. Once you have those slow transforms implemented, go for a unified optimization that would be, for example one big 3D table with tetrahedral interpolation. Pick for example 33 nodes. This is 33 * 33 * 33 * 3 = 106Kbytes, a bargain in today’s computers. Proceed to compute each node by evaluating the floating point transform on the node coordinates, and store a roundup of the obtained high-precision transform. Then you can exercise your assembly skills in generating a highly optimized tetrahedral interpolation. Or just use the GPU, which does trainer at unbeatable speed. Don’t’ waste time optimizing matrix-shaper, it is useless!

Final considerations

That's all you need to get started in the CMM building. But please note that is just the beginning if you plan to build a serious CMM. Advanced CCM may want to do soft proofing or gamut checking, and if you plan to support V4 be prepared to add additional logic, but I found the building blocks approach works very well in supporting all existing features. It makes your CMM very modular and extensible and easy to maintain for a team. In today's software management arena, those are valuable attributes.