https://doi.org/10.2352/ISSN.2169-2629.2017.32.242

An iccMAX Material Profile Example: Converting Spectral Images of Artwork to Paint-Concentration Images

Ben Bodner, Roy S. Berns; Munsell Color Science Laboratory; Rochester Institute of Technology; Rochester, NY

Abstract

The ICC is introducing a new standard, iccMAX, that allows for customized connection spaces. The step-by-step process of implementing a complex linear algorithm in an iccMAX material profile is presented. This provides a reference for creating iccMAX profiles that use the programmable calculator functionality. The algorithm converts spectral reflectance to paint concentrations that produce a spectral match. Several tools are discussed that produce iccMAX profiles for a spectral workflow being developed at the Studio for Scientific Imaging and Archiving of Cultural Heritage at RIT.

Introduction

The International Color Consortium (ICC) provides a widely-used international standard for digital color management. Embedding an ICC profile in a digital image file allows for consistent color reproduction of that image across various devices with different native color spaces. The previous specification, ICC.1, is designed around one Profile Connection Space (PCS) that is based on colorimetry and a single illuminant, observer, and lighting geometry.

The new specification, iccMAX, is more versatile. Profiles can be defined to convert encoded image data to a colorimetric PCS with any illuminant and observer, to a spectral PCS, or to a Material Connection Space (MCS) with flexibility in the number and meaning of channels. iccMAX profiles can also contain complex programmable functions for conversion between connection spaces. These new capabilities provide a standard for color management of spectral images. They also can be used to create analysis tools that are embedded in spectral image files, themselves.

The Studio for Scientific Imaging and Archiving of Cultural Heritage at RIT is developing a workflow for high-resolution spectral imaging of artwork. An iccMAX profile is presented here that converts spectral images of paintings to images whose channels represent component paint concentrations. Because iccMAX has many profile types and countless features, the learning curve for building and using iccMAX profiles is steep. This article presents a linear algorithm, explained concisely in Part I, and describes how to implement it as an iccMAX MCS profile step-by-step in Part II. This example can be used as a reference for the iccMAX programmable calculator and illustrates the usefulness of MCS profiles.

The full iccMAX specification, software for compiling the profile described here, and other sample profiles are currently available by request at http://www.color.org/iccmax/index.xalter Software tools developed at the Studio for Scientific Imaging and Cultural Heritage Archival are available for download at https://www.rit.edu/cos/colorscience/mellon .

Part I: Algorithm Background

Paint Application of Kubelka-Munk Theory

Kubelka-Munk theory is an optical mixing model that is applicable to many subtractive color systems. Colorants are combined in terms of spectral absorption and scattering in the two-constant theory (2KM), or a ratio of the two in single-constant theory (1KM), rather than reflectance or radiance. The single-constant theory assumes that the scattering coefficient of chromatic colorants is negligible compared to the scattering of the substrate, which can lead to error for certain colorants. Berns and Mohammadi[1] described the steps to calculate 2KM and 1KM coefficients for paints, where white paint is analogous to a substrate. They compared the accuracy of both approaches in predicting the spectral properties of mixtures of white with a single chromatic paint. Their results showed that the extra effort required for the 2KM approach was worthwhile in many cases. Abed[2] performed a similar comparison with a larger set of paints and more complex mixtures, and found that the 1KM approach often caused large errors while 2KM performed well in almost all cases.

Estimating Paint Concentrations of a Mixture

Allen developed a generalized linear algorithm to calculate the concentrations of a given set of colorants needed in a mixture to colorimetrically match a sample using any single-constant subtractive model[3], and later adjusted this algorithm to fit specifically with two-constant Kubelka Munk theory[4]. If the colorants are the same as the ones used to create the sample, the resulting concentrations should produce a match in spectral reflectance between the calculated mixture and the sample. The 2KM version of the Allen algorithm assumes a mixture of four colorants and a substrate. Berns[5] adjusted the algorithm for three chromatic paints and one white paint, removing the substrate term.

The absorption-scattering ratio of a sample can be calculated from its internal reflectance. Internal reflectance is the amount of diffuse light in the paint layer, different from the surface reflectance measured with a spectrophotometer. The measured reflectance can be corrected to internal reflectance using the equation described by Saunderson[6]:

$$R_i = \frac{R_m - \kappa_{ins}\kappa_1}{(1-\kappa_1)(1-\kappa_2) + \kappa_2 R_m - \kappa_{ins}\kappa_1\kappa_2} \tag{1}$$

where R_i is internal reflectance, R_m is measured reflectance, and the κ terms are constants that can be optimized for the measurement geometry and subtractive mixing model being used. For the system used in this example, the terms were optimized to $\kappa_1 = .03$ and $\kappa_2 = .65$ by Okumura [7]. For this example, the impact of κ_{ins} was found to be negligible, so it was set to 0 to simply the implementation. The Allen algorithm is divided into a first step where initial concentrations are estimated for a close match, then

an iterative step that converges to an exact match. Both steps rely on values calculated similarly to XYZ tristimulus values that use 2KM coefficients in place of reflectance. These can be thought of as pseudo-tristimulus values X^P, Y^P, and Z^P :

$$\begin{bmatrix} X^P \\ Y^P \\ Z^P \end{bmatrix} = \begin{bmatrix} \bar{x} \\ \bar{y} \\ \bar{z} \end{bmatrix} E(D_k k + D_s s) \tag{2}$$

$\bar{x}, \bar{y}$, and $\bar{z}$ are row vectors containing color matching functions. E is a diagonal matrix containing the spectral radiance of the illuminant used for matching the mixture and sample. k and s are column vectors containing spectral absorption and scattering coefficients, respectively. D_k and D_s are diagonal matrices containing the partial derivatives of internal reflectance of the sample with respect to k and s:

$$D_k = \frac{-2{R_{i,a}}^2}{s_a(1-R_{i,a}^2)}, D_s = \frac{R_{i,a}(1-R_{i,a})}{s_a(1+R_{i,a})} \tag{3}$$

where the subscript a denotes the sample. It's very unlikely that the scattering coefficient s_a of the sample will be known, because only the ratio $(\frac{K}{S})$ can be calculated from reflectance:

$$\left(\frac{K}{S}\right) = \frac{(1-R_i)^2}{2R_i} \tag{4}$$

An assumption is made that $s_a = 1$ at all wavelengths and $k_a = (\frac{K}{S})_a$. When the iterative step of the algorithm is reached, s_a and k_a will be adjusted to more accurate values.

Pseudo-tristimulus values are calculated using equation 2 for the sample denoted by subscript a, the chromatic paints denoted by $p1, p2$, and $p3$, and the white paint denoted by w. k and s of the individual paints should already be known, and the 2KM coefficients that will be used in this example were measured by Okumura[7]. The initial concentration estimates for the chromatic paints are calculated using these pseudo-tristimulus values with an inverse matrix operation:

$$\begin{bmatrix} c_{p1} \\ c_{p2} \\ c_{p3} \end{bmatrix} = \begin{bmatrix} (X^P_{p1} - X^P_w) & (X^P_{p2} - X^P_w) & (X^P_{p3} - X^P_w) \\ (Y^P_{p1} - Y^P_w) & (Y^P_{p2} - Y^P_w) & (Y^P_{p3} - Y^P_w) \\ (Z^P_{p1} - Z^P_w) & (Z^P_{p2} - Z^P_w) & (Z^P_{p3} - Z^P_w) \end{bmatrix}^{-1} \begin{bmatrix} (X^P_a - X^P_w) \\ (Y^P_a - Y^P_w) \\ (Z^P_a - Z^P_w) \end{bmatrix} \tag{5}$$

The sum of all four paint concentrations will equal 1, so the concentration of white is simply $c_w = 1 - (c_{p1} + c_{p2} + c_{p3})$.

In the iterative step, the reflectance of the mixture is calculated using the initial concentrations and compared to the reflectance of the sample. First, the standard 2KM mixture equation is applied using the four paint concentrations to determine $(\frac{K}{S})$ of the calculated mixture

$$\left(\frac{K}{S}\right)_m = \frac{c_{p1}k_{p1} + c_{p2}k_{p2} + c_{p3}k_{p3} + c_w k_w}{c_{p1}s_{p1} + c_{p2}s_{p2} + c_{p3}s_{p3} + c_w s_w} \tag{6}$$

Internal reflectance is calculated using the inverse of equation 4:

$$R_i = 1 + \left(\frac{K}{S}\right) - \sqrt{\left(\frac{K}{S}\right)^2 + 2\left(\frac{K}{S}\right)} \tag{7}$$

and measured reflectance of the mixture is found using the inverse of equation 1:

$$R_m = \frac{((1-\kappa_1)(1-\kappa_2)R_i)}{(1-\kappa_2 R_i)} + \kappa_{ins}\kappa_1 \tag{8}$$

The measured reflectance of the sample and calculated mixture are used to find the XYZ tristimulus values of each, using the same color matching functions and illuminant used when finding pseudo-tristimulus values in equation 2. The inverse matrix from equation 5, which was initially calculated with $s_a = 1$, is recalculated with $s_a = s_m$. The scattering coefficient of the mixture, S_m, is equal to the denominator of equation 6. The inverse matrix is multiplied by a column vector containing the differences in tristimulus values between the sample and mixture:

$$\begin{bmatrix} \Delta c_{p1} \\ \Delta c_{p2} \\ \Delta c_{p3} \end{bmatrix} = \begin{bmatrix} (X^P_{p1} - X^P_w) & (X^P_{p2} - X^P_w) & (X^P_{p3} - X^P_w) \\ (Y^P_{p1} - Y^P_w) & (Y^P_{p2} - Y^P_w) & (Y^P_{p3} - Y^P_w) \\ (Z^P_{p1} - Z^P_w) & (Z^P_{p2} - Z^P_w) & (Z^P_{p3} - Z^P_w) \end{bmatrix}^{-1} \begin{bmatrix} X_a - X_m \\ Y_a - Y_m \\ Z_a - Z_m \end{bmatrix} \tag{9}$$

The Δc values are added to the corresponding concentrations found in equation 5, and the new concentrations are used to update the mixture. The tristimulus values of the updated mixture will be closer to the sample. Equations 6 to 9 are repeated and the concentrations are updated with each iteration. If the paints chosen are capable of reproducing the sample, the difference in tristimulus values between the sample and mixture will converge to 0.

Part II: iccMAX Implementation

Material Connection Spaces

ICC Profiles translate data between color spaces to allow for connections between measurement devices, encoding, and display. Many profiles have predefined relationships between these spaces, such as profiles that translate color defined in spectral channels to colorimetric channels. Material profiles in the iccMAX standard allow the user to define the number and the meaning of channels however they wish, with the expectation that two material profiles intended to connect between one another share at least some of the same user-defined channels.

MVIS and MLNK profiles define the relationship between a material color space and useful output spaces, such as displayed RGB values or spectral data. MID profiles define the relationship between input data and the material color space. This example shows an MID profile that translates input spectral data to paint concentrations, using the algorithm described in the previous section.

Creating an MID Profile

ICC profiles can be written in XML format. Profile information is placed inside tags defined by the ICC, denoted by angle

brackets. This example includes essential tags to create a functioning MID profile that converts color data. Other tags not included here that provide metadata such as copyright information and intended use of the profile may be expected in a compliant iccMAX profile. More detailed information on profile tags is availabe in the iccMAX specification document.

This example will use a spectral image of a painting with 28 channels. The painting consists of four paint types with known spectral absorption and scattering coefficients.

The Header

XML files begin with a tag stating the XML version in use and how it is encoded. The first line of the .XML file is:

```
⟨?xml version="1.0" encoding="UTF-8"?⟩
```

declaring that the file is written in unicode characters. After the XML version declaration, the rest of the file contains the ICC profile code. The entire profile is enclosed in *IccProfile* tags. The profile version, type, and number of input and output channels are enclosed in *Header* tags:

```
⟨IccProfile⟩
⟨Header⟩
⟨ProfileVersion⟩5.0⟨/ProfileVersion⟩
⟨ProfileDeviceClass⟩mid⟨/ProfileDeviceClass⟩
⟨DataColourSpace⟩nc001C⟨/DataColourSpace⟩
⟨MCS⟩mc0003⟨/MCS⟩
⟨/Header⟩
```

Profile version 5.0 allows the use of material profiles. The profile type *MID* is specified in the *ProfileDeviceClass* tag. The *DataColourSpace* tag describes input channels, where *nc* specifies device input channels and 001*C* is the hexidecimal representation of 28, the number of spectral channels in the input image. The *MCS* tag is used to describe the output of the profile, where *mc* specifies that output will be user defined material channels and 0003 is the hexidecimal representation of 3, the number of paint concentration channels in the output.

Channel Definitions

The remaining profile code after the header is enclosed in the *Tags* tag. A detailed description of the output channels is enclosed in the *TagArrayType* tag:

```
⟨tags⟩
⟨tagArrayType⟩
⟨TagSignature⟩mcta⟨/TagSignature⟩
⟨ArraySignature⟩utf8⟨/ArraySignature⟩
⟨ArrayTags⟩
⟨utf8TextType⟩⟨TextData⟩C_redOxd⟨/TextData⟩
⟨/utf8TextType⟩
⟨utf8TextType⟩⟨TextData⟩C_ultBlu⟨/TextData⟩
⟨/utf8TextType⟩
⟨utf8TextType⟩⟨TextData⟩C_ylwOcr⟨/TextData⟩
⟨/utf8TextType⟩
⟨/ArrayTags⟩
⟨/tagArrayType⟩
```

The Tag Signature *mcta* specifies that these are material channels with custom names, and the Array Signature *utf8* specifies the names will be unicode text. The three material channel names are provided in the *ArrayTags* tag, named for the concentration of red oxide, ultramarine blue, yellow ochre. Including a white paint channel would be redundant becaused its value can be found by subtracting the sum of the three chromatic concentrations from 1. An MVIS or MLNK profile that connects this material space to an output space should include some or all of these channel names. The system described in this example would use an MVIS profile that defines concentration channels for all the paint types characterized by Okumura[7] and converts them to display RGB space.

Processing Elements

Processing instructions to convert input channels to material channels are enclosed in the *MultiProcessElementsType* tag:

```
⟨multiProcessElementType⟩
⟨TagSignature⟩A2M0⟨/TagSignature⟩
⟨MultiProcessElements InputChannels = "28"
OutputChannels = "3"⟩
⟨CalculatorElement InputChannels = "28"
OutputChannels = "3"⟩
```

The *MultiProcessElementsType* block begins with the Tag Signature *A2M*0, indicating this processing converts from input data (A) to material data (M). The *MultiProcessElements* tag specifies the number of input and output channels of the processing step. The only processing element used in this example is a *CalculatorElement*, so it has the same number of input and output channels. The tags left open here will be closed at the end of the processing step.

The calculator consists of two parts. First, sub-elements are defined that contain data other than the image data, or instructions other than those in the *MainFunction* of the calculator. The second part is the *MainFunction*, where processing occurs. The *MainFunction* can call upon built in operators to process data and the sub-elements defined earlier. The concentration estimation algorithm requires additional data aside from the input image data. The 2KM coefficients of the four paints, the illuminant spectral radiance, and the color matching functions will be defined using subelements:

```
⟨SubElements⟩
⟨ !−−Element 0 − absorption coefficients −−⟩
⟨MatrixElement InputChannels = "4"
OutputChannels = "28"⟩
⟨MatrixData⟩...⟨/MatrixData⟩
⟨/MatrixElement⟩
⟨ !−−Element 1 − scattering coefficients −−⟩
⟨MatrixElementInputChannels = "4"
OutputChannels = "28"⟩
⟨MatrixData⟩...⟨/MatrixData⟩
⟨/MatrixElement⟩
⟨!−−Element2 − illuminant −−⟩
⟨MatrixElementInputChannels = "1"
OutputChannels = "28"⟩
⟨MatrixData⟩...⟨/MatrixData⟩
```

⟨/*MatrixElement*⟩
⟨! − −*Element*3 − *cmf* − −⟩
⟨*MatrixElementInputChannels* = "28"
OutputChannels = "3"⟩
⟨*MatrixData*⟩...⟨/*MatrixData*⟩
⟨/*MatrixElement*⟩
⟨/*SubElements*⟩

The values in the *MatrixData* tags would be numeric and organized in a matrix. The number of rows corresponds to the output channels specified in the *MatrixElement* tag and the number of columns corresponds to the input channels.

The Stack Calculator

The iccMAX calculator allows for data processing using a customized set of linear operations. This calculator is stack-based, meaning all values used for calculation are held in a one-dimensional array, where the first value in the array is the bottom of the stack and the last value is its top. Typically, only values at the top of the stack can be accessed, and the result of any operation is placed on top of the stack.

This section will describe several different operators used in the implementation of the concentration estimation algorithm. These operators exemplify different categories of operators that interact with the stack in different ways. Each code example is followed by the resulting contents of the stack. Multiple lines of code will be accompanied by the results of each line shown separately. A full list of operators is available in the iccMAX reference specification.

The calculator instructions are enclosed by the *MainFunction* tag:

⟨*MainFunction*⟩
in(0,28) *%Rm*

Stack Contents: $\|R_{m,1},...,R_{m,28}$

The *in* operator retrieves image data. The arguments of the *in* operator specify which channels ill be retrieved. Channels (along with the stack) are indexed starting with 0, so this command retrieves 28 channels of data starting with the first (0th) channel. In otherwords, all channels in the image are placed on the stack. w Operators are separated by spaces, and most are followed by parentheses containing arguments. Line spacing does not affect the calculator, except that all text placed after a % symbol on any given line is ignored by the calculator, which is useful for commenting.

To calculate internal reflectance, as in equation 1, a second copy of the measured reflectance is placed onto the stack and the value κ_2 is placed on top

copy(28,1) .65

$\|R_{m,1},...,R_{m,28},R_{m,1},...,R_{m,28},0.65$

The $copy(q,t)$ operator copies the top q values on the stack t times. Simply typing a number will place it on top of the stack. κ_2 needs to be multiplied by the top set of R_m values.

smul(28)

$\|R_{m,1},...,R_{m,28},0.65\cdot[R_{m,1},...,R_{m,28}]$

The $smul(q)$ operator multiplies the single value on top of the stack by each of the preceding q values of the stack. *smul* is an abbreviation of "single multiplication" and other similar operators follow the same naming convention (eg. *sdiv* and *ssub*).

1 1 .03 .65
sub(2) mul
sadd(28)
div(28) %Results in Ri

$\|R_{m,1},...,R_{m,28},0.65\cdot[R_{m,1},...,R_{m,28}],1,1,.03,.65$
$\|R_{m,1},...,R_{m,28},0.65\cdot[R_{m,1},...,R_{m,28}],(0.97)\cdot(0.35)$
$\|R_{m,1},...,R_{m,28},0.65\cdot[R_{m,1},...,R_{m,28}]+.3395$
$\|[R_{m,1},...,R_{m,28}]/[0.65\cdot[R_{m,1},...,R_{m,28}]+.3395]$

To complete equation 1, the values $[1,1,\kappa_1,\kappa_2]$ are placed on the stack. The $sub(q)$ command subtracts the top q values from the preceding q values. A set of similar arithmetic operators, such as $add(q)$ and $div(q)$, behave in the same manner. This is useful for element-wise operations on pairs of arrays.

Because R_i is used several times throughout the algorithm, it would be useful to store it elsewhere to be accessed later. This can be achieved using the temporary stack:

tsav(0,28) %put Ri on the temp stack

$\|R_{i,1},...,R_{i,28}$

$tsav(q,t)$ copies the top t elements on the stack onto a temporary stack, starting with index q of the temporary stack. Although this data structure is named "the temporary stack" in the iccMAX documentation, it behaves more like a traditional array. Any element of the temporary stack can be accessed at any time, not just the top elements. *tsav* does not affect the values on the main stack. Operators $tadd(q,t)$ and $tget(q,t)$ move values to and retrieve values from the stack, respectively.

For the iccMAX implementation, it's easiest to rearrange the pseudo-tristimulus calculation from equation 2 and the inverse-matrix operation from equation 5 into one equation:

$$\begin{bmatrix}c_{p1}\\c_{p2}\\c_{p3}\end{bmatrix} = (TE[D_k(\Phi_k - k_w u) + D_s(\Phi_s - s_w u)])^{-1} \cdot TE[D_k(k_a - k_w)] \tag{10}$$

T is a matrix of color matching functions as row vectors, Φ_k is a matrix of column vectors k_{p1},k_{p2}, and k_{p3}, Φ_s is a similar matrix of paint s values, and u is row vector $[1,1,1]$. Values for all the terms in equation 10 were stored in the profile's *subelements* block earlier, with the exception of the derivative weight terms D_k and D_s and the sample absorption term k_a. The term $D_s(s_a - s_w)$ is omitted from the equation because s_w and s_a are both equal to 1 at all wavelengths.

As shown in equation 3, D_k and D_s are based on the internal reflectance of the sample. For the initial concentration estimate calculated in equation 10, it is assumed that $k_a = (\frac{k}{s})_a$ and $s_a = 1$ at all wavelengths. Equation 4 shows that $(\frac{k}{s})_a$ also relies on R_i. R_i currently sits on the main stack and the temporary stack. The numerator of equation 4, $(1-R_i)^2$ is found in three steps:

-1 smul(28)
1 sadd(28)
sq(28)

$\|0-[R_{i,1},...,R_{i,28}]$
$\|1-[R_{i,1},...,R_{i,28}]$
$\|[1-[R_{i,1},...,R_{i,28}]]^2$

ssub can't be used to directly find $(1-R_i)$, because it subtracts the top value from the preceding values. The first two lines achieve the equivalent operation $(1+(-R_i))$. $sq(q)$ squares the top q values on the stack. The denominator of equation 4 is $2R_i$, so R_i will need to be retrieved from the temporary stack:

tget(0,28) 2 smul(28)
div(28) 0 0 0 1
mtx(0) %k_w
sub(28) %k_a - k_w

$\|[1-[R_{i,1},...,R_{i,28}]]^2, 2\cdot[R_{i,1},...,R_{i,28}]$
$\|k_{a,1},...,k_{a,28},0,0,0,1$
$\|k_{a,1},...,k_{a,28},k_{w,1},...,k_{w,28}$
$\|k_{a,1}-k_{w,1},...,k_{a,28}-k_{w,28}$

$tget(q,t)$ copies t consecutive values from the temporary stack starting with index q. Values retrieved from the temporary stack are copied, not moved. If more than one set of values is stored on the temporary stack, it's helpful to keep track of indices corresponding to different values through comments. Dividing the bottom half of the stack by the top half with *div* results in k_a.

The $mtx(q)$ operator uses the q sub-element to perform a matrix multiplication on a column vector. If sub-element q is not a *MatrixElement*, an error will occur. The column vector is taken from the stack, and the number of elements used depends on the *InputChannels* setting in the *MatrixElement*. Sub-element 0 was the matrix of paint absorption coefficients, with k_w in the fourth column. Multiplying the matrix by $[0,0,0,1]^T$ places the fourth matrix column on the stack.

Calculating D_k, D_s, and completing the latter half of equation 10 can be accomplished using the operators discussed thus far:

tget(0,28) sq(28) 0 2 sub smul(28) %Dk numerator
1 copy(1,27) tget(0,28) sq(28) sub(28) div(28) %Dk
mul(28) %Dk(k_a - k_w)
1 mtx(2) mul(28) mtx(3) % illuminant and cmf data
tput(28,3) % T E $D_k(k_a-k_w)$

$\|k_{a,1}-k_{w,1},...,k_{a,28}-k_{w,28},-2\cdot[R_{i,1},...,R_{i,28}]^2$
$\|k_{a,1}-k_{w,1},...,k_{a,28}-k_{w,28},D_{k,1},...,D_{k,28}$
$\|D_{k,i}\cdot[k_{a,1}-k_{w,1},...,k_{a,28}-k_{w,28}]$
$\|TED_k(k_a-k_w)$
$\|$

Sub-element 2 is a single column matrix with illuminant data, and sub-element 3 is a 3x28 matrix of color matching functions. The final matrix multiplication results in just three values on the stack, described as pseudo tristimulus values in equation 2. Moving these three values onto the temporary stack leaves the main stack empty.

Calculator Sub-Elements

Users can define their own operators in the *SubElements* block with the tag *CalculatorElement*. A calculator sub-element has all the same properties as the main calculator but it is called as an operator at the user's request in the main calculator. A calculator sub-element has its own stack and can have its own "sub-sub-elements". The number of input channels is defined in the *CalculatorElement* tag, and the values for these input channels are taken from the top of the stack of the main calculator.

Two calculator sub-elements make sense for this implementation. The first takes s_a values from the stack as 28 input channels, and calculates $TE[D_k(\Phi_k-k_wu)+D_s(\Phi_s-s_wu)]$, which will be referred to as matrix B. This would be accomplished using a very similar method to the set of steps described in detail to solve for $TE[D_k(k_a-k_w)]$. This calculator sub-element is useful because the matrix needs to be calculated twice, first with $s_a=1$ at all wavelengths when calculating the initial concentrations, then with $s_a=s_m$ during the iterative step. It should be inserted in the *subelements* block as sub-element 4.

Once matrix B is calculated, the resulting nine values on the stack must be arranged as a 3x3 matrix, inverted, and multiplied by $TE[D_k(k_a-k_w)]$. This is possible using the *solv* operator:

1 copy(1,28) calc(4)
tget(28,3) solv(3,3) pop
tput(31,3) %initial concentrations

$\|matrixB$
$\|C_{p1},C_{p2},C_{p3}$

$calc(4)$ calls sub-element 4 as a calculator, with the 28 input channels representing $s_a=1$ in this case. $tget(28,3)$ places the result of $TE[D_k(k_a-k_w)]$ on the stack. $solv(q,t)$ creates a column vector from the top q values on the stack, and a matrix from the preceding $q\cdot t$ values on the stack, with q rows and t columns. The column vector is multiplied by the inverse or pseudo-inverse of the matrix, and the result is placed on the stack. Additionally, the *solv* operator places a 1 on the stack to indicate the inverse operation was applied successfully. The *pop* operator removes this value from the stack. The three values remaining on the stack will be the initial concentration estimates for the three chromatic paint channels, which are moved to the temporary stack.

For the iterative step, it makes sense to use a second calculator sub-element. This is sub-element 5, and should be inserted in the *subelements* block. This calculator takes 31 input channels from the stack, which should be the 28 image channels containing R_m values followed by the three chromatic paint concentrations found in the previous step. It copies the last three input values to its temporary stack, and calculates measured reflectance R_m of a mixture using equations 6 to 8. XYZ tristimulus values are found for the mixture and image, and the difference ΔXYZ is saved to the temporary stack.

s_m, the scattering coefficient of the mixture, is found using the denominator of equation 6. The matrix B is then updated with the new s_m values using calculator sub-element 4. The updated matrix B is multiplied by the ΔXYZ values, as in equation 9. The resulting ΔC values are added to the corresponding input concentrations, which were saved on the temporary stack, and the updated concentrations are output.

The final lines of the main calculator should place R_i, the image input, and the initial concentrations on the stack, then repeatedly call calculator sub-element 5. No looping functionality is currently available in the iccMAX, so $calc(5)$ must be called multiple times manually. The number of iterations required for convergence depends on the properties of the paints in the mixture, and the level of acceptable error. The tristimulus errors with the initial concentration estimates are typically less than 1%, but some paint combinations can take over 100 iterations to converge.

```
tget(0,28) in(0,28) tget(31,3)
%Ri,Ra,initial concentrations calc(5) calc(5) calc(5).... %re-
peated many times
out(0,3) }
⟨/MainFunction⟩
⟨/CalculatorElement⟩
⟨/MultiProcessElements⟩
⟨/multiProcessElementType⟩
⟨/Tags⟩
⟨/IccProfile⟩
```

The $out(q,t)$ operator outputs the top t values from the stack starting with channel q. In the calculator sub-element, these values are placed on the stack of the main calculator. After the last $calc(5)$ call, the concentration values on top of the stack are output to the *MID* material channels and all remaining open tags from earlier in the profile are closed.

Preliminary Results

A spectral image with known paint concentrations that was measured by Abed [2] was used to test the iccMax implementation of this algorithm. A few problem pixels existed where the measured reflectance in the spectral image fell outside the theoretical gamut of the paints, possibly due to errors in the spectral image capture. For the vast majority of locations in the 400x270 pixel image, the concentrations were calculated accurately, as shown in Figure 2.

Table 1: Algorithm Performance

	Iterations	Error
Mean	0.53	0.6%
Max	5000	100%
90th prctl	0	1.6%

A MatLab implementation of the algorithm was used to evaluate the number of iterations necessary for convergence. The threshold for tristimulus error was set to 1% of the measured tristimulus values, with a maximum of 5000 iterations. As shown in table 1, at least 90% of pixels were below the tristimulus error threshold with the initial concentration estimate and no iterations were required, and 90% of pixels had errors in paint concentration less than 1.6%.

Conclusions and Future Work

A linear algorithm for estimating paint concentrations from spectral data that fits within the iccMAX framework was presented. Preliminary testing of an implementation of the algorithm shows that it can produce accurate results as part of the spectral workflow established at the Studio for Scientific Imaging and Archiving of Cultural Heritage at RIT. Adjustments will be made to the algorithm to allow for artwork composed of more than three

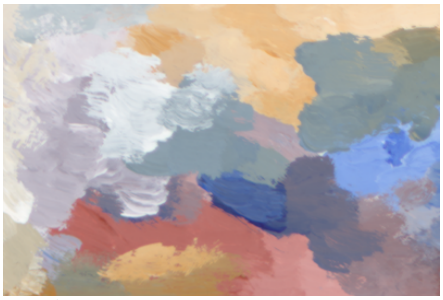

Figure 1: The test image displayed as RGB under D50.

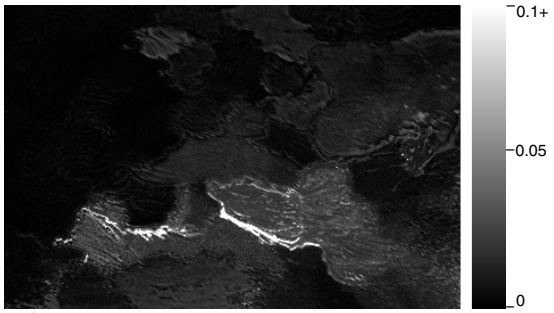

Figure 2: Maximum error of all three concentration channels at every pixel. Concentrations have values between 0 and 1, white represents an error of 0.1

chromatic paints. Several tools are being developed at the Studio that will generate iccMAX profiles based on user selections in a GUI, allowing easy creation of profiles that use different wavelength inputs and paint outputs. iccMAX profiles are also being created for a new high-resolution spectral camera being developed in the Studio. The iccMAX standard will be a vital part of sharing work produced using the Studio's spectral workflow.

References

[1] R.S. Berns and M. Mohammadi, "Evaluating Single- and Two-Constant Kubelka-Munk Turbid Media Theory for Instrumental Based Inpainting"', Studies in Conservation, vol. 52, pp. 299-314, 2009.

[2] F.M. Abed, "Pigment Identification of Paintings Based on Kubelka-Munk Theory and Spectral Images", Ph.D. Dissertation, Rochester Institute of Technology, NY, 2014.

[3] E. Allen, "Basic Equations Used in Computer Color Matching", J. Optical Society of America, vol. 56, no.9, pp. 1256-1259, 1966.

[4] E. Allen, "Basic Equations Used in Computer Color Matching, II. Tristimulus match, Two-Constant Theory", J. Optical Society of America, vol. 64, no. 7, pp. 991-993, 1974.

[5] R.S. Berns, "A Generic Approach to Color Modeling", Color Research and Application, vol. 22, no. 5, pp. 318-325, 1997.

[6] J.L. Saunderson, "Calculation of the Color of Pigmented Plastics," J. Optical Society of America, vol. 32, pp. 727-736, 1942.

[7] Y. Okumura, "Developing a Spectral and Colorimetric Database of Artist Paint Materials" M.S. Thesis, Rochester Institute of Technology Rochester, NY, 2005.