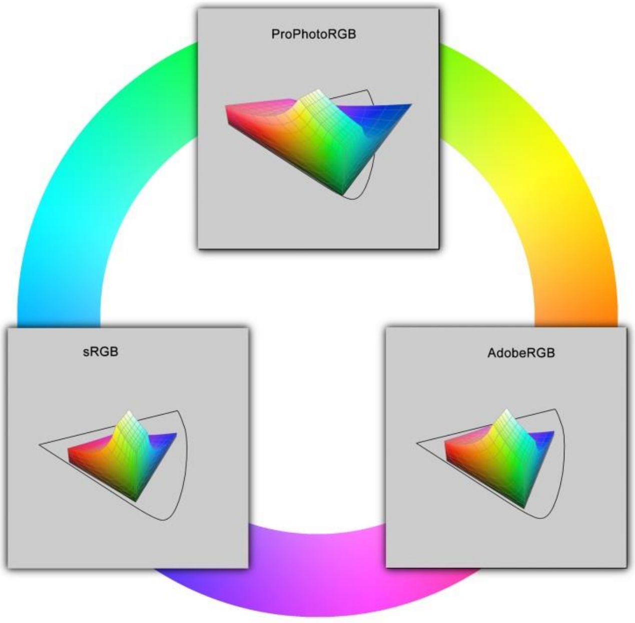
sRGB, AdobeRGB and ProPhotoRGB

The merits of *sRGB*, *AdobeRGB* (*aRGB*) and *ProPhotoRGB* colour spaces are frequently the subject of debate and some confusion. Attempts by the layman to understand the basic principles behind these colour spaces and their associated profiles are often hindered by the complex, 3-dimensional mathematical transforms used to describe them.



In an attempt to understand the basic principles I devised a simpler, linear, 1-dimensional, fictional analogy which describes 3 methods of encoding vehicle speed called (to help the analogy) *sMPH*, *aMPH* and *ProMPH*.

sMPH

Suppose engineers wanted to record the variation in forward speed of a motor vehicle at 1 second intervals over the duration of each journey. The vehicle was equipped with a sensor that provided speed readings from 0.0 to 100.0 mph with 1 decimal place at 1 second intervals. Realising that they didn't need 0.1 mph precision and that they needed to minimise the memory storage requirements for long journeys, the engineers decided they wanted to store each speed reading in a single 8 bit byte (which can only store integer values from 0-255). To achieve this they devised a simple encoding method which multiplies the sensor readings by 2.5 and then rounding to the nearest whole number. They called this process "*Standard MPH encoding*" or *sMPH* for short.

sMPH encoded reading = ROUND(2.5 x Sensor reading)

A speed of 100.0 mph can therefore be encoded in a single 8 bit byte with a value of 250. To recover the original data, the process is simply reversed. They called this process *sMPH* decoding.

Decoded reading = *sMPH* encoded reading / 2.5

sMPH encoding allowed speed readings from 0.0 - 100.0 mph to be stored as single 8 bit bytes with values ranging from 0 - 250. The precision has however been reduced from 0.01mph to 1/2.5 = 0.4 mph. It is this reduction in precision that allows the data to be compressed for storage.

Realising that requirements might change in the future, the engineers decided that files of *sMPH* data should have details of the *sMPH* encoding/decoding process ("the *sMPH* profile") embedded in the file. This would ensure that future users of the data files would know the data inside was *sMPH* encoded and they would know the correct way to recover (decode) the data.

The *sMPH* encoding and decoding process is shown schematically in Figure 1. Note that due to the data compression, 4 different speed values (e.g. 0.2, 0.3, 0.4 and 0.5mph) are typically encoded (and hence decoded) as the same value (e.g. 0.4 mph).

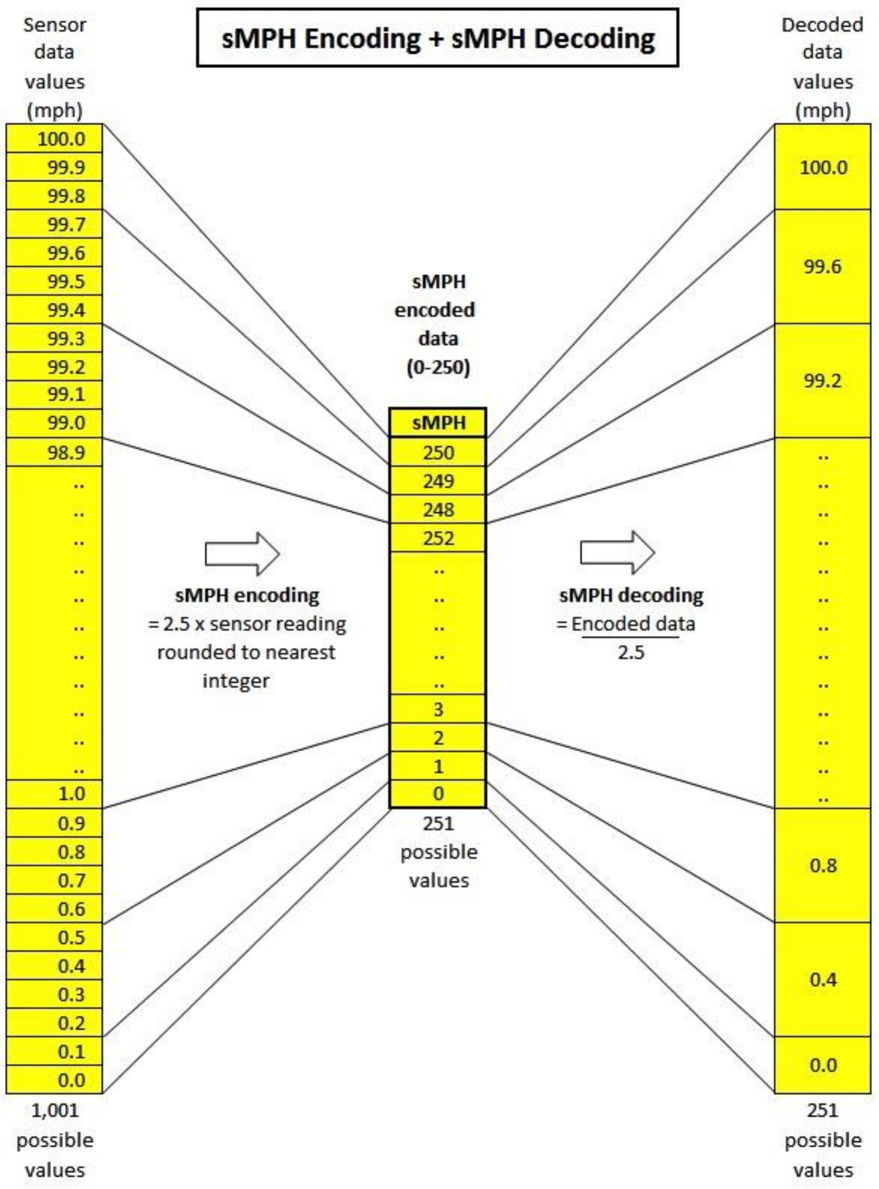


Figure 1

aRGB

As technology advanced engineers decided that they needed to record speeds of vehicles that could go faster than 100mph (up to 125mph) using a similar (1 byte/reading) file format. To accomplish this they devised a new encoding and decoding method which they called "*Advanced MPH*" or *aMPH* for short. To cover the extended speed range they changed the encoding and decoding factor from 2.5 to 2.0

aMPH encoded reading = ROUND(2.0 x Sensor reading)

A speed of 125.0 mph can therefore be *aMPH* encoded in an 8 bit byte with a value of 250. To recover the original data, the process is simply reversed. They called this process is *aMPH* decoding.

Decoded reading = *aMPH* encoded reading / 2.0

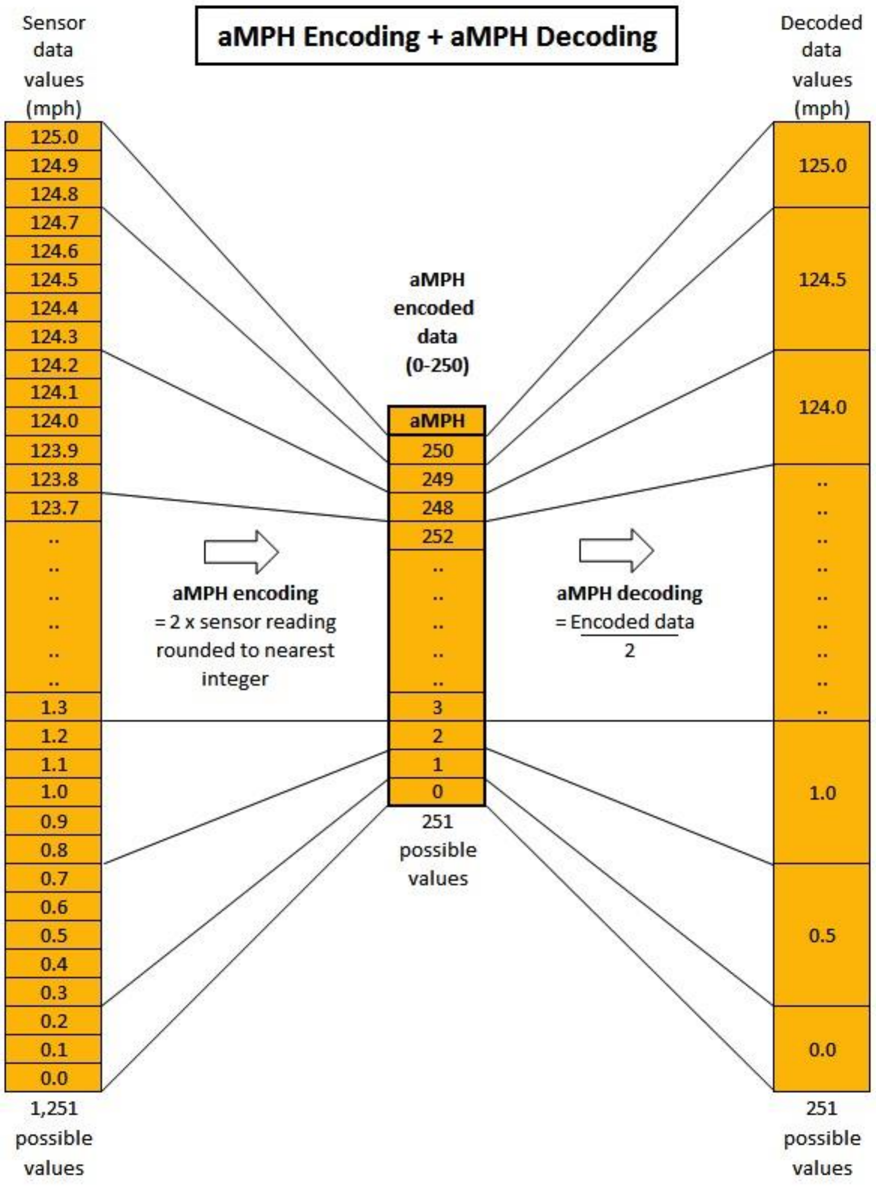


Figure 2

aMPH encoding allows speed readings from 0.0 - 125.0 mph to be stored as single 8 bit bytes with values ranging from 0 - 250. The precision has however been further reduced by the extra data compression from 0.4 mph (under *sMPH* encoding) to 0.5mph (under *aMPH* encoding).

To allow *aMPH* files to be distinguished from *sMPH* files they embedded the details of the "*aMPH* encoding/decoding" process (or *aMPH* profile) in the file. This would ensure that future users of the data files would automatically know the data inside was *aMPH* encoded and the correct way to recover (decode) the data, and should not mistakenly use the *sMPH* process.

The *aMPH* encoding and decoding process is shown schematically in Figure 2. Note that typically 5 different speed values (e.g. 0.3, 0.4, 0.5, 0.6 and 0.7 mph) are now encoded (and hence decoded) as the same value (0.5 mph). So although *aMPH* files can hold data for a 25% wider speed range than *sMPH*, the precision to which those speeds are recorded has now been reduced by 20%.

ProMPH

As technology advanced again it was decided that the system needed to be further adapted to allow speeds up to 150mph to be encoded and decoded. This was achieved by changing the encoding and decoding factor to 5/3 and they called this process *ProMPH* encoding.

ProMPH encoded reading = ROUND(5 x Sensor reading / 3)

A speed of 150.0 mph can therefore be encoded in an 8 bit byte with a value of 250. To recover the original data, the process is simply reversed. They called this process is *ProMPH* decoding.

Decoded reading = 3 x *ProMPH* encoded reading / 5

Note that using *ProMPH* encoding has increased the range of speeds that can be stored by 50% (relative to *sMPH*). But this reduces the precision to which speed is recorded by 33% to 0.6mph. Realising this reduction was becoming significant the engineers recommended that for high precision applications *ProMPH* data should be stored in double bytes (16 bit mode).

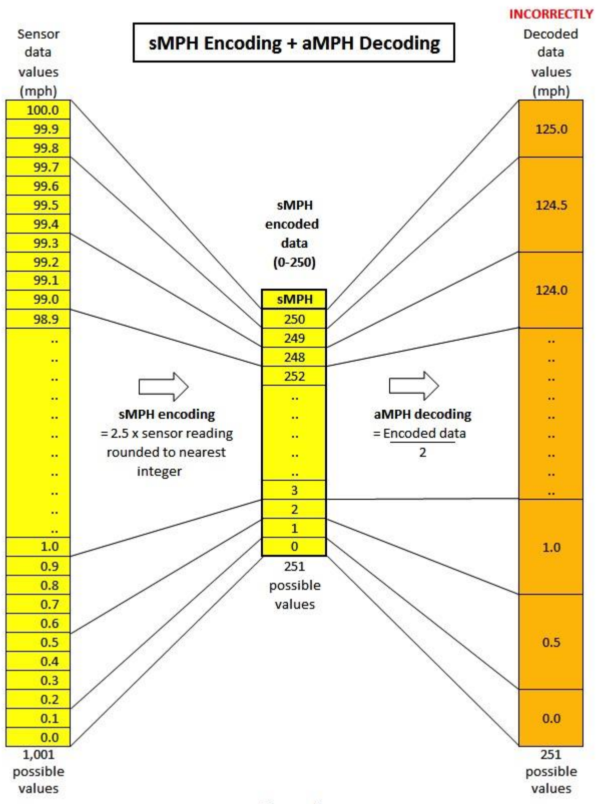
Analogy with *sRGB*, *aRGB* and *ProPhotoRGB*

If *sMPH*, *aMPH* and *ProMPH* are conceptually replaced by *sRGB*, *aRGB* and *ProPhotoRGB* and vehicle speed is replaced with richness of colour (faster speed = richer colour), it's possible to draw parallels between the process described above and those used to store image data. When an *sRGB* image file is created, the image sensor data is encoded using the *sRGB* standard and the *sRGB* profile information is (usually) embedded in the file so that the data can be correctly decoded automatically. Similarly the *aRGB* and *ProPhotoRGB* standards were devised to allow richer and richer colours to be stored in the file, in a similar way to which *aMPH* and *ProMPH* allow higher and higher speeds to be encoded.

The power of the analogy becomes clear when analysing what happens when files are decoded using a different standard to that used to encode the data. For example, what happens if *sRGB* image data is decoded using *aRGB* profile, or vice versa? These two examples are considered below.

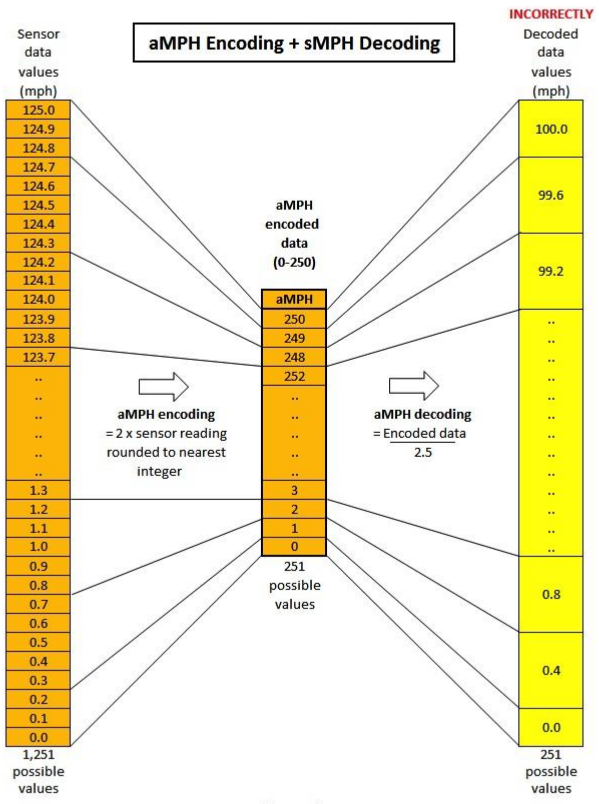
sRGB data decoded using _aRGB_ profile

This is analogous to using the *aMPH* process to decode speed data which was encoded with *sMPH*. The effects of this are shown in the Figure 3 below. Note how a true vehicle speed of 100mph has been incorrectly decoded as 125 mph. The decoded data indicates the vehicle is going faster than it really was. Therefore, by analogy, when *sRGB* data is decoded (rendered) using an *aRGB* profile, the colours will be displayed incorrectly and will appear richer than they should. The situation gets worse if *sRGB* files are rendered using a *ProPhotoRGB* profile as the colours will appear richer still.



sMPH Encoding + aMPH Decoding
Sensor data values (mph)
INCORRECTLY Decoded data values (mph)
sMPH encoded data (0-250)
sMPH encoding = 2.5 x sensor reading rounded to nearest integer
aMPH decoding = Encoded data / 2
1,001 possible values
251 possible values
251 possible values

Figure 3



aMPH Encoding + sMPH Decoding
Sensor data values (mph)
INCORRECTLY Decoded data values (mph)
aMPH encoded data (0-250)
aMPH encoding = 2 x sensor reading rounded to nearest integer
aMPH decoding = Encoded data / 2.5
1,251 possible values
251 possible values
251 possible values

Figure 4

aRGB data decoded using _sRGB_ profile

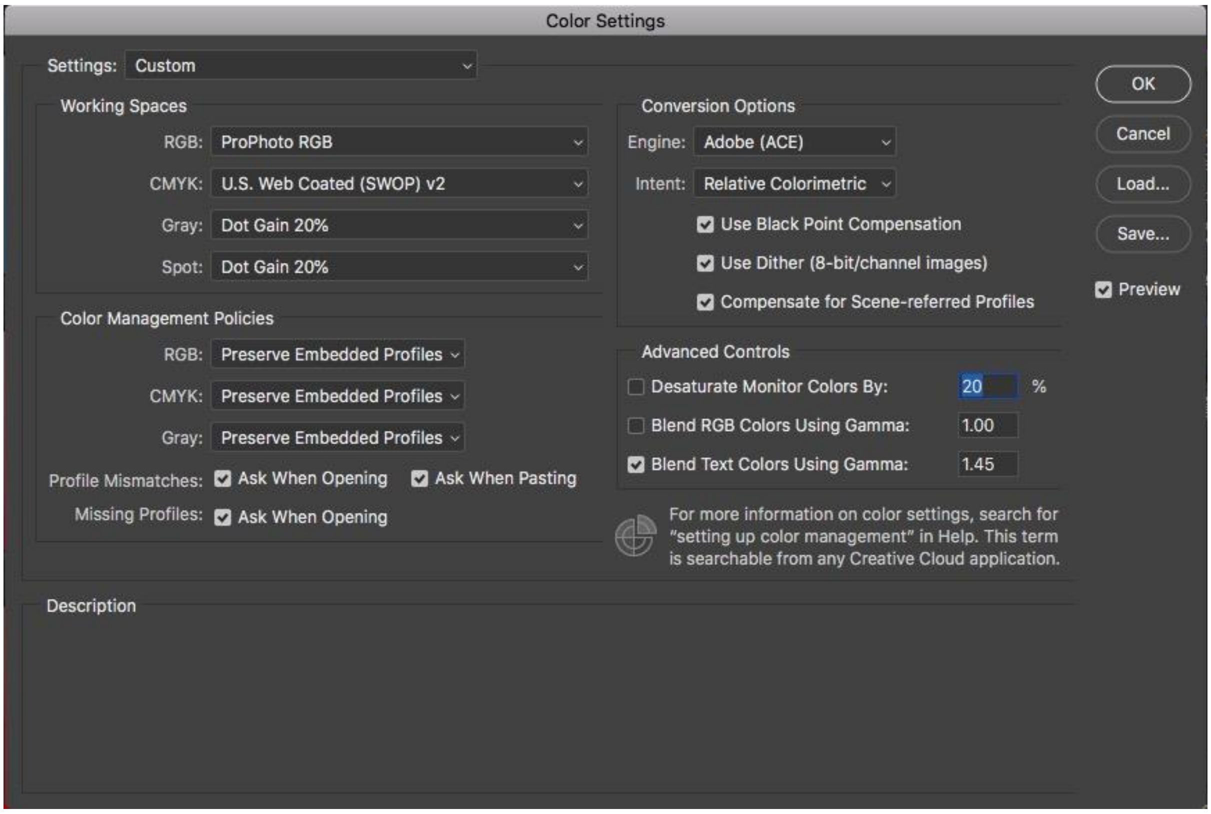
This is analogous to using the *sMPH* process to decode speed data which was encoded with *aMPH*. The effects of this are shown in the Figure 4 above. Note how a true vehicle speed of 125mph has been incorrectly decoded as 100 mph. The decoded data indicates the vehicle is going slower than it really was. Therefore, by analogy, when *aRGB* data is decoded (rendered) using an *sRGB* profile, the colours will be displayed incorrectly and will appear more muted than they should. The situation gets worse if a *ProPhotoRGB* file is rendered using a *sRGB* profile as the colours will appear even more muted.

Missing profiles, profile conversion and profile assignment

If files are correctly encoded and include the relevant profile, incorrect decoding errors shouldn't occur. But there are various scenarios where problems can arise, or compromises are required.

<u>Missing profiles</u>

If the profile used to encode the data is missing from the file, then it's impossible to know how the data should be decoded. Many systems assume *sRGB* as a default profile and will apply that profile to decode the data. This is fine if the data was encoded using *sRGB*, but if *aRGB* or *ProPhotoRGB* was used instead, then the decoded colours will be rendered incorrectly and will appear more muted than they should (analogous to the scenario in Figure 4). To avoid creating files with missing profiles it is advisable to embed the colour profile in the image file when saving. Photoshop provides a tick box to "Embed colour profile" when saving files in the file saving menu. Photoshop can also be set to give warnings when opening files with missing profiles and ask what to do. These settings are found in the Color Management Polices section of the Edit>Color Settings… menu. It is suggested that all boxes in this section should be ticked, as shown in Figure 5.


Color Settings

Settings: Custom

Working Spaces
RGB: ProPhoto RGB
CMYK: U.S. Web Coated (SWOP) v2
Gray: Dot Gain 20%
Spot: Dot Gain 20%

Color Management Policies
RGB: Preserve Embedded Profiles
CMYK: Preserve Embedded Profiles
Gray: Preserve Embedded Profiles
Profile Mismatches: ✓ Ask When Opening ✓ Ask When Pasting
Missing Profiles: ✓ Ask When Opening

Conversion Options
Engine: Adobe (ACE)
Intent: Relative Colorimetric
✓ Use Black Point Compensation
✓ Use Dither (8-bit/channel images)
✓ Compensate for Scene-referred Profiles

Advanced Controls
☐ Desaturate Monitor Colors By: 20 %
☐ Blend RGB Colors Using Gamma: 1.00
✓ Blend Text Colors Using Gamma: 1.45

For more information on color settings, search for "setting up color management" in Help. This term is searchable from any Creative Cloud application.

Description

Figure 5

<u>Profile Assignment</u>

Photoshop allows user to assign a profile (Edit>Assign profile…) to an image with a missing profile, or to overwrite the existing profile. This option must be used with caution. *Colours in the image will be rendered incorrectly if the profile assigned to an image is not the same as the one used to originally encode the image.* Assigning a different profile can be used creatively to make colours appear richer or more muted, but it may be more convenient to adjust the saturation or vibrance sliders instead as these allow more control. The "Convert to Profile..." option should be used to change an image from one profile to another without significantly altering the colours.

Profile Conversion

Photoshop allows users to convert (Edit>Convert to Profile…) images from one profile to another. This operation works as follows. Firstly the image data is decoded using the original profile, then (after some adjustment if required*) the data is encoded using the new profile. **Although the conversion process may have some effect on the colours in the image, there are numerous settings that can be used to control any adjustments that may be needed, as shown in the Conversion Options section of the menu shown in Figure 5. A detailed description of the various options is currently beyond the scope of this document, but there are some potential compromises to be aware of.*

- If a *ProPhotoRGB* image contains a very wide range of colours, then converting to a colour space with less range (e.g. *sRGB*) will cause some alteration in colour. Colours maybe clipped or de-saturated slightly in an attempt to "squeeze" the larger colour range into the smaller colour space. **This process is not reversible.** Converting a *ProPhotoRGB* image to *sRGB* loses colour information which **cannot be recovered** by converting from *sRGB* back *ProPhotoRGB*.

- A similar compromise occurs, but to a lesser degree when converting from *ProPhotoRGB* to *aRGB*, or from *aRGB* to *sRGB*. **Again the process is not reversible.**

Although these compromises may sound daunting a few key points should be remembered.

- The visual differences between correctly rendered s*RGB*, *aRGB* and *ProPhotoRGB* images are relatively small. Indeed, on consumer grade monitors and printers, they can be very difficult to spot. Even on professional grade monitors and printers the visual difference is often far less noticeable than the relative sizes of the colour spaces might suggest. Figure 6 gives an example. This Figure was produced by creating *ProPhotoRGB*, *aRGB* and *sRGB* versions of the same RAW image of a Passport Colour Checker Target and then correctly rendering them all in Photoshop and taking a screenshot. Obviously there are limitations in this comparison as all images have been rendered on a monitor. Therefore, to overcome this limitation I have posted the jpeg files for each image on-line at https://postimg.cc/gallery/1lm43xh4m/ The images can be downloaded and compared on screen or as printouts.

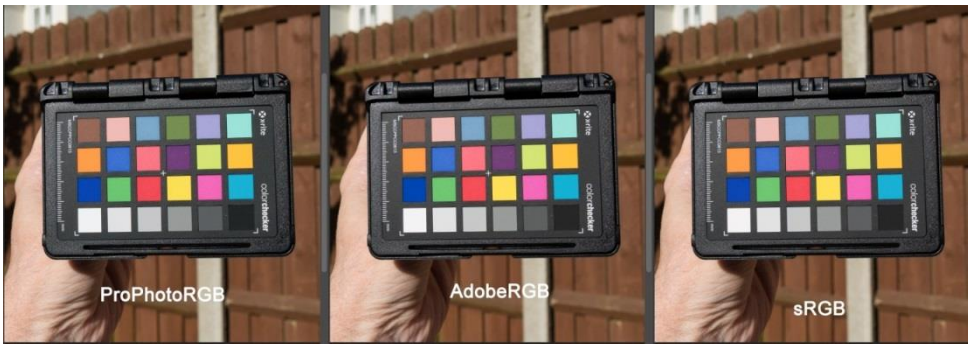


Figure 6

- The visual differences between incorrectly rendered images are much more noticeable and clearly visible on screen, as illustrated by the Figure 7 below. This figure was produced by creating *ProPhotoRGB*, *aRGB* and *sRGB* versions of a RAW image of a Passport Colour Checker Target and then assigning the wrong profiles to some of them. The central image is a screenshot of the *sRGB* image rendered in Photoshop using the correct *sRGB* profile. The surrounding (incorrectly rendered) images are screenshots of the *sRGB*, *aRGB* and *ProPhoto* images rendered in Photoshop with the wrong profile is assigned, as indicated. Note that the differences are now much more obvious than those shown in Figure 6. The jpeg files for each of these images are also available on-line at the URL above.

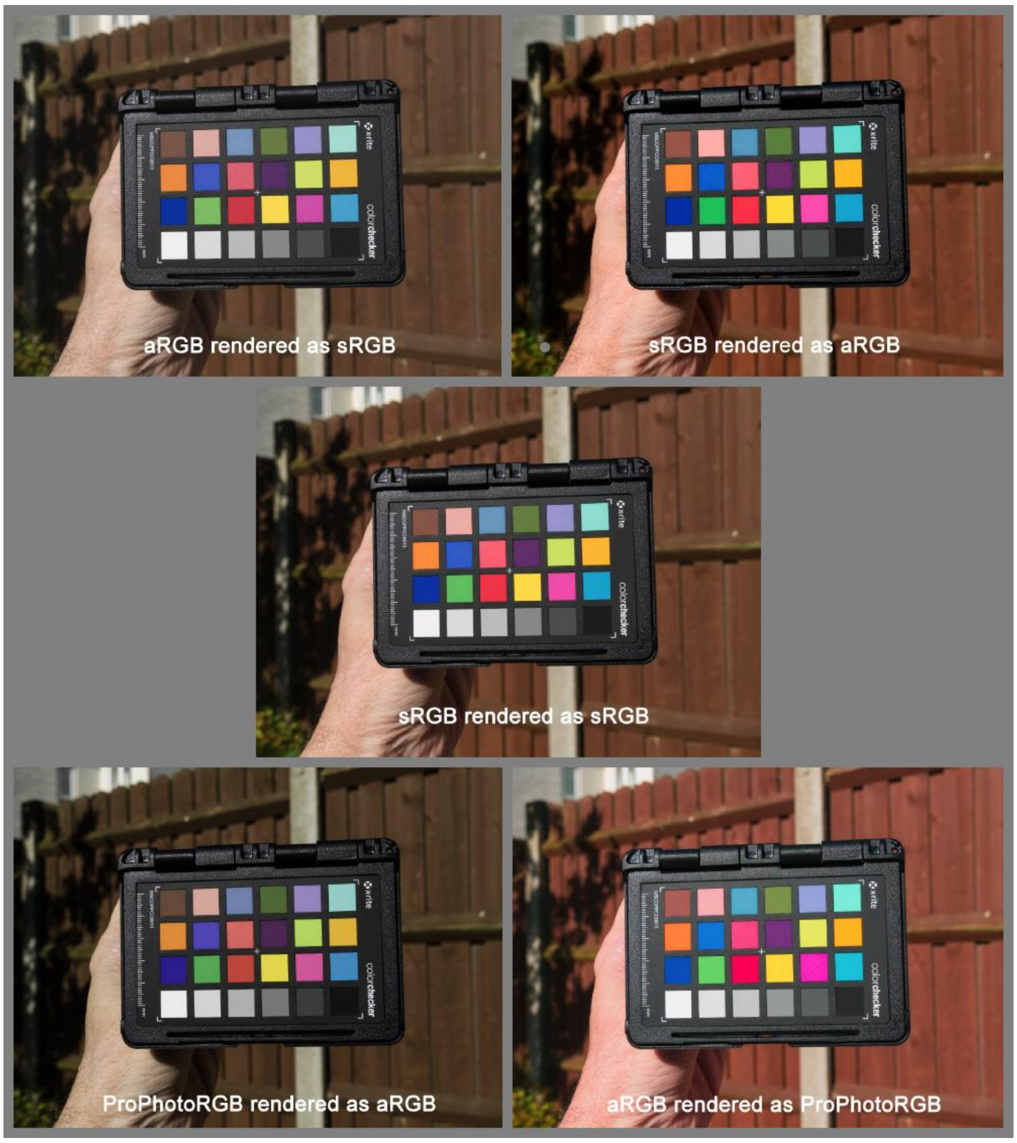


Figure 7

Synthesised images

Some of the images on the internet, which are used to demonstrate the differences between each colour spaces, are produced by generating various spectrum gradients in *sRGB*, *aRGB* and *ProPhotoRGB* workspaces. These images need to be interpreted with caution as they contain a large range of mathematically produced colours which extend right to the edges of each colour space, to deliberately emphasise the differences between these colour spaces. **Such colours can lie outside the range that typical digital cameras are sensitive to, or appear in normal scenes**. The spectrum gradient data they contain is not encoded (converted) to suit a particular colour space. The same numerical RGB values are simply inserted into the image and then decoded using whatever colour space profile is assigned to the image. This is analogous to filling the vehicle speed data file with values from 0-250 and then examining how each decoding method (*sMPH*, *aMPH* or *ProMPH*) interprets this data.

Examples of a spectrum gradient crossed with a luminosity gradient (often referred to as a Granger chart) created in each colour space are shown in Figure 8. (16 bit TIFF versions of these images are also available online at the same URL). There are clear visible differences between them.

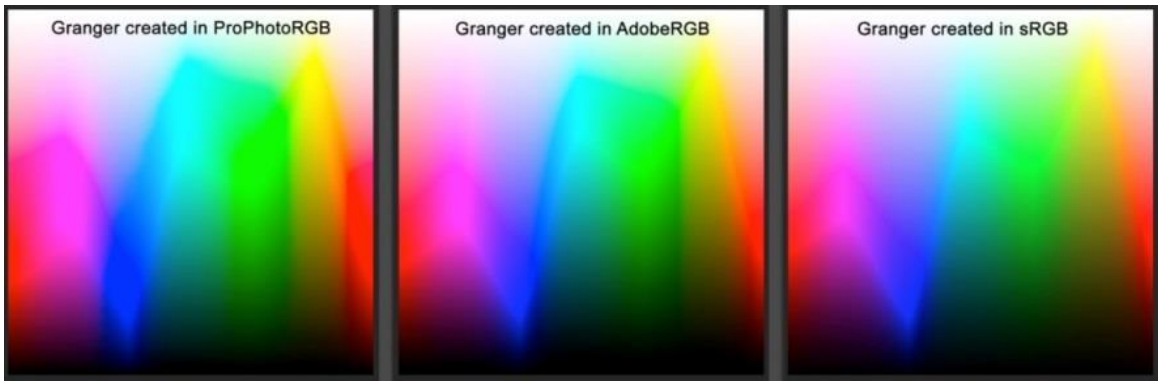


Figure 8

Examples of simple spectrum gradients created in each colour space are shown in Figure 9. (16 bit TIFF versions of these images are also available online at the same URL). Again, there are clear visible differences between them.

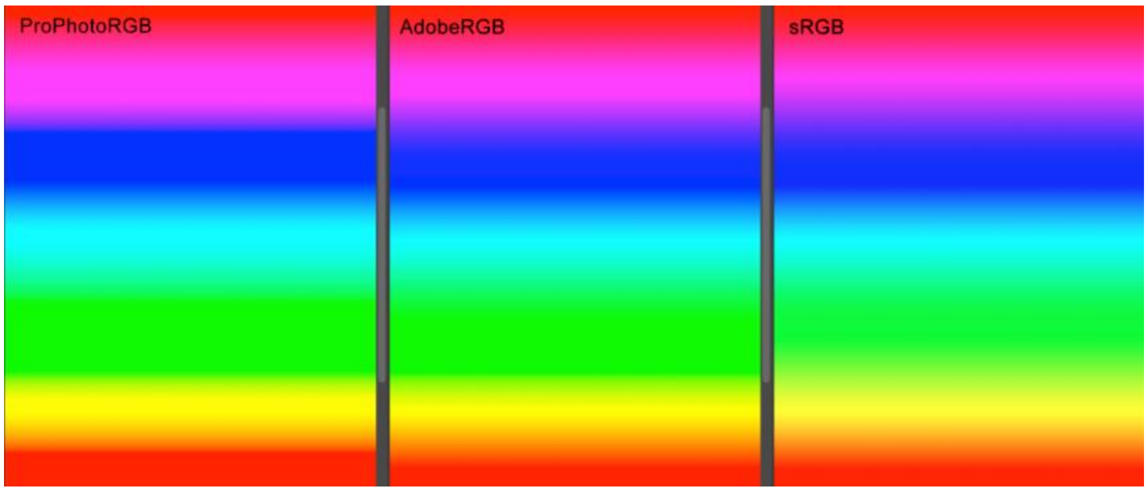


Figure 9

Comparisons between synthesised images like these are often used to justify the decision to use ProPhotoRGB.

However…. the differences between digital camera images of real scenes, taken in RAW format, and then processed in each of the colour spaces and correctly rendered will be less noticeable, especially when viewed or printed on consumer grade monitors and printers. This is illustrated by the following images of a real spectrum created by sunlight passing through a prism and projected onto a white sheet of paper to produce a saturated sweep of natural colours. The RAW image was processed to produce *sRGB*, *AdobeRGB* and *ProPhotoRGB* versions and saved as 16 bit TIFs at the same URL. Figure 10 shows a montage of the three images.

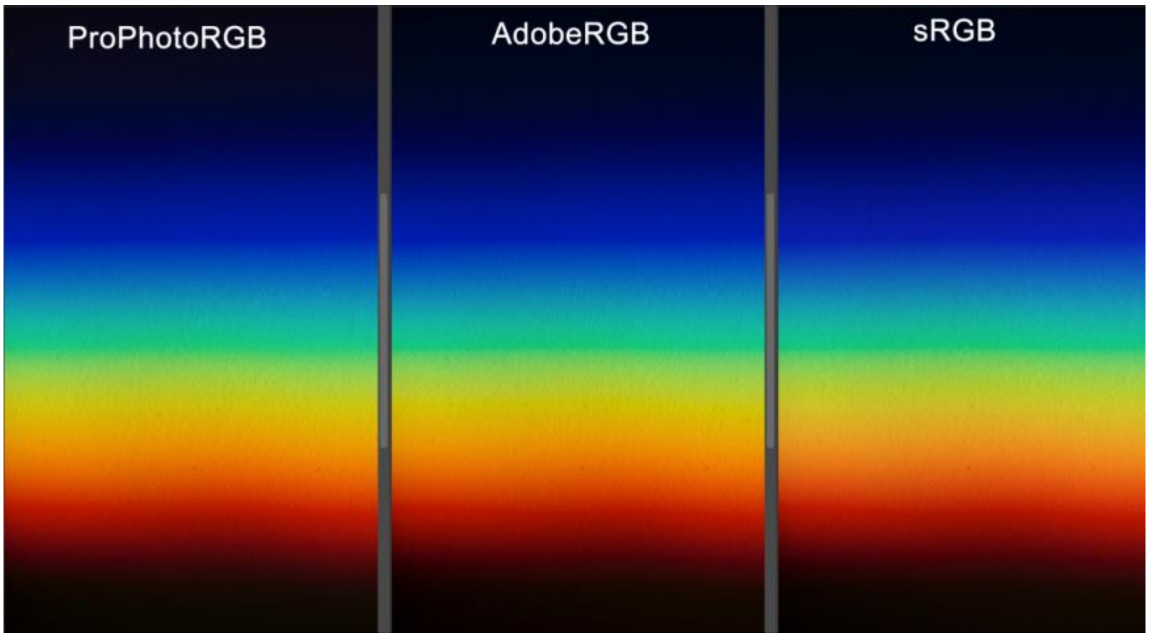


Figure 10

On my screen, and when printed on my printer, the three files are virtually indistinguishable. Perhaps the differences are more noticeable on better monitors or printers (the 16 bit TIFF files can be downloaded from the same URL to make your own comparisons). Maybe I need a more demanding (but still realistic) test image? (I'm working on that!!)