

Re: Saving three 16 bit RGB values in 32 bits::

Source: <http://sci.tech-archive.net/Archive/sci.image.processing/2005-09/msg00049.html>

- *From:* "Roger Hamlett" <rogerspamignored@xxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Fri, 09 Sep 2005 09:31:28 GMT
-

"Wenny Macura" <wmacu@xxxxxxxxxx> wrote in message
[news:gBaUe.166628\\$wr.79983@xxxxxxxxxx](mailto:news:gBaUe.166628$wr.79983@xxxxxxxxxx)

- > I have been fiddling with large 16 bit images for a while.
- > and have been trying to save the image allocation space on the disk.
- >
- > The following algorithm is the result of the struggle :-)
- > Any comments ??

Provided you are prepared to accept the loss of accuracy involved, then of course you can do a conversion like this. However the loss is severe... You are wasting some of the numeric range available, by multiplying by 100000. 65536, gives the tightest packing.

But the key is that you will lose data, and with the formula shown, most of this will be from the blue, when large values are held in the red. The total number of possible values that can exist for the RGB triplet, is not the 'sum' of the numbers, but the product.

Take the example, of a value of 2 in the blue, with a value of 65535 in the red, with an intermediate value (say 32767) in the green. Your formula gives:

$$65535*100000+32767+2/100000 = 6553532767.00002$$

Now a 23bit mantissa allows 2^{23} possible values. 0 to 8388607. Just under 7 digits containing data (This is why the accuracy of single precision numbers like this is normally referred to at 6 3/4 digits). The exponent scales the incoming number to make best use of these digits, normally by finding the first '1' starting from the front of the number, and then not storing this, but storing the 23 subsequent bits (to effectively give 24 bits held). This improved the accuracy to just over 7 digits.

Now for the example given, the normal 32bit FP format, will be stored (in hex) as 4FC34F7D. This converts back to a decimal value of 6553532900.00000. Converting this back to RGB, gives the correct 'R' value (65535), a green value of 32900, but complete loss of the blue data. A change of one bit in the 'blue' in the format given, would change between 6553532767.00000 and 6553532767.00001. A change of one and a half parts in 10^{15} . This would require a 47bit mantissa to reproduce accurately...

- > Saving three 16 bit RGB values in 32 bits:
- >
- > The floating point mantissa is defined as 23 bit number.
- > To sum of three 16 bit numbers requires at most 18 bits.

Re: Saving three 16 bit RGB values in 32 bits::

```
>
> Since the image RGB values are independent of each other
> and the individual color component values are in the range of 0 to
> 65535,
> they can be stored as a single 32 bit floating point number.
>
> This in turn will result in 33% decrement in storage allocation.
>
> The RGB to double algorithm:
>
>  $Fno = R * 100000.0 + G + B / 100000.0$ 
>
> double to RGB
>
>  $R = (int) Fno / 100000.0;$ 
>  $G = (int) ( Fno - R * 100000.0 )$ 
>  $B = (int) (round\ up) ( ( Fno - R * 100000.0 - G ) * 100000.0 ) )$ 
>
> //CODE BEGIN:
>
> // * Copyright (c) 2000–2005
> // * Wenny Macura, Mac–Eng All rights reserved.
> /*
> * double Utility::RGB_ToDBL( int R, int G, int B)
> *
> * used to convert three 16 bit RGB values ( 48 bits )
> * to one double ( 32 bits ) value.
> * this results in saving 2 bytes per pixel or 33% of the image data
> * write the image as CMYK
> * NOTE:
> * the R,G,B values must be less then 100000.
> */
> #define M_CONST 100000.0
>
> double Utility::RGB_ToDBL( int R, int G, int B)
> {
> static double result;
> result = R * M_CONST;
> result += G ;
> result += B /M_CONST;
> return result;
> }
>
> /*
> * int *Utility::DBL_ToRGB( double Value)
> *
> * used to convert double representing RGB values
> * to integer RGB values
> * the return values are:
> * RetVal[0] ~ R
> * RetVal[1] ~ G
```

Re: Saving three 16 bit RGB values in 32 bits::

Re: Saving three 16 bit RGB values in 32 bits::

```
> *RetVal[0] ~ B
> */
>
> int *Utility::DBL_ToRGB( double Value)
> {
> static int Color[4];
> static double Tol = 0.5/M_CONST;
> double mTmp;
> int i_val;
> i_val = (int) ( Value/M_CONST );
> Color[0] = i_val;
> mTmp = Value - (double) i_val * M_CONST;
> i_val = (int) mTmp;
> Color[1] = i_val;
> mTmp -= i_val;
> Color[2] = (int) ( (Tol + mTmp) * M_CONST);
> return Color;
> }
> //CODE END:
>
> Wenny Macura
> wmacu@xxxxxxxxxx
```

Hint. What you post here `_will_` work, but you are using a `_double_` to store the numbers. A 'double' requires 8 bytes to hold (64bits, not 32), giving the larger mantissa required, but resulting in more storage space, not less...

Best Wishes

• **Follow-Ups:**

◆ **[Re: Saving three 16 bit RGB values in 32 bits::](#)**

◇ From: Wenny Macura

• **References:**

◆ **[Saving three 16 bit RGB values in 32 bits::](#)**

◇ From: Wenny Macura

- Prev by Date: **[Re: Saving three 16 bit RGB values in 32 bits::](#)**
- Next by Date: **[Re: Need help: Amber Galileo IR camera \(radiance HS\)](#)**
- Previous by thread: **[Re: Saving three 16 bit RGB values in 32 bits::](#)**
- Next by thread: **[Re: Saving three 16 bit RGB values in 32 bits::](#)**
- Index(es):
 - ◆ **[Date](#)**
 - ◆ **[Thread](#)**