# High-speed Conversion of Floating Point Images to 8-bit (online ID 0265)

Bill Spitzak, Digital Domain *

## Introduction

Most rendering software today destroys precise lighting and shading calculations by doing an inaccurate and low-quality conversion to bytes for display devices. This sketch presents a technique[1] to quickly convert floating point data to a screen image while preserving the correct brightness levels and original detail.

## Screen Gamma

A common misconception of how monitors work is the amount of light emitted by the screen is in linear proportion to the number placed into the image buffer, and that multiplying floating point pixel colors by 255 will produce the screen image. However a power function is the correct mapping and has been standardized[1]. The proper way to convert a floating point brightness is to use the inverse of sRGB and multiply that by 255:

$$to\_byte(x) = 255\times\begin{cases} 12.92x & \text{if } x \le .04045/12.92 \\ 1.055x^{1/2.4} - .055 & \text{if } x > .04045/12.92 \end{cases}$$

It may also be desirable to present high dynamic range image data by compressing the high end. There are many ways to do this. One possibility is:

$$to\_byte\_compressed(x) = \begin{cases} to\_byte(x) & \text{if } x \le .5 \\ to\_byte(1 - 1/4x) & \text{if } x > .5 \end{cases}$$

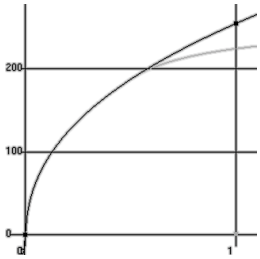


Figure 1: $to\_byte$ (black) and $to\_byte\_compressed$ (gray)

Since the domain of $to\_byte\_compressed()$ is from zero to infinity we would like our conversion to be able to handle this.

## Look-up Table

A look-up table indexed by the top 16 bits of the floating point numbers is used to avoid the need for conditional expressions or calls to pow(). This table, however, is quite large as there are 65536 entries. If this is a concern the table can be made smaller by adding range checks to the algorithm. Limiting the table to $.04045/12.92 \le x \le 1$ (and linearly interpolating outside that range) would reduce the number of entries to 1065.

To fill the table a floating point number is constructed with the correct high 16 bits and the lower bits set to 0x8000, which is halfway through the range that maps to that table entry. This requires care on processors that trap NaN or otherwise fail on some bit patterns in floating point. Some compilers may also have trouble loading arbitrary bit patterns into floating point registers. The floating point value is then converted to a byte, rounded and clamped, and stored in the table.

How accurate is this table? The distance between output values of the table is $\Delta y(x) = 2^{\lfloor \ln x/\ln 2 \rfloor - 7} d(to\_byte)/dx$. The maximum $\Delta y(x)$ when $0 \le x < 1$ is at .5 and is $\approx .656$. Thus every possible byte value is represented, and if values are calculated at the center of the table entries the maximum error is less than $1/3$ of an output value. At common image levels the error is smaller, at 18% gray the maximum error is .15.

## Error Diffusion

Floating point data can represent fine gradations in color that are lost if the values are simply rounded to the nearest byte value. An error diffusion algorithm is used to solve this. Most algorithms are designed to reduce the image to very few levels and involve the weighted distribution of "error" to four or more neighboring pixels[2]. With 256 output levels, it is sufficient to transfer all the error to the next pixel. This allows the conversion of each scan line to be independent and is of course very simple.

The error information is stored by making the table contain 16-bit words equal to $256 \times \lfloor to\_byte(x), 255 \rfloor$. The top 8 bits are then the byte value lower than the desired result and the lower 8 bits are the error offset. Starting the error at 0x80 results in rounding to the nearest byte.

This simple dithering would produce vertical lines in solid shades and horizontal ramps. This was solved by dithering in both directions away from a randomly chosen starting location. Solid areas of black or white must not set the error back to zero or vertical lines will appear in the regions after those.

## Preserving 8-bit Data

It is desirable to have 8-bit files converted to floating point and back to emerge unchanged. If this was not done then repeatedly reading and writing an 8-bit image would gradually deteriorate it. To enforce this the table is modified: each 8-bit pixel value is converted to floating point and the resulting table location is replaced with the exact integer. Since the error is zero, no dithering happens when writing the data back out. This reduces the accuracy of the table by spacing the samples somewhat irregularly. However no entries are out of order and only 1.6% of the table entries are changed. The result is that only a small amount of noise is added to the data.

## Conclusions

This algorithm has proven to be fast enough for real-time preview of compositing results. It has also proven that trivial calculations of blurs and exposure changes in linear floating point produce beautiful and photographically correct results when converted this way. This technique promises to allow future software to produce physically accurate pictures quickly and easily.

## References

[1] International Electrotechnical Commission. The sRGB Specification. IEC 61966-2-1.

[2] Floyd, R., and Steinberg, L. An Adaptive Algorithm for Spatial Gray Scale. *Society for Information Display 1975 Symposium Digest of Technical Papers*, page 36, 1975

*spitzak@d2.com

[1] GPL code is available at http://www.cinenet.net/ spitzak/conversion