PNG-GROUP
PNG-Sci-Vis

PNG Development Group
png-list@dworkin.wustl.edu
03 Feb 1997

PNG Scientific Visualization Chunks, Version 19970203

File: png-scivis-19970203.txt

Status of this Memo

This document is a specification by the PNG development group.

Comments on this document can be sent to the PNG specification maintainers at one of the following addresses:

```
png-list@dworkin.wustl.edu.
png-group@w3.org
png-info@uunet.uu.net
```

Distribution of this memo is unlimited.

At present, the latest version of this document is available on the World Wide Web from

```
ftp://swrinde.nde.swri.edu/pub/png/documents/.
```

Notices

Abstract

This document describes some special-purpose chunk types, of particular interest to the scientific visualization community, that have been registered for use in various PNG (Portable Network Graphics) and related multi-image applications. The chunks that have been registered to date are: "`pCAL`".

Contents

1 Introduction

The chunks described here were registered by the PNG development group as public chunks for use by the scientific visualization community. They are not considered to be of wide enough applicibility to be documented in the PNG specification [PNG] or in the PNG Special-Purpose Public Chunks document [PNG-EXT]. No general-purpose decoder is required or expected to implement these chunks. The format of these chunks has been "frozen" and is not subject to change. Comments on these chunk specifications, and new proposals for additional chunk types, should be sent to the PNG specification maintainers at one of the addresses mentioned above. The basic PNG specification is available from the PNG ftp archive at

`<URL:ftp://swrinde.nde.swri.edu/pub/png/documents/>.`

2 Summary of Sci-Vis Chunks

This table summarizes some properties of the chunks described in this document.

```
chunk name      Multiple   Ordering
                   OK?     constraints

pCAL               No      Before IDAT
```

3 Chunk specifications

This chapter defines the registered Sci-Vis PNG chunks.

3.1 `pCAL` Calibration of pixel values

When a PNG file is being used to store physical data other than color values, such as a two-dimensional temperature field, the "`pCAL`" chunk can be used to record the relationship (mapping) between stored pixel samples, original samples, and actual physical values. The "`pCAL`" data might be used to construct a reference color bar beside the image, or to extract the original physical data values from the file. It is not expected to affect the way the pixels are displayed. Another method should be used if the encoder wants the decoder to modify the sample values for display purposes.

The "`pCAL`" chunk's contents are a text string (purpose) that names the equation, original sample limits "`X0`" and "`X1`", an equation type, a text string (unit), and a set of parameters for the equation:

```
n bytes:  Purpose (Latin-1 text)

1 byte:   null separator

4 bytes:  X0 (signed integer) Lower limit of original
          sample range.  This original sample value is
          mapped to the stored sample value 0.

4 bytes:  X1 (signed integer) Upper limit of original
          sample range.  This original sample value is
          mapped to the stored sample value
          (2^sample_depth - 1).  Must not equal X0.

1 byte:   Equation type (unsigned integer).
     0:   Linear mapping
     1:   Base-e exponential mapping
     2:   Arbitrary-base exponential mapping
     3:   Hypberbolic mapping

1 byte:   N (unsigned integer), number of parameters.

u bytes:  Unit (Latin-1 string).  Symbol or description of
          the unit, eg. K, Population Density, MPa, etc).
          A zero-length string can be used if the data is
          dimensionless.

1 byte:   null separator

p0 bytes: P0 (ASCII text).  First parameter, a real number
          written as a text floating-point value [link to
          Floating-Point Values in PNG extensions document]

1 byte:   null separator

p1 bytes: P1 (ASCII text).  Second parameter.

etc.
```

There is no null separator after the final parameter (or after the "Unit" field, if N=0). The number of parameters present must agree with the "N" field and must be correct for the specified "equation_type".

The "purpose" identifies the equation, which can permit applications or people to choose the appropriate one when more than one "pCAL" chunk is present (this could occur in a multiple-image file, but not in a PNG file). The "purpose" string must follow the format of a "tEXt" keyword, i.e. 1-79 printable Latin-1 characters. One way the "purpose" field could be used s to identify the system of units. A "purpose" string such as "SI" or "English" could be used in the "pCAL" chunk as well as in other chunk types, to permit a decoder to select an appropriate set of chunks based on the contents of their "purpose" fields.

The "pCAL" chunk defines two mappings:

- A mapping from the stored samples, which are unsigned integers in the range [0..M], where "M==2^sample_depth-1", to the original samples, which are signed integers. "X0" and "X1", together with the "sample_depth" for the image, define this mapping.
- A mapping from the original samples to the physical values, which are usually real numbers. The remaining "pCAL" chunk parameters define this mapping.

Encoders will usually set "X0 = 0" and "X1 = M" to indicate that the stored samples are equal to the original samples. Note that "X0" is not constrained to be less than "X1", and neither is constrained to be positive, but they must be different from each other.

The mapping algorithms are

```
original_sample =
   (stored_sample * (X1 - X0) + M/2) / M + X0
```

using integer arithmetic, where (a/b) means (integer(floor(real(a)/real(b)))). Note that this is the same as the "/" operator in the C programming language when "a" and "b" are nonnegative, but not necessarily when "a" or "b" is negative.

```
if equation_type = 0 then
physical_value = P0 + P1 * original_sample/(X1-X0)

else if equation_type = 1 then
physical_value =
    P0 + P1 * EXP(P2 * original_sample/(X1-X0))

else if equation_type = 2 then
physical_value = P0 + P1 * P2^(original_sample/(X1-X0))

else if equation_type = 3 then
physical_value =
    P0 + P1*SINH(P2*(original_sample - P3)/(X1-X0))
```

using floating point arithmetic, in which

- EXP(x) is the exponential function, e^x, in which “e” is the base of natural logarithms (approximately 2.718282)
- SINH(x) is the hyperbolic sine of x,

```
    SINH(x) = 0.5 * ( EXP(x) - EXP(-x) )
```

To map an original sample to a stored sample, the function is:

```
stored_sample =
    ((original_sample - X0) * M + (X1 - X0) / 2) / (X1 - X0)

    (limited to the range [0..M])
```

This mapping is lossless and reversible when `(ABS(X1-X0) <= M)` and the original sample is in the range [X0..X1]. If `(ABS(X1-X0) > M)` then there can be no lossless reversible mapping, but the functions provide the best integer approximations to floating-point affine transformations.

For color_types 2, 3, and 6, the mapping algorithms are applied independently to each of the color sample values. In the case of color type 3 (indexed color), the mapping refers to the RGB samples and not to the index values.

Linear data can be expressed with equation_type 0.

Pure logarithmic data can be expressed either with either equation_type 1 or equation_type 2:

```
X0 = 0                              X0 = 0
X1 = M                              X1 = M
Equation_type = 1                  Equation_type = 2
N  = 3              or with         N  = 3
P0 = 0                              P0 = 0
P1 = min                            P1 = min
P2 = LOGe(max/min)                  P2 = max/min
```

Equation_types 1 and 2 are functionally equivalent; both are defined because authors may find one or the other more convenient.

Using equation type 3, floating point data in the range [-v0..v1] can be reduced (with loss) to a set of integer samples such that the resolution of the stored data is roughly proportional to its magnitude. For example, floating point data ranging from -10^31 to 10^31 (the usual range of floating point numbers on 32-bit machines) can be represented with

```
X0 = 0
X1 = 65535
Equation_type = 3
N  = 4
P0 = 0.0
P1 = 1.0e-30
P2 = 280.0
P3 = 32767.0
```

The resolution near zero is about 10^-33, while the resolution around +/-10^31 is about 10^28. Everywhere the resolution is about 0.4 percent of the magnitude.

Applications should use double precision arithmetic (or take other precautions) while performing the mappings for equation types 1, 2, and 3, to prevent overflow of intermediate results when the parameter "`P1`" is small and the exponential or power functions are large.

If present, the "`pCAL`" chunk must appear before the first "`IDAT`" chunk. Only one instance of the "`pCAL`" chunk is permitted in a PNG stream.

4 References

[PNG]

Boutell, T., et. al., PNG (Portable Network Graphics Format Version 1.0), RFC 2083 <`URL:ftp://ds.internic.net/rfc/rfc2083.txt`>, also available at <`URL:ftp://swrinde.nde.swri.edu/pub/png/documents/`>. This specification has also been published as a W3C Recommendation, which is available at <`URL:http://www.w3.org`>.

[PNG-EXT]

PNG Special-Purpose Public Chunks, <`URL:ftp://swrinde.nde.swri.edu/pub/png/documents/`>.

5 Security considerations

The normal precautions (see the Security considerations section of the PNG specification) should be taken when displaying text contained in the "purpose", "unit", and parameter fields of the "`pCAL`" chunk.

Applications must take care to avoid underflow and overflow of intermediate results when converting data from one form to another according to the "`pCAL`" mappings.

6 Appendix: Sample code

This appendix provides some sample code that can be used in encoding and decoding the PNG Sci-vis chunks. It does not form a part of the specification. In the event of a discrepancy between the sample code in this appendix and the chunk definition, the chunk definition prevails.

6.1 pCAL

This section provides some sample code for the PNG pCAL chunk, written in the "C" Programming Language. The "`pcal_encode`" subroutine takes an array of floating point numbers and produces an array of

16-bit samples that can be stored as grayscale PNG pixels. The "pcal_make_lut" procedure produces a lookup table that can be used to extract the original physical values from PNG pixels.

```
/* Sample code for the PNG pCAL chunk */
#include <math.h>
/* Math.h supplies double precision exp(), pow(), and
 * sinh() functions.  If your math.h doesn't supply sinh(),
 * use sinh(x)=(exp(x)+exp(-x))/2.
 */
unsigned short limit(long low, double x, long high)
{
    if (low < high){
       if(x < low)return low;
       else if (x > high) return high;
       else return x;
    }
    else {
       if(x < high)return high;
       else if( x > low) return low;
       else return x;
    }
}
int pcal_encode (unsigned short *stored_sample, long n,
 float *physical_value, unsigned int m, int equation_type,
 long x0, long x1, float *p)

/* returns 0 (success)
 *        -1 (error, x0==x1)
 *        -2 (unknown equation type)
 * input:
 *         n: number of samples
 *         physical_value[0..n-1]
 *         m: PNG sample depth
 *         equation_type: from pCAL chunk
 *         x0, x1: stored sample to original sample mapping
 *         p[]: equation parameters, from pCAL chunk
 * output:
 *         stored_samples[0..n-1] (caller must allocate space)
 */

{
    double d,dm;  /* force double precision arithmetic */
    long isample, osample, k;
    d=x1-x0;
    dm=m;
```

```
    if (x1 != x0) {
        if(equation_type == 0){
            for (k=0; k<n; k++) {
               isample=.5+d*(physical_value[k] - p[0])/p[1];
               osample=limit(x0, isample, x1);
               stored_sample[k]= floor(((osample-x0)*dm
                                  +floor(d/2))/d);
            }
        }

        else if(equation_type == 1){
            for (k=0; k<n; k++) {
               isample= .5+d*(log(physical_value[k]
                          - p[0])/p[1])/p[2];
               osample=limit(x0, isample, x1);
               stored_sample[k]= floor(((osample-x0)*dm
                                  +floor(d/2))/d);
            }
        }

        else if(equation_type == 2){
            double factor;
            factor=d/log(p[2]);
            for (k=0; k<n; k++) {
               isample=.5+log((physical_value[k]
                         -p[0])/p[1])*factor;
               osample=limit(x0, isample, x1);
               stored_sample[k]= floor(((osample-x0)*dm
                                  +floor(d/2))/d);
            }
        }

        else if(equation_type == 3){
            for (k=0; k<n; k++) {
               isample= .5+p[3]+d*asinh((physical_value[k]
                         -p[0])/p[1])/p[2];
               osample=limit(x0, isample, x1);
               stored_sample[k]= floor(((osample-x0)*dm
                                  +floor(d/2))/d);
            }
        }
        else return (-2); /* ERROR, unknown equation type */
    }
    else return (-1); /* ERROR, x0 == x1 */
    return (0);
}

int pcal_make_lut (float *physical_value, unsigned int m,
             int equation_type, long x0, long x1, float *p)
```

```
/* returns 0 (success)
 *        -1 (error, x0==x1)
 *        -2 (unknown equation type)
 * input:
 *         m: PNG sample depth
 *         equation_type: from pCAL chunk
 *         x0, x1: stored sample to original sample mapping
 *         p[]: equation parameters, from pCAL chunk
 * output:
 *         physical_value[0..m] (caller must allocate space)
 */

{
    double d, dm; /* force double precision arithmetic */
    long sample, osample;
    d=x1-x0;
    dm=m;

    if (x1 != x0) {
        if(equation_type == 0){
            for (sample=0; sample<=m; sample++){
               osample=floor((sample*d+floor(dm/2))/dm) + x0;
               physical_value[sample] = p[0] + p[1]*osample/d;
            }
        }

        else if(equation_type == 1){
            for (sample=0; sample<=m; sample++){
               osample=floor((sample*d+floor(dm/2))/dm) + x0;
               physical_value[sample] = p[0]
                        + p[1]*exp(p[2]*osample/d);
            }
        }
        else if(equation_type == 2){
            for (sample=0; sample<=m; sample++){
               osample=floor((sample*d+floor(dm/2))/dm) + x0;
               physical_value[sample] = p[0]
                        + p[1]*pow(p[2],osample/d);
            }
        }
```

```
        else if(equation_type == 3){
            for (sample=0; sample<=m; sample++){
               osample=floor((sample*d+floor(dm/2))/dm) + x0;
               physical_value[sample] =
                  p[0] + p[1]*sinh(p[2]*(osample-p[3])/d);
            }
        }
        else return (-2); /* ERROR, unknown equation type */
    }
    else return (-1); /* ERROR, X0 == X1 */
    return (0);
}
```

7 Appendix: Rationale

This appendix gives the reasoning behind some of the design decisions in the PNG Sci-vis chunks. It does not form a part of the specification.

7.1 pCAL

This section gives the reasoning behind some of the design decisions in the "`pCAL`" chunk. It does not form a part of the specification.

Redundant equation types

Equation_types 1 and 2 seem to be equivalent. Why have both?

- We don't want to force people to do the exponentiation using log() and exp(), since pow() may provide better accuracy in some floating point math libraries. Thus we have equation type 2, which uses pow(p2, value/(X1-X0)). We also don't want to force people using base-10 logs to store a sufficiently accurate value of log(10) in the pCAL chunk.
- When the base *is* e, we don't want to force people to encode a sufficiently-accurate value of e in the "`pCAL`" chunk, or to use pow() when exp() is sufficient. Thus we have equation type 1.

What are X0 and X1 for?

The X0,X1 mechanism is useful for three things.

- First, the {X0,X1} mechanism provides a way to map stored values back to digitizer output and second, as a way to label a limited range image, and to help recover the original data, losslessly, when the original range is not a power of two. Sometimes the digitized values do not have a range that fills the full depth of a PNG. For example if the full range of an image is 0-800 and one wants to stretch the contrast and increase the bit depth to 16 bits, one can do it and leave a record that this has been done, by setting x0=0 and x1=800. The reason to do this is to know the original granularity in the data which is useful at least as a lower limit on the error in the data.

- A second use is that the {X0,X1} parameters can be used to mark the limited range of an original image that is contained in the current image. In this scenario there is an original image that has important detail in a limited intensity range. The limited range is extracted and contrast stretched to fill the PNG bit depth. There is no way in pCAL to distinguish these cases but at least pCAL can store the relevant information. People can distingush these cases by including a text chunk labelling the image as original or derived.
- Finally, X0 is necessary in several of the equation types to allow a zero offset. Without it, we'd need an extra parameter in equation types 1 and 2. So eliminating the notion of an "original sample" wouldn't really simplify implementations—and having it allows lossless recovery in the cases where there *is* an intermediate exact integer representation.

Integer division

Why define integer division (`(a /b)` means `(integer(floor(real(a) /real(b)))`)?. This is different from many "C" implementations and from all Fortran implementations, which truncate toward zero.

We want to avoid any surprises due to differences in "C" compiler implementations. Also, if we were to depend upon division truncating toward zero, we'd have to account for a discontinuity at original_sample value zero. Zero would represent twice as large a range of physical values:

- zero would represent (-1+epsilon to 1-epsilon)
- positive n would represent (n to n+1-epsilon)
- negative n would represent (n-1+epsilon to n)

where "epsilon" is the difference between "n" and the closest representable real number that is different from "n". But, as we have defined integer division, all samples represent the same range:

- any n represents (n to n+1-epsilon)

8 Appendix: Revision history

- Version 19970203 is the first release of this document. It introduces the "`pCAL`" chunk.

9 Contributors

Names of contributors not already listed in the PNG specification are presented in alphabetical order:

- John Bowler, jbowler@acm.org
- Todd French, tfrench@sandia.gov
- Alaric B. Williams, png@abwillms.demon.co.uk

10 Editor

- Glenn Randers-Pehrson, glennrp@arl.mil or randeg@alumni.rpi.edu

End of PNG Scientific Visualization Chunks Specification.