SCEPNG Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

1 Library Overview

Scope of This Document

This document describes SCEPNG, which provides encoding and decoding features for PNG format image files. Basic procedures for encoding and decoding will be explained.

Purpose and Features

SCEPNG is a library for handling PNG-format image files, which are defined by RFC-2083. PNG is an extremely flexible format, but in order to support it in its entirety, the library's code size would need to be quite large and the library would also be more difficult to use. Therefore, the SCEPNG library is implemented in a way that is necessary and sufficient to support only certain features.

Since the library source code is publicly available, its functionality can be easily extended to implement any missing features.

Main Features

- Only commonly used formats can be decoded
  - RGB888 format is converted to the easy-to-use RGBA8888 format for output.
  - When a CLUT is used, transparent color information is merged and stored in the CLUT.
  - This feature converts CLUT/grayscale/ADAM7 interlace scan output data to RGBA8888 format.
- Supports uncompressed encoding format
  - Output data is always RGB888 (24-bit full color) format PNG.
  - The file size will be larger since the data is uncompressed, but encoding can be performed faster.
  - The data that is created complies with the PNG standard, so it can be handled correctly by system software, etc.

Embedding in Program

The files required for using SCEPNG are as follows.

Filename	Description
scepng.h	Header file
libScePng.a libSceDeflt.a	Library file

Include scepng.h in the source program. When building programs, link libScePng.a and libSceDeflt.a. `-lSceDeflt` needs to be specified after `-lScePng` when using `-l` option of the linker because SCEPNG internally uses libdeflt.

Sample Programs

sample_code/audio_video/api_scepng/dec/

This is a decoding sample. It decodes a PNG file and displays it on the screen.

sample_code/audio_video/api_scepng/enc/

This is an encoding sample. It reads a BMP file and encodes it as PNG data

2 Encoding Procedure Overview

Prepare the Data to be Encoded

Generally, the data in the frame buffer that is being displayed can be input data directly. Since it is usually not necessary to draw this data in real time, the picture quality can also be specially increased (such as by increasing the number of polygons or increasing the bit depth).

Note that the image size cannot be changed during encoding. Use some other appropriate method to perform resizing, etc.

Prepare the Storage Destination for the Encoding Result

Prepare a buffer for storing the encoding result. The buffer size can be verified by calling the `scePngGetEncSize()` function. Note that whenever this function is called with the same arguments, the same value will always be returned.

Also, the only arguments of this function are the image size. In other words, the output data size is always the same regardless of the pixel format.

```
Example: To encode a 480 x 272 size image
char *pngbuf = malloc(scePngGetEncSize(480,272));
```

Call the Encoding Function

Call the `scePngEnc()` function to perform the actual encoding.

```
Example: To encode the image that is stored in the buffer.
int res = scePngEnc(
              pngbuf,
              framebuf,
              480, /* image width */
              272, /* image height */
              480, /* image pitch */
              pixelformat);
```

3 Decoding Procedure Overview

Load the Data to be Decoded

PNG data to be decoded must be loaded in a contiguous area of memory (it cannot be divided and read in parts). Also, since all output is performed at one time, you cannot decode only part of the data.

Example:
```
char pngbuf[100*1024];
int pngsize = sceIoRead(fd, sizeof(pngbuf));
```

Prepare the Output Buffer

The output buffer is also used as an intermediate buffer during decoding. Therefore, an area larger than the image size that is actually output must be prepared. When the image size (number of pixels) is W*H and the number of bits per pixel is n, then only int((W*n+7)/8+1)*H is required for the image part of the output buffer, rather than int((W*n+7)/8)*H (where int(x) is the value of x truncated to its integer value).

Also, if a CLUT was used, then an area of (4<<n) bytes is required in order to record the contents of the CLUT. If the output buffer size is less than the total of these two amounts, an error will be returned during decoding.

If the image to be decoded has not be determined, prepare the required number of bytes for the output buffer obtained with `scePngGetOutputInfo()`, or prepare an approximate buffer size (W*4+1)*H for the expected maximum image size (W*H) and run error handling as necessary. (such as replacing the image with an alternate image to display).

Also, a 4-byte aligned area must be allocated for the decoding buffer. This can be accomplished either by allocating the output buffer as an int-type array or by specifying the aligned(4) attribute for the compiler.

If decode speed is required, allocate the output buffer from the memory with caches enabled. The output buffer is also used as an intermediary buffer during decoding; when a memory with caches disabled is specified, the decode speed may be reduced significantly.

Call the Decoding Function

Perform decoding by calling the `scePngDec()` function with the input/output buffer and its size specified.

Example:
```
int decbuf[80*(80*3+1)/sizeof(int)];
int width, height, format;
int res = scePngDec(
              decbuf, sizeof(decbuf),
              pngbuf, pngsize,
              &width, &height, &format);
```

Decoding Result Format

When decoding is successful, the output buffer begins with the contents of the CLUT (if it exists) followed by the image data.

The CLUT format is always RGBA8888, and if the input had a tRNS chunk, the alpha value is also written. If there was no tRNS chunk, the alpha value is 0 (all 0xff). If the bit depth of the image data is n, the number of CLUTs is (1<<n).

The image data is generally output directly without conversion in the pixel format contained in the PNG. However, only RGB888 format data is automatically converted to RGBA8888 format for output.

The image data has the same width as the PNG file that was input, and the part that is less than a byte is padded. In other words, if the number of bits per pixel is n and an image with a width of W pixels is decoded, then the amount of data (number of bytes) in one raster is int((n*W+7)/8).

If the pixel format is RGB888 full-color and there is a tRNS chunk, the alpha value of the color specified as the transparent color is 0 and the alpha value of every other color is 0xff (this feature is disabled when the pixel format is grayscale or RGB161616).

Precautions When Drawing

As mentioned earlier, PNG decoding results are only padded in byte units. Therefore, the GPU cannot always be used to directly manipulate the results. Moreover, because the results of decoding a 1, 2, or 4-bit CLUT-format PNG are in the big endian format, position is reversed every (horizontal) two pixels when simply performing direct rendering on the GPU. In this case, use the CPU to appropriately arrange or pad the data. If the decode results are in CLUT, grayscale or ADAM7 interlace scan format, the `scePngConvertToRGBA()` function can be used to convert the data into RGBA8888. When the PNG file is in the less-than-8-bit CLUT format, endian conversion will also be carried out upon this conversion.