Fast floating point plugin
1.7

http://www.littlecms.com

Contents

Introduction

Little CMS floating point extensions is a customized plug-in. This add-on implements 4 features:

- Increased throughput for 8 bit transforms on gray, RGB and CMYK
- Support for internal Photoshop 1.15 fixed point format
- Increases throughput of 32 bit floating point color transforms
- Adds dithered 8-bit as output format for certain color spaces (Gray, RGB and CMYK)

Licensing

PLEASE NOTE the license of the plug-in is GPL V3.

https://www.gnu.org/licenses/gpl-3.0.en.html

The requirements of this license are, among others, to release your project's source code. If this is not acceptable for your commercial product, an alternate license is available at a reasonable fee. See the web page Little CMS fast float plugin for further information or contact me at sales@littlecms.com.

Installation

The plug-in comes in lcms2standard distribution. The plug-in itself is contained in "<lcms2root>\plugins" folder.

Visual Studio

There is a Visual studio project ready to be included in solutions. The lcms2 included solution also includes this project.

- <lcms2root>\plugins\fast_float\VC2022

Select the target (Release or debug) and build all.

Linux/Unix/Mac

Use this toggle when running configure on lcms2 distribution. Makefile will do all necessary operations, including a testbed on the "check" target.

- ./configure –with-fastfloat
- make
- make check
- sudo make install

Formats

The following new formats are added by the plug-in.

TYPE_GRAY_15	Gray scale 1 channel
TYPE_GRAY_15_REV	Gray scale, reversed polarity
TYPE_GRAY_15_SE	Gray scale, swapped endianess
TYPE_GRAYA_15	Gray scale plus one alpha channel
TYPE_GRAYA_15_SE	Gray scale plus one alpha channel , swapped endianess
TYPE_GRAYA_15_PLANAR	Gray scale plus one alpha channel , planar
TYPE_RGB_15	RGB 3 channels
TYPE_RGB_15_PLANAR	RGB 3 channels planar
TYPE_RGB_15_SE	RGB 3 channels with swapped endianness
TYPE_BGR_15	RGB 3 channels reversed channel order
TYPE_BGR_15_PLANAR	RGB 3 channels reversed channel order, planar
TYPE_BGR_15_SE	RGB 3 channels reversed channel order, swapped endianness
TYPE_RGBA_15	RGB 3 channels plus one alpha channel ,
TYPE_RGBA_15_PLANAR	RGB 3 channels plus one alpha channel , planar
TYPE_RGBA_15_SE	RGB 3 channels plus one alpha channel , swapped endianness
TYPE_ARGB_15	RGB 3 channels plus one alpha channel
TYPE_ABGR_15	RGB 3 channels reversed channel order plus one alpha channel
TYPE_ABGR_15_PLANAR	RGB 3 channels reversed channel order plus one alpha channel , planar
TYPE_ABGR_15_SE	RGB 3 channels reversed channel order plus one alpha channel , swapped endianness
TYPE_BGRA_15	RGB 3 channels reversed channel order plus one alpha channel
TYPE_BGRA_15_SE	RGB 3 channels reversed channel order plus one alpha channel , swapped endianness
TYPE_CMY_15	CMY 3 channels (no K)
TYPE_YMC_15	CMY 3 channels, reversed order
TYPE_CMY_15_PLANAR	CMY 3 channels (no K), planar
TYPE_CMY_15_SE	CMY 3 channels (no K), swapped endianness
TYPE_CMYK_15	CMYK 4 channels
TYPE_CMYK_15_REV	CMYK 4 channels, reversed
TYPE_CMYK_15_PLANAR	CMYK 4 channels, planar configuration
TYPE_CMYK_15_SE	CMYK 4 channels, endianness of words is swapped (for big endian platforms)
TYPE_KYMC_15	KYMC 4 channels
TYPE_KYMC_15_SE	KYMC 4 channels, endianness of words is swapped (for big endian platforms)
TYPE_KCMY_15	KCMY 4 channels
TYPE_KCMY_15_REV	KCMY 4 channels, reversed
TYPE_KCMY_15_SE	KCMY 4 channels, endianness of words is swapped (for big endian platforms)
TYPE_GRAY_8_DITHER TYPE_RGB_8_DITHER TYPE_RGBA_8_DITHER TYPE_BGR_8_DITHER	Special formatters to activate dither (only meaningful on output direction)

TYPE_ABGR_8_DITHER TYPE_CMYK_8_DITHER TYPE_KYMC_8_DITHER	

15 bit Photoshop format

Photoshop internal format is 1.15 fixed point. This simplifies computation and speeds up some operation. The lcms plug-in provides direct support for following 15 bits types. For further reference to this format, refer to Adobe Photoshop SDK.

TYPE_GRAY_15
TYPE_GRAY_15_REV
TYPE_GRAY_15_SE
TYPE_GRAYA_15
TYPE_GRAYA_15_SE
TYPE_GRAYA_15_PLANAR
TYPE_RGB_15
TYPE_RGB_15_PLANAR
TYPE_RGB_15_SE
TYPE_BGR_15
TYPE_BGR_15_PLANAR
TYPE_BGR_15_SE
TYPE_RGBA_15
TYPE_RGBA_15_PLANAR
TYPE_RGBA_15_SE
TYPE_ARGB_15
TYPE_ABGR_15
TYPE_ABGR_15_PLANAR
TYPE_ABGR_15_SE
TYPE_BGRA_15
TYPE_BGRA_15_SE
TYPE_CMY_15
TYPE_YMC_15
TYPE_CMY_15_PLANAR
TYPE_CMY_15_SE
TYPE_CMYK_15
TYPE_CMYK_15_REV
TYPE_CMYK_15_PLANAR
TYPE_CMYK_15_SE
TYPE_KYMC_15
TYPE_KYMC_15_SE
TYPE_KCMY_15
TYPE_KCMY_15_REV
TYPE_KCMY_15_SE

Fast floating point processing

The plug in intercepts float-to-float color transforms and provides extra throughput on certain cases. Following conditions should be met in order to get an optimized color transform:

- Both input and output formats should be float.
- Optimizable color spaces are Gray, RGB, CMYK and Lab.

As long as those conditions are met, every single profile is prone to be optimized. The test bed application shows the throughput increase obtained in a given platform. Please note that unless both formats are float, the internal lcms2 math being used is 16 bits. This applies to dither as well.

8-bit dither

Certain operations on image data like color conversion (e.g. transforming sRGB to printer CMYK) are best done using 16 bpc precision, especially when lookup tables and interpolation are involved. ICC profiles typically use 16 bpc precision, as do the transformation engines using those profiles. Although true 16 bpc pipelines are being developed, and some are already available as host software, most hardware pipelines today are limited to 8 bpc precision, causing the result of color conversions to be truncated. This truncation to 8 bpc can cause visible and objectionable “banding”, “contouring”, or “posterization” to occur in prints (large areas of “flat” color with abrupt “jumps” in between, where the input shows only smoothly varying gradients). Using true 16 bpc pipelines, the problem does not occur.

In order to minimize this effect a mechanism of error diffusion or “dither” has been implemented in the plug-in. To enable this feature, any of those format specifiers should be used for output only.

TYPE_GRAY_8_DITHER
TYPE_RGB_8_DITHER
TYPE_RGBA_8_DITHER
TYPE_BGR_8_DITHER
TYPE_ABGR_8_DITHER
TYPE_CMYK_8_DITHER
TYPE_KYMC_8_DITHER

Throughput increase guides

- Avoid to use *cmsChangeBuffersFormat()*, Transforms that are polymorphic regarding formats are not optimizable. If you need the same transform operating on 8 and 16 bits, consider creating two transforms. Profiles data tables are already shared and the thoughput gain is huge on 8 bits.

- Whenever possible, use the *cmsDoTransformLineStride()* to apply the color transforms. Use image data blocks as big as possible. Starting the function is costly, but then it goes fast. It is better to do a single call to this function for 10K scanlines that 10K calls for one scanline.

2.8

```
void   cmsDoTransformLineStride(cmsHTRANSFORM  Transform,
                                const void* InputBuffer,
                                 void* OutputBuffer,
                                 cmsUInt32Number PixelsPerLine,
                                 cmsUInt32Number LineCount,
                                 cmsUInt32Number BytesPerLineIn,
                                 cmsUInt32Number BytesPerLineOut,
                                 cmsUInt32Number BytesPerPlaneIn,
                                 cmsUInt32Number BytesPerPlaneOut
```

This function translates bitmaps with complex organization. Each bitmap may contain several lines, and every may have padding. The distance from one line to the next one is BytesPerLine{In/Out}. In planar formats, each line may hold several planes, each plane may have padding. Padding of lines and planes should be same across all bitmap. I.e. all lines in same bitmap have to be padded in same way. This function may be more efficient that repeated calls to cmsDoTransform(), especially when customized plug-ins are being used.

Parameters:

hTransform: Handle to a color transform object.
InputBuffer: A pointer to the input bitmap
OutputBuffer: A pointer to the output bitmap.
PixelsPerLine: The number of pixels for line, which is same on input and in output.
LineCount: The number of lines, which is same on input and output
BytesPerLine{In,Out}: The distance in bytes from one line to the next one.
BytesPerPlaneIn*{In,Out}:* The distance in bytes from one plane to the next one *inside a line. Only applies in planar formats.*

Returns:

None

Sample

```
// Sample usage for 15 bit formatters

// Add this include to access new functionality
#include "lcms2_fast_float.h"



// This is the sample from the tutorial, but adapted for the plug-in
int main(void)
{
      cmsHPROFILE hInProfile, hOutProfile;
      cmsHTRANSFORM hTransform;
      int i;
       cmsUInt16Number YourInputBuffer[3], YourOutputBuffer[3];

      //*** This is the one and only additional line you need in your whole app
      //*** to activate the plug-in
      //***

      cmsPlugin(cmsFastFloatExtensions());

      //***
      //***
     //************************************************************************


      // Convert from AdobeRGB to sRGB in Photoshop internal format
      hInProfile = cmsOpenProfileFromFile("AdobeRGB1998.icc", "r");
      hOutProfile = cmsOpenProfileFromFile("sRGB Color Space Profile.icm", "r");


      hTransform = cmsCreateTransform(hInProfile,
                                      TYPE_RGB_15,    // Note this format is new!
                                      hOutProfile,
                                      TYPE_RGB_15,
                                      INTENT_PERCEPTUAL, 0);

      cmsCloseProfile(hInProfile);
      cmsCloseProfile(hOutProfile);

      YourInputBuffer[0] = 0; YourInputBuffer[1] = 0; YourInputBuffer[2] = 0;
      // Or whatever. Note this is 1fixed15 encoded.

      for (i = 0; i < 10; i++)
      {
            cmsDoTransform(hTransform, YourInputBuffer, YourOutputBuffer, 1);
      }

      // Get rid of resources, etc.
      cmsDeleteTransform(hTransform);

      return 0;
}
```