Little CMS2

Threaded plugin
1.0

http://www.littlecms.com

Contents

Introduction 3
Licensing 3
Installation 3
Visual Studio 3
Linux/Unix/Mac 3
Threaded plugin operation 4
Usage 4
A word of warning 4
Sharing with fast float plug-in 4
Throughput increase guides 5

Introduction

Little CMS threaded is a customized plug-in. This add-on uses modern processors to split costly color transforms between processor cores. As s result throughput is greatly improved for color transforms involving large chunks of memory. If you load the whole image to memory and then apply the color transform to it, this plug-in is for you.

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 plugins for further information or contact me at sales@littlecms.com.

Installation

The plug-in comes in lcms2 standard 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\threaded\VC2022

Select the target (Release or debug) and then 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-threaded
- make
- make check
- sudo make install

Threaded plugin operation

The plug in intercepts all color transforms and provides extra throughput by splitting the work across several cores. You can control how many cores you want to use.

Usage

There is just one plug-in entry point. To install this plugin in your code you need to place this in some initialization place.

```
cmsPlugin(cmsThreadedExtensions(CMS_THREADED_GUESS_MAX_THREADS, 0));
```

The first parameter is the number of cores to use, or

CMS_THREADED_GUESS_MAX_THREADS

to apply some heuristics. This constant equals to -1.

Second parameter, flags, is a reserved field for future use. Set it to zero

Every single profile is prone to be optimized. The test bed application shows the throughput increase obtained for a given platform. Expect a x 4-5 (times 4 to 5) throughput gain in most cases.

A word of warning

Despite throughput gain is huge, splitting the buffers and starting threads has a cost. In fact, if you are going to transform buffers of less that 128Kb, it is better to use just one thread as the cost of starting a new thread exceeds the time of transform processing. The plug-in takes care of this and begins to split threads past the 128Kb limit. **So, if your program transforms only small chunks of memory, you will see no improvement at all.**

Sharing with fast float plug-in

You can use both plug-ins in conjunction for improved throughput gain.

Make sure the threaded plug-in initialization call comes last in plug-in initialization chain.

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