Background

libjpeg-turbo is a derivative of libjpeg that uses SIMD instructions (MMX, SSE2, NEON) to accelerate baseline JPEG compression and decompression on x86, x86-64, and ARM systems. On such systems, libjpeg-turbo is generally 2-4x as fast as the unmodified version of libjpeg, all else being equal.

libjpeg-turbo was originally based on libjpeg/SIMD by Miyasaka Masaru, but the TigerVNC and VirtualGL projects made numerous enhancements to the codec in 2009, including improved support for Mac OS X, 64-bit support, support for 32-bit and big-endian pixel formats (RGBX, XBGR, etc.), accelerated Huffman encoding/decoding, and various bug fixes. The goal was to produce a fully open-source codec that could replace the partially closed-source TurboJPEG/IPP codec used by VirtualGL and TurboVNC. libjpeg-turbo generally achieves 80-120% of the performance of TurboJPEG/IPP. It is faster in some areas but slower in others.

In early 2010, libjpeg-turbo spun off into its own independent project, with the goal of making high-speed JPEG compression/decompression technology available to a broader range of users and developers.

License

Most of libjpeg-turbo inherits the non-restrictive, BSD-style license used by libjpeg (see README.) The TurboJPEG/OSS wrapper (both C and Java versions) and associated test programs bear a similar license, which is reproduced below:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the libjpeg-turbo Project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS", AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Using libjpeg-turbo

****

libjpeg-turbo includes two APIs that can be used to compress and decompress
JPEG images:

  TurboJPEG API: This API provides an easy-to-use interface for compressing
  and decompressing JPEG images in memory. It also provides some functionality
  that would not be straightforward to achieve using the underlying libjpeg
  API, such as generating planar YUV images and performing multiple
  simultaneous lossless transforms on an image. The Java interface for
  libjpeg-turbo is written on top of the TurboJPEG API.

  libjpeg API: This is the de facto industry-standard API for compressing and
  decompressing JPEG images. It is more difficult to use than the TurboJPEG
  API but also more powerful. libjpeg-turbo is both API/ABI-compatible and
  mathematically compatible with libjpeg v6b. It can also optionally be
  configured to be API/ABI-compatible with libjpeg v7 and v8 (see below.)

============================
Replacing libjpeg at Run Time
============================

If a Unix application is dynamically linked with libjpeg, then you can replace
libjpeg with libjpeg-turbo at run time by manipulating LD_LIBRARY_PATH.
For instance:

  [Using libjpeg]
  > time cjpeg <vgl_5674_0098.ppm >vgl_5674_0098.jpg
  real 0m0.392s
  user 0m0.074s
  sys   0m0.020s

  [Using libjpeg-turbo]
  > export LD_LIBRARY_PATH=/opt/libjpeg-turbo/{lib}:$LD_LIBRARY_PATH
  > time cjpeg <vgl_5674_0098.ppm >vgl_5674_0098.jpg
  real 0m0.109s
  user 0m0.029s
  sys   0m0.010s

NOTE: {lib} can be lib, lib32, lib64, or lib/64, depending on the O/S and
architecture.

System administrators can also replace the libjpeg sym links in /usr/{lib} with
links to the libjpeg-turbo dynamic library located in /opt/libjpeg-turbo/{lib}.
This will effectively accelerate every application that uses the libjpeg
dynamic library on the system.

The libjpeg-turbo SDK for Visual C++ installs the libjpeg-turbo DLL
(jpeg62.dll, jpeg7.dll, or jpeg8.dll, depending on whether it was built with
libjpeg v6b, v7, or v8 emulation) into c:\libjpeg-turbo[64]\bin, and the PATH
environment variable can be modified such that this directory is searched
before any others that might contain a libjpeg DLL. However, if a libjpeg
DLL exists in an application's install directory, then Windows will load this
DLL first whenever the application is launched. Thus, if an application ships
with jpeg62.dll, jpeg7.dll, or jpeg8.dll, then back up the application's
version of this DLL and copy c:\libjpeg-turbo[64]\bin\jpeg*.dll into the
application's install directory to accelerate it.

The version of the libjpeg-turbo DLL distributed in the libjpeg-turbo SDK for Visual C++ requires the Visual C++ 2008 C run-time DLL (msvcr90.dll). msvcr90.dll ships with more recent versions of Windows, but users of older Windows releases can obtain it from the Visual C++ 2008 Redistributable Package, which is available as a free download from Microsoft's web site.

NOTE: Features of libjpeg that require passing a C run-time structure, such as a file handle, from an application to libjpeg will probably not work with the version of the libjpeg-turbo DLL distributed in the libjpeg-turbo SDK for Visual C++, unless the application is also built to use the Visual C++ 2008 C run-time DLL. In particular, this affects jpeg_stdio_dest() and jpeg_stdio_src().

Mac applications typically embed their own copies of the libjpeg dylib inside the (hidden) application bundle, so it is not possible to globally replace libjpeg on OS X systems. If an application uses a shared library version of libjpeg, then it may be possible to replace the application's version of it. This would generally involve copying libjpeg.*.dylib from libjpeg-turbo into the appropriate place in the application bundle and using install_name_tool to repoint the dylib to the new directory. This requires an advanced knowledge of OS X and would not survive an upgrade or a re-install of the application. Thus, it is not recommended for most users.

Replacing TurboJPEG/IPP

libjpeg-turbo is a drop-in replacement for the TurboJPEG/IPP SDK used by VirtualGL 2.1.x and TurboVNC 0.6 (and prior.) libjpeg-turbo contains a wrapper library (TurboJPEG/OSS) that emulates the TurboJPEG API using libjpeg-turbo instead of the closed-source Intel Performance Primitives. You can replace the TurboJPEG/IPP package on Linux systems with the libjpeg-turbo package in order to make existing releases of VirtualGL 2.1.x and TurboVNC 0.x use the new codec at run time. Note that the 64-bit libjpeg-turbo packages contain only 64-bit binaries, whereas the TurboJPEG/IPP 64-bit packages contained both 64-bit and 32-bit binaries. Thus, to replace a TurboJPEG/IPP 64-bit package, install both the 64-bit and 32-bit versions of libjpeg-turbo.

You can also build the VirtualGL 2.1.x and TurboVNC 0.6 source code with the libjpeg-turbo SDK instead of TurboJPEG/IPP. It should work identically. libjpeg-turbo also includes static library versions of TurboJPEG/OSS, which are used to build VirtualGL 2.2 and TurboVNC 1.0 and later.

Using libjpeg-turbo in Your Own Programs

For the most part, libjpeg-turbo should work identically to libjpeg, so in most cases, an application can be built against libjpeg and then run against libjpeg-turbo. On Unix systems (including Cygwin), you can build against libjpeg-turbo instead of libjpeg by setting

```
CPATH=/opt/libjpeg-turbo/include
and
LIBRARY_PATH=/opt/libjpeg-turbo/{lib}
```

({lib} = lib32 or lib64, depending on whether you are building a 32-bit or a 64-bit application.)

If using MinGW, then set

```
  CPATH=/c/libjpeg-turbo-gcc[64]/include
  and
  LIBRARY_PATH=/c/libjpeg-turbo-gcc[64]/lib
```

Building against libjpeg-turbo is useful, for instance, if you want to build an application that leverages the libjpeg-turbo colorspace extensions (see below.) On Linux and Solaris systems, you would still need to manipulate LD_LIBRARY_PATH or create appropriate sym links to use libjpeg-turbo at run time. On such systems, you can pass -R /opt/libjpeg-turbo/{lib} to the linker to force the use of libjpeg-turbo at run time rather than libjpeg (also useful if you want to leverage the colorspace extensions), or you can link against the libjpeg-turbo static library.

To force a Linux, Solaris, or MinGW application to link against the static version of libjpeg-turbo, you can use the following linker options:

```
  -Wl,-Bstatic -ljpeg -Wl,-Bdynamic
```

On OS X, simply add /opt/libjpeg-turbo/lib/libjpeg.a to the linker command line (this also works on Linux and Solaris.)

To build Visual C++ applications using libjpeg-turbo, add c:\libjpeg-turbo[64]\include to the system or user INCLUDE environment variable and c:\libjpeg-turbo[64]\lib to the system or user LIB environment variable, and then link against either jpeg.lib (to use the DLL version of libjpeg-turbo) or jpeg-static.lib (to use the static version of libjpeg-turbo.)

Colorspace Extensions

libjpeg-turbo includes extensions that allow JPEG images to be compressed directly from (and decompressed directly to) buffers that use BGR, BGRX, RGBX, XBGR, and XRGB pixel ordering. This is implemented with ten new colorspace constants:

```
  JCS_EXT_RGB   /* red/green/blue */
  JCS_EXT_RGBX  /* red/green/blue/x */
  JCS_EXT_BGR   /* blue/green/red */
  JCS_EXT_BGRX  /* blue/green/red/x */
  JCS_EXT_XBGR  /* x/blue/green/red */
  JCS_EXT_XRGB  /* x/red/green/blue */
  JCS_EXT_RGBA  /* red/green/blue/alpha */
  JCS_EXT_BGRA  /* blue/green/red/alpha */
  JCS_EXT_ABGR  /* alpha/blue/green/red */
  JCS_EXT_ARGB  /* alpha/red/green/blue */
```

Setting cinfo.in_color_space (compression) or cinfo.out_color_space (decompression) to one of these values will cause libjpeg-turbo to read the red, green, and blue values from (or write them to) the appropriate position in the pixel when compressing from/decompressing to an RGB buffer.

Your application can check for the existence of these extensions at compile time with:

```
  #ifdef JCS_EXTENSIONS
```

At run time, attempting to use these extensions with a version of libjpeg that doesn't support them will result in a "Bogus input colorspace" error.

When using the RGBX, BGRX, XBGR, and XRGB colorspaces during decompression, the X byte is undefined, and in order to ensure the best performance, libjpeg-turbo can set that byte to whatever value it wishes. If an application expects the X byte to be used as an alpha channel, then it should specify JCS_EXT_RGBA, JCS_EXT_BGRA, JCS_EXT_ABGR, or JCS_EXT_ARGB. When these colorspace constants are used, the X byte is guaranteed to be 0xFF, which is interpreted as opaque.

Your application can check for the existence of the alpha channel colorspace extensions at compile time with:

```
  #ifdef JCS_ALPHA_EXTENSIONS
```

jcstest.c, located in the libjpeg-turbo source tree, demonstrates how to check for the existence of the colorspace extensions at compile time and run time.

libjpeg v7 and v8 API/ABI support

With libjpeg v7 and v8, new features were added that necessitated extending the compression and decompression structures. Unfortunately, due to the exposed nature of those structures, extending them also necessitated breaking backward ABI compatibility with previous libjpeg releases. Thus, programs that are built to use libjpeg v7 or v8 did not work with libjpeg-turbo, since it is based on the libjpeg v6b code base. Although libjpeg v7 and v8 are still not as widely used as v6b, enough programs (including a few Linux distros) have made the switch that it was desirable to provide support for the libjpeg v7/v8 API/ABI in libjpeg-turbo. Although libjpeg-turbo can now be configured as a drop-in replacement for libjpeg v7 or v8, it should be noted that not all of the features in libjpeg v7 and v8 are supported (see below.)

By passing an argument of --with-jpeg7 or --with-jpeg8 to configure, or an argument of -DWITH_JPEG7=1 or -DWITH_JPEG8=1 to cmake, you can build a version of libjpeg-turbo that emulates the libjpeg v7 or v8 API/ABI, so that programs that are built against libjpeg v7 or v8 can be run with libjpeg-turbo. The following section describes which libjpeg v7+ features are supported and which aren't.

libjpeg v7 and v8 Features:

Fully supported:

-- cjpeg: Separate quality settings for luminance and chrominance
   Note that the libpjeg v7+ API was extended to accommodate this feature only for convenience purposes. It has always been possible to implement this feature with libjpeg v6b (see rdswitch.c for an example.)

-- cjpeg: 32-bit BMP support

-- jpegtran: lossless cropping

-- jpegtran: -perfect option

-- rdjpgcom: -raw option

-- rdjpgcom: locale awareness

Fully supported when using libjpeg v7/v8 emulation:

-- libjpeg: In-memory source and destination managers

Not supported:

-- libjpeg: DCT scaling in compressor
   cinfo.scale_num and cinfo.scale_denom are silently ignored.
   There is no technical reason why DCT scaling cannot be supported, but
   without the SmartScale extension (see below), it would only be able to
   down-scale using ratios of 1/2, 8/15, 4/7, 8/13, 2/3, 8/11, 4/5, and 8/9,
   which is of limited usefulness.

-- libjpeg: SmartScale
   cinfo.block_size is silently ignored.
   SmartScale is an extension to the JPEG format that allows for DCT block
   sizes other than 8x8. It would be difficult to support this feature while
   retaining backward compatibility with libjpeg v6b.

-- libjpeg: IDCT scaling extensions in decompressor
   libjpeg-turbo still supports IDCT scaling with scaling factors of 1/2, 1/4,
   and 1/8 (same as libjpeg v6b.)

-- libjpeg: Fancy downsampling in compressor
   cinfo.do_fancy_downsampling is silently ignored.
   This requires the DCT scaling feature, which is not supported.

-- jpegtran: Scaling
   This requires both the DCT scaling and SmartScale features, which are not
   supported.

-- Lossless RGB JPEG files
   This requires the SmartScale feature, which is not supported.

****
**   Performance pitfalls
****

===============
Restart Markers
===============

The optimized Huffman decoder in libjpeg-turbo does not handle restart markers
in a way that makes the rest of the libjpeg infrastructure happy, so it is
necessary to use the slow Huffman decoder when decompressing a JPEG image that
has restart markers. This can cause the decompression performance to drop by
as much as 20%, but the performance will still be much greater than that of
libjpeg. Many consumer packages, such as PhotoShop, use restart markers when
generating JPEG images, so images generated by those programs will experience
this issue.

==
Fast Integer Forward DCT at High Quality Levels
==

The algorithm used by the SIMD-accelerated quantization function cannot produce correct results whenever the fast integer forward DCT is used along with a JPEG quality of 98-100. Thus, libjpeg-turbo must use the non-SIMD quantization function in those cases. This causes performance to drop by as much as 40%. It is therefore strongly advised that you use the slow integer forward DCT whenever encoding images with a JPEG quality of 98 or higher.