crw2dpx v1.5.0

http://www.cinewrangler.com/

INTRODUCTION

The screenshots used throughout this manual show the Mac OS X version. Rest assured, the Windows and Linux versions look almost the same.

LICENSE CODE

A license can be purchased from right inside the software (using PayPal). But keep in mind that a **license is locked to your computer's hardware ID**. Thus, you can't use it on another machine. However, if you send an email explaining why you need a new license code for your newly bought computer, chances are you'll get a new license code for free (unless you "get a new computer" every other week).

SUPPORTED CAMERAS

Currently the following cameras are supported:

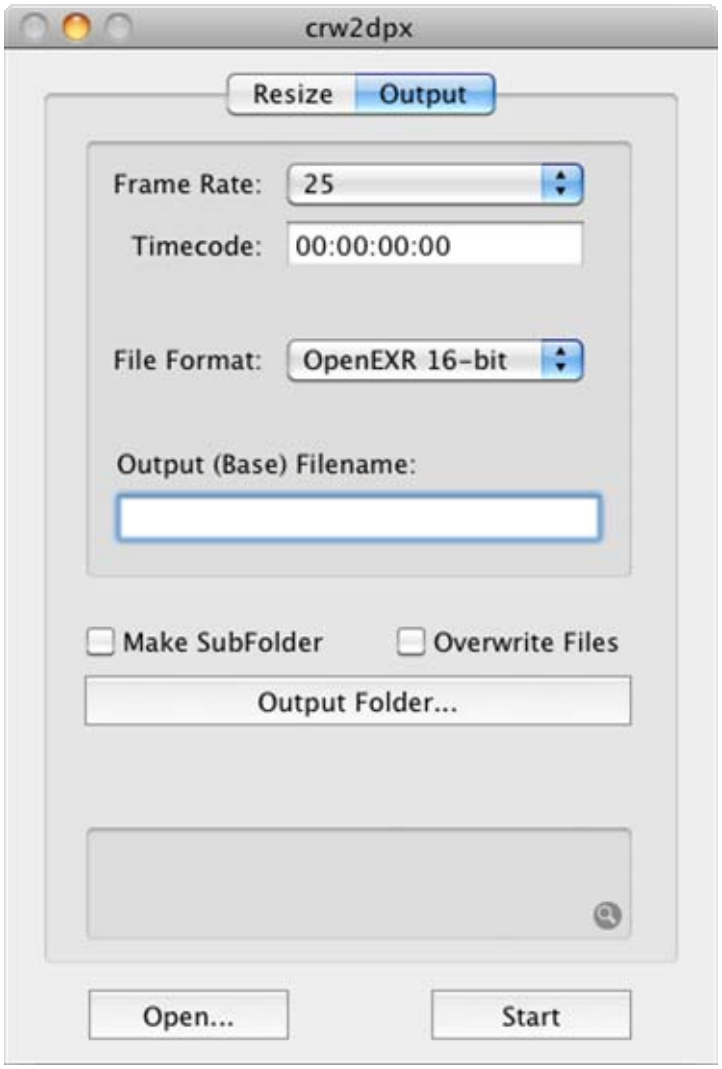
- Canon EOS 100D / Kiss X7 / Rebel SL1
- Canon EOS 20D
- Canon EOS 30D
- Canon EOS 350D / Kiss / Rebel XT
- Canon EOS 40D
- Canon EOS 400D / Rebel XTi
- Canon EOS 450D / Kiss X2 / Rebel XSi
- Canon EOS 50D
- Canon EOS 60D
- Canon EOS 70D
- Canon EOS 500D / Kiss X3 / Rebel T1i
- Canon EOS 550D / Kiss X4 / Rebel T2i
- Canon EOS 600D / Kiss X5 / Rebel T3i
- Canon EOS 650D / Kiss X6 / Rebel T4i
- Canon EOS 700D / Rebel T5i
- Canon EOS 750D / Kiss X8i / Rebel T6i
- Canon EOS 760D / 8000D / Rebel T6s
- Canon EOS 5D
- Canon EOS 5D Mark II
- Canon EOS 5D Mark III
- Canon EOS 5D Mark IV
- Canon EOS 5DS
- Canon EOS 5DS R
- Canon EOS 6D
- Canon EOS 7D
- Canon EOS 7D Mark II
- Canon EOS 1000D / Rebel XS
- Canon EOS 1100D / Kiss X50 / Rebel T3
- Canon EOS 1200D / Kiss X70 / Rebel T5
- Canon EOS 1300D / Kiss X80 / Rebel T6
- Canon EOS-1D Mark II
- Canon EOS-1D Mark III
- Canon EOS-1D Mark IV
- Canon EOS-1Ds Mark II
- Canon EOS-1Ds Mark III
- Canon EOS-1D X
- Canon EOS M
- Canon EOS M3

If you want support for a Canon EOS range camera that is not listed but uses CR2 file format, please provide me with a sample RAW file. Despite the name, the older CRW format is not supported.

FUNCTIONS

OUTPUT TAB

This is the main screen that shows up after you start *crw2dpx GUI*:



From top to bottom your choices are:

Frame Rate — *Frame Rate* of the sequence. For image sequences this gets written to the file headers (if supported by the selected file format). For export to movie files this determines the playback speed.

Timecode — In additional to the frame rate the *Timecode* for the first frame of a sequence can be specified. If supported by the file format, each single image file will then have the correct timecode value (for the given frame rate) embedded into its header.

File Format — For selection of file formats see next section.

Transfer Curve — Selects the transfer or gamma curve to be used. Depending on the selected file format, not all options are available.

Output Filename — The *Output (Base) Filename* specifies the start of the filenames used for a sequence. Each individual file will be named and numbered automatically. Let's assume you give the base name as "`shot_001`" and export to DPX. Your DPX files would then be named "`shot_001.00000001.dpx`", "`shot_001.00000002.dpx`" and so on. The numbers will always be zero padded to 8 digits.
The numbering and correct file extension will be attached automatically. If the *Make SubFolder* checkbox is also checked a sub-folder named "`shot_001`" would be created within the selected output folder and the files would be written to that folder. That way you can easily keep all files of a sequence together in one folder.

When exporting to image file formats (i.e. not video files) it is possible to leave the *Output (Base) Filename* field empty. In that case, the source filenames will be used for the output files (with the file extension swapped out, depending on the output file format). This might come in handy for converting individual stills mages, that don't really represent an image sequence.

Output Folder — The *Output Folder…* button opens a folder open dialog that allows you to select the folder your sequence should be written to. If you're re-doing a conversion and want to overwrite existing files you can check the *Overwrite Files* box. But with this option enabled, there will be no further warning before overwriting a file, so be careful!

Open — Selection of input files is done using the *Open...* button. It allows to either select multiple CR2 files or a folder. In the latter case, all CR2 files in that folder will be converted (subfolders will be ignored).
In the Windows and Linux version only files can be selected from within the open window (because that's the way the native dialogs work). However, to open all CR2 files in a given folder choose *Open Folder…* from the menu.

Please note that all files within one sequence need to be shot with the same camera model! If there is a change of RAW image resolution detected the conversion will stop and an error message will be presented.

All selected CR2 files will be ordered alphabetically. When shooting consecutive frames, that's usually not a problem since the camera numbers them with ascending numbers. However, if your camera's counter went over the IMG_9999.CR2 mark and started at IMG_0001.CR2 again this will cause a problem.

Once a CR2 file or sequence has been opened, some basic file info will be shown just above the Open and Start buttons. Clicking the loupe icon opens an extra window showing more information, such as camera's name and serial number, exposure settings, lens being used…

Start — Finally, the *Start* button starts the conversion or triggers an error message in case your choices just don't seem to make sense.

OUTPUT FILE FORMATS

It's possible to write the output as image sequence or as a movie file. The drop-down list shows formats grouped by file type (i.e. file exension).

Image Sequences

DPX / Cineon

The *DPX* and *Cineon* option are probably the most common formats for ingest into a movie editing pipeline. The color-space for these files is using the sRGB chromaticities with a D55 white point (most apps incorrectly use D65). An 18% grey card gets encoded as code-value 470, 1% black as 95 and 90% white as 685.
The header of DPX files gets padded to 8192 bytes, which is a requirement by some apps in order to provide faster read access for these files.

Exported DPX files contain a bunch of metadata that gets extracted from the CR2 frame. This includes the camera model and its serial number, exposure and focus settings and the owner's name (if present in the CR2 file).

TIFF

The available TIFF formats all use the sRGB chromaticities with a D65 white point. The files have ICC profiles embedded and thus show up correctly in apps like Apple's Preview or Adobe's Photoshop. sRGB and Rec709 both use the official formula with the linear section at the bottom.

Please note that these formats clip black and white to 0 and 100%, respectively.

PNG

The available PNG formats all use the sRGB chromaticities with a D65 white point. The files have ICC profiles embedded and thus show up correctly in apps like Adobe's Photoshop. Apple's Preview seems to use the embedded gamma information for roughly correct display. sRGB and Rec709 both use the official formula with the linear section at the bottom.

Please note that these formats clip black and white to 0 and 100%, respectively.

Targa

Targa files get written using sRGB chromaticities with a D65 white point. File extension is *.tga*.

Please note that these formats clip black and white to 0 and 100%, respectively.

SGI

This is the good old SGI file format that was already around back when serious people were working on IRIX workstations. Compression using run-length encoding (RLE) is not provided because it gives rather bad compression ratios anyway with photographic images (and sometimes even results in files being larger than without compression). File extension is *.sgi*.

Please note that these formats clip black and white to 0 and 100%, respectively.

Radiance RGBE

The Radiance RGBE format (also sometimes known as Radiance HDR) was published by Greg Ward in 1992. It stores a form of floating point data using 8bit for each color component and an additional 8bit for a (common to all components) exponent. It does preserve super-whites (above 100%) but does clip negative values.
A version with compression using run-length encoding (RLE) is also available but usually gives bad compression ratios with photographic images. The code is not particularly speed optimized. File extension is *.hdr*.

OpenEXR

OpenEXR format is a high dynamic-range, floating point file format developed by Industrial Light & Magic (ILM).

Supports linear 16bit half or 32bit float using the same chromaticities as sRGB/Rec709 with D65 white point. A second option is ACES color-space. Header attributes will then be set to comply with the ACES specs (as required by SMPTE 2065-4).

For non-ACES formats there is a choice of different compression methods. Use ZIPS (which is ZIP compression on a per-scanline basis) if you want to work with your files in Nuke since Nuke prefers to be able to read individual scanlines. The ZIP method compresses 16 scanlines together and can achieve a slightly higher compression ratio compared to ZIPS. These two options are lossless.
The PXR24 compression is visually lossless. It truncates the 32bit float data to 24bit and then compresses these 24bits in a lossless way. It's debatable whether a lossy 32bit file is worse than a lossless 16bit file. So give it a try.

Exported EXR files contain a bunch of metadata that gets extracted from the CR2 frame. This includes the camera model and its serial number, exposure and focus settings and the owner's name (if present in the CR2 file).

Movie Files

There are a number of movie file formats supported that result in a single movie file per sequence instead of multiple image files. Apart from ProRes, none of these formats makes use of external codecs or libraries.

QuickTime — Several QuickTime codecs are available to choose from. Basically the offerings divide into codes that write YCbCr data and those that write RGB data.

The following YCbCr formats are supported:

2vuy — Uncompressed 8bit 4:2:2 YCbCr. - This is uncompressed 8bit YCbCr from yesteryear.

v210 — Uncompressed 10bit 4:2:2 YCbCr. - The 10bit version of the 2vuy format. This is used by Final Cut and Premiere. Some Blackmagic and Aja Kona products also support the v210 format.

v410 — Uncompressed 10bit 4:4:4 YCbCr. - This is 10bit uncompressed YCbCr without chroma subsampling. But please note that only very few media players are capable of decoding a v410 encoded file.

Since bayer-pattern RAW data has twice as many green pixel sites as red and blue ones (the rest gets interpolated in the debayering process) writing a 4:2:2 YCbCr format doesn't loose as much image information as one might think.

The following RGB formats are supported:

Animation — The classic lossless codec from the last millennium. Animation uses RLE (run-length encoding), which doesn't give good compression ratios with photographic images. Due to the overhead of storing the RLE information in the file, the totally uncompressed *None* codec (see below) often gives smaller files.

None — Uncompressed 8bit RGB.

PhotoJPEG — Whether *PhotoJPEG* is really an RGB codec can be argued. Many photographers use it; the cameras store the images in that format. But internally it uses YUV encoded data. Just that the famous libjpeg library for handling these files hides all this.

When you export PhotoJPEG from QuickTime Pro you have a quality slider. There are tons of discussions on the internet about at which quality setting which chroma sub-sampling is used. We provide a 100% compression setting that uses 4:4:4 coding (so no loss of chroma information) and a 75% setting with 4:2:2 coding.
The 100% setting actually uses 98%. The reason for labelling it 100% is that common "knowledge" is that only 100% gives 4:4:4 chroma sub sampling. Our code allows independent control of compression quality and chroma sub sampling, so applying a bit more compression to get file size down was an option without having to resort to 4:2:2 sub sampling.

PNG

The *PNG* codec is actually a sequence of 8bit PNG still images (using Rec709 gamma) embedded into a MOV file. As with regular PNG images, it uses the deflate compression algorithm from zlib. The compression is lossless (like the “lossless” from the good old times, not the “lossless” from the marketing guys). Compression ratios depend a lot on image content. Because it uses zlib under the hood compression is rather slow.

ProRes

The (in)famous Apple ProRes codec. Writing these files makes use of Apple’s AVFoundation framework that was introduced in OS X 10.7. Thus earlier version of OS X can not be supported. If you try to write a ProRes file on OS X 10.6 and earlier, an error message will show up to remind you.

When writing ProRes 422 files the image data is handed over to the AVFoundatoin framework as 10bit YCbCr 4:2:2. For ProRes 4444 we submit 16bit RGBA. The RGB to YCbCr conversion is done by some of Apple’s code and we have no control over it.
FFmpeg decodes the supposedly RGB ProRes 4444 files to 10bit YCbCr 4:4:4. QuickTime Player 7 does even display the alpha channel correctly (but it’s all set to 100% opacity anyway). Currently there seems to be no way to encode to ProRes 4444 without an alpha channel using the AVFoundation framework. Doing so would reduce the CPU load for decoding.

See the FAQ section for a discussion of bit depth and RGB versus YCbCr for ProRes.

ProRez

Compressed 4:2:2 or 4:4:4 YCbCr that can be decoded with Apple’s ProRes codec. When writing any of the 422 versions, the encoder gets handed 12bit YCbCr data. For the 4444 versions a fully floating point data path is used. This format is not “the real thing” and is not included in the Mac version of crw2dpx. Please note the spelling, so there is no need to report to the Apple ProRes denunciation portal.

R10k, R10g

These are uncompressed 10bit RGB formats used by AJA Kona capture cards. So in order to play these files in QuickTime player, you might need to install the codecs provided by AJA. The R10g codec actually writes Cineon Log encoded frames and can be considered the movie-file version of DPX files.
Some apps (like Iridas Speedgrade) might support this format without the need for installing a separate codec.
However, it seems that FFmpeg in some versions has a problem with the R10k and R10g codecs. They are available, but the endianness gets identified in a strange (often incorrect) way. This got fixed in the FFmpeg sources in January 2015. We write these files in big-endian, which seems to be more common and works even with older FFmpeg versions.

r210

This is Blackmagic’s uncompressed 10bit RGB format. Uses the DPX type 2 data packing and a rather strange value range of 64 for black and 960 for white. Needs Blackmagic codecs installed for playback. Some apps (like Iridas Speedgrade) might support this format without the need for installing a separate codec.

The QuickTime export options (except for ProRes 422 and 4444 on OS X) do not use any of Apple's QuickTime code. This means that export is not limited to 8 bits and actually *does* write true 10 or 16bit image data if the selected codec can handle it. But please note that at least FCP7 can not read 10bit RGB data. It can only handle 10bit depth when dealing with YCbCr files (and when the correct options are enabled in the FCP settings).
Our internal conversion uses a Rec709 RGB to YCbCr conversion matrix (if applicable) for all HD (and larger) formats. Rec601 is used when exporting to an SD size or in case of the PhotoJPEG codec.

All QuickTime files contain a bunch of metadata that QT7, FCP7/X and Nuke (and probably many other apps) can read. They also contain a proper timecode track.

Hint: Cick on the time display in QuickTime Player 7 in the bottom left… there you can switch to timecode display if the file has a timecode track. Alternatively hit Apple-j and click the tick-box in front of the timecode track to "enable" it, which makes it show up underneath the image area.

AVI

The popular movie file container format on Windows is *AVI*. Supported image data formats in crw2dpx GUI are uncompressed 10bit 4:2:2 YCbCr (also known as v210) and the same in an 8bit version (YVYU).

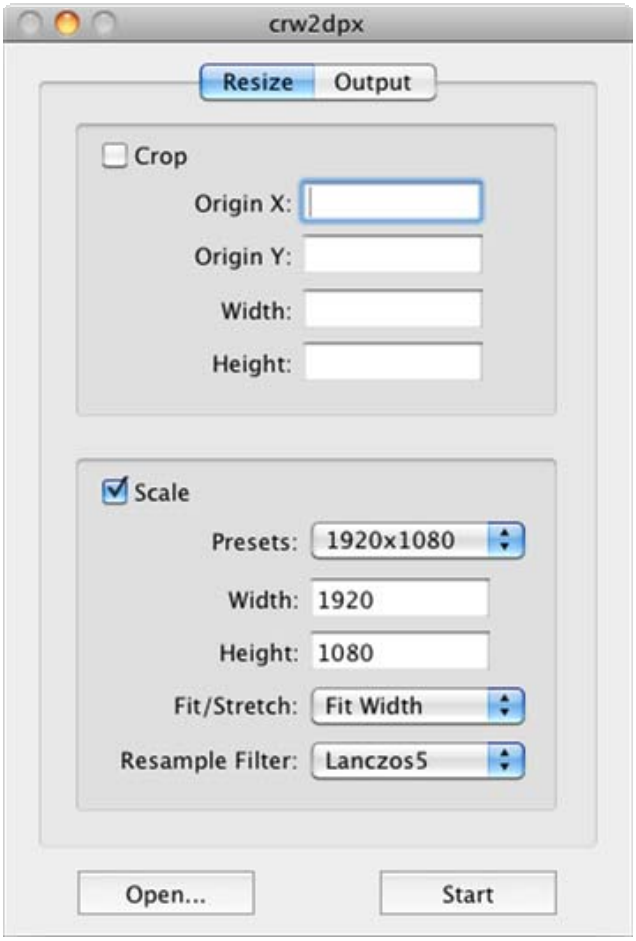
For playback of these files in Windows MediaPlayer or other apps you might need to install a YCbCr / RGB codec. One example might be the "Legacy YCbCr Codec" by Drastic-TV (available from: http://www.drastic.tv/). Premiere Pro seems to play v210 encoded file without any further help.

ARC

The last three movie formats are internal formats used by IFX's Piranha (http://ifxsoftware.com/). If you've never heard about Piranha you probably don't need these file formats.
The provided formats are uncompressed 8bit Rec709 RGB, 10bit Cineon log RGB (D55) or 16bit Half (D65) in one (very) large file.

RESIZE TAB

The resize tab provides options to specify image cropping and / or scaling:



The order of internal processing is cropping followed by scaling. If the crop area is larger than the RAW image you'll be presented an error message.

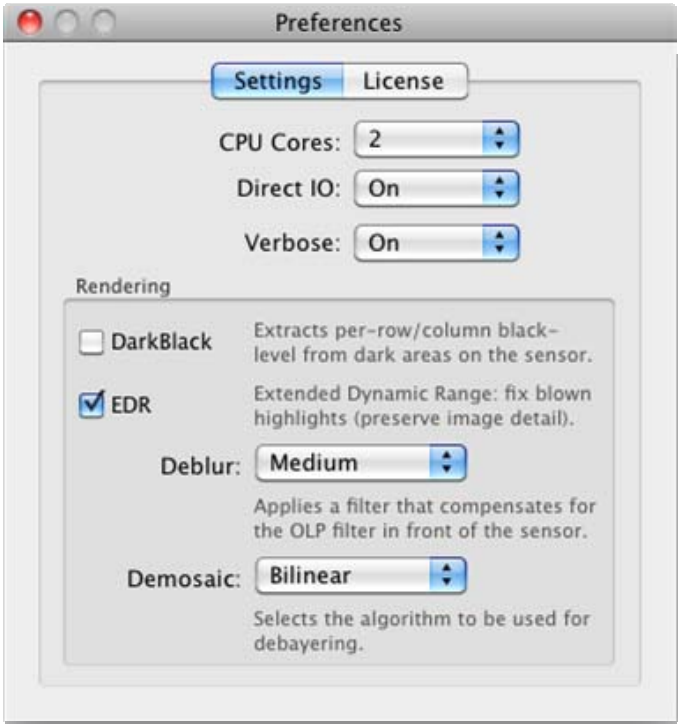
For cropping it is possible to leave fields empty. Just specifying *Width* and *Height* would result in a centred crop area of the given size.

Scaling provides a *Presets* box that lists the most common resolutions. The *Resample Filter* option allows selection of one of the following scaling algorithms: Lanczos2, Lanczos3, Lanczos5, Mitchell-Netravali, Catmull-Rom, Cubic B-Spline and Hermite.
The processing times for each scaling algorithm are different, as is the resulting image. All algorithms will produce different amounts of sharpness and ringing. Check which one suits your image content and don't judge based on processing speed (or lack thereof).

The *Fit/Stretch* option determines how the scaling handles differing aspect ratios between input and output. The default is *Stretch*. The *Fit Width* and *Fit Height* options preserve the aspect ratio. They fit the width (or height) into the specified output size and adjust the height (or width) accordingly by internally calculating additional cropping values.

PREFERENCES

In the preferences window there are a few options that you most likely don't have to deal with on a regular basis.



The top section allows adjusting performance settings to better match your hardware or your working style.

CPU Cores

Unless the *CPU Cores* box is set to *1* all image processing will be spread across as many CPU cores (or real CPUs) as you have specified. However, doubling the number usually does not directly double the processing speed. In order for a CPU to do calculations it needs access to data. In the case of multiple CPUs they might even need to exchange some data between them. All this takes time (with newer hardware architectures being faster). So just experiment and find the sweet spot for your system. The default is half the total number of available CPU cores (unless you run on a single core machine, obviously).

Direct IO

Direct IO is a method for doing file access. Usually (i.e. without it) the operating system is smart enough to cache a file that was accessed, in the expectation that we'll access it again after a short while. So next time one opens the file it will come from RAM (fast) and not disk (slow).
When processing hundreds (or thousands) of frames in a row we just don't need that. So having this option enabled allows your machine to not fill its RAM with data we're not going to need anyway.
There are rare cases with network drives where Direct IO is not supported or just broken. So in case you see failing file reads or writes on network drives, disable this option.

The *Rendering* section has options that effect how the final image will look like.

Dark Black

All cameras have a certain tolerance in their sensors and associated electronics. This means that even without any light hitting the sensor each pixel site will record a slightly different value. The result might be visible vertical or horizontal banding (or both).
To counteract this, the option *DarkBlack* can be enabled to extract separate black-levels specific for each row and column of the sensor. This is done by averaging a few pixel sites for each row and column in an area of the sensor that doesn't see any light. The averaging is required to reduce the impact of noise.
However, keep in mind that this analysis might fail! If there is a dead or stuck pixel in the dark area of the sensor, the averaged value will be way off and you will see more banding instead of less. So use this with caution and re-evaluate its usefulness for each camera you're using.

Deblur

An important step in RAW conversion is sharpening. To avoid moiré a sensor needs an optical-low-pass filter (OLPF). This filter actually blurs the image just the right amount (i.e. it removes fine details that would be beyond the resolving power of the sensor anyway).
In order to get the correct (or desired) perceived sharpness it needs some image processing that does the precise opposite of the OLPF. Unfortunately Canon does not provide any data on the OLPF used in their cameras, so the *Deblur* filter has been arrived at by trial and error. One drawback of this filter that should be mentioned is that it also de-blurs any noise captured in the RAW file. Thus, at very high ISO settings it might be better to disable this filter.
Please note that this is not the typical unsharp-mask (USM) filter that one would use in Photoshop, After Effects or Nuke.

EDR

The third filter that can be employed is the Extended-Dynamic-Range filter, or short *EDR*. It removes the pink cast in "blown" highlights. These are caused by increasing non linearity of the sensor shortly before it clips. And by actually clipping it, of course.
The EDR filter detects clipped green pixel sites (which are the ones that clip first, thus the magenta cast) and reconstructs what's thought to be missing from the neighbouring red and blue pixel sites. Consequently, if the over-exposure additionally clipped one of the other channels, this filter won't be able to reconstruct any meaningful data.

Demosaic

This allows you to choose between two different demosaicing (or *debayer*) algorithms. The default is *Bilinear*, which is actually a slightly advanced bilinear interpolation. It features simple edge detection, to avoid interpolation across sharp edges.
The second option is *LMMSE*, which is short for *Local Minimum Mean Square Error*. This gives slightly less pixelated edges but takes a lot longer to compute.

Frequently Asked Questions

Q: *How can I purchase a license?*

A: A license for crw2dpx GUI can be purchased from inside the software. Just download the demo and each time you start the software it will greet you with a pop-up asking whether you want to purchase a license. Clicking the "buy now" button will open a browser window allowing online purchase using PayPal. Once your payment has been received you'll automatically be sent an email with your license code. Just copy that code into the license code field in the software (under "Preferences").
In order to purchase a license for crw2dpx CLI please get in touch with me using the contact form or via email.

Q: *Can my license code be used on another computer?*

A: A license is locked to your computer's hardware ID. Thus, you can't use it on another machine. However, if you send an email explaining that you need a new license code for your new computer, chances are you'll get one for free (unless you "get a new computer" every week).

Q: *What are the system requirements for GPU rendering? Do I need CUDA or OpenCL?*

A: Rendering on the GPU uses old-school OpenGL. The minimum requirement is OpenGL 2.1 and rectangular textures in 32bit floating point format. The more VRAM the better, 2GB is the bare minimum. Recommended is 4GB and above.
Since only OpenGL is required, GPU rendering should work on any modern GPU no matter what brand.

Q: *I tried GPU rendering for a test frame and it takes longer then CPU. Why is that?*

A: Rendering on the GPU requires a lot of “one time” setup. Shaders need to be compiled and resources allocated. That setup takes time. But it is only required for the first frame in a sequence. Additionally, while the GPU is rendering the CPU will already decompress the next frame. So for a valid speed comparison, try with more than a single frame!

Q: *When using GPU rendering, does the number of CPU cores I select still matter?*

A: Yes, even with GPU rendering the remaining processing can still be split between multiple CPU cores. Those things that still happen on the CPU are unpacking and decompressing the CR2 file, timecode burn-in, the final conversion from 32bit float data to the output format and file writing.

Q: *Does it work with sRAW files?*

A: Earlier versions of crw2dpx CLI (up to 1.1.4) did support sRAW files. Because sRAW files store already debayered image data in a chroma-subsampled format (YCbCr 4:2:2) the available image quality is limited compared to regular RAW files. Maintaining the extra code needed for the sRAW format didn't seem to be justified given the limited use these files seem to seeing.

Q: *Does it support Nikon (.NEF) RAW files?*

A: Currently the only file format supported is CR2. If enough people ask for Nikon NEF support it might get added in a future version. So just ask.

Q: *Compared to the in-camera JPEGs the files written by crw2dpx seem too dark! Why is that?*

A: The JPEG files written by the camera have the selected "Picture Style" applied to them, whereas the files written by crw2dpx have not.
Even with "Picture Style" set to "Flat" or "Neutral" there seems to be some processing happening inside the camera. If you want your RAW files to look like the in-camera JPEGs, then just use the JPEGs.

Q: *Are the Apple ProRes files 12bit RGB? FFmpeg says it's 10bit YUV!*

A: It depends. You should ask Apple's marketing department about that. – All 12bit RGB samples that are out there (basically files recorded by an Arri Alexa) seem to be no different than so called 10bit or YUV files. The only difference is that the colour matrix flag (in the QuickTime file's 'nclc' atom and inside the ProRes frame header) is set to zero (meaning "Reserved" following Apple's documentation). This makes for example MediaInfo display them as RGB. But you can use a Hex editor and change the values to 1 for Rec709 and now you have a YUV file that still displays correctly. After having written a decoder and encoder from scratch, I would say the files are all the same internally. If you read the marketing claims carefully, Apple says the codecs preserve 10bits (or even 12bits) of RGB. Of course you can have RGB to YUV conversion as part of the compression process. JPEG does this as well. It's just a black box that takes RGB as input and delivers RGB as output. Under the hood it does YUV conversion. So if JPEG is RGB then ProRes is as well.
The bit depth issue is a similar one. The input data to the DCT needs to be in a 0 – 4095 range which is 12bits. But it doesn't have to be integer and can be floating point as well. So you don't need to get that 12bit "granularity". The different flavours of the ProRes codec don't seem to have a bit depth as such. It's just the decoder that limits the versions with higher compression to 10bit output because the very fine detail got lost in the compression process anyway. If you do the whole decoding in floating point (possibly using double precision for the DCT) then the question of bit-depth really boils down to how much the data was compressed. The higher the bitrate, the more "bits" will survive.

Q: *Since your software uses dcRAW under the hood anyway, why don't you support the other RAW formats that dcRAW is supporting?*

A: There is no dcRAW under the hood.

Q: *Since your software uses FFmpeg under the hood anyway, why don't you support the other export file format that FFmpeg is supporting?*

A: There is no FFmpeg under the hood.

Command-Line Interface

There is a simple command-line interface that allows to pre-set the GUI values. The Linux and Windows version can be run from the terminal just normally. The OS X version, due to it being an application bundle, requires some trickery:

```
# cd /path/to/the/binary
# ./crw2dpx.app/Contents/MacOS/crw2dpx -h
```

However, if you need to use the command-line quite often, you can just add the following line to your `.bashrc` file (in your home directory):

```
export PATH=/Applications/crw2dpx.app/Contents/MacOS:$PATH
```

This assumes that you copied the crw2dpx application bundle into the Applications folder. It instructs the terminal to look inside the application bundle when looking for a binary. So from now on you can just run crw2dpx directly from the terminal, no matter which directory your're currently in.

With the –h command you can get a list of available command-line options. If a value is not supplied the usual GUI values will be used. Specifying for example a cropping size will automatically tick the 'Crop' checkbox in the 'Resize' tab.

Some features can only be enabled using the command-line. The reason behind this is that I want to avoid filling the GUI with tons of buttons and menus for options which only get used rarely or by very few people.

The following options can only be enable through the command-line interface:

`-clamp <num>`	Turns on clamping of image values. The number parameter specifies whether to only clip at black or also at white.
`-icc <filename>`	This allows you to apply an ICC profile instead of using a generic matrix conversion for the conversion from your camera's color space to the output color space. Only special "input profiles", or sometimes called "camera profiles", are supported. For some cameras it is possible to find these profiles online if you don't know how to make your own.
`-tcburn`	Turns on timecode burn-in. The current timecode will be rendered to the bottom center of each frame.
`-start-num <num>`	This sets the frame number of the first frame. Usually the first frame of an image sequence is frame 1. If you want to start with another number, this is how you can set it.
`-num-digits <num>`	Determines the number of digits used when naming / numbering an image sequence. The default is 8.

There are some more options that you can get listed using the '-h' parameter. Sometimes a client requests a special feature and this is how it gets added. So have a look!

PRESETS

Actually the command line interface can be used to create some sort of presets. If you want to have a preset for scaling to 720p you could make a shell script (Linux and OS X) or a batch file (Windows) that sets the options when starting crw2dpx. The content of a batch file would look like this:

```
crw2dpx.exe -scale 1280 720 %*
```

The `%*` at the end appends all extra command-line parameters that you might supply to the batch file.
Simply create a text file with these lines and save it to the directory where the crw2dpx.exe is located. File extension needs to be ".bat".

A shell script for Linux would look like this:

```
#!/bin/bash
./crw2dpx –scale 1280 720 $@
```

Here the $@ is responsible for appending any extra arguments that you passed to the shell script.
Simply create a text file with these lines and save it to the directory where the crw2dpx binary is located. File extension used is usually ".sh". You also need to make the file executable using the "`chmod u+x preset_filename.sh`" command.