MvidMovieMaker Manual

The mvidmoviemaker utility is a command line application for MacOSX used to convert a Quicktime animation codec movie into a Maxvid formatted movie.

USAGE:

```
$ mvidmoviemaker Superwalk_ANI.mov Superwalk.mvid
  wrote Superwalk.mvid
```

EXECUTIVE SUMMARY

Objective

Processing video on an embedded system like iOS is difficult because CPU resources and memory are limited. The mvidmoviemaker command line utility for MacOSX can be used to pre-process and optimize memory layout of video data so that preprocessing can be skipped when a video is loaded on an iOS device.

Goals

The primary goal of the mvidmoviemaker application is to enable optimized loading of video on IOS devices by doing all the time consuming preprocessing on the desktop instead of on the embedded device. Many video operations are so time consuming and memory intensive that they simply cannot be implemented on the embedded device.

Solution

The mvidmoviemaker application makes it possible to create an optimized Maxvid movie file from a Quicktime movie or from a series of images or. An input Quicktime file must make use of the lossless Animation codec, since a movie processing chain must not suffer from generational loss due to multiple lossy decoding/encoding steps. Any modern video processing tool will be able to export a Quicktime video that makes use of the Animation codec.

LOSSLESS IS IMPORTANT

Digital video has come a long way, but the basic problems are still largely the same as in the days of black and white television. In short, digital video is a huge! Digital video is so big that dealing with video in real time will push a computer system to the limits. Complex digital video delivery systems like H.264 make the final step of delivering data easier because of space savings due to compression. But delivery codecs like H.264 do not address the problems involved in digital video production. The most basic issue is that video data must be processed in a lossless manner during production. For example, a video passed through logic to crop a movie into a smaller aspect ratio must not alter the actual colors in the video. A lossless processing chain is critical because multiple rounds of encoding to and decoding from H.264 would result in unacceptable loss of quality. Exchanging video data between programs requires a lossless format, the mvidmoviemaker application provides a lossless container that it can be decoded on iOS and can be converted back to a Quicktime file without any generational data loss.

Many many long hours were spent in making sure that the mvidmoviemaker utility is able to process video data in a lossless fashion. One can make use of the mvidmoviemaker utility to convert data either from a Quicktime container or from image files and store the result in a Maxvid movie file. One can then extract each frame of the Maxvid movie file as a series of PNG images or convert the entire movie back to a Quicktime container without actually changing the video data. While that sounds straightforward, actually producing a bug free implementation that does all this was surprising difficult due to a shocking number of bugs and misfeatures in the Quicktime APIs. The following examples demonstrate how to convert images to a Maxvid file and how to convert a Quicktime file to a Maxvid file.

MVID FROM IMAGES EXAMPLE

Converting a series of images into a movie file involves invoking the command like mvidmoviemaker application, indicating the image files to read, the name of the output file, and the frame rate of the movie in frames per second.

```
$ mvidmoviemaker Frame01.png MyMovie.mvid -fps 5
```

The command above writes a movie named MyMovie.mvid by reading frame files named Frame01.png, Frame02.png, Frame03.png and so on. The number of frames in the movie file depends on how many frames are found. The -fps 5 arguments indicate that the movie frame rate will be set to five frames per second.

MVID FROM MOV EXAMPLE

Converting a Quicktime movie file into a Maxvid movie file involves invoking mvidmoviemaker with the input and output filenames.

```
$ mvidmoviemaker MyMovie.mov MyMovie.mvid
```

The command above writes a movie named MyMovie.mvid by reading frame data from MyMovie.mov. Note that the -fps argument is not required since the frame rate of the output movie will default to the frame rate of the input movie. The bit depth of the input Quicktime movie (16, 24, or 32 bits per pixel) will be used as the default bit depth of the Maxvid file.

EXTRACTING FRAMES

In the examples above, a Maxvid file was created from a series of images stored on disk or inside a Quicktime file. But how does one extract the images once video data is in mvid format? A simple command is all that is required.

```
$ mvidmoviemaker -extract MyMovie.mvid
wrote Frame01.png
wrote Frame02.png
...
```

The -extract argument passed to mvidmoviemaker tells the program to decode each frame of data and then write the data to a PNG image file. If a movie frame rate is 10 fps and the movie is 10 seconds long, then a total of 100 frames will be extracted.

Extracting frame images from a Maxvid file is useful, one might want to examine the content of the frame images or even pass the frames to another program that only accepts input frames as image files. But be aware that extracting as a series of images can result in a ton of disk space being used up.

CONVERT MVID TO QUICKTIME

In the examples above, a Quicktime file that makes use of the Animation codec was converted to a Maxvid file. This example shows how to convert a Maxvid file back to a Quicktime container that contains Animation codec data.

```
$ mvidmoviemaker MyMovie.mvid MyMovie.mov
```

It is just that easy! The command above will write a Quicktime file and encode data using the lossless Animation codec. The Quicktime file written by this command can be played with the Quicktime player in the Finder on MacOSX. This Quicktime file can also be imported directly by other video software like ffmpeg, After Effects, and others.

There are a couple of things to note about exporting to a Quicktime file. If an existing Quicktime file is converted to Maxvid and then back to Quicktime, one will notice that the exported file can be significantly larger than both the original Quicktime file and the Maxvid file. The size difference is because frame deltas are not written when exporting, so the resulting Quicktime file can be very large. In practice, this is not much of a concern since an exported Quicktime file is an intermediate file used to view the contents of a movie with the Quicktime player or to support file interchange with another video software package. Typically, one might want to archive the original Quicktime file and possibly the Maxvid file, but an exported Quicktime would not typically be saved since it can always be regenerated from the Maxvid file.

The second thing to note about a Quicktime file exported from a Maxvid file is that the Quicktime file will always be tagged as using pixel colors in the sRGB colorspace. Colorspaces are a really complex subject and will be covered in more depth later in this document. For now, just be aware that if a Quicktime file exported from a Maxvid file looks a little different in the Quicktime player as compared to the original Quicktime file, then the difference is the result of the sRGB colorspace. The actual pixel values are the same as in the original Quicktime file, but Quicktime has been around for a long time and colorspace support is really buggy and only partially implemented in many Quicktime encoders.

Basically, mvidmoviemaker will write the Quicktime file with the proper sRGB colorspace tags and then Quicktime will display the pixel data as best it can to approximate the colors that will appear when the movie is displayed on an iOS device. Note that your monitor needs to be properly calibrated for best results.

DISPLAY MAXVID INFO

It can be very useful to print out the header information for a Maxvid file. The following command shows how to display Maxvid header information.

```
$ mvidmoviemaker -info Superwalk.mvid
MVID:                Superwalk.mvid
Version:             1
Width:               86
Height:              114
BitsPerPixel:        32
ColorSpace:          sRGB
Duration:            6.0000s
FrameDuration:       1.0000s
FPS:                 1.0000
Frames:              6
AllKeyFrames:        FALSE
```

The Width and Height values indicate the pixel width and height of the movie. The BitsPerPixel value is either 16, 24, or 32. The Duration value indicates how long in seconds that the video would play for. The FrameDuration and FPS values are related, at 30 frames per second a video would have a FrameDuration of 1/30. The Frames value indicates how many frames the video contains. The AllKeyFrames boolean is TRUE only in the case where a video is make up entirely of keyframes, meaning the video contains no frame to frame deltas.

UPGRADE MAXVID FILE

A Maxvid file created with AVAnimator 1.0 should still work with AVAnimator 2.0 since much care was taken to maintain binary format compatibility. But, in some cases one might want to upgrade from version 0 to version 1 for use with AVAnimator 2.0. This can be accomplished easily with the following:

```
$ mvidmoviemaker -upgrade Superwalk.mvid
Wrote: Superwalk.mvid
```

The upgrade command will write over the original input file once the upgrade is complete. It is also possible to pass a second Maxvid filename argument to indicate the name of the output file. Currently, only adler checksums in certain odd cases differ from Maxvid version 0 to version 1.

ADLER CHECKSUMS

Each frame in a Maxvid file is stored with an adler checksum that provides a handy way to verify correctness of the decoded result. One can print these frame checksums easily.

```
$ mvidmoviemaker -adler Superwalk.mvid
0xC2A54D0E
0x163C8B19
0x8137FCDF
0xF85AB53D
0x868A1D35
0xB1106836
```

The adler command provides a handy way to compare two movies if the adler checksums are identical then all pixels in all frames are identical.

SHOW PIXEL VALUES

There may be times when a developer wants to see the actual pixel values encoded in a Maxvid file. Typically, this functionality only be needed for debug purposes.

```
$ mvidmoviemaker -pixels Superwalk.mvid
File Superwalk.mvid, 32BPP, 6 FRAMES
FRAME 1
ROW 0
COLUMN 0: HEX 0x00000000, RGBA = (0, 0, 0, 0)
COLUMN 1: HEX 0x00000000, RGBA = (0, 0, 0, 0)
…
```

The pixels command shows raw pixel values for every pixel in every row and for every frame.

CROP A VIDEO

The crop operation will create a new video that is a subset of the input video. The video size is not scaled as part of a crop operation, the width or height can stay the same or get smaller as a result of a crop.

```
$ mvidmoviemaker -crop "0 0 43 57" Superwalk.mvid CropSuperwalk.mvid
Wrote: CropSuperwalk.mvid
```

The invocation above shows a crop of a 86x114 input video down to a size of 43x57. The result is that the upper left quadrant of the original video is retained in the cropped video. Note that a crop would never change the pixel depth.

RESIZE A VIDEO

The resize operation is typically used to shrink a larger video down to a specific size. A video does not normally look good when scaled up to a larger size. The following shows a typical resize command with the target width and height are passed in as an argument.

```
$ mvidmoviemaker -resize "43 57" Superwalk.mvid HalfSuperwalk.mvid
Wrote: HalfSuperwalk.mvid
```

The invocation above shows an input video of dimensions 86x114 that has been scaled down to a size of 43x57 exactly one half the original size.

Because shrinking a movie to half size is a useful operation, resize supports a shortcut for this specific usage.

```
$ mvidmoviemaker -resize HALF Superwalk.mvid HalfSuperwalk.mvid
Wrote: HalfSuperwalk.mvid
```

The invocation with the HALF argument will create the same output movie at 43x57.

Previously, it was noted that scaling a video up would not typically be done due to poor quality of the resulting scaled movie. But, in the case of doubling the size of the movie, special logic can be used to replicate each pixel as 4 pixels in double size movie. The following example shows passing DOUBLE to the resize option to access this special case logic.

```
$ mvidmoviemaker -resize DOUBLE Superwalk.mvid DoubleSuperwalk.mvid
Wrote: DoubleSuperwalk.mvid
```

The invocation above writes a 172x228 movie where each pixel from the original is mapped to 4 pixels in the output. One might use this double size logic to create a double size representation of a movie and then scale the double size movie down to a smaller size that is still larger than the original. One might also double size a video and then encode the results as H.264 which could then be scaled down to the original size in order to encode with higher precision for a video that contained text or fine edges that could be blurred by H.264 encoding.

VIDEO DELTA

The rdelta operation can be used to compare all the pixels in all the frames of two different movies. Typically, this command would be used to validate the correctness/quality of other operations, as the command will print a message to indicate that all pixels are exactly the same. Assume that one used the scale operations described above to create a double size movie and then cut that size in half. The following could be used to verify that the resized movie is exactly the same as the original.

```
$ mvidmoviemaker -rdelta Superwalk.mvid HalfDoubleSuperwalk.mvid Delta.mvid
Found 0 modified pixels
Wrote Delta.mvid
```

The invocation above shows that expanding by 2x and then scaling down results in exactly same pixels as the original movie. The Delta.mvid in this case is exactly the same as the original.

The following commands could be used to see an example where the two videos are not exactly the same, such that the delta movie would appear with transparent red pixels over the pixels that changed. The scale up used here would be lossy.

```
$ mvidmoviemaker -resize "172 228" Superwalk.mvid BadDoubleSuperwalk.mvid
```

```
Wrote: BadDoubleSuperwalk.mvid
$ mvidmoviemaker -resize HALF BadDoubleSuperwalk.mvid BadHalfDoubleSuperwalk.mvid
Wrote: BadHalfDoubleSuperwalk.mvid
$ mvidmoviemaker -rdelta Superwalk.mvid BadHalfDoubleSuperwalk.mvid Delta.mvid
Found 18022 modified pixels
Wrote Delta.mvid
$ mvidmoviemaker Delta.mvid Delta.mov
Delta.mvid
Delta.mov
```

One can now view the results by opening Delta.mov in the Quicktime player. The partial red transparent pixels indicate where pixels differ in the two movies.

In the case of a lossy conversion like encoding to H.264, the rdelta command can be used to see the results of the compression. This could be useful for example if one wanted to compare the results of H.264 with various CRF quality settings.

ALPHA SPLIT / JOIN

The H.264 format does not support an alpha channel by default. AVAnimator 2.0 includes some exciting new functionality that supports H.264 encoding of videos with an alpha channel using a pair of encoded videos. To create these split RGB+A videos, one would use the mvidmoviemaker utility to split the RGB and alpha components.

```
$ mvidmoviemaker -splitalpha Superwalk.mvid
Split Superwalk.mvid RGB+A as Superwalk_rgb.mvid and Superwalk_alpha.mvid
Wrote Superwalk_rgb.mvid
Wrote Superwalk_alpha.mvid
```

One could then convert these split videos into Quicktime movies and then pass them to ffmpeg to encode to H.264. The join operation is the opposite of a split.

```
$ mvidmoviemaker -joinalpha Superwalk.mvid
Combining Superwalk_rgb.mvid and Superwalk_alpha.mvid as Superwalk.mvid
Wrote Superwalk.mvid
```

The join operation is exactly what the client would do at runtime to decode the two movies and join the components back together one pixel at a time.

Note that one would normally make use of the ext_ffmpeg_splitalpha_encode_crf.sh script supplied with the AVAnimatorUtils tools to execute all the alpha split steps and encode with H.264. This script does the -joinalpha step as part of the encoding process.

ENCODING OPTIONS

By default, mvidmoviemaker will attempt to detect the most optimal bits per pixel setting based on the actual pixel values used in the video. For example, if a Quicktime Animation is stored as 32BPP but all the pixel values are fully opaque, then a 24BPP movie will be written. Most of the time, automatic detection is very useful because other tools in the process may not implement this type of checking.

But, in the case where an input 32BPP movie is scanned and it is discovered that a 24BPP movie would be more optimal, it means that mvidmoviemaker needs to write the movie twice. If a very large movie is to be encoded, then it is faster to explicitly pass the known 24BPP argument to avoid this double write. The next example shows how to pass a specific bits per pixel value to override the detection logic.

```
$ mvidmoviemaker Superwalk.mov Superwalk.mvid -bpp 24
```

The explicit bpp argument can also be used to reduce the bits per pixel down to 16BPP or remove the alpha channel by converting a 32BPP movie down to 24BPP.

Normally, the frame rate detection logic is able to determine the frame rate based on the display times in a Quicktime file. But, in some odd cases one might want to explicitly set the frame rate when encoding. The following command shows explicitly setting the frame rate to 15 frames per second:

```
$ mvidmoviemaker Superwalk.mov Superwalk.mvid -fps 15
```

In the case of encoding a series of images, a -fps or -framerate argument is always required.

One can explicitly create an encoded video with only keyframes when encoding from images using the -keyframe option:

```
$ mvidmoviemaker -extract Superwalk.mvid Superwalk
wrote Superwalk0001.png
wrote Superwalk0002.png
…
$ mvidmoviemaker Superwalk0001.png SuperwalkKeyframes.mvid -keyframe 1 -fps 1
done writing 6 frames to SuperwalkKeyframes.mvid
$ mvidmoviemaker -info SuperwalkKeyframes.mvid | grep AllKeyFrames
AllKeyFrames:       TRUE
```

COLORSPACES

The sRGB colorspace is the only colorspace supported on iOS hardware. All RGB pixel values sent to iOS video hardware are assumed to be in the sRGB colorspace. So, how does one deal with colors defined in another colorspace like Adobe RGB? The trick is just to avoid the problem on the iOS device. All colorspace conversion is dealt with on the desktop, both for performance reasons and because MacOSX includes ColorSync which does the conversion automatically.

Images tagged with a specific colorspace and Quicktime files tagged with a specific colorspace are automatically converted to sRGB when encoding to Maxvid format. One can import images with weird custom colorspaces as long as the images are tagged with an embedded color profile. But, what about content that is not tagged? Basically, there is no perfect answer for how to deal with untagged content. The approach AVAnimator takes is to simply assume that all untagged content is already in the sRGB colorspace. Defaulting to sRGB makes it possible to import untagged Quicktime files exported from a tool like After Effects and assume that the color values are in the sRGB colorspace. Certain tools like After Effects are unable to embed a color profile in a Quicktime file, so import logic must make a default assumption as to the colorspace. An animator will need to double check that external tools like After Effects are configured to emit Quicktime files in the sRGB colorspace, then everything will convert automatically.