The MPEG Library
Version 1.2*

Greg Ward
(greg@bic.mni.mcgill.ca)

February, 1996

Contents

1 Introduction and Background

The MPEG Library is based on an effort from the University of California at Berkeley to create a portable, software-based MPEG decoder [2]. This resulted in the widely distributed (and widely modified) `mpeg_play`, a highly-optimized MPEG decoder that was specifically geared towards displaying under X Windows. The value of having a portable, software MPEG decoder is amply demonstrated by the number of programs that have been adapted from this original Berkeley source (including ports to the Linux SVGA library, Silicon Graphics hardware, and a non-display MPEG information utility). However, the utility of the decoder was limited by the difficulty of extracting the useful, MPEG-related source code from the X11-specific, display-related source. Essentially what was needed was a simple interface that would allow a programmer to extract

* Also covers version 1.3

frames from an MPEG stream (either before or after converting to RGB colour space), and then to do with the image data as he or she saw fit.

The MPEG Library is intended to fill this need. It was developed at the Montreal Neurological Institute in the summer of 1994 in order to facilitate the development of a high-performance, feature-heavy MPEG player for Silicon Graphics workstations. Since then, the Library has found a use in numerous applications, notably as one of several optional libraries used for extending the well-known ImageMagick suite of graphics applications.

2 Programming with the MPEG Library

Using the Library is quite straightforward, and is analogous to the way in which files have been traditionally handled: you open an MPEG stream to initialize internal data structures, and then read frames until the stream is exhausted. At any point, you can rewind the stream to start over; however, random access is not allowed. (This is not due to a fundamental weakness with MPEG; however, due to the nature of the decoding engine at the heart of the MPEG Library, don't expect to see it implemented here any time soon.) When you are finished with the stream, you close it to clean up.

Here is a simple example program to open an MPEG stream (named by the first command-line argument) and read all frames from it. Since displaying images is as non-portable as it is desirable, I have included calls to dummy routines `InitializeDisplay()` and `ShowFrame()`; actually defining these is up to you.

```
#include <stdio.h>
#include "mpeg.h"

int main (int argc, char *argv[])
{
   FILE       *mpeg;
   ImageDesc   img;
   Boolean     moreframes = TRUE;
   char       *pixels;

   mpeg = fopen (argv[1], "r");
   SetMPEGOption (MPEG_DITHER, FULL_COLOR_DITHER);
   OpenMPEG (mpeg, &img);

   InitializeDisplay (img.Width, img.Height);
   pixels = (char *) malloc (img.Size);
   while (moreframes)
   {
      moreframes = GetMPEGFrame (pixels);
      DisplayFrame (img.Width, img.Height, pixels)
   }
   CloseMPEG ();
   fclose (mpeg);
```

```
}
```

For a concrete example, you might wish to consult `easympeg.c`, a very simple SGI-specific MPEG player included with the Library. Also, I have omitted any error-checking or handling here; again, consult `easympeg.c` for a more realistic example.[1]

Note in particular the following points about the above code:

- The caller must take care of opening and closing the file containing the MPEG stream; the Library assumes that it is passed a file ready for reading.
- The `ImageDesc` structure contains all the information that should be needed to display frames from the MPEG stream (although not necessarily all the information you could possibly want to know about an MPEG stream).
- `SetMPEGOption()` can be used to control somewhat the decoding of frames. In addition to selecting a dithering mode, you can also select the luminance and chrominance ranges used for dithering. Also, note that `SetMPEGOption()` should be called *before* `OpenMPEG()` when setting the dithering method.
- The MPEG data can be decoded using a variety of dithering methods. (Note that in this context, *dithering* refers to converting from the luminance-chromaticity, or YCrCb, colour space in which MPEG data is encoded, to the more conventional RGB scheme.)
- You don't need to pass any parameters to `GetMPEGFrame()` or `CloseMPEG()` to tell it which MPEG stream you mean; this is because the Berkeley decoding engine (and hence the MPEG Library itself) depends heavily on global variables, and unfortunately cannot decode more than one MPEG at a time.

More detailed information is provided in sections below.

3 Concepts and Data Formats

This section deals with the main concepts needed to control the MPEG Library and to display the data it returns. It does *not* deal with the details of how MPEG streams are encoded, stored, or decoded.

3.1 Dithering modes

A large number of dithering modes (in fact, all the modes provided by the original `mpeg_play`) are available. A few produce nonsensical results, but all have been fully tested in the context of the MPEG Library and found to agree with the results given by `mpeg_play`.

"Dithering" in this context is the conversion from luminance-chrominance colour space (aka YCrCb, YIQ, or YUV, which is how MPEG streams are encoded and is the same space used by NTSC television signals) to some form of RGB space. The implementors of `mpeg_play` found

[1]For an even more realistic (but of course considerably larger) example, take a look at `glmpeg_play`. This is the full-featured MPEG player that was the impetus for creating the MPEG Library; it is available by from the same location as the library itself: `ftp://ftp.bic.mni.mcgill.ca/pub/mpeg`.

that outright conversion to red/green/blue values takes both more time and memory than any other method they experimented with, so most modes are colour mapped. This means that `OpenMPEG()` will create a colour map which can be accessed by the user via the `ColormapEntry` pointer in `ImageDesc`, and that the pixel values returned by `GetMPEGFrame()` are indeces into this colour map. The dithering mode affects the quality of the decoded images, the number of bits used per pixel, and the colour depth of the image.

The dithering mode is selected with `SetMPEGOption()`, using the `MPEG_DITHER` option and one of the following values:

`ORDERED_DITHER:`	8-bit colour-mapped; reasonable quality; decoding is almost as fast as `GRAY_DITHER`
`ORDERED2_DITHER:`	8-bit colour-mapped; reasonable quality
`MBORDERED_DITHER:`	8-bit colour-mapped; reasonable quality
`FS4_DITHER:`	8-bit colour-mapped; colours are all wrong
`FS2_DITHER:`	8-bit colour-mapped; colours are all wrong
`FS2FAST_DITHER:`	8-bit colour mapped using Floyd-Steinberg error diffusion; reasonable quality
`HYBRID_DITHER:`	8-bit colour-mapped; passable colour
`HYBRID2_DITHER:`	8-bit colour-mapped; slightly worse than `HYBRID_DITHER`
`Twox2_DITHER:`	8-bit colour-mapped with pixels doubled; poor quality
`GRAY_DITHER:`	a 256-shade grayscale rendering; nice quality and fastest decoding
`FULL_COLOR_DITHER:`	a high-quality 24-bit colour rendering; results in slowest decoding
`MONO_DITHER:`	1-bit monochrome dithering; use as last resort for 1-bit displays
`THRESHOLD_DITHER:`	??

The descriptions here are my entirely subjective judgments of the image quality with each dithering mode. "Reasonable" quality is better than "passable." Your mileage may vary. "8-bit" or "24-bit" here refers to the colour depth in the final images, i.e. the minimum number of bits allocated to each pixel. Authoritative information about the actual pixel size can be found in the `ImageDesc` structure filled in by `OpenMPEG()`; for instance, if you select `FULL_COLOR_DITHER`, the colour depth is 24 bits, but 32 bits are allocated per pixel. Thus, the `PixelSize` field in `ImageDesc` will be 32, and the `Depth` field will be 24.

Note that the dithering mode must be set *before* `OpenMPEG()` is called. For example, to select gray-scale dithering and then open the file `example.mpg` as an MPEG stream:

```
char       filename[] = "example.mpg";
FILE       *mpeg;
ImageDesc  image;

mpeg = fopen (filename, "r");
SetMPEGOption (MPEG_DITHER, (int) GRAY_DITHER);
OpenMPEG (&image);
```

3.2 Colour maps

Most dithering modes result in images whose pixel values are indeces to an 8-bit colour map. This colour map is accessed via the `ImageDesc` structure, and it is created by `OpenMPEG()` based on the dithering type selected by `SetMPEGOption()` (this is why the dithering type must be set before calling `OpenMPEG()`). Note that "8-bit" refers to the size of the colour map: each pixel in the colour-mapped images is 8 bits long, so the colour map itself has 256 entries.

The colour map is accessed via the `Colormap` field of `ImageDesc`, which points to an array of `ColormapSize` colour map entries. Each colour map entry is a structure of the form

```
typedef struct
{
   short red, green, blue;
} ColormapEntry;
```

and the colour map is created when `OpenMPEG()` is called. If no colour map is created (i.e., the dithering mode is `FULL_COLOR_DITHER`), then `ColormapSize` will be -1 and `Colormap` will be `NULL`. For example:

```
char      *filename;
FILE      *MPEG;
ImageDesc MPEGInfo;

filename = argv[1];

/* Prepare to read and decode an MPEG stream */

MPEG = fopen (filename, "rb");
if (!OpenMPEG (MPEG, &MPEGInfo))
   exit;

/* Do we have a colour-mapped mode? */

if (MPEGInfo.Colormap != NULL)
{
   for (i = 0; i < MPEGInfo.ColormapSize; i++)
   {
      mapcolor (i, MPEGInfo.Colormap[i].red,
```

```
                              MPEGInfo.Colormap[i].green,
                              MPEGInfo.Colormap[i].blue);
      }
   }
   /* ... */
```

Here, we assume that the function `mapcolor()` is available to set the system colour map.

3.3 Image data format

The image data, as returned by `GetMPEGFrame()`, is formatted in a straightforward way. Pixels are stored in row-major order, starting at the upper left-hand corner of the image. The number of bits allocated per pixel is given by the `PixelSize` field of `ImageDesc`. This is illustrated in Figure 1, which shows a sample 8×10-pixel image, with the offset into the image data for each pixel. If the pixels are 8 bits each, then this will be a simple byte offset.

0	1	2	3	4	5	6	7	8	9
10									
20									
30									
40									
50									
60									
70									79

Figure 1: Illustration of image data layout for a sample 8×10 image. The number at each pixel is just the offset into the image data array.

4 Programming Reference

4.1 The `ImageDesc` structure

Relevant declarations:

```
typedef struct
{
   unsigned char red, green, blue;
} ColormapEntry;

typedef struct
{
   int  Height;             /* in pixels */
   int  Width;
```

```
    int  Depth;              /* image depth (bits) */
    int  PixelSize;          /* bits actually stored per pixel */
    int  Size;               /* bytes for whole image */
    int  BitmapPad;          /* "quantum" of a scanline -- */
                             /* each scanline starts on an even */
                             /* interval of this many bits */
    int  ColormapSize;
    ColormapEntry *Colormap; /* an array of ColormapSize entries */
} ImageDesc;
```

This structure provides (hopefully) all the information needed to display an MPEG stream, although it doesn't necessarily provide all the information you could possibly want to know about such a stream. However, that's not the intent of the MPEG Library; if you really need to know, for instance, just how many intra-frames are in a particular MPEG, you might want to take a look at the `mpegstat` program, which was also based on the Berkeley X11 player.[2]

Here is the list of fields in the structure:

`Height:` the height, in pixels, of the movie.

`Width:` the width, in pixels, of the movie. Note that due to the block nature of MPEG encoding, the height and width will always be multiples of 16.

`Depth:` the number of bits per pixel that are actually relevant to the display. For most dithering methods, this will be 8 (i.e., we usually use an 8-bit colour map); for full-colour dithering, it will be 24.

`PixelSize:` the number of bits (not bytes!) of storage allocated per pixel.

`Size:` the size, in bytes, of one entire unencoded frame. This is simply equal to `Height*Width*PixelSize/8`. (Note: currently, `BitmapPad` is ignored in the calculation of `Size`.)

`BitmapPad:` the "quantum" of a scan line; i.e., each scan line starts on an even interval of this many bits.

`ColormapSize:` the number of entries in the colour map. This is usually 128, but for most dithering methods it can be indirectly modified by the user of the Library. It is zero in non-colourmapped modes.

`Colormap:` the table used to map pixel values to red/green/blue values (which are themselves stored as bytes in the `ColormapEntry` structure. It is `NULL` in non-colourmapped modes.

4.2 SetMPEGOption()

Function prototype:

```
void SetMPEGOption (MPEGOptionEnum Option, int Value)
```

`Option` should be one of:

[2] `mpegstat` should also be available at `ftp://ftp.bic.mni.mcgill.ca/pub/mpeg`.

MPEG_DITHER: Sets the dithering mode, which controls how YCrCb values are converted to RGB space. `Value` should be a `DitherEnum` value, cast to `int`. Dithering modes are explained above, in section 3.1.

MPEG_LUM_RANGE:

MPEG_CR_RANGE:

MPEG_CB_RANGE: These set the ranges of luminance and chromaticity values. The defaults are 8, 4, and 4. (I do not understand the effects of changing these; my experiments indicate that doing so garbles perfectly good colour maps.)

Notes:

`SetMPEGOption()` allows you to set a variety of options related to MPEG decoding. The possible values for `Option` are described above; the possible values for `Value` value are dependent on what `Option` you are setting. Whatever `Value` is, it should of course be cast to an `int`.

4.3 `OpenMPEG()`

Function prototype:

```
Boolean OpenMPEG (FILE *MPEGfile, ImageDesc *Image)
```

Arguments:

`MPEGfile`: A file that is already open for reading, positioned at the beginning of an MPEG stream.

`Image`: Pointer to a user-declared `ImageDesc` structure. You shouldn't change any of the fields in `*Image` yourself, either before or after calling `OpenMPEG()`; use `SetMPEGOption()` instead.

Notes:

`OpenMPEG()` prepares an MPEG stream for decoding. It initializes internal data structures for decoding and dithering and—if applicable—creates a colour map. After calling `OpenMPEG()`, the following fields in `*Image` will be set: `Height`, `Width`, `Depth`, `PixelSize`, `Size`, `BitmapPad`, `ColormapSize`, and `Colormap`.

4.4 `GetMPEGFrame()`

Function prototype:

```
Boolean GetMPEGFrame (char *Frame)
```

Arguments:

`Frame`: Pointer to a user-allocated chunk of memory. Must have enough room for the decoded image, which can be determined from the `Size` field of `ImageDesc`.

Notes:

Decodes the next frame from the movie. Returns `TRUE` if there are any frames left to decode in the movie, or `FALSE` if the decoded frame is the last frame in the movie. That is, for a movie with N frames `GetMPEGFrame()` will return `TRUE` $N-1$ times, and then the call to decode the last frame will return `FALSE`. After that, the behaviour of `GetMPEGFrame()` is undefined (unless you call `RewindMPEG()`.)

4.5 `RewindMPEG()`

Function prototype:

```
void RewindMPEG (FILE *MPEGfile, ImageDesc *Image)
```

Arguments:

`MPEGfile`: The open, readable stream pointer that was also passed to `OpenMPEG()`.

`Image`: The image descriptor that was passed to and filled in by `OpenMPEG()`.

Notes:

Repositions `MPEGfile`'s file-offset pointer to point to the beginning of the stream, and reinitializes internal MPEG Library structures to prepare reading the MPEG again. The first call to `GetMPEGFrame()` after calling `RewindMPEG()` will return the first frame of the movie, as though `OpenMPEG()` had just been called.

Acknowledgements

Most of the credit for this package should go to the authors of `mpeg_play`; all I can really take credit for is shuffling code around and coming up with a reasonably intelligent interface to the decoding engine that does the real work.

Thanks to Magnus Heldestat for contributing a sped-up version of 24bit.c, resulting in faster decoding/dithering of full-colour images.

References

[1] Didier LeGall, "MPEG–A Video Compression Standard for Multimedia Applications," *Communications of the ACM*, April 1991, Vol 34 Num 4, pp. 46–58.

[2] Ketan Patel, Brian C. Smith, and Lawrence A. Rowe, "Performace of a Software MPEG Video Decoder", *ACM Multimedia '93 Conference*.