

Feig's scaled 2-D DCT now tested with the Berkeley MPEG-Player

Central issue of the JPEG or MPEG decoders is the discrete cosine transform. Very often discussed, quite a variety of different methods exist.

By a lot of mathematical optimizations, its computational costs could be reduced from 1024 multiplication's and 896 additions for a full 8x8 DCT down to 54 multiplication's, 464 additions and 6 left shifts required by Feig's true 2-D method based on the discrete Fourier transform.

Many programs, like the Berkeley MPEG Player, even some MPEG Hardware use a 1-D Algorithm with 12 multiplication's and 32 additions per row/column. This method could be optimized to take advantage of the sparse nature of the dct matrices, so that the question now is "**Can the Feig's scaled 2-D DCT method still compete ?**".

Contents:

1. [Show me the results, I can' wait !](#) (Click here to take a look at the speed comparison diagrams first)
 2. [Introduction](#)
 3. [The Berkeley MPEG-Player.](#)
 4. [Runtime investigation of the original.](#)
 5. [Runtime investigation of the MPEG-Player with Feig's scaled 2-D DCT](#)
 6. [The Still A Lot To Do List](#)
 7. [Download section](#)
 8. [Other related links](#)
-

Introduction

People being familiar with the MPEG compression technique should skip the following section. I wrote these additional lines for people who found themselves on these pages, having no idea what the technical slang is all about.

Data compression methods in general try to compromise between processing speed and disk space usage/network loading.

Just because of the large amount of data that has to be dealt with when graphical information are to be stored or transmitted, picture compression methods have been subject of special interest since ever.

Thus, quite a few people have been working on that field in since 1982. In order to avoid the creation of competing, independently developed standards a joint collaboration between CCITT and ISO took the initiative to establish an international research group back in 1986.

This international group, called JPEG (Joint Photographic Experts Group) selected by competitive contests the *Adaptive Discrete Cosine Transform approach* as being the basis for their image compression standard out of 12 other different proposals, that have been contributed by the participating teams.

Applying this method, the JPEG Data compression today (which is now officially admitted by ISO, no 10918) can reduce picture data from 24 bits/pixel, in true color mode, to lower than 0.75 bit/pixel, thereby still providing an excellent quality.

In easy words, the basic idea behind the JPEG Compression Standard is to figure out, which information's are worth storing and which can be omitted (which are for example the, for human eyes invisible, high contrast frequencies, the same color information's in a part of a picture etc.).

Readers unfamiliar with the above mentioned Adaptive Discrete Cosine Transform, it's theory and implementation are encouraged to read

- [[some other papers from the author of the java MPEG-Player, Dr. Jörg Anders](#)]
- E.Feig,E.Linzer: Discrete Cosine Transform Algorithms for Image Data Compression, Proceedings Electronic Imaging '90 Eas, pp.84-7, Boston, MA(Oct.29th-Nov.1rst,1990)
- W.B:PenneBaker,J.L.Mitchell, JPEG Still Image Compression Standard, VAN Nostrand Reinhold 1993

The JPEG image compression standard was extended to the MPEG system from the Motion Pictures Experts Group to meet the needs of a compression system for image sequences:

In addition to the JPEG compression, MPEG compares each picture (frame) during the encoding process, making a decision whether the whole picture must be encoded or just the difference (e.g. unchanged parts of pictures or even moving objects inclusive translation vector).

The result of this hybride method is usually 3 times better then a plain JPEG compression.

On the one hand, it is now clear that MPEG decoding means a lot more than just running the discrete cosine transform on each macroblock.

On the other, this work attempts to show that even after years there still can be room for improvement (concerning the DCT code) and tries to give some creative inspirations for further work on the mpeg decoding software.

The Berkeley MPEG-Player

Before any modifications to a software system can be made, it is essential to know its structure (software engineers rather say "architecture").

Because the necessary papers from the program designers (specification, architecture model) are not available, at least they can not be found in the package, a code inspection has to be done.

When taking a first look into the source code, the entire program appears as a product of the programming style of the late 80's. Someone who expected a clear and a practical design is rather confronted with a mess.

Modules are full of global variables (like flags, switches, file pointers,), where it is hard to guess where they are related to. At this level of complexity, for instance, it is strongly recommended to add a configuration structure (or object) and its manager.

The User Interface, ctrlbar was integrated dispassionately, the dither modules as well.

From deep inside the decoder it has to be jumped back into the main.c module to branch crisscross into a dither module right after and back in this circle stack to main.c .

Some people badly played with **#ifdef**'s and **#endif**'s every time they added functionality, maybe not loose the 0.3% speed with this flexibility and all the way they programmed like this:

```

        vid_stream->future->locked |= FUTURE_LOCK;
        vid_stream->current = vid_stream->past;
#ifdef NOCONTROLS
        ExecuteDisplay(vid_stream, 1, xinfo);
#else
        ExecuteDisplay(vid_stream, xinfo);
#endif /* !NOCONTROLS */
    }
    } else {
#ifdef NOCONTROLS
        ExecuteDisplay(vid_stream, 1, xinfo);
#else
        ExecuteDisplay(vid_stream, xinfo);
#endif /* !NOCONTROLS */
    }
}

```

Finally no coding convention at all seems to be valid. No one knows, whether it should be programmed in ANSI or K&R standard. Some rattled people did both and wrote very readable things like:

```

#ifdef __STDC__
int MakeFloatClockTime(unsigned char hiBit, unsigned long low4Bytes,
                      double * floatClockTime)
#else
int MakeFloatClockTime(hiBit, low4Bytes, floatClockTime)
    unsigned char hiBit;
    unsigned long low4Bytes;
    double *floatClockTime;
#endif
{
    if (hiBit != 0 && hiBit != 1) {
        *floatClockTime = 0.0;
        return 1;
    }
    *floatClockTime
        = (double)hiBit*FLOAT_0x10000*FLOAT_0x10000 + (double)low4Bytes;
}

```

Result of the 'reengineering' is the picture below. The upper part of the picture shows the basic video decoder block diagram taken from ISO 11172-2, page VIII. The part of the mpeg_play call graph below shows where (which function and which module) the components of the video decoder can be found in mpeg_play:

In readfile.c a buffered bitstream is implemented. It provides the functions `get_more_data()`, `get_bits()` ... used by the parsers that are spread over a couple of different modules.

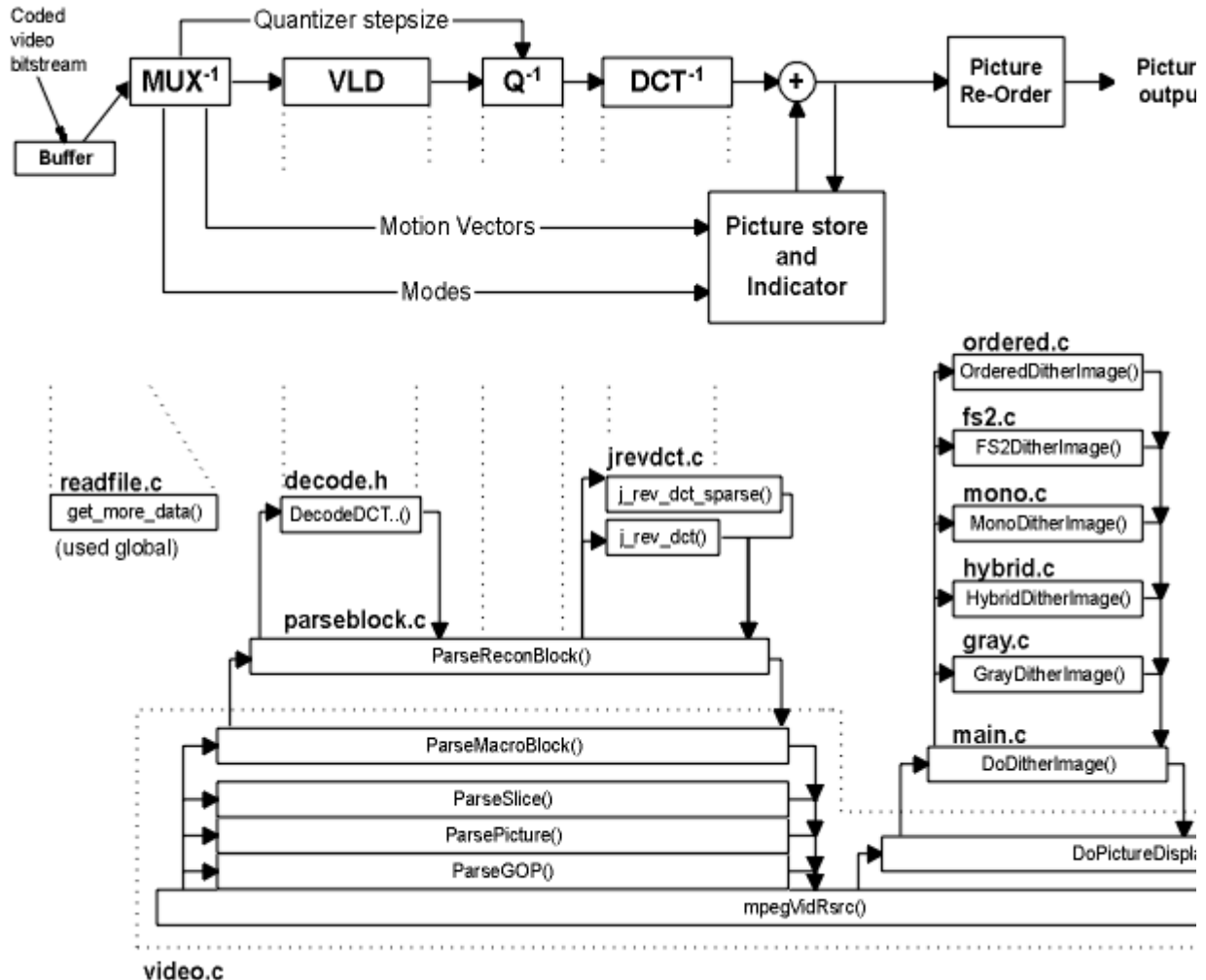
The **main()** function calls **mpegVidRsrc()** in video.c, the function which implements the demultiplexer MUX⁻¹ and where the stream is parsed first. Depending on the sequence code (Group Of Pictures, Picture Start, Sequence Start) the related subparsers are called, for instance `ParseMacroBlock()`.

ParseMacroBlock() calls `ParseReconBlock()` in parseblock.c 6 times, 4 times for the luminance blocks and 2 times for the chrominance blocks (*).

(*) = not in gray dithering.

ParseReconblock() uses the macros in decode.h, which implements the variable length decoder VLD for sequence headers, motion vectors, dct values, et cetera.

After the dct values are decoded, ParseReconBlock() does the dequantisation Q^{-1} and calls `j_rev_dct()` or `j_rev_dct_sparse()` in `jrevdct.c`, the functions that have to be replaced in this experiment.



To make it short, we have to change at least `parseblock.c` and `jrevdct.c`. But, it turned out that the **Feig's scaled 2-D DCT won't work any more, neither with 16 bit DCT values nor with 8 bit quantization tables**. Because the the intra- and nonintra quantization tables are multiplied with the DFT constants, the result does no more fit into unsigned char - and after the next multiplication (that is the dequantization step) the DCT values will not fit into 16 bit.

Because of `mpeg_play`'s design, these major changes will affect not just `parseblock.c` and `jrevdct.c`, but files like `proto.h` `video.h` `video.c`, as well.

Runtime Investigations of the MPEG-Player

Finally, to find out, whether Feig's scaled 2-D DCT is faster than the "Practical Fast 1-D DCT Algorithm with 11 Multiplication's" in C. Loeffler, A. Ligtenberg and G. Moschytz, the frame rates of the two different `mpeg_play`'s are compared.

Therefore the MPEG-Player must be invoked with the following options:

```
./mpeg_play -framerate 0 -dither none -no_display videos/paris.mpg
```

A possible result would be:

```
148
Done!
```

```
Real Time Spent (After Initializations): 1.698300 secs.
Avg. Frames/Sec: 87.145969
```

First, it is of course interesting to know, how a speedup of the DCT code would affect the behavior of the entire program, how much time the DCT costs, respectively.

A very easy and effective method is to use the GNU-Profiler. It must be said that it is not the exactest for fast executed functions, because the times, which are displayed, are propagated along the edges of the estimated call graph.

To use it, `mpeg_play` must be compiled with the `'-pg'` option and linked with the profiler code in the `gmon` library.

In the **Makefile** the following two definitions must be changed:

```
CC = gcc -pg
```

```
LIBS = -L/usr/lib/X11 -lXext -lX11 -lgmon
```

To get the output from the profiler the program must terminate correctly, but:

In Version 2.3 of `mpeg_play` a bug in `main.c` keeps `mpeg_play` sticking in a loop, it does not terminate.

This can be fixed up by the patch or by changing the lines 884-890 from

```
#ifndef NOCONTROLS
  if (ControlShow == CTRLBAR_NONE) {
    while (TRUE) {
      for (i=0;i < numInput; i++) {
        while (theStream[i]->film_has_ended != TRUE) {
          mpegVidRsrc(0, theStream[i], 0, &xinfo[i]);
        }
        if (loopFlag) {
          rewind(theStream[i]->input);
        }
      }
    }
  }
#endif
```

to

```
#ifndef NOCONTROLS
  if (ControlShow == CTRLBAR_NONE) {
    while (workToDo) {
      workToDo=FALSE;
      for (i=0;i < numInput; i++) {
        while (theStream[i]->film_has_ended != TRUE) {
          workToDo=TRUE;
          mpegVidRsrc(0, theStream[i], 0, &xinfo[i]);
        }
      }
      if (loopFlag) {
        rewind(theStream[i]->input);
      }
    }
  }
#endif
```

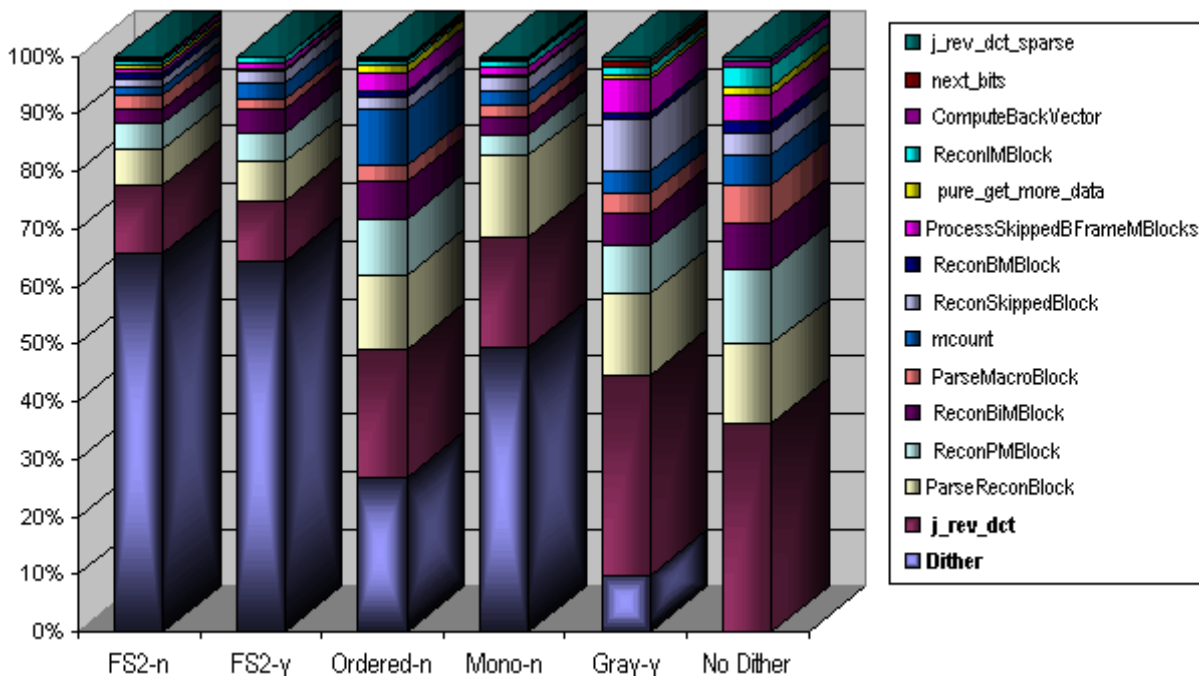
After running `mpeg_play` the results can be examined at with:

```
gprof mpeg_play
```

The following table is a summary of several tests with the same video `paris.mpg`:

Dither Type	Display video ?	Time Dither (%)	Time DCT ⁻¹ (%)	Links to Results
Floyd Steinberg 2	no	65.84	11.91	Click here for details
Floyd Steinberg 2	yes	64.11	10.98	Click here for details
Ordered	no	26.48	21.92	Click here for details
Mono	no	49.40	19.28	Click here for details
Gray	yes	9.49	34.18	Click here for details
None	no	-	35.71	Click here for details

...graphically represented gives:



Conclusions:

1. Once the picture is decoded and dithered in the memory, no time is recognizably wasted for drawing the picture, because the X-Shared-Memory mechanism is used (the slower ExecuteDisplay() for FS2-y in the 2nd column now takes 2-3% time from dither() and dct())
2. Floyd Steinberg 2 is very expensive compared to Gray, because in gray mode no color informations are processed, the chrominance blocks are skipped and the luminance matrices are mapped directly into the window memory.
3. **The maximum CPU-Time consumed by DCT⁻¹ during the MPEG decoding seems not raise above 35 % for normal video streams.**

The last statement is very important, because it sets the limits of the speedup that could be achieved even when a very very fast DCT⁻¹ is invented.

Runtime Investigations of the modified MPEG-Player

As written before, the following files were modified:

- [main.c](#) (mpeg_play now terminates correctly when invoked with -no_display)
- [jrevidct.c](#) (switched to Feig's scaled 2-D DCT)
- [parseblock.c](#) (see above)
- [video.h](#) (data structures changed, now 32 bits per DCT_ELEMENT, 16 bits per quantization element and)
- [video.c](#) (retrieval of the new quantization matrices adapted for the DFT quantization)
- [proto.h](#) (functions changed and added)
- [jrevidct.h-tmpl](#) (template where the DCT constants are substituted using the mechanism from)
- [compute.y](#) (the java mpeg player)
- [expand_const.c](#) (see above)
- [Makefile](#) (constant substitution added)

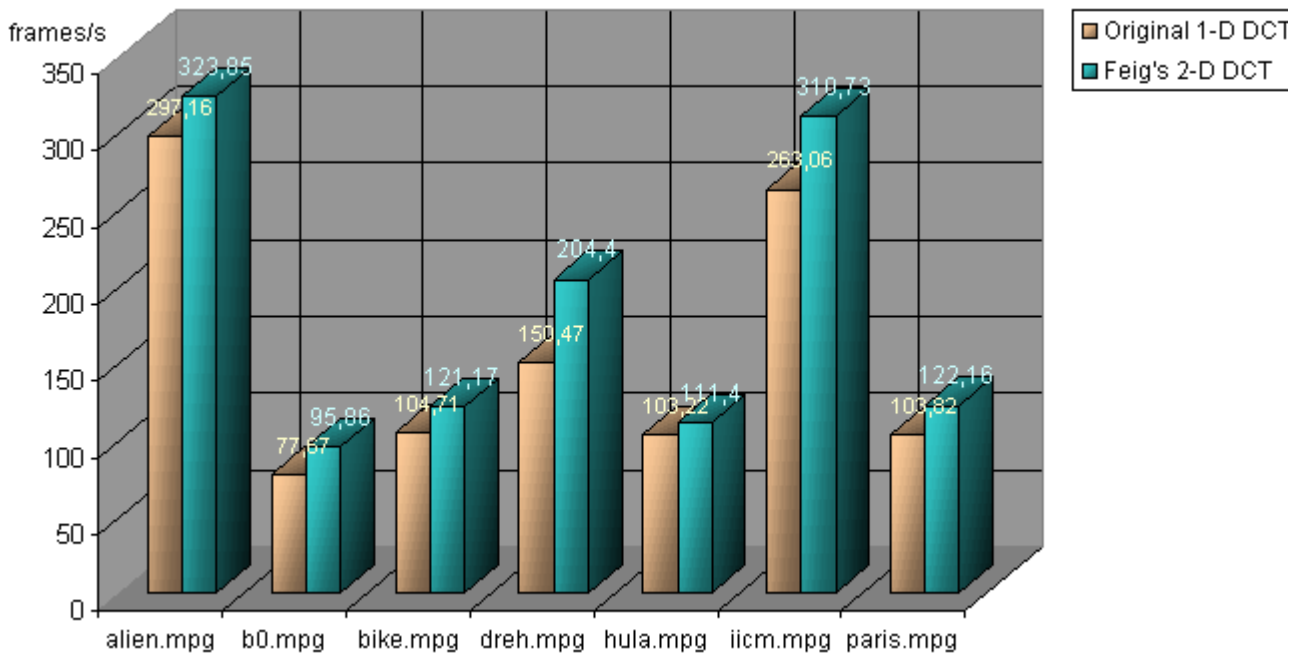
If you already have the source code for mpeg_play_2.3 you can use the patch from the download section. To apply the changes type:

```
patch -d mpeg_play < patch_dct
```

After running the two versions of the program on an Intel PII with Linux 2.0.33, the following table could be filled in:

Stream	Stream Size	Frames	Resolution	calls to jrevidct(0)	calls to jdct_sparse	Sparse factor %	Frames/s Berkeley DCT	Frames/s Feigs DCT	Speedup
alien.mpg	366123	253	160 x 128	99826	21678	21.7	297.16	323.85	1.089
b0.mpg	719188	100	368 x 240	93786	32251	34.4	77.67	95.86	1.234
bike.mpg	642590	150	352 x 240	99352	24041	24.2	104.71	121.17	1.157
dreh.mpg	50456	15	256 x 192	5733	1309	22.8	150.47	204.40	1.358
hula.mpg	148076	40	352 x 240	34296	9712	28.3	103.22	111.40	1.079
iicm.mpg	1761522	801	160 x 128	258738	125806	48.6	263.06	310.73	1.181
paris.mpg	690185	148	320 x 240	72251	12910	17.9	103.82	122.16	1.177

Another eye candy will enlighten us:



As seen on the other picture, the dithering, huffman decoding and file-I/O consumes a lot of CPU cycles, so that even when no dithering is done, the mpeg_play works only 35 % on the DCT. The more it is surprising to see, that mpeg_play can run at a 1.35 times higher frame rate, when changing the DCT approach and doing some additional code rearrangements. This is also a very good result, since the quantization tables changed from char[][] to short [][] and the dct matrices changed from short[][] to long[][].

to-do list:

Im not quite sure, whether I can continue working on that field or not, so here are some Items that would need to be treated.

1. This implementation of the algorithm is still unchecked concerning the JPEG generic compliance tests.
See ISO DIS 10918 Part 2.
2. The speed comparison should be done on other architectures, too.
3. The 7 videos should be converted to plain DCT values. Maybe mpeg_play can be used for this when adding enough fwrite's in j_rev_dct() and j_rev_dct_sparse(), dumping out the dct_tables and the variable pos for the sparse function.
The operation speed of the 2 DCT's can be compared more exactly in a special DCT test program without any parser, huffman decoder or dither modules.
4. MPEG_PLAY is a great program, but after years it is terribly out of order now. Doing the necessary redesign would help using this program as basis for browser plug-in's, video software, and other multimedia applications.

Download Section

Click on the following Items for download...

- [patch_dft.gz](#) (A patchfile for installing Feig's scaled 2-D DCT into mpeg_play_2.3)
- [mpeg_play_2.3.tgz](#) (Source Code of the Berkeley Player)
- [mpeg_play_dft.tgz](#) (already patched source code)
- [dct_testbench.tgz](#) (for people who want to play around with the DCT, several matrix operations included)
- [doc-html.tgz](#) (this documentation, packed and compressed)

Some Related Links:

This section is reserved for future completion. If it's really urgent try [yahoo](#).
Nevertheless.
Have Fun !

Rene' Stöckel <rst@hrz.tu-chemnitz.de>, 24.10.1998
