https://ntrs.nasa.gov/search.jsp?R=19960008037 2020-06-16T05:25:22+00:00Z

NASA-CR-199828

INTERIM
IN-61-CR
OCIT.
6379
P-348

(NASA-CR-199828) TECHNIQUES FOR VIDEO COMPRESSION Final Report, 15 Dec. 1993 - 15 Sep. 1995 (Auburn Univ.) 348 p

N96-15203

Unclas

G3/61 0081138

NASA
NATIONAL AERONAUTICS & SPACE ADMINISTRATION

Report Documentation Page

1. REPORT NO.	2. GOVERNMENT ACCESSION NO	3. RECIPIENT'S CATALOG NO

4. TITLE AND SUBTITLE

Techniques for Video Compression

5. REASON DATE

6. PERFORMING ORGANIZATION CODE

7. AUTHOR(S)

Chwan-Hwa "John" Wu

8. PERFORMING ORGANIZATION REPORT NO.

10. WORK UNIT NO.

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Department of Electrical Engineering
Auburn University, AL 36849-5201

11. CONTRACT OR GRANT NO.

NASA-39131 #26

13. TYPE OF REPORT AND PERIOD COVERED

12/15/93 - 09/15/95

12. SPONSORING AGENCY NAME AND ADDRESS

NASA/MSFC
Marshall Space Flight Center, AL 35812

14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES

16. ABSTRACT

In this report, we present our study on multiprocessor implementation of a MPEG2 encoding algorithm. First, we compare two approaches to implementing video standards, VLSI technology and multiprocessor processing, in terms of design complexity, applications, and cost. Then we evaluate the functional modules of MPEG2 encoding process in terms of their computation time. Two crucial modules are identified based on this evaluation. Then we present our experimental study on the multiprocessor implementation of the two crucial modules. Data partitioning is used for job assignment. Experimental results show that high speedup ratio and good scalability can be achieved by using this kind of job assignment strategy.

17. KEY WORDS (SUGGESTED BY AUTHORS)	18. DISTRIBUTION STATEMENT

19. SECURITY CLASSIF. (OF THIS REPORT)	20. SECURITY CLASSIF. (OF THIS PAGE)	21. NO. OF PAGES	22. PRICE

NASA FORM 1626 OCT 86

AUBURN UNIVERSITY

Auburn, Al — **ACCOUNT NO.** 4-20559

R & D STATUS REPORT

Report No. ______

FOR THE PERIOD OF : 12/15/93 - 09/15/95 **PREPARED:** ______

CONTRACT NO. NASA - 39131 — **DELIVERY ORDER NO:** 26

DELIVERY ORDER TITLE: Techniques for Video Compression

RESEARCH ACTIVITIES PERFORMED:

In this report, we present our study on multiprocessor implementation of a MPEG2 encoding algorithm. First, we compare two approaches to implementing video standards, VLSI technology and multiprocessor processing, in terms of design complexity, applications, and cost. Then we evaluate the functional modules of MPEG2 encoding process in terms of their computation time. Two crucial modules are identified based on this evaluation. Then we present our experimental study on the multiprocessor implementation of the two crucial modules. Data partitioning is used for job assignment. Experimental results show that high speedup ratio and good scalability can be achieved by using this kind of job assignment strategy.

PROBLEMS ENCOUNTERED:

RESEARCH ACTIVITIES PLANNED NEXT MONTH:

Principal Investigator Chwan-Hwa "John" Wu — Date: 09/15/95

Prepared for: NASA/MSFC
Organization: AP29-H, CN22D, AT01,
ATTN: CC01, EM11, EB55,
NASA Scientific
Marshall Space Flight Center,
AL 35812

TPR/RA88 — Attachment can be appended

Speed Up MPEG2 Encoding Process By Multiprocessor Computation

1. Introduction

Presently, most of the implementations of video encoder or decoder are based on VLSI technology. Many video compression and decompression chips have been designed and produced [3], [4], [6], [14]. VLSI implementation has the advantage of high performance, high efficiency, low power dissipation, and even low cost for large quantities. It is suitable for video telephone, video conference, and various multimedia applications. However, it also has the disadvantage of high design complexity, non-flexibility, and high cost if the application is not in large quantities. It is well known that video compression standards like P*64 [18], and MPEG [1], [2], [15] are unbalanced compression technology. The processing required at encoder side is more than that at the decoder side. The design of the encoder is more complex than that of the decoder. In addition, the MPEG standard is also unbalanced in terms of intended applications (CD-ROM, CTV, HDTV, etc.). That is, a large number of end users needs only a decoder to decode the compressed video. Encoder is needed only by a relatively small number of video producers. This further increases the cost of the encoder. For some applications, such as video telephony and video conferencing, where both encoder and decoder must present for the system to work, performance and efficiency requirement may make it necessary to implement the encoder and decoder, or codec, with VLSI technology. For applications intended by MPEG, it is not necessary to implement the encoder with VLSI technology. In this case, a multiprocessor implementation might be a good choice for encoder.

The multiprocessor implementation has several advantages. The first advantage of this approach is its programmability. The implementation of this type is typically software programming on general purpose multiprocessor systems or symmetric multiprocessor (SMP) servers [5], [7]. This is especially good for improving and updating the compression algorithm. Since digital video compression is a very young industry and the

compression standards (P*64, MPEG1, and MPEG2) only defined the syntax of the encoded video bitstream, much room has been left for designers to improve and enhance the encoder. Dedicated VLSI implementation may have the high risk in terms of system upgrading. Due to this reason, many manufacturers have taken the approach of programmable VLSI implementation. In this approach, compression algorithms are implemented by microcode downloaded to special purpose processors that are suitable for video signal processing. One example of this VSP (Video Signal Processing) approach is IIT's VP (Vision Processor) and VC (Vision Controller) [11], and VCP (Vision Controller and Processor) [4]. However, it is still not generic enough to implement compression algorithms other than DCT (Discrete Cosine Transform) based methods. The programmability with multiprocessor implementation dose not have this limit; hence, any updated algorithm can be implemented. This means different compression algorithms or even different applications can run on the same system. Since video compression and decompression are integrated parts of multimedia applications, the capability of sharing system resources will be helpful for reducing system cost. TI's MVP (Multimedia Video Processor) [5] is a new development in this respect. For MPEG2 standard, programmability also facilitates the flexible implementation of different profiles and levels.

The second advantage of multiprocessor implementation is its scalability in terms of system performance. For the efficiency of signal processing, most available video compression algorithms process image data on the basis of uniform blocks. For instance, MPEG compresses video data macro block by macro block (MB). The processing of each MB is fully independent. Since a frame of video usually contains a large number of MBs, this provides a very good data partition basis for job assignment. If we denote the total number of available processors as N, and the number of MB's of a frame of video as M, then $N << M$ usually holds. The speed up ratio of using N processors over using uniprocessor is simply N if we divide M MB's uniformly among N processors (ignoring the communication overhead at this moment). This is significant because the value of N can vary in a relatively wide range and it can be fairly large. For applications like HDTV with high image resolutions, the amount of computation at the encoder side significant. A VLSI implementation that is designed for a predefined maximum data rate, whether

programmable or not, cannot process the data without degrading image quality (e.g. lower frame rate) if the data rate exceeds the designed one. It is impossible to increase the throughput by simply raising the chip's working clock rate. A multiprocessor implementation, however, can increase the system throughput simply by adding enough number of processors. This scalability is also helpful for upgrading the system as the encoding algorithm getting more and more intelligent and thus requiring more computation power. New functions can also be added by simply adding more processors.

The third advantage of multiprocessor implementation is that it reduces the implementation complexity. This is especially true for highly unbalanced MPEG2 [2] encoder. Once the target hardware system is available, the implementation is mainly a programming issue which is easier than the design and fabrication of a VLSI chip. If one starts with a uniprocessor program, then the parallel programming will be straight forward if the job assignment is based on data partition.

The above argument is based on the assumption that a multiprocessor hardware environment is already available. Yes, a multiprocessor hardware might have the image of bulky and expensive. However, recent development has shown that multiprocessor processing is migrating toward desktop and server systems and even receiving attention from the embedded-system community [8]. Small-scale systems with 4, 8, or 16 processors are popular [9] The high computation requirement of multimedia applications has pushed the computation toward parallelism. The aforementioned MVP from TI is a typical example. It includes a 32-bit RISC master processor and four 32-bit DSPs. It aims at real-time multimedia applications: video conferencing, video compression and decompression for MPEG1 or 2, video server, 3D graphics, etc. [5]. These developments in hardware architecture together with the above mentioned advantages make the multiprocessor implementation of video encoding algorithms a very appealing approach.

In the following, we first present our evaluation of the modules of a MPEG2 encoding process in terms of computation time in section 2. In particular, the two crucial modules, Motion Estimation (ME) and TRansform (TR) which includes DCT type estimation and forward DCT are analyzed in terms of parallel implementation. Section 3 describes our multiprocessor implementation of the ME module and the TR module. This

is followed by the experimental results and analysis in section 4. Section 5 gives a comparison of MPEG2 and Indeo Video technologies [13], [20] based on our experiments. We summarize and discuss our results in section 6.

2 Evaluation and Analysis of the Modules in MPEG2 Encoding Process

For the purpose of performance evaluation, the computation time of each modules in MPEG2 encoding process was recorded by running the program that implements the MPEG2 encoding algorithm on a single-processor machine. The uncompressed video data we used for the evaluation was two clips of frames captured, respectively, from the two given video tapes, **EFE Flight Experiment for air bubbles in a tank and glass tube** (NASA1) and the **Various Drop Physics Module test** (NASA2). The frame resolution of the digital video is 256 by 240 pixels. Table 2.1 and table 2.2 list the computation time and its corresponding percentage for each module of the MPEG2 encoder. Figure 2.1 and figure 2.2 further illustrate the percentage of computation load for each module.

In tables 2.1 and 2.2, *ME* stands for motion estimation module, *Pred* for prediction module, *TR* for DCT type estimation and forward DCT module, *Q&VLC* for quantization and variable length coding module, *IQ* for inverse quantization module, and *IDCT* for inverse DCT. From the tables and figures, it is clear that the crucial modules for the MPEG2 encoding process are the module ME and the module TR. This can also be recognized from the analysis of their computations. Figure 2.3 to 2.5 illustrate the flow charts of the module ME for frame pictures without using dual prime motion vectors (since dual prime motion vectors are only permitted for continuous P type pictures).

As seen from figure 2.3, no motion estimation is carried out for I type pictures since they are to be coded with only the information of themselves. For P type pictures, motion estimation is carried out MB by MB for both frame coding mode and field coding mode. Motion estimation is to find out, by means of some search technique, the best match of the MB in a reference picture. The search is usually limited within a window centered around the location of the MB. This best match is determined by the minimum

Table 2.1 Time and percentage by modules of MPEG2 encoding process (NASA1)

Functional	I type pictures		P type pictures		B type pictures	
Modules	Time(sec.)	Percentage	Time(sec.)	Percentage	Time(sec.)	Percentage
ME	0.05	1.2%	10.756	75.8%	8.614	72.3%
Pred	0	0.0%	0.046	0.3%	0.0825	0.7%
TR	2.609	66.1%	2.326	16.4%	2.335	19.6%
Q&VLC	0.851	21.5%	0.668	4.7%	0.567	4.8%
IQ	0.165	4.2%	0.165	1.2%	0.101	0.8%
IDCT	0.275	7.0%	0.229	1.6%	0.218	1.8%

Table 2.2 Time and percentage by modules of MPEG2 encoding process (NASA2)

Functional	I type pictures		P type pictures		B type pictures	
Modules	Time(sec.)	Percentage	Time(sec.)	Percentage	Time(sec.)	Percentage
ME	0.028	0.75%	7.754	69.7%	7.012	68.2%
Pred	0.028	0.75%	0.037	0.4%	0.078	0.7%
TR	2.397	60.4%	2.307	20.7%	2.321	22.6%
Q&VLC	0.934	24.5%	0.659	5.9%	0.540	5.3%
IQ	0.192	5.0%	0.138	1.2%	0.110	1.1%
IDCT	0.33	8.6%	0.229	2.1%	0.220	2.1%

distance defined as summed absolute differences between the pixel values of the MB and those of the reference picture and is sometimes called the prediction of the MB from the reference picture. The corresponding displacement between the MB and its best match, called Moving Vector (MV), can be used for motion compensation during decoding. The search process is a computational intensive process. To get better results by comparison, this search process is repeated for frame coding mode and various prediction combinations in field coding mode as shown in figure 2.4. This repetition gravitates the computation

load. The worst situation, however, happens when doing motion estimation for B type pictures. In this case, two reference pictures are to be searched, corresponding to *forward prediction* and *backward prediction* respectively as shown in figure 2.5. These two blocks each includes the computations corresponding to the blocks encircled by the dashed rectangle in figure 2.4. The symbols dmc, dmcftt, dmcftb, dmcft, dmcfbt, dmcfbb, dmcfb,

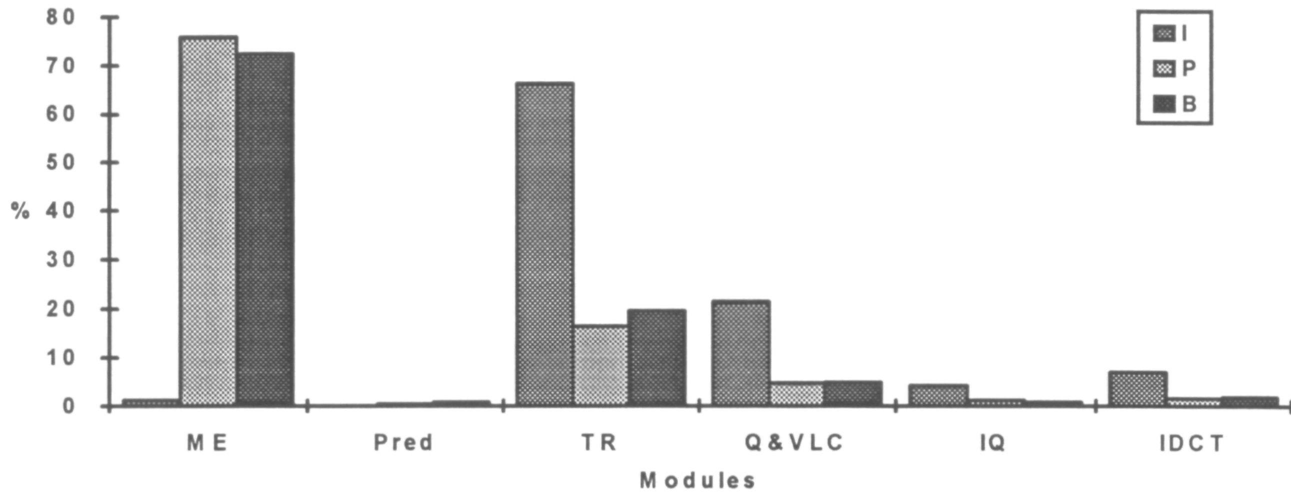


Fig. 2.1 Chart for percentage of time spent by different modules with video clip NASA1

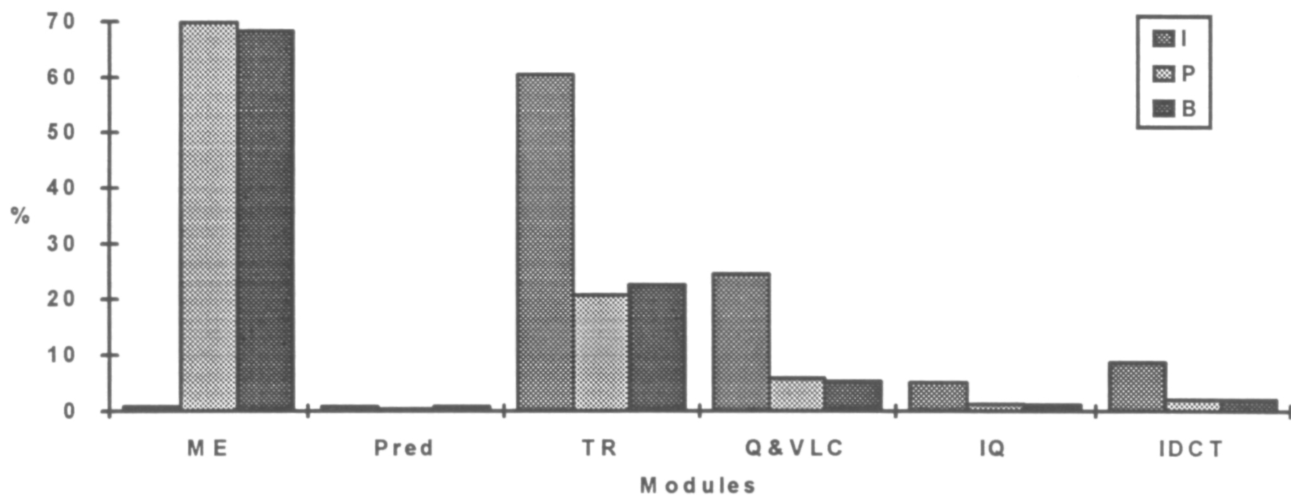


Fig. 2.2 Chart for percentage of time spent by different modules with video clip NASA2

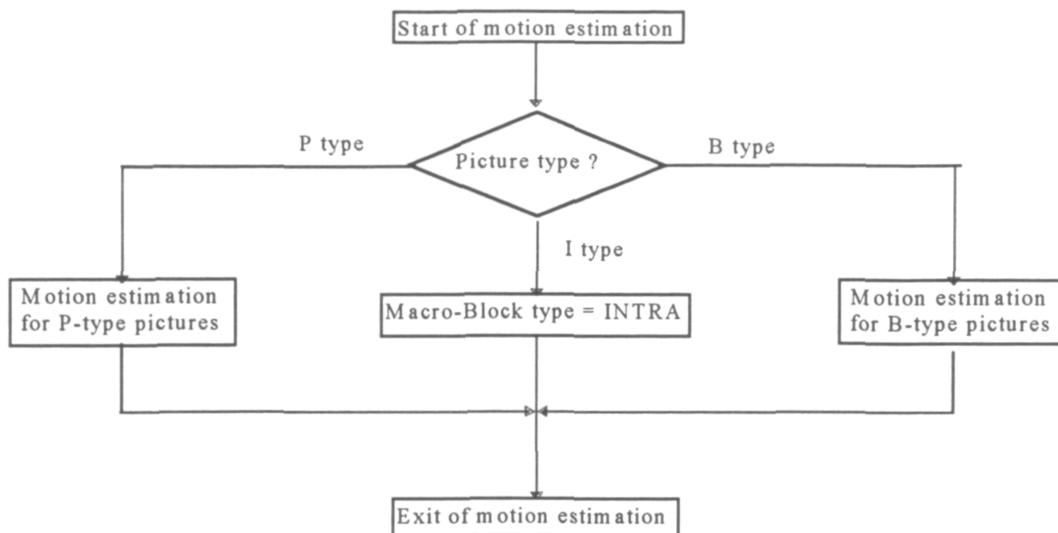


Fig. 2.3 Flow chart for motion estimation

and dmcf in figure 2.4 and the symbols dmc_f, dmc_bi, dmcf_bi_t, dmcf_bi_b, dmcf_bi, and dmcf_b in figure 2.5 are the minimum distances evaluated at various estimation modes.

To implement the function of motion estimation is more than just searching for the best match of the MB and getting the MVs. As seen from figures 2.4 and 2.5, motion estimation also includes the estimation of the variance of the MB (*Var* in the figures 2.4 and 2.5), the summed squared motion compensated prediction error (*Smcp* in the figures 2.4 and 2.5), and the summed squared prediction error without motion compensation (*Sp* in the figures 2.4 and 2.5). Decisions about the MB types such as *intra frame* or *field* (see figure 2.4 and 2.5), *prediction with motion compensation* (see figure 2.4 and 2.5), *and prediction without motion compensation* (see figure 2.4) are made based on the comparison of these evaluations. From the point of view of computation, however, most computation load comes from the search processes.

For the TR module the computation is straightforward. The computation load mainly comes from the six 8×8 double precision floating point multiplications and additions carried out in the forward DCT part for each MB.

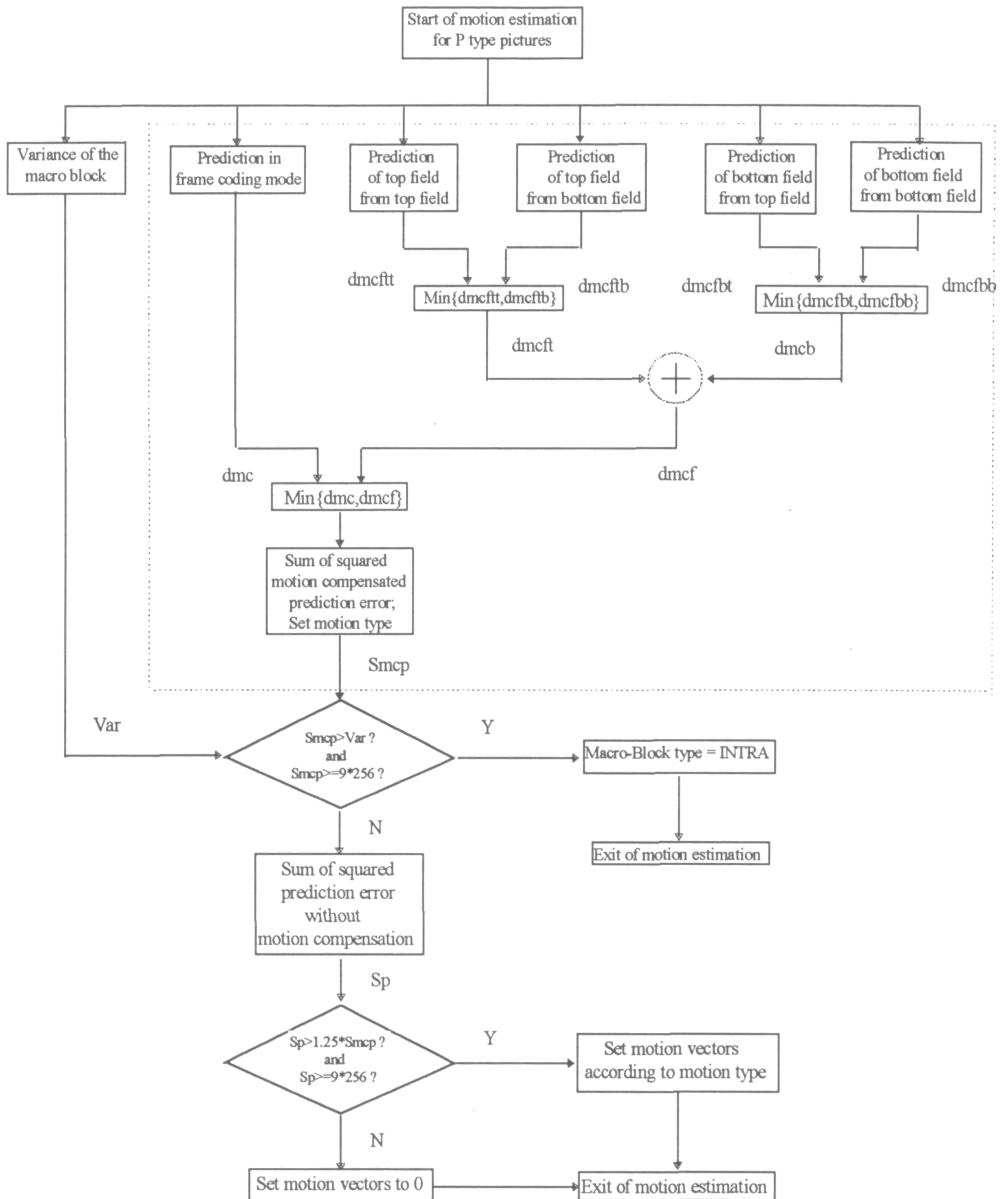


Fig. 2.4 Flow chart for motion estimation of P type pictures

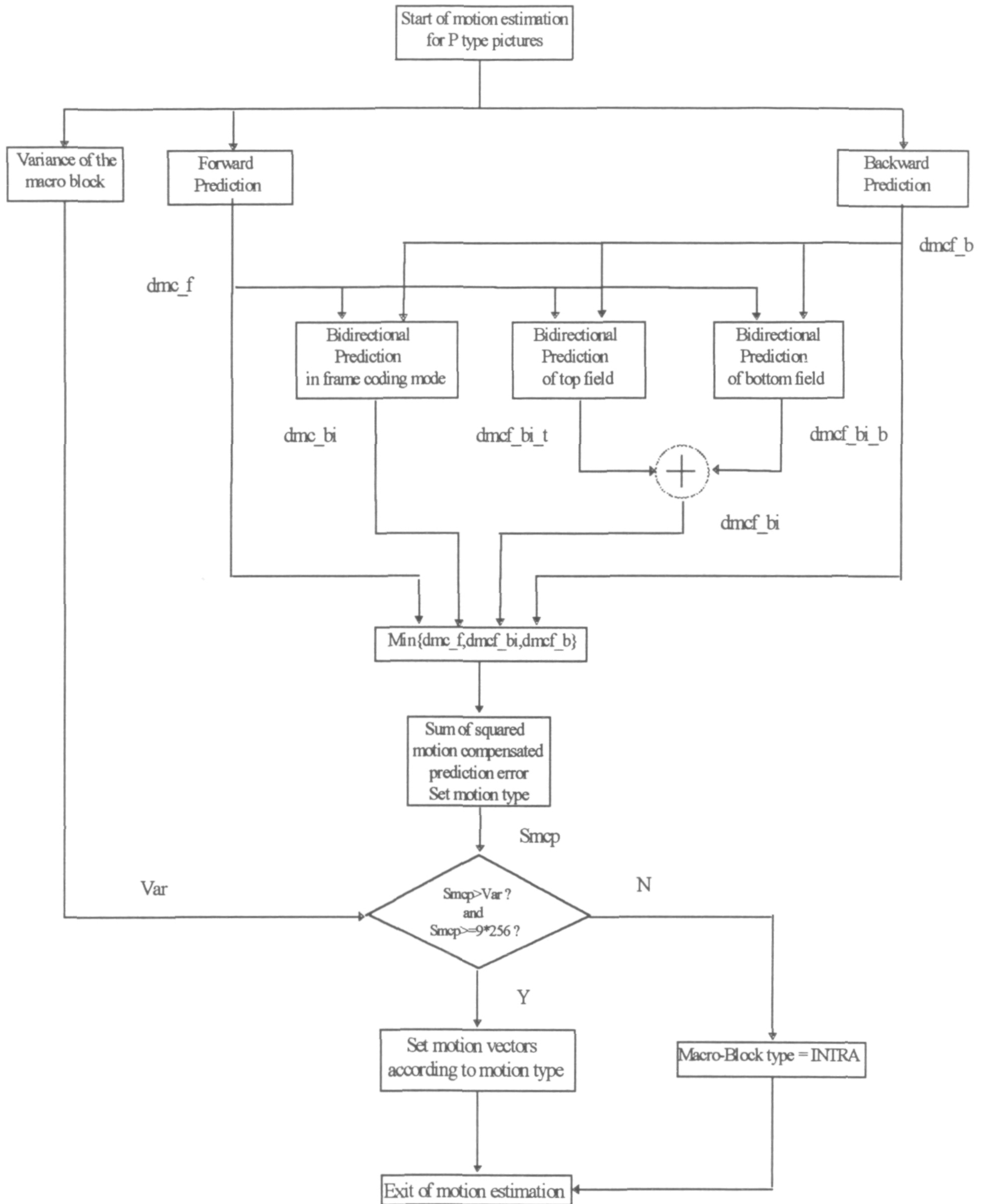


Fig. 2.5 Flow chart for motion estimation of B type pictures

Parallelism can be easily identified at various levels within the two modules. Take ME module for example, the six functional blocks at the first stage of the flow chart in the case of P type pictures in figure 2.4 can be computed in parallel. In the case of B type pictures in figure 2.5, the first stage of computation has more blocks that can be computed in parallel (remember each of the two prediction blocks in figure 2.5 contains the functional block encircled by the dashed lines in figure 2.4). Parallelism can also be observed if we go further into the lower levels. The exploitation of this kind of parallelism is called program partitioning [12]. However, caution must be taken when one tries to utilize the parallelism because the results from different parallel tasks must merge together to make comparisons, which implies that some kind of data communication will be required. If, with an implementation, the data communication takes more time than the computation does, this implementation is not useful in terms of speeding up the process.

To benefit from a specific parallel implementation of an algorithm with parallelism, one must carefully divide the task such that the job assignment well fits the underlying implementation. For the module ME we have tried to parallelize it by assigning the functional blocks (see figure 2.4 and 2.5) that can be concurrently computed to different processors of a multiprocessor computation machine. It turned out, however, that the total time spent with multiple processors was even more than that with single processor. This implied that the Inter Processor Communication (IPC) time spent was more than the computation time. The granularity of this program partitioning does not fit the computation power of the individual processor of the multiprocessor system. Even if the granularity well fits the computation power, one could not expect a high speedup with the limited parallelism in the module ME. This is clear from figures 2.4 and 2.5. It is not difficult to see that similar conclusion could also be made if we tried to parallelize the TR module by program partitioning. The above analyses suggest that program partitioning is not a good way of parallelizing the ME and TR modules.

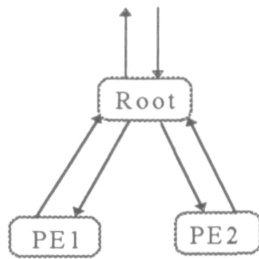
3. Multiprocessor Implementation of The ME module and TR module

Based on the discussion in previous sections, we chose to study the multiprocessor implementation of MPEG2 encoding algorithm. Because of their dominance in computation time (see Figs. 2.1 and 2.2), we only considered the two crucial modules, ME and TR, in our study.

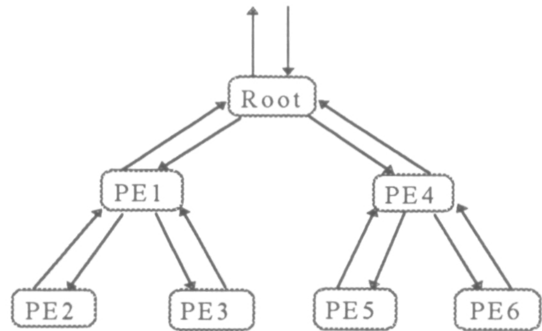
The study is carried out on a multiprocessor computation machine built with INMOS T805 processors on an IBM PC compatible host computer. The processing speed of the processor is 25 MHz. It has 4 bi-directional links that can be used to connect it with other processors. The bandwidth of each link is 20 M bits/s. The processes on a processor may communicate with the processes on other processors by sending and receiving messages through the corresponding links. Extreme care must be taken when one programs the communication because a miss match in the type or the number of transmission unit would result in a deadlock. All the communications with the outside world must be done through a root processor that is connected with the host computer. This makes it difficult to debug the programs on all the processors other than the root processor. The program can be written in C or FORTRAN. In this study we used the Parallel C Version 2.2.2 by 3L Ltd.

In order to gain a high speedup, we did our parallel implementation by data partitioning. That is, to have the processors run the same program but process different parts of a whole data set. This is possible because the compression algorithm processes a frame of video MB by MB independently. As discussed in section 1, dividing MBs among processors will result in high speed up and good scalability. In addition, it also contributes to the development and maintainability of the parallel program since the programs on most of the processors would be much the same except for the communication parts.

Specifically, we divided M MBs uniformly among N processors. The picture resolution used for this study was 256 by 240. A MB is a square block of 16 by 16 pixels. This results in a total number of MBs $M = 256$. We did experiments with different values



a. N = 3; a processor processes 1/3 of the MBs in a frame.



b. N = 7; a processor processes 1/7 of the MBs in a frame.

Fig. 3.1 Configurations of porcessors for different values of N. Each processor runs the same algorithm but processes 1/N part of a frame of image data.

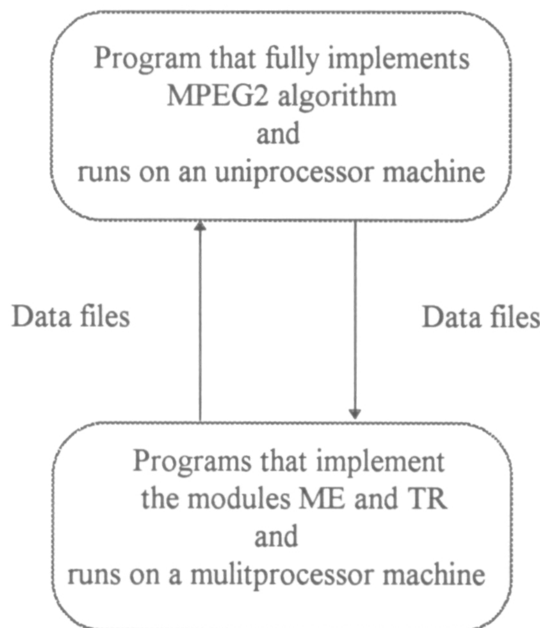


Fig. 3.2 Computation evironment for parallel implementation of the modules ME and TR.

of N. For $N = 1$, a processor would process 256 MBs. For $N = 3$, two of the processors would process 85 MBs and one of them processes 86. For $N = 7$, 3 of them process 36

MBs and 4 of them process 37. Due to the same reason, the parallelization for module TR was also done by data partitioning in the same way.

The MPEG2 standard was designed to deal with both frame type pictures and field type pictures. To simplify parallel programming and debugging, we only considered frame type pictures in our study on the parallel implementation. However, we included the field coding mode for frame type pictures in the study.

The processors were configured in binary tree structures. Figure 3.1 shows the configurations of the processors and the corresponding data partitioning for each frame of image when $N = 3$ and 7.

The code running on each processor is the same as the corresponding part of the uniprocessor program except for the instructions for data communication. The root processor, however, has some extra code for input and output purpose. Since the implementation is only for two modules of the encoder, the parallel program could not run independently without a proper "environment". To provide the necessary environmental data for the two modules, files were used to record the necessary data for the two modules from the running uniprocessor program. When the parallel program is running, it reads the data from the files and write its results into other files which are read in by the uniprocessor program to verify the correctness of the parallel implementation. This process is illustrated by the diagram shown in figure 3.2.

For the ME module, several reference frames (only the luminance Y component) have to be sent through the communication links from the root processor to each internal processor so that the motion estimation could be carried out properly. For TR module, two frames of Y, U, and V components is also to be sent to each processor to carry out the DCT type estimation as well as DCT transform. In addition, results from internal processors must be sent through the links to the root processor. As will be seen in the next section, these constituted a large amount of communication.

4. Experimental Results

The experiments on the parallel implementation of the two modules of ME and TR were carried out with two clips of video and each contains 20 frames of pictures captured from the two given tapes, **EFE Flight Experiment for air bubbles in a tank and glass tubes** (NASA1) and the **Various Drop Physics Module test** (NASA2), at the resolution of 256 by 240. The frame rate that could be reached at this resolution is about 5 frames per second (fps). Due to memory limitation, long period video clips with large number of frames of relatively high resolutions could only be captured at relatively low frame rate so that the captured image data could be written to the extended memory by means of DOS RAMDRIVE function.

The communication times and computation times for different number of processors were recorded for both clips. These are listed in Tables 5.1-5.4. In the tables the *N* is the total number of processors used in the computation. The recorded times under the column of $N = 1$ are the computation times that the corresponding module spent for computation with single processor implementation. Thus the data under this column is served as the basis for the evaluation of speed up with multiprocessor implementations. The observed processing times under the columns of *Total* for $N = 3$ and $N = 7$ are the times that the module spent for both computation and communication. To analyze the effect of communication time on the speed up, the computation times and the communication times were also recorded. However, care must be taken in interpreting these times because they were only the observed ones at the root processor. For example, the total processing time may not be equal to the sum of computation time and the communication time due to the overlaps between the computation time and the communication time at different processors. From these tables following observation can be made:

1) The computation time decreases with the increase of processor number *N* since the job assigned to each individual processor is getting less with larger values of *N*. On the other hand, the communication time increases with the increase of processor number N since the communication is more with larger values of *N*.

Table 4.1 Observed times for module ME with video clip NASA1

Frame # & Type	Observed Times (in 64 micro second)						
	N = 1	N = 3			N = 7		
	Comp.	Comp.	Comm.	Total	Comp.	Comm.	Total
0/I	9298	3758	353	4119	1594	456	2052
1/B	1726499	608208	13666	623555	222634	87616	310261
2/B	1667899	564497	25226	589733	223656	60216	283883
3/P	2176734	761409	20162	781586	266318	126774	393106
4/B	1704901	594757	14331	610033	222658	75855	298521
5/B	1709886	605068	13666	620390	227556	71925	299494
6/P	2245627	819969	8340	829329	282689	129278	411974
7/B	1681560	580457	18730	599201	220872	70019	290976
8/B	1712448	607102	13666	622436	223284	84928	308224
9/P	2158411	780529	8340	789916	290949	93699	384662
10/B	1685779	597337	13666	612623	223260	77911	301182
11/B	1740238	604031	17806	621843	223977	84129	308115
12/P	2071705	737668	8350	747027	277342	89206	366561
13/B	1715725	603100	13666	618391	217243	93271	310525
14/B	1672303	580850	13666	596138	222976	77022	290010
15/I	9297	3761	353	4115	1598	457	2051
16/B	1707198	608378	13666	623692	224936	79215	304159
17/B	1703552	609250	13666	624568	226959	77586	304556
18/P	2245690	807575	8340	816961	282624	192532	412759
19/P	2161123	789158	7351	795845	280270	107748	388028

Table 4.2 Observed times for module TR with video clip NASA1

Frame # & Type	Observed Times (in 64 micro second)						
	N = 1	N = 3			N = 7		
	Comp.	Comp.	Comm.	Total	Comp.	Comm.	Total
0/I	212095	73368	20468	93836	29805	37591	67457
1/B	210362	72599	20468	93067	29490	37658	67210
2/B	210290	72511	20467	92977	29474	37623	67157
3/P	210471	72667	20468	93134	29538	37608	67207
4/B	210367	72611	20468	93078	29498	37640	67199
5/B	210331	72576	20468	93043	29495	37621	67179
6/P	210616	72702	20467	93169	29533	37645	67241
7/B	210412	72630	20468	93096	29511	37581	67201
8/B	210354	72597	20467	93066	29489	37646	67198
9/P	210486	72662	20468	93128	29549	37581	67193
10/B	210234	72558	20467	93027	29479	37645	67186
11/B	210316	72579	20467	93045	29503	37613	67177
12/P	210368	72624	20467	93090	29538	37575	67174
13/B	210193	72568	20467	93035	29496	37620	67177
14/B	210140	72558	20468	93025	29504	37597	67162
15/I	212089	73365	20468	93831	29806	37594	67462
16/B	210192	72568	20468	93036	29494	37722	67178
17/B	210316	72613	20468	93083	29533	37586	67179
18/P	210424	72674	20467	93142	29564	37589	67214
19/P	210631	72746	20468	93213	29570	37595	67228

Table 4.3 Observed times for module ME with video clip NASA2

Frame # & Type	Observed Times (in 64 micro second)						
	N = 1	N = 3			N = 7		
	Comp.	Comp.	Comm.	Total	Comp.	Comm.	Total
0/I	11036	3759	353	4117	1594	457	1757
1/B	1329425	411151	106898	518058	156552	91181	247333
2/B	1329300	411504	99104	510619	159552	86394	245230
3/P	1532219	462179	171375	633565	166616	138722	305144
4/B	1313115	427000	79853	506863	167279	78029	244909
5/B	1409249	460962	82179	543155	175841	88539	263981
6/P	1584186	507640	155232	662882	179109	141350	320264
7/B	1319275	417844	93692	511624	158928	91933	250537
8/B	1336928	415553	109288	524848	162610	86055	248266
9/P	1500753	463243	149203	612459	166069	131378	297251
10/B	1292721	597337	99308	505557	155853	86208	241662
11/B	1350369	420413	104466	524889	159085	93553	252246
12/P	1437746	416761	187726	604500	158214	129354	286375
13/B	1357313	429484	87307	516800	161506	84393	245501
14/B	1284958	399774	93883	493666	155391	82502	237497
15/I	11036	3762	353	4113	1590	457	1758
16/B	1361928	445798	60995	506805	168397	86858	254857
17/B	1297323	396599	123732	520341	156135	87560	243193
18/P	1502195	466425	152996	619431	165749	146695	312246
19/P	1600491	509835	116726	626571	181291	118312	299410

Table 4.4 Observed times for module TR with video clip NASA2

Frame # & Type	Observed Times (in 64 micro second)						
	N = 1	N = 3			N = 7		
	Comp.	Comp.	Comm.	Total	Comp.	Comm.	Total
0/I	212109	73352	20467	93817	29772	37697	67479
1/B	211025	72967	20468	93436	29667	37651	67330
2/B	211007	72963	20467	93431	29671	37636	67317
3/P	211190	73033	20468	93502	29676	37674	67360
4/B	211031	72971	20467	93440	29679	37633	67320
5/B	211029	72975	20468	93441	29679	37634	67324
6/P	211153	73014	20468	93481	29683	37653	67349
7/B	211105	73009	20468	93475	29690	37562	67335
8/B	211082	72993	20468	93460	29690	37627	67328
9/P	211215	73063	20468	93532	29696	37640	67358
10/B	211161	73044	20467	93513	29690	37655	67356
11/B	211117	73005	20467	93470	29689	37636	67335
12/P	211205	73023	20469	93492	29692	37642	67346
13/B	211126	73032	20467	93500	29680	37663	67357
14/B	211100	73001	20467	93468	29679	37648	67336
15/I	212128	73361	20468	93829	29774	37697	67479
16/B	211067	72963	20468	93431	29682	37624	67315
17/B	211043	72970	20468	93438	29680	37628	67318
18/P	211235	73067	20468	93535	29687	37669	67367
19/P	211178	73045	20467	93511	29695	37651	67356

2) For the module ME, obvious variations are observed with different picture types. The computation time spent with I type pictures is very small because no motion estimation is carried out for them. The computation time with P type pictures is even more than that with B type pictures although the computation is more complex with B type pictures than with P type ones. This is because the search window with B type pictures is smaller.

3) For the module TR, no obvious variation can be observed with different picture types.

The variations in computation time and communication time with different number of processors are further illustrated by the plots shown in figures 4.1 to 4.4 for video clip NASA1, and figures 4.7 to 4.10 for video clip NASA2. It can be seen from these figures that as the number of processors grows, the decrease of the observed total processing time slows down due to the contribution of the growing communication time. This made the observed speedup tend to saturate with the increase of the processor number as illustrated by the plots shown in figures 4.5 and 4.6 for video clip NASA1, and figure 4.11 and 4.12 for video clip NASA2. From these plots of speedup vs. processor number it is also seen that a higher Computation time to Communication time Ratio (CCR) as seen in the case of module ME has a higher speed up ratio than a lower CCR does as seen in the case of module TR.

MPEG standards provide means of controlling compressed data rate. A data rate corresponds to a quality of reconstructed video and a compression ratio. The higher the data rate, the higher the quality of the reconstructed video but the lower the compression ratio and vice versa. Table 5.1 lists the compression ratio and video quality description corresponding to different data rate. The sample reconstructed images of the two video clips are also shown in figures 5.1 through 5.6.

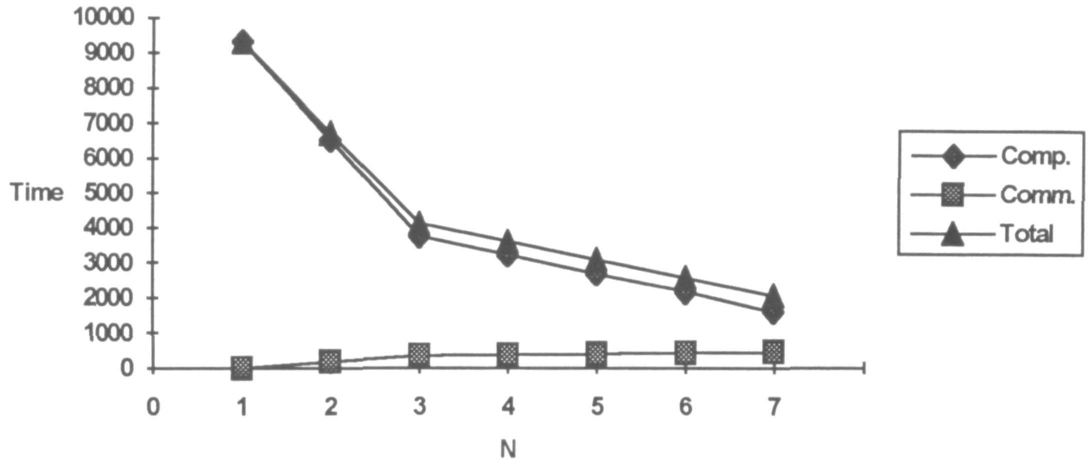


Fig. 4.1 Observed times for module ME with video clip NASA1
(time unit = 64 microseconds).

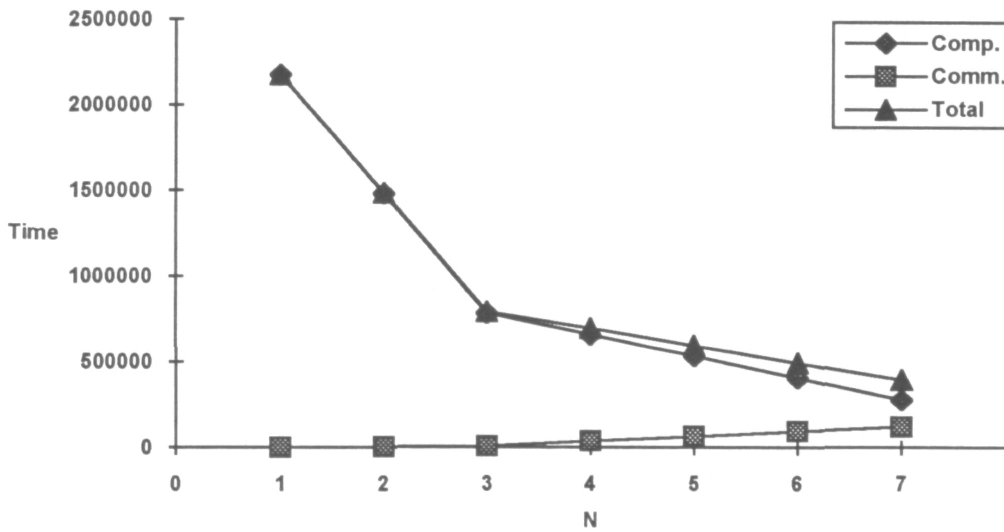


Fig. 4.2 Observed times for module ME with video clip NASA1.
(time unit = 64 microseconds)

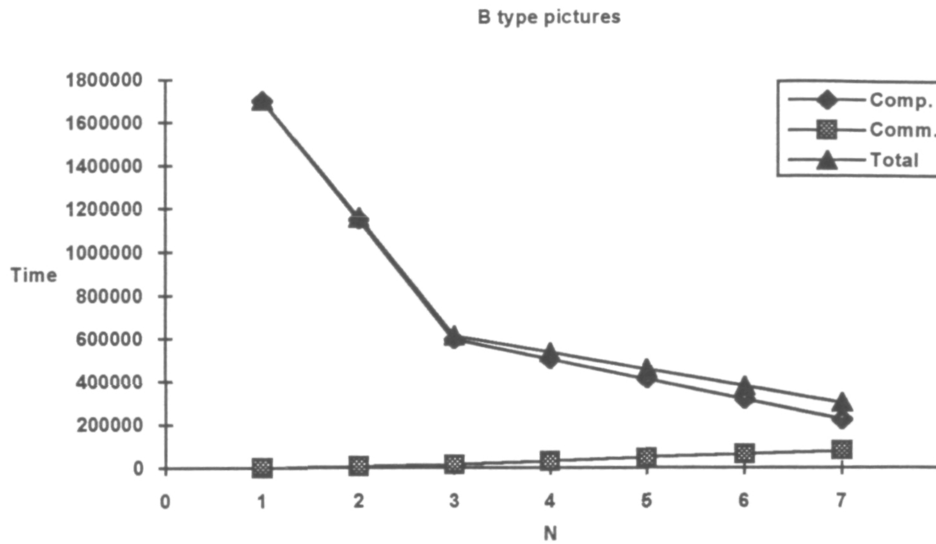


Fig. 4.3 Observed times for module ME with video clip NASA1.
(time unit = 64 microseconds)

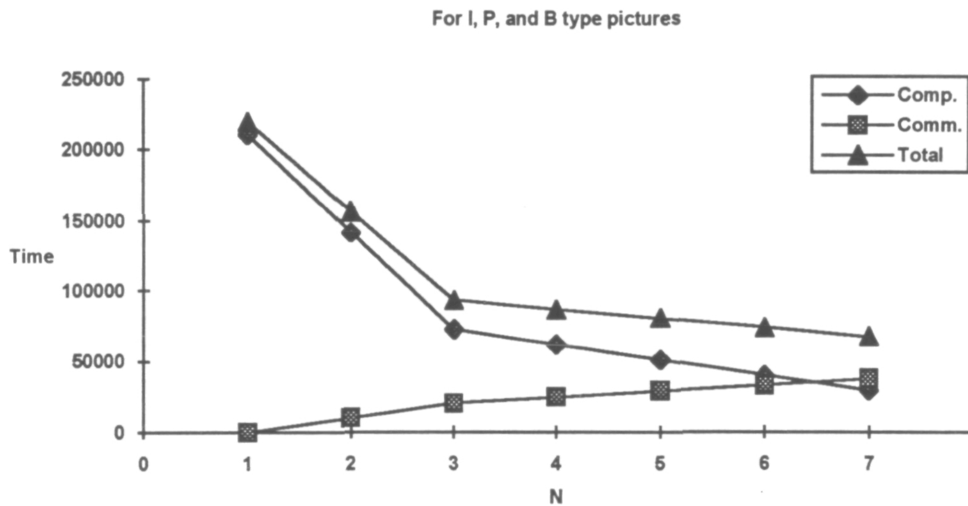


Fig. 4.4 Observed times for module TR with video clip NASA1.
(time unit = 64 microseconds)

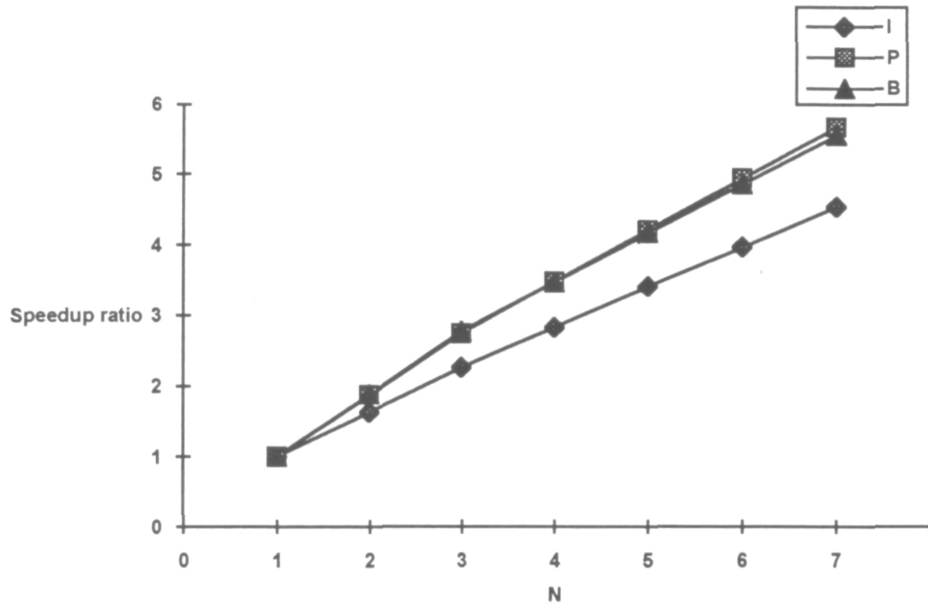


Fig. 4.5 Speed up ratio for module ME with video clip NASA1.

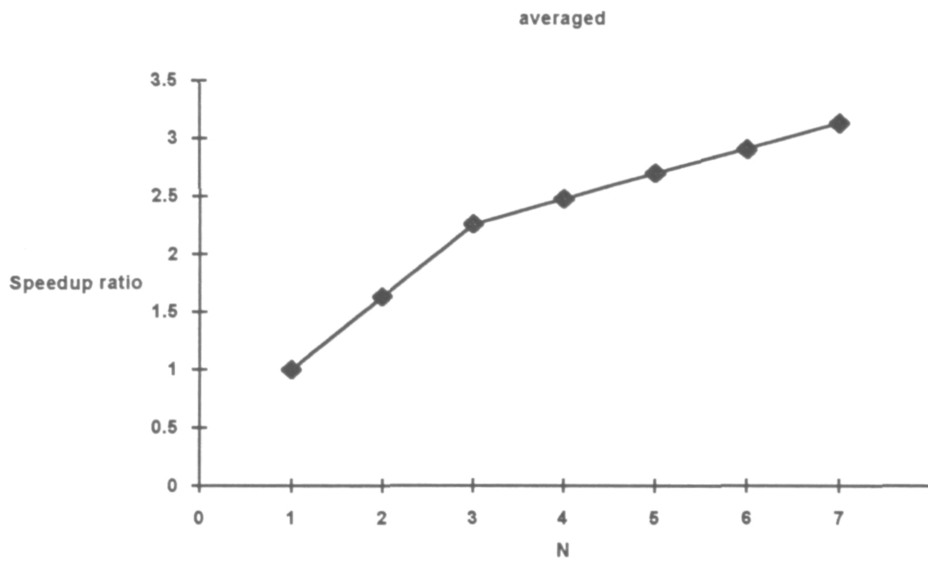


Fig. 4.6 Speed up ratio for module TR with video clip NASA1.

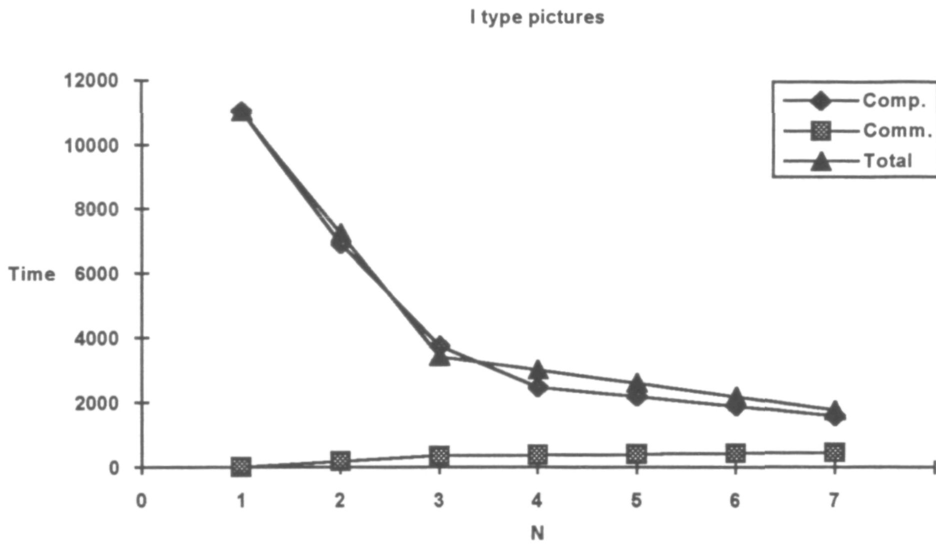


Fig. 4.7 Observed times for module ME with video clip NASA2.
(time unit = 64 microseconds)

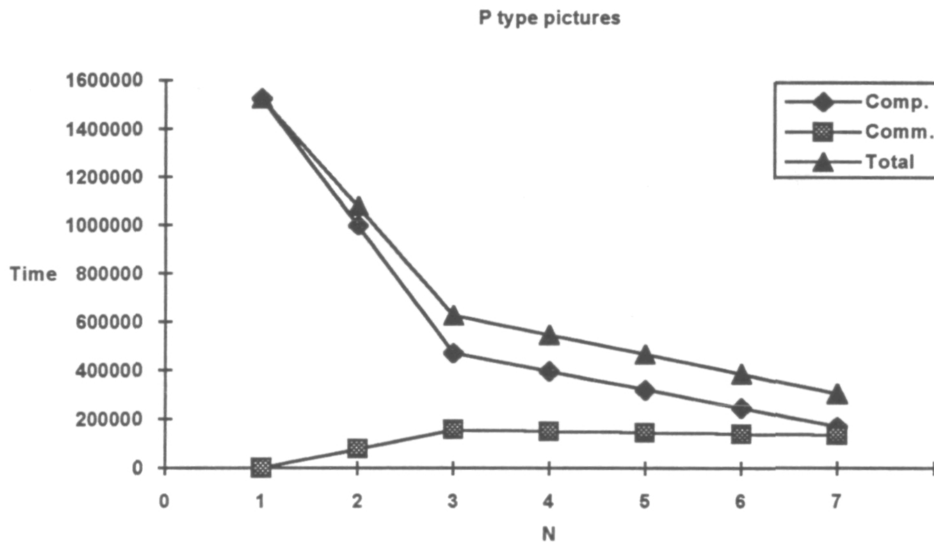


Fig. 4.8 Observed times for module ME with video clip NASA2.
(time unit = 64 microseconds)

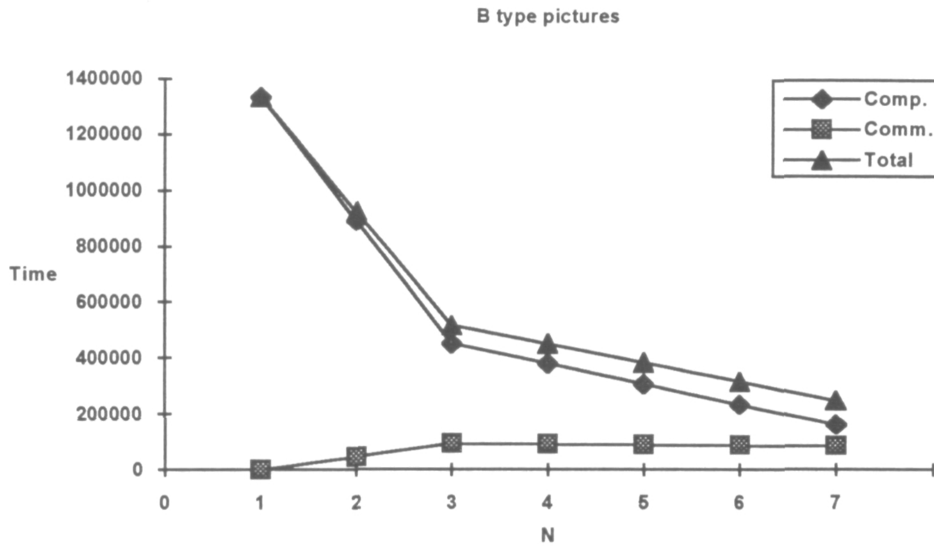


Fig. 4.9 Observed times for module ME with video clip NASA2.
(time unit = 64 microseconds)

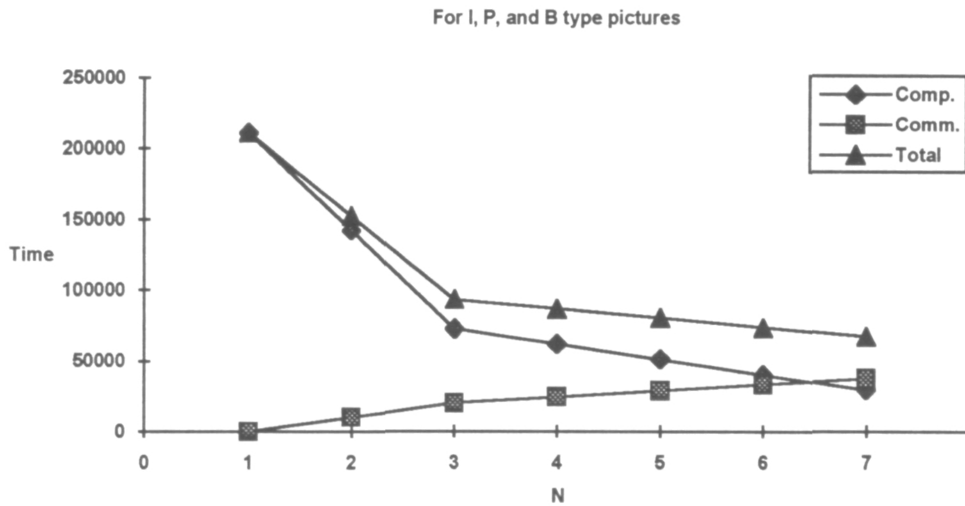


Fig. 4.10 Observed times for module TR with video clip NASA2.
(time unit = 64 microseconds)

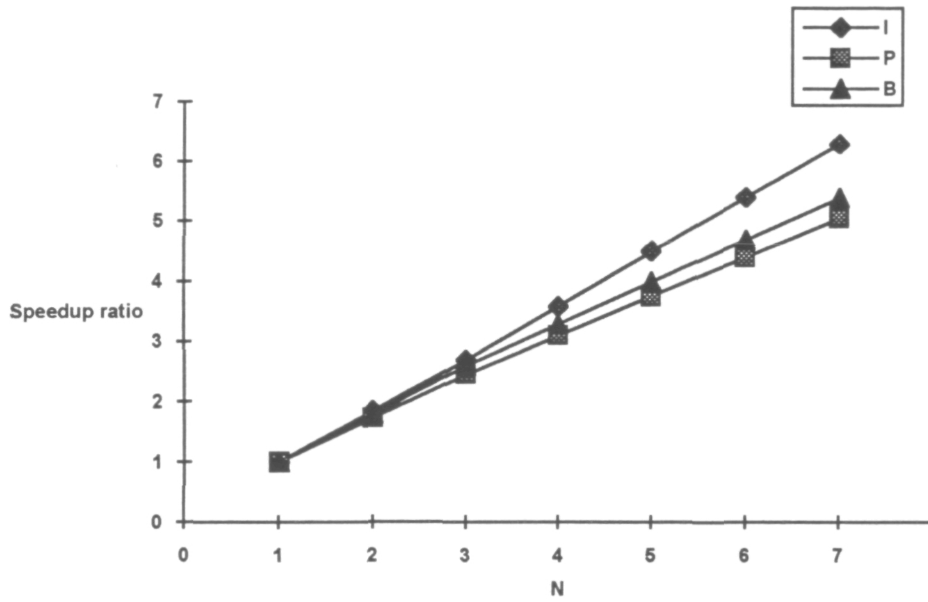


Fig. 4.11 Speed up ratio for module ME with video clip NASA2.

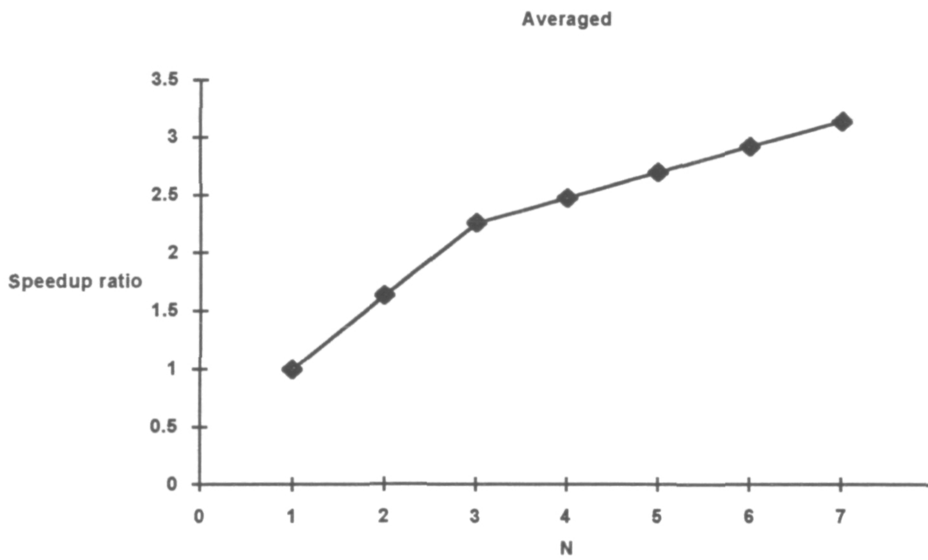


Fig. 4.12 Speed up ratio for module TR with video clip NASA2.

5. Comparison of MPEG2 with Indeo Video

MPEG and Indeo Video are two widely used video compression techniques. We have studied Indeo Video technology during this contract [13], [20]. In the following we will compare these two technologies in terms of compression ratio and reconstructed image quality. Table 5.1 and table 5.2 list the statistics we obtained through our experiments. The corresponding samples of the reconstructed as well as the corresponding original images with both technologies are shown in figures 5.1 through 5.6 for MPEG2 and figures 5.7 through 5.12 for Indeo Video respectively.

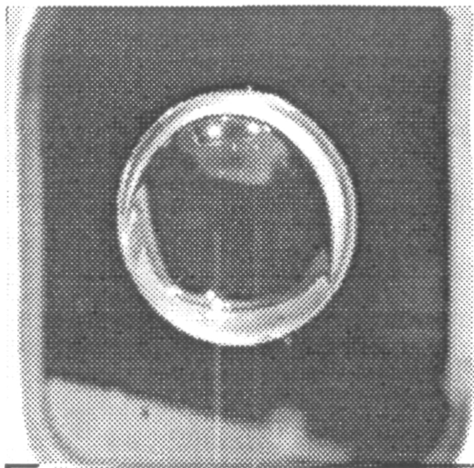
From the statistics shown in the tables and the qualities of the images shown in the figures, we can see that MPEG produced better image qualities than the Indeo Video did. Severe artifacts pop up with Indeo Video compression at moderate compression ratios where no artifacts are visible with MPEG compression. The MPEG compression produced images with reasonable qualities at fairly high compression ratios where Indeo Video compression produced unrecognizable images. Our experiments suggested that the MPEG is superiors to Indeo Video in terms of compression ratio and reconstructed image quality. In addition, MPEG is a MB oriented coding which facilitates data partitioning that leads to good scalability. Another important issue is that MPEG is an international standard that is well expected to be widely accepted by the world

Table 5.1 Compression ratios and qualities at different data rates with MPEG2.

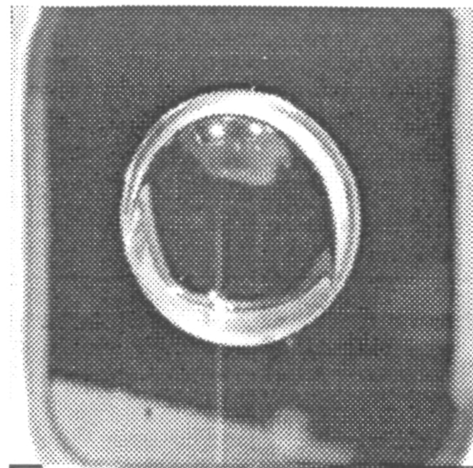
	NASA1		NASA2	
Data rate (Mbps)	Compression ratio	Quality description	Compression ratio	Quality description
4	5.53	very good	5.63	very good
2	11.05	good	11.04	good
1	22.06	invisible block artifact	22.05	invisible block artifact
0.5	44.17	visible block artifact	43.77	visible block artifact
0.1	65.40	obvious block artifact	85.44	obvious block artifact

Table 5.2 Compression ratios and qualities at different quantization levels with Indeo Video.

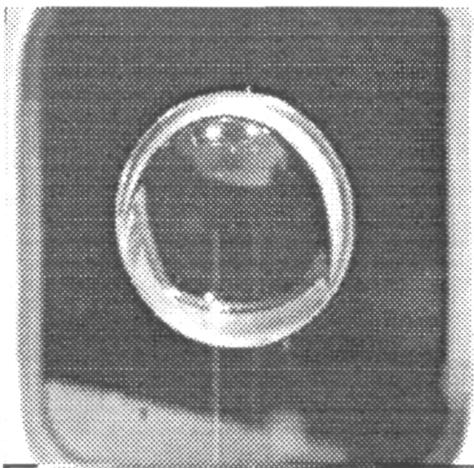
Quatiza-tion level	NASA1		NASA2	
	Compression ratio	Quality description	Compression ratio	Quality description
2	6.39	very good	5.58	very good
8	8.83	visible artifact	7.26	visible artifact
16	10.14	very bad	13.76	very bad
18	22.77	very bad	22.10	very bad



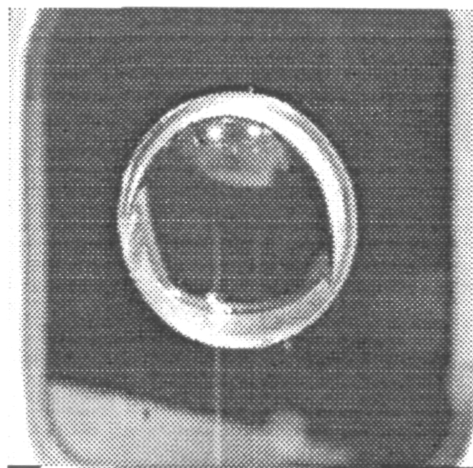
a. (Compression ratio = 0, original)



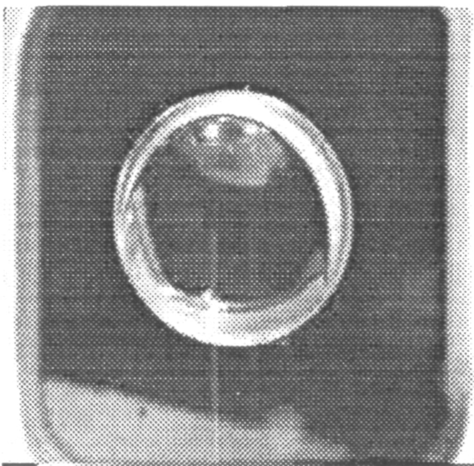
b. (Compression ratio = 5.53)



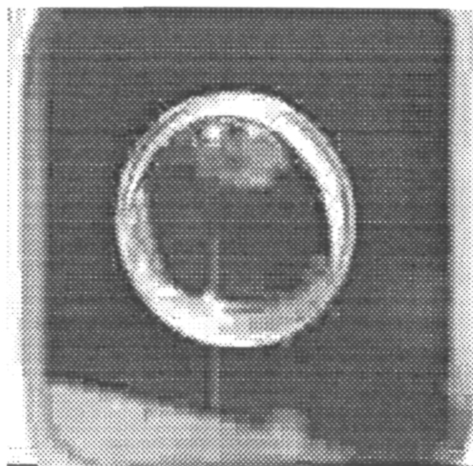
c. (Compression ratio = 11.05)



d. (Compression ratio = 22.06)

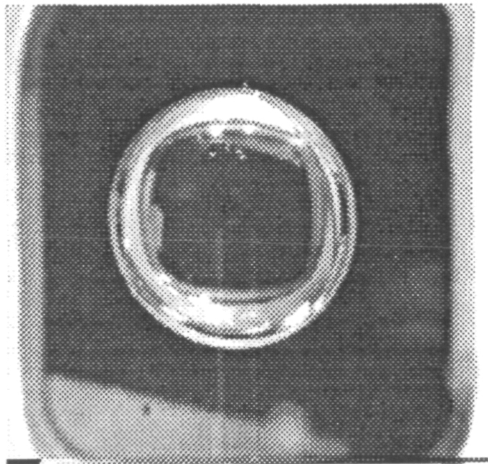


e. (Compression ratio = 44.17)

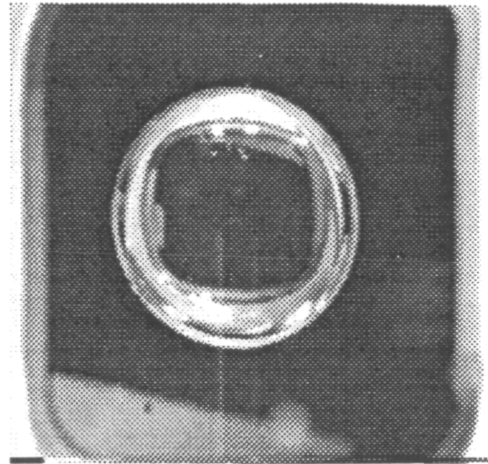


f. (Compression ratio = 65.40)

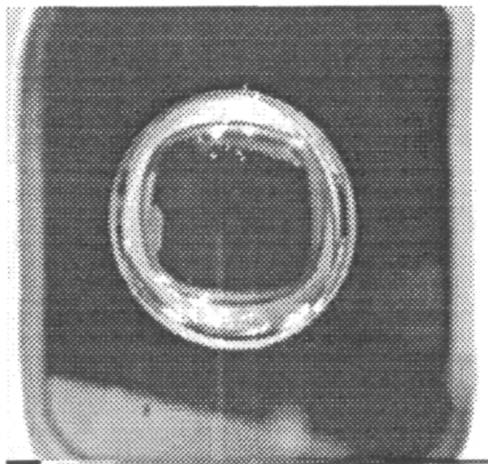
Fig. 5.1 Original & reconstructed frames for an I type picture from NASA1 with MPEG2.



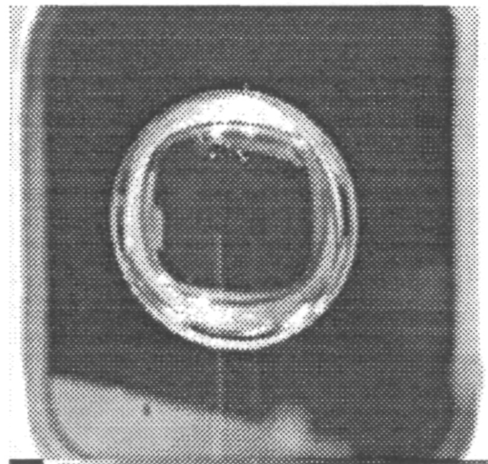
a. (Compression ratio = 0, original)



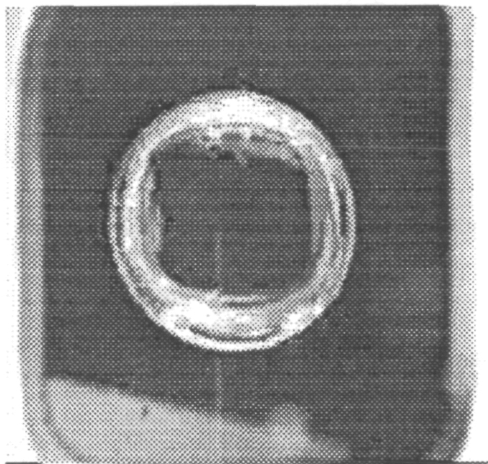
b. (Compression ratio = 5.53)



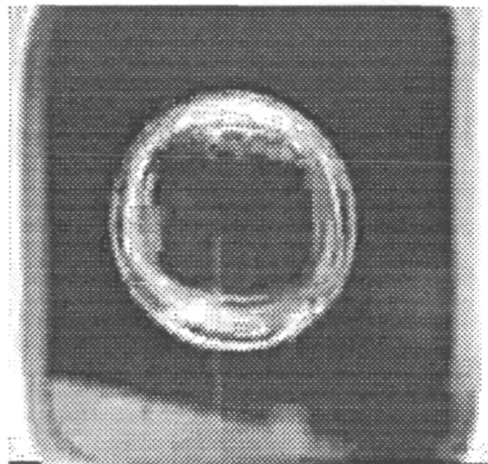
c. (Compression ratio = 11.05)



d. (Compression ratio = 22.06)

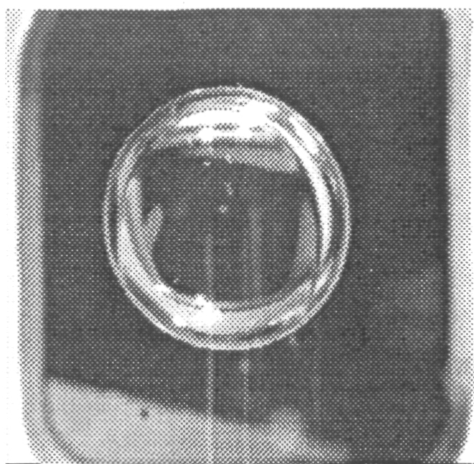


e. (Compression ratio = 44.17)

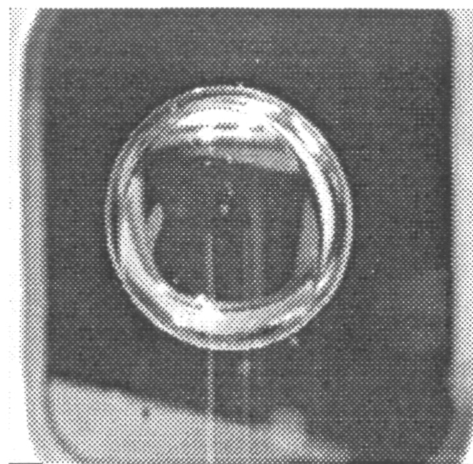


f. (Compression ratio = 65.40)

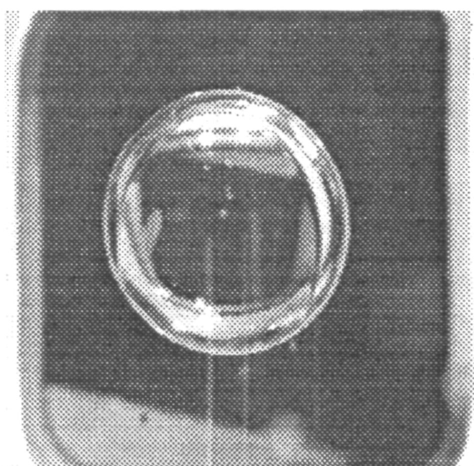
Fig. 5.2 Original & reconstructed frames for a B type picture from NASA1 with MPEG2.



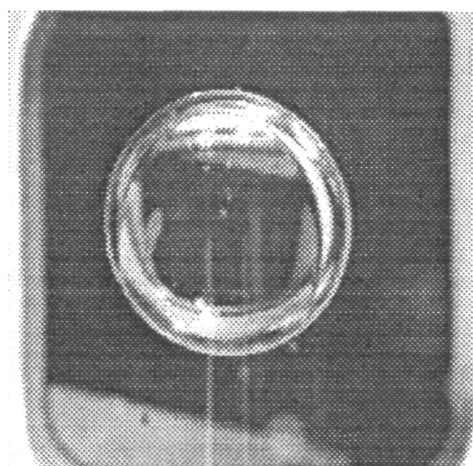
a. (Compression ratio = 0, original)



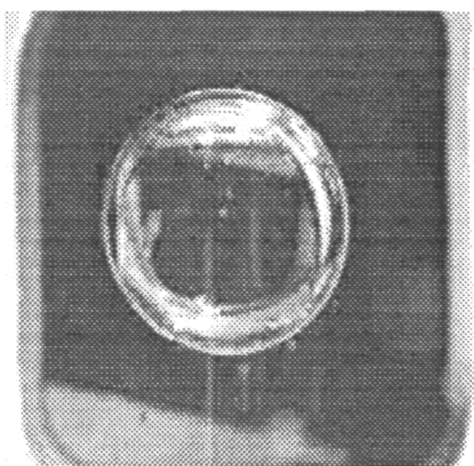
b. (Compression ratio = 5.53)



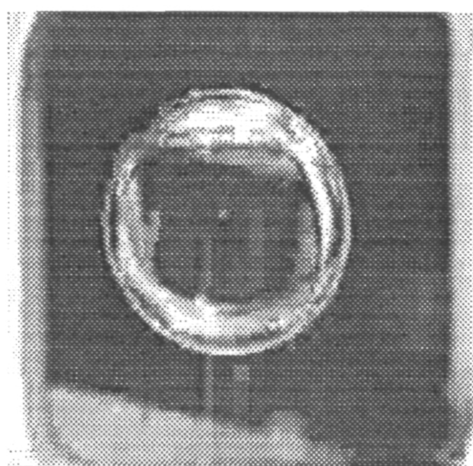
c. (Compression ratio = 11.05)



d. (Compression ratio = 22.06)

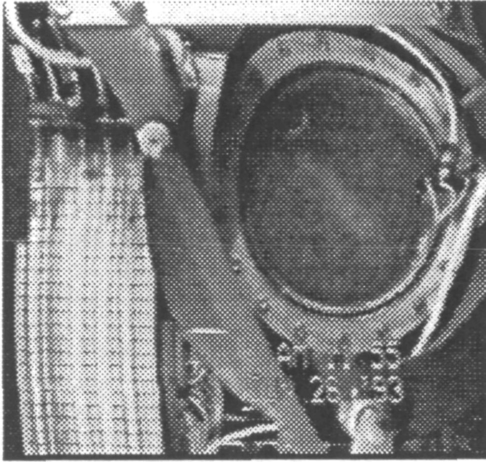


e. (Compression ratio = 44.17)

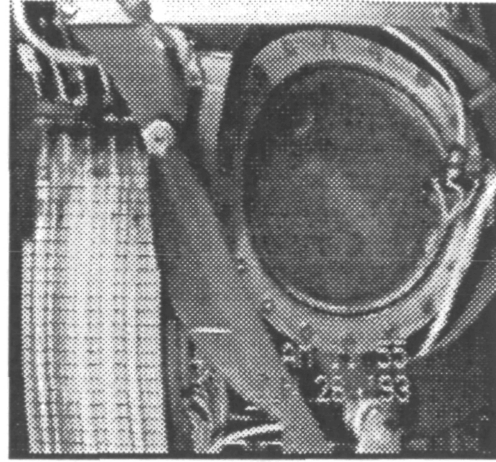


f. (Compression ratio = 65.40)

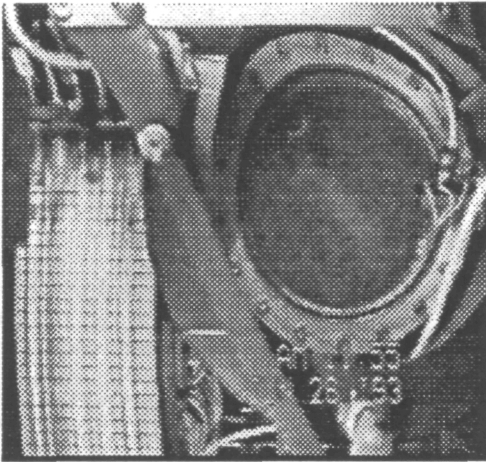
Fig. 5.3 Original & reconstructed frames for a P type picture from NASA1 with MPEG2.



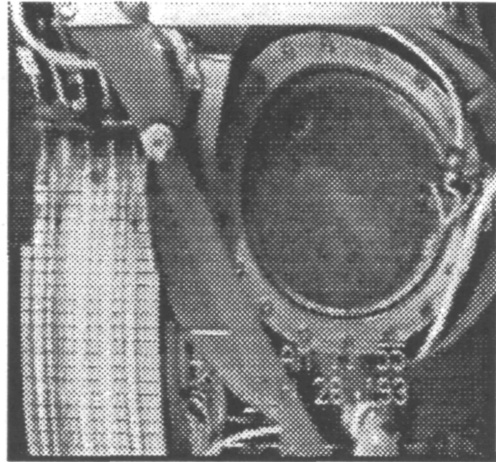
a. (Compression ratio = 0, original)



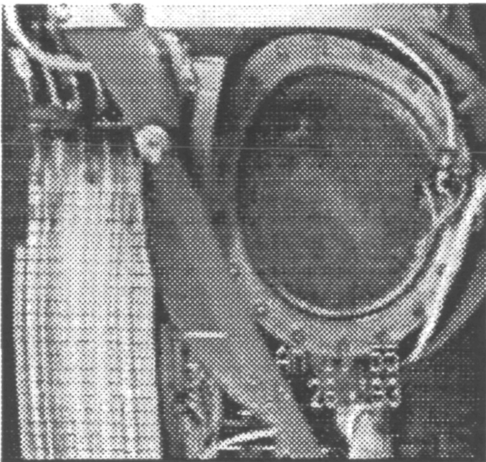
b. (Compression ratio = 5.63)



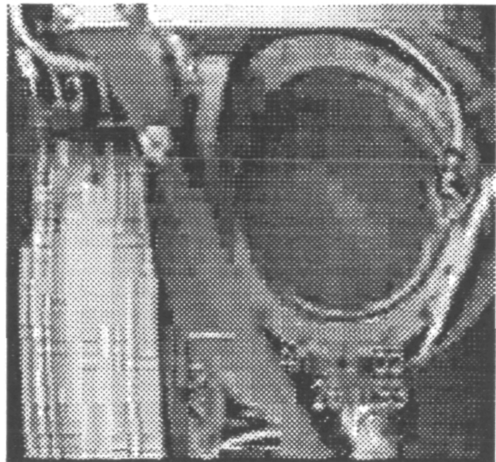
c. (Compression ratio = 11.04)



d. (Compression ratio = 22.05)

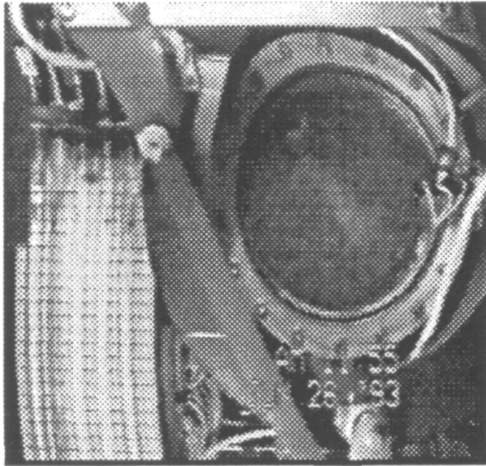


e. (Compression ratio = 43.77)

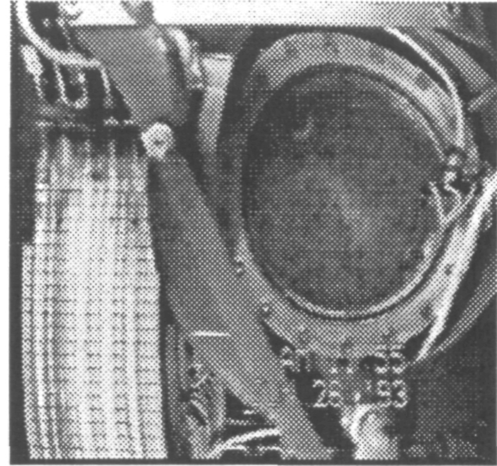


f. (Compression ratio = 85.44)

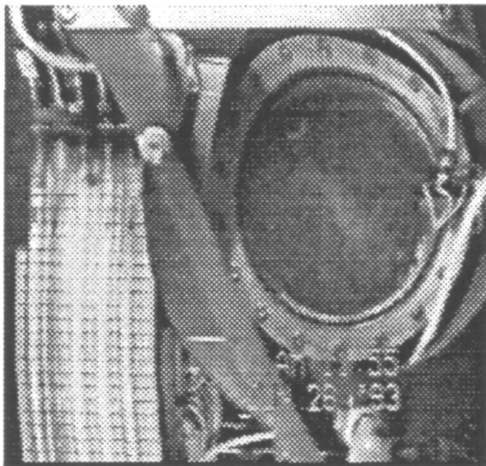
Fig. 5.4 Original & reconstructed frames for an I type picture from NASA2 with MPEG2.



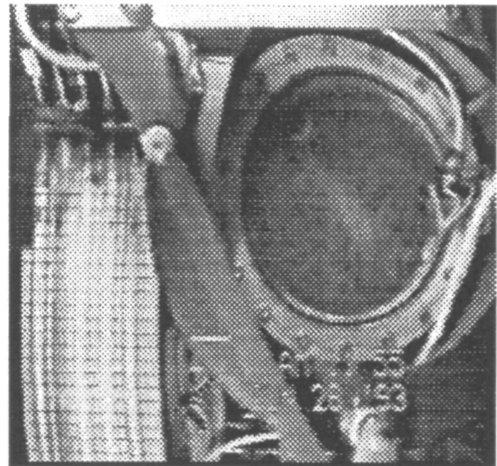
a. (Compression ratio = 0, original)



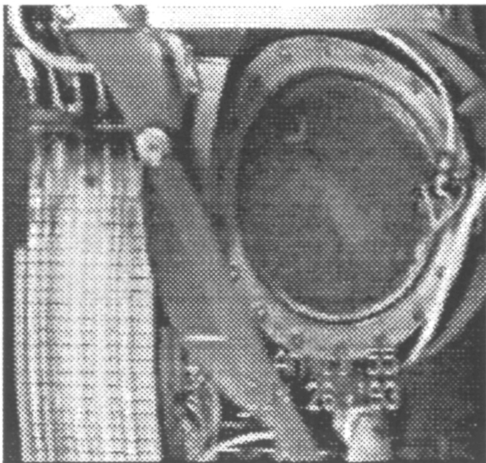
b. (Compression ratio = 5.63)



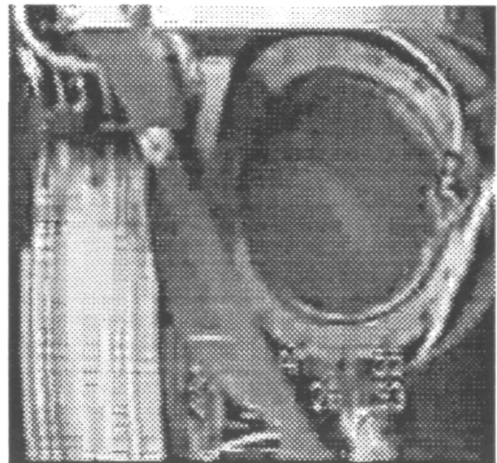
c. (Compression ratio = 11.05)



d. (Compression ratio = 22.05)

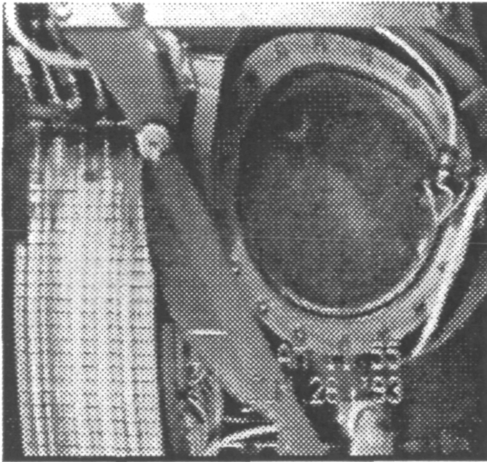


e. (Compression ratio = 44.17)

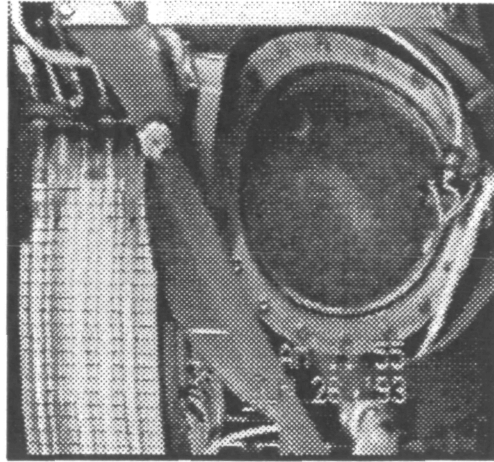


f. (Compression ratio = 85.44)

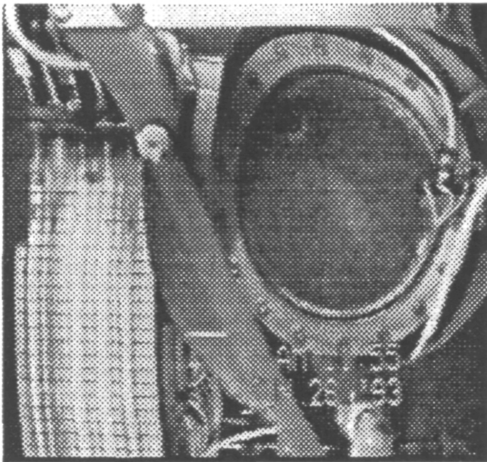
Fig. 5.5 Original & reconstructed frames for a B type picture from NASA2 with MPEG2.



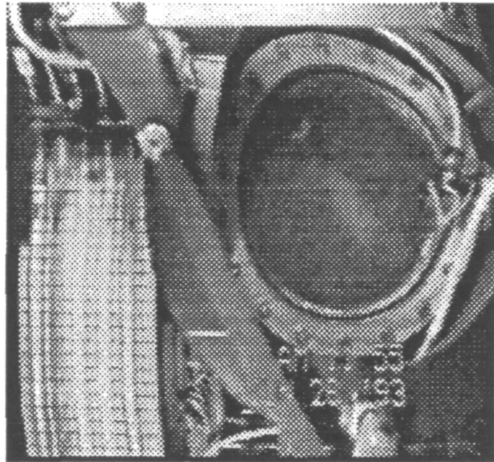
a. (Compression ratio = 0, original)



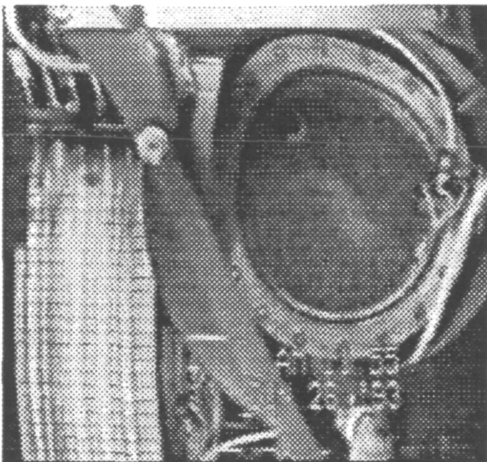
b. (Compression ratio = 5.63)



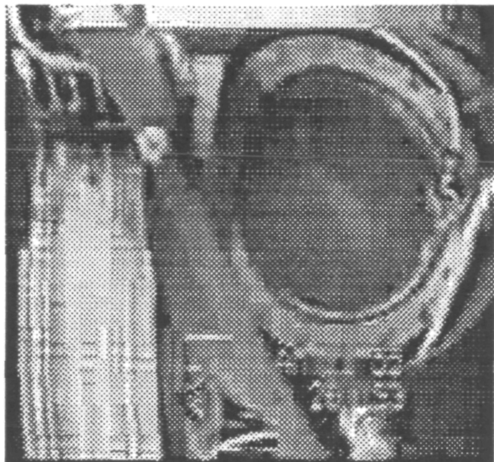
c. (Compression ratio = 11.04)



d. (Compression ratio = 22.05)

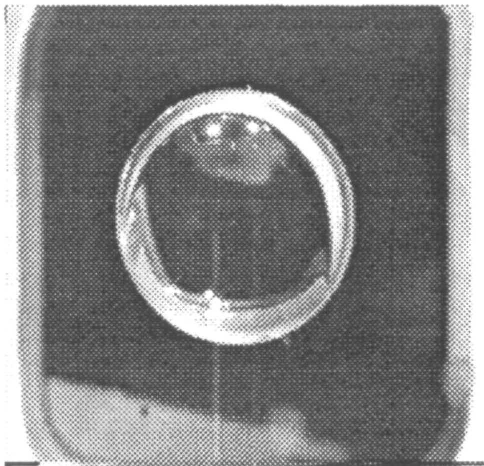


e. (Compression ratio = 43.77)

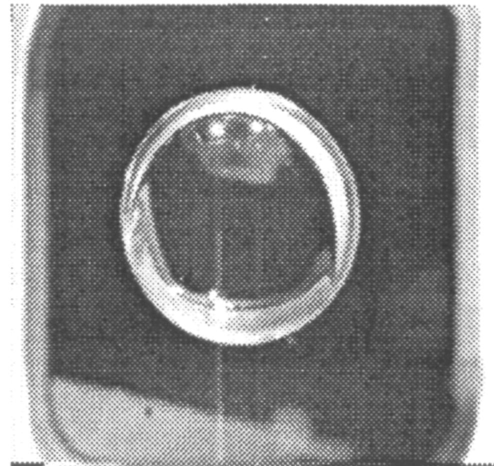


f. (Compression ratio = 85.44)

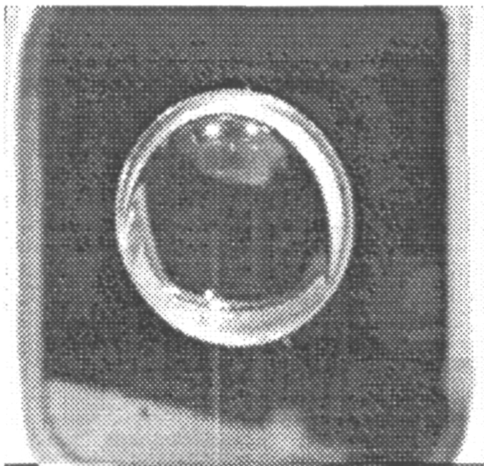
Fig. 5.6 Original & reconstructed frames for a P type picture from NASA2 with MPEG2.



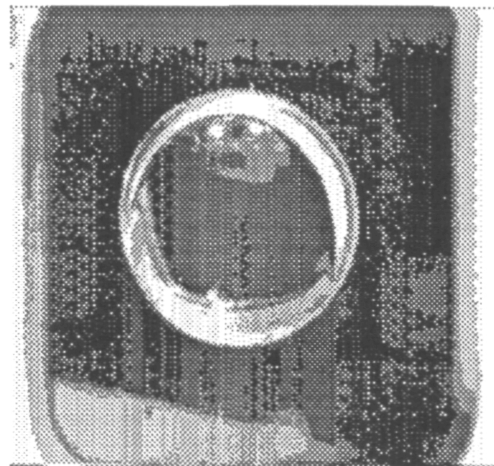
a. (Compression ratio = 0, original)



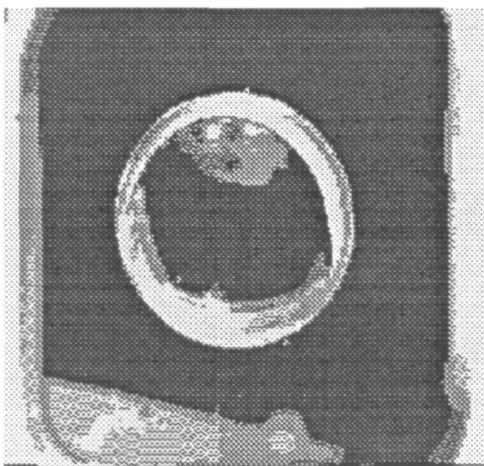
b. (Compression ratio = 6.39)



c. (Compression ratio = 8.83)

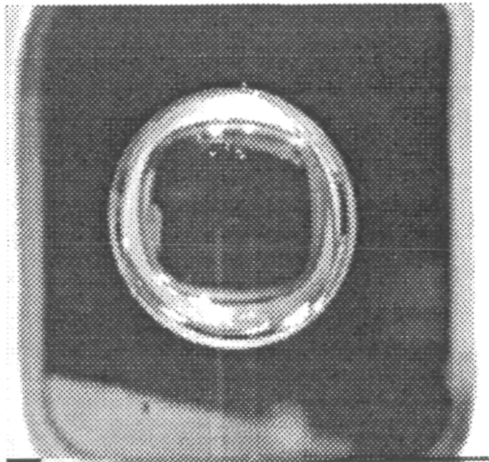


d. (Compression ratio = 10.14)

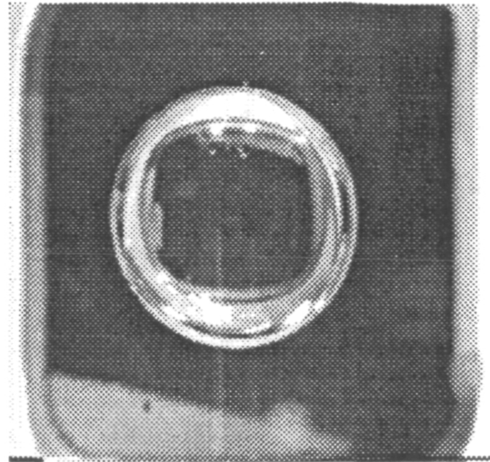


e. (Compression ratio = 22.77)

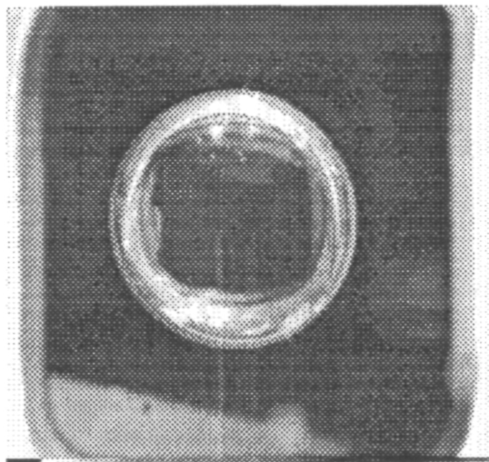
Fig. 5.7 Original & reconstructed frames for a reference picture from NASA1 with Indeo Video.



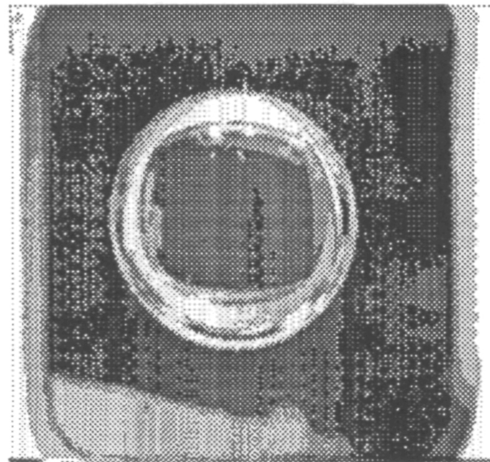
a. (Compression ratio = 0, original)



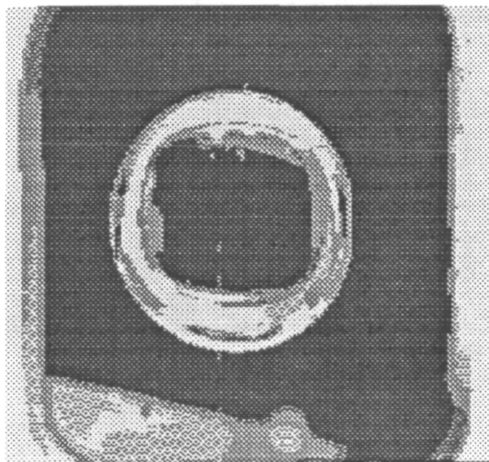
b. (Compression ratio = 6.39)



c. (Compression ratio = 8.83)

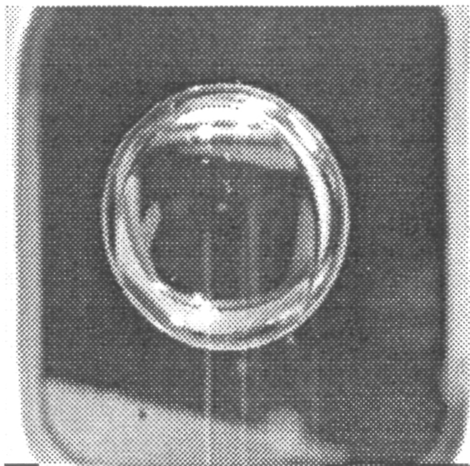


d. (Compression ratio = 10.14)

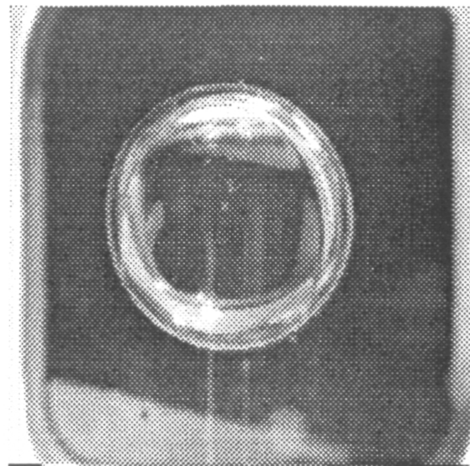


e. (Compression ratio = 22.77)

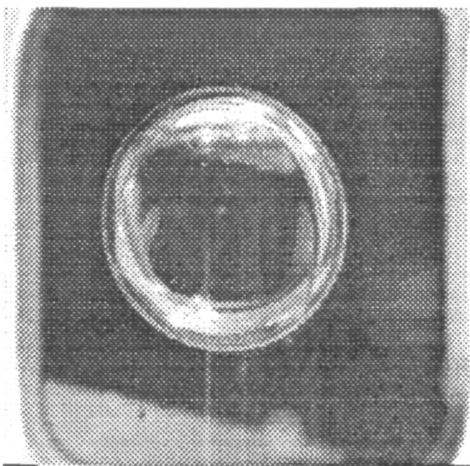
Fig. 5.8 Original & reconstructed frames for a relative picture from NASA1 with Indeo Video.



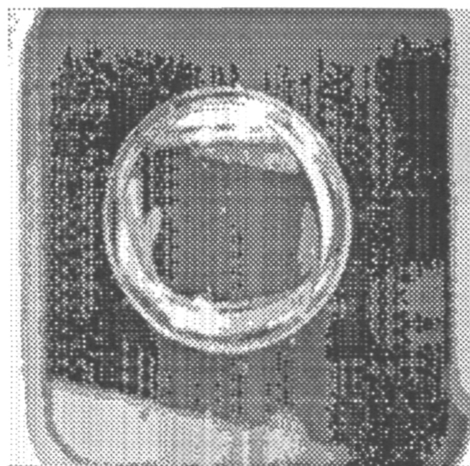
a. (Compression ratio = 0, original)



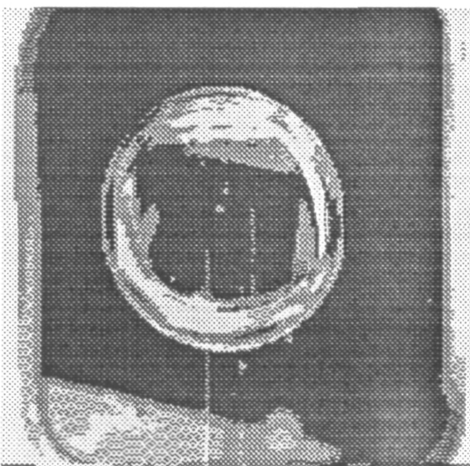
b. (Compression ratio = 6.39)



c. (Compression ratio = 8.83)

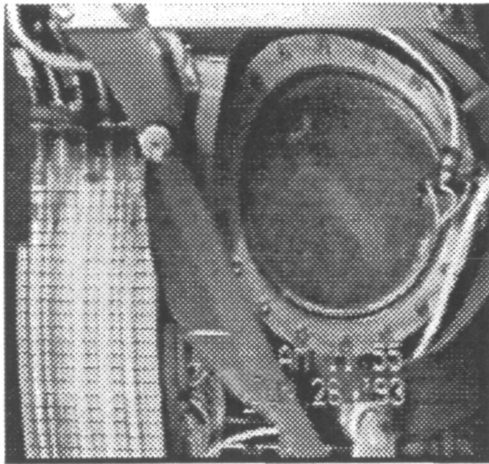


d. (Compression ratio = 10.14)

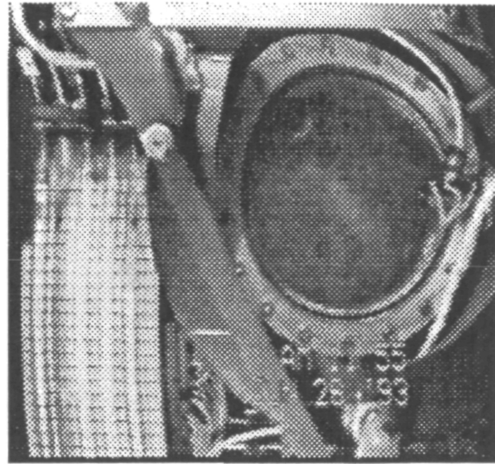


e. (Compression ratio = 22.77)

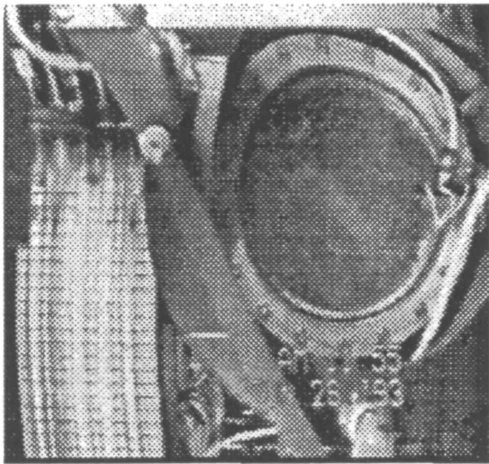
Fig. 5.9 Original & reconstructed frames for a relative picture from NASA1 with Indeo Video.



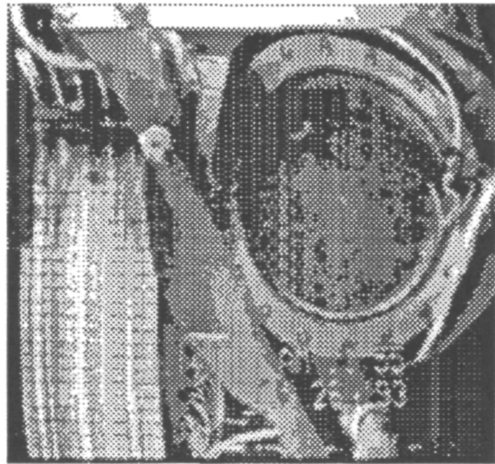
a. (Compression ratio = 0, original)



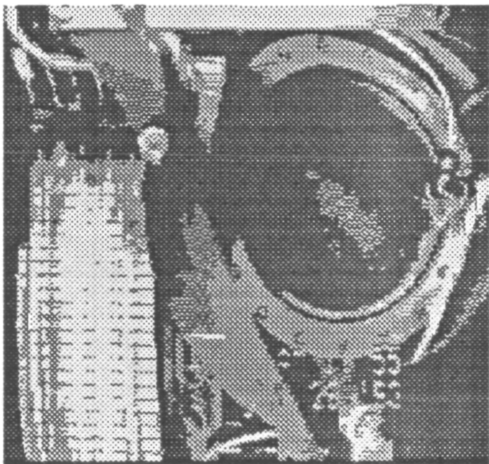
b. (Compression ratio = 5.58)



c. (Compression ratio = 7.26)

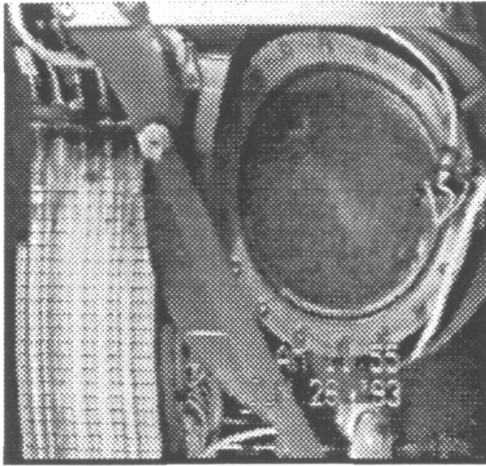


d. (Compression ratio = 13.76)

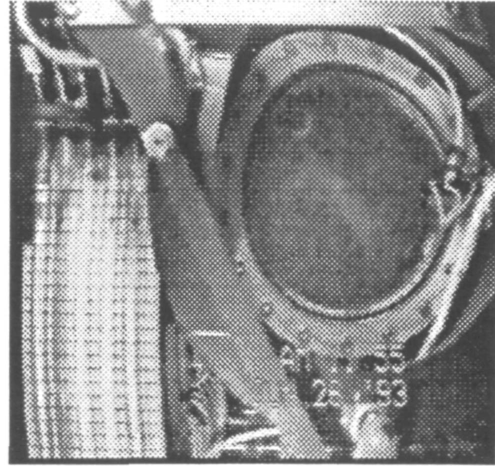


e. (Compression ratio = 22.10)

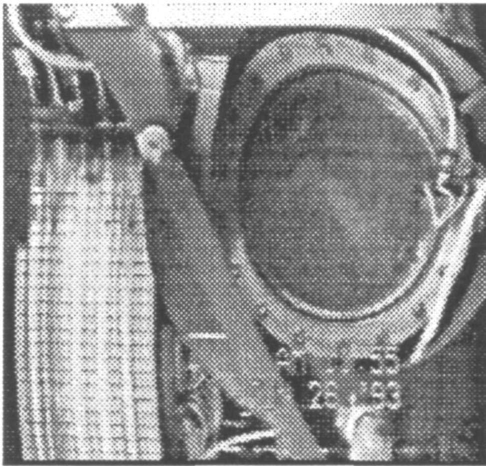
Fig. 5.10 Original & reconstructed frames for a reference picture from NASA2 with Indeo Video.



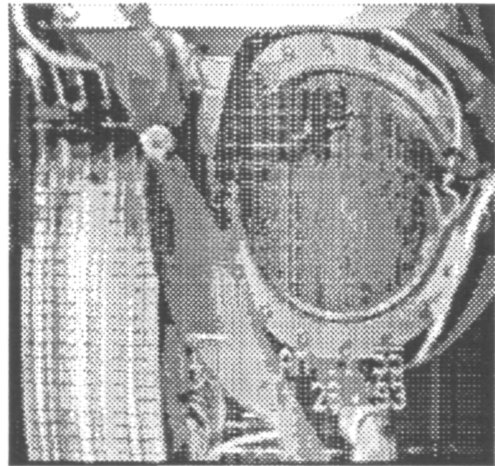
a. (Compression ratio = 0, original)



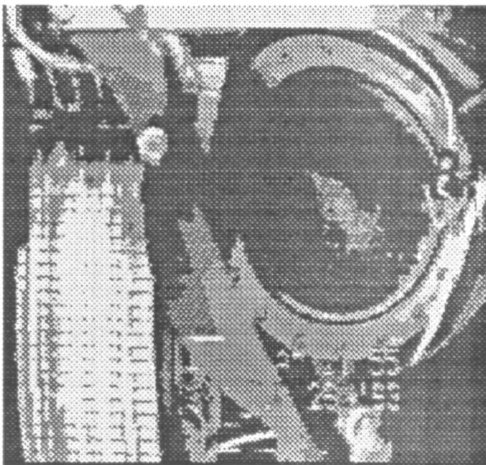
b. (Compression ratio = 5.58)



c. (Compression ratio = 7.26)

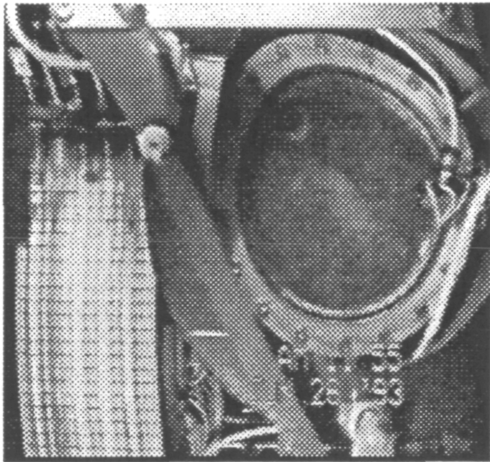


d. (Compression ratio = 13.76)

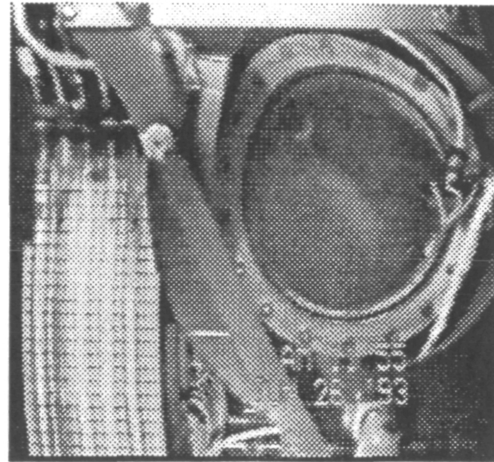


e. (Compression ratio = 22.10)

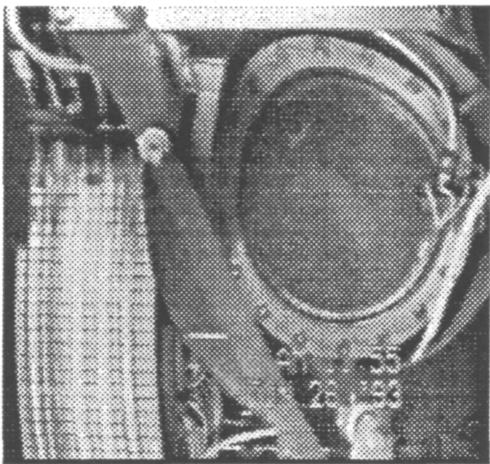
Fig. 5.11 Original & reconstructed frames for a relative picture from NASA2 with Indeo Video.



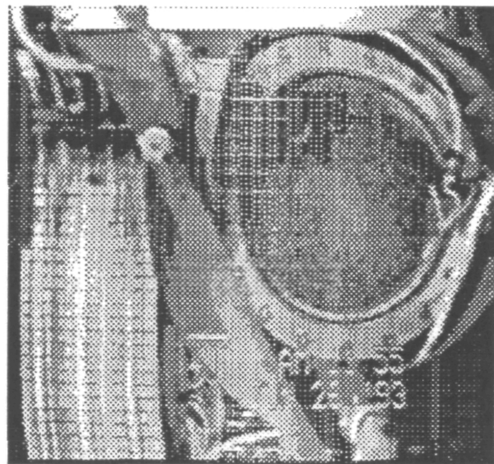
a. (Compression ratio = 0, original)



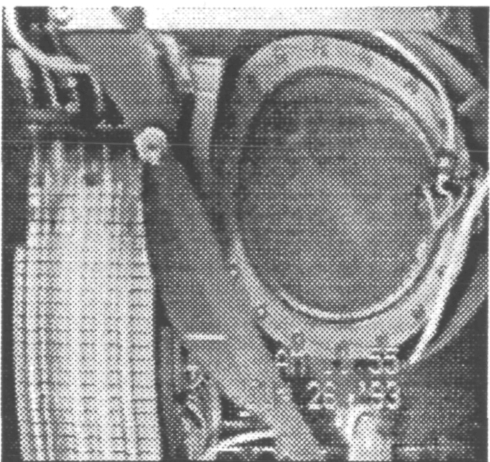
b. (Compression ratio = 5.58)



c. (Compression ratio = 7.26)



d. (Compression ratio = 13.76)



e. (Compression ratio = 22.10)

Fig. 5.12 Original & reconstructed frames for a relative picture from NASA2 with Indeo Video.

6. Summary and Discussions

In this report, we evaluated the different modules of the MPEG2 encoder algorithm with two clips of video each containing 20 frames of pictures captured from the two given types, **EFE Flight Experiment for air bubbles in a tank and glass tubes** (NASA1) and the **Various Drop Physics Module test** (NASA2), at the resolution of 256 by 240. The measured times reconfirmed that the two crucial modules are the ME module and the TR module that take more than 90 % of the total computation time of the MPEG2 encoding process.

Approaches to implementing video standards were analyzed in terms of complexity, applications, and cost. It was argued that, for applications intended by MPEG multiprocessor implementation might be a good choice for encoder because it is highly unbalanced both in terms of computation and in terms of application as well. This approach has the advantages of programmability, good scalability for job assignment by data partitioning, and less design complexity.

Experiments were carried out with the parallel implementation of the two crucial modules of the encoding process. We adopted a strategy of data partitioning for job assignment. Instead of partitioning algorithms that exhibits limited parallelism, we divided the picture frames uniformly on the basis of MBs which are a basic coding unit of the major image/video compression standards such as JPEG [16], [19], H.261 [21], as well as MPEG1[1] and MPEG2 [2]. This type of job partitioning also made our parallel programming easier. The most benefit of this type of data partitioning is that good scalability can be achieved as seen from the previous section.

Compared with Indeo video technology [13], [20], MPEG can achieve higher compression ratio at the same reconstructed video quality as the Indeo video. In addition, its MB oriented coding facilitates data partitioning that leads to good scalability. Finally, MPEG is an international standard that is well expected to be widely accepted.

7. References

[1] "ISO/IEC 11172-2: Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbits/s - Part 2: Video", *International Standards Organization*, August 93.

[2] "ISO/IEC DIS 13818-2: Generic coding of moving pictures and associated audio information - Part 2: Video", *International Standards Organization*, May 94.

[3] Thierry Fautier, "VLSI Implementation of MPEG Decoders", in *Proc. IEEE ISCAS 94 Tutorials*, May 1994.

[4] Bryan Ackland, "VLSI Architectures for Multimedia and Video Conferencing", in *Proc. IEEE ISCAS 94 Tutorials*, May 1994.

[5] Dave Bursky, "Parallelism Pushes DSP Throughput", *Electronic Design Vol.42, No.* , Mach 21, 1994, pp-151-154.

[6] Peter Pirsch, "VLSI Architectures for Video Compression --- A Survey", *Proc. IEEE, Vol. 83, No. 2*, Feb. 1995, pp 220-246.

[7] Frans Sijstermans and Jan Van Der Meer, "CD-I Full-Motion Video Encoding on a Parallel Computer", *Comm. ACM, Vol. 34, NO. 4*, April 1991, pp 82-91.

[8] Jack Shandle, "Parallel Processing Comes Down to Earth", *Electronic Design Vol. 41, No. 17*, Aug. 19, 1993, pp 56-62.

[9] Dan Strassberg, "To Multiprocess or not To Mulitprocess ?", *EDN*, June 23, 1994, pp 64-72.

[10] Jeffrey Child, "Realtime Video Compression Poses Challenges to designers and Vendors Alike", *Computer Design, Vol. 32, No. 7*, July 1993, pp 67-82.

[11] Doug Bailey, "The Realtime Virtues of Programmability", *Computer Design, Vol. 32, No. 7*, July 1993, pp 71-76.

[12] Kai Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, Inc., 1993.

[13] Quaterly report for this contract, 06-16-94 to 09-15-94.

[14] Peter A. Ruetz, Po Tong, Douglas Bailey, Daniel A. Luthi, and Peng H. Ang, "A High-Performance Full-Motion Video Compression Chip Set", *IEEE Trans. Circuits Syst. Video Technol.*, Vol. 2, No. 2, June !992.

[15] Rangarahan Aravind, Glenn L. Cash, Dnald L. Duttweiler, Hsueh-Ming Hang, Barry G. Haskell, and Atul Puri, "Image and video Coding Standards", *AT&T Technical Journal*, January/February 1993, pp 67-88.

[16] Gregory K. Wallace, "The JPEG Still Picture Compression Standard", *Communications of The ACM*, Vol. 34, No. 4, April 1991, pp 31-45.

[17] Didier Le Gall, "MPEG: A Video Compression Standard for Multimedia Applications", *Communications of The ACM*, Vol. 34, No. 4, April 1991, pp 47-58.

[18] Ming Liou, "Overview of the P*64 kbits/s Video Coding Standard", *Communications of The ACM*, Vol. 34, No. 4, April 1991, pp 60-63.

[19] William B. Pennebaker, and Joan L. Mitchell, *JPEG Still Image Data Compression Standard*, Van Nostrand Reinhold, 1993.

[20] Mark J. Bunzel, and Sandra K. Morris, *Multimedia Applications Development Using Indeo ™ Video and DVI Technology*, McGraw-Hill, Inc., 1994.

[21] S. Okubo, "Reference Model Methodology --- A Tool for the Collaborative Creation of Video Coding Standards", ", *Proc. IEEE, Vol. 83, No. 2*, Feb. 1995, pp 139-150+.

Appendix C Code Programs That Were Used In the Experiments

This appendix includes the programs that we used for our experiments. The parallel implementation was based on uniprocessor programs that were adapted from those developed by the MPEG Software Simulation Group. **Part one** is for module ME and **part two** for module TR. Each part includes 3 sets of programs. The first one runs on personal computers with 32 bit C code compiler. It is for providing data for the second set of programs that is a parallel implementation of module ME and runs on INMOS T805 based parallel computers with 3L Ltd.'s parallel C compiler version 2.2.2. Like the first set of programs, the last set runs on personal computers with 32 bit C compiler. It reads data produced by the second set of programs and produces compressed data that is MPEG2 compatible and can be decoded by any MPEG2 decoding program. The purpose of the last set of programs is thus for verifying the correctness of the parallel program that implements the ME module.

The part one has following three sets of programs:

1. The first set of programs for providing data. It includes programs: **encoder.cpp**, **putseq1.cpp**, **motion.cpp**, **transfrm.cpp**, and **putpic.cpp**.
2. The second set of programs for parallel implementation of module ME with 7 processors. It includes programs: **m0.c**, **m01.c**, **m02.c**, **m011.c**, **m012.c**, **m021.c**, and **m022.c**
3. The third set of programs for verifying the parallel programs implementing module ME. It includes programs: **encoder.cpp**, **putseq2.cpp**, **transfrm.cpp**, and **putpic.cpp**

The part two has following three sets of programs:

1. The first set of programs for providing data. It includes programs: **encoder.cpp**, **putseq4.cpp**, **motion.cpp**, **transfrm.cpp**, and **putpic.cpp**.
2. The second set of programs for parallel implementation of module TR with 7 processors. It includes programs: **t0.c**, **t01.c**, **t02.c**, **t011.c**, **t012.c**, **t021.c**, and **t022.c**.

3 The third set of programs for verifying the parallel programs implementing module TR. It includes programs: **encoder.cpp**, **putseq5.cpp**, **motion.cpp**, and **putpic.cpp.**

```
MPEG-2 Test Sequence, 30 frames/sec /* parameter file for running encoding programs */
ns2_%02d   /* name of source files */
q%d        /* name of reconstructed images ("-": don't store) */
-          /* name of intra quant matrix file     ("-": default matrix) */
-          /* name of non intra quant matrix file ("-". default matrix) */
stat.out  /* name of statistics file ("-": stdout ) */
0          /* input picture file format: 0=*.Y,*.U,*.V, 1=*.yuv, 2=*.ppm */
20         /* number of frames */
0          /* number of first frame */
00:00:00:00 /* timecode of first frame */
15         /* N (# of frames in GOP) */
3          /* M (I/P frame distance) */
0          /* ISO/IEC 11172-2 stream */
0          /* 0:frame pictures, 1:field pictures */
256        /* horizontal_size */
240        /* vertical_size */
2          /* aspect_ratio_information 1=square pel, 2=4:3, 3=16:9, 4=2.11:1 */
5          /* frame_rate_code 1=23.976, 2=24, 3=25, 4=29.97, 5=30 frames/sec. */
4000000.0 /* bit_rate (bits/s) */
112        /* vbv_buffer_size (in multiples of 16 kbit) */
0          /* low_delay  */
0          /* constrained_parameters_flag */
4          /* Profile ID: Simple = 5, Main = 4, SNR = 3, Spatial = 2, High = 1 */
8          /* Level ID:   Low = 10, Main = 8, High 1440 = 6, High = 4         */
0          /* progressive_sequence */
1          /* chroma_format: 1=4:2:0, 2=4:2:2, 3=4:4:4 */
2          /* video_format: 0=comp., 1=PAL, 2=NTSC, 3=SECAM, 4=MAC, 5=unspec. */
5          /* color_primaries */
5          /* transfer_characteristics */
4          /* matrix_coefficients */
256        /* display_horizontal_size */
240        /* display_vertical_size */
0          /* intra_dc_precision (0: 8 bit, 1: 9 bit, 2: 10 bit, 3: 11 bit */
1          /* top_field_first */
0 0 0      /* frame_pred_frame_dct (I P B) */
0 0 0      /* concealment_motion_vectors (I P B) */
1 1 1      /* q_scale_type  (I P B) */
1 0 0      /* intra_vlc_format (I P B)*/
0 0 0      /* alternate_scan (I P B) */
0          /* repeat_first_field */
0          /* progressive_frame */
0          /* P distance between complete intra slice refresh */
0          /* rate control: r (reaction parameter) */
0          /* rate control: avg_act (initial average activity) */
0          /* rate control: Xi (initial I frame global complexity measure) */
0          /* rate control: Xp (initial P frame global complexity measure) */
0          /* rate control: Xb (initial B frame global complexity measure) */
0          /* rate control: d0i (initial I frame virtual buffer fullness) */
0          /* rate control: d0p (initial P frame virtual buffer fullness) */
0          /* rate control: d0b (initial B frame virtual buffer fullness) */
2 2 11 11 /* P:  forw_hor_f_code forw_vert_f_code search_width/height */
1 1 3  3  /* B1: forw_hor_f_code forw_vert_f_code search_width/height */
1 1 7  7  /* B1: back_hor_f_code back_vert_f_code search_width/height */
1 1 7  7  /* B2: forw_hor_f_code forw_vert_f_code search_width/height */
```

```
1 1 3  3  /* B2: back_hor_f_code back_vert_f_code search_width/height */
```

```
/* config.h, configuration defines                    */

/* define NON_ANSI_COMPILER for compilers without function prototyping */
/* #define NON_ANSI_COMPILER */

#ifdef NON_ANSI_COMPILER
#define _ANSI_ARGS_(x) ()
#else
#define _ANSI_ARGS_(x) x
#endif
```

```
/* global h, global variables, function prototypes                    */

/* choose between declaration (GLOBAL undefined)
 * and definition (GLOBAL defined)
 * GLOBAL is defined in exactly one file (mpeg2enc.c)
 */

#include "c:\nick\mpeg\org\mpeg2enc.h"

#ifndef GLOBAL
#define EXTERN extern
#else
#define EXTERN
#endif

/* prototypes of global functions */

void range_checks _ANSI_ARGS_((void));
void profile_and_level_checks _ANSI_ARGS_(());

void init_fdct _ANSI_ARGS_((void));
void fdct _ANSI_ARGS_((short *block));

void idct _ANSI_ARGS_((short *block));
void init_idct _ANSI_ARGS_((void));

void motion_estimation _ANSI_ARGS_((unsigned char *oldorg,
                                     unsigned char *neworg,
                                     unsigned char *oldref,
                                     unsigned char *newref,
                                     unsigned char *cur,
                                     int sxf,
                                     int syf,
                                     int sxb,
                                     int syb,
                                     struct mbinfo *mbi));

void error _ANSI_ARGS_((char *text));

void predict _ANSI_ARGS_((unsigned char *reff[],
                          unsigned char *refb[],
                          unsigned char *cur[3],
                          int secondfield, struct mbinfo *mbi));

void initbits _ANSI_ARGS_((void));
void putbits _ANSI_ARGS_(( int val, int n));
void alignbits _ANSI_ARGS_((void));
int  bitcount _ANSI_ARGS_((void)),

void putseqhdr _ANSI_ARGS_((void));
void putseqext _ANSI_ARGS_((void));
void putseqdispext _ANSI_ARGS_((void));
void putuserdata _ANSI_ARGS_((char *userdata));
void putgophdr _ANSI_ARGS_((int frame, int closed_gop));
```

```
void putpicthdr _ANSI_ARGS_((void));
void putpictcodext _ANSI_ARGS_((void));
void putseqend _ANSI_ARGS_((void));

void putintrablk _ANSI_ARGS_((short *blk, int cc));
void putnonintrablk _ANSI_ARGS_((short *blk));
void putmv _ANSI_ARGS_((int dmv, int f_code));

void putpict _ANSI_ARGS_((unsigned char *frame));

void putseq _ANSI_ARGS_((void));

void putDClum _ANSI_ARGS_((int val));
void putDCchrom _ANSI_ARGS_((int val));
void putACfirst _ANSI_ARGS_((int run, int val));
void putAC _ANSI_ARGS_((int run, int signed_level, int vlcformat));
void putaddrinc _ANSI_ARGS_((int addrinc));
void putmbtype _ANSI_ARGS_((int pict_type, int mb_type));
void putmotioncode _ANSI_ARGS_((int motion_code));
void putdmv _ANSI_ARGS_((int dmv));
void putcbp _ANSI_ARGS_(( int cbp));

int quant_intra _ANSI_ARGS_((short *src, short *dst, int dc_prec,
  unsigned char *quant_mat, int mquant));
int quant_non_intra _ANSI_ARGS_((short *src, short *dst,
  unsigned char *quant_mat, int mquant));
void iquant_intra _ANSI_ARGS_((short *src, short *dst, int dc_prec,
  unsigned char *quant_mat, int mquant));
void iquant_non_intra _ANSI_ARGS_((short *src, short *dst,
  unsigned char *quant_mat, int mquant));

void rc_init_seq _ANSI_ARGS_((void));
void rc_init_GOP _ANSI_ARGS_((int np, int nb));
void rc_init_pict _ANSI_ARGS_((unsigned char *frame));
void rc_update_pict _ANSI_ARGS_((void));
int rc_start_mb _ANSI_ARGS_((void));
int rc_calc_mquant _ANSI_ARGS_((int j));
void vbv_end_of_picture _ANSI_ARGS_((void));
void calc_vbv_delay _ANSI_ARGS_((void));

void readframe _ANSI_ARGS_((char *fname, unsigned char *frame[]));

void transform _ANSI_ARGS_((unsigned char *pred[],
                    unsigned char *cur[],
                    struct mbinfo *mbi,
                    short blocks[][64],
                    char forward_invers));
void itransform _ANSI_ARGS_((unsigned char *pred[],
                    unsigned char *cur[],
                    struct mbinfo *mbi,
                    short blocks[][64]));
void dct_type_estimation _ANSI_ARGS_((unsigned char *pred,
                         unsigned char *cur,
                         struct mbinfo *mbi));
```

```
/* zig-zag scan */
EXTERN unsigned char zig_zag_scan[64]
#ifdef GLOBAL
=
{
 0,1,8,16,9,2,3,10,17,24,32,25,18,11,4,5,
 12,19,26,33,40,48,41,34,27,20,13,6,7,14,21,28,
 35,42,49,56,57,50,43,36,29,22,15,23,30,37,44,51,
 58,59,52,45,38,31,39,46,53,60,61,54,47,55,62,63
}
#endif
;

/* alternate scan */
EXTERN unsigned char alternate_scan[64]
#ifdef GLOBAL
=
{
 0,8,16,24,1,9,2,10,17,25,32,40,48,56,57,49,
 41,33,26,18,3,11,4,12,19,27,34,42,50,58,35,43,
 51,59,20,28,5,13,6,14,21,29,36,44,52,60,37,45,
 53,61,22,30,7,15,23,31,38,46,54,62,39,47,55,63
}
#endif
;

/* default intra quantization matrix */
EXTERN unsigned char default_intra_quantizer_matrix[64]
#ifdef GLOBAL
=
{
  8, 16, 19, 22, 26, 27, 29, 34,
 16, 16, 22, 24, 27, 29, 34, 37,
 19, 22, 26, 27, 29, 34, 34, 38,
 22, 22, 26, 27, 29, 34, 37, 40,
 22, 26, 27, 29, 32, 35, 40, 48,
 26, 27, 29, 32, 35, 40, 48, 58,
 26, 27, 29, 34, 38, 46, 56, 69,
 27, 29, 35, 38, 46, 56, 69, 83
}
#endif
;

/* non-linear quantization coefficient table */
EXTERN unsigned char non_linear_mquant_table[32]
#ifdef GLOBAL
=
{
  0, 1, 2, 3, 4, 5, 6, 7,
  8,10,12,14,16,18,20,22,
 24,28,32,36,40,44,48,52,
 56,64,72,80,88,96,104,112
}
```

```
#endif
;

/* non-linear mquant table for mapping from scale to code
 * since reconstruction levels are not bijective with the index map,
 * it is up to the designer to determine most of the quantization levels
 */

EXTERN unsigned char map_non_linear_mquant[113]
#ifdef GLOBAL
=
{
0,1,2,3,4,5,6,7,8,8,9,9,10,10,11,11,12,12,13,13,14,14,15,15,16,16,
16,17,17,17,18,18,18,18,19,19,19,19,20,20,20,20,21,21,21,21,22,22,
22,22,23,23,23,23,24,24,24,24,24,24,24,25,25,25,25,25,25,25,26,26,
26,26,26,26,26,26,27,27,27,27,27,27,27,27,28,28,28,28,28,28,28,29,
29,29,29,29,29,29,29,29,29,30,30,30,30,30,30,30,31,31,31,31,31
}
#endif
;

/* picture data arrays */

/* reconstructed frames */
EXTERN unsigned char *newrefframe[3], *oldrefframe[3], *auxframe[3];
/* original frames */
EXTERN unsigned char *neworgframe[3], *oldorgframe[3], *auxorgframe[3];
/* prediction of current frame */
EXTERN unsigned char *predframe[3];
/* 8*8 block data */
EXTERN short (*blocks)[64];
/* intra / non_intra quantization matrices */
EXTERN unsigned char intra_q[64], inter_q[64];
EXTERN unsigned char chrom_intra_q[64],chrom_inter_q[64];
/* prediction values for DCT coefficient (0,0) */
EXTERN int dc_dct_pred[3];
/* macroblock side information array */
EXTERN struct mbinfo *mbinfo;
/* motion estimation parameters */
EXTERN struct motion_data *motion_data;
/* clipping (=saturation) table */
EXTERN unsigned char *clp;

/* name strings */
EXTERN char id_string[256], tplorg[256], tplref[256];
EXTERN char iqname[256], niqname[256];
EXTERN char statname[256];
EXTERN char errortext[256];

EXTERN FILE *outfile, *statfile; /* file descriptors */
EXTERN int inputtype; /* format of input frames */

EXTERN int quiet; /* suppress warnings */
```

```
/* coding model parameters */

EXTERN int N, /* number of frames in Group of Pictures */
EXTERN int M; /* distance between I/P frames */
EXTERN int P; /* intra slice refresh interval */
EXTERN int nframes; /* total number of frames to encode */
EXTERN int frame0, tc0; /* number and timecode of first frame */
EXTERN int mpeg1; /* ISO/IEC IS 11172-2 sequence */
EXTERN int fieldpic; /* use field pictures */

/* sequence specific data (sequence header) */

EXTERN int horizontal_size, vertical_size; /* frame size (pels) */
EXTERN int width, height; /* encoded frame size (pels) multiples of 16 or 32 */
EXTERN int chrom_width,chrom_height,block_count;
EXTERN int mb_width, mb_height, /* frame size (macroblocks) */
EXTERN int width2, height2, mb_height2, chrom_width2; /* picture size */
EXTERN int aspectratio; /* aspect ratio information (pel or display) */
EXTERN int frame_rate_code; /* coded value of frame rate */
EXTERN double frame_rate; /* frames per second */
EXTERN double bit_rate; /* bits per second */
EXTERN unsigned int vbv_buffer_size; /* size of VBV buffer (* 16 kbit) */
EXTERN int constrparms; /* constrained parameters flag (MPEG-1 only) */
EXTERN int load_iquant, load_niquant; /* use non-default quant. matrices */
EXTERN int load_ciquant,load_cniquant;


/* sequence specific data (sequence extension) */

EXTERN int profile, level; /* syntax / parameter constraints */
EXTERN int prog_seq; /* progressive sequence */
EXTERN int chroma_format;
EXTERN int low_delay; /* no B pictures, skipped pictures */


/* sequence specific data (sequence display extension) */

EXTERN int video_format; /* component, PAL, NTSC, SECAM or MAC */
EXTERN int color_primaries; /* source primary chromaticity coordinates */
EXTERN int transfer_characteristics; /* opto-electronic transfer char. (gamma) */
EXTERN int matrix_coefficients; /* Eg,Eb,Er / Y,Cb,Cr matrix coefficients */
EXTERN int display_horizontal_size, display_vertical_size; /* display size */


/* picture specific data (picture header) */

EXTERN int temp_ref; /* temporal reference */
EXTERN int pict_type; /* picture coding type (I, P or B) */
EXTERN int vbv_delay; /* video buffering verifier delay (1/90000 seconds) */


/* picture specific data (picture coding extension) */
```

```
EXTERN int forw_hor_f_code, forw_vert_f_code;
EXTERN int back_hor_f_code, back_vert_f_code; /* motion vector ranges */
EXTERN int dc_prec; /* DC coefficient precision for intra coded blocks */
EXTERN int pict_struct; /* picture structure (frame, top / bottom field) */
EXTERN int topfirst; /* display top field first */
/* use only frame prediction and frame DCT (I,P,B,current) */
EXTERN int frame_pred_dct_tab[3], frame_pred_dct;
EXTERN int conceal_tab[3]; /* use concealment motion vectors (I,P,B) */
EXTERN int qscale_tab[3], q_scale_type; /* linear/non-linear quantizaton table */
EXTERN int intravlc_tab[3], intravlc; /* intra vlc format (I,P,B,current) */
EXTERN int altscan_tab[3], altscan; /* alternate scan (I,P,B,current) */
EXTERN int repeatfirst; /* repeat first field after second field */
EXTERN int prog_frame; /* progressive frame */
```

```
/* mpg2enc.h, defines and types                          */

#include <time.h>

#define PICTURE_START_CODE 0x100L
#define SLICE_MIN_START    0x101L
#define SLICE_MAX_START    0x1AFL
#define USER_START_CODE    0x1B2L
#define SEQ_START_CODE     0x1B3L
#define EXT_START_CODE     0x1B5L
#define SEQ_END_CODE       0x1B7L
#define GOP_START_CODE     0x1B8L
#define ISO_END_CODE       0x1B9L
#define PACK_START_CODE    0x1BAL
#define SYSTEM_START_CODE 0x1BBL

/* picture coding type */
#define I_TYPE 1
#define P_TYPE 2
#define B_TYPE 3
#define D_TYPE 4

/* picture structure */
#define TOP_FIELD     1
#define BOTTOM_FIELD  2
#define FRAME_PICTURE 3

/* macroblock type */
#define MB_INTRA     1
#define MB_PATTERN   2
#define MB_BACKWARD 4
#define MB_FORWARD  8
#define MB_QUANT    16

/* motion_type */
#define MC_FIELD 1
#define MC_FRAME 2
#define MC_16X8  2
#define MC_DMV   3

/* mv_format */
#define MV_FIELD 0
#define MV_FRAME 1

/* chroma_format */
#define CHROMA420 1
#define CHROMA422 2
#define CHROMA444 3

/* extension start code IDs */

#define SEQ_ID      1
#define DISP_ID     2
#define QUANT_ID    3
```

```
#define SEQSCAL_ID  5
#define PANSCAN_ID  7
#define CODING_ID   8
#define SPATSCAL_ID 9
#define TEMPSCAL_ID 10

/* inputtype */
#define T_Y_U_V 0
#define T_YUV   1
#define T_PPM   2

/* macroblock information */
struct mbinfo {
  int mb_type; /* intra/forward/backward/interpolated */
  int motion_type; /* frame/field/16x8/dual_prime */
  int dct_type; /* field/frame DCT */
  int mquant; /* quantization parameter */
  unsigned int cbp; /* coded block pattern */
  int skipped; /* skipped macroblock */
  int MV[2][2][2]; /* motion vectors */
  int mv_field_sel[2][2]; /* motion vertical field select */
  int dmvector[2]; /* dual prime vectors */
  double act; /* activity measure */
  float var; /* for debugging */
};

/* motion data */
struct motion_data {
  int forw_hor_f_code,forw_vert_f_code; /* vector range */
  int sxf,syf; /* search range */
  int back_hor_f_code,back_vert_f_code;
  int sxb,syb;
};
```

```
/* vlc.h, variable length code tables (used by routines in putvlc.c)        */

/* type definitions for variable length code table entries */

typedef struct
{
  unsigned char code; /* right justified */
  char len;
} VLCtable;

/* for codes longer than 8 bits (excluding leading zeroes) */
typedef struct
{
  unsigned short code; /* right justified */
  char len;
} sVLCtable;


/* data from ISO/IEC 13818-2 DIS, Annex B, variable length code tables */

/* Table B-1, variable length codes for macroblock_address_increment
 *
 * indexed by [macroblock_address_increment-1]
 * 'macroblock_escape' is treated elsewhere
 */

static VLCtable addrinctab[33]=
{
  {0x01,1},  {0x03,3},  {0x02,3},  {0x03,4},
  {0x02,4},  {0x03,5},  {0x02,5},  {0x07,7},
  {0x06,7},  {0x0b,8},  {0x0a,8},  {0x09,8},
  {0x08,8},  {0x07,8},  {0x06,8},  {0x17,10},
  {0x16,10}, {0x15,10}, {0x14,10}, {0x13,10},
  {0x12,10}, {0x23,11}, {0x22,11}, {0x21,11},
  {0x20,11}, {0x1f,11}, {0x1e,11}, {0x1d,11},
  {0x1c,11}, {0x1b,11}, {0x1a,11}, {0x19,11},
  {0x18,11}
};


/* Table B-2, B-3, B-4 variable length codes for macroblock_type
 *
 * indexed by [macroblock_type]
 */

static VLCtable mbtypetab[3][32]=
{
 /* I */
 {
  {0,0}, {1,1}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0},
  {0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0},
  {0,0}, {1,2}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0},
  {0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0}
 },
```

```
 /* P */
 {
  {0,0}, {3,5}, {1,2}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0},
  {1,3}, {0,0}, {1,1}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0},
  {0,0}, {1,6}, {1,5}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0},
  {0,0}, {0,0}, {2,5}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0}
 },
 /* B */
 {
  {0,0}, {3,5}, {0,0}, {0,0}, {2,3}, {0,0}, {3,3}, {0,0},
  {2,4}, {0,0}, {3,4}, {0,0}, {2,2}, {0,0}, {3,2}, {0,0},
  {0,0}, {1,6}, {0,0}, {0,0}, {0,0}, {0,0}, {2,6}, {0,0},
  {0,0}, {0,0}, {3,6}, {0,0}, {0,0}, {0,0}, {2,5}, {0,0}
 }
};


/* Table B-5 ... B-8 variable length codes for macroblock_type in
 *  scalable sequences
 *
 * not implemented
 */

/* Table B-9, variable length codes for coded_block_pattern
 *
 * indexed by [coded_block_pattern]
 */

static VLCtable cbptable[64]=
{
  {0x01,9}, {0x0b,5}, {0x09,5}, {0x0d,6},
  {0x0d,4}, {0x17,7}, {0x13,7}, {0x1f,8},
  {0x0c,4}, {0x16,7}, {0x12,7}, {0x1e,8},
  {0x13,5}, {0x1b,8}, {0x17,8}, {0x13,8},
  {0x0b,4}, {0x15,7}, {0x11,7}, {0x1d,8},
  {0x11,5}, {0x19,8}, {0x15,8}, {0x11,8},
  {0x0f,6}, {0x0f,8}, {0x0d,8}, {0x03,9},
  {0x0f,5}, {0x0b,8}, {0x07,8}, {0x07,9},
  {0x0a,4}, {0x14,7}, {0x10,7}, {0x1c,8},
  {0x0e,6}, {0x0e,8}, {0x0c,8}, {0x02,9},
  {0x10,5}, {0x18,8}, {0x14,8}, {0x10,8},
  {0x0e,5}, {0x0a,8}, {0x06,8}, {0x06,9},
  {0x12,5}, {0x1a,8}, {0x16,8}, {0x12,8},
  {0x0d,5}, {0x09,8}, {0x05,8}, {0x05,9},
  {0x0c,5}, {0x08,8}, {0x04,8}, {0x04,9},
  {0x07,3}, {0x0a,5}, {0x08,5}, {0x0c,6}
};


/* Table B-10, variable length codes for motion_code
 *
 * indexed by [abs(motion_code)]
 * sign of motion_code is treated elsewhere
 */
```

```
static VLCtable motionvectab[17]=
{
 {0x01,1}, {0x01,2}, {0x01,3}, {0x01,4},
 {0x03,6}, {0x05,7}, {0x04,7}, {0x03,7},
 {0x0b,9}, {0x0a,9}, {0x09,9}, {0x11,10},
 {0x10,10}, {0x0f,10}, {0x0e,10}, {0x0d,10},
 {0x0c,10}
};


/* Table B-11, variable length codes for dmvector
 *
 * treated elsewhere
 */

/* Table B-12, variable length codes for dct_dc_size_luminance
 *
 * indexed by [dct_dc_size_luminance]
 */

static sVLCtable DClumtab[12]=
{
 {0x0004,3}, {0x0000,2}, {0x0001,2}, {0x0005,3}, {0x0006,3}, {0x000e,4},
 {0x001e,5}, {0x003e,6}, {0x007e,7}, {0x00fe,8}, {0x01fe,9}, {0x01ff,9}
};


/* Table B-13, variable length codes for dct_dc_size_chrominance
 *
 * indexed by [dct_dc_size_chrominance]
 */

static sVLCtable DCchromtab[12]=
{
 {0x0000,2}, {0x0001,2}, {0x0002,2}, {0x0006,3}, {0x000e,4}, {0x001e,5},
 {0x003e,6}, {0x007e,7}, {0x00fe,8}, {0x01fe,9}, {0x03fe,10},{0x03ff,10}
};


/* Table B-14, DCT coefficients table zero
 *
 * indexed by [run][level-1]
 * split into two tables (dct_code_tab1, dct_code_tab2) to reduce size
 * 'first DCT coefficient' condition and 'End of Block' are treated elsewhere
 * codes do not include s (sign bit)
 */

static VLCtable dct_code_tab1[2][40]=
{
 /* run = 0, level = 1...40 */
 {
  {0x03, 2}, {0x04, 4}, {0x05, 5}, {0x06, 7},
  {0x26, 8}, {0x21, 8}, {0x0a,10}, {0x1d,12},
```

```
 {0x18,12}, {0x13,12}, {0x10,12}, {0x1a,13},
 {0x19,13}, {0x18,13}, {0x17,13}, {0x1f,14},
 {0x1e,14}, {0x1d,14}, {0x1c,14}, {0x1b,14},
 {0x1a,14}, {0x19,14}, {0x18,14}, {0x17,14},
 {0x16,14}, {0x15,14}, {0x14,14}, {0x13,14},
 {0x12,14}, {0x11,14}, {0x10,14}, {0x18,15},
 {0x17,15}, {0x16,15}, {0x15,15}, {0x14,15},
 {0x13,15}, {0x12,15}, {0x11,15}, {0x10,15}
 },
/* run = 1, level = 1...18 */
 {
 {0x03, 3}, {0x06, 6}, {0x25, 8}, {0x0c,10},
 {0x1b,12}, {0x16,13}, {0x15,13}, {0x1f,15},
 {0x1e,15}, {0x1d,15}, {0x1c,15}, {0x1b,15},
 {0x1a,15}, {0x19,15}, {0x13,16}, {0x12,16},
 {0x11,16}, {0x10,16}, {0x00, 0}, {0x00, 0},
 {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0},
 {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0},
 {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0},
 {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0},
 {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}
 }
};

static VLCtable dct_code_tab2[30][5]=
{
 /* run = 2...31, level = 1...5 */
 {{0x05, 4}, {0x04, 7}, {0x0b,10}, {0x14,12}, {0x14,13}},
 {{0x07, 5}, {0x24, 8}, {0x1c,12}, {0x13,13}, {0x00, 0}},
 {{0x06, 5}, {0x0f,10}, {0x12,12}, {0x00, 0}, {0x00, 0}},
 {{0x07, 6}, {0x09,10}, {0x12,13}, {0x00, 0}, {0x00, 0}},
 {{0x05, 6}, {0x1e,12}, {0x14,16}, {0x00, 0}, {0x00, 0}},
 {{0x04, 6}, {0x15,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x07, 7}, {0x11,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x05, 7}, {0x11,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x27, 8}, {0x10,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x23, 8}, {0x1a,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x22, 8}, {0x19,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x20, 8}, {0x18,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x0e,10}, {0x17,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x0d,10}, {0x16,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x08,10}, {0x15,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x1f,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x1a,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x19,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x17,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x16,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x1f,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x1e,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x1d,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x1c,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x1b,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x1f,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x1e,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
```

```
 {{0x1d,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x1c,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x1b,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}}
};


/* Table B-15, DCT coefficients table one
 *
 * indexed by [run][level-1]
 * split into two tables (dct_code_tab1a, dct_code_tab2a) to reduce size
 * 'End of Block' is treated elsewhere
 * codes do not include s (sign bit)
 */

static VLCtable dct_code_tab1a[2][40]=
{
 /* run = 0, level = 1...40 */
 {
  {0x02, 2}, {0x06, 3}, {0x07, 4}, {0x1c, 5},
  {0x1d, 5}, {0x05, 6}, {0x04, 6}, {0x7b, 7},
  {0x7c, 7}, {0x23, 8}, {0x22, 8}, {0xfa, 8},
  {0xfb, 8}, {0xfe, 8}, {0xff, 8}, {0x1f,14},
  {0x1e,14}, {0x1d,14}, {0x1c,14}, {0x1b,14},
  {0x1a,14}, {0x19,14}, {0x18,14}, {0x17,14},
  {0x16,14}, {0x15,14}, {0x14,14}, {0x13,14},
  {0x12,14}, {0x11,14}, {0x10,14}, {0x18,15},
  {0x17,15}, {0x16,15}, {0x15,15}, {0x14,15},
  {0x13,15}, {0x12,15}, {0x11,15}, {0x10,15}
 },
 /* run = 1, level = 1...18 */
 {
  {0x02, 3}, {0x06, 5}, {0x79, 7}, {0x27, 8},
  {0x20, 8}, {0x16,13}, {0x15,13}, {0x1f,15},
  {0x1e,15}, {0x1d,15}, {0x1c,15}, {0x1b,15},
  {0x1a,15}, {0x19,15}, {0x13,16}, {0x12,16},
  {0x11,16}, {0x10,16}, {0x00, 0}, {0x00, 0},
  {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0},
  {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0},
  {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0},
  {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0},
  {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}
 }
};

static VLCtable dct_code_tab2a[30][5]=
{
 /* run = 2...31, level = 1...5 */
 {{0x05, 5}, {0x07, 7}, {0xfc, 8}, {0x0c,10}, {0x14,13}},
 {{0x07, 5}, {0x26, 8}, {0x1c,12}, {0x13,13}, {0x00, 0}},
 {{0x06, 6}, {0xfd, 8}, {0x12,12}, {0x00, 0}, {0x00, 0}},
 {{0x07, 6}, {0x04, 9}, {0x12,13}, {0x00, 0}, {0x00, 0}},
 {{0x06, 7}, {0x1e,12}, {0x14,16}, {0x00, 0}, {0x00, 0}},
 {{0x04, 7}, {0x15,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
 {{0x05, 7}, {0x11,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
```

```
  {{0x78, 7}, {0x11,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x7a, 7}, {0x10,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x21, 8}, {0x1a,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x25, 8}, {0x19,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x24, 8}, {0x18,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x05, 9}, {0x17,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x07, 9}, {0x16,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x0d,10}, {0x15,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x1f,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x1a,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x19,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x17,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x16,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x1f,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x1e,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x1d,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x1c,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x1b,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x1f,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x1e,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x1d,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x1c,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
  {{0x1b,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}}
};
```

```
/*      const1.h        */

#define WORKSPACE 2048
#define END   0
```

```
/**********************************/
/*                                */
/*        encoder.cpp             */
/*                                */
/**********************************/

#include <stdio.h>
#include <stdlib.h>

#define GLOBAL /* used by global.h */
#include "c:\nick\mpeg\org\config.h"
#include "c:\nick\mpeg\org\global.h"

/* private prototypes */
static void init _ANSI_ARGS_((void));
static void readparmfile _ANSI_ARGS_((char *fname));
static void readquantmat _ANSI_ARGS_((void));


void main(int argc, char *argv[] )
{
 if (argc!=3)
 {
  printf("Usage: mpeg2encode in.par out.m2v\n");
  exit(0);
 }

 /* read parameter file */
 readparmfile(argv[1]);

 /* read quantization matrices */
 readquantmat();

 /* open output file */
 if (!(outfile=fopen(argv[2],"wb")))
 {
  sprintf(errortext,"Couldn't create output file %s",argv[2]);
  error(errortext);
 }

 init();
 putseq();

 fclose(outfile);

}

static void init()
{
 int i, size;
 static int block_count_tab[3] = {6,8,12};

 initbits();
 init_fdct();
```

```
init_idct();

/* round picture dimensions to nearest multiple of 16 or 32 */
mb_width = (horizontal_size+15)/16;
mb_height = prog_seq ? (vertical_size+15)/16 : 2*((vertical_size+31)/32);
mb_height2 = fieldpic ? mb_height>>1 : mb_height, /* for field pictures */
width = 16*mb_width;
height = 16*mb_height;

chrom_width = (chroma_format==CHROMA444) ? width : width>>1;
chrom_height = (chroma_format!=CHROMA420) ? height : height>>1;

height2 = fieldpic ? height>>1 : height;
width2 = fieldpic ? width<<1 : width;
chrom_width2 = fieldpic ? chrom_width<<1 : chrom_width;

block_count = block_count_tab[chroma_format-1],

/* clip table */
if (!(clp = (unsigned char *)malloc(1024)))
  error("malloc failed\n");
clp+= 384;
for (i=-384; i<640; i++)
  clp[i] = (i<0) ? 0 : ((i>255) ? 255 : i),

for (i=0; i<3; i++)
{
  size = (i==0) ? width*height : chrom_width*chrom_height;

  if (!(newrefframe[i] = (unsigned char *)malloc(size)))
    error("malloc failed\n"),
  if (!(oldrefframe[i] = (unsigned char *)malloc(size)))
    error("malloc failed\n");
  if (!(auxframe[i] = (unsigned char *)malloc(size)))
    error("malloc failed\n");
  if (!(neworgframe[i] = (unsigned char *)malloc(size)))
    error("malloc failed\n");
  if (!(oldorgframe[i] = (unsigned char *)malloc(size)))
    error("malloc failed\n");
  if (!(auxorgframe[i] = (unsigned char *)malloc(size)))
    error("malloc failed\n");
  if (!(predframe[i] = (unsigned char *)malloc(size)))
    error("malloc failed\n");
}

mbinfo = (struct mbinfo *)malloc(mb_width*mb_height2*sizeof(struct mbinfo));

if (!mbinfo) error("malloc failed\n"),

blocks =
  (short (*)[64])malloc(mb_width*mb_height2*block_count*sizeof(short [64]));

if (!blocks) error("malloc failed\n");
```

```
}

void error(char *text)
{
 fprintf(stderr,text);
 putc('\n',stderr);
 exit(1);
}

static void readparmfile(char *fname)
{
 int i;
 int h,m,s,f;
 FILE *fd;
 char line[256];
 static double ratetab[8]=
   {24000.0/1001.0,24.0,25.0,30000.0/1001.0,30.0,50.0,60000.0/1001.0,60.0};
 extern int r,Xi,Xb,Xp,d0i,d0p,d0b; /* rate control */
 extern double avg_act; /* rate control */

 if (!(fd = fopen(fname,"r")))
 {
  sprintf(errortext,"Couldn't open parameter file %s",fname);
  error(errortext);
 }

 fgets(id_string,254,fd);
 fgets(line,254,fd); sscanf(line,"%s",tplorg);
 fgets(line,254,fd); sscanf(line,"%s",tplref);
 fgets(line,254,fd); sscanf(line,"%s",iqname);
 fgets(line,254,fd); sscanf(line,"%s",niqname);
 fgets(line,254,fd); sscanf(line,"%s",statname);
 fgets(line,254,fd); sscanf(line,"%d",&inputtype);
 fgets(line,254,fd); sscanf(line,"%d",&nframes);
 fgets(line,254,fd); sscanf(line,"%d",&frame0);
 fgets(line,254,fd); sscanf(line,"%d:%d:%d:%d",&h,&m,&s,&f);
 fgets(line,254,fd); sscanf(line,"%d",&N);
 fgets(line,254,fd); sscanf(line,"%d",&M);
 fgets(line,254,fd); sscanf(line,"%d",&mpeg1);
 fgets(line,254,fd); sscanf(line,"%d",&fieldpic);
 fgets(line,254,fd); sscanf(line,"%d",&horizontal_size),
 fgets(line,254,fd); sscanf(line,"%d",&vertical_size);
 fgets(line,254,fd); sscanf(line,"%d",&aspectratio);
 fgets(line,254,fd); sscanf(line,"%d",&frame_rate_code);
 fgets(line,254,fd); sscanf(line,"%lf",&bit_rate);
 fgets(line,254,fd); sscanf(line,"%d",&vbv_buffer_size);
 fgets(line,254,fd); sscanf(line,"%d",&low_delay);
 fgets(line,254,fd); sscanf(line,"%d",&constrparms);
 fgets(line,254,fd); sscanf(line,"%d",&profile);
 fgets(line,254,fd); sscanf(line,"%d",&level);
 fgets(line,254,fd); sscanf(line,"%d",&prog_seq);
 fgets(line,254,fd); sscanf(line,"%d",&chroma_format);
 fgets(line,254,fd); sscanf(line,"%d",&video_format);
 fgets(line,254,fd); sscanf(line,"%d",&color_primaries);
```

```
fgets(line,254,fd); sscanf(line,"%d",&transfer_characteristics);
fgets(line,254,fd); sscanf(line,"%d",&matrix_coefficients);
fgets(line,254,fd); sscanf(line,"%d",&display_horizontal_size);
fgets(line,254,fd); sscanf(line,"%d",&display_vertical_size);
fgets(line,254,fd); sscanf(line,"%d",&dc_prec);
fgets(line,254,fd); sscanf(line,"%d",&topfirst),
fgets(line,254,fd); sscanf(line,"%d %d %d",
  frame_pred_dct_tab,frame_pred_dct_tab+1,frame_pred_dct_tab+2),

fgets(line,254,fd); sscanf(line,"%d %d %d",
  conceal_tab,conceal_tab+1,conceal_tab+2);

fgets(line,254,fd); sscanf(line,"%d %d %d",
  qscale_tab,qscale_tab+1,qscale_tab+2);

fgets(line,254,fd); sscanf(line,"%d %d %d",
  intravlc_tab,intravlc_tab+1,intravlc_tab+2);
fgets(line,254,fd); sscanf(line,"%d %d %d",
  altscan_tab,altscan_tab+1,altscan_tab+2);
fgets(line,254,fd); sscanf(line,"%d",&repeatfirst);
fgets(line,254,fd); sscanf(line,"%d",&prog_frame);
/* intra slice interval refresh period */
fgets(line,254,fd); sscanf(line,"%d",&P);
fgets(line,254,fd); sscanf(line,"%d",&r);
fgets(line,254,fd); sscanf(line,"%lf",&avg_act);
fgets(line,254,fd); sscanf(line,"%d",&Xi);
fgets(line,254,fd); sscanf(line,"%d",&Xp);
fgets(line,254,fd); sscanf(line,"%d",&Xb);
fgets(line,254,fd); sscanf(line,"%d",&d0i);
fgets(line,254,fd); sscanf(line,"%d",&d0p);
fgets(line,254,fd); sscanf(line,"%d",&d0b);

if (N<1) error("N must be positive");
if (M<1) error("M must be positive");
if (N%M != 0) error("N must be an integer multiple of M");

motion_data = (struct motion_data *)malloc(M*sizeof(struct motion_data));
if (!motion_data) error("malloc failed\n");

for (i=0; i<M; i++)
{
  fgets(line,254,fd);
  sscanf(line,"%d %d %d %d",
    &motion_data[i].forw_hor_f_code, &motion_data[i].forw_vert_f_code,
    &motion_data[i].sxf, &motion_data[i].syf),

  if (i!=0)
  {
    fgets(line,254,fd);
    sscanf(line,"%d %d %d %d",
      &motion_data[i].back_hor_f_code, &motion_data[i].back_vert_f_code,
      &motion_data[i].sxb, &motion_data[i].syb);
  }
}
```

```
fclose(fd);

/* make flags boolean (x!=0 -> x=1) */
mpeg1 = !!mpeg1;
fieldpic = !!fieldpic;
low_delay = !!low_delay;
constrparms = !!constrparms;
prog_seq = !!prog_seq;
topfirst = !!topfirst;

for (i=0; i<3; i++)
{
 frame_pred_dct_tab[i] = !!frame_pred_dct_tab[i];
 conceal_tab[i] = !!conceal_tab[i];
 qscale_tab[i] = !!qscale_tab[i];
 intravlc_tab[i] = !!intravlc_tab[i];
 altscan_tab[i] = !!altscan_tab[i];
}
repeatfirst = !!repeatfirst;
prog_frame = !!prog_frame;

/* make sure MPEG specific parameters are valid */
range_checks();

frame_rate = ratetab[frame_rate_code-1];

/* timecode -> frame number */
tc0 = h;
tc0 = 60*tc0 + m;
tc0 = 60*tc0 + s;
tc0 = (int)(frame_rate+0.5)*tc0 + f;

if (!mpeg1)
{
 profile_and_level_checks();
}
else
{
 /* MPEG-1 */
 if (constrparms)
 {
  if (horizontal_size>768
     || vertical_size>576
     || ((horizontal_size+15)/16)*((vertical_size+15)/16)>396
     || ((horizontal_size+15)/16)*((vertical_size+15)/16)*frame_rate>396*25.0
     || frame_rate>30.0)
  {
   if (!quiet)
    fprintf(stderr,"Warning: setting constrained_parameters_flag = 0\n");
   constrparms = 0;
  }
 }
```

```
 if (constrparms)
 {
  for (i=0; i<M; i++)
  {
   if (motion_data[i].forw_hor_f_code>4)
   {
    constrparms = 0;
    break;
   }

   if (motion_data[i].forw_vert_f_code>4)
   {
    constrparms = 0;
    break;
   }

   if (i!=0)
   {
    if (motion_data[i].back_hor_f_code>4)
    {
     constrparms = 0;
     break;
    }

    if (motion_data[i].back_vert_f_code>4)
    {
     constrparms = 0;
     break;
    }
   }
  }
 }
}

/* relational checks */

if (!mpeg1 && constrparms) constrparms = 0;

if (prog_seq && !prog_frame) prog_frame = 1;

if (prog_frame && fieldpic) fieldpic = 0;

if (!prog_frame && repeatfirst) repeatfirst = 0;

if (prog_frame)
for (i=0; i<3; i++)
if (!frame_pred_dct_tab[i]) frame_pred_dct_tab[i] = 1,

if (prog_seq && !repeatfirst && topfirst) topfirst = 0;

/* search windows */
for (i=0; i<M; i++)
{
 if (motion_data[i].sxf > (4<<motion_data[i].forw_hor_f_code)-1)
```

```
    motion_data[i].sxf = (4<<motion_data[i].forw_hor_f_code)-1;

   if (motion_data[i].syf > (4<<motion_data[i].forw_vert_f_code)-1)
    motion_data[i].syf = (4<<motion_data[i].forw_vert_f_code)-1;

   if (i!=0)
   {
    if (motion_data[i].sxb > (4<<motion_data[i].back_hor_f_code)-1)
      motion_data[i].sxb = (4<<motion_data[i].back_hor_f_code)-1;

    if (motion_data[i].syb > (4<<motion_data[i].back_vert_f_code)-1)
      motion_data[i].syb = (4<<motion_data[i].back_vert_f_code)-1;
   }
  }

}

static void readquantmat()
{
 int i,v;
 FILE *fd;

 if (iqname[0]=='-')
 {
  /* use default intra matrix */
  load_iquant = 0;
  for (i=0; i<64; i++)
    intra_q[i] = default_intra_quantizer_matrix[i];
 }
 else
 {
  /* read customized intra matrix */
  load_iquant = 1;
  if (!(fd = fopen(iqname,"r")))
  {
   sprintf(errortext,"Couldn't open quant matrix file %s",iqname);
   error(errortext);
  }

  for (i=0; i<64; i++)
  {
   fscanf(fd,"%d",&v);
   if (v<1 || v>255)
   error("invalid value in quant matrix");
   intra_q[i] = v,
  }

  fclose(fd);
 }

 if (niqname[0]=='-')
 {
  /* use default non-intra matrix */
  load_niquant = 0;
```

```
   for (i=0, i<64; i++)
     inter_q[i] = 16;
 }
 else
 {
   /* read customized non-intra matrix */
   load_niquant = 1;
   if (!(fd = fopen(niqname,"r")))
   {
     sprintf(errortext,"Couldn't open quant matrix file %s",niqname);
     error(errortext);
   }

   for (i=0; i<64; i++)
   {
     fscanf(fd,"%d",&v);
     if (v<1 || v>255)
     error("invalid value in quant matrix");
     inter_q[i] = v;
   }

   fclose(fd);
 }
}

/* check for (level independent) parameter limits */
void range_checks()
{
 int i;

 /* range and value checks */

 if (horizontal_size<1 || horizontal_size>16383)
   error("horizontal_size must be between 1 and 16383");
 if (mpeg1 && horizontal_size>4095)
   error("horizontal_size must be less than 4096 (MPEG-1)");
 if ((horizontal_size&4095)==0)
   error("horizontal_size must not be a multiple of 4096");
 if (chroma_format!=CHROMA444 && horizontal_size%2 != 0)
   error("horizontal_size must be a even (4:2:0 / 4:2.2)");

 if (vertical_size<1 || vertical_size>16383)
   error("vertical_size must be between 1 and 16383");
 if (mpeg1 && vertical_size>4095)
   error("vertical size must be less than 4096 (MPEG-1)");
 if ((vertical_size&4095)==0)
   error("vertical_size must not be a multiple of 4096");
 if (chroma_format==CHROMA420 && vertical_size%2 != 0)
   error("vertical_size must be a even (4:2:0)");
 if(fieldpic)
 {
   if (vertical_size%2 != 0)
     error("vertical_size must be a even (field pictures)");
   if (chroma_format==CHROMA420 && vertical_size%4 != 0)
```

```
    error("vertical_size must be a multiple of 4 (4:2:0 field pictures)");
  }

  if (mpeg1)
  {
    if (aspectratio<1 || aspectratio>14)
      error("pel_aspect_ratio must be between 1 and 14 (MPEG-1)");
  }
  else
  {
    if (aspectratio<1 || aspectratio>4)
      error("aspect_ratio_information must be 1, 2, 3 or 4");
  }

  if (frame_rate_code<1 || frame_rate_code>8)
    error("frame_rate code must be between 1 and 8");

  if (bit_rate<=0.0)
    error("bit_rate must be positive");
  if (bit_rate > ((1<<30)-1)*400.0)
    error("bit_rate must be less than 429 Gbit/s");
  if (mpeg1 && bit_rate > ((1<<18)-1)*400.0)
    error("bit_rate must be less than 104 Mbit/s (MPEG-1)");

  if (vbv_buffer_size<1 || vbv_buffer_size>0x3ffff)
    error("vbv_buffer_size must be in range 1..(2^18-1)");
  if (mpeg1 && vbv_buffer_size>=1024)
    error("vbv_buffer_size must be less than 1024 (MPEG-1)");

  if (chroma_format<CHROMA420 || chroma_format>CHROMA444)
    error("chroma_format must be in range 1...3");

  if (video_format<0 || video_format>4)
    error("video_format must be in range 0...4");

  if (color_primaries<1 || color_primaries>7 || color_primaries==3)
    error("color_primaries must be in range 1...2 or 4...7");

  if (transfer_characteristics<1 || transfer_characteristics>7
      || transfer_characteristics==3)
    error("transfer_characteristics must be in range 1...2 or 4...7");

  if (matrix_coefficients<1 || matrix_coefficients>7 || matrix_coefficients==3)
    error("matrix_coefficients must be in range 1...2 or 4...7");

  if (display_horizontal_size<0 || display_horizontal_size>16383)
    error("display_horizontal_size must be in range 0...16383");
  if (display_vertical_size<0 || display_vertical_size>16383)
    error("display_vertical_size must be in range 0...16383");

  if (dc_prec<0 || dc_prec>3)
    error("intra_dc_precision must be in range 0...3");

  for (i=0; i<M; i++)
```

```
 {
  if (motion_data[i].forw_hor_f_code<1 || motion_data[i].forw_hor_f_code>9)
   error("f_code must be between 1 and 9");
  if (motion_data[i].forw_vert_f_code<1 || motion_data[i] forw_vert_f_code>9)
   error("f_code must be between 1 and 9");
  if (mpeg1 && motion_data[i].forw_hor_f_code>7)
   error("f_code must be le less than 8");
  if (mpeg1 && motion_data[i].forw_vert_f_code>7)
   error("f_code must be le less than 8");
  if (motion_data[i].sxf<=0)
   error("search window must be positive"); /* doesn't belong here */
  if (motion_data[i].syf<=0)
   error("search window must be positive");
  if (i!=0)
  {
   if (motion_data[i].back_hor_f_code<1 || motion_data[i].back_hor_f_code>9)
    error("f_code must be between 1 and 9");
   if (motion_data[i].back_vert_f_code<1 || motion_data[i].back_vert_f_code>9)
    error("f_code must be between 1 and 9");
   if (mpeg1 && motion_data[i].back_hor_f_code>7)
    error("f_code must be le less than 8");
   if (mpeg1 && motion_data[i].back_vert_f_code>7)
    error("f_code must be le less than 8");
   if (motion_data[i].sxb<=0)
    error("search window must be positive");
   if (motion_data[i].syb<=0)
    error("search window must be positive");
  }
 }
}

/* identifies valid profile / level combinations */
static char profile_level_defined[5][4] =
{
/* HL  H-14 ML  LL */
 {1,  1,  1,  0}, /* HP   */
 {0,  1,  0,  0}, /* Spat */
 {0,  0,  1,  1}, /* SNR  */
 {1,  1,  1,  1}, /* MP   */
 {0,  0,  1,  0}  /* SP   */
};

static struct level_limits {
 int hor_f_code;
 int vert_f_code;
 int hor_size;
 int vert_size;
 int sample_rate;
 int bit_rate; /* Mbit/s */
 int vbv_buffer_size; /* 16384 bit steps */
} maxval_tab[4] =
{
 {9, 5, 1920, 1152, 62668800, 80, 597}, /* HL */
 {9, 5, 1440, 1152, 47001600, 60, 448}, /* H-14 */
```

```
  {8, 5,  720,  576, 10368000, 15, 112}, /* ML */
  {7, 4,  352,  288,  3041280,  4,  29}  /* LL */
};

#define SP   5
#define MP   4
#define SNR  3
#define SPAT 2
#define HP   1

#define LL  10
#define ML   8
#define H14  6
#define HL   4

void profile_and_level_checks()
{
  int i;
  struct level_limits *maxval;

  if (profile<0 || profile>15)
    error("profile must be between 0 and 15");

  if (level<0 || level>15)
    error("level must be between 0 and 15");

  if (profile>=8)
  {
    if (!quiet)
      fprintf(stderr,"Warning: profile uses a reserved value, conformance checks skipped\n");
    return;
  }

  if (profile<HP || profile>SP)
    error("undefined Profile");

  if (profile==SNR || profile==SPAT)
    error("This encoder currently generates no scalable bitstreams");

  if (level<HL || level>LL || level&1)
    error("undefined Level");

  maxval = &maxval_tab[(level-4) >> 1];

  /* check profile@level combination */
  if(!profile_level_defined[profile-1][(level-4) >> 1])
    error("undefined profile@level combination");


  /* profile (syntax) constraints */

  if (profile==SP && M!=1)
    error("Simple Profile does not allow B pictures");
```

```
if (profile!=HP && chroma_format!=CHROMA420)
  error("chroma format must be 4 2·0 in specified Profile");

if (profile==HP && chroma_format==CHROMA444)
  error("chroma format must be 4.2:0 or 4:2:2 in High Profile"),

if (profile>=MP) /* SP, MP: constrained repeat_first_field */
{
  if (frame_rate_code<=2 && repeatfirst)
    error("repeat_first_first must be zero");
  if (frame_rate_code<=6 && prog_seq && repeatfirst)
    error("repeat_first_first must be zero");
}

if (profile!=HP && dc_prec==3)
  error("11 bit DC precision only allowed in High Profile");


/* level (parameter value) constraints */

/* Table 8-8 */
if (frame_rate_code>5 && level>=ML)
  error("Picture rate greater than permitted in specified Level");

for (i=0; i<M; i++)
{
  if (motion_data[i].forw_hor_f_code > maxval->hor_f_code)
    error("forward horizontal f_code greater than permitted in specified Level");

  if (motion_data[i].forw_vert_f_code > maxval->vert_f_code)
    error("forward vertical f_code greater than permitted in specified Level");

  if (i!=0)
  {
    if (motion_data[i].back_hor_f_code > maxval->hor_f_code)
      error("backward horizontal f_code greater than permitted in specified Level");

    if (motion_data[i].back_vert_f_code > maxval->vert_f_code)
      error("backward vertical f_code greater than permitted in specified Level");
  }
}

/* Table 8-10 */
if (horizontal_size > maxval->hor_size)
  error("Horizontal size is greater than permitted in specified Level");

if (vertical_size > maxval->vert_size)
  error("Horizontal size is greater than permitted in specified Level"),

/* Table 8-11 */
if (horizontal_size*vertical_size*frame_rate > maxval->sample_rate)
  error("Sample rate is greater than permitted in specified Level");

/* Table 8-12 */
```

```
 if (bit_rate> 1.0e6 * maxval->bit_rate)
   error("Bit rate is greater than permitted in specified Level");

 /* Table 8-13 */
 if (vbv_buffer_size > maxval->vbv_buffer_size)
   error("vbv_buffer_size exceeds High Level limit");
}
```

```
/**********************************/
/*                                */
/*        putseq1.cpp             */
/*                                */
/**********************************/

#include <stdio.h>
#include <string.h>
#include "c:\nick\mpeg\org\config.h"
#include "c:\nick\mpeg\org\global.h"

void putseq()
{
 /* this routine assumes (N % M) == 0 */
 int i, j, k, f, f0, n, np, nb;
 int sxf, syf, sxb, syb;
 FILE *f_d, *f_status;
 char name[128], name1[128];
 unsigned char *neworg[3], *newref[3];

 rc_init_seq(); /* initialize rate control */

 /* sequence header, sequence extension and sequence display extension */
 putseqhdr();
 if (!mpeg1)
 {
  putseqext();
  putseqdispext();
 }

/* loop through all frames in encoding/decoding order */

 for (i=0; i<nframes; i++)
 {
  /* f0: lowest frame number in current GOP
   *
   * first GOP contains N-(M-1) frames,
   * all other GOPs contain N frames
   */
  f0 = N*((i+(M-1))/N) - (M-1);

  if (f0<0)
   f0=0;

  if (i==0 || (i-1)%M==0)
  {
   /* I or P frame */
   for (j=0; j<3; j++)
   {
    /* shuffle reference frames */
    neworg[j] = oldorgframe[j];
    newref[j] = oldrefframe[j];
    oldorgframe[j] = neworgframe[j];
    oldrefframe[j] = newrefframe[j];
```

```
    neworgframe[j] = neworg[j];
    newrefframe[j] = newref[j];
   }

   /* f. frame number in display order */
   f = (i==0) ? 0 : i+M-1;
   if (f>=nframes)
     f = nframes - 1;

   if (i==f0) /* first displayed frame in GOP is I */
   {
     /* I frame */
     pict_type = I_TYPE;
     forw_hor_f_code = forw_vert_f_code = 15;
     back_hor_f_code = back_vert_f_code = 15;

     /* n: number of frames in current GOP
      *
      * first GOP contains (M-1) less (B) frames
      */
     n = (i==0) ? N-(M-1) : N;

     /* last GOP may contain less frames */
     if (n > nframes-f0)
       n = nframes-f0;

     /* number of P frames */
     if (i==0)
       np = (n + 2*(M-1))/M - 1; /* first GOP */
     else
       np = (n + (M-1))/M - 1;

     /* number of B frames */
     nb = n - np - 1;

     rc_init_GOP(np,nb);

     putgophdr(f0,i==0); /* set closed_GOP in first GOP only */
   }
   else
   {
     /* P frame */
     pict_type = P_TYPE;
     forw_hor_f_code = motion_data[0].forw_hor_f_code;
     forw_vert_f_code = motion_data[0].forw_vert_f_code;
     back_hor_f_code = back_vert_f_code = 15;
     sxf = motion_data[0].sxf;
     syf = motion_data[0].syf;
   }
 }
 else
 {
   /* B frame */
   for (j=0; j<3; j++)
```

```
    {
     neworg[j] = auxorgframe[j],
     newref[j] = auxframe[j];
    }

    /* f: frame number in display order */
    f = i - 1;
    pict_type = B_TYPE;
    n = (i-2)%M + 1; /* first B: n=1, second B. n=2, ... */
    forw_hor_f_code = motion_data[n].forw_hor_f_code;
    forw_vert_f_code = motion_data[n].forw_vert_f_code;
    back_hor_f_code = motion_data[n].back_hor_f_code;
    back_vert_f_code = motion_data[n] back_vert_f_code;
    sxf = motion_data[n].sxf;
    syf = motion_data[n].syf;
    sxb = motion_data[n].sxb;
    syb = motion_data[n].syb;
   }

   temp_ref = f - f0;
   frame_pred_dct = frame_pred_dct_tab[pict_type-1];
   q_scale_type = qscale_tab[pict_type-1],
   intravlc = intravlc_tab[pict_type-1];
   altscan = altscan_tab[pict_type-1];

/* read in next picture to be coded */

   sprintf(name,tplorg,f+frame0);
   readframe(name,neworg);

/* for frame pictures only */

   pict_struct = FRAME_PICTURE;


/* record status data for parallel program
   that implements module ME
*/

   sprintf(name,tplorg,i);
   sprintf(name1,"c:\\nick\\mpeg\\dmot\\%s.sta",name);
        if (!(f_status = fopen(name1,"w/0")))
   {
    sprintf(errortext,"Couldn't open %s\n",name1);
    error(errortext);
   }

   fprintf(f_status,"%d %d %d %d %d %d %d %d \n",
   pict_type, pict_struct, sxf, syf, sxb, syb, f, frame_pred_dct);

        fclose(f_status),


/* record referenc frames for parallel program
```

```
   that implements module ME
 */

   sprintf(name,tplorg,f+frame0);

   sprintf(name1,"c:\\nick\\mpeg\\dmot\\%s.oof",name),
   if (!(f_d = fopen(name1,"wb")))
   {
    sprintf(errortext,"Couldn't open %s\n",name1);
    error(errortext);
   }
   fwrite(oldorgframe[0],1,width*height,f_d);
   fclose(f_d);

   sprintf(name1,"c:\\nick\\mpeg\\dmot\\%s.nof",name);
   if (!(f_d = fopen(name1,"wb")))
   {
    sprintf(errortext,"Couldn't open %s\n",name1);
    error(errortext);
   }
   fwrite(neworgframe[0],1,width*height,f_d);
   fclose(f_d);

   sprintf(name1,"c:\\nick\\mpeg\\dmot\\%s.off",name);
   if (!(f_d = fopen(name1,"wb")))
   {
    sprintf(errortext,"Couldn't open %s\n",name1);
    error(errortext);
   }
   fwrite(oldrefframe[0],1,width*height,f_d);
   fclose(f_d);

   sprintf(name1,"c:\\nick\\mpeg\\dmot\\%s.nff",name);
   if (!(f_d = fopen(name1,"wb")))
   {
    sprintf(errortext,"Couldn't open %s\n",name1);
    error(errortext);
   }
   fwrite(newrefframe[0],1,width*height,f_d);
   fclose(f_d);

   sprintf(name1,"c:\\nick\\mpeg\\dmot\\%s.nor",name);
   if (!(f_d = fopen(name1,"wb")))
   {
    sprintf(errortext,"Couldn't open %s\n",name1);
    error(errortext);
   }
   fwrite(neworg[0],1,width*height,f_d);
   fclose(f_d);


/* motion estimation ( ME module ) */
    motion_estimation(oldorgframe[0], neworgframe[0],
                 oldrefframe[0], newrefframe[0],
```

```
                    neworg[0],
                    sxf, syf,
                    sxb, syb,
                    mbinfo);

    /* prediction */
      predict(oldrefframe,newrefframe,predframe,0,mbinfo);

    /* transform ( TR module ) */
      transform(predframe,neworg,mbinfo,blocks,1);

    /* quantization & variable length coding */
      putpict(neworg[0]);

    /* inverse quantization */
      for (k=0; k<mb_height*mb_width; k++)
      {
        if (mbinfo[k].mb_type & MB_INTRA)
          for (j=0; j<block_count; j++)
            iquant_intra(blocks[k*block_count+j],blocks[k*block_count+j],
                         dc_prec,intra_q,mbinfo[k].mquant);
        else
          for (j=0;j<block_count;j++)
            iquant_non_intra(blocks[k*block_count+j],blocks[k*block_count+j],
                             inter_q,mbinfo[k].mquant);
      }

    /* inverse transform */
      transform(predframe,newref,mbinfo,blocks,0);

  }

  putseqend();
}
```

```
/**********************************/
/*                                */
/*         putseq2.cpp            */
/*                                */
/**********************************/

#include <stdio.h>
#include <string.h>
#include "c:\nick\mpeg\org\config.h"
#include "c:\nick\mpeg\org\global.h"

void putseq()
{
 /* this routine assumes (N % M) == 0 */
 int i, j, k, f, f0, n, np, nb;
 int a, p, q;
 FILE *f_mbi,
 char name[128], name1[128];
 unsigned char *neworg[3], *newref[3];

 rc_init_seq(); /* initialize rate control */

 /* sequence header, sequence extension and sequence display extension */
 putseqhdr();
 if (!mpeg1)
 {
  putseqext();
  putseqdispext();
 }

/* loop through all frames in encoding/decoding order */
 for (i=0; i<nframes; i++)
 {
  /* f0: lowest frame number in current GOP
   *
   * first GOP contains N-(M-1) frames,
   * all other GOPs contain N frames
   */
  f0 = N*((i+(M-1))/N) - (M-1);

  if (f0<0) f0=0;

  if (i==0 || (i-1)%M==0)
  {
   /* I or P frame */
   for (j=0; j<3; j++)
   {
    /* shuffle reference frames */
    neworg[j] = oldorgframe[j];
    newref[j] = oldrefframe[j];
    oldorgframe[j] = neworgframe[j];
    oldrefframe[j] = newrefframe[j];
    neworgframe[j] = neworg[j];
    newrefframe[j] = newref[j];
```

```
    }

    /* f: frame number in display order */
    f = (i==0) ? 0 : i+M-1;
    if (f>=nframes) f = nframes - 1;

    if (i==f0) /* first displayed frame in GOP is I */
    {
     /* I frame */
     pict_type = I_TYPE;
     forw_hor_f_code = forw_vert_f_code = 15;
     back_hor_f_code = back_vert_f_code = 15;

     /* n: number of frames in current GOP
      *
      * first GOP contains (M-1) less (B) frames
      */
     n = (i==0) ? N-(M-1) : N;

     /* last GOP may contain less frames */
     if (n > nframes-f0)
       n = nframes-f0;

     /* number of P frames */
     if (i==0)
       np = (n + 2*(M-1))/M - 1; /* first GOP */
     else
       np = (n + (M-1))/M - 1;

     /* number of B frames */
     nb = n - np - 1;

     rc_init_GOP(np,nb);

     putgophdr(f0,i==0); /* set closed_GOP in first GOP only */
    }
    else
    {
     /* P frame */
     pict_type = P_TYPE;
     forw_hor_f_code = motion_data[0].forw_hor_f_code;
     forw_vert_f_code = motion_data[0].forw_vert_f_code;
     back_hor_f_code = back_vert_f_code = 15;
    }
  }
  else
  {
   /* B frame */
   for (j=0; j<3; j++)
   {
    neworg[j] = auxorgframe[j];
    newref[j] = auxframe[j];
   }
```

```
    /* f: frame number in display order */
    f = i - 1;
    pict_type = B_TYPE;
    n = (i-2)%M + 1; /* first B: n=1, second B: n=2, ... */
    forw_hor_f_code = motion_data[n].forw_hor_f_code;
    forw_vert_f_code = motion_data[n].forw_vert_f_code;
    back_hor_f_code = motion_data[n].back_hor_f_code;
    back_vert_f_code = motion_data[n].back_vert_f_code;
   }

   temp_ref = f - f0;
   frame_pred_dct = frame_pred_dct_tab[pict_type-1],
   q_scale_type = qscale_tab[pict_type-1],
   intravlc = intravlc_tab[pict_type-1];
   altscan = altscan_tab[pict_type-1];

/*  read in next frame for coding   */
   sprintf(name,tplorg,f+frame0);
   readframe(name,neworg);

/*  for frame pictures only */
   pict_struct = FRAME_PICTURE,

/*  read in data produced by the parallel program
   that implements module ME
*/
    sprintf(name1,"c:\\nick\\mpeg\\mot_out\\%s mbi",name);
    if (!(f_mbi = fopen(name1,"r")))
    {
     sprintf(errortext,"Couldn't open %s\n",name1);
     error(errortext);
    }

    a=0;
    for (p=0; p<mb_height2; p++)
    {
     for (q=0; q<mb_width; q++)
     {
      fscanf(f_mbi,"%d %d %d %d %d %d %d %d %d %d %d %d %d %d \n",
      &mbinfo[a].mb_type,
      &mbinfo[a].motion_type,
      &mbinfo[a].MV[0][0][0],
      &mbinfo[a].MV[0][0][1],
      &mbinfo[a].MV[0][1][0],
      &mbinfo[a].MV[0][1][1],
      &mbinfo[a] MV[1][0][0],
      &mbinfo[a].MV[1][0][1],
      &mbinfo[a].MV[1][1][0],
      &mbinfo[a].MV[1][1][1],
      &mbinfo[a].mv_field_sel[0][0],
      &mbinfo[a].mv_field_sel[0][1],
      &mbinfo[a].mv_field_sel[1][0],
      &mbinfo[a].mv_field_sel[1][1]);
```

```
      a++,
     }
    }

    fclose(f_mbi);

/* prediction */
    predict(oldrefframe,newrefframe,predframe,0,mbinfo);

/* transform (TR module) */
    transform(predframe,neworg,mbinfo,blocks,1);

/* quantization & variable length coding */
    putpict(neworg[0]);

/* inverse quantization */
    for (k=0; k<mb_height*mb_width, k++)
    {
      if (mbinfo[k].mb_type & MB_INTRA)
        for (j=0; j<block_count; j++)
          iquant_intra(blocks[k*block_count+j],blocks[k*block_count+j],
                  dc_prec,intra_q,mbinfo[k].mquant),
      else
        for (j=0;j<block_count;j++)
          iquant_non_intra(blocks[k*block_count+j],blocks[k*block_count+j],
                  inter_q,mbinfo[k].mquant);
    }

/* inverse transform */
    transform(predframe,newref,mbinfo,blocks,0);

  }

  putseqend();
}
```

```
/*********************************/
/*                               */
/*         putseq4.cpp           */
/*                               */
/*********************************/

#include <stdio.h>
#include <string.h>
#include "c:\nick\mpeg\org\config.h"
#include "c:\nick\mpeg\org\global.h"

void putseq()
{
 /* this routine assumes (N % M) == 0 */
 int i, j, k, f, f0, n, np, nb;
 int a, p, q, b, c;
 int sxf, syf, sxb, syb;
 FILE *f_mbi, *f_d, *f_status, *f_blk;
 char name[128], name1[128];
 unsigned char *neworg[3], *newref[3];

 rc_init_seq(); /* initialize rate control */

 /* sequence header, sequence extension and sequence display extension */
 putseqhdr();
 if (!mpeg1)
 {
  putseqext();
  putseqdispext();
 }


/* loop through all frames in encoding/decoding order */
 for (i=0; i<nframes; i++)
 {
  /* f0: lowest frame number in current GOP
   *
   * first GOP contains N-(M-1) frames,
   * all other GOPs contain N frames
   */
  f0 = N*((i+(M-1))/N) - (M-1);

  if (f0<0) f0=0;

  if (i==0 || (i-1)%M==0)
  {
   /* I or P frame */
   for (j=0; j<3; j++)
   {
    /* shuffle reference frames */
    neworg[j] = oldorgframe[j];
    newref[j] = oldrefframe[j];
    oldorgframe[j] = neworgframe[j];
    oldrefframe[j] = newrefframe[j];
```

```
      neworgframe[j] = neworg[j];
      newrefframe[j] = newref[j];
    }

    /* f: frame number in display order */
    f = (i==0) ? 0 : i+M-1;
    if (f>=nframes) f = nframes - 1;

    if (i==f0) /* first displayed frame in GOP is I */
    {
      /* I frame */
      pict_type = I_TYPE;
      forw_hor_f_code = forw_vert_f_code = 15;
      back_hor_f_code = back_vert_f_code = 15;

      /* n: number of frames in current GOP
       *
       * first GOP contains (M-1) less (B) frames
       */
      n = (i==0) ? N-(M-1) : N,

      /* last GOP may contain less frames */
      if (n > nframes-f0)
        n = nframes-f0;

      /* number of P frames */
      if (i==0)
        np = (n + 2*(M-1))/M - 1; /* first GOP */
      else
        np = (n + (M-1))/M - 1,

      /* number of B frames */
      nb = n - np - 1;

      rc_init_GOP(np,nb);

      putgophdr(f0,i==0), /* set closed_GOP in first GOP only */
    }
    else
    {
      /* P frame */
      pict_type = P_TYPE;
      forw_hor_f_code = motion_data[0].forw_hor_f_code;
      forw_vert_f_code = motion_data[0].forw_vert_f_code;
      back_hor_f_code = back_vert_f_code = 15;
      sxf = motion_data[0].sxf;
      syf = motion_data[0].syf;
    }
  }
  else
  {
    /* B frame */
    for (j=0; j<3; j++)
    {
```

```
      neworg[j] = auxorgframe[j];
      newref[j] = auxframe[j];
     }

     /* f: frame number in display order */
     f = i - 1;
     pict_type = B_TYPE;
     n = (i-2)%M + 1; /* first B: n=1, second B: n=2, ... */
     forw_hor_f_code = motion_data[n].forw_hor_f_code;
     forw_vert_f_code = motion_data[n].forw_vert_f_code;
     back_hor_f_code = motion_data[n].back_hor_f_code;
     back_vert_f_code = motion_data[n].back_vert_f_code;
     sxf = motion_data[n].sxf;
     syf = motion_data[n].syf;
     sxb = motion_data[n].sxb;
     syb = motion_data[n].syb;
    }

   temp_ref = f - f0,
   frame_pred_dct = frame_pred_dct_tab[pict_type-1];
   q_scale_type = qscale_tab[pict_type-1];
   intravlc = intravlc_tab[pict_type-1];
   altscan = altscan_tab[pict_type-1];

/*  read in next frame for coding   */
   sprintf(name,tplorg,f+frame0);
   readframe(name,neworg);

/*  for frame pictures only   */
     pict_struct = FRAME_PICTURE;

/* motion estimation (module ME) */
     motion_estimation(oldorgframe[0], neworgframe[0],
                 oldrefframe[0], newrefframe[0],
                 neworg[0],
                 sxf, syf,
                 sxb, syb,
                 mbinfo),

/* predictionn */
   predict(oldrefframe,newrefframe,predframe,0,mbinfo);

/*  write status data for the parallel program
   that implements module TR
*/
   sprintf(name,tplorg,i);
   sprintf(name1,"c:\\nick\\mpeg\\dtrn\\%s.sta",name);
   if (!(f_status = fopen(name1,"w/0")))
   {
    sprintf(errortext,"Couldn't open %s\n",name1);
    error(errortext);
   }

   fprintf(f_status,"%d %d %d \n", f, pict_struct, frame_pred_dct);
```

```
  fclose(f_status);


/*  write image data for the parallel program
    that implements module TR
*/
   sprintf(name,tplorg,f+frame0);

   sprintf(name1,"c:\\nick\\mpeg\\dtrn\\%sy.nor",name),
   if (!(f_d = fopen(name1,"wb")))
   {
    sprintf(errortext,"Couldn't open %s\n",name1);
    error(errortext);
   }
   fwrite(neworg[0],1,width*height,f_d);
   fclose(f_d);

   sprintf(name1,"c:\\nick\\mpeg\\dtrn\\%su.nor",name);
   if (!(f_d = fopen(name1,"wb")))
   {
    sprintf(errortext,"Couldn't open %s\n",name1);
    error(errortext);
   }
   fwrite(neworg[1],1,chrom_width*chrom_height,f_d);
   fclose(f_d);

   sprintf(name1,"c:\\nick\\mpeg\\dtrn\\%sv.nor",name);
   if (!(f_d = fopen(name1,"wb")))
   {
    sprintf(errortext,"Couldn't open %s\n",name1);
    error(errortext);
   }
   fwrite(neworg[2],1,chrom_width*chrom_height,f_d);
   fclose(f_d);

   sprintf(name1,"c:\\nick\\mpeg\\dtrn\\%sy.prf",name);
   if (!(f_d = fopen(name1,"wb")))
   {
    sprintf(errortext,"Couldn't open %s\n",name1);
    error(errortext),
   }
   fwrite(predframe[0],1,width*height,f_d),
   fclose(f_d);

   sprintf(name1,"c:\\nick\\mpeg\\dtrn\\%su.prf",name);
   if (!(f_d = fopen(name1,"wb")))
   {
    sprintf(errortext,"Couldn't open %s\n",name1);
    error(errortext),
   }
   fwrite(predframe[1],1,chrom_width*chrom_height,f_d),
   fclose(f_d);

   sprintf(name1,"c:\\nick\\mpeg\\dtrn\\%sv.prf",name);
```

```
    if (!(f_d = fopen(name1,"wb")))
    {
     sprintf(errortext,"Couldn't open %s\n",name1);
     error(errortext);
    }
    fwrite(predframe[2],1,chrom_width*chrom_height,f_d);
    fclose(f_d);

/* transform (module TR) */
    transform(predframe,neworg,mbinfo,blocks,1);

/* quantization & variable length coding */
    putpict(neworg[0]),

/* inverse quantization */
      for (k=0; k<mb_height*mb_width; k++)
      {
        if (mbinfo[k].mb_type & MB_INTRA)
          for (j=0, j<block_count; j++)
            iquant_intra(blocks[k*block_count+j],blocks[k*block_count+j],
                         dc_prec,intra_q,mbinfo[k].mquant);
        else
          for (j=0;j<block_count;j++)
            iquant_non_intra(blocks[k*block_count+j],blocks[k*block_count+j],
                             inter_q,mbinfo[k].mquant);
      }

/* inverse transform */
   transform(predframe,newref,mbinfo,blocks,0);

  }

  putseqend();
}
```

```
/**********************************/
/*                                */
/*         putseq5.cpp            */
/*                                */
/**********************************/

#include <stdio.h>
#include <string.h>
#include "c:\nick\mpeg\org\config.h"
#include "c:\nick\mpeg\org\global h"

void putseq()
{
 /* this routine assumes (N % M) == 0 */
 int i, j, k, f, f0, n, np, nb;
 int a, p, q, b, c;
 int sxf, syf, sxb, syb;
 FILE *f_mbi, *f_blk;
 char name[128], name1[128];
 unsigned char *neworg[3], *newref[3];

 rc_init_seq(); /* initialize rate control */

 /* sequence header, sequence extension and sequence display extension */
 putseqhdr();
 if (!mpeg1)
 {
  putseqext(),
  putseqdispext();
 }

/* loop through all frames in encoding/decoding order */
 for (i=0; i<nframes; i++)
 {
  /* f0: lowest frame number in current GOP
   *
   * first GOP contains N-(M-1) frames,
   * all other GOPs contain N frames
   */
  f0 = N*((i+(M-1))/N) - (M-1);

  if (f0<0) f0=0;

  if (i==0 || (i-1)%M==0)
  {
   /* I or P frame */
   for (j=0; j<3; j++)
   {
    /* shuffle reference frames */
    neworg[j] = oldorgframe[j],
    newref[j] = oldrefframe[j];
    oldorgframe[j] = neworgframe[j],
    oldrefframe[j] = newrefframe[j];
    neworgframe[j] = neworg[j];
```

```
      newrefframe[j] = newref[j];
    }

    /* f: frame number in display order */
    f = (i==0) ? 0 : i+M-1;
    if (f>=nframes) f = nframes - 1;

    if (i==f0) /* first displayed frame in GOP is I */
    {
      /* I frame */
      pict_type = I_TYPE;
      forw_hor_f_code = forw_vert_f_code = 15;
      back_hor_f_code = back_vert_f_code = 15;

      /* n: number of frames in current GOP
       *
       * first GOP contains (M-1) less (B) frames
       */
      n = (i==0) ? N-(M-1) : N;

      /* last GOP may contain less frames */
      if (n > nframes-f0) n = nframes-f0;

      /* number of P frames */
      if (i==0)
        np = (n + 2*(M-1))/M - 1; /* first GOP */
      else
        np = (n + (M-1))/M - 1;

      /* number of B frames */
      nb = n - np - 1;

      rc_init_GOP(np,nb);

      putgophdr(f0,i==0); /* set closed_GOP in first GOP only */
    }
    else
    {
      /* P frame */
      pict_type = P_TYPE;
      forw_hor_f_code = motion_data[0].forw_hor_f_code;
      forw_vert_f_code = motion_data[0].forw_vert_f_code;
      back_hor_f_code = back_vert_f_code = 15;
      sxf = motion_data[0].sxf;
      syf = motion_data[0].syf;
    }
  }
  else
  {
    /* B frame */
    for (j=0; j<3; j++)
    {
      neworg[j] = auxorgframe[j],
      newref[j] = auxframe[j];
```

```
    }

    /* f: frame number in display order */
    f = i - 1;
    pict_type = B_TYPE;
    n = (i-2)%M + 1; /* first B: n=1, second B: n=2, ... */
    forw_hor_f_code = motion_data[n].forw_hor_f_code;
    forw_vert_f_code = motion_data[n].forw_vert_f_code;
    back_hor_f_code = motion_data[n].back_hor_f_code;
    back_vert_f_code = motion_data[n].back_vert_f_code;
    sxf = motion_data[n].sxf;
    syf = motion_data[n].syf;
    sxb = motion_data[n].sxb;
    syb = motion_data[n].syb;
   }

   temp_ref = f - f0;
   frame_pred_dct = frame_pred_dct_tab[pict_type-1],
   q_scale_type = qscale_tab[pict_type-1];
   intravlc = intravlc_tab[pict_type-1];
   altscan = altscan_tab[pict_type-1];

/*  read in next frame for coding   */
   sprintf(name,tplorg,f+frame0),
   readframe(name,neworg);

/*  for frame pictures only   */
   pict_struct = FRAME_PICTURE;

/*  motion estimation (module ME)  */

    motion_estimation(oldorgframe[0], neworgframe[0],
                oldrefframe[0], newrefframe[0],
                neworg[0],
                sxf, syf,
                sxb, syb,
                mbinfo);


/*  prediction   */
    predict(oldrefframe,newrefframe,predframe,0,mbinfo);

/*  read in status data produced by the parallel program
   that implements module TR
*/
    sprintf(name1,"c:\\nick\\mpeg\\trn_out\\%s.mbi",name);
    if (!(f_mbi = fopen(name1,"r")))
    {
     sprintf(errortext,"Couldn't open %s\n",name1);
     error(errortext);
    }

    sprintf(name1,"c:\\nick\\mpeg\\trn_out\\%s.blk",name);
    if (!(f_blk = fopen(name1,"r")))
```

```
    {
     sprintf(errortext,"Couldn't open %s\n",name1);
     error(errortext);
    }

    a=0;
    for (p=0; p<mb_height2; p++)
    for (q=0, q<mb_width; q++)
    {
     fscanf(f_mbi,"%d ", &mbinfo[a].dct_type);

     for (b=0; b<block_count; b++)
     for (c=0; c<64; c++)
     fscanf(f_blk,"%d ", &blocks[a*block_count+b][c]);

     a++;
    }

    fclose(f_mbi);
    fclose(f_blk);

/*  quantization & variable length coding   */
    putpict(neworg[0]);

/*  inverse quantization   */
    for (k=0; k<mb_height*mb_width; k++)
    {
      if (mbinfo[k].mb_type & MB_INTRA)
        for (j=0; j<block_count; j++)
          iquant_intra(blocks[k*block_count+j],blocks[k*block_count+j],
                  dc_prec,intra_q,mbinfo[k].mquant);
      else
        for (j=0;j<block_count;j++)
          iquant_non_intra(blocks[k*block_count+j],blocks[k*block_count+j],
                    inter_q,mbinfo[k].mquant);
    }

/*  inverse transform   */
    transform(predframe,newref,mbinfo,blocks,0);

  }

  putseqend();
}
```

```
/**********************************/
/*                                */
/*        motion.cpp              */
/*                                */
/**********************************/

#include <stdio.h>
#include "c:\nick\mpeg\org\config.h"
#include "c:\nick\mpeg\org\global.h"

/* private prototypes */

static void frame_estimate _ANSI_ARGS_((unsigned char *org,
                    unsigned char *ref,
                    unsigned char *mb,
                    int i,
                    int j,
                    int sx,
                    int sy,
                    int *iminp,
                    int *jminp,
                    int *imintp,
                    int *jmintp,
                    int *iminbp,
                    int *jminbp,
                    float *dframep,
                    float *dfieldp,
                    int *tselp, int *bselp,
                    int imins[2][2],
                    int jmins[2][2]));

static float fullsearch _ANSI_ARGS_((unsigned char *org,
                    unsigned char *ref,
                    unsigned char *blk,
                    int lx,
                    int i0,
                    int j0,
                    int sx,
                    int sy,
                    char h,
                    int xmax,
                    int ymax,
                    int *iminp,
                    int *jminp));

static float dist1 _ANSI_ARGS_((unsigned char *blk1,
                    unsigned char *blk2,
                    int lx,
                    char hx,
                    char hy,
                    char h,
                    float distlim));

static float dist2 _ANSI_ARGS_((unsigned char *blk1,
```

```
                        unsigned char *blk2,
                        int lx,
                        char hx,
                        char hy,
                        char h));

static float bdist1 _ANSI_ARGS_((unsigned char *pf,
                        unsigned char *pb,
                        unsigned char *p2,
                        int lx,
                        int hxf,
                        int hyf,
                        int hxb,
                        int hyb,
                        char h));

static float bdist2 _ANSI_ARGS_((unsigned char *pf,
                        unsigned char *pb,
                        unsigned char *p2,
                        int lx,
                        int hxf,
                        int hyf,
                        int hxb,
                        int hyb,
                        char h));

static float variance _ANSI_ARGS_((unsigned char *p, int lx));

/*
 * motion estimation for progressive and interlaced frame pictures
 *
 * oldorg: source frame for forward prediction (used for P and B frames)
 * neworg: source frame for backward prediction (B frames only)
 * oldref: reconstructed frame for forward prediction (P and B frames)
 * newref: reconstructed frame for backward prediction (B frames only)
 * cur:    current frame (the one for which the prediction is formed)
 * sxf,syf: forward search window (frame coordinates)
 * sxb,syb: backward search window (frame coordinates)
 * mbi:    pointer to macroblock info structure
 *
 * results:
 * mbi->
 *  mb_type: 0, MB_INTRA, MB_FORWARD, MB_BACKWARD, MB_FORWARD|MB_BACKWARD
 *  MV[][][]: motion vectors (frame format)
 *  mv_field_sel: top/bottom field (for field prediction)
 *  motion_type: MC_FRAME, MC_FIELD
 *
 * uses global vars: pict_type, frame_pred_dct
 */
void motion_estimation(unsigned char *oldorg, unsigned char *neworg,
                unsigned char *oldref, unsigned char *newref,
                unsigned char *cur,
                int sxf, int syf,
```

```
                  int sxb, int syb,
                  struct mbinfo *mbi)
{
 int i, j;
 int imin,jmin,iminf,jminf,iminr,jminr;
 int imint,jmint,iminb,jminb;
 int imintf,jmintf,iminbf,jminbf;
 int imintr,jmintr,iminbr,jminbr,
 float var,v0;
 float dmc,dmcf,dmcr,dmci,vmc,vmcf,vmcr,vmci;
 float dmcfield,dmcfieldf,dmcfieldr,dmcfieldi,
 int tsel,bsel,tself,bself,tselr,bselr;
 unsigned char *mb;
 int imins[2][2],jmins[2][2];


 /* loop through all macroblocks of the picture */

 for (j=0, j<height2; j+=16)
 {
  for (i=0; i<width; i+=16)
  {

   /*  if (pict_struct==FRAME_PICTURE)  */

   mb = cur + i + width*j;

   var = variance(mb,width);

   /*  for I_type pictures     */

   if (pict_type==I_TYPE)
   mbi->mb_type = MB_INTRA;


   /*  for P_type pictures     */

   if (pict_type==P_TYPE)
   {
    frame_estimate(oldorg,oldref,mb,i,j,sxf,syf,
               &imin,&jmin,&imint,&jmint,&iminb,&jminb,
               &dmc,&dmcfield,&tsel,&bsel,imins,jmins);

    /* select between frame and field prediction */
    if (dmc<=dmcfield)
    {
     mbi->motion_type = MC_FRAME;
     vmc = dist2(oldref+(imin>>1)+width*(jmin>>1),mb,
             width,imin&1,jmin&1,16);
    }
    else
    {
     mbi->motion_type = MC_FIELD;
     dmc = dmcfield;
```

```
 vmc = dist2(oldref+(tsel?width:0)+(imint>>1)+(width<<1)*(jmint>>1),
          mb,width<<1,imint&1,jmint&1,8);
 vmc+= dist2(oldref+(bsel?width:0)+(iminb>>1)+(width<<1)*(jminb>>1),
          mb+width,width<<1,iminb&1,jminb&1,8);
}

/* select between intra or non-intra coding:
 *
 * selection is based on intra block variance (var) vs
 * prediction error variance (vmc)
 *
 * blocks with small prediction error are always coded non-intra
 * even if variance is smaller (is this reasonable?)
 */

if (vmc>var && vmc>=9*256)
mbi->mb_type = MB_INTRA;
else
{
 /* select between MC / No-MC
  *
  * use No-MC if var(No-MC) <= 1.25*var(MC)
  * (i.e slightly biased towards No-MC)
  *
  * blocks with small prediction error are always coded as No-MC
  * (requires no motion vectors, allows skipping)
  */

 v0 = dist2(oldref+i+width*j,mb,width,0,0,16);
 if (4*v0>5*vmc && v0>=9*256)
 {
  /* use MC */
  var = vmc;
  mbi->mb_type = MB_FORWARD;
  if (mbi->motion_type==MC_FRAME)
  {
   mbi->MV[0][0][0] = imin - (i<<1);
   mbi->MV[0][0][1] = jmin - (j<<1);
  }
  else /* if (mbi->motion_type==MC_FIELD) */
  {
   /* these are FRAME vectors */
   mbi->MV[0][0][0] = imint - (i<<1);
   mbi->MV[0][0][1] = (jmint<<1) - (j<<1);
   mbi->MV[1][0][0] = iminb - (i<<1);
   mbi->MV[1][0][1] = (jminb<<1) - (j<<1);
   mbi->mv_field_sel[0][0] = tsel;
   mbi->mv_field_sel[1][0] = bsel,
  }
 }
 else
 {
  /* No-MC */
  var = v0;
```

```
  mbi->mb_type = 0,
  mbi->motion_type = MC_FRAME,
  mbi->MV[0][0][0] = 0;
  mbi->MV[0][0][1] = 0;
 }
}
}


/* for B_type pictures    */

if (pict_type==B_TYPE)
{
 /* forward prediction */
 frame_estimate(oldorg,oldref,mb,i,j,sxf,syf,
            &iminf,&jminf,&imintf,&jmintf,&iminbf,&jminbf,
            &dmcf,&dmcfieldf,&tself,&bself,imins,jmins);

 /* backward prediction */
 frame_estimate(neworg,newref,mb,i,j,sxb,syb,
            &iminr,&jminr,&imintr,&jmintr,&iminbr,&jminbr,
            &dmcr,&dmcfieldr,&tselr,&bselr,imins,jmins);

 /* calculate interpolated distance */
 /* frame */
 dmci = bdist1(oldref+(iminf>>1)+width*(jminf>>1),
           newref+(iminr>>1)+width*(jminr>>1),
           mb,width,iminf&1,jminf&1,iminr&1,jminr&1,16);

 /* top field */
 dmcfieldi = bdist1(oldref+(imintf>>1)+(tself?width:0)+(width<<1)*(jmintf>>1),
              newref+(imintr>>1)+(tselr?width:0)+(width<<1)*(jmintr>>1),
              mb,width<<1,imintf&1,jmintf&1,imintr&1,jmintr&1,8);

 /* bottom field */
 dmcfieldi+= bdist1(oldref+(iminbf>>1)+(bself?width:0)+(width<<1)*(jminbf>>1),
              newref+(iminbr>>1)+(bselr?width:0)+(width<<1)*(jminbr>>1),
              mb+width,width<<1,iminbf&1,jminbf&1,iminbr&1,jminbr&1,8),

 /* select prediction type of minimum distance from the
  * six candidates (field/frame * forward/backward/interpolated)
  */

 if (dmci<dmcfieldi && dmci<dmcf && dmci<dmcfieldf
    && dmci<dmcr && dmci<dmcfieldr)
 {
  /* frame, interpolated */
  mbi->mb_type = MB_FORWARD|MB_BACKWARD;
  mbi->motion_type = MC_FRAME,
  vmc = bdist2(oldref+(iminf>>1)+width*(jminf>>1),
           newref+(iminr>>1)+width*(jminr>>1),
           mb,width,iminf&1,jminf&1,iminr&1,jminr&1,16);
 }
 else if (dmcfieldi<dmcf && dmcfieldi<dmcfieldf
```

```
      && dmcfieldi<dmcr && dmcfieldi<dmcfieldr)
 {
  /* field, interpolated */
  mbi->mb_type = MB_FORWARD|MB_BACKWARD;
  mbi->motion_type = MC_FIELD;
  vmc = bdist2(oldref+(imintf>>1)+(tself?width:0)+(width<<1)*(jmintf>>1),
            newref+(imintr>>1)+(tselr?width:0)+(width<<1)*(jmintr>>1),
            mb,width<<1,imintf&1,jmintf&1,imintr&1,jmintr&1,8);
  vmc+= bdist2(oldref+(iminbf>>1)+(bself?width:0)+(width<<1)*(jminbf>>1),
            newref+(iminbr>>1)+(bselr?width:0)+(width<<1)*(jminbr>>1),
            mb+width,width<<1,iminbf&1,jminbf&1,iminbr&1,jminbr&1,8);
 }
 else if (dmcf<dmcfieldf && dmcf<dmcr && dmcf<dmcfieldr)
 {
  /* frame, forward */
  mbi->mb_type = MB_FORWARD;
  mbi->motion_type = MC_FRAME;
  vmc = dist2(oldref+(iminf>>1)+width*(jminf>>1),mb,
           width,iminf&1,jminf&1,16);
 }
 else if (dmcfieldf<dmcr && dmcfieldf<dmcfieldr)
 {
  /* field, forward */
  mbi->mb_type = MB_FORWARD;
  mbi->motion_type = MC_FIELD;
  vmc = dist2(oldref+(tself?width:0)+(imintf>>1)+(width<<1)*(jmintf>>1),
           mb,width<<1,imintf&1,jmintf&1,8);
  vmc+= dist2(oldref+(bself?width:0)+(iminbf>>1)+(width<<1)*(jminbf>>1),
           mb+width,width<<1,iminbf&1,jminbf&1,8),
 }
 else if (dmcr<dmcfieldr)
 {
  /* frame, backward */
  mbi->mb_type = MB_BACKWARD;
  mbi->motion_type = MC_FRAME;
  vmc = dist2(newref+(iminr>>1)+width*(jminr>>1),mb,
           width,iminr&1,jminr&1,16);
 }
 else
 {
  /* field, backward */
  mbi->mb_type = MB_BACKWARD;
  mbi->motion_type = MC_FIELD;
  vmc = dist2(newref+(tselr?width:0)+(imintr>>1)+(width<<1)*(jmintr>>1),
           mb,width<<1,imintr&1,jmintr&1,8);
  vmc+= dist2(newref+(bselr?width:0)+(iminbr>>1)+(width<<1)*(jminbr>>1),
           mb+width,width<<1,iminbr&1,jminbr&1,8);
 }

/* select between intra or non-intra coding:
 *
 * selection is based on intra block variance (var) vs.
 * prediction error variance (vmc)
 *
```

```
 * blocks with small prediction error are always coded non-intra
 * even if variance is smaller (is this reasonable?)
 */

if (vmc>var && vmc>=9*256)
 mbi->mb_type = MB_INTRA;
else
{
 var = vmc;
 if (mbi->motion_type==MC_FRAME)
 {
  /* forward */
  mbi->MV[0][0][0] = iminf - (i<<1);
  mbi->MV[0][0][1] = jminf - (j<<1);
  /* backward */
  mbi->MV[0][1][0] = iminr - (i<<1);
  mbi->MV[0][1][1] = jminr - (j<<1);
 }
 else
 {
  /* these are FRAME vectors */
  /* forward */
  mbi->MV[0][0][0] = imintf - (i<<1);
  mbi->MV[0][0][1] = (jmintf<<1) - (j<<1);
  mbi->MV[1][0][0] = iminbf - (i<<1);
  mbi->MV[1][0][1] = (jminbf<<1) - (j<<1);
  mbi->mv_field_sel[0][0] = tself;
  mbi->mv_field_sel[1][0] = bself;
  /* backward */
  mbi->MV[0][1][0] = imintr - (i<<1);
  mbi->MV[0][1][1] = (jmintr<<1) - (j<<1);
  mbi->MV[1][1][0] = iminbr - (i<<1);
  mbi->MV[1][1][1] = (jminbr<<1) - (j<<1);
  mbi->mv_field_sel[0][1] = tselr;
  mbi->mv_field_sel[1][1] = bselr;
 }
}
}

mbi->var = var;


mbi++;

}

if (!quiet)
{
 putc('.',stderr);
 fflush(stderr);
}

}
```

```
  if (!quiet)  putc('\n',stderr);
}


/*
 * frame picture motion estimation
 *
 * org: top left pel of source reference frame
 * ref: top left pel of reconstructed reference frame
 * mb:  macroblock to be matched
 * i,j: location of mb relative to ref (=center of search window)
 * sx,sy: half widths of search window
 * iminp,jminp,dframep: location and value of best frame prediction
 * imintp,jmintp,tselp: location of best field pred. for top field of mb
 * iminbp,jminbp,bselp: location of best field pred. for bottom field of mb
 * dfieldp: value of field prediction
 */
static void frame_estimate(unsigned char *org, unsigned char *ref,
                    unsigned char *mb,
                    int i, int j,
                    int sx, int sy,
                    int *iminp, int *jminp,
                    int *imintp, int *jmintp,
                    int *iminbp, int *jminbp,
                    float *dframep, float *dfieldp,
                    int *tselp, int *bselp,
                    int imins[2][2], int jmins[2][2] )
{
  float dt,db,dmint,dminb;
  int imint,iminb,jmint,jminb;

  /* frame prediction */
  *dframep = fullsearch(org,ref,mb,width,i,j,sx,sy,16,width,height,
                  iminp,jminp);

  /* predict top field from top field */
  dt = fullsearch(org,ref,mb,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
             &imint,&jmint);

  /* predict top field from bottom field */
  db = fullsearch(org+width,ref+width,mb,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
             &iminb,&jminb);

  imins[0][0] = imint;
  jmins[0][0] = jmint;
  imins[1][0] = iminb;
  jmins[1][0] = jminb;

  /* select prediction for top field */
  if (dt<=db)
  {
   dmint=dt; *imintp=imint; *jmintp=jmint; *tselp=0;
  }
  else
```

```
{
 dmint=db; *imintp=iminb; *jmintp=jminb; *tselp=1;
}

/* predict bottom field from top field */
dt = fullsearch(org,ref,mb+width,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
            &imint,&jmint);

/* predict bottom field from bottom field */
db = fullsearch(org+width, ref+width, mb+width,
            width<<1, i, j>>1, sx, sy>>1, 8, width, height>>1,
            &iminb,&jminb);

imins[0][1] = imint;
jmins[0][1] = jmint;
imins[1][1] = iminb;
jmins[1][1] = jminb;

/* select prediction for bottom field */
if (db<=dt)
{
 dminb=db, *iminbp=iminb; *jminbp=jminb; *bselp=1,
}
else
{
 dminb=dt; *iminbp=imint; *jminbp=jmint; *bselp=0;
}

 *dfieldp=dmint+dminb;
}



/*
 * full search block matching
 *
 * blk: top left pel of (16*h) block
 * h: height of block
 * lx: distance (in bytes) of vertically adjacent pels in ref,blk
 * org: top left pel of source reference picture
 * ref: top left pel of reconstructed reference picture
 * i0,j0: center of search window
 * sx,sy: half widths of search window
 * xmax,ymax: right/bottom limits of search area
 * iminp,jminp: pointers to where the result is stored
 *          result is given as half pel offset from ref(0,0)
 *          i.e. NOT relative to (i0,j0)
 */
static float fullsearch(unsigned char *org, unsigned char *ref,
                unsigned char *blk,
                int lx, int i0, int j0,
                int sx, int sy,
                char h,
                int xmax, int ymax,
```

```
                  int *iminp, int *jminp)
{
 int i,j,imin,jmin,ilow,ihigh,jlow,jhigh;
 float d,dmin;
 int k,l,sxy;

 ilow = i0 - sx;
 ihigh = i0 + sx;

 if (ilow<0)
  ilow = 0;

 if (ihigh>xmax-16)
  ihigh = xmax-16;

 jlow = j0 - sy;
 jhigh = j0 + sy;

 if (jlow<0)
  jlow = 0;

 if (jhigh>ymax-h)
  jhigh = ymax-h;

 /* full pel search, spiraling outwards */

 imin = i0;
 jmin = j0;
 dmin = dist1(org+imin+lx*jmin,blk,lx,0,0,h,65536.0);

 sxy = (sx>sy) ? sx : sy,

 for (l=1; l<=sxy; l++)
 {
  i = i0 - l;
  j = j0 - l;
  for (k=0, k<8*l; k++)
  {
   if (i>=ilow && i<=ihigh && j>=jlow && j<=jhigh)
   {
    d = dist1(org+i+lx*j,blk,lx,0,0,h,dmin);

    if (d<dmin)
    {
     dmin = d;
     imin = i;
     jmin = j;
    }
   }

   if      (k<2*l) i++;
   else if (k<4*l) j++;
   else if (k<6*l) i--;
   else            j--;
```

```
    }
  }

  /* half pel */
  dmin = 65536.0;
  imin <<= 1;
  jmin <<= 1;
  ilow = imin - (imin>0);
  ihigh = imin + (imin<((xmax-16)<<1));
  jlow = jmin - (jmin>0);
  jhigh = jmin + (jmin<((ymax-h)<<1));

  for (j=jlow; j<=jhigh; j++)
    for (i=ilow; i<=ihigh; i++)
    {
      d = dist1(ref+(i>>1)+lx*(j>>1),blk,lx,i&1,j&1,h,dmin),

      if (d<dmin)
      {
        dmin = d;
        imin = i;
        jmin = j;
      }
    }

  *iminp = imin;
  *jminp = jmin;

  return dmin;
}

/*
 * total absolute difference between two (16*h) blocks
 * including optional half pel interpolation of blk1 (hx,hy)
 * blk1,blk2: addresses of top left pels of both blocks
 * lx:       distance (in bytes) of vertically adjacent pels
 * hx,hy:    flags for horizontal and/or vertical interpolation
 * h:        height of block (usually 8 or 16)
 * distlim:  bail out if sum exceeds this value
 */
static float dist1(unsigned char *blk1,
                   unsigned char *blk2,
                   int lx,
                   char hx,
                   char hy,
                   char h,
                   float distlim )
{
  unsigned char *p1,*p1a,*p2;
  char i, j;
  int v;
  float s;

  s = 0,
```

```
p1 = blk1;
p2 = blk2;

if (!hx && !hy)
 for (j=0; j<h; j++)
 {
  if ((v = p1[0]  - p2[0])<0)  v = -v; s+= (float) v;
  if ((v = p1[1]  - p2[1])<0)  v = -v; s+= (float) v;
  if ((v = p1[2]  - p2[2])<0)  v = -v; s+= (float) v;
  if ((v = p1[3]  - p2[3])<0)  v = -v; s+= (float) v;
  if ((v = p1[4]  - p2[4])<0)  v = -v; s+= (float) v;
  if ((v = p1[5]  - p2[5])<0)  v = -v; s+= (float) v;
  if ((v = p1[6]  - p2[6])<0)  v = -v; s+= (float) v;
  if ((v = p1[7]  - p2[7])<0)  v = -v; s+= (float) v;
  if ((v = p1[8]  - p2[8])<0)  v = -v; s+= (float) v;
  if ((v = p1[9]  - p2[9])<0)  v = -v; s+= (float) v;
  if ((v = p1[10] - p2[10])<0) v = -v; s+= (float) v;
  if ((v = p1[11] - p2[11])<0) v = -v; s+= (float) v;
  if ((v = p1[12] - p2[12])<0) v = -v; s+= (float) v;
  if ((v = p1[13] - p2[13])<0) v = -v; s+= (float) v;
  if ((v = p1[14] - p2[14])<0) v = -v; s+= (float) v;
  if ((v = p1[15] - p2[15])<0) v = -v; s+= (float) v;

  if (s >= distlim)
   break;

  p1+= lx;
  p2+= lx;
 }
else if (hx && !hy)
 for (j=0; j<h; j++)
 {
  for (i=0; i<16; i++)
  {
   v = (p1[i]+p1[i+1]+1)>>1 - p2[i];
   if (v>=0)
    s+= (float) v;
   else
    s-= (float) v;
  }
  p1+= lx;
  p2+= lx;
 }
else if (!hx && hy)
{
 p1a = p1 + lx;
 for (j=0; j<h; j++)
 {
  for (i=0; i<16; i++)
  {
   v = (p1[i]+p1a[i]+1)>>1 - p2[i];
   if (v>=0)
    s+= (float) v;
   else
```

```
        s-= (float) v;
      }
      p1 = p1a;
      p1a+= lx;
      p2+= lx,
    }
  }
  else /* if (hx && hy) */
  {
    p1a = p1 + lx;
    for (j=0; j<h; j++)
    {
      for (i=0; i<16, i++)
      {
        v = (p1[i]+p1[i+1]+p1a[i]+p1a[i+1]+2)>>2 - p2[i];
        if (v>=0)
          s+= (float) v;
        else
          s-= (float) v;
      }
      p1 = p1a;
      p1a+= lx;
      p2+= lx;
    }
  }

  return s;
}

/*
 * total squared difference between two (16*h) blocks
 * including optional half pel interpolation of blk1 (hx,hy)
 * blk1,blk2: addresses of top left pels of both blocks
 * lx:      distance (in bytes) of vertically adjacent pels
 * hx,hy.    flags for horizontal and/or vertical interpolation
 * h:       height of block (usually 8 or 16)
 */
static float dist2(unsigned char *blk1,
                   unsigned char *blk2,
                   int lx,
                   char hx,
                   char hy,
                   char h )
{
  unsigned char *p1,*p1a,*p2;
  char i, j;
  int v;
  float s;

  s = 0,
  p1 = blk1;
  p2 = blk2;
  if (!hx && !hy)
    for (j=0; j<h; j++)
```

```
  {
   for (i=0; i<16; i++)
   {
    v = p1[i] - p2[i];
    s+= (float) (v*v);
   }
   p1+= lx;
   p2+= lx;
  }
 else if (hx && !hy)
  for (j=0; j<h; j++)
  {
   for (i=0, i<16; i++)
   {
    v = (p1[i]+p1[i+1]+1)>>1 - p2[i];
    s+= (float) (v*v);
   }
   p1+= lx;
   p2+= lx;
  }
 else if (!hx && hy)
 {
  p1a = p1 + lx;
  for (j=0; j<h; j++)
  {
   for (i=0; i<16; i++)
   {
    v = (p1[i]+p1a[i]+1)>>1 - p2[i];
    s+= (float) (v*v);
   }
   p1 = p1a;
   p1a+= lx;
   p2+= lx;
  }
 }
 else /* if (hx && hy) */
 {
  p1a = p1 + lx;
  for (j=0, j<h; j++)
  {
   for (i=0; i<16; i++)
   {
    v = (p1[i]+p1[i+1]+p1a[i]+p1a[i+1]+2)>>2 - p2[i];
    s+= (float) (v*v);
   }
   p1 = p1a;
   p1a+= lx;
   p2+= lx;
  }
 }

 return s;
}
```

```
/*
 * absolute difference error between a (16*h) block and a bidirectional
 * prediction
 *
 * p2: address of top left pel of block
 * pf,hxf,hyf: address and half pel flags of forward ref. block
 * pb,hxb,hyb: address and half pel flags of backward ref. block
 * h: height of block
 * lx: distance (in bytes) of vertically adjacent pels in p2,pf,pb
 */
static float bdist1(unsigned char *pf, unsigned char *pb, unsigned char *p2,
                    int lx, int hxf,
                    int hyf, int hxb,
                    int hyb, char h )
{
  unsigned char *pfa,*pfb,*pfc,*pba,*pbb,*pbc;
  char i, j;
  int v;
  float s;

  pfa = pf + hxf,
  pfb = pf + lx*hyf;
  pfc = pfb + hxf,

  pba = pb + hxb;
  pbb = pb + lx*hyb,
  pbc = pbb + hxb;

  s = 0;

  for (j=0; j<h; j++)
  {
    for (i=0; i<16; i++)
    {
      v = ((*pf++ + *pfa++ + *pfb++ + *pfc++ + 2)>>2 +
           (*pb++ + *pba++ + *pbb++ + *pbc++ + 2)>>2 + 1)>>1
          - *p2++,

      if (v>=0)
        s+= (float) v;
      else
        s-= (float) v;
    }
    p2+= lx-16;
    pf+= lx-16;
    pfa+= lx-16,
    pfb+= lx-16;
    pfc+= lx-16;
    pb+= lx-16;
    pba+= lx-16;
    pbb+= lx-16;
    pbc+= lx-16;
  }
```

```
 return s;
}

/*
 * squared error between a (16*h) block and a bidirectional
 * prediction
 *
 * p2: address of top left pel of block
 * pf,hxf,hyf: address and half pel flags of forward ref. block
 * pb,hxb,hyb: address and half pel flags of backward ref. block
 * h: height of block
 * lx: distance (in bytes) of vertically adjacent pels in p2,pf,pb
 */
static float bdist2(unsigned char *pf, unsigned char *pb, unsigned char *p2,
                  int lx, int hxf,
                  int hyf, int hxb,
                  int hyb, char h )
{
 unsigned char *pfa,*pfb,*pfc,*pba,*pbb,*pbc;
 char i, j;
 int v;
 float s;

 pfa = pf + hxf;
 pfb = pf + lx*hyf;
 pfc = pfb + hxf;

 pba = pb + hxb;
 pbb = pb + lx*hyb;
 pbc = pbb + hxb;

 s = 0;

 for (j=0; j<h; j++)
 {
  for (i=0; i<16; i++)
  {
   v = ((*pf++ + *pfa++ + *pfb++ + *pfc++ + 2)>>2 +
        (*pb++ + *pba++ + *pbb++ + *pbc++ + 2)>>2 + 1)>>1
       - *p2++;
   s+= (float) (v*v);
  }
  p2+= lx-16;
  pf+= lx-16;
  pfa+= lx-16;
  pfb+= lx-16;
  pfc+= lx-16;
  pb+= lx-16;
  pba+= lx-16,
  pbb+= lx-16,
  pbc+= lx-16;
 }

 return s;
```

```
}

/*
 * variance of a (16*16) block, multiplied by 256
 * p:  address of top left pel of block
 * lx: distance (ın bytes) of vertically adjacent pels
 */
static float variance(unsigned char *p, int lx )
{
  char i, j;
  unsigned int v;
  float s, s2;

  s = s2 = 0,

  for (j=0; j<16; j++)
  {
    for (ı=0; i<16, ı++)
    {
      v = *p++;
      s+= (float) v;
      s2+= (float) (v*v);
    }
    p+= lx-16;
  }
  s=0.0625*s;
  return s2 - s*s,
}
```

```
/**********************************/
/*                                */
/*      transfrm.cpp              */
/*                                */
/**********************************/

#include <stdio.h>
#include <math.h>
#include "c:\nick\mpeg\org\config.h"
#include "c:\nick\mpeg\org\global.h"


/* private prototypes*/
static void add_sub_pred _ANSI_ARGS_((unsigned char *pred,
                         unsigned char *cur,
                         int lx, short *blk,
                         char add_sub));
/* private prototypes */
static void predict_mb _ANSI_ARGS_((unsigned char *oldref[],
                        unsigned char *newref[],
                        unsigned char *cur[],
                        int lx, int bx, int by,
                        int pict_type, int pict_struct,
                        int mb_type, int motion_type, int secondfield,
                        int PMV[2][2][2],
                        int mv_field_sel[2][2],
                        int dmvector[2]));

static void pred _ANSI_ARGS_((unsigned char *src[],
                  int sfield,
                  unsigned char *dst[],
                  int dfield,
                  int lx, int w, int h, int x, int y,
                  int dx, int dy, int addflag));

static void pred_comp _ANSI_ARGS_((unsigned char *src, unsigned char *dst,
  int lx, int w, int h, int x, int y, int dx, int dy, int addflag));

static void calc_DMV _ANSI_ARGS_((int DMV[][2],
                      int *dmvector,
                      int mvx, int mvy));

static void clearblock _ANSI_ARGS_((unsigned char *cur[], int i0, int j0));

/*
 * select between frame and field DCT
 *
 * preliminary version: based on inter-field correlation
 */
/* subtract prediction and transform prediction error */
/* inverse transform prediction error and add prediction */

void transform(unsigned char *pred[],
          unsigned char *cur[],
```

```
              struct mbinfo *mbi,
              short blocks[][64],
              char forward_invers )
{
  int i, j, i1, j1, k, n, cc, offs, lx;

/*
 * select between frame and field DCT
 *
 * preliminary version: based on inter-field correlation
 */

  if (forward_invers)
  dct_type_estimation(pred[0], cur[0], mbi ) ;

/* subtract prediction and transform prediction error */
/* inverse transform prediction error and add prediction */

  k = 0;
  for (j=0; j<height2; j+=16)
    for (i=0, i<width; i+=16)
    {
      for (n=0; n<block_count; n++)
      {
        cc = (n<4) ? 0 : (n&1)+1, /* color component index */
        if (cc==0)
        {
          /* luminance */
          if ((pict_struct==FRAME_PICTURE) && mbi[k].dct_type)
          {
            /* field DCT */
            offs = i + ((n&1)<<3) + width*(j+((n&2)>>1));
            lx = width<<1;
          }
          else
          {
            /* frame DCT */
            offs = i + ((n&1)<<3) + width2*(j+((n&2)<<2));
            lx = width2,
          }

          if (pict_struct==BOTTOM_FIELD)
            offs += width;
        }
        else
        {
          /* chrominance */

          /* scale coordinates */
          i1 = (chroma_format==CHROMA444) ? i : i>>1;
          j1 = (chroma_format!=CHROMA420) ? j · j>>1;

          if ((pict_struct==FRAME_PICTURE) && mbi[k].dct_type
              && (chroma_format!=CHROMA420))
```

```
        {
         /* field DCT */
         offs = i1 + (n&8) + chrom_width*(j1+((n&2)>>1));
         lx = chrom_width<<1;
        }
        else
        {
         /* frame DCT */
         offs = i1 + (n&8) + chrom_width2*(j1+((n&2)<<2)),
         lx = chrom_width2;
        }

        if (pict_struct==BOTTOM_FIELD)
         offs += chrom_width;
       }

       if (forward_invers) /* 1 for forward DCT */
       {
        add_sub_pred(pred[cc]+offs,
                 cur[cc]+offs,
                 lx,
                 blocks[k*block_count+n],
                 0);
        fdct(blocks[k*block_count+n]);
       }
       else /* 0 for invers DCT */
       {
        idct(blocks[k*block_count+n]);
        add_sub_pred(pred[cc]+offs,
                 cur[cc]+offs,
                 lx,
                 blocks[k*block_count+n],
                 1);
       }
      }

     k++;
    }
}


/* add prediction and prediction error, saturate to 0...255 */
/* subtract prediction from block data */
static void add_sub_pred(unsigned char *pred,
                  unsigned char *cur,
                  int lx, short *blk,
                  char add_sub )
{
 int i, j;

 for (j=0; j<8; j++)
 {
  for (i=0; i<8; i++)
```

```
    {
     if (add_sub) /* 1 for addition */
     cur[i] = clp[ blk[i] + pred[i] ],
     else /* 0 for subtractction */
     blk[i] = cur[i] - pred[i];
    }

   blk+= 8;
   cur+= lx;
   pred+= lx;
  }
}


/*
 * select between frame and field DCT
 *
 * preliminary version· based on inter-field correlation
 */
void dct_type_estimation(unsigned char *pred,
                         unsigned char *cur,
                         struct mbinfo *mbi )
{
 short blk0[128], blk1[128];
 int i, j, i0, j0, k, offs, s0, s1, sq0, sq1, s01;
 double d, r;

 k = 0;

 for (j0=0; j0<height2; j0+=16)
  for (i0=0; i0<width; i0+=16)
  {
   if (frame_pred_dct || pict_struct!=FRAME_PICTURE)
     mbi[k].dct_type = 0;
   else
   {
     /* interlaced frame picture */
     /*
      * calculate prediction error (cur-pred) for top (blk0)
      * and bottom field (blk1)
      */
     for (j=0, j<8; j++)
     {
      offs = width*((j<<1)+j0) + i0;
      for (i=0; i<16; i++)
      {
        blk0[16*j+i] = cur[offs] - pred[offs];
        blk1[16*j+i] = cur[offs+width] - pred[offs+width];
        offs++;
      }
     }
     /* correlate fields */
     s0=s1=sq0=sq1=s01=0;
```

```
      for (i=0; i<128; i++)
      {
        s0+= blk0[i];
        sq0+= blk0[i]*blk0[i];
        s1+= blk1[i];
        sq1+= blk1[i]*blk1[i];
        s01+= blk0[i]*blk1[i];
      }

      d = (sq0-(s0*s0)/128.0)*(sq1-(s1*s1)/128.0);

      if (d>0.0)
      {
        r = (s01-(s0*s1)/128.0)/sqrt(d);
        if (r>0.5)
          mbi[k].dct_type = 0; /* frame DCT */
        else
          mbi[k].dct_type = 1; /* field DCT */
      }
      else
        mbi[k].dct_type = 1; /* field DCT */
    }
    k++;
  }
}

/* fdctref.c, forward discrete cosine transform, double precision          */

#ifndef PI
# ifdef M_PI
#  define PI M_PI
# else
#  define PI 3.14159265358979323846
# endif
#endif


/* private data */
static double c[8][8]; /* transform coefficients */

void init_fdct()
{
  int i, j;
  double s;

  for (i=0; i<8; i++)
  {
    s = (i==0) ? sqrt(0.125) : 0.5;

    for (j=0, j<8; j++)
      c[i][j] = s * cos((PI/8.0)*i*(j+0.5)),
  }
}
```

```
void fdct(short *block)
{
 int i, j, k;
 double s;
 double tmp[64];

 for (i=0; i<8; i++)
  for (j=0; j<8; j++)
  {
   s = 0.0;

   for (k=0; k<8; k++)
    s += c[j][k] * block[8*i+k];

   tmp[8*i+j] = s;
  }

 for (j=0; j<8; j++)
  for (i=0; i<8; i++)
  {
   s = 0.0;

   for (k=0, k<8, k++)
    s += c[i][k] * tmp[8*k+j];

   block[8*i+j] = (int)floor(s+0.499999);
   /*
    * reason for adding 0.499999 instead of 0.5.
    * s is quite often x.5 (at least for i and/or j = 0 or 4)
    * and setting the rounding threshold exactly to 0.5 leads to an
    * extremely high arithmetic implementation dependency of the result;
    * s being between x.5 and x.500001 (which is now incorrectly rounded
    * downwards instead of upwards) is assumed to occur less often
    * (if at all)
    */
  }
}


/* idct.c, inverse fast discrete cosine transform                    */

/**********************************************************/
/* inverse two dimensional DCT, Chen-Wang algorithm     */
/* (cf. IEEE ASSP-32, pp 803-816, Aug. 1984)            */
/* 32-bit integer arithmetic (8 bit coefficients)       */
/* 11 mults, 29 adds per DCT                            */
/*                         sE, 18.8.91    */
/**********************************************************/
/* coefficients extended to 12 bit for IEEE1180-1990    */
/* compliance                  sE, 2.1.94     */
/**********************************************************/

/* this code assumes >> to be a two's-complement arithmetic */
```

```
/* right shift: (-2)>>1 == -1 , (-3)>>1 == -2              */


/* row (horizontal) IDCT
 *
 *         7                    pi        1
 * dst[k] = sum c[l] * src[l] * cos( -- * ( k + - ) * l )
 *        l=0                   8        2
 *
 * where: c[0]   = 128
 *        c[1..7] = 128*sqrt(2)
 */

#define W1 2841 /* 2048*sqrt(2)*cos(1*pi/16) */
#define W2 2676 /* 2048*sqrt(2)*cos(2*pi/16) */
#define W3 2408 /* 2048*sqrt(2)*cos(3*pi/16) */
#define W5 1609 /* 2048*sqrt(2)*cos(5*pi/16) */
#define W6 1108 /* 2048*sqrt(2)*cos(6*pi/16) */
#define W7 565  /* 2048*sqrt(2)*cos(7*pi/16) */

/* private data */
static short iclip[1024]; /* clipping table */
static short *iclp;

/* private prototypes */
static void idctrow _ANSI_ARGS_((short *blk));
static void idctcol _ANSI_ARGS_((short *blk));


static void idctrow(short *blk)
{
 int x0, x1, x2, x3, x4, x5, x6, x7, x8;

 /* shortcut */
 if (!((x1 = blk[4]<<11) | (x2 = blk[6]) | (x3 = blk[2]) |
       (x4 = blk[1]) | (x5 = blk[7]) | (x6 = blk[5]) | (x7 = blk[3])))
 {
  blk[0]=blk[1]=blk[2]=blk[3]=blk[4]=blk[5]=blk[6]=blk[7]=blk[0]<<3;
  return;
 }

 x0 = (blk[0]<<11) + 128; /* for proper rounding in the fourth stage */

 /* first stage */
 x8 = W7*(x4+x5);
 x4 = x8 + (W1-W7)*x4;
 x5 = x8 - (W1+W7)*x5;
 x8 = W3*(x6+x7);
 x6 = x8 - (W3-W5)*x6;
 x7 = x8 - (W3+W5)*x7;

 /* second stage */
 x8 = x0 + x1;
 x0 -= x1;
```

```
  x1 = W6*(x3+x2);
  x2 = x1 - (W2+W6)*x2,
  x3 = x1 + (W2-W6)*x3,
  x1 = x4 + x6,
  x4 -= x6;
  x6 = x5 + x7,
  x5 -= x7;

  /* third stage */
  x7 = x8 + x3;
  x8 -= x3;
  x3 = x0 + x2;
  x0 -= x2;
  x2 = (181*(x4+x5)+128)>>8,
  x4 = (181*(x4-x5)+128)>>8;

  /* fourth stage */
  blk[0] = (x7+x1)>>8;
  blk[1] = (x3+x2)>>8,
  blk[2] = (x0+x4)>>8,
  blk[3] = (x8+x6)>>8;
  blk[4] = (x8-x6)>>8;
  blk[5] = (x0-x4)>>8;
  blk[6] = (x3-x2)>>8;
  blk[7] = (x7-x1)>>8;
}

/* column (vertical) IDCT
 *
 *             7                         pi         1
 * dst[8*k] = sum c[l] * src[8*l] * cos( -- * ( k + - ) * l )
 *            l=0                        8          2
 *
 * where: c[0]    = 1/1024
 *        c[1..7] = (1/1024)*sqrt(2)
 */
static void idctcol(short *blk)
{
  int x0, x1, x2, x3, x4, x5, x6, x7, x8,

  /* shortcut */
  if (!((x1 = (blk[8*4]<<8)) | (x2 = blk[8*6]) | (x3 = blk[8*2]) |
        (x4 = blk[8*1]) | (x5 = blk[8*7]) | (x6 = blk[8*5]) | (x7 = blk[8*3])))
  {
    blk[8*0]=blk[8*1]=blk[8*2]=blk[8*3]=blk[8*4]=blk[8*5]=blk[8*6]=blk[8*7]=
      iclp[(blk[8*0]+32)>>6];
    return;
  }

  x0 = (blk[8*0]<<8) + 8192,

  /* first stage */
  x8 = W7*(x4+x5) + 4;
  x4 = (x8+(W1-W7)*x4)>>3;
```

```
    x5 = (x8-(W1+W7)*x5)>>3,
    x8 = W3*(x6+x7) + 4;
    x6 = (x8-(W3-W5)*x6)>>3;
    x7 = (x8-(W3+W5)*x7)>>3;

    /* second stage */
    x8 = x0 + x1;
    x0 -= x1;
    x1 = W6*(x3+x2) + 4;
    x2 = (x1-(W2+W6)*x2)>>3;
    x3 = (x1+(W2-W6)*x3)>>3;
    x1 = x4 + x6;
    x4 -= x6;
    x6 = x5 + x7;
    x5 -= x7;

    /* third stage */
    x7 = x8 + x3;
    x8 -= x3;
    x3 = x0 + x2;
    x0 -= x2;
    x2 = (181*(x4+x5)+128)>>8;
    x4 = (181*(x4-x5)+128)>>8;

    /* fourth stage */
    blk[8*0] = iclp[(x7+x1)>>14];
    blk[8*1] = iclp[(x3+x2)>>14];
    blk[8*2] = iclp[(x0+x4)>>14];
    blk[8*3] = iclp[(x8+x6)>>14];
    blk[8*4] = iclp[(x8-x6)>>14];
    blk[8*5] = iclp[(x0-x4)>>14];
    blk[8*6] = iclp[(x3-x2)>>14];
    blk[8*7] = iclp[(x7-x1)>>14],
}

/* two dimensional inverse discrete cosine transform */
void idct(short *block)
{
  int i;

  for (i=0; i<8; i++)
    idctrow(block+8*i);

  for (i=0; i<8; i++)
    idctcol(block+i);
}

void init_idct()
{
  int i;

  iclp = iclip+512;
  for (i= -512; i<512; i++)
    iclp[i] = (i<-256) ? -256 : ((i>255) ? 255 : i);
```

```
}

/* form prediction for a complete picture (frontend for predict_mb)
 *
 * reff: reference frame for forward prediction
 * refb: reference frame for backward prediction
 * cur:  destination (current) frame
 * secondfield: predict second field of a frame
 * mbi:  macroblock info
 *
 * Notes:
 * - cf. predict_mb
 */

void predict(unsigned char *reff[], unsigned char *refb[],
        unsigned char *cur[3], int secondfield,
        struct mbinfo *mbi )
{
 int i, j, k;

 k = 0;

 /* loop through all macroblocks of the picture */
 for (j=0; j<height2; j+=16)
  for (i=0; i<width; i+=16)
  {
   predict_mb(reff,refb,cur,width,i,j,pict_type,pict_struct,
          mbi[k].mb_type,mbi[k].motion_type,secondfield,
          mbi[k].MV,mbi[k].mv_field_sel,mbi[k].dmvector);

   k++;
  }
}

/* form prediction for one macroblock
 *
 * oldref: reference frame for forward prediction
 * newref: reference frame for backward prediction
 * cur:   destination (current) frame
 * lx:    frame width (identical to global var `width')
 * bx,by:  picture (field or frame) coordinates of macroblock to be predicted
 * pict_type: I, P or B
 * pict_struct: FRAME_PICTURE, TOP_FIELD, BOTTOM_FIELD
 * mb_type.    MB_FORWARD, MB_BACKWARD, MB_INTRA
 * motion_type: MC_FRAME, MC_FIELD, MC_16X8, MC_DMV
 * secondfield: predict second field of a frame
 * PMV[2][2][2]: motion vectors (in half pel picture coordinates)
 * mv_field_sel[2][2]: motion vertical field selects (for field predictions)
 * dmvector: differential motion vectors (for dual prime)
 *
 * Notes:
 * - when predicting a P type picture which is the second field of
 *   a frame, the same parity reference field is in oldref, while the
```

```
 *   opposite parity reference field is assumed to be in newref!
 * - intra macroblocks are modelled to have a constant prediction of 128
 *   for all pels; this results in a DC DCT coefficient symmetric to 0
 * - vectors for field prediction in frame pictures are in half pel frame
 *   coordinates (vertical component is twice the field value and always
 *   even)
 *
 * already covers dual prime (not yet used)
 */

static void predict_mb(unsigned char *oldref[], unsigned char *newref[],
                       unsigned char *cur[],
                       int lx, int bx, int by,
                       int pict_type, int pict_struct,
                       int mb_type, int motion_type,
                       int secondfield,
                       int PMV[2][2][2],
                       int mv_field_sel[2][2] ,
                       int dmvector[2] )
{
  int addflag, currentfield;
  unsigned char **predframe;
  int DMV[2][2];

  if (mb_type&MB_INTRA)
  {
    clearblock(cur,bx,by);
    return;
  }

  addflag = 0; /* first prediction is stored, second is added and averaged */

  if ((mb_type & MB_FORWARD) || (pict_type==P_TYPE))
  {
    /* forward prediction, including zero MV in P pictures */

    if (pict_struct==FRAME_PICTURE)
    {
      /* frame picture */

      if ((motion_type==MC_FRAME) || !(mb_type & MB_FORWARD))
      {
        /* frame-based prediction in frame picture */
        pred(oldref,0,cur,0,
          lx,16,16,bx,by,PMV[0][0][0],PMV[0][0][1],0);
      }
      else if (motion_type==MC_FIELD)
      {
        /* field-based prediction in frame picture
         *
         * note scaling of the vertical coordinates (by, PMV[][0][1])
         * from frame to field!
         */
```

```
    /* top field prediction */
    pred(oldref,mv_field_sel[0][0],cur,0,
      lx<<1,16,8,bx,by>>1,PMV[0][0][0],PMV[0][0][1]>>1,0);

    /* bottom field prediction */
    pred(oldref,mv_field_sel[1][0],cur,1,
      lx<<1,16,8,bx,by>>1,PMV[1][0][0],PMV[1][0][1]>>1,0),
  }
  else if (motion_type==MC_DMV)
  {
    /* dual prime prediction */

    /* calculate derived motion vectors */
    calc_DMV(DMV,dmvector,PMV[0][0][0],PMV[0][0][1]>>1);

    /* predict top field from top field */
    pred(oldref,0,cur,0,
      lx<<1,16,8,bx,by>>1,PMV[0][0][0],PMV[0][0][1]>>1,0),

    /* predict bottom field from bottom field */
    pred(oldref,1,cur,1,
      lx<<1,16,8,bx,by>>1,PMV[0][0][0],PMV[0][0][1]>>1,0);

    /* predict and add to top field from bottom field */
    pred(oldref,1,cur,0,
      lx<<1,16,8,bx,by>>1,DMV[0][0],DMV[0][1],1);

    /* predict and add to bottom field from top field */
    pred(oldref,0,cur,1,
      lx<<1,16,8,bx,by>>1,DMV[1][0],DMV[1][1],1);
  }
  else
  {
    /* invalid motion_type in frame picture */
    if (!quiet)
      fprintf(stderr,"invalid motion_type\n"),
  }
}
else /* TOP_FIELD or BOTTOM_FIELD */
{
  /* field picture */

  currentfield = (pict_struct==BOTTOM_FIELD);

  /* determine which frame to use for prediction */
  if ((pict_type==P_TYPE) && secondfield
     && (currentfield!=mv_field_sel[0][0]))
    predframe = newref; /* same frame */
  else
    predframe = oldref; /* previous frame */

  if ((motion_type==MC_FIELD) || !(mb_type & MB_FORWARD))
  {
    /* field-based prediction in field picture */
```

```
      pred(predframe,mv_field_sel[0][0],cur,currentfield,
        lx<<1,16,16,bx,by,PMV[0][0][0],PMV[0][0][1],0);
    }
    else if (motion_type==MC_16X8)
    {
      /* 16 x 8 motion compensation in field picture */

      /* upper half */
      pred(predframe,mv_field_sel[0][0],cur,currentfield,
        lx<<1,16,8,bx,by,PMV[0][0][0],PMV[0][0][1],0);

      /* determine which frame to use for lower half prediction */
      if ((pict_type==P_TYPE) && secondfield
          && (currentfield!=mv_field_sel[1][0]))
        predframe = newref; /* same frame */
      else
        predframe = oldref; /* previous frame */

      /* lower half */
      pred(predframe,mv_field_sel[1][0],cur,currentfield,
        lx<<1,16,8,bx,by+8,PMV[1][0][0],PMV[1][0][1],0);
    }
    else if (motion_type==MC_DMV)
    {
      /* dual prime prediction */

      /* determine which frame to use for prediction */
      if (secondfield)
        predframe = newref; /* same frame */
      else
        predframe = oldref; /* previous frame */

      /* calculate derived motion vectors */
      calc_DMV(DMV,dmvector,PMV[0][0][0],PMV[0][0][1]);

      /* predict from field of same parity */
      pred(oldref,currentfield,cur,currentfield,
        lx<<1,16,16,bx,by,PMV[0][0][0],PMV[0][0][1],0);

      /* predict from field of opposite parity */
      pred(predframe,!currentfield,cur,currentfield,
        lx<<1,16,16,bx,by,DMV[0][0],DMV[0][1],1);
    }
    else
    {
      /* invalid motion_type in field picture */
      if (!quiet)
        fprintf(stderr,"invalid motion_type\n");
    }
  }
  addflag = 1; /* next prediction (if any) will be averaged with this one */
}

if (mb_type & MB_BACKWARD)
```

```
{
 /* backward prediction */

 if (pict_struct==FRAME_PICTURE)
 {
  /* frame picture */

  if (motion_type==MC_FRAME)
  {
   /* frame-based prediction in frame picture */
   pred(newref,0,cur,0,
    lx,16,16,bx,by,PMV[0][1][0],PMV[0][1][1],addflag),
  }
  else
  {
   /* field-based prediction in frame picture
    *
    * note scaling of the vertical coordinates (by, PMV[][1][1])
    * from frame to field!
    */

   /* top field prediction */
   pred(newref,mv_field_sel[0][1],cur,0,
    lx<<1,16,8,bx,by>>1,PMV[0][1][0],PMV[0][1][1]>>1,addflag);

   /* bottom field prediction */
   pred(newref,mv_field_sel[1][1],cur,1,
    lx<<1,16,8,bx,by>>1,PMV[1][1][0],PMV[1][1][1]>>1,addflag);
  }
 }
 else /* TOP_FIELD or BOTTOM_FIELD */
 {
  /* field picture */

  currentfield = (pict_struct==BOTTOM_FIELD);

  if (motion_type==MC_FIELD)
  {
   /* field-based prediction in field picture */
   pred(newref,mv_field_sel[0][1],cur,currentfield,
    lx<<1,16,16,bx,by,PMV[0][1][0],PMV[0][1][1],addflag);
  }
  else if (motion_type==MC_16X8)
  {
   /* 16 x 8 motion compensation in field picture */

   /* upper half */
   pred(newref,mv_field_sel[0][1],cur,currentfield,
    lx<<1,16,8,bx,by,PMV[0][1][0],PMV[0][1][1],addflag);

   /* lower half */
   pred(newref,mv_field_sel[1][1],cur,currentfield,
    lx<<1,16,8,bx,by+8,PMV[1][1][0],PMV[1][1][1],addflag);
  }
```

```
      else
      {
        /* invalid motion_type in field picture */
        if (!quiet)
          fprintf(stderr,"invalid motion_type\n");
      }
    }
  }
}

/* predict a rectangular block (all three components)
 *
 * src:     source frame (Y,U,V)
 * sfield·  source field select (0: frame or top field, 1: bottom field)
 * dst:     destination frame (Y,U,V)
 * dfield:  destination field select (0: frame or top field, 1: bottom field)
 *
 * the following values are in luminance picture (frame or field) dimensions
 * lx:      distance of vertically adjacent pels (selects frame or field pred.)
 * w,h:     width and height of block (only 16x16 or 16x8 are used)
 * x,y:     coordinates of destination block
 * dx,dy:   half pel motion vector
 * addflag: store or add (= average) prediction
 */
static void pred(unsigned char *src[], int sfield,
                 unsigned char *dst[], int dfield,
                 int lx, int w, int h,
                 int x, int y, int dx, int dy, int addflag )
{
  int cc;

  for (cc=0; cc<3; cc++)
  {
    if (cc==1)
    {
      /* scale for color components */
      if (chroma_format==CHROMA420)
      {
        /* vertical */
        h >>= 1; y >>= 1; dy /= 2;
      }
      if (chroma_format!=CHROMA444)
      {
        /* horizontal */
        w >>= 1; x >>= 1; dx /= 2;
        lx >>= 1;
      }
    }
    pred_comp(src[cc]+(sfield?lx>>1:0),dst[cc]+(dfield?lx>>1:0),
      lx,w,h,x,y,dx,dy,addflag);
  }
}

/* low level prediction routine
```

```
 *
 * src      prediction source
 * dst:     prediction destination
 * lx:      line width (for both src and dst)
 * x,y:     destination coordinates
 * dx,dy:   half pel motion vector
 * w,h:     size of prediction block
 * addflag: store or add prediction
 */

static void pred_comp(unsigned char *src, unsigned char *dst,
                      int lx, int w, int h,
                      int x, int y,
                      int dx, int dy, int addflag )
{
  int xint, xh, yint, yh;
  int i, j;
  unsigned char *s, *d;

  /* half pel scaling */
  xint = dx>>1; /* integer part */
  xh = dx & 1;  /* half pel flag */
  yint = dy>>1;
  yh = dy & 1;

  /* origins */
  s = src + lx*(y+yint) + (x+xint); /* motion vector */
  d = dst + lx*y + x;

  if (!xh && !yh)
    if (addflag)
      for (j=0; j<h; j++)
      {
        for (i=0; i<w; i++)
          d[i] = (unsigned int)(d[i]+s[i]+1)>>1;
        s+= lx;
        d+= lx;
      }
    else
      for (j=0; j<h; j++)
      {
        for (i=0; i<w; i++)
          d[i] = s[i];
        s+= lx;
        d+= lx;
      }
  else if (!xh && yh)
    if (addflag)
      for (j=0; j<h; j++)
      {
        for (i=0; i<w, i++)
          d[i] = (d[i] + ((unsigned int)(s[i]+s[i+lx]+1)>>1)+1)>>1;
        s+= lx;
        d+= lx;
```

```
      }
    else
      for (j=0; j<h; j++)
      {
        for (i=0, i<w; i++)
          d[i] = (unsigned int)(s[i]+s[i+lx]+1)>>1;
        s+= lx;
        d+= lx;
      }
  else if (xh && !yh)
    if (addflag)
      for (j=0; j<h; j++)
      {
        for (i=0; i<w, i++)
          d[i] = (d[i] + ((unsigned int)(s[i]+s[i+1]+1)>>1)+1)>>1;
        s+= lx;
        d+= lx;
      }
    else
      for (j=0; j<h; j++)
      {
        for (i=0; i<w; i++)
          d[i] = (unsigned int)(s[i]+s[i+1]+1)>>1;
        s+= lx;
        d+= lx;
      }
  else /* if (xh && yh) */
    if (addflag)
      for (j=0; j<h; j++)
      {
        for (i=0; i<w; i++)
          d[i] = (d[i] + ((unsigned int)(s[i]+s[i+1]+s[i+lx]+s[i+lx+1]+2)>>2)+1)>>1;
        s+= lx;
        d+= lx;
      }
    else
      for (j=0; j<h; j++)
      {
        for (i=0; i<w; i++)
          d[i] = (unsigned int)(s[i]+s[i+1]+s[i+lx]+s[i+lx+1]+2)>>2;
        s+= lx;
        d+= lx;
      }
}

/* calculate derived motion vectors (DMV) for dual prime prediction
 * dmvector[2]: differential motion vectors (-1,0,+1)
 * mvx,mvy: motion vector (for same parity)
 *
 * DMV[2][2]: derived motion vectors (for opposite parity)
 *
 * uses global variables pict_struct and topfirst
 *
```

```
 * Notes:
 *  - all vectors are in field coordinates (even for frame pictures)
 */

static void calc_DMV(int DMV[][2], int *dmvector, int mvx, int mvy )
{
  if (pict_struct==FRAME_PICTURE)
  {
    if (topfirst)
    {
      /* vector for prediction of top field from bottom field */
      DMV[0][0] = ((mvx  +(mvx>0))>>1) + dmvector[0];
      DMV[0][1] = ((mvy  +(mvy>0))>>1) + dmvector[1] - 1;

      /* vector for prediction of bottom field from top field */
      DMV[1][0] = ((3*mvx+(mvx>0))>>1) + dmvector[0];
      DMV[1][1] = ((3*mvy+(mvy>0))>>1) + dmvector[1] + 1,
    }
    else
    {
      /* vector for prediction of top field from bottom field */
      DMV[0][0] = ((3*mvx+(mvx>0))>>1) + dmvector[0];
      DMV[0][1] = ((3*mvy+(mvy>0))>>1) + dmvector[1] - 1;

      /* vector for prediction of bottom field from top field */
      DMV[1][0] = ((mvx  +(mvx>0))>>1) + dmvector[0];
      DMV[1][1] = ((mvy  +(mvy>0))>>1) + dmvector[1] + 1;
    }
  }
  else
  {
    /* vector for prediction from field of opposite 'parity' */
    DMV[0][0] = ((mvx+(mvx>0))>>1) + dmvector[0];
    DMV[0][1] = ((mvy+(mvy>0))>>1) + dmvector[1];

    /* correct for vertical field shift */
    if (pict_struct==TOP_FIELD)
      DMV[0][1]--;
    else
      DMV[0][1]++;
  }
}

static void clearblock(unsigned char *cur[], int i0, int j0 )
{
  int i, j, w, h;
  unsigned char *p;

  p = cur[0] + ((pict_struct==BOTTOM_FIELD) ? width : 0) + i0 + width2*j0,

  for (j=0; j<16; j++)
  {
    for (i=0; i<16; i++)
      p[i] = 128;
```

```
    p+= width2;
  }

  w = h = 16;

  if (chroma_format!=CHROMA444)
  {
    i0>>=1; w>>=1;
  }

  if (chroma_format==CHROMA420)
  {
    j0>>=1; h>>=1;
  }

  p = cur[1] + ((pict_struct==BOTTOM_FIELD) ? chrom_width : 0) + i0
          + chrom_width2*j0;

  for (j=0; j<h; j++)
  {
    for (i=0; i<w; i++)
      p[i] = 128;
    p+= chrom_width2;
  }

  p = cur[2] + ((pict_struct==BOTTOM_FIELD) ? chrom_width : 0) + i0
          + chrom_width2*j0;

  for (j=0; j<h; j++)
  {
    for (i=0; i<w; i++)
      p[i] = 128;
    p+= chrom_width2;
  }
}
```

```
/*********************************/
/*                               */
/*        putpic.cpp             */
/*                               */
/*********************************/

#include <stdio.h>
#include <math.h>
#include "c:\nick\mpeg\org\config.h"
#include "c:\nick\mpeg\org\global.h"
#include "c:\nick\mpeg\org\vlc.h"


extern FILE *outfile; /* the only global var we need here */

/* private data */
static unsigned char outbfr;
static int outcnt;
static int bytecnt;

/* private prototypes */
static void iquant1_intra _ANSI_ARGS_((short *src, short *dst, int dc_prec,
                        unsigned char *quant_mat, int mquant));
static void iquant1_non_intra _ANSI_ARGS_((short *src, short *dst,
                        unsigned char *quant_mat, int mquant));
static void calc_actj _ANSI_ARGS_((unsigned char *frame));
static double var_sblk _ANSI_ARGS_((unsigned char *p, int lx));

static void putDC _ANSI_ARGS_((sVLCtable *tab, int val));

static int frametotc _ANSI_ARGS_((int frame));

static void putmvs _ANSI_ARGS_((int MV[2][2][2],
                    int PMV[2][2][2],
                    int mv_field_sel[2][2],
                    int dmvector[2],
                    int s, int motion_type,
                    int hor_f_code, int vert_f_code));

/* quantization / variable length encoding of a complete picture */
void putpict(unsigned char *frame)
{
 int i, j, k, comp, cc;
 int mb_type;
 int PMV[2][2][2];
 int prev_mquant;
 int cbp, MBAinc;

 rc_init_pict(frame), /* set up rate control */

 /* picture header and picture coding extension */
 putpicthdr();

 if (!mpeg1) putpictcodext(),
```

```
prev_mquant = rc_start_mb(); /* initialize quantization parameter */

k = 0;

for (j=0; j<mb_height2; j++)
{
 /* macroblock row loop */

 for (i=0; i<mb_width; i++)
 {
  /* macroblock loop */
  if (i==0)
  {
   /* slice header (6.2.4) */
   alignbits();

   if (mpeg1 || vertical_size<=2800)
    putbits(SLICE_MIN_START+j,32); /* slice_start_code */
   else
   {
    putbits(SLICE_MIN_START+(j&127),32); /* slice_start_code */
    putbits(j>>7,3); /* slice_vertical_position_extension */
   }

   /* quantiser_scale_code */
   putbits(q_scale_type ? map_non_linear_mquant[prev_mquant]
                  : prev_mquant >> 1, 5);

   putbits(0,1); /* extra_bit_slice */

   /* reset predictors */

   for (cc=0; cc<3; cc++)
    dc_dct_pred[cc] = 0;

   PMV[0][0][0]=PMV[0][0][1]=PMV[1][0][0]=PMV[1][0][1]=0;
   PMV[0][1][0]=PMV[0][1][1]=PMV[1][1][0]=PMV[1][1][1]=0;

   MBAinc = i + 1; /* first MBAinc denotes absolute position */
  }

  mb_type = mbinfo[k].mb_type;

  /* determine mquant (rate control) */
  mbinfo[k].mquant = rc_calc_mquant(k);

  /* quantize macroblock */
  if (mb_type & MB_INTRA)
  {
   for (comp=0; comp<block_count, comp++)
    quant_intra(blocks[k*block_count+comp],blocks[k*block_count+comp],
            dc_prec,intra_q,mbinfo[k].mquant);
   mbinfo[k].cbp = cbp = (1<<block_count) - 1;
```

```
}
else
{
  cbp = 0;
  for (comp=0;comp<block_count;comp++)
    cbp = (cbp<<1) | quant_non_intra(blocks[k*block_count+comp],
                                    blocks[k*block_count+comp],
                                    inter_q,mbinfo[k].mquant);

  mbinfo[k].cbp = cbp,

  if (cbp)
    mb_type|= MB_PATTERN;
}

/* output mquant if it has changed */
if (cbp && prev_mquant!=mbinfo[k].mquant)
  mb_type|= MB_QUANT;

/* check if macroblock can be skipped */
if (i!=0 && i!=mb_width-1 && !cbp)
{
  /* no DCT coefficients and neither first nor last macroblock of slice */

  if (pict_type==P_TYPE && !(mb_type&MB_FORWARD))
  {
    /* P picture, no motion vectors -> skip */

    /* reset predictors */

    for (cc=0; cc<3; cc++)
      dc_dct_pred[cc] = 0;

    PMV[0][0][0]=PMV[0][0][1]=PMV[1][0][0]=PMV[1][0][1]=0;
    PMV[0][1][0]=PMV[0][1][1]=PMV[1][1][0]=PMV[1][1][1]=0;

    mbinfo[k].mb_type = mb_type;
    mbinfo[k].skipped = 1;
    MBAinc++;
    k++;
    continue;
  }

  if (pict_type==B_TYPE && pict_struct==FRAME_PICTURE
      && mbinfo[k].motion_type==MC_FRAME
      && ((mbinfo[k-1].mb_type^mb_type)&(MB_FORWARD|MB_BACKWARD))==0
      && (!(mb_type&MB_FORWARD) ||
         (PMV[0][0][0]==mbinfo[k].MV[0][0][0] &&
          PMV[0][0][1]==mbinfo[k].MV[0][0][1]))
      && (!(mb_type&MB_BACKWARD) ||
         (PMV[0][1][0]==mbinfo[k] MV[0][1][0] &&
          PMV[0][1][1]==mbinfo[k].MV[0][1][1])))
  {
    /* conditions for skipping in B frame pictures:
```

```
    * - must be frame predicted
    * - must be the same prediction type (forward/backward/interp )
    *   as previous macroblock
    * - relevant vectors (forward/backward/both) have to be the same
    *   as in previous macroblock
    */

   mbinfo[k].mb_type = mb_type;
   mbinfo[k].skipped = 1;
   MBAinc++;
   k++;
   continue;
  }

  if (pict_type==B_TYPE && pict_struct!=FRAME_PICTURE
     && mbinfo[k].motion_type==MC_FIELD
     && ((mbinfo[k-1].mb_type^mb_type)&(MB_FORWARD|MB_BACKWARD))==0
     && (!(mb_type&MB_FORWARD) ||
        (PMV[0][0][0]==mbinfo[k].MV[0][0][0] &&
         PMV[0][0][1]==mbinfo[k].MV[0][0][1] &&
         mbinfo[k].mv_field_sel[0][0]==(pict_struct==BOTTOM_FIELD)))
     && (!(mb_type&MB_BACKWARD) ||
        (PMV[0][1][0]==mbinfo[k].MV[0][1][0] &&
         PMV[0][1][1]==mbinfo[k].MV[0][1][1] &&
         mbinfo[k].mv_field_sel[0][1]==(pict_struct==BOTTOM_FIELD))))
  {
   /* conditions for skipping in B field pictures:
    * - must be field predicted
    * - must be the same prediction type (forward/backward/interp.)
    *   as previous macroblock
    * - relevant vectors (forward/backward/both) have to be the same
    *   as in previous macroblock
    * - relevant motion_vertical_field_selects have to be of same
    *   parity as current field
    */

   mbinfo[k].mb_type = mb_type;
   mbinfo[k].skipped = 1;
   MBAinc++;
   k++;
   continue;
  }
 }

/* macroblock cannot be skipped */
mbinfo[k].skipped = 0;

/* there's no VLC for 'No MC, Not Coded':
 * we have to transmit (0,0) motion vectors
 */
if (pict_type==P_TYPE && !cbp && !(mb_type&MB_FORWARD))
  mb_type|= MB_FORWARD,

putaddrinc(MBAinc); /* macroblock_address_increment */
```

```
MBAinc = 1;

putmbtype(pict_type,mb_type); /* macroblock type */

if (mb_type & (MB_FORWARD|MB_BACKWARD) && !frame_pred_dct)
  putbits(mbinfo[k].motion_type,2);

if (pict_struct==FRAME_PICTURE && cbp && !frame_pred_dct)
  putbits(mbinfo[k].dct_type,1);

if (mb_type & MB_QUANT)
{
  putbits(q_scale_type ? map_non_linear_mquant[mbinfo[k].mquant]
                 : mbinfo[k].mquant>>1,5);
  prev_mquant = mbinfo[k].mquant;
}

if (mb_type & MB_FORWARD)
{
  /* forward motion vectors, update predictors */
  putmvs(mbinfo[k].MV,PMV,mbinfo[k].mv_field_sel,mbinfo[k].dmvector,0,
    mbinfo[k].motion_type,forw_hor_f_code,forw_vert_f_code);
}

if (mb_type & MB_BACKWARD)
{
  /* backward motion vectors, update predictors */
  putmvs(mbinfo[k].MV,PMV,mbinfo[k].mv_field_sel,mbinfo[k].dmvector,1,
    mbinfo[k].motion_type,back_hor_f_code,back_vert_f_code);
}

if (mb_type & MB_PATTERN)
{
  putcbp((cbp >> (block_count-6)) & 63);
  if (chroma_format!=CHROMA420)
    putbits(cbp,block_count-6);
}

for (comp=0; comp<block_count; comp++)
{
  /* block loop */
  if (cbp & (1<<(block_count-1-comp)))
  {
    if (mb_type & MB_INTRA)
    {
      cc = (comp<4) ? 0 : (comp&1)+1;
      putintrablk(blocks[k*block_count+comp],cc),
    }
    else
      putnonintrablk(blocks[k*block_count+comp]),
  }
}

/* reset predictors */
```

```
    if (!(mb_type & MB_INTRA))
      for (cc=0; cc<3; cc++)
        dc_dct_pred[cc] = 0;

    if (mb_type & MB_INTRA || (pict_type==P_TYPE && !(mb_type & MB_FORWARD)))
    {
      PMV[0][0][0]=PMV[0][0][1]=PMV[1][0][0]=PMV[1][0][1]=0;
      PMV[0][1][0]=PMV[0][1][1]=PMV[1][1][0]=PMV[1][1][1]=0,
    }

    mbinfo[k].mb_type = mb_type;
    k++;
   }
  }

  rc_update_pict();
  vbv_end_of_picture();
}


/* output motion vectors (6.2.5.2, 6.3.16.2)
 *
 * this routine also updates the predictions for motion vectors (PMV)
 */

static void putmvs(int MV[2][2][2],
                   int PMV[2][2][2],
                   int mv_field_sel[2][2],
                   int dmvector[2],
                   int s, int motion_type,
                   int hor_f_code, int vert_f_code )
{
  if (pict_struct==FRAME_PICTURE)
  {
    if (motion_type==MC_FRAME)
    {
      /* frame prediction */
      putmv(MV[0][s][0]-PMV[0][s][0],hor_f_code);
      putmv(MV[0][s][1]-PMV[0][s][1],vert_f_code);
      PMV[0][s][0]=PMV[1][s][0]=MV[0][s][0];
      PMV[0][s][1]=PMV[1][s][1]=MV[0][s][1];
    }
    else if (motion_type==MC_FIELD)
    {
      /* field prediction */
      putbits(mv_field_sel[0][s],1);
      putmv(MV[0][s][0]-PMV[0][s][0],hor_f_code);
      putmv((MV[0][s][1]>>1)-(PMV[0][s][1]>>1),vert_f_code);
      putbits(mv_field_sel[1][s],1);
      putmv(MV[1][s][0]-PMV[1][s][0],hor_f_code);
      putmv((MV[1][s][1]>>1)-(PMV[1][s][1]>>1),vert_f_code);
      PMV[0][s][0]=MV[0][s][0];
      PMV[0][s][1]=MV[0][s][1];
      PMV[1][s][0]=MV[1][s][0];
```

```
      PMV[1][s][1]=MV[1][s][1],
    }
    else
    {
      /* dual prime prediction */
      putmv(MV[0][s][0]-PMV[0][s][0],hor_f_code);
      putdmv(dmvector[0]);
      putmv((MV[0][s][1]>>1)-(PMV[0][s][1]>>1),vert_f_code),
      putdmv(dmvector[1]);
      PMV[0][s][0]=PMV[1][s][0]=MV[0][s][0];
      PMV[0][s][1]=PMV[1][s][1]=MV[0][s][1];
    }
  }
  else
  {
    /* field picture */
    if (motion_type==MC_FIELD)
    {
      /* field prediction */
      putbits(mv_field_sel[0][s],1);
      putmv(MV[0][s][0]-PMV[0][s][0],hor_f_code),
      putmv(MV[0][s][1]-PMV[0][s][1],vert_f_code);
      PMV[0][s][0]=PMV[1][s][0]=MV[0][s][0];
      PMV[0][s][1]=PMV[1][s][1]=MV[0][s][1];
    }
    else if (motion_type==MC_16X8)
    {
      /* 16x8 prediction */
      putbits(mv_field_sel[0][s],1);
      putmv(MV[0][s][0]-PMV[0][s][0],hor_f_code);
      putmv(MV[0][s][1]-PMV[0][s][1],vert_f_code);
      putbits(mv_field_sel[1][s],1);
      putmv(MV[1][s][0]-PMV[1][s][0],hor_f_code);
      putmv(MV[1][s][1]-PMV[1][s][1],vert_f_code),
      PMV[0][s][0]=MV[0][s][0];
      PMV[0][s][1]=MV[0][s][1],
      PMV[1][s][0]=MV[1][s][0];
      PMV[1][s][1]=MV[1][s][1];
    }
    else
    {
      /* dual prime prediction */
      putmv(MV[0][s][0]-PMV[0][s][0],hor_f_code),
      putdmv(dmvector[0]);
      putmv(MV[0][s][1]-PMV[0][s][1],vert_f_code);
      putdmv(dmvector[1]);
      PMV[0][s][0]=PMV[1][s][0]=MV[0][s][0],
      PMV[0][s][1]=PMV[1][s][1]=MV[0][s][1];
    }
  }
}

/* generate variable length codes for an intra-coded block (6.2.6, 6.3.17) */
void putintrablk(short *blk, int cc)
```

```
{
  int n, dct_diff, run, signed_level;

  /* DC coefficient (7.2.1) */
  dct_diff = blk[0] - dc_dct_pred[cc]; /* difference to previous block */
  dc_dct_pred[cc] = blk[0],

  if (cc==0)
    putDClum(dct_diff);
  else
    putDCchrom(dct_diff);

  /* AC coefficients (7.2.2) */
  run = 0;
  for (n=1; n<64; n++)
  {
    /* use appropriate entropy scanning pattern */
    signed_level = blk[(altscan ? alternate_scan : zig_zag_scan)[n]];
    if (signed_level!=0)
    {
      putAC(run,signed_level,intravlc);
      run = 0;
    }
    else
      run++; /* count zero coefficients */
  }

  /* End of Block -- normative block punctuation */
  if (intravlc)
    putbits(6,4); /* 0110 (Table B-15) */
  else
    putbits(2,2); /* 10 (Table B-14) */
}

/* generate variable length codes for a non-intra-coded block (6.2 6, 6.3.17) */
void putnonintrablk(short *blk)
{
  int n, run, signed_level, first;

  run = 0;
  first = 1;

  for (n=0; n<64; n++)
  {
    /* use appropriate entropy scanning pattern */
    signed_level = blk[(altscan ? alternate_scan : zig_zag_scan)[n]];

    if (signed_level!=0)
    {
      if (first)
      {
        /* first coefficient in non-intra block */
        putACfirst(run,signed_level);
        first = 0;
```

```
      }
      else
        putAC(run,signed_level,0);

      run = 0,
    }
    else
      run++; /* count zero coefficients */
  }

  /* End of Block -- normative block punctuation  */
  putbits(2,2);
}

/* generate variable length code for a motion vector component (7.6.3.1) */
void putmv(int dmv, int f_code)
{
  int r_size, f, vmin, vmax, dv, temp, motion_code, motion_residual,

  r_size = f_code - 1; /* number of fixed length code ('residual') bits */
  f = 1<<r_size;
  vmin = -16*f; /* lower range limit */
  vmax = 16*f - 1; /* upper range limit */
  dv = 32*f;

  /* fold vector difference into [vmin...vmax] */
  if (dmv>vmax)
    dmv-= dv;
  else if (dmv<vmin)
    dmv+= dv;

  /* check value */
  if (dmv<vmin || dmv>vmax)
    if (!quiet)
      fprintf(stderr,"invalid motion vector\n");

  /* split dmv into motion_code and motion_residual */
  temp = ((dmv<0) ? -dmv : dmv) + f - 1;
  motion_code = temp>>r_size;
  if (dmv<0)
    motion_code = -motion_code;
  motion_residual = temp & (f-1);

  putmotioncode(motion_code); /* variable length code */

  if (r_size!=0 && motion_code!=0)
    putbits(motion_residual,r_size), /* fixed length code */
}


/* generate sequence header (6.2.2.1, 6.3.3)
 *
 * matrix download not implemented
 */
```

```
void putseqhdr()
{
  int i;

  alignbits(),
  putbits(SEQ_START_CODE,32); /* sequence_header_code */
  putbits(horizontal_size,12); /* horizontal_size_value */
  putbits(vertical_size,12); /* vertical_size_value */
  putbits(aspectratio,4); /* aspect_ratio_information */
  putbits(frame_rate_code,4); /* frame_rate_code */
  putbits((int)ceil(bit_rate/400.0),18); /* bit_rate_value */
  putbits(1,1); /* marker_bit */
  putbits(vbv_buffer_size,10); /* vbv_buffer_size_value */
  putbits(constrparms,1); /* constrained_parameters_flag */

  putbits(load_iquant,1); /* load_intra_quantizer_matrix */
  if (load_iquant)
    for (i=0, i<64; i++)  /* matrices are always downloaded in zig-zag order */
      putbits(intra_q[zig_zag_scan[i]],8); /* intra_quantizer_matrix */

  putbits(load_niquant,1); /* load_non_intra_quantizer_matrix */
  if (load_niquant)
    for (i=0; i<64; i++)
      putbits(inter_q[zig_zag_scan[i]],8); /* non_intra_quantizer_matrix */
}

/* generate sequence extension (6.2.2.3, 6.3.5) header (MPEG-2 only) */
void putseqext()
{
  alignbits();
  putbits(EXT_START_CODE,32), /* extension_start_code */
  putbits(SEQ_ID,4); /* extension_start_code_identifier */
  putbits((profile<<4)|level,8); /* profile_and_level_indication */
  putbits(prog_seq,1); /* progressive sequence */
  putbits(chroma_format,2); /* chroma_format */
  putbits(horizontal_size>>12,2); /* horizontal_size_extension */
  putbits(vertical_size>>12,2); /* vertical_size_extension */
  putbits(((int)ceil(bit_rate/400.0))>>18,12); /* bit_rate_extension */
  putbits(1,1); /* marker_bit */
  putbits(vbv_buffer_size>>10,8); /* vbv_buffer_size_extension */
  putbits(0,1); /* low_delay  -- currently not implemented */
  putbits(0,2); /* frame_rate_extension_n */
  putbits(0,5); /* frame_rate_extension_d */
}

/* generate sequence display extension (6.2.2.4, 6.3.6)
 *
 * content not yet user setable
 */
void putseqdispext()
{
  alignbits();
  putbits(EXT_START_CODE,32); /* extension_start_code */
  putbits(DISP_ID,4); /* extension_start_code_identifier */
```

```
  putbits(video_format,3); /* video_format */
  putbits(1,1); /* colour_description */
  putbits(color_primaries,8); /* colour_primaries */
  putbits(transfer_characteristics,8); /* transfer_characteristics */
  putbits(matrix_coefficients,8); /* matrix_coefficients */
  putbits(display_horizontal_size,14); /* display_horizontal_size */
  putbits(1,1); /* marker_bit */
  putbits(display_vertical_size,14); /* display_vertical_size */
}

/* output a zero terminated string as user data (6.2.2.2.2, 6.3.4.1)
 *
 * string must not emulate start codes
 */
void putuserdata(char *userdata)
{
  alignbits();
  putbits(USER_START_CODE,32); /* user_data_start_code */
  while (*userdata)
    putbits(*userdata++,8),
}

/* generate group of pictures header (6.2 2.6, 6.3.9)
 *
 * uses tc0 (timecode of first frame) and frame0 (number of first frame)
 */
void putgophdr(int frame, int closed_gop)
{
  int tc;

  alignbits();
  putbits(GOP_START_CODE,32); /* group_start_code */
  tc = frametotc(tc0+frame);
  putbits(tc,25); /* time_code */
  putbits(closed_gop,1); /* closed_gop */
  putbits(0,1); /* broken_link */
}

/* convert frame number to time_code
 *
 * drop_frame not implemented
 */
static int frametotc(int frame)
{
  int fps, pict, sec, minute, hour, tc;

  fps = (int)(frame_rate+0.5);
  pict = frame%fps;
  frame = (frame-pict)/fps;
  sec = frame%60;
  frame = (frame-sec)/60;
  minute = frame%60;
  frame = (frame-minute)/60;
  hour = frame%24;
```

```
 tc = (hour<<19) | (minute<<13) | (1<<12) | (sec<<6) | pict;

 return tc;
}

/* generate picture header (6.2.3, 6.3.10) */
void putpicthdr()
{
 alignbits();
 putbits(PICTURE_START_CODE,32); /* picture_start_code */
 calc_vbv_delay();
 putbits(temp_ref,10); /* temporal_reference */
 putbits(pict_type,3); /* picture_coding_type */
 putbits(vbv_delay,16); /* vbv_delay */

 if (pict_type==P_TYPE || pict_type==B_TYPE)
 {
  putbits(0,1); /* full_pel_forward_vector */
  if (mpeg1)
   putbits(forw_hor_f_code,3);
  else
   putbits(7,3); /* forward_f_code */
 }

 if (pict_type==B_TYPE)
 {
  putbits(0,1); /* full_pel_backward_vector */
  if (mpeg1)
   putbits(back_hor_f_code,3);
  else
   putbits(7,3); /* backward_f_code */
 }

 putbits(0,1); /* extra_bit_picture */
}

/* generate picture coding extension (6.2.3.1, 6.3.11)
 *
 * composite display information (v_axis etc.) not implemented
 */
void putpictcodext()
{
 alignbits();
 putbits(EXT_START_CODE,32); /* extension_start_code */
 putbits(CODING_ID,4); /* extension_start_code_identifier */
 putbits(forw_hor_f_code,4); /* forward_horizontal_f_code */
 putbits(forw_vert_f_code,4); /* forward_vertical_f_code */
 putbits(back_hor_f_code,4); /* backward_horizontal_f_code */
 putbits(back_vert_f_code,4); /* backward_vertical_f_code */
 putbits(dc_prec,2); /* intra_dc_precision */
 putbits(pict_struct,2); /* picture_structure */
 putbits((pict_struct==FRAME_PICTURE)?topfirst:0,1); /* top_field_first */
 putbits(frame_pred_dct,1); /* frame_pred_frame_dct */
 putbits(0,1); /* concealment_motion_vectors -- currently not implemented */
```

```
  putbits(q_scale_type,1); /* q_scale_type */
  putbits(intravlc,1), /* intra_vlc_format */
  putbits(altscan,1); /* alternate_scan */
  putbits(repeatfirst,1), /* repeat_first_field */
  putbits(prog_frame,1); /* chroma_420_type */
  putbits(prog_frame,1); /* progressive_frame */
  putbits(0,1); /* composite_display_flag */
}

/* generate sequence_end_code (6.2.2) */
void putseqend()
{
  alignbits();
  putbits(SEQ_END_CODE,32),
}

/* initialize buffer, call once before first putbits or alignbits */
void initbits()
{
  outcnt = 8;
  bytecnt = 0;
}

/* write rightmost n (0<=n<=32) bits of val to outfile */
void putbits(int val, int n)
{
  int i;
  unsigned int mask;

  mask = 1 << (n-1); /* selects first (leftmost) bit */

  for (i=0; i<n; i++)
  {
    outbfr <<= 1;

    if (val & mask)
      outbfr|= 1;

    mask >>= 1; /* select next bit */
    outcnt--;

    if (outcnt==0) /* 8 bit buffer full */
    {
      putc(outbfr,outfile);
      outcnt = 8;
      bytecnt++;
    }
  }
}

/* zero bit stuffing to next byte boundary (5.2.3, 6.2 1) */
void alignbits()
{
  if (outcnt!=8)
```

```
    putbits(0,outcnt);
  }

/* return total number of generated bits */
int bitcount()
{
  return 8*bytecnt + (8-outcnt);
}

/* generate variable length code for luminance DC coefficient */
void putDClum(int val)
{
  putDC(DClumtab,val);
}

/* generate variable length code for chrominance DC coefficient */
void putDCchrom(int val)
{
  putDC(DCchromtab,val);
}

/* generate variable length code for DC coefficient (7.2.1) */
static void putDC(sVLCtable *tab, int val )
{
  int absval, size;

  absval = (val<0) ? -val : val; /* abs(val) */

  if (absval>2047 || (mpeg1 && absval>255))
  {
    /* should never happen */
    sprintf(errortext,"DC value out of range (%d)\n",val);
    error(errortext);
  }

  /* compute dct_dc_size */
  size = 0;

  while (absval)
  {
    absval >>= 1;
    size++;
  }

  /* generate VLC for dct_dc_size (Table B-12 or B-13) */
  putbits(tab[size].code,tab[size].len);

  /* append fixed length code (dc_dct_differential) */
  if (size!=0)
  {
    if (val>=0)
      absval = val;
    else
      absval = val + (1<<size) - 1; /* val + (2 ^ size) - 1 */
```

```
    putbits(absval,size);
  }
}

/* generate variable length code for first coefficient
 * of a non-intra block (7.2.2.2) */
void putACfirst(int run, int val)
{
  if (run==0 && (val==1 || val==-1)) /* these are treated differently */
    putbits(2|(val<0),2); /* generate '1s' (s=sign), (Table B-14, line 2) */
  else
    putAC(run,val,0); /* no difference for all others */
}

/* generate variable length code for other DCT coefficients (7.2.2) */
void putAC(int run, int signed_level, int vlcformat )
{
  int level, len;
  VLCtable *ptab;

  level = (signed_level<0) ? -signed_level : signed_level; /* abs(signed_level) */

  /* make sure run and level are valid */
  if (run<0 || run>63 || level==0 || level>2047 || (mpeg1 && level>255))
  {
    sprintf(errortext,"AC value out of range (run=%d, signed_level=%d)\n",
      run,signed_level);
    error(errortext);
  }

  len = 0;

  if (run<2 && level<41)
  {
    /* vlcformat selects either of Table B-14 / B-15 */
    if (vlcformat)
      ptab = &dct_code_tab1a[run][level-1];
    else
      ptab = &dct_code_tab1[run][level-1];

    len = ptab->len;
  }
  else if (run<32 && level<6)
  {
    /* vlcformat selects either of Table B-14 / B-15 */
    if (vlcformat)
      ptab = &dct_code_tab2a[run-2][level-1];
    else
      ptab = &dct_code_tab2[run-2][level-1];

    len = ptab->len;
  }

  if (len!=0) /* a VLC code exists */
```

```
  {
   putbits(ptab->code,len);
   putbits(signed_level<0,1); /* sign */
  }
  else
  {
   /* no VLC for this (run, level) combination: use escape coding (7.2.2.3) */
   putbits(1l,6); /* Escape */
   putbits(run,6); /* 6 bit code for run */
   if (mpeg1)
   {
    /* ISO/IEC 11172-2 uses a 8 or 16 bit code */
    if (signed_level>127)
      putbits(0,8);
    if (signed_level<-127)
      putbits(128,8);
    putbits(signed_level,8);
   }
   else
   {
    /* ISO/IEC 13818-2 uses a 12 bit code, Table B-16 */
    putbits(signed_level,12);
   }
  }
}

/* generate variable length code for macroblock_address_increment (6.3.16) */
void putaddrinc(int addrinc)
{
 while (addrinc>33)
 {
  putbits(0x08,11); /* macroblock_escape */
  addrinc-= 33;
 }

 putbits(addrinctab[addrinc-1].code,addrinctab[addrinc-1].len);
}

/* generate variable length code for macroblock_type (6.3.16.1) */
void putmbtype(int pict_type, int mb_type)
{
 putbits(mbtypetab[pict_type-1][mb_type].code,
      mbtypetab[pict_type-1][mb_type].len);
}

/* generate variable length code for motion_code (6.3.16.3) */
void putmotioncode(int motion_code)
{
 int abscode;

 abscode = (motion_code>=0) ? motion_code : -motion_code; /* abs(motion_code) */
 putbits(motionvectab[abscode].code,motionvectab[abscode].len);
 if (motion_code!=0)
  putbits(motion_code<0,1); /* sign, 0=positive, 1=negative */
```

```
}

/* generate variable length code for dmvector[t] (6 3.16 3), Table B-11 */
void putdmv(int dmv)
{
 if (dmv==0)
  putbits(0,1);
 else if (dmv>0)
  putbits(2,2);
 else
  putbits(3,2);
}

/* generate variable length code for coded_block_pattern (6.3.16.4)
 *
 * 4:2:2, 4:4:4 not implemented
 */
void putcbp(int cbp)
{
 putbits(cbptable[cbp] code,cbptable[cbp].len);
}

/* rate control variables */
int Xi, Xp, Xb, r, d0i, d0p, d0b;
double avg_act;
static int R, T, d;
static double actsum;
static int Np, Nb, S, Q;
static int prev_mquant;

void rc_init_seq()
{
 /* reaction parameter (constant) */
 if (r==0)  r = (int)floor(2.0*bit_rate/frame_rate + 0.5);

 /* average activity */
 if (avg_act==0.0)  avg_act = 400 0;

 /* remaining # of bits in GOP */
 R = 0;

 /* global complexity measure */
 if (Xi==0) Xi = (int)floor(160.0*bit_rate/115.0 + 0.5);
 if (Xp==0) Xp = (int)floor( 60.0*bit_rate/115.0 + 0.5);
 if (Xb==0) Xb = (int)floor( 42.0*bit_rate/115.0 + 0.5),

 /* virtual buffer fullness */
 if (d0i==0) d0i = (int)floor(10.0*r/31 0 + 0.5);
 if (d0p==0) d0p = (int)floor(10.0*r/31.0 + 0.5),
 if (d0b==0) d0b = (int)floor(1.4*10.0*r/31.0 + 0.5),

}

void rc_init_GOP(int np, int nb )
```

```
{
  R += (int) floor((1 + np + nb) * bit_rate / frame_rate + 0.5);
  Np = fieldpic ? 2*np+1 : np;
  Nb = fieldpic ? 2*nb : nb;

}

/* Note: we need to substitute K for the 1.4 and 1.0 constants -- this can
   be modified to fit image content */

/* Step 1: compute target bits for current picture being coded */
void rc_init_pict(unsigned char *frame )
{
  double Tmin;

  switch (pict_type)
  {
  case I_TYPE:
    T = (int) floor(R/(1.0+Np*Xp/(Xi*1.0)+Nb*Xb/(Xi*1.4)) + 0.5);
    d = d0i;
    break;
  case P_TYPE:
    T = (int) floor(R/(Np+Nb*1.0*Xb/(1.4*Xp)) + 0.5);
    d = d0p;
    break;
  case B_TYPE:
    T = (int) floor(R/(Nb+Np*1.4*Xp/(1.0*Xb)) + 0.5);
    d = d0b;
    break;
  }

  Tmin = (int) floor(bit_rate/(8.0*frame_rate) + 0.5);

  if (T<Tmin)
    T = Tmin;

  S = bitcount();
  Q = 0;

  calc_actj(frame);
  actsum = 0.0;

}

static void calc_actj(unsigned char *frame )
{
  int i,j,k;
  unsigned char *p;
  double actj,var;

  k = 0;

  for (j=0; j<height2; j+=16)
    for (i=0; i<width; i+=16)
```

```
    {
     p = frame + ((pict_struct==BOTTOM_FIELD)?width:0) + i + width2*j;

     /* take minimum spatial activity measure of luminance blocks */

     actj = var_sblk(p,width2);
     var = var_sblk(p+8,width2);
     if (var<actj) actj = var;
     var = var_sblk(p+8*width2,width2);
     if (var<actj) actj = var;
     var = var_sblk(p+8*width2+8,width2);
     if (var<actj) actj = var;

     if (!fieldpic && !prog_seq)
     {
      /* field */
      var = var_sblk(p,width<<1);
      if (var<actj) actj = var;
      var = var_sblk(p+8,width<<1);
      if (var<actj) actj = var;
      var = var_sblk(p+width,width<<1);
      if (var<actj) actj = var;
      var = var_sblk(p+width+8,width<<1);
      if (var<actj) actj = var;
     }

     actj+= 1.0;

     mbinfo[k++].act = actj;
    }
  }

void rc_update_pict()
{
 double X;

 S = bitcount() - S; /* total # of bits in picture */
 R-= S; /* remaining # of bits in GOP */
 X = (int) floor(S*((0.5*(double)Q)/(mb_width*mb_height2)) + 0.5);
 d+= S - T;
 avg_act = actsum/(mb_width*mb_height2);

 switch (pict_type)
 {
 case I_TYPE:
  Xi = X;
  d0i = d;
  break;
 case P_TYPE:
  Xp = X,
  d0p = d;
  Np--;
  break;
 case B_TYPE:
```

```
   Xb = X;
   d0b = d;
   Nb--;
   break;
  }

 }

/* compute initial quantization stepsize (at the beginning of picture) */
int rc_start_mb()
{
 int mquant;

 if (q_scale_type)
 {
  mquant = (int) floor(2.0*d*31.0/r + 0.5);

  /* clip mquant to legal (linear) range */
  if (mquant<1) mquant = 1;
  if (mquant>112) mquant = 112;

  /* map to legal quantization level */
  mquant = non_linear_mquant_table[map_non_linear_mquant[mquant]];
 }
 else
 {
  mquant = (int) floor(d*31.0/r + 0.5);
  mquant <<= 1;

  /* clip mquant to legal (linear) range */
  if (mquant<2) mquant = 2;
  if (mquant>62) mquant = 62;

  prev_mquant = mquant;
 }

 return mquant;
}

/* Step 2: measure virtual buffer - estimated buffer discrepancy */
int rc_calc_mquant(int j)
{
 int mquant;
 double dj, Qj, actj, N_actj;

 /* measure virtual buffer discrepancy from uniform distribution model */
 dj = d + (bitcount()-S) - j*(T/(mb_width*mb_height2));

 /* scale against dynamic range of mquant and the bits/picture count */
 Qj = dj*31.0/r;

 actj = mbinfo[j].act;
 actsum+= actj;
```

```
 /* compute normalized activity */
 N_actj = (2.0*actj+avg_act)/(actj+2.0*avg_act);

 if (q_scale_type)
 {
  /* modulate mquant with combined buffer and local activity measures */
  mquant = (int) floor(2.0*Qj*N_actj + 0.5);

  /* clip mquant to legal (linear) range */
  if (mquant<1) mquant = 1,
  if (mquant>112) mquant = 112,

  /* map to legal quantization level */
  mquant = non_linear_mquant_table[map_non_linear_mquant[mquant]];
 }
 else
 {
  /* modulate mquant with combined buffer and local activity measures */
  mquant = (int) floor(Qj*N_actj + 0.5);
  mquant <<= 1;

  /* clip mquant to legal (linear) range */
  if (mquant<2) mquant = 2;
  if (mquant>62) mquant = 62;

  /* ignore small changes in mquant */
  if (mquant>=8 && (mquant-prev_mquant)>=-4 && (mquant-prev_mquant)<=4)
   mquant = prev_mquant;

  prev_mquant = mquant;
 }

 Q+= mquant; /* for calculation of average mquant */

 return mquant;
}

/* compute variance of 8x8 block */
static double var_sblk(unsigned char *p, int lx)
{
 int i, j;
 unsigned int v, s, s2;

 s = s2 = 0;

 for (j=0; j<8; j++)
 {
  for (i=0, i<8; i++)
  {
   v = *p++;
   s+= v;
   s2+= v*v;
  }
  p+= lx - 8;
```

```
 }

 return s2/64.0 - (s/64.0)*(s/64.0),
}

/* VBV calculations
 *
 * generates warnings if underflow or overflow occurs
 */

/* vbv_end_of_picture
 *
 * - has to be called directly after writing picture_data()
 * - needed for accurate VBV buffer overflow calculation
 * - assumes there is no byte stuffing prior to the next start code
 */

static int bitcnt_EOP;

void vbv_end_of_picture()
{
 bitcnt_EOP = bitcount();
 bitcnt_EOP = (bitcnt_EOP + 7) & ~7; /* account for bit stuffing */
}

/* calc_vbv_delay
 *
 * has to be called directly after writing the picture start code, the
 * reference point for vbv_delay
 */

void calc_vbv_delay()
{
 double picture_delay;
 static double next_ip_delay; /* due to frame reordering delay */
 static double decoding_time;

 /* number of 1/90000 s ticks until next picture is to be decoded */
 if (pict_type == B_TYPE)
 {
  if (prog_seq)
  {
   if (!repeatfirst)
    picture_delay = 90000.0/frame_rate, /* 1 frame */
   else
   {
    if (!topfirst)
     picture_delay = 90000.0*2.0/frame_rate; /* 2 frames */
    else
     picture_delay = 90000.0*3.0/frame_rate; /* 3 frames */
   }
  }
  else
  {
```

```
      /* interlaced */
      if (fieldpic)
        picture_delay = 90000.0/(2.0*frame_rate); /* 1 field */
      else
      {
        if (!repeatfirst)
          picture_delay = 90000.0*2.0/(2.0*frame_rate); /* 2 flds */
        else
          picture_delay = 90000.0*3.0/(2.0*frame_rate); /* 3 flds */
      }
    }
  }
  else
  {
    /* I or P picture */
    if (fieldpic)
    {
      if(topfirst==(pict_struct==TOP_FIELD))
      {
        /* first field */
        picture_delay = 90000.0/(2.0*frame_rate);
      }
      else
      {
        /* second field */
        /* take frame reordering delay into account */
        picture_delay = next_ip_delay - 90000.0/(2 0*frame_rate);
      }
    }
    else
    {
      /* frame picture */
      /* take frame reordering delay into account*/
      picture_delay = next_ip_delay;
    }

    if (!fieldpic || topfirst!=(pict_struct==TOP_FIELD))
    {
      /* frame picture or second field */
      if (prog_seq)
      {
        if (!repeatfirst)
          next_ip_delay = 90000.0/frame_rate;
        else
        {
          if (!topfirst)
            next_ip_delay = 90000.0*2.0/frame_rate;
          else
            next_ip_delay = 90000.0*3.0/frame_rate;
        }
      }
      else
      {
        if (fieldpic)
```

```
        next_ip_delay = 90000.0/(2.0*frame_rate);
      else
      {
        if (!repeatfirst)
          next_ip_delay = 90000.0*2.0/(2.0*frame_rate);
        else
          next_ip_delay = 90000.0*3.0/(2.0*frame_rate);
      }
    }
  }
}

if (decoding_time==0.0)
{
  /* first call of calc_vbv_delay */
  /* we start with a 7/8 filled VBV buffer (12.5% back-off) */
  picture_delay = ((vbv_buffer_size*16384*7)/8)*90000.0/bit_rate;
  if (fieldpic)
    next_ip_delay = (int)(90000.0/frame_rate+0.5);
}

/* VBV checks */

/* check for underflow (previous picture) */
if (!low_delay && (decoding_time < bitcnt_EOP*90000.0/bit_rate))
{
  /* picture not completely in buffer at intended decoding time */
  if (!quiet)
    fprintf(stderr,"vbv_delay underflow! (decoding_time=%.1f, t_EOP=%.1f\n)",
      decoding_time, bitcnt_EOP*90000.0/bit_rate);
}

/* when to decode current frame */
decoding_time += picture_delay;

/* warning: bitcount() may overflow (e.g. after 9 min. at 8 Mbit/s */
vbv_delay = ( unsigned int)(decoding_time - bitcount()*90000.0/bit_rate);

/* check for overflow (current picture) */
if ((decoding_time - bitcnt_EOP*90000.0/bit_rate)
    > (vbv_buffer_size*16384)*90000.0/bit_rate)
{
  if (!quiet)
    fprintf(stderr,"vbv_delay overflow!\n");
}

if (vbv_delay<0)
{
  if (!quiet)
    fprintf(stderr,"vbv_delay underflow: %d\n",vbv_delay);
  vbv_delay = 0;
}

if (vbv_delay>65535)
```

```
 {
  if (!quiet)
            fprintf(stderr,"vbv_delay overflow: %d\n",vbv_delay);
  vbv_delay = 65535,
 }
}

/* Test Model 5 quantization
 *
 * this quantizer has a bias of 1/8 stepsize towards zero
 * (except for the DC coefficient)
 */
int quant_intra(short *src, short *dst, int dc_prec,
         unsigned char *quant_mat, int mquant )
{
 int i;
 int x, y, d;

 x = src[0];
 d = 8>>dc_prec; /* intra_dc_mult */
 dst[0] = (x>=0) ? (x+(d>>1))/d : -((-x+(d>>1))/d); /* round(x/d) */

 for (i=1; i<64; i++)
 {
  x = src[i];
  d = quant_mat[i];
  y = (32*(x>=0 ? x : -x) + (d>>1))/d; /* round(32*x/quant_mat) */
  d = (3*mquant+2)>>2;
  y = (y+d)/(2*mquant); /* (y+0.75*mquant) / (2*mquant) */

  /* clip to syntax limits */
  if (y > 255)
  {
   if (mpeg1)
    y = 255;
   else if (y > 2047)
    y = 2047;
  }

  dst[i] = (x>=0) ? y : -y;

#if 0
  /* this quantizer is virtually identical to the above */
  if (x<0)
   x = -x;
  d = mquant*quant_mat[i];
  y = (16*x + ((3*d)>>3)) / d;
  dst[i] = (src[i]<0) ? -y : y;
#endif
 }

 return 1;
}
```

```
int quant_non_intra(short *src, short *dst,
            unsigned char *quant_mat, int mquant )
{
  int i;
  int x, y, d;
  int nzflag;

  nzflag = 0;

  for (i=0; i<64; i++)
  {
    x = src[i];
    d = quant_mat[i];
    y = (32*(x>=0 ? x : -x) + (d>>1))/d; /* round(32*x/quant_mat) */
    y /= (2*mquant);

    /* clip to syntax limits */
    if (y > 255)
    {
      if (mpeg1)
        y = 255;
      else if (y > 2047)
        y = 2047;
    }

    if ((dst[i] = (x>=0 ? y : -y)) != 0)
      nzflag=1;
  }

  return nzflag;
}

/* MPEG-2 inverse quantization */
void iquant_intra(short *src, short *dst, int dc_prec,
            unsigned char *quant_mat, int mquant )
{
  int i, val, sum;

  if (mpeg1)
    iquant1_intra(src,dst,dc_prec,quant_mat,mquant);
  else
  {
    sum = dst[0] = src[0] << (3-dc_prec),
    for (i=1; i<64; i++)
    {
      val = (int)(src[i]*quant_mat[i]*mquant)/16;
      sum+= dst[i] = (val>2047) ? 2047 : ((val<-2048) ? -2048 : val);
    }

    /* mismatch control */
    if ((sum&1)==0)
      dst[63]^= 1;
  }
}
```

```
void iquant_non_intra(short *src, short *dst,
                      unsigned char *quant_mat, int mquant )
{
  int i, val, sum;

  if (mpeg1)
    iquant1_non_intra(src,dst,quant_mat,mquant);
  else
  {
    sum = 0;
    for (i=0; i<64; i++)
    {
      val = src[i];
      if (val!=0)
        val = (int)((2*val+(val>0 ? 1 : -1))*quant_mat[i]*mquant)/32;
      sum+= dst[i] = (val>2047) ? 2047 : ((val<-2048) ? -2048 : val);
    }

    /* mismatch control */
    if ((sum&1)==0)
      dst[63]^= 1;
  }
}

/* MPEG-1 inverse quantization */
static void iquant1_intra(short *src, short *dst, int dc_prec,
                          unsigned char *quant_mat, int mquant )
{
  int i, val;

  dst[0] = src[0] << (3-dc_prec);
  for (i=1; i<64; i++)
  {
    val = (int)(src[i]*quant_mat[i]*mquant)/16;

    /* mismatch control */
    if ((val&1)==0 && val!=0)
      val+= (val>0) ? -1 : 1;

    /* saturation */
    dst[i] = (val>2047) ? 2047 : ((val<-2048) ? -2048 : val);
  }
}

static void iquant1_non_intra(short *src, short *dst,
                              unsigned char *quant_mat, int mquant )
{
  int i, val;

  for (i=0; i<64; i++)
  {
    val = src[i];
    if (val!=0)
```

```
  {
   val = (int)((2*val+(val>0 ? 1 : -1))*quant_mat[i]*mquant)/32;

   /* mismatch control */
   if ((val&1)==0 && val!=0)
     val+= (val>0) ? -1 : 1;
  }

  /* saturation */
  dst[i] = (val>2047) ? 2047 : ((val<-2048) ? -2048 : val),
 }
}
```

```
/********************************/
/*                              */
/*      m0.c residing at Root   */
/*                              */
/********************************/

#include "c:\tc2v2\dos.h"
#include "c:\tc2v2\float.h"
#include "c:\tc2v2\math.h"
#include "c:\tc2v2\ctype.h"
#include "c:\tc2v2\stdio.h"
#include "c:\tc2v2\stdlib.h"
#include "c:\tc2v2\string.h"
#include "c:\tc2v2\chan.h"
#include "c:\tc2v2\thread.h"
#include "c:\tc2v2\time.h"
#include "c:\tc2v2\errno.h"

#define GLOBAL /* used by global.h */

#include "c:\nick\h\const1.h"
#include "c:\nick\h\config.h"
#include "c:\nick\h\global.h"

static void init _ANSI_ARGS_((void));
static void readparmfile _ANSI_ARGS_((char *fname));

char    *neworg[1];

void main(argc,argv,envp,ins,in_ports,outs,out_ports)
int argc,ins,outs;
char *argv[],*envp[];
CHAN *in_ports[],*out_ports[];
{
 if (argc!=2)
 {
  printf("Usage: mpeg2encode in.par\n");
  exit(0);
 }

 /* read parameter file */
 readparmfile(argv[1]),

 init();
 putseq();

}

void error(char *text)
{
 fprintf(stderr,text);
 putc('\n',stderr);
 exit(1);
}
```

```
static void readparmfile(char *fname)
{
 int i;
 int h,m,s,f;
 FILE *fd;
 char line[256];
 int r,Xi,Xb,Xp,d0i,d0p,d0b; /* rate control */
 double avg_act; /* rate control */

 sprintf(line,"c:\\nick\\par\\%s",fname);
 if (!(fd = fopen(line,"r")))
 {
  sprintf(errortext,"Couldn't open parameter file %s",line);
  error(errortext);
 }

 fgets(id_string,254,fd);
 fgets(line,254,fd); sscanf(line,"%s",tplorg);
 fgets(line,254,fd); sscanf(line,"%s",tplref);
 fgets(line,254,fd); sscanf(line,"%s",iqname);
 fgets(line,254,fd); sscanf(line,"%s",niqname);
 fgets(line,254,fd); sscanf(line,"%s",statname),
 fgets(line,254,fd); sscanf(line,"%d",&inputtype);
 fgets(line,254,fd); sscanf(line,"%d",&nframes);
 fgets(line,254,fd); sscanf(line,"%d",&frame0);
 fgets(line,254,fd); sscanf(line,"%d:%d:%d:%d",&h,&m,&s,&f);
 fgets(line,254,fd); sscanf(line,"%d",&N);
 fgets(line,254,fd); sscanf(line,"%d",&M);
 fgets(line,254,fd); sscanf(line,"%d",&mpeg1);
 fgets(line,254,fd); sscanf(line,"%d",&fieldpic);
 fgets(line,254,fd); sscanf(line,"%d",&horizontal_size);
 fgets(line,254,fd); sscanf(line,"%d",&vertical_size);
 fgets(line,254,fd); sscanf(line,"%d",&aspectratio);
 fgets(line,254,fd); sscanf(line,"%d",&frame_rate_code);
 fgets(line,254,fd), sscanf(line,"%lf",&bit_rate);
 fgets(line,254,fd); sscanf(line,"%d",&vbv_buffer_size);
 fgets(line,254,fd); sscanf(line,"%d",&low_delay);
 fgets(line,254,fd); sscanf(line,"%d",&constrparms);
 fgets(line,254,fd); sscanf(line,"%d",&profile);
 fgets(line,254,fd); sscanf(line,"%d",&level);
 fgets(line,254,fd); sscanf(line,"%d",&prog_seq);
 fgets(line,254,fd); sscanf(line,"%d",&chroma_format);
 fgets(line,254,fd); sscanf(line,"%d",&video_format);
 fgets(line,254,fd); sscanf(line,"%d",&color_primaries);
 fgets(line,254,fd); sscanf(line,"%d",&transfer_characteristics);
 fgets(line,254,fd); sscanf(line,"%d",&matrix_coefficients);
 fgets(line,254,fd); sscanf(line,"%d",&display_horizontal_size);
 fgets(line,254,fd); sscanf(line,"%d",&display_vertical_size);
 fgets(line,254,fd); sscanf(line,"%d",&dc_prec);
 fgets(line,254,fd); sscanf(line,"%d",&topfirst);
 fgets(line,254,fd); sscanf(line,"%d %d %d",
  frame_pred_dct_tab,frame_pred_dct_tab+1,frame_pred_dct_tab+2);
```

```
 fgets(line,254,fd); sscanf(line,"%d %d %d",
  conceal_tab,conceal_tab+1,conceal_tab+2);

 fgets(line,254,fd); sscanf(line,"%d %d %d",
  qscale_tab,qscale_tab+1,qscale_tab+2);

 fgets(line,254,fd); sscanf(line,"%d %d %d",
  intravlc_tab,intravlc_tab+1,intravlc_tab+2);
 fgets(line,254,fd); sscanf(line,"%d %d %d",
  altscan_tab,altscan_tab+1,altscan_tab+2);
 fgets(line,254,fd); sscanf(line,"%d",&repeatfirst);
 fgets(line,254,fd); sscanf(line,"%d",&prog_frame);
/* intra slice interval refresh period */
 fgets(line,254,fd); sscanf(line,"%d",&P);
 fgets(line,254,fd); sscanf(line,"%d",&r);
 fgets(line,254,fd); sscanf(line,"%lf",&avg_act);
 fgets(line,254,fd); sscanf(line,"%d",&Xi);
 fgets(line,254,fd); sscanf(line,"%d",&Xp);
 fgets(line,254,fd); sscanf(line,"%d",&Xb);
 fgets(line,254,fd); sscanf(line,"%d",&d0i);
 fgets(line,254,fd); sscanf(line,"%d",&d0p);
 fgets(line,254,fd); sscanf(line,"%d",&d0b),

 if (N<1) error("N must be positive");
 if (M<1) error("M must be positive"),
 if (N%M != 0) error("N must be an integer multiple of M");

 motion_data = (struct motion_data *)malloc(M*sizeof(struct motion_data));
 if (!motion_data) error("malloc failed motion_data\n");

 for (i=0; i<M; i++)
 {
  fgets(line,254,fd);
  sscanf(line,"%d %d %d %d",
   &motion_data[i] forw_hor_f_code, &motion_data[i].forw_vert_f_code,
   &motion_data[i].sxf, &motion_data[i].syf);

  if (i!=0)
  {
   fgets(line,254,fd);
   sscanf(line,"%d %d %d %d",
    &motion_data[i].back_hor_f_code, &motion_data[i].back_vert_f_code,
    &motion_data[i].sxb, &motion_data[i].syb);
  }
 }

 fclose(fd);

 /* make flags boolean (x!=0 -> x=1) */
 fieldpic = !!fieldpic;
 prog_seq = !!prog_seq;
 topfirst = !!topfirst;

 for (i=0, i<3; i++)
```

```
  {
   frame_pred_dct_tab[i] = !!frame_pred_dct_tab[i];
   conceal_tab[i] = !!conceal_tab[i];
   qscale_tab[i] = !!qscale_tab[i];
   intravlc_tab[i] = !!intravlc_tab[i];
   altscan_tab[i] = !!altscan_tab[i];
  }
  repeatfirst = !!repeatfirst;
  prog_frame = !!prog_frame;

  chan_out_word(M,Link1Output);
  chan_out_word(M,Link2Output);

  chan_out_word(nframes,Link1Output);
  chan_out_word(nframes,Link2Output);

}


static void init()
{
  int i, size, a;

  /* round picture dimensions to nearest multiple of 16 or 32 */
  mb_width = (horizontal_size+15)/16;
  mb_height = prog_seq ? (vertical_size+15)/16 : 2*((vertical_size+31)/32);
  mb_height2 = fieldpic ? mb_height>>1 : mb_height; /* for field pictures */
  width = 16*mb_width;
  height = 16*mb_height;

  height2 = fieldpic ? height>>1 : height;
  width2 = fieldpic ? width<<1 : width;

  size = width*height ;

  if (!(newrefframe[0] = (char *)malloc(size)))
     error("malloc failed for newrefframe\n");
  if (!(oldrefframe[0] = (char *)malloc(size)))
     error("malloc failed oldrefframe\n");
  if (!(neworgframe[0] = (char *)malloc(size)))
     error("malloc failed for neworgframe\n");
  if (!(oldorgframe[0] = (char *)malloc(size)))
     error("malloc failed\n");
  if (!(neworg[0] = (char *)malloc(size)))
     error("malloc failed for auxorgframe\n");

  mbinfo = (struct mbinfo *)malloc(mb_width*mb_height2*sizeof(struct mbinfo));
  if (!mbinfo) error("malloc failed mbinfo\n");

  for (a=0; a<mb_width*mb_height2; a++)
  {
   mbinfo[a].mb_type = mbinfo[a].motion_type = 0;
   mbinfo[a].MV[0][0][0] = mbinfo[a].MV[0][0][1] = mbinfo[a].MV[0][1][0] = 0;
   mbinfo[a].MV[0][1][1] = mbinfo[a].MV[1][0][0] = mbinfo[a].MV[1][0][1] = 0;
```

```
    mbinfo[a].MV[1][1][0] = mbinfo[a].MV[1][1][1] = 0;
    mbinfo[a].mv_field_sel[0][0] = mbinfo[a].mv_field_sel[0][1] = 0,
    mbinfo[a].mv_field_sel[1][0] = mbinfo[a] mv_field_sel[1][1] = 0,

   }

  chan_out_word(mb_width,Link1Output);
  chan_out_word(mb_width,Link2Output);

  chan_out_word(mb_height,Link1Output);
  chan_out_word(mb_height,Link2Output);

  chan_out_word(mb_height2,Link1Output);
  chan_out_word(mb_height2,Link2Output);

  chan_out_word(width,Link1Output);
  chan_out_word(width,Link2Output);

  chan_out_word(height,Link1Output);
  chan_out_word(height,Link2Output),

  chan_out_word(width2,Link1Output);
  chan_out_word(width2,Link2Output);

  chan_out_word(height2,Link1Output);
  chan_out_word(height2,Link2Output);

}


void putseq()
{
/* this routine assumes (N % M) == 0 */
  int i, j, k, f, f0, n, np, nb;
  int a;
  int sxf, syf, sxb, syb,
  FILE *f_mbi, *f_d, *f_status;
  char name[128], name1[128];
  int finished, size;

  size = width*height;

/* loop through all frames in encoding/decoding order */
  for (i=0; i<nframes; i++)
  {
    chan_out_word(i,Link1Output);
    chan_out_word(i,Link2Output),

/*  read in data from the first program     */
    sprintf(name,tplorg,i);
    sprintf(name1,"c:\\nick\\dmot\\%s.sta",name);
    if (!(f_status = fopen(name1,"r")))
    {
     sprintf(errortext,"Couldn't open %s\n",name1);
```

```
 error(errortext);
 }

fscanf(f_status,"%d %d %d %d %d %d %d %d ",
&pict_type,
&pict_struct,
&sxf, &syf,
&sxb, &syb,
&f,
&frame_pred_dct);

        fclose(f_status);

fprintf(f_time,"frame:%d\n", f);

        sprintf(name,tplorg,f+frame0);

sprintf(name1,"c:\\nick\\dmot\\%s.oof",name);
if (!(f_d = fopen(name1,"rb")))
{
 sprintf(errortext,"Couldn't open %s\n",name1);
 error(errortext);
}
fread(oldorgframe[0],1,size,f_d);
fclose(f_d);

sprintf(name1,"c:\\nick\\dmot\\%s.nof",name);
if (!(f_d = fopen(name1,"rb")))
{
 sprintf(errortext,"Couldn't open %s\n",name1);
 error(errortext);
}
fread(neworgframe[0],1,size,f_d);
fclose(f_d),

sprintf(name1,"c:\\nick\\dmot\\%s.off",name);
if (!(f_d = fopen(name1,"rb")))
{
 sprintf(errortext,"Couldn't open %s\n",name1);
 error(errortext);
}
fread(oldrefframe[0],1,size,f_d);
fclose(f_d);

sprintf(name1,"c:\\nick\\dmot\\%s.nff",name);
if (!(f_d = fopen(name1,"rb")))
{
 sprintf(errortext,"Couldn't open %s\n",name1);
 error(errortext);
}
fread(newrefframe[0],1,size,f_d);
fclose(f_d);

sprintf(name1,"c:\\nick\\dmot\\%s.nor",name);
```

```
  if (!(f_d = fopen(name1,"rb")))
  {
   sprintf(errortext,"Couldn't open %s\n",name1);
   error(errortext);
  }
  fread(neworg[0],1,size,f_d);
  fclose(f_d);

  chan_out_word(pict_type,Link1Output);
  chan_out_word(pict_type,Link2Output);

  chan_out_word(sxf,Link1Output);
  chan_out_word(sxf,Link2Output);

  chan_out_word(syf,Link1Output);
  chan_out_word(syf,Link2Output);

  chan_out_word(sxb,Link1Output);
  chan_out_word(sxb,Link2Output);

  chan_out_word(syb,Link1Output);
  chan_out_word(syb,Link2Output);

  if ( pict_type != I_TYPE )
  {
   chan_out_message(size,neworg[0],Link1Output);
   chan_out_message(size,neworg[0],Link2Output);

   chan_out_message(size,oldorgframe[0],Link1Output);
   chan_out_message(size,oldorgframe[0],Link2Output);

   chan_out_message(size,oldrefframe[0],Link1Output);
   chan_out_message(size,oldrefframe[0],Link2Output);

   if ( pict_type != P_TYPE )
   {
    chan_out_message(size,neworgframe[0],Link1Output);
    chan_out_message(size,neworgframe[0],Link2Output);

    chan_out_message(size,newrefframe[0],Link1Output);
    chan_out_message(size,newrefframe[0],Link2Output);
   }
  }

/* loop through 36 macroblocks of the picture */

  motion_estimation(oldorgframe[0], neworgframe[0],
            oldrefframe[0], newrefframe[0],
            neworg[0],
            sxf, syf,
            sxb, syb,
            mbinfo);

   for (a=36; a<146; a++)
```

```
{
 chan_in_word(&mbinfo[a].mb_type,Link1Input);
 chan_in_word(&mbinfo[a].motion_type,Link1Input);
 chan_in_word(&mbinfo[a].MV[0][0][0],Link1Input);
 chan_in_word(&mbinfo[a].MV[0][0][1],Link1Input);
 chan_in_word(&mbinfo[a].MV[0][1][0],Link1Input);
 chan_in_word(&mbinfo[a].MV[0][1][1],Link1Input),
 chan_in_word(&mbinfo[a].MV[1][0][0],Link1Input);
 chan_in_word(&mbinfo[a].MV[1][0][1],Link1Input);
 chan_in_word(&mbinfo[a].MV[1][1][0],Link1Input);
 chan_in_word(&mbinfo[a].MV[1][1][1],Link1Input);
 chan_in_word(&mbinfo[a].mv_field_sel[0][0],Link1Input);
 chan_in_word(&mbinfo[a].mv_field_sel[0][1],Link1Input);
 chan_in_word(&mbinfo[a].mv_field_sel[1][0],Link1Input);
 chan_in_word(&mbinfo[a].mv_field_sel[1][1],Link1Input);
}

for (a=146, a<256; a++)
{
 chan_in_word(&mbinfo[a].mb_type,Link2Input);
 chan_in_word(&mbinfo[a].motion_type,Link2Input);
 chan_in_word(&mbinfo[a].MV[0][0][0],Link2Input);
 chan_in_word(&mbinfo[a].MV[0][0][1],Link2Input);
 chan_in_word(&mbinfo[a].MV[0][1][0],Link2Input);
 chan_in_word(&mbinfo[a].MV[0][1][1],Link2Input);
 chan_in_word(&mbinfo[a].MV[1][0][0],Link2Input);
 chan_in_word(&mbinfo[a].MV[1][0][1],Link2Input);
 chan_in_word(&mbinfo[a].MV[1][1][0],Link2Input);
 chan_in_word(&mbinfo[a].MV[1][1][1],Link2Input);
 chan_in_word(&mbinfo[a].mv_field_sel[0][0],Link2Input);
 chan_in_word(&mbinfo[a].mv_field_sel[0][1],Link2Input);
 chan_in_word(&mbinfo[a].mv_field_sel[1][0],Link2Input);
 chan_in_word(&mbinfo[a].mv_field_sel[1][1],Link2Input);
}

/*  write data for the third program                */

  sprintf(name1,"c:\\nick\\mot_out\\%s.mbi",name);
  if (!(f_mbi = fopen(name1,"w/0")))
  {
   sprintf(errortext,"Couldn't open %s\n",name1),
   error(errortext);
  }

  for (a=0; a<mb_width*mb_height2; a++)
  {
   fprintf(f_mbi,"%d %d %d %d %d %d %d %d %d %d %d %d %d %d \n",
   mbinfo[a].mb_type,
   mbinfo[a].motion_type,
   mbinfo[a].MV[0][0][0],
   mbinfo[a].MV[0][0][1],
   mbinfo[a].MV[0][1][0],
   mbinfo[a].MV[0][1][1],
   mbinfo[a].MV[1][0][0],
```

```
    mbinfo[a].MV[1][0][1],
    mbinfo[a].MV[1][1][0],
    mbinfo[a].MV[1][1][1],
    mbinfo[a].mv_field_sel[0][0],
    mbinfo[a].mv_field_sel[0][1],
    mbinfo[a].mv_field_sel[1][0],
    mbinfo[a].mv_field_sel[1][1]);
   }

   fclose(f_mbi);

  }

 chan_out_word(i,Link1Output);
 chan_out_word(i,Link2Output);

 chan_in_word(&finished,Link1Input);
 chan_in_word(&finished,Link2Input);

}/*end of main()*/


static void frame_estimate _ANSI_ARGS_((char *org,
                               char *ref,
                               char *mb,
                               int i, int j,
                               int sx, int sy,
                               int *iminp, int *jminp,
                               int *imintp, int *jmintp,
                               int *iminbp, int *jminbp,
                               float *dframep, float *dfieldp,
                               int *tselp, int *bselp,
                               int imins[2][2], int jmins[2][2]));

static float fullsearch _ANSI_ARGS_((char *org,
                             char *ref,
                             char *blk,
                             int lx,
                             int i0,
                             int j0,
                             int sx,
                             int sy,
                             char h,
                             int xmax,
                             int ymax,
                             int *iminp,
                             int *jminp));

static float dist1 _ANSI_ARGS_((char *blk1,
                       char *blk2,
                       int lx,
                       char hx,
                       char hy,
                       char h,
```

```
                    float distlim));

static float dist2 _ANSI_ARGS_((char *blk1,
                    char *blk2,
                    int lx,
                    char hx,
                    char hy,
                    char h)),

static float bdist1 _ANSI_ARGS_((char *pf,
                    char *pb,
                    char *p2,
                    int lx,
                    int hxf,
                    int hyf,
                    int hxb,
                    int hyb,
                    char h));

static float bdist2 _ANSI_ARGS_((char *pf,
                    char *pb,
                    char *p2,
                    int lx,
                    int hxf,
                    int hyf,
                    int hxb,
                    int hyb,
                    char h));

static float variance _ANSI_ARGS_((char *p, int lx));


void motion_estimation(char *oldorg, char *neworg,
                char *oldref, char *newref,
                char *cur,
                int sxf,
                int syf,
                int sxb,
                int syb,
                struct mbinfo *mbi)
{
 int i, j;
 int imin,jmin,iminf,jminf,iminr,jminr,
 int imint,jmint,iminb,jminb;
 int imintf,jmintf,iminbf,jminbf;
 int imintr,jmintr,iminbr,jminbr;
 float var,v0;
 float dmc,dmcf,dmcr,dmci,vmc,vmcf,vmcr,vmci;
 float dmcfield,dmcfieldf,dmcfieldr,dmcfieldi;
 int tsel,bsel,tself,bself,tselr,bselr;
 char *mb;
 int imins[2][2],jmins[2][2];
 int a, ii, jj;
```

```
/* loop through 36 macroblocks of the picture */

 for(a=0; a<36; a++)
 {
  jj = a>>4;
  ii = a - (jj<<4);
  j = jj<<4;
  i = ii<<4;

  mb = cur + i + width*j;

  var = variance(mb,width);

  /*  for I_type pictures    */

  if (pict_type==I_TYPE)
  mbi->mb_type = MB_INTRA;

  /*  for P_type pictures    */

  if (pict_type==P_TYPE)
  {
   frame_estimate(oldorg,oldref,mb,i,j,sxf,syf,
               &imin,&jmin,&imint,&jmint,&iminb,&jminb,
               &dmc,&dmcfield,&tsel,&bsel,imins,jmins);

  /* select between frame and field prediction */
  if (dmc<=dmcfield)
  {
   mbi->motion_type = MC_FRAME;
   vmc = dist2(oldref+(imin>>1)+width*(jmin>>1),mb,
           width,imin&1,jmin&1,16);
  }
  else
  {
   mbi->motion_type = MC_FIELD;
   dmc = dmcfield;
   vmc = dist2(oldref+(tsel?width:0)+(imint>>1)+(width<<1)*(jmint>>1),
           mb,width<<1,imint&1,jmint&1,8);
   vmc+= dist2(oldref+(bsel?width:0)+(iminb>>1)+(width<<1)*(jminb>>1),
           mb+width,width<<1,iminb&1,jminb&1,8);
  }

  if (vmc>var && vmc>=9*256)
  mbi->mb_type = MB_INTRA;
  else
  {
   v0 = dist2(oldref+i+width*j,mb,width,0,0,16);
   if (4*v0>5*vmc && v0>=9*256)
   {
    /* use MC */
    var = vmc;
    mbi->mb_type = MB_FORWARD;
    if (mbi->motion_type==MC_FRAME)
```

```
  {
   mbi->MV[0][0][0] = imin - (i<<1);
   mbi->MV[0][0][1] = jmin - (j<<1);
  }
  else /* if (mbi->motion_type==MC_FIELD) */
  {
   /* these are FRAME vectors */
   mbi->MV[0][0][0] = imint - (i<<1);
   mbi->MV[0][0][1] = (jmint<<1) - (j<<1);
   mbi->MV[1][0][0] = iminb - (i<<1);
   mbi->MV[1][0][1] = (jminb<<1) - (j<<1);
   mbi->mv_field_sel[0][0] = tsel;
   mbi->mv_field_sel[1][0] = bsel;
  }
 }
 else
 {
  /* No-MC */
  var = v0;
  mbi->mb_type = 0;
  mbi->motion_type = MC_FRAME;
  mbi->MV[0][0][0] = 0;
  mbi->MV[0][0][1] = 0;
 }
}
}

/*  for B_type pictures     */

if (pict_type==B_TYPE)
{
 /* forward prediction */
 frame_estimate(oldorg,oldref,mb,i,j,sxf,syf,
           &iminf,&jminf,&imintf,&jmintf,&iminbf,&jminbf,
           &dmcf,&dmcfieldf,&tself,&bself,imins,jmins);

 /* backward prediction */
 frame_estimate(neworg,newref,mb,i,j,sxb,syb,
           &iminr,&jminr,&imintr,&jmintr,&iminbr,&jminbr,
           &dmcr,&dmcfieldr,&tselr,&bselr,imins,jmins);

 /* calculate interpolated distance */
 /* frame */
 dmci = bdist1(oldref+(iminf>>1)+width*(jminf>>1),
          newref+(iminr>>1)+width*(jminr>>1),
          mb,width,iminf&1,jminf&1,iminr&1,jminr&1,16);

 /* top field
 dmcfieldi = bdist1(oldref+(imintf>>1)+(tself?width:0)+(width<<1)*(jmintf>>1),
               newref+(imintr>>1)+(tselr?width:0)+(width<<1)*(jmintr>>1),
               mb,width<<1,imintf&1,jmintf&1,imintr&1,jmintr&1,8);

 /* bottom field
```

```
dmcfieldi+= bdist1(oldref+(iminbf>>1)+(bself?width:0)+(width<<1)*(jminbf>>1),
            newref+(iminbr>>1)+(bselr?width:0)+(width<<1)*(jminbr>>1),
            mb+width,width<<1,iminbf&1,jminbf&1,iminbr&1,jminbr&1,8);

/* select prediction type of minimum distance from the
 * six candidates (field/frame * forward/backward/interpolated)
 */

if (dmci<dmcfieldi && dmci<dmcf && dmci<dmcfieldf
  && dmci<dmcr && dmci<dmcfieldr)
{
 /* frame, interpolated */
 mbi->mb_type = MB_FORWARD|MB_BACKWARD;
 mbi->motion_type = MC_FRAME;
 vmc = bdist2(oldref+(iminf>>1)+width*(jminf>>1),
        newref+(iminr>>1)+width*(jminr>>1),
        mb,width,iminf&1,jminf&1,iminr&1,jminr&1,16);
}
else if (dmcfieldi<dmcf && dmcfieldi<dmcfieldf
     && dmcfieldi<dmcr && dmcfieldi<dmcfieldr)
{
 /* field, interpolated */
 mbi->mb_type = MB_FORWARD|MB_BACKWARD;
 mbi->motion_type = MC_FIELD;
 vmc = bdist2(oldref+(imintf>>1)+(tself?width:0)+(width<<1)*(jmintf>>1),
        newref+(imintr>>1)+(tselr?width:0)+(width<<1)*(jmintr>>1),
        mb,width<<1,imintf&1,jmintf&1,imintr&1,jmintr&1,8);
 vmc+= bdist2(oldref+(iminbf>>1)+(bself?width:0)+(width<<1)*(jminbf>>1),
        newref+(iminbr>>1)+(bselr?width:0)+(width<<1)*(jminbr>>1),
        mb+width,width<<1,iminbf&1,jminbf&1,iminbr&1,jminbr&1,8);
}
else if (dmcf<dmcfieldf && dmcf<dmcr && dmcf<dmcfieldr)
{
 /* frame, forward */
 mbi->mb_type = MB_FORWARD;
 mbi->motion_type = MC_FRAME;
 vmc = dist2(oldref+(iminf>>1)+width*(jminf>>1),mb,
       width,iminf&1,jminf&1,16);
}
else if (dmcfieldf<dmcr && dmcfieldf<dmcfieldr)
{
 /* field, forward */
 mbi->mb_type = MB_FORWARD;
 mbi->motion_type = MC_FIELD;
 vmc = dist2(oldref+(tself?width:0)+(imintf>>1)+(width<<1)*(jmintf>>1),
       mb,width<<1,imintf&1,jmintf&1,8);
 vmc+= dist2(oldref+(bself?width:0)+(iminbf>>1)+(width<<1)*(jminbf>>1),
       mb+width,width<<1,iminbf&1,jminbf&1,8);
}
else if (dmcr<dmcfieldr)
{
 /* frame, backward */
 mbi->mb_type = MB_BACKWARD;
 mbi->motion_type = MC_FRAME;
```

```
 vmc = dist2(newref+(iminr>>1)+width*(jminr>>1),mb,
          width,iminr&1,jminr&1,16);
 }
 else
 {
 /* field, backward */
 mbi->mb_type = MB_BACKWARD;
 mbi->motion_type = MC_FIELD;
 vmc = dist2(newref+(tselr?width:0)+(imintr>>1)+(width<<1)*(jmintr>>1),
          mb,width<<1,imintr&1,jmintr&1,8);
 vmc+= dist2(newref+(bselr?width:0)+(iminbr>>1)+(width<<1)*(jminbr>>1),
          mb+width,width<<1,iminbr&1,jminbr&1,8);
 }


 if (vmc>var && vmc>=9*256)
 mbi->mb_type = MB_INTRA;
 else
 {
 var = vmc;
 if (mbi->motion_type==MC_FRAME)
 {
  /* forward */
  mbi->MV[0][0][0] = iminf - (i<<1);
  mbi->MV[0][0][1] = jminf - (j<<1);
  /* backward */
  mbi->MV[0][1][0] = iminr - (i<<1);
  mbi->MV[0][1][1] = jminr - (j<<1);
 }
 else
 {
  /* these are FRAME vectors */
  /* forward */
  mbi->MV[0][0][0] = imintf - (i<<1);
  mbi->MV[0][0][1] = (jmintf<<1) - (j<<1);
  mbi->MV[1][0][0] = iminbf - (i<<1);
  mbi->MV[1][0][1] = (jminbf<<1) - (j<<1);
  mbi->mv_field_sel[0][0] = tself;
  mbi->mv_field_sel[1][0] = bself;
  /* backward */
  mbi->MV[0][1][0] = imintr - (i<<1);
  mbi->MV[0][1][1] = (jmintr<<1) - (j<<1);
  mbi->MV[1][1][0] = iminbr - (i<<1);
  mbi->MV[1][1][1] = (jminbr<<1) - (j<<1);
  mbi->mv_field_sel[0][1] = tselr;
  mbi->mv_field_sel[1][1] = bselr;
 }
 }
}

mbi->var = var;

mbi++;
```

```
  if (!quiet)
  {
   putc('.',stderr);
   fflush(stderr);
  }

 }

 if (!quiet)  putc('\n',stderr);
}


static void frame_estimate(char *org, char *ref,
                          char *mb,
                          int i, int j,
                          int sx, int sy,
                          int *iminp, int *jminp,
                          int *imintp, int *jmintp,
                          int *iminbp, int *jminbp,
                          float *dframep, float *dfieldp,
                          int *tselp, int *bselp,
                          int imins[2][2], int jmins[2][2] )
{
 float dt,db,dmint,dminb;
 int imint,iminb,jmint,jminb;

 /* frame prediction  */
 *dframep = fullsearch(org,ref,mb,width,i,j,sx,sy,16,width,height,
                  iminp,jminp);

 /* predict top field from top field  */
 dt = fullsearch(org,ref,mb,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
           &imint,&jmint);

 /* predict top field from bottom field  */
 db = fullsearch(org+width,ref+width,mb,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
           &iminb,&jminb);

 imins[0][0] = imint;
 jmins[0][0] = jmint;
 imins[1][0] = iminb;
 jmins[1][0] = jminb;

 /* select prediction for top field */
 if (dt<=db)
 {
  dmint=dt; *imintp=imint; *jmintp=jmint; *tselp=0;
 }
 else
 {
  dmint=db; *imintp=iminb; *jmintp=jminb; *tselp=1;
 }

 /* predict bottom field from top field  */
```

```
 dt = fullsearch(org,ref,mb+width,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
          &imint,&jmint);

 /* predict bottom field from bottom field  */
 db = fullsearch(org+width, ref+width, mb+width,
          width<<1, i, j>>1, sx, sy>>1, 8, width, height>>1,
          &iminb,&jminb);

 imins[0][1] = imint;
 jmins[0][1] = jmint;
 imins[1][1] = iminb;
 jmins[1][1] = jminb;

 /* select prediction for bottom field */
 if (db<=dt)
 {
  dminb=db; *iminbp=iminb; *jminbp=jminb; *bselp=1;
 }
 else
 {
  dminb=dt; *iminbp=imint; *jminbp=jmint; *bselp=0;
 }

 *dfieldp=dmint+dminb;
}


static float fullsearch(char *org, char *ref,
              char *blk,
              int lx,
              int i0,
              int j0,
              int sx,
              int sy,
              char h,
              int xmax,
              int ymax,
              int *iminp,
              int *jminp)
{
 int i,j,imin,jmin,ilow,ihigh,jlow,jhigh;
 float d,dmin;
 int k,l,sxy;
 int ii, jj;

 ilow = i0 - sx;
 ihigh = i0 + sx;

 if (ilow<0) ilow = 0;

 if (ihigh>xmax-16) ihigh = xmax-16;

 jlow = j0 - sy;
 jhigh = j0 + sy;
```

```
if (jlow<0) jlow = 0;

if (jhigh>ymax-h) jhigh = ymax-h,

/* full pel search, spiraling outwards */

imin = i0;
jmin = j0;
dmin = dist1(org+imin+lx*jmin,blk,lx,0,0,h,65536.0);

sxy = (sx>sy) ? sx : sy;

for (l=1; l<=sxy; l++)
{
 i = i0 - l;
 j = j0 - l;
 for (k=0; k<8*l; k++)
 {
  if (i>=ilow && i<=ihigh && j>=jlow && j<=jhigh)
  {
   d = dist1(org+i+lx*j,blk,lx,0,0,h,dmin);

   if (d<dmin)
   {
    dmin = d;
    imin = i;
    jmin = j;
   }
  }

  if      (k<2*l) i++;
  else if (k<4*l) j++;
  else if (k<6*l) i--;
  else            j--;
 }
}

/* half pel */
dmin = 65536.0;
imin = imin << 1;
jmin = jmin << 1;
ilow = imin - (imin>0);
ihigh = imin + (imin<((xmax-16)<<1));
jlow = jmin - (jmin>0);
jhigh = jmin + (jmin<((ymax-h)<<1));

for (j=jlow; j<=jhigh; j++)
 for (i=ilow; i<=ihigh; i++)
 {
  ii = i >> 1 ;
  jj = j >> 1 ;
  d = dist1(ref+ii+lx*jj,blk,lx,i&1,j&1,h,dmin),
```

```
    if (d<dmin)
    {
     dmin = d;
     imin = i;
     jmin = j;
    }
   }

 *iminp = imin;
 *jminp = jmin;

 return dmin;
}


static float dist1(char *blk1,
           char *blk2,
           int lx,
           char hx,
           char hy,
           char h,
           float distlim )
{
 char *p1,*p1a,*p2;
 char i, j;
 int v;
 float s;

 s = 0;
 p1 = blk1;
 p2 = blk2;

 if (!hx && !hy)
  for (j=0; j<h; j++)
  {
   if ((v = p1[0]  - p2[0])<0)  v = -v; s+= (float) v;
   if ((v = p1[1]  - p2[1])<0)  v = -v; s+= (float) v;
   if ((v = p1[2]  - p2[2])<0)  v = -v; s+= (float) v;
   if ((v = p1[3]  - p2[3])<0)  v = -v; s+= (float) v,
   if ((v = p1[4]  - p2[4])<0)  v = -v; s+= (float) v;
   if ((v = p1[5]  - p2[5])<0)  v = -v; s+= (float) v;
   if ((v = p1[6]  - p2[6])<0)  v = -v; s+= (float) v;
   if ((v = p1[7]  - p2[7])<0)  v = -v; s+= (float) v;
   if ((v = p1[8]  - p2[8])<0)  v = -v; s+= (float) v;
   if ((v = p1[9]  - p2[9])<0)  v = -v; s+= (float) v;
   if ((v = p1[10] - p2[10])<0) v = -v; s+= (float) v;
   if ((v = p1[11] - p2[11])<0) v = -v; s+= (float) v;
   if ((v = p1[12] - p2[12])<0) v = -v; s+= (float) v;
   if ((v = p1[13] - p2[13])<0) v = -v; s+= (float) v;
   if ((v = p1[14] - p2[14])<0) v = -v; s+= (float) v;
   if ((v = p1[15] - p2[15])<0) v = -v, s+= (float) v;

   if (s >= distlim)
    break;
```

```
    p1+= lx;
    p2+= lx;
  }
else if (hx && !hy)
  for (j=0; j<h; j++)
  {
    for (i=0, i<16; i++)
    {
      v = (int) ((p1[i]+p1[i+1]+1) / 2 - p2[i]);
      if (v>=0) s+= (float) v;
      else s-= (float) v;
    }
    p1+= lx;
    p2+= lx;
  }
else if (!hx && hy)
{
  p1a = p1 + lx;
  for (j=0; j<h; j++)
  {
    for (i=0; i<16; i++)
    {
      v = (int ) ((p1[i]+p1a[i]+1) / 2 - p2[i]);
      if (v>=0)
        s+= (float) v;
      else
        s-= (float) v;
    }
    p1 = p1a;
    p1a+= lx;
    p2+= lx;
  }
}
else /* if (hx && hy) */
{
  p1a = p1 + lx;
  for (j=0; j<h; j++)
  {
    for (i=0; i<16; i++)
    {
      v = (int) ((p1[i]+p1[i+1]+p1a[i]+p1a[i+1]+2) / 4 - p2[i]);
      if (v>=0)
        s+= (float) v;
      else
        s-= (float) v;
    }
    p1 = p1a;
    p1a+= lx;
    p2+= lx;
  }
}

return s;
```

```
}

static float dist2(char *blk1,
                   char *blk2,
                   int lx,
                   char hx,
                   char hy,
                   char h )
{
 char *p1,*p1a,*p2;
 char i, j;
 int v;
 float s;

 s = 0;
 p1 = blk1;
 p2 = blk2;
 if (!hx && !hy)
  for (j=0; j<h; j++)
  {
   for (i=0; i<16; i++)
   {
    v = p1[i] - p2[i];
    s+= (float) (v*v);
   }
   p1+= lx;
   p2+= lx;
  }
 else if (hx && !hy)
  for (j=0; j<h; j++)
  {
   for (i=0, i<16; i++)
   {
    v = (int) ((p1[i]+p1[i+1]+1) / 2 - p2[i]);
    s+= (float) (v*v),
   }
   p1+= lx,
   p2+= lx;
  }
 else if (!hx && hy)
 {
  p1a = p1 + lx;
  for (j=0; j<h; j++)
  {
   for (i=0; i<16; i++)
   {
    v = (int) ((p1[i]+p1a[i]+1) / 2 - p2[i]),
    s+= (float) (v*v);
   }
   p1 = p1a;
   p1a+= lx;
   p2+= lx;
  }
 }
```

```
  else /* if (hx && hy) */
  {
    p1a = p1 + lx;
    for (j=0; j<h; j++)
    {
      for (i=0; i<16; i++)
      {
        v = (int) ((p1[i]+p1[i+1]+p1a[i]+p1a[i+1]+2) / 4 - p2[i]);
        s+= (float) (v*v);
      }
      p1 = p1a;
      p1a+= lx;
      p2+= lx;
    }
  }

  return s;
}

static float bdist1(char *pf, char *pb, char *p2,
                    int lx, int hxf,
                    int hyf, int hxb,
                    int hyb, char h )
{
  char *pfa,*pfb,*pfc,*pba,*pbb,*pbc;
  char i, j;
  int v;
  float s;

  pfa = pf + hxf;
  pfb = pf + lx*hyf;
  pfc = pfb + hxf;

  pba = pb + hxb;
  pbb = pb + lx*hyb;
  pbc = pbb + hxb;

  s = 0;

  for (j=0; j<h; j++)
  {
    for (i=0; i<16; i++)
    {
      v = (int) (((*pf++ + *pfa++ + *pfb++ + *pfc++ + 2) / 4 +
                 (*pb++ + *pba++ + *pbb++ + *pbc++ + 2) / 4 + 1) / 2
                - *p2++);

      if (v>=0)
        s+= (float) v;
      else
        s-= (float) v;
    }
    p2+= lx-16;
    pf+= lx-16;
```

```
    pfa+= lx-16;
    pfb+= lx-16;
    pfc+= lx-16;
    pb+= lx-16;
    pba+= lx-16;
    pbb+= lx-16;
    pbc+= lx-16;
   }

  return s;
 }

static float bdist2(char *pf, char *pb, char *p2,
                    int lx, int hxf,
                    int hyf, int hxb,
                    int hyb, char h )
{
  char *pfa,*pfb,*pfc,*pba,*pbb,*pbc;
  char i, j;
  int v;
  float s;

  pfa = pf + hxf;
  pfb = pf + lx*hyf;
  pfc = pfb + hxf;

  pba = pb + hxb;
  pbb = pb + lx*hyb;
  pbc = pbb + hxb;

  s = 0;

  for (j=0; j<h; j++)
  {
   for (i=0; i<16; i++)
   {
    v = (int) (((*pf++ + *pfa++ + *pfb++ + *pfc++ + 2) / 4 +
               (*pb++ + *pba++ + *pbb++ + *pbc++ + 2) / 4 + 1) / 2
               - *p2++);
    s+= (float) (v*v);
   }
   p2+= lx-16;
   pf+= lx-16;
   pfa+= lx-16;
   pfb+= lx-16;
   pfc+= lx-16;
   pb+= lx-16;
   pba+= lx-16;
   pbb+= lx-16;
   pbc+= lx-16;
  }

  return s,
}
```

```
static float variance(char *p, int lx )
{
 char i, j;
 int v;
 float s, s2;

 s = s2 = 0;

 for (j=0; j<16; j++)
 {
  for (i=0; i<16; i++)
  {
   v = *p++;
   s+= (float) v;
   s2+= (float) (v*v);
  }
  p+= lx-16;
 }
 s=0.0625*s;
 return s2 - s*s;
}
```

```
/********************************/
/*                              */
/*      m01.c reside at PE1     */
/*                              */
/********************************/

#include "c:\tc2v2\dos.h"
#include "c:\tc2v2\float.h"
#include "c:\tc2v2\math.h"
#include "c:\tc2v2\ctype.h"
#include "c:\tc2v2\stdio.h"
#include "c:\tc2v2\stdlib.h"
#include "c:\tc2v2\string.h"
#include "c:\tc2v2\chan.h"
#include "c:\tc2v2\thread.h"
#include "c:\tc2v2\time.h"
#include "c:\tc2v2\errno.h"

#define GLOBAL /* used by global.h */

#include "c:\nick\h\const1.h"
#include "c:\nick\h\config.h"
#include "c:\nick\h\global.h"

static void init _ANSI_ARGS_((void));

char    *neworg[1];

void main(argc,argv,envp,ins,in_ports,outs,out_ports)
int argc,ins,outs;
char *argv[],*envp[];
CHAN *in_ports[],*out_ports[];
{
 /* read parameter file */
 chan_in_word(&M,Link0Input);
 chan_out_word(M,Link1Output);
 chan_out_word(M,Link2Output);

 chan_in_word(&nframes,Link0Input);
 chan_out_word(nframes,Link1Output);
 chan_out_word(nframes,Link2Output);

 init();
 putseq();

}

static void init()
{
 int i, size, a;

 chan_in_word(&mb_width,Link0Input);
 chan_out_word(mb_width,Link1Output);
 chan_out_word(mb_width,Link2Output);
```

```
 chan_in_word(&mb_height,Link0Input);
 chan_out_word(mb_height,Link1Output);
 chan_out_word(mb_height,Link2Output);

 chan_in_word(&mb_height2,Link0Input);
 chan_out_word(mb_height2,Link1Output);
 chan_out_word(mb_height2,Link2Output);

 chan_in_word(&width,Link0Input);
 chan_out_word(width,Link1Output);
 chan_out_word(width,Link2Output);

 chan_in_word(&height,Link0Input);
 chan_out_word(height,Link1Output);
 chan_out_word(height,Link2Output),

 chan_in_word(&width2,Link0Input);
 chan_out_word(width2,Link1Output);
 chan_out_word(width2,Link2Output);

 chan_in_word(&height2,Link0Input);
 chan_out_word(height2,Link1Output);
 chan_out_word(height2,Link2Output);

 size = width*height ;

 newrefframe[0] = (char *)malloc(size);
 oldrefframe[0] = (char *)malloc(size);
 neworgframe[0] = (char *)malloc(size);
 oldorgframe[0] = (char *)malloc(size);
 neworg[0] = (char *)malloc(size);

 mbinfo = (struct mbinfo *)malloc(110*sizeof(struct mbinfo));

 for (a=0; a<110; a++)
 {
  mbinfo[a].mb_type = mbinfo[a].motion_type = 0;
  mbinfo[a].MV[0][0][0] = mbinfo[a].MV[0][0][1] = mbinfo[a].MV[0][1][0] = 0;
  mbinfo[a].MV[0][1][1] = mbinfo[a].MV[1][0][0] = mbinfo[a].MV[1][0][1] = 0;
  mbinfo[a].MV[1][1][0] = mbinfo[a].MV[1][1][1] = 0;
  mbinfo[a].mv_field_sel[0][0] = mbinfo[a].mv_field_sel[0][1] = 0;
  mbinfo[a].mv_field_sel[1][0] = mbinfo[a].mv_field_sel[1][1] = 0;
 }


}


void putseq()
{
/* this routine assumes (N % M) == 0 */
 int i, j, k, f, f0, n, np, nb;
 int a;
```

```
int sxf, syf, sxb, syb;
int size, loop;

size = width*height;


/* loop through all frames in encoding/decoding order */
do{
   chan_in_word(&loop,Link0Input);
   chan_out_word(loop,Link1Output);
   chan_out_word(loop,Link2Output);

   if(loop==nframes) break;

   chan_in_word(&pict_type,Link0Input);
   chan_out_word(pict_type,Link1Output),
   chan_out_word(pict_type,Link2Output);

   chan_in_word(&sxf,Link0Input);
   chan_out_word(sxf,Link1Output),
   chan_out_word(sxf,Link2Output);

   chan_in_word(&syf,Link0Input);
   chan_out_word(syf,Link1Output);
   chan_out_word(syf,Link2Output);

   chan_in_word(&sxb,Link0Input);
   chan_out_word(sxb,Link1Output);
   chan_out_word(sxb,Link2Output);

   chan_in_word(&syb,Link0Input);
   chan_out_word(syb,Link1Output);
   chan_out_word(syb,Link2Output);

   if ( pict_type != I_TYPE )
   {
    chan_in_message(size,neworg[0],Link0Input);
    chan_out_message(size,neworg[0],Link1Output);
    chan_out_message(size,neworg[0],Link2Output);

    chan_in_message(size,oldorgframe[0],Link0Input);
    chan_out_message(size,oldorgframe[0],Link1Output);
    chan_out_message(size,oldorgframe[0],Link2Output);

    chan_in_message(size,oldrefframe[0],Link0Input);
    chan_out_message(size,oldrefframe[0],Link1Output);
    chan_out_message(size,oldrefframe[0],Link2Output);

    if ( pict_type != P_TYPE )
    {
     chan_in_message(size,neworgframe[0],Link0Input);
     chan_out_message(size,neworgframe[0],Link1Output);
     chan_out_message(size,neworgframe[0],Link2Output),
```

```
  chan_in_message(size,newrefframe[0],Link0Input);
  chan_out_message(size,newrefframe[0],Link1Output),
  chan_out_message(size,newrefframe[0],Link2Output);
 }
 }

/* loop through 36 macroblocks of the picture */

 motion_estimation(oldorgframe[0], neworgframe[0],
          oldrefframe[0], newrefframe[0],
          neworg[0],
          sxf, syf,
          sxb, syb,
          mbinfo);

  for (a=36; a<73; a++)
  {
   chan_in_word(&mbinfo[a].mb_type,Link1Input);
   chan_in_word(&mbinfo[a].motion_type,Link1Input);
   chan_in_word(&mbinfo[a].MV[0][0][0],Link1Input);
   chan_in_word(&mbinfo[a].MV[0][0][1],Link1Input);
   chan_in_word(&mbinfo[a].MV[0][1][0],Link1Input);
   chan_in_word(&mbinfo[a].MV[0][1][1],Link1Input);
   chan_in_word(&mbinfo[a].MV[1][0][0],Link1Input);
   chan_in_word(&mbinfo[a].MV[1][0][1],Link1Input);
   chan_in_word(&mbinfo[a].MV[1][1][0],Link1Input);
   chan_in_word(&mbinfo[a].MV[1][1][1],Link1Input);
   chan_in_word(&mbinfo[a].mv_field_sel[0][0],Link1Input);
   chan_in_word(&mbinfo[a].mv_field_sel[0][1],Link1Input);
   chan_in_word(&mbinfo[a].mv_field_sel[1][0],Link1Input);
   chan_in_word(&mbinfo[a].mv_field_sel[1][1],Link1Input);
  }

  for (a=73; a<110; a++)
  {
   chan_in_word(&mbinfo[a].mb_type,Link2Input);
   chan_in_word(&mbinfo[a].motion_type,Link2Input);
   chan_in_word(&mbinfo[a].MV[0][0][0],Link2Input);
   chan_in_word(&mbinfo[a].MV[0][0][1],Link2Input);
   chan_in_word(&mbinfo[a].MV[0][1][0],Link2Input);
   chan_in_word(&mbinfo[a].MV[0][1][1],Link2Input);
   chan_in_word(&mbinfo[a].MV[1][0][0],Link2Input);
   chan_in_word(&mbinfo[a].MV[1][0][1],Link2Input);
   chan_in_word(&mbinfo[a].MV[1][1][0],Link2Input);
   chan_in_word(&mbinfo[a].MV[1][1][1],Link2Input);
   chan_in_word(&mbinfo[a].mv_field_sel[0][0],Link2Input);
   chan_in_word(&mbinfo[a].mv_field_sel[0][1],Link2Input);
   chan_in_word(&mbinfo[a].mv_field_sel[1][0],Link2Input);
   chan_in_word(&mbinfo[a].mv_field_sel[1][1],Link2Input);
  }


  for (a=0; a<110; a++)
  {
```

```
      chan_out_word(mbinfo[a].mb_type,Link0Output);
      chan_out_word(mbinfo[a] motion_type,Link0Output);
      chan_out_word(mbinfo[a].MV[0][0][0],Link0Output);
      chan_out_word(mbinfo[a] MV[0][0][1],Link0Output);
      chan_out_word(mbinfo[a].MV[0][1][0],Link0Output);
      chan_out_word(mbinfo[a].MV[0][1][1],Link0Output);
      chan_out_word(mbinfo[a] MV[1][0][0],Link0Output);
      chan_out_word(mbinfo[a].MV[1][0][1],Link0Output);
      chan_out_word(mbinfo[a].MV[1][1][0],Link0Output);
      chan_out_word(mbinfo[a] MV[1][1][1],Link0Output),
      chan_out_word(mbinfo[a] mv_field_sel[0][0],Link0Output);
      chan_out_word(mbinfo[a].mv_field_sel[0][1],Link0Output);
      chan_out_word(mbinfo[a].mv_field_sel[1][0],Link0Output);
      chan_out_word(mbinfo[a].mv_field_sel[1][1],Link0Output);
     }

  }while(loop<nframes),

  chan_in_word(&loop,Link1Input);
  chan_in_word(&loop,Link2Input);
  chan_out_word(loop,Link0Output);

}/*end of main()*/


static void frame_estimate _ANSI_ARGS_((char *org,
                              char *ref,
                              char *mb,
                              int i, int j,
                              int sx, int sy,
                              int *iminp, int *jminp,
                              int *imintp, int *jmintp,
                              int *iminbp, int *jminbp,
                              float *dframep, float *dfieldp,
                              int *tselp, int *bselp,
                              int imins[2][2], int jmins[2][2]));

static float fullsearch _ANSI_ARGS_((char *org,
                            char *ref,
                            char *blk,
                            int lx,
                            int i0,
                            int j0,
                            int sx,
                            int sy,
                            char h,
                            int xmax,
                            int ymax,
                            int *iminp,
                            int *jminp));

static float dist1 _ANSI_ARGS_((char *blk1,
                        char *blk2,
                        int lx,
```

```
                              char hx,
                              char hy,
                              char h,
                              float distlim));

static float dist2 _ANSI_ARGS_((char *blk1,
                              char *blk2,
                              int lx,
                              char hx,
                              char hy,
                              char h));

static float bdist1 _ANSI_ARGS_((char *pf,
                               char *pb,
                               char *p2,
                               int lx,
                               int hxf,
                               int hyf,
                               int hxb,
                               int hyb,
                               char h));

static float bdist2 _ANSI_ARGS_((char *pf,
                               char *pb,
                               char *p2,
                               int lx,
                               int hxf,
                               int hyf,
                               int hxb,
                               int hyb,
                               char h));

static float variance _ANSI_ARGS_((char *p, int lx));


void motion_estimation(char *oldorg, char *neworg,
                    char *oldref, char *newref,
                    char *cur,
                    int sxf,
                    int syf,
                    int sxb,
                    int syb,
                    struct mbinfo *mbi)
{
  int i, j;
  int imin,jmin,iminf,jminf,iminr,jminr;
  int imint,jmint,iminb,jminb;
  int imintf,jmintf,iminbf,jminbf,
  int imintr,jmintr,iminbr,jminbr;
  float var,v0;
  float dmc,dmcf,dmcr,dmci,vmc,vmcf,vmcr,vmci;
  float dmcfield,dmcfieldf,dmcfieldr,dmcfieldi;
  int tsel,bsel,tself,bself,tselr,bselr;
  char *mb;
```

```
  int imins[2][2],jmins[2][2];
  int a, ii, jj;

/* loop through 36 macroblocks of the picture */

  for(a=36; a<72; a++)
  {
   jj = a>>4;
   ii = a - (jj<<4);
   j = jj<<4;
   i = ii<<4;

   mb = cur + i + width*j;

   var = variance(mb,width);

   /*  for I_type pictures     */

   if (pict_type==I_TYPE)
   mbi->mb_type = MB_INTRA;

   /*  for P_type pictures     */

   if (pict_type==P_TYPE)
   {
    frame_estimate(oldorg,oldref,mb,i,j,sxf,syf,
                &imin,&jmin,&imint,&jmint,&iminb,&jminb,
                &dmc,&dmcfield,&tsel,&bsel,imins,jmins);

    /* select between frame and field prediction */
    if (dmc<=dmcfield)
    {
     mbi->motion_type = MC_FRAME;
     vmc = dist2(oldref+(imin>>1)+width*(jmin>>1),mb,
             width,imin&1,jmin&1,16);
    }
    else
    {
     mbi->motion_type = MC_FIELD;
     dmc = dmcfield;
     vmc = dist2(oldref+(tsel?width:0)+(imint>>1)+(width<<1)*(jmint>>1),
             mb,width<<1,imint&1,jmint&1,8);
     vmc+= dist2(oldref+(bsel?width:0)+(iminb>>1)+(width<<1)*(jminb>>1),
             mb+width,width<<1,iminb&1,jminb&1,8);
    }


    if (vmc>var && vmc>=9*256)
    mbi->mb_type = MB_INTRA;
    else
    {
     v0 = dist2(oldref+i+width*j,mb,width,0,0,16);
     if (4*v0>5*vmc && v0>=9*256)
     {
```

```
    /* use MC */
    var = vmc;
    mbi->mb_type = MB_FORWARD;
    if (mbi->motion_type==MC_FRAME)
    {
     mbi->MV[0][0][0] = imin - (i<<1);
     mbi->MV[0][0][1] = jmin - (j<<1);
    }
    else /* if (mbi->motion_type==MC_FIELD) */
    {
     /* these are FRAME vectors */
     mbi->MV[0][0][0] = imint - (i<<1);
     mbi->MV[0][0][1] = (jmint<<1) - (j<<1);
     mbi->MV[1][0][0] = iminb - (i<<1);
     mbi->MV[1][0][1] = (jminb<<1) - (j<<1);
     mbi->mv_field_sel[0][0] = tsel;
     mbi->mv_field_sel[1][0] = bsel;
    }
   }
   else
   {
    /* No-MC */
    var = v0;
    mbi->mb_type = 0;
    mbi->motion_type = MC_FRAME;
    mbi->MV[0][0][0] = 0;
    mbi->MV[0][0][1] = 0;
   }
  }
 }


/*  for B_type pictures     */

if (pict_type==B_TYPE)
{
 /* forward prediction */
 frame_estimate(oldorg,oldref,mb,i,j,sxf,syf,
          &iminf,&jminf,&imintf,&jmintf,&iminbf,&jminbf,
          &dmcf,&dmcfieldf,&tself,&bself,imins,jmins);

/* backward prediction */
frame_estimate(neworg,newref,mb,i,j,sxb,syb,
          &iminr,&jminr,&imintr,&jmintr,&iminbr,&jminbr,
          &dmcr,&dmcfieldr,&tselr,&bselr,imins,jmins);

/* calculate interpolated distance */
/* frame */
dmci = bdist1(oldref+(iminf>>1)+width*(jminf>>1),
         newref+(iminr>>1)+width*(jminr>>1),
         mb,width,iminf&1,jminf&1,iminr&1,jminr&1,16);

/* top field
dmcfieldi = bdist1(oldref+(imintf>>1)+(tself?width:0)+(width<<1)*(jmintf>>1),
```

```
            newref+(imintr>>1)+(tselr?width:0)+(width<<1)*(jmintr>>1),
            mb,width<<1,imintf&1,jmintf&1,imintr&1,jmintr&1,8);

/* bottom field
dmcfieldi+= bdist1(oldref+(iminbf>>1)+(bself?width:0)+(width<<1)*(jminbf>>1),
            newref+(iminbr>>1)+(bselr?width:0)+(width<<1)*(jminbr>>1),
            mb+width,width<<1,iminbf&1,jminbf&1,iminbr&1,jminbr&1,8),

if (dmci<dmcfieldi && dmci<dmcf && dmci<dmcfieldf
   && dmci<dmcr && dmci<dmcfieldr)
{
 /* frame, interpolated */
 mbi->mb_type = MB_FORWARD|MB_BACKWARD;
 mbi->motion_type = MC_FRAME;
 vmc = bdist2(oldref+(iminf>>1)+width*(jminf>>1),
          newref+(iminr>>1)+width*(jminr>>1),
          mb,width,iminf&1,jminf&1,iminr&1,jminr&1,16);
}
else if (dmcfieldi<dmcf && dmcfieldi<dmcfieldf
       && dmcfieldi<dmcr && dmcfieldi<dmcfieldr)
{
 /* field, interpolated */
 mbi->mb_type = MB_FORWARD|MB_BACKWARD;
 mbi->motion_type = MC_FIELD;
 vmc = bdist2(oldref+(imintf>>1)+(tself?width:0)+(width<<1)*(jmintf>>1),
          newref+(imintr>>1)+(tselr?width:0)+(width<<1)*(jmintr>>1),
          mb,width<<1,imintf&1,jmintf&1,imintr&1,jmintr&1,8),
 vmc+= bdist2(oldref+(iminbf>>1)+(bself?width:0)+(width<<1)*(jminbf>>1),
          newref+(iminbr>>1)+(bselr?width:0)+(width<<1)*(jminbr>>1),
          mb+width,width<<1,iminbf&1,jminbf&1,iminbr&1,jminbr&1,8);
}
else if (dmcf<dmcfieldf && dmcf<dmcr && dmcf<dmcfieldr)
{
 /* frame, forward */
 mbi->mb_type = MB_FORWARD;
 mbi->motion_type = MC_FRAME;
 vmc = dist2(oldref+(iminf>>1)+width*(jminf>>1),mb,
         width,iminf&1,jminf&1,16);
}
else if (dmcfieldf<dmcr && dmcfieldf<dmcfieldr)
{
 /* field, forward */
 mbi->mb_type = MB_FORWARD;
 mbi->motion_type = MC_FIELD;
 vmc = dist2(oldref+(tself?width:0)+(imintf>>1)+(width<<1)*(jmintf>>1),
         mb,width<<1,imintf&1,jmintf&1,8);
 vmc+= dist2(oldref+(bself?width:0)+(iminbf>>1)+(width<<1)*(jminbf>>1),
         mb+width,width<<1,iminbf&1,jminbf&1,8);
}
else if (dmcr<dmcfieldr)
{
 /* frame, backward */
 mbi->mb_type = MB_BACKWARD;
 mbi->motion_type = MC_FRAME;
```

```
  vmc = dist2(newref+(iminr>>1)+width*(jminr>>1),mb,
            width,iminr&1,jminr&1,16);
 }
 else
 {
  /* field, backward */
  mbi->mb_type = MB_BACKWARD;
  mbi->motion_type = MC_FIELD;
  vmc = dist2(newref+(tselr?width:0)+(imintr>>1)+(width<<1)*(jmintr>>1),
            mb,width<<1,imintr&1,jmintr&1,8);
  vmc+= dist2(newref+(bselr?width:0)+(iminbr>>1)+(width<<1)*(jminbr>>1),
            mb+width,width<<1,iminbr&1,jminbr&1,8);
 }

 if (vmc>var && vmc>=9*256)
 mbi->mb_type = MB_INTRA;
 else
 {
  var = vmc;
  if (mbi->motion_type==MC_FRAME)
  {
   /* forward */
   mbi->MV[0][0][0] = iminf - (i<<1);
   mbi->MV[0][0][1] = jminf - (j<<1);
   /* backward */
   mbi->MV[0][1][0] = iminr - (i<<1);
   mbi->MV[0][1][1] = jminr - (j<<1);
  }
  else
  {
   /* these are FRAME vectors */
   /* forward */
   mbi->MV[0][0][0] = imintf - (i<<1);
   mbi->MV[0][0][1] = (jmintf<<1) - (j<<1);
   mbi->MV[1][0][0] = iminbf - (i<<1);
   mbi->MV[1][0][1] = (jminbf<<1) - (j<<1);
   mbi->mv_field_sel[0][0] = tself;
   mbi->mv_field_sel[1][0] = bself;
   /* backward */
   mbi->MV[0][1][0] = imintr - (i<<1);
   mbi->MV[0][1][1] = (jmintr<<1) - (j<<1);
   mbi->MV[1][1][0] = iminbr - (i<<1);
   mbi->MV[1][1][1] = (jminbr<<1) - (j<<1);
   mbi->mv_field_sel[0][1] = tselr;
   mbi->mv_field_sel[1][1] = bselr;
  }
 }
}

mbi->var = var;

mbi++;

}
```

```
}

static void frame_estimate(char *org, char *ref,
                    char *mb,
                    int i, int j,
                    int sx, int sy,
                    int *iminp, int *jminp,
                    int *imintp, int *jmintp,
                    int *iminbp, int *jminbp,
                    float *dframep, float *dfieldp,
                    int *tselp, int *bselp,
                    int imins[2][2], int jmins[2][2] )
{
 float dt,db,dmint,dminb;
 int imint,iminb,jmint,jminb;

 /* frame prediction   */
 *dframep = fullsearch(org,ref,mb,width,i,j,sx,sy,16,width,height,
                iminp,jminp);

 /* predict top field from top field   */
 dt = fullsearch(org,ref,mb,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
            &imint,&jmint);

 /* predict top field from bottom field  */
 db = fullsearch(org+width,ref+width,mb,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
            &iminb,&jminb);

 imins[0][0] = imint;
 jmins[0][0] = jmint;
 imins[1][0] = iminb;
 jmins[1][0] = jminb;

 /* select prediction for top field */
 if (dt<=db)
 {
  dmint=dt; *imintp=imint; *jmintp=jmint; *tselp=0;
 }
 else
 {
  dmint=db; *imintp=iminb; *jmintp=jminb; *tselp=1,
 }

 /* predict bottom field from top field   */
 dt = fullsearch(org,ref,mb+width,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
            &imint,&jmint);

 /* predict bottom field from bottom field   */
 db = fullsearch(org+width, ref+width, mb+width,
            width<<1, i, j>>1, sx, sy>>1, 8, width, height>>1,
            &iminb,&jminb);

 imins[0][1] = imint;
```

```
  jmins[0][1] = jmint;
  imins[1][1] = iminb;
  jmins[1][1] = jminb;

  /* select prediction for bottom field */
  if (db<=dt)
  {
    dminb=db; *iminbp=iminb; *jminbp=jminb; *bselp=1;
  }
  else
  {
    dminb=dt; *iminbp=imint; *jminbp=jmint; *bselp=0;
  }

  *dfieldp=dmint+dminb;
}


static float fullsearch(char *org, char *ref,
                        char *blk,
                        int lx,
                        int i0,
                        int j0,
                        int sx,
                        int sy,
                        char h,
                        int xmax,
                        int ymax,
                        int *iminp,
                        int *jminp)
{
  int i,j,imin,jmin,ilow,ihigh,jlow,jhigh;
  float d,dmin;
  int k,l,sxy;
  int ii, jj;

  ilow = i0 - sx;
  ihigh = i0 + sx;

  if (ilow<0) ilow = 0;

  if (ihigh>xmax-16) ihigh = xmax-16;

  jlow = j0 - sy;
  jhigh = j0 + sy;

  if (jlow<0) jlow = 0;

  if (jhigh>ymax-h) jhigh = ymax-h;

  /* full pel search, spiraling outwards */

  imin = i0;
  jmin = j0;
```

```
dmin = dist1(org+imin+lx*jmin,blk,lx,0,0,h,65536.0);

sxy = (sx>sy) ? sx : sy;

for (l=1; l<=sxy; l++)
{
  i = i0 - l;
  j = j0 - l;
  for (k=0; k<8*l; k++)
  {
    if (i>=ilow && i<=ihigh && j>=jlow && j<=jhigh)
    {
      d = dist1(org+i+lx*j,blk,lx,0,0,h,dmin),

      if (d<dmin)
      {
        dmin = d;
        imin = i;
        jmin = j;
      }
    }

    if      (k<2*l) i++;
    else if (k<4*l) j++;
    else if (k<6*l) i--;
    else            j--;
  }
}

/* half pel */
dmin = 65536.0;
imin = imin << 1;
jmin = jmin << 1;
ilow = imin - (imin>0);
ihigh = imin + (imin<((xmax-16)<<1));
jlow = jmin - (jmin>0);
jhigh = jmin + (jmin<((ymax-h)<<1));

for (j=jlow; j<=jhigh; j++)
  for (i=ilow; i<=ihigh; i++)
  {
   ii = i >> 1 ;
   jj = j >> 1 ;
   d = dist1(ref+ii+lx*jj,blk,lx,i&1,j&1,h,dmin);

    if (d<dmin)
    {
      dmin = d;
      imin = i;
      jmin = j;
    }
  }

*iminp = imin;
```

```
  *jminp = jmin;

  return dmin,
}

static float dist1(char *blk1,
                 char *blk2,
                 int lx,
                 char hx,
                 char hy,
                 char h,
                 float distlim )
{
 char *p1,*p1a,*p2;
 char i, j;
 int v;
 float s;

 s = 0;
 p1 = blk1;
 p2 = blk2;

 if (!hx && !hy)
  for (j=0; j<h, j++)
  {
   if ((v = p1[0]  - p2[0])<0)  v = -v; s+= (float) v;
   if ((v = p1[1]  - p2[1])<0)  v = -v; s+= (float) v;
   if ((v = p1[2]  - p2[2])<0)  v = -v; s+= (float) v;
   if ((v = p1[3]  - p2[3])<0)  v = -v; s+= (float) v;
   if ((v = p1[4]  - p2[4])<0)  v = -v; s+= (float) v;
   if ((v = p1[5]  - p2[5])<0)  v = -v; s+= (float) v,
   if ((v = p1[6]  - p2[6])<0)  v = -v; s+= (float) v;
   if ((v = p1[7]  - p2[7])<0)  v = -v; s+= (float) v;
   if ((v = p1[8]  - p2[8])<0)  v = -v; s+= (float) v;
   if ((v = p1[9]  - p2[9])<0)  v = -v; s+= (float) v;
   if ((v = p1[10] - p2[10])<0) v = -v; s+= (float) v;
   if ((v = p1[11] - p2[11])<0) v = -v; s+= (float) v;
   if ((v = p1[12] - p2[12])<0) v = -v; s+= (float) v;
   if ((v = p1[13] - p2[13])<0) v = -v; s+= (float) v;
   if ((v = p1[14] - p2[14])<0) v = -v; s+= (float) v;
   if ((v = p1[15] - p2[15])<0) v = -v, s+= (float) v;

   if (s >= distlim)
    break;

   p1+= lx,
   p2+= lx;
  }
 else if (hx && !hy)
  for (j=0; j<h; j++)
  {
   for (i=0; i<16; i++)
   {
    v = (int) ((p1[i]+p1[i+1]+1) / 2 - p2[i]);
```

```
      if (v>=0) s+= (float) v,
      else s-= (float) v;
     }
     p1+= lx;
     p2+= lx;
    }
  else if (!hx && hy)
  {
    p1a = p1 + lx;
    for (j=0, j<h; j++)
    {
     for (i=0; i<16; i++)
     {
      v = (int ) ((p1[i]+p1a[i]+1) / 2 - p2[i]);
      if (v>=0)
        s+= (float) v;
      else
        s-= (float) v;
     }
     p1 = p1a;
     p1a+= lx;
     p2+= lx;
    }
  }
  else /* if (hx && hy) */
  {
    p1a = p1 + lx,
    for (j=0; j<h; j++)
    {
     for (i=0; i<16; i++)
     {
      v = (int) ((p1[i]+p1[i+1]+p1a[i]+p1a[i+1]+2) / 4 - p2[i]);
      if (v>=0)
        s+= (float) v;
      else
        s-= (float) v;
     }
     p1 = p1a;
     p1a+= lx;
     p2+= lx;
    }
  }

  return s;
}

static float dist2(char *blk1,
                   char *blk2,
                   int lx,
                   char hx,
                   char hy,
                   char h )
{
  char *p1,*p1a,*p2;
```

```
char i, j;
int v;
float s;

s = 0,
p1 = blk1,
p2 = blk2;
if (!hx && !hy)
  for (j=0; j<h; j++)
  {
    for (i=0; i<16; i++)
    {
      v = p1[i] - p2[i];
      s+= (float) (v*v);
    }
    p1+= lx;
    p2+= lx;
  }
else if (hx && !hy)
  for (j=0; j<h; j++)
  {
    for (i=0, i<16; i++)
    {
      v = (int) ((p1[i]+p1[i+1]+1) / 2 - p2[i]);
      s+= (float) (v*v);
    }
    p1+= lx;
    p2+= lx;
  }
else if (!hx && hy)
{
  p1a = p1 + lx;
  for (j=0; j<h; j++)
  {
    for (i=0; i<16; i++)
    {
      v = (int) ((p1[i]+p1a[i]+1) / 2 - p2[i]);
      s+= (float) (v*v);
    }
    p1 = p1a;
    p1a+= lx;
    p2+= lx;
  }
}
else /* if (hx && hy) */
{
  p1a = p1 + lx;
  for (j=0; j<h; j++)
  {
    for (i=0; i<16; i++)
    {
      v = (int) ((p1[i]+p1[i+1]+p1a[i]+p1a[i+1]+2) / 4 - p2[i]);
      s+= (float) (v*v);
    }
```

```
      p1 = p1a;
      p1a+= lx;
      p2+= lx;
    }
  }

  return s;
}

static float bdist1(char *pf, char *pb, char *p2,
                    int lx, int hxf,
                    int hyf, int hxb,
                    int hyb, char h )
{
  char *pfa,*pfb,*pfc,*pba,*pbb,*pbc;
  char i, j;
  int v;
  float s;

  pfa = pf + hxf;
  pfb = pf + lx*hyf;
  pfc = pfb + hxf;

  pba = pb + hxb;
  pbb = pb + lx*hyb;
  pbc = pbb + hxb;

  s = 0;

  for (j=0; j<h; j++)
  {
    for (i=0; i<16; i++)
    {
      v = (int) (((*pf++ + *pfa++ + *pfb++ + *pfc++ + 2) / 4 +
                 (*pb++ + *pba++ + *pbb++ + *pbc++ + 2) / 4 + 1) / 2
                - *p2++);

      if (v>=0)
        s+= (float) v;
      else
        s-= (float) v;
    }
    p2+= lx-16;
    pf+= lx-16;
    pfa+= lx-16;
    pfb+= lx-16;
    pfc+= lx-16;
    pb+= lx-16;
    pba+= lx-16;
    pbb+= lx-16;
    pbc+= lx-16;
  }

  return s;
```

```
}

static float bdist2(char *pf, char *pb, char *p2,
                  int lx, int hxf,
                  int hyf, int hxb,
                  int hyb, char h )
{
  char *pfa,*pfb,*pfc,*pba,*pbb,*pbc;
  char i, j;
  int v;
  float s;

  pfa = pf + hxf;
  pfb = pf + lx*hyf;
  pfc = pfb + hxf;

  pba = pb + hxb;
  pbb = pb + lx*hyb,
  pbc = pbb + hxb;

  s = 0;

  for (j=0, j<h; j++)
  {
    for (i=0; i<16; i++)
    {
      v = (int) (((*pf++ + *pfa++ + *pfb++ + *pfc++ + 2) / 4 +
              (*pb++ + *pba++ + *pbb++ + *pbc++ + 2) / 4 + 1) / 2
             - *p2++);
      s+= (float) (v*v);
    }
    p2+= lx-16;
    pf+= lx-16;
    pfa+= lx-16;
    pfb+= lx-16;
    pfc+= lx-16;
    pb+= lx-16;
    pba+= lx-16;
    pbb+= lx-16;
    pbc+= lx-16;
  }

  return s;
}

static float variance(char *p, int lx )
{
  char i, j;
  int v;
  float s, s2;

  s = s2 = 0;

  for (j=0; j<16; j++)
```

```
 {
  for (i=0; i<16; i++)
  {
   v = *p++;
   s+= (float) v;
   s2+= (float) (v*v);
  }
  p+= lx-16;
 }
 s=0.0625*s;
 return s2 - s*s;
}
```

```
/********************************/
/*                              */
/*   m011.c residing at PE2     */
/*                              */
/********************************/

#include "c:\tc2v2\dos.h"
#include "c:\tc2v2\float.h"
#include "c:\tc2v2\math.h"
#include "c:\tc2v2\ctype.h"
#include "c.\tc2v2\stdio.h"
#include "c:\tc2v2\stdlib.h"
#include "c:\tc2v2\string.h"
#include "c:\tc2v2\chan.h"
#include "c:\tc2v2\thread.h"
#include "c:\tc2v2\time.h"
#include "c:\tc2v2\errno.h"

#define GLOBAL /* used by global.h */

#include "c:\nick\h\const1.h"
#include "c:\nick\h\config.h"
#include "c:\nick\h\global.h"

static void init _ANSI_ARGS_((void));

char  *neworg[1];

void main(argc,argv,envp,ins,in_ports,outs,out_ports)
int argc,ins,outs;
char *argv[],*envp[];
CHAN *in_ports[],*out_ports[];
{
 /* read parameter file */
 chan_in_word(&M,Link0Input);
 chan_in_word(&nframes,Link0Input);

 init();
 putseq();

}

static void init()
{
 int i, size, a;

 chan_in_word(&mb_width,Link0Input);
 chan_in_word(&mb_height,Link0Input);
 chan_in_word(&mb_height2,Link0Input);
 chan_in_word(&width,Link0Input);
 chan_in_word(&height,Link0Input);
 chan_in_word(&width2,Link0Input);
 chan_in_word(&height2,Link0Input);
```

```
  size = width*height ;

  newrefframe[0] = (char *)malloc(size);
  oldrefframe[0] = (char *)malloc(size);
  neworgframe[0] = (char *)malloc(size);
  oldorgframe[0] = (char *)malloc(size);
  neworg[0] = (char *)malloc(size);

  mbinfo = (struct mbinfo *)malloc(37*sizeof(struct mbinfo));

  for (a=0; a<37; a++)
  {
   mbinfo[a].mb_type = mbinfo[a].motion_type = 0;
   mbinfo[a].MV[0][0][0] = mbinfo[a].MV[0][0][1] = mbinfo[a].MV[0][1][0] = 0,
   mbinfo[a].MV[0][1][1] = mbinfo[a].MV[1][0][0] = mbinfo[a].MV[1][0][1] = 0;
   mbinfo[a].MV[1][1][0] = mbinfo[a].MV[1][1][1] = 0;
   mbinfo[a].mv_field_sel[0][0] = mbinfo[a].mv_field_sel[0][1] = 0;
   mbinfo[a].mv_field_sel[1][0] = mbinfo[a].mv_field_sel[1][1] = 0;
  }

}


void putseq()
{
/* this routine assumes (N % M) == 0 */
  int i, j, k, f, f0, n, np, nb;
  int a;
  int sxf, syf, sxb, syb;
  int size, loop;

  size = width*height;


/* loop through all frames in encoding/decoding order */
do{
   chan_in_word(&loop,Link0Input);

   if(loop==nframes) break,

   chan_in_word(&pict_type,Link0Input);
   chan_in_word(&sxf,Link0Input);
   chan_in_word(&syf,Link0Input);
   chan_in_word(&sxb,Link0Input);
   chan_in_word(&syb,Link0Input);

   if ( pict_type != I_TYPE )
   {
    chan_in_message(size,neworg[0],Link0Input);
    chan_in_message(size,oldorgframe[0],Link0Input);
    chan_in_message(size,oldrefframe[0],Link0Input);
    if ( pict_type != P_TYPE )
    {
     chan_in_message(size,neworgframe[0],Link0Input);
```

```
     chan_in_message(size,newrefframe[0],Link0Input);
    }
   }

/* loop through 37 macroblocks of the picture */

   motion_estimation(oldorgframe[0], neworgframe[0],
                oldrefframe[0], newrefframe[0],
                neworg[0],
                sxf, syf,
                sxb, syb,
                mbinfo);


    for (a=0; a<37; a++)
    {
     chan_out_word(mbinfo[a].mb_type,Link0Output);
     chan_out_word(mbinfo[a].motion_type,Link0Output);
     chan_out_word(mbinfo[a].MV[0][0][0],Link0Output);
     chan_out_word(mbinfo[a].MV[0][0][1],Link0Output);
     chan_out_word(mbinfo[a].MV[0][1][0],Link0Output);
     chan_out_word(mbinfo[a].MV[0][1][1],Link0Output);
     chan_out_word(mbinfo[a].MV[1][0][0],Link0Output);
     chan_out_word(mbinfo[a].MV[1][0][1],Link0Output);
     chan_out_word(mbinfo[a].MV[1][1][0],Link0Output);
     chan_out_word(mbinfo[a].MV[1][1][1],Link0Output);
     chan_out_word(mbinfo[a].mv_field_sel[0][0],Link0Output);
     chan_out_word(mbinfo[a].mv_field_sel[0][1],Link0Output);
     chan_out_word(mbinfo[a].mv_field_sel[1][0],Link0Output);
     chan_out_word(mbinfo[a].mv_field_sel[1][1],Link0Output);
    }

  }while(loop<nframes);

  chan_out_word(loop,Link0Output);

}

/* motion.c, motion estimation                                  */

/* private prototypes */

static void frame_estimate _ANSI_ARGS_((char *org,
                              char *ref,
                              char *mb,
                              int i, int j,
                              int sx, int sy,
                              int *iminp, int *jminp,
                              int *imintp, int *jmintp,
                              int *iminbp, int *jminbp,
                              float *dframep, float *dfieldp,
                              int *tselp, int *bselp,
                              int imins[2][2], int jmins[2][2]));
```

```
static float fullsearch _ANSI_ARGS_((char *org,
                                char *ref,
                                char *blk,
                                int lx,
                                int i0,
                                int j0,
                                int sx,
                                int sy,
                                char h,
                                int xmax,
                                int ymax,
                                int *iminp,
                                int *jminp));

static float dist1 _ANSI_ARGS_((char *blk1,
                            char *blk2,
                            int lx,
                            char hx,
                            char hy,
                            char h,
                            float distlim));

static float dist2 _ANSI_ARGS_((char *blk1,
                            char *blk2,
                            int lx,
                            char hx,
                            char hy,
                            char h));

static float bdist1 _ANSI_ARGS_((char *pf,
                             char *pb,
                             char *p2,
                             int lx,
                             int hxf,
                             int hyf,
                             int hxb,
                             int hyb,
                             char h)),

static float bdist2 _ANSI_ARGS_((char *pf,
                             char *pb,
                             char *p2,
                             int lx,
                             int hxf,
                             int hyf,
                             int hxb,
                             int hyb,
                             char h));

static float variance _ANSI_ARGS_((char *p, int lx));


void motion_estimation(char *oldorg, char *neworg,
                char *oldref, char *newref,
```

```
                    char *cur,
                    int sxf,
                    int syf,
                    int sxb,
                    int syb,
                    struct mbinfo *mbi)
{
 int i, j;
 int imin,jmin,iminf,jminf,iminr,jminr;
 int imint,jmint,iminb,jminb;
 int imintf,jmintf,iminbf,jminbf;
 int imintr,jmintr,iminbr,jminbr;
 float var,v0;
 float dmc,dmcf,dmcr,dmci,vmc,vmcf,vmcr,vmci;
 float dmcfield,dmcfieldf,dmcfieldr,dmcfieldi;
 int tsel,bsel,tself,bself,tselr,bselr;
 char *mb;
 int imins[2][2],jmins[2][2];
 int a, ii, jj;

 /* loop through all macroblocks of the picture */

 for(a=72; a<109; a++)
 {
  jj = a>>4;
  ii = a - (jj<<4);
  j = jj<<4;
  i = ii<<4;

  mb = cur + i + width*j;

  var = variance(mb,width);

  /*  for I_type pictures     */

  if (pict_type==I_TYPE)
  mbi->mb_type = MB_INTRA;

  /*  for P_type pictures     */

  if (pict_type==P_TYPE)
  {
   frame_estimate(oldorg,oldref,mb,i,j,sxf,syf,
              &imin,&jmin,&imint,&jmint,&iminb,&jminb,
              &dmc,&dmcfield,&tsel,&bsel,imins,jmins),

   /* select between frame and field prediction */
   if (dmc<=dmcfield)
   {
    mbi->motion_type = MC_FRAME;
    vmc = dist2(oldref+(imin>>1)+width*(jmin>>1),mb,
           width,imin&1,jmin&1,16);
   }
   else
```

```
 {
  mbi->motion_type = MC_FIELD;
  dmc = dmcfield;
  vmc = dist2(oldref+(tsel?width:0)+(imint>>1)+(width<<1)*(jmint>>1),
          mb,width<<1,imint&1,jmint&1,8);
  vmc+= dist2(oldref+(bsel?width:0)+(iminb>>1)+(width<<1)*(jminb>>1),
          mb+width,width<<1,iminb&1,jminb&1,8);
 }

 if (vmc>var && vmc>=9*256)
 mbi->mb_type = MB_INTRA;
 else
 {
  v0 = dist2(oldref+i+width*j,mb,width,0,0,16);
  if (4*v0>5*vmc && v0>=9*256)
  {
   /* use MC */
   var = vmc;
   mbi->mb_type = MB_FORWARD;
   if (mbi->motion_type==MC_FRAME)
   {
    mbi->MV[0][0][0] = imin - (i<<1);
    mbi->MV[0][0][1] = jmin - (j<<1);
   }
   else /* if (mbi->motion_type==MC_FIELD) */
   {
    /* these are FRAME vectors */
    mbi->MV[0][0][0] = imint - (i<<1);
    mbi->MV[0][0][1] = (jmint<<1) - (j<<1);
    mbi->MV[1][0][0] = iminb - (i<<1);
    mbi->MV[1][0][1] = (jminb<<1) - (j<<1);
    mbi->mv_field_sel[0][0] = tsel;
    mbi->mv_field_sel[1][0] = bsel;
   }
  }
  else
  {
   /* No-MC */
   var = v0;
   mbi->mb_type = 0;
   mbi->motion_type = MC_FRAME;
   mbi->MV[0][0][0] = 0;
   mbi->MV[0][0][1] = 0;
  }
 }
}


/*  for B_type pictures    */

if (pict_type==B_TYPE)
{
 /* forward prediction */
 frame_estimate(oldorg,oldref,mb,i,j,sxf,syf,
```

```
        &iminf,&jminf,&imintf,&jmintf,&iminbf,&jminbf,
        &dmcf,&dmcfieldf,&tself,&bself,imins,jmins);

/* backward prediction */
frame_estimate(neworg,newref,mb,i,j,sxb,syb,
        &iminr,&jminr,&imintr,&jmintr,&iminbr,&jminbr,
        &dmcr,&dmcfieldr,&tselr,&bselr,imins,jmins);

/* calculate interpolated distance */
/* frame */
dmci = bdist1(oldref+(iminf>>1)+width*(jminf>>1),
       newref+(iminr>>1)+width*(jminr>>1),
       mb,width,iminf&1,jminf&1,iminr&1,jminr&1,16);

/* top field
dmcfieldi = bdist1(oldref+(imintf>>1)+(tself?width:0)+(width<<1)*(jmintf>>1),
           newref+(imintr>>1)+(tselr?width:0)+(width<<1)*(jmintr>>1),
           mb,width<<1,imintf&1,jmintf&1,imintr&1,jmintr&1,8);

/* bottom field
dmcfieldi+= bdist1(oldref+(iminbf>>1)+(bself?width:0)+(width<<1)*(jminbf>>1),
           newref+(iminbr>>1)+(bselr?width:0)+(width<<1)*(jminbr>>1),
           mb+width,width<<1,iminbf&1,jminbf&1,iminbr&1,jminbr&1,8);

if (dmci<dmcfieldi && dmci<dmcf && dmci<dmcfieldf
  && dmci<dmcr && dmci<dmcfieldr)
{
 /* frame, interpolated */
 mbi->mb_type = MB_FORWARD|MB_BACKWARD;
 mbi->motion_type = MC_FRAME;
 vmc = bdist2(oldref+(iminf>>1)+width*(jminf>>1),
       newref+(iminr>>1)+width*(jminr>>1),
       mb,width,iminf&1,jminf&1,iminr&1,jminr&1,16);
}
else if (dmcfieldi<dmcf && dmcfieldi<dmcfieldf
     && dmcfieldi<dmcr && dmcfieldi<dmcfieldr)
{
 /* field, interpolated */
 mbi->mb_type = MB_FORWARD|MB_BACKWARD;
 mbi->motion_type = MC_FIELD;
 vmc = bdist2(oldref+(imintf>>1)+(tself?width:0)+(width<<1)*(jmintf>>1),
       newref+(imintr>>1)+(tselr?width:0)+(width<<1)*(jmintr>>1),
       mb,width<<1,imintf&1,jmintf&1,imintr&1,jmintr&1,8);
 vmc+= bdist2(oldref+(iminbf>>1)+(bself?width:0)+(width<<1)*(jminbf>>1),
       newref+(iminbr>>1)+(bselr?width:0)+(width<<1)*(jminbr>>1),
       mb+width,width<<1,iminbf&1,jminbf&1,iminbr&1,jminbr&1,8);
}
else if (dmcf<dmcfieldf && dmcf<dmcr && dmcf<dmcfieldr)
{
 /* frame, forward */
 mbi->mb_type = MB_FORWARD;
 mbi->motion_type = MC_FRAME;
 vmc = dist2(oldref+(iminf>>1)+width*(jminf>>1),mb,
      width,iminf&1,jminf&1,16);
```

```
}
else if (dmcfieldf<dmcr && dmcfieldf<dmcfieldr)
{
 /* field, forward */
 mbi->mb_type = MB_FORWARD,
 mbi->motion_type = MC_FIELD;
 vmc = dist2(oldref+(tself?width:0)+(imintf>>1)+(width<<1)*(jmintf>>1),
          mb,width<<1,imintf&1,jmintf&1,8);
 vmc+= dist2(oldref+(bself?width:0)+(iminbf>>1)+(width<<1)*(jminbf>>1),
          mb+width,width<<1,iminbf&1,jminbf&1,8);
}
else if (dmcr<dmcfieldr)
{
 /* frame, backward */
 mbi->mb_type = MB_BACKWARD;
 mbi->motion_type = MC_FRAME;
 vmc = dist2(newref+(iminr>>1)+width*(jminr>>1),mb,
          width,iminr&1,jminr&1,16);
}
else
{
 /* field, backward */
 mbi->mb_type = MB_BACKWARD,
 mbi->motion_type = MC_FIELD;
 vmc = dist2(newref+(tselr?width:0)+(imintr>>1)+(width<<1)*(jmintr>>1),
          mb,width<<1,imintr&1,jmintr&1,8);
 vmc+= dist2(newref+(bselr?width 0)+(iminbr>>1)+(width<<1)*(jminbr>>1),
          mb+width,width<<1,iminbr&1,jminbr&1,8);
}

if (vmc>var && vmc>=9*256)
mbi->mb_type = MB_INTRA;
else
{
 var = vmc;
 if (mbi->motion_type==MC_FRAME)
 {
  /* forward */
  mbi->MV[0][0][0] = iminf - (i<<1);
  mbi->MV[0][0][1] = jminf - (j<<1);
  /* backward */
  mbi->MV[0][1][0] = iminr - (i<<1);
  mbi->MV[0][1][1] = jminr - (j<<1);
 }
 else
 {
  /* these are FRAME vectors */
  /* forward */
  mbi->MV[0][0][0] = imintf - (i<<1);
  mbi->MV[0][0][1] = (jmintf<<1) - (j<<1);
  mbi->MV[1][0][0] = iminbf - (i<<1);
  mbi->MV[1][0][1] = (jminbf<<1) - (j<<1);
  mbi->mv_field_sel[0][0] = tself;
  mbi->mv_field_sel[1][0] = bself,
```

```
    /* backward */
    mbi->MV[0][1][0] = imintr - (i<<1);
    mbi->MV[0][1][1] = (jmintr<<1) - (j<<1);
    mbi->MV[1][1][0] = iminbr - (i<<1);
    mbi->MV[1][1][1] = (jminbr<<1) - (j<<1);
    mbi->mv_field_sel[0][1] = tselr;
    mbi->mv_field_sel[1][1] = bselr;
   }
  }
 }

 mbi->var = var;

 mbı++;

 }
}


static void frame_estimate(char *org, char *ref,
                    char *mb,
                    int i, int j,
                    int sx, int sy,
                    int *iminp, int *jminp,
                    int *ımintp, int *jmintp,
                    int *iminbp, int *jminbp,
                    float *dframep, float *dfieldp,
                    int *tselp, int *bselp,
                    int imins[2][2], int jmins[2][2] )
{
 float dt,db,dmint,dminb;
 int imint,iminb,jmint,jminb;

 /* frame prediction   */
 *dframep = fullsearch(org,ref,mb,width,i,j,sx,sy,16,width,height,
               iminp,jminp);

 /* predict top field from top field   */
 dt = fullsearch(org,ref,mb,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
          &imint,&jmint);

 /* predıct top field from bottom field  */
 db = fullsearch(org+width,ref+width,mb,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
          &iminb,&jminb);

 imins[0][0] = imint;
 jmins[0][0] = jmint;
 imins[1][0] = iminb;
 jmins[1][0] = jminb;

 /* select prediction for top field */
 if (dt<=db)
 {
  dmint=dt; *imintp=imint; *jmintp=jmint; *tselp=0;
```

```
  }
  else
  {
   dmint=db, *imintp=iminb; *jmintp=jminb, *tselp=1;
  }

  /* predict bottom field from top field   */
  dt = fullsearch(org,ref,mb+width,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
               &imint,&jmint);

  /* predict bottom field from bottom field   */
  db = fullsearch(org+width, ref+width, mb+width,
               width<<1, i, j>>1, sx, sy>>1, 8, width, height>>1,
               &iminb,&jminb);

  imins[0][1] = imint;
  jmins[0][1] = jmint;
  imins[1][1] = iminb;
  jmins[1][1] = jminb;

  /* select prediction for bottom field */
  if (db<=dt)
  {
   dminb=db; *iminbp=iminb; *jminbp=jminb; *bselp=1;
  }
  else
  {
   dminb=dt; *iminbp=imint; *jminbp=jmint; *bselp=0;
  }

  *dfieldp=dmint+dminb;
}


static float fullsearch(char *org, char *ref,
                     char *blk,
                     int lx,
                     int i0,
                     int j0,
                     int sx,
                     int sy,
                     char h,
                     int xmax,
                     int ymax,
                     int *iminp,
                     int *jminp)
{
 int i,j,imin,jmin,ilow,ihigh,jlow,jhigh;
 float d,dmin;
 int k,l,sxy;
 int ii, jj;

 ilow = i0 - sx;
 ihigh = i0 + sx;
```

```
if (ilow<0)
  ilow = 0;

if (ihigh>xmax-16)
  ihigh = xmax-16;

jlow = j0 - sy;
jhigh = j0 + sy;

if (jlow<0)
  jlow = 0;

if (jhigh>ymax-h)
  jhigh = ymax-h;

/* full pel search, spiraling outwards */

imin = i0;
jmin = j0;
dmin = dist1(org+imin+lx*jmin,blk,lx,0,0,h,65536.0);

sxy = (sx>sy) ? sx : sy;

for (l=1; l<=sxy; l++)
{
  i = i0 - l;
  j = j0 - l;
  for (k=0; k<8*l; k++)
  {
    if (i>=ilow && i<=ihigh && j>=jlow && j<=jhigh)
    {
      d = dist1(org+i+lx*j,blk,lx,0,0,h,dmin);

      if (d<dmin)
      {
        dmin = d;
        imin = i;
        jmin = j;
      }
    }

    if      (k<2*l) i++;
    else if (k<4*l) j++;
    else if (k<6*l) i--;
    else            j--;
  }
}

/* half pel */
dmin = 65536.0;
imin = imin << 1;
jmin = jmin << 1;
ilow = imin - (imin>0);
```

```
 ihigh = imin + (imin<((xmax-16)<<1));
 jlow = jmin - (jmin>0);
 jhigh = jmin + (jmin<((ymax-h)<<1));

 for (j=jlow; j<=jhigh; j++)
   for (i=ilow; i<=ihigh; i++)
   {
    ii = i >> 1 ;
    jj = j >> 1 ;
    d = dist1(ref+ii+lx*jj,blk,lx,i&1,j&1,h,dmin);

     if (d<dmin)
     {
       dmin = d;
       imin = i;
       jmin = j;
     }
   }

 *iminp = imin;
 *jminp = jmin;

 return dmin;
}


static float dist1(char *blk1,
                   char *blk2,
                   int lx,
                   char hx,
                   char hy,
                   char h,
                   float distlim )
{
 char *p1,*p1a,*p2;
 char i, j;
 int v;
 float s;

 s = 0;
 p1 = blk1;
 p2 = blk2;

 if (!hx && !hy)
   for (j=0; j<h; j++)
   {
     if ((v = p1[0]  - p2[0])<0)  v = -v, s+= (float) v;
     if ((v = p1[1]  - p2[1])<0)  v = -v; s+= (float) v;
     if ((v = p1[2]  - p2[2])<0)  v = -v; s+= (float) v;
     if ((v = p1[3]  - p2[3])<0)  v = -v; s+= (float) v;
     if ((v = p1[4]  - p2[4])<0)  v = -v; s+= (float) v;
     if ((v = p1[5]  - p2[5])<0)  v = -v; s+= (float) v;
     if ((v = p1[6]  - p2[6])<0)  v = -v; s+= (float) v;
     if ((v = p1[7]  - p2[7])<0)  v = -v; s+= (float) v;
```

```
    if ((v = p1[8]  - p2[8])<0)  v = -v; s+= (float) v;
    if ((v = p1[9]  - p2[9])<0)  v = -v; s+= (float) v;
    if ((v = p1[10] - p2[10])<0) v = -v; s+= (float) v;
    if ((v = p1[11] - p2[11])<0) v = -v; s+= (float) v;
    if ((v = p1[12] - p2[12])<0) v = -v; s+= (float) v;
    if ((v = p1[13] - p2[13])<0) v = -v; s+= (float) v;
    if ((v = p1[14] - p2[14])<0) v = -v; s+= (float) v;
    if ((v = p1[15] - p2[15])<0) v = -v; s+= (float) v;

    if (s >= distlim)
     break;

    p1+= lx;
    p2+= lx;
   }
 else if (hx && !hy)
  for (j=0; j<h; j++)
  {
   for (i=0; i<16; i++)
   {
    v = (int) ((p1[i]+p1[i+1]+1) / 2 - p2[i]);
    if (v>=0) s+= (float) v;
    else s-= (float) v;
   }
   p1+= lx;
   p2+= lx;
  }
 else if (!hx && hy)
 {
  p1a = p1 + lx;
  for (j=0; j<h; j++)
  {
   for (i=0; i<16; i++)
   {
    v = (int ) ((p1[i]+p1a[i]+1) / 2 - p2[i]);
    if (v>=0)
     s+= (float) v;
    else
     s-= (float) v;
   }
   p1 = p1a;
   p1a+= lx;
   p2+= lx;
  }
 }
 else /* if (hx && hy) */
 {
  p1a = p1 + lx;
  for (j=0; j<h, j++)
  {
   for (i=0; i<16; i++)
   {
    v = (int) ((p1[i]+p1[i+1]+p1a[i]+p1a[i+1]+2) / 4 - p2[i]);
    if (v>=0)
```

```
        s+= (float) v;
      else
        s-= (float) v;
    }
    p1 = p1a;
    p1a+= lx;
    p2+= lx;
  }
 }

 return s;
}


static float dist2(char *blk1,
              char *blk2,
              int lx,
              char hx,
              char hy,
              char h )
{
 char *p1,*p1a,*p2;
 char i, j;
 int v;
 float s;

 s = 0;
 p1 = blk1;
 p2 = blk2;
 if (!hx && !hy)
  for (j=0, j<h; j++)
  {
   for (i=0; i<16; i++)
   {
    v = p1[i] - p2[i];
    s+= (float) (v*v);
   }
   p1+= lx;
   p2+= lx;
  }
 else if (hx && !hy)
  for (j=0; j<h; j++)
  {
   for (i=0; i<16; i++)
   {
    v = (int) ((p1[i]+p1[i+1]+1) / 2 - p2[i]);
    s+= (float) (v*v);
   }
   p1+= lx;
   p2+= lx;
  }
 else if (!hx && hy)
 {
  p1a = p1 + lx;
```

```
    for (j=0; j<h; j++)
    {
      for (i=0; i<16; i++)
      {
        v = (int) ((p1[i]+p1a[i]+1) / 2 - p2[i]);
        s+= (float) (v*v);
      }
      p1 = p1a;
      p1a+= lx;
      p2+= lx;
    }
  }
  else /* if (hx && hy) */
  {
    p1a = p1 + lx;
    for (j=0; j<h; j++)
    {
      for (i=0; i<16; i++)
      {
        v = (int) ((p1[i]+p1[i+1]+p1a[i]+p1a[i+1]+2) / 4 - p2[i]);
        s+= (float) (v*v);
      }
      p1 = p1a;
      p1a+= lx;
      p2+= lx;
    }
  }

  return s;
}


static float bdist1(char *pf, char *pb, char *p2,
                    int lx, int hxf,
                    int hyf, int hxb,
                    int hyb, char h )
{
  char *pfa,*pfb,*pfc,*pba,*pbb,*pbc;
  char i, j;
  int v;
  float s;

  pfa = pf + hxf;
  pfb = pf + lx*hyf;
  pfc = pfb + hxf;

  pba = pb + hxb;
  pbb = pb + lx*hyb;
  pbc = pbb + hxb;

  s = 0;

  for (j=0; j<h; j++)
  {
```

```
   for (i=0; i<16; i++)
   {
    v = (int) (((*pf++ + *pfa++ + *pfb++ + *pfc++ + 2) / 4 +
              (*pb++ + *pba++ + *pbb++ + *pbc++ + 2) / 4 + 1) / 2
             - *p2++);

    if (v>=0)
      s+= (float) v;
    else
      s-= (float) v;
   }
   p2+= lx-16;
   pf+= lx-16;
   pfa+= lx-16;
   pfb+= lx-16;
   pfc+= lx-16;
   pb+= lx-16;
   pba+= lx-16;
   pbb+= lx-16;
   pbc+= lx-16;
  }

  return s;
}


static float bdist2(char *pf, char *pb, char *p2,
                    int lx, int hxf,
                    int hyf, int hxb,
                    int hyb, char h )
{
 char *pfa,*pfb,*pfc,*pba,*pbb,*pbc;
 char i, j;
 int v;
 float s;

 pfa = pf + hxf;
 pfb = pf + lx*hyf;
 pfc = pfb + hxf;

 pba = pb + hxb;
 pbb = pb + lx*hyb;
 pbc = pbb + hxb;

 s = 0;

 for (j=0; j<h; j++)
 {
  for (i=0; i<16; i++)
  {
   v = (int) (((*pf++ + *pfa++ + *pfb++ + *pfc++ + 2) / 4 +
             (*pb++ + *pba++ + *pbb++ + *pbc++ + 2) / 4 + 1) / 2
            - *p2++);
   s+= (float) (v*v);
```

```
    }
    p2+= lx-16,
    pf+= lx-16;
    pfa+= lx-16;
    pfb+= lx-16,
    pfc+= lx-16;
    pb+= lx-16;
    pba+= lx-16;
    pbb+= lx-16;
    pbc+= lx-16;
  }

  return s;
}


static float variance(char *p, int lx )
{
  char i, j;
  int v;
  float s, s2;

  s = s2 = 0;

  for (j=0; j<16; j++)
  {
    for (i=0; i<16; i++)
    {
      v = *p++;
      s+= (float) v;
      s2+= (float) (v*v);
    }
    p+= lx-16;
  }
  s=0.0625*s;
  return s2 - s*s;
}
```

```
/*********************************/
/*                               */
/*      m012.c residing at PE3   */
/*                               */
/*********************************/

#include "c:\tc2v2\dos.h"
#include "c:\tc2v2\float.h"
#include "c:\tc2v2\math.h"
#include "c:\tc2v2\ctype.h"
#include "c:\tc2v2\stdio.h"
#include "c:\tc2v2\stdlib.h"
#include "c:\tc2v2\string.h"
#include "c:\tc2v2\chan.h"
#include "c:\tc2v2\thread.h"
#include "c:\tc2v2\time.h"
#include "c:\tc2v2\errno h"

#define GLOBAL /* used by global.h */

#include "c:\nick\h\const1.h"
#include "c:\nick\h\config.h"
#include "c:\nick\h\global.h"

static void init _ANSI_ARGS_((void));

char  *neworg[1];

void main(argc,argv,envp,ins,in_ports,outs,out_ports)
int argc,ins,outs;
char *argv[],*envp[];
CHAN *in_ports[],*out_ports[],
{
 /* read parameter file */
 chan_in_word(&M,Link0Input);
 chan_in_word(&nframes,Link0Input);

 init(),
 putseq();

}

static void init()
{
 int i, size, a;

 chan_in_word(&mb_width,Link0Input);
 chan_in_word(&mb_height,Link0Input);
 chan_in_word(&mb_height2,Link0Input),
 chan_in_word(&width,Link0Input);
 chan_in_word(&height,Link0Input),
 chan_in_word(&width2,Link0Input);
 chan_in_word(&height2,Link0Input);
```

```
  size = width*height ;

  newrefframe[0] = (char *)malloc(size);
  oldrefframe[0] = (char *)malloc(size);
  neworgframe[0] = (char *)malloc(size);
  oldorgframe[0] = (char *)malloc(size);
  neworg[0] = (char *)malloc(size);

  mbinfo = (struct mbinfo *)malloc(37*sizeof(struct mbinfo));

  for (a=0; a<37; a++)
  {
   mbinfo[a].mb_type = mbinfo[a].motion_type = 0;
   mbinfo[a].MV[0][0][0] = mbinfo[a].MV[0][0][1] = mbinfo[a].MV[0][1][0] = 0;
   mbinfo[a].MV[0][1][1] = mbinfo[a].MV[1][0][0] = mbinfo[a].MV[1][0][1] = 0;
   mbinfo[a].MV[1][1][0] = mbinfo[a].MV[1][1][1] = 0;
   mbinfo[a].mv_field_sel[0][0] = mbinfo[a].mv_field_sel[0][1] = 0;
   mbinfo[a].mv_field_sel[1][0] = mbinfo[a].mv_field_sel[1][1] = 0,
  }

}


void putseq()
{
/* this routine assumes (N % M) == 0 */
  int i, j, k, f, f0, n, np, nb;
  int a;
  int sxf, syf, sxb, syb;
  int size, loop,

  size = width*height;


/* loop through all frames in encoding/decoding order */
do{
   chan_in_word(&loop,Link0Input);

   if(loop==nframes) break;

   chan_in_word(&pict_type,Link0Input);
   chan_in_word(&sxf,Link0Input);
   chan_in_word(&syf,Link0Input);
   chan_in_word(&sxb,Link0Input);
   chan_in_word(&syb,Link0Input);

   if ( pict_type != I_TYPE )
   {
    chan_in_message(size,neworg[0],Link0Input);
    chan_in_message(size,oldorgframe[0],Link0Input);
    chan_in_message(size,oldrefframe[0],Link0Input);
    if ( pict_type != P_TYPE )
    {
     chan_in_message(size,neworgframe[0],Link0Input);
```

```
      chan_in_message(size,newrefframe[0],Link0Input),
     }
    }

/* loop through 37 macroblocks of the picture */

    motion_estimation(oldorgframe[0], neworgframe[0],
                oldrefframe[0], newrefframe[0],
                neworg[0],
                sxf, syf,
                sxb, syb,
                mbinfo);

     for (a=0; a<37; a++)
     {
      chan_out_word(mbinfo[a].mb_type,Link0Output);
      chan_out_word(mbinfo[a].motion_type,Link0Output);
      chan_out_word(mbinfo[a].MV[0][0][0],Link0Output);
      chan_out_word(mbinfo[a].MV[0][0][1],Link0Output);
      chan_out_word(mbinfo[a].MV[0][1][0],Link0Output);
      chan_out_word(mbinfo[a].MV[0][1][1],Link0Output);
      chan_out_word(mbinfo[a].MV[1][0][0],Link0Output);
      chan_out_word(mbinfo[a].MV[1][0][1],Link0Output);
      chan_out_word(mbinfo[a].MV[1][1][0],Link0Output);
      chan_out_word(mbinfo[a].MV[1][1][1],Link0Output);
      chan_out_word(mbinfo[a].mv_field_sel[0][0],Link0Output);
      chan_out_word(mbinfo[a].mv_field_sel[0][1],Link0Output);
      chan_out_word(mbinfo[a].mv_field_sel[1][0],Link0Output);
      chan_out_word(mbinfo[a].mv_field_sel[1][1],Link0Output);
     }

  }while(loop<nframes);

   chan_out_word(loop,Link0Output);

}

static void frame_estimate _ANSI_ARGS_((char *org,
                          char *ref,
                          char *mb,
                          int i, int j,
                          int sx, int sy,
                          int *iminp, int *jminp,
                          int *imintp, int *jmintp,
                          int *iminbp, int *jminbp,
                          float *dframep, float *dfieldp,
                          int *tselp, int *bselp,
                          int imins[2][2], int jmins[2][2]));

static float fullsearch _ANSI_ARGS_((char *org,
                         char *ref,
                         char *blk,
                         int lx,
                         int i0,
```

```
                              int j0,
                              int sx,
                              int sy,
                              char h,
                              int xmax,
                              int ymax,
                              int *iminp,
                              int *jminp));

static float dist1 _ANSI_ARGS_((char *blk1,
                         char *blk2,
                         int lx,
                         char hx,
                         char hy,
                         char h,
                         float distlim));

static float dist2 _ANSI_ARGS_((char *blk1,
                         char *blk2,
                         int lx,
                         char hx,
                         char hy,
                         char h));

static float bdist1 _ANSI_ARGS_((char *pf,
                          char *pb,
                          char *p2,
                          int lx,
                          int hxf,
                          int hyf,
                          int hxb,
                          int hyb,
                          char h));

static float bdist2 _ANSI_ARGS_((char *pf,
                          char *pb,
                          char *p2,
                          int lx,
                          int hxf,
                          int hyf,
                          int hxb,
                          int hyb,
                          char h));

static float variance _ANSI_ARGS_((char *p, int lx));


void motion_estimation(char *oldorg, char *neworg,
                 char *oldref, char *newref,
                 char *cur,
                 int sxf,
                 int syf,
                 int sxb,
                 int syb,
```

```
                 struct mbinfo *mbi)
{
 int i, j;
 int imin,jmin,iminf,jminf,iminr,jminr;
 int imint,jmint,iminb,jminb;
 int imintf,jmintf,iminbf,jminbf,
 int imintr,jmintr,iminbr,jminbr;
 float var,v0;
 float dmc,dmcf,dmcr,dmci,vmc,vmcf,vmcr,vmci;
 float dmcfield,dmcfieldf,dmcfieldr,dmcfieldi;
 int tsel,bsel,tself,bself,tselr,bselr,
 char *mb;
 int imins[2][2],jmins[2][2];
 int a, ii, jj;

/* loop through 37 macroblocks of the picture */

 for(a=109; a<146; a++)
 {
  jj = a>>4;
  ii = a - (jj<<4),
  j = jj<<4;
  i = ii<<4,

  mb = cur + i + width*j;

  var = variance(mb,width);

  /* for I_type pictures    */

  if (pict_type==I_TYPE)
  mbi->mb_type = MB_INTRA;

  /* for P_type pictures    */

  if (pict_type==P_TYPE)
  {
   frame_estimate(oldorg,oldref,mb,i,j,sxf,syf,
             &imin,&jmin,&imint,&jmint,&iminb,&jminb,
             &dmc,&dmcfield,&tsel,&bsel,imins,jmins);

  /* select between frame and field prediction */
  if (dmc<=dmcfield)
  {
   mbi->motion_type = MC_FRAME;
   vmc = dist2(oldref+(imin>>1)+width*(jmin>>1),mb,
           width,imin&1,jmin&1,16);
  }
  else
  {
   mbi->motion_type = MC_FIELD,
   dmc = dmcfield;
   vmc = dist2(oldref+(tsel?width:0)+(imint>>1)+(width<<1)*(jmint>>1),
           mb,width<<1,imint&1,jmint&1,8);
```

```
 vmc+= dist2(oldref+(bsel?width:0)+(iminb>>1)+(width<<1)*(jminb>>1),
        mb+width,width<<1,iminb&1,jminb&1,8);
 }

 if (vmc>var && vmc>=9*256)
 mbi->mb_type = MB_INTRA;
 else
 {
  v0 = dist2(oldref+i+width*j,mb,width,0,0,16);
  if (4*v0>5*vmc && v0>=9*256)
  {
   /* use MC */
   var = vmc;
   mbi->mb_type = MB_FORWARD;
   if (mbi->motion_type==MC_FRAME)
   {
    mbi->MV[0][0][0] = imin - (i<<1);
    mbi->MV[0][0][1] = jmin - (j<<1);
   }
   else /* if (mbi->motion_type==MC_FIELD) */
   {
    /* these are FRAME vectors */
    mbi->MV[0][0][0] = imint - (i<<1);
    mbi->MV[0][0][1] = (jmint<<1) - (j<<1);
    mbi->MV[1][0][0] = iminb - (i<<1),
    mbi->MV[1][0][1] = (jminb<<1) - (j<<1);
    mbi->mv_field_sel[0][0] = tsel;
    mbi->mv_field_sel[1][0] = bsel;
   }
  }
  else
  {
   /* No-MC */
   var = v0;
   mbi->mb_type = 0;
   mbi->motion_type = MC_FRAME;
   mbi->MV[0][0][0] = 0,
   mbi->MV[0][0][1] = 0;
  }
 }
}


/*  for B_type pictures     */

if (pict_type==B_TYPE)
{
 /* forward prediction */
 frame_estimate(oldorg,oldref,mb,i,j,sxf,syf,
          &iminf,&jminf,&imintf,&jmintf,&iminbf,&jminbf,
          &dmcf,&dmcfieldf,&tself,&bself,imins,jmins);

 /* backward prediction */
 frame_estimate(neworg,newref,mb,i,j,sxb,syb,
```

```
            &iminr,&jminr,&imintr,&jmintr,&iminbr,&jminbr,
            &dmcr,&dmcfieldr,&tselr,&bselr,imins,jmins);

/* calculate interpolated distance */
/* frame */
dmci = bdist1(oldref+(iminf>>1)+width*(jminf>>1),
          newref+(iminr>>1)+width*(jminr>>1),
          mb,width,iminf&1,jminf&1,iminr&1,jminr&1,16);

/* top field
dmcfieldi = bdist1(oldref+(imintf>>1)+(tself?width:0)+(width<<1)*(jmintf>>1),
              newref+(imintr>>1)+(tselr?width:0)+(width<<1)*(jmintr>>1),
              mb,width<<1,imintf&1,jmintf&1,imintr&1,jmintr&1,8);

/* bottom field
dmcfieldi+= bdist1(oldref+(iminbf>>1)+(bself?width:0)+(width<<1)*(jminbf>>1),
              newref+(iminbr>>1)+(bselr?width:0)+(width<<1)*(jminbr>>1),
              mb+width,width<<1,iminbf&1,jminbf&1,iminbr&1,jminbr&1,8);


if (dmci<dmcfieldi && dmci<dmcf && dmci<dmcfieldf
   && dmci<dmcr && dmci<dmcfieldr)
{
 /* frame, interpolated */
 mbi->mb_type = MB_FORWARD|MB_BACKWARD;
 mbi->motion_type = MC_FRAME;
 vmc = bdist2(oldref+(iminf>>1)+width*(jminf>>1),
          newref+(iminr>>1)+width*(jminr>>1),
          mb,width,iminf&1,jminf&1,iminr&1,jminr&1,16);
}
else if (dmcfieldi<dmcf && dmcfieldi<dmcfieldf
      && dmcfieldi<dmcr && dmcfieldi<dmcfieldr)
{
 /* field, interpolated */
 mbi->mb_type = MB_FORWARD|MB_BACKWARD;
 mbi->motion_type = MC_FIELD;
 vmc = bdist2(oldref+(imintf>>1)+(tself?width:0)+(width<<1)*(jmintf>>1),
          newref+(imintr>>1)+(tselr?width:0)+(width<<1)*(jmintr>>1),
          mb,width<<1,imintf&1,jmintf&1,imintr&1,jmintr&1,8);
 vmc+= bdist2(oldref+(iminbf>>1)+(bself?width:0)+(width<<1)*(jminbf>>1),
          newref+(iminbr>>1)+(bselr?width:0)+(width<<1)*(jminbr>>1),
          mb+width,width<<1,iminbf&1,jminbf&1,iminbr&1,jminbr&1,8);
}
else if (dmcf<dmcfieldf && dmcf<dmcr && dmcf<dmcfieldr)
{
 /* frame, forward */
 mbi->mb_type = MB_FORWARD;
 mbi->motion_type = MC_FRAME;
 vmc = dist2(oldref+(iminf>>1)+width*(jminf>>1),mb,
         width,iminf&1,jminf&1,16);
}
else if (dmcfieldf<dmcr && dmcfieldf<dmcfieldr)
{
 /* field, forward */
```

```
 mbi->mb_type = MB_FORWARD;
 mbi->motion_type = MC_FIELD;
 vmc = dist2(oldref+(tself?width:0)+(imintf>>1)+(width<<1)*(jmintf>>1),
        mb,width<<1,imintf&1,jmintf&1,8);
 vmc+= dist2(oldref+(bself?width:0)+(iminbf>>1)+(width<<1)*(jminbf>>1),
        mb+width,width<<1,iminbf&1,jminbf&1,8);
}
else if (dmcr<dmcfieldr)
{
 /* frame, backward */
 mbi->mb_type = MB_BACKWARD;
 mbi->motion_type = MC_FRAME;
 vmc = dist2(newref+(iminr>>1)+width*(jminr>>1),mb,
        width,iminr&1,jminr&1,16);
}
else
{
 /* field, backward */
 mbi->mb_type = MB_BACKWARD;
 mbi->motion_type = MC_FIELD;
 vmc = dist2(newref+(tselr?width:0)+(imintr>>1)+(width<<1)*(jmintr>>1),
        mb,width<<1,imintr&1,jmintr&1,8);
 vmc+= dist2(newref+(bselr?width:0)+(iminbr>>1)+(width<<1)*(jminbr>>1),
        mb+width,width<<1,iminbr&1,jminbr&1,8);
}

if (vmc>var && vmc>=9*256)
mbi->mb_type = MB_INTRA;
else
{
 var = vmc;
 if (mbi->motion_type==MC_FRAME)
 {
  /* forward */
  mbi->MV[0][0][0] = iminf - (i<<1);
  mbi->MV[0][0][1] = jminf - (j<<1);
  /* backward */
  mbi->MV[0][1][0] = iminr - (i<<1);
  mbi->MV[0][1][1] = jminr - (j<<1);
 }
 else
 {
  /* these are FRAME vectors */
  /* forward */
  mbi->MV[0][0][0] = imintf - (i<<1);
  mbi->MV[0][0][1] = (jmintf<<1) - (j<<1);
  mbi->MV[1][0][0] = iminbf - (i<<1);
  mbi->MV[1][0][1] = (jminbf<<1) - (j<<1);
  mbi->mv_field_sel[0][0] = tself;
  mbi->mv_field_sel[1][0] = bself;
  /* backward */
  mbi->MV[0][1][0] = imintr - (i<<1);
  mbi->MV[0][1][1] = (jmintr<<1) - (j<<1);
  mbi->MV[1][1][0] = iminbr - (i<<1);
```

```
      mbi->MV[1][1][1] = (jminbr<<1) - (j<<1);
      mbi->mv_field_sel[0][1] = tselr;
      mbi->mv_field_sel[1][1] = bselr;
     }
    }
   }

   mbi->var = var;

   mbi++;

  }
 }


static void frame_estimate(char *org, char *ref,
                           char *mb,
                           int i, int j,
                           int sx, int sy,
                           int *iminp, int *jminp,
                           int *imintp, int *jmintp,
                           int *iminbp, int *jminbp,
                           float *dframep, float *dfieldp,
                           int *tselp, int *bselp,
                           int imins[2][2], int jmins[2][2] )
{
  float dt,db,dmint,dminb;
  int imint,iminb,jmint,jminb;

  /* frame prediction   */
  *dframep = fullsearch(org,ref,mb,width,i,j,sx,sy,16,width,height,
                        iminp,jminp);

  /* predict top field from top field   */
  dt = fullsearch(org,ref,mb,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
                  &imint,&jmint);

  /* predict top field from bottom field  */
  db = fullsearch(org+width,ref+width,mb,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
                  &iminb,&jminb);

  imins[0][0] = imint;
  jmins[0][0] = jmint,
  imins[1][0] = iminb;
  jmins[1][0] = jminb;

  /* select prediction for top field */
  if (dt<=db)
  {
    dmint=dt; *imintp=imint; *jmintp=jmint; *tselp=0;
  }
  else
  {
    dmint=db; *imintp=iminb, *jmintp=jminb; *tselp=1;
```

```
    }

    /* predict bottom field from top field    */
    dt = fullsearch(org,ref,mb+width,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
                &imint,&jmint);

    /* predict bottom field from bottom field   */
    db = fullsearch(org+width, ref+width, mb+width,
                width<<1, i, j>>1, sx, sy>>1, 8, width, height>>1,
                &iminb,&jminb);

    imins[0][1] = imint;
    jmins[0][1] = jmint;
    imins[1][1] = iminb;
    jmins[1][1] = jminb;

    /* select prediction for bottom field */
    if (db<=dt)
    {
      dminb=db; *iminbp=iminb; *jminbp=jminb; *bselp=1;
    }
    else
    {
      dminb=dt; *iminbp=imint, *jminbp=jmint; *bselp=0;
    }

    *dfieldp=dmint+dminb;
}


static float fullsearch(char *org, char *ref,
                  char *blk,
                  int lx,
                  int i0,
                  int j0,
                  int sx,
                  int sy,
                  char h,
                  int xmax,
                  int ymax,
                  int *iminp,
                  int *jminp)
{
  int i,j,imin,jmin,ilow,ihigh,jlow,jhigh;
  float d,dmin;
  int k,l,sxy;
  int ii, jj;

  ilow = i0 - sx,
  ihigh = i0 + sx;

  if (ilow<0) ilow = 0;

  if (ihigh>xmax-16) ihigh = xmax-16;
```

```
jlow = j0 - sy,
jhigh = j0 + sy,

if (jlow<0) jlow = 0;

if (jhigh>ymax-h) jhigh = ymax-h;

/* full pel search, spiraling outwards */

imin = i0;
jmin = j0;
dmin = dist1(org+imin+lx*jmin,blk,lx,0,0,h,65536.0);

sxy = (sx>sy) ? sx : sy;

for (l=1; l<=sxy; l++)
{
 i = i0 - l;
 j = j0 - l;
 for (k=0, k<8*l; k++)
 {
  if (i>=ilow && i<=ihigh && j>=jlow && j<=jhigh)
  {
   d = dist1(org+i+lx*j,blk,lx,0,0,h,dmin),

   if (d<dmin)
   {
    dmin = d;
    imin = i;
    jmin = j;
   }
  }

  if      (k<2*l) i++;
  else if (k<4*l) j++;
  else if (k<6*l) i--;
  else            j--;
 }
}

/* half pel */
dmin = 65536.0;
imin = imin << 1;
jmin = jmin << 1;
ilow = imin - (imin>0);
ihigh = imin + (imin<((xmax-16)<<1));
jlow = jmin - (jmin>0);
jhigh = jmin + (jmin<((ymax-h)<<1));

for (j=jlow; j<=jhigh; j++)
 for (i=ilow; i<=ihigh; i++)
 {
  ii = i >> 1 ;
```

```
    jj = j >> 1 ;
    d = dist1(ref+ii+lx*jj,blk,lx,i&1,j&1,h,dmin);

     if (d<dmin)
     {
       dmin = d;
       imin = i;
       jmin = j;
     }
   }

  *iminp = imin,
  *jminp = jmin,

  return dmin;
}

static float dist1(char *blk1,
               char *blk2,
               int lx,
               char hx,
               char hy,
               char h,
               float distlim )
{
 char *p1,*p1a,*p2;
 char i, j;
 int v;
 float s;

 s = 0;
 p1 = blk1;
 p2 = blk2;

 if (!hx && !hy)
   for (j=0; j<h; j++)
   {
     if ((v = p1[0]  - p2[0])<0)  v = -v; s+= (float) v;
     if ((v = p1[1]  - p2[1])<0)  v = -v; s+= (float) v;
     if ((v = p1[2]  - p2[2])<0)  v = -v; s+= (float) v;
     if ((v = p1[3]  - p2[3])<0)  v = -v; s+= (float) v;
     if ((v = p1[4]  - p2[4])<0)  v = -v; s+= (float) v;
     if ((v = p1[5]  - p2[5])<0)  v = -v; s+= (float) v;
     if ((v = p1[6]  - p2[6])<0)  v = -v; s+= (float) v;
     if ((v = p1[7]  - p2[7])<0)  v = -v; s+= (float) v;
     if ((v = p1[8]  - p2[8])<0)  v = -v; s+= (float) v;
     if ((v = p1[9]  - p2[9])<0)  v = -v; s+= (float) v;
     if ((v = p1[10] - p2[10])<0) v = -v; s+= (float) v;
     if ((v = p1[11] - p2[11])<0) v = -v; s+= (float) v;
     if ((v = p1[12] - p2[12])<0) v = -v; s+= (float) v;
     if ((v = p1[13] - p2[13])<0) v = -v; s+= (float) v;
     if ((v = p1[14] - p2[14])<0) v = -v; s+= (float) v;
     if ((v = p1[15] - p2[15])<0) v = -v; s+= (float) v;
```

```
    if (s >= distlim)
      break;

    p1+= lx;
    p2+= lx;
  }
else if (hx && !hy)
  for (j=0; j<h; j++)
  {
    for (i=0; i<16; i++)
    {
      v = (int) ((p1[i]+p1[i+1]+1) / 2 - p2[i]);
      if (v>=0) s+= (float) v;
      else s-= (float) v;
    }
    p1+= lx;
    p2+= lx;
  }
else if (!hx && hy)
{
  p1a = p1 + lx;
  for (j=0; j<h; j++)
  {
    for (i=0; i<16; i++)
    {
      v = (int ) ((p1[i]+p1a[i]+1) / 2 - p2[i]),
      if (v>=0)
        s+= (float) v;
      else
        s-= (float) v;
    }
    p1 = p1a;
    p1a+= lx;
    p2+= lx;
  }
}
else /* if (hx && hy) */
{
  p1a = p1 + lx;
  for (j=0, j<h; j++)
  {
    for (i=0; i<16; i++)
    {
      v = (int) ((p1[i]+p1[i+1]+p1a[i]+p1a[i+1]+2) / 4 - p2[i]);
      if (v>=0)
        s+= (float) v;
      else
        s-= (float) v;
    }
    p1 = p1a;
    p1a+= lx;
    p2+= lx;
  }
}
```

```
  return s;
}

static float dist2(char *blk1,
                   char *blk2,
                   int lx,
                   char hx,
                   char hy,
                   char h )
{
  char *p1,*p1a,*p2;
  char i, j;
  int v;
  float s;

  s = 0;
  p1 = blk1;
  p2 = blk2;
  if (!hx && !hy)
    for (j=0; j<h; j++)
    {
      for (i=0; i<16; i++)
      {
        v = p1[i] - p2[i];
        s+= (float) (v*v);
      }
      p1+= lx;
      p2+= lx;
    }
  else if (hx && !hy)
    for (j=0; j<h; j++)
    {
      for (i=0; i<16; i++)
      {
        v = (int) ((p1[i]+p1[i+1]+1) / 2 - p2[i]);
        s+= (float) (v*v),
      }
      p1+= lx;
      p2+= lx;
    }
  else if (!hx && hy)
  {
    p1a = p1 + lx;
    for (j=0; j<h; j++)
    {
      for (i=0; i<16; i++)
      {
        v = (int) ((p1[i]+p1a[i]+1) / 2 - p2[i]);
        s+= (float) (v*v);
      }
      p1 = p1a;
      p1a+= lx;
      p2+= lx;
```

```
    }
  }
  else /* if (hx && hy) */
  {
    p1a = p1 + lx;
    for (j=0; j<h; j++)
    {
      for (i=0, i<16; i++)
      {
        v = (int) ((p1[i]+p1[i+1]+p1a[i]+p1a[i+1]+2) / 4 - p2[i]);
        s+= (float) (v*v);
      }
      p1 = p1a;
      p1a+= lx;
      p2+= lx;
    }
  }

  return s;
}

static float bdist1(char *pf, char *pb, char *p2,
                    int lx, int hxf,
                    int hyf, int hxb,
                    int hyb, char h )
{
  char *pfa,*pfb,*pfc,*pba,*pbb,*pbc;
  char i, j;
  int v;
  float s;

  pfa = pf + hxf;
  pfb = pf + lx*hyf;
  pfc = pfb + hxf;

  pba = pb + hxb;
  pbb = pb + lx*hyb;
  pbc = pbb + hxb;

  s = 0;

  for (j=0; j<h; j++)
  {
    for (i=0; i<16; i++)
    {
      v = (int) (((*pf++ + *pfa++ + *pfb++ + *pfc++ + 2) / 4 +
                (*pb++ + *pba++ + *pbb++ + *pbc++ + 2) / 4 + 1) / 2
               - *p2++);

      if (v>=0)
        s+= (float) v;
      else
        s-= (float) v;
    }
```

```
    p2+= lx-16;
    pf+= lx-16,
    pfa+= lx-16;
    pfb+= lx-16;
    pfc+= lx-16;
    pb+= lx-16;
    pba+= lx-16;
    pbb+= lx-16;
    pbc+= lx-16;
  }

  return s;
}

static float bdist2(char *pf, char *pb, char *p2,
                    int lx, int hxf,
                    int hyf, int hxb,
                    int hyb, char h )
{
  char *pfa,*pfb,*pfc,*pba,*pbb,*pbc;
  char i, j;
  int v,
  float s;

  pfa = pf + hxf;
  pfb = pf + lx*hyf;
  pfc = pfb + hxf;

  pba = pb + hxb;
  pbb = pb + lx*hyb;
  pbc = pbb + hxb;

  s = 0;

  for (j=0; j<h; j++)
  {
    for (i=0; i<16; i++)
    {
      v = (int) (((*pf++ + *pfa++ + *pfb++ + *pfc++ + 2) / 4 +
                (*pb++ + *pba++ + *pbb++ + *pbc++ + 2) / 4 + 1) / 2
                - *p2++);
      s+= (float) (v*v);
    }
    p2+= lx-16;
    pf+= lx-16;
    pfa+= lx-16;
    pfb+= lx-16;
    pfc+= lx-16;
    pb+= lx-16;
    pba+= lx-16;
    pbb+= lx-16;
    pbc+= lx-16;
  }
```

```
 return s;
}

static float variance(char *p, int lx )
{
 char i, j;
 int v;
 float s, s2;

 s = s2 = 0;

 for (j=0, j<16; j++)
 {
  for (i=0; i<16; i++)
  {
   v = *p++;
   s+= (float) v;
   s2+= (float) (v*v);
  }
  p+= lx-16;
 }
 s=0.0625*s;
 return s2 - s*s;
}
```

```
/*********************************/
/*                               */
/*   m02.c residing at PE4       */
/*                               */
/*********************************/

#include "c:\tc2v2\dos.h"
#include "c:\tc2v2\float.h"
#include "c:\tc2v2\math.h"
#include "c:\tc2v2\ctype.h"
#include "c:\tc2v2\stdio.h"
#include "c:\tc2v2\stdlib.h"
#include "c:\tc2v2\string.h"
#include "c:\tc2v2\chan.h"
#include "c:\tc2v2\thread.h"
#include "c:\tc2v2\time.h"
#include "c:\tc2v2\errno.h"

#define GLOBAL /* used by global.h */

#include "c:\nick\h\const1.h"
#include "c:\nick\h\config.h"
#include "c:\nick\h\global.h"

static void init _ANSI_ARGS_((void));

char   *neworg[1];

void main(argc,argv,envp,ins,in_ports,outs,out_ports)
int argc,ins,outs;
char *argv[],*envp[];
CHAN *in_ports[],*out_ports[];
{
 /* read parameter file */
 chan_in_word(&M,Link0Input);
 chan_out_word(M,Link1Output);
 chan_out_word(M,Link2Output);

 chan_in_word(&nframes,Link0Input);
 chan_out_word(nframes,Link1Output);
 chan_out_word(nframes,Link2Output);

 init(),
 putseq();

}

static void init()
{
 int i, size, a;

 chan_in_word(&mb_width,Link0Input);
 chan_out_word(mb_width,Link1Output);
 chan_out_word(mb_width,Link2Output);
```

```
  chan_in_word(&mb_height,Link0Input);
  chan_out_word(mb_height,Link1Output),
  chan_out_word(mb_height,Link2Output);

  chan_in_word(&mb_height2,Link0Input);
  chan_out_word(mb_height2,Link1Output);
  chan_out_word(mb_height2,Link2Output);

  chan_in_word(&width,Link0Input);
  chan_out_word(width,Link1Output);
  chan_out_word(width,Link2Output);

  chan_in_word(&height,Link0Input);
  chan_out_word(height,Link1Output);
  chan_out_word(height,Link2Output);

  chan_in_word(&width2,Link0Input);
  chan_out_word(width2,Link1Output);
  chan_out_word(width2,Link2Output);

  chan_in_word(&height2,Link0Input);
  chan_out_word(height2,Link1Output);
  chan_out_word(height2,Link2Output);

  size = width*height ;

  newrefframe[0] = (char *)malloc(size);
  oldrefframe[0] = (char *)malloc(size);
  neworgframe[0] = (char *)malloc(size);
  oldorgframe[0] = (char *)malloc(size);
  neworg[0] = (char *)malloc(size);

  mbinfo = (struct mbinfo *)malloc(110*sizeof(struct mbinfo));

  for (a=0; a<110; a++)
  {
   mbinfo[a].mb_type = mbinfo[a].motion_type = 0;
   mbinfo[a].MV[0][0][0] = mbinfo[a].MV[0][0][1] = mbinfo[a].MV[0][1][0] = 0;
   mbinfo[a].MV[0][1][1] = mbinfo[a] MV[1][0][0] = mbinfo[a].MV[1][0][1] = 0;
   mbinfo[a].MV[1][1][0] = mbinfo[a] MV[1][1][1] = 0;
   mbinfo[a].mv_field_sel[0][0] = mbinfo[a].mv_field_sel[0][1] = 0;
   mbinfo[a].mv_field_sel[1][0] = mbinfo[a].mv_field_sel[1][1] = 0;
  }


}


void putseq()
{
/* this routine assumes (N % M) == 0 */
 int i, j, k, f, f0, n, np, nb;
 int a;
```

```
int sxf, syf, sxb, syb;
int size, loop,

size = width*height;


/* loop through all frames in encoding/decoding order */
do{
   chan_in_word(&loop,Link0Input);
   chan_out_word(loop,Link1Output);
   chan_out_word(loop,Link2Output);

   if(loop==nframes) break;

   chan_in_word(&pict_type,Link0Input);
   chan_out_word(pict_type,Link1Output);
   chan_out_word(pict_type,Link2Output);

   chan_in_word(&sxf,Link0Input);
   chan_out_word(sxf,Link1Output);
   chan_out_word(sxf,Link2Output);

   chan_in_word(&syf,Link0Input);
   chan_out_word(syf,Link1Output);
   chan_out_word(syf,Link2Output);

   chan_in_word(&sxb,Link0Input);
   chan_out_word(sxb,Link1Output);
   chan_out_word(sxb,Link2Output);

   chan_in_word(&syb,Link0Input);
   chan_out_word(syb,Link1Output);
   chan_out_word(syb,Link2Output);

   if ( pict_type != I_TYPE )
   {
    chan_in_message(size,neworg[0],Link0Input);
    chan_out_message(size,neworg[0],Link1Output);
    chan_out_message(size,neworg[0],Link2Output);

    chan_in_message(size,oldorgframe[0],Link0Input);
    chan_out_message(size,oldorgframe[0],Link1Output);
    chan_out_message(size,oldorgframe[0],Link2Output);

    chan_in_message(size,oldrefframe[0],Link0Input);
    chan_out_message(size,oldrefframe[0],Link1Output);
    chan_out_message(size,oldrefframe[0],Link2Output);

    if ( pict_type != P_TYPE )
    {
     chan_in_message(size,neworgframe[0],Link0Input);
     chan_out_message(size,neworgframe[0],Link1Output);
     chan_out_message(size,neworgframe[0],Link2Output);
```

```
    chan_in_message(size,newrefframe[0],Link0Input);
    chan_out_message(size,newrefframe[0],Link1Output);
    chan_out_message(size,newrefframe[0],Link2Output);
   }
  }

/* loop through 36 macroblocks of the picture */

  motion_estimation(oldorgframe[0], neworgframe[0],
                oldrefframe[0], newrefframe[0],
                neworg[0],
                sxf, syf,
                sxb, syb,
                mbinfo);

   for (a=36; a<73; a++)
   {
    chan_in_word(&mbinfo[a].mb_type,Link1Input);
    chan_in_word(&mbinfo[a].motion_type,Link1Input);
    chan_in_word(&mbinfo[a].MV[0][0][0],Link1Input);
    chan_in_word(&mbinfo[a].MV[0][0][1],Link1Input);
    chan_in_word(&mbinfo[a].MV[0][1][0],Link1Input);
    chan_in_word(&mbinfo[a].MV[0][1][1],Link1Input);
    chan_in_word(&mbinfo[a].MV[1][0][0],Link1Input);
    chan_in_word(&mbinfo[a].MV[1][0][1],Link1Input);
    chan_in_word(&mbinfo[a].MV[1][1][0],Link1Input);
    chan_in_word(&mbinfo[a].MV[1][1][1],Link1Input);
    chan_in_word(&mbinfo[a].mv_field_sel[0][0],Link1Input);
    chan_in_word(&mbinfo[a].mv_field_sel[0][1],Link1Input);
    chan_in_word(&mbinfo[a].mv_field_sel[1][0],Link1Input);
    chan_in_word(&mbinfo[a].mv_field_sel[1][1],Link1Input);
   }

   for (a=73; a<110; a++)
   {
    chan_in_word(&mbinfo[a].mb_type,Link2Input);
    chan_in_word(&mbinfo[a].motion_type,Link2Input);
    chan_in_word(&mbinfo[a].MV[0][0][0],Link2Input);
    chan_in_word(&mbinfo[a].MV[0][0][1],Link2Input);
    chan_in_word(&mbinfo[a].MV[0][1][0],Link2Input);
    chan_in_word(&mbinfo[a].MV[0][1][1],Link2Input);
    chan_in_word(&mbinfo[a].MV[1][0][0],Link2Input);
    chan_in_word(&mbinfo[a].MV[1][0][1],Link2Input);
    chan_in_word(&mbinfo[a].MV[1][1][0],Link2Input);
    chan_in_word(&mbinfo[a].MV[1][1][1],Link2Input);
    chan_in_word(&mbinfo[a].mv_field_sel[0][0],Link2Input);
    chan_in_word(&mbinfo[a].mv_field_sel[0][1],Link2Input);
    chan_in_word(&mbinfo[a].mv_field_sel[1][0],Link2Input);
    chan_in_word(&mbinfo[a].mv_field_sel[1][1],Link2Input);
   }


   for (a=0; a<110; a++)
   {
```

```
    chan_out_word(mbinfo[a].mb_type,Link0Output);
    chan_out_word(mbinfo[a] motion_type,Link0Output);
    chan_out_word(mbinfo[a].MV[0][0][0],Link0Output);
    chan_out_word(mbinfo[a].MV[0][0][1],Link0Output);
    chan_out_word(mbinfo[a].MV[0][1][0],Link0Output);
    chan_out_word(mbinfo[a].MV[0][1][1],Link0Output);
    chan_out_word(mbinfo[a].MV[1][0][0],Link0Output);
    chan_out_word(mbinfo[a].MV[1][0][1],Link0Output);
    chan_out_word(mbinfo[a].MV[1][1][0],Link0Output);
    chan_out_word(mbinfo[a].MV[1][1][1],Link0Output);
    chan_out_word(mbinfo[a].mv_field_sel[0][0],Link0Output);
    chan_out_word(mbinfo[a].mv_field_sel[0][1],Link0Output);
    chan_out_word(mbinfo[a].mv_field_sel[1][0],Link0Output);
    chan_out_word(mbinfo[a].mv_field_sel[1][1],Link0Output);
    }

  }while(loop<nframes);

  chan_in_word(&loop,Link1Input);
  chan_in_word(&loop,Link2Input);
  chan_out_word(loop,Link0Output);

}/*end of main()*/


static void frame_estimate _ANSI_ARGS_((char *org,
                              char *ref,
                              char *mb,
                              int i, int j,
                              int sx, int sy,
                              int *iminp, int *jminp,
                              int *imintp, int *jmintp,
                              int *iminbp, int *jminbp,
                              float *dframep, float *dfieldp,
                              int *tselp, int *bselp,
                              int imins[2][2], int jmins[2][2]));

static float fullsearch _ANSI_ARGS_((char *org,
                              char *ref,
                              char *blk,
                              int lx,
                              int i0,
                              int j0,
                              int sx,
                              int sy,
                              char h,
                              int xmax,
                              int ymax,
                              int *iminp,
                              int *jminp));

static float dist1 _ANSI_ARGS_((char *blk1,
                          char *blk2,
                          int lx,
```

```
                    char hx,
                    char hy,
                    char h,
                    float distlim));

static float dist2 _ANSI_ARGS_((char *blk1,
                    char *blk2,
                    int lx,
                    char hx,
                    char hy,
                    char h));

static float bdist1 _ANSI_ARGS_((char *pf,
                    char *pb,
                    char *p2,
                    int lx,
                    int hxf,
                    int hyf,
                    int hxb,
                    int hyb,
                    char h));

static float bdist2 _ANSI_ARGS_((char *pf,
                    char *pb,
                    char *p2,
                    int lx,
                    int hxf,
                    int hyf,
                    int hxb,
                    int hyb,
                    char h));

static float variance _ANSI_ARGS_((char *p, int lx));


void motion_estimation(char *oldorg, char *neworg,
              char *oldref, char *newref,
              char *cur,
              int sxf,
              int syf,
              int sxb,
              int syb,
              struct mbinfo *mbi)
{
 int i, j;
 int imin,jmin,iminf,jminf,iminr,jminr;
 int imint,jmint,iminb,jminb;
 int imintf,jmintf,iminbf,jminbf;
 int imintr,jmintr,iminbr,jminbr;
 float var,v0,
 float dmc,dmcf,dmcr,dmci,vmc,vmcf,vmcr,vmci;
 float dmcfield,dmcfieldf,dmcfieldr,dmcfieldi;
 int tsel,bsel,tself,bself,tselr,bselr;
 char *mb;
```

```
  int imins[2][2],jmins[2][2];
  int a, ii, jj;

/* loop through 36 macroblocks of the picture */

  for(a=146; a<182; a++)
  {
   jj = a>>4;
   ii = a - (jj<<4);
   j = jj<<4;
   i = ii<<4;

   mb = cur + i + width*j;

   var = variance(mb,width);

   /*  for I_type pictures     */

   if (pict_type==I_TYPE)
   mbi->mb_type = MB_INTRA;

   /*  for P_type pictures     */

   if (pict_type==P_TYPE)
   {
    frame_estimate(oldorg,oldref,mb,i,j,sxf,syf,
               &imin,&jmin,&imint,&jmint,&iminb,&jminb,
               &dmc,&dmcfield,&tsel,&bsel,imins,jmins);

    /* select between frame and field prediction */
    if (dmc<=dmcfield)
    {
     mbi->motion_type = MC_FRAME;
     vmc = dist2(oldref+(imin>>1)+width*(jmin>>1),mb,
             width,imin&1,jmin&1,16);
    }
    else
    {
     mbi->motion_type = MC_FIELD;
     dmc = dmcfield;
     vmc = dist2(oldref+(tsel?width:0)+(imint>>1)+(width<<1)*(jmint>>1),
             mb,width<<1,imint&1,jmint&1,8);
     vmc+= dist2(oldref+(bsel?width.0)+(iminb>>1)+(width<<1)*(jminb>>1),
             mb+width,width<<1,iminb&1,jminb&1,8);
    }

    if (vmc>var && vmc>=9*256)
    mbi->mb_type = MB_INTRA;
    else
    {
     v0 = dist2(oldref+i+width*j,mb,width,0,0,16);
     if (4*v0>5*vmc && v0>=9*256)
     {
      /* use MC */
```

```
  var = vmc;
  mbi->mb_type = MB_FORWARD;
  if (mbi->motion_type==MC_FRAME)
  {
   mbi->MV[0][0][0] = imin - (i<<1);
   mbi->MV[0][0][1] = jmin - (j<<1);
  }
  else /* if (mbi->motion_type==MC_FIELD) */
  {
   /* these are FRAME vectors */
   mbi->MV[0][0][0] = imint - (i<<1);
   mbi->MV[0][0][1] = (jmint<<1) - (j<<1);
   mbi->MV[1][0][0] = iminb - (i<<1);
   mbi->MV[1][0][1] = (jminb<<1) - (j<<1);
   mbi->mv_field_sel[0][0] = tsel;
   mbi->mv_field_sel[1][0] = bsel;
  }
 }
 else
 {
  /* No-MC */
  var = v0;
  mbi->mb_type = 0;
  mbi->motion_type = MC_FRAME;
  mbi->MV[0][0][0] = 0;
  mbi->MV[0][0][1] = 0;
 }
}
}


/*  for B_type pictures    */

if (pict_type==B_TYPE)
{
 /* forward prediction */
 frame_estimate(oldorg,oldref,mb,i,j,sxf,syf,
             &iminf,&jminf,&imintf,&jmintf,&iminbf,&jminbf,
             &dmcf,&dmcfieldf,&tself,&bself,imins,jmins);

 /* backward prediction */
 frame_estimate(neworg,newref,mb,i,j,sxb,syb,
             &iminr,&jminr,&imintr,&jmintr,&iminbr,&jminbr,
             &dmcr,&dmcfieldr,&tselr,&bselr,imins,jmins);

 /* calculate interpolated distance */
 /* frame */
 dmci = bdist1(oldref+(iminf>>1)+width*(jminf>>1),
            newref+(iminr>>1)+width*(jminr>>1),
            mb,width,iminf&1,jminf&1,iminr&1,jminr&1,16);

 /* top field
 dmcfieldi = bdist1(oldref+(imintf>>1)+(tself?width:0)+(width<<1)*(jmintf>>1),
                newref+(imintr>>1)+(tselr?width:0)+(width<<1)*(jmintr>>1),
```

```
        mb,width<<1,imintf&1,jmintf&1,imintr&1,jmintr&1,8);

/* bottom field
dmcfieldi+= bdist1(oldref+(iminbf>>1)+(bself?width:0)+(width<<1)*(jminbf>>1),
        newref+(iminbr>>1)+(bselr?width:0)+(width<<1)*(jminbr>>1),
        mb+width,width<<1,iminbf&1,jminbf&1,iminbr&1,jminbr&1,8);


if (dmci<dmcfieldi && dmci<dmcf && dmci<dmcfieldf
   && dmci<dmcr && dmci<dmcfieldr)
{
 /* frame, interpolated */
 mbi->mb_type = MB_FORWARD|MB_BACKWARD;
 mbi->motion_type = MC_FRAME;
 vmc = bdist2(oldref+(iminf>>1)+width*(jminf>>1),
        newref+(iminr>>1)+width*(jminr>>1),
        mb,width,iminf&1,jminf&1,iminr&1,jminr&1,16);
}
else if (dmcfieldi<dmcf && dmcfieldi<dmcfieldf
      && dmcfieldi<dmcr && dmcfieldi<dmcfieldr)
{
 /* field, interpolated */
 mbi->mb_type = MB_FORWARD|MB_BACKWARD;
 mbi->motion_type = MC_FIELD;
 vmc = bdist2(oldref+(imintf>>1)+(tself?width:0)+(width<<1)*(jmintf>>1),
        newref+(imintr>>1)+(tselr?width:0)+(width<<1)*(jmintr>>1),
        mb,width<<1,imintf&1,jmintf&1,imintr&1,jmintr&1,8);
 vmc+= bdist2(oldref+(iminbf>>1)+(bself?width:0)+(width<<1)*(jminbf>>1),
        newref+(iminbr>>1)+(bselr?width:0)+(width<<1)*(jminbr>>1),
        mb+width,width<<1,iminbf&1,jminbf&1,iminbr&1,jminbr&1,8);
}
else if (dmcf<dmcfieldf && dmcf<dmcr && dmcf<dmcfieldr)
{
 /* frame, forward */
 mbi->mb_type = MB_FORWARD;
 mbi->motion_type = MC_FRAME;
 vmc = dist2(oldref+(iminf>>1)+width*(jminf>>1),mb,
        width,iminf&1,jminf&1,16);
}
else if (dmcfieldf<dmcr && dmcfieldf<dmcfieldr)
{
 /* field, forward */
 mbi->mb_type = MB_FORWARD;
 mbi->motion_type = MC_FIELD;
 vmc = dist2(oldref+(tself?width:0)+(imintf>>1)+(width<<1)*(jmintf>>1),
        mb,width<<1,imintf&1,jmintf&1,8);
 vmc+= dist2(oldref+(bself?width:0)+(iminbf>>1)+(width<<1)*(jminbf>>1),
        mb+width,width<<1,iminbf&1,jminbf&1,8);
}
else if (dmcr<dmcfieldr)
{
 /* frame, backward */
 mbi->mb_type = MB_BACKWARD,
 mbi->motion_type = MC_FRAME;
```

```
   vmc = dist2(newref+(iminr>>1)+width*(jminr>>1),mb,
            width,iminr&1,jminr&1,16);
  }
  else
  {
   /* field, backward */
   mbi->mb_type = MB_BACKWARD;
   mbi->motion_type = MC_FIELD,
   vmc = dist2(newref+(tselr?width:0)+(imintr>>1)+(width<<1)*(jmintr>>1),
            mb,width<<1,imintr&1,jmintr&1,8);
   vmc+= dist2(newref+(bselr?width:0)+(iminbr>>1)+(width<<1)*(jminbr>>1),
            mb+width,width<<1,iminbr&1,jminbr&1,8);
  }


  if (vmc>var && vmc>=9*256)
  mbi->mb_type = MB_INTRA;
  else
  {
   var = vmc;
   if (mbi->motion_type==MC_FRAME)
   {
    /* forward */
    mbi->MV[0][0][0] = iminf - (i<<1);
    mbi->MV[0][0][1] = jminf - (j<<1);
    /* backward */
    mbi->MV[0][1][0] = iminr - (i<<1);
    mbi->MV[0][1][1] = jminr - (j<<1);
   }
   else
   {
    /* these are FRAME vectors */
    /* forward */
    mbi->MV[0][0][0] = imintf - (i<<1);
    mbi->MV[0][0][1] = (jmintf<<1) - (j<<1),
    mbi->MV[1][0][0] = iminbf - (i<<1);
    mbi->MV[1][0][1] = (jminbf<<1) - (j<<1);
    mbi->mv_field_sel[0][0] = tself;
    mbi->mv_field_sel[1][0] = bself;
    /* backward */
    mbi->MV[0][1][0] = imintr - (i<<1);
    mbi->MV[0][1][1] = (jmintr<<1) - (j<<1);
    mbi->MV[1][1][0] = iminbr - (i<<1);
    mbi->MV[1][1][1] = (jminbr<<1) - (j<<1);
    mbi->mv_field_sel[0][1] = tselr;
    mbi->mv_field_sel[1][1] = bselr;
   }
  }
 }

 mbi->var = var;

 mbi++;
```

```
  }

}


static void frame_estimate(char *org, char *ref,
                char *mb,
                int i, int j,
                int sx, int sy,
                int *iminp, int *jminp,
                int *imintp, int *jmintp,
                int *iminbp, int *jminbp,
                float *dframep, float *dfieldp,
                int *tselp, int *bselp,
                int imins[2][2], int jmins[2][2] )
{
 float dt,db,dmint,dminb;
 int imint,iminb,jmint,jminb;

 /* frame prediction   */
 *dframep = fullsearch(org,ref,mb,width,i,j,sx,sy,16,width,height,
              iminp,jminp);

 /* predict top field from top field   */
 dt = fullsearch(org,ref,mb,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
         &imint,&jmint);

 /* predict top field from bottom field   */
 db = fullsearch(org+width,ref+width,mb,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
         &iminb,&jminb);

 imins[0][0] = imint;
 jmins[0][0] = jmint;
 imins[1][0] = iminb;
 jmins[1][0] = jminb;

 /* select prediction for top field */
 if (dt<=db)
 {
  dmint=dt; *imintp=imint; *jmintp=jmint; *tselp=0;
 }
 else
 {
  dmint=db; *imintp=iminb; *jmintp=jminb; *tselp=1;
 }

 /* predict bottom field from top field   */
 dt = fullsearch(org,ref,mb+width,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
         &imint,&jmint);

 /* predict bottom field from bottom field   */
 db = fullsearch(org+width, ref+width, mb+width,
         width<<1, i, j>>1, sx, sy>>1, 8, width, height>>1,
         &iminb,&jminb),
```

```
  imins[0][1] = imint;
  jmins[0][1] = jmint;
  imins[1][1] = iminb;
  jmins[1][1] = jminb;

  /* select prediction for bottom field */
  if (db<=dt)
  {
    dminb=db; *iminbp=iminb; *jminbp=jminb; *bselp=1;
  }
  else
  {
    dminb=dt; *iminbp=imint; *jminbp=jmint; *bselp=0;
  }

  *dfieldp=dmint+dminb;
}


static float fullsearch(char *org, char *ref,
                        char *blk,
                        int lx,
                        int i0,
                        int j0,
                        int sx,
                        int sy,
                        char h,
                        int xmax,
                        int ymax,
                        int *iminp,
                        int *jminp)
{
  int i,j,imin,jmin,ilow,ihigh,jlow,jhigh;
  float d,dmin;
  int k,l,sxy;
  int ii, jj;

  ilow = i0 - sx;
  ihigh = i0 + sx;

  if (ilow<0) ilow = 0;

  if (ihigh>xmax-16) ihigh = xmax-16;

  jlow = j0 - sy;
  jhigh = j0 + sy;

  if (jlow<0) jlow = 0;

  if (jhigh>ymax-h) jhigh = ymax-h;

  /* full pel search, spiraling outwards */
```

```
imin = i0;
jmin = j0;
dmin = dist1(org+imin+lx*jmin,blk,lx,0,0,h,65536.0);

sxy = (sx>sy) ? sx : sy;

for (l=1, l<=sxy; l++)
{
 i = i0 - l;
 j = j0 - l;
 for (k=0, k<8*l; k++)
 {
  if (i>=ilow && i<=ihigh && j>=jlow && j<=jhigh)
  {
   d = dist1(org+i+lx*j,blk,lx,0,0,h,dmin);

   if (d<dmin)
   {
    dmin = d;
    imin = i;
    jmin = j;
   }
  }

  if      (k<2*l) i++;
  else if (k<4*l) j++;
  else if (k<6*l) i--;
  else            j--;
 }
}

/* half pel */
dmin = 65536.0;
imin = imin << 1;
jmin = jmin << 1;
ilow = imin - (imin>0);
ihigh = imin + (imin<((xmax-16)<<1));
jlow = jmin - (jmin>0);
jhigh = jmin + (jmin<((ymax-h)<<1)),

for (j=jlow; j<=jhigh; j++)
 for (i=ilow; i<=ihigh; i++)
 {
  ii = i >> 1 ;
  jj = j >> 1 ;
  d = dist1(ref+ii+lx*jj,blk,lx,i&1,j&1,h,dmin),

   if (d<dmin)
   {
    dmin = d;
    imin = i;
    jmin = j;
   }
 }
```

```
 *iminp = imin;
 *jminp = jmin;

 return dmin;
}


static float dist1(char *blk1,
                   char *blk2,
                   int lx,
                   char hx,
                   char hy,
                   char h,
                   float distlim )
{
 char *p1,*p1a,*p2;
 char i, j;
 int v;
 float s;

 s = 0;
 p1 = blk1;
 p2 = blk2;

 if (!hx && !hy)
  for (j=0; j<h, j++)
  {
   if ((v = p1[0]  - p2[0])<0)  v = -v; s+= (float) v;
   if ((v = p1[1]  - p2[1])<0)  v = -v; s+= (float) v;
   if ((v = p1[2]  - p2[2])<0)  v = -v; s+= (float) v;
   if ((v = p1[3]  - p2[3])<0)  v = -v; s+= (float) v;
   if ((v = p1[4]  - p2[4])<0)  v = -v; s+= (float) v;
   if ((v = p1[5]  - p2[5])<0)  v = -v; s+= (float) v;
   if ((v = p1[6]  - p2[6])<0)  v = -v; s+= (float) v;
   if ((v = p1[7]  - p2[7])<0)  v = -v; s+= (float) v;
   if ((v = p1[8]  - p2[8])<0)  v = -v; s+= (float) v;
   if ((v = p1[9]  - p2[9])<0)  v = -v; s+= (float) v;
   if ((v = p1[10] - p2[10])<0) v = -v; s+= (float) v;
   if ((v = p1[11] - p2[11])<0) v = -v; s+= (float) v;
   if ((v = p1[12] - p2[12])<0) v = -v; s+= (float) v;
   if ((v = p1[13] - p2[13])<0) v = -v; s+= (float) v;
   if ((v = p1[14] - p2[14])<0) v = -v; s+= (float) v;
   if ((v = p1[15] - p2[15])<0) v = -v; s+= (float) v;

   if (s >= distlim)
    break;

   p1+= lx;
   p2+= lx;
  }
 else if (hx && !hy)
  for (j=0; j<h; j++)
  {
```

```
      for (i=0; i<16; i++)
      {
        v = (int) ((p1[i]+p1[i+1]+1) / 2 - p2[i]);
        if (v>=0) s+= (float) v;
        else s-= (float) v;
      }
      p1+= lx;
      p2+= lx;
    }
  else if (!hx && hy)
  {
    p1a = p1 + lx;
    for (j=0; j<h; j++)
    {
      for (i=0; i<16; i++)
      {
        v = (int ) ((p1[i]+p1a[i]+1) / 2 - p2[i]);
        if (v>=0)
          s+= (float) v;
        else
          s-= (float) v;
      }
      p1 = p1a;
      p1a+= lx;
      p2+= lx;
    }
  }
  else /* if (hx && hy) */
  {
    p1a = p1 + lx;
    for (j=0; j<h; j++)
    {
      for (i=0; i<16; i++)
      {
        v = (int) ((p1[i]+p1[i+1]+p1a[i]+p1a[i+1]+2) / 4 - p2[i]);
        if (v>=0)
          s+= (float) v;
        else
          s-= (float) v;
      }
      p1 = p1a;
      p1a+= lx;
      p2+= lx;
    }
  }

  return s;
}


static float dist2(char *blk1,
                   char *blk2,
                   int lx,
                   char hx,
```

```
                char hy,
                char h )
{
 char *p1,*p1a,*p2,
 char i, j;
 int v;
 float s;

 s = 0;
 p1 = blk1;
 p2 = blk2;
 if (!hx && !hy)
  for (j=0; j<h; j++)
  {
   for (i=0; i<16; i++)
   {
    v = p1[i] - p2[i];
    s+= (float) (v*v);
   }
   p1+= lx;
   p2+= lx;
  }
 else if (hx && !hy)
  for (j=0; j<h; j++)
  {
   for (i=0; i<16; i++)
   {
    v = (int) ((p1[i]+p1[i+1]+1) / 2 - p2[i]);
    s+= (float) (v*v);
   }
   p1+= lx;
   p2+= lx;
  }
 else if (!hx && hy)
 {
  p1a = p1 + lx;
  for (j=0; j<h; j++)
  {
   for (i=0; i<16, i++)
   {
    v = (int) ((p1[i]+p1a[i]+1) / 2 - p2[i]),
    s+= (float) (v*v);
   }
   p1 = p1a;
   p1a+= lx;
   p2+= lx;
  }
 }
 else /* if (hx && hy) */
 {
  p1a = p1 + lx;
  for (j=0; j<h; j++)
  {
   for (i=0; i<16; i++)
```

```
    {
     v = (int) ((p1[i]+p1[i+1]+p1a[i]+p1a[i+1]+2) / 4 - p2[i]),
     s+= (float) (v*v);
    }
    p1 = p1a;
    p1a+= lx;
    p2+= lx;
   }
  }

  return s;
}


static float bdist1(char *pf, char *pb, char *p2,
                    int lx, int hxf,
                    int hyf, int hxb,
                    int hyb, char h )
{
  char *pfa,*pfb,*pfc,*pba,*pbb,*pbc;
  char i, j;
  int v;
  float s;

  pfa = pf + hxf;
  pfb = pf + lx*hyf;
  pfc = pfb + hxf;

  pba = pb + hxb;
  pbb = pb + lx*hyb;
  pbc = pbb + hxb;

  s = 0;

  for (j=0; j<h; j++)
  {
   for (i=0; i<16, i++)
   {
    v = (int) (((*pf++ + *pfa++ + *pfb++ + *pfc++ + 2) / 4 +
               (*pb++ + *pba++ + *pbb++ + *pbc++ + 2) / 4 + 1) / 2
              - *p2++);

    if (v>=0)
      s+= (float) v;
    else
      s-= (float) v;
   }
   p2+= lx-16;
   pf+= lx-16;
   pfa+= lx-16;
   pfb+= lx-16;
   pfc+= lx-16;
   pb+= lx-16;
   pba+= lx-16;
```

```
    pbb+= lx-16;
    pbc+= lx-16;
  }

  return s;
}


static float bdist2(char *pf, char *pb, char *p2,
                    int lx, int hxf,
                    int hyf, int hxb,
                    int hyb, char h )
{
  char *pfa,*pfb,*pfc,*pba,*pbb,*pbc;
  char i, j;
  int v;
  float s;

  pfa = pf + hxf;
  pfb = pf + lx*hyf;
  pfc = pfb + hxf;

  pba = pb + hxb;
  pbb = pb + lx*hyb;
  pbc = pbb + hxb;

  s = 0;

  for (j=0; j<h; j++)
  {
    for (i=0; i<16; i++)
    {
      v = (int) (((*pf++ + *pfa++ + *pfb++ + *pfc++ + 2) / 4 +
               (*pb++ + *pba++ + *pbb++ + *pbc++ + 2) / 4 + 1) / 2
              - *p2++);
      s+= (float) (v*v);
    }
    p2+= lx-16,
    pf+= lx-16;
    pfa+= lx-16;
    pfb+= lx-16;
    pfc+= lx-16;
    pb+= lx-16;
    pba+= lx-16;
    pbb+= lx-16;
    pbc+= lx-16;
  }

  return s;
}


static float variance(char *p, int lx )
{
```

```
  char i, j;
  int v;
  float s, s2;

  s = s2 = 0;

  for (j=0; j<16, j++)
  {
   for (i=0; i<16; i++)
   {
    v = *p++;
    s+= (float) v;
    s2+= (float) (v*v),
   }
   p+= lx-16;
  }
  s=0.0625*s;
  return s2 - s*s;
}
```

```
/********************************/
/*                              */
/*      m021.c residing at PE5  */
/*                              */
/********************************/

#include "c:\tc2v2\dos.h"
#include "c:\tc2v2\float.h"
#include "c:\tc2v2\math.h"
#include "c:\tc2v2\ctype.h"
#include "c:\tc2v2\stdio h"
#include "c:\tc2v2\stdlib.h"
#include "c:\tc2v2\string.h"
#include "c:\tc2v2\chan.h"
#include "c:\tc2v2\thread.h"
#include "c:\tc2v2\time.h"
#include "c:\tc2v2\errno.h"

#define GLOBAL /* used by global.h */

#include "c:\nick\h\const1.h"
#include "c:\nick\h\config h"
#include "c:\nick\h\global.h"

static void init _ANSI_ARGS_((void));

char  *neworg[1],

void main(argc,argv,envp,ins,in_ports,outs,out_ports)
int argc,ins,outs;
char *argv[],*envp[];
CHAN *in_ports[],*out_ports[];
{
 /* read parameter file */
 chan_in_word(&M,Link0Input);
 chan_in_word(&nframes,Link0Input);

 init();
 putseq();

}

static void init()
{
 int i, size, a;

 chan_in_word(&mb_width,Link0Input);
 chan_in_word(&mb_height,Link0Input);
 chan_in_word(&mb_height2,Link0Input);
 chan_in_word(&width,Link0Input);
 chan_in_word(&height,Link0Input);
 chan_in_word(&width2,Link0Input);
 chan_in_word(&height2,Link0Input);
```

```
  size = width*height ;

  newrefframe[0] = (char *)malloc(size);
  oldrefframe[0] = (char *)malloc(size);
  neworgframe[0] = (char *)malloc(size);
  oldorgframe[0] = (char *)malloc(size);
  neworg[0] = (char *)malloc(size);

  mbinfo = (struct mbinfo *)malloc(37*sizeof(struct mbinfo));

  for (a=0; a<37; a++)
  {
   mbinfo[a].mb_type = mbinfo[a].motion_type = 0;
   mbinfo[a].MV[0][0][0] = mbinfo[a].MV[0][0][1] = mbinfo[a].MV[0][1][0] = 0;
   mbinfo[a].MV[0][1][1] = mbinfo[a].MV[1][0][0] = mbinfo[a].MV[1][0][1] = 0;
   mbinfo[a].MV[1][1][0] = mbinfo[a].MV[1][1][1] = 0;
   mbinfo[a].mv_field_sel[0][0] = mbinfo[a].mv_field_sel[0][1] = 0;
   mbinfo[a].mv_field_sel[1][0] = mbinfo[a].mv_field_sel[1][1] = 0;
  }

}


void putseq()
{
  /* this routine assumes (N % M) == 0 */
  int i, j, k, f, f0, n, np, nb;
  int a;
  int sxf, syf, sxb, syb;
  int size, loop;

  size = width*height;


/* loop through all frames in encoding/decoding order */
do{
    chan_in_word(&loop,Link0Input);

    if(loop==nframes) break;

    chan_in_word(&pict_type,Link0Input);
    chan_in_word(&sxf,Link0Input);
    chan_in_word(&syf,Link0Input),
    chan_in_word(&sxb,Link0Input);
    chan_in_word(&syb,Link0Input);

    if ( pict_type != I_TYPE )
    {
     chan_in_message(size,neworg[0],Link0Input);
     chan_in_message(size,oldorgframe[0],Link0Input);
     chan_in_message(size,oldrefframe[0],Link0Input);

     if ( pict_type != P_TYPE )
     {
```

```
     chan_in_message(size,neworgframe[0],Link0Input);
     chan_in_message(size,newrefframe[0],Link0Input);
    }
   }

/* loop through 37 macroblocks of the picture */

   motion_estimation(oldorgframe[0], neworgframe[0],
               oldrefframe[0], newrefframe[0],
               neworg[0],
               sxf, syf,
               sxb, syb,
               mbinfo);

    for (a=0; a<37; a++)
    {
     chan_out_word(mbinfo[a].mb_type,Link0Output);
     chan_out_word(mbinfo[a].motion_type,Link0Output);
     chan_out_word(mbinfo[a].MV[0][0][0],Link0Output);
     chan_out_word(mbinfo[a].MV[0][0][1],Link0Output);
     chan_out_word(mbinfo[a].MV[0][1][0],Link0Output);
     chan_out_word(mbinfo[a].MV[0][1][1],Link0Output),
     chan_out_word(mbinfo[a].MV[1][0][0],Link0Output),
     chan_out_word(mbinfo[a] MV[1][0][1],Link0Output),
     chan_out_word(mbinfo[a].MV[1][1][0],Link0Output);
     chan_out_word(mbinfo[a].MV[1][1][1],Link0Output);
     chan_out_word(mbinfo[a].mv_field_sel[0][0],Link0Output);
     chan_out_word(mbinfo[a].mv_field_sel[0][1],Link0Output);
     chan_out_word(mbinfo[a].mv_field_sel[1][0],Link0Output);
     chan_out_word(mbinfo[a].mv_field_sel[1][1],Link0Output);
    }

  }while(loop<nframes);

  chan_out_word(loop,Link0Output);

}


static void frame_estimate _ANSI_ARGS_((char *org,
                              char *ref,
                              char *mb,
                              int i, int j,
                              int sx, int sy,
                              int *iminp, int *jminp,
                              int *imintp, int *jmintp,
                              int *iminbp, int *jminbp,
                              float *dframep, float *dfieldp,
                              int *tselp, int *bselp,
                              int imins[2][2], int jmins[2][2]));

static float fullsearch _ANSI_ARGS_((char *org,
                           char *ref,
                           char *blk,
```

```
                      int lx,
                      int i0,
                      int j0,
                      int sx,
                      int sy,
                      char h,
                      int xmax,
                      int ymax,
                      int *iminp,
                      int *jminp));

static float dist1 _ANSI_ARGS_((char *blk1,
                    char *blk2,
                    int lx,
                    char hx,
                    char hy,
                    char h,
                    float distlim));

static float dist2 _ANSI_ARGS_((char *blk1,
                    char *blk2,
                    int lx,
                    char hx,
                    char hy,
                    char h));

static float bdist1 _ANSI_ARGS_((char *pf,
                     char *pb,
                     char *p2,
                     int lx,
                     int hxf,
                     int hyf,
                     int hxb,
                     int hyb,
                     char h));

static float bdist2 _ANSI_ARGS_((char *pf,
                     char *pb,
                     char *p2,
                     int lx,
                     int hxf,
                     int hyf,
                     int hxb,
                     int hyb,
                     char h));

static float variance _ANSI_ARGS_((char *p, int lx));


void motion_estimation(char *oldorg, char *neworg,
              char *oldref, char *newref,
              char *cur,
              int sxf,
              int syf,
```

```
                    int sxb,
                    int syb,
                    struct mbinfo *mbi)
{
 int i, j, p, q;
 int imin,jmin,iminf,jminf,iminr,jminr;
 int imint,jmint,iminb,jminb;
 int imintf,jmintf,iminbf,jminbf;
 int imintr,jmintr,iminbr,jminbr;
 float var,v0,
 float dmc,dmcf,dmcr,dmci,vmc,vmcf,vmcr,vmci;
 float dmcfield,dmcfieldf,dmcfieldr,dmcfieldi;
 int tsel,bsel,tself,bself,tselr,bselr;
 char *mb;
 int imins[2][2],jmins[2][2];
 int a, ii, jj;

/* loop through 37 macroblocks of the picture */

 for(a=182; a<219; a++)
 {
  jj = a>>4;
  ii = a - (jj<<4);
  j = jj<<4;
  i = ii<<4;

  mb = cur + i + width*j;

  var = variance(mb,width);

  /*  for I_type pictures     */

  if (pict_type==I_TYPE)
  mbi->mb_type = MB_INTRA;

  /*  for P_type pictures     */

  if (pict_type==P_TYPE)
  {
   frame_estimate(oldorg,oldref,mb,i,j,sxf,syf,
              &imin,&jmin,&imint,&jmint,&iminb,&jminb,
              &dmc,&dmcfield,&tsel,&bsel,imins,jmins);

   /* select between frame and field prediction */
   if (dmc<=dmcfield)
   {
    mbi->motion_type = MC_FRAME;
    vmc = dist2(oldref+(imin>>1)+width*(jmin>>1),mb,
             width,imin&1,jmin&1,16);
   }
   else
   {
    mbi->motion_type = MC_FIELD,
    dmc = dmcfield;
```

```
  vmc = dist2(oldref+(tsel?width:0)+(imint>>1)+(width<<1)*(jmint>>1),
            mb,width<<1,imint&1,jmint&1,8);
  vmc+= dist2(oldref+(bsel?width:0)+(iminb>>1)+(width<<1)*(jminb>>1),
            mb+width,width<<1,iminb&1,jminb&1,8);
  }

 if (vmc>var && vmc>=9*256)
 mbi->mb_type = MB_INTRA;
 else
 {
  v0 = dist2(oldref+i+width*j,mb,width,0,0,16);
  if (4*v0>5*vmc && v0>=9*256)
  {
   /* use MC */
   var = vmc;
   mbi->mb_type = MB_FORWARD;
   if (mbi->motion_type==MC_FRAME)
   {
    mbi->MV[0][0][0] = imin - (i<<1);
    mbi->MV[0][0][1] = jmin - (j<<1);
   }
   else /* if (mbi->motion_type==MC_FIELD) */
   {
    /* these are FRAME vectors */
    mbi->MV[0][0][0] = imint - (i<<1);
    mbi->MV[0][0][1] = (jmint<<1) - (j<<1);
    mbi->MV[1][0][0] = iminb - (i<<1);
    mbi->MV[1][0][1] = (jminb<<1) - (j<<1);
    mbi->mv_field_sel[0][0] = tsel;
    mbi->mv_field_sel[1][0] = bsel;
   }
  }
  else
  {
   /* No-MC */
   var = v0;
   mbi->mb_type = 0;
   mbi->motion_type = MC_FRAME;
   mbi->MV[0][0][0] = 0;
   mbi->MV[0][0][1] = 0;
  }
 }
}


/*  for B_type pictures     */

if (pict_type==B_TYPE)
{
 /* forward prediction */
 frame_estimate(oldorg,oldref,mb,i,j,sxf,syf,
            &iminf,&jminf,&imintf,&jmintf,&iminbf,&jminbf,
            &dmcf,&dmcfieldf,&tself,&bself,imins,jmins);
```

```
/* backward prediction */
frame_estimate(neworg,newref,mb,i,j,sxb,syb,
          &iminr,&jminr,&imintr,&jmintr,&iminbr,&jminbr,
          &dmcr,&dmcfieldr,&tselr,&bselr,imins,jmins);

/* calculate interpolated distance */
/* frame */
dmci = bdist1(oldref+(iminf>>1)+width*(jminf>>1),
        newref+(iminr>>1)+width*(jminr>>1),
        mb,width,iminf&1,jminf&1,iminr&1,jminr&1,16);

/* top field
dmcfieldi = bdist1(oldref+(imintf>>1)+(tself?width:0)+(width<<1)*(jmintf>>1),
            newref+(imintr>>1)+(tselr?width:0)+(width<<1)*(jmintr>>1),
            mb,width<<1,imintf&1,jmintf&1,imintr&1,jmintr&1,8);

/* bottom field
dmcfieldi+= bdist1(oldref+(iminbf>>1)+(bself?width:0)+(width<<1)*(jminbf>>1),
            newref+(iminbr>>1)+(bselr?width:0)+(width<<1)*(jminbr>>1),
            mb+width,width<<1,iminbf&1,jminbf&1,iminbr&1,jminbr&1,8);


if (dmci<dmcfieldi && dmci<dmcf && dmci<dmcfieldf
   && dmci<dmcr && dmci<dmcfieldr)
{
 /* frame, interpolated */
 mbi->mb_type = MB_FORWARD|MB_BACKWARD;
 mbi->motion_type = MC_FRAME;
 vmc = bdist2(oldref+(iminf>>1)+width*(jminf>>1),
        newref+(iminr>>1)+width*(jminr>>1),
        mb,width,iminf&1,jminf&1,iminr&1,jminr&1,16);
}
else if (dmcfieldi<dmcf && dmcfieldi<dmcfieldf
      && dmcfieldi<dmcr && dmcfieldi<dmcfieldr)
{
 /* field, interpolated */
 mbi->mb_type = MB_FORWARD|MB_BACKWARD;
 mbi->motion_type = MC_FIELD;
 vmc = bdist2(oldref+(imintf>>1)+(tself?width:0)+(width<<1)*(jmintf>>1),
        newref+(imintr>>1)+(tselr?width:0)+(width<<1)*(jmintr>>1),
        mb,width<<1,imintf&1,jmintf&1,imintr&1,jmintr&1,8);
 vmc+= bdist2(oldref+(iminbf>>1)+(bself?width:0)+(width<<1)*(jminbf>>1),
        newref+(iminbr>>1)+(bselr?width:0)+(width<<1)*(jminbr>>1),
        mb+width,width<<1,iminbf&1,jminbf&1,iminbr&1,jminbr&1,8);
}
else if (dmcf<dmcfieldf && dmcf<dmcr && dmcf<dmcfieldr)
{
 /* frame, forward */
 mbi->mb_type = MB_FORWARD;
 mbi->motion_type = MC_FRAME;
 vmc = dist2(oldref+(iminf>>1)+width*(jminf>>1),mb,
       width,iminf&1,jminf&1,16);
}
else if (dmcfieldf<dmcr && dmcfieldf<dmcfieldr)
```

```
{
 /* field, forward */
 mbi->mb_type = MB_FORWARD;
 mbi->motion_type = MC_FIELD;
 vmc = dist2(oldref+(tself?width:0)+(imintf>>1)+(width<<1)*(jmintf>>1),
         mb,width<<1,imintf&1,jmintf&1,8);
 vmc+= dist2(oldref+(bself?width:0)+(iminbf>>1)+(width<<1)*(jminbf>>1),
         mb+width,width<<1,iminbf&1,jminbf&1,8);
}
else if (dmcr<dmcfieldr)
{
 /* frame, backward */
 mbi->mb_type = MB_BACKWARD;
 mbi->motion_type = MC_FRAME;
 vmc = dist2(newref+(iminr>>1)+width*(jminr>>1),mb,
         width,iminr&1,jminr&1,16);
}
else
{
 /* field, backward */
 mbi->mb_type = MB_BACKWARD;
 mbi->motion_type = MC_FIELD;
 vmc = dist2(newref+(tselr?width:0)+(imintr>>1)+(width<<1)*(jmintr>>1),
         mb,width<<1,imintr&1,jmintr&1,8);
 vmc+= dist2(newref+(bselr?width:0)+(iminbr>>1)+(width<<1)*(jminbr>>1),
         mb+width,width<<1,iminbr&1,jminbr&1,8);
}


if (vmc>var && vmc>=9*256)
mbi->mb_type = MB_INTRA;
else
{
 var = vmc;
 if (mbi->motion_type==MC_FRAME)
 {
  /* forward */
  mbi->MV[0][0][0] = iminf - (i<<1);
  mbi->MV[0][0][1] = jminf - (j<<1);
  /* backward */
  mbi->MV[0][1][0] = iminr - (i<<1);
  mbi->MV[0][1][1] = jminr - (j<<1);
 }
 else
 {
  /* these are FRAME vectors */
  /* forward */
  mbi->MV[0][0][0] = imintf - (i<<1);
  mbi->MV[0][0][1] = (jmintf<<1) - (j<<1);
  mbi->MV[1][0][0] = iminbf - (i<<1);
  mbi->MV[1][0][1] = (jminbf<<1) - (j<<1);
  mbi->mv_field_sel[0][0] = tself;
  mbi->mv_field_sel[1][0] = bself;
  /* backward */
```

```
      mbi->MV[0][1][0] = imintr - (i<<1);
      mbi->MV[0][1][1] = (jmintr<<1) - (j<<1);
      mbi->MV[1][1][0] = iminbr - (i<<1);
      mbi->MV[1][1][1] = (jminbr<<1) - (j<<1);
      mbi->mv_field_sel[0][1] = tselr;
      mbi->mv_field_sel[1][1] = bselr;
     }
    }
   }

   mbi->var = var;

   mbi++;

  }
}


static void frame_estimate(char *org, char *ref,
                     char *mb,
                     int i, int j,
                     int sx, int sy,
                     int *iminp, int *jminp,
                     int *imintp, int *jmintp,
                     int *iminbp, int *jminbp,
                     float *dframep, float *dfieldp,
                     int *tselp, int *bselp,
                     int imins[2][2], int jmins[2][2] )
{
 float dt,db,dmint,dminb;
 int imint,iminb,jmint,jminb,

 /* frame prediction   */
 *dframep = fullsearch(org,ref,mb,width,i,j,sx,sy,16,width,height,
                  iminp,jminp);

 /* predict top field from top field   */
 dt = fullsearch(org,ref,mb,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
             &imint,&jmint);

 /* predict top field from bottom field  */
 db = fullsearch(org+width,ref+width,mb,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
             &iminb,&jminb);

 imins[0][0] = imint;
 jmins[0][0] = jmint;
 imins[1][0] = iminb;
 jmins[1][0] = jminb;

 /* select prediction for top field */
 if (dt<=db)
 {
  dmint=dt; *imintp=imint; *jmintp=jmint; *tselp=0;
 }
```

```
  else
  {
   dmint=db; *imintp=iminb; *jmintp=jminb; *tselp=1;
  }

  /* predict bottom field from top field    */
  dt = fullsearch(org,ref,mb+width,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
              &imint,&jmint);

  /* predict bottom field from bottom field   */
  db = fullsearch(org+width, ref+width, mb+width,
              width<<1, i, j>>1, sx, sy>>1, 8, width, height>>1,
              &iminb,&jminb);

  imins[0][1] = imint;
  jmins[0][1] = jmint;
  imins[1][1] = iminb;
  jmins[1][1] = jminb;

  /* select prediction for bottom field */
  if (db<=dt)
  {
   dminb=db; *iminbp=iminb; *jminbp=jminb; *bselp=1;
  }
  else
  {
   dminb=dt; *iminbp=imint; *jminbp=jmint; *bselp=0;
  }

  *dfieldp=dmint+dminb;
}


static float fullsearch(char *org, char *ref,
                    char *blk,
                    int lx,
                    int i0,
                    int j0,
                    int sx,
                    int sy,
                    char h,
                    int xmax,
                    int ymax,
                    int *iminp,
                    int *jminp)
{
  int i,j,imin,jmin,ilow,ihigh,jlow,jhigh;
  float d,dmin;
  int k,l,sxy;
  int ii, jj;

  ilow = i0 - sx;
  ihigh = i0 + sx;
```

```
if (ilow<0) ilow = 0;

if (ihigh>xmax-16) ihigh = xmax-16;

jlow = j0 - sy;
jhigh = j0 + sy;

if (jlow<0) jlow = 0;

if (jhigh>ymax-h) jhigh = ymax-h,

/* full pel search, spiraling outwards */

imin = i0;
jmin = j0;
dmin = dist1(org+imin+lx*jmin,blk,lx,0,0,h,65536.0);

sxy = (sx>sy) ? sx : sy;

for (l=1; l<=sxy; l++)
{
 i = i0 - l;
 j = j0 - l;
 for (k=0; k<8*l; k++)
 {
  if (i>=ilow && i<=ihigh && j>=jlow && j<=jhigh)
  {
   d = dist1(org+i+lx*j,blk,lx,0,0,h,dmin);

   if (d<dmin)
   {
    dmin = d;
    imin = i;
    jmin = j;
   }
  }

  if      (k<2*l) i++;
  else if (k<4*l) j++;
  else if (k<6*l) i--;
  else            j--;
 }
}

/* half pel */
dmin = 65536.0;
imin = imin << 1;
jmin = jmin << 1;
ilow = imin - (imin>0);
ihigh = imin + (imin<((xmax-16)<<1));
jlow = jmin - (jmin>0),
jhigh = jmin + (jmin<((ymax-h)<<1)),

for (j=jlow; j<=jhigh; j++)
```

```
    for (i=ilow; i<=ihigh, i++)
    {
     ii = i >> 1 ;
     jj = j >> 1 ;
     d = dist1(ref+ii+lx*jj,blk,lx,i&1,j&1,h,dmin);

      if (d<dmin)
      {
        dmin = d;
        imin = i;
        jmin = j;
      }
    }

  *iminp = imin;
  *jminp = jmin;

  return dmin,
}


static float dist1(char *blk1,
                   char *blk2,
                   int lx,
                   char hx,
                   char hy,
                   char h,
                   float distlim )
{
  char *p1,*p1a,*p2;
  char i, j;
  int v;
  float s;

  s = 0;
  p1 = blk1;
  p2 = blk2;

  if (!hx && !hy)
    for (j=0; j<h; j++)
    {
      if ((v = p1[0]  - p2[0])<0)  v = -v; s+= (float) v;
      if ((v = p1[1]  - p2[1])<0)  v = -v; s+= (float) v;
      if ((v = p1[2]  - p2[2])<0)  v = -v; s+= (float) v;
      if ((v = p1[3]  - p2[3])<0)  v = -v; s+= (float) v;
      if ((v = p1[4]  - p2[4])<0)  v = -v; s+= (float) v;
      if ((v = p1[5]  - p2[5])<0)  v = -v; s+= (float) v;
      if ((v = p1[6]  - p2[6])<0)  v = -v; s+= (float) v;
      if ((v = p1[7]  - p2[7])<0)  v = -v; s+= (float) v;
      if ((v = p1[8]  - p2[8])<0)  v = -v; s+= (float) v;
      if ((v = p1[9]  - p2[9])<0)  v = -v; s+= (float) v;
      if ((v = p1[10] - p2[10])<0) v = -v; s+= (float) v,
      if ((v = p1[11] - p2[11])<0) v = -v; s+= (float) v,
      if ((v = p1[12] - p2[12])<0) v = -v; s+= (float) v;
```

```
      if ((v = p1[13] - p2[13])<0) v = -v; s+= (float) v;
      if ((v = p1[14] - p2[14])<0) v = -v; s+= (float) v,
      if ((v = p1[15] - p2[15])<0) v = -v; s+= (float) v;

      if (s >= distlim)
        break;

      p1+= lx;
      p2+= lx;
    }
  else if (hx && !hy)
    for (j=0; j<h; j++)
    {
      for (i=0, i<16; i++)
      {
        v = (int) ((p1[i]+p1[i+1]+1) / 2 - p2[i]);
        if (v>=0) s+= (float) v;
        else s-= (float) v;
      }
      p1+= lx;
      p2+= lx;
    }
  else if (!hx && hy)
  {
    p1a = p1 + lx;
    for (j=0; j<h; j++)
    {
      for (i=0; i<16; i++)
      {
        v = (int ) ((p1[i]+p1a[i]+1) / 2 - p2[i]);
        if (v>=0)
          s+= (float) v;
        else
          s-= (float) v;
      }
      p1 = p1a;
      p1a+= lx;
      p2+= lx;
    }
  }
  else /* if (hx && hy) */
  {
    p1a = p1 + lx;
    for (j=0; j<h; j++)
    {
      for (i=0; i<16; i++)
      {
        v = (int) ((p1[i]+p1[i+1]+p1a[i]+p1a[i+1]+2) / 4 - p2[i]);
        if (v>=0)
          s+= (float) v;
        else
          s-= (float) v;
      }
      p1 = p1a;
```

```
    p1a+= lx;
    p2+= lx;
   }
  }

  return s;
}


static float dist2(char *blk1,
                   char *blk2,
                   int lx,
                   char hx,
                   char hy,
                   char h )
{
  char *p1,*p1a,*p2;
  char i, j;
  int v;
  float s;

  s = 0;
  p1 = blk1;
  p2 = blk2;
  if (!hx && !hy)
   for (j=0; j<h; j++)
   {
    for (i=0; i<16; i++)
    {
     v = p1[i] - p2[i];
     s+= (float) (v*v);
    }
    p1+= lx;
    p2+= lx;
   }
  else if (hx && !hy)
   for (j=0; j<h; j++)
   {
    for (i=0; i<16; i++)
    {
     v = (int) ((p1[i]+p1[i+1]+1) / 2 - p2[i]);
     s+= (float) (v*v);
    }
    p1+= lx;
    p2+= lx;
   }
  else if (!hx && hy)
  {
   p1a = p1 + lx;
   for (j=0; j<h; j++)
   {
    for (i=0; i<16; i++)
    {
     v = (int) ((p1[i]+p1a[i]+1) / 2 - p2[i]);
```

```
      s+= (float) (v*v);
    }
    p1 = p1a;
    p1a+= lx,
    p2+= lx;
  }
 }
 else /* if (hx && hy) */
 {
  p1a = p1 + lx;
  for (j=0, j<h; j++)
  {
   for (i=0; i<16; i++)
   {
    v = (int) ((p1[i]+p1[i+1]+p1a[i]+p1a[i+1]+2) / 4 - p2[i]);
    s+= (float) (v*v),
   }
   p1 = p1a;
   p1a+= lx;
   p2+= lx;
  }
 }

 return s;
}


static float bdist1(char *pf, char *pb, char *p2,
                    int lx, int hxf,
                    int hyf, int hxb,
                    int hyb, char h )
{
 char *pfa,*pfb,*pfc,*pba,*pbb,*pbc,
 char i, j;
 int v;
 float s;

 pfa = pf + hxf;
 pfb = pf + lx*hyf;
 pfc = pfb + hxf;

 pba = pb + hxb;
 pbb = pb + lx*hyb;
 pbc = pbb + hxb;

 s = 0;

 for (j=0, j<h; j++)
 {
  for (i=0; i<16; i++)
  {
   v = (int) (((*pf++ + *pfa++ + *pfb++ + *pfc++ + 2) / 4 +
             (*pb++ + *pba++ + *pbb++ + *pbc++ + 2) / 4 + 1) / 2
            - *p2++);
```

```
    if (v>=0)
      s+= (float) v;
    else
      s-= (float) v;
   }
   p2+= lx-16;
   pf+= lx-16;
   pfa+= lx-16;
   pfb+= lx-16;
   pfc+= lx-16;
   pb+= lx-16;
   pba+= lx-16;
   pbb+= lx-16;
   pbc+= lx-16;
  }

  return s;
}


static float bdist2(char *pf, char *pb, char *p2,
                    int lx, int hxf,
                    int hyf, int hxb,
                    int hyb, char h )
{
  char *pfa,*pfb,*pfc,*pba,*pbb,*pbc;
  char i, j;
  int v;
  float s;

  pfa = pf + hxf;
  pfb = pf + lx*hyf;
  pfc = pfb + hxf;

  pba = pb + hxb;
  pbb = pb + lx*hyb;
  pbc = pbb + hxb;

  s = 0;

  for (j=0, j<h; j++)
  {
   for (i=0; i<16; i++)
   {
    v = (int) (((*pf++ + *pfa++ + *pfb++ + *pfc++ + 2) / 4 +
               (*pb++ + *pba++ + *pbb++ + *pbc++ + 2) / 4 + 1) / 2
               - *p2++);
    s+= (float) (v*v);
   }
   p2+= lx-16,
   pf+= lx-16;
   pfa+= lx-16;
   pfb+= lx-16;
```

```
    pfc+= lx-16;
    pb+= lx-16;
    pba+= lx-16;
    pbb+= lx-16;
    pbc+= lx-16;
  }

  return s;
}


static float variance(char *p, int lx )
{
  char i, j;
  int v;
  float s, s2;

  s = s2 = 0;

  for (j=0; j<16; j++)
  {
    for (i=0; i<16; i++)
    {
      v = *p++;
      s+= (float) v;
      s2+= (float) (v*v);
    }
    p+= lx-16;
  }
  s=0.0625*s;
  return s2 - s*s;
}
```

```
/*******************************/
/*                             */
/*      m022.c residing at PE6 */
/*                             */
/*******************************/

#include "c:\tc2v2\dos.h"
#include "c:\tc2v2\float.h"
#include "c:\tc2v2\math.h"
#include "c:\tc2v2\ctype.h"
#include "c:\tc2v2\stdio.h"
#include "c:\tc2v2\stdlib.h"
#include "c:\tc2v2\string.h"
#include "c:\tc2v2\chan.h"
#include "c:\tc2v2\thread.h"
#include "c:\tc2v2\time.h"
#include "c:\tc2v2\errno.h"

#define GLOBAL /* used by global.h */

#include "c:\nick\h\const1.h"
#include "c:\nick\h\config.h"
#include "c:\nick\h\global.h"

static void init _ANSI_ARGS_((void));

char *neworg[1];

void main(argc,argv,envp,ins,in_ports,outs,out_ports)
int argc,ins,outs;
char *argv[],*envp[];
CHAN *in_ports[],*out_ports[];
{
 /* read parameter file */
 chan_in_word(&M,Link0Input);
 chan_in_word(&nframes,Link0Input);

 init();
 putseq();

}

static void init()
{
 int i, size, a;

 chan_in_word(&mb_width,Link0Input);
 chan_in_word(&mb_height,Link0Input);
 chan_in_word(&mb_height2,Link0Input);
 chan_in_word(&width,Link0Input),
 chan_in_word(&height,Link0Input);
 chan_in_word(&width2,Link0Input);
 chan_in_word(&height2,Link0Input);
```

```
  size = width*height ;

  newrefframe[0] = (char *)malloc(size);
  oldrefframe[0] = (char *)malloc(size);
  neworgframe[0] = (char *)malloc(size);
  oldorgframe[0] = (char *)malloc(size);
  neworg[0] = (char *)malloc(size);

  mbinfo = (struct mbinfo *)malloc(37*sizeof(struct mbinfo));

  for (a=0; a<37; a++)
  {
   mbinfo[a].mb_type = mbinfo[a].motion_type = 0;
   mbinfo[a].MV[0][0][0] = mbinfo[a].MV[0][0][1] = mbinfo[a].MV[0][1][0] = 0;
   mbinfo[a].MV[0][1][1] = mbinfo[a].MV[1][0][0] = mbinfo[a].MV[1][0][1] = 0;
   mbinfo[a].MV[1][1][0] = mbinfo[a].MV[1][1][1] = 0;
   mbinfo[a].mv_field_sel[0][0] = mbinfo[a].mv_field_sel[0][1] = 0;
   mbinfo[a].mv_field_sel[1][0] = mbinfo[a].mv_field_sel[1][1] = 0,
  }

}


void putseq()
{
/* this routine assumes (N % M) == 0 */
  int i, j, k, f, f0, n, np, nb;
  int a;
  int sxf, syf, sxb, syb;
  int size, loop;

  size = width*height;


/* loop through all frames in encoding/decoding order */
do{
   chan_in_word(&loop,Link0Input);

   if(loop==nframes) break;

   chan_in_word(&pict_type,Link0Input);
   chan_in_word(&sxf,Link0Input);
   chan_in_word(&syf,Link0Input);
   chan_in_word(&sxb,Link0Input);
   chan_in_word(&syb,Link0Input);

   if ( pict_type != I_TYPE )
   {
    chan_in_message(size,neworg[0],Link0Input);
    chan_in_message(size,oldorgframe[0],Link0Input);
    chan_in_message(size,oldrefframe[0],Link0Input);
    if ( pict_type != P_TYPE )
    {
     chan_in_message(size,neworgframe[0],Link0Input);
```

```
    chan_in_message(size,newrefframe[0],Link0Input),
   }
  }

/* loop through 37 macroblocks of the picture */

   motion_estimation(oldorgframe[0], neworgframe[0],
                 oldrefframe[0], newrefframe[0],
                 neworg[0],
                 sxf, syf,
                 sxb, syb,
                 mbinfo);

    for (a=0; a<37; a++)
    {
     chan_out_word(mbinfo[a].mb_type,Link0Output);
     chan_out_word(mbinfo[a].motion_type,Link0Output);
     chan_out_word(mbinfo[a].MV[0][0][0],Link0Output);
     chan_out_word(mbinfo[a].MV[0][0][1],Link0Output);
     chan_out_word(mbinfo[a].MV[0][1][0],Link0Output);
     chan_out_word(mbinfo[a] MV[0][1][1],Link0Output);
     chan_out_word(mbinfo[a].MV[1][0][0],Link0Output);
     chan_out_word(mbinfo[a].MV[1][0][1],Link0Output);
     chan_out_word(mbinfo[a].MV[1][1][0],Link0Output);
     chan_out_word(mbinfo[a].MV[1][1][1],Link0Output);
     chan_out_word(mbinfo[a].mv_field_sel[0][0],Link0Output);
     chan_out_word(mbinfo[a].mv_field_sel[0][1],Link0Output);
     chan_out_word(mbinfo[a].mv_field_sel[1][0],Link0Output);
     chan_out_word(mbinfo[a].mv_field_sel[1][1],Link0Output);
    }

  }while(loop<nframes);

  chan_out_word(loop,Link0Output);

}

static void frame_estimate _ANSI_ARGS_((char *org,
                              char *ref,
                              char *mb,
                              int i, int j,
                              int sx, int sy,
                              int *iminp, int *jminp,
                              int *imintp, int *jmintp,
                              int *iminbp, int *jminbp,
                              float *dframep, float *dfieldp,
                              int *tselp, int *bselp,
                              int imins[2][2], int jmins[2][2]));

static float fullsearch _ANSI_ARGS_((char *org,
                            char *ref,
                            char *blk,
                            int lx,
                            int i0,
```

```
                    int j0,
                    int sx,
                    int sy,
                    char h,
                    int xmax,
                    int ymax,
                    int *iminp,
                    int *jminp));

static float dist1 _ANSI_ARGS_((char *blk1,
                    char *blk2,
                    int lx,
                    char hx,
                    char hy,
                    char h,
                    float distlim));

static float dist2 _ANSI_ARGS_((char *blk1,
                    char *blk2,
                    int lx,
                    char hx,
                    char hy,
                    char h));

static float bdist1 _ANSI_ARGS_((char *pf,
                    char *pb,
                    char *p2,
                    int lx,
                    int hxf,
                    int hyf,
                    int hxb,
                    int hyb,
                    char h));

static float bdist2 _ANSI_ARGS_((char *pf,
                    char *pb,
                    char *p2,
                    int lx,
                    int hxf,
                    int hyf,
                    int hxb,
                    int hyb,
                    char h));

static float variance _ANSI_ARGS_((char *p, int lx));


void motion_estimation(char *oldorg, char *neworg,
            char *oldref, char *newref,
            char *cur,
            int sxf,
            int syf,
            int sxb,
            int syb,
```

```
                struct mbinfo *mbi)
{
 int i, j, p, q;
 int imin,jmin,iminf,jminf,iminr,jminr;
 int imint,jmint,iminb,jminb;
 int imintf,jmintf,iminbf,jminbf;
 int imintr,jmintr,iminbr,jminbr;
 float var,v0;
 float dmc,dmcf,dmcr,dmci,vmc,vmcf,vmcr,vmci;
 float dmcfield,dmcfieldf,dmcfieldr,dmcfieldi;
 int tsel,bsel,tself,bself,tselr,bselr;
 char *mb;
 int imins[2][2],jmins[2][2];
 int a, ii, jj;

/* loop through 37 macroblocks of the picture */

 for(a=219; a<256; a++)
 {
  jj = a>>4;
  ii = a - (jj<<4);
  j = jj<<4;
  i = ii<<4;

  mb = cur + i + width*j;

  var = variance(mb,width);

  /*  for I_type pictures    */

  if (pict_type==I_TYPE)
  mbi->mb_type = MB_INTRA;

  /*  for P_type pictures     */

  if (pict_type==P_TYPE)
  {
   frame_estimate(oldorg,oldref,mb,i,j,sxf,syf,
              &imin,&jmin,&imint,&jmint,&iminb,&jminb,
              &dmc,&dmcfield,&tsel,&bsel,imins,jmins);

   /* select between frame and field prediction */
   if (dmc<=dmcfield)
   {
    mbi->motion_type = MC_FRAME;
    vmc = dist2(oldref+(imin>>1)+width*(jmin>>1),mb,
            width,imin&1,jmin&1,16),
   }
   else
   {
    mbi->motion_type = MC_FIELD,
    dmc = dmcfield;
    vmc = dist2(oldref+(tsel?width:0)+(imint>>1)+(width<<1)*(jmint>>1),
            mb,width<<1,imint&1,jmint&1,8);
```

```
  vmc+= dist2(oldref+(bsel?width.0)+(iminb>>1)+(width<<1)*(jminb>>1),
          mb+width,width<<1,iminb&1,jminb&1,8);
 }

 if (vmc>var && vmc>=9*256)
 mbi->mb_type = MB_INTRA;
 else
 {
  v0 = dist2(oldref+i+width*j,mb,width,0,0,16);
  if (4*v0>5*vmc && v0>=9*256)
  {
   /* use MC */
   var = vmc;
   mbi->mb_type = MB_FORWARD;
   if (mbi->motion_type==MC_FRAME)
   {
    mbi->MV[0][0][0] = imin - (i<<1);
    mbi->MV[0][0][1] = jmin - (j<<1);
   }
   else /* if (mbi->motion_type==MC_FIELD) */
   {
    /* these are FRAME vectors */
    mbi->MV[0][0][0] = imint - (i<<1);
    mbi->MV[0][0][1] = (jmint<<1) - (j<<1);
    mbi->MV[1][0][0] = iminb - (i<<1);
    mbi->MV[1][0][1] = (jminb<<1) - (j<<1);
    mbi->mv_field_sel[0][0] = tsel;
    mbi->mv_field_sel[1][0] = bsel;
   }
  }
  else
  {
   /* No-MC */
   var = v0;
   mbi->mb_type = 0;
   mbi->motion_type = MC_FRAME;
   mbi->MV[0][0][0] = 0;
   mbi->MV[0][0][1] = 0;
  }
 }
}


/*  for B_type pictures     */

if (pict_type==B_TYPE)
{
 /* forward prediction */
 frame_estimate(oldorg,oldref,mb,i,j,sxf,syf,
           &iminf,&jminf,&imintf,&jmintf,&iminbf,&jminbf,
           &dmcf,&dmcfieldf,&tself,&bself,imins,jmins);

 /* backward prediction */
 frame_estimate(neworg,newref,mb,i,j,sxb,syb,
```

```
        &iminr,&jminr,&imintr,&jmintr,&iminbr,&jminbr,
        &dmcr,&dmcfieldr,&tselr,&bselr,imins,jmins);

/* calculate interpolated distance */
/* frame */
dmci = bdist1(oldref+(iminf>>1)+width*(jminf>>1),
       newref+(iminr>>1)+width*(jminr>>1),
       mb,width,iminf&1,jminf&1,iminr&1,jminr&1,16);

/* top field
dmcfieldi = bdist1(oldref+(imintf>>1)+(tself?width:0)+(width<<1)*(jmintf>>1),
            newref+(imintr>>1)+(tselr?width:0)+(width<<1)*(jmintr>>1),
            mb,width<<1,imintf&1,jmintf&1,imintr&1,jmintr&1,8);

/* bottom field
dmcfieldi+= bdist1(oldref+(iminbf>>1)+(bself?width:0)+(width<<1)*(jminbf>>1),
            newref+(iminbr>>1)+(bselr?width:0)+(width<<1)*(jminbr>>1),
            mb+width,width<<1,iminbf&1,jminbf&1,iminbr&1,jminbr&1,8);

/* select prediction type of minimum distance from the
 * six candidates (field/frame * forward/backward/interpolated)
 */

if (dmci<dmcfieldi && dmci<dmcf && dmci<dmcfieldf
   && dmci<dmcr && dmci<dmcfieldr)
{
 /* frame, interpolated */
 mbi->mb_type = MB_FORWARD|MB_BACKWARD;
 mbi->motion_type = MC_FRAME;
 vmc = bdist2(oldref+(iminf>>1)+width*(jminf>>1),
        newref+(iminr>>1)+width*(jminr>>1),
        mb,width,iminf&1,jminf&1,iminr&1,jminr&1,16);
}
else if (dmcfieldi<dmcf && dmcfieldi<dmcfieldf
      && dmcfieldi<dmcr && dmcfieldi<dmcfieldr)
{
 /* field, interpolated */
 mbi->mb_type = MB_FORWARD|MB_BACKWARD;
 mbi->motion_type = MC_FIELD;
 vmc = bdist2(oldref+(imintf>>1)+(tself?width:0)+(width<<1)*(jmintf>>1),
        newref+(imintr>>1)+(tselr?width:0)+(width<<1)*(jmintr>>1),
        mb,width<<1,imintf&1,jmintf&1,imintr&1,jmintr&1,8);
 vmc+= bdist2(oldref+(iminbf>>1)+(bself?width:0)+(width<<1)*(jminbf>>1),
        newref+(iminbr>>1)+(bselr?width:0)+(width<<1)*(jminbr>>1),
        mb+width,width<<1,iminbf&1,jminbf&1,iminbr&1,jminbr&1,8);
}
else if (dmcf<dmcfieldf && dmcf<dmcr && dmcf<dmcfieldr)
{
 /* frame, forward */
 mbi->mb_type = MB_FORWARD;
 mbi->motion_type = MC_FRAME;
 vmc = dist2(oldref+(iminf>>1)+width*(jminf>>1),mb,
        width,iminf&1,jminf&1,16);
}
```

```
else if (dmcfieldf<dmcr && dmcfieldf<dmcfieldr)
{
 /* field, forward */
 mbi->mb_type = MB_FORWARD;
 mbi->motion_type = MC_FIELD;
 vmc = dist2(oldref+(tself?width:0)+(imintf>>1)+(width<<1)*(jmintf>>1),
         mb,width<<1,imintf&1,jmintf&1,8);
 vmc+= dist2(oldref+(bself?width:0)+(iminbf>>1)+(width<<1)*(jminbf>>1),
         mb+width,width<<1,iminbf&1,jminbf&1,8);
}
else if (dmcr<dmcfieldr)
{
 /* frame, backward */
 mbi->mb_type = MB_BACKWARD;
 mbi->motion_type = MC_FRAME;
 vmc = dist2(newref+(iminr>>1)+width*(jminr>>1),mb,
         width,iminr&1,jminr&1,16);
}
else
{
 /* field, backward */
 mbi->mb_type = MB_BACKWARD;
 mbi->motion_type = MC_FIELD;
 vmc = dist2(newref+(tselr?width:0)+(imintr>>1)+(width<<1)*(jmintr>>1),
         mb,width<<1,imintr&1,jmintr&1,8);
 vmc+= dist2(newref+(bselr?width:0)+(iminbr>>1)+(width<<1)*(jminbr>>1),
         mb+width,width<<1,iminbr&1,jminbr&1,8);
}


if (vmc>var && vmc>=9*256)
mbi->mb_type = MB_INTRA;
else
{
 var = vmc;
 if (mbi->motion_type==MC_FRAME)
 {
  /* forward */
  mbi->MV[0][0][0] = iminf - (i<<1),
  mbi->MV[0][0][1] = jminf - (j<<1);
  /* backward */
  mbi->MV[0][1][0] = iminr - (i<<1);
  mbi->MV[0][1][1] = jminr - (j<<1),
 }
 else
 {
  /* these are FRAME vectors */
  /* forward */
  mbi->MV[0][0][0] = imintf - (i<<1);
  mbi->MV[0][0][1] = (jmintf<<1) - (j<<1);
  mbi->MV[1][0][0] = iminbf - (i<<1);
  mbi->MV[1][0][1] = (jminbf<<1) - (j<<1);
  mbi->mv_field_sel[0][0] = tself;
  mbi->mv_field_sel[1][0] = bself;
```

```
    /* backward */
    mbi->MV[0][1][0] = imintr - (i<<1);
    mbi->MV[0][1][1] = (jmintr<<1) - (j<<1);
    mbi->MV[1][1][0] = iminbr - (i<<1);
    mbi->MV[1][1][1] = (jminbr<<1) - (j<<1);
    mbi->mv_field_sel[0][1] = tselr;
    mbi->mv_field_sel[1][1] = bselr,
   }
  }
 }

 mbi->var = var,

 mbi++;

 }
}


static void frame_estimate(char *org, char *ref,
                        char *mb,
                        int i, int j,
                        int sx, int sy,
                        int *iminp, int *jminp,
                        int *imintp, int *jmintp,
                        int *iminbp, int *jminbp,
                        float *dframep, float *dfieldp,
                        int *tselp, int *bselp,
                        int imins[2][2], int jmins[2][2] )
{
 float dt,db,dmint,dminb;
 int imint,iminb,jmint,jminb;

 /* frame prediction   */
 *dframep = fullsearch(org,ref,mb,width,i,j,sx,sy,16,width,height,
                 iminp,jminp);

 /* predict top field from top field   */
 dt = fullsearch(org,ref,mb,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
           &imint,&jmint);

 /* predict top field from bottom field  */
 db = fullsearch(org+width,ref+width,mb,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
           &iminb,&jminb);

 imins[0][0] = imint;
 jmins[0][0] = jmint;
 imins[1][0] = iminb;
 jmins[1][0] = jminb;

 /* select prediction for top field */
 if (dt<=db)
 {
  dmint=dt; *imintp=imint; *jmintp=jmint; *tselp=0;
```

```
  }
  else
  {
   dmint=db; *imintp=iminb; *jmintp=jminb; *tselp=1;
  }

  /* predict bottom field from top field    */
  dt = fullsearch(org,ref,mb+width,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
              &imint,&jmint);

  /* predict bottom field from bottom field   */
  db = fullsearch(org+width, ref+width, mb+width,
              width<<1, i, j>>1, sx, sy>>1, 8, width, height>>1,
              &iminb,&jminb);

  imins[0][1] = imint;
  jmins[0][1] = jmint;
  imins[1][1] = iminb;
  jmins[1][1] = jminb;

  /* select prediction for bottom field */
  if (db<=dt)
  {
   dminb=db, *iminbp=iminb; *jminbp=jminb; *bselp=1;
  }
  else
  {
   dminb=dt; *iminbp=imint; *jminbp=jmint; *bselp=0;
  }

  *dfieldp=dmint+dminb;
}


static float fullsearch(char *org, char *ref,
                   char *blk,
                   int lx,
                   int i0,
                   int j0,
                   int sx,
                   int sy,
                   char h,
                   int xmax,
                   int ymax,
                   int *iminp,
                   int *jminp)
{
 int i,j,imin,jmin,ilow,ihigh,jlow,jhigh;
 float d,dmin;
 int k,l,sxy;
 int ii, jj;

 ilow = i0 - sx;
 ihigh = i0 + sx;
```

```
 if (ilow<0) ilow = 0,

 if (ihigh>xmax-16) ihigh = xmax-16;

 jlow = j0 - sy;
 jhigh = j0 + sy;

 if (jlow<0) jlow = 0;

 if (jhigh>ymax-h) jhigh = ymax-h;

 /* full pel search, spiraling outwards */

 imin = i0;
 jmin = j0;
 dmin = dist1(org+imin+lx*jmin,blk,lx,0,0,h,65536.0);

 sxy = (sx>sy) ? sx : sy;

 for (l=1; l<=sxy; l++)
 {
  i = i0 - l;
  j = j0 - l;
  for (k=0; k<8*l; k++)
  {
   if (i>=ilow && i<=ihigh && j>=jlow && j<=jhigh)
   {
    d = dist1(org+i+lx*j,blk,lx,0,0,h,dmin);

    if (d<dmin)
    {
     dmin = d;
     imin = i;
     jmin = j;
    }
   }

   if      (k<2*l) i++;
   else if (k<4*l) j++;
   else if (k<6*l) i--;
   else            j--;
  }
 }

 /* half pel */
 dmin = 65536.0;
 imin = imin << 1;
 jmin = jmin << 1;
 ilow = imin - (imin>0);
 ihigh = imin + (imin<((xmax-16)<<1));
 jlow = jmin - (jmin>0);
 jhigh = jmin + (jmin<((ymax-h)<<1));
```

```
  for (j=jlow; j<=jhigh; j++)
    for (i=ilow; i<=ihigh; i++)
    {
     ii = i >> 1 ;
     jj = j >> 1 ,
     d = dist1(ref+ii+lx*jj,blk,lx,i&1,j&1,h,dmin);

      if (d<dmin)
      {
        dmin = d;
        imin = i;
        jmin = j;
      }
    }

  *iminp = imin;
  *jminp = jmin;

  return dmin;
}


static float dist1(char *blk1,
                   char *blk2,
                   int lx,
                   char hx,
                   char hy,
                   char h,
                   float distlim )
{
  char *p1,*p1a,*p2;
  char i, j;
  int v;
  float s;

  s = 0,
  p1 = blk1;
  p2 = blk2;

  if (!hx && !hy)
    for (j=0; j<h; j++)
    {
      if ((v = p1[0]  - p2[0])<0)  v = -v; s+= (float) v,
      if ((v = p1[1]  - p2[1])<0)  v = -v; s+= (float) v;
      if ((v = p1[2]  - p2[2])<0)  v = -v; s+= (float) v;
      if ((v = p1[3]  - p2[3])<0)  v = -v; s+= (float) v;
      if ((v = p1[4]  - p2[4])<0)  v = -v; s+= (float) v;
      if ((v = p1[5]  - p2[5])<0)  v = -v; s+= (float) v;
      if ((v = p1[6]  - p2[6])<0)  v = -v; s+= (float) v;
      if ((v = p1[7]  - p2[7])<0)  v = -v; s+= (float) v;
      if ((v = p1[8]  - p2[8])<0)  v = -v; s+= (float) v;
      if ((v = p1[9]  - p2[9])<0)  v = -v; s+= (float) v;
      if ((v = p1[10] - p2[10])<0) v = -v; s+= (float) v;
      if ((v = p1[11] - p2[11])<0) v = -v, s+= (float) v;
```

```
    if ((v = p1[12] - p2[12])<0) v = -v; s+= (float) v,
    if ((v = p1[13] - p2[13])<0) v = -v, s+= (float) v,
    if ((v = p1[14] - p2[14])<0) v = -v; s+= (float) v;
    if ((v = p1[15] - p2[15])<0) v = -v, s+= (float) v;

    if (s >= distlim)
      break;

    p1+= lx;
    p2+= lx;
  }
else if (hx && !hy)
  for (j=0; j<h, j++)
  {
    for (i=0; i<16; i++)
    {
      v = (int) ((p1[i]+p1[i+1]+1) / 2 - p2[i]);
      if (v>=0) s+= (float) v;
      else s-= (float) v;
    }
    p1+= lx;
    p2+= lx;
  }
else if (!hx && hy)
{
  p1a = p1 + lx;
  for (j=0; j<h; j++)
  {
    for (i=0; i<16; i++)
    {
      v = (int ) ((p1[i]+p1a[i]+1) / 2 - p2[i]);
      if (v>=0)
        s+= (float) v;
      else
        s-= (float) v;
    }
    p1 = p1a;
    p1a+= lx;
    p2+= lx;
  }
}
else /* if (hx && hy) */
{
  p1a = p1 + lx;
  for (j=0; j<h; j++)
  {
    for (i=0; i<16; i++)
    {
      v = (int) ((p1[i]+p1[i+1]+p1a[i]+p1a[i+1]+2) / 4 - p2[i]);
      if (v>=0)
        s+= (float) v;
      else
        s-= (float) v;
    }
```

```
      p1 = p1a;
      p1a+= lx;
      p2+= lx;
    }
  }

  return s,
}


static float dist2(char *blk1,
                   char *blk2,
                   int lx,
                   char hx,
                   char hy,
                   char h )
{
  char *p1,*p1a,*p2;
  char i, j;
  int v;
  float s;

  s = 0;
  p1 = blk1;
  p2 = blk2;
  if (!hx && !hy)
    for (j=0; j<h; j++)
    {
      for (i=0; i<16; i++)
      {
        v = p1[i] - p2[i];
        s+= (float) (v*v);
      }
      p1+= lx;
      p2+= lx;
    }
  else if (hx && !hy)
    for (j=0; j<h; j++)
    {
      for (i=0; i<16; i++)
      {
        v = (int) ((p1[i]+p1[i+1]+1) / 2 - p2[i]);
        s+= (float) (v*v);
      }
      p1+= lx;
      p2+= lx;
    }
  else if (!hx && hy)
  {
    p1a = p1 + lx;
    for (j=0; j<h; j++)
    {
      for (i=0; i<16; i++)
      {
```

```
      v = (int) ((p1[i]+p1a[i]+1) / 2 - p2[i]);
      s+= (float) (v*v);
    }
    p1 = p1a;
    p1a+= lx;
    p2+= lx;
  }
 }
 else /* if (hx && hy) */
 {
  p1a = p1 + lx;
  for (j=0, j<h; j++)
  {
   for (i=0; i<16; i++)
   {
    v = (int) ((p1[i]+p1[i+1]+p1a[i]+p1a[i+1]+2) / 4 - p2[i]);
    s+= (float) (v*v);
   }
   p1 = p1a;
   p1a+= lx;
   p2+= lx;
  }
 }

 return s;
}


static float bdist1(char *pf, char *pb, char *p2,
                    int lx, int hxf,
                    int hyf, int hxb,
                    int hyb, char h )
{
 char *pfa,*pfb,*pfc,*pba,*pbb,*pbc;
 char i, j;
 int v;
 float s;

 pfa = pf + hxf;
 pfb = pf + lx*hyf;
 pfc = pfb + hxf;

 pba = pb + hxb;
 pbb = pb + lx*hyb;
 pbc = pbb + hxb;

 s = 0;

 for (j=0; j<h; j++)
 {
  for (i=0; i<16; i++)
  {
   v = (int) (((*pf++ + *pfa++ + *pfb++ + *pfc++ + 2) / 4 +
         (*pb++ + *pba++ + *pbb++ + *pbc++ + 2) / 4 + 1) / 2
```

```
              - *p2++);

      if (v>=0)
        s+= (float) v,
      else
        s-= (float) v;
    }
    p2+= lx-16;
    pf+= lx-16;
    pfa+= lx-16;
    pfb+= lx-16;
    pfc+= lx-16;
    pb+= lx-16;
    pba+= lx-16,
    pbb+= lx-16;
    pbc+= lx-16;
  }

  return s;
}


static float bdist2(char *pf, char *pb, char *p2,
                int lx, int hxf,
                int hyf, int hxb,
                int hyb, char h )
{
  char *pfa,*pfb,*pfc,*pba,*pbb,*pbc;
  char i, j;
  int v;
  float s;

  pfa = pf + hxf;
  pfb = pf + lx*hyf;
  pfc = pfb + hxf;

  pba = pb + hxb;
  pbb = pb + lx*hyb,
  pbc = pbb + hxb;

  s = 0;

  for (j=0; j<h; j++)
  {
    for (i=0; i<16; i++)
    {
      v = (int) (((*pf++ + *pfa++ + *pfb++ + *pfc++ + 2) / 4 +
              (*pb++ + *pba++ + *pbb++ + *pbc++ + 2) / 4 + 1) / 2
              - *p2++);
      s+= (float) (v*v);
    }
    p2+= lx-16;
    pf+= lx-16;
    pfa+= lx-16,
```

```
    pfb+= lx-16;
    pfc+= lx-16,
    pb+= lx-16;
    pba+= lx-16;
    pbb+= lx-16;
    pbc+= lx-16;
  }

  return s;
}


static float variance(char *p, int lx )
{
  char i, j;
  int v;
  float s, s2;

  s = s2 = 0;

  for (j=0; j<16; j++)
  {
   for (i=0; i<16; i++)
   {
    v = *p++;
    s+= (float) v;
    s2+= (float) (v*v);
   }
   p+= lx-16;
  }
  s=0.0625*s;
  return s2 - s*s,
}
```

```
/*********************************/
/*                               */
/*        t0.c residing at Root  */
/*                               */
/*********************************/

#include "c:\tc2v2\dos.h"
#include "c:\tc2v2\float.h"
#include "c:\tc2v2\math.h"
#include "c:\tc2v2\ctype.h"
#include "c:\tc2v2\stdio.h"
#include "c:\tc2v2\stdlib.h"
#include "c:\tc2v2\string.h"
#include "c:\tc2v2\chan.h"
#include "c:\tc2v2\thread.h"
#include "c:\tc2v2\time.h"
#include "c:\tc2v2\errno.h"

#define GLOBAL /* used by global.h */

#include "c:\nick\h\const1.h"
#include "c:\nick\h\config.h"
#include "c:\nick\h\global.h"

static void init _ANSI_ARGS_((void));
static void readparmfile _ANSI_ARGS_((char *fname));

void main(argc,argv,envp,ins,in_ports,outs,out_ports)
int argc,ins,outs;
char *argv[],*envp[];
CHAN *in_ports[],*out_ports[];
{
 if (argc!=2)
 {
  printf("Usage: mpeg2encode in.par\n");
  exit(0);
 }

 /* read parameter file */
 readparmfile(argv[1]);

 init();
 putseq();

}

void error(char *text)
{
 fprintf(stderr,text),
 putc('\n',stderr);
 exit(1);
}

static void readparmfile(char *fname)
```

```
{
 int i;
 int h,m,s,f;
 FILE *fd;
 char line[256];
 int r,Xi,Xb,Xp,d0i,d0p,d0b; /* rate control */
 double avg_act; /* rate control */

 sprintf(line,"c:\\nick\\par\\%s",fname);
 if (!(fd = fopen(line,"r")))
 {
  sprintf(errortext,"Couldn't open parameter file %s",line);
  error(errortext);
 }

 fgets(id_string,254,fd);
 fgets(line,254,fd); sscanf(line,"%s",tplorg);
 fgets(line,254,fd); sscanf(line,"%s",tplref);
 fgets(line,254,fd); sscanf(line,"%s",iqname);
 fgets(line,254,fd); sscanf(line,"%s",niqname);
 fgets(line,254,fd), sscanf(line,"%s",statname);
 fgets(line,254,fd), sscanf(line,"%d",&inputtype);
 fgets(line,254,fd); sscanf(line,"%d",&nframes);
 fgets(line,254,fd); sscanf(line,"%d",&frame0);
 fgets(line,254,fd); sscanf(line,"%d:%d:%d:%d",&h,&m,&s,&f);
 fgets(line,254,fd); sscanf(line,"%d",&N);
 fgets(line,254,fd); sscanf(line,"%d",&M);
 fgets(line,254,fd); sscanf(line,"%d",&mpeg1);
 fgets(line,254,fd); sscanf(line,"%d",&fieldpic),
 fgets(line,254,fd); sscanf(line,"%d",&horizontal_size);
 fgets(line,254,fd); sscanf(line,"%d",&vertical_size);
 fgets(line,254,fd); sscanf(line,"%d",&aspectratio);
 fgets(line,254,fd); sscanf(line,"%d",&frame_rate_code);
 fgets(line,254,fd); sscanf(line,"%lf",&bit_rate);
 fgets(line,254,fd); sscanf(line,"%d",&vbv_buffer_size);
 fgets(line,254,fd); sscanf(line,"%d",&low_delay);
 fgets(line,254,fd); sscanf(line,"%d",&constrparms);
 fgets(line,254,fd); sscanf(line,"%d",&profile);
 fgets(line,254,fd); sscanf(line,"%d",&level);
 fgets(line,254,fd); sscanf(line,"%d",&prog_seq);
 fgets(line,254,fd); sscanf(line,"%d",&chroma_format);
 fgets(line,254,fd); sscanf(line,"%d",&video_format);
 fgets(line,254,fd); sscanf(line,"%d",&color_primaries);
 fgets(line,254,fd); sscanf(line,"%d",&transfer_characteristics),
 fgets(line,254,fd); sscanf(line,"%d",&matrix_coefficients);
 fgets(line,254,fd); sscanf(line,"%d",&display_horizontal_size);
 fgets(line,254,fd); sscanf(line,"%d",&display_vertical_size);
 fgets(line,254,fd); sscanf(line,"%d",&dc_prec);
 fgets(line,254,fd); sscanf(line,"%d",&topfirst);
 fgets(line,254,fd); sscanf(line,"%d %d %d",
  frame_pred_dct_tab,frame_pred_dct_tab+1,frame_pred_dct_tab+2);

 fgets(line,254,fd); sscanf(line,"%d %d %d",
  conceal_tab,conceal_tab+1,conceal_tab+2);
```

```
 fgets(line,254,fd); sscanf(line,"%d %d %d",
   qscale_tab,qscale_tab+1,qscale_tab+2);

 fgets(line,254,fd); sscanf(line,"%d %d %d",
   intravlc_tab,intravlc_tab+1,intravlc_tab+2);
 fgets(line,254,fd); sscanf(line,"%d %d %d",
   altscan_tab,altscan_tab+1,altscan_tab+2);
 fgets(line,254,fd); sscanf(line,"%d",&repeatfirst);
 fgets(line,254,fd), sscanf(line,"%d",&prog_frame);
/* intra slice interval refresh period */
 fgets(line,254,fd); sscanf(line,"%d",&P);
 fgets(line,254,fd); sscanf(line,"%d",&r);
 fgets(line,254,fd); sscanf(line,"%lf",&avg_act);
 fgets(line,254,fd); sscanf(line,"%d",&Xi);
 fgets(line,254,fd); sscanf(line,"%d",&Xp);
 fgets(line,254,fd), sscanf(line,"%d",&Xb),
 fgets(line,254,fd); sscanf(line,"%d",&d0i);
 fgets(line,254,fd); sscanf(line,"%d",&d0p);
 fgets(line,254,fd); sscanf(line,"%d",&d0b);

 if (N<1)
  error("N must be positive");
 if (M<1)
  error("M must be positive");
 if (N%M != 0)
  error("N must be an integer multiple of M");

 motion_data = (struct motion_data *)malloc(M*sizeof(struct motion_data));
 if (!motion_data) error("malloc failed\n");

 for (i=0, i<M; i++)
 {
  fgets(line,254,fd);
  sscanf(line,"%d %d %d %d",
    &motion_data[i].forw_hor_f_code, &motion_data[i].forw_vert_f_code,
    &motion_data[i].sxf, &motion_data[i].syf);

  if (i!=0)
  {
   fgets(line,254,fd);
   sscanf(line,"%d %d %d %d",
     &motion_data[i].back_hor_f_code, &motion_data[i].back_vert_f_code,
     &motion_data[i].sxb, &motion_data[i].syb);
  }
 }

 fclose(fd);

 /* make flags boolean (x!=0 -> x=1) */
 mpeg1 = !!mpeg1;
 fieldpic = !!fieldpic;
 low_delay = !!low_delay;
 constrparms = !!constrparms;
```

```
  prog_seq = !!prog_seq;
  topfirst = !!topfirst;

  for (i=0; i<3, i++)
  {
    frame_pred_dct_tab[i] = !!frame_pred_dct_tab[i];
    conceal_tab[i] = !!conceal_tab[i];
    qscale_tab[i] = !!qscale_tab[i];
    intravlc_tab[i] = !!intravlc_tab[i];
    altscan_tab[i] = !!altscan_tab[i];
  }
  repeatfirst = !!repeatfirst;
  prog_frame = !!prog_frame;

  chan_out_word(M,Link1Output);
  chan_out_word(M,Link2Output);

  chan_out_word(nframes,Link1Output);
  chan_out_word(nframes,Link2Output);

  chan_out_word(chroma_format,Link1Output);
  chan_out_word(chroma_format,Link2Output);

}


static void init()
{
  int i, size;
  static int block_count_tab[3] = {6,8,12};

  init_fdct();

  /* round picture dimensions to nearest multiple of 16 or 32 */
  mb_width = (horizontal_size+15)/16;
  mb_height = prog_seq ? (vertical_size+15)/16 : 2*((vertical_size+31)/32);
  mb_height2 = fieldpic ? mb_height>>1 : mb_height; /* for field pictures */
  width = 16*mb_width;
  height = 16*mb_height;

  chrom_width = (chroma_format==CHROMA444) ? width : width>>1;
  chrom_height = (chroma_format!=CHROMA420) ? height : height>>1;

  height2 = fieldpic ? height>>1 : height;
  width2 = fieldpic ? width<<1 : width;
  chrom_width2 = fieldpic ? chrom_width<<1 : chrom_width;

  block_count = block_count_tab[chroma_format-1];

  /* clip table */
  if (!(clp = (char *)malloc(1024)))
    error("malloc failed\n");
  clp+= 384;
  for (i=-384; i<640; i++)
```

```
  clp[i] = (i<0) ? 0 : ((i>255) ? 255 : i);

 for (i=0; i<3; i++)
 {
  size = (i==0) ? width*height : chrom_width*chrom_height;

  if (!(auxorgframe[i] = (char *)malloc(size)))
    error("malloc failed for auxorgframe\n");
  if (!(predframe[i] = (char *)malloc(size)))
    error("malloc failed for predframe\n");
 }

 mbinfo = (struct mbinfo *)malloc(mb_width*mb_height2*sizeof(struct mbinfo));
 if (!mbinfo) error("malloc failed mbinfo\n");

 blocks = (int (*)[64])malloc(mb_width*mb_height2*block_count*sizeof(int [64]));
 if (!blocks) error("malloc failed for blocks\n");

 chan_out_word(mb_width,Link1Output);
 chan_out_word(mb_width,Link2Output);

 chan_out_word(mb_height,Link1Output);
 chan_out_word(mb_height,Link2Output);

 chan_out_word(mb_height2,Link1Output);
 chan_out_word(mb_height2,Link2Output);

 chan_out_word(width,Link1Output);
 chan_out_word(width,Link2Output);

 chan_out_word(height,Link1Output);
 chan_out_word(height,Link2Output);

 chan_out_word(width2,Link1Output),
 chan_out_word(width2,Link2Output);

 chan_out_word(height2,Link1Output),
 chan_out_word(height2,Link2Output);

 chan_out_word(chrom_width,Link1Output);
 chan_out_word(chrom_width,Link2Output);

 chan_out_word(chrom_height,Link1Output);
 chan_out_word(chrom_height,Link2Output);

 chan_out_word(chrom_width2,Link1Output);
 chan_out_word(chrom_width2,Link2Output);

 chan_out_word(block_count,Link1Output);
 chan_out_word(block_count,Link2Output);

}
```

```
void putseq()
{
 /* this routine assumes (N % M) == 0 */
 int i, j, f;
 unsigned int a, b, c, p, q;
 FILE *f_mbi, *f_d, *f_status, *f_blk,
 char name[128], name1[128];
 char *neworg[3];
 int finished ;

 for (j=0; j<3; j++)
 neworg[j] = auxorgframe[j];

/* loop through all frames in encoding/decoding order */
 for (i=0; i<nframes; i++)
 {
  chan_out_word(i,Link1Output);
  chan_out_word(i,Link2Output);


/*  read in data from the first program      */
  sprintf(name,tplorg,i);
  sprintf(name1,"c:\\nick\\dtrn\\%s.st",name);
  if (!(f_status = fopen(name1,"r")))
  {
   sprintf(errortext,"Couldn't open %s\n",name1);
   error(errortext);
  }

  fscanf(f_status,"%d %d %d ", &f, &pict_struct, &frame_pred_dct);
  fclose(f_status);

  fprintf(f_time,"frame:%d\n", f);

  sprintf(name,tplorg,f+frame0);

  sprintf(name1,"c:\\nick\\dtrn\\%sy.nor",name);
  if (!(f_d = fopen(name1,"rb")))
  {
   sprintf(errortext,"Couldn't open %s\n",name1);
   error(errortext);
  }
  fread(neworg[0],1,width*height,f_d);
  fclose(f_d);

  sprintf(name1,"c:\\nick\\dtrn\\%su.nor",name);
  if (!(f_d = fopen(name1,"rb")))
  {
   sprintf(errortext,"Couldn't open %s\n",name1);
   error(errortext);
  }
  fread(neworg[1],1,chrom_width*chrom_height,f_d);
  fclose(f_d),
```

```
sprintf(name1,"c:\\nick\\dtrn\\%sv.nor",name);
if (!(f_d = fopen(name1,"rb")))
{
 sprintf(errortext,"Couldn't open %s\n",name1);
 error(errortext);
}
fread(neworg[2],1,chrom_width*chrom_height,f_d);
fclose(f_d),

sprintf(name1,"c:\\nick\\dtrn\\%sy.prf",name);
if (!(f_d = fopen(name1,"rb")))
{
 sprintf(errortext,"Couldn't open %s\n",name1);
 error(errortext);
}
fread(predframe[0],1,width*height,f_d);
fclose(f_d);

sprintf(name1,"c:\\nick\\dtrn\\%su prf",name);
if (!(f_d = fopen(name1,"rb")))
{
 sprintf(errortext,"Couldn't open %s\n",name1);
 error(errortext);
}
fread(predframe[1],1,chrom_width*chrom_height,f_d);
fclose(f_d);

sprintf(name1,"c:\\nick\\dtrn\\%sv.prf",name);
if (!(f_d = fopen(name1,"rb")))
{
 sprintf(errortext,"Couldn't open %s\n",name1);
 error(errortext);
}
fread(predframe[2],1,chrom_width*chrom_height,f_d);
fclose(f_d);

chan_out_word(pict_struct,Link1Output);
chan_out_word(pict_struct,Link2Output);

chan_out_word(frame_pred_dct,Link1Output);
chan_out_word(frame_pred_dct,Link2Output);

chan_out_message(width*height,neworg[0],Link1Output);
chan_out_message(width*height,neworg[0],Link2Output).

chan_out_message(chrom_width*chrom_height,neworg[1],Link1Output);
chan_out_message(chrom_width*chrom_height,neworg[1],Link2Output);

chan_out_message(chrom_width*chrom_height,neworg[2],Link1Output);
chan_out_message(chrom_width*chrom_height,neworg[2],Link2Output);

chan_out_message(width*height,predframe[0],Link1Output);
chan_out_message(width*height,predframe[0],Link2Output);
```

```
    chan_out_message(chrom_width*chrom_height,predframe[1],Link1Output),
    chan_out_message(chrom_width*chrom_height,predframe[1],Link2Output);

    chan_out_message(chrom_width*chrom_height,predframe[2],Link1Output);
    chan_out_message(chrom_width*chrom_height,predframe[2],Link2Output);

/* loop through 36 macroblocks of the picture */

     transform(predframe,neworg,mbinfo,blocks,1);

     for (a=36; a<146; a++)
     {
      chan_in_word(&mbinfo[a].dct_type,Link1Input);

      for (b=0; b<block_count; b++)
      for (c=0; c<64; c++)
      chan_in_word(&blocks[a*block_count+b][c],Link1Input);
     }

     for (a=146; a<256; a++)
     {
      chan_in_word(&mbinfo[a].dct_type,Link2Input);

      for (b=0; b<block_count; b++)
      for (c=0; c<64; c++)
      chan_in_word(&blocks[a*block_count+b][c],Link2Input);
     }

/*  write data for the third program                    */

     sprintf(name1,"c:\\nick\\trn_out\\%s.mbi",name);
     if (!(f_mbi = fopen(name1,"w/0")))
     {
      sprintf(errortext,"Couldn't open %s\n",name1);
      error(errortext);
     }
     sprintf(name1,"c:\\nick\\trn_out\\%s.blk",name);
     if (!(f_blk = fopen(name1,"w/0")))
     {
      sprintf(errortext,"Couldn't open %s\n",name1);
      error(errortext);
     }


     for (a=0; a<mb_height2*mb_width; a++)
     {
      fprintf(f_mbi,"%d ", mbinfo[a].dct_type);

      for (b=0; b<block_count; b++)
      {
       for (c=0; c<64; c++)
       fprintf(f_blk,"%d ", blocks[a*block_count+b][c]),

       fprintf(f_blk,"\n");
```

```
      }
     }

     fclose(f_mbi),
     fclose(f_blk);

  }

  fclose(f_time);

  chan_out_word(i,Link1Output);
  chan_out_word(i,Link2Output);

  chan_in_word(&finished,Link1Input);
  chan_in_word(&finished,Link2Input);


}


void dct_type_estimation _ANSI_ARGS_((char *pred,
                                      char *cur,
                                      struct mbinfo *mbi)),

static void add_sub_pred _ANSI_ARGS_((char *pred,
                                      char *cur,
                                      int lx, int *blk,
                                      char add_sub));

/* subtract prediction and transform prediction error */
/* inverse transform prediction error and add prediction */

void transform(char *pred[],
               char *cur[],
               struct mbinfo *mbi,
               int blocks[][64],
               char forward_invers )
{
 int i, j, i1, j1, n, cc, offs, lx;
 int k, ii, jj;

 if (forward_invers)
 dct_type_estimation(pred[0], cur[0], mbi ) ;

/* subtract prediction and transform prediction error */
/* inverse transform prediction error and add prediction */

/* loop through 36 macroblocks of the picture */

   for(k=0; k<36; k++)
   {
   jj = k>>4;
   ii = k - (jj<<4);
   j = jj<<4;
```

```
i = i1<<4;

for (n=0; n<block_count; n++)
{
 cc = (n<4) ? 0 : (n&1)+1, /* color component index */
 if (cc==0)
 {
  /* luminance */
  if ((pict_struct==FRAME_PICTURE) && mbi[k].dct_type)
  {
   /* field DCT */
   offs = i + ((n&1)<<3) + width*(j+((n&2)>>1));
   lx = width<<1;
  }
  else
  {
   /* frame DCT */
   offs = i + ((n&1)<<3) + width2*(j+((n&2)<<2));
   lx = width2;
  }

  if (pict_struct==BOTTOM_FIELD)
  offs += width;
 }
 else
 {
  /* chrominance */

  /* scale coordinates */
  i1 = (chroma_format==CHROMA444) ? i : i>>1;
  j1 = (chroma_format!=CHROMA420) ? j : j>>1;

  if ((pict_struct==FRAME_PICTURE) && mbi[k].dct_type
     && (chroma_format!=CHROMA420))
  {
   /* field DCT */
   offs = i1 + (n&8) + chrom_width*(j1+((n&2)>>1));
   lx = chrom_width<<1;
  }
  else
  {
   /* frame DCT */
   offs = i1 + (n&8) + chrom_width2*(j1+((n&2)<<2));
   lx = chrom_width2;
  }

  if (pict_struct==BOTTOM_FIELD)
  offs += chrom_width;
 }

 if (forward_invers) /* 1 for forward DCT */
 {
 add_sub_pred(pred[cc]+offs,
          cur[cc]+offs,
```

```
              lx,
              blocks[k*block_count+n],
              0);

      fdct(blocks[k*block_count+n]);
     }
     else /* 0 for invers DCT */
     {
/*      idct(blocks[k*block_count+n]);*/
      add_sub_pred(pred[cc]+offs,
              cur[cc]+offs,
              lx,
              blocks[k*block_count+n],
              1);
     }
    }
   }
}


/* add prediction and prediction error, saturate to 0. .255 */
/* subtract prediction from block data */
static void add_sub_pred(char *pred,
                  char *cur,
                  int lx, int *blk,
                  char add_sub )
{
 char i, j;

 for (j=0; j<8; j++)
 {
  for (i=0; i<8; i++)
  {
   if (add_sub) /* 1 for addition */
   cur[i] = clp[ blk[i] + pred[i] ];
   else /* 0 for subtractction */
   blk[i] = cur[i] - pred[i];
  }

  blk+= 8;
  cur+= lx;
  pred+= lx;
 }
}


void dct_type_estimation(char *pred,
                  char *cur,
                  struct mbinfo *mbi )
{
 int blk0[128], blk1[128];
 int i, j, i0, j0, offs;
 double s0, s1, sq0, sq1, s01;
```

```
double d, r;
int k, ii, jj;

  for(k=0; k<36; k++)
  {
  jj = k>>4;
  ii = k - (jj<<4);
  j0 = jj<<4;
  i0 = ii<<4;

  if (frame_pred_dct || pict_struct!=FRAME_PICTURE)
  mbi[k].dct_type = 0;
  else
  {
   for (j=0; j<8; j++)
   {
    offs = width*((j<<1)+j0) + i0;
    for (i=0, i<16; i++)
    {
     blk0[16*j+i] = cur[offs] - pred[offs];
     blk1[16*j+i] = cur[offs+width] - pred[offs+width];
     offs++;
    }
   }
   /* correlate fields */
   s0=s1=sq0=sq1=s01=0;

   for (i=0; i<128; i++)
   {
    s0+= (double) blk0[i];
    sq0+= (double) (blk0[i]*blk0[i]);
    s1+= (double) blk1[i];
    sq1+= (double) (blk1[i]*blk1[i]);
    s01+= (double) (blk0[i]*blk1[i]);
   }

   s0 = 0.125 * s0;
   s1 = 0.125 * s1;
   d = (sq0-(s0*s0)/2.0)*(sq1-(s1*s1)/2.0);

   if (d>0.0)
   {
    r = (s01-(s0*s1)/2.0)/sqrt(d);
    if (r>0.5)
    mbi[k].dct_type = 0; /* frame DCT */
    else
    mbi[k].dct_type = 1; /* field DCT */
   }
   else
   mbi[k].dct_type = 1; /* field DCT */
  }
  }
}
```

```c
#ifndef PI
# ifdef M_PI
#  define PI M_PI
# else
#  define PI 3 14159265358979323846
# endif
#endif


static double c[8][8]; /* transform coefficients */

void init_fdct()
{
  char i, j;
  double s;

  for (i=0; i<8; i++)
  {
   s = (i==0) ? sqrt(0.125) : 0.5;

   for (j=0; j<8; j++)
   c[i][j] = s * cos((PI/8.0)*i*(j+0.5));
  }
}

void fdct(int *block)
{
  char i, j, k;
  double s;
  double tmp[64];

  for (i=0; i<8; i++)
   for (j=0; j<8; j++)
   {
    s = 0.0;

    for (k=0; k<8; k++)
    s += c[j][k] * block[8*i+k];

    tmp[8*i+j] = s;
   }

  for (j=0; j<8; j++)
   for (i=0; i<8; i++)
   {
    s = 0.0,

    for (k=0; k<8; k++)
    s += c[i][k] * tmp[8*k+j];

    block[8*i+j] = (int)floor(s+0.499999);

   }
}
```

```
/********************************/
/*                              */
/*      t01 c residing at PE1   */
/*                              */
/********************************/


#include "c:\tc2v2\dos.h"
#include "c:\tc2v2\float.h"
#include "c:\tc2v2\math.h"
#include "c:\tc2v2\ctype.h"
#include "c:\tc2v2\stdio.h"
#include "c:\tc2v2\stdlib.h"
#include "c:\tc2v2\string.h"
#include "c:\tc2v2\chan.h"
#include "c:\tc2v2\thread.h"
#include "c:\tc2v2\time.h"
#include "c:\tc2v2\errno.h"

#define GLOBAL /* used by global.h */

#include "c:\nick\h\const1.h"
#include "c:\nick\h\config.h"
#include "c:\nick\h\global.h"

static void init _ANSI_ARGS_((void));
static void readparmfile _ANSI_ARGS_((char *fname));

void main(argc,argv,envp,ins,in_ports,outs,out_ports)
int argc,ins,outs;
char *argv[],*envp[];
CHAN *in_ports[],*out_ports[];
{
 /* read parameter file */
 chan_in_word(&M,Link0Input);
 chan_out_word(M,Link1Output);
 chan_out_word(M,Link2Output);

 chan_in_word(&nframes,Link0Input);
 chan_out_word(nframes,Link1Output);
 chan_out_word(nframes,Link2Output);

 chan_in_word(&chroma_format,Link0Input);
 chan_out_word(chroma_format,Link1Output);
 chan_out_word(chroma_format,Link2Output);

 init();
 putseq();

}


static void init()
{
```

```
int i, size;

init_fdct();

chan_in_word(&mb_width,Link0Input);
chan_out_word(mb_width,Link1Output);
chan_out_word(mb_width,Link2Output);

chan_in_word(&mb_height,Link0Input);
chan_out_word(mb_height,Link1Output);
chan_out_word(mb_height,Link2Output);

chan_in_word(&mb_height2,Link0Input);
chan_out_word(mb_height2,Link1Output);
chan_out_word(mb_height2,Link2Output);

chan_in_word(&width,Link0Input);
chan_out_word(width,Link1Output),
chan_out_word(width,Link2Output);

chan_in_word(&height,Link0Input);
chan_out_word(height,Link1Output);
chan_out_word(height,Link2Output);

chan_in_word(&width2,Link0Input);
chan_out_word(width2,Link1Output);
chan_out_word(width2,Link2Output);

chan_in_word(&height2,Link0Input);
chan_out_word(height2,Link1Output);
chan_out_word(height2,Link2Output);

chan_in_word(&chrom_width,Link0Input);
chan_out_word(chrom_width,Link1Output);
chan_out_word(chrom_width,Link2Output);

chan_in_word(&chrom_height,Link0Input);
chan_out_word(chrom_height,Link1Output);
chan_out_word(chrom_height,Link2Output);

chan_in_word(&chrom_width2,Link0Input);
chan_out_word(chrom_width2,Link1Output);
chan_out_word(chrom_width2,Link2Output);

chan_in_word(&block_count,Link0Input);
chan_out_word(block_count,Link1Output);
chan_out_word(block_count,Link2Output);

/* clip table */
clp = (char *)malloc(1024);
clp+= 384;
for (i=-384; i<640; i++)
  clp[i] = (i<0) ? 0 : ((i>255) ? 255 : i);
```

```
 for (i=0, i<3; i++)
 {
  size = (i==0) ? width*height . chrom_width*chrom_height;

  auxorgframe[i] = (char *)malloc(size);
  predframe[i] = (char *)malloc(size);
 }

 mbinfo = (struct mbinfo *)malloc(110*sizeof(struct mbinfo));
 blocks = (int (*)[64])malloc(110*block_count*sizeof(int [64])),


}


void putseq()
{
 /* this routine assumes (N % M) == 0 */
 int i, j, f;
 unsigned int a, b, c;
 char *neworg[3];
 int loop;

 for (j=0; j<3; j++)
 neworg[j] = auxorgframe[j];


/* loop through all frames in encoding/decoding order */
do{
   chan_in_word(&loop,Link0Input);
   chan_out_word(loop,Link1Output),
   chan_out_word(loop,Link2Output);

   if(loop==nframes) break;

   chan_in_word(&pict_struct,Link0Input);
   chan_out_word(pict_struct,Link1Output);
   chan_out_word(pict_struct,Link2Output);

   chan_in_word(&frame_pred_dct,Link0Input);
   chan_out_word(frame_pred_dct,Link1Output);
   chan_out_word(frame_pred_dct,Link2Output);

   chan_in_message(width*height,neworg[0],Link0Input);
   chan_out_message(width*height,neworg[0],Link1Output);
   chan_out_message(width*height,neworg[0],Link2Output);

   chan_in_message(chrom_width*chrom_height,neworg[1],Link0Input);
   chan_out_message(chrom_width*chrom_height,neworg[1],Link1Output);
   chan_out_message(chrom_width*chrom_height,neworg[1],Link2Output);

   chan_in_message(chrom_width*chrom_height,neworg[2],Link0Input);
   chan_out_message(chrom_width*chrom_height,neworg[2],Link1Output);
   chan_out_message(chrom_width*chrom_height,neworg[2],Link2Output);
```

```
  chan_in_message(width*height,predframe[0],Link0Input);
  chan_out_message(width*height,predframe[0],Link1Output);
  chan_out_message(width*height,predframe[0],Link2Output),

  chan_in_message(chrom_width*chrom_height,predframe[1],Link0Input);
  chan_out_message(chrom_width*chrom_height,predframe[1],Link1Output),
  chan_out_message(chrom_width*chrom_height,predframe[1],Link2Output);

  chan_in_message(chrom_width*chrom_height,predframe[2],Link0Input);
  chan_out_message(chrom_width*chrom_height,predframe[2],Link1Output);
  chan_out_message(chrom_width*chrom_height,predframe[2],Link2Output);

/* loop through 36 macroblocks of the picture */

    transform(predframe,neworg,mbinfo,blocks,1);

    for (a=36; a<73; a++)
    {
     chan_in_word(&mbinfo[a].dct_type,Link1Input);

     for (b=0; b<block_count; b++)
     for (c=0; c<64; c++)
     chan_in_word(&blocks[a*block_count+b][c],Link1Input),
    }

    for (a=73; a<110; a++)
    {
     chan_in_word(&mbinfo[a].dct_type,Link2Input);

     for (b=0; b<block_count; b++)
     for (c=0, c<64; c++)
     chan_in_word(&blocks[a*block_count+b][c],Link2Input);
    }

    for (a=0, a<110; a++)
    {
     chan_out_word(mbinfo[a].dct_type,Link0Output);

     for (b=0, b<block_count; b++)
     for (c=0; c<64; c++)
     chan_out_word(blocks[a*block_count+b][c],Link0Output);
    }

 }while(loop<nframes);

 chan_in_word(&loop,Link1Input),
 chan_in_word(&loop,Link2Input);
 chan_out_word(loop,Link0Output);

}/*end of main()*/

void dct_type_estimation _ANSI_ARGS_((char *pred,
                              char *cur,
```

```
                              struct mbinfo *mbi));

static void add_sub_pred _ANSI_ARGS_((char *pred,
                              char *cur,
                              int lx, int *blk,
                              char add_sub));
/* subtract prediction and transform prediction error */
/* inverse transform prediction error and add prediction */

void transform(char *pred[],
        char *cur[],
        struct mbinfo *mbi,
        int blocks[][64],
        char forward_invers )
{
 int i, j, i1, j1, n, cc, offs, lx;
 int k, ii, jj;


 if (forward_invers)
 dct_type_estimation(pred[0], cur[0], mbi ) ;

/* subtract prediction and transform prediction error */
/* inverse transform prediction error and add prediction */

/* loop through 36 macroblocks of the picture */

   for(k=0; k<36; k++)
   {
   jj = (k+36)>>4;
   ii = (k+36) - (jj<<4);
   j = jj<<4;
   i = ii<<4;

   for (n=0; n<block_count; n++)
   {
    cc = (n<4) ? 0 : (n&1)+1; /* color component index */
    if (cc==0)
    {
     /* luminance */
     if ((pict_struct==FRAME_PICTURE) && mbi[k].dct_type)
     {
      /* field DCT */
      offs = i + ((n&1)<<3) + width*(j+((n&2)>>1));
      lx = width<<1;
     }
     else
     {
      /* frame DCT */
      offs = i + ((n&1)<<3) + width2*(j+((n&2)<<2));
      lx = width2;
     }

     if (pict_struct==BOTTOM_FIELD)
```

```
          offs += width;
        }
        else
        {
          /* chrominance */

          /* scale coordinates */
          i1 = (chroma_format==CHROMA444) ? i : i>>1;
          j1 = (chroma_format!=CHROMA420) ? j : j>>1;

          if ((pict_struct==FRAME_PICTURE) && mbi[k].dct_type
             && (chroma_format!=CHROMA420))
          {
            /* field DCT */
            offs = i1 + (n&8) + chrom_width*(j1+((n&2)>>1));
            lx = chrom_width<<1;
          }
          else
          {
            /* frame DCT */
            offs = i1 + (n&8) + chrom_width2*(j1+((n&2)<<2));
            lx = chrom_width2;
          }

          if (pict_struct==BOTTOM_FIELD)
          offs += chrom_width;
        }

        if (forward_invers) /* 1 for forward DCT */
        {
         add_sub_pred(pred[cc]+offs,
                 cur[cc]+offs,
                 lx,
                 blocks[k*block_count+n],
                 0);

         fdct(blocks[k*block_count+n]);
        }
        else /* 0 for invers DCT */
        {
/*        idct(blocks[k*block_count+n]);*/
         add_sub_pred(pred[cc]+offs,
                 cur[cc]+offs,
                 lx,
                 blocks[k*block_count+n],
                 1),
        }
      }
    }
}


/* add prediction and prediction error, saturate to 0...255 */
/* subtract prediction from block data */
```

```
static void add_sub_pred(char *pred,
                         char *cur,
                         int lx, int *blk,
                         char add_sub )
{
  char i, j;

  for (j=0; j<8; j++)
  {
    for (i=0; i<8; i++)
    {
     if (add_sub) /* 1 for addition */
     cur[i] = clp[ blk[i] + pred[i] ];
     else /* 0 for subtractction */
     blk[i] = cur[i] - pred[i];
    }

    blk+= 8;
    cur+= lx;
    pred+= lx;
  }
}


void dct_type_estimation(char *pred,
                         char *cur,
                         struct mbinfo *mbi )
{
  int blk0[128], blk1[128];
  int i, j, i0, j0, offs;
  double s0, s1, sq0, sq1, s01;
  double d, r;
  int k, ii, jj;

     for(k=0; k<36; k++)
     {
      jj = (k+36)>>4;
      ii = (k+36) - (jj<<4);
      j0 = jj<<4;
      i0 = ii<<4;

      if (frame_pred_dct || pict_struct!=FRAME_PICTURE)
      mbi[k].dct_type = 0;
      else
      {
        for (j=0; j<8; j++)
        {
          offs = width*((j<<1)+j0) + i0;
          for (i=0; i<16; i++)
          {
            blk0[16*j+i] = cur[offs] - pred[offs];
            blk1[16*j+i] = cur[offs+width] - pred[offs+width];
            offs++;
          }
```

```
      }
      /* correlate fields */
      s0=s1=sq0=sq1=s01=0;

      for (i=0; i<128; i++)
      {
        s0+= (double) blk0[i];
        sq0+= (double) (blk0[i]*blk0[i]);
        s1+= (double) blk1[i];
        sq1+= (double) (blk1[i]*blk1[i]);
        s01+= (double) (blk0[i]*blk1[i]);
      }

      s0 = 0.125 * s0;
      s1 = 0.125 * s1;
      d = (sq0-(s0*s0)/2.0)*(sq1-(s1*s1)/2.0);

      if (d>0.0)
      {
        r = (s01-(s0*s1)/2.0)/sqrt(d);
        if (r>0.5)
        mbi[k].dct_type = 0; /* frame DCT */
        else
        mbi[k].dct_type = 1; /* field DCT */
      }
      else
      mbi[k].dct_type = 1; /* field DCT */
    }
  }
}


#ifndef PI
# ifdef M_PI
#  define PI M_PI
# else
#  define PI 3.14159265358979323846
# endif
#endif


static double c[8][8]; /* transform coefficients */

void init_fdct()
{
  char i, j,
  double s;

  for (i=0; i<8; i++)
  {
    s = (i==0) ? sqrt(0.125) : 0.5;

    for (j=0; j<8; j++)
    c[i][j] = s * cos((PI/8.0)*i*(j+0.5));
```

```
 }
}

void fdct(int *block)
{
 char i, j, k;
 double s;
 double tmp[64];

 for (i=0; i<8; i++)
  for (j=0; j<8; j++)
  {
   s = 0.0;

   for (k=0; k<8; k++)
   s += c[j][k] * block[8*i+k];

   tmp[8*i+j] = s;
  }

 for (j=0; j<8; j++)
  for (i=0; i<8; i++)
  {
   s = 0.0;

   for (k=0; k<8; k++)
   s += c[i][k] * tmp[8*k+j];

   block[8*i+j] = (int)floor(s+0.499999);

  }
}
```

```
/********************************/
/*                              */
/*        t011.c residing at PE2 */
/*                              */
/********************************/

#include "c:\tc2v2\dos.h"
#include "c:\tc2v2\float.h"
#include "c:\tc2v2\math.h"
#include "c:\tc2v2\ctype.h"
#include "c:\tc2v2\stdio.h"
#include "c:\tc2v2\stdlib.h"
#include "c:\tc2v2\string.h"
#include "c:\tc2v2\chan.h"
#include "c.\tc2v2\thread.h"
#include "c:\tc2v2\time.h"
#include "c:\tc2v2\errno.h"

#define GLOBAL /* used by global.h */

#include "c:\nick\h\const1.h"
#include "c:\nick\h\config.h"
#include "c:\nick\h\global.h"

/* private prototypes */
static void init _ANSI_ARGS_((void));
static void readparmfile _ANSI_ARGS_((char *fname)),

void main(argc,argv,envp,ins,in_ports,outs,out_ports)
int argc,ins,outs;
char *argv[],*envp[];
CHAN *in_ports[],*out_ports[];
{
 /* read parameter file */
 chan_in_word(&M,Link0Input);
 chan_in_word(&nframes,Link0Input);
 chan_in_word(&chroma_format,Link0Input);

 init();
 putseq();

}


static void init()
{
 int i, size;

 init_fdct();

 chan_in_word(&mb_width,Link0Input);
 chan_in_word(&mb_height,Link0Input);
 chan_in_word(&mb_height2,Link0Input);
 chan_in_word(&width,Link0Input);
```

```
chan_in_word(&height,Link0Input),
chan_in_word(&width2,Link0Input);
chan_in_word(&height2,Link0Input);
chan_in_word(&chrom_width,Link0Input);
chan_in_word(&chrom_height,Link0Input);
chan_in_word(&chrom_width2,Link0Input);
chan_in_word(&block_count,Link0Input);

/* clip table */
clp = (char *)malloc(1024);
clp+= 384;
for (i=-384; i<640; i++)
  clp[i] = (i<0) ? 0 : ((i>255) ? 255 · i);

for (i=0, i<3; i++)
{
  size = (i==0) ? width*height : chrom_width*chrom_height;

  auxorgframe[i] = (char *)malloc(size);
  predframe[i] = (char *)malloc(size);
}

mbinfo = (struct mbinfo *)malloc(37*sizeof(struct mbinfo));
blocks = (int (*)[64])malloc(37*block_count*sizeof(int [64]));
}


void putseq()
{
 /* this routine assumes (N % M) == 0 */
 int i, j, f;
 unsigned int a, b, c;
 char *neworg[3],
 int loop;

 for (j=0, j<3; j++)
 neworg[j] = auxorgframe[j];


/* loop through all frames in encoding/decoding order */
do{
   chan_in_word(&loop,Link0Input);

   if(loop==nframes) break;

   chan_in_word(&pict_struct,Link0Input),
   chan_in_word(&frame_pred_dct,Link0Input);
   chan_in_message(width*height,neworg[0],Link0Input);
   chan_in_message(chrom_width*chrom_height,neworg[1],Link0Input);
   chan_in_message(chrom_width*chrom_height,neworg[2],Link0Input);
   chan_in_message(width*height,predframe[0],Link0Input);
   chan_in_message(chrom_width*chrom_height,predframe[1],Link0Input);
   chan_in_message(chrom_width*chrom_height,predframe[2],Link0Input);
```

```
/* loop through 37 macroblocks of the picture */

      transform(predframe,neworg,mbinfo,blocks,1);

      for (a=0; a<37; a++)
      {
       chan_out_word(mbinfo[a].dct_type,Link0Output);

       for (b=0; b<block_count; b++)
       for (c=0; c<64; c++)
       chan_out_word(blocks[a*block_count+b][c],Link0Output);
      }

  }while(loop<nframes);

   chan_out_word(loop,Link0Output);

}/*end of main()*/

void dct_type_estimation _ANSI_ARGS_((char *pred,
                                      char *cur,
                                      struct mbinfo *mbi));

static void add_sub_pred _ANSI_ARGS_((char *pred,
                                      char *cur,
                                      int lx, int *blk,
                                      char add_sub)),
/* subtract prediction and transform prediction error */
/* inverse transform prediction error and add prediction */

void transform(char *pred[],
             char *cur[],
             struct mbinfo *mbi,
             int blocks[][64],
             char forward_invers )
{
 int i, j, i1, j1, n, cc, offs, lx;
 int k, ii, jj;

 if (forward_invers)
 dct_type_estimation(pred[0], cur[0], mbi ) ;

/* subtract prediction and transform prediction error */
/* inverse transform prediction error and add prediction */

/* loop through 37 macroblocks of the picture */

    for(k=0, k<37; k++)
    {
    jj = (k+72)>>4;
    ii = (k+72) - (jj<<4);
    j = jj<<4;
    i = ii<<4;
```

```
for (n=0; n<block_count; n++)
{
 cc = (n<4) ? 0 : (n&1)+1; /* color component index */
 if (cc==0)
 {
  /* luminance */
  if ((pict_struct==FRAME_PICTURE) && mbi[k].dct_type)
  {
   /* field DCT */
   offs = i + ((n&1)<<3) + width*(j+((n&2)>>1));
   lx = width<<1;
  }
  else
  {
   /* frame DCT */
   offs = i + ((n&1)<<3) + width2*(j+((n&2)<<2));
   lx = width2;
  }

  if (pict_struct==BOTTOM_FIELD)
  offs += width;
 }
 else
 {
  /* chrominance */

  /* scale coordinates */
  i1 = (chroma_format==CHROMA444) ? i : i>>1;
  j1 = (chroma_format!=CHROMA420) ? j : j>>1;

  if ((pict_struct==FRAME_PICTURE) && mbi[k].dct_type
     && (chroma_format!=CHROMA420))
  {
   /* field DCT */
   offs = i1 + (n&8) + chrom_width*(j1+((n&2)>>1));
   lx = chrom_width<<1;
  }
  else
  {
   /* frame DCT */
   offs = i1 + (n&8) + chrom_width2*(j1+((n&2)<<2));
   lx = chrom_width2;
  }

  if (pict_struct==BOTTOM_FIELD)
  offs += chrom_width;
 }

if (forward_invers) /* 1 for forward DCT */
{
 add_sub_pred(pred[cc]+offs,
          cur[cc]+offs,
          lx,
          blocks[k*block_count+n],
```

```
                    0);

          fdct(blocks[k*block_count+n]);
         }
         else /* 0 for invers DCT */
         {
/*          idct(blocks[k*block_count+n]);*/
          add_sub_pred(pred[cc]+offs,
                    cur[cc]+offs,
                    lx,
                    blocks[k*block_count+n],
                    1);
         }
       }
     }
}


/* add prediction and prediction error, saturate to 0...255 */
/* subtract prediction from block data */
static void add_sub_pred(char *pred,
                    char *cur,
                    int lx, int *blk,
                    char add_sub )
{
 char i, j;

 for (j=0; j<8; j++)
 {
  for (i=0; i<8; i++)
  {
   if (add_sub) /* 1 for addition */
   cur[i] = clp[ blk[i] + pred[i] ];
   else /* 0 for subtractction */
   blk[i] = cur[i] - pred[i];
  }

  blk+= 8;
  cur+= lx;
  pred+= lx;
 }
}


void dct_type_estimation(char *pred,
                    char *cur,
                    struct mbinfo *mbi )
{
 int blk0[128], blk1[128];
 int i, j, i0, j0, offs;
 double s0, s1, sq0, sq1, s01;
 double d, r;
 int k, ii, jj;
```

```
for(k=0; k<37; k++)
{
jj = (k+72)>>4,
ii = (k+72) - (jj<<4);
j0 = jj<<4;
i0 = ii<<4;

if (frame_pred_dct || pict_struct!=FRAME_PICTURE)
mbi[k].dct_type = 0;
else
{
 for (j=0; j<8; j++)
 {
  offs = width*((j<<1)+j0) + i0;
  for (i=0; i<16; i++)
  {
   blk0[16*j+i] = cur[offs] - pred[offs];
   blk1[16*j+i] = cur[offs+width] - pred[offs+width];
   offs++;
  }
 }
 /* correlate fields */
 s0=s1=sq0=sq1=s01=0;

 for (i=0, i<128; i++)
 {
  s0+= (double) blk0[i];
  sq0+= (double) (blk0[i]*blk0[i]);
  s1+= (double) blk1[i];
  sq1+= (double) (blk1[i]*blk1[i]),
  s01+= (double) (blk0[i]*blk1[i]);
 }

 s0 = 0.125 * s0;
 s1 = 0.125 * s1;
 d = (sq0-(s0*s0)/2.0)*(sq1-(s1*s1)/2.0);

 if (d>0.0)
 {
  r = (s01-(s0*s1)/2.0)/sqrt(d);
  if (r>0.5)
  mbi[k].dct_type = 0; /* frame DCT */
  else
  mbi[k].dct_type = 1, /* field DCT */
 }
 else
 mbi[k] dct_type = 1; /* field DCT */
}
}
}

#ifndef PI
```

```
# ifdef M_PI
#  define PI M_PI
# else
#  define PI 3.14159265358979323846
# endif
#endif


static double c[8][8]; /* transform coefficients */

void init_fdct()
{
 char i, j;
 double s;

 for (i=0, i<8; i++)
 {
  s = (i==0) ? sqrt(0.125) : 0.5;

  for (j=0; j<8; j++)
  c[i][j] = s * cos((PI/8.0)*i*(j+0.5));
 }
}

void fdct(int *block)
{
 char i, j, k;
 double s;
 double tmp[64];

 for (i=0; i<8; i++)
  for (j=0; j<8; j++)
  {
   s = 0.0;

   for (k=0, k<8; k++)
   s += c[j][k] * block[8*i+k];

   tmp[8*i+j] = s;
  }

 for (j=0, j<8; j++)
  for (i=0; i<8; i++)
  {
   s = 0.0;

   for (k=0, k<8; k++)
   s += c[i][k] * tmp[8*k+j];

   block[8*i+j] = (int)floor(s+0.499999);

  }
}
```

```
/********************************/
/*                              */
/*       t012.c residing at PE3 */
/*                              */
/********************************/

#include "c:\tc2v2\dos.h"
#include "c:\tc2v2\float.h"
#include "c:\tc2v2\math.h"
#include "c:\tc2v2\ctype.h"
#include "c:\tc2v2\stdio.h"
#include "c:\tc2v2\stdlib.h"
#include "c:\tc2v2\string.h"
#include "c:\tc2v2\chan.h"
#include "c:\tc2v2\thread.h"
#include "c:\tc2v2\time.h"
#include "c:\tc2v2\errno.h"

#define GLOBAL /* used by global.h */

#include "c:\nick\h\const1.h"
#include "c:\nick\h\config.h"
#include "c:\nick\h\global h"

static void init _ANSI_ARGS_((void));
static void readparmfile _ANSI_ARGS_((char *fname));

void main(argc,argv,envp,ins,in_ports,outs,out_ports)
int argc,ins,outs;
char *argv[],*envp[];
CHAN *in_ports[],*out_ports[];
{
 /* read parameter file */
 chan_in_word(&M,Link0Input);
 chan_in_word(&nframes,Link0Input);
 chan_in_word(&chroma_format,Link0Input);

 init();
 putseq();

}


static void init()
{
 int i, size;

 init_fdct();

 chan_in_word(&mb_width,Link0Input);
 chan_in_word(&mb_height,Link0Input);
 chan_in_word(&mb_height2,Link0Input);
 chan_in_word(&width,Link0Input);
 chan_in_word(&height,Link0Input);
```

```
  chan_in_word(&width2,Link0Input);
  chan_in_word(&height2,Link0Input);
  chan_in_word(&chrom_width,Link0Input);
  chan_in_word(&chrom_height,Link0Input);
  chan_in_word(&chrom_width2,Link0Input);
  chan_in_word(&block_count,Link0Input),

  /* clip table */
  clp = (char *)malloc(1024);
  clp+= 384;
  for (i=-384; i<640; i++)
    clp[i] = (i<0) ? 0 : ((i>255) ? 255 : i);

  for (i=0; i<3; i++)
  {
    size = (i==0) ? width*height : chrom_width*chrom_height;

    auxorgframe[i] = (char *)malloc(size);
    predframe[i] = (char *)malloc(size);
  }

  mbinfo = (struct mbinfo *)malloc(37*sizeof(struct mbinfo));
  blocks = (int (*)[64])malloc(37*block_count*sizeof(int [64]));


}


void putseq()
{
 /* this routine assumes (N % M) == 0 */
 int i, j, f;
 unsigned int a, b, c;
 char *neworg[3];
 int loop;

 for (j=0; j<3; j++)
 neworg[j] = auxorgframe[j];


/* loop through all frames in encoding/decoding order */
do{
   chan_in_word(&loop,Link0Input);

   if(loop==nframes) break;

   chan_in_word(&pict_struct,Link0Input);
   chan_in_word(&frame_pred_dct,Link0Input);
   chan_in_message(width*height,neworg[0],Link0Input);
   chan_in_message(chrom_width*chrom_height,neworg[1],Link0Input);
   chan_in_message(chrom_width*chrom_height,neworg[2],Link0Input);
   chan_in_message(width*height,predframe[0],Link0Input);
   chan_in_message(chrom_width*chrom_height,predframe[1],Link0Input);
   chan_in_message(chrom_width*chrom_height,predframe[2],Link0Input);
```

```
/* loop through 37 macroblocks of the picture */

      transform(predframe,neworg,mbinfo,blocks,1);

      for (a=0, a<37, a++)
      {
       chan_out_word(mbinfo[a].dct_type,Link0Output);

       for (b=0; b<block_count; b++)
       for (c=0; c<64; c++)
       chan_out_word(blocks[a*block_count+b][c],Link0Output);
      }

   }while(loop<nframes);

   chan_out_word(loop,Link0Output);

}/*end of main()*/

void dct_type_estimation _ANSI_ARGS_((char *pred,
                                      char *cur,
                                      struct mbinfo *mbi)),

static void add_sub_pred _ANSI_ARGS_((char *pred,
                                      char *cur,
                                      int lx, int *blk,
                                      char add_sub));
/* subtract prediction and transform prediction error */
/* inverse transform prediction error and add prediction */

void transform(char *pred[],
               char *cur[],
               struct mbinfo *mbi,
               int blocks[][64],
               char forward_invers )
{
  int i, j, i1, j1, n, cc, offs, lx;
  int k, ii, jj;

  if (forward_invers)
  dct_type_estimation(pred[0], cur[0], mbi ) ;

/* subtract prediction and transform prediction error */
/* inverse transform prediction error and add prediction */

/* loop through 37 macroblocks of the picture */

     for(k=0; k<37; k++)
     {
      jj = (k+109)>>4;
      ii = (k+109) - (jj<<4);
      j = jj<<4;
      i = ii<<4;
```

```
for (n=0; n<block_count; n++)
{
 cc = (n<4) ? 0 : (n&1)+1; /* color component index */
 if (cc==0)
 {
  /* luminance */
  if ((pict_struct==FRAME_PICTURE) && mbi[k].dct_type)
  {
   /* field DCT */
   offs = i + ((n&1)<<3) + width*(j+((n&2)>>1));
   lx = width<<1;
  }
  else
  {
   /* frame DCT */
   offs = i + ((n&1)<<3) + width2*(j+((n&2)<<2));
   lx = width2;
  }

  if (pict_struct==BOTTOM_FIELD)
  offs += width;
 }
 else
 {
  /* chrominance */

  /* scale coordinates */
  i1 = (chroma_format==CHROMA444) ? i : i>>1;
  j1 = (chroma_format!=CHROMA420) ? j : j>>1;

  if ((pict_struct==FRAME_PICTURE) && mbi[k].dct_type
     && (chroma_format!=CHROMA420))
  {
   /* field DCT */
   offs = i1 + (n&8) + chrom_width*(j1+((n&2)>>1));
   lx = chrom_width<<1;
  }
  else
  {
   /* frame DCT */
   offs = i1 + (n&8) + chrom_width2*(j1+((n&2)<<2));
   lx = chrom_width2;
  }

  if (pict_struct==BOTTOM_FIELD)
  offs += chrom_width,
 }

 if (forward_invers) /* 1 for forward DCT */
 {
 add_sub_pred(pred[cc]+offs,
          cur[cc]+offs,
          lx,
```

```
              blocks[k*block_count+n],
              0);

      fdct(blocks[k*block_count+n]);
     }
     else /* 0 for invers DCT */
     {
/*      idct(blocks[k*block_count+n]);*/
      add_sub_pred(pred[cc]+offs,
              cur[cc]+offs,
              lx,
              blocks[k*block_count+n],
              1);
     }
    }
   }
}



/* add prediction and prediction error, saturate to 0...255 */
/* subtract prediction from block data */
static void add_sub_pred(char *pred,
               char *cur,
               int lx, int *blk,
               char add_sub )
{
 char i, j;

 for (j=0; j<8; j++)
 {
  for (i=0; i<8; i++)
  {
   if (add_sub) /* 1 for addition */
   cur[i] = clp[ blk[i] + pred[i] ];
   else /* 0 for subtractction */
   blk[i] = cur[i] - pred[i];
  }

  blk+= 8;
  cur+= lx;
  pred+= lx;
 }
}


void dct_type_estimation(char *pred,
               char *cur,
               struct mbinfo *mbi )
{
 int blk0[128], blk1[128];
 int i, j, i0, j0, offs;
 double s0, s1, sq0, sq1, s01;
 double d, r;
```

```
int k, ii, jj;

  for(k=0; k<37; k++)
  {
  jj = (k+109)>>4;
  ii = (k+109) - (jj<<4);
  j0 = jj<<4,
  i0 = ii<<4;

  if (frame_pred_dct || pict_struct!=FRAME_PICTURE)
  mbi[k].dct_type = 0;
  else
  {
   for (j=0; j<8; j++)
   {
    offs = width*((j<<1)+j0) + i0;
    for (i=0; i<16; i++)
    {
     blk0[16*j+i] = cur[offs] - pred[offs];
     blk1[16*j+i] = cur[offs+width] - pred[offs+width];
     offs++;
    }
   }
   /* correlate fields */
   s0=s1=sq0=sq1=s01=0;

   for (i=0; i<128; i++)
   {
    s0+= (double) blk0[i];
    sq0+= (double) (blk0[i]*blk0[i]);
    s1+= (double) blk1[i];
    sq1+= (double) (blk1[i]*blk1[i]);
    s01+= (double) (blk0[i]*blk1[i]);
   }

   s0 = 0.125 * s0;
   s1 = 0.125 * s1;
   d = (sq0-(s0*s0)/2.0)*(sq1-(s1*s1)/2.0);

   if (d>0.0)
   {
    r = (s01-(s0*s1)/2.0)/sqrt(d);
    if (r>0.5)
    mbi[k].dct_type = 0; /* frame DCT */
    else
    mbi[k].dct_type = 1; /* field DCT */
   }
   else
   mbi[k].dct_type = 1; /* field DCT */
  }
 }
}
```

```
#ifndef PI
# ifdef M_PI
#  define PI M_PI
# else
#  define PI 3.14159265358979323846
# endif
#endif


static double c[8][8]; /* transform coefficients */

void init_fdct()
{
  char i, j;
  double s;

  for (i=0; i<8; i++)
  {
    s = (i==0) ? sqrt(0.125) : 0.5;

    for (j=0; j<8; j++)
    c[i][j] = s * cos((PI/8 0)*i*(j+0.5));
  }
}

void fdct(int *block)
{
  char i, j, k;
  double s;
  double tmp[64],

  for (i=0; i<8; i++)
    for (j=0; j<8; j++)
    {
      s = 0.0;

      for (k=0; k<8; k++)
      s += c[j][k] * block[8*i+k];

      tmp[8*i+j] = s;
    }

  for (j=0; j<8; j++)
    for (i=0; i<8, i++)
    {
      s = 0.0,

      for (k=0; k<8; k++)
      s += c[i][k] * tmp[8*k+j];

      block[8*i+j] = (int)floor(s+0.499999);

    }
}
```

```
/********************************/
/*                              */
/*      t02.c residing at PE4   */
/*                              */
/********************************/

#include "c:\tc2v2\dos.h"
#include "c:\tc2v2\float.h"
#include "c:\tc2v2\math.h"
#include "c:\tc2v2\ctype.h"
#include "c:\tc2v2\stdio.h"
#include "c:\tc2v2\stdlib.h"
#include "c:\tc2v2\string.h"
#include "c:\tc2v2\chan.h"
#include "c:\tc2v2\thread.h"
#include "c:\tc2v2\time.h"
#include "c:\tc2v2\errno.h"

#define GLOBAL /* used by global.h */

#include "c:\nick\h\const1.h"
#include "c:\nick\h\config.h"
#include "c:\nick\h\global.h"

static void init _ANSI_ARGS_((void));
static void readparmfile _ANSI_ARGS_((char *fname));

void main(argc,argv,envp,ins,in_ports,outs,out_ports)
int argc,ins,outs;
char *argv[],*envp[];
CHAN *in_ports[],*out_ports[];
{
 /* read parameter file */
 chan_in_word(&M,Link0Input);
 chan_out_word(M,Link1Output);
 chan_out_word(M,Link2Output);

 chan_in_word(&nframes,Link0Input);
 chan_out_word(nframes,Link1Output);
 chan_out_word(nframes,Link2Output);

 chan_in_word(&chroma_format,Link0Input);
 chan_out_word(chroma_format,Link1Output);
 chan_out_word(chroma_format,Link2Output);

 init();
 putseq();

}


static void init()
{
 int i, size;
```

```
init_fdct();

chan_in_word(&mb_width,Link0Input);
chan_out_word(mb_width,Link1Output),
chan_out_word(mb_width,Link2Output);

chan_in_word(&mb_height,Link0Input);
chan_out_word(mb_height,Link1Output),
chan_out_word(mb_height,Link2Output);

chan_in_word(&mb_height2,Link0Input);
chan_out_word(mb_height2,Link1Output);
chan_out_word(mb_height2,Link2Output);

chan_in_word(&width,Link0Input);
chan_out_word(width,Link1Output);
chan_out_word(width,Link2Output);

chan_in_word(&height,Link0Input);
chan_out_word(height,Link1Output);
chan_out_word(height,Link2Output);

chan_in_word(&width2,Link0Input);
chan_out_word(width2,Link1Output);
chan_out_word(width2,Link2Output);

chan_in_word(&height2,Link0Input);
chan_out_word(height2,Link1Output);
chan_out_word(height2,Link2Output);

chan_in_word(&chrom_width,Link0Input);
chan_out_word(chrom_width,Link1Output);
chan_out_word(chrom_width,Link2Output);

chan_in_word(&chrom_height,Link0Input);
chan_out_word(chrom_height,Link1Output);
chan_out_word(chrom_height,Link2Output);

chan_in_word(&chrom_width2,Link0Input);
chan_out_word(chrom_width2,Link1Output);
chan_out_word(chrom_width2,Link2Output),

chan_in_word(&block_count,Link0Input),
chan_out_word(block_count,Link1Output);
chan_out_word(block_count,Link2Output);

/* clip table */
clp = (char *)malloc(1024);
clp+= 384;
for (i=-384; i<640; i++)
  clp[i] = (i<0) ? 0 : ((i>255) ? 255 : i);

for (i=0; i<3; i++)
```

```
 {
  size = (i==0) ? width*height : chrom_width*chrom_height;

  auxorgframe[i] = (char *)malloc(size);
  predframe[i] = (char *)malloc(size);
 }

 mbinfo = (struct mbinfo *)malloc(110*sizeof(struct mbinfo));
 blocks = (int (*)[64])malloc(110*block_count*sizeof(int [64]));


}


void putseq()
{
 /* this routine assumes (N % M) == 0 */
 int i, j, f;
 unsigned int a, b, c;
 char *neworg[3];
 int loop;

 for (j=0; j<3; j++)
 neworg[j] = auxorgframe[j];


/* loop through all frames in encoding/decoding order */
do{
  chan_in_word(&loop,Link0Input);
  chan_out_word(loop,Link1Output);
  chan_out_word(loop,Link2Output);

  if(loop==nframes) break;

  chan_in_word(&pict_struct,Link0Input);
  chan_out_word(pict_struct,Link1Output),
  chan_out_word(pict_struct,Link2Output);

  chan_in_word(&frame_pred_dct,Link0Input);
  chan_out_word(frame_pred_dct,Link1Output),
  chan_out_word(frame_pred_dct,Link2Output);

  chan_in_message(width*height,neworg[0],Link0Input);
  chan_out_message(width*height,neworg[0],Link1Output);
  chan_out_message(width*height,neworg[0],Link2Output);

  chan_in_message(chrom_width*chrom_height,neworg[1],Link0Input);
  chan_out_message(chrom_width*chrom_height,neworg[1],Link1Output);
  chan_out_message(chrom_width*chrom_height,neworg[1],Link2Output);

  chan_in_message(chrom_width*chrom_height,neworg[2],Link0Input);
  chan_out_message(chrom_width*chrom_height,neworg[2],Link1Output);
  chan_out_message(chrom_width*chrom_height,neworg[2],Link2Output);
```

```
    chan_in_message(width*height,predframe[0],Link0Input);
    chan_out_message(width*height,predframe[0],Link1Output),
    chan_out_message(width*height,predframe[0],Link2Output),

    chan_in_message(chrom_width*chrom_height,predframe[1],Link0Input),
    chan_out_message(chrom_width*chrom_height,predframe[1],Link1Output);
    chan_out_message(chrom_width*chrom_height,predframe[1],Link2Output);

    chan_in_message(chrom_width*chrom_height,predframe[2],Link0Input);
    chan_out_message(chrom_width*chrom_height,predframe[2],Link1Output);
    chan_out_message(chrom_width*chrom_height,predframe[2],Link2Output);

/* loop through 36 macroblocks of the picture */

     transform(predframe,neworg,mbinfo,blocks,1);

     for (a=36; a<73; a++)
     {
      chan_in_word(&mbinfo[a].dct_type,Link1Input);

      for (b=0; b<block_count; b++)
      for (c=0; c<64; c++)
      chan_in_word(&blocks[a*block_count+b][c],Link1Input);
     }

     for (a=73; a<110; a++)
     {
      chan_in_word(&mbinfo[a].dct_type,Link2Input);

      for (b=0; b<block_count; b++)
      for (c=0; c<64; c++)
      chan_in_word(&blocks[a*block_count+b][c],Link2Input);
     }

    for (a=0; a<110; a++)
    {
     chan_out_word(mbinfo[a].dct_type,Link0Output);

     for (b=0; b<block_count; b++)
     for (c=0; c<64; c++)
     chan_out_word(blocks[a*block_count+b][c],Link0Output);
    }

  }while(loop<nframes);

  chan_in_word(&loop,Link1Input);
  chan_in_word(&loop,Link2Input);
  chan_out_word(loop,Link0Output);

}/*end of main()*/

void dct_type_estimation _ANSI_ARGS_((char *pred,
                              char *cur,
                              struct mbinfo *mbi));
```

```
static void add_sub_pred _ANSI_ARGS_((char *pred,
                                     char *cur,
                                     int lx, int *blk,
                                     char add_sub));
/* subtract prediction and transform prediction error */
/* inverse transform prediction error and add prediction */

void transform(char *pred[],
               char *cur[],
               struct mbinfo *mbi,
               int blocks[][64],
               char forward_invers )
{
  int i, j, i1, j1, n, cc, offs, lx;
  int k, ii, jj;

  if (forward_invers)
  dct_type_estimation(pred[0], cur[0], mbi ) ;

/* subtract prediction and transform prediction error */
/* inverse transform prediction error and add prediction */

/* loop through 36 macroblocks of the picture */

    for(k=0; k<36; k++)
    {
     jj = (k+146)>>4;
     ii = (k+146) - (jj<<4);
     j = jj<<4;
     i = ii<<4;

     for (n=0; n<block_count; n++)
     {
       cc = (n<4) ? 0 : (n&1)+1; /* color component index */
       if (cc==0)
       {
         /* luminance */
         if ((pict_struct==FRAME_PICTURE) && mbi[k].dct_type)
         {
           /* field DCT */
           offs = i + ((n&1)<<3) + width*(j+((n&2)>>1));
           lx = width<<1;
         }
         else
         {
           /* frame DCT */
           offs = i + ((n&1)<<3) + width2*(j+((n&2)<<2));
           lx = width2;
         }

         if (pict_struct==BOTTOM_FIELD)
         offs += width;
       }
```

```
        else
        {
         /* chrominance */

         /* scale coordinates */
         i1 = (chroma_format==CHROMA444) ? i : i>>1,
         j1 = (chroma_format!=CHROMA420) ? j : j>>1;

         if ((pict_struct==FRAME_PICTURE) && mbi[k].dct_type
            && (chroma_format!=CHROMA420))
         {
          /* field DCT */
          offs = i1 + (n&8) + chrom_width*(j1+((n&2)>>1)),
          lx = chrom_width<<1;
         }
         else
         {
          /* frame DCT */
          offs = i1 + (n&8) + chrom_width2*(j1+((n&2)<<2));
          lx = chrom_width2;
         }

         if (pict_struct==BOTTOM_FIELD)
         offs += chrom_width;
        }

        if (forward_invers) /* 1 for forward DCT */
        {
         add_sub_pred(pred[cc]+offs,
                 cur[cc]+offs,
                 lx,
                 blocks[k*block_count+n],
                 0);

         fdct(blocks[k*block_count+n]);
        }
        else /* 0 for invers DCT */
        {
/*        idct(blocks[k*block_count+n]);*/
         add_sub_pred(pred[cc]+offs,
                 cur[cc]+offs,
                 lx,
                 blocks[k*block_count+n],
                 1);
        }
      }
    }
}


/* add prediction and prediction error, saturate to 0...255 */
/* subtract prediction from block data */
static void add_sub_pred(char *pred,
```

```
                    char *cur,
                    int lx, int *blk,
                    char add_sub )
{
 char i, j;

 for (j=0; j<8; j++)
 {
  for (i=0; i<8; i++)
  {
   if (add_sub) /* 1 for addition */
   cur[i] = clp[ blk[i] + pred[i] ];
   else /* 0 for subtractction */
   blk[i] = cur[i] - pred[i];
  }

  blk+= 8;
  cur+= lx;
  pred+= lx;
 }
}


void dct_type_estimation(char *pred,
                    char *cur,
                    struct mbinfo *mbi )
{
 int blk0[128], blk1[128],
 int i, j, i0, j0, offs;
 double s0, s1, sq0, sq1, s01;
 double d, r;
 int k, ii, jj;

   for(k=0; k<36; k++)
   {
    jj = (k+146)>>4;
    ii = (k+146) - (jj<<4);
    j0 = jj<<4;
    i0 = ii<<4;

    if (frame_pred_dct || pict_struct!=FRAME_PICTURE)
    mbi[k].dct_type = 0;
    else
    {
     for (j=0; j<8, j++)
     {
      offs = width*((j<<1)+j0) + i0;
      for (i=0; i<16; i++)
      {
       blk0[16*j+i] = cur[offs] - pred[offs];
       blk1[16*j+i] = cur[offs+width] - pred[offs+width];
       offs++,
      }
     }
```

```
      /* correlate fields */
      s0=s1=sq0=sq1=s01=0;

      for (i=0; i<128; i++)
      {
       s0+= (double) blk0[i];
       sq0+= (double) (blk0[i]*blk0[i]),
       s1+= (double) blk1[i];
       sq1+= (double) (blk1[i]*blk1[i]);
       s01+= (double) (blk0[i]*blk1[i]);
      }

      s0 = 0.125 * s0;
      s1 = 0.125 * s1;
      d = (sq0-(s0*s0)/2.0)*(sq1-(s1*s1)/2.0),

      if (d>0.0)
      {
       r = (s01-(s0*s1)/2.0)/sqrt(d);
       if (r>0.5)
       mbi[k].dct_type = 0; /* frame DCT */
       else
       mbi[k].dct_type = 1; /* field DCT */
      }
      else
      mbi[k].dct_type = 1; /* field DCT */
    }
  }
}


#ifndef PI
# ifdef M_PI
#  define PI M_PI
# else
#  define PI 3.14159265358979323846
# endif
#endif


static double c[8][8]; /* transform coefficients */

void init_fdct()
{
 char i, j;
 double s;

 for (i=0; i<8; i++)
 {
  s = (i==0) ? sqrt(0.125) : 0.5;

  for (j=0; j<8; j++)
  c[i][j] = s * cos((PI/8.0)*i*(j+0.5));
 }
```

```
}

void fdct(int *block)
{
 char i, j, k;
 double s;
 double tmp[64];

 for (i=0; i<8, i++)
  for (j=0; j<8; j++)
  {
   s = 0.0;

   for (k=0; k<8; k++)
   s += c[j][k] * block[8*i+k];

   tmp[8*i+j] = s;
  }

 for (j=0; j<8; j++)
  for (i=0; i<8; i++)
  {
   s = 0.0;

   for (k=0; k<8; k++)
   s += c[i][k] * tmp[8*k+j];

   block[8*i+j] = (int)floor(s+0.499999);

  }
}
```

```
/**********************************/
/*                                */
/*   t021.c residing at PE5       */
/*                                */
/**********************************/

#include "c:\tc2v2\dos.h"
#include "c:\tc2v2\float.h"
#include "c:\tc2v2\math.h"
#include "c:\tc2v2\ctype.h"
#include "c:\tc2v2\stdio.h"
#include "c:\tc2v2\stdlib.h"
#include "c:\tc2v2\string.h"
#include "c:\tc2v2\chan.h"
#include "c:\tc2v2\thread.h"
#include "c \tc2v2\time.h"
#include "c:\tc2v2\errno.h"

#define GLOBAL /* used by global.h */

#include "c:\nick\h\const1.h"
#include "c:\nick\h\config.h"
#include "c:\nick\h\global.h"

static void init _ANSI_ARGS_((void));
static void readparmfile _ANSI_ARGS_((char *fname));

void main(argc,argv,envp,ins,in_ports,outs,out_ports)
int argc,ins,outs;
char *argv[],*envp[];
CHAN *in_ports[],*out_ports[];
{
 /* read parameter file */
 chan_in_word(&M,Link0Input);
 chan_in_word(&nframes,Link0Input);
 chan_in_word(&chroma_format,Link0Input);

 init();
 putseq();

}


static void init()
{
 int i, size;

 init_fdct();

 chan_in_word(&mb_width,Link0Input);
 chan_in_word(&mb_height,Link0Input);
 chan_in_word(&mb_height2,Link0Input);
 chan_in_word(&width,Link0Input);
 chan_in_word(&height,Link0Input);
```

```
  chan_in_word(&width2,Link0Input);
  chan_in_word(&height2,Link0Input);
  chan_in_word(&chrom_width,Link0Input);
  chan_in_word(&chrom_height,Link0Input);
  chan_in_word(&chrom_width2,Link0Input);
  chan_in_word(&block_count,Link0Input);

  /* clip table */
  clp = (char *)malloc(1024);
  clp+= 384;
  for (i=-384; i<640; i++)
    clp[i] = (i<0) ? 0 : ((i>255) ? 255 : i);

  for (i=0; i<3; i++)
  {
    size = (i==0) ? width*height : chrom_width*chrom_height;

    auxorgframe[i] = (char *)malloc(size);
    predframe[i] = (char *)malloc(size);
  }

  mbinfo = (struct mbinfo *)malloc(37*sizeof(struct mbinfo));
  blocks = (int (*)[64])malloc(37*block_count*sizeof(int [64]));


}


void putseq()
{
  /* this routine assumes (N % M) == 0 */
  int i, j, f;
  unsigned int a, b, c;
  char *neworg[3];
  int loop;

  for (j=0; j<3; j++)
  neworg[j] = auxorgframe[j];


/* loop through all frames in encoding/decoding order */
do{
   chan_in_word(&loop,Link0Input);

   if(loop==nframes) break;

   chan_in_word(&pict_struct,Link0Input);
   chan_in_word(&frame_pred_dct,Link0Input);
   chan_in_message(width*height,neworg[0],Link0Input);
   chan_in_message(chrom_width*chrom_height,neworg[1],Link0Input);
   chan_in_message(chrom_width*chrom_height,neworg[2],Link0Input);
   chan_in_message(width*height,predframe[0],Link0Input);
   chan_in_message(chrom_width*chrom_height,predframe[1],Link0Input);
   chan_in_message(chrom_width*chrom_height,predframe[2],Link0Input);
```

```
/* loop through 37 macroblocks of the picture */

     transform(predframe,neworg,mbinfo,blocks,1);

     for (a=0; a<37; a++)
     {
      chan_out_word(mbinfo[a].dct_type,Link0Output);

      for (b=0; b<block_count; b++)
      for (c=0; c<64, c++)
      chan_out_word(blocks[a*block_count+b][c],Link0Output);
     }

  }while(loop<nframes),

   chan_out_word(loop,Link0Output),

}/*end of main()*/

void dct_type_estimation _ANSI_ARGS_((char *pred,
                                char *cur,
                                struct mbinfo *mbi));

static void add_sub_pred _ANSI_ARGS_((char *pred,
                                char *cur,
                                int lx, int *blk,
                                char add_sub));
/* subtract prediction and transform prediction error */
/* inverse transform prediction error and add prediction */

void transform(char *pred[],
            char *cur[],
            struct mbinfo *mbi,
            int blocks[][64],
            char forward_invers )
{
 int i, j, i1, j1, n, cc, offs, lx;
 int k, ii, jj;

 if (forward_invers)
 dct_type_estimation(pred[0], cur[0], mbi ) ;

/* subtract prediction and transform prediction error */
/* inverse transform prediction error and add prediction */

/* loop through 37 macroblocks of the picture */

    for(k=0; k<37; k++)
    {
     jj = (k+182)>>4;
     ii = (k+182) - (jj<<4),
     j = jj<<4;
     i = ii<<4;
```

```
for (n=0; n<block_count, n++)
{
 cc = (n<4) ? 0 : (n&1)+1; /* color component index */
 if (cc==0)
 {
  /* luminance */
  if ((pict_struct==FRAME_PICTURE) && mbi[k].dct_type)
  {
   /* field DCT */
   offs = i + ((n&1)<<3) + width*(j+((n&2)>>1));
   lx = width<<1;
  }
  else
  {
   /* frame DCT */
   offs = i + ((n&1)<<3) + width2*(j+((n&2)<<2));
   lx = width2;
  }

  if (pict_struct==BOTTOM_FIELD)
  offs += width;
 }
 else
 {
  /* chrominance */

  /* scale coordinates */
  i1 = (chroma_format==CHROMA444) ? i : i>>1,
  j1 = (chroma_format!=CHROMA420) ? j : j>>1;

  if ((pict_struct==FRAME_PICTURE) && mbi[k].dct_type
     && (chroma_format!=CHROMA420))
  {
   /* field DCT */
   offs = i1 + (n&8) + chrom_width*(j1+((n&2)>>1));
   lx = chrom_width<<1;
  }
  else
  {
   /* frame DCT */
   offs = i1 + (n&8) + chrom_width2*(j1+((n&2)<<2));
   lx = chrom_width2;
  }

  if (pict_struct==BOTTOM_FIELD)
  offs += chrom_width;
 }

 if (forward_invers) /* 1 for forward DCT */
 {
 add_sub_pred(pred[cc]+offs,
          cur[cc]+offs,
          lx,
```

```
                blocks[k*block_count+n],
                0);

        fdct(blocks[k*block_count+n]),
        }
        else /* 0 for invers DCT */
        {
/*          idct(blocks[k*block_count+n]);*/
        add_sub_pred(pred[cc]+offs,
                cur[cc]+offs,
                lx,
                blocks[k*block_count+n],
                1);
        }
      }
    }
}


/* add prediction and prediction error, saturate to 0...255 */
/* subtract prediction from block data */
static void add_sub_pred(char *pred,
                  char *cur,
                  int lx, int *blk,
                  char add_sub )
{
  char i, j;

  for (j=0; j<8; j++)
  {
    for (i=0; i<8; i++)
    {
     if (add_sub) /* 1 for addition */
     cur[i] = clp[ blk[i] + pred[i] ];
     else /* 0 for subtractction */
     blk[i] = cur[i] - pred[i];
    }

    blk+= 8;
    cur+= lx;
    pred+= lx;
  }
}


void dct_type_estimation(char *pred,
                  char *cur,
                  struct mbinfo *mbi )
{
  int blk0[128], blk1[128];
  int i, j, i0, j0, offs;
  double s0, s1, sq0, sq1, s01;
  double d, r;
```

```
int k, ii, jj,

  for(k=0, k<37; k++)
  {
   jj = (k+182)>>4;
   ii = (k+182) - (jj<<4);
   j0 = jj<<4;
   i0 = ii<<4;

   if (frame_pred_dct || pict_struct!=FRAME_PICTURE)
   mbi[k].dct_type = 0;
   else
   {
    for (j=0; j<8; j++)
    {
     offs = width*((j<<1)+j0) + i0;
     for (i=0; i<16; i++)
     {
      blk0[16*j+i] = cur[offs] - pred[offs];
      blk1[16*j+i] = cur[offs+width] - pred[offs+width],
      offs++;
     }
    }
    /* correlate fields */
    s0=s1=sq0=sq1=s01=0;

    for (i=0; i<128; i++)
    {
     s0+= (double) blk0[i];
     sq0+= (double) (blk0[i]*blk0[i]);
     s1+= (double) blk1[i];
     sq1+= (double) (blk1[i]*blk1[i]);
     s01+= (double) (blk0[i]*blk1[i]);
    }

    s0 = 0.125 * s0;
    s1 = 0.125 * s1;
    d = (sq0-(s0*s0)/2.0)*(sq1-(s1*s1)/2.0);

    if (d>0.0)
    {
     r = (s01-(s0*s1)/2 0)/sqrt(d);
     if (r>0.5)
     mbi[k].dct_type = 0; /* frame DCT */
     else
     mbi[k].dct_type = 1; /* field DCT */
    }
    else
    mbi[k].dct_type = 1; /* field DCT */
   }
  }
}
```

```
#ifndef PI
# ifdef M_PI
#  define PI M_PI
# else
#  define PI 3.14159265358979323846
# endif
#endif


static double c[8][8], /* transform coefficients */

void init_fdct()
{
 char i, j;
 double s;

 for (i=0; i<8; i++)
 {
  s = (i==0) ? sqrt(0.125) : 0.5;

  for (j=0; j<8; j++)
  c[i][j] = s * cos((PI/8.0)*i*(j+0.5));
 }
}

void fdct(int *block)
{
 char i, j, k;
 double s;
 double tmp[64];

 for (i=0, i<8; i++)
  for (j=0; j<8; j++)
  {
   s = 0.0;

   for (k=0; k<8; k++)
   s += c[j][k] * block[8*i+k];

   tmp[8*i+j] = s;
  }

 for (j=0; j<8; j++)
  for (i=0; i<8; i++)
  {
   s = 0.0;

   for (k=0; k<8; k++)
   s += c[i][k] * tmp[8*k+j];

   block[8*i+j] = (int)floor(s+0.499999);

  }
}
```

```
/********************************/
/*                              */
/*   t022 c residing at PE6     */
/*                              */
/********************************/

#include "c:\tc2v2\dos.h"
#include "c:\tc2v2\float.h"
#include "c:\tc2v2\math.h"
#include "c:\tc2v2\ctype.h"
#include "c:\tc2v2\stdio.h"
#include "c:\tc2v2\stdlib.h"
#include "c:\tc2v2\string h"
#include "c:\tc2v2\chan.h"
#include "c:\tc2v2\thread.h"
#include "c:\tc2v2\time.h"
#include "c:\tc2v2\errno.h"

#define GLOBAL /* used by global.h */

#include "c:\nick\h\const1.h"
#include "c:\nick\h\config.h"
#include "c:\nick\h\global h"

static void init _ANSI_ARGS_((void));
static void readparmfile _ANSI_ARGS_((char *fname));

void main(argc,argv,envp,ins,in_ports,outs,out_ports)
int argc,ins,outs;
char *argv[],*envp[];
CHAN *in_ports[],*out_ports[];
{
 /* read parameter file */
 chan_in_word(&M,Link0Input);
 chan_in_word(&nframes,Link0Input);
 chan_in_word(&chroma_format,Link0Input);

 init();
 putseq();

}


static void init()
{
 int i, size;

 init_fdct(),

 chan_in_word(&mb_width,Link0Input);
 chan_in_word(&mb_height,Link0Input);
 chan_in_word(&mb_height2,Link0Input);
 chan_in_word(&width,Link0Input);
 chan_in_word(&height,Link0Input);
```

```
  chan_in_word(&width2,Link0Input);
  chan_in_word(&height2,Link0Input),
  chan_in_word(&chrom_width,Link0Input);
  chan_in_word(&chrom_height,Link0Input);
  chan_in_word(&chrom_width2,Link0Input),
  chan_in_word(&block_count,Link0Input);

  /* clip table */
  clp = (char *)malloc(1024);
  clp+= 384;
  for (i=-384; i<640; i++)
    clp[i] = (i<0) ? 0 : ((i>255) ? 255 : i);

  for (i=0; i<3; i++)
  {
    size = (i==0) ? width*height : chrom_width*chrom_height;

    auxorgframe[i] = (char *)malloc(size);
    predframe[i] = (char *)malloc(size);
  }

  mbinfo = (struct mbinfo *)malloc(37*sizeof(struct mbinfo));
  blocks = (int (*)[64])malloc(37*block_count*sizeof(int [64]));


}

void putseq()
{
 /* this routine assumes (N % M) == 0 */
 int i, j, f;
 unsigned int a, b, c;
 char *neworg[3];
 int loop;

 for (j=0; j<3; j++)
 neworg[j] = auxorgframe[j];


/* loop through all frames in encoding/decoding order */
do{
   chan_in_word(&loop,Link0Input),

   if(loop==nframes) break;

   chan_in_word(&pict_struct,Link0Input);
   chan_in_word(&frame_pred_dct,Link0Input);
   chan_in_message(width*height,neworg[0],Link0Input);
   chan_in_message(chrom_width*chrom_height,neworg[1],Link0Input);
   chan_in_message(chrom_width*chrom_height,neworg[2],Link0Input);
   chan_in_message(width*height,predframe[0],Link0Input);
   chan_in_message(chrom_width*chrom_height,predframe[1],Link0Input);
   chan_in_message(chrom_width*chrom_height,predframe[2],Link0Input);
```

```
/* loop through 37 macroblocks of the picture */

      transform(predframe,neworg,mbinfo,blocks,1);

      for (a=0, a<37; a++)
      {
       chan_out_word(mbinfo[a].dct_type,Link0Output);

       for (b=0, b<block_count; b++)
       for (c=0; c<64; c++)
       chan_out_word(blocks[a*block_count+b][c],Link0Output);
      }

  }while(loop<nframes);

   chan_out_word(loop,Link0Output);

}/*end of main()*/

void dct_type_estimation _ANSI_ARGS_((char *pred,
                                      char *cur,
                                      struct mbinfo *mbi));

static void add_sub_pred _ANSI_ARGS_((char *pred,
                                      char *cur,
                                      int lx, int *blk,
                                      char add_sub));

void transform(char *pred[],
               char *cur[],
               struct mbinfo *mbi,
               int blocks[][64],
               char forward_invers )
{
  int i, j, i1, j1, n, cc, offs, lx;
  int k, ii, jj;

  if (forward_invers)
  dct_type_estimation(pred[0], cur[0], mbi ) ;

/* subtract prediction and transform prediction error */
/* inverse transform prediction error and add prediction */

/* loop through 37 macroblocks of the picture */

     for(k=0; k<37; k++)
     {
     jj = (k+219)>>4;
     ii = (k+219) - (jj<<4);
     j = jj<<4;
     i = ii<<4,

     for (n=0; n<block_count; n++)
     {
```

```
cc = (n<4) ? 0 : (n&1)+1; /* color component index */
if (cc==0)
{
  /* luminance */
  if ((pict_struct==FRAME_PICTURE) && mbi[k].dct_type)
  {
    /* field DCT */
    offs = i + ((n&1)<<3) + width*(j+((n&2)>>1));
    lx = width<<1,
  }
  else
  {
    /* frame DCT */
    offs = i + ((n&1)<<3) + width2*(j+((n&2)<<2));
    lx = width2;
  }

  if (pict_struct==BOTTOM_FIELD)
  offs += width;
}
else
{
  /* chrominance */

  /* scale coordinates */
  i1 = (chroma_format==CHROMA444) ? i : i>>1;
  j1 = (chroma_format!=CHROMA420) ? j : j>>1;

  if ((pict_struct==FRAME_PICTURE) && mbi[k].dct_type
      && (chroma_format!=CHROMA420))
  {
    /* field DCT */
    offs = i1 + (n&8) + chrom_width*(j1+((n&2)>>1));
    lx = chrom_width<<1;
  }
  else
  {
    /* frame DCT */
    offs = i1 + (n&8) + chrom_width2*(j1+((n&2)<<2));
    lx = chrom_width2;
  }

  if (pict_struct==BOTTOM_FIELD)
  offs += chrom_width;
}

if (forward_invers) /* 1 for forward DCT */
{
 add_sub_pred(pred[cc]+offs,
          cur[cc]+offs,
          lx,
          blocks[k*block_count+n],
          0);
```

```
        fdct(blocks[k*block_count+n]);
       }
       else /* 0 for invers DCT */
       {
/*         idct(blocks[k*block_count+n]);*/
        add_sub_pred(pred[cc]+offs,
                     cur[cc]+offs,
                     lx,
                     blocks[k*block_count+n],
                     1);
       }
      }
    }
}


/* add prediction and prediction error, saturate to 0...255 */
/* subtract prediction from block data */
static void add_sub_pred(char *pred,
                     char *cur,
                     int lx, int *blk,
                     char add_sub )
{
  char i, j;

  for (j=0; j<8; j++)
  {
    for (i=0; i<8; i++)
    {
     if (add_sub) /* 1 for addition */
     cur[i] = clp[ blk[i] + pred[i] ];
     else /* 0 for subtractction */
     blk[i] = cur[i] - pred[i];
    }

    blk+= 8,
    cur+= lx;
    pred+= lx;
  }
}


void dct_type_estimation(char *pred,
                     char *cur,
                     struct mbinfo *mbi )
{
  int blk0[128], blk1[128];
  int i, j, i0, j0, offs;
  double s0, s1, sq0, sq1, s01;
  double d, r;
  int k, ii, jj;

    for(k=0, k<37; k++)
```

```
{
jj = (k+219)>>4;
ii = (k+219) - (jj<<4);
j0 = jj<<4;
i0 = ii<<4;

if (frame_pred_dct || pict_struct!=FRAME_PICTURE)
mbi[k].dct_type = 0;
else
{
 for (j=0; j<8; j++)
 {
  offs = width*((j<<1)+j0) + i0;
  for (i=0; i<16; i++)
  {
   blk0[16*j+i] = cur[offs] - pred[offs];
   blk1[16*j+i] = cur[offs+width] - pred[offs+width];
   offs++;
  }
 }
 /* correlate fields */
 s0=s1=sq0=sq1=s01=0;

 for (i=0; i<128; i++)
 {
  s0+= (double) blk0[i];
  sq0+= (double) (blk0[i]*blk0[i]);
  s1+= (double) blk1[i];
  sq1+= (double) (blk1[i]*blk1[i]);
  s01+= (double) (blk0[i]*blk1[i]);
 }

 s0 = 0.125 * s0;
 s1 = 0.125 * s1;
 d = (sq0-(s0*s0)/2.0)*(sq1-(s1*s1)/2.0),

 if (d>0.0)
 {
  r = (s01-(s0*s1)/2.0)/sqrt(d);
  if (r>0.5)
  mbi[k].dct_type = 0; /* frame DCT */
  else
  mbi[k].dct_type = 1; /* field DCT */
 }
 else
 mbi[k].dct_type = 1; /* field DCT */
}
}
}

#ifndef PI
# ifdef M_PI
#  define PI M_PI
```

```
# else
#  define PI 3.14159265358979323846
# endif
#endif


static double c[8][8]; /* transform coefficients */

void init_fdct()
{
  char i, j;
  double s;

  for (i=0; i<8; i++)
  {
    s = (i==0) ? sqrt(0.125) : 0.5;

    for (j=0; j<8; j++)
    c[i][j] = s * cos((PI/8.0)*i*(j+0.5));
  }
}

void fdct(int *block)
{
  char i, j, k;
  double s;
  double tmp[64];

  for (i=0; i<8; i++)
    for (j=0; j<8; j++)
    {
      s = 0.0;

      for (k=0; k<8; k++)
      s += c[j][k] * block[8*i+k];

      tmp[8*i+j] = s;
    }

  for (j=0; j<8; j++)
    for (i=0; i<8; i++)
    {
      s = 0.0;

      for (k=0; k<8; k++)
      s += c[i][k] * tmp[8*k+j];

      block[8*i+j] = (int)floor(s+0.499999);

    }
}
```