

```

/*****
*
*  lencod.c
*  Main program for H.26L encoder
*  Copyright (C) 1999  Telenor Satellite Services,Norway
*
*  Contacts:
*  Inge Lille-Langøy          <inge.lille-Langoy@telenor.com>
*
*  Telenor Satellite Services
*  Keyzers gt.13              tel.:   +47 23 13 86 98
*  N-0130 Oslo,Norway        fax.:   +47 22 77 79 80
*
*****/
#include <string.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <sys/timeb.h>
#include <stdlib.h>
#include "global.h"

void main(int argc,char **argv)
{
    char config_filename[100];
    int bit_use[2][2] ;
    int i,j,k;
    int scale_factor;
    char name[20];

    time_t ltime1;          /* for time measurement */
    time_t ltime2;
    struct _timeb tstruct1;
    struct _timeb tstruct2;
    char timebuf[128];
    int tot_time=0;         /* time for total encoding session */
    int tmp_time;

    struct inp_par    *inp;    /* input parameters from input configuration file
    */
    struct stat_par    *stat;   /* statistics
    */
    struct snr_par     *snr;    /* SNR values
    */
    struct img_par     *img;    /* image parameters
    */

    /* allocate memory */
    inp = (struct inp_par *) calloc(1, sizeof(struct inp_par));
    stat = (struct stat_par *)calloc(1, sizeof(struct stat_par));

```

```

snr = (struct snr_par *) calloc(1, sizeof(struct snr_par));
img = (struct img_par *) calloc(1, sizeof(struct img_par));

if (argc != 2)
{
    printf("Usage: %s <config.dat> \n",argv[0]);
    printf("<config.dat> defines encoder parameters\n");
    exit(-1);
}
strcpy(config_filename,argv[1]);

read_input(inp,config_filename);

printf("-----\n");
printf(" Encoder config file           : %s \n",config_filename);

printf("-----\n");
printf(" Input YUV file                   : %s \n",inp->infile);
printf(" Output H.26L bitstream           : %s \n",inp->outfile);
if (inp->write_dec)
    printf(" Output YUV file(debug)           : test.dec \n");
printf(" Output log file                   : log.dat \n");
printf(" Output statistics file            : stat.dat \n");

printf("-----\n");

img->framerate=INIT_FRAME_RATE; /* The basic frame rate (of
the original sequence) */

if (inp->img_format == QCIF)
    scale_factor=1;
else
    scale_factor=2; /* CIF */

img->width      = IMG_WIDTH      * scale_factor;
img->width_cr   = IMG_WIDTH_CR   * scale_factor;
img->height     = IMG_HEIGHT     * scale_factor;
img->height_cr  = IMG_HEIGHT_CR  * scale_factor;

init(img);

/* Prediction mode is set to -1 outside the frame, indicating that no prediction
can be made from this part*/

for (i=0; i < img->width/BLOCK_SIZE+1; i++)

```

```

{
    img->ipredmode[i+1][0]=-1;
}
for (j=0; j < img->height/BLOCK_SIZE+1; j++)
{
    img->ipredmode[0][j+1]=-1;
}

printf("Frame  Bit/pic  QP   SnrY   SnrU   SnrV   Bitrate  Time(ms)\n");

img->mb_y_upd=0;

for (img->number=0; img->number < inp->no_frames; img->number++)
{
    _ftime( &tstruct1 );    /* start time ms */
    time( &ltime1 );        /* start time s   */

    image(img,inp,stat);

    /* New inloop filter is used for all frames frames */
    loopfilter_new(img);

    /* The 1/4 pixel upsampling of the decoded frame is produced.
       This will be used both for motion search and prediction for future frames
    */
    oneforthpix(img);

    find_snr(snr,img,inp);    /* find SNR for the last image */

    /* write decoded image to file, just for debugging */
    if (inp->write_dec)
    {
        for (i=0; i < img->height; i++)
            for (j=0; j < img->width; j++)
                fputc(min(imgY[i][j],255),p_dec);

        for (k=0; k < 2; ++k)
            for (i=0; i < img->height/2; i++)
                for (j=0; j < img->width/2; j++)
                    fputc(min(imgUV[k][i][j],255),p_dec);
    }

    time( &ltime2 );          /* end time sec */
    _ftime( &tstruct2 );      /* end time ms */
    tmp_time=(ltime2*1000+tstruct2.millitm) - (ltime1*1000+tstruct1.millitm);
    tot_time=tot_time + tmp_time;

    /* Output to terminal */

    stat->bitr=

```

```

(float)stat->bit_ctr_0*img->framerate/(inp->no_frames*(inp->jumpd+1)) +
    stat->bit_ctr*img->framerate/((img->number+1)*(inp->jumpd+1));

printf(" %2d %8d %5d %7.4f %7.4f %7.4f %8.0f %5d\n",
    img->number*(inp->jumpd+1), stat->bit_ctr-stat->bit_ctr_n,
    img->qp, snr->snr_y, snr->snr_u, snr->snr_v, stat->bitr, tmp_time);

if (img->number == 0)
{
    stat->bitr0=stat->bitr;          /* store bitrate for first frame */
    stat->bit_ctr_0=stat->bit_ctr;    /* store bit usage for first frame */
    stat->bit_ctr=0;
}
stat->bit_ctr_n=stat->bit_ctr;      /* store last bit usage */
}

/* write End Of Sequence */

sprintf(tracestring, "----- End of seq ---- " );
put(tracestring,LEN_STARTCODE,EOS);

bit_use[0][0]=1;

bit_use[1][0]=max(1,inp->no_frames-1);

/* Accumulate bit usage for inter and intra frames */

bit_use[0][1]=bit_use[1][1]=0;

for (i=0; i < 9; i++)
    bit_use[1][1] += stat->bit_use_mode_inter[i];

for (j=0;j<2;j++)
{
    bit_use[j][1]+=stat->bit_use_head_mode[j];
    bit_use[j][1]+=stat->tmp_bit_use_cbp[j];
    bit_use[j][1]+=stat->bit_use_coeffY[j];
    bit_use[j][1]+=stat->bit_use_coeffC[j];
}

if (inp->no_frames > 1)
{
stat->bitrate=(bit_use[0][1]+bit_use[1][1])*(float)img->framerate/(inp->no_frames*(
inp->jumpd+1));
}

```

```

fprintf(stdout,"-----\n");
fprintf(stdout, " Freq. for encoded bitstream      :
%1.0f\n",(float)img->framerate/(float)(inp->jumpd+1));
if(inp->hadamard)
    fprintf(stdout," Hadamard transform          : Used\n");
else
    fprintf(stdout," Hadamard transform          : Not used\n");

if(inp->img_format==0)
    fprintf(stdout," Image format                : QCIF\n");
else if(inp->img_format==1)
    fprintf(stdout," Image format                : CIF\n");

if(inp->intra_upd)
    fprintf(stdout," Error robustness            : On\n");
else if(inp->img_format==1)
    fprintf(stdout," Error robustness            : Off\n");

fprintf(stdout, " Search range                    : %d\n",inp->search_range);

fprintf(stdout, " No of ref. frames used in pred      : %d\n",inp->no_multpred);

fprintf(stdout, " Total encoding time for the seq.   : %d\n",tot_time);

fprintf(stdout,"----- Average data all frames
-----\n");
fprintf(stdout," SNR Y(dB)                        : %5.2f\n",snr->snr_ya);
fprintf(stdout," SNR U(dB)                        : %5.2f\n",snr->snr_ua);
fprintf(stdout," SNR V(dB)                        : %5.2f\n",snr->snr_va);
fprintf(stdout," Bit usage                        :
%5.2f\n",(float)(bit_use[1][1]+bit_use[0][1])/img->number);
fprintf(stdout," Bit rate                        : %5.2f\n",stat->bitrate);

fprintf(stdout,"-----\n");
fprintf(stdout,"Exit Telenor encoder ver 4.0\n");

/*
status file
*/
if ((p_stat=fopen("stat.dat","wb"))==0)
{
    printf("Error open file %s \n",p_stat);
    exit(0);
}
fprintf(p_stat," -----
\n");
fprintf(p_stat," This file contains statistics for the last encoded sequence

```

```

\n");
    fprintf(p_stat, " -----
\n");
    fprintf(p_stat, " Sequence : %s\n",inp->infile);
    fprintf(p_stat, " No.of coded pictures : %d\n",inp->no_frames);
    fprintf(p_stat, " Freq. for encoded bitstream :
%3.0f\n", (float)img->framerate/(float)(inp->jumpd+1));
    fprintf(p_stat, " Bitate(kb/s) : %6.2f\n",stat->bitrate);

    if(inp->hadamard)
        fprintf(p_stat, " Hadamard transform : Used\n");
    else
        fprintf(p_stat, " Hadamard transform : Not used\n");

/*
if(inp->postfilter)
    fprintf(p_stat, " Postfilter : Used\n");
else
    fprintf(p_stat, " Postfilter : Not used\n");
*/

if(inp->img_format==0)
    fprintf(p_stat, " Image format : QCIF\n");
else if(inp->img_format==1)
    fprintf(p_stat, " Image format : CIF\n");

if(inp->intra_upd)
    fprintf(p_stat, " Error robustness : On\n");
else if(inp->img_format==1)
    fprintf(p_stat, " Error robustness : Off\n");

fprintf(p_stat, " Search range : %d\n",inp->search_range);
fprintf(p_stat, " No of frame used in pred : %d\n",inp->no_multpred);

fprintf(p_stat, " -----|-----|-----| \n");
fprintf(p_stat, " Item | Intra | All frames | \n");
fprintf(p_stat, " -----|-----|-----| \n");
fprintf(p_stat, " SNR Y(dB) |");
fprintf(p_stat, " %5.2f |",snr->snr_y1);
fprintf(p_stat, " %5.2f | \n",snr->snr_ya);
fprintf(p_stat, " SNR U/V (dB) |");
fprintf(p_stat, " %5.2f/%5.2f |",snr->snr_u1,snr->snr_v1);
fprintf(p_stat, " %5.2f/%5.2f | \n",snr->snr_ua,snr->snr_va);
/* QUANT. */
fprintf(p_stat, " Average quant |");
fprintf(p_stat, " %5d |",absm(inp->qp0));
fprintf(p_stat, " %5.2f
\n", (float)stat->quant1/max(1.0, (float)stat->quant0));
/* MODE */

```

```

fprintf(p_stat, "\n");
fprintf(p_stat, " -----|-----|\n");
fprintf(p_stat, "    Intra          |    Mode used    |\n");
fprintf(p_stat, " -----|-----|\n");

fprintf(p_stat, " Mode 0  intra old\t| %5d          |\n", stat->mode_use_intra[0]);

for (i=1; i<24; i++)
    fprintf(p_stat, " Mode %d intra new\t| %5d\n", i, stat->mode_use_intra[i]);

fprintf(p_stat, " -----|-----|-----|\n");
fprintf(p_stat, "    Inter          |    Mode used    | Vector bit use|\n");
fprintf(p_stat, " -----|-----|-----|\n");

fprintf(p_stat, "\n Mode 0  (copy)  \t| %5d          | %5.0f\n",
stat->mode_use_inter[0], (float)stat->bit_use_mode_inter[0]/(float)bit_use[1][0]);
;
fprintf(p_stat, "\n Mode 1  (16x16) \t| %5d          | %5.0f\n",
stat->mode_use_inter[1], (float)stat->bit_use_mode_inter[1]/(float)bit_use[1][0]);
;
fprintf(p_stat, "\n Mode 2  (16x8)  \t| %5d          | %5.0f\n",
stat->mode_use_inter[2], (float)stat->bit_use_mode_inter[2]/(float)bit_use[1][0]);
;
fprintf(p_stat, "\n Mode 3  (8x16)  \t| %5d          | %5.0f\n",
stat->mode_use_inter[3], (float)stat->bit_use_mode_inter[3]/(float)bit_use[1][0]);
;
fprintf(p_stat, "\n Mode 4  (8x8)   \t| %5d          | %5.0f\n",
stat->mode_use_inter[4], (float)stat->bit_use_mode_inter[4]/(float)bit_use[1][0]);
;
fprintf(p_stat, "\n Mode 5  (8x4)   \t| %5d          | %5.0f\n",
stat->mode_use_inter[5], (float)stat->bit_use_mode_inter[5]/(float)bit_use[1][0]);
;
fprintf(p_stat, "\n Mode 6  (4x8)   \t| %5d          | %5.0f\n",
stat->mode_use_inter[6], (float)stat->bit_use_mode_inter[6]/(float)bit_use[1][0]);
;
fprintf(p_stat, "\n Mode 7  (4x4)   \t| %5d          | %5.0f\n",
stat->mode_use_inter[7], (float)stat->bit_use_mode_inter[7]/(float)bit_use[1][0]);
;
fprintf(p_stat, "\n Mode 8 intra old\t| %5d          |
-----|", stat->mode_use_inter[8], (float)stat->bit_use_mode_inter[8]/(float)
bit_use[1][0]);
for (i=9; i<33; i++)
    fprintf(p_stat, "\n Mode %d intr.new \t| %5d\n", i, stat->mode_use_inter[i]);

fprintf(p_stat, "\n");
/*      BITS. */
fprintf(p_stat, " -----|-----|-----|\n");
fprintf(p_stat, " Bit usage:          | Intra          | Inter          |\n");

```

```

fprintf(p_stat," -----|-----|-----|\n");
fprintf(p_stat," Header+modeinfo\t");
fprintf(p_stat," %5d      |",stat->bit_use_head_mode[0]);
fprintf(p_stat," %5d      |",stat->bit_use_head_mode[1]/bit_use[1][0]);

fprintf(p_stat,"\n");
fprintf(p_stat," CBP Y/C      |");
for (j=0; j < 2; j++)
{
    fprintf(p_stat," %5d      |",stat->tmp_bit_use_cbp[j]/bit_use[j][0]);
}
fprintf(p_stat,"\n");

/* bit usage for coefficients */
fprintf(p_stat," Coeffs. Y      | %7d      | %7d
\n",stat->bit_use_coeffY[0]/bit_use[0][0],stat->bit_use_coeffY[1]/bit_use[1][0]);
fprintf(p_stat," Coeffs. C      | %7d      | %7d
\n",stat->bit_use_coeffC[0]/bit_use[0][0],stat->bit_use_coeffC[1]/bit_use[1][0]);

fprintf(p_stat," -----|-----|-----|\n");
fprintf(p_stat," Total pr.pict.      |");
for (i=0; i < 2; i++)
{
    fprintf(p_stat," %5d      |",bit_use[i][1]/bit_use[i][0]);
}
fprintf(p_stat,"\n");
fprintf(p_stat," -----|-----|-----|\n");

/*
write to log file
*/
if (fopen("log.dat","r")==0)                /* check if file exist */
{
    if ((p_log=fopen("log.dat","a"))==0)      /* append new statistic at the
end */
    {
        printf("Error open file %s  \n",p_log);
        exit(0);
    }
    else                                     /* Create header for new log
file */
    {
        fprintf(p_log,"
-----
----- \n");
        fprintf(p_log,"|          Encoder statistics. This file is generated during
first encoding session, new sessions will be appended
| \n");
        fprintf(p_log,"
-----

```

```

----- \n");
    fprintf(p_log,"| Date | Time | Sequence
|#Img|Quant1|QuantN|Format|Hadamard|Search r|#Ref |Freq |Intra upd|SNRY 1|SNRU
1|SNRV 1|SNRY N|SNRU N|SNRV N|#Bitr 1|#Bitr N|\n");
    fprintf(p_log,"
----- \n");
}
}
else
    p_log=fopen("log.dat","a"); /* File exist,just open for
appending */

    _strdate( timebuf );
    fprintf(p_log,"| %1.5s |",timebuf );

    _strtime( timebuf);
    fprintf(p_log," % 1.5s |",timebuf);

    for (i=0;i<20;i++)
        name[i]=inp->infile[i+max(0,strlen(inp->infile)-20)]; /* write last part of
path, max 20 chars */
    fprintf(p_log,"%20.20s|",name);

    fprintf(p_log,"%3d |",inp->no_frames);
    fprintf(p_log," %2d |",inp->qp0);
    fprintf(p_log," %2d |",inp->qpN);
    if(inp->img_format==0)
        fprintf(p_log," QCIF |");
    else
        fprintf(p_log," CIF |");

    if (inp->hadamard==1)
        fprintf(p_log," ON |");
    else
        fprintf(p_log," OFF |");

    fprintf(p_log," %2d |",inp->search_range );

    fprintf(p_log," %2d |",inp->no_multpred);

    fprintf(p_log," %2d |",img->framerate/(inp->jumpd+1));

    if (inp->intra_upd==1)
        fprintf(p_log," ON |");
    else
        fprintf(p_log," OFF |");

    fprintf(p_log,"%5.3f|",snr->snr_y1);

```

```

    fprintf(p_log,"%5.3f|",snr->snr_u1);
    fprintf(p_log,"%5.3f|",snr->snr_v1);
    fprintf(p_log,"%5.3f|",snr->snr_ya);
    fprintf(p_log,"%5.3f|",snr->snr_ua);
    fprintf(p_log,"%5.3f|",snr->snr_va);
    fprintf(p_log,"%7.0f|",stat->bitr0);
    fprintf(p_log,"%7.0f|\n",stat->bitrate);
}

/*****
*
*   Name :      read_input()
*
*   Description: Read input from configuration file
*
*   Input      : Name of configuration filename
*
*   Output     : none
*
*****/

void read_input(struct inp_par *inp,char config_filename[])
{
    FILE *fd;
    int bck_tmp;
    int i,j;

    for (i=0;i<8;i++)
        for (j=0;j<2;j++)
            inp->bkc_size[i][j]=0;      /* init block size array */

    if((fd=fopen(config_filename,"r")) == NULL)      /* read the decoder
configuration file */
    {
        printf("Error: Control file %s not found\n",config_filename);
        exit(0);
    }

    /* read input parameters */
    fscanf(fd,"%ld",&inp->no_frames);      /* number of frames to be
encoded */
    fscanf(fd,"%*[^\\n]");      /* discard everything to
the beginning of ewnt line */

    fscanf(fd,"%ld",&inp->qp0);      /* QP of first frame (max 31)
*/
    fscanf(fd,"%*[^\\n]");
    if (inp->qp0 > 31 && inp->qp0 <= 0)
    {

```

```

    printf("Error input parameter quant_0,check configuration file\n");
    exit (0);
}

fscanf(fd,"%ld",&inp->qpN); /* QP of remaining frames (max
31) */
fscanf(fd,"%*[^\\n]");
if (inp->qpN > 31 && inp->qpN <= 0)
{
    printf("Error input parameter quant_n,check configuration file\n");
    exit (0);
}

fscanf(fd,"%ld",&inp->jumpd); /* number of frames to skip
in input sequence (e.g 2 takes frame 0,3,6,9...) */
fscanf(fd,"%*[^\\n]");

fscanf(fd,"%ld",&inp->hadamard); /* 0: 'normal' SAD in 1/3
pixel search. 1: use 4x4 Hadamard transform and 'Sum of absolute transform
differens' in 1/3 pixel search */
fscanf(fd,"%*[^\\n]");

fscanf(fd,"%ld",&inp->search_range);
/* search range-integer pel search and 16x16 blocks. The search window is
generally around the predicted vector.
Max vector is 2xsearch_range. For 8x8 and 4x4 block sizes the search range is
1/2 of that for 16x16 blocks. */
fscanf(fd,"%*[^\\n]");
if (inp->search_range > 16)
{
    printf("Error in input parameter search_range, check configuration file\n");
    exit (0);
}

fscanf(fd,"%ld",&inp->no_multpred); /* 1: prediction from the
last frame only. 2: prediction from the last or second last frame etc. Maximum 5
frames */
fscanf(fd,"%*[^\\n]");
if(inp->no_multpred > MAX_MULT_PRED)
{
    printf("Error : input parameter 'no_multpred' exceeds limit, check
configuration file \n");
    exit(0);
}

fscanf(fd,"%ld",&inp->postfilter); /* No postfilter in this
version, not used */
fscanf(fd,"%*[^\\n]");

fscanf(fd,"%ld",&inp->img_format); /* 0: QCIF. 1: CIF */

```

```

fscanf(fd,"%*[^\\n]");
if (inp->img_format != QCIF && inp->img_format != CIF)
{
    printf("Unsupported image format=%d,use QCIF=0 or CIF=1 \\n",inp->img_format);
    exit(0);
}

fscanf(fd,"%ld",&inp->intra_upd);
/* For error robustness.
0: no special action.
1: One GOB/frame is intra
coded as regular 'update'.
2: One GOB every 2 frames
is intra coded etc.
In connection with this
intra update,restrictions is put on
motion vectors to prevent errors to propagate from the past */

fscanf(fd,"%*[^\\n]");
fscanf(fd,"%ld",&inp->write_dec);
/* 0: Do not write decoded
image to file
1: Write decoded image to file */
/* read block sizes for motion search */
fscanf(fd,"%*[^\\n]");
fscanf(fd,"%ld",&bck_tmp);
if (bck_tmp)
{
    inp->blc_size[1][0]=16;
    inp->blc_size[1][1]=16;
}
fscanf(fd,"%ld",&bck_tmp);
fscanf(fd,"%*[^\\n]");
fscanf(fd,"%ld",&bck_tmp);
if (bck_tmp)
{
    inp->blc_size[2][0]=16;
    inp->blc_size[2][1]= 8;
}
fscanf(fd,"%*[^\\n]");
fscanf(fd,"%ld",&bck_tmp);
if (bck_tmp)
{
    inp->blc_size[3][0]= 8;
    inp->blc_size[3][1]=16;
}
fscanf(fd,"%*[^\\n]");
fscanf(fd,"%ld",&bck_tmp);
if (bck_tmp)
{
    inp->blc_size[4][0]= 8;

```

```

    inp->blc_size[4][1]= 8;
}
fscanf(fd,"%*[^\\n]");
fscanf(fd,"%ld",&bck_tmp);
if (bck_tmp)
{
    inp->blc_size[5][0]= 8;
    inp->blc_size[5][1]= 4;
}
fscanf(fd,"%*[^\\n]");
fscanf(fd,"%ld",&bck_tmp);
if (bck_tmp)
{
    inp->blc_size[6][0]= 4;
    inp->blc_size[6][1]= 8;
}
fscanf(fd,"%*[^\\n]");
fscanf(fd,"%ld",&bck_tmp);
if (bck_tmp)
{
    inp->blc_size[7][0]= 4;
    inp->blc_size[7][1]= 4;
}
fscanf(fd,"%*[^\\n]");

fscanf(fd,"%s",inp->infile);                                /* YUV 4:2:0 input format
*/
fscanf(fd,"%*[^\\n]");

fscanf(fd,"%s",inp->outfile);                                /* H.26L compressed output
bitsream */

if ((p_in=fopen(inp->infile,"rb"))==0)
{
    printf("Input file %s does not exist \\n",inp->infile);
    exit(0);
}
if ((p_out=fopen(inp->outfile,"wb"))==0)                    /* bitstream */
{
    printf("Error open file %s  \\n",inp->outfile);
    exit(0);
}
if ((p_dec=fopen("test.dec","wb"))==0)                      /* decodet picture,for
debugging */
{
    printf("Error open file test.dec \\n");
    exit(0);
}
if ((p_trace=fopen("trace.txt","wb"))==0)
{

```

```

        printf("Error open trace file trace.txt \n");
    }
}

```

```

/*****
*
* Name :          init()
*
* Description: Some matrices are initilized here.
*
* Input      : none
*
* Output     : none
*
*****/

```

```

void init(struct img_par *img,struct stat_par *stat)
{
    const int SPIRAL0[9][2]=
    {
        {0,0},{0,-1},{0,1},{-1,0},{1,0},{-1,1},{1,1},{-1,-1},{1,-1}
    };
    int i,j,k,l,i2,l2,ii,ind;

    k=1;
    for (l=1; l < 41; l++)
    {
        l2=l+1;

        for (i=-l+1; i < l; i++)
        {
            for (j=-l; j < l+1; j += l2)
            {
                /* img->spiral_search contains the vector positions organized to perform a
spiral search */
                img->spiral_search[0][k]= i;
                img->spiral_search[1][k]= j;
                k++;
            }
        }

        for (j=-l; j <= l; j++)
        {
            for (i=-l;i <= l; i += l2)
            {
                img->spiral_search[0][k]= i;
                img->spiral_search[1][k]= j;
                k++;
            }
        }
    }
}

```

```

}

/* img->lpfilter is the table used for loop and post filter. */

for (k=0; k < 6; k++)
{
    for (i=1; i <= 100; i++)
    {
        ii=min(k,i);
        img->lpfilter[k][i+300]= ii;
        img->lpfilter[k][300-i]=-ii;
    }
}
/* img->mv_bituse[] is the number of bits used for motion vectors.
It is used to add a portion to the SAD depending on bit usage in the motion
search
*/

img->mv_bituse[0]=1;
ind=0;
for (i=0; i < 9; i++)
{
    ii= 2*i + 3;
    for (j=1; j <= (int)pow(2,i); j++)
    {
        ind++;
        img->mv_bituse[ind]=ii;
    }
}

/*      quad(0:255) SNR quad array */

for (i=0; i < 256; ++i) /* fix from TML1 / TML2 sw, truncation removed */
{
    i2=i*i;
    img->quad[i]=i2;
}
}

```