```
/***********************************************************************
*
*  image.c for H.26L decoder.
*  Copyright (C) 1999  Telenor Satellite Services, Norway
*
*  Contacts:
*  Inge Lille-Langøy               <inge.lille-langoy@telenor.com>
*
*  Telenor Satellite Services
*  P.O.Box 6914 St.Olavs plass
*  N-0130 Oslo, Norway
*
***********************************************************************/

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

#include "global.h"
#include "image.h"

int image(struct img_par *img,struct inp_par *inp)
{
  int len,info;
  int scale_factor;
  int i,j;

  info=get_info(&len);
  img->format=(info&ICIF_MASK)>>1;                /* image format , 0= QCIF, 1=CIF
        */
  img->qp=(info&QP_MASK)>>2;                      /* read quant parameter for
current frame */
  img->tr=(info&TR_MASK)>>7;                      /* 8 bit temporal reference
        */

  info=get_info(&len);

  img->type = (int) pow(2,(len/2))+info-1;        /* find image type INTRA/INTER*/
  if (img->type>3)
      return SEARCH_SYNC;
  if (img->format == QCIF)
    scale_factor=1;                               /* QCIF */
  else
    scale_factor=2;                               /* CIF  */

  img->width      = IMG_WIDTH     * scale_factor; /* luma   */
  img->height     = IMG_HEIGHT    * scale_factor;

  img->width_cr   = IMG_WIDTH_CR  * scale_factor; /* chroma */
  img->height_cr  = IMG_HEIGHT_CR * scale_factor;
```


INTEL-1012
9,641,841

```
  for(i=0;i<img->width/BLOCK_SIZE+1;i++)          /* set edge to -1, indicate
nothing to predict from */
    img->ipredmode[i+1][0]=-1;
  for(j=0;j<img->height/BLOCK_SIZE+1;j++)
    img->ipredmode[0][j+1]=-1;

  for (i=1;i<90;i++)
  {
    for (j=1;j<74;j++)
    {
      loopb[i+1][j+1]=0;
    }
  }
  for (i=1;i<46;i++)
  {
    for (j=1;j<38;j++)
    {
      loopc[i+1][j+1]=0;
    }
  }


  for (img->mb_y=0;img->mb_y < img->height/MB_BLOCK_SIZE;img->mb_y++)
  {
    img->block_y = img->mb_y * BLOCK_SIZE;        /* vertical luma block posision
      */
    img->pix_c_y=img->mb_y *MB_BLOCK_SIZE/2;      /* vertical chroma macroblock
posision  */
    img->pix_y=img->mb_y *MB_BLOCK_SIZE;          /* vertical luma macroblock
posision    */

    for (img->mb_x=0;img->mb_x < img->width/MB_BLOCK_SIZE;img->mb_x++)
    {
      img->block_x=img->mb_x*BLOCK_SIZE;          /* luma block
      */
      img->pix_c_x=img->mb_x*MB_BLOCK_SIZE/2;
      img->pix_x=img->mb_x *MB_BLOCK_SIZE;        /* horisontal luma macroblock
posision    */

      if(macroblock(img)==SEARCH_SYNC)            /* decode one frame */
        return SEARCH_SYNC;                       /* error in bitstream, look for
synk word next frame */
    }
  }

 loopfilter_new(img);

  oneforthpix(img);
```

```
  return DECODING_OK;
}

/************************************************************************
*
*  Name :       oneforthpix()
*
*  Description: Upsample 4 times and store in buffer for multiple
*               reference frames
*
************************************************************************/
void oneforthpix(struct img_par *img)
{
  int imgY_tmp[288][704];
  int i,j,i2,j2;
  int is,ie2,je2;
  int uv;

  img->frame_cycle=img->number % MAX_MULT_PRED;

  for (j=0; j < img->height; j++)
  {
    for (i=0; i < img->width; i++)
    {
      i2=i*2;
      is=(ONE_FOURTH_TAP[0][0]*(imgY[j][i
]+imgY[j][min(img->width-1,i+1)])+

ONE_FOURTH_TAP[1][0]*(imgY[j][max(0,i-1)]+imgY[j][min(img->width-1,i+2)])+

ONE_FOURTH_TAP[2][0]*(imgY[j][max(0,i-2)]+imgY[j][min(img->width-1,i+3)])+16)/32;

      imgY_tmp[j][i2  ]=imgY[j][i];             /* 1/1 pix pos */
      imgY_tmp[j][i2+1]=max(0,min(255,is));     /* 1/2 pix pos */

    }
  }
  for (i=0; i < (img->width-1)*2+1; i++)
  {
    for (j=0; j < img->height; j++)
    {
      j2=j*4;
      is=(ONE_FOURTH_TAP[0][1]*(imgY_tmp[j
][i]+imgY_tmp[min(img->height-1,j+1)][i])+

ONE_FOURTH_TAP[1][1]*(imgY_tmp[max(0,j-1)][i]+imgY_tmp[min(img->height-1,j+2)][i])+

ONE_FOURTH_TAP[2][1]*(imgY_tmp[max(0,j-2)][i]+imgY_tmp[min(img->height-1,j+3)][i])+
16)/32;
```

```
      mref[img->frame_cycle][j2  ][i*2]=imgY_tmp[j][i];     /* 1/2 pix */
      mref[img->frame_cycle][j2+2][i*2]=max(0,min(255,is)); /* 1/2 pix */
    }
  }

  /* 1/4 pix */
  /* luma */
  ie2=(img->width-1)*4;
  je2=(img->height-1)*4;
  for (j=0;j<je2+1;j+=2)
    for (i=0;i<ie2;i+=2)

mref[img->frame_cycle][j][i+1]=(mref[img->frame_cycle][j][i]+mref[img->frame_cycle]
[j][min(ie2,i+2)])/2;


  for (i=0;i<ie2+1;i++)
  {
    for (j=0;j<je2;j+=2)
    {

mref[img->frame_cycle][j+1][i]=(mref[img->frame_cycle][j][i]+mref[img->frame_cycle]
[min(je2,j+2)][i])/2;

      /* "funny posision" */
      if( ((i&3) == 3)&&(((j+1)&3) == 3))
      {
        mref[img->frame_cycle][j+1][i] = (mref[img->frame_cycle][j-2][i-3]+
                                          mref[img->frame_cycle][j+2][i-3]+
                                          mref[img->frame_cycle][j-2][i+1]+
                                          mref[img->frame_cycle][j+2][i+1]+2)/4;
      }
    }
  }



  /*  Chroma: */

  for (uv=0; uv < 2; uv++)
  {
    for (i=0; i < img->width_cr; i++)
    {
      for (j=0; j < img->height_cr; j++)
      {
        mcef[img->frame_cycle][uv][j][i]=imgUV[uv][j][i];/* just copy 1/1 pix,
interpolate "online" */
      }
    }
```

```
  }
}



/*********************************************************************
*
*  Name :       find_snr()
*
*  Description: Find PSNR for all three components.Compare decoded frame with
*               the original sequence. Read inp->jumpd frames to reflect frame
skipping.
*
*
*********************************************************************/
void find_snr(struct snr_par *snr,struct img_par *img,FILE *p_ref,int postfilter)
{
  int i,j;
  int diff_y,diff_u,diff_v;
  int i2,uv;
  int skip_frame;
  static int tr_old=0;
  byte imgY_ref[288][352];
  byte imgUV_ref[2][144][176];

  if (img->number == 0)
    skip_frame = 1;                                 /* start with the first frame
*/
  else
  {
    /* skip frames which are not encoded */
    skip_frame =  img->tr - tr_old;
    if (skip_frame < 0)
      skip_frame = img->tr + (256 - tr_old);         /* tr wraps at 255 */


    tr_old = img->tr;
  }

  for (i2=0; i2 < skip_frame; ++i2)
  {
    for (j=0; j < img->height; ++j)
    {
      for (i=0; i < img->width; ++i)
      {
        imgY_ref[j][i]=fgetc(p_ref);
      }
    }
    for (uv=0; uv < 2; ++uv)
    {
```

```
      for (j=0; j < img->height_cr ; ++j)
      {
        for (i=0; i < img->width_cr; ++i)
        {
          imgUV_ref[uv][j][i]=fgetc(p_ref);
        }
      }
    }
  }

  img->quad[0]=0;
  diff_y=0;
  for (i=0; i < img->width; ++i)
  {
    for (j=0; j < img->height; ++j)
    {
      diff_y += img->quad[abs(imgY[j][i]-imgY_ref[j][i])];
    }
  }

  /*  Chroma */

  diff_u=0;
  diff_v=0;

  for (i=0; i < img->width_cr; ++i)
  {
    for (j=0; j < img->height_cr; ++j)
    {
        diff_u += img->quad[abs(imgUV_ref[0][j][i]-imgUV[0][j][i])];
        diff_v += img->quad[abs(imgUV_ref[1][j][i]-imgUV[1][j][i])];
    }
  }

  /*  Collecting SNR statistics */

  if (diff_y != 0)
  {

snr->snr_y=(float)(10*log10(65025*(float)(img->width)*(img->height)/(float)diff_y))
;        /* luma snr for current frame */
  }

  if (diff_u != 0)
  {

snr->snr_u=(float)(10*log10(65025*(float)(img->width)*(img->height)/(float)(4*diff_
u)));    /*  chroma snr for current frame  */

snr->snr_v=(float)(10*log10(65025*(float)(img->width)*(img->height)/(float)(4*diff_
```

```
v)));    /*  chroma snr for current frame  */
  }


  if (img->number == 0) /* first */
  {

snr->snr_y1=(float)(10*log10(65025*(float)(img->width)*(img->height)/(float)diff_y)
);       /* keep luma snr for first frame */

snr->snr_u1=(float)(10*log10(65025*(float)(img->width)*(img->height)/(float)(4*diff
_u)));   /* keep chroma snr for first frame */

snr->snr_v1=(float)(10*log10(65025*(float)(img->width)*(img->height)/(float)(4*diff
_v)));   /* keep chroma snr for first frame */
    snr->snr_ya=snr->snr_y1;
    snr->snr_ua=snr->snr_u1;
    snr->snr_va=snr->snr_v1;
  }
  else
  {
    snr->snr_ya=(float)(snr->snr_ya*img->number+snr->snr_y)/(img->number+1);
/* average snr chroma for all frames except first */
    snr->snr_ua=(float)(snr->snr_ua*img->number+snr->snr_u)/(img->number+1);
/* average snr luma for all frames except first  */
    snr->snr_va=(float)(snr->snr_va*img->number+snr->snr_v)/(img->number+1);
/* average snr luma for all frames except first  */
  }
}
/************************************************************************
*
*  Name :       loopfilter_new()
*
*  Description: Filter to reduce blocking artifacts. The filter strengh
*               is QP dependent.
*
************************************************************************/
void loopfilter_new(struct img_par *img)
{
  byte imgY_tmp[288][352];                      /* temp luma image */
  byte imgUV_tmp[2][144][176];                  /* temp chroma image */

  int i,j,j4,k,i4;
  int uv;

  /* luma hor */
  for(j=0;j<img->height;j++)
  {
    for(i=0;i<img->width;i++)
    {
```

```
    imgY_tmp[j][i]=imgY[j][i];
  }
}

for(j=0;j<img->height;j++)
{
  j4=j/4;
  for(i=4;i<img->width;i=i+4)
  {
    i4=i/4;
    if((loopb[i4][j4+1]!=0)&&(loopb[i4+1][j4+1]!=0))
    {
      for(k=0;k<8;k++)
        img->li[k]=imgY_tmp[j][i-4+k];
      loop(img,loopb[i/4][j4+1],loopb[i/4+1][j4+1]);
      for (k=2;k<6;k++)
        imgY[j][i-4+k]=img->lu[k];
    }
  }
}

 /* luma vert */
for(j=0;j<img->height;j++)
{
  for(i=0;i<img->width;i++)
  {
    imgY_tmp[j][i]=imgY[j][i];

  }
}

for(i=0;i<img->width;i++)
{
  i4=i/4;
  for(j=4;j<img->height;j=j+4)
  {
    j4=j/4;
    if((loopb[i4+1][j4]!=0)&&(loopb[i4+1][j4+1]!=0))
    {
      for(k=0;k<8;k++)
        img->li[k]=imgY_tmp[j-4+k][i];
      loop(img,loopb[i4+1][j/4],loopb[i4+1][j/4+1]);
      for (k=2;k<6;k++)
        imgY[j-4+k][i]=img->lu[k];
    }
  }
}

/* horizontal chroma */
for (uv=0;uv<2;uv++)
```

```
{
  for(j=0;j<img->height_cr;j++)
  {
    for(i=0;i<img->width_cr;i++)
    {
      imgUV_tmp[uv][j][i]=imgUV[uv][j][i];
    }
  }

  for(j=0;j<img->height_cr;j++)
  {
    j4=j/4;
    for(i=4;i<img->width_cr;i=i+4)
    {
      i4=i/4;
      if((loopc[i4][j4+1]!=0)&&(loopc[i4+1][j4+1]!=0))
      {
        for(k=0;k<8;k++)
          img->li[k]=imgUV_tmp[uv][j][i-4+k];
        loop(img,loopc[i/4][j4+1],loopc[i/4+1][j4+1]);
        for (k=2;k<6;k++)
          imgUV[uv][j][i-4+k]=img->lu[k];
      }
    }
  }

  /* vertical chroma */
  for(j=0;j<img->height_cr;j++)
  {
    for(i=0;i<img->width_cr;i++)
    {
      imgUV_tmp[uv][j][i]=imgUV[uv][j][i];
    }
  }

  for(i=0;i<img->width_cr;i++)
  {
    i4=i/4;
    for(j=4;j<img->height_cr;j=j+4)
    {
      j4=j/4;
      if((loopc[i4+1][j4]!=0)&&(loopc[i4+1][j4+1]!=0))
      {
        for(k=0;k<8;k++)
          img->li[k]=imgUV_tmp[uv][j-4+k][i];
        loop(img,loopc[i4+1][j/4],loopc[i4+1][j/4+1]);
        for (k=2;k<6;k++)
          imgUV[uv][j-4+k][i]=img->lu[k];
      }
    }
```

```
    }
  }
}
/*************************************************************************
*
*  Name :       loop()
*
*  Description: Filter 4 from 8 pix input
*
*************************************************************************/
void loop(struct img_par *img,int ibl,int ibr)
{
  int i0,i1,iaa,ia;
  int ii,iii;
  int il3,il4,icl,ir3,ir4,icr,idl,idr,idlr,lim;
  int isl,isr,islr,id,il1,ir1,il2,ir2;


  lim=LIM[img->qp];

  /* overall activity */
  i0=4*(img->li[2]-img->li[5])-12*(img->li[3]-img->li[4]);
  i1=9*(img->li[2]+img->li[5]-     img->li[3]-img->li[4]);
  iaa=abs(i0)+abs(i1);

  if((ibl==3) || (ibr==3))
    ia=1;
  else
    ia=0;
  for(ii=3;ii>0;ii--)
  {
    iii=(int)pow(2,ii-1);
    if(iaa<(iii*lim))
      ia=4-ii;
  }

  il3=12*abs(img->li[1]+img->li[3]-2*img->li[2]);
  i0=4*(img->li[0]-img->li[3])-12*(img->li[1]-img->li[2]);
  i1=9*(img->li[0]+img->li[3]-     img->li[1]-img->li[2]);
  il4=abs(i0)+abs(i1);
  icl=1;
  if(il3<lim)
    icl++;
  if(il4<lim)
    icl++;

  ir3=12*abs(img->li[4]+img->li[6]-2*img->li[5]);
  i0=4*(img->li[4]-img->li[7])-12*(img->li[5]-img->li[6]);
  i1=9*(img->li[4]+img->li[7]-     img->li[5]-img->li[6]);
  ir4=abs(i0)+abs(i1);
```

```
  icr=1;
  if(ir3<lim)
    icr++;
  if (ir4<lim)
    icr++;

  idl=min(ia,icl);
  idr=min(ia,icr);
  idlr=(idl+idr)/2;

  isl=FILTER_STR[img->qp][ibl];
  isr=FILTER_STR[img->qp][ibr];

  islr=(isl+isr+icl+icr-2)/2;

  /* start filtering */
/* edge */
  id =0;
  if(idlr==2 || idlr==1)
    id=(3*(img->li[2]-img->li[5])-8*(img->li[3]-img->li[4]))/16;
  if(idlr==3 )
    id=(img->li[1]+img->li[2]-8*(img->li[3]-img->li[4])-img->li[5]-img->li[6])/16;
  id =max(-islr,min(id,islr));
  il1=img->li[3]+id;
  ir1=img->li[4]-id;
  img->li[3]=il1;/* set back left edge  */
  img->li[4]=ir1;/* set back right edge */

/* center left */
  id =0;
  if(idl==2)
    id=5*(img->li[1]-2*img->li[2]+img->li[3])/16;
  if(idl==3 )
    id=3*(img->li[0]+img->li[1]-4*img->li[2]+img->li[3]+img->li[4])/16;
  id =max(-isl,min(id,isl));
  il2=img->li[2]+id;
/* center right  */
  id =0;
  if(idr==2)
    id=5*(img->li[4]-2*img->li[5]+img->li[6])/16;
  if(idr==3 )
    id=3*(img->li[3]+img->li[4]-4*img->li[5]+img->li[6]+img->li[7])/16;
  id =max(-isr,min(id,isr));
  ir2=img->li[5]+id;

  img->lu[2]=max(0,min(255,il2));
  img->lu[3]=max(0,min(255,il1));
  img->lu[4]=max(0,min(255,ir1));
  img->lu[5]=max(0,min(255,ir2));
```

```
}

/************************************************************************
*
*  Name :       loopfilter()
*
*  Description: Filter to reduce blocking artifacts. The filter strengh
*               is QP dependent.Applied only for intra frames.
*               From TML1
*
************************************************************************/
void loopfilter(struct img_par *img)
{
  int i,j;
  int id,uv;

  /*  Luma horizontal */

  for (j=0; j < img->height; j++)
  {
    for (i=3; i < img->width-1 ; i += BLOCK_SIZE)
    {
      id=(imgY[j][i-1]-imgY[j][i+2]-((imgY[j][i]-imgY[j][i+1])*4))/8;
      id=img->lpfilter[FILTER_STR[img->qp][1]][id+300];
      imgY[j][i]= max(0,min(imgY[j][i]+id,255));
      imgY[j][i+1]= max(0,min(imgY[j][i+1]-id,255));
    }
  }

  /*  Luma vertical */

  for (i=0; i < img->width; i++)
  {
    for (j=3; j < img->height-1; j += BLOCK_SIZE)
    {
      id=(imgY[j-1][i]-imgY[j+2][i]-((imgY[j][i]-imgY[j+1][i])*4))/8;
      id=img->lpfilter[FILTER_STR[img->qp][1]][id+300];
      imgY[j][i]= max(0,min(imgY[j][i]+id,255));
      imgY[j+1][i]= max(0,min(imgY[j+1][i]-id,255));
    }
  }

  /*  Chroma horizontal */

  for (uv=0; uv < 2; uv++)
  {
    for (j=0; j < img->height_cr; j++)
    {
      for (i=3; i < img->width_cr-1; i += BLOCK_SIZE)
      {
```

```
id=(imgUV[uv][j][i-1]-imgUV[uv][j][i+2]-((imgUV[uv][j][i]-imgUV[uv][j][i+1])*4))/8;
        id=img->lpfilter[FILTER_STR[img->qp][1]][id+300];
        imgUV[uv][j][i]= max(0,min(imgUV[uv][j][i]+id,255));
        imgUV[uv][j][i+1]= max(0,min(imgUV[uv][j][i+1]-id,255));
      }
    }

    /*  Chroma vertical */

    for (i=0; i < img->width_cr; i++)
    {
      for (j=3; j < img->height_cr-1; j += BLOCK_SIZE)
      {

id=(imgUV[uv][j-1][i]-imgUV[uv][j+2][i]-((imgUV[uv][j][i]-imgUV[uv][j+1][i])*4))/8;
        id=img->lpfilter[FILTER_STR[img->qp][1]][id+300];
        imgUV[uv][j][i]= max(0,min(imgUV[uv][j][i]+id,255));
        imgUV[uv][j+1][i]= max(0,min(imgUV[uv][j+1][i]-id,255));
      }
    }
  }
}
```