/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \file mv\_search.c

\*

\* \brief

\* Motion Vector Search, unified for B and P Pictures

\*

\* \author

\* Main contributors (see contributors.h for copyright, address and affiliation details)

\* - Stephan Wenger <stewe@cs.tu-berlin.de>

\* - Inge Lille-Langoy <inge.lille-langoy@telenor.com>

\* - Rickard Sjoberg <rickard.sjoberg@era.ericsson.se>

\* - Stephan Wenger <stewe@cs.tu-berlin.de>

\* - Jani Lainema <jani.lainema@nokia.com>

\* - Detlev Marpe

\* - Thomas Wedi <wedi@tnt.uni-hannover.de>

\* - Heiko Schwarz

\* - Alexis Michael Tourapis <alexismt@ieee.org>

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#include "contributors.h"

#include <math.h>

#include <limits.h>

#include <time.h>

#include "global.h"

#include "image.h"

#include "mv\_search.h"

#include "refbuf.h"

#include "memalloc.h"

#include "mb\_access.h"

#include "macroblock.h"

#include "mc\_prediction.h"

#include "conformance.h"

#include "mode\_decision.h"

// Motion estimation distortion header file

#include "me\_distortion.h"

#include "me\_distortion\_otf.h"

// Motion estimation search algorithms

#include "me\_epzs.h"

#include "me\_epzs\_int.h"

#include "me\_fullfast.h"

#include "me\_fullfast\_otf.h"

#include "me\_fullsearch.h"

#include "me\_umhex.h"

#include "me\_umhexsmp.h"

#include "rdoq.h"

static const short bx0[5][4] = {{0,0,0,0}, {0,0,0,0}, {0,0,0,0}, {0,2,0,0}, {0,2,0,2}};

static const short by0[5][4] = {{0,0,0,0}, {0,0,0,0}, {0,2,0,0}, {0,0,0,0}, {0,0,2,2}};

static distblk GetSkipCostMB (Macroblock \*currMB, int lambda);

static distblk BiPredBlockMotionSearch(Macroblock \*currMB, MEBlock \*, MotionVector\*, int, int , int\*);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Set search range. This needs to be changed to provide 2D support

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void get\_search\_range(MEBlock \*mv\_block, InputParameters \*p\_Inp, short ref, int blocktype)

{

SearchWindow \*searchRange = &mv\_block->searchRange;

\*searchRange = mv\_block->p\_Vid->searchRange;

//----- set search range ---

if (p\_Inp->full\_search == 1)

{

int scale = (imin(ref, 1) + 1);

searchRange->min\_x /= scale;

searchRange->max\_x /= scale;

searchRange->min\_y /= scale;

searchRange->max\_y /= scale;

}

else if (p\_Inp->full\_search != 2)

{

int scale = ((imin(ref, 1) + 1) \* imin(2, blocktype));

searchRange->min\_x /= scale;

searchRange->max\_x /= scale;

searchRange->min\_y /= scale;

searchRange->max\_y /= scale;

}

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Set search range. This needs to be changed to provide 2D support

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

static inline void set\_me\_parameters( PicMotionParams \*\*motion, const MotionVector \*all\_mv, int list, char ref, int step\_h, int step\_v, int pic\_block\_y, int pic\_block\_x)

{

int i, j;

// Set first line

for (j = pic\_block\_y; j < pic\_block\_y + step\_v; j++)

{

for (i=pic\_block\_x; i<pic\_block\_x + step\_h; i++)

{

motion[j][i].mv[list] = \*all\_mv;

motion[j][i].ref\_idx[list] = ref;

}

}

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Set ME access method

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void set\_access\_method(int \*access\_method, MotionVector \*blk, int min\_x, int min\_y, int max\_x, int max\_y)

{

if ( (blk->mv\_x > min\_x) && (blk->mv\_x < max\_x) && (blk->mv\_y > min\_y) && (blk->mv\_y < max\_y))

{

\*access\_method = FAST\_ACCESS;

}

else

{

\*access\_method = UMV\_ACCESS;

}

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Initialize ME engine

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void init\_ME\_engine(Macroblock \*currMB)

{

InputParameters \*p\_Inp = currMB->p\_Inp;

VideoParameters \*p\_Vid = currMB->p\_Vid;

switch (p\_Inp->SearchMode[p\_Vid->view\_id])

{

case EPZS:

EPZS\_setup\_engine(currMB, p\_Inp);

break;

case UM\_HEX:

currMB->IntPelME = UMHEXIntegerPelBlockMotionSearch;

currMB->BiPredME = UMHEXBipredIntegerPelBlockMotionSearch;

currMB->SubPelBiPredME = sub\_pel\_bipred\_motion\_estimation;

currMB->SubPelME = UMHEXSubPelBlockME;

break;

case UM\_HEX\_SIMPLE:

currMB->IntPelME = smpUMHEXIntegerPelBlockMotionSearch;

currMB->BiPredME = smpUMHEXBipredIntegerPelBlockMotionSearch;

currMB->SubPelBiPredME = sub\_pel\_bipred\_motion\_estimation;

currMB->SubPelME = smpUMHEXSubPelBlockME;

break;

case FULL\_SEARCH:

currMB->IntPelME = full\_search\_motion\_estimation;

currMB->BiPredME = full\_search\_bipred\_motion\_estimation;

currMB->SubPelBiPredME = sub\_pel\_bipred\_motion\_estimation;

currMB->SubPelME = sub\_pel\_motion\_estimation;

break;

case FAST\_FULL\_SEARCH:

default:

currMB->IntPelME = fast\_full\_search\_motion\_estimation;

currMB->BiPredME = full\_search\_bipred\_motion\_estimation;

currMB->SubPelBiPredME = sub\_pel\_bipred\_motion\_estimation;

currMB->SubPelME = sub\_pel\_motion\_estimation;

currMB->p\_SetupFastFullPelSearch = (p\_Inp->OnTheFlyFractMCP) ? (SetupFastFullPelSearch\_otf):(setup\_fast\_full\_search);

break;

}

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Prepare Motion Estimation parameters for single list ME

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void PrepareMEParams(Slice \*currSlice, MEBlock \*mv\_block, int ChromaMEEnable, int list, int ref)

{

if (mv\_block->apply\_weights)

{

mv\_block->weight\_luma = currSlice->wp\_weight[list][ref][0];

mv\_block->offset\_luma = currSlice->wp\_offset[list][ref][0];

if ( ChromaMEEnable)

{

mv\_block->weight\_cr[0] = currSlice->wp\_weight[list][ref][1];

mv\_block->weight\_cr[1] = currSlice->wp\_weight[list][ref][2];

mv\_block->offset\_cr[0] = currSlice->wp\_offset[list][ref][1];

mv\_block->offset\_cr[1] = currSlice->wp\_offset[list][ref][2];

}

}

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Prepare Motion Estimation parameters for bipred list ME

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void PrepareBiPredMEParams(Slice \*currSlice, MEBlock \*mv\_block, int ChromaMEEnable, int list, int list\_offset, int ref)

{

if (mv\_block->apply\_weights)

{

if (list == LIST\_0)

{

mv\_block->weight1 = currSlice->wbp\_weight[list\_offset ][ref][0][0];

mv\_block->weight2 = currSlice->wbp\_weight[list\_offset + LIST\_1][ref][0][0];

mv\_block->offsetBi = (currSlice->wp\_offset[list\_offset ][ref][0] + currSlice->wp\_offset[list\_offset + LIST\_1][ref][0] + 1)>>1;

if ( ChromaMEEnable)

{

mv\_block->weight1\_cr[0] = currSlice->wbp\_weight[list\_offset ][ref][0][1];

mv\_block->weight1\_cr[1] = currSlice->wbp\_weight[list\_offset ][ref][0][2];

mv\_block->weight2\_cr[0] = currSlice->wbp\_weight[list\_offset + LIST\_1][ref][0][1];

mv\_block->weight2\_cr[1] = currSlice->wbp\_weight[list\_offset + LIST\_1][ref][0][2];

mv\_block->offsetBi\_cr[0] = (currSlice->wp\_offset[list\_offset ][ref][1] + currSlice->wp\_offset[list\_offset + LIST\_1][ref][1] + 1) >> 1;

mv\_block->offsetBi\_cr[1] = (currSlice->wp\_offset[list\_offset ][ref][2] + currSlice->wp\_offset[list\_offset + LIST\_1][ref][2] + 1) >> 1;

}

}

else

{

mv\_block->weight1 = currSlice->wbp\_weight[list\_offset + LIST\_1][0 ][ref][0];

mv\_block->weight2 = currSlice->wbp\_weight[list\_offset ][0 ][ref][0];

mv\_block->offsetBi = (currSlice->wp\_offset[list\_offset + LIST\_1][0][0] + currSlice->wp\_offset[list\_offset][0][0] + 1)>>1;

if ( ChromaMEEnable)

{

mv\_block->weight1\_cr[0] = currSlice->wbp\_weight[list\_offset + LIST\_1][0 ][ref][1];

mv\_block->weight1\_cr[1] = currSlice->wbp\_weight[list\_offset + LIST\_1][0 ][ref][2];

mv\_block->weight2\_cr[0] = currSlice->wbp\_weight[list\_offset ][0 ][ref][1];

mv\_block->weight2\_cr[1] = currSlice->wbp\_weight[list\_offset ][0 ][ref][2];

mv\_block->offsetBi\_cr[0] = (currSlice->wp\_offset[list\_offset + LIST\_1][0 ][1] + currSlice->wp\_offset[list\_offset ][0 ][1] + 1) >> 1;

mv\_block->offsetBi\_cr[1] = (currSlice->wp\_offset[list\_offset + LIST\_1][0 ][2] + currSlice->wp\_offset[list\_offset ][0 ][2] + 1) >> 1;

}

}

}

else

{

mv\_block->weight1 = (short) (1 << currSlice->luma\_log\_weight\_denom);

mv\_block->weight2 = (short) (1 << currSlice->luma\_log\_weight\_denom);

mv\_block->offsetBi = 0;

if ( ChromaMEEnable)

{

mv\_block->weight1\_cr[0] = 1<<currSlice->chroma\_log\_weight\_denom;

mv\_block->weight1\_cr[1] = 1<<currSlice->chroma\_log\_weight\_denom;

mv\_block->weight2\_cr[0] = 1<<currSlice->chroma\_log\_weight\_denom;

mv\_block->weight2\_cr[1] = 1<<currSlice->chroma\_log\_weight\_denom;

mv\_block->offsetBi\_cr[0] = 0;

mv\_block->offsetBi\_cr[1] = 0;

}

}

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Get current block spatial neighbors

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void get\_neighbors(Macroblock \*currMB, // <-- current Macroblock

PixelPos \*block, // <--> neighbor blocks

int mb\_x, // <-- block x position

int mb\_y, // <-- block y position

int blockshape\_x // <-- block width

)

{

VideoParameters \*p\_Vid = currMB->p\_Vid;

int \*mb\_size = p\_Vid->mb\_size[IS\_LUMA];

get4x4Neighbour(currMB, mb\_x - 1, mb\_y , mb\_size, &block[0]);

get4x4Neighbour(currMB, mb\_x, mb\_y - 1, mb\_size, &block[1]);

get4x4Neighbour(currMB, mb\_x + blockshape\_x, mb\_y - 1, mb\_size, &block[2]);

get4x4Neighbour(currMB, mb\_x - 1, mb\_y - 1, mb\_size, &block[3]);

if (mb\_y > 0)

{

if (mb\_x < 8) // first column of 8x8 blocks

{

if (mb\_y == 8 )

{

if (blockshape\_x == MB\_BLOCK\_SIZE)

block[2].available = 0;

}

else if (mb\_x + blockshape\_x == 8)

{

block[2].available = 0;

}

}

else if (mb\_x + blockshape\_x == MB\_BLOCK\_SIZE)

{

block[2].available = 0;

}

}

if (!block[2].available)

{

block[2] = block[3];

}

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Initialize the motion search

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void init\_motion\_search\_module (VideoParameters \*p\_Vid, InputParameters \*p\_Inp)

{

int bits;

int i\_min, i\_max,k;

int i, l;

int search\_range\_orig = p\_Inp->SepViewInterSearch ? imax( p\_Inp->search\_range[0], p\_Inp->search\_range[1] ) : p\_Inp->search\_range[0];

int search\_range = search\_range\_orig;

int max\_search\_points = imax(9, (2 \* search\_range + 1) \* (2 \* search\_range + 1));

int max\_ref\_bits = 1 + 2 \* (int)floor(log(imax(16, p\_Vid->max\_num\_references + 1)) / log(2) + 1e-10);

int max\_ref = (1<<((max\_ref\_bits>>1)+1))-1;

int number\_of\_subpel\_positions = 4 \* (2\*search\_range+3);

int max\_mv\_bits = 3 + 2 \* (int)ceil (log(number\_of\_subpel\_positions + 1) / log(2) + 1e-10);

int max\_mvd = p\_Inp->UseMVLimits? imax(4\*imax(p\_Inp->SetMVXLimit, p\_Inp->SetMVYLimit), ((1<<( max\_mv\_bits >>1) ) - 1)): ((1<<( max\_mv\_bits >>1)) - 1);

p\_Vid->max\_mvd = max\_mvd;

p\_Vid->imgpel\_abs\_range = (imax(p\_Vid->max\_pel\_value\_comp[0],p\_Vid->max\_pel\_value\_comp[1]) + 1) \* 64;

//===== CREATE ARRAYS =====

//-----------------------------

if ((p\_Vid->spiral\_search = (MotionVector\*)calloc(max\_search\_points, sizeof(MotionVector))) == NULL)

no\_mem\_exit("init\_motion\_search\_module: p\_Vid->spiral\_search");

if ((p\_Vid->spiral\_hpel\_search = (MotionVector\*)calloc(max\_search\_points, sizeof(MotionVector))) == NULL)

no\_mem\_exit("init\_motion\_search\_module: p\_Vid->spiral\_hpel\_search");

if ((p\_Vid->spiral\_qpel\_search = (MotionVector\*)calloc(max\_search\_points, sizeof(MotionVector))) == NULL)

no\_mem\_exit("init\_motion\_search\_module: p\_Vid->spiral\_qpel\_search");

if ((p\_Vid->mvbits = (int\*)calloc(2 \* max\_mvd + 1, sizeof(int))) == NULL)

no\_mem\_exit("init\_motion\_search\_module: p\_Vid->mvbits");

if ((p\_Vid->refbits = (int\*)calloc(max\_ref, sizeof(int))) == NULL)

no\_mem\_exit("init\_motion\_search\_module: p\_Vid->refbits");

#if (JM\_MEM\_DISTORTION)

if ((p\_Vid->imgpel\_abs = (int\*)calloc(p\_Vid->imgpel\_abs\_range, sizeof(int))) == NULL)

no\_mem\_exit("init\_motion\_search\_module: p\_Vid->imgpel\_abs");

if ((p\_Vid->imgpel\_quad = (int\*)calloc(p\_Vid->imgpel\_abs\_range, sizeof(int))) == NULL)

no\_mem\_exit("init\_motion\_search\_module: p\_Vid->imgpel\_quad");

p\_Vid->imgpel\_abs += p\_Vid->imgpel\_abs\_range / 2;

p\_Vid->imgpel\_quad += p\_Vid->imgpel\_abs\_range / 2;

#endif

if (p\_Vid->max\_num\_references)

get\_mem4Ddistblk (&p\_Vid->motion\_cost, 8, 2, p\_Vid->max\_num\_references, 4);

//--- set array offsets ---

p\_Vid->mvbits += max\_mvd;

//===== INIT ARRAYS =====

//---------------------------

//--- init array: motion vector bits ---

p\_Vid->mvbits[0] = 1;

for (bits = 3; bits <= max\_mv\_bits; bits += 2)

{

i\_max = (short) (1 << (bits >> 1));

i\_min = i\_max >> 1;

for (i = i\_min; i < i\_max; i++)

p\_Vid->mvbits[-i] = p\_Vid->mvbits[i] = bits;

}

//--- init array: reference frame bits ---

p\_Vid->refbits[0] = 1;

for (bits=3; bits<=max\_ref\_bits; bits+=2)

{

i\_max = (short) (1 << ((bits >> 1) + 1)) - 1;

i\_min = i\_max >> 1;

for (i = i\_min; i < i\_max; i++)

p\_Vid->refbits[i] = bits;

}

#if (JM\_MEM\_DISTORTION)

//--- init array: absolute value ---

p\_Vid->imgpel\_abs[0] = 0;

for (i=1; i<p\_Vid->imgpel\_abs\_range / 2; i++)

{

p\_Vid->imgpel\_abs[i] = p\_Vid->imgpel\_abs[-i] = i;

}

//--- init array: square value ---

p\_Vid->imgpel\_quad[0] = 0;

for (i=1; i<p\_Vid->imgpel\_abs\_range / 2; i++)

{

p\_Vid->imgpel\_quad[i] = p\_Vid->imgpel\_quad[-i] = i \* i;

}

#endif

//--- init array: search pattern ---

p\_Vid->spiral\_search[0].mv\_x = p\_Vid->spiral\_search[0].mv\_y = 0;

p\_Vid->spiral\_hpel\_search[0].mv\_x = p\_Vid->spiral\_hpel\_search[0].mv\_y = 0;

p\_Vid->spiral\_qpel\_search[0].mv\_x = p\_Vid->spiral\_qpel\_search[0].mv\_y = 0;

for (k=1, l=1; l <= imax(1,search\_range); l++)

{

for (i=-l+1; i< l; i++)

{

p\_Vid->spiral\_search[k].mv\_x = (short) i;

p\_Vid->spiral\_search[k].mv\_y = (short) -l;

p\_Vid->spiral\_hpel\_search[k].mv\_x = (short) (i<<1);

p\_Vid->spiral\_hpel\_search[k].mv\_y = (short) -(l<<1);

p\_Vid->spiral\_qpel\_search[k].mv\_x = (short) (i<<2);

p\_Vid->spiral\_qpel\_search[k++].mv\_y = (short) -(l<<2);

p\_Vid->spiral\_search[k].mv\_x = (short) i;

p\_Vid->spiral\_search[k].mv\_y = (short) l;

p\_Vid->spiral\_hpel\_search[k].mv\_x = (short) (i<<1);

p\_Vid->spiral\_hpel\_search[k].mv\_y = (short) (l<<1);

p\_Vid->spiral\_qpel\_search[k].mv\_x = (short) (i<<2);

p\_Vid->spiral\_qpel\_search[k++].mv\_y = (short) (l<<2);

}

for (i=-l; i<=l; i++)

{

p\_Vid->spiral\_search[k].mv\_x = (short) -l;

p\_Vid->spiral\_search[k].mv\_y = (short) i;

p\_Vid->spiral\_hpel\_search[k].mv\_x = (short) -(l<<1);

p\_Vid->spiral\_hpel\_search[k].mv\_y = (short) (i<<1);

p\_Vid->spiral\_qpel\_search[k].mv\_x = (short) -(l<<2);

p\_Vid->spiral\_qpel\_search[k++].mv\_y = (short) (i<<2);

p\_Vid->spiral\_search[k].mv\_x = (short) l;

p\_Vid->spiral\_search[k].mv\_y = (short) i;

p\_Vid->spiral\_hpel\_search[k].mv\_x = (short) (l<<1);

p\_Vid->spiral\_hpel\_search[k].mv\_y = (short) (i<<1);

p\_Vid->spiral\_qpel\_search[k].mv\_x = (short) (l<<2);

p\_Vid->spiral\_qpel\_search[k++].mv\_y = (short) (i<<2);

}

}

// set global variable prior to ME

p\_Vid->start\_me\_refinement\_hp = (p\_Inp->ChromaMEEnable == 1 || p\_Inp->MEErrorMetric[F\_PEL] != p\_Inp->MEErrorMetric[H\_PEL] ) ? 0 : 1;

p\_Vid->start\_me\_refinement\_qp = (p\_Inp->ChromaMEEnable == 1 || p\_Inp->MEErrorMetric[H\_PEL] != p\_Inp->MEErrorMetric[Q\_PEL] ) ? 0 : 1;

select\_distortion(p\_Vid, p\_Inp);

// Setup Distortion Metrics depending on refinement level

if( p\_Inp->OnTheFlyFractMCP )

{

for (i=0; i<3; i++)

{

switch( p\_Inp->MEErrorMetric[i])

{

case ERROR\_SAD:

p\_Vid->computeUniPred[i] = computeSAD\_otf ;

p\_Vid->computeUniPred[i + 3] = computeSADWP\_otf ;

p\_Vid->computeBiPred1[i] = computeBiPredSAD1\_otf ;

p\_Vid->computeBiPred2[i] = computeBiPredSAD2\_otf ;

break;

case ERROR\_SSE:

p\_Vid->computeUniPred[i] = computeSSE\_otf;

p\_Vid->computeUniPred[i + 3] = computeSSEWP\_otf;

p\_Vid->computeBiPred1[i] = computeBiPredSSE1\_otf;

p\_Vid->computeBiPred2[i] = computeBiPredSSE2\_otf;

break;

case ERROR\_SATD :

default:

p\_Vid->computeUniPred[i] = computeSATD\_otf ;

p\_Vid->computeUniPred[i + 3] = computeSATDWP\_otf ;

p\_Vid->computeBiPred1[i] = computeBiPredSATD1\_otf ;

p\_Vid->computeBiPred2[i] = computeBiPredSATD2\_otf ;

break;

}

}

}

else

{

for (i=0; i<3; i++)

{

switch(p\_Inp->MEErrorMetric[i])

{

case ERROR\_SAD:

p\_Vid->computeUniPred[i] = computeSAD;

p\_Vid->computeUniPred[i + 3] = computeSADWP;

p\_Vid->computeBiPred1[i] = computeBiPredSAD1;

p\_Vid->computeBiPred2[i] = computeBiPredSAD2;

break;

case ERROR\_SSE:

p\_Vid->computeUniPred[i] = computeSSE;

p\_Vid->computeUniPred[i + 3] = computeSSEWP;

p\_Vid->computeBiPred1[i] = computeBiPredSSE1;

p\_Vid->computeBiPred2[i] = computeBiPredSSE2;

break;

case ERROR\_SATD :

default:

p\_Vid->computeUniPred[i] = computeSATD;

p\_Vid->computeUniPred[i + 3] = computeSATDWP;

p\_Vid->computeBiPred1[i] = computeBiPredSATD1;

p\_Vid->computeBiPred2[i] = computeBiPredSATD2;

break;

}

}

}

if (!p\_Inp->IntraProfile)

{

if(p\_Inp->SearchMode[0] == FAST\_FULL\_SEARCH || p\_Inp->SearchMode[1] == FAST\_FULL\_SEARCH)

initialize\_fast\_full\_search (p\_Vid, p\_Inp);

if (p\_Inp->SearchMode[0] == UM\_HEX || p\_Inp->SearchMode[1] == UM\_HEX)

UMHEX\_DefineThreshold(p\_Vid);

}

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Free memory used by motion search

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void

clear\_motion\_search\_module (VideoParameters \*p\_Vid, InputParameters \*p\_Inp)

{

//int search\_range = p\_Inp->search\_range;

//int number\_of\_subpel\_positions = 4 \* (2\*search\_range+3);

//int max\_mv\_bits = 3 + 2 \* (int)ceil (log(number\_of\_subpel\_positions + 1) / log(2) + 1e-10);

int max\_mvd = p\_Vid->max\_mvd; //(1<<( max\_mv\_bits >>1) ) - 1;

//--- correct array offset ---

p\_Vid->mvbits -= max\_mvd;

#if (JM\_MEM\_DISTORTION)

p\_Vid->imgpel\_abs -= p\_Vid->imgpel\_abs\_range / 2;

p\_Vid->imgpel\_quad -= p\_Vid->imgpel\_abs\_range / 2;

#endif

//--- delete arrays ---

free (p\_Vid->spiral\_search);

free (p\_Vid->spiral\_hpel\_search);

free (p\_Vid->spiral\_qpel\_search);

free (p\_Vid->mvbits);

free (p\_Vid->refbits);

#if (JM\_MEM\_DISTORTION)

free (p\_Vid->imgpel\_abs);

free (p\_Vid->imgpel\_quad);

#endif

if (p\_Vid->motion\_cost)

free\_mem4Ddistblk (p\_Vid->motion\_cost);

if ((p\_Inp->SearchMode[0] == FAST\_FULL\_SEARCH || p\_Inp->SearchMode[1] == FAST\_FULL\_SEARCH) && (!p\_Inp->IntraProfile) )

clear\_fast\_full\_search (p\_Vid);

}

static inline int mv\_bit\_cost(Macroblock \*currMB, MotionVector \*\*all\_mv, int cur\_list, short cur\_ref, int by, int bx, int step\_v0, int step\_v, int step\_h0, int step\_h, int mvd\_bits)

{

int v, h;

MotionVector predMV;

PixelPos block[4]; // neighbor blocks

VideoParameters \*p\_Vid = currMB->p\_Vid;

for (v=by; v<by + step\_v0; v+=step\_v)

{

for (h=bx; h<bx + step\_h0; h+=step\_h)

{

get\_neighbors(currMB, block, h, v, step\_h);

// Lets recompute MV predictor. This should avoid any problems with alterations of the motion vectors after ME

currMB->GetMVPredictor (currMB, block, &predMV, cur\_ref, p\_Vid->enc\_picture->mv\_info, cur\_list, h, v, step\_h, step\_v);

mvd\_bits += p\_Vid->mvbits[ all\_mv[v>>2][h>>2].mv\_x - predMV.mv\_x ];

mvd\_bits += p\_Vid->mvbits[ all\_mv[v>>2][h>>2].mv\_y - predMV.mv\_y ];

}

}

return mvd\_bits;

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Motion Cost for Bidirectional modes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

distblk BPredPartitionCost (Macroblock \*currMB,

int blocktype,

int block8x8,

short ref\_l0,

short ref\_l1,

int lambda\_factor,

int list)

{

VideoParameters \*p\_Vid = currMB->p\_Vid;

Slice \*currSlice = currMB->p\_Slice;

DecodedPictureBuffer \*p\_Dpb = p\_Vid->p\_Dpb\_layer[p\_Vid->dpb\_layer\_id];

imgpel \*\*cur\_img = p\_Vid->pCurImg;

short pic\_pix\_x, pic\_pix\_y;

short v, h;

distblk mcost;

int mvd\_bits = 0;

short parttype = (short) (blocktype < 4 ? blocktype : 4);

short step\_h0 = block\_size[ parttype][0];

short step\_v0 = block\_size[ parttype][1];

short step\_h = block\_size[blocktype][0];

short step\_v = block\_size[blocktype][1];

short by0\_part = by0[parttype][block8x8] << 2;

short bx0\_part = bx0[parttype][block8x8] << 2;

short block\_size\_x = block\_size[blocktype][0];

short block\_size\_y = block\_size[blocktype][1];

MotionVector \*\*all\_mv\_l0 = currSlice->bipred\_mv[list][LIST\_0][ref\_l0][blocktype];

MotionVector \*\*all\_mv\_l1 = currSlice->bipred\_mv[list][LIST\_1][ref\_l1][blocktype];

imgpel \*\*mb\_pred = currSlice->mb\_pred[0];

// List0

mvd\_bits = mv\_bit\_cost(currMB, all\_mv\_l0, LIST\_0, ref\_l0, by0\_part, bx0\_part, step\_v0, step\_v, step\_h0, step\_h, mvd\_bits);

// List1

mvd\_bits = mv\_bit\_cost(currMB, all\_mv\_l1, LIST\_1, ref\_l1, by0\_part, bx0\_part, step\_v0, step\_v, step\_h0, step\_h, mvd\_bits);

mcost = weighted\_cost (lambda\_factor, mvd\_bits);

// Get prediction

for (v = by0\_part; v < by0\_part + step\_v0; v = (short) (v + block\_size\_y))

{

for (h = bx0\_part; h < bx0\_part + step\_h0; h = (short) (h + block\_size\_x))

{

p\_Dpb->pf\_luma\_prediction\_bi (currMB, h, v, block\_size\_x, block\_size\_y, blocktype, blocktype, ref\_l0, ref\_l1, list);

}

}

//----- cost of residual signal -----

if ((!currSlice->p\_Inp->Transform8x8Mode) || (blocktype>4))

{

short diff16[16];

short \*diff;

pic\_pix\_y = currMB->opix\_y;

pic\_pix\_x = currMB->pix\_x;

for (v = by0\_part; v < by0\_part + step\_v0; v += 4)

{

for (h = bx0\_part; h < bx0\_part + step\_h0; h += 4)

{

diff = diff16;

calcDifference(cur\_img, pic\_pix\_x+h, pic\_pix\_y+v, mb\_pred, h, v, 4, 4, diff);

mcost += p\_Vid->distortion4x4 (diff16, DISTBLK\_MAX);

}

}

}

else

{

short diff64[64];

short \*diff;

pic\_pix\_y = currMB->opix\_y;

pic\_pix\_x = currMB->pix\_x;

for (v = by0\_part; v < by0\_part + step\_v0; v += 8)

{

for (h = bx0\_part; h < bx0\_part + step\_h0; h += 8)

{

diff = diff64;

calcDifference(cur\_img, pic\_pix\_x+h, pic\_pix\_y+v, mb\_pred, h, v, 8, 8, diff);

mcost += p\_Vid->distortion8x8(diff64, DISTBLK\_MAX);

}

}

}

return mcost;

}

void update\_mv\_block(Macroblock \*currMB, MEBlock \*mv\_block, int h, int v)

{

mv\_block->block\_x = (short) h;

mv\_block->block\_y = (short) v;

mv\_block->pos\_x = (short) (currMB->pix\_x + (h << 2));

mv\_block->pos\_y = (short) (currMB->opix\_y + (v << 2));

mv\_block->pos\_x2 = (short) (mv\_block->pos\_x >> 2);

mv\_block->pos\_y2 = (short) (mv\_block->pos\_y >> 2);

mv\_block->pos\_x\_padded = (short) (mv\_block->pos\_x << 2);

mv\_block->pos\_y\_padded = (short) (mv\_block->pos\_y << 2);

mv\_block->pos\_cr\_x = (short) (mv\_block->pos\_x >> currMB->p\_Vid->shift\_cr\_x);

mv\_block->pos\_cr\_y = (short) (mv\_block->pos\_y >> currMB->p\_Vid->shift\_cr\_y);

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Init motion vector block

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void init\_mv\_block(Macroblock \*currMB, MEBlock \*mv\_block, short blocktype, int list, char ref\_idx, short mb\_x, short mb\_y)

{

InputParameters \*p\_Inp = currMB->p\_Inp;

VideoParameters \*p\_Vid = currMB->p\_Vid;

Slice \*currSlice = currMB->p\_Slice;

mv\_block->blocktype = blocktype;

mv\_block->blocksize\_x = block\_size[blocktype][0]; // horizontal block size

mv\_block->blocksize\_y = block\_size[blocktype][1]; // vertical block size

// update position info

update\_mv\_block(currMB, mv\_block, mb\_x, mb\_y);

mv\_block->list = (char) list;

mv\_block->ref\_idx = ref\_idx;

mv\_block->mv[LIST\_0].mv\_x = 0;

mv\_block->mv[LIST\_0].mv\_y = 0;

mv\_block->mv[LIST\_1].mv\_x = 0;

mv\_block->mv[LIST\_1].mv\_y = 0;

// Init WP parameters

mv\_block->p\_Vid = p\_Vid;

mv\_block->p\_Slice = currSlice;

mv\_block->cost = INT\_MAX;

mv\_block->search\_pos2 = 9;

mv\_block->search\_pos4 = 9;

if (p\_Inp->ChromaMEEnable)

get\_mem2Dpel(&mv\_block->orig\_pic, 3, mv\_block->blocksize\_x \* mv\_block->blocksize\_y);

else

get\_mem2Dpel(&mv\_block->orig\_pic, 1, mv\_block->blocksize\_x \* mv\_block->blocksize\_y);

mv\_block->ChromaMEEnable = p\_Inp->ChromaMEEnable;

mv\_block->apply\_bi\_weights = p\_Inp->UseWeightedReferenceME && ((currSlice->slice\_type == B\_SLICE) && p\_Vid->active\_pps->weighted\_bipred\_idc != 0);

mv\_block->apply\_weights = p\_Inp->UseWeightedReferenceME && ( currSlice->weighted\_prediction != 0 );

if (p\_Inp->ChromaMEEnable)

{

mv\_block->blocksize\_cr\_x = (short) (mv\_block->blocksize\_x >> p\_Vid->shift\_cr\_x);

mv\_block->blocksize\_cr\_y = (short) (mv\_block->blocksize\_y >> p\_Vid->shift\_cr\_y);

mv\_block->ChromaMEWeight = p\_Inp->ChromaMEWeight;

}

if (mv\_block->apply\_weights)

{

// If implicit WP, single list weights are always non weighted

if ((currSlice->slice\_type == B\_SLICE) && (p\_Vid->active\_pps->weighted\_bipred\_idc == 2))

{

mv\_block->computePredFPel = p\_Vid->computeUniPred[F\_PEL];

mv\_block->computePredHPel = p\_Vid->computeUniPred[H\_PEL];

mv\_block->computePredQPel = p\_Vid->computeUniPred[Q\_PEL];

}

else

{

mv\_block->computePredFPel = p\_Vid->computeUniPred[F\_PEL + 3];

mv\_block->computePredHPel = p\_Vid->computeUniPred[H\_PEL + 3];

mv\_block->computePredQPel = p\_Vid->computeUniPred[Q\_PEL + 3];

}

mv\_block->computeBiPredFPel = p\_Vid->computeBiPred2[F\_PEL];

mv\_block->computeBiPredHPel = p\_Vid->computeBiPred2[H\_PEL];

mv\_block->computeBiPredQPel = p\_Vid->computeBiPred2[Q\_PEL];

}

else

{

mv\_block->computePredFPel = p\_Vid->computeUniPred[F\_PEL];

mv\_block->computePredHPel = p\_Vid->computeUniPred[H\_PEL];

mv\_block->computePredQPel = p\_Vid->computeUniPred[Q\_PEL];

mv\_block->computeBiPredFPel = p\_Vid->computeBiPred1[F\_PEL];

mv\_block->computeBiPredHPel = p\_Vid->computeBiPred1[H\_PEL];

mv\_block->computeBiPredQPel = p\_Vid->computeBiPred1[Q\_PEL];

}

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* free motion vector block

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void free\_mv\_block(MEBlock \*mv\_block)

{

if (mv\_block->orig\_pic)

{

free\_mem2Dpel(mv\_block->orig\_pic);

}

}

void get\_original\_block(VideoParameters \*p\_Vid, MEBlock \*mv\_block)

{

//==================================

//===== GET ORIGINAL BLOCK =====

//==================================

imgpel \*orig\_pic\_tmp = mv\_block->orig\_pic[0];

int bsx = mv\_block->blocksize\_x;

int pic\_pix\_x = mv\_block->pos\_x;

int i, j;

imgpel \*\*cur\_img = &p\_Vid->pCurImg[mv\_block->pos\_y];

for (j = 0; j < mv\_block->blocksize\_y; j++)

{

memcpy(orig\_pic\_tmp,&cur\_img[j][pic\_pix\_x], bsx \* sizeof(imgpel));

orig\_pic\_tmp += bsx;

}

if ( p\_Vid->p\_Inp->ChromaMEEnable )

{

bsx = mv\_block->blocksize\_cr\_x;

pic\_pix\_x = mv\_block->pos\_cr\_x;

// copy the original cmp1 and cmp2 data to the orig\_pic matrix

for ( i = 1; i<=2; i++)

{

cur\_img = &p\_Vid->pImgOrg[i][mv\_block->pos\_cr\_y];

orig\_pic\_tmp = mv\_block->orig\_pic[i];

for (j = 0; j < mv\_block->blocksize\_cr\_y; j++)

{

memcpy(orig\_pic\_tmp, &(cur\_img[j][pic\_pix\_x]), bsx \* sizeof(imgpel));

orig\_pic\_tmp += bsx;

}

}

}

}

void CheckSearchRange(VideoParameters \*p\_Vid, MotionVector \*pPredMV, MotionVector \*pSWC, MEBlock \*mv\_block)

{

int iMaxMVD = p\_Vid->max\_mvd - 2;

SearchWindow \*searchRange = &mv\_block->searchRange;

int left = pSWC->mv\_x + searchRange->min\_x;

int right = pSWC->mv\_x + searchRange->max\_x;

int top = pSWC->mv\_y + searchRange->min\_y;

int down = pSWC->mv\_y + searchRange->max\_y;

left = iClip3(pPredMV->mv\_x - iMaxMVD, pPredMV->mv\_x + iMaxMVD, left);

right = iClip3(pPredMV->mv\_x - iMaxMVD, pPredMV->mv\_x + iMaxMVD, right);

top = iClip3(pPredMV->mv\_y - iMaxMVD, pPredMV->mv\_y + iMaxMVD, top);

down = iClip3(pPredMV->mv\_y - iMaxMVD, pPredMV->mv\_y + iMaxMVD, down);

if(left<right && top<down)

{

pSWC->mv\_x = (short) ((left + right)>>1);

pSWC->mv\_y = (short) ((top + down)>>1);

searchRange->min\_x = left - pSWC->mv\_x;

searchRange->max\_x = imin(pSWC->mv\_x-left, right-pSWC->mv\_x);

searchRange->min\_y = top - pSWC->mv\_y;

searchRange->max\_y = imin(pSWC->mv\_y-top, down-pSWC->mv\_y);

}

else

{

\*pSWC = \*pPredMV;

}

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Block motion search

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

distblk //!< minimum motion cost after search

BlockMotionSearch (Macroblock \*currMB, //!< Current Macroblock

MEBlock \*mv\_block, //!< Motion estimation information block

int mb\_x, //!< x-coordinate inside macroblock

int mb\_y, //!< y-coordinate inside macroblock

int\* lambda\_factor) //!< lagrangian parameter for determining motion cost

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// FILE \* motion\_vector\_fp;

// motion\_vector\_fp = fopen("/Users/liangsiyang/Documents/USC-learning/EE-669/HW3/motion\_vector.dat", "a");

// if (motion\_vector\_fp == NULL) {

// printf("ERROR in creating motion vector storage file!\n");

// }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// each 48-pel line stores the 16 luma pels (at 0) followed by 8 or 16 crcb[0] (at 16) and crcb[1] (at 32) pels

// depending on the type of chroma subsampling used: YUV 4:4:4, 4:2:2, and 4:2:0

Slice \*currSlice = currMB->p\_Slice;

VideoParameters \*p\_Vid = currMB->p\_Vid;

InputParameters \*p\_Inp = currMB->p\_Inp;

int i, j;

distblk max\_value = DISTBLK\_MAX;

distblk min\_mcost = max\_value;

int block\_x = (mb\_x>>2);

int block\_y = (mb\_y>>2);

int bsx = mv\_block->blocksize\_x;

int bsy = mv\_block->blocksize\_y;

short pic\_pix\_x = (short) (currMB->pix\_x + mb\_x);

int blocktype = mv\_block->blocktype;

int list = mv\_block->list;

short ref = mv\_block->ref\_idx;

MotionVector \*mv = &mv\_block->mv[list], pred;

MotionVector \*\*all\_mv = &currSlice->all\_mv[list][ref][blocktype][block\_y];

distblk \*prevSad = (p\_Inp->SearchMode[p\_Vid->view\_id] == EPZS)? currSlice->p\_EPZS->distortion[list + currMB->list\_offset][blocktype - 1]: NULL;

get\_neighbors(currMB, mv\_block->block, mb\_x, mb\_y, bsx);

PrepareMEParams(currSlice, mv\_block, p\_Inp->ChromaMEEnable, list + currMB->list\_offset, ref);

//==================================

//===== GET ORIGINAL BLOCK =====

//==================================

if (blocktype > 4)

get\_original\_block(p\_Vid, mv\_block);

//===========================================

//===== GET MOTION VECTOR PREDICTOR =====

//===========================================

if (p\_Inp->SearchMode[p\_Vid->view\_id] == UM\_HEX)

{

p\_Vid->p\_UMHex->UMHEX\_blocktype = blocktype;

p\_Vid->p\_UMHex->bipred\_flag = 0;

UMHEXSetMotionVectorPredictor(currMB, &pred, p\_Vid->enc\_picture->mv\_info, ref, list, mb\_x, mb\_y, bsx, bsy, mv\_block);

}

else if (p\_Inp->SearchMode[p\_Vid->view\_id] == UM\_HEX\_SIMPLE)

{

smpUMHEX\_setup(currMB, ref, list, block\_y, block\_x, blocktype, currSlice->all\_mv );

currMB->GetMVPredictor (currMB, mv\_block->block, &pred, ref, p\_Vid->enc\_picture->mv\_info, list, mb\_x, mb\_y, bsx, bsy);

}

else

{

currMB->GetMVPredictor (currMB, mv\_block->block, &pred, ref, p\_Vid->enc\_picture->mv\_info, list, mb\_x, mb\_y, bsx, bsy);

}

//==================================

//===== INTEGER-PEL SEARCH =====

//==================================

if (p\_Inp->EPZSSubPelGrid)

{

\*mv = pred;

}

else

{

#if (JM\_INT\_DIVIDE)

mv->mv\_x = (short) (((pred.mv\_x + 2) >> 2) \* 4);

mv->mv\_y = (short) (((pred.mv\_y + 2) >> 2) \* 4);

#else

mv->mv\_x = (short) ((pred.mv\_x / 4) \* 4);

mv->mv\_y = (short) ((pred.mv\_y / 4) \* 4);

#endif

}

if (p\_Inp->DisableMEPrediction == TRUE)

{

mv->mv\_x = 0;

mv->mv\_y = 0;

}

if (!p\_Inp->rdopt)

{

MotionVector center = \*mv;

//--- adjust search center so that the (0,0)-vector is inside ---

mv->mv\_x = (short) iClip3 (mv\_block->searchRange.min\_x, mv\_block->searchRange.max\_x, mv->mv\_x);

mv->mv\_y = (short) iClip3 (mv\_block->searchRange.min\_y, mv\_block->searchRange.max\_y, mv->mv\_y);

//mvbits overflow checking;

if((mv->mv\_x != center.mv\_x) || (mv->mv\_y != center.mv\_y))

CheckSearchRange(p\_Vid, &center, mv, mv\_block);

}

// valid search range limits could be precomputed once during the initialization process

clip\_mv\_range(p\_Vid, 0, mv, Q\_PEL);

//--- perform motion search ---

min\_mcost = currMB->IntPelME (currMB, &pred, mv\_block, min\_mcost, lambda\_factor[F\_PEL]);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* changes \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// fwrite(&mv->mv\_x, sizeof(short), 1, motion\_vector\_fp);

//

// fwrite(&mv->mv\_y, sizeof(short), 1, motion\_vector\_fp);

//

// fclose(motion\_vector\_fp);

// //printf("\nInteger-pel motion vector x=%d, y=%d\n", mv->mv\_x, mv->mv\_y);

//

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* end changes \*\*f\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//==============================

//===== SUB-PEL SEARCH =====

//==============================

mv\_block->ChromaMEEnable = (p\_Inp->ChromaMEEnable == ME\_YUV\_FP\_SP ) ? TRUE : FALSE; // set it externally

if (!p\_Inp->DisableSubpelME[p\_Vid->view\_id])

{

if (p\_Inp->SearchMode[p\_Vid->view\_id] != EPZS || (ref == 0 || currSlice->structure != FRAME || (ref > 0 && min\_mcost < 3.5 \* prevSad[pic\_pix\_x >> 2])))

{

if ( !p\_Vid->start\_me\_refinement\_hp )

{

min\_mcost = max\_value;

}

min\_mcost = currMB->SubPelME (currMB, &pred, mv\_block, min\_mcost, lambda\_factor);

}

}

// clip mvs after me is performed (is not exactly the best)

// better solution is to modify search window appropriately

clip\_mv\_range(p\_Vid, 0, mv, Q\_PEL);

if (!p\_Inp->rdopt)

{

// Get the skip mode cost

if (blocktype == 1 && (currSlice->slice\_type == P\_SLICE|| (currSlice->slice\_type == SP\_SLICE) ))

{

distblk cost;

FindSkipModeMotionVector (currMB);

cost = GetSkipCostMB (currMB, lambda\_factor[Q\_PEL]);

if (cost < min\_mcost)

{

min\_mcost = cost;

\*mv = currSlice->all\_mv [0][0][0][0][0];

}

}

}

//===============================================

//===== SET MV'S AND RETURN MOTION COST =====

//===============================================

// Set first line

for (i=block\_x; i < block\_x + (bsx>>2); i++)

{

all\_mv[0][i] = \*mv;

}

// set all other lines

for (j=1; j < (bsy>>2); j++)

{

memcpy(&all\_mv[j][block\_x], &all\_mv[0][block\_x], (bsx>>2) \* sizeof(MotionVector));

}

// Bipred ME consideration: returns minimum bipred cost

if (is\_bipred\_enabled(p\_Vid, blocktype) && (ref == 0))

{

BiPredBlockMotionSearch(currMB, mv\_block, &pred, mb\_x, mb\_y, lambda\_factor);

}

return min\_mcost;

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Bi-predictive motion search

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

static distblk BiPredBlockMotionSearch(Macroblock \*currMB, //!< Current Macroblock

MEBlock \*mv\_block,

MotionVector \*pred\_mv, //!< current list motion vector predictor

int mb\_x, //!< x-coordinate inside macroblock

int mb\_y, //!< y-coordinate inside macroblock

int\* lambda\_factor) //!< lagrangian parameter for determining motion cost

{

VideoParameters \*p\_Vid = currMB->p\_Vid;

InputParameters \*p\_Inp = currMB->p\_Inp;

Slice \*currSlice = currMB->p\_Slice;

int list = mv\_block->list;

int i, j;

short bipred\_type = list ? 0 : 1;

MotionVector \*\*\*\*\* bipred\_mv = currSlice->bipred\_mv[bipred\_type];

distblk min\_mcostbi = DISTBLK\_MAX;

MotionVector \*mv = &mv\_block->mv[list];

MotionVector bimv, tempmv;

MotionVector pred\_mv1, pred\_mv2, pred\_bi;

MotionVector \*bi\_mv1 = NULL, \*bi\_mv2 = NULL;

short iterlist = (short) list;

int block\_x = (mb\_x>>2);

int block\_y = (mb\_y>>2);

int blocktype = mv\_block->blocktype;

int bsx = mv\_block->blocksize\_x;

int bsy = mv\_block->blocksize\_y;

//PixelPos block[4]; // neighbor blocks

//get\_neighbors(currMB, mv\_block->block, mb\_x, mb\_y, bsx);

if (p\_Inp->SearchMode[p\_Vid->view\_id] == UM\_HEX)

{

p\_Vid->p\_UMHex->bipred\_flag = 1;

UMHEXSetMotionVectorPredictor(currMB, &pred\_bi, p\_Vid->enc\_picture->mv\_info, 0, list ^ 1, mb\_x, mb\_y, bsx, bsy, mv\_block);

}

else

currMB->GetMVPredictor (currMB, mv\_block->block, &pred\_bi, 0, p\_Vid->enc\_picture->mv\_info, list ^ 1, mb\_x, mb\_y, bsx, bsy);

if ((p\_Inp->SearchMode[p\_Vid->view\_id] != EPZS) || (p\_Inp->EPZSSubPelGrid == 0))

{

mv->mv\_x = ((mv->mv\_x + 2) >> 2) \* 4;

mv->mv\_y = ((mv->mv\_y + 2) >> 2) \* 4;

bimv.mv\_x = ((pred\_bi.mv\_x + 2) >> 2) \* 4;

bimv.mv\_y = ((pred\_bi.mv\_y + 2) >> 2) \* 4;

}

else

{

bimv = pred\_bi;

}

//Bi-predictive motion Refinements

for (mv\_block->iteration\_no = 0; mv\_block->iteration\_no <= p\_Inp->BiPredMERefinements; mv\_block->iteration\_no++)

{

if (mv\_block->iteration\_no & 0x01)

{

pred\_mv1 = \*pred\_mv;

pred\_mv2 = pred\_bi;

bi\_mv1 = mv;

bi\_mv2 = &bimv;

iterlist = (short) list;

}

else

{

pred\_mv1 = pred\_bi;

pred\_mv2 = \*pred\_mv;

bi\_mv1 = &bimv;

bi\_mv2 = mv;

iterlist = (short) (list ^ 1);

}

tempmv = \*bi\_mv1;

PrepareBiPredMEParams(currSlice, mv\_block, mv\_block->ChromaMEEnable, iterlist, currMB->list\_offset, mv\_block->ref\_idx);

// Get bipred mvs for list iterlist given previously computed mvs from other list

min\_mcostbi = currMB->BiPredME (currMB, iterlist,

&pred\_mv1, &pred\_mv2, bi\_mv1, bi\_mv2, mv\_block,

(p\_Inp->BiPredMESearchRange[p\_Vid->view\_id] <<2)>>mv\_block->iteration\_no, min\_mcostbi, lambda\_factor[F\_PEL]);

if (mv\_block->iteration\_no > 0 && (tempmv.mv\_x == bi\_mv1->mv\_x) && (tempmv.mv\_y == bi\_mv1->mv\_y))

{

break;

}

}

if (!p\_Inp->DisableSubpelME[p\_Vid->view\_id])

{

if (p\_Inp->BiPredMESubPel)

{

if ( !p\_Vid->start\_me\_refinement\_hp )

min\_mcostbi = DISTBLK\_MAX;

PrepareBiPredMEParams(currSlice, mv\_block, mv\_block->ChromaMEEnable, iterlist, currMB->list\_offset, mv\_block->ref\_idx);

min\_mcostbi = currMB->SubPelBiPredME (currMB, mv\_block, iterlist, &pred\_mv1, &pred\_mv2, bi\_mv1, bi\_mv2, min\_mcostbi, lambda\_factor);

}

if (p\_Inp->BiPredMESubPel==2)

{

if ( !p\_Vid->start\_me\_refinement\_qp )

min\_mcostbi = DISTBLK\_MAX;

PrepareBiPredMEParams(currSlice, mv\_block, mv\_block->ChromaMEEnable, iterlist ^ 1, currMB->list\_offset, mv\_block->ref\_idx);

min\_mcostbi = currMB->SubPelBiPredME (currMB, mv\_block, iterlist ^ 1, &pred\_mv2, &pred\_mv1, bi\_mv2, bi\_mv1, min\_mcostbi, lambda\_factor);

}

}

clip\_mv\_range(p\_Vid, 0, bi\_mv1, Q\_PEL);

clip\_mv\_range(p\_Vid, 0, bi\_mv2, Q\_PEL);

for (j=block\_y; j < block\_y + (bsy>>2); j++)

{

for (i=block\_x ; i < block\_x + (bsx>>2); i++)

{

bipred\_mv[iterlist ][(short) mv\_block->ref\_idx][blocktype][j][i] = \*bi\_mv1;

bipred\_mv[iterlist ^ 1][(short) mv\_block->ref\_idx][blocktype][j][i] = \*bi\_mv2;

}

}

return min\_mcostbi;

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Motion Cost for Bidirectional modes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

distblk BIDPartitionCost (Macroblock \*currMB,

int blocktype,

int block8x8,

char cur\_ref[2],

int lambda\_factor)

{

VideoParameters \*p\_Vid = currMB->p\_Vid;

Slice \*currSlice = currMB->p\_Slice;

DecodedPictureBuffer \*p\_Dpb = p\_Vid->p\_Dpb\_layer[p\_Vid->dpb\_layer\_id];

imgpel \*\*cur\_img = p\_Vid->pCurImg;

short pic\_pix\_x, pic\_pix\_y;

int v, h;

distblk mcost;

int mvd\_bits = 0;

int parttype = (blocktype < 4 ? blocktype : 4);

int step\_h0 = block\_size[ parttype][0];

int step\_v0 = block\_size[ parttype][1];

int step\_h = block\_size[blocktype][0];

int step\_v = block\_size[blocktype][1];

int bx = bx0[parttype][block8x8] << 2;

int by = by0[parttype][block8x8] << 2;

short block\_size\_x = block\_size[blocktype][0]; // this is the same as step\_h and could be removed

short block\_size\_y = block\_size[blocktype][1]; // this is the same as step\_v and could be removed

MotionVector \*\*all\_mv\_l0 = currSlice->all\_mv [LIST\_0][(int) cur\_ref[LIST\_0]][blocktype];

MotionVector \*\*all\_mv\_l1 = currSlice->all\_mv [LIST\_1][(int) cur\_ref[LIST\_1]][blocktype];

short bipred\_me = 0; //no bipred for this case

imgpel \*\*mb\_pred = currSlice->mb\_pred[0];

int list\_mode[2];

list\_mode[0] = blocktype;

list\_mode[1] = blocktype;

//----- cost for motion vector bits -----

// Should write a separate, small function to do this processing

// List0

mvd\_bits = mv\_bit\_cost(currMB, all\_mv\_l0, LIST\_0, cur\_ref[LIST\_0], by, bx, step\_v0, step\_v, step\_h0, step\_h, mvd\_bits);

// List1

mvd\_bits = mv\_bit\_cost(currMB, all\_mv\_l1, LIST\_1, cur\_ref[LIST\_1], by, bx, step\_v0, step\_v, step\_h0, step\_h, mvd\_bits);

mcost = weighted\_cost (lambda\_factor, mvd\_bits);

// Get prediction

for (v = by; v < by + step\_v0; v += block\_size\_y)

{

for (h = bx; h < bx + step\_h0; h += block\_size\_x)

{

p\_Dpb->pf\_luma\_prediction (currMB, h, v, block\_size\_x, block\_size\_y, 2, list\_mode, cur\_ref, bipred\_me);

}

}

//----- cost of residual signal -----

if ((!currSlice->p\_Inp->Transform8x8Mode) || (blocktype>4))

{

short diff16[16];

short \*diff;

pic\_pix\_y = (short) currMB->opix\_y;

pic\_pix\_x = (short) currMB->pix\_x;

for (v= by; v < by + step\_v0; v += BLOCK\_SIZE)

{

for (h = bx; h < bx + step\_h0; h += BLOCK\_SIZE)

{

diff = diff16;

calcDifference(cur\_img, pic\_pix\_x+h, pic\_pix\_y+v, mb\_pred, h,v, 4, 4, diff);

mcost += p\_Vid->distortion4x4 (diff16, DISTBLK\_MAX);

}

}

}

else

{

short diff64[64];

short \*diff;

pic\_pix\_y = (short) currMB->opix\_y;

pic\_pix\_x = (short) currMB->pix\_x;

for (v= by; v < by + step\_v0; v += BLOCK\_SIZE\_8x8)

{

for (h = bx; h < bx + step\_h0; h += BLOCK\_SIZE\_8x8)

{

diff = diff64;

calcDifference(cur\_img, pic\_pix\_x+h, pic\_pix\_y+v, mb\_pred, h, v, 8, 8, diff);

mcost += p\_Vid->distortion8x8(diff64, DISTBLK\_MAX);

}

}

}

return mcost;

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Get cost for skip mode for an macroblock

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

static distblk GetSkipCostMB (Macroblock \*currMB, int lambda)

{

Slice \*currSlice = currMB->p\_Slice;

VideoParameters \*p\_Vid = currMB->p\_Vid;

InputParameters \*p\_Inp = currMB->p\_Inp;

DecodedPictureBuffer \*p\_Dpb = p\_Vid->p\_Dpb\_layer[p\_Vid->dpb\_layer\_id];

distblk cost = 0;

int block;

imgpel \*\*mb\_pred = currSlice->mb\_pred[0];

char cur\_ref[2] = {0, 0};

int list\_mode[2] = {0, 0};

//===== prediction of 16x16 skip block =====

p\_Dpb->pf\_luma\_prediction (currMB, 0, 0, MB\_BLOCK\_SIZE, MB\_BLOCK\_SIZE, 0, list\_mode, cur\_ref, 0);

if (p\_Inp->Transform8x8Mode == 0)

{

short diff16[16];

short \*diff;

int block\_y, block\_x;//, i, j;

int mb\_x, mb\_y;

int pic\_pix\_y = currMB->opix\_y;

int pic\_pix\_x = currMB->pix\_x;

for(block = 0;block < 4; block++)

{

mb\_y = (block >> 1)<<3;

mb\_x = (block & 0x01)<<3;

for (block\_y = mb\_y; block\_y < mb\_y + 8; block\_y += 4)

{

for (block\_x = mb\_x; block\_x < mb\_x + 8; block\_x += 4)

{

diff = diff16;

//===== get displaced frame difference ======

calcDifference(p\_Vid->pCurImg, pic\_pix\_x+block\_x, pic\_pix\_y+block\_y, mb\_pred, block\_x, block\_y, 4, 4, diff);

cost += p\_Vid->distortion4x4 (diff16, DISTBLK\_MAX);

}

}

}

}

else

{

short diff64[64];

short \*diff;

//int i, j;

int mb\_x, mb\_y;

int pic\_pix\_y = currMB->opix\_y;

int pic\_pix\_x = currMB->pix\_x;

for(block = 0;block < 4;block++)

{

mb\_y = (block >> 1)<<3;

mb\_x = (block & 0x01)<<3;

//===== get displaced frame difference ======

diff = diff64;

calcDifference(p\_Vid->pCurImg, pic\_pix\_x+mb\_x, pic\_pix\_y+mb\_y, mb\_pred, mb\_x, mb\_y, 8, 8, diff);

cost += p\_Vid->distortion8x8 (diff64, DISTBLK\_MAX);

}

}

//cost -= ((lambda\_factor[Q\_PEL] + 4096) >> 13);

cost -= weight\_cost(lambda, 8);

return cost;

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Find motion vector for the Skip mode

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void FindSkipModeMotionVector (Macroblock \*currMB)

{

Slice \*currSlice = currMB->p\_Slice;

VideoParameters \*p\_Vid = currMB->p\_Vid;

PicMotionParams \*\*motion = p\_Vid->enc\_picture->mv\_info;

int bx, by;

MotionVector \*\*all\_mv = currSlice->all\_mv[0][0][0];

MotionVector pmv;

int zeroMotionAbove;

int zeroMotionLeft;

PixelPos mb[4];

int a\_mv\_y = 0;

int a\_ref\_idx = 0;

int b\_mv\_y = 0;

int b\_ref\_idx = 0;

get\_neighbors(currMB, mb, 0, 0, 16);

if (mb[0].available)

{

a\_mv\_y = motion[mb[0].pos\_y][mb[0].pos\_x].mv[LIST\_0].mv\_y;

a\_ref\_idx = motion[mb[0].pos\_y][mb[0].pos\_x].ref\_idx[LIST\_0];

if (currMB->mb\_field && !p\_Vid->mb\_data[mb[0].mb\_addr].mb\_field)

{

a\_mv\_y /=2;

a\_ref\_idx \*=2;

}

if (!currMB->mb\_field && p\_Vid->mb\_data[mb[0].mb\_addr].mb\_field)

{

a\_mv\_y \*= 2;

a\_ref\_idx >>=1;

}

}

if (mb[1].available)

{

b\_mv\_y = motion[mb[1].pos\_y][mb[1].pos\_x].mv[LIST\_0].mv\_y;

b\_ref\_idx = motion[mb[1].pos\_y][mb[1].pos\_x].ref\_idx[LIST\_0];

if (currMB->mb\_field && !p\_Vid->mb\_data[mb[1].mb\_addr].mb\_field)

{

b\_mv\_y /=2;

b\_ref\_idx \*=2;

}

if (!currMB->mb\_field && p\_Vid->mb\_data[mb[1].mb\_addr].mb\_field)

{

b\_mv\_y \*=2;

b\_ref\_idx >>=1;

}

}

zeroMotionLeft = !mb[0].available ? 1 : a\_ref\_idx==0 && motion[mb[0].pos\_y][mb[0].pos\_x].mv[LIST\_0].mv\_x ==0 && a\_mv\_y==0 ? 1 : 0;

zeroMotionAbove = !mb[1].available ? 1 : b\_ref\_idx==0 && motion[mb[1].pos\_y][mb[1].pos\_x].mv[LIST\_0].mv\_x ==0 && b\_mv\_y==0 ? 1 : 0;

if (zeroMotionAbove || zeroMotionLeft)

{

memset(&all\_mv [0][0], 0, 16 \* sizeof(MotionVector)); // 4 \* 4

}

else

{

currMB->GetMVPredictor (currMB, mb, &pmv, 0, motion, LIST\_0, 0, 0, 16, 16);

for (by = 0;by < 4;by++)

for (bx = 0;bx < 4;bx++)

{

all\_mv [by][bx] = pmv;

}

}

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Get cost for direct mode for an 8x8 block

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

distblk GetDirectCost8x8 (Macroblock \*currMB, int block, distblk \*cost8x8)

{

Slice \*currSlice = currMB->p\_Slice;

VideoParameters \*p\_Vid = currMB->p\_Vid;

InputParameters \*p\_Inp = currMB->p\_Inp;

DecodedPictureBuffer \*p\_Dpb = p\_Vid->p\_Dpb\_layer[p\_Vid->dpb\_layer\_id];

int pic\_pix\_y, pic\_pix\_x, i, j;

distblk cost = 0;

int mb\_y = (block >> 1)<<3;

int mb\_x = (block & 0x01)<<3;

imgpel \*\*mb\_pred = currSlice->mb\_pred[0];

int list\_mode[2] = {0, 0};

// Check if valid

for (j=(currMB->opix\_y + mb\_y) >> 2; j < (currMB->opix\_y + mb\_y + 8) >> 2; j++)

{

for (i=(currMB->pix\_x + mb\_x) >> 2; i < (currMB->pix\_x + mb\_x + 8) >> 2; i++)

{

if (currSlice->direct\_pdir[j][i] < 0)

{

\*cost8x8 = DISTBLK\_MAX;

return DISTBLK\_MAX; //mode not allowed

}

}

}

//===== Generate direct prediction =====

for (j = mb\_y; j < mb\_y + 8; j += 4)

{

pic\_pix\_y = (currMB->opix\_y + j) >> 2;

for (i = mb\_x; i < mb\_x + 8; i += 4)

{

pic\_pix\_x = (currMB->pix\_x + i) >> 2;

p\_Dpb->pf\_luma\_prediction (currMB, i, j, 4, 4, currSlice->direct\_pdir[pic\_pix\_y][pic\_pix\_x],

list\_mode, currSlice->direct\_ref\_idx[pic\_pix\_y][pic\_pix\_x], 0);

}

}

if(p\_Inp->Transform8x8Mode)

{

short diff16[4][16];

short diff64[64];

short \*tmp64 = diff64;

short \*tmp16[4]; //{diff16[0], diff16[1], diff16[2], diff16[3]};

int index;

tmp16[0] = diff16[0];

tmp16[1] = diff16[1];

tmp16[2] = diff16[2];

tmp16[3] = diff16[3];

pic\_pix\_y = currMB->opix\_y;

pic\_pix\_x = currMB->pix\_x;

//===== get displaced frame difference ======

//p\_Dpb->pf\_calcDifference(p\_Vid->pCurImg, pic\_pix\_x+mb\_x, pic\_pix\_y+mb\_y, mb\_pred, mb\_x, mb\_y, 8, 8, tmp64);

for (j = mb\_y; j < 8 + mb\_y; j++)

{

for (i = mb\_x; i < 8 + mb\_x; i++)

{

index = 2 \* ((j - mb\_y)> 3) + ((i - mb\_x)> 3);

\*tmp64++ = \*(tmp16[index])++ = (short) (p\_Vid->pCurImg[pic\_pix\_y + j][pic\_pix\_x + i] - mb\_pred[j][i]);

}

}

cost += p\_Vid->distortion4x4 (diff16[0], DISTBLK\_MAX);

cost += p\_Vid->distortion4x4 (diff16[1], DISTBLK\_MAX);

cost += p\_Vid->distortion4x4 (diff16[2], DISTBLK\_MAX);

cost += p\_Vid->distortion4x4 (diff16[3], DISTBLK\_MAX);

\*cost8x8 += p\_Vid->distortion8x8 (diff64, DISTBLK\_MAX);

}

else

{

int block\_y, block\_x;

short diff16[16];

short \*diff;

for (block\_y=mb\_y; block\_y < mb\_y + 8; block\_y += 4)

{

pic\_pix\_y = currMB->opix\_y + block\_y;

for (block\_x=mb\_x; block\_x<mb\_x+8; block\_x+=4)

{

pic\_pix\_x = currMB->pix\_x + block\_x;

diff = diff16;

//===== get displaced frame difference ======

calcDifference(p\_Vid->pCurImg, pic\_pix\_x, pic\_pix\_y, mb\_pred, block\_x, block\_y, 4, 4, diff);

cost += p\_Vid->distortion4x4 (diff16, DISTBLK\_MAX);

}

}

}

return cost;

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Get cost for direct mode for an macroblock

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

distblk GetDirectCostMB (Macroblock \*currMB)

{

Slice \*currSlice = currMB->p\_Slice;

InputParameters \*p\_Inp = currSlice->p\_Inp;

int i;

distblk cost = 0;

distblk cost8x8 = 0;

int bslice = currSlice->slice\_type == B\_SLICE;

#if (MVC\_EXTENSION\_ENABLE)

int \*InterSearch = p\_Inp->InterSearch[(currSlice->p\_Vid->num\_of\_layers > 1) ? currSlice->view\_id : 0][bslice];

#else

int \*InterSearch = p\_Inp->InterSearch[0][bslice];

#endif

for (i=0; i<4; i++)

{

cost += GetDirectCost8x8 (currMB, i, &cost8x8);

if (cost8x8 == DISTBLK\_MAX) return DISTBLK\_MAX;

}

switch(p\_Inp->Transform8x8Mode)

{

case 1: // Mixture of 8x8 & 4x4 transform

if((cost8x8 < cost)||

!(InterSearch[5] &&

InterSearch[6] &&

InterSearch[7])

)

{

cost = cost8x8; //return 8x8 cost

}

break;

case 2: // 8x8 Transform only

cost = cost8x8;

break;

default: // 4x4 Transform only

break;

}

return cost;

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Motion search for a macroblock partition

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void PartitionMotionSearch (Macroblock \*currMB,

int blocktype,

int block8x8,

int \*lambda\_factor)

{

VideoParameters \*p\_Vid = currMB->p\_Vid;

Slice \*currSlice = currMB->p\_Slice;

#if GET\_METIME

TIME\_T me\_time\_start;

TIME\_T me\_time\_end;

int64 me\_tmp\_time;

gettime( &me\_time\_start ); // start time ms

#endif

if (currSlice->rdoq\_motion\_copy == 1)

{

PicMotionParams \*\*motion = p\_Vid->enc\_picture->mv\_info;

short by = by0[blocktype][block8x8];

short bx = bx0[blocktype][block8x8];

short step\_h = (part\_size[blocktype][0]);

short step\_v = (part\_size[blocktype][1]);

short pic\_block\_y = currMB->block\_y + by;

short pic\_block\_x = currMB->block\_x + bx;

int list\_offset = currMB->list\_offset;

int numlists = (currSlice->slice\_type == B\_SLICE) ? 2 : 1;

distblk \*m\_cost;

short list = LIST\_0;

short ref = 0;

//===== LOOP OVER REFERENCE FRAMES =====

for (list = 0; list < numlists; list++)

{

for (ref=0; ref < currSlice->listXsize[list+list\_offset]; ref++)

{

m\_cost = &p\_Vid->motion\_cost[blocktype][list][ref][block8x8];

//===== LOOP OVER SUB MACRO BLOCK partitions

updateMV\_mp(currMB, m\_cost, ref, list, bx, by, blocktype, block8x8);

set\_me\_parameters(motion, &currSlice->all\_mv[list][ref][blocktype][by][bx], list, (char) ref, step\_h, step\_v, pic\_block\_y, pic\_block\_x);

}

}

}

else

{

InputParameters \*p\_Inp = currMB->p\_Inp;

short by = by0[blocktype][block8x8];

short bx = bx0[blocktype][block8x8];

short step\_h = (part\_size[blocktype][0]);

short step\_v = (part\_size[blocktype][1]);

short pic\_block\_y = currMB->block\_y + by;

short pic\_block\_x = currMB->block\_x + bx;

int list\_offset = currMB->list\_offset;

int numlists = (currSlice->slice\_type == B\_SLICE) ? 2 : 1;

short list = LIST\_0;

short ref = 0;

MEBlock mv\_block;

distblk \*m\_cost;

PicMotionParams \*\*motion = p\_Vid->enc\_picture->mv\_info;

// Set flag for 8x8 Hadamard consideration for SATD (only used when 8x8 integer transform is used for encoding)

mv\_block.test8x8 = p\_Inp->Transform8x8Mode;

init\_mv\_block(currMB, &mv\_block, (short) blocktype, list, (char) ref, bx, by);

if (p\_Inp->SearchMode[p\_Vid->view\_id] == EPZS)

{

if (p\_Inp->EPZSSubPelGrid)

currMB->IntPelME = EPZS\_integer\_motion\_estimation;

else

currMB->IntPelME = EPZS\_motion\_estimation;

}

get\_original\_block(p\_Vid, &mv\_block);

//--- motion search for block ---

{

//===== LOOP OVER REFERENCE FRAMES =====

for (list = 0; list < numlists; list++)

{

//----- set arrays -----

mv\_block.list = (char) list;

for (ref=0; ref < currSlice->listXsize[list+list\_offset]; ref++)

{

mv\_block.ref\_idx = (char) ref;

m\_cost = &p\_Vid->motion\_cost[blocktype][list][ref][block8x8];

{

//----- set search range ---

get\_search\_range(&mv\_block, p\_Inp, ref, blocktype);

//===== LOOP OVER MACROBLOCK partitions

\*m\_cost = BlockMotionSearch (currMB, &mv\_block, bx<<2, by<<2, lambda\_factor);

}

//--- set motion vectors and reference frame ---

set\_me\_parameters(motion, &currSlice->all\_mv[list][ref][blocktype][by][bx], list, (char) ref, step\_h, step\_v, pic\_block\_y, pic\_block\_x);

}

}

}

free\_mv\_block(&mv\_block);

}

#if GET\_METIME

gettime(&me\_time\_end); // end time ms

me\_tmp\_time = timediff (&me\_time\_start, &me\_time\_end);

p\_Vid->me\_tot\_time += me\_tmp\_time;

p\_Vid->me\_time += me\_tmp\_time;

#endif

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Motion search for a submacroblock partition

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void SubPartitionMotionSearch (Macroblock \*currMB,

int blocktype,

int block8x8,

int \*lambda\_factor)

{

Slice \*currSlice = currMB->p\_Slice;

VideoParameters \*p\_Vid = currMB->p\_Vid;

#if GET\_METIME

TIME\_T me\_time\_start;

TIME\_T me\_time\_end;

int64 me\_tmp\_time;

gettime( &me\_time\_start ); // start time ms

#endif

if (currSlice->rdoq\_motion\_copy == 1)

{

int parttype = 4;

PicMotionParams \*\*motion = p\_Vid->enc\_picture->mv\_info;

short by = by0[parttype][block8x8];

short bx = bx0[parttype][block8x8];

short step\_h = (part\_size[blocktype][0]);

short step\_v = (part\_size[blocktype][1]);

int list\_offset = currMB->list\_offset;

int numlists = (currSlice->slice\_type == B\_SLICE) ? 2 : 1;

distblk \*m\_cost;

MotionVector \*all\_mv;

short list = LIST\_0;

short ref = 0;

short step\_h0 = (part\_size[ parttype][0]);

short step\_v0 = (part\_size[ parttype][1]);

int v, h;

int pic\_block\_y;

//===== LOOP OVER REFERENCE FRAMES =====

for (list = 0; list < numlists; list++)

{

for (ref=0; ref < currSlice->listXsize[list+list\_offset]; ref++)

{

m\_cost = &p\_Vid->motion\_cost[blocktype][list][ref][block8x8];

//===== LOOP OVER SUB MACRO BLOCK partitions

for (v=by; v<by + step\_v0; v += step\_v)

{

pic\_block\_y = currMB->block\_y + v;

for (h=bx; h<bx+step\_h0; h+=step\_h)

{

all\_mv = &currSlice->all\_mv[list][ref][blocktype][v][h];

updateMV\_mp(currMB, m\_cost, ref, list, h, v, blocktype, block8x8);

//--- set motion vectors and reference frame (for motion vector prediction) ---

set\_me\_parameters(motion, all\_mv, list, (char) ref, step\_h, step\_v, pic\_block\_y, currMB->block\_x + h);

} // h

} // v

}

}

}

else

{

InputParameters \*p\_Inp = currMB->p\_Inp;

PicMotionParams \*\*motion = p\_Vid->enc\_picture->mv\_info;

int parttype = 4;

short by = by0[parttype][block8x8];

short bx = bx0[parttype][block8x8];

short step\_h0 = (part\_size[ parttype][0]);

short step\_v0 = (part\_size[ parttype][1]);

short step\_h = (part\_size[blocktype][0]);

short step\_v = (part\_size[blocktype][1]);

int list\_offset = currMB->list\_offset;

int numlists = (currSlice->slice\_type == B\_SLICE) ? 2 : 1;

MotionVector \*all\_mv;

short list = LIST\_0;

short ref = 0;

MEBlock mv\_block;

distblk \*m\_cost;

distblk mcost;

int v, h;

int pic\_block\_y;

// Set if 8x8 transform will be used if SATD is used

mv\_block.test8x8 = p\_Inp->Transform8x8Mode && blocktype == 4;

if (p\_Inp->SearchMode[p\_Vid->view\_id] == EPZS)

{

if (p\_Inp->EPZSSubPelGrid)

{

if (blocktype > 4)

currMB->IntPelME = EPZS\_integer\_subMB\_motion\_estimation;

else

currMB->IntPelME = EPZS\_integer\_motion\_estimation;

}

else

{

if (blocktype > 4)

currMB->IntPelME = EPZS\_subMB\_motion\_estimation;

else

currMB->IntPelME = EPZS\_motion\_estimation;

}

}

init\_mv\_block(currMB, &mv\_block, (short) blocktype, list, (char) ref, bx, by);

if (blocktype == 4)

get\_original\_block(p\_Vid, &mv\_block);

//===== LOOP OVER REFERENCE FRAMES =====

for (list=0; list<numlists;list++)

{

mv\_block.list = (char) list;

for (ref=0; ref < currSlice->listXsize[list+list\_offset]; ref++)

{

mv\_block.ref\_idx = (char) ref;

m\_cost = &p\_Vid->motion\_cost[blocktype][list][ref][block8x8];

//----- set search range ---

get\_search\_range(&mv\_block, p\_Inp, ref, blocktype);

//----- init motion cost -----

\*m\_cost = 0;

//===== LOOP OVER SUB MACRO BLOCK partitions

for (v=by; v<by + step\_v0; v += step\_v)

{

pic\_block\_y = currMB->block\_y + v;

for (h=bx; h<bx+step\_h0; h+=step\_h)

{

all\_mv = &currSlice->all\_mv[list][ref][blocktype][v][h];

//--- motion search for block ---

update\_mv\_block(currMB, &mv\_block, h, v);

{

//----- set search range ---

get\_search\_range(&mv\_block, p\_Inp, ref, blocktype);

mcost = BlockMotionSearch (currMB, &mv\_block, h<<2, v<<2, lambda\_factor);

\*m\_cost += mcost;

}

//--- set motion vectors and reference frame (for motion vector prediction) ---

set\_me\_parameters(motion, all\_mv, list, (char) ref, step\_h, step\_v, pic\_block\_y, currMB->block\_x + h);

}

}

if ((p\_Inp->Transform8x8Mode == 1) && p\_Inp->RDOQ\_CP\_MV && (blocktype == 4))

{

if (currMB->luma\_transform\_size\_8x8\_flag)

{

currSlice->tmp\_mv8[list][ref][by][bx] = currSlice->all\_mv[list][ref][blocktype][by][bx];

currSlice->motion\_cost8[list][ref][block8x8] = \*m\_cost;

}

else

{

currSlice->tmp\_mv4[list][ref][by][bx] = currSlice->all\_mv[list][ref][blocktype][by][bx];

currSlice->motion\_cost4[list][ref][block8x8] = \*m\_cost;

}

}

}

}

free\_mv\_block(&mv\_block);

}

#if GET\_METIME

gettime(&me\_time\_end); // end time ms

me\_tmp\_time = timediff (&me\_time\_start, &me\_time\_end);

p\_Vid->me\_tot\_time += me\_tmp\_time;

p\_Vid->me\_time += me\_tmp\_time;

#endif

}

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \file md\_high.c

\*

\* \brief

\* Main macroblock mode decision functions and helpers

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#include <math.h>

#include <limits.h>

#include <float.h>

#include "global.h"

#include "rdopt\_coding\_state.h"

#include "intrarefresh.h"

#include "image.h"

#include "ratectl.h"

#include "mode\_decision.h"

#include "mode\_decision\_p8x8.h"

#include "fmo.h"

#include "me\_umhex.h"

#include "me\_umhexsmp.h"

#include "macroblock.h"

#include "md\_common.h"

#include "conformance.h"

#include "vlc.h"

#include "rdopt.h"

#include "mv\_search.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "me\_fullfast.h"

extern int block\_total\_number = 0;

extern long long frame\_total\_SAD = 0;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \brief

\* Mode Decision for a macroblock

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void encode\_one\_macroblock\_high (Macroblock \*currMB)

{

Slice \*currSlice = currMB->p\_Slice;

VideoParameters \*p\_Vid = currMB->p\_Vid;

InputParameters \*p\_Inp = currMB->p\_Inp;

PicMotionParams \*\*motion = p\_Vid->enc\_picture->mv\_info;

RDOPTStructure \*p\_RDO = currSlice->p\_RDO;

int max\_index = 9;

int block, index, mode, i, j;

RD\_PARAMS enc\_mb;

distblk bmcost[5] = {DISTBLK\_MAX};

distblk cost=0;

distblk min\_cost = DISTBLK\_MAX;

int intra1 = 0;

int mb\_available[3];

short bslice = (short) (currSlice->slice\_type == B\_SLICE);

short pslice = (short) ((currSlice->slice\_type == P\_SLICE) || (currSlice->slice\_type == SP\_SLICE));

short intra = (short) ((currSlice->slice\_type == I\_SLICE) || (currSlice->slice\_type == SI\_SLICE) || (pslice && currMB->mb\_y == p\_Vid->mb\_y\_upd && p\_Vid->mb\_y\_upd != p\_Vid->mb\_y\_intra));

int lambda\_mf[3];

imgpel \*\*mb\_pred = currSlice->mb\_pred[0];

Block8x8Info \*b8x8info = p\_Vid->b8x8info;

char chroma\_pred\_mode\_range[2];

short inter\_skip = 0;

BestMode md\_best;

Info8x8 best;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int current\_block\_SAD = 0;

int skip\_all\_intra\_mode = 0; // if it = 1, skip all intra modes, else do intra modes

int total\_block\_number = p\_Vid->p\_Inp->source.width[0]/16 \* p\_Vid->p\_Inp->source.height[0]/16;

int me\_choice = 0; // if me\_choice = 1, do ME only for 16x16, 16x8, 8x16 blocks, else do ME for all blocks \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

init\_md\_best(&md\_best);

// Init best (need to create simple function)

best.pdir = 0;

best.bipred = 0;

best.ref[LIST\_0] = 0;

best.ref[LIST\_1] = -1;

intra |= RandomIntra (p\_Vid, currMB->mbAddrX); // Forced Pseudo-Random Intra

//===== Setup Macroblock encoding parameters =====

init\_enc\_mb\_params(currMB, &enc\_mb, intra);

if (p\_Inp->AdaptiveRounding)

{

reset\_adaptive\_rounding(p\_Vid);

}

if (currSlice->mb\_aff\_frame\_flag)

{

reset\_mb\_nz\_coeff(p\_Vid, currMB->mbAddrX);

}

//===== S T O R E C O D I N G S T A T E =====

//---------------------------------------------------

currSlice->store\_coding\_state (currMB, currSlice->p\_RDO->cs\_cm);

if (!intra)

{

//===== set skip/direct motion vectors =====

if (enc\_mb.valid[0])

{

if (bslice)

currSlice->Get\_Direct\_Motion\_Vectors (currMB);

else

FindSkipModeMotionVector (currMB);

}

if (p\_Inp->CtxAdptLagrangeMult == 1)

{

get\_initial\_mb16x16\_cost(currMB);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

current\_block\_SAD = p\_Vid->p\_ffast\_me->BlockSAD[0][0][1][0][0];

// compute w value for different QP

double w\_value;

switch (p\_Vid->p\_Inp->qp[0]) {

case 28:

w\_value = 1.2;

break;

case 32:

w\_value = 1.0;

break;

case 36:

w\_value = 0.8;

break;

case 40:

w\_value = 0.6;

break;

default:

w\_value = 1.0;

break;

}

// printf("num of macroblocks in a silce = %d\n",currSlice->num\_mb);

// printf("SAD of current Mb = %d\n",current\_block\_SAD);

// printf("block\_total\_num = %d\n",block\_total\_number);

frame\_total\_SAD += current\_block\_SAD;

// printf("frame\_total SAD = %lld\n",frame\_total\_SAD);

if (block\_total\_number != 0) {

if (frame\_total\_SAD/block\_total\_number >= w\_value\* current\_block\_SAD) {

me\_choice = 1;

}else{

me\_choice = 0;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//===== MOTION ESTIMATION FOR 16x16, 16x8, 8x16 BLOCKS =====

for (mode = 1; mode < 4; mode++)

{

best.mode = (char) mode;

best.bipred = 0;

b8x8info->best[mode][0].bipred = 0;

if (enc\_mb.valid[mode])

{

for (cost=0, block=0; block<(mode==1?1:2); block++)

{

update\_lambda\_costs(currMB, &enc\_mb, lambda\_mf);

PartitionMotionSearch (currMB, mode, block, lambda\_mf);

//--- set 4x4 block indices (for getting MV) ---

j = (block==1 && mode==2 ? 2 : 0);

i = (block==1 && mode==3 ? 2 : 0);

//--- get cost and reference frame for List 0 prediction ---

bmcost[LIST\_0] = DISTBLK\_MAX;

list\_prediction\_cost(currMB, LIST\_0, block, mode, &enc\_mb, bmcost, best.ref);

if (bslice)

{

//--- get cost and reference frame for List 1 prediction ---

bmcost[LIST\_1] = DISTBLK\_MAX;

list\_prediction\_cost(currMB, LIST\_1, block, mode, &enc\_mb, bmcost, best.ref);

// Compute bipredictive cost between best list 0 and best list 1 references

list\_prediction\_cost(currMB, BI\_PRED, block, mode, &enc\_mb, bmcost, best.ref);

// currently Bi predictive ME is only supported for modes 1, 2, 3 and ref 0

if (is\_bipred\_enabled(p\_Vid, mode))

{

get\_bipred\_cost(currMB, mode, block, i, j, &best, &enc\_mb, bmcost);

}

else

{

bmcost[BI\_PRED\_L0] = DISTBLK\_MAX;

bmcost[BI\_PRED\_L1] = DISTBLK\_MAX;

}

// Determine prediction list based on mode cost

determine\_prediction\_list(bmcost, &best, &cost);

}

else // if (bslice)

{

best.pdir = 0;

cost += bmcost[LIST\_0];

}

assign\_enc\_picture\_params(currMB, mode, &best, 2 \* block);

//----- set reference frame and direction parameters -----

set\_block8x8\_info(b8x8info, mode, block, &best);

//--- set reference frames and motion vectors ---

if (mode>1 && block == 0)

currSlice->set\_ref\_and\_motion\_vectors (currMB, motion, &best, block);

} // for (block=0; block<(mode==1?1:2); block++)

if (cost < min\_cost)

{

md\_best.mode = (byte) mode;

md\_best.cost = cost;

currMB->best\_mode = (short) mode;

min\_cost = cost;

if (p\_Inp->CtxAdptLagrangeMult == 1)

{

adjust\_mb16x16\_cost(currMB, cost);

}

}

} // if (enc\_mb.valid[mode])

} // for (mode=1; mode<4; mode++)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (me\_choice == 0)

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (enc\_mb.valid[P8x8])

{

currMB->valid\_8x8 = FALSE;

if (p\_Inp->Transform8x8Mode)

{

ResetRD8x8Data(p\_Vid, p\_RDO->tr8x8);

currMB->luma\_transform\_size\_8x8\_flag = TRUE; //switch to 8x8 transform size

//===========================================================

// Check 8x8 partition with transform size 8x8

//===========================================================

//===== LOOP OVER 8x8 SUB-PARTITIONS (Motion Estimation & Mode Decision) =====

for (block = 0; block < 4; block++)

{

currSlice->submacroblock\_mode\_decision(currMB, &enc\_mb, p\_RDO->tr8x8, p\_RDO->cofAC8x8ts[block], block, &cost);

if(!currMB->valid\_8x8)

break;

set\_subblock8x8\_info(b8x8info, P8x8, block, p\_RDO->tr8x8);

}

}// if (p\_Inp->Transform8x8Mode)

currMB->valid\_4x4 = FALSE;

if (p\_Inp->Transform8x8Mode != 2)

{

currMB->luma\_transform\_size\_8x8\_flag = FALSE; //switch to 8x8 transform size

ResetRD8x8Data(p\_Vid, p\_RDO->tr4x4);

//=================================================================

// Check 8x8, 8x4, 4x8 and 4x4 partitions with transform size 4x4

//=================================================================

//===== LOOP OVER 8x8 SUB-PARTITIONS (Motion Estimation & Mode Decision) =====

for (block = 0; block < 4; block++)

{

currSlice->submacroblock\_mode\_decision(currMB, &enc\_mb, p\_RDO->tr4x4, p\_RDO->coefAC8x8[block], block, &cost);

if(!currMB->valid\_4x4)

break;

set\_subblock8x8\_info(b8x8info, P8x8, block, p\_RDO->tr4x4);

}

}// if (p\_Inp->Transform8x8Mode != 2)

if (p\_Inp->RCEnable)

rc\_store\_diff(currSlice->diffy, &p\_Vid->pCurImg[currMB->opix\_y], currMB->pix\_x, mb\_pred);

p\_Vid->giRDOpt\_B8OnlyFlag = FALSE;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

else // if (!intra)

{

min\_cost = DISTBLK\_MAX;

}

// Set Chroma mode

set\_chroma\_pred\_mode(currMB, enc\_mb, mb\_available, chroma\_pred\_mode\_range);

//========= C H O O S E B E S T M A C R O B L O C K M O D E =========

//-------------------------------------------------------------------------

// printf("curSAD = %d\n",p\_Vid->p\_ffast\_me->BlockSAD[0][0][1][0][0]);

for (currMB->c\_ipred\_mode = chroma\_pred\_mode\_range[0]; currMB->c\_ipred\_mode<=chroma\_pred\_mode\_range[1]; currMB->c\_ipred\_mode++)

{

// bypass if c\_ipred\_mode is not allowed

if ( (p\_Vid->yuv\_format != YUV400) &&

( ((!intra || !p\_Inp->IntraDisableInterOnly) && p\_Inp->ChromaIntraDisable == 1 && currMB->c\_ipred\_mode!=DC\_PRED\_8)

|| (currMB->c\_ipred\_mode == VERT\_PRED\_8 && !mb\_available[0])

|| (currMB->c\_ipred\_mode == HOR\_PRED\_8 && !mb\_available[1])

|| (currMB->c\_ipred\_mode == PLANE\_8 && (!mb\_available[1] || !mb\_available[0] || !mb\_available[2]))))

continue;

//===== GET BEST MACROBLOCK MODE =====

for (index=0; index < max\_index; index++)

{

mode = mb\_mode\_table[index];

//printf("mode %d rdcost = %7.3f\n", mode, (double) currMB->min\_rdcost);

if (enc\_mb.valid[mode])

{

//printf(" mode %d is valid\n", mode);

if (p\_Vid->yuv\_format != YUV400)

{

currMB->i16mode = 0;

}

// Skip intra modes in inter slices if best mode is inter <P8x8 with cbp equal to 0

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (!intra & (current\_block\_SAD < currMB->min\_rdcost)) {

skip\_all\_intra\_mode = 1;

}else{

skip\_all\_intra\_mode = 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (currSlice->P444\_joined)

{

if (p\_Inp->SkipIntraInInterSlices && !intra && mode >= I16MB

&& currMB->best\_mode <=3 && currMB->best\_cbp == 0 && currSlice->cmp\_cbp[1] == 0 && currSlice->cmp\_cbp[2] == 0 && (currMB->min\_rdcost < weighted\_cost(enc\_mb.lambda\_mdfp,5)))

continue;

}

else

{

// if (p\_Inp->SkipIntraInInterSlices | skip\_all\_intra\_mode)

if (p\_Inp->SkipIntraInInterSlices)

{

if (!intra && mode >= I4MB)

{

if (currMB->best\_mode <=3 && currMB->best\_cbp == 0 && (currMB->min\_rdcost < weighted\_cost(enc\_mb.lambda\_mdfp, 5)))

{

continue;

}

else if (currMB->best\_mode == 0 && (currMB->min\_rdcost < weighted\_cost(enc\_mb.lambda\_mdfp,6)))

{

continue;

}

}

}

}

compute\_mode\_RD\_cost(currMB, &enc\_mb, (short) mode, &inter\_skip);

}

//printf(" best %d %7.2f\n", currMB->best\_mode, (double) currMB->min\_rdcost);

}// for (index=0; index<max\_index; index++)

}// for (currMB->c\_ipred\_mode=DC\_PRED\_8; currMB->c\_ipred\_mode<=chroma\_pred\_mode\_range[1]; currMB->c\_ipred\_mode++)

restore\_nz\_coeff(currMB);

intra1 = is\_intra(currMB);

//===== S E T F I N A L M A C R O B L O C K P A R A M E T E R S ======

//---------------------------------------------------------------------------

update\_qp\_cbp\_tmp(currMB, p\_RDO->cbp);

currSlice->set\_stored\_mb\_parameters (currMB);

// Rate control

if(p\_Inp->RCEnable && p\_Inp->RCUpdateMode <= MAX\_RC\_MODE)

rc\_store\_mad(currMB);

//===== Decide if this MB will restrict the reference frames =====

if (p\_Inp->RestrictRef)

update\_refresh\_map(currMB, intra, intra1);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

block\_total\_number ++;

if (block\_total\_number == total\_block\_number) {

block\_total\_number = 0;

frame\_total\_SAD = 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}