Inter Prediction

2016.12.07

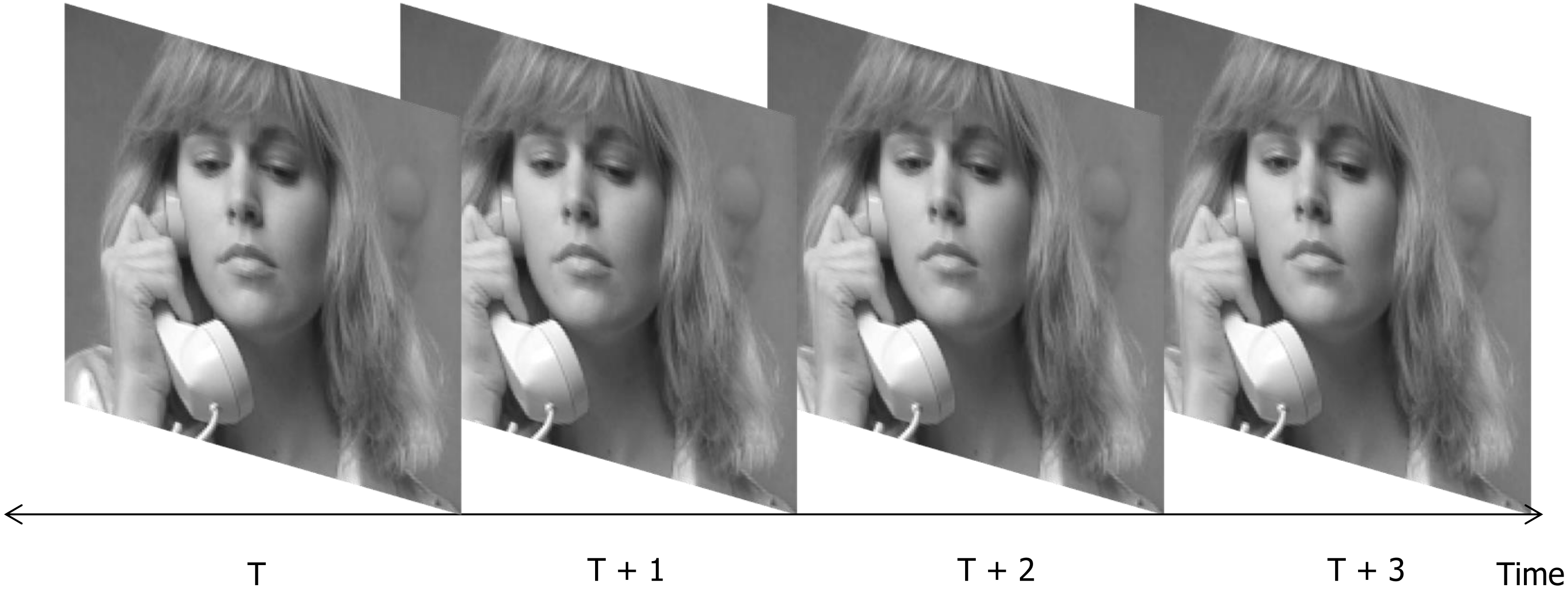
Dongkyu Lee (dongkyu@media.kw.ac.kr)
Jongseok Lee (suk2080@media.kw.ac.kr)
Kangseok Choi (smail91@media.kw.ac.kr)

Multimedia LAB

VIA-Multimedia Center, Kwangwoon University

Video format

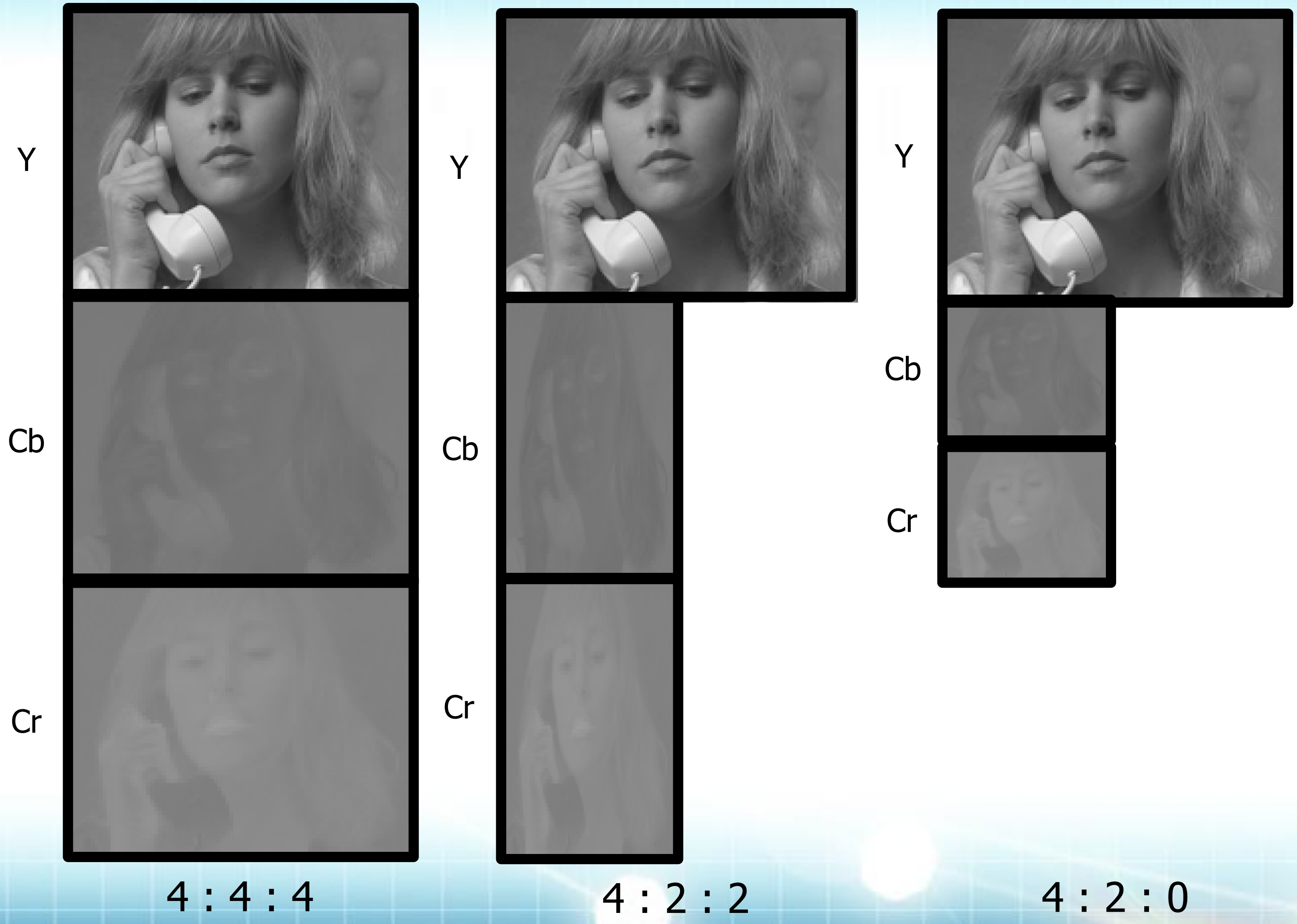
- Basic concept of the video file



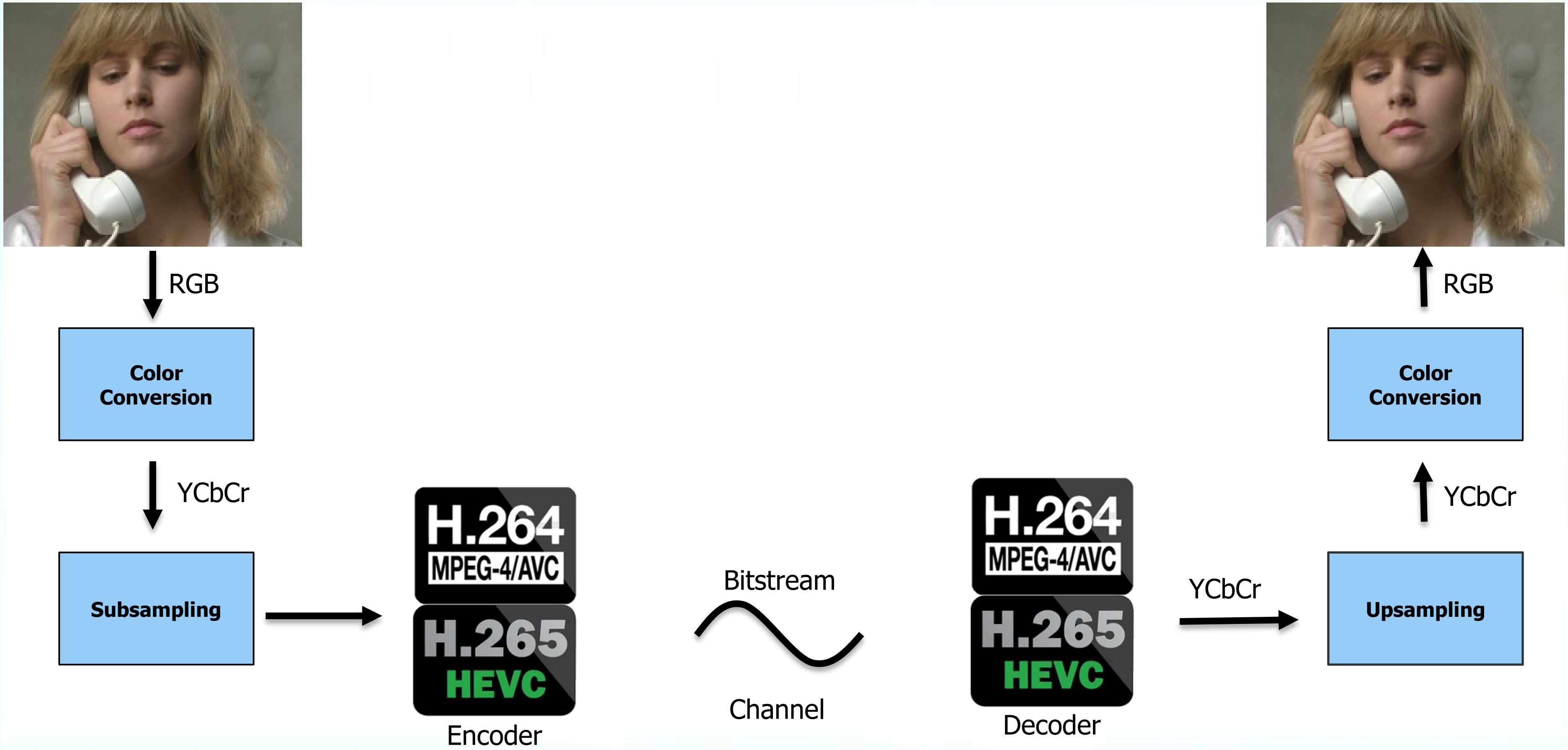
RGB to YUV(YCbCr)

- RGB to YUV (integer)
  - $Y' = \left((66 \times R + 129 \times G + 25 \times B + 128) \gg 8\right) + 16$
  - $U = \left((-38 \times R - 74 \times G + 112 \times B + 128) \gg 8\right) + 128$
  - $V = \left((112 \times R - 94 \times G - 18 \times B + 128) \gg 8\right) + 128$
- YUV to RGB (integer)
  - $C = Y' - 16$
  - $D = U - 128$
  - $E = V - 128$
  - $R = clamp\left((298 \times C + 409 \times E + 128) \gg 8\right)$
  - $G = clamp\left((298 \times C - 100 \times D - 208 \times E + 128) \gg 8\right)$
  - $B = clamp\left((298 \times C + 516 \times D + 128) \gg 8\right)$

Subsampling



CODEC



RGB

Color Conversion

YCbCr

Subsampling

Encoder

Bitstream

Channel

Decoder

YCbCr

Upsampling

YCbCr

Color Conversion

RGB

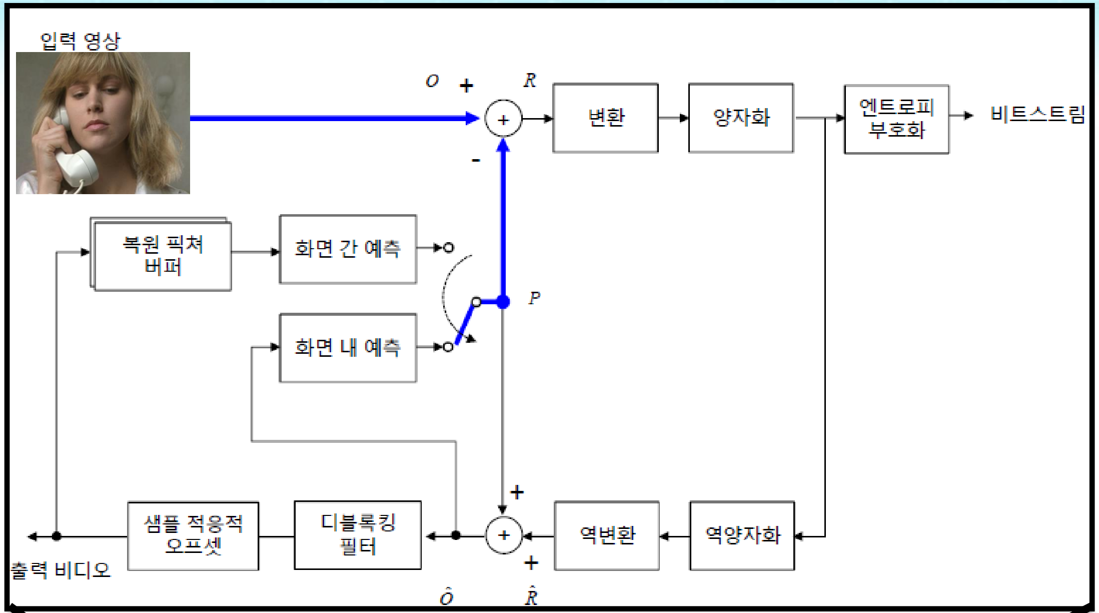
CODEC

RGB

Color Conversion

YCbCr

Subsampling



H.264 MPEG-4/AVC

H.265 HEVC

Encoder

Bitstream

Channel

H.264 MPEG-4/AVC

H.265 HEVC

Decoder

YCbCr

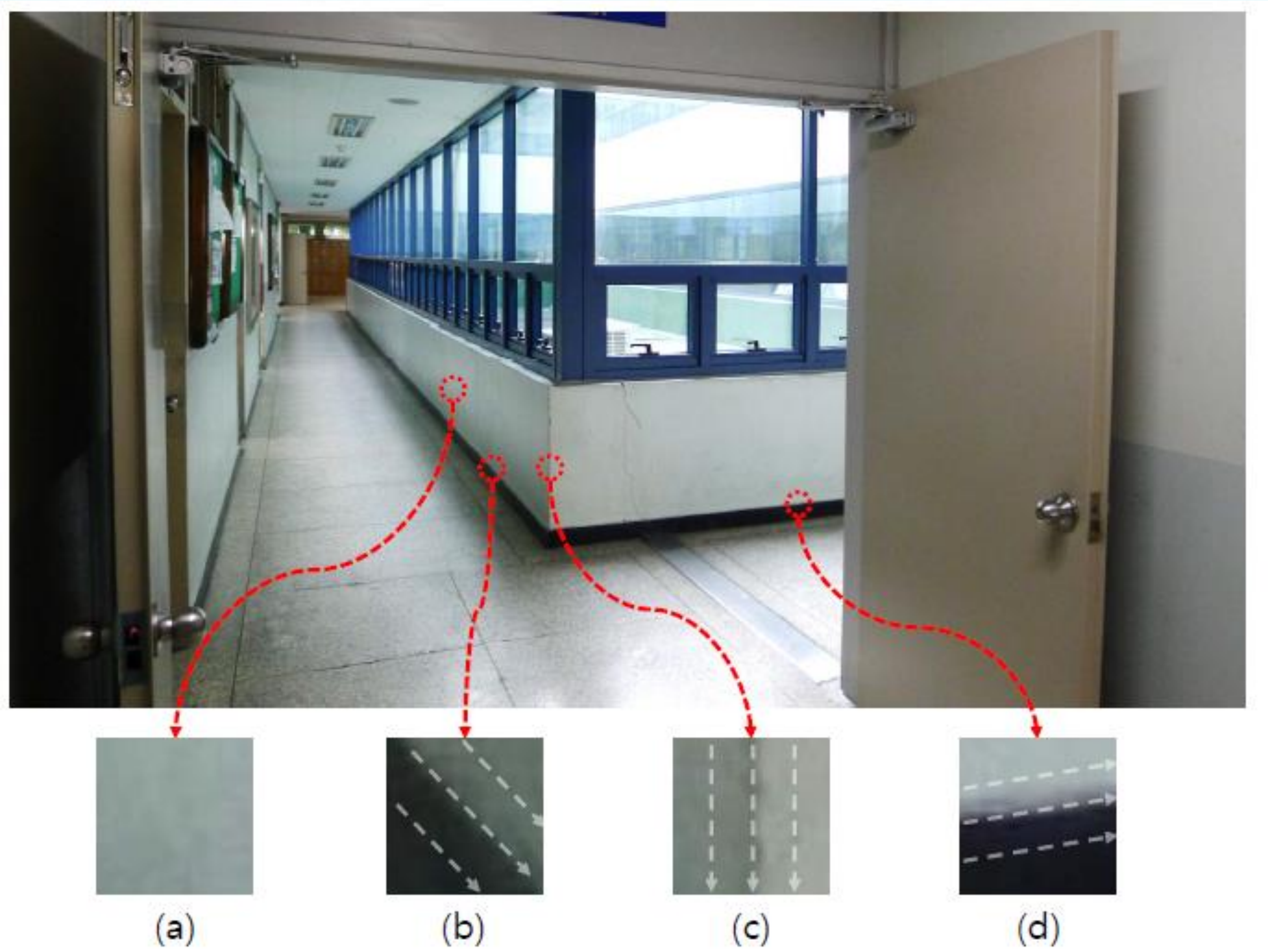
Upsampling

YCbCr

Color Conversion

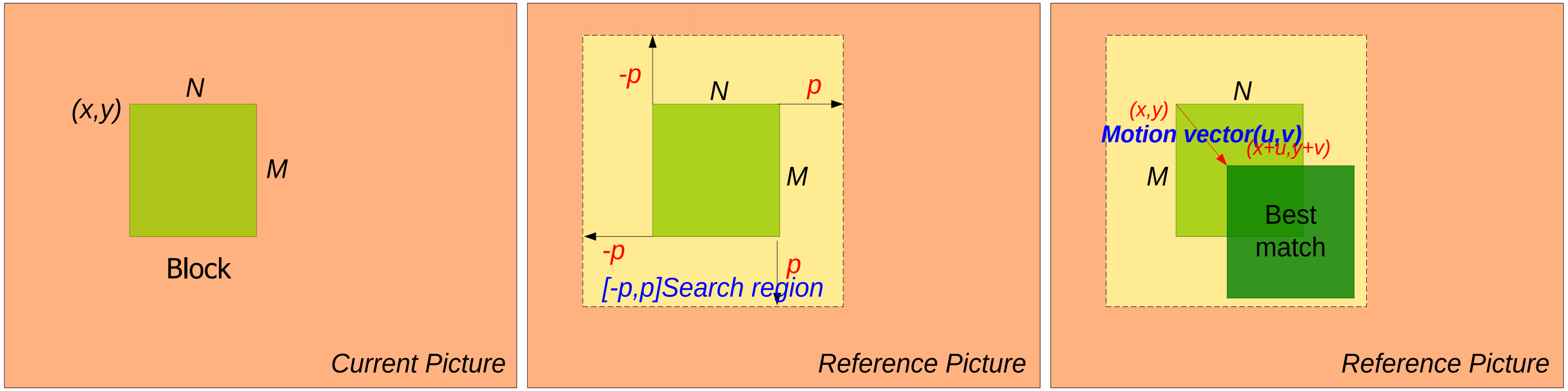
RGB

Intra Prediction

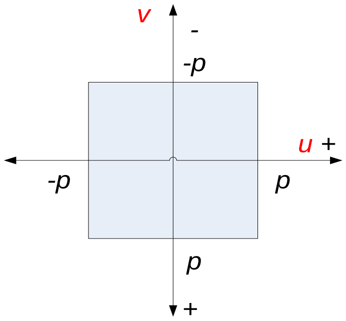


Motion-Compensated Prediction

❖ Motion estimation



$$\text{where}\begin{cases} N \times M \ : \text{block size} \\ (x, y) : \text{location of the block region(left-top corner)} \\ [-p, p] : \text{search region around the macroblock in the current picture} \\ \text{Motion vector : a vector from } (x, y) \text{ to } (x+u, y+v) \end{cases}$$



Motion-Compensated Prediction

- The matching criterion
  - Mean absolute error(MAE)

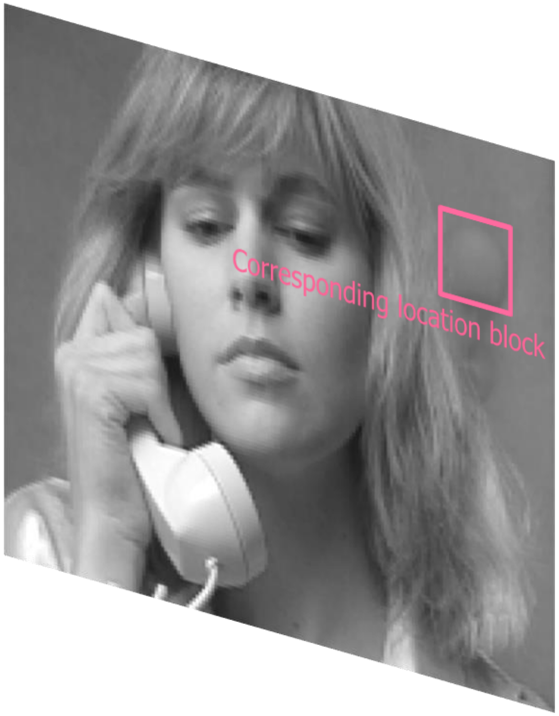
$$MAE(i,\ j) = \frac{1}{MN}\sum_{k=0}^{M-1}\sum_{l=0}^{N-1} |\,C(x+k,\ y+l) - R(x+i+k,\ y+j+l)\,|$$

$$\text{where}\begin{cases} C(x+k,\ y+l) \text{ : pixels of the block in the current frame} \\ R(x+i+k,\ y+j+l) \text{ : pixels in the reference picture} \\ -p \le i,\ j \le p \end{cases}$$

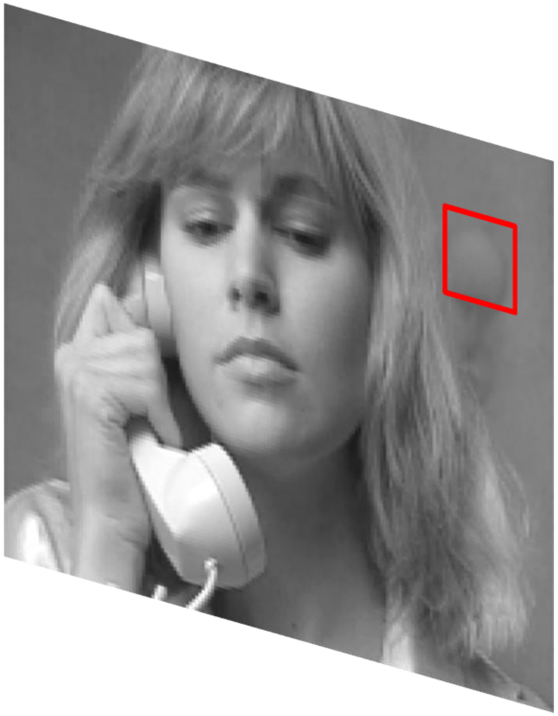
  - Best matching block
    - Block $R(x + i, y + j)$ for which $MAE(i, j)$ is minimized
  - Motion vector
    - The coordinates (i, j) for which MAE is minimized

Motion-Compensated Prediction

- Example



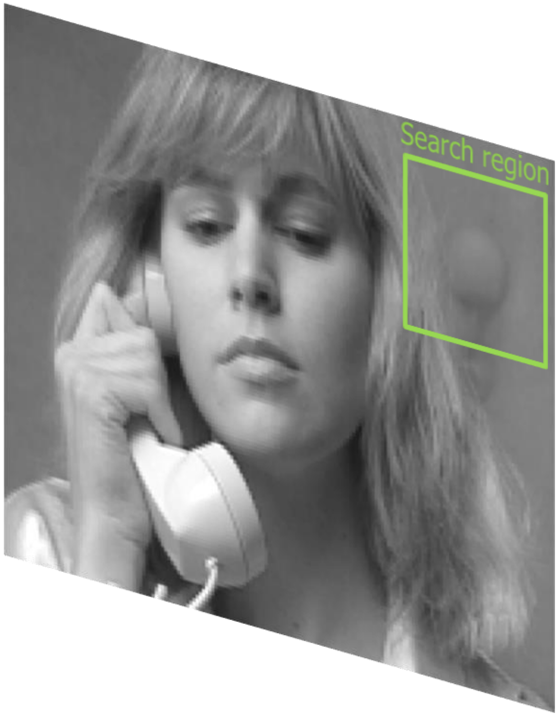
Previous frame



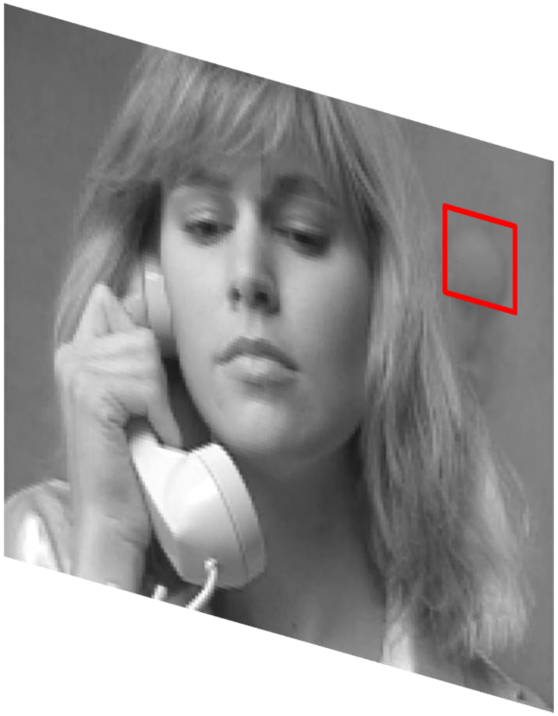
Current frame

Motion-Compensated Prediction

- Example



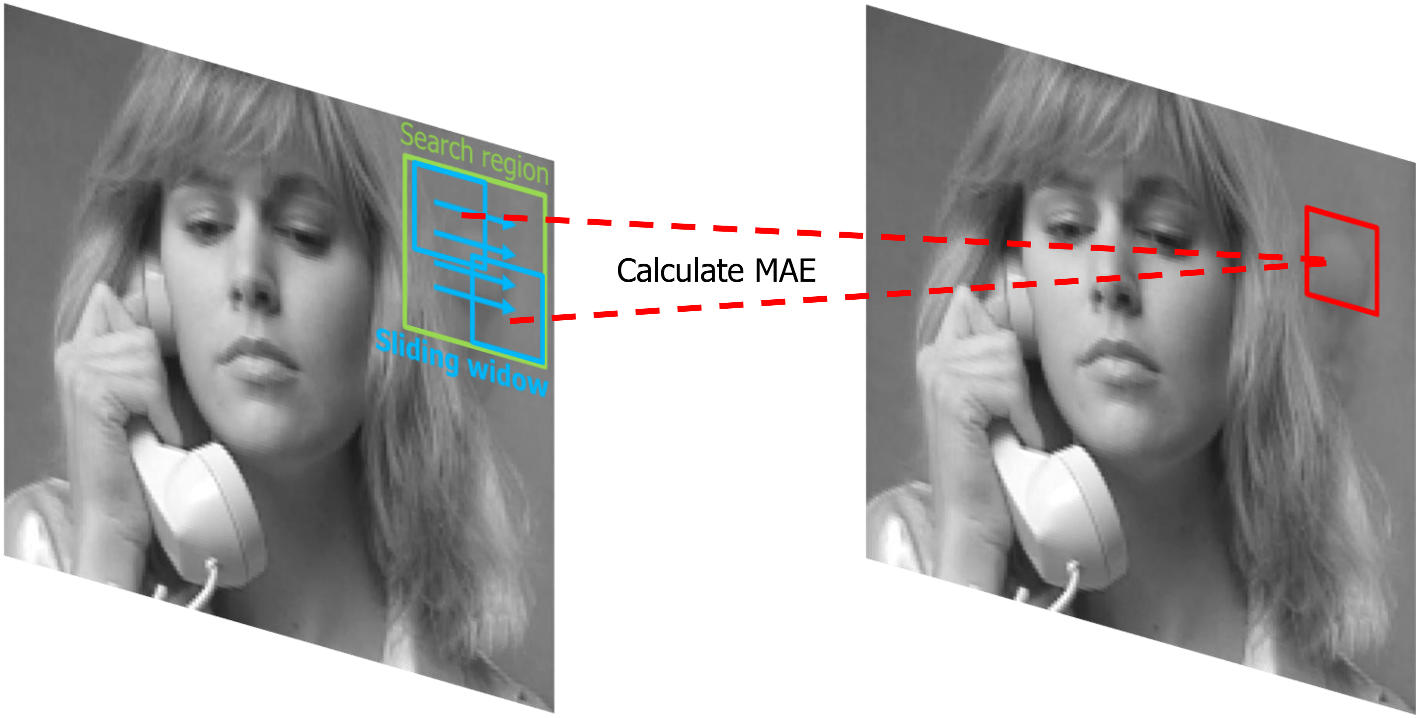
Previous frame



Current frame

Motion-Compensated Prediction

- Example

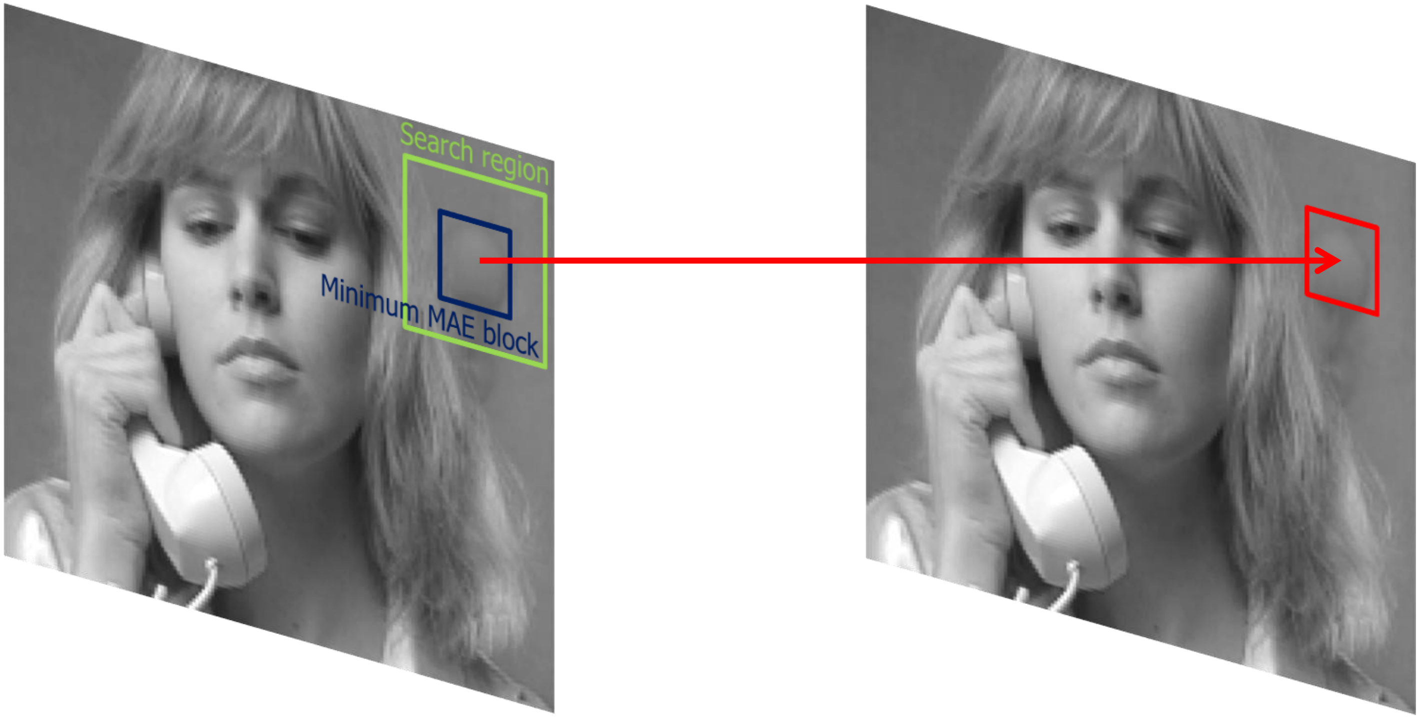


Previous frame

Current frame

Motion-Compensated Prediction

- Example

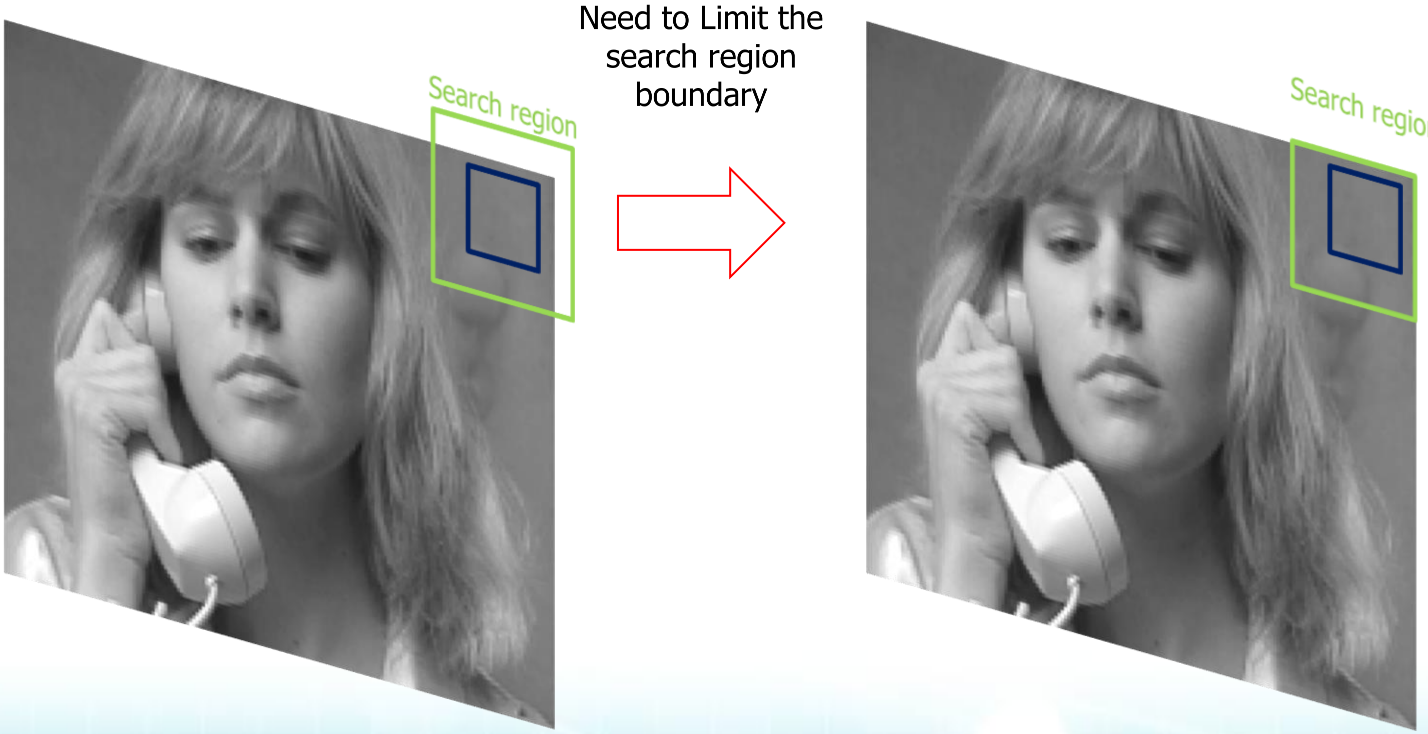


Previous frame

Current frame

Motion-Compensated Prediction

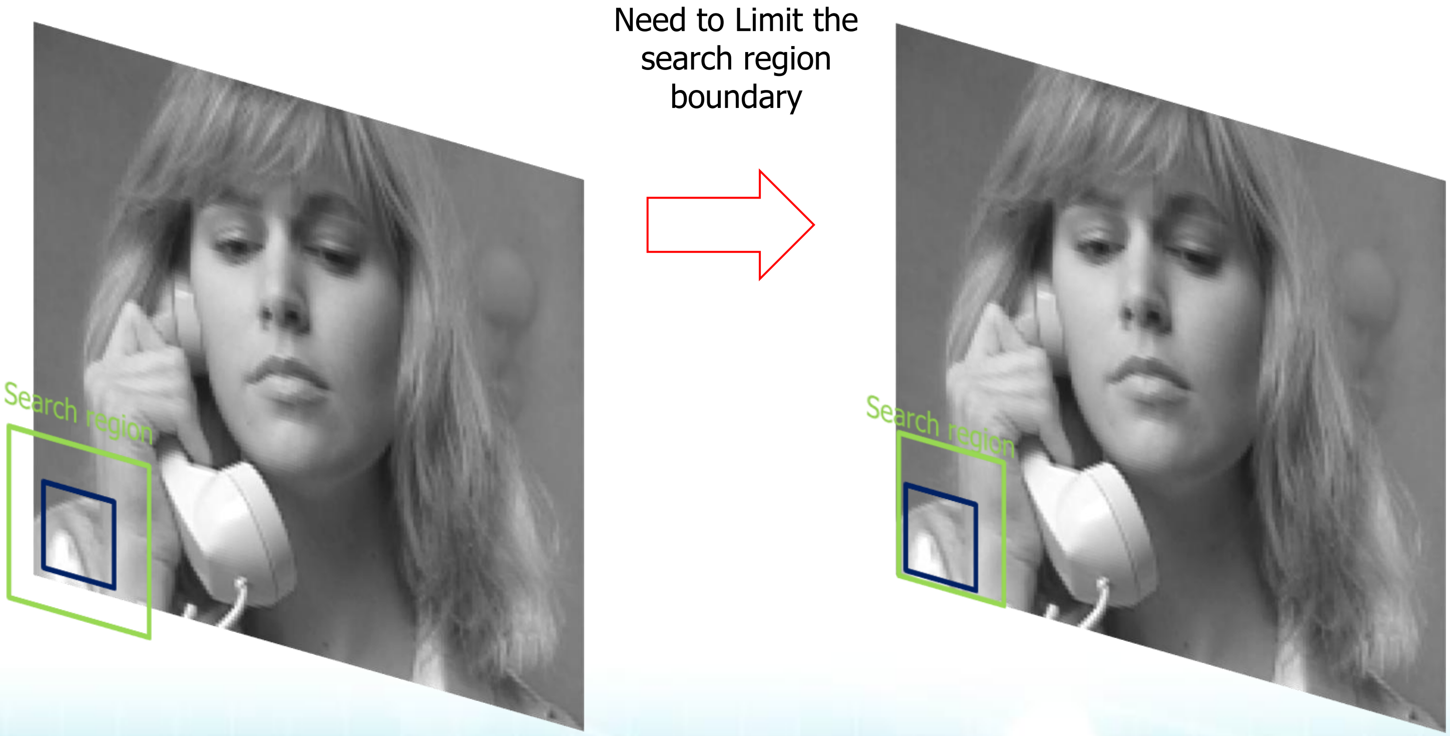
- Example
  - Adaptive Search Range decision



Need to Limit the search region boundary

Motion-Compensated Prediction

- Example
  - Adaptive Search Range decision

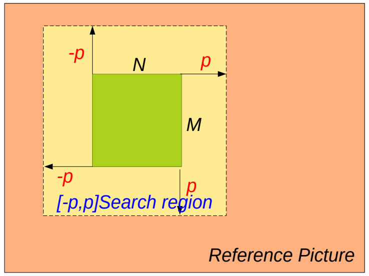


Need to Limit the search region boundary

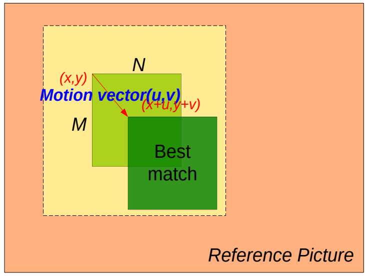
Example Code

```
#include <stdio.h>
#include <math.h>               //  header file
#include <stdlib.h>
#include <string.h>

//Parameter
#define WIDTH  352              //  CIF frame size
#define HEIGHT 288

#define BLOCK_SIZE 4                               //prediction block size
#define SR         16                              //Search Range

#define cWIDTH      (WIDTH>>1)                     //Chroma frame size
#define cHEIGHT     (HEIGHT>>1)                    //Chroma frame size

#define cBLOCK_SIZE (BLOCK_SIZE>>1)                //Chroma prediction block size
#define cSR         (SR>>1)                        //Chroma Search Range

#define Clip(x) ( x < 0 ? 0 : ( x > 255 ? 255 : x))

typedef unsigned char BYTE;

typedef struct MV   // motion vector structure
{
    int x,y;
}MV;

BYTE** MemAlloc_2D(int width, int height);                 // 2D memory allocation
void MemFree_2D(BYTE** arr, int height);                   // 2D memory free

float GetPSNR(BYTE** img_ori, BYTE** img_dist, int width, int height);  //PSNR value

int Read_Frame(FILE *fp_in, BYTE** img_in, int width, int height);                          // 1 frame read from input file
void Write_Frame(FILE *fp_out, BYTE** img_in, int width, int height);                       // 1 frame write on output file
void RGB_to_YUV(BYTE** img_in, BYTE** img_out, int height, int width);                      // Image color conversion RGB444 to YUV444
void YUV_to_RGB(BYTE** img_in, BYTE** img_out, int width, int height);                      // Image color conversion YUV444 to RGB444

void YUV444_to_420(BYTE** img_in, BYTE** img_Y, BYTE** img_U420, BYTE** img_V420, int width, int height);      // Chroma sampling  4:4:4 -> 4:2:0
void YUV420_to_444(BYTE** img_Y, BYTE** img_U420, BYTE** img_V420, BYTE** img_out, int width, int height);     // Chroma sampling  4:2:0 -> 4:4:4

void InterPrediction(BYTE** img_ori, BYTE** img_ref,BYTE** img_pred, BYTE** img_resi, BYTE** img_recon, int width, int height, int block_size, int search_range);   // inter-prediction
```

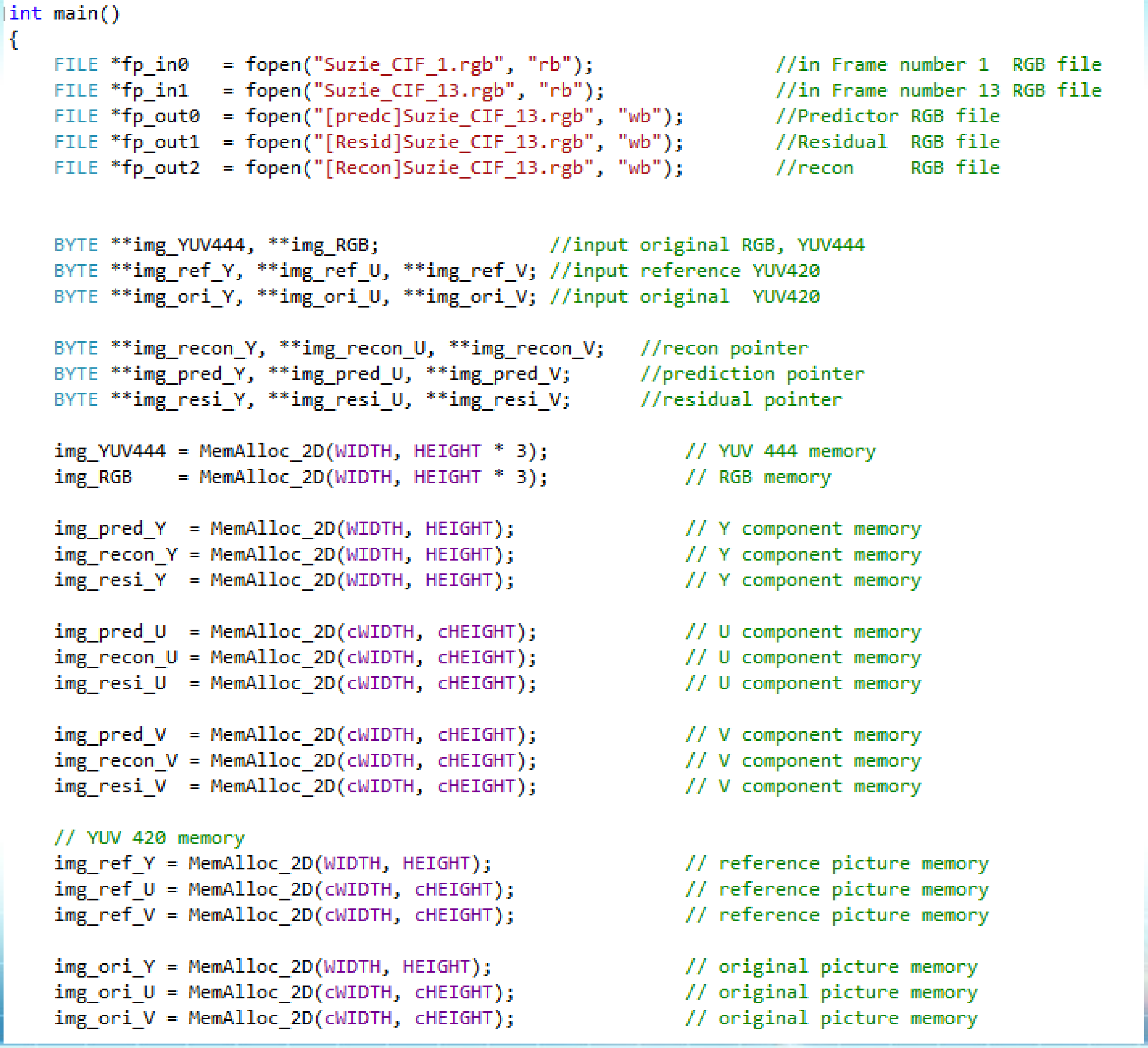
P = 16
N,M = 4



MV
structure
Has (x,y)

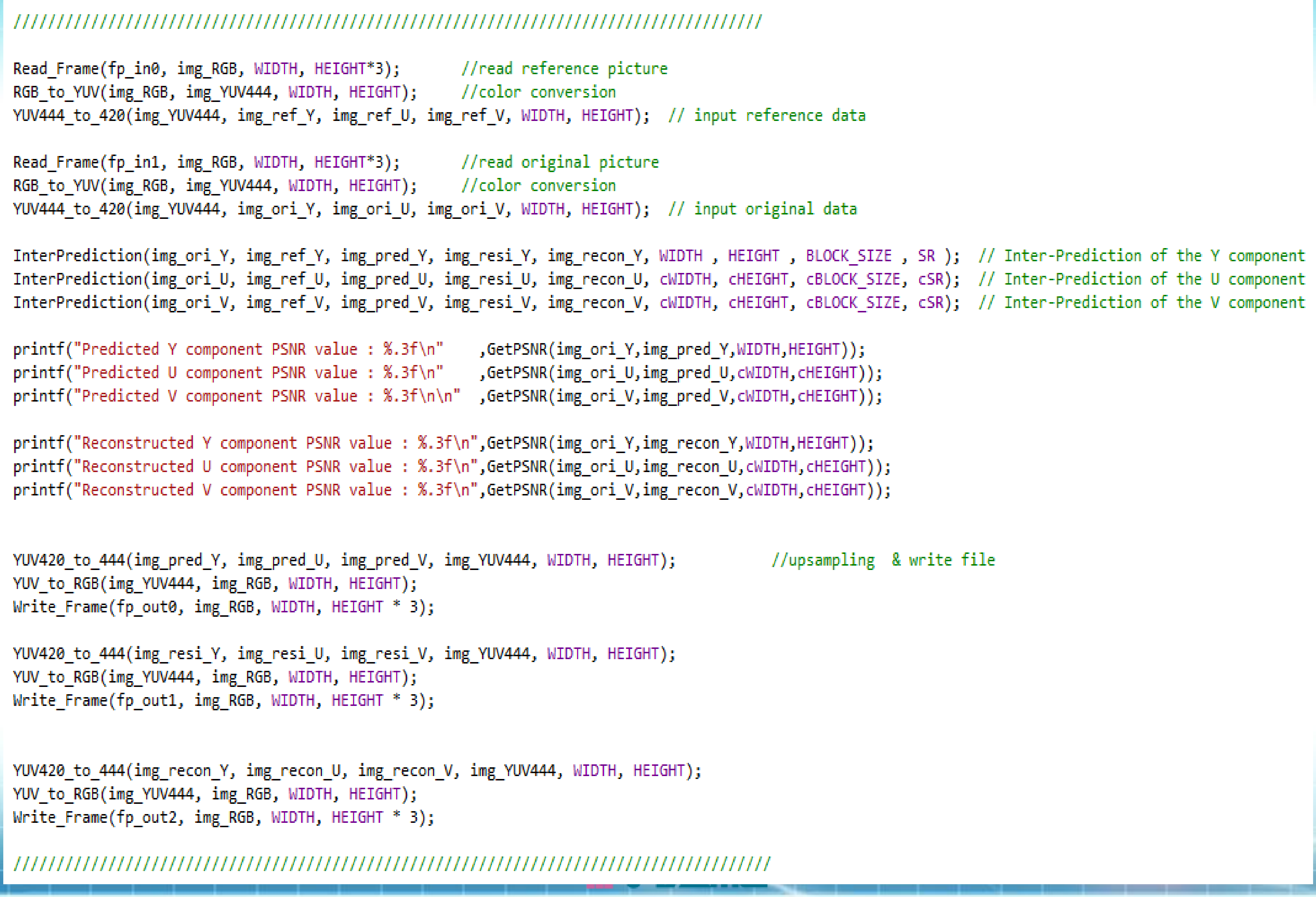


Example Code



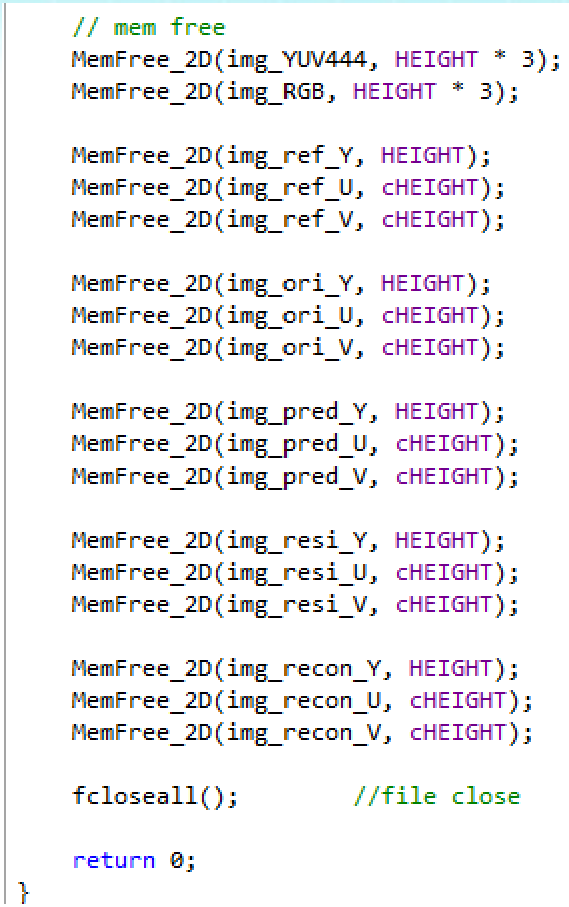
```
int main()
{
    FILE *fp_in0   = fopen("Suzie_CIF_1.rgb", "rb");                //in Frame number 1  RGB file
    FILE *fp_in1   = fopen("Suzie_CIF_13.rgb", "rb");               //in Frame number 13 RGB file
    FILE *fp_out0  = fopen("[predc]Suzie_CIF_13.rgb", "wb");        //Predictor RGB file
    FILE *fp_out1  = fopen("[Resid]Suzie_CIF_13.rgb", "wb");        //Residual  RGB file
    FILE *fp_out2  = fopen("[Recon]Suzie_CIF_13.rgb", "wb");        //recon     RGB file


    BYTE **img_YUV444, **img_RGB;               //input original RGB, YUV444
    BYTE **img_ref_Y, **img_ref_U, **img_ref_V; //input reference YUV420
    BYTE **img_ori_Y, **img_ori_U, **img_ori_V; //input original  YUV420

    BYTE **img_recon_Y, **img_recon_U, **img_recon_V;   //recon pointer
    BYTE **img_pred_Y, **img_pred_U, **img_pred_V;      //prediction pointer
    BYTE **img_resi_Y, **img_resi_U, **img_resi_V;      //residual pointer

    img_YUV444 = MemAlloc_2D(WIDTH, HEIGHT * 3);            // YUV 444 memory
    img_RGB    = MemAlloc_2D(WIDTH, HEIGHT * 3);            // RGB memory

    img_pred_Y  = MemAlloc_2D(WIDTH, HEIGHT);               // Y component memory
    img_recon_Y = MemAlloc_2D(WIDTH, HEIGHT);               // Y component memory
    img_resi_Y  = MemAlloc_2D(WIDTH, HEIGHT);               // Y component memory

    img_pred_U  = MemAlloc_2D(cWIDTH, cHEIGHT);             // U component memory
    img_recon_U = MemAlloc_2D(cWIDTH, cHEIGHT);             // U component memory
    img_resi_U  = MemAlloc_2D(cWIDTH, cHEIGHT);             // U component memory

    img_pred_V  = MemAlloc_2D(cWIDTH, cHEIGHT);             // V component memory
    img_recon_V = MemAlloc_2D(cWIDTH, cHEIGHT);             // V component memory
    img_resi_V  = MemAlloc_2D(cWIDTH, cHEIGHT);             // V component memory

    // YUV 420 memory
    img_ref_Y = MemAlloc_2D(WIDTH, HEIGHT);                 // reference picture memory
    img_ref_U = MemAlloc_2D(cWIDTH, cHEIGHT);               // reference picture memory
    img_ref_V = MemAlloc_2D(cWIDTH, cHEIGHT);               // reference picture memory

    img_ori_Y = MemAlloc_2D(WIDTH, HEIGHT);                 // original picture memory
    img_ori_U = MemAlloc_2D(cWIDTH, cHEIGHT);               // original picture memory
    img_ori_V = MemAlloc_2D(cWIDTH, cHEIGHT);               // original picture memory
```

Example Code



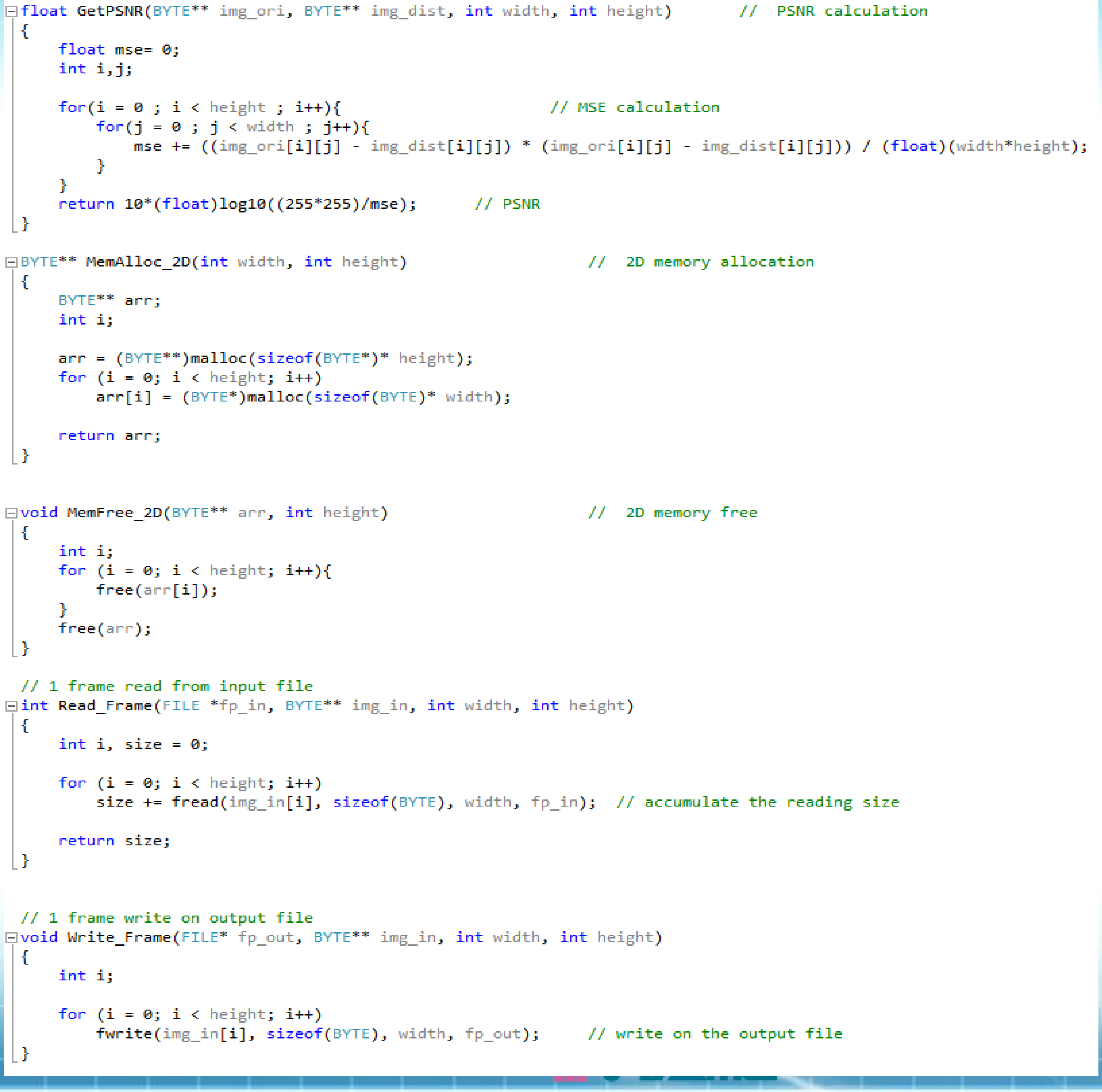
```
/////////////////////////////////////////////////////////////////////////////////////////

Read_Frame(fp_in0, img_RGB, WIDTH, HEIGHT*3);       //read reference picture
RGB_to_YUV(img_RGB, img_YUV444, WIDTH, HEIGHT);     //color conversion
YUV444_to_420(img_YUV444, img_ref_Y, img_ref_U, img_ref_V, WIDTH, HEIGHT);  // input reference data

Read_Frame(fp_in1, img_RGB, WIDTH, HEIGHT*3);       //read original picture
RGB_to_YUV(img_RGB, img_YUV444, WIDTH, HEIGHT);     //color conversion
YUV444_to_420(img_YUV444, img_ori_Y, img_ori_U, img_ori_V, WIDTH, HEIGHT);  // input original data

InterPrediction(img_ori_Y, img_ref_Y, img_pred_Y, img_resi_Y, img_recon_Y, WIDTH , HEIGHT , BLOCK_SIZE , SR );  // Inter-Prediction of the Y component
InterPrediction(img_ori_U, img_ref_U, img_pred_U, img_resi_U, img_recon_U, cWIDTH, cHEIGHT, cBLOCK_SIZE, cSR);  // Inter-Prediction of the U component
InterPrediction(img_ori_V, img_ref_V, img_pred_V, img_resi_V, img_recon_V, cWIDTH, cHEIGHT, cBLOCK_SIZE, cSR);  // Inter-Prediction of the V component

printf("Predicted Y component PSNR value : %.3f\n"    ,GetPSNR(img_ori_Y,img_pred_Y,WIDTH,HEIGHT));
printf("Predicted U component PSNR value : %.3f\n"    ,GetPSNR(img_ori_U,img_pred_U,cWIDTH,cHEIGHT));
printf("Predicted V component PSNR value : %.3f\n\n"  ,GetPSNR(img_ori_V,img_pred_V,cWIDTH,cHEIGHT));

printf("Reconstructed Y component PSNR value : %.3f\n",GetPSNR(img_ori_Y,img_recon_Y,WIDTH,HEIGHT));
printf("Reconstructed U component PSNR value : %.3f\n",GetPSNR(img_ori_U,img_recon_U,cWIDTH,cHEIGHT));
printf("Reconstructed V component PSNR value : %.3f\n",GetPSNR(img_ori_V,img_recon_V,cWIDTH,cHEIGHT));


YUV420_to_444(img_pred_Y, img_pred_U, img_pred_V, img_YUV444, WIDTH, HEIGHT);           //upsampling  & write file
YUV_to_RGB(img_YUV444, img_RGB, WIDTH, HEIGHT);
Write_Frame(fp_out0, img_RGB, WIDTH, HEIGHT * 3);

YUV420_to_444(img_resi_Y, img_resi_U, img_resi_V, img_YUV444, WIDTH, HEIGHT);
YUV_to_RGB(img_YUV444, img_RGB, WIDTH, HEIGHT);
Write_Frame(fp_out1, img_RGB, WIDTH, HEIGHT * 3);


YUV420_to_444(img_recon_Y, img_recon_U, img_recon_V, img_YUV444, WIDTH, HEIGHT);
YUV_to_RGB(img_YUV444, img_RGB, WIDTH, HEIGHT);
Write_Frame(fp_out2, img_RGB, WIDTH, HEIGHT * 3);

//////////////////////////////////////////////////////////////////////////////////////////
```

Example Code



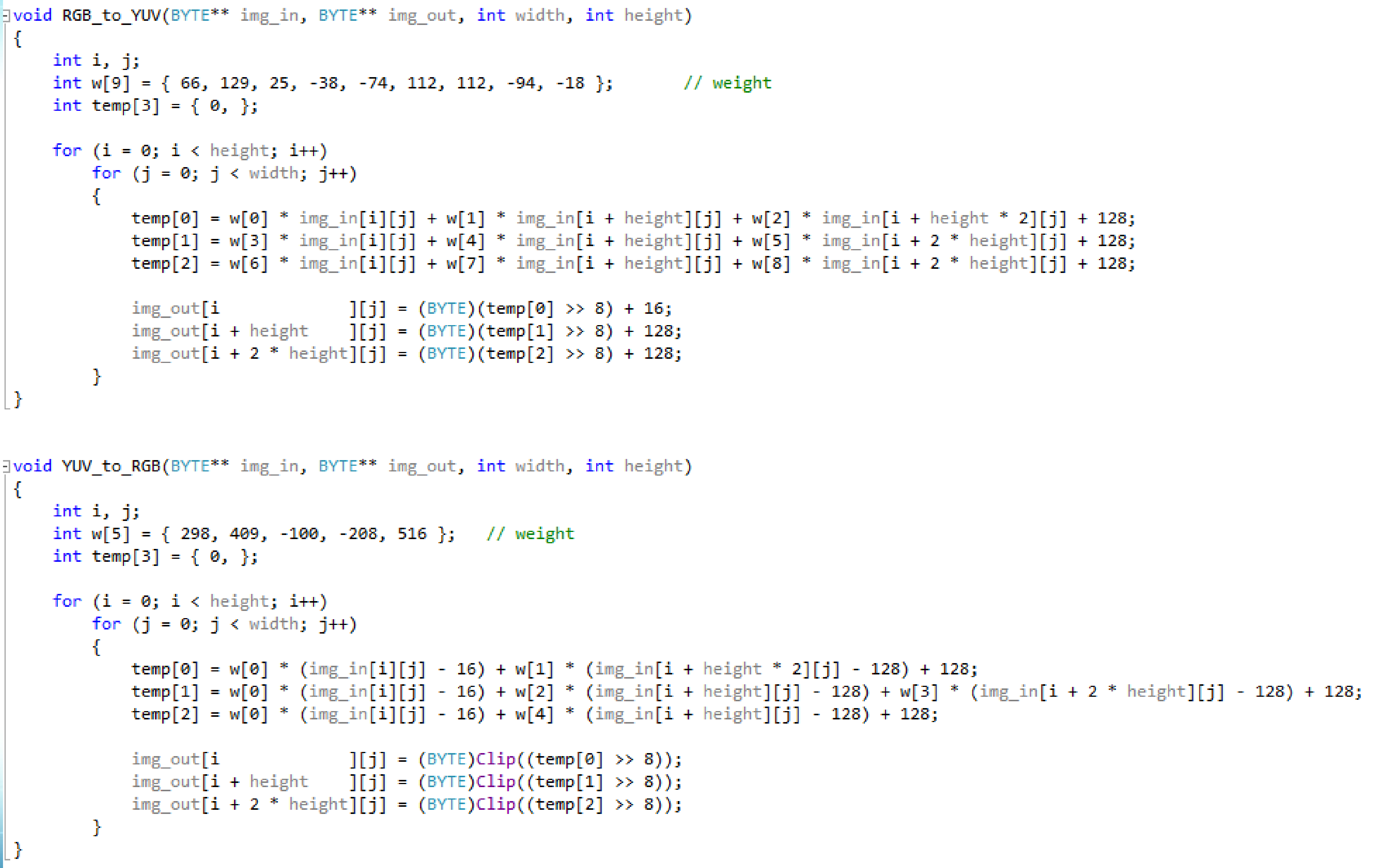
```
    // mem free
    MemFree_2D(img_YUV444, HEIGHT * 3);
    MemFree_2D(img_RGB, HEIGHT * 3);

    MemFree_2D(img_ref_Y, HEIGHT);
    MemFree_2D(img_ref_U, cHEIGHT);
    MemFree_2D(img_ref_V, cHEIGHT);

    MemFree_2D(img_ori_Y, HEIGHT);
    MemFree_2D(img_ori_U, cHEIGHT);
    MemFree_2D(img_ori_V, cHEIGHT);

    MemFree_2D(img_pred_Y, HEIGHT);
    MemFree_2D(img_pred_U, cHEIGHT);
    MemFree_2D(img_pred_V, cHEIGHT);

    MemFree_2D(img_resi_Y, HEIGHT);
    MemFree_2D(img_resi_U, cHEIGHT);
    MemFree_2D(img_resi_V, cHEIGHT);

    MemFree_2D(img_recon_Y, HEIGHT);
    MemFree_2D(img_recon_U, cHEIGHT);
    MemFree_2D(img_recon_V, cHEIGHT);

    fcloseall();        //file close

    return 0;
}
```

Example Code



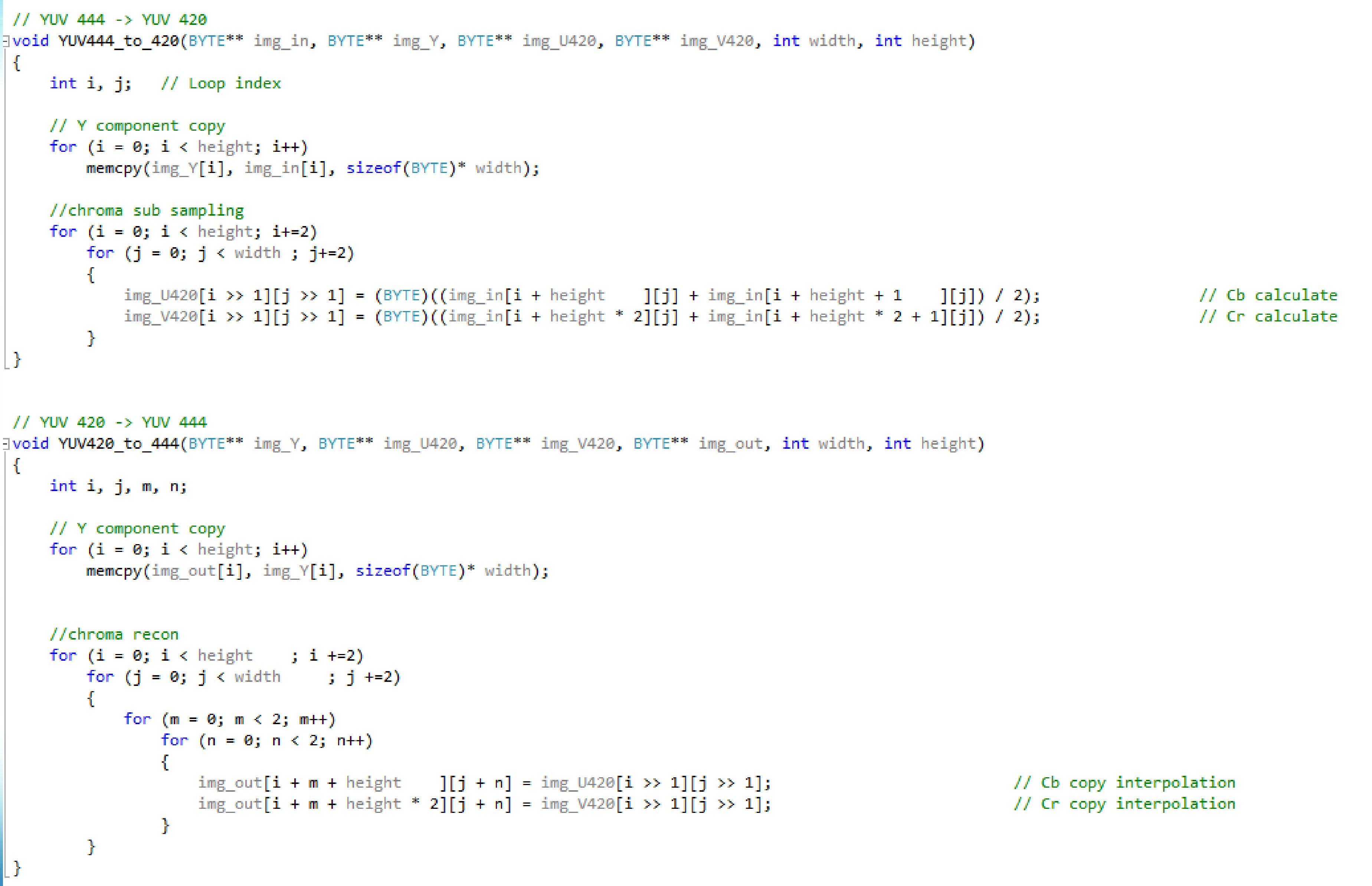
```c
float GetPSNR(BYTE** img_ori, BYTE** img_dist, int width, int height)       //  PSNR calculation
{
    float mse= 0;
    int i,j;

    for(i = 0 ; i < height ; i++){                          // MSE calculation
        for(j = 0 ; j < width ; j++){
            mse += ((img_ori[i][j] - img_dist[i][j]) * (img_ori[i][j] - img_dist[i][j])) / (float)(width*height);
        }
    }
    return 10*(float)log10((255*255)/mse);      // PSNR
}

BYTE** MemAlloc_2D(int width, int height)                   //  2D memory allocation
{
    BYTE** arr;
    int i;

    arr = (BYTE**)malloc(sizeof(BYTE*)* height);
    for (i = 0; i < height; i++)
        arr[i] = (BYTE*)malloc(sizeof(BYTE)* width);

    return arr;
}


void MemFree_2D(BYTE** arr, int height)                     //  2D memory free
{
    int i;
    for (i = 0; i < height; i++){
        free(arr[i]);
    }
    free(arr);
}

// 1 frame read from input file
int Read_Frame(FILE *fp_in, BYTE** img_in, int width, int height)
{
    int i, size = 0;

    for (i = 0; i < height; i++)
        size += fread(img_in[i], sizeof(BYTE), width, fp_in);  // accumulate the reading size

    return size;
}


// 1 frame write on output file
void Write_Frame(FILE* fp_out, BYTE** img_in, int width, int height)
{
    int i;

    for (i = 0; i < height; i++)
        fwrite(img_in[i], sizeof(BYTE), width, fp_out);     // write on the output file
}
```

Example Code



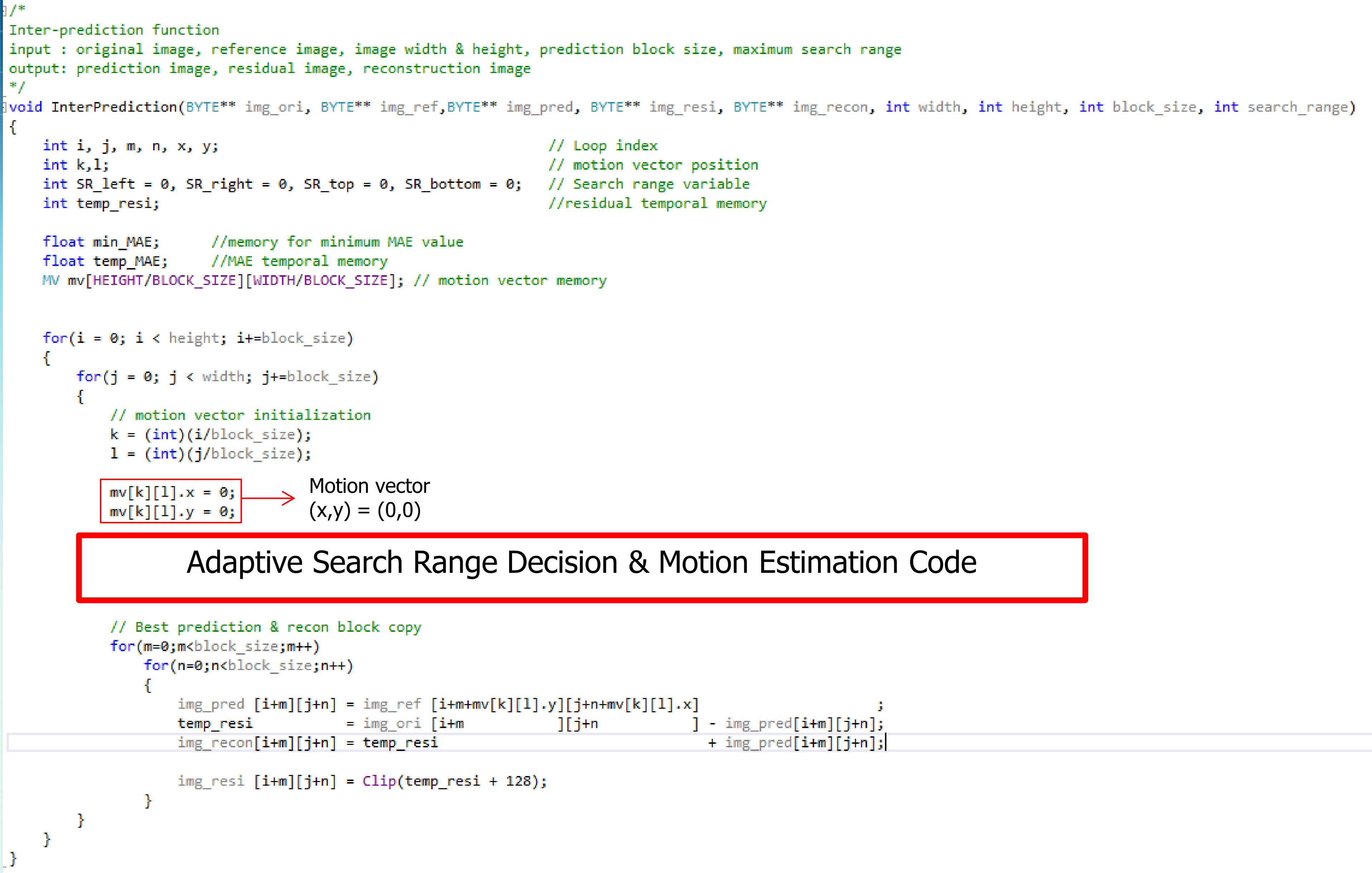
```
void RGB_to_YUV(BYTE** img_in, BYTE** img_out, int width, int height)
{
    int i, j;
    int w[9] = { 66, 129, 25, -38, -74, 112, 112, -94, -18 };       // weight
    int temp[3] = { 0, };

    for (i = 0; i < height; i++)
        for (j = 0; j < width; j++)
        {
            temp[0] = w[0] * img_in[i][j] + w[1] * img_in[i + height][j] + w[2] * img_in[i + height * 2][j] + 128;
            temp[1] = w[3] * img_in[i][j] + w[4] * img_in[i + height][j] + w[5] * img_in[i + 2 * height][j] + 128;
            temp[2] = w[6] * img_in[i][j] + w[7] * img_in[i + height][j] + w[8] * img_in[i + 2 * height][j] + 128;

            img_out[i             ][j] = (BYTE)(temp[0] >> 8) + 16;
            img_out[i + height    ][j] = (BYTE)(temp[1] >> 8) + 128;
            img_out[i + 2 * height][j] = (BYTE)(temp[2] >> 8) + 128;
        }
}


void YUV_to_RGB(BYTE** img_in, BYTE** img_out, int width, int height)
{
    int i, j;
    int w[5] = { 298, 409, -100, -208, 516 };   // weight
    int temp[3] = { 0, };

    for (i = 0; i < height; i++)
        for (j = 0; j < width; j++)
        {
            temp[0] = w[0] * (img_in[i][j] - 16) + w[1] * (img_in[i + height * 2][j] - 128) + 128;
            temp[1] = w[0] * (img_in[i][j] - 16) + w[2] * (img_in[i + height][j] - 128) + w[3] * (img_in[i + 2 * height][j] - 128) + 128;
            temp[2] = w[0] * (img_in[i][j] - 16) + w[4] * (img_in[i + height][j] - 128) + 128;

            img_out[i             ][j] = (BYTE)Clip((temp[0] >> 8));
            img_out[i + height    ][j] = (BYTE)Clip((temp[1] >> 8));
            img_out[i + 2 * height][j] = (BYTE)Clip((temp[2] >> 8));
        }
}
```

Example Code



```
// YUV 444 -> YUV 420
void YUV444_to_420(BYTE** img_in, BYTE** img_Y, BYTE** img_U420, BYTE** img_V420, int width, int height)
{
    int i, j;   // Loop index

    // Y component copy
    for (i = 0; i < height; i++)
        memcpy(img_Y[i], img_in[i], sizeof(BYTE)* width);

    //chroma sub sampling
    for (i = 0; i < height; i+=2)
        for (j = 0; j < width ; j+=2)
        {
            img_U420[i >> 1][j >> 1] = (BYTE)((img_in[i + height    ][j] + img_in[i + height + 1    ][j]) / 2);                // Cb calculate
            img_V420[i >> 1][j >> 1] = (BYTE)((img_in[i + height * 2][j] + img_in[i + height * 2 + 1][j]) / 2);                // Cr calculate
        }
}


// YUV 420 -> YUV 444
void YUV420_to_444(BYTE** img_Y, BYTE** img_U420, BYTE** img_V420, BYTE** img_out, int width, int height)
{
    int i, j, m, n;

    // Y component copy
    for (i = 0; i < height; i++)
        memcpy(img_out[i], img_Y[i], sizeof(BYTE)* width);


    //chroma recon
    for (i = 0; i < height    ; i +=2)
        for (j = 0; j < width     ; j +=2)
        {
            for (m = 0; m < 2; m++)
                for (n = 0; n < 2; n++)
                {
                    img_out[i + m + height    ][j + n] = img_U420[i >> 1][j >> 1];                          // Cb copy interpolation
                    img_out[i + m + height * 2][j + n] = img_V420[i >> 1][j >> 1];                          // Cr copy interpolation
                }
        }
}
```

Example Code



```
/*
Inter-prediction function
input : original image, reference image, image width & height, prediction block size, maximum search range
output: prediction image, residual image, reconstruction image
*/
void InterPrediction(BYTE** img_ori, BYTE** img_ref,BYTE** img_pred, BYTE** img_resi, BYTE** img_recon, int width, int height, int block_size, int search_range)
{
    int i, j, m, n, x, y;                                           // Loop index
    int k,l;                                                        // motion vector position
    int SR_left = 0, SR_right = 0, SR_top = 0, SR_bottom = 0;      // Search range variable
    int temp_resi;                                                  //residual temporal memory

    float min_MAE;      //memory for minimum MAE value
    float temp_MAE;     //MAE temporal memory
    MV mv[HEIGHT/BLOCK_SIZE][WIDTH/BLOCK_SIZE]; // motion vector memory


    for(i = 0; i < height; i+=block_size)
    {
        for(j = 0; j < width; j+=block_size)
        {
            // motion vector initialization
            k = (int)(i/block_size);
            l = (int)(j/block_size);

            mv[k][l].x = 0;
            mv[k][l].y = 0;
```

Motion vector
(x,y) = (0,0)

Adaptive Search Range Decision & Motion Estimation Code

```
            // Best prediction & recon block copy
            for(m=0;m<block_size;m++)
                for(n=0;n<block_size;n++)
                {
                    img_pred [i+m][j+n] = img_ref [i+m+mv[k][l].y][j+n+mv[k][l].x]                     ;
                    temp_resi           = img_ori [i+m           ][j+n           ] - img_pred[i+m][j+n];
                    img_recon[i+m][j+n] = temp_resi                                + img_pred[i+m][j+n];

                    img_resi [i+m][j+n] = Clip(temp_resi + 128);
                }
        }
    }
}
```

Result

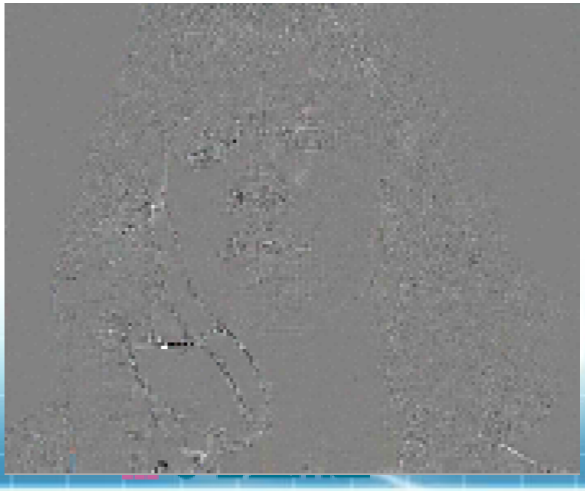
Reference image

Original image

Result

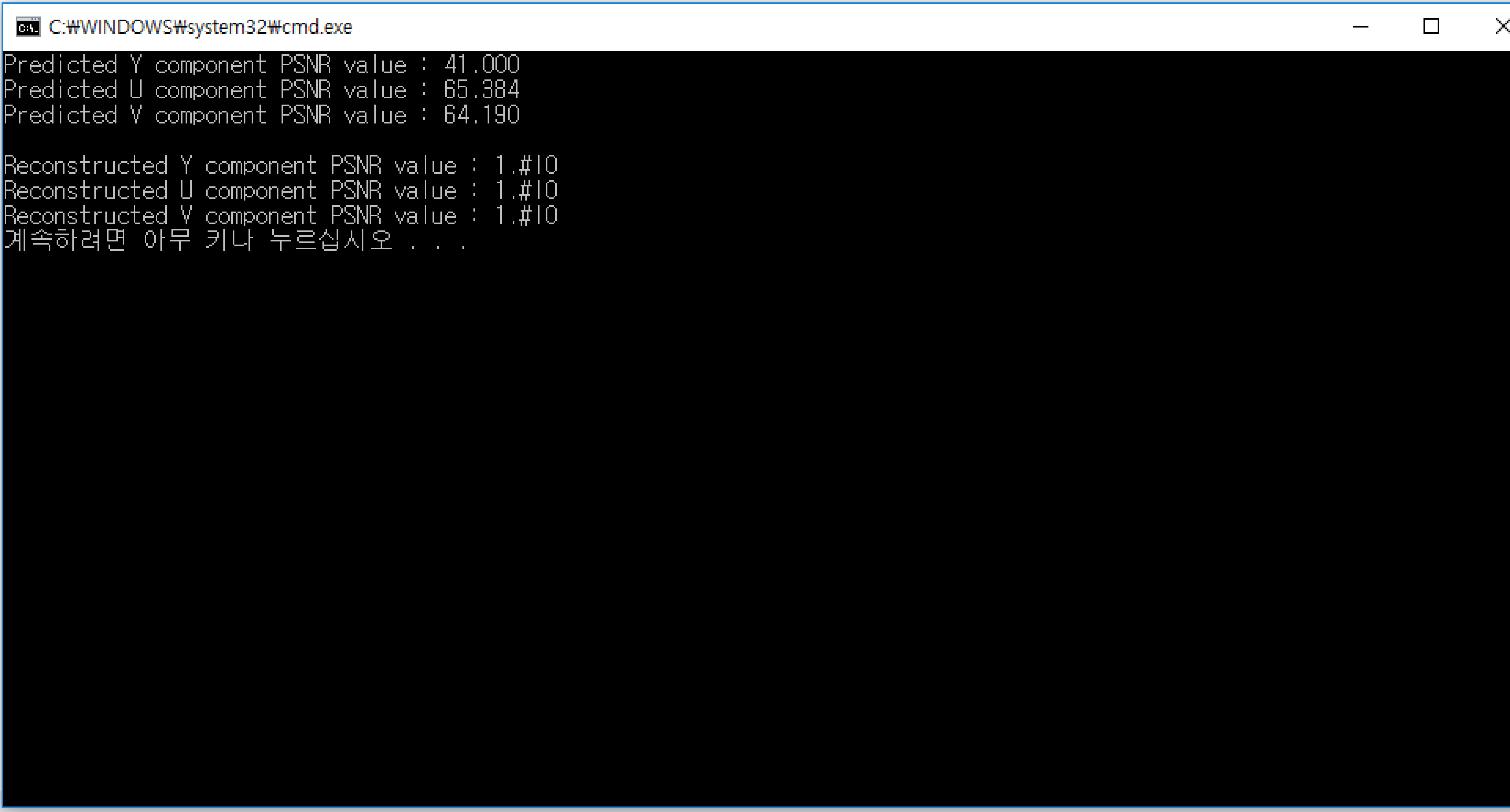
Precidtion image

Recon image



Residual image

Result



```
Predicted Y component PSNR value : 41.000
Predicted U component PSNR value : 65.384
Predicted V component PSNR value : 64.190

Reconstructed Y component PSNR value : 1.#IO
Reconstructed U component PSNR value : 1.#IO
Reconstructed V component PSNR value : 1.#IO
계속하려면 아무 키나 누르십시오 . . .
```