KWANGWOON UNIVERSITY

Intra Prediction Part.2

2018.10.23

Seoungjun Oh(sjoh@kw.ac.kr)
Wooju Lee (krosea@kw.ac.kr)

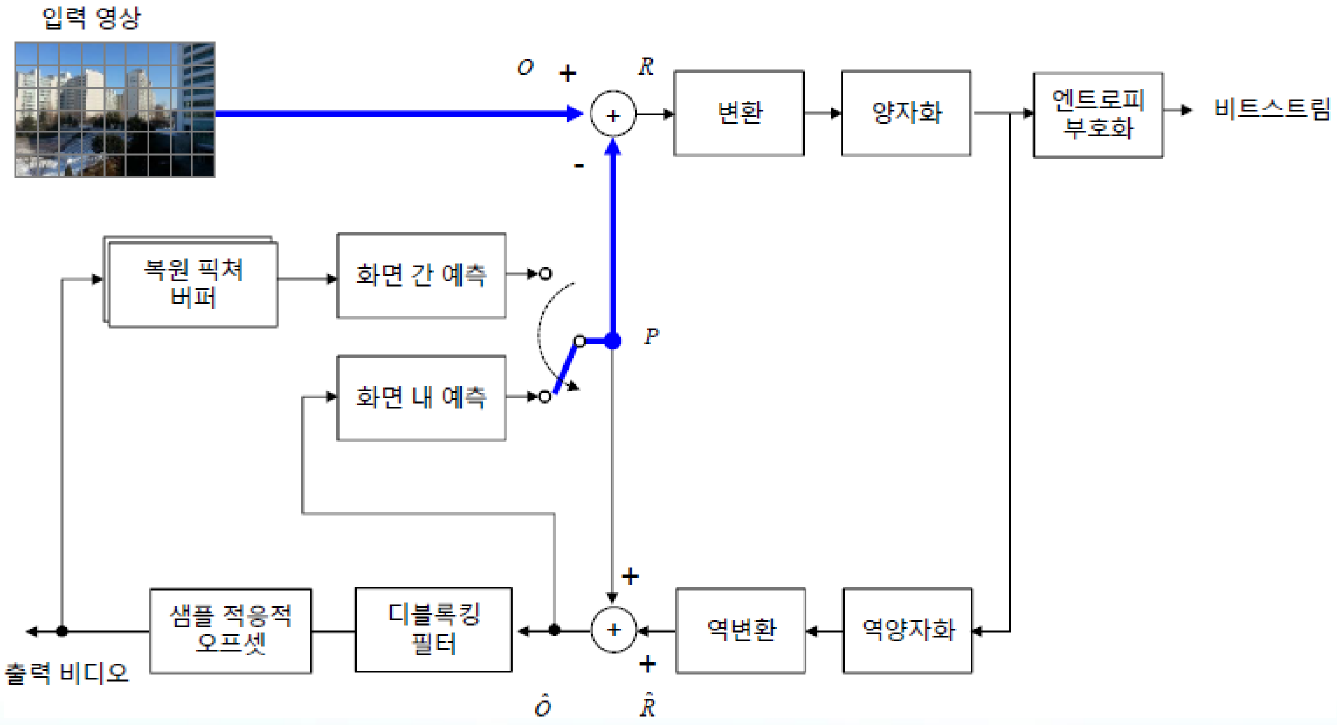
Multimedia LAB
VIA-Multimedia Center, Kwangwoon University

Contents

- Basic Codec
- Make Predictors
- Best Mode Selection
- Reconstruction

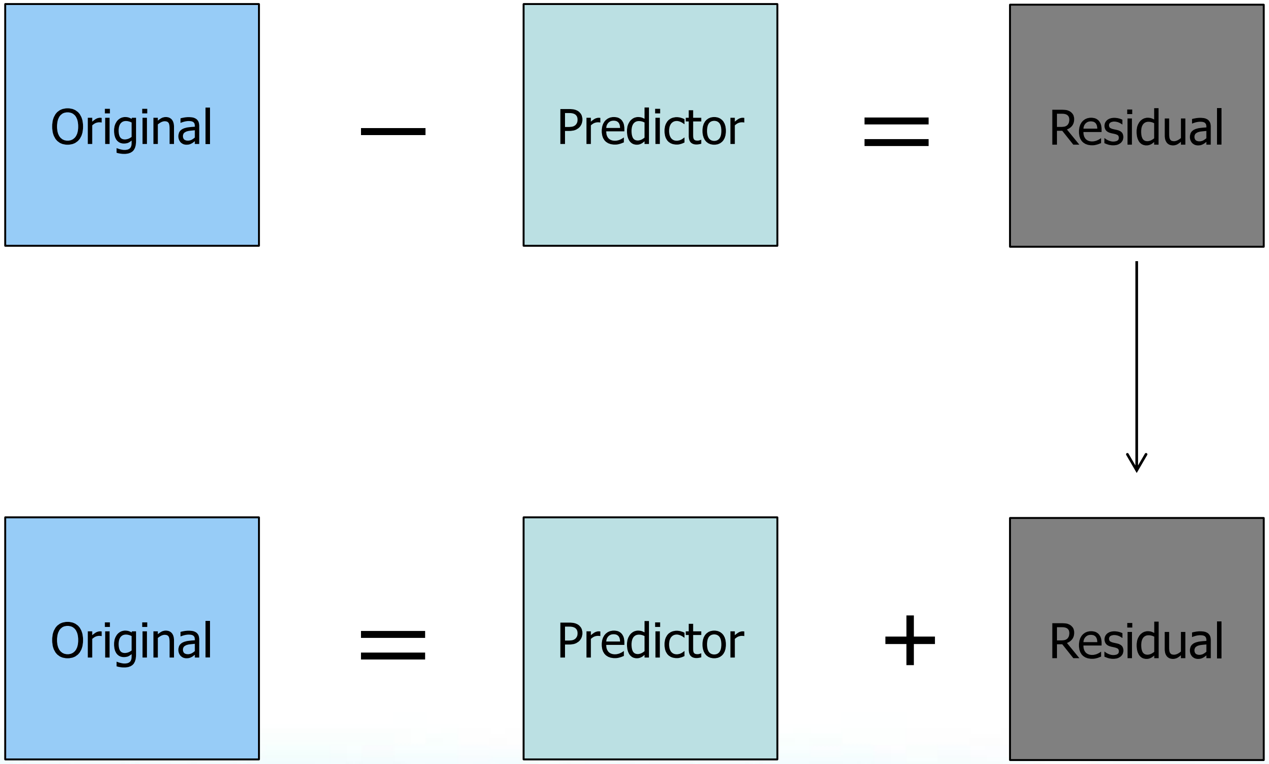
Basic Codec

❖ Structure of Encoder



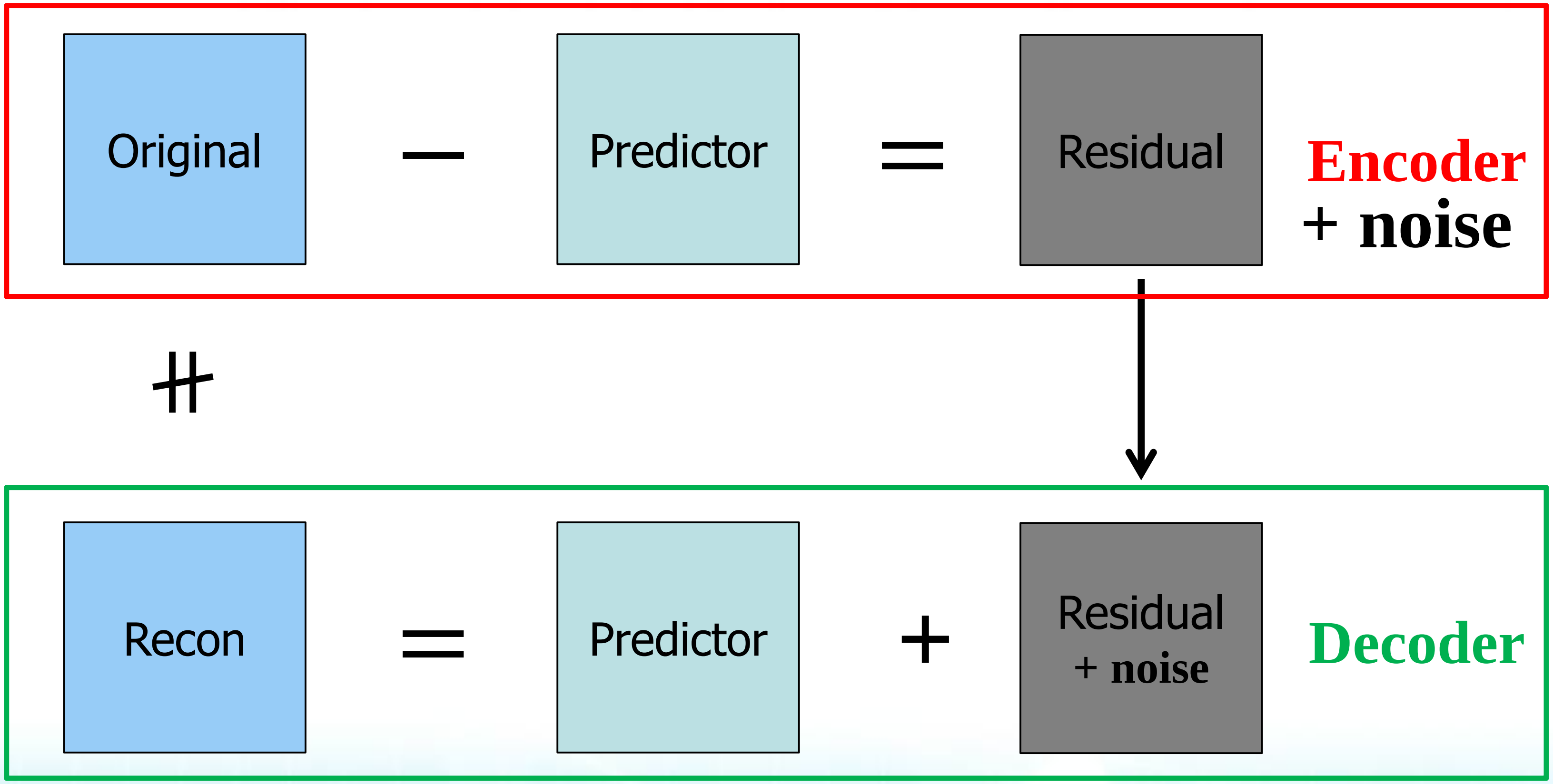
Basic Codec

- Basic Image/Video Codec



Basic Codec

- Basic Image/Video Codec



Original − Predictor = Residual

Encoder + noise

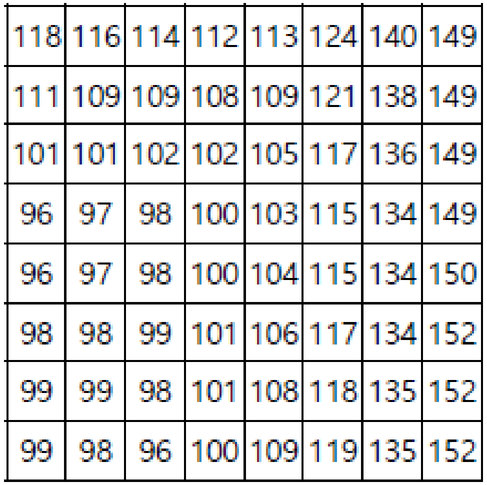
≠

Recon = Predictor + Residual + noise

Decoder

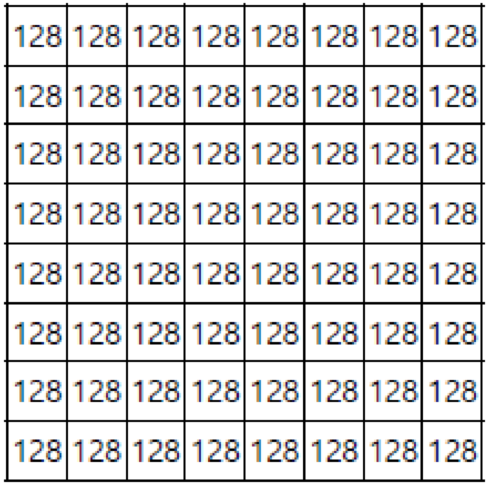
Basic Codec

❖ Structure of Encoder



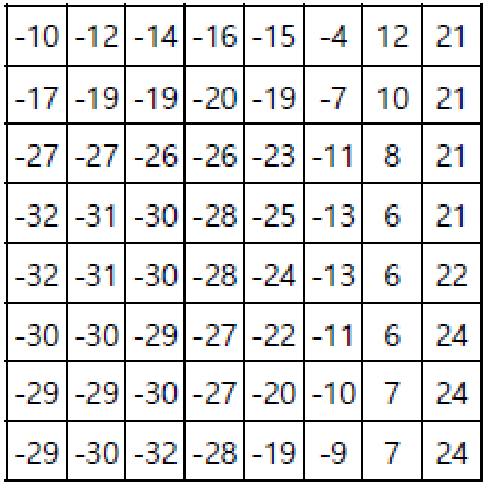
118	116	114	112	113	124	140	149
111	109	109	108	109	121	138	149
101	101	102	102	105	117	136	149
96	97	98	100	103	115	134	149
96	97	98	100	104	115	134	150
98	98	99	101	106	117	134	152
99	99	98	101	108	118	135	152
99	98	96	100	109	119	135	152

(a) 8x8 CU 픽셀 값



128	128	128	128	128	128	128	128
128	128	128	128	128	128	128	128
128	128	128	128	128	128	128	128
128	128	128	128	128	128	128	128
128	128	128	128	128	128	128	128
128	128	128	128	128	128	128	128
128	128	128	128	128	128	128	128
128	128	128	128	128	128	128	128

(b) 8x8 PU 예측 값 (Intra_Planar모드)

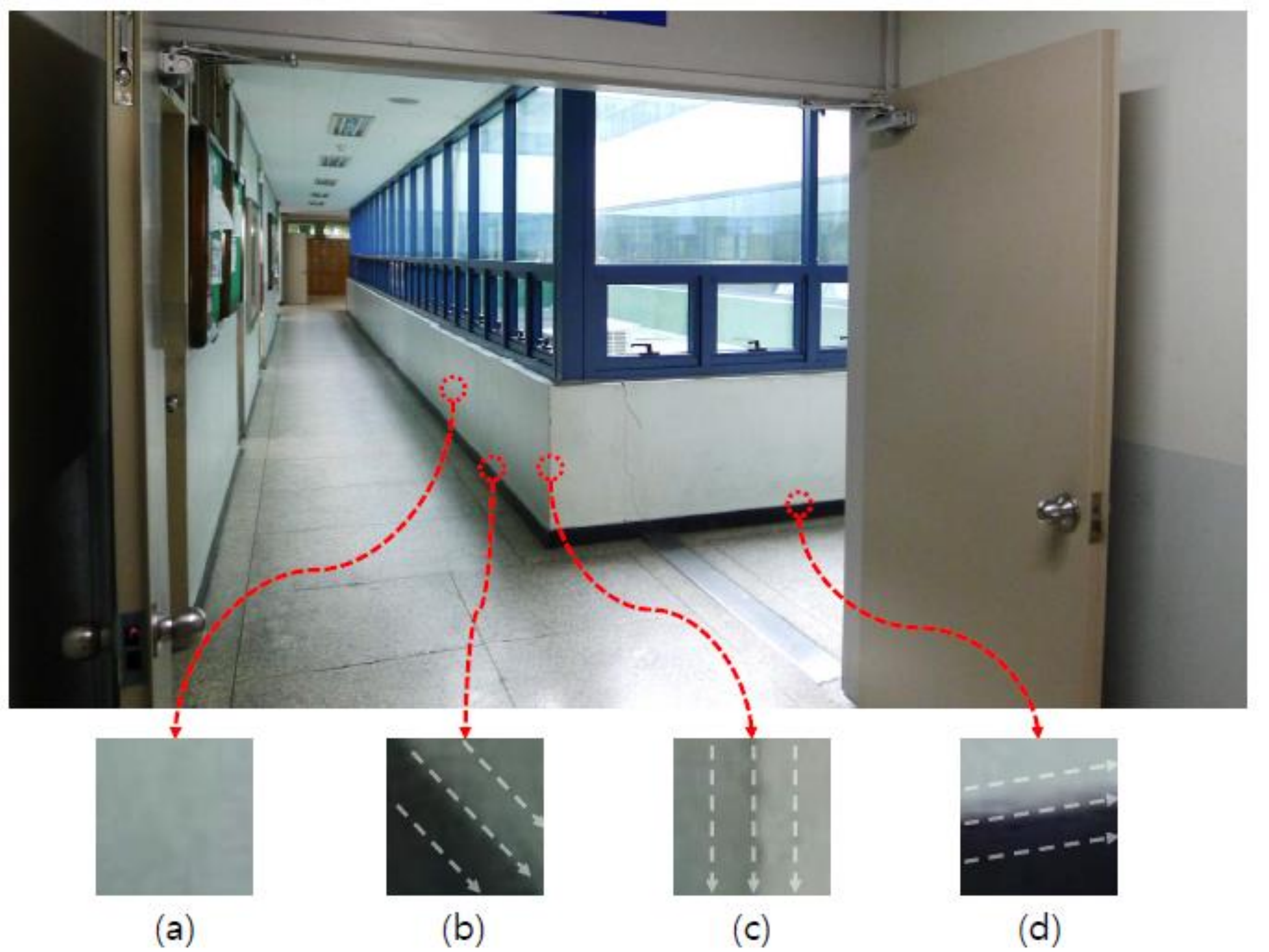


-10	-12	-14	-16	-15	-4	12	21
-17	-19	-19	-20	-19	-7	10	21
-27	-27	-26	-26	-23	-11	8	21
-32	-31	-30	-28	-25	-13	6	21
-32	-31	-30	-28	-24	-13	6	22
-30	-30	-29	-27	-22	-11	6	24
-29	-29	-30	-27	-20	-10	7	24
-29	-30	-32	-28	-19	-9	7	24

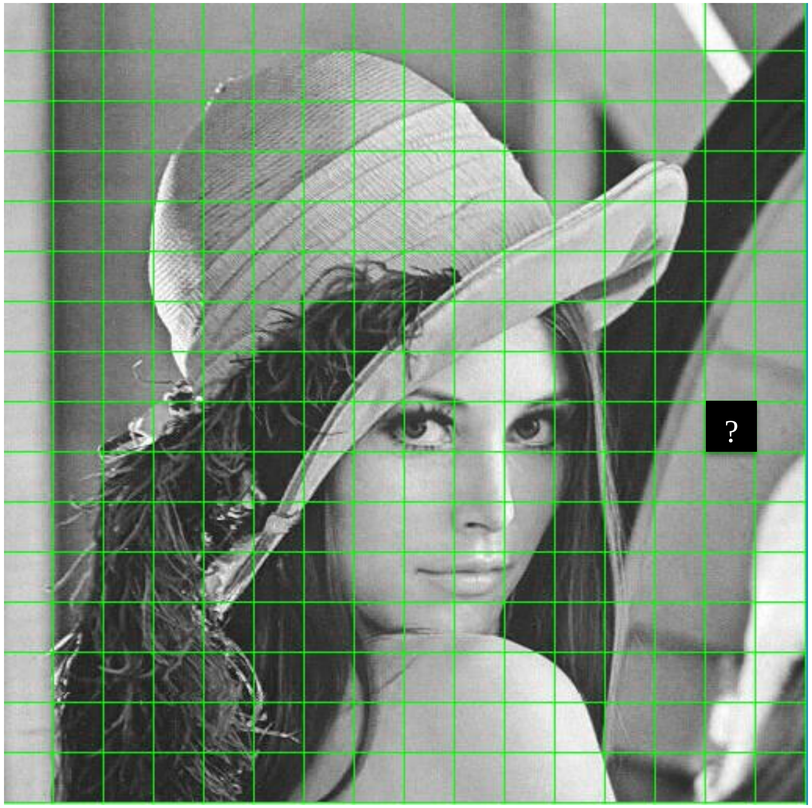
(c) 8x8 차분 블록 계수 값

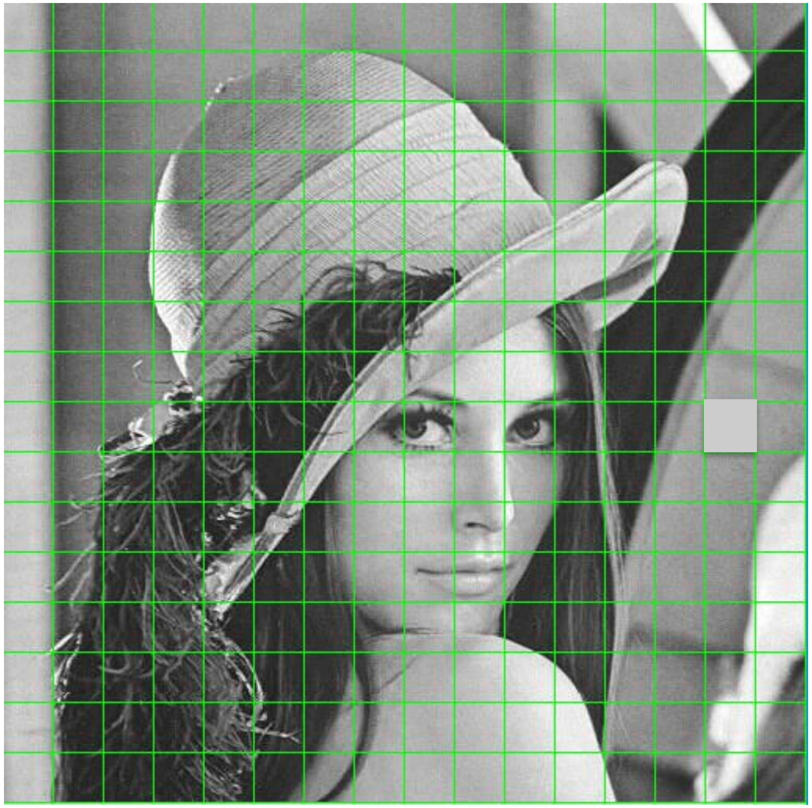
Original image — Predictor = Residual

Intra Prediction

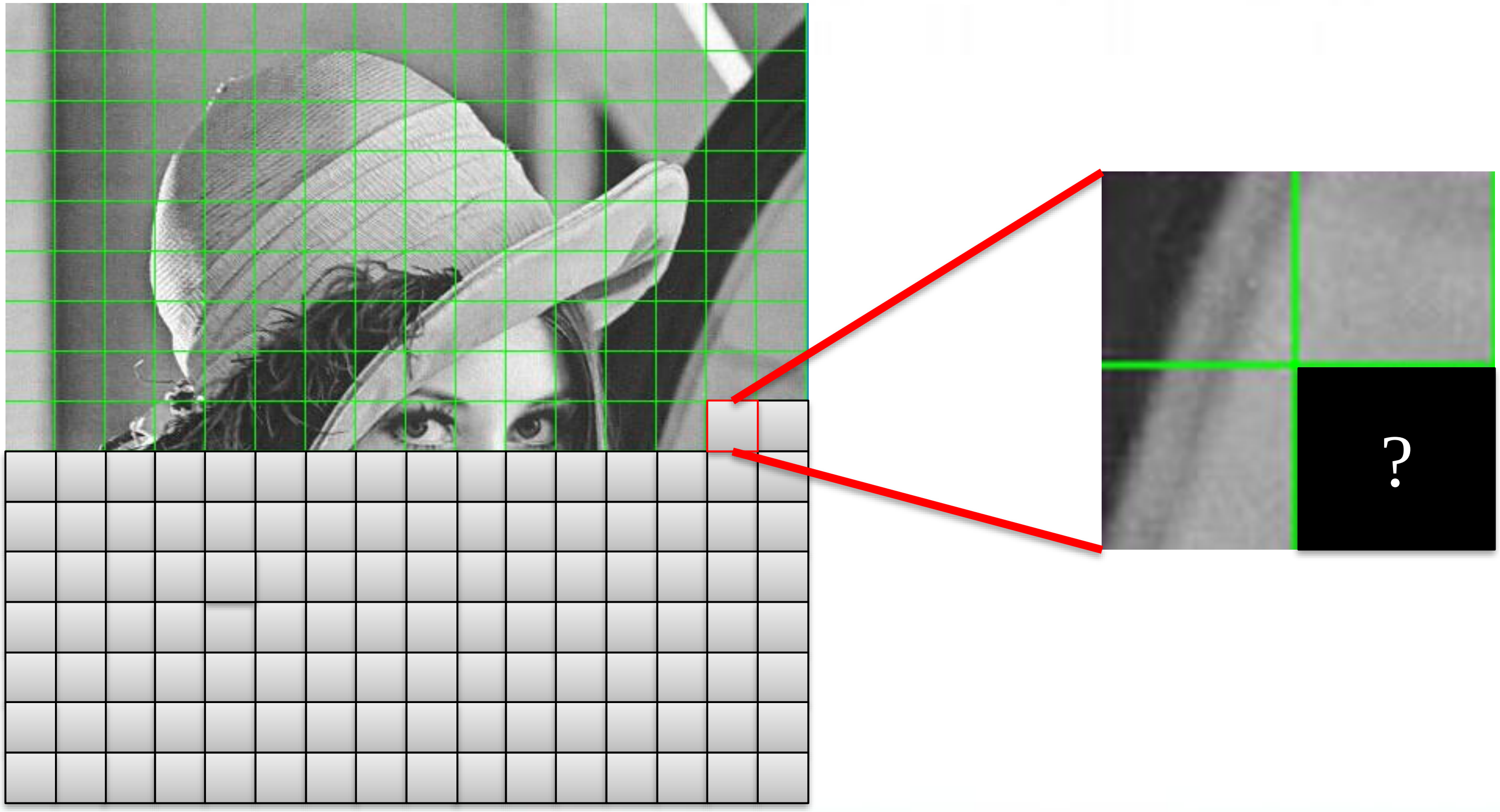


Intra Prediction



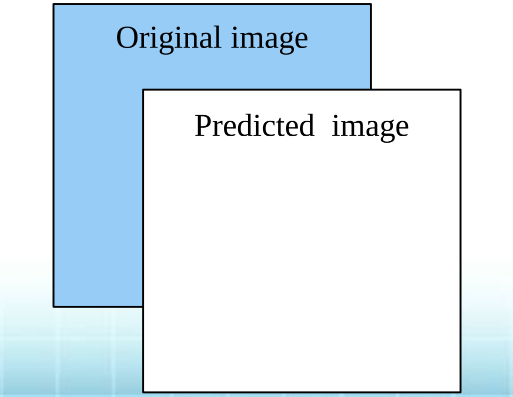


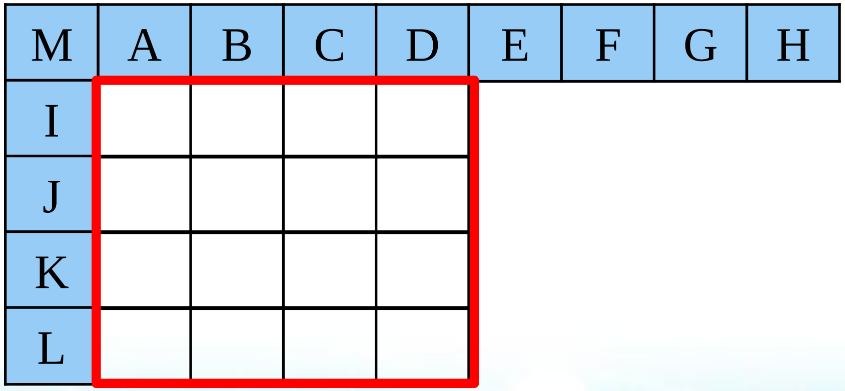
Intra Prediction



Make predictors

- Block size : 4x4
- 🟦 : reference sample values at an original image
  - If reference sample is NOT available, pad a reference sample value with 128
- □ : predictor sample values





M	A	B	C	D	E	F	G	H
I								
J								
K								
L								

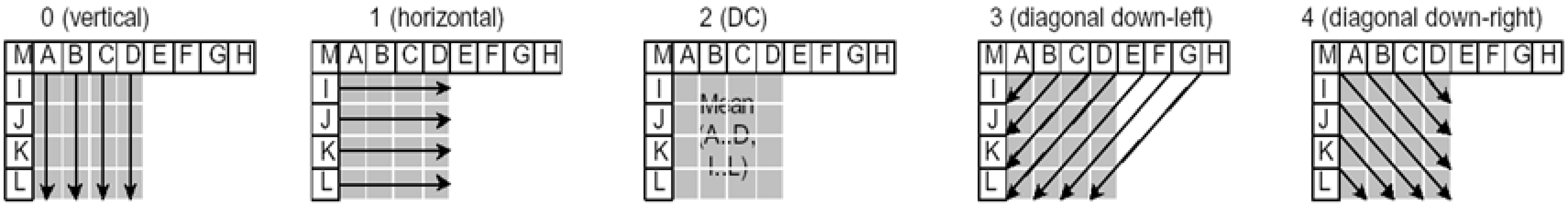
Make predictors

- Diagonal Left Down

M	A	B	C	D	E	F	G	H
I	B	C	D	E				
J	C	D	E	F				
K	D	E	F	G				
L	E	F	G	H				

- Diagonal Right Down

M	A	B	C	D
I	M	A	B	C
J	I	M	A	B
K	J	I	M	A
L	K	J	I	M



Best mode selection

- Find the best mode using Sum of Absolute Difference (SAD)

$$o_{i,j} \quad , \quad p^{0}_{i,j} \quad \cdots \quad p^{N}_{i,j}$$

$$\text{bestmode} = \arg\min_{k}\left(\sum_{j=0}^{M}\sum_{i=0}^{M} | o(i,j) - pred_{k}(i,j) |\right)$$

i, j : pixel position

k : prediction mode number

$c_{i,j}$: pixel value of the current block

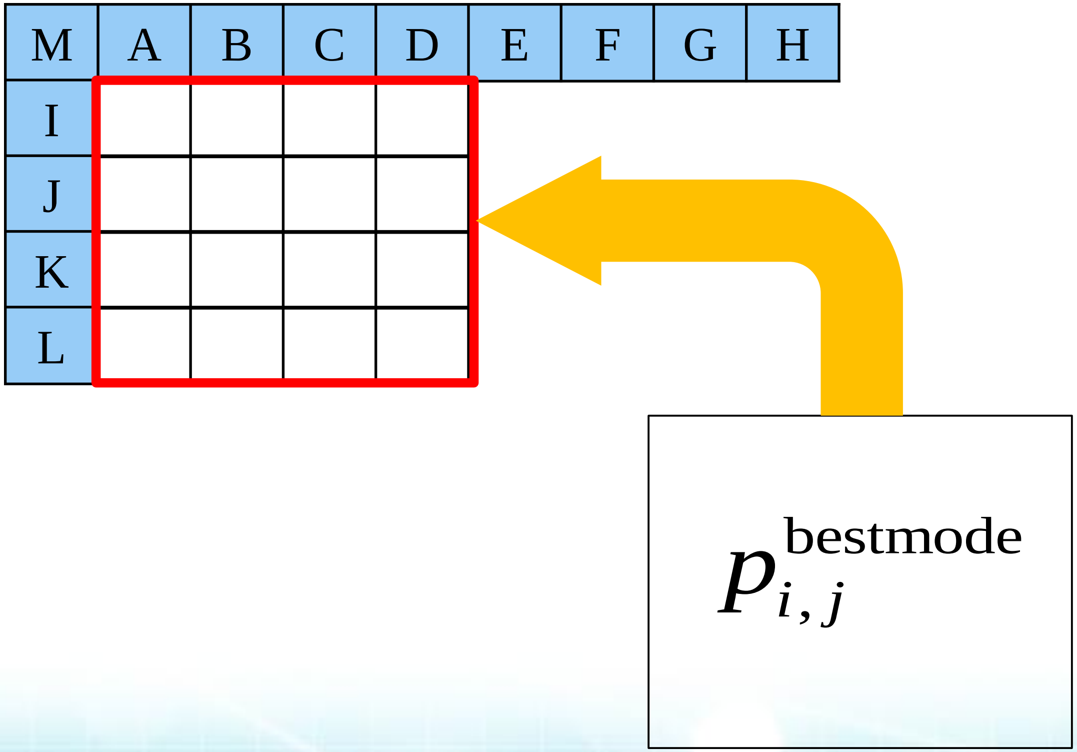
$p^{k}_{i,j}$: pixel value of the predictor mode k

N : the number of mode

M : block size

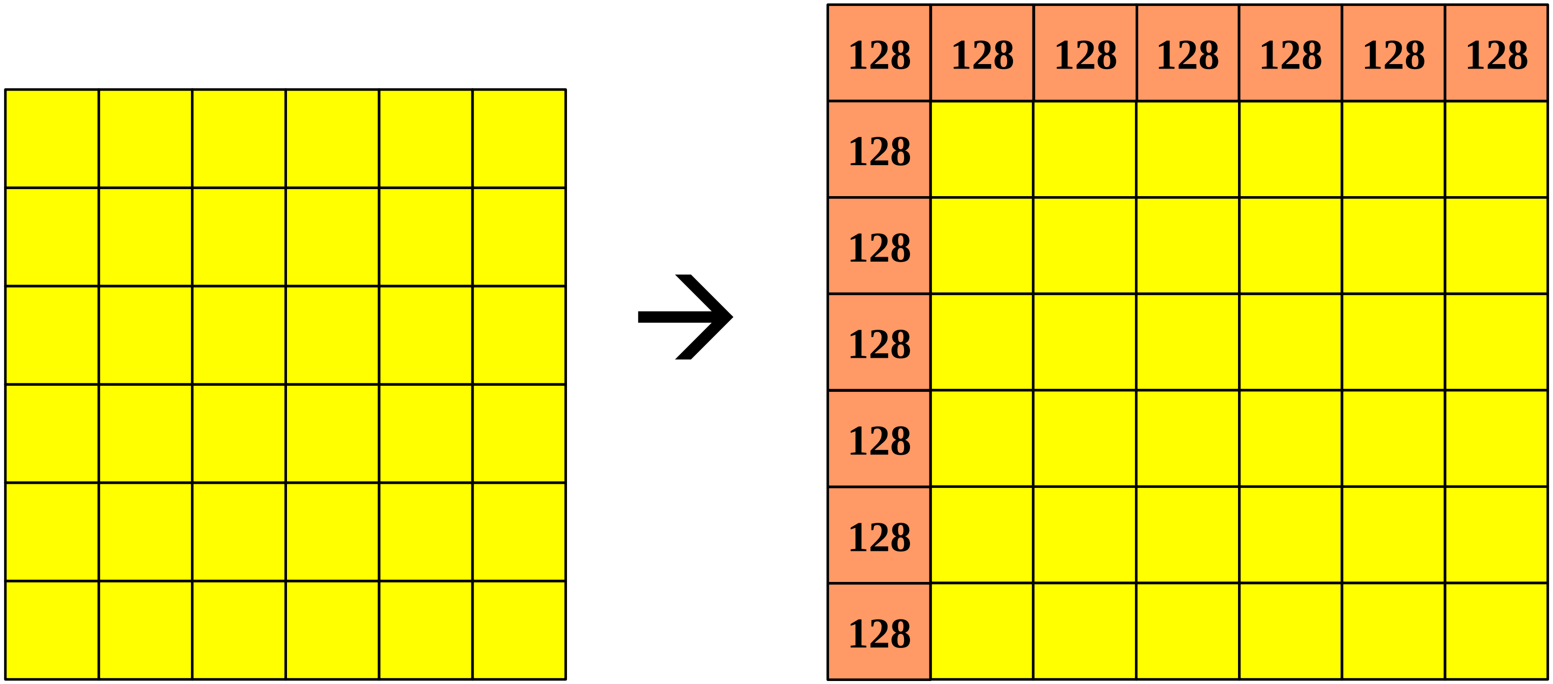
Reconstruction

- Make the new image using the predictor which is best mode



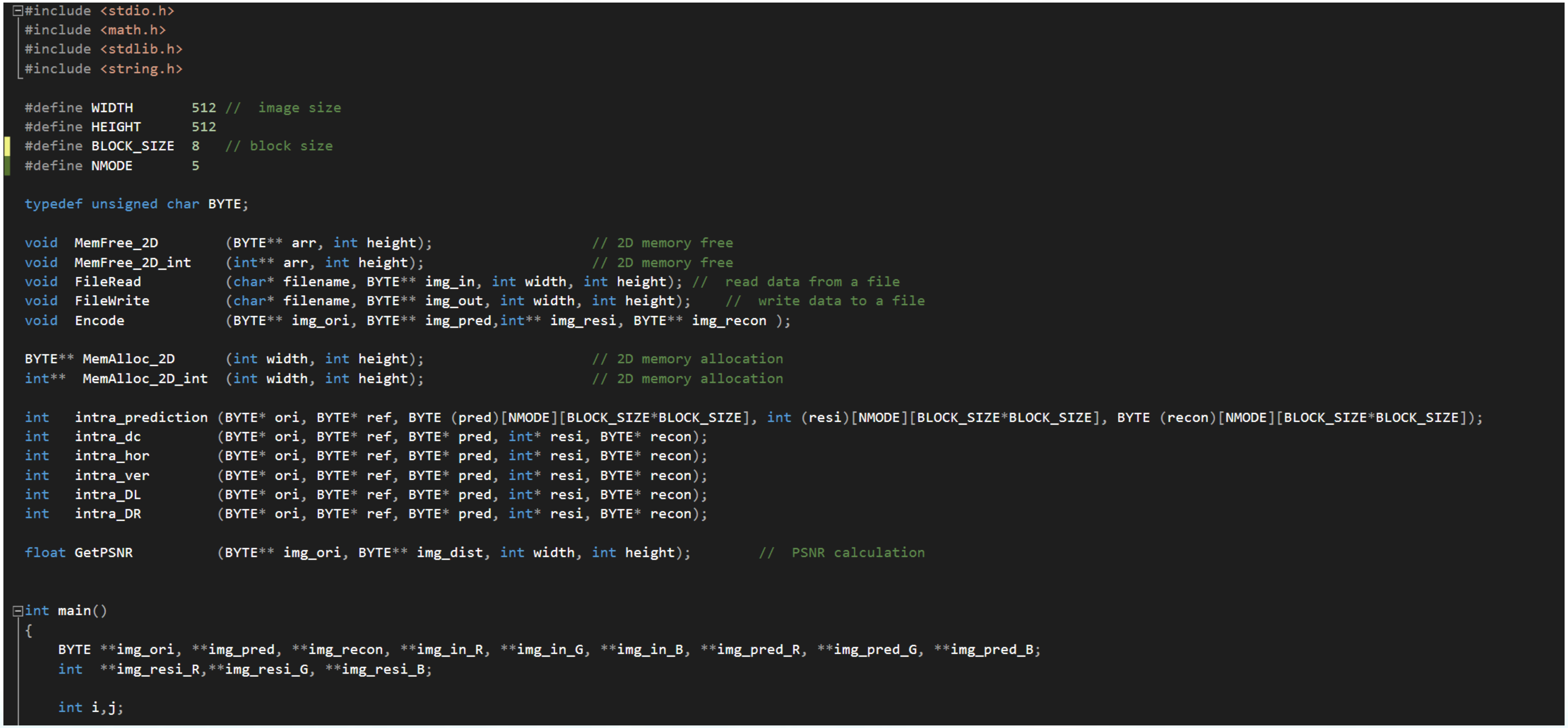
Guideline

- Padding the image
  - 8bit → 128



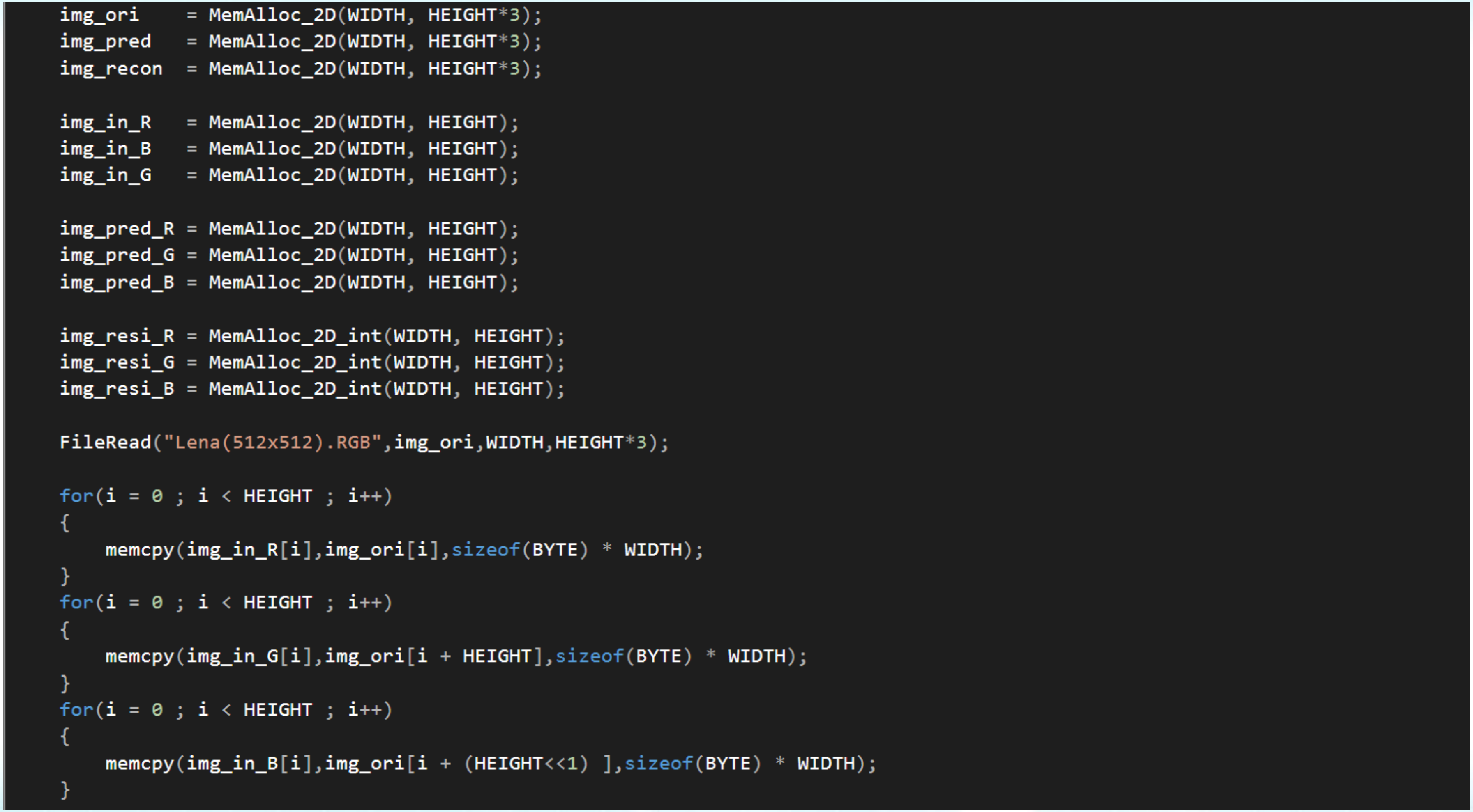
128	128	128	128	128	128	128
128						
128						
128						
128						
128						
128						

Programming Guide



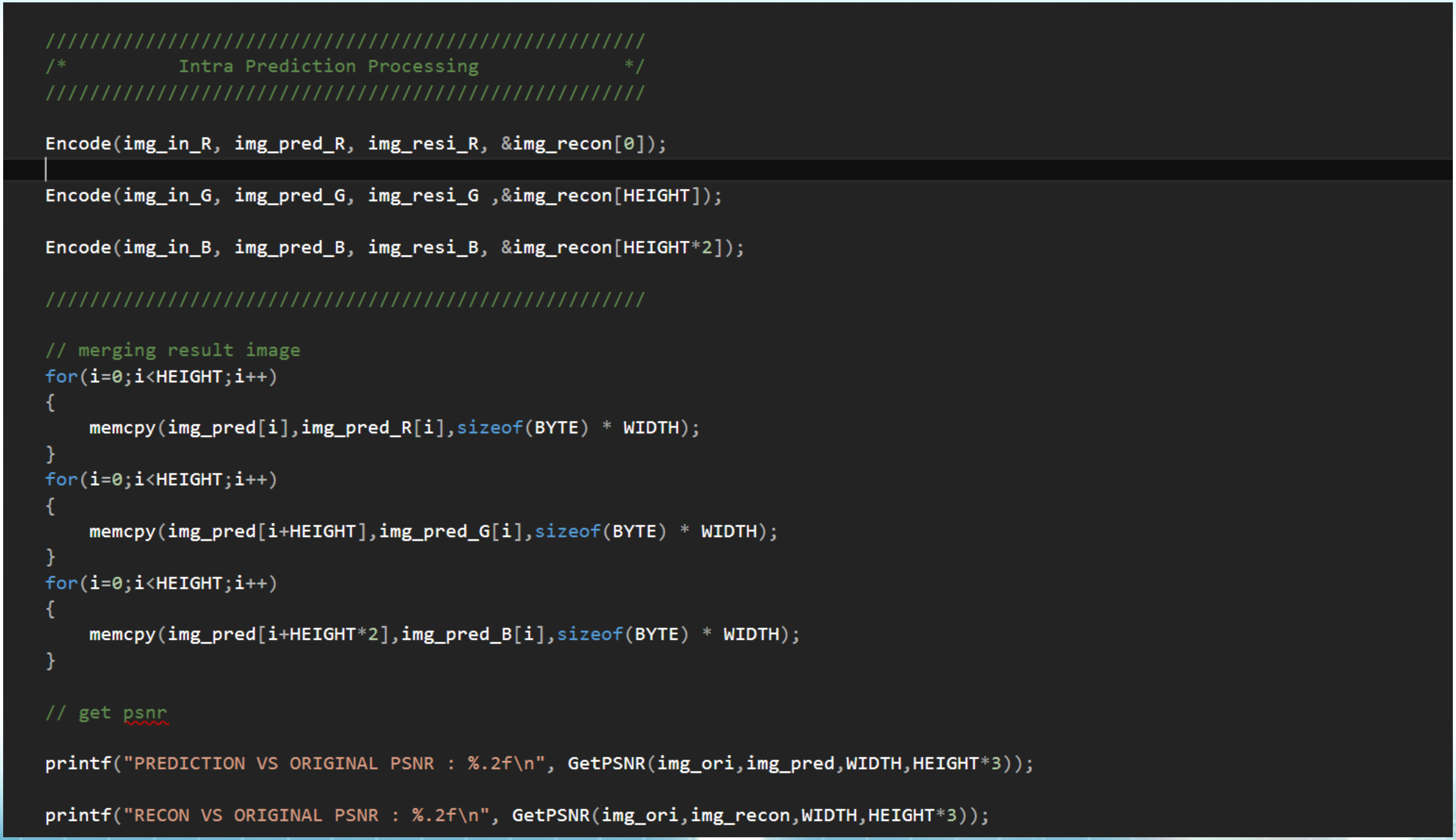
```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

#define WIDTH       512 //  image size
#define HEIGHT      512
#define BLOCK_SIZE  8   // block size
#define NMODE       5

typedef unsigned char BYTE;

void  MemFree_2D        (BYTE** arr, int height);                   // 2D memory free
void  MemFree_2D_int    (int** arr, int height);                    // 2D memory free
void  FileRead          (char* filename, BYTE** img_in, int width, int height); //  read data from a file
void  FileWrite         (char* filename, BYTE** img_out, int width, int height);    //  write data to a file
void  Encode            (BYTE** img_ori, BYTE** img_pred,int** img_resi, BYTE** img_recon );

BYTE** MemAlloc_2D      (int width, int height);                    // 2D memory allocation
int**  MemAlloc_2D_int  (int width, int height);                    // 2D memory allocation

int   intra_prediction (BYTE* ori, BYTE* ref, BYTE (pred)[NMODE][BLOCK_SIZE*BLOCK_SIZE], int (resi)[NMODE][BLOCK_SIZE*BLOCK_SIZE], BYTE (recon)[NMODE][BLOCK_SIZE*BLOCK_SIZE]);
int   intra_dc         (BYTE* ori, BYTE* ref, BYTE* pred, int* resi, BYTE* recon);
int   intra_hor        (BYTE* ori, BYTE* ref, BYTE* pred, int* resi, BYTE* recon);
int   intra_ver        (BYTE* ori, BYTE* ref, BYTE* pred, int* resi, BYTE* recon);
int   intra_DL         (BYTE* ori, BYTE* ref, BYTE* pred, int* resi, BYTE* recon);
int   intra_DR         (BYTE* ori, BYTE* ref, BYTE* pred, int* resi, BYTE* recon);

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


int main()
{
    BYTE **img_ori, **img_pred, **img_recon, **img_in_R, **img_in_G, **img_in_B, **img_pred_R, **img_pred_G, **img_pred_B;
    int  **img_resi_R,**img_resi_G, **img_resi_B;

    int i,j;
```

Programming Guide



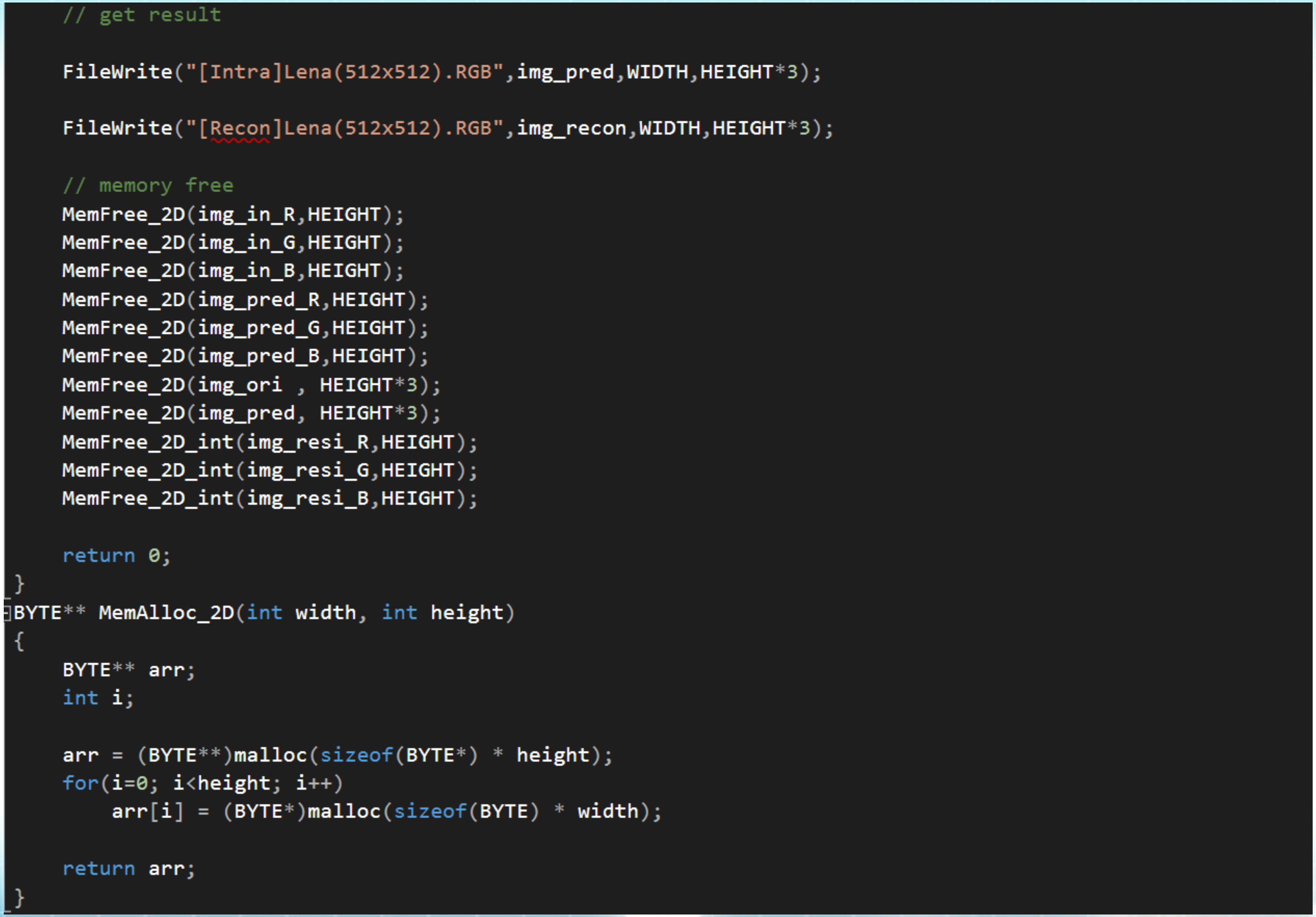
```
    img_ori    = MemAlloc_2D(WIDTH, HEIGHT*3);
    img_pred   = MemAlloc_2D(WIDTH, HEIGHT*3);
    img_recon  = MemAlloc_2D(WIDTH, HEIGHT*3);

    img_in_R   = MemAlloc_2D(WIDTH, HEIGHT);
    img_in_B   = MemAlloc_2D(WIDTH, HEIGHT);
    img_in_G   = MemAlloc_2D(WIDTH, HEIGHT);

    img_pred_R = MemAlloc_2D(WIDTH, HEIGHT);
    img_pred_G = MemAlloc_2D(WIDTH, HEIGHT);
    img_pred_B = MemAlloc_2D(WIDTH, HEIGHT);

    img_resi_R = MemAlloc_2D_int(WIDTH, HEIGHT);
    img_resi_G = MemAlloc_2D_int(WIDTH, HEIGHT);
    img_resi_B = MemAlloc_2D_int(WIDTH, HEIGHT);

    FileRead("Lena(512x512).RGB",img_ori,WIDTH,HEIGHT*3);

    for(i = 0 ; i < HEIGHT ; i++)
    {
        memcpy(img_in_R[i],img_ori[i],sizeof(BYTE) * WIDTH);
    }
    for(i = 0 ; i < HEIGHT ; i++)
    {
        memcpy(img_in_G[i],img_ori[i + HEIGHT],sizeof(BYTE) * WIDTH);
    }
    for(i = 0 ; i < HEIGHT ; i++)
    {
        memcpy(img_in_B[i],img_ori[i + (HEIGHT<<1) ],sizeof(BYTE) * WIDTH);
    }
```

Programming Guide



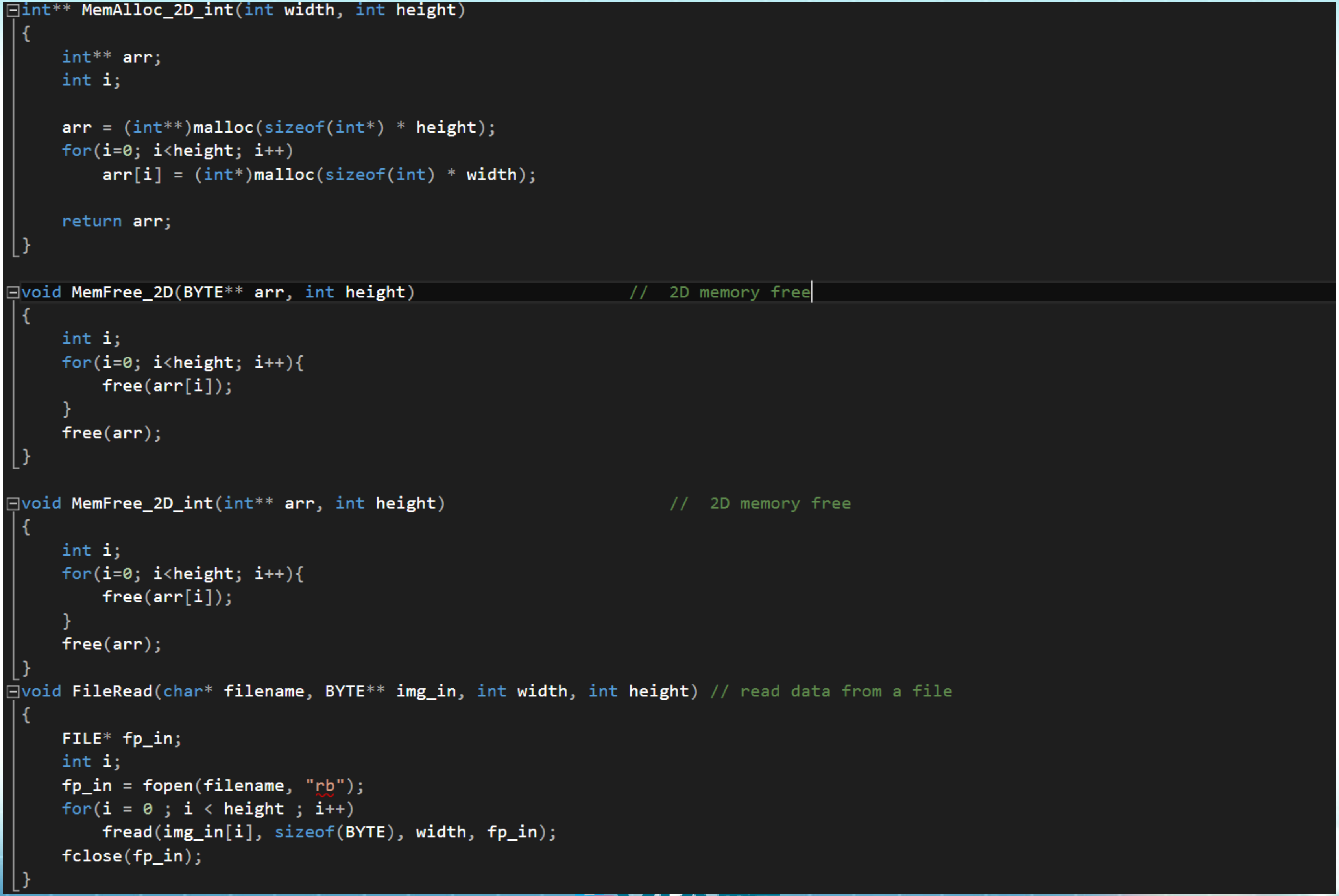
```
//////////////////////////////////////////////////////
/*          Intra Prediction Processing             */
//////////////////////////////////////////////////////

Encode(img_in_R, img_pred_R, img_resi_R, &img_recon[0]);

Encode(img_in_G, img_pred_G, img_resi_G ,&img_recon[HEIGHT]);

Encode(img_in_B, img_pred_B, img_resi_B, &img_recon[HEIGHT*2]);

//////////////////////////////////////////////////////

// merging result image
for(i=0;i<HEIGHT;i++)
{
    memcpy(img_pred[i],img_pred_R[i],sizeof(BYTE) * WIDTH);
}
for(i=0;i<HEIGHT;i++)
{
    memcpy(img_pred[i+HEIGHT],img_pred_G[i],sizeof(BYTE) * WIDTH);
}
for(i=0;i<HEIGHT;i++)
{
    memcpy(img_pred[i+HEIGHT*2],img_pred_B[i],sizeof(BYTE) * WIDTH);
}

// get psnr

printf("PREDICTION VS ORIGINAL PSNR : %.2f\n", GetPSNR(img_ori,img_pred,WIDTH,HEIGHT*3));

printf("RECON VS ORIGINAL PSNR : %.2f\n", GetPSNR(img_ori,img_recon,WIDTH,HEIGHT*3));
```

Programming Guide



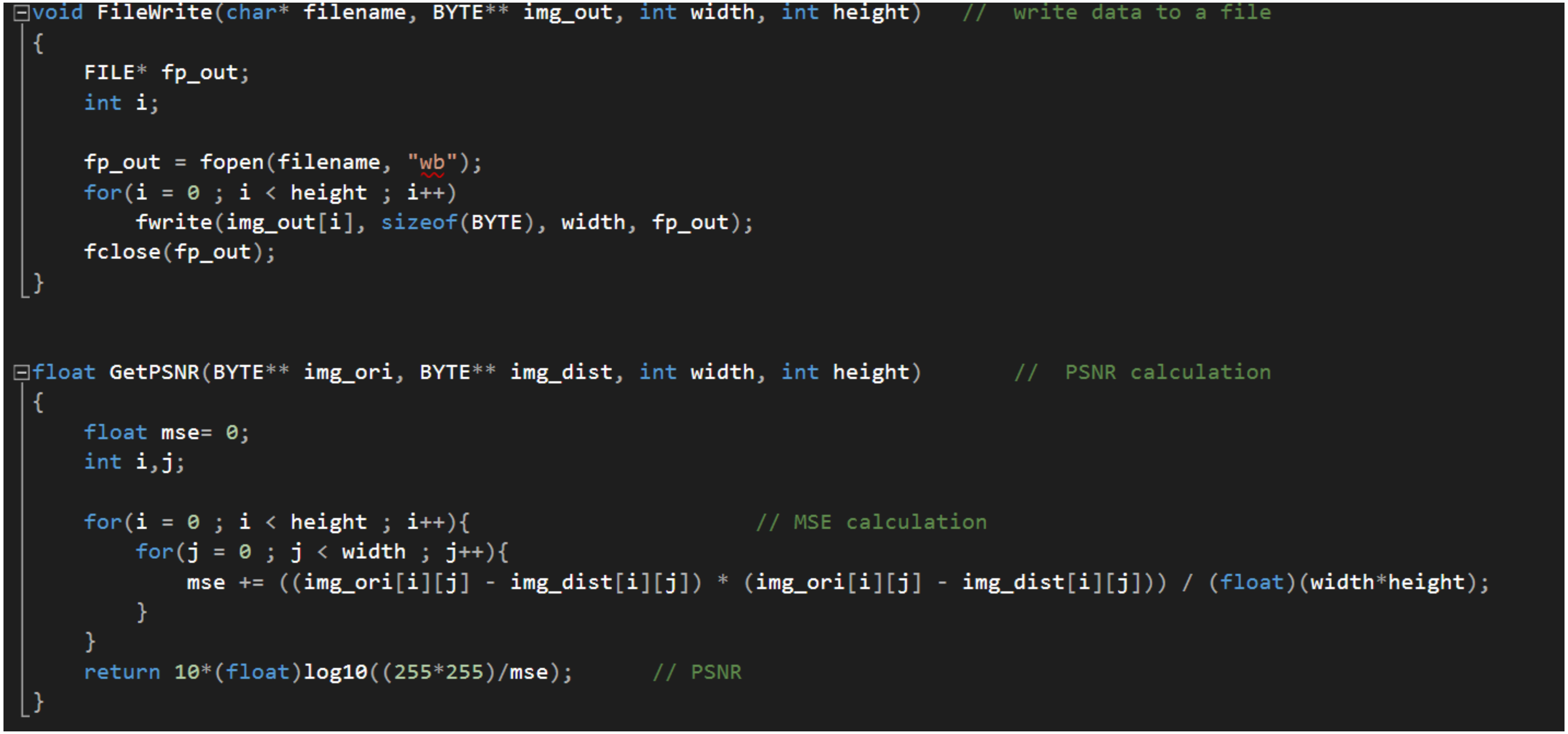
```c
    // get result

    FileWrite("[Intra]Lena(512x512).RGB",img_pred,WIDTH,HEIGHT*3);

    FileWrite("[Recon]Lena(512x512).RGB",img_recon,WIDTH,HEIGHT*3);

    // memory free
    MemFree_2D(img_in_R,HEIGHT);
    MemFree_2D(img_in_G,HEIGHT);
    MemFree_2D(img_in_B,HEIGHT);
    MemFree_2D(img_pred_R,HEIGHT);
    MemFree_2D(img_pred_G,HEIGHT);
    MemFree_2D(img_pred_B,HEIGHT);
    MemFree_2D(img_ori , HEIGHT*3);
    MemFree_2D(img_pred, HEIGHT*3);
    MemFree_2D_int(img_resi_R,HEIGHT);
    MemFree_2D_int(img_resi_G,HEIGHT);
    MemFree_2D_int(img_resi_B,HEIGHT);

    return 0;
}
BYTE** MemAlloc_2D(int width, int height)
{
    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;
}
```

Programming Guide



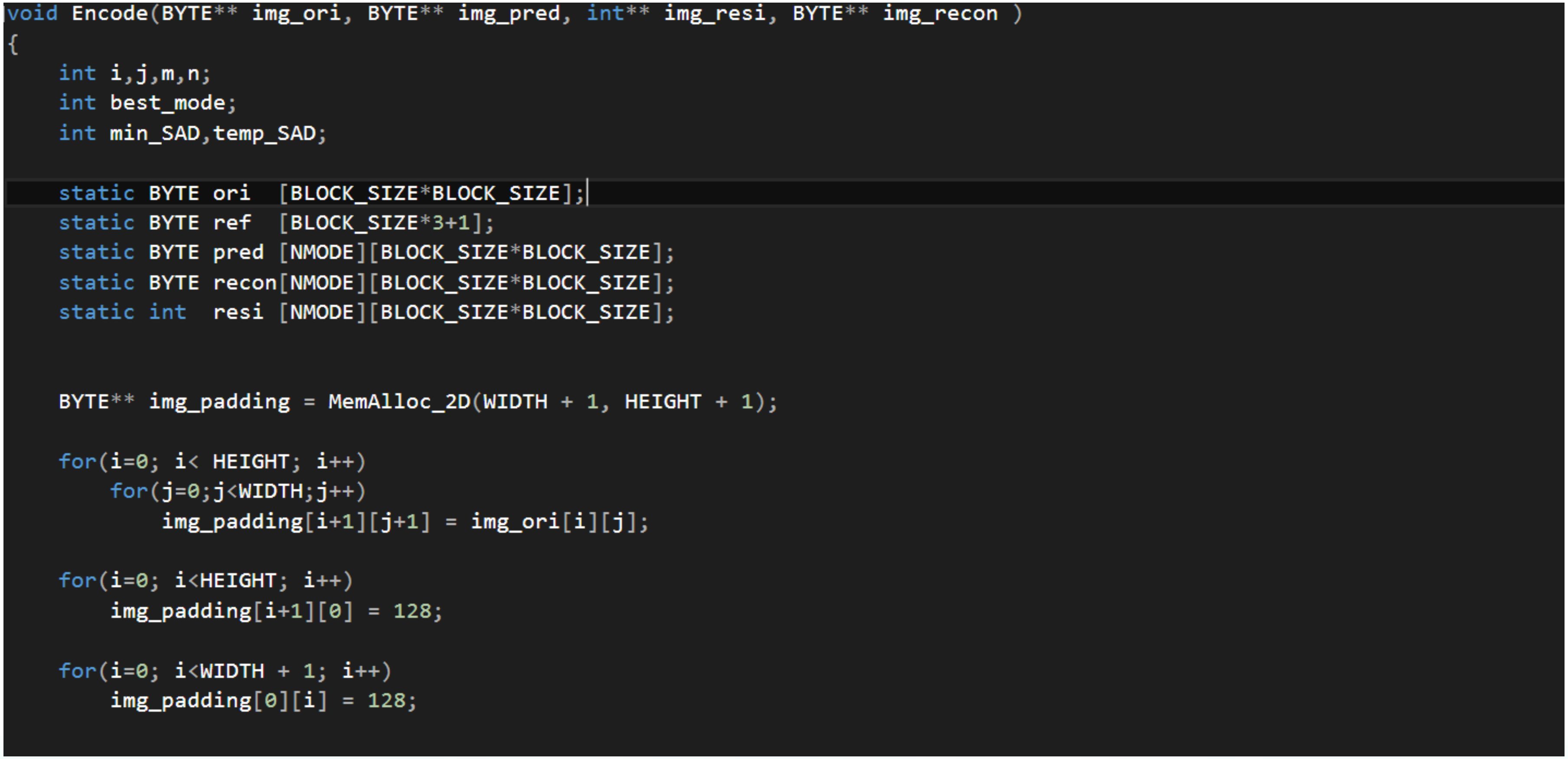
```
int** MemAlloc_2D_int(int width, int height)
{
    int** arr;
    int i;

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

void MemFree_2D_int(int** arr, int height)                    //  2D memory free
{
    int i;
    for(i=0; i<height; i++){
        free(arr[i]);
    }
    free(arr);
}
void FileRead(char* filename, BYTE** img_in, int width, int height) // read data from a file
{
    FILE* fp_in;
    int i;
    fp_in = fopen(filename, "rb");
    for(i = 0 ; i < height ; i++)
        fread(img_in[i], sizeof(BYTE), width, fp_in);
    fclose(fp_in);
}
```

Programming Guide



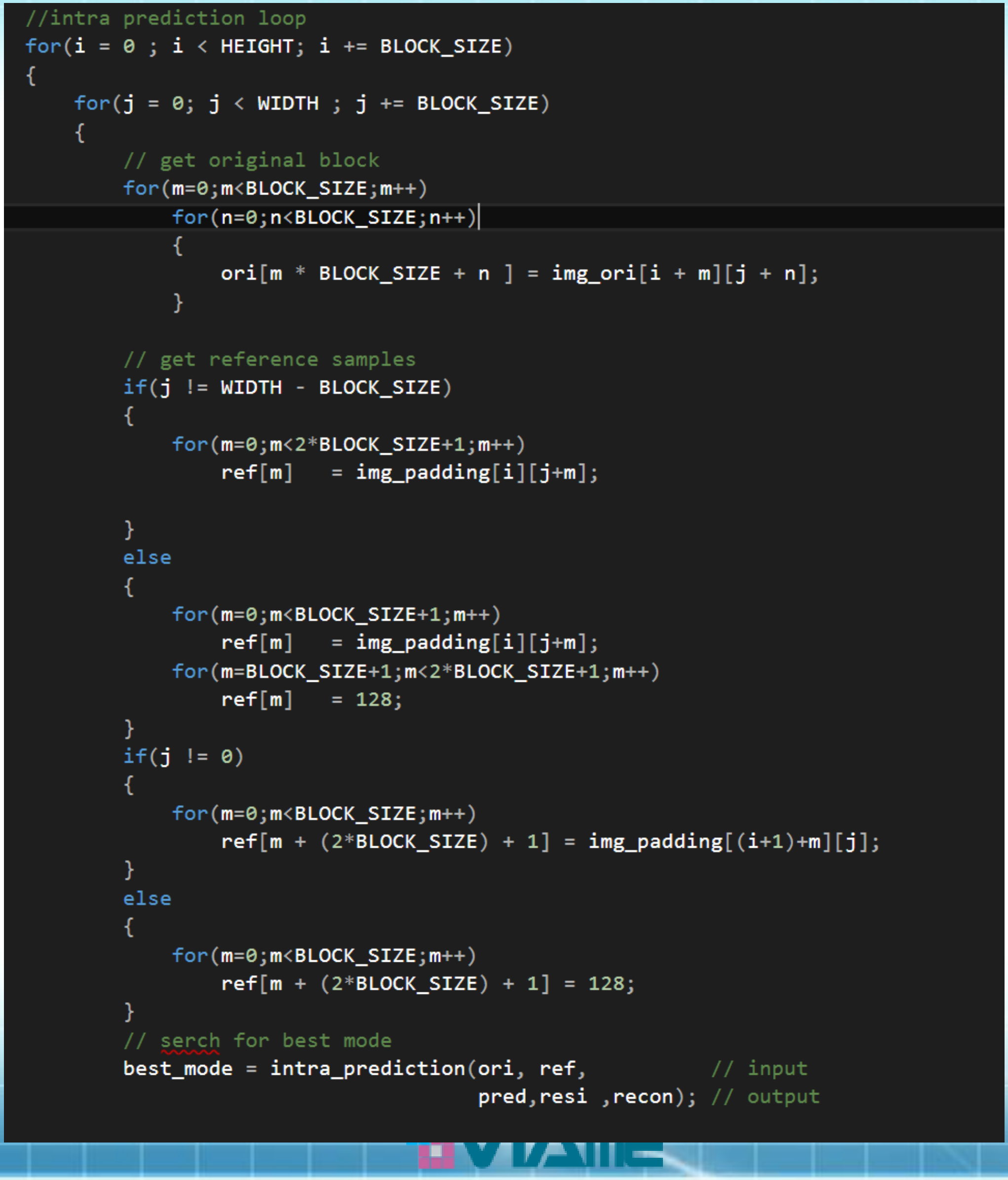
```
void FileWrite(char* filename, BYTE** img_out, int width, int height)   //  write data to a file
{
    FILE* fp_out;
    int i;

    fp_out = fopen(filename, "wb");
    for(i = 0 ; i < height ; i++)
        fwrite(img_out[i], sizeof(BYTE), width, fp_out);
    fclose(fp_out);
}


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
}
```

Programming Guide



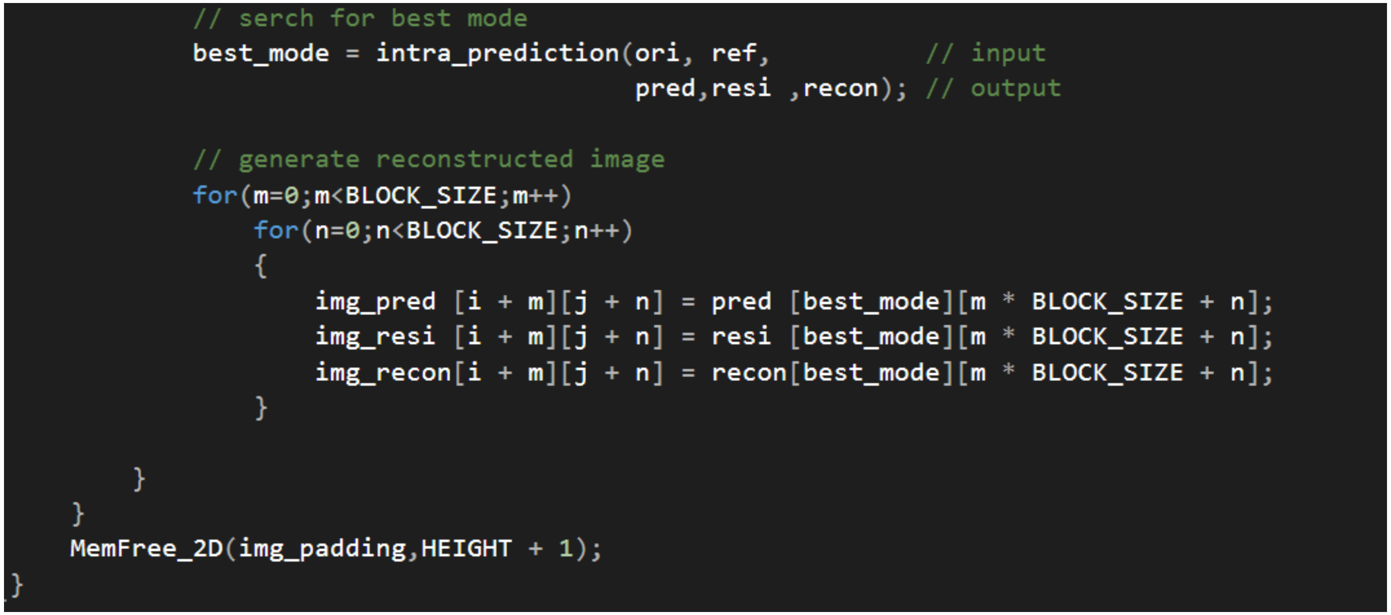
```
void Encode(BYTE** img_ori, BYTE** img_pred, int** img_resi, BYTE** img_recon )
{
    int i,j,m,n;
    int best_mode;
    int min_SAD,temp_SAD;

    static BYTE ori  [BLOCK_SIZE*BLOCK_SIZE];
    static BYTE ref  [BLOCK_SIZE*3+1];
    static BYTE pred [NMODE][BLOCK_SIZE*BLOCK_SIZE];
    static BYTE recon[NMODE][BLOCK_SIZE*BLOCK_SIZE];
    static int  resi [NMODE][BLOCK_SIZE*BLOCK_SIZE];


    BYTE** img_padding = MemAlloc_2D(WIDTH + 1, HEIGHT + 1);

    for(i=0; i< HEIGHT; i++)
        for(j=0;j<WIDTH;j++)
            img_padding[i+1][j+1] = img_ori[i][j];

    for(i=0; i<HEIGHT; i++)
        img_padding[i+1][0] = 128;

    for(i=0; i<WIDTH + 1; i++)
        img_padding[0][i] = 128;
```

Programming Guide



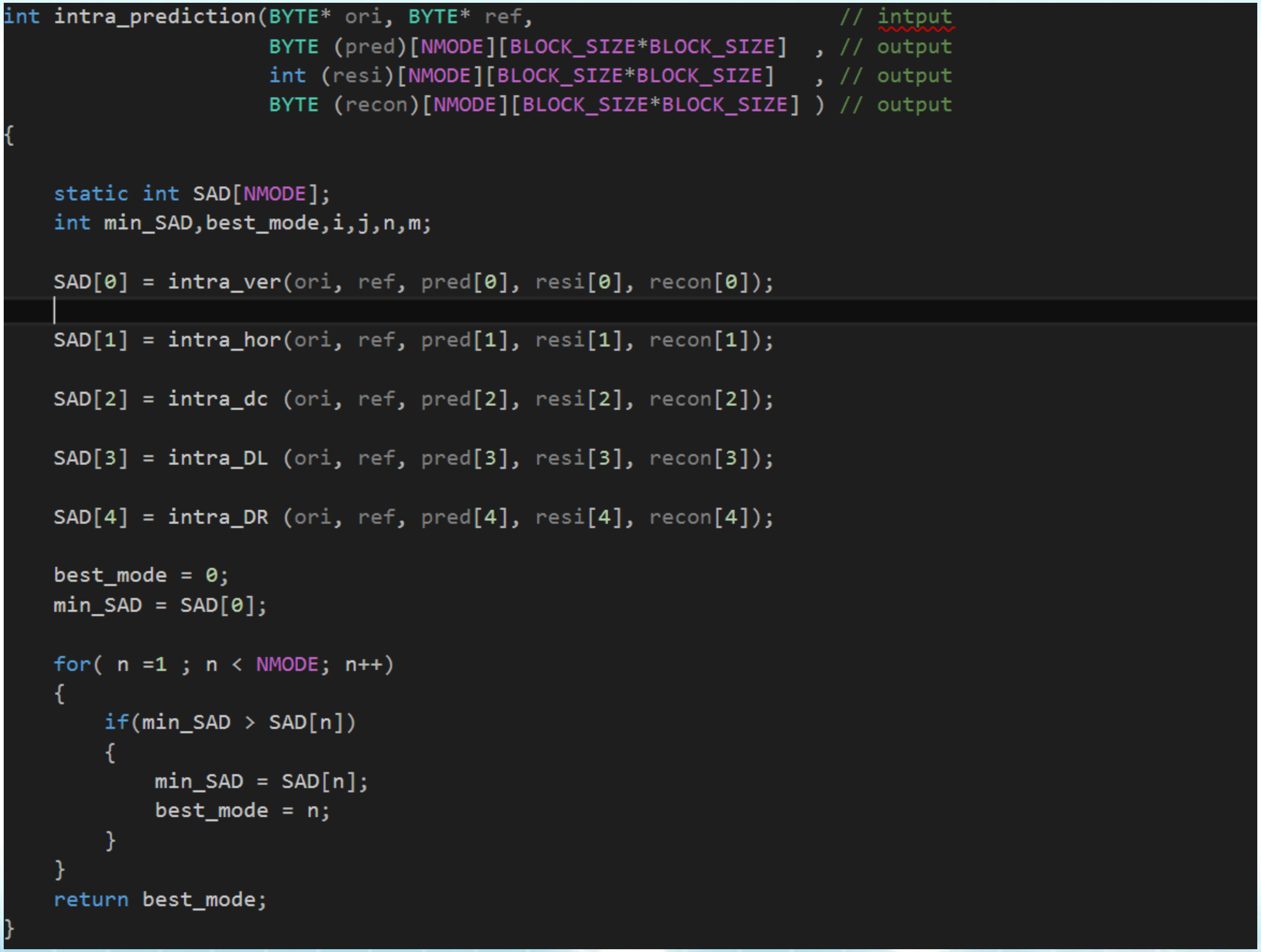
```
//intra prediction loop
for(i = 0 ; i < HEIGHT; i += BLOCK_SIZE)
{
    for(j = 0; j < WIDTH ; j += BLOCK_SIZE)
    {
        // get original block
        for(m=0;m<BLOCK_SIZE;m++)
            for(n=0;n<BLOCK_SIZE;n++)
            {
                ori[m * BLOCK_SIZE + n ] = img_ori[i + m][j + n];
            }

        // get reference samples
        if(j != WIDTH - BLOCK_SIZE)
        {
            for(m=0;m<2*BLOCK_SIZE+1;m++)
                ref[m]   = img_padding[i][j+m];

        }
        else
        {
            for(m=0;m<BLOCK_SIZE+1;m++)
                ref[m]   = img_padding[i][j+m];
            for(m=BLOCK_SIZE+1;m<2*BLOCK_SIZE+1;m++)
                ref[m]   = 128;
        }
        if(j != 0)
        {
            for(m=0;m<BLOCK_SIZE;m++)
                ref[m + (2*BLOCK_SIZE) + 1] = img_padding[(i+1)+m][j];
        }
        else
        {
            for(m=0;m<BLOCK_SIZE;m++)
                ref[m + (2*BLOCK_SIZE) + 1] = 128;
        }
        // serch for best mode
        best_mode = intra_prediction(ori, ref,          // input
                                     pred,resi ,recon); // output
```

Programming Guide



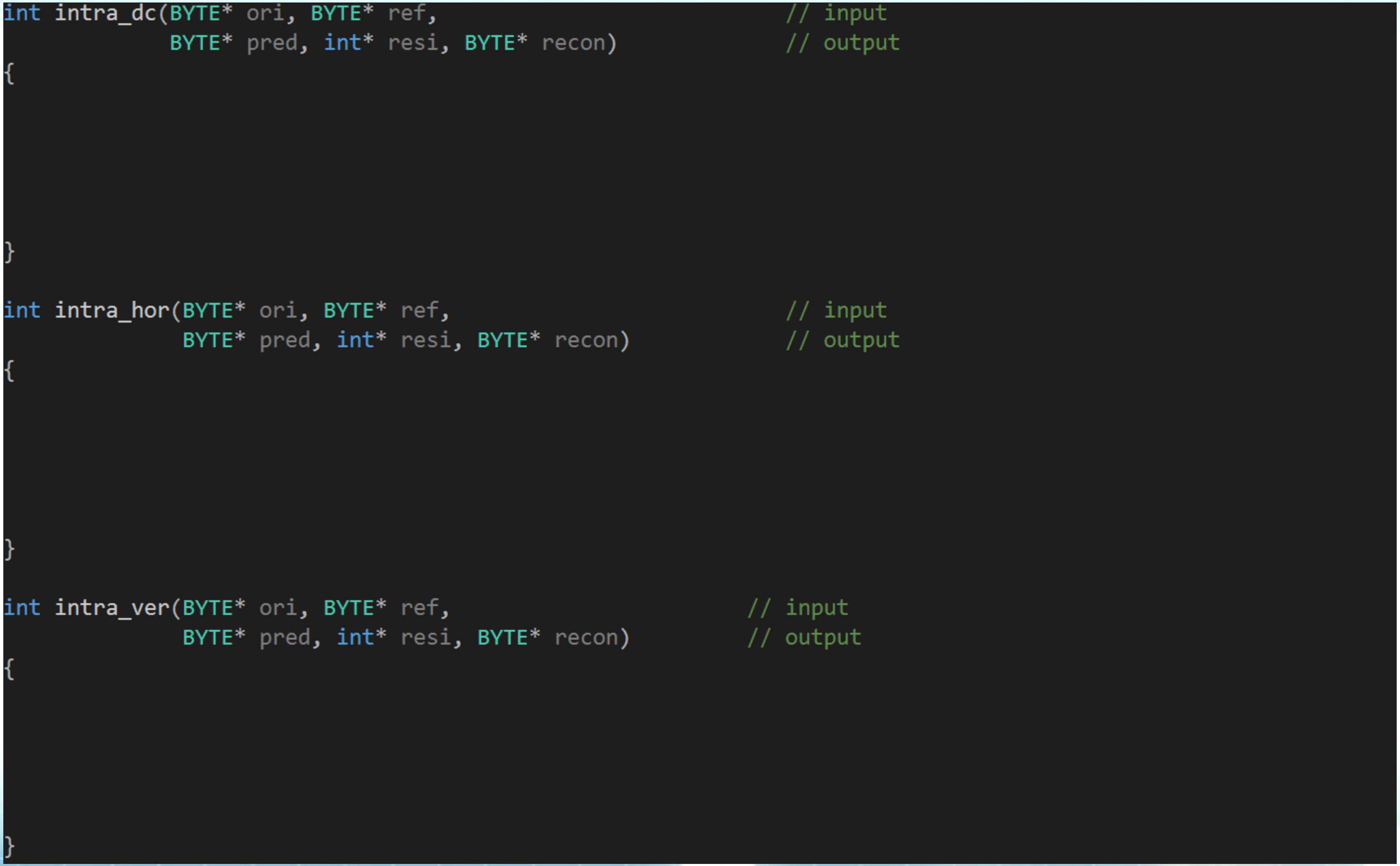
```
            // serch for best mode
            best_mode = intra_prediction(ori, ref,          // input
                                         pred,resi ,recon); // output

            // generate reconstructed image
            for(m=0;m<BLOCK_SIZE;m++)
                for(n=0;n<BLOCK_SIZE;n++)
                {
                    img_pred [i + m][j + n] = pred [best_mode][m * BLOCK_SIZE + n];
                    img_resi [i + m][j + n] = resi [best_mode][m * BLOCK_SIZE + n];
                    img_recon[i + m][j + n] = recon[best_mode][m * BLOCK_SIZE + n];
                }

        }
    }
    MemFree_2D(img_padding,HEIGHT + 1);
}
```

Programming Guide



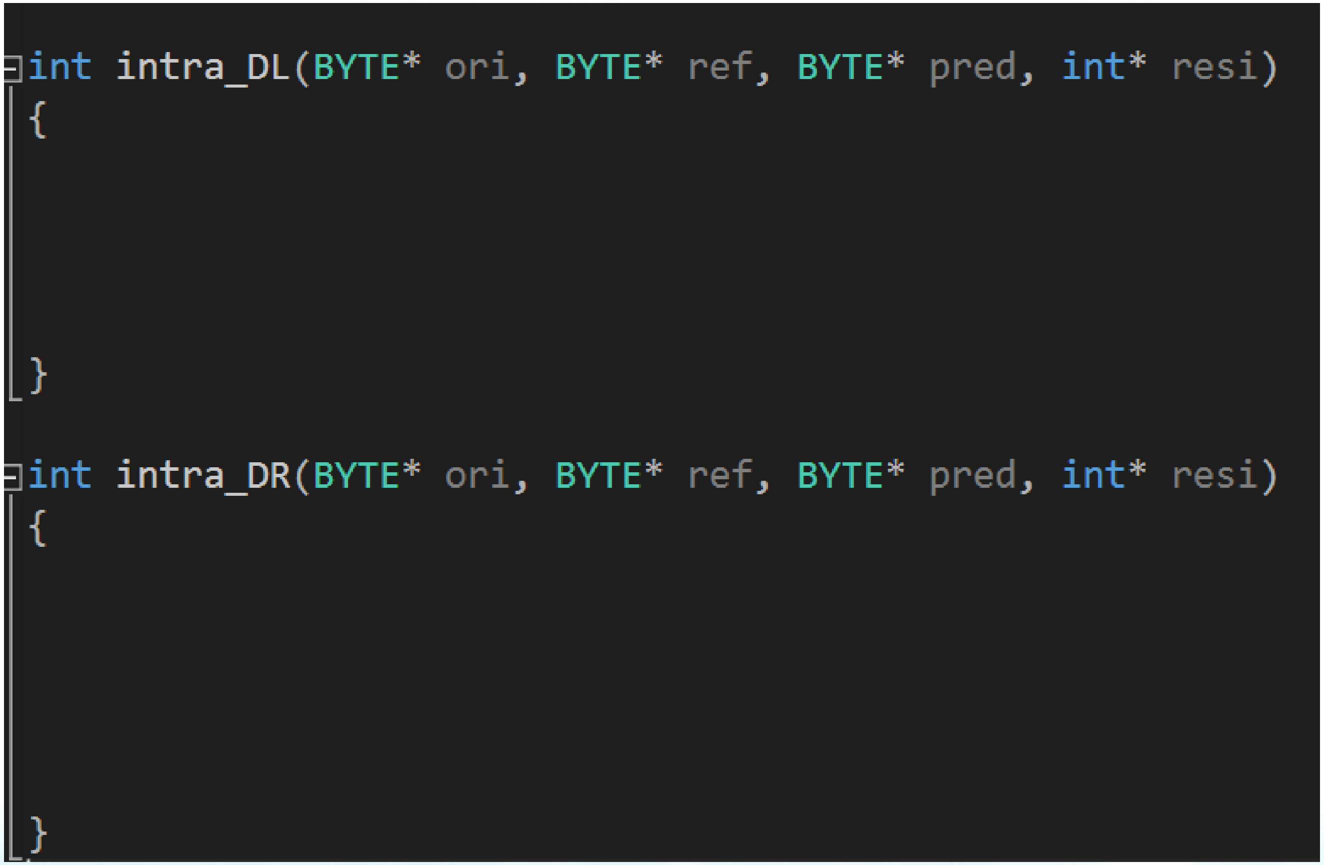
```
int intra_prediction(BYTE* ori, BYTE* ref,                             // intput
                     BYTE (pred)[NMODE][BLOCK_SIZE*BLOCK_SIZE]  , // output
                     int (resi)[NMODE][BLOCK_SIZE*BLOCK_SIZE]   , // output
                     BYTE (recon)[NMODE][BLOCK_SIZE*BLOCK_SIZE] ) // output
{

    static int SAD[NMODE];
    int min_SAD,best_mode,i,j,n,m;

    SAD[0] = intra_ver(ori, ref, pred[0], resi[0], recon[0]);

    SAD[1] = intra_hor(ori, ref, pred[1], resi[1], recon[1]);

    SAD[2] = intra_dc (ori, ref, pred[2], resi[2], recon[2]);

    SAD[3] = intra_DL (ori, ref, pred[3], resi[3], recon[3]);

    SAD[4] = intra_DR (ori, ref, pred[4], resi[4], recon[4]);

    best_mode = 0;
    min_SAD = SAD[0];

    for( n =1 ; n < NMODE; n++)
    {
        if(min_SAD > SAD[n])
        {
            min_SAD = SAD[n];
            best_mode = n;
        }
    }
    return best_mode;
}
```

Programming Guide



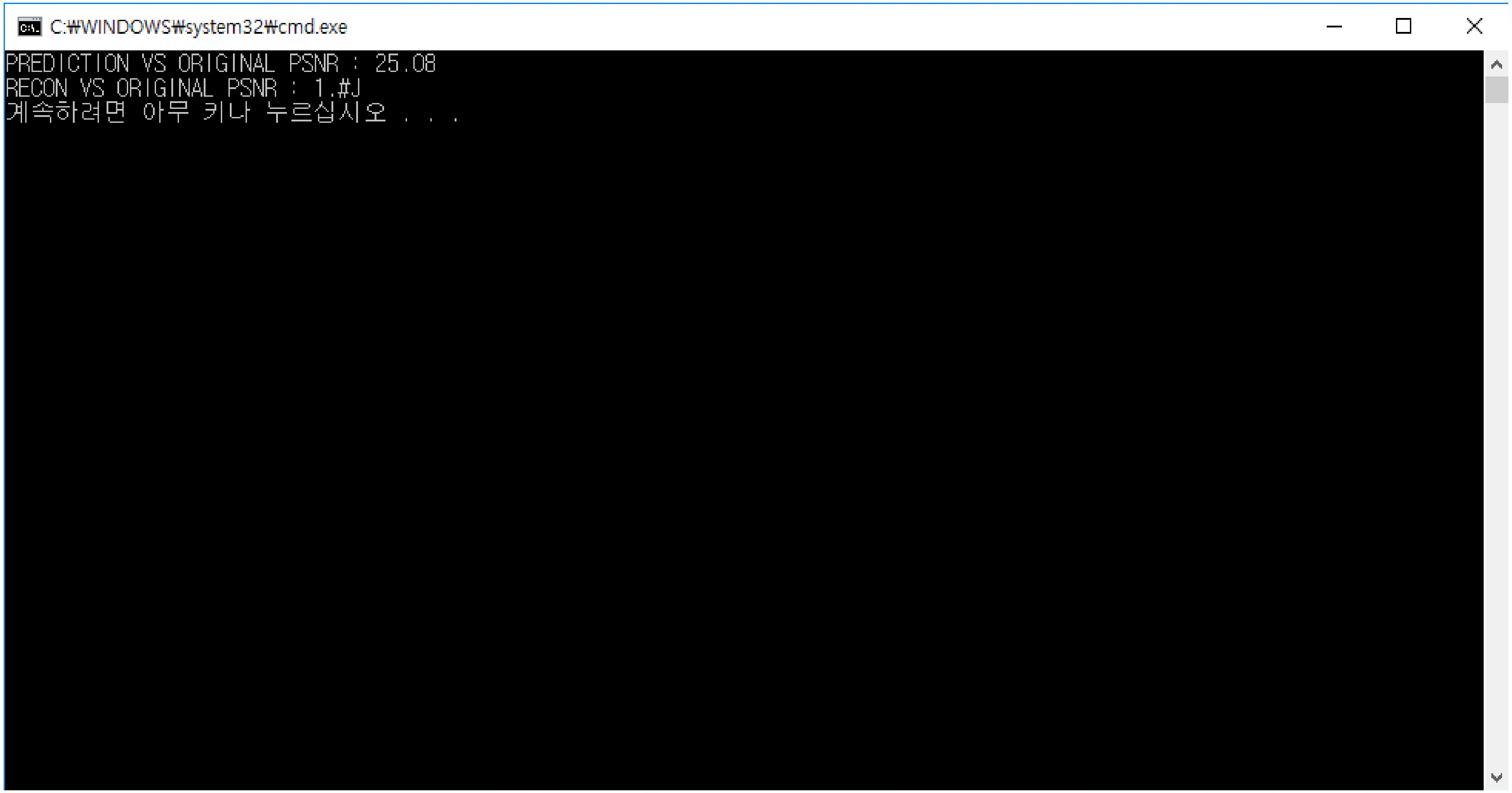
```
int intra_dc(BYTE* ori, BYTE* ref,                           // input
             BYTE* pred, int* resi, BYTE* recon)             // output
{





}

int intra_hor(BYTE* ori, BYTE* ref,                          // input
              BYTE* pred, int* resi, BYTE* recon)            // output
{





}

int intra_ver(BYTE* ori, BYTE* ref,                       // input
              BYTE* pred, int* resi, BYTE* recon)         // output
{





}
```

Programming Guide



```
int intra_DL(BYTE* ori, BYTE* ref, BYTE* pred, int* resi)
{





}

int intra_DR(BYTE* ori, BYTE* ref, BYTE* pred, int* resi)
{





}
```

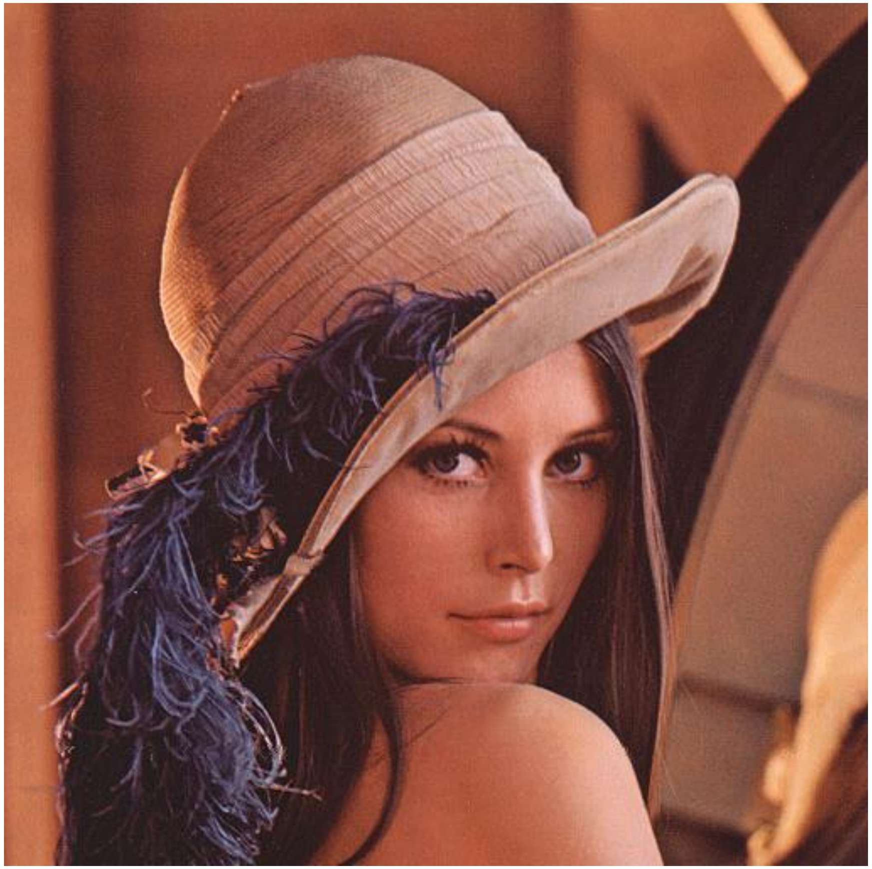
Result



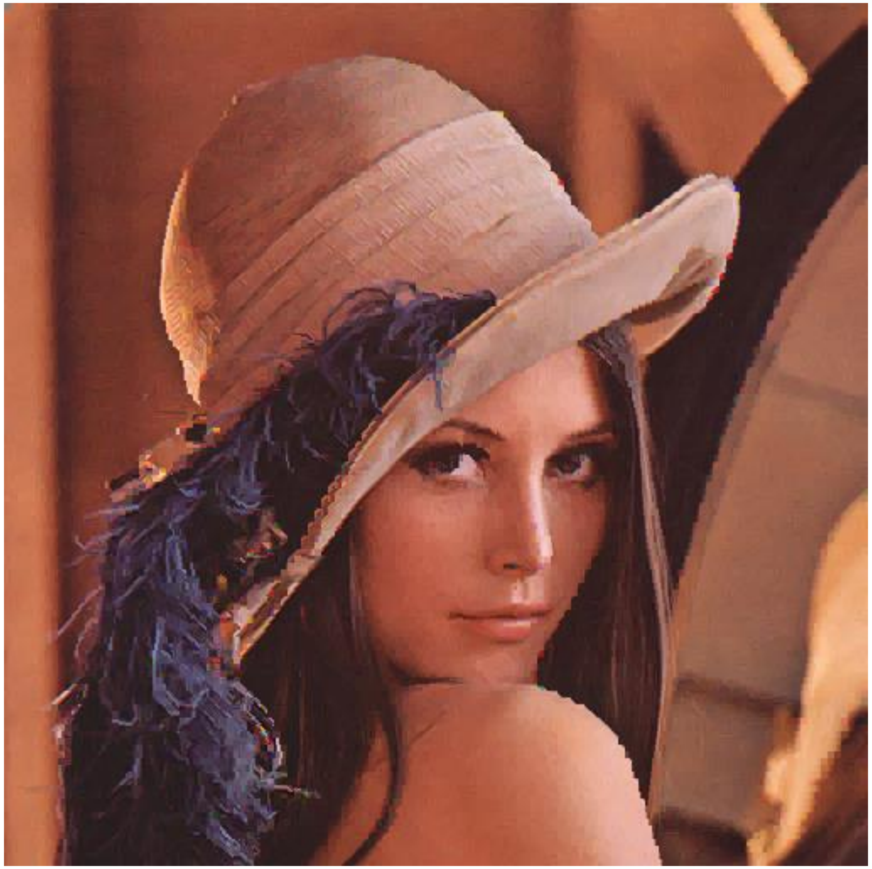
C:₩WINDOWS₩system32₩cmd.exe

```
PREDICTION VS ORIGINAL PSNR : 25.08
RECON VS ORIGINAL PSNR : 1.#J
계속하려면 아무 키나 누르십시오 . . .
```

Result

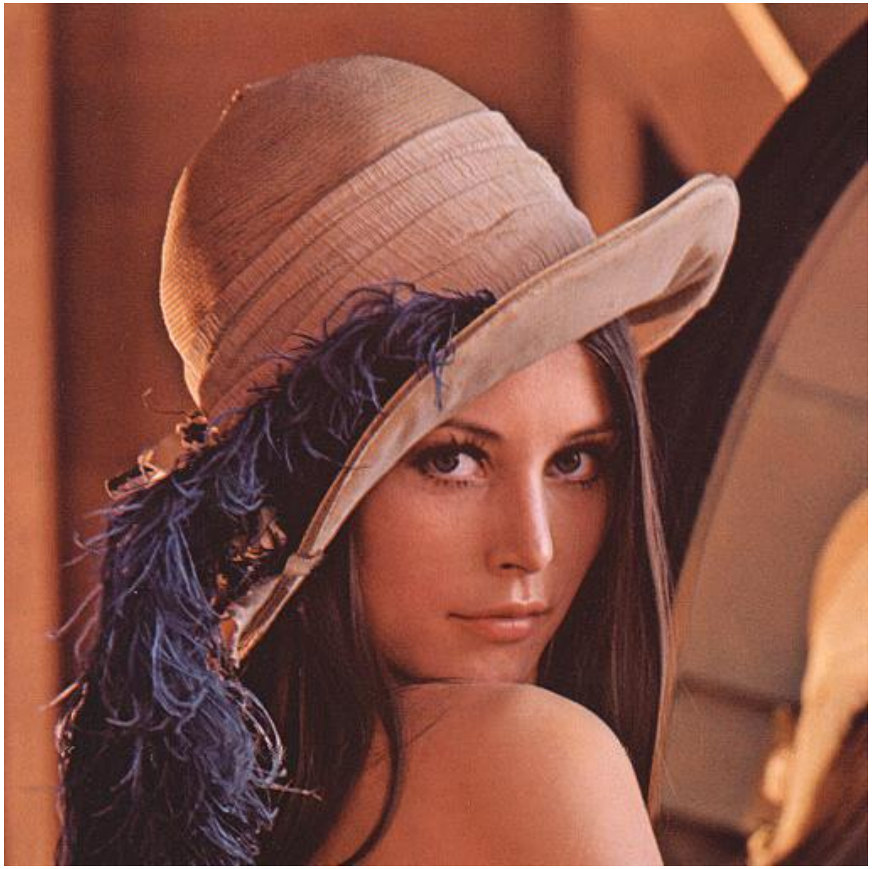


Original

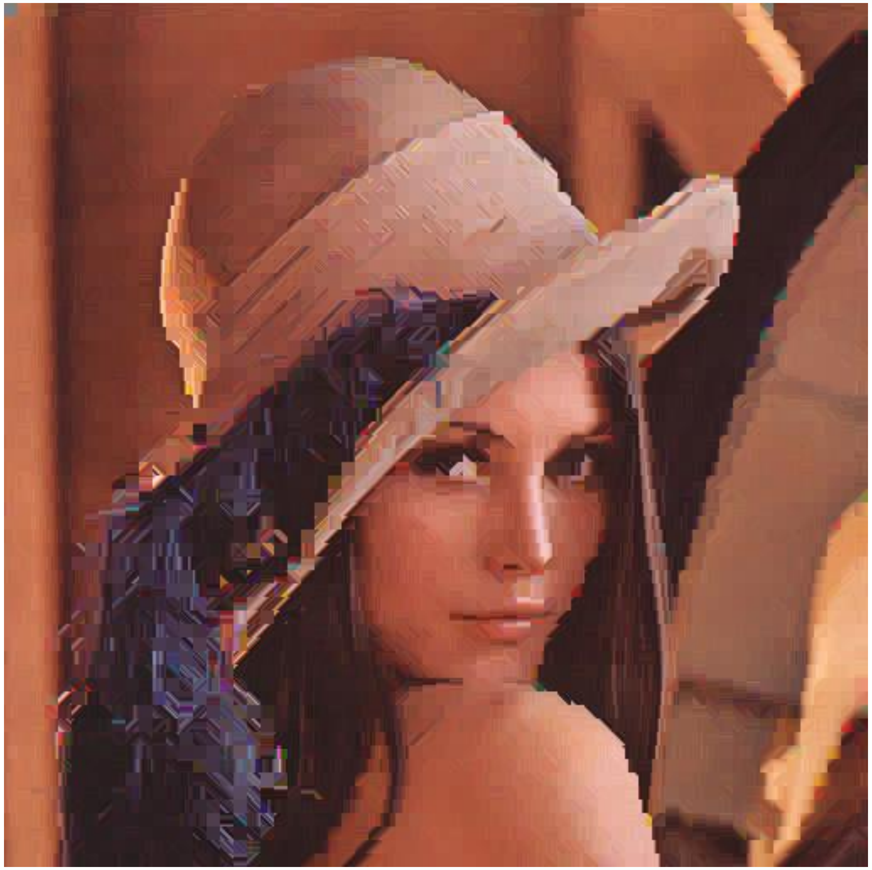


Intra Prediction (4x4)
PSNR = 27.59dB

Result

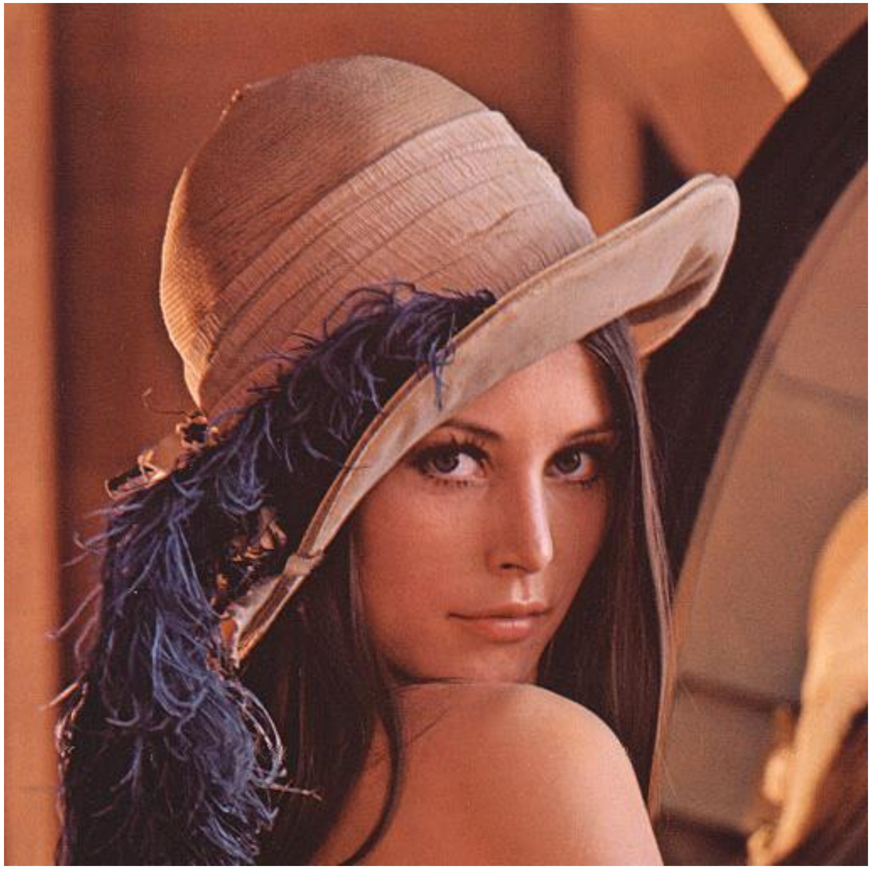


Original

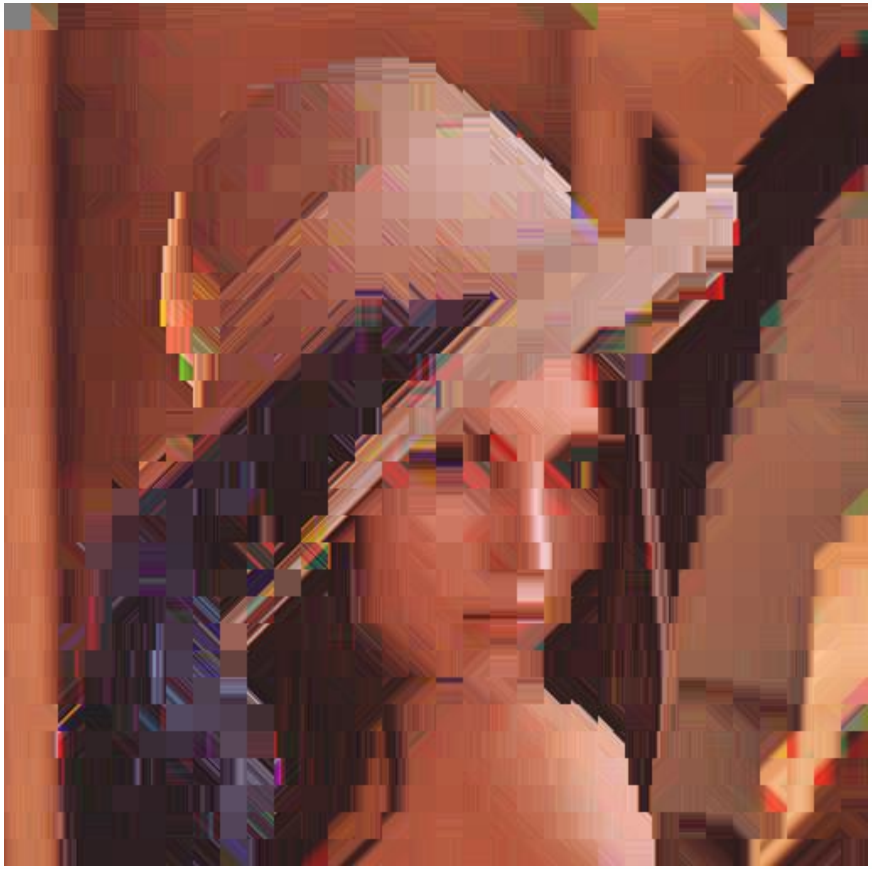


Intra Prediction (8x8)
PSNR = 25.08dB

Result

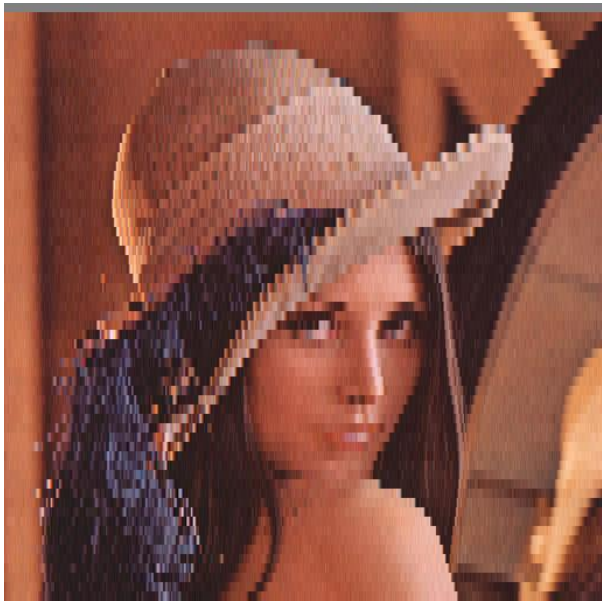


Original

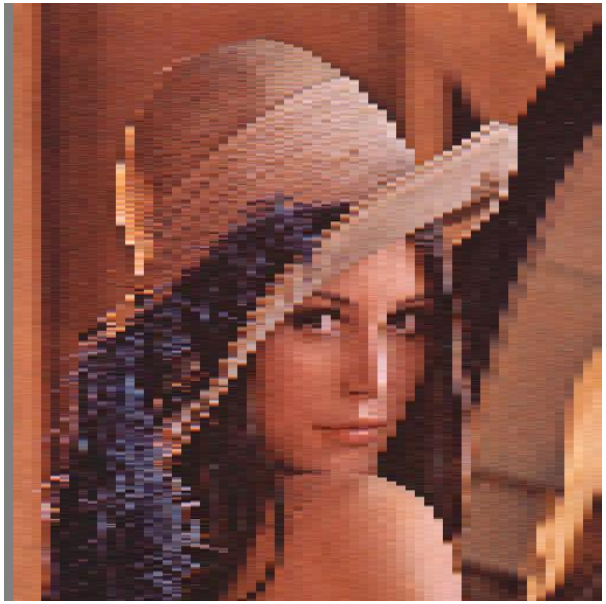


Intra Prediction (8x8)
PSNR = 22.66dB

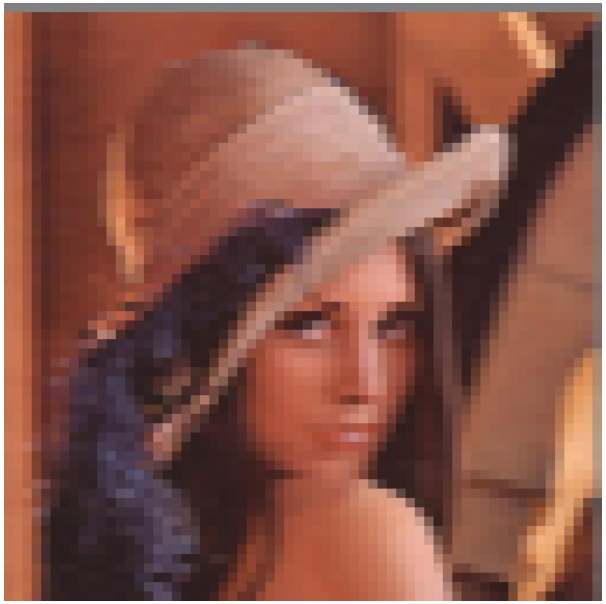
Result



Mode 0 (Vertical)
PSNR = 21.28 dB

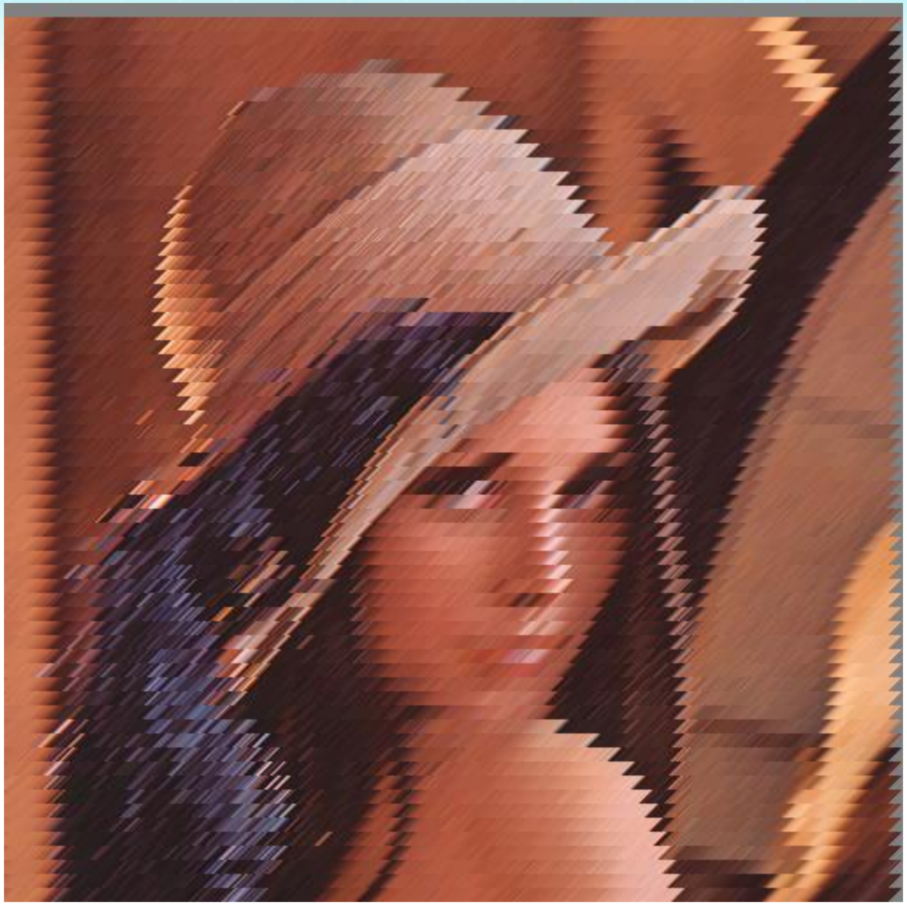


Mode 1 (Horizontal)
PSNR = 19.83 dB

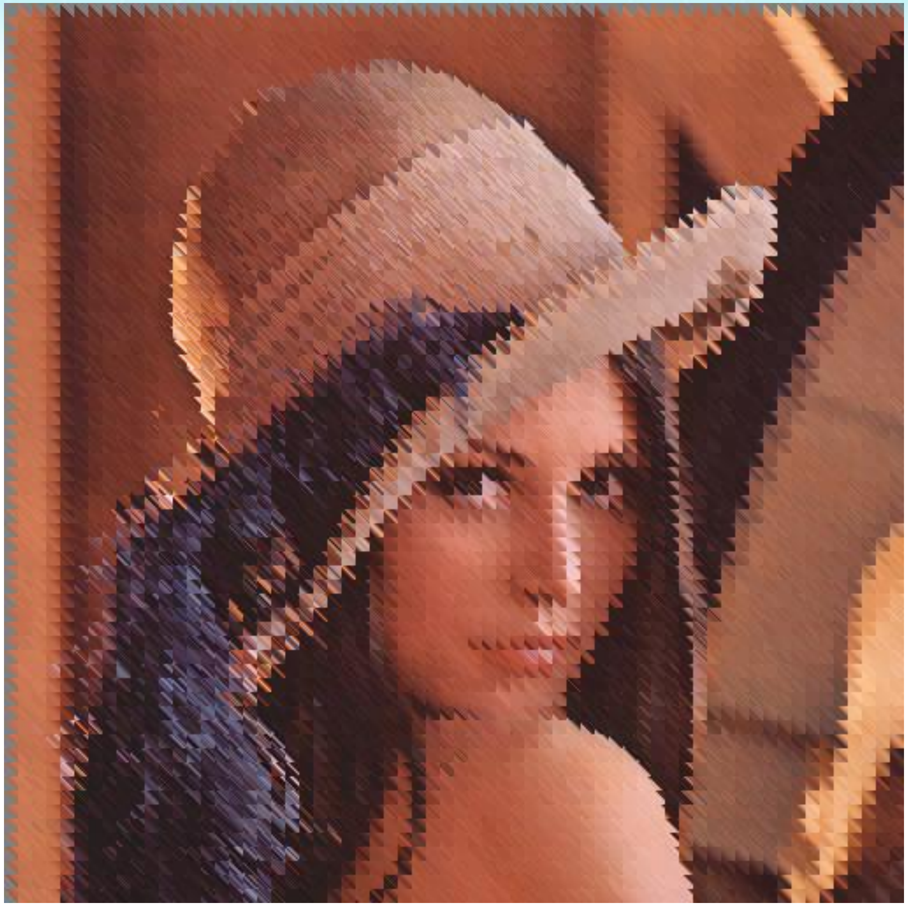


Mode 2 (Horizontal)
PSNR = 21.31 dB

Result



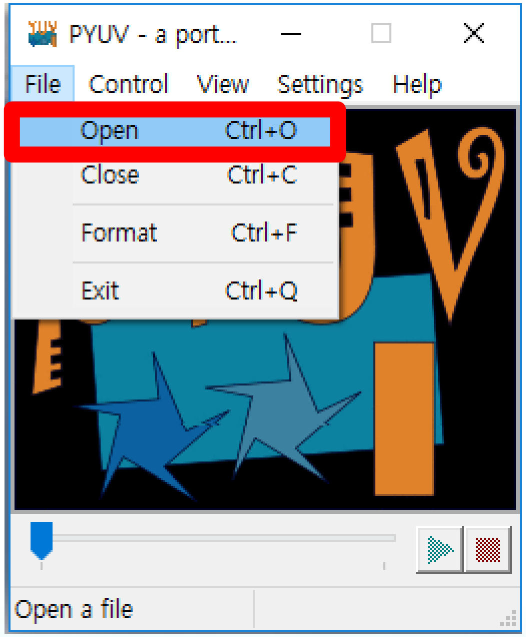
Mode 3 (Diagonal Left)
PSNR = 19.70 dB



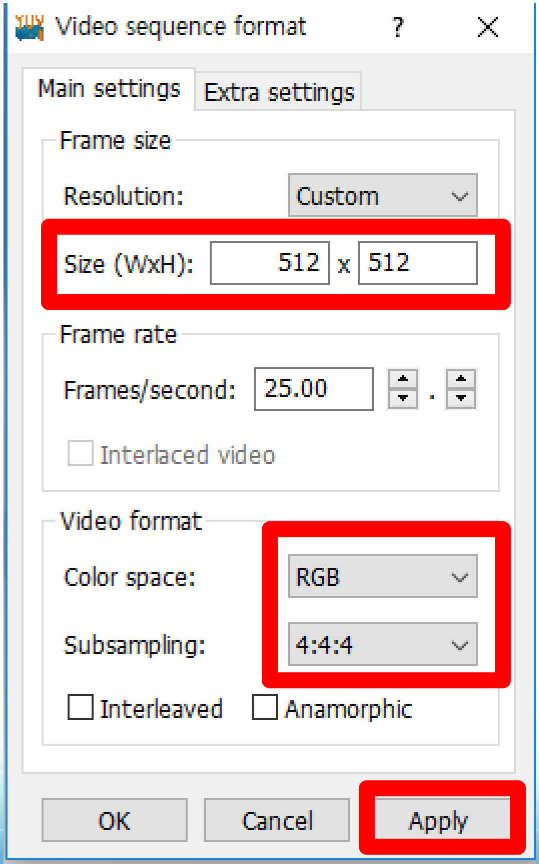
Mode 4 (Diagonal Right)
PSNR = 20.40 dB

pYUV.exe

- RAW file player
  - Freeware for a researcher



Open a file



Frame size

Resolution: Custom

Size (WxH): 512 x 512

Frame rate

Frames/second: 25.00

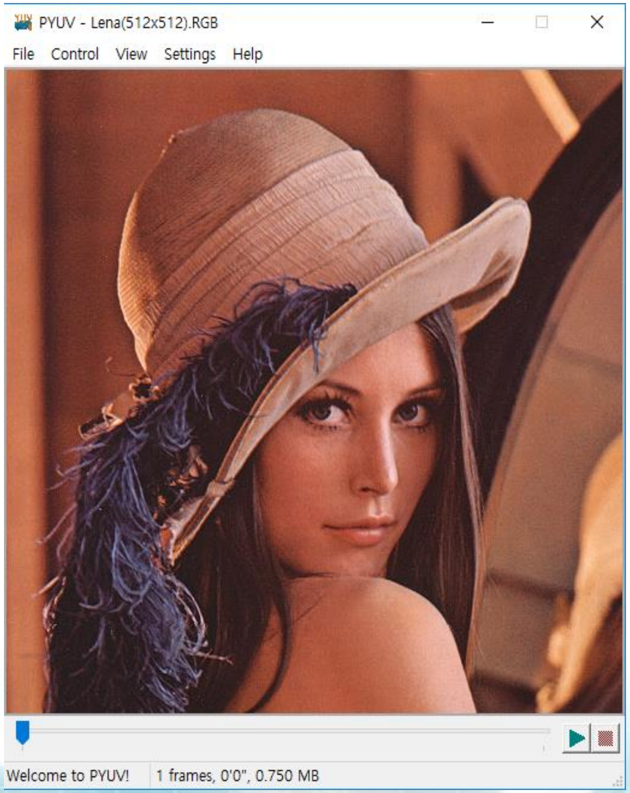
☐ Interlaced video

Video format

Color space: RGB

Subsampling: 4:4:4

☐ Interleaved ☐ Anamorphic



Welcome to PYUV!

1 frames, 0'0", 0.750 MB