영상처리 프로그래밍

<Assignment 11>

Jongseok Lee(suk2080@kw.ac.kr)

Yong-Jo Ahn (yjahn@digitalinsights.co.kr)

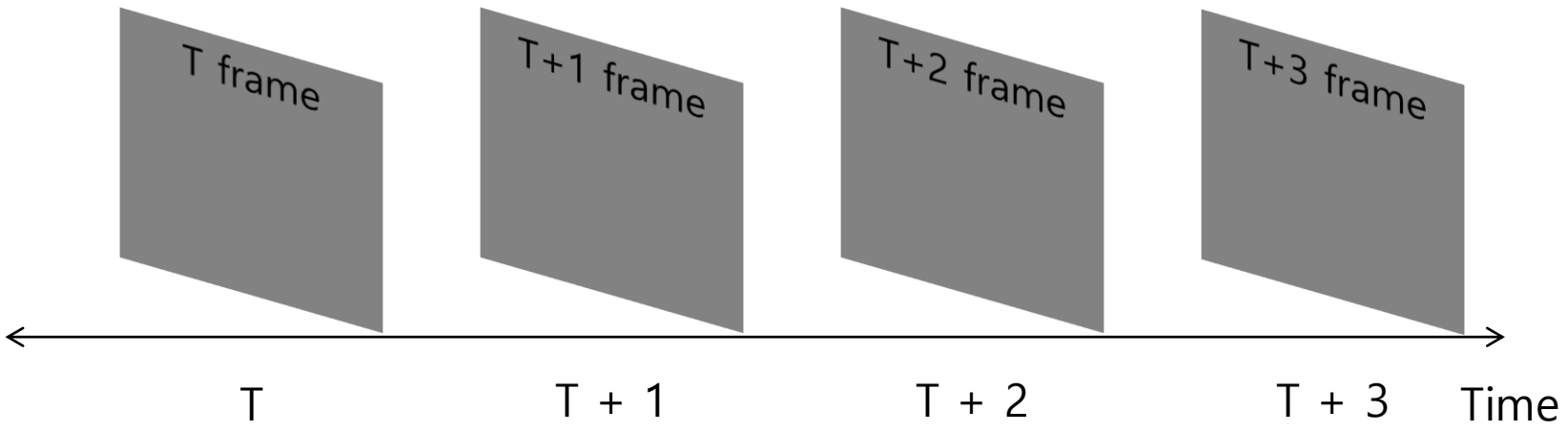
2018-05-30

Contents

- Video format
- Human Visual System
- Basic Codec
- Color Conversion
- Subsampling
- Example Code
- Assignments

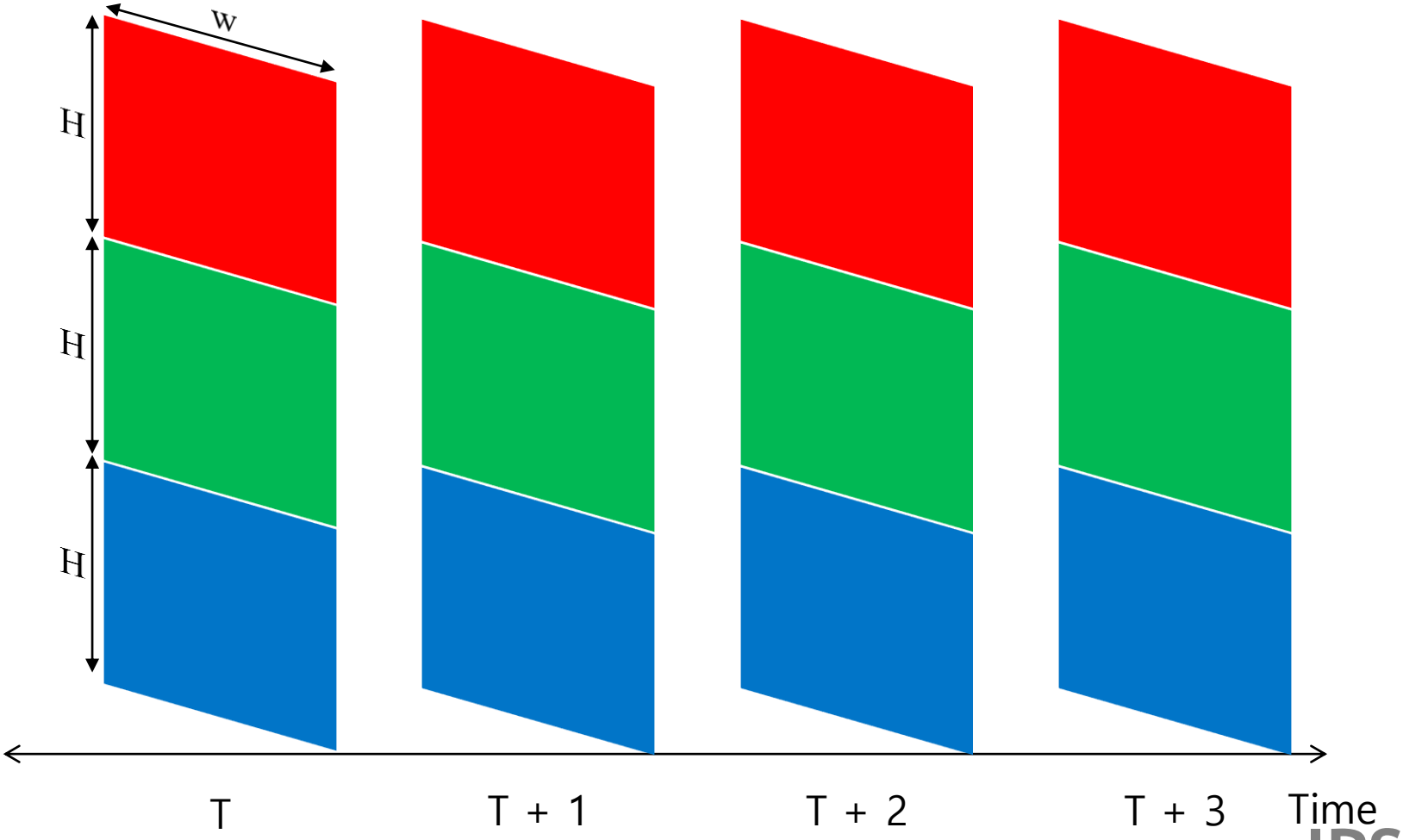
Video format

- Basic concept of the Video file

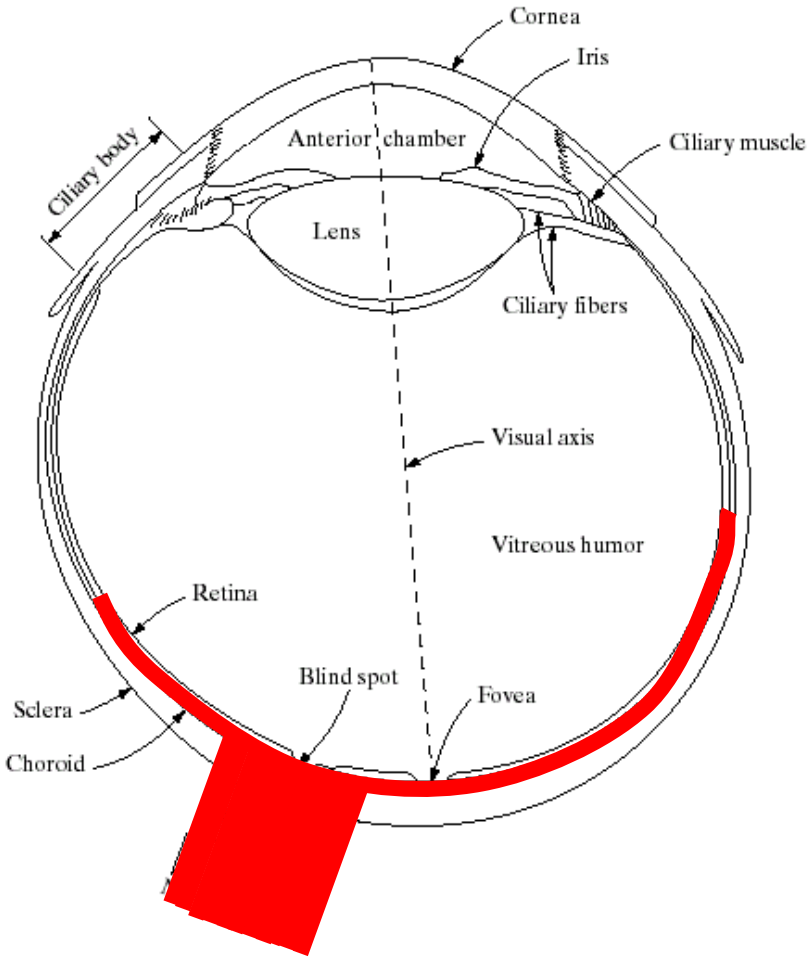


Video format

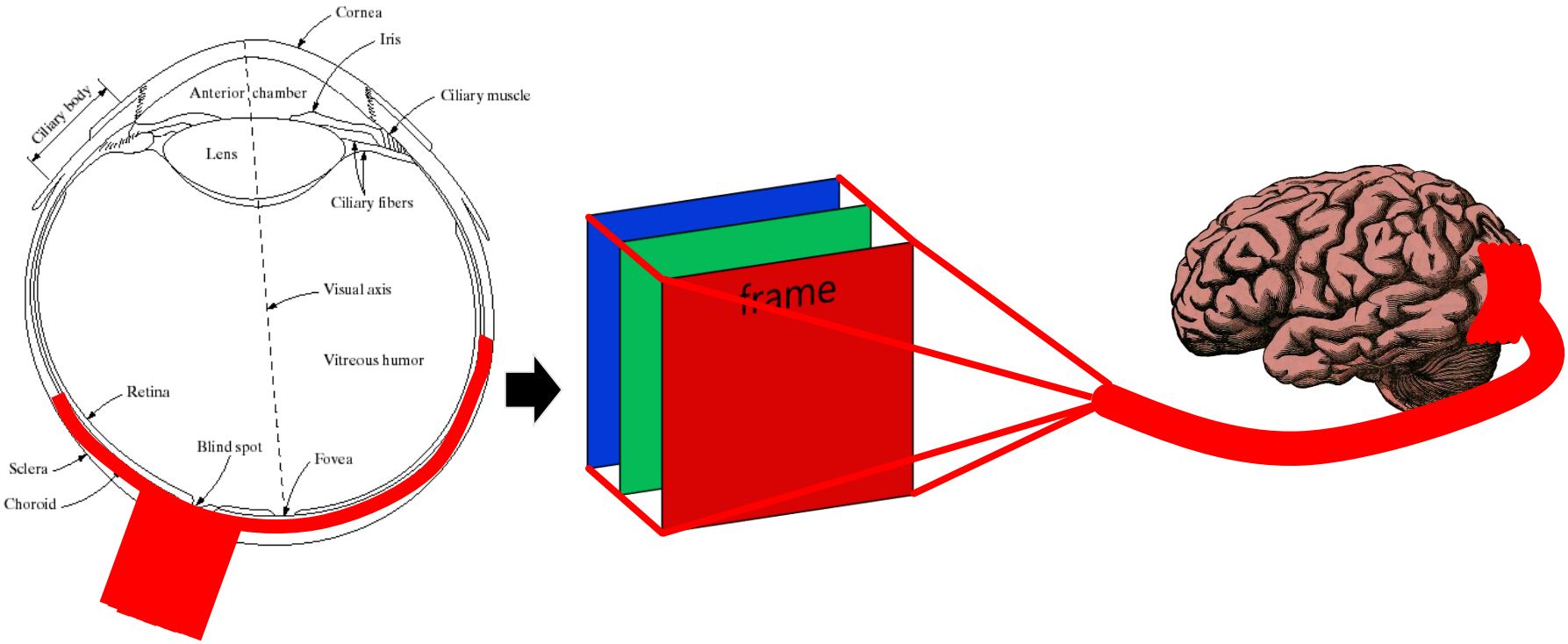
- Basic concept of the Video file (RGB)



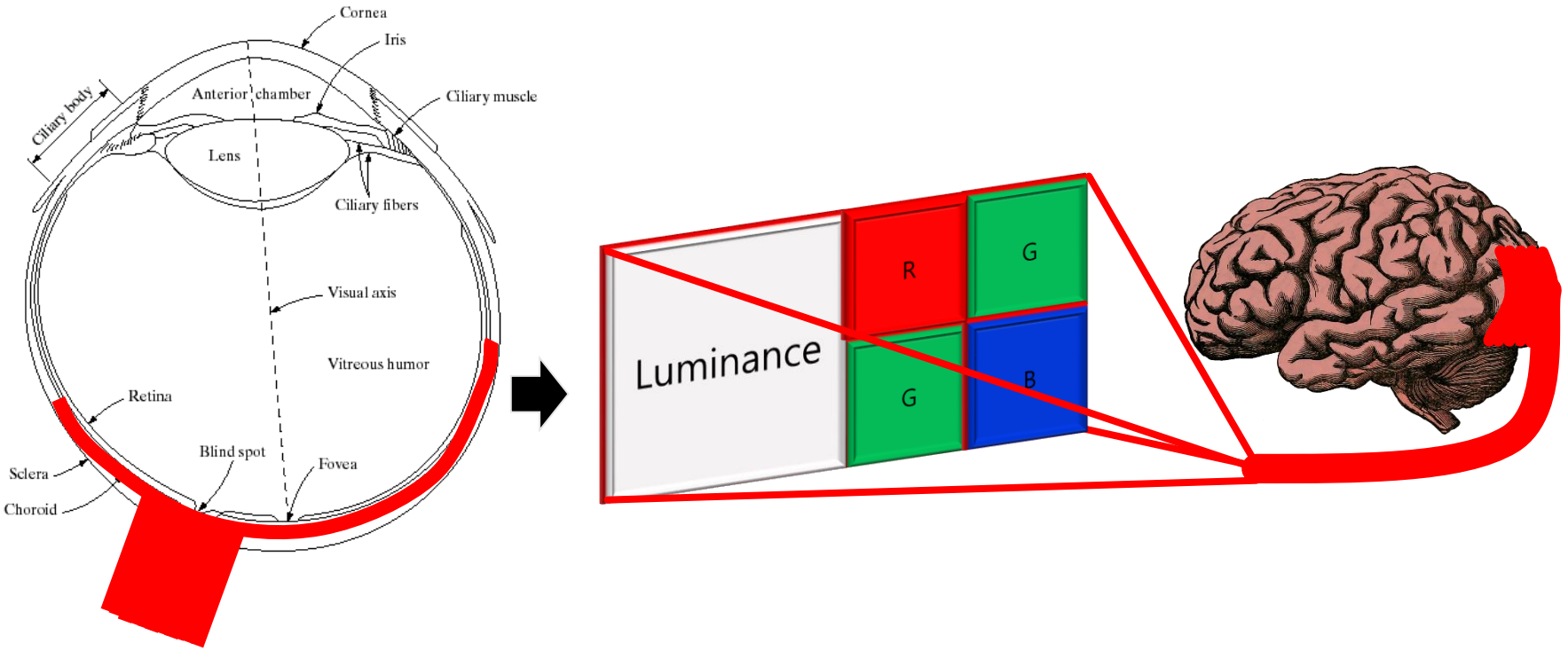
Human Visual System



Human Visual System

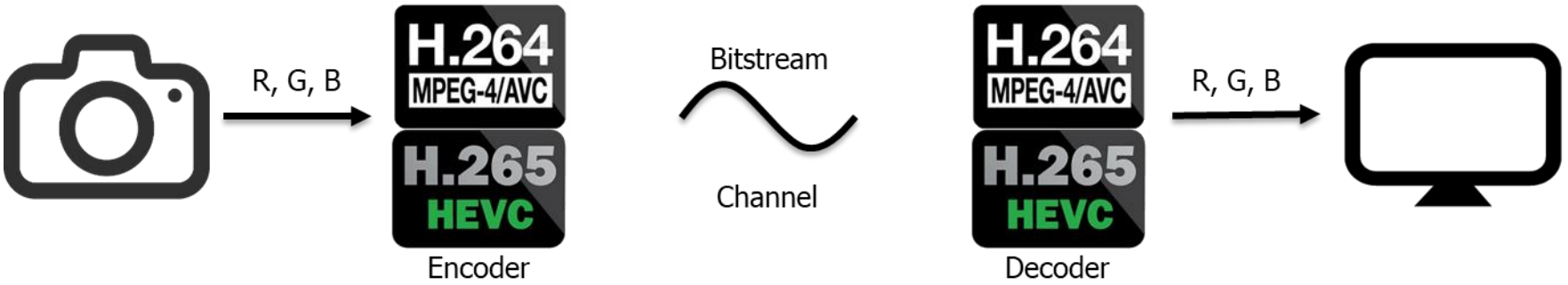


Human Visual System



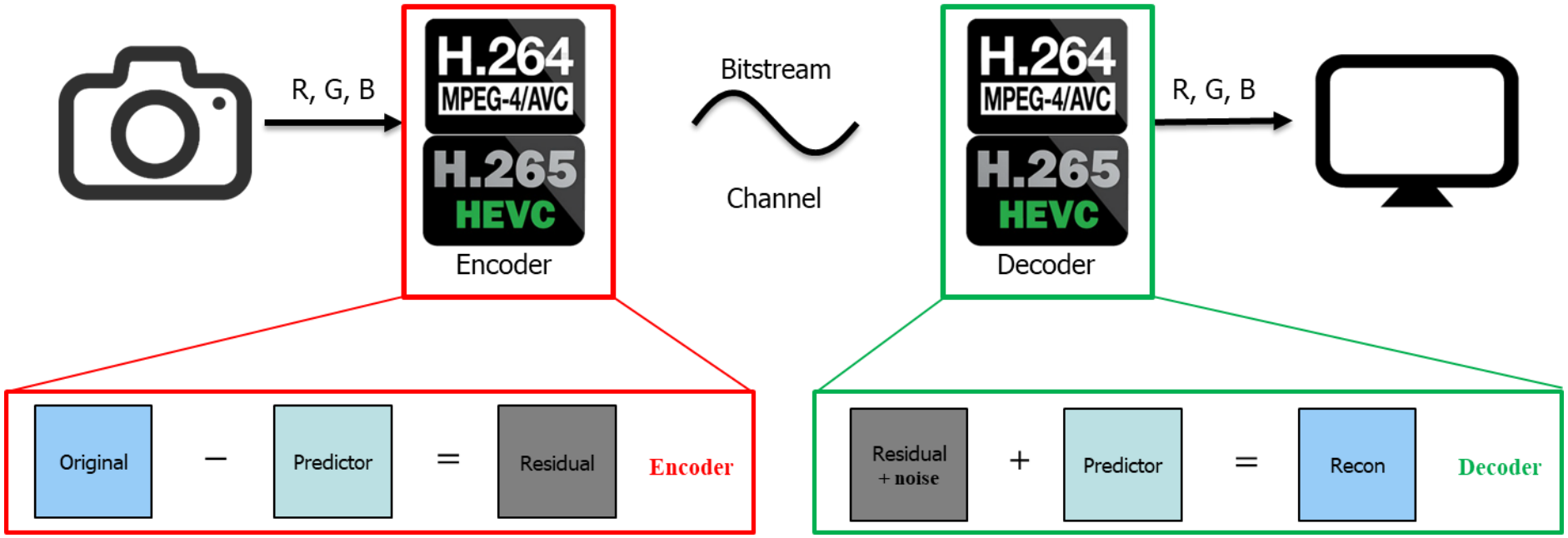
Basic Codec

- Basic Image/Video Codec



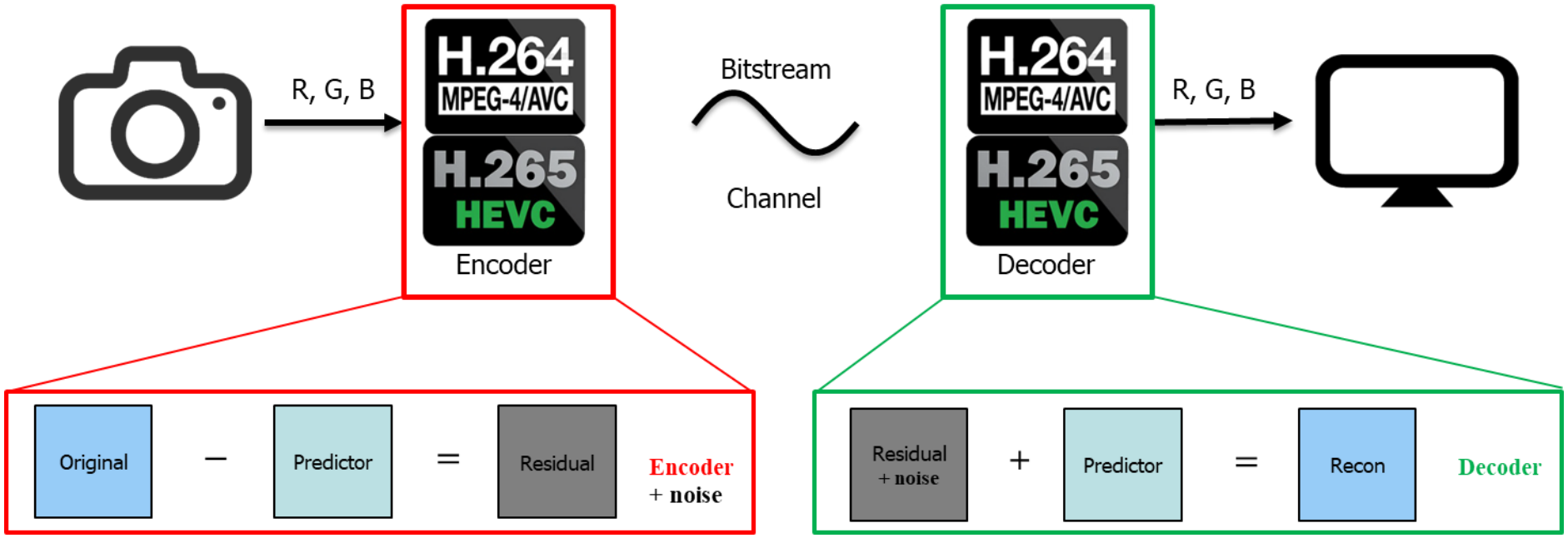
Basic Codec

- Basic Image/Video Codec



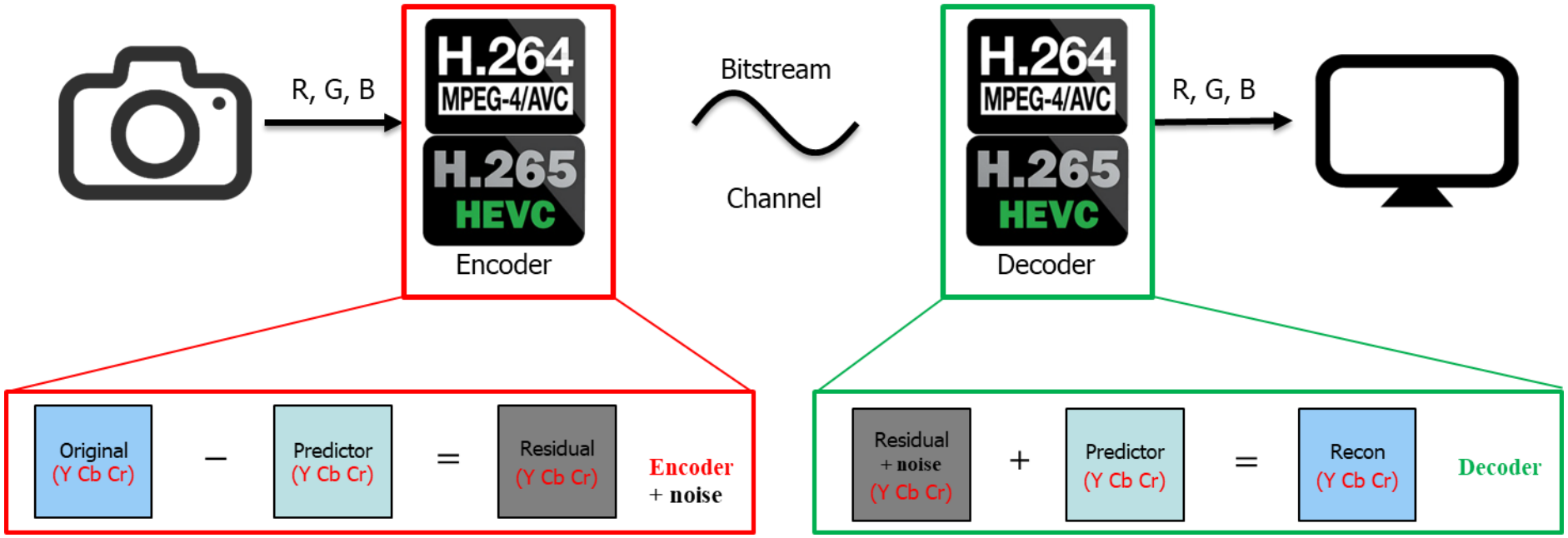
Basic Codec

- Basic Image/Video Codec



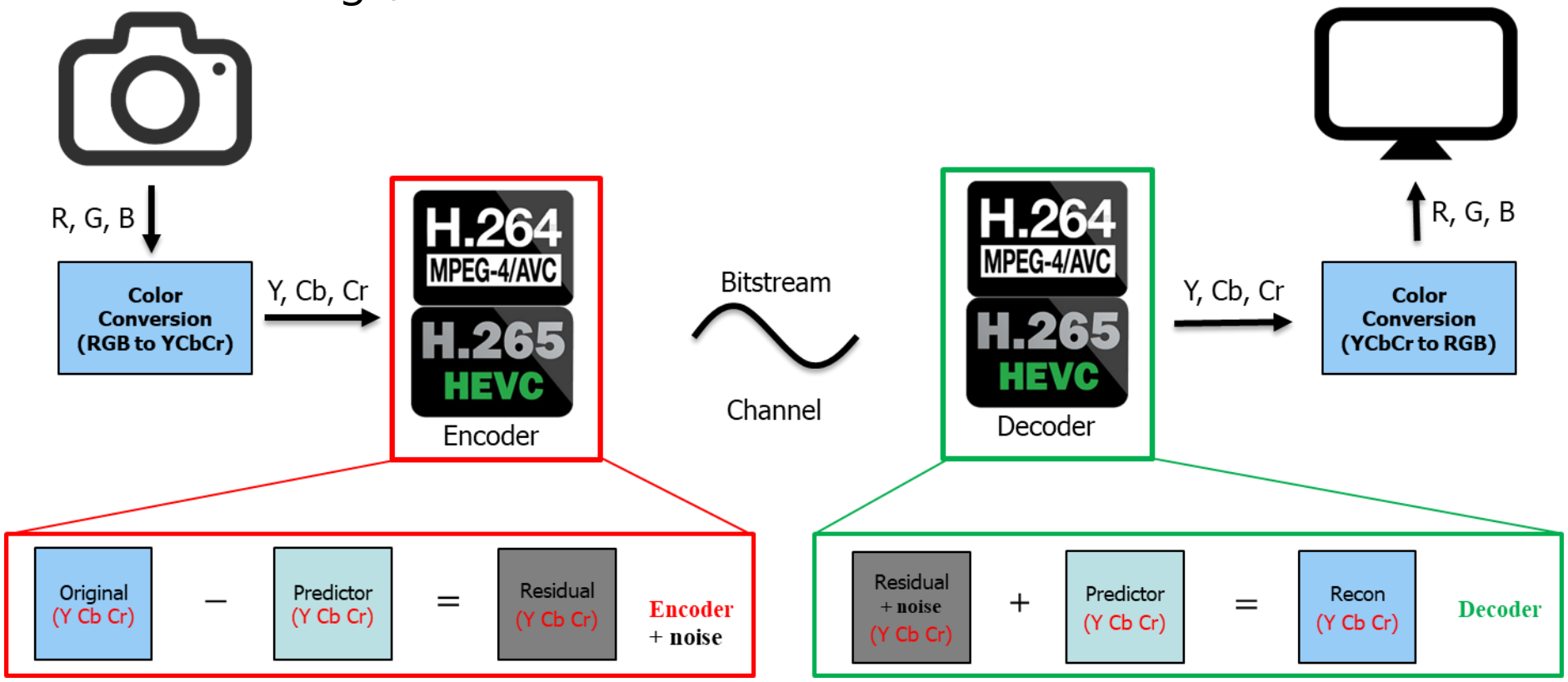
Basic Codec

- Basic Image/Video Codec



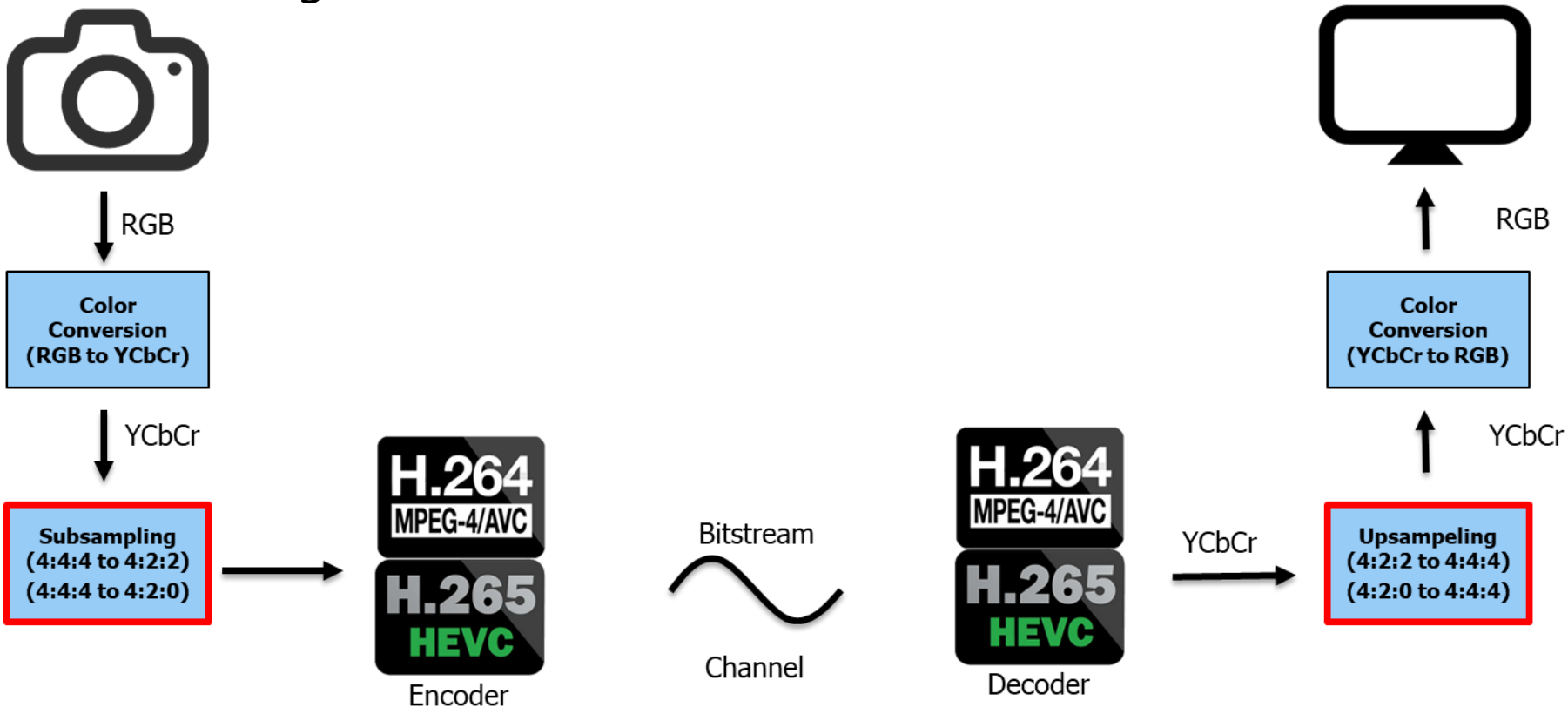
Basic Codec

- Basic Image/Video Codec



Basic Codec

- Basic Image/Video Codec



Color Conversion

- RGB to YUV (integer)

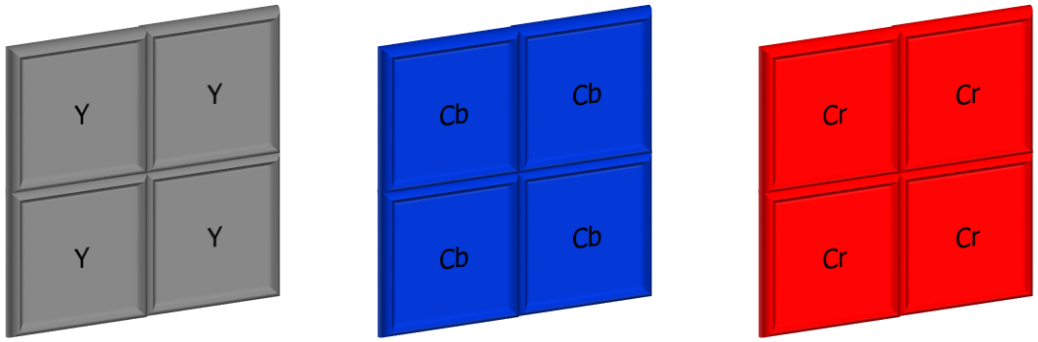
$$Y' = \left((66 \times R + 129 \times G + 25 \times B + 128) \gg 8\right) + 16$$
$$U = \left((-38 \times R - 74 \times G + 112 \times B + 128) \gg 8\right) + 128$$
$$V = \left((112 \times R - 94 \times G - 18 \times B + 128) \gg 8\right) + 128$$

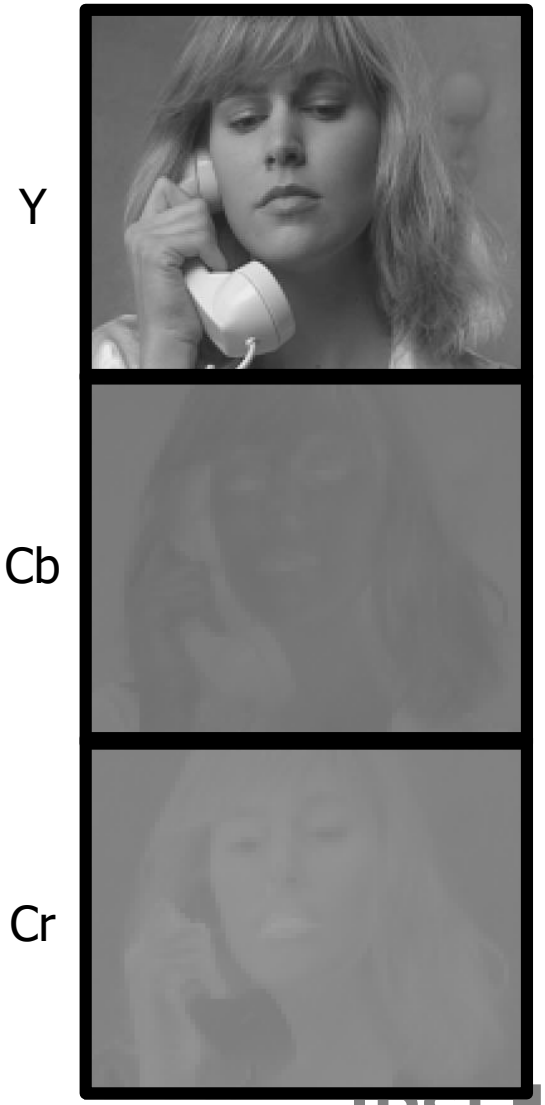
- YUV to RGB (integer)

$$C = Y' - 16$$
$$D = U - 128$$
$$E = V - 128$$
$$R = clamp\left((298 \times C + 409 \times E + 128) \gg 8\right)$$
$$G = clamp\left((298 \times C - 100 \times D - 208 \times E + 128) \gg 8\right)$$
$$B = clamp\left((298 \times C + 516 \times D + 128) \gg 8\right)$$

Subsampling

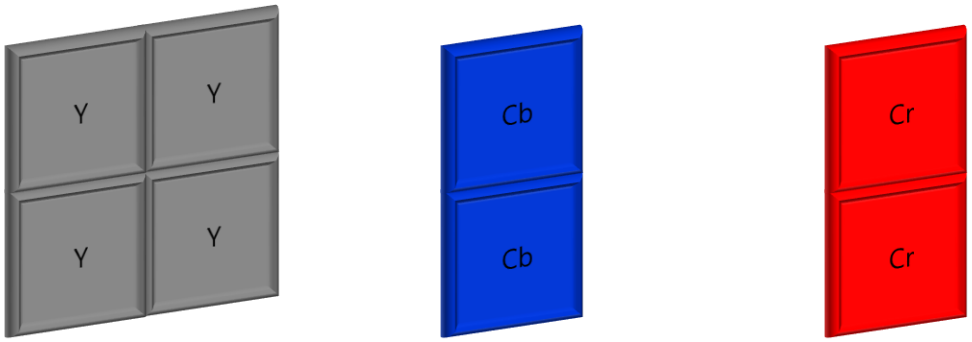
- 4:4:4 color component

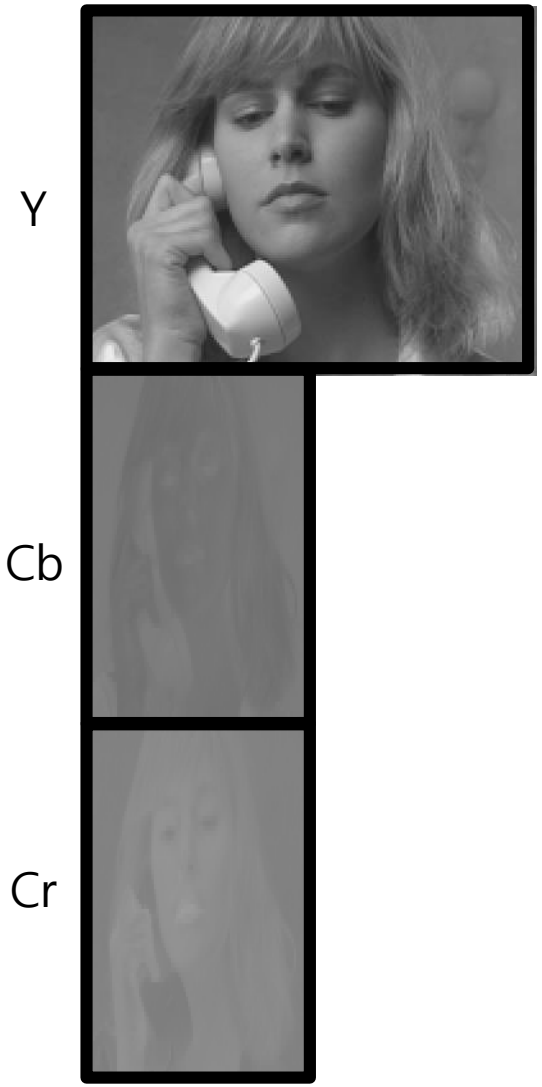




Subsampling

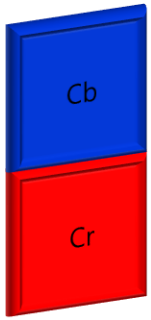
- 4:4:4 color component





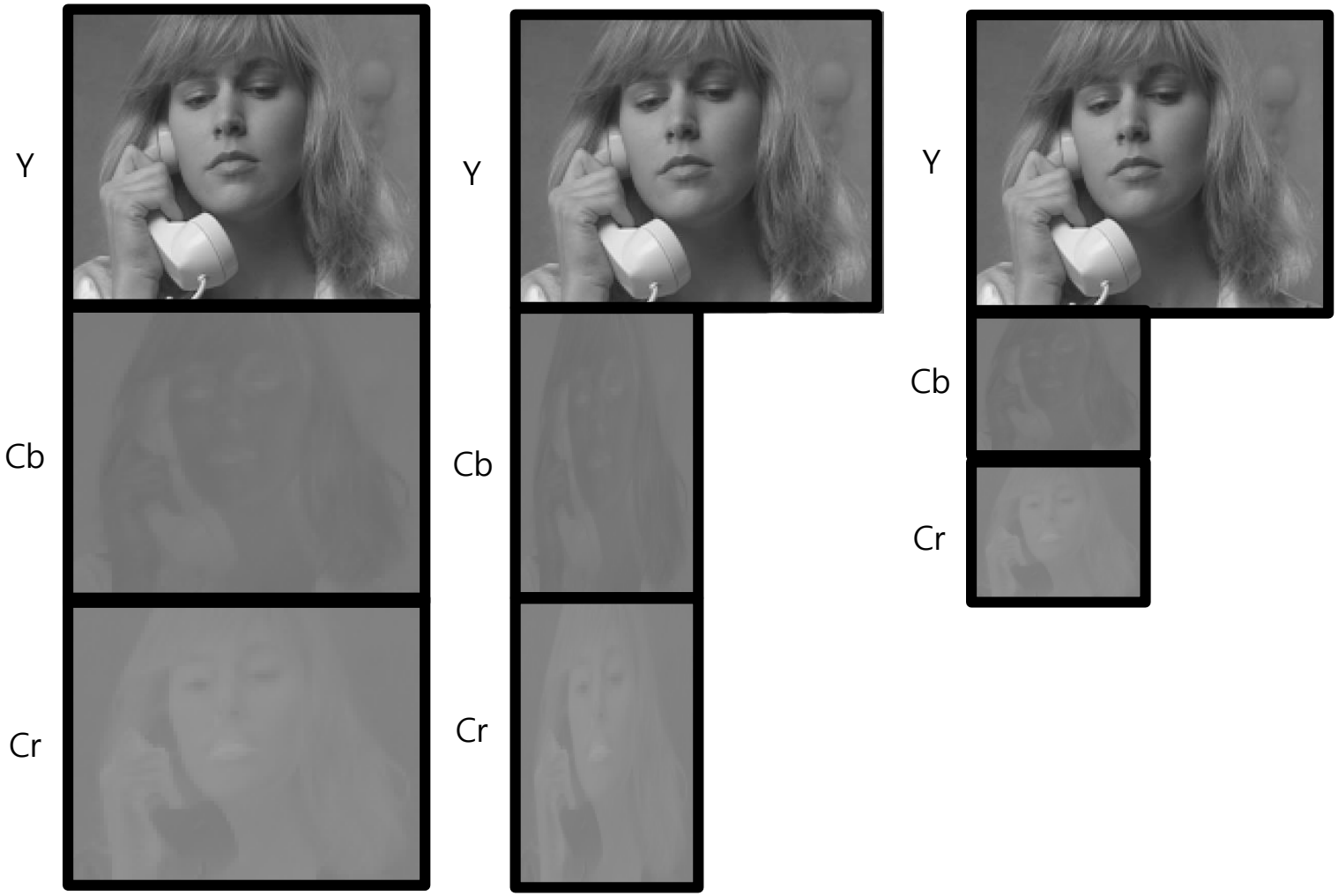
Subsampling

- 4:2:0 color component



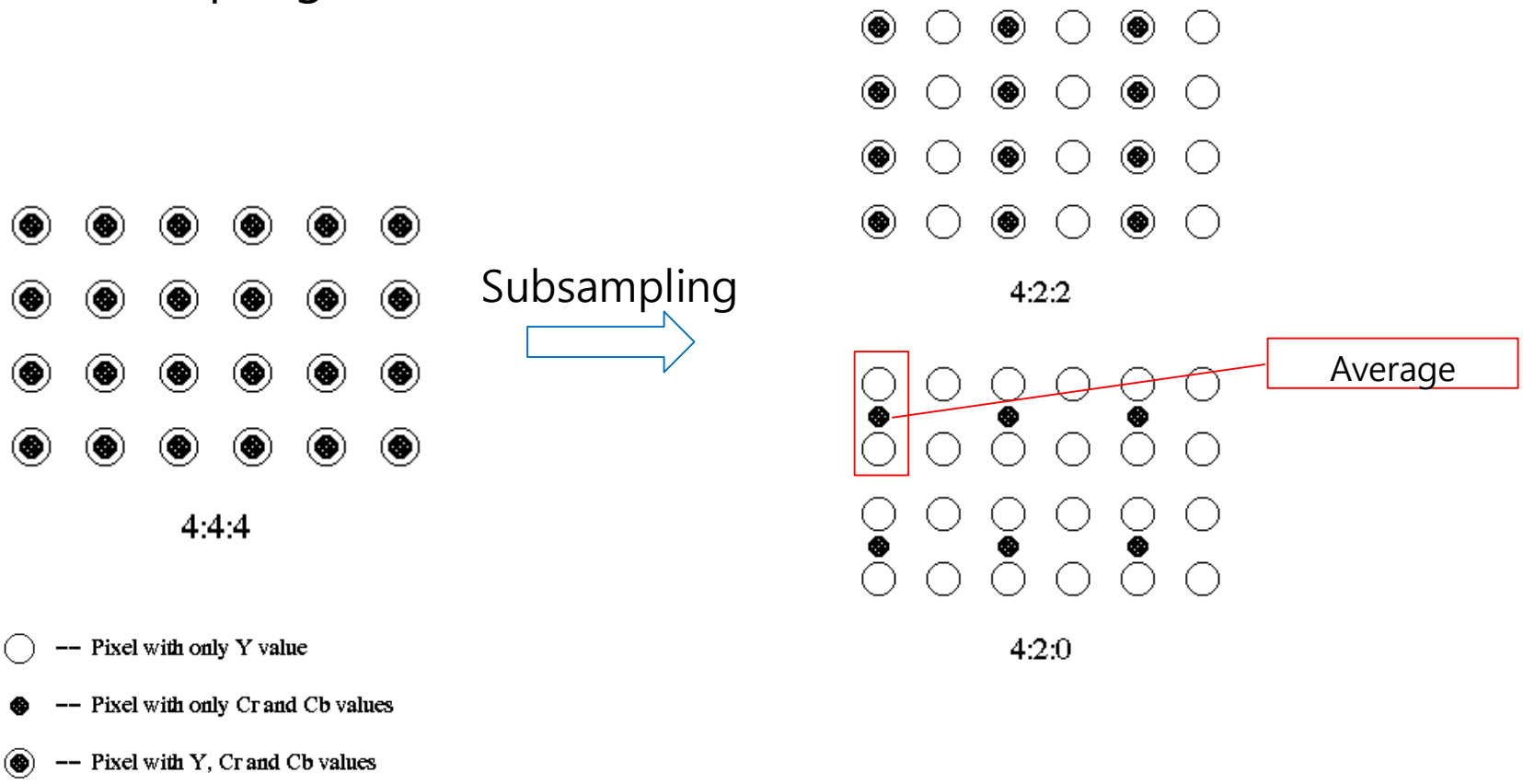
4 : 2 : 0

Subsampling



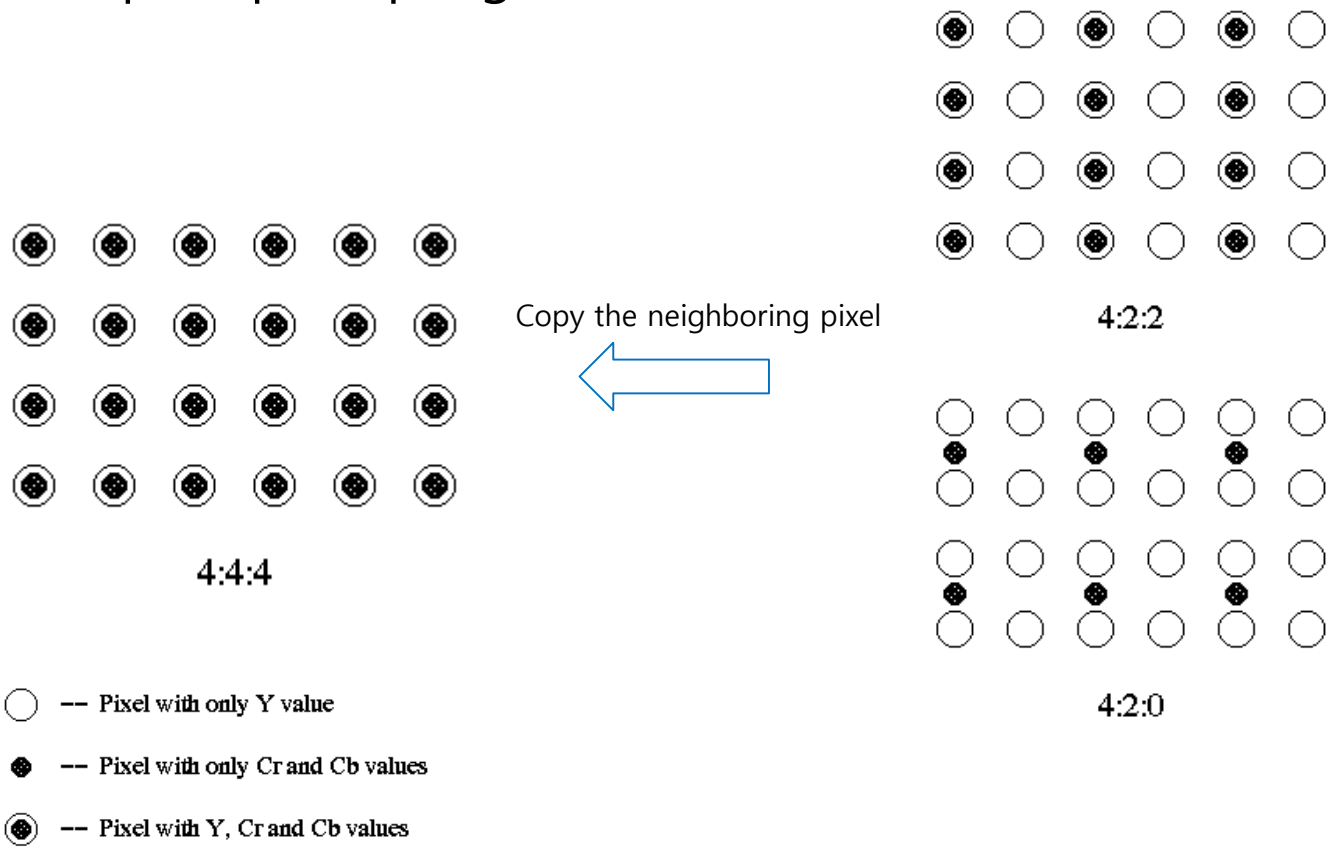
Subsampling

- Subsampling Method



Subsampling

- Simple upsampling method



Example Code

```c
#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 Clip(x) ( x < 0 ? 0 : ( x > 255 ? 255 : x))

typedef unsigned char BYTE;

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

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 YUV444_to_422(BYTE** img_in, BYTE** img_Y, BYTE** img_U422, BYTE** img_V422, int width, int height);      // Chroma sampling   4:4:4 -> 4:2:2
void YUV422_to_444(BYTE** img_Y, BYTE** img_U422, BYTE** img_V422, BYTE** img_out, int width, int height);      // Chroma sampling   4:2:2 -> 4:4:4

int main()
{
    FILE *fp_in   = fopen("Suzie_CIF_150_30.rgb", "rb");            //in RGB file
    FILE *fp_out0 = fopen("[YUV444]Suzie_CIF_150_30.yuv", "wb");    //out yuv 444 file
    FILE *fp_out1 = fopen("[YUV420]Suzie_CIF_150_30.yuv", "wb");    //out yuv 420 file
    FILE *fp_out2 = fopen("[YUV422]Suzie_CIF_150_30.yuv", "wb");    //out yuv 422 file
    FILE *fp_out3 = fopen("[Recon_420]Suzie_CIF_150_30.rgb", "wb"); //recon RGB file
    FILE *fp_out4 = fopen("[Recon_422]Suzie_CIF_150_30.rgb", "wb"); //recon RGB file


    BYTE **img_YUV444, **img_RGB;           // in : RGB444      out : YUV444, YUV420, YUV422, recon RGB
    BYTE **img_Y, **img_U420, **img_V420;   // 420 memory pointer
    BYTE **img_U422, **img_V422;            // 422 memory pointer

    int size = 1;  // loop condition

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

    // YUV 420 memory
    img_Y = MemAlloc_2D(WIDTH, HEIGHT);
    img_U420 = MemAlloc_2D(WIDTH>>1, HEIGHT>>1);
    img_V420 = MemAlloc_2D(WIDTH>>1, HEIGHT>>1);

    // YUV 422 memory
    img_U422 = MemAlloc_2D(WIDTH >> 1, HEIGHT);
    img_V422 = MemAlloc_2D(WIDTH >> 1, HEIGHT);
```

Example Code

```
    while (size = Read_Frame(fp_in, img_RGB, WIDTH, HEIGHT * 3)) //Loop RGB444 -> YUV444 -> YUV420 -> YUV444 -> RGB444
    {                                                            //                    -> YUV422 -> YUV444 -> RGB444
        RGB_to_YUV(img_RGB, img_YUV444, WIDTH, HEIGHT);                 //Color conversion
        Write_Frame(fp_out0, img_YUV444, WIDTH, HEIGHT * 3);            //YUV444

        //Chroma subsampling 420 & 422
        YUV444_to_420(img_YUV444, img_Y, img_U420, img_V420, WIDTH, HEIGHT);
        YUV444_to_422(img_YUV444, img_Y, img_U422, img_V422, WIDTH, HEIGHT);

        //YUV420 Write
        Write_Frame(fp_out1, img_Y, WIDTH, HEIGHT);                             // Y
        Write_Frame(fp_out1, img_U420, WIDTH >> 1, HEIGHT >> 1);                // U420
        Write_Frame(fp_out1, img_V420, WIDTH >> 1, HEIGHT >> 1);                // V420

        //YUV422 Write
        Write_Frame(fp_out2, img_Y, WIDTH, HEIGHT);                             // Y
        Write_Frame(fp_out2, img_U422, WIDTH >> 1, HEIGHT);                     // U422
        Write_Frame(fp_out2, img_V422, WIDTH >> 1, HEIGHT);                     // V422

        YUV420_to_444(img_Y, img_U420, img_V420, img_YUV444, WIDTH, HEIGHT);    // YUV 420 -> 444
        YUV_to_RGB(img_YUV444, img_RGB, WIDTH, HEIGHT);
        Write_Frame(fp_out3, img_RGB, WIDTH, HEIGHT * 3);                       // 420 -> 444 -> recon RGB444

        YUV422_to_444(img_Y, img_U422, img_V422, img_YUV444, WIDTH, HEIGHT);    // YUV 422 -> 444
        YUV_to_RGB(img_YUV444, img_RGB, WIDTH, HEIGHT);                         //Color conversion
        Write_Frame(fp_out4, img_RGB, WIDTH, HEIGHT * 3);                       // 422 -> 444 -> recon RGB444
    }


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

    MemFree_2D(img_Y, HEIGHT);
    MemFree_2D(img_U420, HEIGHT>>1);
    MemFree_2D(img_V420, HEIGHT>>1);

    MemFree_2D(img_U422, HEIGHT);
    MemFree_2D(img_V422, HEIGHT);

    fcloseall();        //file close

    return 0;
}
```

Example Code

```
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
        }
    }
}
```

Assignments

- Assignment1
  - Example code를 참고하여 RGB영상 변환 및 관련 내용 학습
    - RGB to YUV 444
    - RGB to YUV 422
    - RGB to YUV 420

- Assignment2
  - YUV 영상을 입력으로 받는 MFC 기반 YUV 플레이어 구현
    - 다양한 subsampling을 지원하는 YUV 플레이어 구현
      - **RGB**
      - **YUV444**
      - YUV422
      - YUV420

END OF PRESENTATION

Q&A