KWANGWOON UNIVERSITY

Video Subsampling

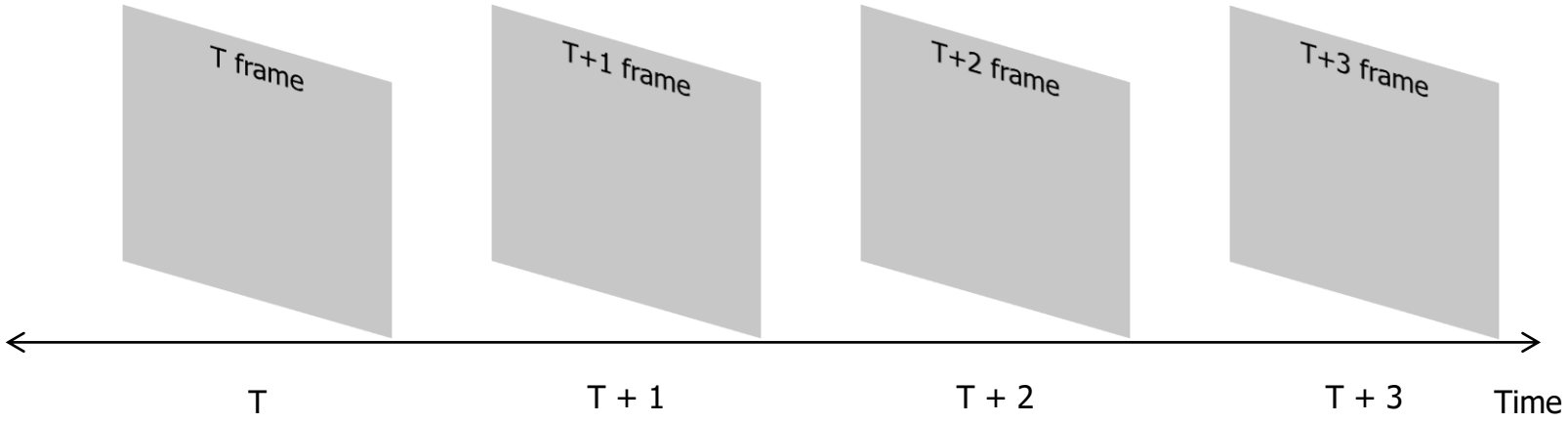
2018.11.13

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

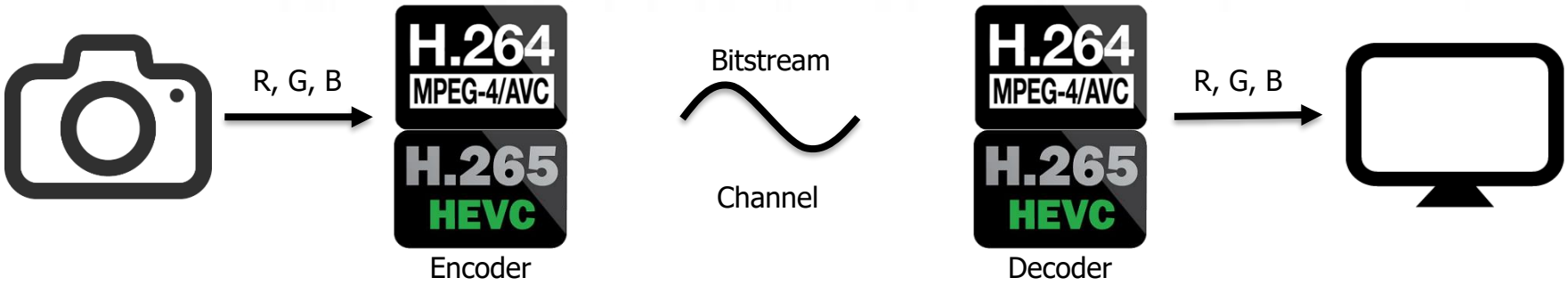
Multimedia LAB
VIA-Multimedia Center, Kwangwoon University

Video format

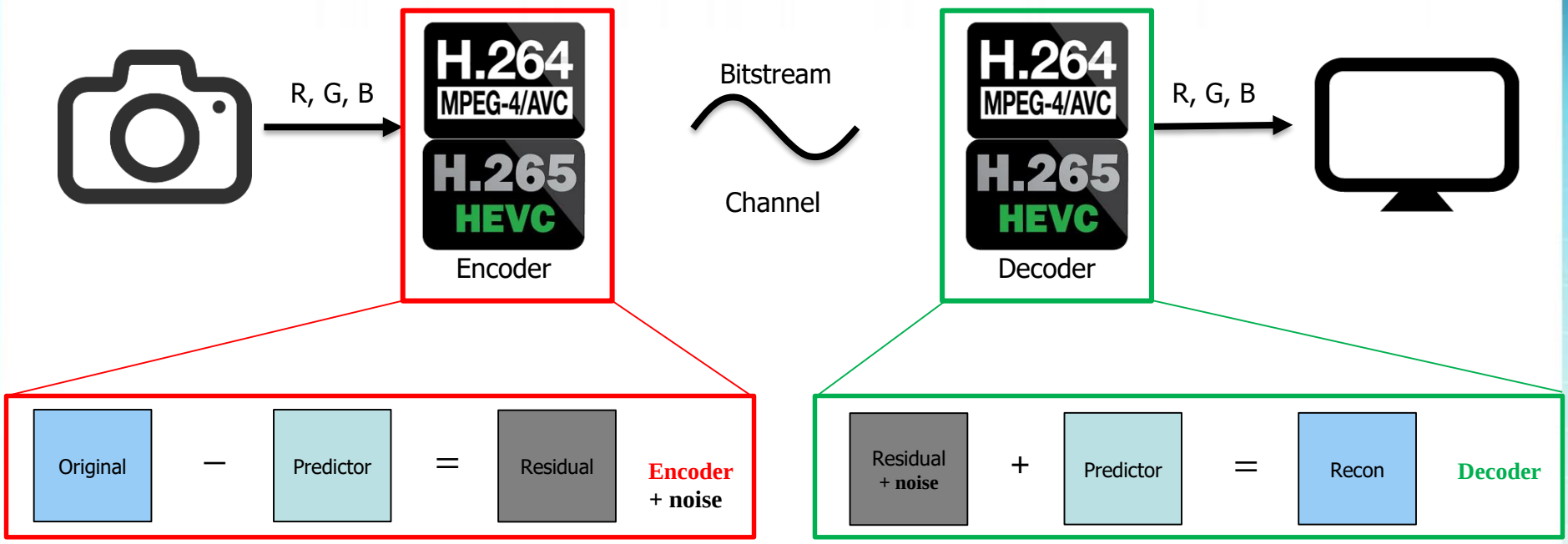
- Basic concept of the video file



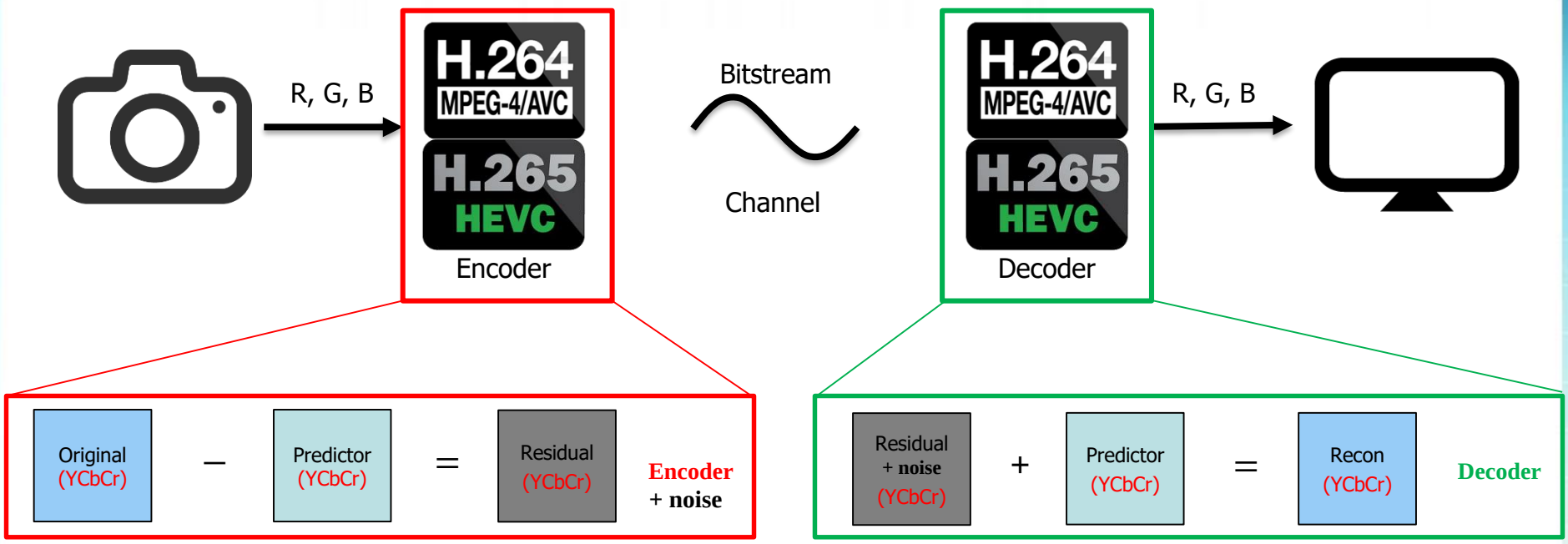
CODEC


R, G, B
H.264
MPEG-4/AVC
H.265
HEVC
Encoder
Bitstream
Channel
H.264
MPEG-4/AVC
H.265
HEVC
Decoder
R, G, B

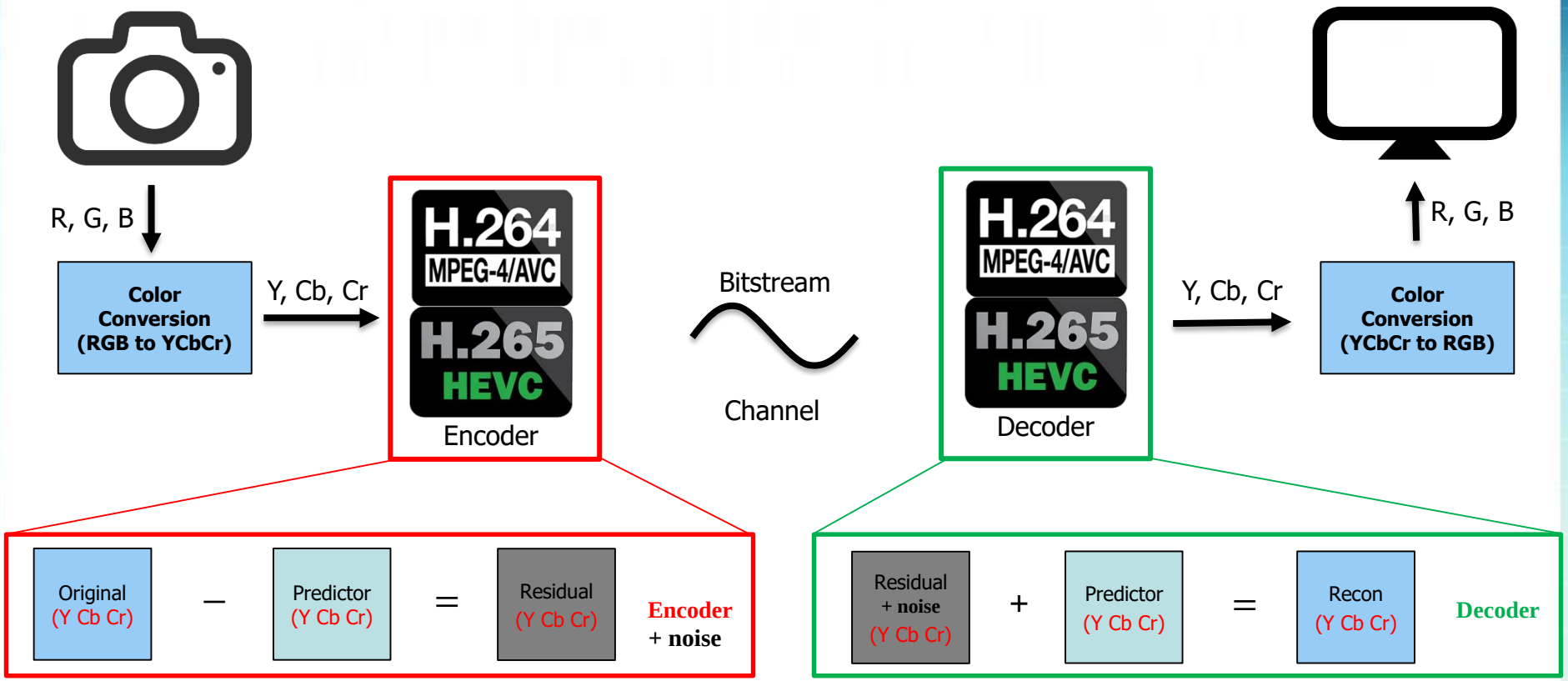
CODEC


R, G, B
H.264
MPEG-4/AVC
H.265
HEVC
Encoder
Bitstream
Channel
H.264
MPEG-4/AVC
H.265
HEVC
Decoder
R, G, B
Original
−
Predictor
=
Residual
Encoder
+ noise
Residual
+ noise
+
Predictor
=
Recon
Decoder

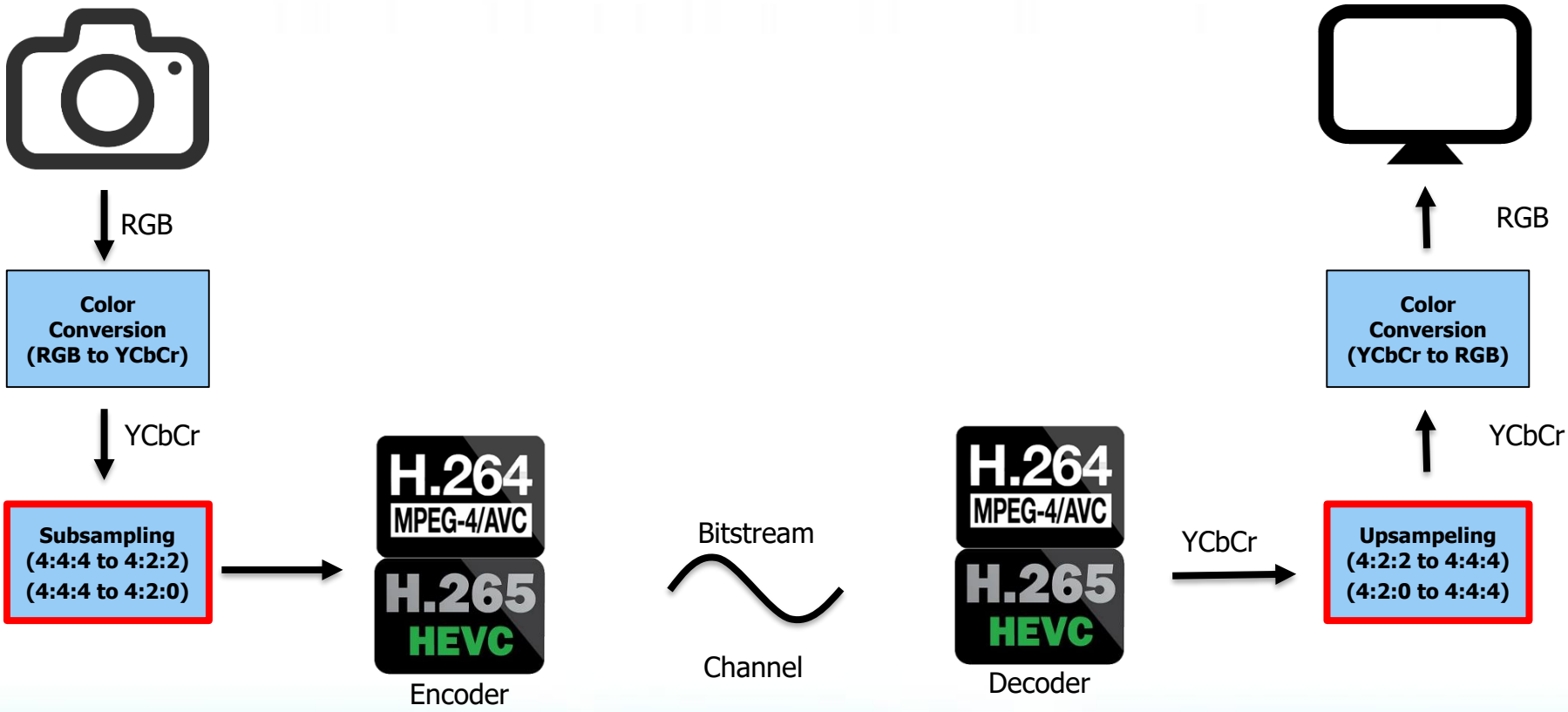
CODEC


R, G, B
H.264
MPEG-4/AVC
H.265
HEVC
Encoder
Bitstream
Channel
H.264
MPEG-4/AVC
H.265
HEVC
Decoder
R, G, B
Original
(YCbCr)
–
Predictor
(YCbCr)
=
Residual
(YCbCr)
Encoder
+ noise
Residual
+ noise
(YCbCr)
+
Predictor
(YCbCr)
=
Recon
(YCbCr)
Decoder

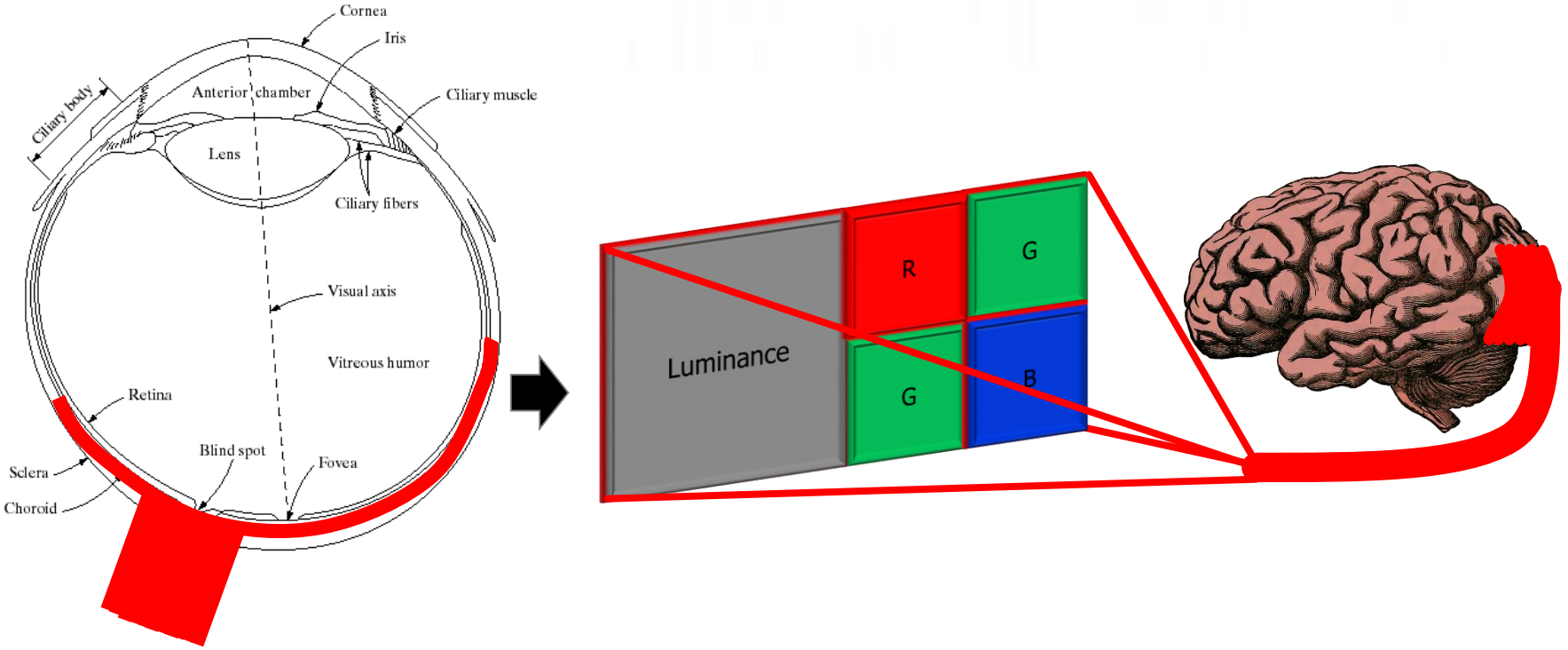
CODEC


R, G, B
Color Conversion (RGB to YCbCr)
Y, Cb, Cr
H.264 MPEG-4/AVC
H.265 HEVC
Encoder
Bitstream
Channel
H.264 MPEG-4/AVC
H.265 HEVC
Decoder
Y, Cb, Cr
Color Conversion (YCbCr to RGB)
R, G, B
Original (Y Cb Cr)
−
Predictor (Y Cb Cr)
=
Residual (Y Cb Cr)
Encoder + noise
Residual + noise (Y Cb Cr)
+
Predictor (Y Cb Cr)
=
Recon (Y Cb Cr)
Decoder

CODEC

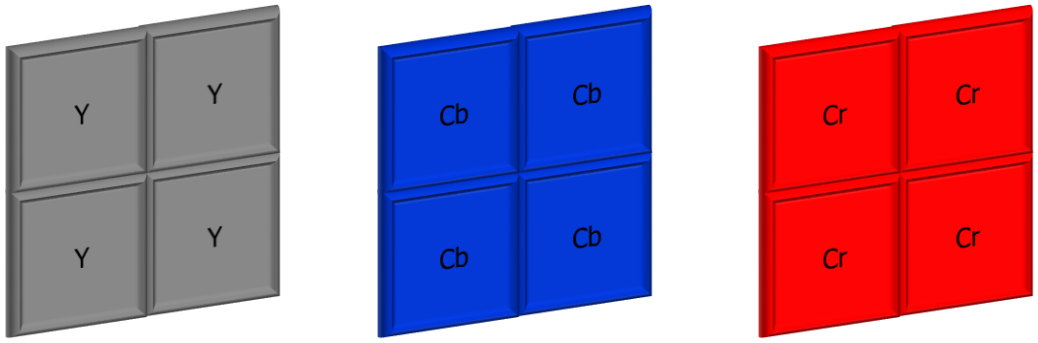

RGB
Color Conversion (RGB to YCbCr)
YCbCr
Subsampling (4:4:4 to 4:2:2) (4:4:4 to 4:2:0)
H.264 MPEG-4/AVC
H.265 HEVC
Encoder
Bitstream
Channel
H.264 MPEG-4/AVC
H.265 HEVC
Decoder
YCbCr
Upsampeling (4:2:2 to 4:4:4) (4:2:0 to 4:4:4)
YCbCr
Color Conversion (YCbCr to RGB)
RGB

Elements of Visual Perception

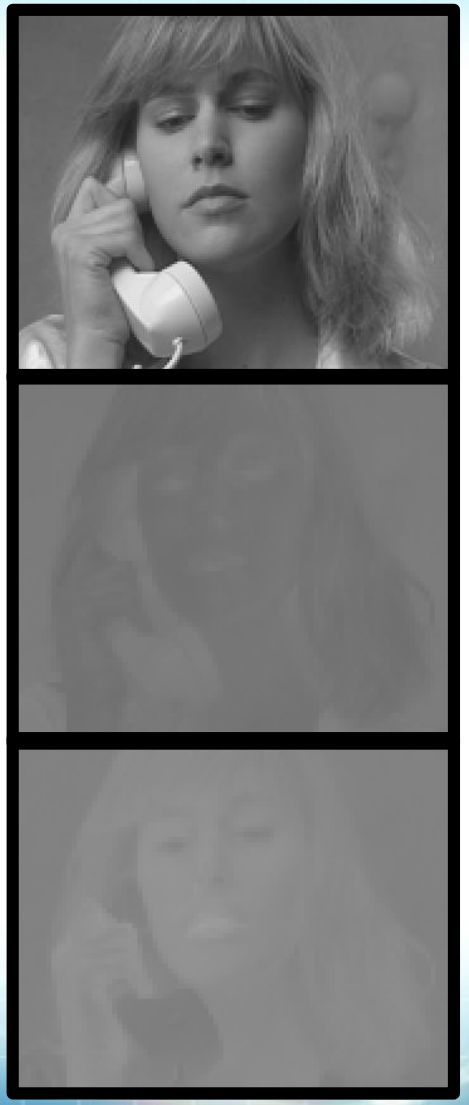


Subsampling

- 4:4:4 color component

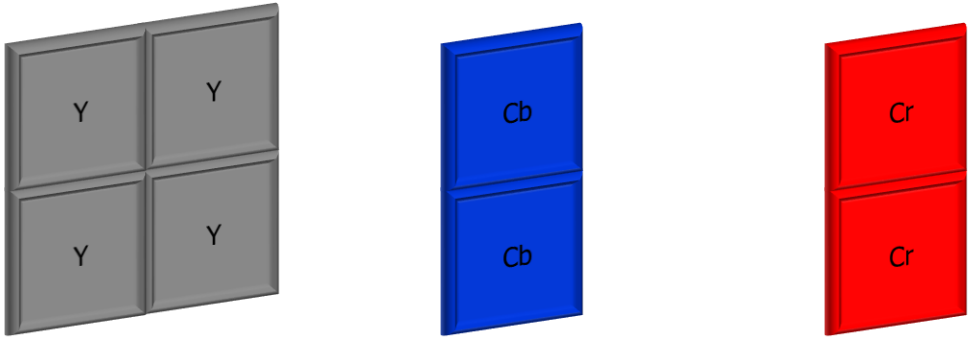


4 : 4 : 4

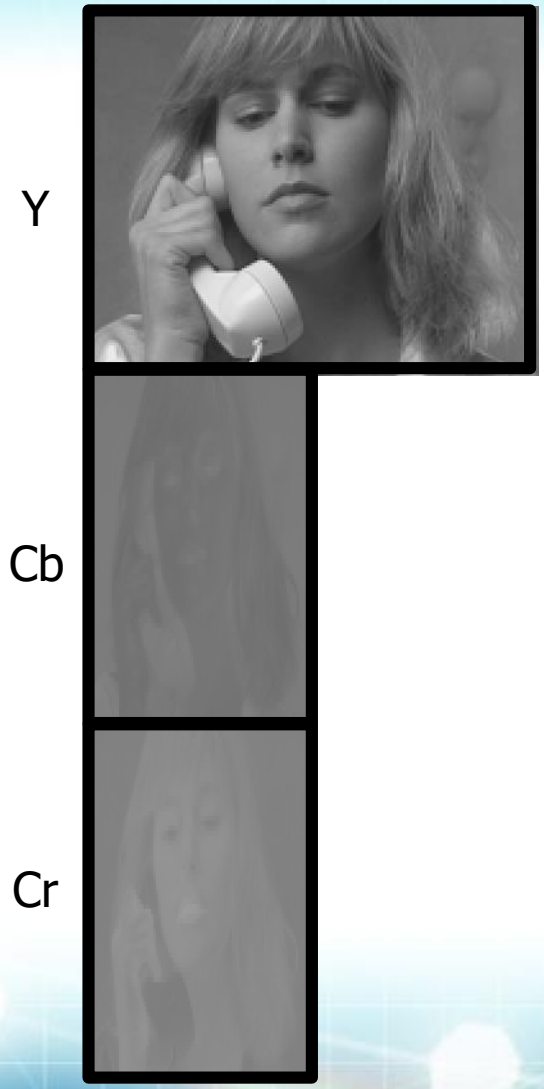


Subsampling

- 4:2:2 color component

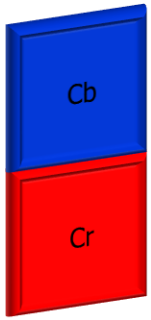


4 : 2 : 2



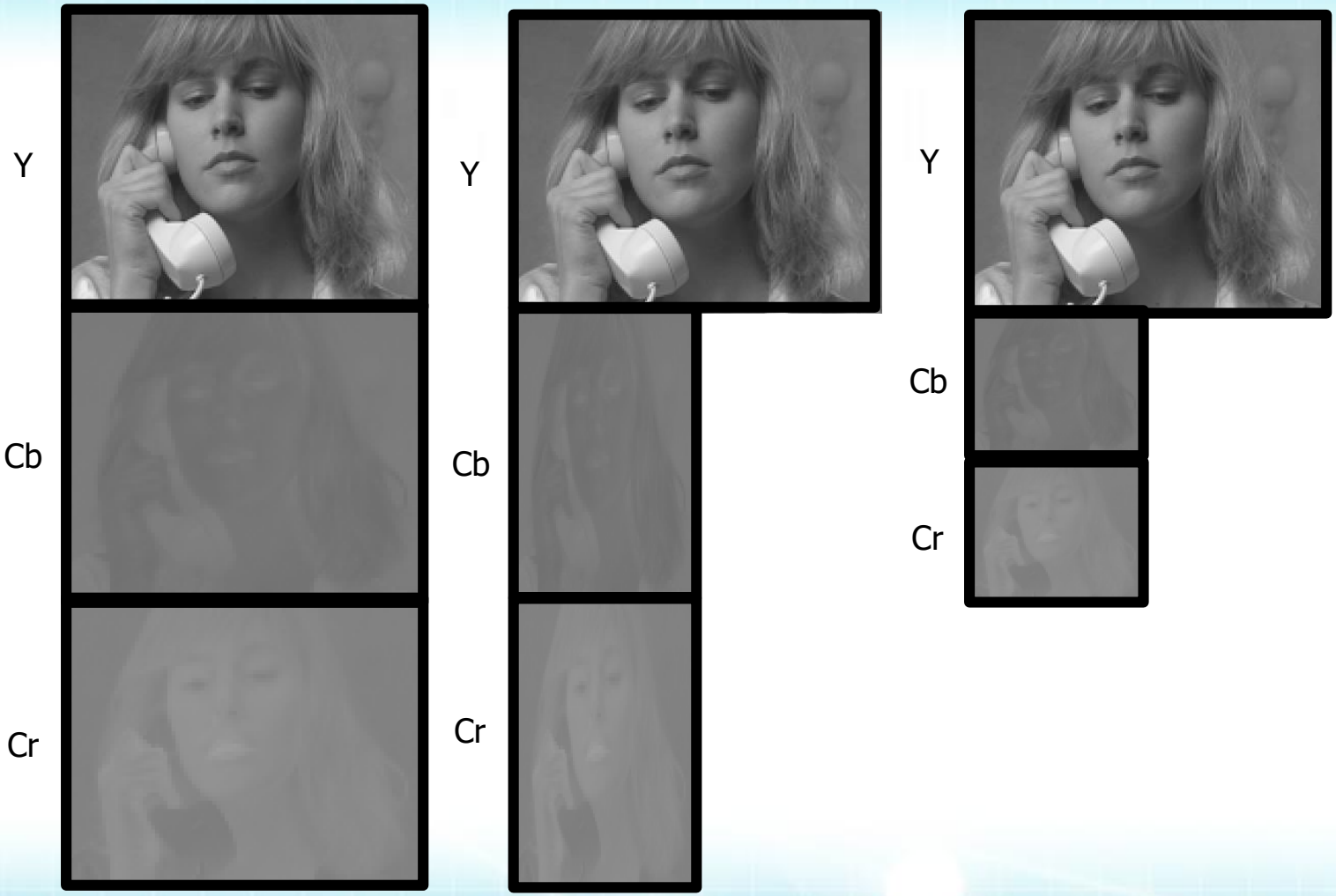
Subsampling

- 4:2:0 color component

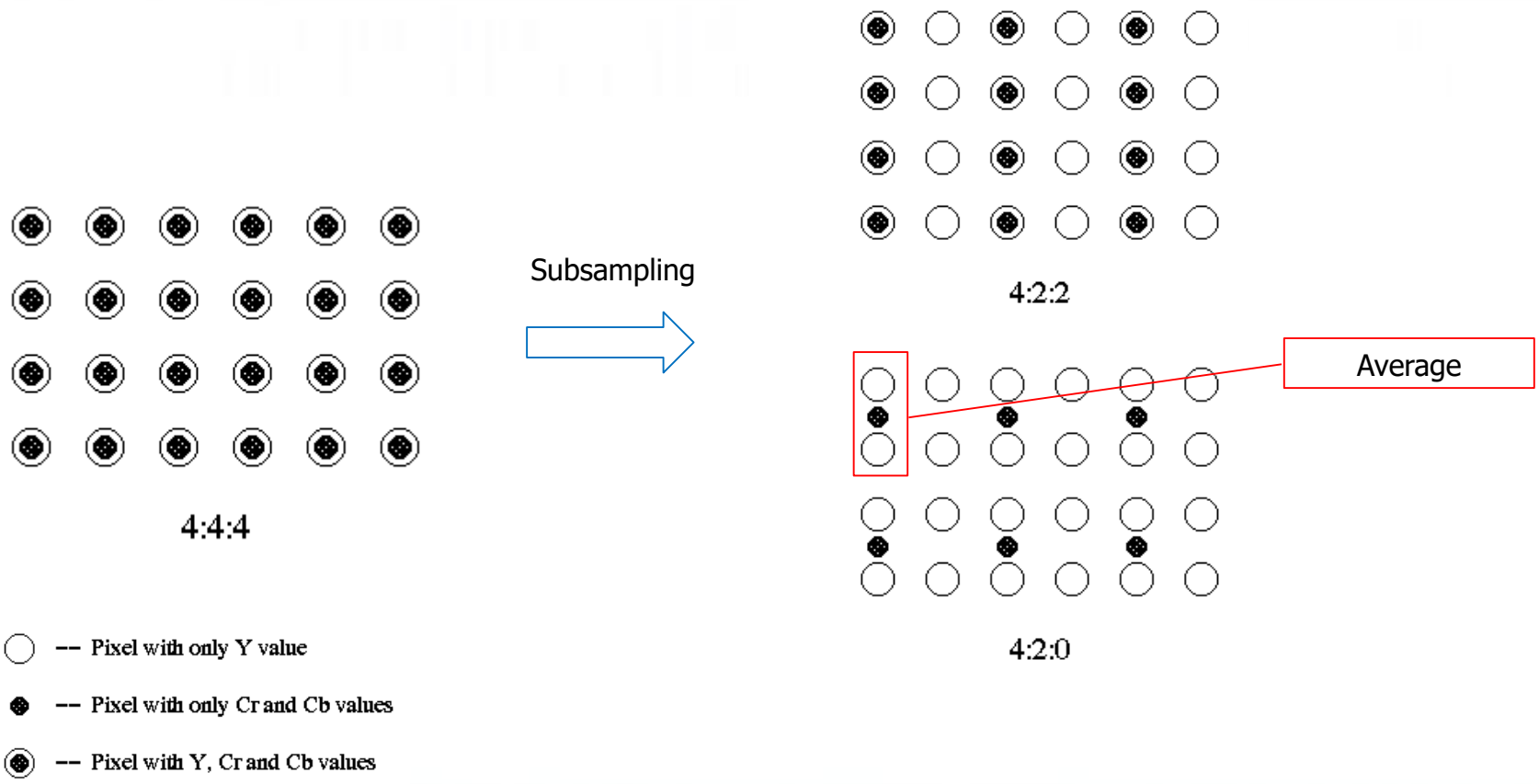


4 : 2 : 0

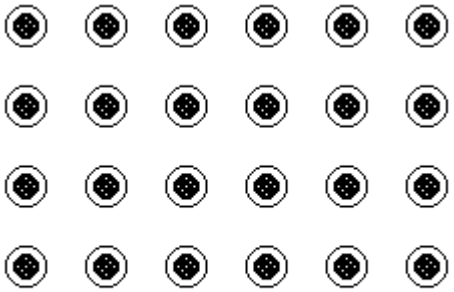
Subsampling



Subsampling

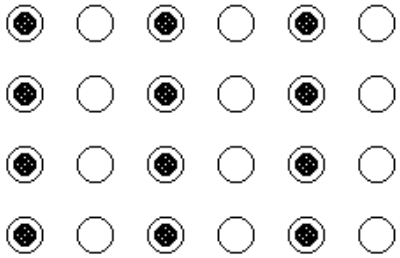


Subsampling

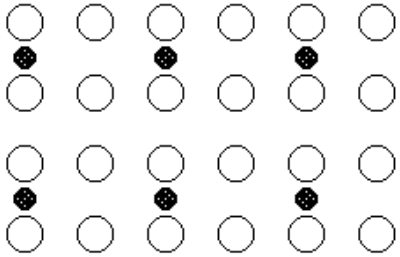


4:4:4

Copy the neighboring pixel



4:2:2



4:2:0

○ -- Pixel with only Y value

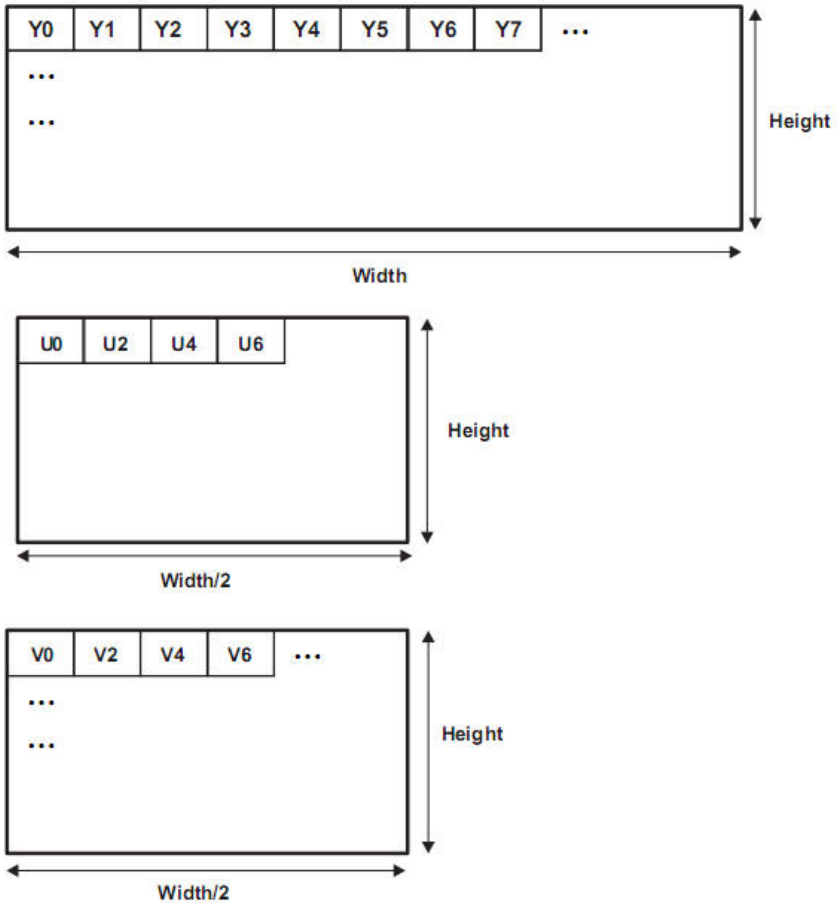
● -- Pixel with only Cr and Cb values

◉ -- Pixel with Y, Cr and Cb values

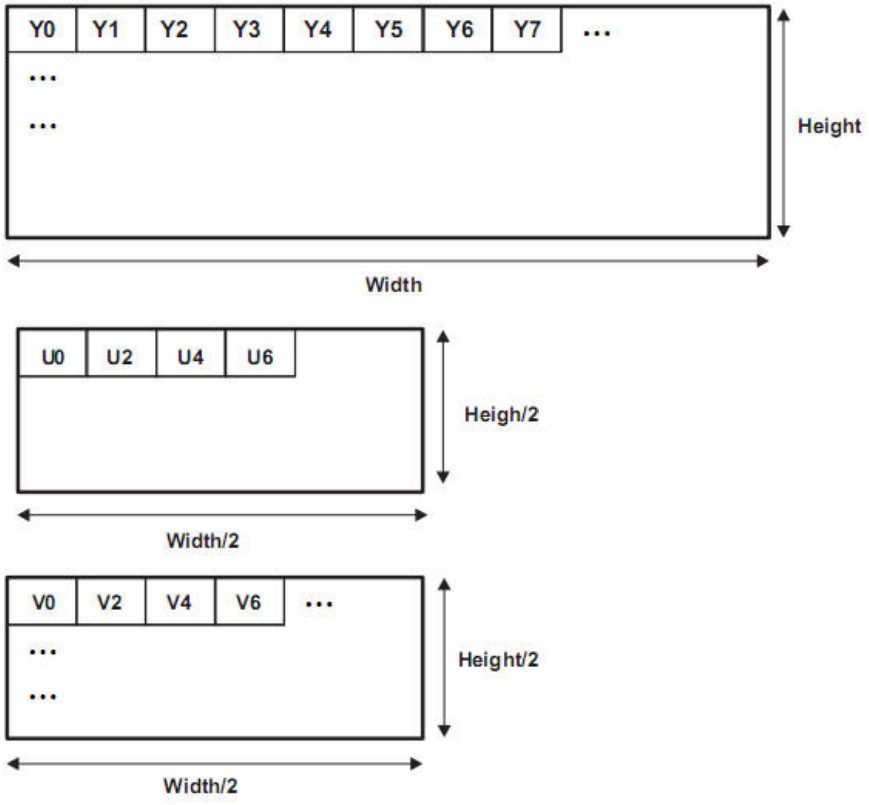
Subsampling

❖ File structure

4 : 2 : 2



4 : 2 : 0



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

```
    ////////////////////
    /*Processing Loop*/
    ////////////////////

    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)
{



}

void YUV_to_RGB(BYTE** img_in, BYTE** img_out, int width, int height)
{



}
```

Assignment 11 Color conversion

Assignment 11 Color conversion

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)
{

}

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

}
```

Assignment 12 Video Subsampling

Assignment 12 Video Subsampling

Example code

```
// YUV 444 -> YUV 422
void YUV444_to_422(BYTE** img_in, BYTE** img_Y, BYTE** img_U422, BYTE** img_V422, int width, int height)
{
```

Assignment 12 Video Subsampling

```
}

// YUV 422 -> YUV 444
void YUV422_to_444(BYTE** img_Y, BYTE** img_U422, BYTE** img_V422, BYTE** img_out, int width, int height)
{
```

Assignment 12 Video Subsampling

```
}
```