**Image processing**

**DIGITAL IMAGE FORMAT INTRODUCTIONS**

An image can also be described in terms of vector graphics or raster graphics. An image stored in raster form is sometimes called a bitmap. An image map is a file containing information that associates different locations on a specified image with hypertext links.

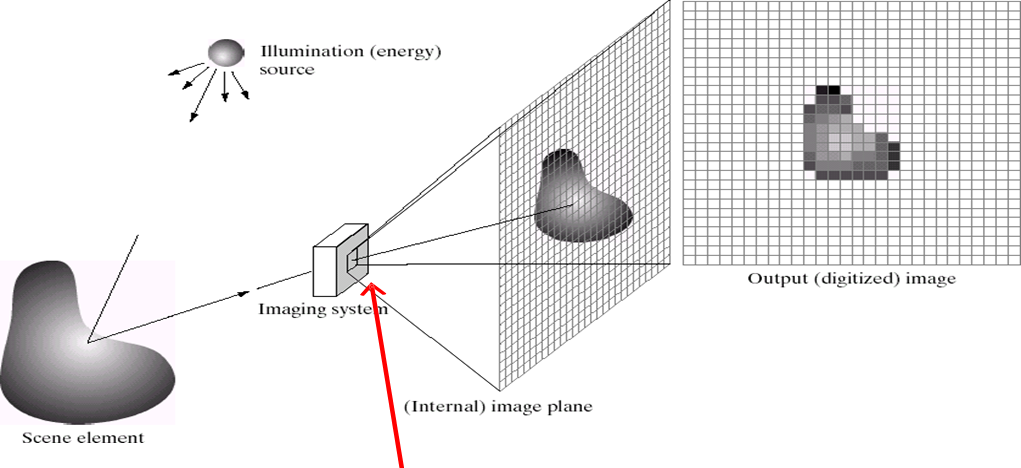
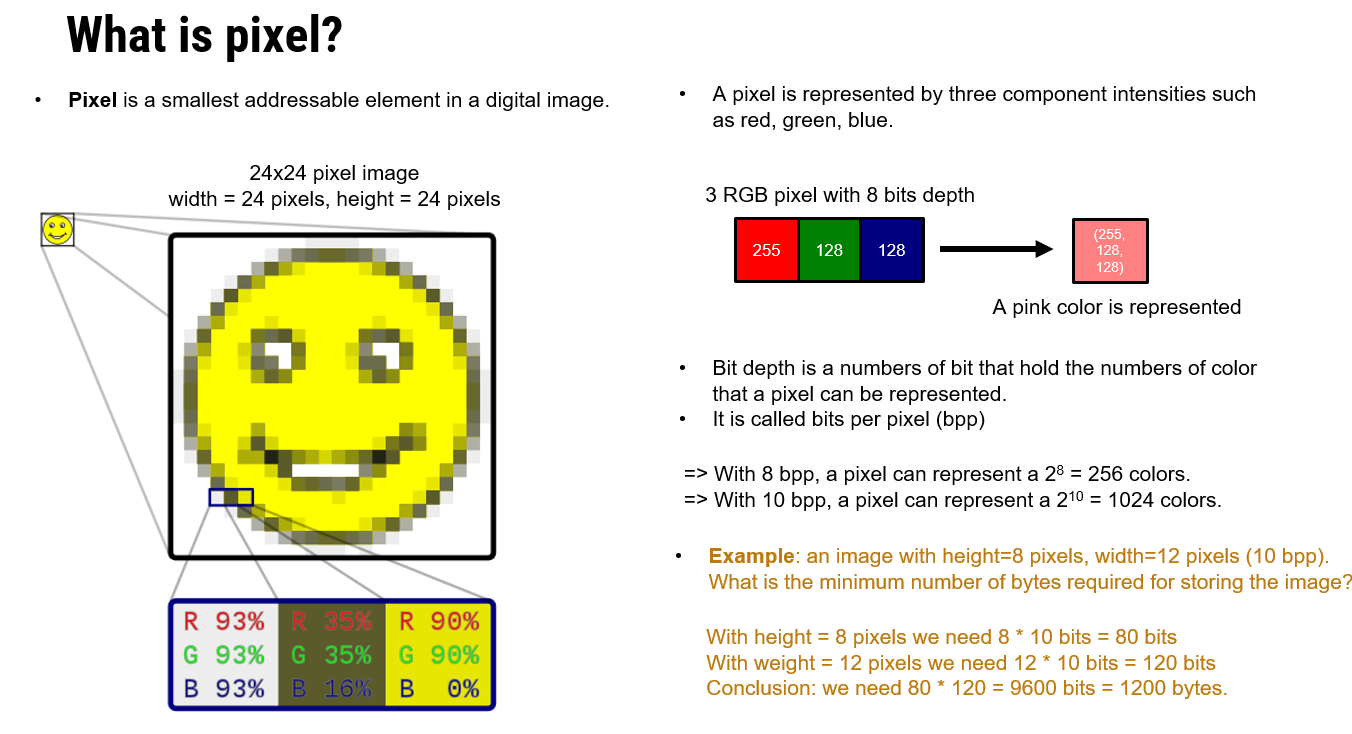


Figure 1: camera convert light to image

**Image format**

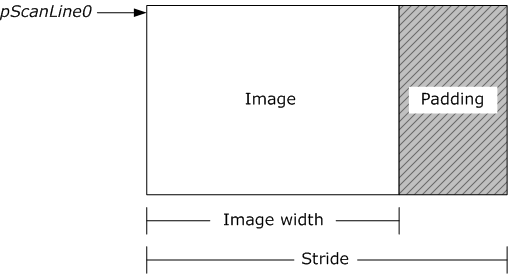
* Hundreds of image file formats. Examples
* Tagged Image File Format (**TIFF**)
* Graphics Interchange Format (**GIF**)
* Portable Network Graphics (**PNG**)
* **JPEG**, **BMP**, Portable Bitmap Format (**PBM**), etc
* Image pixel values can be
* **Grayscale**: 0 – 255 range
* **Binary**: 0 or 1
* **Color**: RGB colors in 0‐255 range (or other color model)
* **Application specific** (e.g. floating point values in astronomy)



!with binary image we have 1bit per pixel, but black and white needs upto 8bpp

**What is stride**

The stride is the number of bytes from one row of pixels in memory to the next row of pixels in memory. Stride is also called pitch. If padding bytes are present, the stride is wider than the width of the image, as shown in the following illustration. <https://learn.microsoft.com/en-us/windows/win32/medfound/image-stride>



Reason for stride:

When we have a matrix of pixel, for example . But if the data will store in a 1 line array (or another way that different with 3x3)  we will need ‘stride’ to tell computer where is the end of every line (3, 6, 9)

In short, stride means how many bytes per row (padding included)

<https://swiftunboxed.com/internals/size-stride-alignment/>



In which:

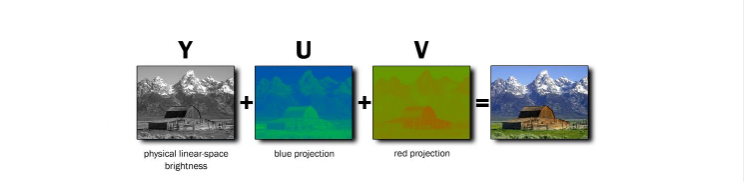
* width is the number of pixels on each row
* bpp : bits per pixel
* [] is round up.

We assump that our image running on 32-bit architech (4 bytes) so to meet alignment in data structure

NV16 will devide into 2 planar so [bpp is 8](https://www.kernel.org/doc/html/latest/userspace-api/media/v4l/pixfmt-yuv-planar.html)

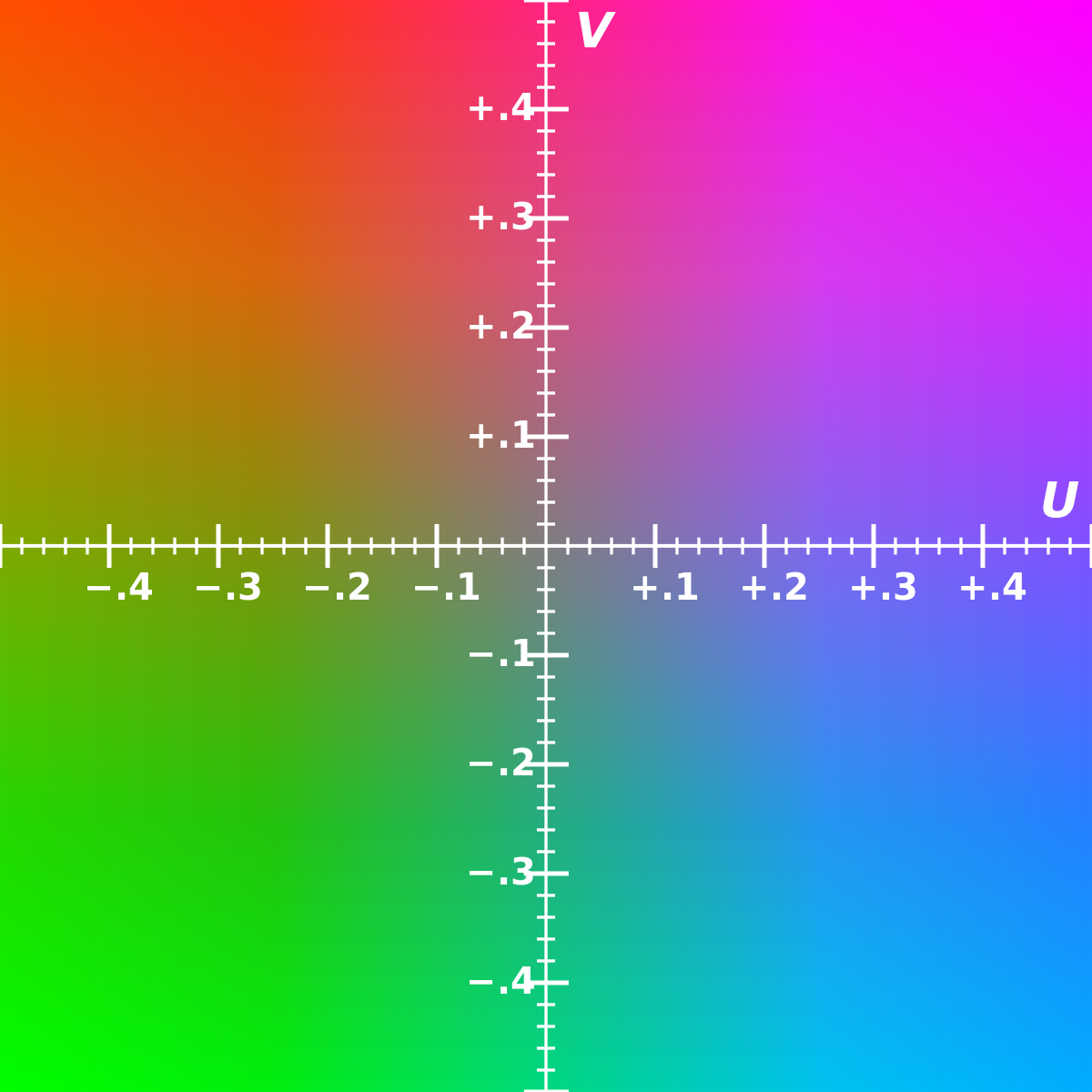
[RGB24](https://learn.microsoft.com/en-us/dotnet/api/system.windows.media.pixelformats.rgb24?view=windowsdesktop-8.0) bpp is 24

**What is YUV**:



1. Y (Luminance): This component represents the brightness or intensity of the image. It contains grayscale information and determines the overall brightness level of the picture.
2. U (Chrominance Blue): This component represents the blue color information. It describes the difference between the luminance and the blue component.
3. V (Chrominance Red): This component represents the red color information. It describes the difference between the luminance and the red component.

Y: brightness of that pixel (grayscale)

UV:  UV only displays tone color, not the brightness so that color light dark blue or light red is not here. U-axis calls blue axis and V-axis is red axis

* Why YUV is still be used in these day:

if goal is compressing video down to manageable bitrates with something simple enough that it can be mostly decoded in hw, YUV420. In terms of workloads, videos and JPEG photographs are typically in YUV. Mostly for efficiency and due to legacy/backwards compatibility reasons. PNG (graphics) are stored in RGB, as they are often used for graphics or charts for web use, where we want something lossless/best quaility.

* **Planar format**:
  + Semi-planar: first plane contain lyma (Y), the second plane contain chroma (like Cr and Cb, 2 of them stay seperately). EX: NV12, NV16 -> suit for graphic card

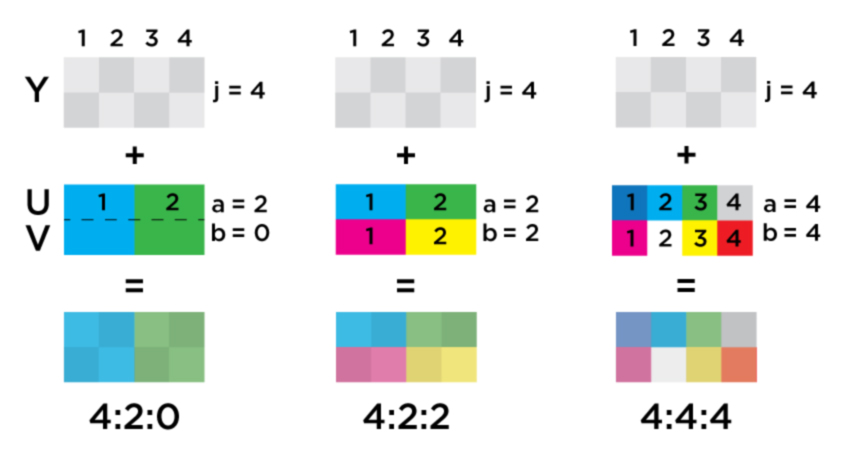
These formats are commonly referred to as NV formats (NV12, NV16, ...). They use two planes, and store the luma components in the first plane and the chroma components in the second plane. The Cb and Cr components are interleaved in the chroma plane, with Cb and Cr always stored in pairs. The chroma order is exposed as different formats.

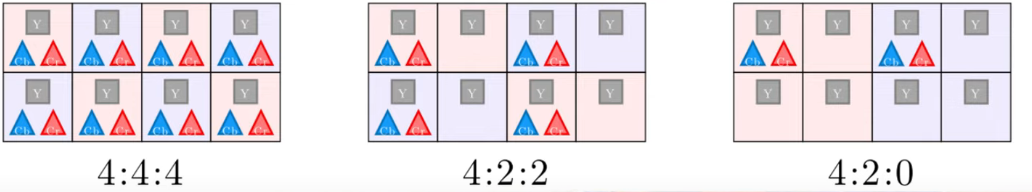
* + Fully planar: Use 3 plane to contain luma (Y), chroma red (Cr), chroma blue (Cb). EX: I420, I422, I444 -> suit for compressing video
  + Within a plane, components are stored in pixel order, which may be linear or tiled. Padding may be supported at the end of the lines, and the line stride of the chroma planes may be constrained by the line stride of the luma plane.
  + If there is no note or suffix, all plane need to stay together
* **Chroma subsampling** (reduce file size)

In this technique, we compress information in chroma and keep luma (due to human eyes are more sensitive to luma changes). This will reduce file size up to 50% while still keep picture clarity

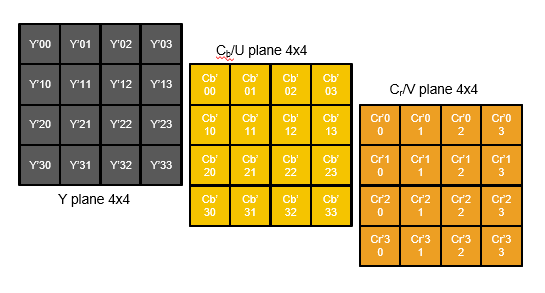
This technique usually come with notation J:a:b for 2x4 pixel. In which:

* J: amount of luma samples being taken
* a: samples are taken in the upper row of pixels
* b: how many samples are taken in the lower row





Most video display at 4:2:0, some camera record at 4:2:2, some top-notch game display at 4:4:4



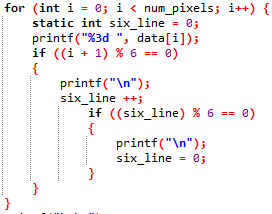
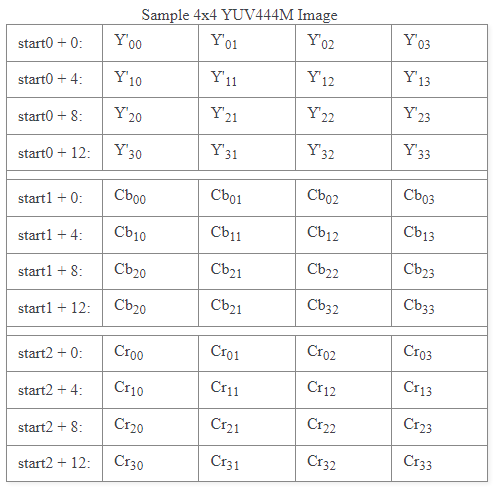
YUV444 need 3 byte per pixel to store data

While NV16 need 2 bytes to store data per pixel

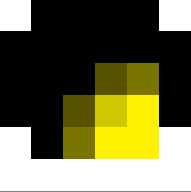
**Demo output for png file 6x6 pixel in YUV444 and NV16 (YUV422); convert yuv444 to nv16 (in ex1 down below) using C program:**

<https://www.kernel.org/doc/html/latest/userspace-api/media/v4l/pixfmt-yuv-planar.html>

<https://blog.csdn.net/fanbird2008/article/details/8232673>

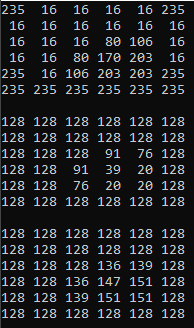
 

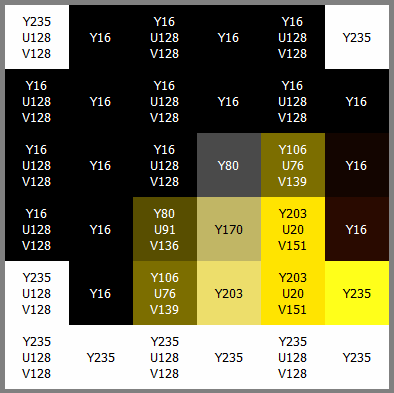
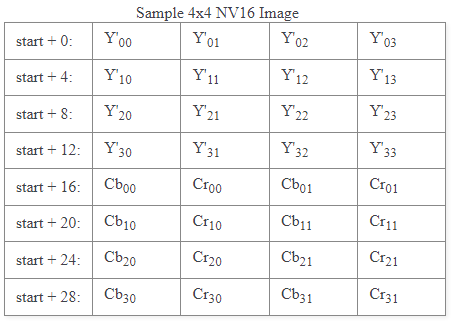
With this sample for 4x4, we can see that amount of data is 3\*(amount of pixel, in this example is 6\*6 = 36 pixel). We use this C program to see the result and compare with YUView

Sample file: 

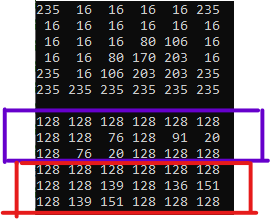
PNG file (in RGB): 

YUV444: 

 Data store in .yuv file devided into 3 part respectively : Y -> U -> V

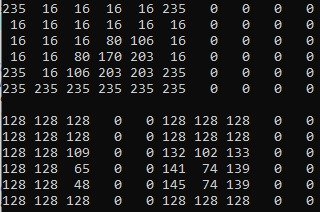
NV16:  

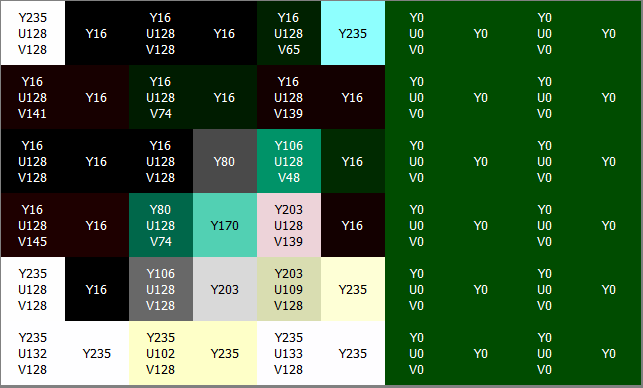
As we can see, NV16 only have half of chroma info compare to YUV444 so it needs 2\*(amount of pixels) to store image data

 So the first part is the first 6 line that keep all Y(luma) info, coming right next to that is the blue part – which is Cb (chroma blue) or U (blue axis); and the last part is red zone – which is Cr (chroma red) or V (red axis)

We can see that the color in NV16 (YUV422) is not completely the same with YUV444 and PNG file (these file need a tons of storage to store color)

Stride in NV16, for example we use above sample to add padding = 10





Ex1:

Here is the command to convert png file to yuv

/d/github/ffmpeg-6.1.1-essentials\_build/bin/ffmpeg.exe -i imageprocessing\_requirement1.png -pix\_fmt yuv444p imageprocessing\_requirement1\_yuv444p.yuv

List of pixel format that ffmpeg support: <https://ffmpeg.org/pipermail/ffmpeg-devel/2007-May/035617.html> (old)

<https://gist.github.com/dericed/3319386>

ffmpeg -pix\_fmts

Pixel formats:

I.... = Supported Input format for conversion

.O... = Supported Output format for conversion

..H.. = Hardware accelerated format

...P. = Paletted format

....B = Bitstream format

FLAGS NAME NB\_COMPONENTS BITS\_PER\_PIXEL BIT\_DEPTHS

-----

IO... yuv420p 3 12 8-8-8

IO... yuyv422 3 16 8-8-8

IO... rgb24 3 24 8-8-8

IO... bgr24 3 24 8-8-8

IO... yuv422p 3 16 8-8-8

IO... yuv444p 3 24 8-8-8

IO... yuv410p 3 9 8-8-8

IO... yuv411p 3 12 8-8-8

……

IO... yuva444p16le 4 64 16-16-16-16

..H.. vdpau 0 0 0

IO... xyz12le 3 36 12-12-12

IO... xyz12be 3 36 12-12-12

IO... nv16 3 16 8-8

..... nv20le 3 20 10-10-10

..... nv20be 3 20 10-10-10

IO... rgba64be 4 64 16-16-16-16

IO... rgba64le 4 64 16-16-16-16

IO... gbrap14le 4 56 14-14-14-14

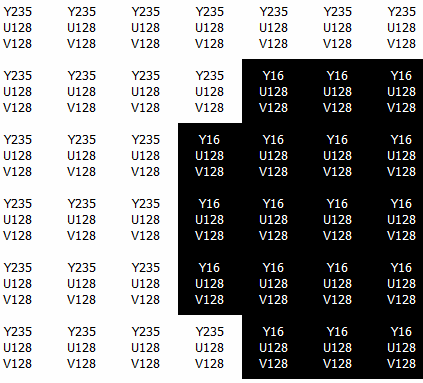
……

Update: ffmpeg now have input nv16 available

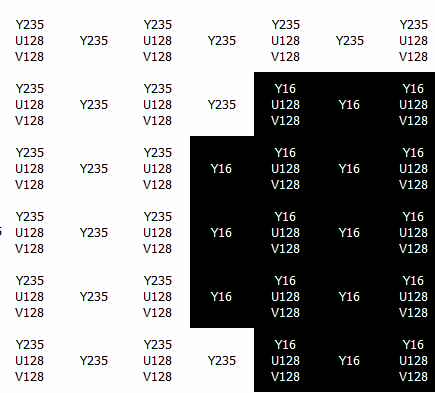
<https://patchwork.ffmpeg.org/project/ffmpeg/patch/20220601203337.1116134-1-matthieu.bouron@gmail.com/>

But in this exercise we will use C program to convert yuv444p to nv16 (yuv422)

Take part of image to compare, this is yuv444:



This is nv16:



Ex2:

We will use this formular to calculate stride for NV16

Because things inside brackets will be calculated first -> We can round up the width to align with align by add (align – 1) first and then devide by align

For example, compare between width = 0 and width = 1, align is 256

Width = 0, -> then devided by 256 lead to result is 0 -> stride =0

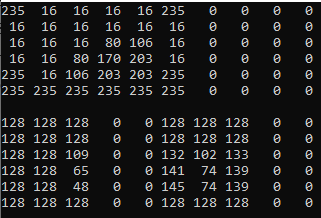
Width = 1, -> then devided by 256 lead to result is 1 -> stride = 256

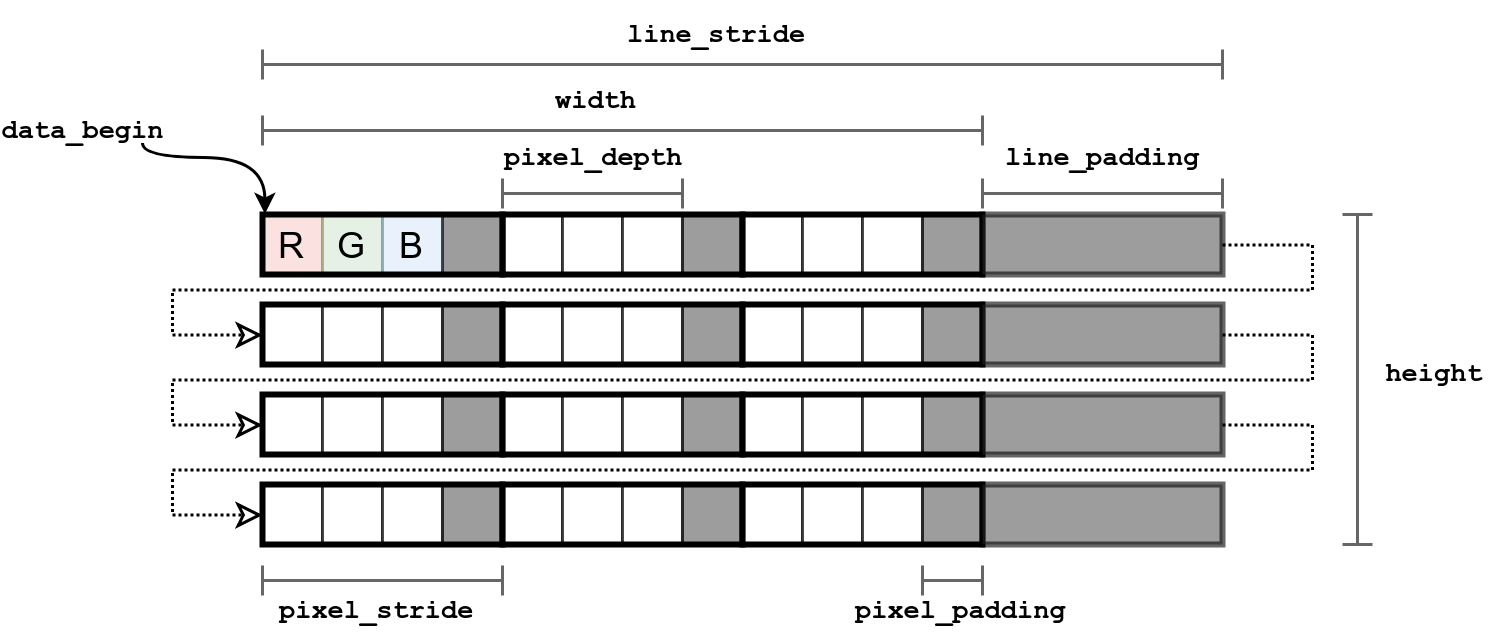
So that stride will be align with 256

In the second part, we will add padding to cover the rest of size that doesn’t have image data by 0x0.

* With NV16 (YUV422): this type is devided into 3 part
  + First part is Y with 100% width length
  + Second part is Cb with 50% width length
  + Last part is Cr witg 50% width length

So we need to use for loop and fread to insert Y into first part to the left, Cb and Cr will insert half of width + half of Y padding



**Ex2: rgb24  
**In RGB24 these is a little bit different in stride: each pixel need 3 color data red green blue in a row (instead of in a plane in YUV) -> stride need triple compare to above

And the data to print out in each row also equal to 3 times of pixel in a row

**Appendix**

C program to check data and convert from yuv444 to nv16 Ex1

#include <stdio.h>

#include <stdlib.h>

int main() {

    FILE \*file;

    unsigned char \*data;

    int width = 3856;

    int height = 1964;

    int num\_pixels = width \* height \* 3; // time with plane (YUV444 has 3 plane), nv16 has 2

    // open file to convert

    file = fopen("imageprocessing\_requirement1\_yuv444p.yuv", "rb");

    if (!file) {

        fprintf(stderr, "Cannot open\n");

        return 1;

    }

    // allocate mem

    data = (unsigned char \*)malloc(num\_pixels);

    if (!data) {

        fprintf(stderr, "error\n");

        fclose(file);

        return 1;

    }

    // read

    fread(data, sizeof(unsigned char), num\_pixels, file);

    // close

    fclose(file);

    // create output file

    FILE \*output\_file;

    unsigned char \*data\_out = (unsigned char \*)malloc(num\_pixels\*2/3);//only need 2/3 compare to yuv444

    output\_file = fopen("imageprocessing\_requirement1\_nv16.yuv", "wb");

    if (output\_file == NULL) {

        perror("Failed to open output file");

        fclose(output\_file);

        return 1;

    }

    //write to new file

        for (int i = 0; i < num\_pixels; i++) {

        static int data\_out\_index = 0;

        if(i<(num\_pixels/3)){

            data\_out[data\_out\_index]=data[i];

            data\_out\_index++;

        } else {

            if(i%2==0){

                data\_out[data\_out\_index]=data[i];

                data\_out\_index++;

            }

        }

    }

    // write data\_out into ouput\_file

    fwrite(data\_out, sizeof(unsigned char), num\_pixels\*2/3, output\_file);

    fclose(output\_file);

    // release

    free(data);

    free(data\_out);

    return 0;

}

**Apendix**

Add padding nv16

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

//this code is to add padding and calculate stride and size

int main() {

    int width = 3856; // replace by image width

    int height = 1964; // replace by image height

    int bpp = 8; // bits per pixel for: NV16 format - 8, RGB24 - 24

    int align = 256; //must align with \_\_ value

    int num\_of\_planar = 2;//nv16 has 2, rgb24 has 3

    // Calculate stride, align-1 is to round up stride to align with 256

    int stride = ((width + align-1) / align)\*align;

    // Calculate size (in bytes) and print out stride and size - EX2\_1

    int size = stride \* height \* num\_of\_planar;

    printf("Stride: %d bits\n", stride);

    printf("Size: %d bits\n", size);

//EX2\_2 :

    // Allocate memory for the buffer: 'image' - which align with 'stride' inside 'size'

    unsigned char \*image = malloc(size);

    if (image == NULL) {

        printf("Failed to allocate memory for image\n");

        return 1;

    }

    // Initialize the image data to 0

    memset(image, 0x0, size);

    // Open the NV16 file

    FILE \*file = fopen("imageprocessing\_requirement1\_nv16.yuv", "rb");

    if (file == NULL) {

        printf("Failed to open file\n");

        free(image);

        return 1;

    }

    // Read the image data from the file

    for (int i = 0; i < height; ++i) {

        if (fread(image + i \* stride, 1, width, file) != width) {

            printf("Failed to read image data from file\n");

            free(image);

            fclose(file);

            return 1;

        }

    }

    for (int i = height; i < height\*num\_of\_planar; ++i) {

        if (fread(image + i \* stride, 1, width/2, file) != width/2) {

            printf("Failed to read image data from file2\n");

            free(image);

            fclose(file);

            return 1;

        }

        if (fread(image + i \* stride + width/2 + (stride-width+1)/2, 1, width/2, file) != width/2) {

            printf("Failed to read image data from file3\n");

            free(image);

            fclose(file);

            return 1;

        }

    }

    // Close the input file

    fclose(file);

    // Open the output file

    file = fopen("imageprocessing\_requirement1\_nv16\_padded.yuv", "wb");

    if (file == NULL) {

        printf("Failed to open output file\n");

        free(image);

        return 1;

    }

    // Write the image data to the file

    if (fwrite(image, 1, size, file) != size) {

        printf("Failed to write image data to file\n");

    }

    // Clean up

    free(image);

    fclose(file);

    return 0;

}

Add padding to rgb24

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

//this code is to add padding and calculate stride and size

int main() {

    int width = 3856; // replace by image width

    int height = 1964; // replace by image height

    int bpp = 8; // bits per pixel for: NV16 format - 8, RGB24 - 24

    int align = 256; //must align with \_\_ value

    int num\_of\_planar = 2;//nv16 has 2, rgb24 has 3

    // Calculate stride, align-1 is to round up stride to align with 256

    int stride = ((width + align-1) / align)\*align;

    // Calculate size (in bytes) and print out stride and size - EX2\_1

    int size = stride \* height \* num\_of\_planar;

    printf("Stride: %d bits\n", stride);

    printf("Size: %d bits\n", size);

//EX2\_2 :

    // Allocate memory for the buffer: 'image' - which align with 'stride' inside 'size'

    unsigned char \*image = malloc(size);

    if (image == NULL) {

        printf("Failed to allocate memory for image\n");

        return 1;

    }

    // Initialize the image data to 0

    memset(image, 0x0, size);

    // Open the NV16 file

    FILE \*file = fopen("imageprocessing\_requirement1\_nv16.yuv", "rb");

    if (file == NULL) {

        printf("Failed to open file\n");

        free(image);

        return 1;

    }

    // Read the image data from the file

    for (int i = 0; i < height; ++i) {

        if (fread(image + i \* stride, 1, width, file) != width) {

            printf("Failed to read image data from file\n");

            free(image);

            fclose(file);

            return 1;

        }

    }

    for (int i = height; i < height\*num\_of\_planar; ++i) {

        if (fread(image + i \* stride, 1, width/2, file) != width/2) {

            printf("Failed to read image data from file2\n");

            free(image);

            fclose(file);

            return 1;

        }

        if (fread(image + i \* stride + width/2 + (stride-width+1)/2, 1, width/2, file) != width/2) {

            printf("Failed to read image data from file3\n");

            free(image);

            fclose(file);

            return 1;

        }

    }

    // Close the input file

    fclose(file);

    // Open the output file

    file = fopen("imageprocessing\_requirement1\_nv16\_padded.yuv", "wb");

    if (file == NULL) {

        printf("Failed to open output file\n");

        free(image);

        return 1;

    }

    // Write the image data to the file

    if (fwrite(image, 1, size, file) != size) {

        printf("Failed to write image data to file\n");

    }

    // Clean up

    free(image);

    fclose(file);

    return 0;

}