COMP 7503 programming assignment

Name: Ng Argens
University number: 3035072143

Name: Leung Hui Homn
University number: 2005279759

Programming task 1:

Conversion of RGB to YUV and YUV to RGB

Discussion of the source code in C++:
Conversion of RGB to YUV

```
int i, j;
    unsigned char r, g, b;

for(j = 0; j < height; j++) {
        for(i = 0; i < width; i++) {
            b = pRGB[(i + j * width) * 3] ;
            g = pRGB[(i + j * width) * 3 + 1] ;
            r = pRGB[(i + j * width) * 3 + 2] ;

            pYUV[(i + j * width) * 3 + 0] = unsigned char (0.299000 * r + 0.587000 * g +
0.114000 * b);

            pYUV[(i + j * width) * 3 + 1] = unsigned char (-0.168736 * r -0.331264 * g +
0.500000 * b + 128);

            pYUV[(i + j * width) * 3 + 2] = unsigned char (0.500000 * r -0.418688 * g -0.081312
* b + 128);
```

Conversion of YUV to RGB

```
    int i, j ;
    unsigned char y, u, v;
    float f;

    for(j = 0; j < height; j++) {
        for(i = 0; i < width; i++) {

            y = pYUV[(i + j * width) * 3 + 0];
            u = pYUV[(i + j * width) * 3 + 1];
            v = pYUV[(i + j * width) * 3 + 2];

            f = y + 1.4075 * (v - 128);

pRGB[(i + j * width) * 3 + 2] = unsigned char (f < 0 ? 0 : f > 255 ? 255 : f); //r

            f = y - 0.3455 * (u - 128) - 0.7169 * (v - 128);

pRGB[(i + j * width) * 3 + 1] = unsigned char (f < 0 ? 0 : f > 255 ? 255 : f); //g
```

```
                    f = y + 1.7790 * (u - 128);

pRGB[(i + j * width) * 3 + 0] = unsigned char (f < 0 ? 0 : f > 255 ? 255 : f); //b

                }
         }
```

Key points of the source code:
1. Conversion formula used

R = Y + 1.4075 * (V - 128)
G = Y - 0.3455 * (U - 128) - (0.7169 * (V - 128))
B = Y + 1.7790 * (U - 128)

Y = R * .299000 + G * .587000 + B * .114000
U = R * -.168736 + G * -.331264 + B * .500000 + 128
V = R * .500000 + G * -.418688 + B * -.081312 + 128

The above formula are what we used for our conversion of RGB to YUV and YUV to RGB. It is sourced from a color space programming website (http://softpixel.com/~cwright/programming/colorspace/yuv/)

2. Adjustment for YUV numbers
In the formula, we add 128 to convert the range of U and V to 0 to 255.

3. Setting boundary for RGB values during YUV to RGB conversion
As the Y, U and V values are not exact after rounding, the RGB values after the YUV to RGB conversion could be negative or higher than 255. Hence, we set boundaries for the RGB values from 0 to 255 by the statement of (unsigned char (f < 0 ? 0 : f > 255 ? 255 : f)) to avoid overflows or underflows.

Programming task 2:

Summary of the compression algorithm:
First, we use a filtering method adopted by PNG image file. This will transform the data into another set of data which has a skewed distribution to zero. Then, we adopt run-length encoding to compress the data.

Key points of the source code:
1. Five predictors used per scan line
Like PNG compression method, we used five predictors per scan line.

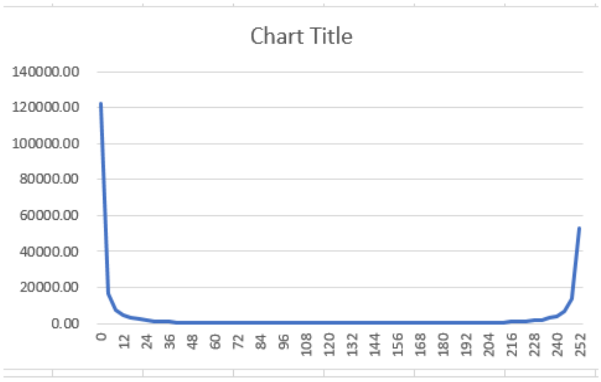
a. **None** – the raw byte passes through unaltered
b. **Sub** – Based on difference between itself and the pixel on its left (Pixel A)
c. **Up** – Based on difference between itself and the pixel above (Pixel B)
d. **Average** – Based on difference between itself and the mean of the pixel on the left and the pixel above
e. **Paeth** – A, B, or C, whichever is closest to p = A + B – C where C is the pixel on the top left

Note that we use squared difference to ensure the value of the differences is a positive number. We tested our transformed data. The output data has a skewed distribution to 0 and small negative numbers (denoted as 255 in the chart).

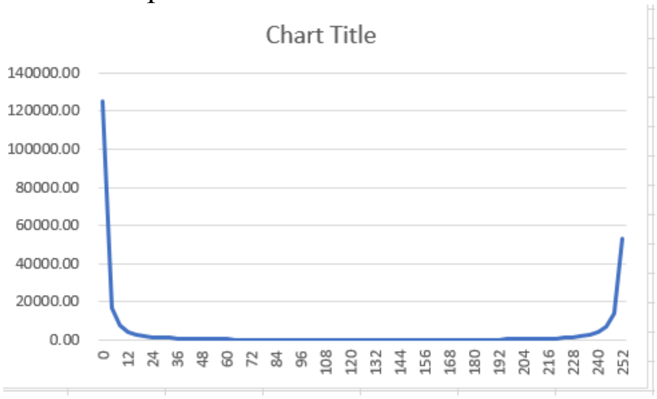
“Red Panda” image has the least skewed distribution presumably because its similarity across scanline is the least.

“Beach” image:

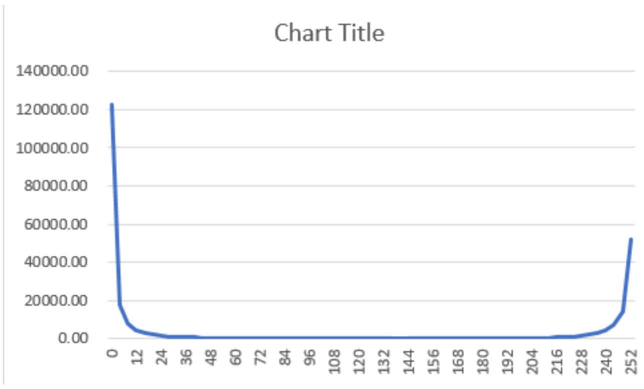
Blue component result



Green component result

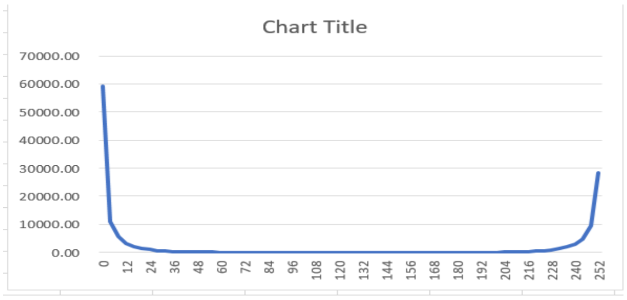


Red component result

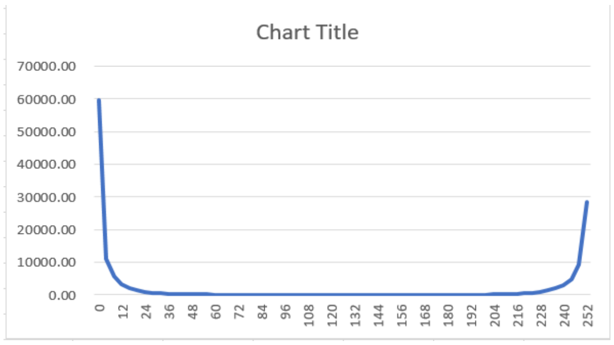


"Red panda" image:

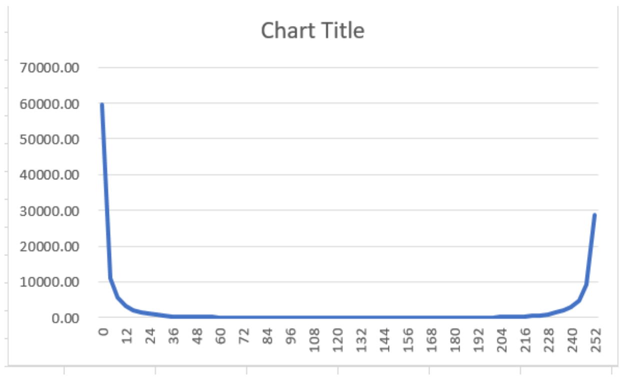
Blue component result



Green component result

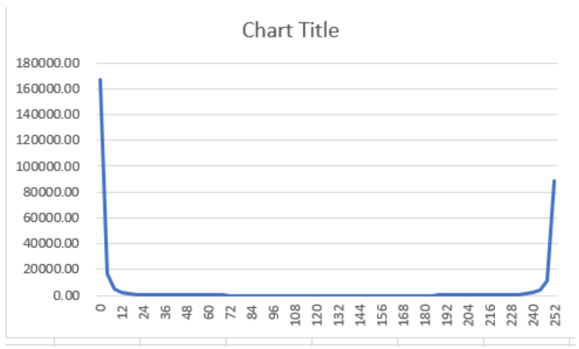


Red component result

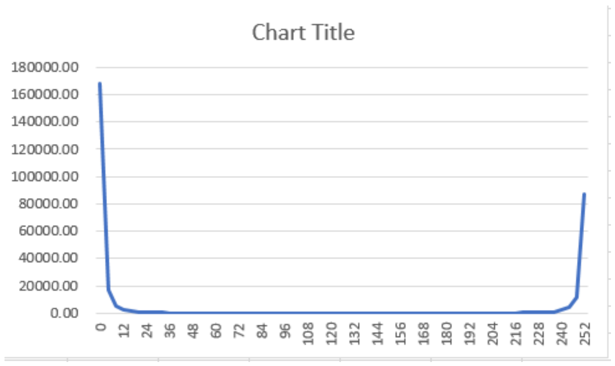


“Sunset” image:

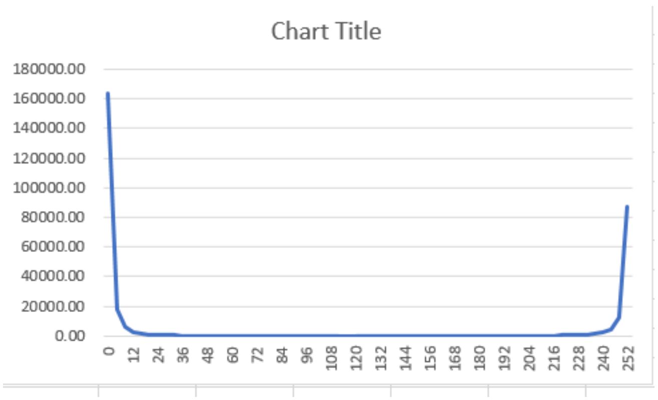
Blue component result



Green component reslt

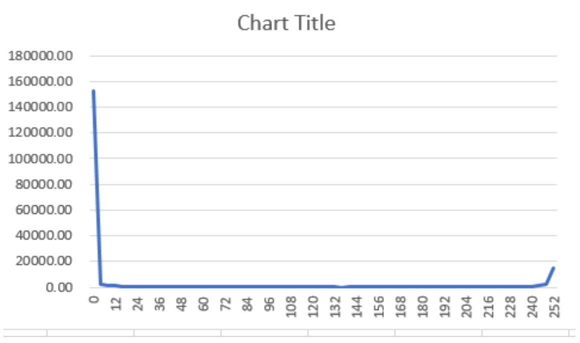


Red component result

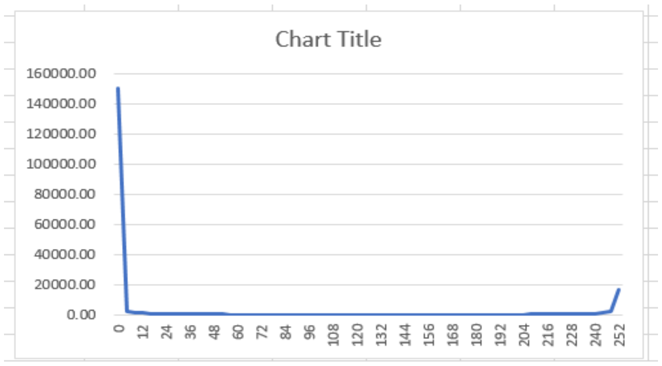


“Tuxinu” image:

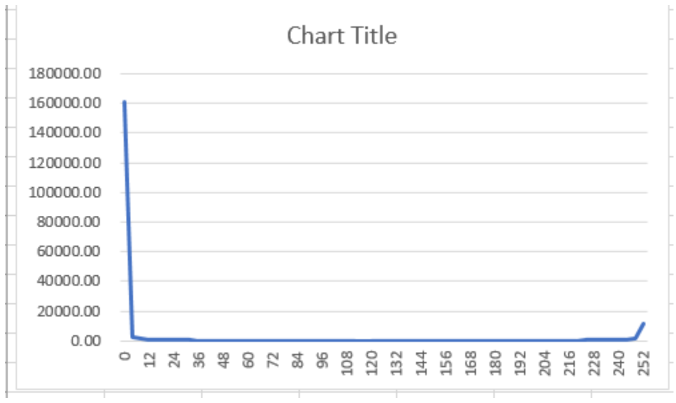
Blue component result



Green component result



Red component result



2. Bit operations by shift coding
To maximize compression efficiency, we added an algorithm to do shift coding.

3. Channel pre-processing
We compress the data for the R, G and B components separately. We name that in the respective channel Red, channel Green and channel Blue in our source code. This will make the data more compressible as each R, G and B components are more likely to have same or very similar values in an image.

Experiment 1

Summary and conclusion of analysis: Based on our analysis, converting RGB image into YUV image would be more efficient for image compression.

Summary of compression results of YUV and individual components:

	Beach	Red Panda	Sunset	Tuxinu
Original	1.53	1.5	2.04	2.29
Y	1.56	1.51	2.06	2.28
U	2.79	3.36	3.37	3.32
V	3.07	3.41	3.39	3.35

Based on our results, the YUV image has much higher compression ratio than RGB. On the other hand, the individual U and V components have higher compression ratio than the Y component. In our view, the higher compression ratio for YUV image was due to the higher compression rate for the U and V components of YUV.

Explanation of the results:

Separation of luma and chroma component likely to lead to more efficient compression

In each image, the luma component and the chroma components usually have their own distribution patterns. The luma component would depends on the intensity and angle of the light source while the chroma component would depend on the colour distribution pattern of the objects which reflected the lights. Hence, the luma components and the chroma components usually have a different set of repetitive patterns. If we seperate the luma and chroma component, the compression would likely to be more efficient.

However, as each RGB component shares part of the luma components and the chroma components, RGB is more likely to have a lower compression ratio.

YUV image could have lower resolution than RGB with limited impact to the image quality

As human eyes are more sensitive to luma components than chroma components, there would be limited impact to image quality if we reduce the chroma component reslution. For example, we could use various YUV sampling method such as 4:1:1, 4:2:0, 4:22 and 4:4:0 to reduce the chroma component resolutions. This would lead to an image of even smaller size after compression.

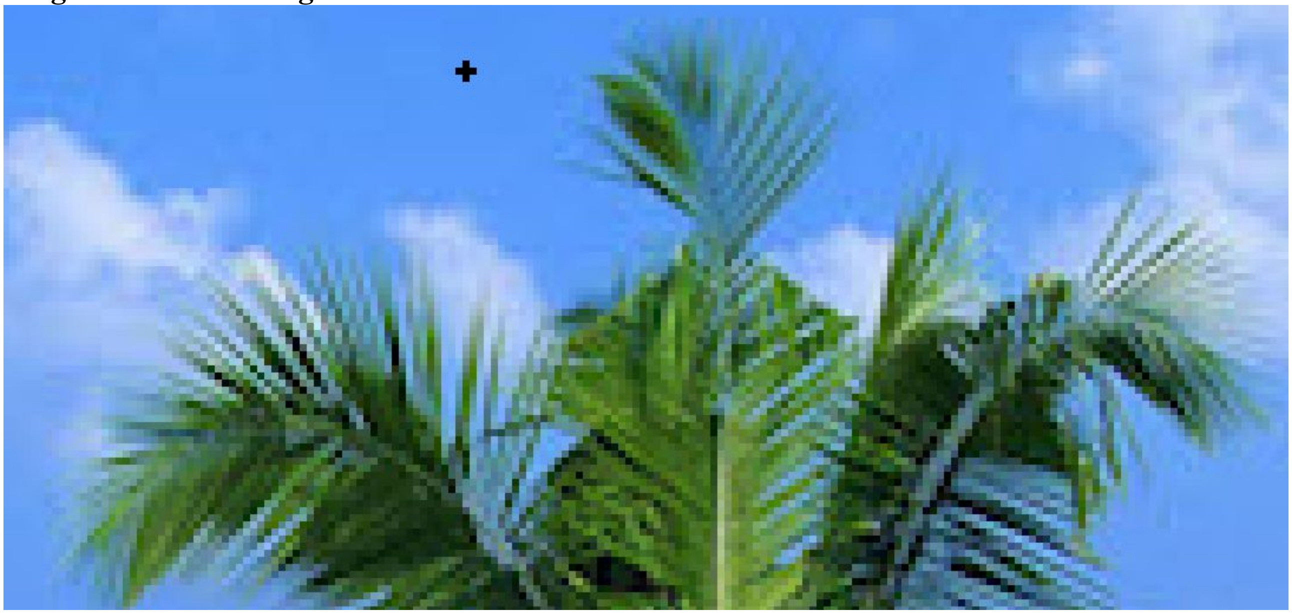
Experiment 2

Summary and conclusion of analysis: Based on our analysis, using 4:2:0 image would lead to lower image quality. There are artefacts on the fine edges of the images.

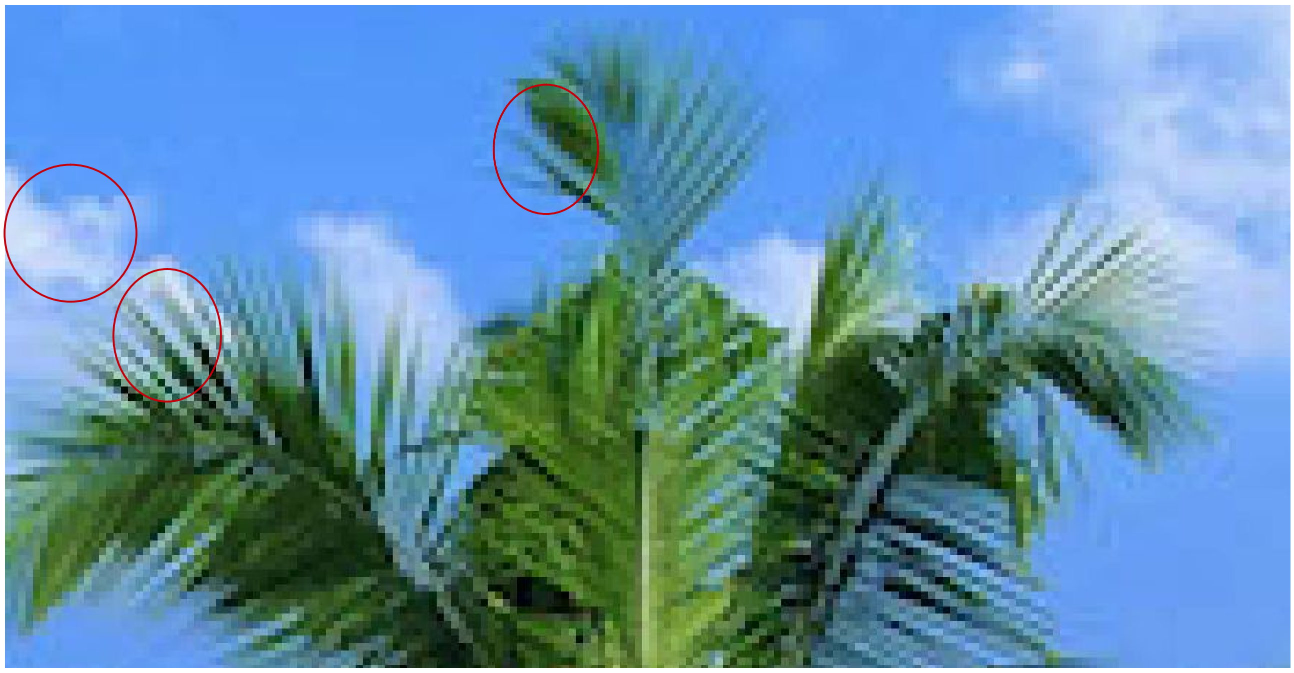
Comparison of images

The artefacts are highlighted in each of the photos under 4:2:0 resolution.

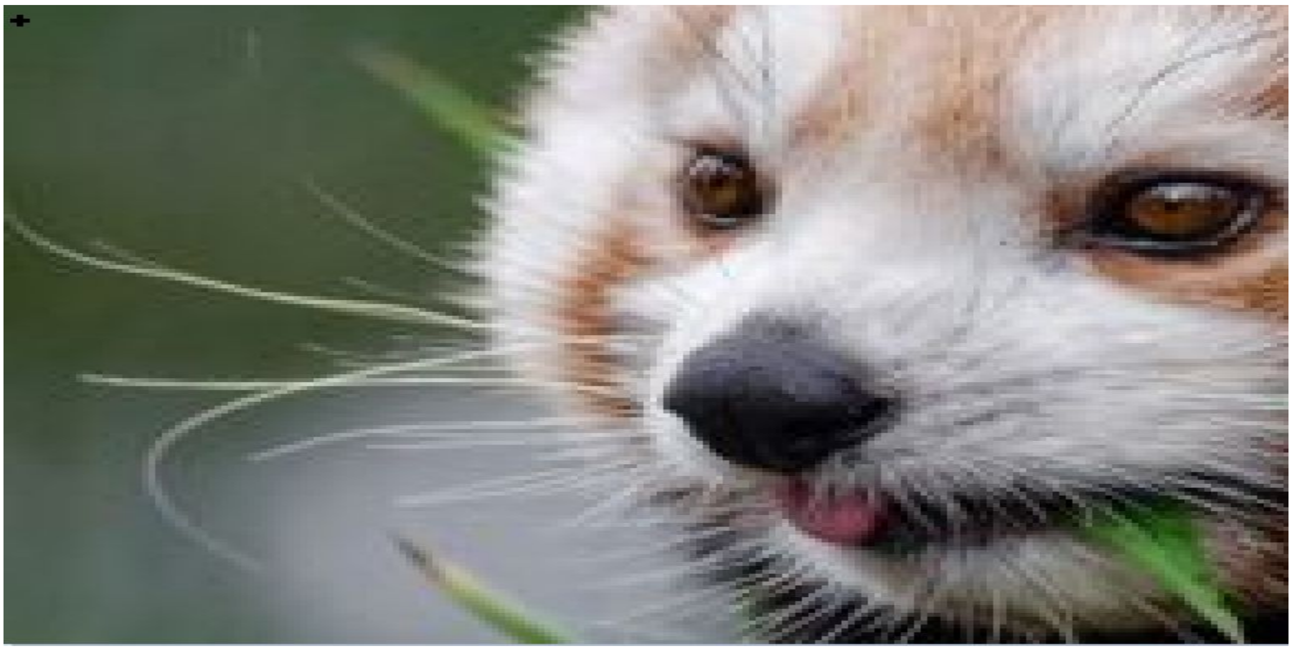
Original "beach" images:



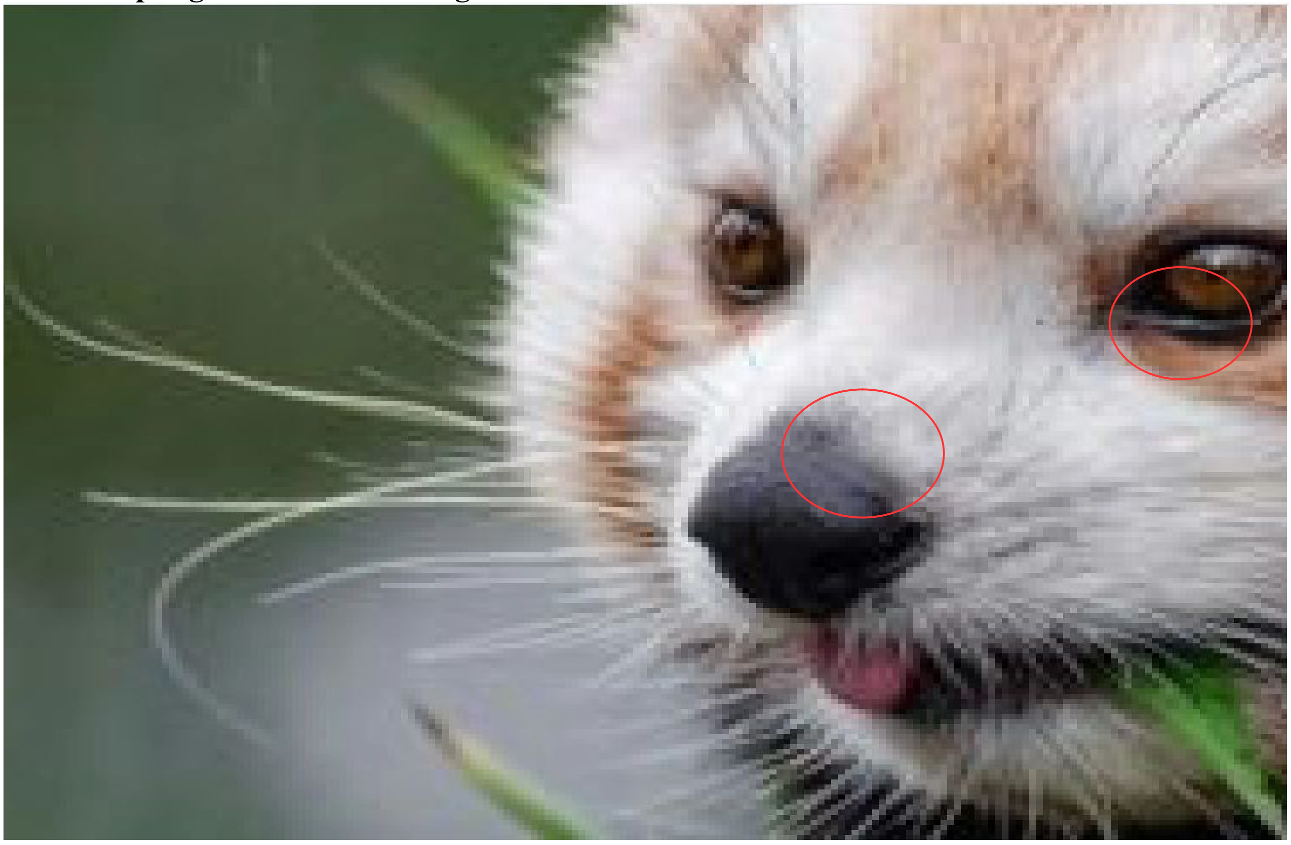
4:2:0 sampling "beach" images:



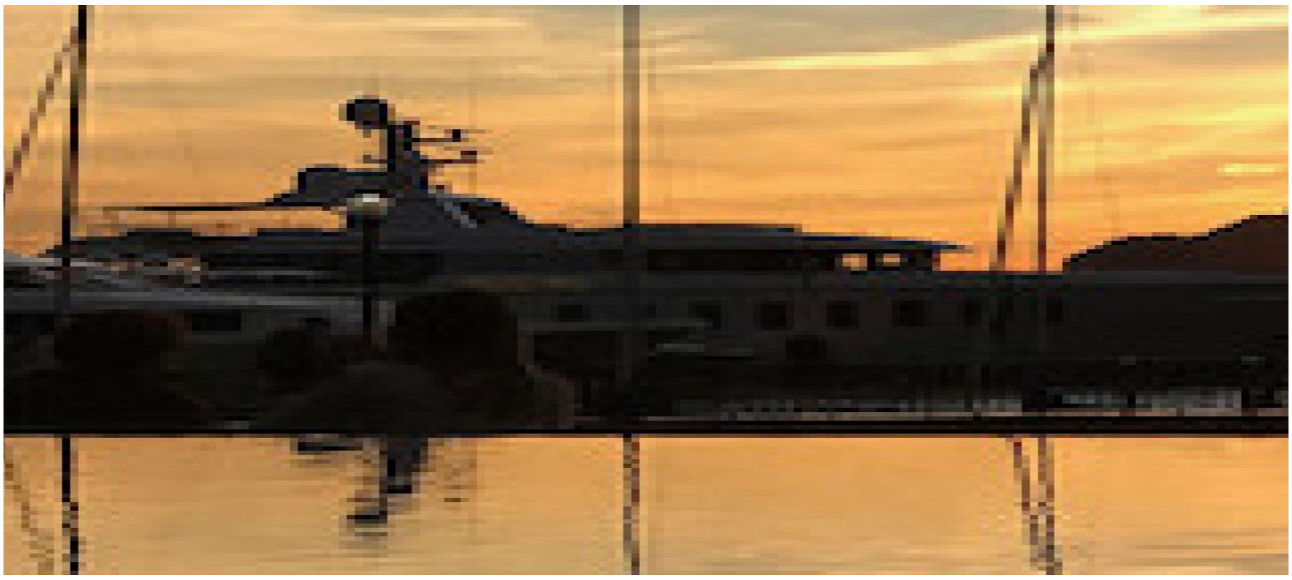
Original “Red Panda” images:



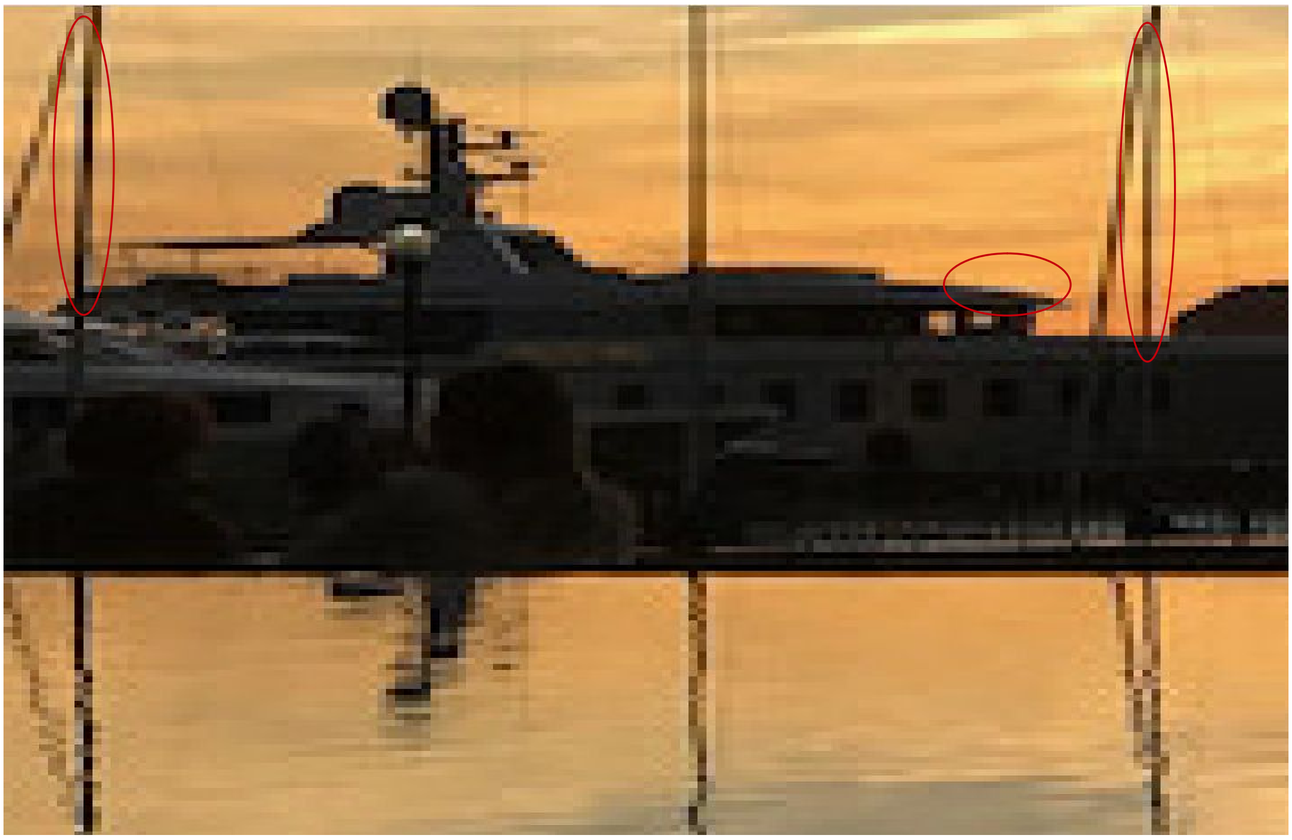
4:2:0 sampling “Red Panda” images:



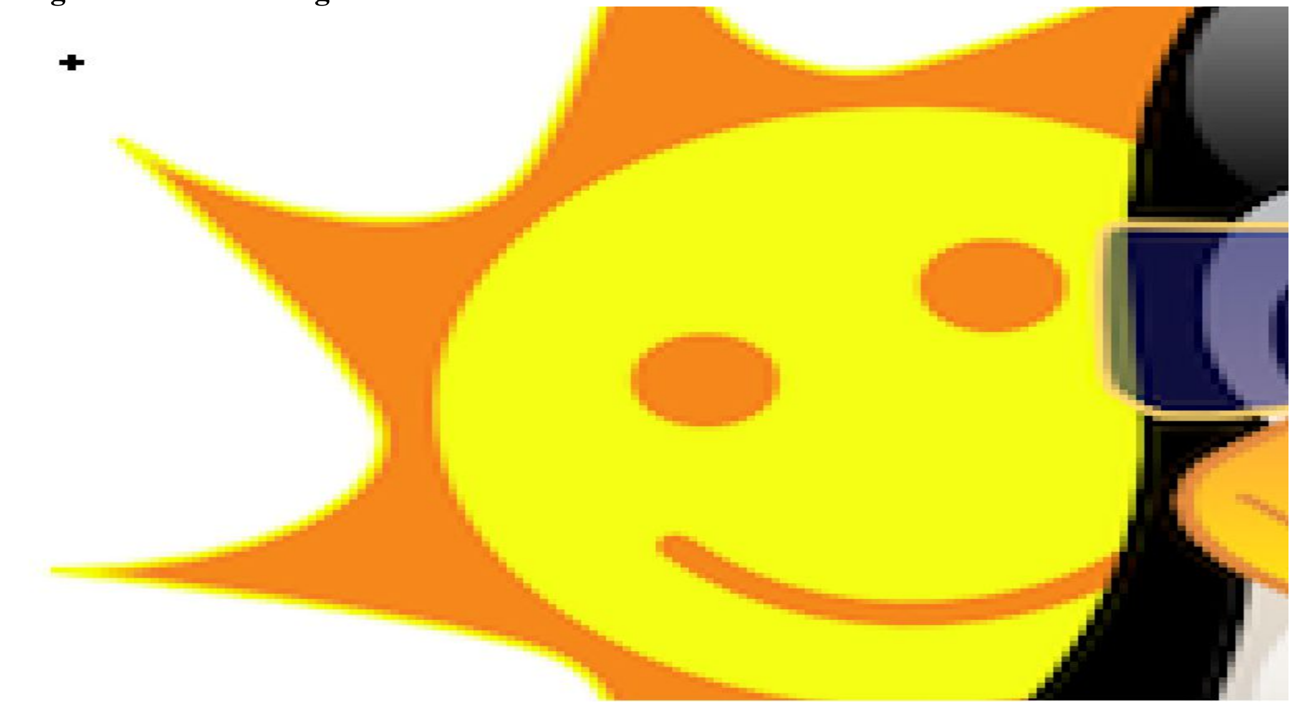
Original “sunset” images:



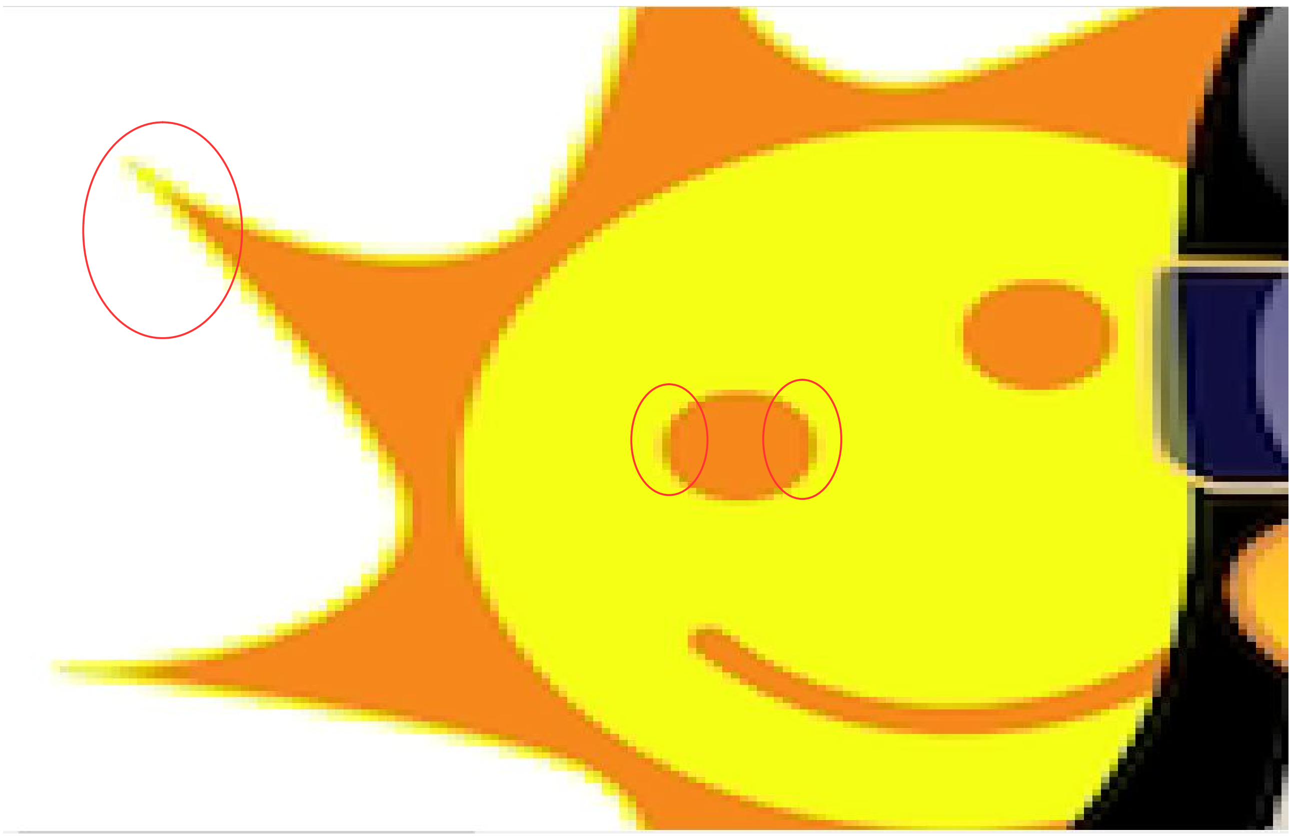
4:2:0 sampling “sunset” images:



Original "Tuxinu" images:

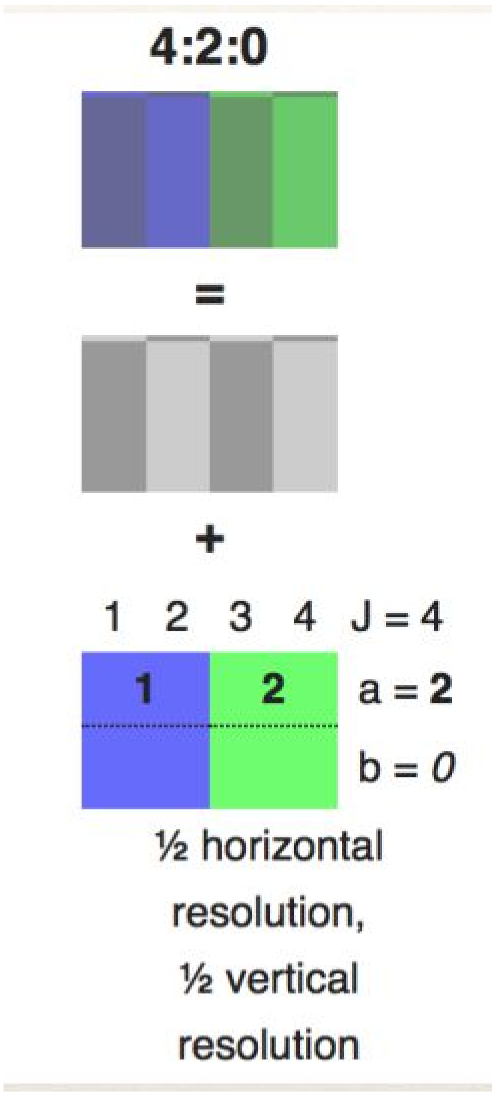


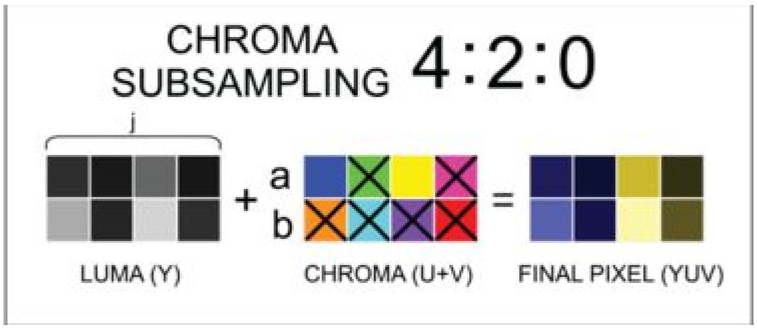
4:2:0 sampling "Tuxinu" images:



Explanation of the results:
Lower resolution for 4:2:0 quality
The 4:2:0 image uses less pixels for the chroma component U and V components. In a hypothetical 4x2 pixels image, 4:2:0 sampling takes two chroma samples from the top row of pixels and none from the bottom row. The distribution of the luma and chroma component would be as below. It can reduce the chroma information to 25% of the uncompressed chroma.

Impact of lower resolution used for the colour component
In general, the image quality would not reduce as much as human are more sensitive to luma component. In the image area where colour is the same or similar to the neighbouring pixel, sharing the same chroma component with neighbouring pixel would not adversely affect the image quality. However, there would be artifact at fine edges where there is sharp color transition. This is because sharing a neighbouring pixel's chroma component would just produce an approximate colour compared to the original pixel.





Source:

https://www.videomaker.com/article/f6/15788-the-anatomy-of-chroma-subsampling

http://softpixel.com/~cwright/programming/colorspace/yuv/