Group 7:

Albert Hernández

Anthony Vega

José Pablo Vernavá

Natalia Rodríguez

Documentation: rgb2yuv using NEON™ Intrinsics

Note: we used the Warrior release of Yocto.

RGB Format:

RGB is one of the most well-known color systems in the world and is extensively used in modern technology since it combines red, Green and blue light to create the colors that can be seen in TV screens, computer monitors and smartphones. RGB has three color channels that can be represented from 0 to 255 when the files are encoded in 24 bits (8 bits per channel). RGB files are typically encoded in 8, 12, 16 or 24 bits per pixel and for this project the 24 bits per pixel format is used and is written as RGB888 (8 bits per color channel).

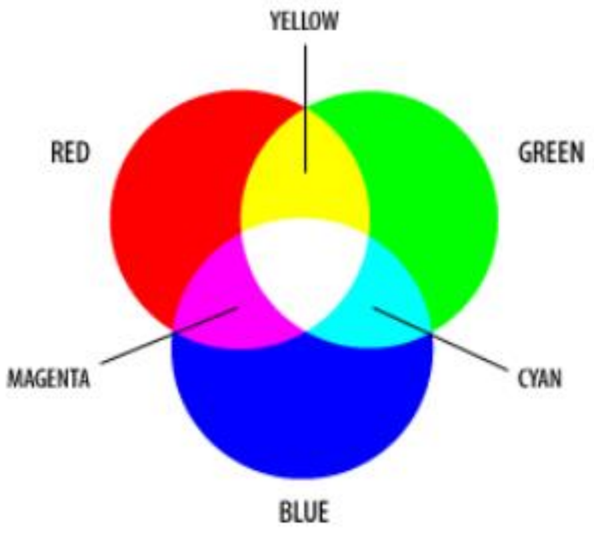

Fig. 1. RGB color space.

YUV Format

YUV is a color encoding system that encodes a color image or video taking human perception into account. The term was historically used for a specific analog encoding of color information in television systems. Now, it is commonly used in the computer industry to describe file-formats that are encoded using YCbCr, which is a family of color spaces.

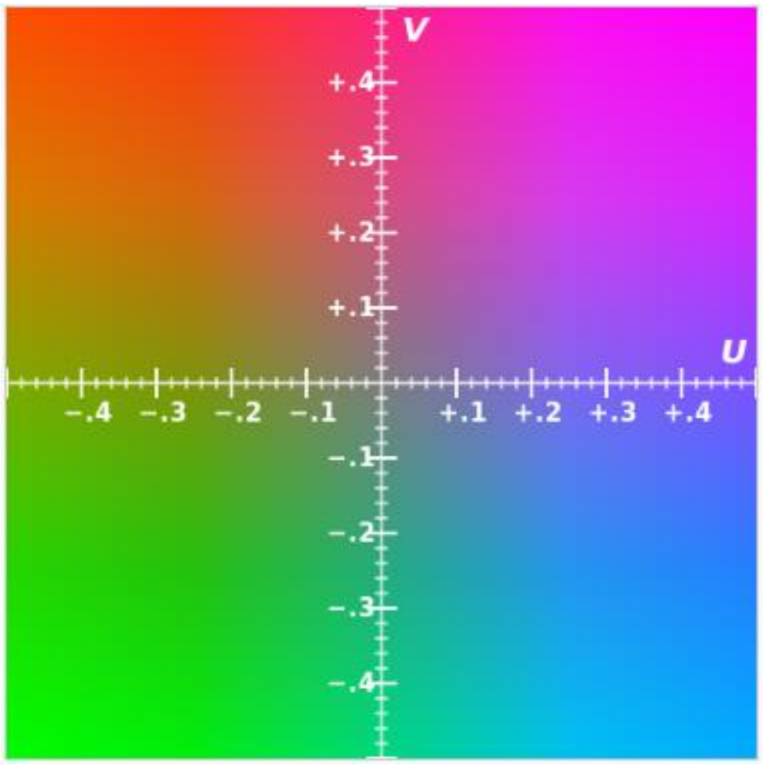

Fig. 2. Example of UV color plane, Y value = 0.5.

The "rgb2yuv(char *input_image, char *output_image)" conversion function using Intrinsics we implemented does the following:

1. Opens the input image to a file handle.
2. Starts reading the file one character at a time using the file handle until the end of file is reached. A counter is incremented for each character read.
3. Creates an "unsigned char" array based on the character counter value. This array will store the RGB values of the input image file.
4. Returns the file handle to the first character of the input image file.
5. Reads one character at a time again until the end of file is reached, but this time, the characters read are stored to the array.
6. At this point the array contains all RGB values on the file.
7. Loops through the RGB array, reading 8 pixels (24 bytes) at a time, loading them into 3 registers (uint8x8x3_t), so one register contains the Red components, another one the Green components and the last one the Blue components.

8. The Y,U, and V components are then calculated separately using Intrinsics instructions to mimic the following equations:

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

   a. For Y component: since it has only positive coefficients, we used “vmull_u8” (multiplication), “vmlal_u8” (multiplication and add), “vshrn_n_u16” (shift right) and “vadd_u8” (add).

   b. For U component: since it has positive and negative coefficients, we used “vmull_u8” (multiplication), “vmlsq_u16” (multiplication and subtract), “vshrn_n_u16” (shift right) and “vadd_u8” (add).

   c. For V component: since it has positive and negative coefficients, we used “vmull_u8” (multiplication), “vmlsq_u16” (multiplication and subtract), “vshrn_n_u16” (shift right) and “vadd_u8” (add).

   Note: according to the instruction used, the proper casts were made.

9. Then, the 3 components (Y, U, V) are combined into a single array as follows:

Y[0]	**Y[1]**	**…**	**Y[n]**	**U[0]**	**U[1]**	**…**	**U[n]**	**V[0]**	**V[1]**	**…**	**V[n]**

10. Finally, the output array is written to the output file.

We used the RGB8 format for the RBG image and YUV444p format for the converted YUV image.

In order to see the images in rawpixels.net, the following format values need to be used:

	RGB888	**YUV444p**
Predefined format	RGB8	YUV444p
Pixel format	RGBA	YUV
Ignore Alpha	Unchecked	
Alpha First		
Little Endian		
bpp1,bpp2,bpp3	8	
bpp4	0	
Pixel plane	Packed	Planar
Alignment	1	
Subsampling H		
Subsampling V		

The command line command to generate the sample image is as follows:

```
# raspiyuv -w <WIDTH> -h <HEIGHT> -rgb -o <FILENAME>
```

For example, to capture a 640 x 480 pixels image named “image.rgb”, you need to execute the following command:

```
# raspiyuv -w 640 -h 480 -rgb -o image.rgb
```

The following are the sample input RGB image and the YUV image obtained:

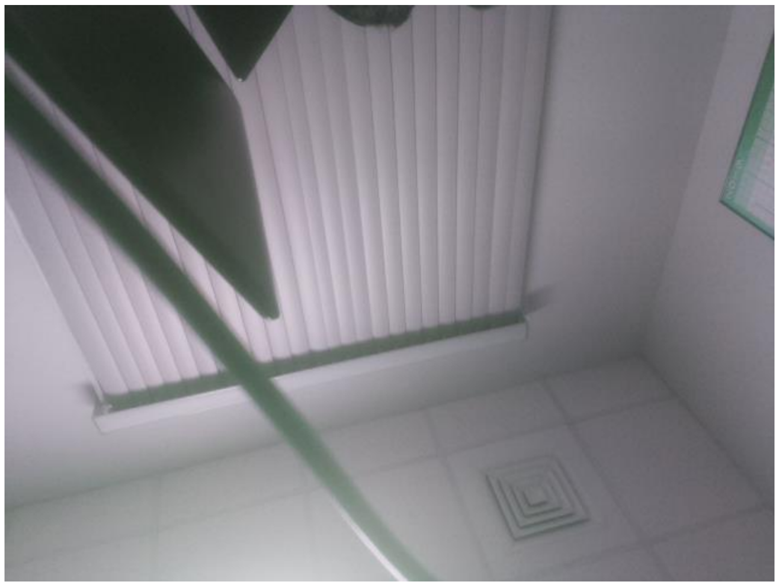

Fig. 3. Sample input RGB image to be converted.

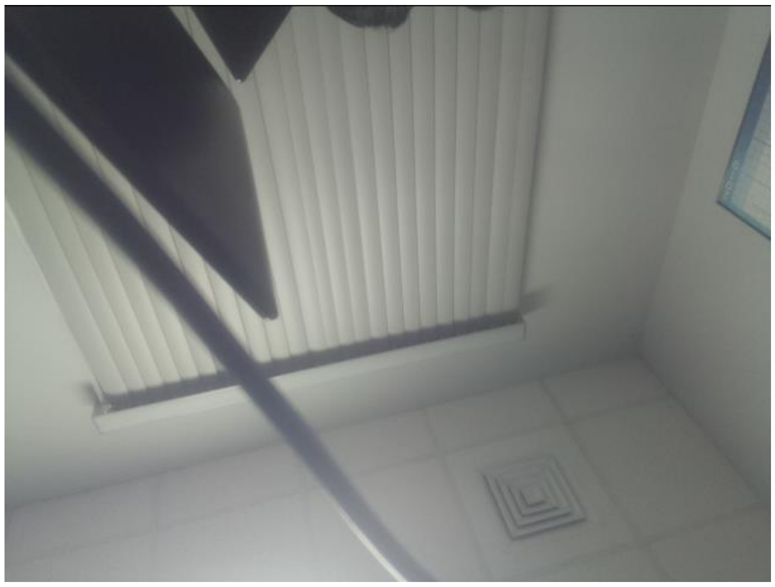

Fig. 4. YUV image obtained after the conversion.

A timer was started right before the call to “rgb2yuv” function and stopped right after it. The average processing time of the “rgb2yuv” function that uses Intrinsics was **205 ms**.