## Input Format: RGB24

- Each pixel is represented by 24 bits (1 byte per color component)

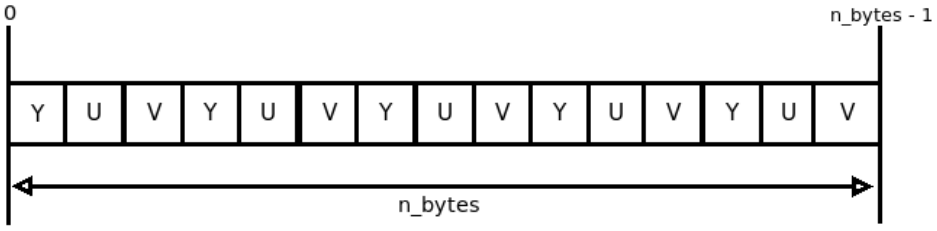
| Read Byte | | | | | | | | Green Byte | | | | | | | | Blue Byte | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

- For a 640x480 image:
  - Total pixels: n_pixels = 640 * 480 = 307,2 kB
  - Total bytes: n_bytes = 640 * 480 * 3 = 921,6 kB
- Source image obtained using the following bash command:
  - *raspiyuv -w 640 -h 480 -bgr -o rgb_image.bgr*
- Pixel Format: BGR

| B | G | R | B | G | R | B | G | R | … | B | G | R | B | G | R | B | G | R |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

## Output Format: YUV

- YUV is a color encoding system based on a color space composed on three components.
  - Y: “Brightness” component, known as “luma”.
  - U: Chrominance (color) component.
  - V: Chrominance (color) component.
- YUV 4:4:4 Packed
  - Y, U and V components for each pixel




## Neon:

- Neon is an advanced Single Instruction Multiple Data (SIMD) architecture provided by ARM processors. Is highly used in video and audio encoding/decoding and graphics.
- Neon registers are considered vectors. Instruction set is intended to perform operations in all lanes of the registers (vectors).
- Neon can be used by enable compiler’s auto-vectorization, C intrinsics or Neon Assembly code.

## Conversion Algorithm

- **RGB to YUV conversion is done through the following equation:**

Y = ( ( 66 * R + 129 * G + 25 * B + 128) >> 8) + 16
U = ( ( -38 * R - 74 * G + 112 * B + 128) >> 8) + 128 (Eq.1)
V = ( ( 112 * R - 94 * G - 18 * B + 128) >> 8) + 128

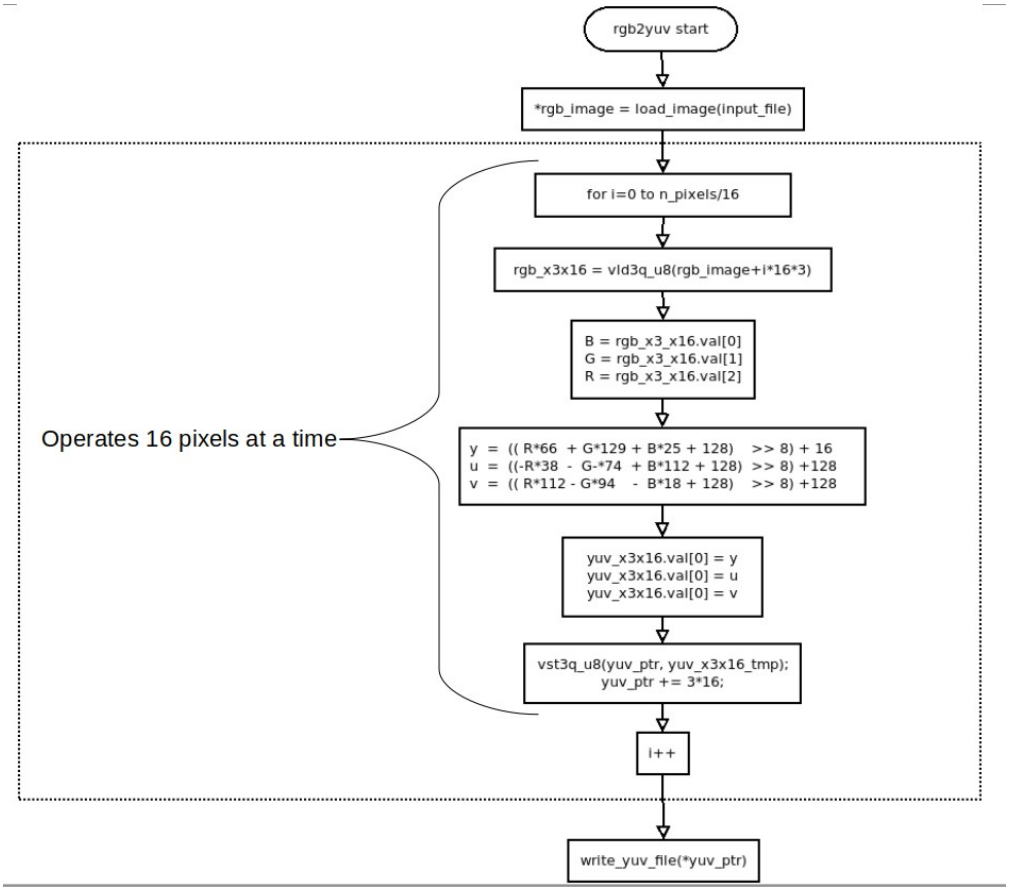
Equation 1 (same used on the plain C implementation) is used to compute the YUV components for each pixel of the output image.

The main difference with algorithm implemented using NEON and the one implemented using “plain C” is that the conversion is applied to 16 pixels per loop cycle instead of performing it one pixel at a time.

The other major difference is the “Pixel Plane” format, the algorithm that uses NEON implements a “Packed Pixel Plane” while the algorithm implemented using “plain C” uses a “Planar Pixel Plane” format. The reason why it was decided to use “Packed” over “Planar” is to exploit the “vst3q_u8” function which stores multiple 3-element structures (Y, U and V) with interleaving. Conversion multiplications and adds are performed through the “vmlaq_s16” function.

Is also important to mention that 16bits data structures (unsigned for Y and signed for U and V) were used in order to handle the multiplications (coefficient*color_component) involved in the conversion process.

The following flow chart illustrates the conversion process.



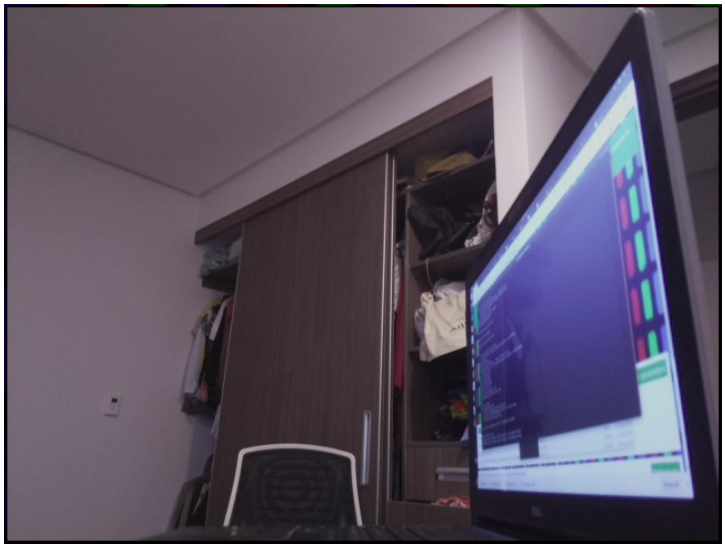
## Results

- Sample Input Image (image.bgr)

- Image parameter (used to visualize image in http://rawpixels.net/)

| | |
|---|---|
| **Raw Data** | image.rgb |
| **Width** | 640 |
| **Height** | 480 |
| **Predefined Format** | RGB24 |
| **Pixel Format** | BGRA |
| **Ignore Alpha** | No |
| **Alpha First** | No |
| **Pixel Plane** | Packed |

- Sample Output Image (output.yuv)



- Image parameter (used to visualize image in http://rawpixels.net/)

| | |
|---|---|
| **Raw Data** | output.yuv |
| **Width** | 640 |
| **Height** | 480 |
| **Predefined Format** | YUV444p |
| **Pixel Format** | YUV |
| **Ignore Alpha** | No |
| **Alpha First** | No |
| **Pixel Plane** | Packed |

- Execution time:
  - Following table shows the average time spent in five iterations executing the conversion of image.bgr into output.yuv.

| Iteration | Time (s) | Average |
|---|---|---|
| 0 | 0.101039 | 0.1015354 |
| 1 | 0.104798 | |
| 2 | 0.100448 | |
| 3 | 0.101011 | |
| 4 | 0.100381 | |

  - rgb2yuv_c vs. rgb2yuv_Neon:

| Process | Average time (s) | Difference |
|---|---|---|
| rgb2yuv_c | 0.092435 | ~30% |
| rgb2yuv_Neon | 0.060323 | |

## References


[1] "Converting Between YUV and RGB | Computer Graphics | Imaging", *Scribd*, 2019. [Online]. Available: https://www.scribd.com/document/117222158/Converting-Between-YUV-and-RGB. [Accessed: 27- Jun- 2019].

[2] A. Ltd., "SIMD ISAs | Optimizing C Code with Neon Intrinsics – Arm Developer", *ARM Developer*, 2019. [Online]. Available: https://developer.arm.com/architectures/instruction-sets/simd-isas/neon/neon-programmers-guide-for-armv8-a/optimizing-c-code-with-neon-intrinsics/rgb-deinterleaving. [Accessed: 26- Jun- 2019].

[3] "yszheda/rgb2yuv-neon",*GitHub*, 2019. [Online]. Available: https://github.com/yszheda/rgb2yuv-neon/blob/master/yuv444.cpp. [Accessed: 26- Jun- 2019].