Coding exercise for C++ developer

In order for us to have a more grounded understanding of not only your current skills but also your comfort in learning the technology we work with in 8i on a day to day basis, we would like you to engage on the following scenario:

A lot of our work involves the processing of images from computer vision cameras. As an example we have provided you with some still frames and would like you to write a small application to process them a bit. This application should:

- Have a graphical user interface written using Qt 5
  - We base our software on *Qt 5.12*. **DO SPECIFY** if you use any features that require a version newer than that or we might not be able to build it ourselves
  - You can use either *QWidgets* or *QML* for the UI, whichever you're more comfortable
  - As a build system you're free to use either *CMake* or *qmake*. While it's part of Qt, we wouldn't recommend you use *qbs* as none of us have used it and aren't familiar with it, so if your build turns out to need some massaging for us to double check, we might not be able to verify
  - Other than Qt we ask that you do not use any external libraries. This is mostly to ensure we're able to build your code without problems. If you DO want to use something like *CUDA, OpenMP, ISPC or TBB* for increased performance, make sure we can build it on an *OpenSUSE Leap 15.2*. None of this is required or expected. Qt has facilities for all the image-related functionality and even if you WANT to fiddle with parallelisation, it's own threading API and/or regular *C++ STL* should have you quite comfortably covered
- Your application must be able to load the provided PNG images
- The images we've provided are straight from a computer vision cameras and need to be "debayered" (see appendix)
- One of the images is a "clean plate", a picture of the green screen without anything in it. Using that, we want your application to be able to do a crude foreground extraction
  - This can be done by comparing if a pixel on the "clean plate" is equal (same colour) to the same pixel on the "shot image". It only makes sense to do this after the images have been debayered
  - Between shadows, potential green screen movement, natural sensor and lighting variance, etc... this is not an exact comparison. As such, we'd like your application to have some sort of configuration, such as a slider, for the threshold of what is considered to be foreground and background.
- Your application needs to be able to save the debayered images and the "extracted foreground" as PNG files. How the discarded background is treated (matted into grey, replaced with specific colour/transparency, etc…) is up to you.

- Your application needs to be able to display the debayered image and the “extracted foreground” before the user decides to save it or not
    - Not required but desirable bonuses would be responsiveness on the update of the preview as well as the ability to zoom, move and rotate the image (as well as saving them in the correct orientation)

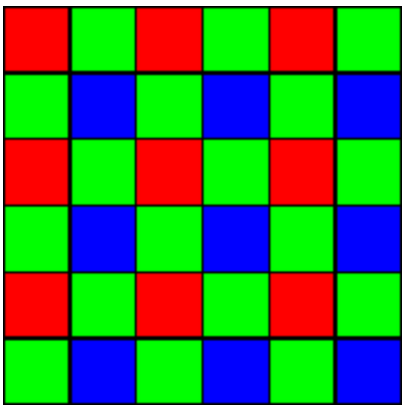
The time constraint on this is **three days**. Please provide us with the source code for your work and an example output (no need for binaries, we’ll compile it on our end).

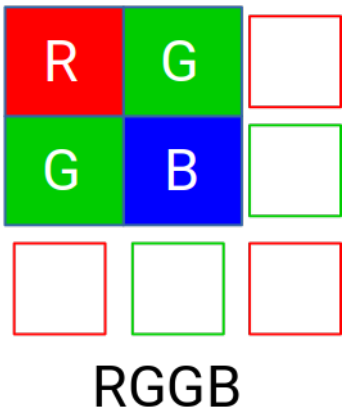
Feel free to contact us if any of the guidelines here are unclear. Good luck and happy coding

Appendix – The Bayer Filter

Coloured photos from digital cameras really are more of a trick than anything else. For the vast majority of cameras, the sensors are only able to detect general light levels. In order to produce coloured images from them, a special filter is placed over the sensor which filters out all but one "rgb component" of light. These are arranged as a grid over the sensor so that each pixel on the output represents only one of Red, Green or Blue colour.

Different camera/sensor manufacturers will choose different filter patterns according to their needs, research in technology, etc… The cameras we captured the provided images with use what is known as a "RGGB" pattern:





This means that the very first pixel on the image's top left corner has a red colour value, the following pixel on that row has a green value and so and so on… Pixels are grouped on 2x2 groups of four, consisting of two "rows" and two "columns"

There are various different strategies in how to "translate" that information so that the image can be "demosaiced" or "debayered" into an actual colour image. The simplest one is to treat each of those 4 pixel groups as a single pixel in the output image whose values for Red, Blue and Green are, respectively, the value of the colour in pixel "0,0", the value for pixel "1,1" and the average of pixels "1,0" and "0,1".

The most obvious disadvantage of this very simple strategy is that you end up reducing your resolution by half in both dimensions. It can also be tricky around the edges if your image has any dimensions which are even, which is not the case for the provided images which are 4112 wide and 3008 tall. **For the purposes of this exercise, this strategy is good enough and this drawback is not an issue**.

As you work, in order to check whether the colours you're getting out are expected here is a reference, keep in mind that as this was taken with a different camera under different conditions, the exact tone is likely to be very different, but as long as your green screen looks green and Barbie's dress looks pink, etc… you're on the right track

If you want to read a bit more on this, the Wikipedia page on the subject is a good enough place to start with some history and examples of alternative filters. The page on Demosaicing has some visual examples of the process and ideas but we believe the information provided here to be enough