DSP2 SS2020 – Exercise 1: CMake, OpenCV and Grayscale

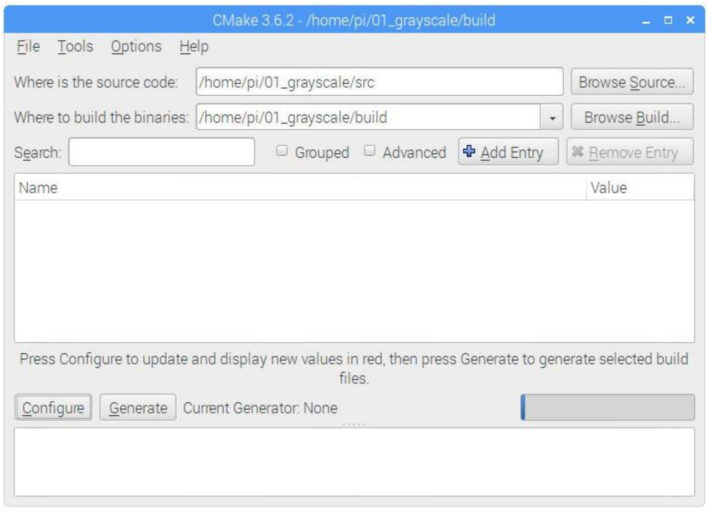
Part 1: Get the Files

1. Download the exercise (`01_grayscale.zip`) from OPAL[1]
2. Extract the archive file (with file explorer or in terminal with command `unzip 01_grayscale.zip`)
3. Afterwards you will find a new directory `01_grayscale` with the following content:
   a. build: an empty directory for the program you are going to compile
   b. data: a folder with the image (lena.tiff) which we are going to use as input
   c. src: this directory contains the source files

Part 2: Create the project with CMake

1. Start CMake (e.g. in terminal with command `cmake-gui`)
2. Set the path to source and build

   Note: In the pictures the path "/home/pi" us used. Replace with the correct path on your computer, e.g. "/home/dsp2/Downloads" for the default download folder of the given Ubuntu image.



CMake 3.6.2 - /home/pi/01_grayscale/build

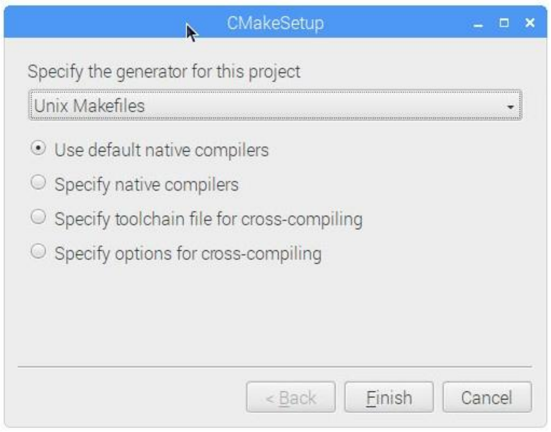
Where is the source code: /home/pi/01_grayscale/src

Where to build the binaries: /home/pi/01_grayscale/build

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: None

3. Configure (use default settings in the following window)



CMakeSetup

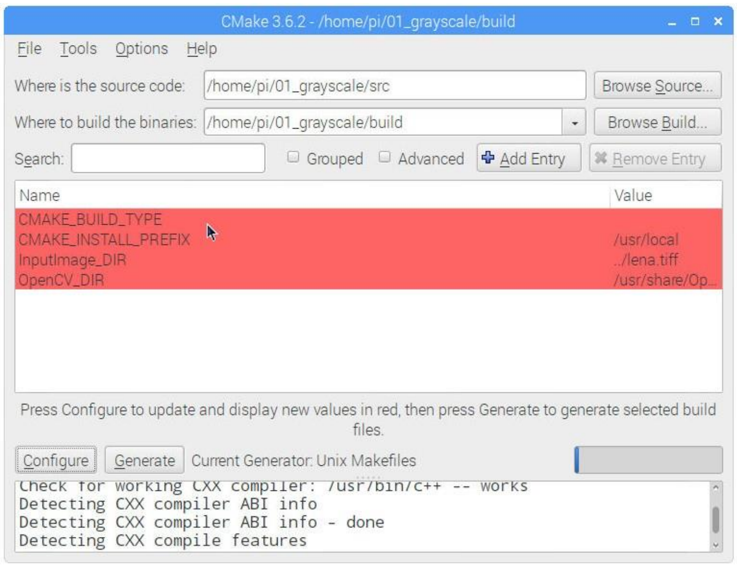
Specify the generator for this project: Unix Makefiles

(●) Use default native compilers
( ) Specify native compilers
( ) Specify toolchain file for cross-compiling
( ) Specify options for cross-compiling

[1] OPAL → Technische Universität → Chemnitz Fakultät für Elektrotechnik und Informationstechnik → Professur Digital- und Schaltungstechnik | Chair of Digital Signal Processing and Circuit Technology → DST_DSP_2

4. Configure again (the red markings will disappear)

   Note: Newly found settings are marked red. In This case red color does not mean there was an error.



Where is the source code: /home/pi/01_grayscale/src

Where to build the binaries: /home/pi/01_grayscale/build

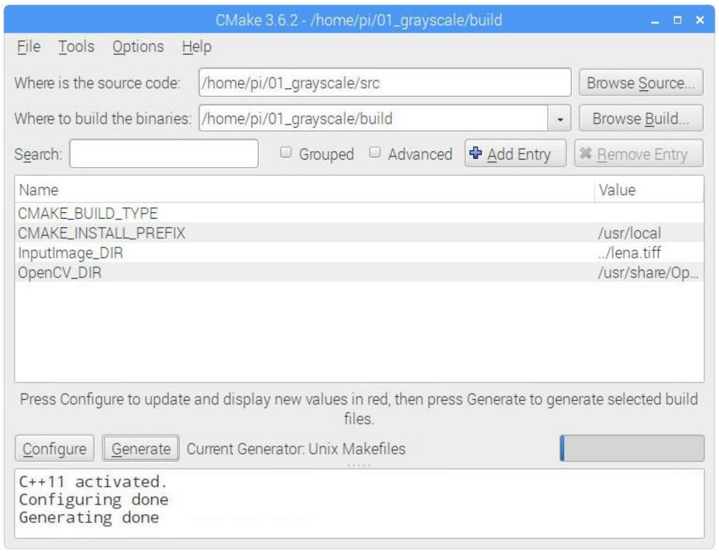
Name	Value
CMAKE_BUILD_TYPE	
CMAKE_INSTALL_PREFIX	/usr/local
InputImage_DIR	../lena.tiff
OpenCV_DIR	/usr/share/Op...

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Unix Makefiles

```
Check for working CXX compiler: /usr/bin/c++ -- works
Detecting CXX compiler ABI info
Detecting CXX compiler ABI info - done
Detecting CXX compile features
```

5. Generate



Where is the source code: /home/pi/01_grayscale/src

Where to build the binaries: /home/pi/01_grayscale/build

Name	Value
CMAKE_BUILD_TYPE	
CMAKE_INSTALL_PREFIX	/usr/local
InputImage_DIR	../lena.tiff
OpenCV_DIR	/usr/share/Op...

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Unix Makefiles

```
C++11 activated.
Configuring done
Generating done
```

Note: Once the project is created, you will not have to use CMake (and these 5 steps) for this project again, even when you change the source code.

Part 3. Compile and start the program

1. Use the terminal and go to the build directory
   (e.g. `cd ~/Downloads/01_grayscale/build`).
2. Compile the program using the command `make`
3. Start the program (in the build directory) with `./grayscale`

Note: After changing the source code you have to use `make` again.

Part 4: The source code (folder src)

Files:

1. CMakeLists.txt
   - the project description file
2. Timer.h
   - Provides an easy way to measure the time (you do not need to modify this file at all).
   - `INIT_TIMER` (without ";") initializes the timer
   - `START_TIMER` starts the time counting
   - `STOP_TIMER(<text>)` displays the time elapsed between START_TIMER and STOP_TIMER with the additional text <text>. Use <text> for a short but meaningful description.
3. main.cpp
   - Function `main(…)`: You do not have to change anything.
   - Function `grayscale cpp(…)`
     This is the place where you have to insert your code.

Description of main.cpp

- `cv::Mat img = cv::imread("../data/lena.tiff");`
  creates the matrix img and imports the image (24-bit BGR color) from file
- `cv::cvtColor(img, imgGray, CV_BGR2GRAY);`
  converts the BGR color image (img) to a grayscale image (imgGray) with OpenCV
- `grayscale_cpp(…)`
  This function should convert the BGR color image to a grayscale image with standard C++ code. It is your task to write this code.
- `cv::imshow("description", img);` : displays the image (matrix)
- `cv::waitKey();`
  without it, the program (and the displayed images) will be terminated immediately

Part 5: Complete the function grayscale_cpp(…)

The function already

- create the output image,
- has two loops to access each pixel of the image,
- read the 3 values (0, …, 255) for the BGR color of each pixel and
- write an integer value ("gray") to the output.

For now, the output image is always black because the variable "gray" is always zero. Your task is to apply grayscaling by calculating the correct "gray" value based on the pixel color (values b, g and r).

Good luck and feel free to play with the code.

Note:

In OpenCV, the channel weights for RGB-to-grayscale conversion are: R*0.299, G*0.587 and B*0.114 (77/256*R, 150/256*G and 29/256*B).

Further information:

OpenCV API Reference: http://docs.opencv.org/2.4/modules/refman.html