IMGS cmake Build Environment

Carl Salvaggio, Ph.D.

Rochester Institute of Technology
College of Science
Chester F. Carlson Center for Imaging Science
54 Lomb Memorial Drive
Rochester, NY 14623-5604, USA

A general-purpose, skeletonized build environment for developing `C++` functions and applications for imaging-related purposes can be found at

`https://github.com/csalvaggio/rit`

This repository is specifically seeded to be used for the classes *IMGS.180/Introduction to Computing and Control*, *IMGS.361/Image Processing and Computer Vision I*, and *IMGS.362/Image Processing and Computer Vision II*, however, this environment can be used and expanded for any imaging-related `C++` code developed.

This build environment takes advantage of several `cmake` functions, authored by Dr. Philip Salvaggio, that were developed to provide convenient shorthand notations for `cmake` management. These `cmake` functions are located at

```
rit
└── cmake
    ├── CMakeUtils.cmake
    ├── CompilerFlags.cmake
    ├── Executables.cmake
    ├── Libraries.cmake
    └── Options.cmake
```

These `cmake` functions allow for the easy insertion of additional `C++` functions throughout the directory structure developed for the *Image Processing and Computer Vision* sequence of classes.

An exemplar of the general directory structure is depicted below

```
rit
├── build
├── cmake
├── CMakeLists.txt
├── imgs
│   ├── apps
│   │   ├── bilateral_filter
│   │   └── CMakeLists.txt
│   │       ├── bilateral_filter.cpp
│   │       └── CMakeLists.txt
│   ├── CMakeLists.txt
│   ├── ipcv
│   │   ├── bilateral_filtering
│   │   │   ├── BilateralFilter.cpp
│   │   │   ├── BilateralFilter.h
│   │   │   └── CMakeLists.txt
│   │   └── CMakeLists.txt
│   ├── numerical
│   ├── physics
│   ├── plot
│   ├── radiometry
│   └── third_party
└── README.md
```

You will notice that there are many `CMakeLists.txt` files throughout this directory structure. Let's take a look at each of them, and their including subdirectory, and discuss their purpose.

Let's begin with

```
rit
├── CMakeLists.txt
└── imgs
```

This file should remain untouched for the most part. It's purpose is to incorporate the `cmake` functions referred to above, define the `rit` project, the required version of `cmake`, the default user-defined project library prefix, find the required packages, set the standards to be followed, and to add the `imgs` subdirectory, at this current level, to the project.

Listing 1: `imgs/CMakeLists.txt`

```
cmake_minimum_required(VERSION 3.1)

project(rit VERSION 0.0.0)
```

```
include(GNUInstallDirs)

include(cmake/CMakeUtils.cmake)

rit_set_library_prefix("librit_")

find_package(Boost REQUIRED
  COMPONENTS
    program_options
    iostreams
    filesystem
    system
)
find_package(Eigen3 REQUIRED NO_MODULE)
find_package(OpenCV REQUIRED)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

add_subdirectory(imgs)
```

The next `CMakeLists.txt` file,

```
rit
└── imgs
    ├── apps
    ├── CMakeLists.txt
    └── ipcv
```

adds the `apps` and `ipcv` subdirectories to the environment. Like the previous file, this file should not need to be modified.

Listing 2: `imgs/imgs/CMakeLists.txt`

```
add_subdirectory(apps)
add_subdirectory(ipcv)
add_subdirectory(numerical)
add_subdirectory(physics)
add_subdirectory(plot)
add_subdirectory(radiometry)
add_subdirectory(third_party)
```

The `ipcv` directory exists to contain `C++` functions that perform image processing and computer vision tasks. The other directories contain functions that perform tasks relevant to their titles. These functions, each housed in their own directory, typically represent

an action that produces a result. For example, the `BilateralFilter.cpp` code in the `bilateral_filtering` directory shown below takes an input image and produces a bilateral filtered version of that image in the destination parameter or the returned object.

```
rit
└── imgs
    └── ipcv
        ├── bilateral_filtering
        │   ├── BilateralFilter.cpp
        │   ├── BilateralFilter.h
        │   └── CMakeLists.txt
        ├── CMakeLists.txt
        └── ipcv.h
```

The `CMakeLists.txt` file at the `ipcv` directory level adds the `bilateral_filtering` directory to the environment

Listing 3: `imgs/imgs/ipcv/CMakeLists.txt`

```
add_subdirectory(bilateral_filtering)
```

Any additional functions that are to be included in the `ipcv` directory should have an entry just like the one shown above included in this `CMakeLists.txt` file. The same is true for functions added to the other topical area directories.

The `CMakeLists.txt` file that is located in the `bilateral_filtering` directory contains the following

Listing 4: `imgs/imgs/ipcv/bilateral_filtering/CMakeLists.txt`

```
rit_add_library(ipcv_bilateral_filtering
  SOURCES
    BilateralFilter.cpp
  HEADERS
    BilateralFilter.h
)

target_link_libraries(ipcv_bilateral_filtering
  PUBLIC
    opencv_core
)
```

This file is specific to the function being defined, and names the library that is created for the individual function. The defined `cmake` function `rit_add_library` defines the name of the library to be created for the function in this directory. The library name should consist of the prefix `ipcv_` followed by the name of the containing directory.

The SOURCES and HEADERS should define the names of the C++ and corresponding header files (named using camel-case format).

The defined cmake function target_link_libraries lists the current library being created along with any additional public libraries that are needed by the function being defined (in this case, the core OpenCV library).

The aggregate header file, ipcv.h, should be modified to include the header file defined here, namely

Listing 5: imgs/imgs/ipcv/ipcv.h

```
/** Aggregate interface file for IPCV library
 *
 *  \file ipcv/ipcv.h
 *  \author Carl Salvaggio, Ph.D. (salvaggio@cis.rit.edu)
 *  \date 17 Mar 2018
 */

#pragma once

#include "ipcv/bilateral_filtering/BilateralFilter.h"
```

This should be repeated for each function/header added to the ipcv directory.

One last set of CMakeLists.txt files to look at are those in the apps directory. The apps directory contains user-level applications to perform stand-alone tasks or tasks involving those functions defined in the ipcv directory.

For the following tree

```
rit
└── imgs
    └── apps
        ├── bilateral_filter
        │   ├── bilateral_filter.cpp
        │   └── CMakeLists.txt
        └── CMakeLists.txt
```

the CMakeLists.txt at the apps level contains an entry for each application being developed. For example,

Listing 6: imgs/imgs/apps/CMakeLists.txt

```
add_subdirectory(bilateral_filter)
```

Add an entry to this file each time you create a new application.

The CMakeLists.txt in the application directory

Listing 7: imgs/imgs/apps/bilateral_filter/CMakeLists.txt

```
rit_add_executable(bilateral_filter
  SOURCES
    bilateral_filter.cpp
)

target_link_libraries(bilateral_filter
  Boost::filesystem
  Boost::program_options
  rit::ipcv_bilateral_filtering
  opencv_core
  opencv_highgui
  opencv_imgcodecs
)
```

is similar to the one located in each of the `ipcv` directories, except that this file uses the defined `cmake` function `rit_add_executable` to define the user-level executable file that can be run.

Of particular note here, you must modify the name of the executable to be created (in this case `blilateral_filter`) as well as the source file name under the `SOURCES` designation. You must also include all libraries that are required for linking, including any libraries created in the `ipcv` directory (in this case `rit::ipcv_bilateral_filtering`).

Compilation of the project must occur in the `build` directory. If this directory does not exist, create it at this location

```
rit
└── build
```

In the `build` directory, type the command `cmake ..` the first time (and any time after which you may delete and recreate this directory). The following structure will be created

```
rit
└── build
    ├── bin
    ├── build.ninja
    ├── CMakeCache.txt
    ├── CMakeFiles
    ├── cmake_install.cmake
    ├── imgs
    ├── lib64
    └── rules.ninja
```

All function-specific libraries created will be located in the `lib64` directory. The user-level applications will be located in the `bin` directory. For example, for the application described above, the user may type `bin/bilateral_filter` to execute that application while located

in the `build` directory.