Introduction to CImg

March 20, 2009

1 C/C++ : Reminder

1.1 Reminder on syntax

When defining a function, the type of the returned object should be given (type_return). If the function does not return anything, the type void should be used. The parameters (param1, param2, ...) of one function should be given too, along with the type for each parameter (type_input1, type_input2, ...). As before there may be no parameters. Once the function prototype is given, the set of computations to do is given between braces ({}).

```
type_return foo(type_input1 param1, type_input2 param2)
{
    <code>// list of tasks to do
}
```

As a general rule, when a variable or an object is defined for the first time its type should be given. It should be done only once.

Loops are one of the most common operator that may be used. The type of the value used for the loop should be given. It will be defined only inside the loop.

```
for(int i=0 ; i<100 ; i++)
{
    <code>// list of tasks to do
}
```

An application is created by defining a single main method, which should be named main.

```
void main()
{
    <code>
}
```

1.2 Input/Output on command line

It is possible to write anything inside a command line window. User may enter some information too. This allows to avoid compiling a program each time you want to change some parameters. First, the file that contains these methods should be included.

```
#include <iostream>
```

Then writing on the command line window is done using the command *cout*.

```
std::cout << "Hello";   // write "hello" inside command line
int val = 5;
std::cout << val;       // write 5
std::cout << std::endl; // go to next line
```

Reading values is done by the command *cin*.

```
int val;
std::cin >> val;        // set val to parameter given by user
```

1.3 Class

The real definition of a class will not be given, only a simpler explanation will be given, which will be sufficient for this work. In our case we will see a class as a container, with some data known only inside the class, along with some methods allowing to work on data inside this class.

For example let us consider a class Matrix3x3, with a method rand() that allows to fill a matrix object with random values. First a matrix instance will be defined, and then the created object will be filled with random values.

```
Matrix3x3 M;     // creates an instance of class Matrix3x3
M.rand();        // fill M instance with random values
```

1.3.1 Templated Class

The class that we will use are templated class. Knowing what this means is not necessary for work. You may simply choose to use one templated class as a normal class.

Classes may be defined as templated. Let us imagine that our previous class Matrix3x3 works only with float values. If we want to use matrix with double values, we would have to create a new class Matrix3x3dbl, that will be mostly identical to our class Matrix3x3. Templated class permit to avoid that. It allows to specify the basis type that will managed inside a class as an additional parameter. In this case we will define a templated class Matrix3x3<T> only once, with T defining the type of parameter to consider. In our example one will only have to call Matrix3x3<float> and Matrix3x3<double> when creating instances of any of previous class types.

2 The CImg Library

The CImg library is an image library for c++ that includes lots of useful methods for image processing [5]. Other libraries exist, in particular one may cite opencv [4] and itk/vtk [3], which are often used for image processing. CImg was chosen because of its easy use.

Basic methods of the class CImg will be presented, for more details, please refer to official documentation [6].

2.1 Using CImg

First of all, CImg library should be downloaded and installed from the CImg website [5]. Then you should download and install ImageMagick software. This software is used by the CImg library for conversion of some usual image types [2].

We will now go through the library. First, your code should include one single file *CImg.h* and indicates that its namespace *cimg library* has to be used.

```
#include "CImg.h"
using namespace cimg_library;
```

This library contains 2 classes that will be useful for us:

- The class CImg<T>: this class represents an image (up to 4 dimensions wide), each pixel being of type T
- The class CImgDisplay: This class represents a window which can display CImg images and handles some events

2.2 Declaration of Images

The CImg class is a templated class CImg<T>, where T refers to the variable type used for the pixels. A standard type for gray level images is float, because it allows easier use of filtering, and more precision than integer types. In case of color images, the type T would be a container for RGB components, for example. There are several methods to create image object, the most useful are given next.

```
CImg<float> img;             // default declaration of image
CImg<float> img(300,200);    // 300x200 pixel image (random values)
CImg<float> img("img.png");  // directly loading a file
CImg<float> img2(img);       // copying a previous image
```

2.3 Input and Output of images

Loading and saving of images is very simple and a vast number of image formats are supported. Let assume an image is defined by any possible way. Then loading and saving is possible using loading and saving method.

```
img.load("img.png");  // loading from file
img.save ("img.png"); // saving to file
```

Please note that loading an image will erase any content that the variable img previously contained.

2.4 Processing at pixel level

The pixel value of any point inside a loaded image may be accessed by its coordinates.

```
img(x, y);
img(x, y, z);
```

The class contains a number of basis operations to modify pixel values. One may add a constant to each pixel of one image, or computes the sum of 2 images (the images should have same size). The basis operators are defined, addition, multiplication, division and subtraction.

```
CImg<float> img("img.png");
CImg<float> img2("img2.png");
img = img+5;      // add 5 to each pixel value inside img
img += 5;         // same thing, but inplace, without copy of the image
img += img2;      // sum of pairwise pixels
```

2.5 Other useful methods

In this part we will present other useful methods.

- Access to image dimensions

```
unsigned width  = img.dimx(); // access image width
unsigned height = img.dimy(); // access image height
unsigned height = img.dimz(); // access image depth
```

- Filling an image with a value.

```
img.fill(0.0f); // fill the image with 0
```

2.6 Displaying image

The CImg library contains a class CImgDisplay, that allows to display images along with some user interaction. This class contains a number of possibilities, but we will focus only on display for now.

A window with an image is created by defining a CImgDisplay object, with parameter defined by the image to display first, and the title of the window next. This last parameter is optional.

```
CImgDisplay main_disp(image, "title");
```

The interface between the window and the user is processed by events. If nothing is added the window will close as soon as the user interacts with the window. The next loop permits to avoid that. It allows to prevent window closing, for as long as it is not voluntarily closed. One may add additional commands each time the user interacts with the window.

```
while (!main_disp.is_closed)
{
     main_disp.wait();
     <command>
}
```

Once an image is displayed, it remains disconnected from the initial image that was used to create the display. In order to modify or refresh this image, one has to explicitly refresh the display.

```
main_disp << img;
```

3 Compiling a CImg program

Explanations for compilation with diverse compilers are given at the end of the page CImg Library Overview on the CImg website for compilation using command line [1].

3.1 Compiling with Code::Block

Compilation has been checked using *Code::Blocks 8.02*, with *mingw32-g++ v3.4.5* as compiler. A recent compiler is required for compilation. In particular, *v3.4.2* may not work.

An additional libray has to be linked for compilation. Open project options (right click on the project). Click on "Build option" inside the tab "Build targets" (figure 1). Then add a graphical library inside "Linker Settings" (gdi32 for Windows, X11 for Unix) (figure 2).

3.2 Compiling with Visual C++

The libraries required for CImg use are already included by default, thus there is nothing to do. However you should rather use empty console project, rather than one with precompiled headers. This choice is done inside project wizard.

3.3 Compiling with Dev-cpp

By default, the version of the compiler that is included in installer for Dev-cpp is not recent enough to handle CImg. It should work by installing a more recent compiler, for example *mingw32-g++ v3.4.5*. Then graphical libraries should be linked, as for compilation with Code::Block.

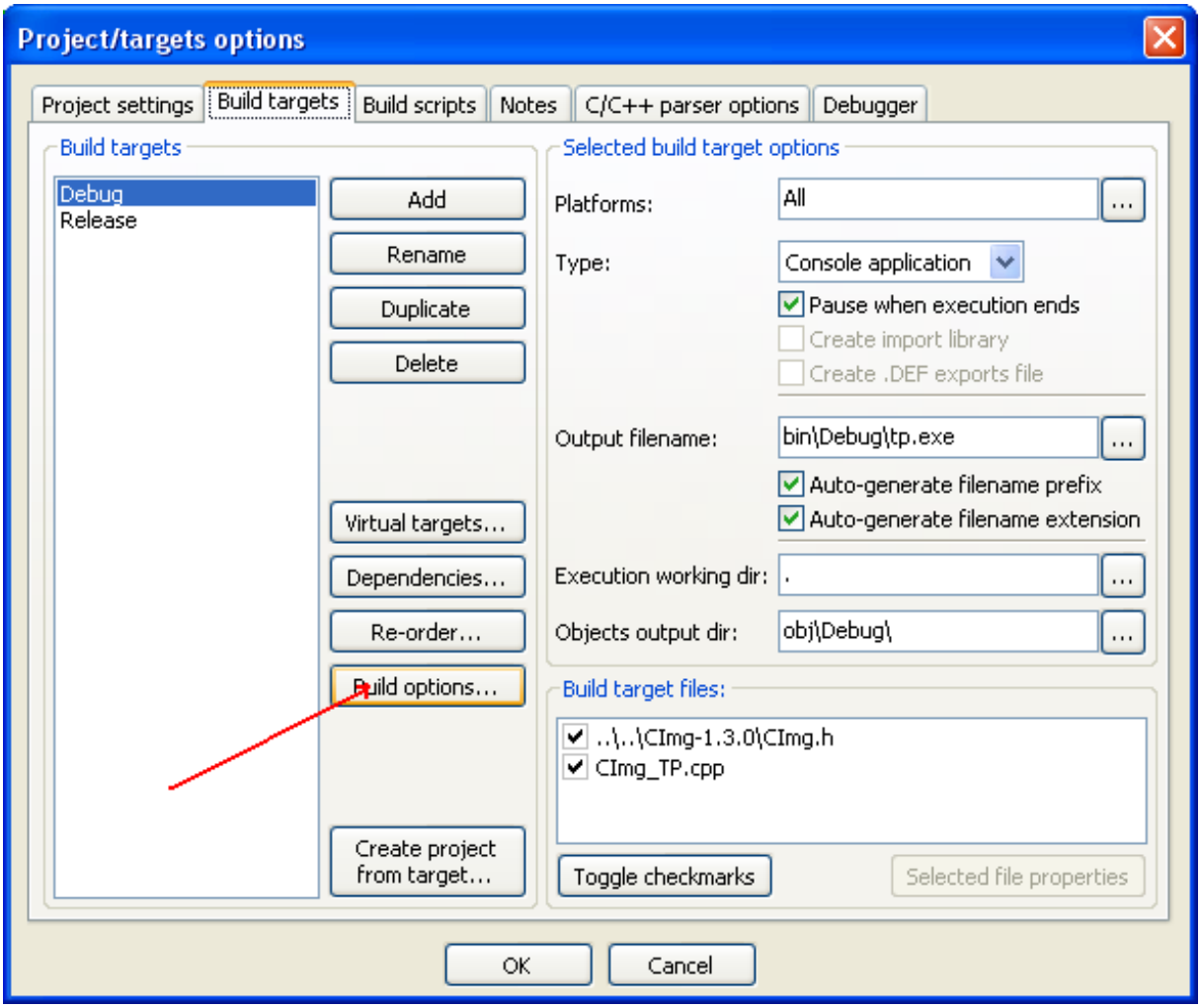

Figure 1: Code Block: Step 1.

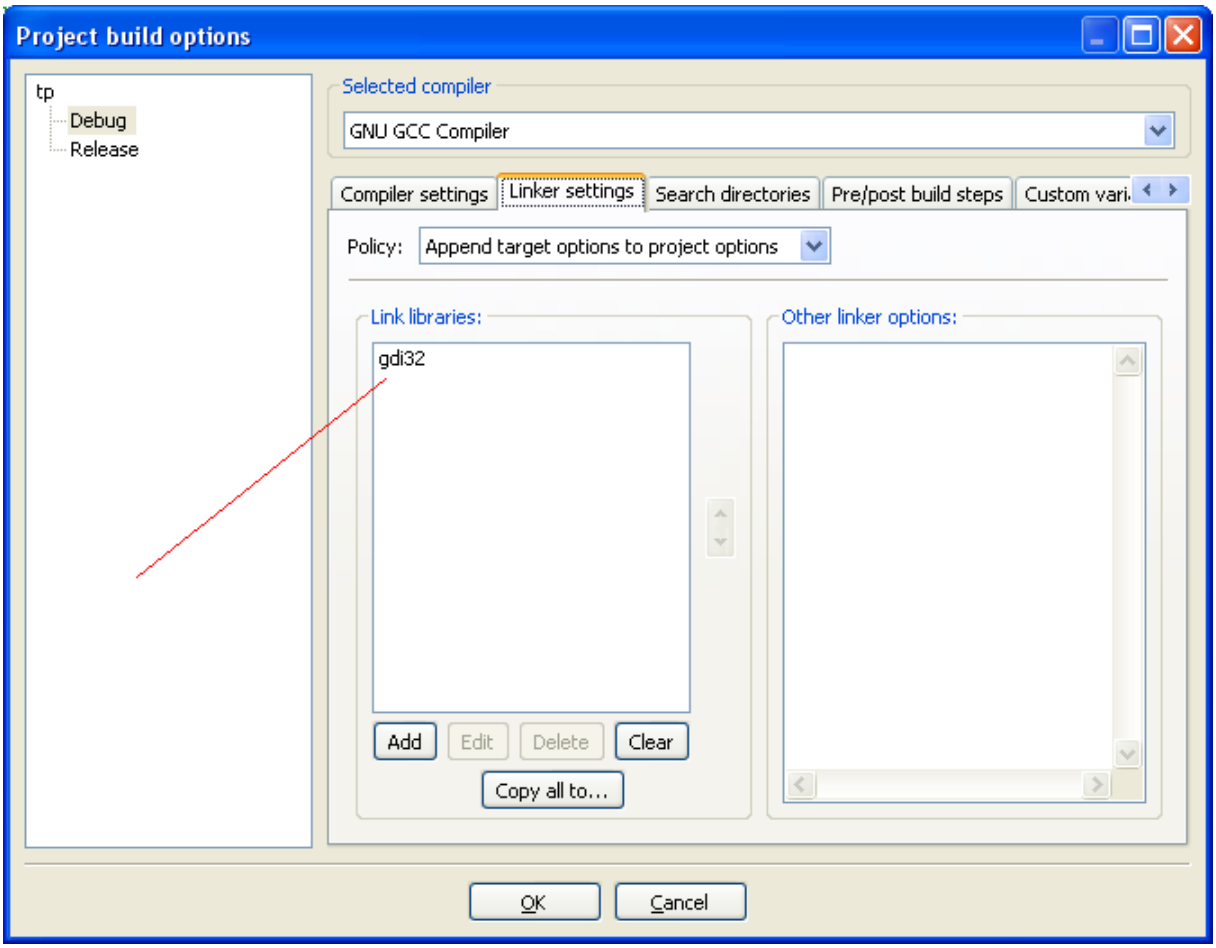

Figure 2: Code Block: Step 2.

3.4 Compiling on a Mac

Compilation on Mac may be done using *g++*, for recent versions. It will work with v3.4.5, but will not compile with v3.4.2.

```
g++ -o <appli_name> -O2 -I<include_path> <file_to_compile>
 -L/usr/X11R6/lib -lm -lpthread -lX11
```

4 Work

4.1 Work to do

You should send us back a report for the day after next lab session. A correction will be posted the morning after. The report should contain your experiments and the code you did for the last question (denoising using Gaussian filter).

4.2 Using the library

The goal of this part is only to get used to some basis methods. Try to open a file, to save it, to display it.

4.3 Gaussian Filtering

The class CImg already contains methods to apply Gaussian filters, but you should not use them. Gaussian filtering is the convolution of a Gaussian mask on an image. The formula for convolution is given as a reminder 1. For an image I, the convoluted image J by a kernel K of size $m \times n$ is defined as:

$$J(x, y) = \sum_{i=1}^{m} \sum_{j=1}^{n} I(x + i - 1, y + j - 1)K(i, j) \tag{1}$$

4.3.1 Defining the Gaussian Mask

Gaussian filters aim to smooth an image, by using the local values around one pixel. Thus Gaussian masks should have an odd size in order to give more weight on the central voxel. A Gaussian kernel may be isotropic or anisotropic, meaning that we may use either a same standard deviation on each axis, or diverse values. In this work we will use only isotropic Gaussian kernel.

Create a method that initializes an isotropic Gaussian kernel. This method should take a radius (to define the mask size) and standard deviation as parameters.

In order to compute exponential and to obtain the value for π, a file should be included. Do not forget mask normalization, otherwise all pixel values inside the image will be multiplied.

```
#define _USE_MATH_DEFINES // numerical definition for pi (M_PI)
#include "math.h"         // definition for exponential
```

4.3.2 Studying the effect of Gaussian filtering

For diverse Gaussian Kernel study how the filtered image is modified. You may use the convolution methods defined inside class CImg *convolve* or *get_convolve*.

4.3.3 Gaussian filtering for denoising

Gaussian filtering is a simple step that may be used to remove noise from images. Do as previously, but for a noisy image.

Noise may be added to an image using the method *noise*.

References

[1] Cimg: Compilation. `http://cimg.sourceforge.net/reference/group__cimg__overview.html`.

[2] ImageMagick Software. `http://www.imagemagick.org/script/index.php`.

[3] Insight Toolkit. `http://www.itk.org/`.

[4] Open Computer Vision Library. `http://sourceforge.net/projects/opencvlibrary/`.

[5] The Cimg Library. `http://cimg.sourceforge.net/index.shtml`.

[6] The Cimg Library Reference. `http://cimg.sourceforge.net/reference/index.html`.