

The Clmg Library

`_cimg_version`

Generated by Doxygen 1.8.11

Contents

Chapter 1

Main Page

This is the reference documentation of [the CImg Library](#), the C++ template image processing library. This documentation have been generated using the tool [doxygen](#). It contains a detailed description of all classes and functions of the CImg Library. If you have downloaded the CImg package, you actually have a local copy of these pages in the `CImg/html/reference/` directory.

Use the menu above to navigate through the documentation pages. As a first step, you may look at the list of [available modules](#).

You may be interested also in the [presentation slides](#) presenting an overview of the CImg Library capabilities.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

CIImg Library Overview	??
FAQ : Frequently Asked Questions.	??
Setting Environment Variables	??
How to use CIImg library with Visual C++ 2005 Express Edition ?.	??
Tutorial : Getting Started.	??
Using Drawing Functions.	??
Using Image Loops.	??
Using Display Windows.	??
How pixel data are stored with CIImg.	??
Files IO in CIImg.	??
Retrieving Command Line Arguments.	??

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

cimg_library	Contains <i>all classes and functions</i> of the CImg library	??
cimg_library::cimg	Contains <i>low-level</i> functions and variables of the CImg Library	??

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Clmg< T >	??
Clmg< charT >	??
Clmg< doubleT >	??
Clmg< floatT >	??
Clmg< intT >	??
Clmg< uintT >	??
Clmg< ulongT >	??
ClmgDisplay	??
ClmgException	??
ClmgList< T >	??
ClmgList< boolT >	??
ClmgList< charT >	??
ClmgList< doubleT >	??
ClmgList< ulongT >	??

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

CImg< T >	Class representing an image (up to 4 dimensions wide), each pixel being of type T	??
CImgDisplay	Allow the creation of windows, display images on them and manage user events (keyboard, mouse and windows events)	??
CImgException	Instances of CImgException are thrown when errors are encountered in a CImg function call	??
CImgList< T >	Represent a list of images CImg<T>	??

Chapter 6

Module Documentation

6.1 CImg Library Overview

The **CImg Library** is an image processing library, designed for C++ programmers. It provides useful classes and functions to load/save, display and process various types of images.

6.1.1 Library structure

The CImg Library consists in a single header file `CImg.h` providing a set of C++ template classes that can be used in your own sources, to load/save, process and display images or list of images. Very portable (Unix/X11, Windows, MacOS X, FreeBSD,...), efficient, simple to use, it's a pleasant toolkit for coding image processing stuff in C++.

The header file `CImg.h` contains all the classes and functions that compose the library itself. This is one originality of the CImg Library. This particularly means that :

- No pre-compilation of the library is needed, since the compilation of the CImg functions is done at the same time as the compilation of your own C++ code.
- No complex dependencies have to be handled : Just include the `CImg.h` file, and you get a working C++ image processing toolkit.
- The compilation is done on the fly : only CImg functionalities really used by your program are compiled and appear in the compiled executable program. This leads to very compact code, without any unused stuff.
- Class members and functions are inlined, leading to better performance during the program execution.

The CImg Library is structured as follows :

- All library classes and functions are defined in the namespace `cimg_library`. This namespace encapsulates the library functionalities and avoid any class name collision that could happen with other includes. Generally, one uses this namespace as a default namespace :

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

- The namespace `cimg_library::cimg` defines a set of *low-level* functions and variables used by the library. Documented functions in this namespace can be safely used in your own program. But, **never** use the `cimg_library::cimg` namespace as a default namespace, since it contains functions whose names are already defined in the standard C/C++ library.

- The class `cimg_library::CImg` represents images up to 4-dimensions wide, containing pixels of type `T` (template parameter). This is actually the main class of the library.
- The class `cimg_library::CImgList` represents lists of `cimg_library::CImg<T>` images. It can be used for instance to store different frames of an image sequence.
- The class `cimg_library::CImgDisplay` is able to display images or image lists into graphical display windows. As you may guess, the code of this class is highly system-dependent but this is transparent for the programmer, as environment variables are automatically set by the CImg library (see also [Setting Environment Variables](#)).
- The class `cimg_library::CImgException` (and its subclasses) are used by the library to throw exceptions when errors occur. Those exceptions can be caught with a `try { .. } catch (CImgException) { .. }` block. Subclasses define precisely the type of encountered errors.

Knowing these four classes is **enough** to get benefit of the CImg Library functionalities.

6.1.2 CImg version of "Hello world".

Below is some very simple code that creates a "Hello World" image. This shows you basically how a CImg program looks like.

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

int main() {
    CImg<unsigned char> img(640,400,1,3); // Define a 640x400 color image with 8 bits per
        color component.
    img.fill(0); // Set pixel values to 0 (color : black)
    unsigned char purple[] = { 255,0,255 }; // Define a purple color
    img.draw_text(100,100,"Hello World",purple); // Draw a purple "Hello world" at coordinates (100,100).
    img.display("My first CImg code"); // Display the image in a display window.
    return 0;
}
```

Which can be also written in a more compact way as :

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

int main() {
    const unsigned char purple[] = { 255,0,255 };
    CImg<unsigned char>(640,400,1,3,0).draw_text(100,100,"Hello World",purple).
        display("My first CImg code");
    return 0;
}
```

Generally, you can write very small code that performs complex image processing tasks. The CImg Library is very simple to use and provides a lot of interesting algorithms for image manipulation.

6.1.3 How to compile ?

The CImg library is a very light and user-friendly library : only standard system libraries are used. It avoids handling complex dependencies and problems with library compatibility. The only thing you need is a (quite modern) C++ compiler :

- **Microsoft Visual C++ 6.0, Visual Studio.NET and Visual Express Edition** : Use the project files and solution files provided in the CImg Library package (directory "compilation/") to see how it works.

- **Intel ICL compiler** : Use the following command to compile a Clmg-based program with ICL :

```
icl /Ox hello_world.cpp user32.lib gdi32.lib
```

- **g++ (MingW windows version)** : Use the following command to compile a Clmg-based program with g++, on Windows :

```
g++ -o hello_word.exe hello_world.cpp -O2 -lgdi32
```

- **g++ (Linux version)** : Use the following command to compile a Clmg-based program with g++, on Linux :

```
g++ -o hello_word.exe hello_world.cpp -O2 -L/usr/X11R6/lib -lm -lpthread -lX11
```

- **g++ (Solaris version)** : Use the following command to compile a Clmg-based program with g++, on Solaris :

```
g++ -o hello_word.exe hello_world.cpp -O2 -lm -lpthread -R/usr/X11R6/lib -lrt -lnsl -lsocket
```

- **g++ (Mac OS X version)** : Use the following command to compile a Clmg-based program with g++, on Mac OS X :

```
g++ -o hello_word.exe hello_world.cpp -O2 -lm -lpthread -I/usr/X11R6/include -L/usr/X11R6/lib -lm -lpthread -lX11
```

- **Dev-Cpp** : Use the project file provided in the Clmg library package to see how it works.

If you are using other compilers and encounter problems, please [write me](#) since maintaining compatibility is one of the priorities of the Clmg Library. Nevertheless, old compilers that do not respect the C++ standard will not support the Clmg Library.

6.1.4 What's next ?

If you are ready to get more, and to start writing more serious programs with Clmg, you are invited to go to the [Tutorial : Getting Started](#). section.

6.2 FAQ : Frequently Asked Questions.

6.2.1 FAQ Summary

- General information and availability
 - What is the CImg Library ?
 - What platforms are supported ?
 - How is CImg distributed ?
 - What kind of people are concerned by CImg ?
 - What are the specificities of the CeCILL license ?
 - Who is behind CImg ?
- C++ related questions
 - What is the level of C++ knowledge needed to use CImg ?
 - How to use CImg in my own C++ program ?
 - Why is CImg entirely contained in a single header file ?
- Other resources
 - Translations

6.2.2 1. General information and availability

6.2.2.1 1.1. What is the Cimg Library ?

The Cimg Library is an *open-source C++ toolkit for image processing*.

It mainly consists in a (big) single header file `CImg.h` providing a set of C++ classes and functions that can be used in your own sources, to load/save, manage/process and display generic images. It's actually a very simple and pleasant toolkit for coding image processing stuff in C++ : Just include the header file `CImg.h`, and you are ready to handle images in your C++ programs.

6.2.2.2 1.2. What platforms are supported ?

Cimg has been designed with *portability* in mind. It is regularly tested on different architectures and compilers, and should also work on any decent OS having a decent C++ compiler. Before each release, the Cimg Library is compiled under these different configurations :

- PC Linux 32 bits, with g++.
- PC Windows 32 bits, with Visual C++ 6.0.
- PC Windows 32 bits, with Visual C++ Express Edition.
- Sun SPARC Solaris 32 bits, with g++.
- Mac PPC with OS X and g++.

Cimg has a minimal number of dependencies. In its minimal version, it can be compiled only with standard C++ headers. Anyway, it has interesting extension capabilities and can use external libraries to perform specific tasks more efficiently (Fourier Transform computation using FFTW for instance).

6.2.2.3 1.3. How is CImg distributed ?

The CImg Library is freely distributed as a complete .zip compressed package, hosted at the [CImg server](#). The package is distributed under the [CeCILL license](#).

This package contains :

- The main library file [CImg.h](#) (C++ header file).
- Several C++ source code showing [examples of using CImg](#).
- A complete library documentation, in [HTML](#) and [PDF](#) formats.
- Additional [library plug-ins](#) that can be used to extend library capabilities for specific uses.

The CImg Library is a quite lightweight library which is easy to maintain (due to its particular structure), and thus has a fast rythm of release. A new version of the CImg package is released approximately every three months.

6.2.2.4 1.4. What kind of people are concerned by CImg ?

The CImg library is an *image processing* library, primarily intended for computer scientists or students working in the fields of image processing or computer vision, and knowing bases of C++. As the library is handy and really easy to use, it can be also used by any programmer needing occasional tools for dealing with images in C++, since there are no standard library yet for this purpose.

6.2.2.5 1.5. What are the specificities of the CeCILL license ?

The [CeCILL license](#) governs the use of the CImg Library. This is an *open-source* license which gives you rights to access, use, modify and redistribute the source code, under certains conditions. There are two different variants of the CeCILL license used in CImg (namely [CeCILL](#) and [CeCILL-C](#), all open-source), corresponding to different constraints on the source files :

- The [CeCILL-C](#) license is the most permissive one, close to the *GNU LGPL license*, and *applies **only** on the main library file [CImg.h](#)*. Basically, this license allows to use [CImg.h](#) in a closed-source product without forcing you to redistribute the entire software source code. Anyway, if one modifies the [CImg.h](#) source file, one has to redistribute the modified version of the file that must be governed by the same [CeCILL-C](#) license.
- The [CeCILL](#) license applies to all other files (source examples, plug-ins and documentation) of the CImg Library package, and is close (even *compatible*) with the *GNU GPL license*. It *does not allow* the use of these files in closed-source products.

You are invited to read the complete descriptions of the the [CeCILL-C](#) and [CeCILL](#) licenses before releasing a software based on the CImg Library.

6.2.2.6 1.6. Who is behind CImg ?

CImg has been started by [David Tschumperle](#) at the beginning of his PhD thesis, in October 1999. He is still the main coordinator of the project. Since the first release, a growing number of contributors has appeared. Due to the very simple and compact form of the library, submitting a contribution is quite easy and can be fastly integrated into the supported releases. List of contributors can be found on the front page.

6.2.3 2. C++ related questions

6.2.3.1 2.1 What is the level of C++ knowledge needed to use CImg ?

The CImg Library has been designed using C++ templates and object-oriented programming techniques, but in a very accessible level. There are only public classes without any derivation (just like C structures) and there is at most one template parameter for each CImg class (defining the pixel type of the images). The design is simple but clean, making the library accessible even for non professional C++ programmers, while proposing strong extension capabilities for C++ experts.

6.2.3.2 2.2 How to use CImg in my own C++ program ?

Basically, you need to add these two lines in your C++ source code, in order to be able to work with CImg images :

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

6.2.3.3 2.3 Why is CImg entirely contained in a single header file ?

People are often surprised to see that the complete code of the library is contained in a single (big) C++ header file `CImg.h`. There are good practical and technical reasons to do that. Some arguments are listed below to justify this approach, so (I hope) you won't think this is a awkwardly C++ design of the CImg library :

- First, the library is based on *template datatypes* (images with generic pixel type), meaning that the programmer is free to decide what type of image he instanciates in his code. Even if there are roughly a limited number of fully supported types (basically, the "atomic" types of C++ : *unsigned char*, *int*, *float*, ...), this is *not imaginable* to pre-compile the library classes and functions for *all possible atomic datatypes*, since many functions and methods can have two or three arguments having different template parameters. This really means *a huge number* of possible combinations. The size of the object binary file generated to cover all possible cases would be just *colossal*. Is the STL library a pre-compiled one ? No, CImg neither. CImg is not using a classical *.cpp* and *.h* mechanism, just like the STL. Architectures of C++ *template-based* libraries are somewhat special in this sense. This is a proven technical fact.
- Second, why CImg does not have several header files, just like the STL does (one for each class for instance) ? This would be possible of course. There are only 4 classes in CImg, the two most important being *CImg<T>* and *CImgList<T>* representing respectively an image and a collection of images. But contrary to the STL library, these two CImg classes are strongly *inter-dependent*. All CImg algorithms are actually not defined as separate functions acting on containers (as the STL does with his header `<algorithm>`), but are directly methods of the image and image collection classes. This inter-dependence practically means that you will undoubtedly need these two main classes at the same time if you are using CImg. If they were defined in separate header files, you would be forced to include both of them. What is the gain then ? No gain. Concerning the two other classes : You can disable the third most important class *CImgDisplay* of the CImg library, by setting the compilation macro *cimg_display* to 0, avoiding thus to compile this class if you don't use display capabilities of CImg in your code. But to be honest, this is a quite small class and doing this doesn't save much compilation time. The last and fourth class is *CImgException*, which is only few lines long and is obviously required in almost all methods of CImg. Including this one is *mandatory*. As a consequence, having a single header file instead of several ones is just a way for you to avoid including all of them, without any consequences on compilation time. This is both good technical and practical reasons to do like this.

- Third, having a single header file has plenty of advantages : Simplicity for the user, and for the developers (maintenance is in fact easier). Look at the `CImg.h` file, it looks like a mess at a first glance, but it is in fact very well organized and structured. Finding pieces of code in CImg functions or methods is particularly easy and fast. Also, how about the fact that library installation problems just disappear ? Just bring `CImg.h` with you, put it in your source directory, and the library is ready to go !

I admit the compilation time of CImg-based programs can be sometime long, but don't think that it is due to the fact that you are using a single header file. Using several header files wouldn't arrange anything since you would need all of them. Having a pre-compiled library object would be the only solution to speed up compilation time, but it is not possible at all, due to the too much generic nature of the library.

6.2.4 3. Other resources

6.2.4.1 3.1 Translations

This FAQ has been translated to [Serbo-Croatian](#) language by [Web Geeks](#) .

6.3 Setting Environment Variables

The CImg library is a multiplatform library, working on a wide variety of systems. This implies the existence of some *environment variables* that must be correctly defined depending on your current system. Most of the time, the CImg Library defines these variables automatically (for popular systems). Anyway, if your system is not recognized, you will have to set the environment variables by hand. Here is a quick explanations of environment variables.

Setting the environment variables is done with the `#define` keyword. This setting must be done *before including the file `CImg.h`* in your source code. For instance, defining the environment variable `cimg_display` would be done like this :

```
#define cimg_display 0
#include "CImg.h"
...
```

Here are the different environment variables used by the CImg Library :

- **cimg_OS** : This variable defines the type of your Operating System. It can be set to **1** (*Unix*), **2** (*Windows*), or **0** (*Other configuration*). It should be actually auto-detected by the CImg library. If this is not the case (`cimg_OS=0`), you will probably have to tune the environment variables described below.
- **cimg_display** : This variable defines the type of graphical library used to display images in windows. It can be set to **0** (no display library available), **1** (X11-based display) or **2** (Windows-GDI display). If you are running on a system without X11 or Windows-GDI ability, please set this variable to **0**. This will disable the display support, since the CImg Library doesn't contain the necessary code to display images on systems other than X11 or Windows GDI.
- **cimg_use_vt100** : This variable tells the library if the system terminal has VT100 color capabilities. It can be *defined* or *not defined*. Define this variable to get colored output on your terminal, when using the CImg Library.
- **cimg_verbosity** : This variable defines the level of run-time debug messages that will be displayed by the CImg Library. It can be set to **0** (no debug messages), **1** (normal debug messages displayed on standard error), **2** (normal debug messages displayed in modal windows, which is the default value), or **3** (high debug messages). Note that setting this value to **3** may slow down your program since more debug tests are made by the library (particularly to check if pixel access is made outside image boundaries). See also `CImgException` to better understand how debug messages are working.
- **cimg_plugin** : This variable tells the library to use a plugin file to add features to the `CImg<T>` class. Define it with the path of your plugin file, if you want to add member functions to the `CImg<T>` class, without having to modify directly the "`CImg.h`" file. An include of the plugin file is performed in the `CImg<T>` class. If `cimg_plugin` is not specified (default), no include is done.
- **cimglist_plugin** : Same as `cimg_plugin`, but to add features to the `CImgList<T>` class.
- **cimgdisplay_plugin** : Same as `cimg_plugin`, but to add features to the `CImgDisplay<T>` class.

All these compilation variables can be checked, using the function `cimg_library::cimg::info()`, which displays a list of the different configuration variables and their values on the standard error output.

6.4 How to use Clmg library with Visual C++ 2005 Express Edition ?.

6.4.1 How to use Clmg library with Visual C++ 2005 Express Edition ?

This section has been written by Vincent Garcia and Alexandre Fournier from I3S/Sophia_Antipolis.

- Download Clmg library
- Download and install Visual C++ 2005 Express Edition
- Download and install Microsoft Windows SDK
- Configure Visual C++ to take into account Microsoft SDK
 - 1. Go to menu "Tools -> options"
 - 2. Select option "Projects and Solutions -> VC++ Directories"
 - 3. In the select liste "Show directories for", choose "include files", and add C:\Program Files\Microsoft Platform SDK\Include (adapt if needed)
 - 4. In the select liste "Show directories for", choose "library files", and add C:\Program Files\Microsoft Platform SDK\Lib (adapt if needed) Edit file C:\Program Files\Microsoft Visual Studio 8\VC\VCProjectDefaults\corewin_express.vsprops (adapt if needed)
 - 6. 7. Remplace the line AdditionalDependencies="kernel32.lib" /> by AdditionalDependencies="kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib" />
- Restart Visual C++
- Import Clmg library in your main file

6.5 Tutorial : Getting Started.

Let's start to write our first program to get the idea. This will demonstrate how to load and create images, as well as handle image display and mouse events. Assume we want to load a color image `lena.jpg`, smooth it, display it in a windows, and enter an event loop so that clicking a point in the image will draw the (R,G,B) intensity profiles of the corresponding image line (in another window). Yes, that sounds quite complex for a first code, but don't worry, it will be very simple using the `CImg` library ! Well, just look at the code below, it does the task :

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

int main() {
    CImg<unsigned char> image("lena.jpg"), visu(500,400,1,3,0);
    const unsigned char red[] = { 255,0,0 }, green[] = { 0,255,0 }, blue[] = { 0,0,255 };
    image.blur(2.5);
    CImgDisplay main_disp(image,"Click a point"), draw_disp(visu,"Intensity profile");
    while (!main_disp.is_closed() && !draw_disp.is_closed()) {
        main_disp.wait();
        if (main_disp.button() && main_disp.mouse_y()>=0) {
            const int y = main_disp.mouse_y();
            visu.fill(0).draw_graph(image.get_crop(0,y,0,0,image.width()-1,y,0,0),red,1,1,0,255,0);
            visu.draw_graph(image.get_crop(0,y,0,1,image.width()-1,y,0,1),green,1,1,0,255,0);
            visu.draw_graph(image.get_crop(0,y,0,2,image.width()-1,y,0,2),blue,1,1,0,255,0).display(draw_disp);
        }
    }
    return 0;
}
```

Here is a screenshot of the resulting program :

And here is the detailed explanation of the source, line by line :

```
#include "CImg.h"
```

Include the main and only header file of the `CImg` library.

```
using namespace cimg_library;
```

Use the library namespace to ease the declarations afterward.

```
int main() {
```

Definition of the main function.

```
CImg<unsigned char> image("lena.jpg"), visu(500,400,1,3,0);
```

Creation of two instances of images of `unsigned char` pixels. The first image `image` is initialized by reading an image file from the disk. Here, `lena.jpg` must be in the same directory as the current program. Note that you must also have installed the *ImageMagick* package in order to be able to read JPG images. The second image `visu` is initialized as a black color image with dimension `dx=500`, `dy=400`, `dz=1` (here, it is a 2D image, not a 3D one), and `dv=3` (each pixel has 3 'vector' channels R,G,B). The last argument in the constructor defines the default value of the pixel values (here 0, which means that `visu` will be initially black).

```
const unsigned char red[] = { 255,0,0 }, green[] = { 0,255,0 }, blue[] = { 0,0,255 };
```

Definition of three different colors as array of `unsigned char`. This will be used to draw plots with different colors.


```
image.blur(2.5);
```

Blur the image, with a gaussian blur and a standard variation of 2.5. Note that most of the CImg functions have two versions : one that acts in-place (which is the case of blur), and one that returns the result as a new image (the name of the function begins then with `get_`). In this case, one could have also written `image = image.get_blur(2.5);` (more expensive, since it needs an additional copy operation).

```
CImgDisplay main_disp(image,"Click a point"), draw_disp(visu,"Intensity profile");
```

Creation of two display windows, one for the input image image, and one for the image visu which will be display intensity profiles. By default, CImg displays handles events (mouse,keyboard,..). On Windows, there is a way to create fullscreen displays.

```
while (!main_disp.is_closed() && !draw_disp.is_closed()) {
```

Enter the event loop, the code will exit when one of the two display windows is closed.

```
main_disp.wait();
```

Wait for an event (mouse, keyboard,..) in the display window main_disp.

```
if (main_disp.button() && main_disp.mouse_y()>=0) {
```

Test if the mouse button has been clicked on the image area. One may distinguish between the 3 different mouse buttons, but in this case it is not necessary

```
const int y = main_disp.mouse_y();
```

Get the image line y-coordinate that has been clicked.

```
visu.fill(0).draw_graph(image.get_crop(0,y,0,0,image.width()-1,y,0,0),red,1,0,256,0);
```

This line illustrates the pipeline property of most of the CImg class functions. The first function `fill(0)` simply sets all pixel values with 0 (i.e. clear the image visu). The interesting thing is that it returns a reference to visu and then, can be pipelined with the function `draw_graph()` which draws a plot in the image visu. The plot data are given by another image (the first argument of `draw_graph()`). In this case, the given image is the red-component of the line y of the original image, retrieved by the function `get_crop()` which returns a sub-image of the image image. Remember that images coordinates are 4D (x,y,z,v) and for color images, the R,G,B channels are respectively given by `v=0`, `v=1` and `v=2`.

```
visu.draw_graph(image.get_crop(0,y,0,1,image.width()-1,y,0,1),green,1,0,256,0);
```

Plot the intensity profile for the green channel of the clicked line.

```
visu.draw_graph(image.get_crop(0,y,0,2,image.width()-1,y,0,2),blue,1,0,256,0).display(draw_disp);
```

Same thing for the blue channel. Note how the function (which return a reference to visu) is pipelined with the function `display()` that just paints the image visu in the corresponding display window.

```
...till the end
```

I don't think you need more explanations !

As you have noticed, the CImg library allows to write very small and intuitive code. Note also that this source will perfectly work on Unix and Windows systems. Take also a look to the examples provided in the CImg package (directory `examples/`). It will show you how CImg-based code can be surprisingly small. Moreover, there is surely one example close to what you want to do. A good start will be to look at the file `CImg_demo.cpp` which contains small and various examples of what you can do with the CImg Library. All CImg classes are used in this source, and the code can be easily modified to see what happens.

6.6 Using Drawing Functions.

6.6.1 Using Drawing Functions.

This section tells more about drawing features in CImg images. Drawing functions list can be found in [the CImg functions list](#) (section **Drawing** Functions), and are all defined on a common basis. Here are the important points to understand before using drawing functions :

- Drawing is performed on the instance image. Drawing functions parameters are defined as *const* variables and return a reference to the current instance (**this*), so that drawing functions can be pipelined (see examples below). Drawing is usually done in 2D color images but can be performed in 3D images with any vector-valued dimension, and with any possible pixel type.
- A color parameter is always needed to draw features in an image. The color must be defined as a C-style array whose dimension is at least

6.7 Using Image Loops.

The CImg Library provides different macros that define useful iterative loops over an image. Basically, it can be used to replace one or several `for (. .)` instructions, but it also proposes interesting extensions to classical loops. Below is a list of all existing loop macros, classified in four different categories :

- [Loops over the pixel buffer](#)
- [Loops over image dimensions](#)
- [Loops over interior regions and borders.](#)
- [Loops using neighborhoods.](#)

6.7.1 Loops over the pixel buffer

Loops over the pixel buffer are really basic loops that iterate a pointer on the pixel data buffer of a `cimg_library::CImg` image. Two macros are defined for this purpose :

- **`cimg_for(img,ptr,T)`** : This macro loops over the pixel data buffer of the image `img`, using a pointer `T* ptr`, starting from the beginning of the buffer (first pixel) till the end of the buffer (last pixel).
 - `img` must be a (non empty) `cimg_library::CImg` image of pixels `T`.
 - `ptr` is a pointer of type `T*`. This kind of loop should not appear a lot in your own source code, since this is a low-level loop and many functions of the CImg class may be used instead. Here is an example of use :


```
CImg<float> img(320,200);
cimg_for(img,ptr,float) { *ptr=0; }           // Equivalent to 'img.fill(0);'
```
- **`cimg_rof(img,ptr,T)`** : This macro does the same as `cimg_for()` but from the end to the beginning of the pixel buffer.
- **`cimg_foroff(img,off)`** : This macro loops over the pixel data buffer of the image `img`, using an offset , starting from the beginning of the buffer (first pixel, `off=0`) till the end of the buffer (last pixel value, `off = img.size()-1`).
 - `img` must be a (non empty) `cimg_library::CImg<T>` image of pixels `T`.
 - `off` is an inner-loop variable, only defined inside the scope of the loop.

Here is an example of use :

```
CImg<float> img(320,200);
cimg_foroff(img,off) { img[off]=0; }           // Equivalent to 'img.fill(0);'
```

6.7.2 Loops over image dimensions

The following loops are probably the most used loops in image processing programs. They allow to loop over the image along one or several dimensions, along a raster scan course. Here is the list of such loop macros for a single dimension :

- **cimg_forX(img,x)** : equivalent to : `for (int x = 0; x<img.width(); x++)`.
- **cimg_forY(img,y)** : equivalent to : `for (int y = 0; y<img.height(); y++)`.
- **cimg_forZ(img,z)** : equivalent to : `for (int z = 0; z<img.depth(); z++)`.
- **cimg_forC(img,v)** : equivalent to : `for (int v = 0; v<img.spectrum(); v++)`.

Combinations of these macros are also defined as other loop macros, allowing to loop directly over 2D, 3D or 4D images :

- **cimg_forXY(img,x,y)** : equivalent to : `cimg_forY(img, y) cimg_forX(img, x)`.
- **cimg_forXZ(img,x,z)** : equivalent to : `cimg_forZ(img, z) cimg_forX(img, x)`.
- **cimg_forYZ(img,y,z)** : equivalent to : `cimg_forZ(img, z) cimg_forY(img, y)`.
- **cimg_forXC(img,x,v)** : equivalent to : `cimg_forC(img, v) cimg_forX(img, x)`.
- **cimg_forYC(img,y,v)** : equivalent to : `cimg_forC(img, v) cimg_forY(img, y)`.
- **cimg_forZC(img,z,v)** : equivalent to : `cimg_forC(img, v) cimg_forZ(img, z)`.
- **cimg_forXYZ(img,x,y,z)** : equivalent to : `cimg_forZ(img, z) cimg_forXY(img, x, y)`.
- **cimg_forXYC(img,x,y,v)** : equivalent to : `cimg_forC(img, v) cimg_forXY(img, x, y)`.
- **cimg_forXZC(img,x,z,v)** : equivalent to : `cimg_forC(img, v) cimg_forXZ(img, x, z)`.
- **cimg_forYZC(img,y,z,v)** : equivalent to : `cimg_forC(img, v) cimg_forYZ(img, y, z)`.
- **cimg_forXYZC(img,x,y,z,v)** : equivalent to : `cimg_forC(img, v) cimg_forXYZ(img, x, y, z)`.
- For all these loops, `x,y,z` and `v` are inner-defined variables only visible inside the scope of the loop. They don't have to be defined before the call of the macro.
- `img` must be a (non empty) [cimg_library::CImg](#) image.

Here is an example of use that creates an image with a smooth color gradient :

```
CImg<unsigned char> img(256,256,1,3);           // Define a 256x256 color image
cimg_forXYC(img,x,y,v) { img(x,y,v) = (x+y)*(v+1)/6; }
img.display("Color gradient");
```

6.7.3 Loops over interior regions and borders.

Similar macros are also defined to loop only on the border of an image, or inside the image (excluding the border). The border may be several pixel wide :

- **cimg_for_insideX(img,x,n)** : Loop along the x-axis, except for pixels inside a border of *n* pixels wide.
- **cimg_for_insideY(img,y,n)** : Loop along the y-axis, except for pixels inside a border of *n* pixels wide.
- **cimg_for_insideZ(img,z,n)** : Loop along the z-axis, except for pixels inside a border of *n* pixels wide.
- **cimg_for_insideC(img,v,n)** : Loop along the v-axis, except for pixels inside a border of *n* pixels wide.
- **cimg_for_insideXY(img,x,y,n)** : Loop along the (x,y)-axes, excepted for pixels inside a border of *n* pixels wide.
- **cimg_for_insideXYZ(img,x,y,z,n)** : Loop along the (x,y,z)-axes, excepted for pixels inside a border of *n* pixels wide.

And also :

- **cimg_for_borderX(img,x,n)** : Loop along the x-axis, only for pixels inside a border of *n* pixels wide.
- **cimg_for_borderY(img,y,n)** : Loop along the y-axis, only for pixels inside a border of *n* pixels wide.
- **cimg_for_borderZ(img,z,n)** : Loop along the z-axis, only for pixels inside a border of *n* pixels wide.
- **cimg_for_borderC(img,v,n)** : Loop along the z-axis, only for pixels inside a border of *n* pixels wide.
- **cimg_for_borderXY(img,x,y,n)** : Loop along the (x,y)-axes, only for pixels inside a border of *n* pixels wide.
- **cimg_for_borderXYZ(img,x,y,z,n)** : Loop along the (x,y,z)-axes, only for pixels inside a border of *n* pixels wide.
- For all these loops, *x*, *y*, *z* and *v* are inner-defined variables only visible inside the scope of the loop. They don't have to be defined before the call of the macro.
- *img* must be a (non empty) [cimg_library::CImg](#) image.
- The constant *n* stands for the size of the border.

Here is an example of use, to create a 2d grayscale image with two different intensity gradients :

```
CImg<> img(256,256);
cimg_for_insideXY(img,x,y,50) img(x,y) = x+y;
cimg_for_borderXY(img,x,y,50) img(x,y) = x-y;
img.display();
```

6.7.4 Loops using neighborhoods.

Inside an image loop, it is often useful to get values of neighborhood pixels of the current pixel at the loop location. The CImg Library provides a very smart and fast mechanism for this purpose, with the definition of several loop macros that remember the neighborhood values of the pixels. The use of these macros can highly optimize your code, and also simplify your program.

6.7.4.1 Neighborhood-based loops for 2D images

For 2D images, the neighborhood-based loop macros are :

- **cimg_for2x2(img,x,y,z,v,I,T)** : Loop along the (x,y)-axes using a centered 2x2 neighborhood.
- **cimg_for3x3(img,x,y,z,v,I,T)** : Loop along the (x,y)-axes using a centered 3x3 neighborhood.
- **cimg_for4x4(img,x,y,z,v,I,T)** : Loop along the (x,y)-axes using a centered 4x4 neighborhood.
- **cimg_for5x5(img,x,y,z,v,I,T)** : Loop along the (x,y)-axes using a centered 5x5 neighborhood.

For all these loops, `x` and `y` are inner-defined variables only visible inside the scope of the loop. They don't have to be defined before the call of the macro. `img` is a non empty `CImg<T>` image. `z` and `v` are constants that define on which image slice and vector channel the loop must apply (usually both 0 for grayscale 2D images). Finally, `I` is the 2x2, 3x3, 4x4 or 5x5 neighborhood of type `T` that will be updated with the correct pixel values during the loop (see [Defining neighborhoods](#)).

6.7.4.2 Neighborhood-based loops for 3D images

For 3D images, the neighborhood-based loop macros are :

- **cimg_for2x2x2(img,x,y,z,v,I,T)** : Loop along the (x,y,z)-axes using a centered 2x2x2 neighborhood.
- **cimg_for3x3x3(img,x,y,z,v,I,T)** : Loop along the (x,y,z)-axes using a centered 3x3x3 neighborhood.

For all these loops, `x`, `y` and `z` are inner-defined variables only visible inside the scope of the loop. They don't have to be defined before the call of the macro. `img` is a non empty `CImg<T>` image. `v` is a constant that defines on which image channel the loop must apply (usually 0 for grayscale 3D images). Finally, `I` is the 2x2x2 or 3x3x3 neighborhood of type `T` that will be updated with the correct pixel values during the loop (see [Defining neighborhoods](#)).

6.7.4.3 Defining neighborhoods

A neighborhood is defined as an instance of a class having `operator[]` defined. This particularly includes classical C-array, as well as `CImg<T>` objects.

For instance, a 3x3 neighborhood can be defined either as a `'float[9]'` or a `'CImg<float>(3,3)'` variable.

6.7.4.4 Using alternate variable names

There are also some useful macros that can be used to define variables that reference the neighborhood elements. There are :

- **CImg_2x2(I,type)** : Define a 2x2 neighborhood named I, of type type.
- **CImg_3x3(I,type)** : Define a 3x3 neighborhood named I, of type type.
- **CImg_4x4(I,type)** : Define a 4x4 neighborhood named I, of type type.
- **CImg_5x5(I,type)** : Define a 5x5 neighborhood named I, of type type.
- **CImg_2x2x2(I,type)** : Define a 2x2x2 neighborhood named I, of type type.
- **CImg_3x3x3(I,type)** : Define a 3x3x3 neighborhood named I, of type type.

Actually, I is a *generic name* for the neighborhood. In fact, these macros declare a *set* of new variables. For instance, defining a 3x3 neighborhood `CImg_3x3(I, float)` declares 9 different float variables `Ipp,Icp,Inp,Ipc,Icc,Inc,Ipn,Icn,Inn` which correspond to each pixel value of a 3x3 neighborhood. Variable indices are *p,c* or *n*, and stand respectively for '*previous*', '*current*' and '*next*'. First indice denotes the *x-axis*, second indice denotes the *y-axis*. Then, the names of the variables are directly related to the position of the corresponding pixels in the neighborhood. For 3D neighborhoods, a third indice denotes the *z-axis*. Then, inside a neighborhood loop, you will have the following equivalence :

- `Ipp = img(x-1,y-1)`
- `Icn = img(x,y+1)`
- `Inp = img(x+1,y-1)`
- `Inpc = img(x+1,y-1,z)`
- `Ippn = img(x-1,y-1,z+1)`
- and so on...

For bigger neighborhoods, such as 4x4 or 5x5 neighborhoods, two additionnal indices are introduced : *a* (stands for '*after*') and *b* (stands for '*before*'), so that :

- `Ibb = img(x-2,y-2)`
- `Ina = img(x+1,y+2)`
- and so on...

The value of a neighborhood pixel outside the image range (image border problem) is automatically set to the same values as the nearest valid pixel in the image (this is also called the *Neumann border condition*).

6.7.4.5 Example codes

More than a long discussion, the above example will demonstrate how to compute the gradient norm of a 3D volume using the `cimg_for3x3x3()` loop macro :

```
CImg<float> volume("IRM.hdr");    // Load an IRM volume from an Analyze7.5 file
CImg_3x3x3(I,float);             // Define a 3x3x3 neighborhood
CImg<float> gradnorm(volume);     // Create an image with same size as 'volume'
cimg_for3x3x3(volume,x,y,z,0,I,float) { // Loop over the volume, using the neighborhood I
    const float ix = 0.5f*(Incc-Ipcc); // Compute the derivative along the x-axis.
    const float iy = 0.5f*(Icnc-Ipcp); // Compute the derivative along the y-axis.
    const float iz = 0.5f*(Iccn-Iccp); // Compute the derivative along the z-axis.
    gradnorm(x,y,z) = std::sqrt(ix*ix+iy*iy+iz*iz); // Set the gradient norm in the destination image
}
gradnorm.display("Gradient norm");
```

And the following example shows how to deal with neighborhood references to blur a color image by averaging pixel values on a 5x5 neighborhood.

```
CImg<unsigned char> src("image_color.jpg"), dest(src,false); // Image definitions.
typedef unsigned char uchar; // Avoid space in the second parameter of the macro CImg_5x5x1
    below.
CImg<> N(5,5); // Define a 5x5 neighborhood as a 5x5 image.
cimg_forC(src,k) // Standard loop on color channels
    cimg_for5x5(src,x,y,0,k,N,float) // 5x5 neighborhood loop.
        dest(x,y,k) = N.sum()/(5*5); // Averaging pixels to filter the color image.
CImgList<unsigned char> visu(src,dest);
visu.display("Original + Filtered"); // Display both original and filtered image.
```

As you can see, explaining the use of the `CImg` neighborhood macros is actually more difficult than using them !

6.8 Using Display Windows.

When opening a display window, you can choose the way the pixel values will be normalized before being displayed on the screen. Screen displays only support color values between [0,255], and some

When displaying an image into the display window using `CImgDisplay::display()`, values of the image pixels can be eventually linearly normalized between [0,255] for visualization purposes. This may be useful for instance when displaying `CImg<double>` images with pixel values between [0,1]. The normalization behavior depends on the value of `normalize` which can be either 0,1 or 2 :

- 0 : No pixel normalization is performed when displaying an image. This is the fastest process, but you must be sure your displayed image have pixel values inside the range [0,255].
- 1 : Pixel value normalization is done for each new image display. Image pixels are not modified themselves, only displayed pixels are normalized.
- 2 : Pixel value normalization is done for the first image display, then the normalization parameters are kept and used for all the next image displays.

6.9 How pixel data are stored with CImg.

First, `CImg<T>` are *very* basic structures, which means that there are no memory tricks, weird memory alignments or disk caches used to store pixel data of images. When an image is instanced, all its pixel values are stored in memory at the same time (yes, you should avoid working with huge images when dealing with CImg, if you have only 64kb of RAM).

A `CImg<T>` is basically a 4th-dimensional array (width,height,depth,dim), and its pixel data are stored linearly in a single memory buffer of general size (width*height*depth*dim). Nothing more, nothing less. The address of this memory buffer can be retrieved by the function `CImg<T>::data()`. As each image value is stored as a type `T` (`T` being known by the programmer of course), this pointer is a `T*`, or a `const T*` if your image is 'const'. so, `T *ptr = img.data()` gives you the pointer to the first value of the image 'img'. The overall size of the used memory for one instance image (in bytes) is then `'width*height*depth*dim*sizeof(T)'`.

Now, the ordering of the pixel values in this buffer follows these rules : The values are *not* interleaved, and are ordered first along the X,Y,Z and V axis respectively (corresponding to the width,height,depth,dim dimensions), starting from the upper-left pixel to the bottom-right pixel of the instane image, with a classical scanline run.

So, a color image with dim=3 and depth=1, will be stored in memory as :

`R1R2R3R4R5R6.....G1G2G3G4G5G6.....B1B2B3B4B5B6....` (i.e following a 'planar' structure)

and *not* as `R1G1B1R2G2B2R3G3B3...` (interleaved channels), where `R1 = img(0,0,0,0)` is the first upper-left pixel of the red component of the image, `R2 = img(1,0,0,0)`, `G1 = img(0,0,0,1)`, `G2 = img(1,0,0,1)`, `B1 = img(0,0,0,2)`, and so on...

Another example, a (1x5x1x1) `CImg<T>` (column vector A) will be stored as : `A1A2A3A4A5` where `A1 = img(0,0)`, `A2 = img(0,1)`, ... , `A5 = img(0,4)`.

As you see, it is *very* simple and intuitive : no interleaving, no padding, just simple. This is cool not only because it is simple, but this has in fact a number of interesting properties. For instance, a 2D color image is stored in memory exactly as a 3D scalar image having a depth=3, meaning that when you are dealing with 2D color images, you can write `'img(x,y,k)'` instead of `'img(x,y,0,k)'` to access the kth channel of the (x,y) pixel. More generally, if you have one dimension that is 1 in your image, you can just skip it in the call to the operator(). Similarly, values of a column vector stored as an image with width=depth=spectrum=1 can be accessed by `'img(y)'` instead of `'img(0,y)'`. This is very convenient.

Another cool thing is that it allows you to work easily with 'shared' images. A shared image is a `CImg<T>` instance that shares its memory with another one (the 'base' image). Destroying a shared image does nothing in fact. Shared images is a convenient way of modifying only *portions* (consecutive in memory) of an image. For instance, if 'img' is a 2D color image, you can write :

```
img.get_shared_channel(0).blur(2); img.get_shared_channels(1,2).mirror('x');
```

which just blur the red channel of the image, and mirror the two others along the X-axis. This is possible since channels of an image are not interleaved but are stored as different consecutive planes in memory, so you see that constructing a shared image is possible (and trivial).

6.10 Files IO in CImg.

The CImg Library can NATIVELY handle the following file formats :

- RAW : consists in a very simple header (in ascii), then the image data.
- ASC (Ascii)
- HDR (Analyze 7.5)
- INR (Inrimage)
- PPM/PGM (Portable Pixmap)
- BMP (uncompressed)
- PAN (Pandore-5)
- DLM (Matlab ASCII)

If ImageMagick is installed, The CImg Library can save image in formats handled by ImageMagick : JPG, GIF, PNG, TIF,...

6.11 Retrieving Command Line Arguments.

The CImg library offers facilities to retrieve command line arguments in a console-based program, as it is a commonly needed operation. Three macros `cimg_usage()`, `cimg_help()` and `cimg_option()` are defined for this purpose. Using these macros allows to easily retrieve options values from the command line. Invoking the compiled executable with the option `-h` or `-help` will automatically display the program usage, followed by the list of requested options.

6.11.1 The `cimg_usage()` macro

The macro `cimg_usage(usage)` may be used to describe the program goal and usage. It is generally inserted one time after the `int main(int argc, char **argv)` definition.

Parameters

<code>usage</code>	: A string describing the program goal and usage.
--------------------	---

Precondition

The function where `cimg_usage()` is used must have correctly defined `argc` and `argv` variables.

6.11.2 The `cimg_help()` macro

The macro `cimg_help(str)` will display the string `str` only if the `-help` or `-help` option are invoked when running the program.

6.11.3 The `cimg_option()` macro

The macro `cimg_option(name, default, usage)` may be used to retrieve an option value from the command line.

Parameters

<code>name</code>	: The name of the option to be retrieved from the command line.
<code>default</code>	: The default value returned by the macro if no options <code>name</code> has been specified when running the program.
<code>usage</code>	: A brief explanation of the option. If <code>usage==0</code> , the option won't appear on the option list when invoking the executable with options <code>-h</code> or <code>-help</code> (hidden option).

Returns

`cimg_option()` returns an object that has the *same type* as the default value `default`. The return value is equal to the one specified on the command line. If no such option have been specified, the return value is equal to the default value `default`. Warning, this can be confusing in some situations (look at the end of the next section).

Precondition

The function where `cimg_option()` is used must have correctly defined `argc` and `argv` variables.

6.11.4 Example of use

The code below uses the macros `cimg_usage()` and `cimg_option()`. It loads an image, smoothes it and quantifies it with a specified number of values.

```
#include "CImg.h"
using namespace cimg_library;
int main(int argc, char **argv) {
    cimg_usage("Retrieve command line arguments");
    const char* filename = cimg_option("-i", "image.gif", "Input image file");
    const char* output = cimg_option("-o", (char*)0, "Output image file");
    const double sigma = cimg_option("-s", 1.0, "Standard variation of the gaussian smoothing");
    const int nlevels = cimg_option("-n", 16, "Number of quantification levels");
    const bool hidden = cimg_option("-hidden", false, 0); // This is a hidden option

    CImg<unsigned char> img(filename);
    img.blur(sigma).quantize(nlevels);
    if (output) img.save(output); else img.display("Output image");
    if (hidden) std::fprintf(stderr, "You found me !\n");
    return 0;
}
```

Invoking the corresponding executable with `test -h -hidden -n 20 -i foo.jpg` will display :

```
./test -h -hidden -n 20 -i foo.jpg

test : Retrieve command line arguments (Oct 16 2004, 12:34:26)

-i      = foo.jpg      : Input image file
-o      = 0            : Output image file
-s      = 1            : Standard variation of the gaussian smoothing
-n      = 20           : Number of quantification levels

You found me !
```

Warning

As the type of object returned by the macro `cimg_option(option, default, usage)` is defined by the type of `default`, undesired casts may appear when writing code such as :

```
const double sigma = cimg_option("-val", 0, "A floating point value");
```

In this case, `sigma` will always be equal to an integer (since the default value `0` is an integer). When passing a float value on the command line, a *float to integer* cast is then done, truncating the given parameter to an integer value (this is surely not a desired behavior). You must specify `0.0` as the default value in this case.

6.11.5 How to learn more about command line options ?

You should take a look at the examples `examples/gmic.cpp` provided in the CImg Library package. This is a command line based image converter which intensively uses the `cimg_option()` and `cimg_usage()` macros to retrieve command line parameters.

Chapter 7

Namespace Documentation

7.1 cimg_library Namespace Reference

Contains *all classes and functions* of the CImg library.

Namespaces

- [cimg](#)

Contains low-level functions and variables of the CImg Library.

Classes

- struct [CImg](#)

Class representing an image (up to 4 dimensions wide), each pixel being of type T.

- struct [CImgDisplay](#)

Allow the creation of windows, display images on them and manage user events (keyboard, mouse and windows events).

- struct [CImgException](#)

Instances of [CImgException](#) are thrown when errors are encountered in a CImg function call.

- struct [CImgList](#)

Represent a list of images CImg<T>.

7.1.1 Detailed Description

Contains *all classes and functions* of the CImg library.

This namespace is defined to avoid functions and class names collisions that could happen with the inclusion of other C++ header files. Anyway, it should not happen often and you should reasonably start most of your CImg-based programs with

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

to simplify the declaration of CImg Library objects afterwards.

7.2 cimg_library::cimg Namespace Reference

Contains *low-level* functions and variables of the CImg Library.

Functions

- `std::FILE * output (std::FILE *file)`
Get/set default output stream for the CImg library messages.
- `void info ()`
Print information about CImg environnement variables.
- `template<typename T >`
`void unused (const T &,...)`
Avoid warning messages due to unused parameters. Do nothing actually.
- `unsigned int & exception_mode (const unsigned int mode)`
Set current CImg exception mode.
- `unsigned int & exception_mode ()`
Return current CImg exception mode.
- `unsigned int & openmp_mode ()`
Return current CImg openmp mode.
- `int dialog (const char *const title, const char *const msg, const char *const button1_label, const char *const button2_label, const char *const button3_label, const char *const button4_label, const char *const button5_label, const char *const button6_label, const bool is_centered)`
*Display a simple dialog box, and wait for the user's response [**specialization**].*
- `double eval (const char *const expression, const double x, const double y, const double z, const double c)`
Evaluate math expression.
- `void warn (const char *const format,...)`
Display a warning message on the default output stream.
- `int system (const char *const command, const char *const module_name=0)`
- `template<typename T >`
`T & temporary (const T &)`
Return a reference to a temporary variable of type T.
- `template<typename T >`
`void swap (T &a, T &b)`
Exchange values of variables a and b.
- `template<typename T1 , typename T2 >`
`void swap (T1 &a1, T1 &b1, T2 &a2, T2 &b2)`
Exchange values of variables (a1,a2) and (b1,b2).
- `template<typename T1 , typename T2 , typename T3 >`
`void swap (T1 &a1, T1 &b1, T2 &a2, T2 &b2, T3 &a3, T3 &b3)`
Exchange values of variables (a1,a2,a3) and (b1,b2,b3).
- `template<typename T1 , typename T2 , typename T3 , typename T4 >`
`void swap (T1 &a1, T1 &b1, T2 &a2, T2 &b2, T3 &a3, T3 &b3, T4 &a4, T4 &b4)`
Exchange values of variables (a1,a2,...,a4) and (b1,b2,...,b4).
- `template<typename T1 , typename T2 , typename T3 , typename T4 , typename T5 >`
`void swap (T1 &a1, T1 &b1, T2 &a2, T2 &b2, T3 &a3, T3 &b3, T4 &a4, T4 &b4, T5 &a5, T5 &b5)`
Exchange values of variables (a1,a2,...,a5) and (b1,b2,...,b5).
- `template<typename T1 , typename T2 , typename T3 , typename T4 , typename T5 , typename T6 >`
`void swap (T1 &a1, T1 &b1, T2 &a2, T2 &b2, T3 &a3, T3 &b3, T4 &a4, T4 &b4, T5 &a5, T5 &b5, T6 &a6, T6 &b6)`
Exchange values of variables (a1,a2,...,a6) and (b1,b2,...,b6).

- `template<typename T1 , typename T2 , typename T3 , typename T4 , typename T5 , typename T6 , typename T7 >`
`void swap (T1 &a1, T1 &b1, T2 &a2, T2 &b2, T3 &a3, T3 &b3, T4 &a4, T4 &b4, T5 &a5, T5 &b5, T6 &a6, T6 &b6, T7 &a7, T7 &b7)`
Exchange values of variables (a1,a2,...,a7) and (b1,b2,...,b7).
- `template<typename T1 , typename T2 , typename T3 , typename T4 , typename T5 , typename T6 , typename T7 , typename T8 >`
`void swap (T1 &a1, T1 &b1, T2 &a2, T2 &b2, T3 &a3, T3 &b3, T4 &a4, T4 &b4, T5 &a5, T5 &b5, T6 &a6, T6 &b6, T7 &a7, T7 &b7, T8 &a8, T8 &b8)`
Exchange values of variables (a1,a2,...,a8) and (b1,b2,...,b8).
- `bool endianness ()`
Return the endianness of the current architecture.
- `template<typename T >`
`void invert_endianness (T *const buffer, const unsigned long size)`
Reverse endianness of all elements in a memory buffer.
- `template<typename T >`
`T & invert_endianness (T &a)`
Reverse endianness of a single variable.
- `unsigned long time ()`
Return the value of a system timer, with a millisecond precision.
- `unsigned long tic ()`
Start tic/toc timer for time measurement between code instructions.
- `unsigned long toc ()`
End tic/toc timer and displays elapsed time from last call to tic().
- `void sleep (const unsigned int milliseconds)`
Sleep for a given numbers of milliseconds.
- `long wait (const unsigned int milliseconds)`
Wait for a given number of milliseconds since the last call to wait().
- `double rand (const double val_min, const double val_max)`
Return a random variable uniformly distributed between [val_min,val_max].
- `double rand (const double val_max=1)`
Return a random variable uniformly distributed between [0,val_max].
- `double grand ()`
Return a random variable following a gaussian distribution and a standard deviation of 1.
- `unsigned int prand (const double z)`
Return a random variable following a Poisson distribution of parameter z.
- `template<typename T >`
`T rol (const T &a, const unsigned int n=1)`
Bitwise-rotate value on the left.
- `template<typename T >`
`T ror (const T &a, const unsigned int n=1)`
Bitwise-rotate value on the right.
- `template<typename T >`
`T abs (const T &a)`
Return absolute value of a value.
- `template<typename T >`
`T sqr (const T &val)`
Return square of a value.
- `int xln (const int x)`
Return $1 + \log_{10}(x)$ of a value x.
- `template<typename t1 , typename t2 >`
`cimg::superset< t1, t2 >::type min (const t1 &a, const t2 &b)`
Return the minimum between two values.

- `template<typename t1 , typename t2 , typename t3 >`
`cimg::superset2< t1, t2, t3 >::type min` (const t1 &a, const t2 &b, const t3 &c)
Return the minimum between three values.
- `template<typename t1 , typename t2 , typename t3 , typename t4 >`
`cimg::superset3< t1, t2, t3, t4 >::type min` (const t1 &a, const t2 &b, const t3 &c, const t4 &d)
Return the minimum between four values.
- `template<typename t1 , typename t2 >`
`cimg::superset< t1, t2 >::type max` (const t1 &a, const t2 &b)
Return the maximum between two values.
- `template<typename t1 , typename t2 , typename t3 >`
`cimg::superset2< t1, t2, t3 >::type max` (const t1 &a, const t2 &b, const t3 &c)
Return the maximum between three values.
- `template<typename t1 , typename t2 , typename t3 , typename t4 >`
`cimg::superset3< t1, t2, t3, t4 >::type max` (const t1 &a, const t2 &b, const t3 &c, const t4 &d)
Return the maximum between four values.
- `template<typename T >`
`T sign` (const T &x)
Return the sign of a value.
- `template<typename T >`
`unsigned long nearest_pow2` (const T &x)
Return the nearest power of 2 higher than given value.
- `double sinc` (const double x)
Return the sinc of a given value.
- `template<typename T >`
`T mod` (const T &x, const T &m)
Return the modulo of a value.
- `template<typename T >`
`T minmod` (const T &a, const T &b)
Return the min-mod of two values.
- `double log2` (const double x)
Return base-2 logarithm of a value.
- `template<typename T >`
`T round` (const T &x, const double y=1, const int rounding_type=0)
Return rounded value.
- `double hypot` (const double x, const double y)
Return $\sqrt{x^2 + y^2}$.
- `double factorial` (const int n)
Return the factorial of n.
- `double permutations` (const int k, const int n, const bool with_order)
Return the number of permutations of k objects in a set of n objects.
- `double fibonacci` (const int n)
Return fibonacci number.
- `char lowercase` (const char x)
Convert ascii character to lower case.
- `void lowercase` (char *const str)
Convert C-string to lower case.
- `char uppercase` (const char x)
Convert ascii character to upper case.
- `void uppercase` (char *const str)
Convert C-string to upper case.
- `double atof` (const char *const str)
Read value in a C-string.

- int [strncasecmp](#) (const char *const str1, const char *const str2, const int l)
Compare the first l characters of two C-strings, ignoring the case.
- int [strcasecmp](#) (const char *const str1, const char *const str2)
Compare two C-strings, ignoring the case.
- char * [strellipsize](#) (char *const str, const unsigned int l=64, const bool is_ending=true)
Ellipsize a string.
- char * [strellipsize](#) (const char *const str, char *const res, const unsigned int l=64, const bool is_ending=true)
Ellipsize a string.
- bool [strpare](#) (char *const str, const char delimiter, const bool is_symmetric, const bool is_iterative)
Remove delimiters on the start and/or end of a C-string.
- void [strwindows_reserved](#) (char *const str, const char c='_')
Replace reserved characters (for Windows filename) by another character.
- void [strunescape](#) (char *const str)
Replace escape sequences in C-strings by their binary ascii values.
- const char * [basename](#) (const char *const s, const char separator= '/')
Return the basename of a filename.
- std::FILE * [fopen](#) (const char *const path, const char *const mode)
Open a file.
- int [fclose](#) (std::FILE *file)
Close a file.
- int [fseek](#) (FILE *stream, long offset, int origin)
Version of 'fseek()' that supports >=64bits offsets everywhere (for Windows).
- long [ftell](#) (FILE *stream)
Version of 'ftell()' that supports >=64bits offsets everywhere (for Windows).
- bool [is_directory](#) (const char *const path)
Check if a path is a directory.
- bool [is_file](#) (const char *const path)
Check if a path is a file.
- int [fdate](#) (const char *const path, const unsigned int attr)
Get last write time of a given file or directory.
- int [date](#) (const unsigned int attr)
Get current local time.
- const char * [temporary_path](#) (const char *const user_path, const bool reinit_path)
Get/set path to store temporary files.
- const char * [imagemagick_path](#) (const char *const user_path, const bool reinit_path)
Get/set path to the Program Files/ directory (Windows only).
- const char * [graphicsmagick_path](#) (const char *const user_path, const bool reinit_path)
Get/set path to the GraphicsMagick's gm binary.
- const char * [medcon_path](#) (const char *const user_path, const bool reinit_path)
Get/set path to the XMedcon's medcon binary.
- const char * [ffmpeg_path](#) (const char *const user_path, const bool reinit_path)
Get/set path to the FFmpeg's ffmpeg binary.
- const char * [gzip_path](#) (const char *const user_path, const bool reinit_path)
Get/set path to the gzip binary.
- const char * [gunzip_path](#) (const char *const user_path, const bool reinit_path)
Get/set path to the gunzip binary.
- const char * [dcraw_path](#) (const char *const user_path, const bool reinit_path)
Get/set path to the dcraw binary.
- const char * [wget_path](#) (const char *const user_path, const bool reinit_path)
Get/set path to the wget binary.
- const char * [curl_path](#) (const char *const user_path, const bool reinit_path)

- Get/set path to the `curl` binary.*

 - `const char * split_filename (const char *const filename, char *const body=0)`

Split filename into two C-strings `body` and `extension`.
- `char * number_filename (const char *const filename, const int number, const unsigned int digits, char *const str)`

Generate a numbered version of a filename.
- `template<typename T >
size_t fread (T *const ptr, const size_t nmemb, std::FILE *stream)`

Read data from file.
- `template<typename T >
size_t fwrite (const T *ptr, const size_t nmemb, std::FILE *stream)`

Write data to file.
- `void fempty (std::FILE *const file, const char *const filename)`

Create an empty file.
- `const char * ftype (std::FILE *const file, const char *const filename)`

Try to guess format from an image file.
- `char * load_network (const char *const url, char *const filename_local, const unsigned int timeout, const bool try_fallback, const char *const referer)`

Load file from network as a local temporary file.
- `const char * option (const char *const name, const int argc, const char *const *const argv, const char *const default, const char *const usage, const bool reset_static)`

Return options specified on the command line.
- `CImgList< char > files (const char *const path, const bool is_pattern=false, const unsigned int mode=2, const bool include_path=false)`

Return list of files/directories in specified directory.
- `template<typename t >
int dialog (const char *const title, const char *const msg, const char *const button1_label, const char *const button2_label, const char *const button3_label, const char *const button4_label, const char *const button5_label, const char *const button6_label, const CImg< t > &logo, const bool is_centered=false)`

Display a simple dialog box, and wait for the user's response.

Variables

- `const unsigned int keyESC = 1U`

Keycode for the `ESC` key (architecture-dependent).
- `const unsigned int keyF1 = 2U`

Keycode for the `F1` key (architecture-dependent).
- `const unsigned int keyF2 = 3U`

Keycode for the `F2` key (architecture-dependent).
- `const unsigned int keyF3 = 4U`

Keycode for the `F3` key (architecture-dependent).
- `const unsigned int keyF4 = 5U`

Keycode for the `F4` key (architecture-dependent).
- `const unsigned int keyF5 = 6U`

Keycode for the `F5` key (architecture-dependent).
- `const unsigned int keyF6 = 7U`

Keycode for the `F6` key (architecture-dependent).
- `const unsigned int keyF7 = 8U`

Keycode for the `F7` key (architecture-dependent).
- `const unsigned int keyF8 = 9U`

Keycode for the `F8` key (architecture-dependent).

- const unsigned int **keyF9** = 10U
Keycode for the F9 key (architecture-dependent).
- const unsigned int **keyF10** = 11U
Keycode for the F10 key (architecture-dependent).
- const unsigned int **keyF11** = 12U
Keycode for the F11 key (architecture-dependent).
- const unsigned int **keyF12** = 13U
Keycode for the F12 key (architecture-dependent).
- const unsigned int **keyPAUSE** = 14U
Keycode for the PAUSE key (architecture-dependent).
- const unsigned int **key1** = 15U
Keycode for the 1 key (architecture-dependent).
- const unsigned int **key2** = 16U
Keycode for the 2 key (architecture-dependent).
- const unsigned int **key3** = 17U
Keycode for the 3 key (architecture-dependent).
- const unsigned int **key4** = 18U
Keycode for the 4 key (architecture-dependent).
- const unsigned int **key5** = 19U
Keycode for the 5 key (architecture-dependent).
- const unsigned int **key6** = 20U
Keycode for the 6 key (architecture-dependent).
- const unsigned int **key7** = 21U
Keycode for the 7 key (architecture-dependent).
- const unsigned int **key8** = 22U
Keycode for the 8 key (architecture-dependent).
- const unsigned int **key9** = 23U
Keycode for the 9 key (architecture-dependent).
- const unsigned int **key0** = 24U
Keycode for the 0 key (architecture-dependent).
- const unsigned int **keyBACKSPACE** = 25U
Keycode for the BACKSPACE key (architecture-dependent).
- const unsigned int **keyINSERT** = 26U
Keycode for the INSERT key (architecture-dependent).
- const unsigned int **keyHOME** = 27U
Keycode for the HOME key (architecture-dependent).
- const unsigned int **keyPAGEUP** = 28U
Keycode for the PAGEUP key (architecture-dependent).
- const unsigned int **keyTAB** = 29U
Keycode for the TAB key (architecture-dependent).
- const unsigned int **keyQ** = 30U
Keycode for the Q key (architecture-dependent).
- const unsigned int **keyW** = 31U
Keycode for the W key (architecture-dependent).
- const unsigned int **keyE** = 32U
Keycode for the E key (architecture-dependent).
- const unsigned int **keyR** = 33U
Keycode for the R key (architecture-dependent).
- const unsigned int **keyT** = 34U
Keycode for the T key (architecture-dependent).
- const unsigned int **keyY** = 35U

- Keycode for the Y key (architecture-dependent).*

 - const unsigned int `keyU` = 36U
- Keycode for the U key (architecture-dependent).*

 - const unsigned int `keyI` = 37U
- Keycode for the I key (architecture-dependent).*

 - const unsigned int `keyO` = 38U
- Keycode for the O key (architecture-dependent).*

 - const unsigned int `keyP` = 39U
- Keycode for the P key (architecture-dependent).*

 - const unsigned int `keyDELETE` = 40U
- Keycode for the DELETE key (architecture-dependent).*

 - const unsigned int `keyEND` = 41U
- Keycode for the END key (architecture-dependent).*

 - const unsigned int `keyPAGEDOWN` = 42U
- Keycode for the PAGEDOWN key (architecture-dependent).*

 - const unsigned int `keyCAPSLOCK` = 43U
- Keycode for the CAPSLOCK key (architecture-dependent).*

 - const unsigned int `keyA` = 44U
- Keycode for the A key (architecture-dependent).*

 - const unsigned int `keyS` = 45U
- Keycode for the S key (architecture-dependent).*

 - const unsigned int `keyD` = 46U
- Keycode for the D key (architecture-dependent).*

 - const unsigned int `keyF` = 47U
- Keycode for the F key (architecture-dependent).*

 - const unsigned int `keyG` = 48U
- Keycode for the G key (architecture-dependent).*

 - const unsigned int `keyH` = 49U
- Keycode for the H key (architecture-dependent).*

 - const unsigned int `keyJ` = 50U
- Keycode for the J key (architecture-dependent).*

 - const unsigned int `keyK` = 51U
- Keycode for the K key (architecture-dependent).*

 - const unsigned int `keyL` = 52U
- Keycode for the L key (architecture-dependent).*

 - const unsigned int `keyENTER` = 53U
- Keycode for the ENTER key (architecture-dependent).*

 - const unsigned int `keySHIFTLEFT` = 54U
- Keycode for the SHIFTLEFT key (architecture-dependent).*

 - const unsigned int `keyZ` = 55U
- Keycode for the Z key (architecture-dependent).*

 - const unsigned int `keyX` = 56U
- Keycode for the X key (architecture-dependent).*

 - const unsigned int `keyC` = 57U
- Keycode for the C key (architecture-dependent).*

 - const unsigned int `keyV` = 58U
- Keycode for the V key (architecture-dependent).*

 - const unsigned int `keyB` = 59U
- Keycode for the B key (architecture-dependent).*

 - const unsigned int `keyN` = 60U
- Keycode for the N key (architecture-dependent).*

- const unsigned int [keyM](#) = 61U
Keycode for the M key (architecture-dependent).
- const unsigned int [keySHIFTRIGHT](#) = 62U
Keycode for the SHIFTRIGHT key (architecture-dependent).
- const unsigned int [keyARROWUP](#) = 63U
Keycode for the ARROWUP key (architecture-dependent).
- const unsigned int [keyCTRLLEFT](#) = 64U
Keycode for the CTRLLEFT key (architecture-dependent).
- const unsigned int [keyAPPLEFT](#) = 65U
Keycode for the APPLEFT key (architecture-dependent).
- const unsigned int [keyALT](#) = 66U
Keycode for the ALT key (architecture-dependent).
- const unsigned int [keySPACE](#) = 67U
Keycode for the SPACE key (architecture-dependent).
- const unsigned int [keyALTGR](#) = 68U
Keycode for the ALTGR key (architecture-dependent).
- const unsigned int [keyAPPRIGHT](#) = 69U
Keycode for the APPRIGHT key (architecture-dependent).
- const unsigned int [keyMENU](#) = 70U
Keycode for the MENU key (architecture-dependent).
- const unsigned int [keyCTRLRIGHT](#) = 71U
Keycode for the CTRLRIGHT key (architecture-dependent).
- const unsigned int [keyARROWLEFT](#) = 72U
Keycode for the ARROWLEFT key (architecture-dependent).
- const unsigned int [keyARROWDOWN](#) = 73U
Keycode for the ARROWDOWN key (architecture-dependent).
- const unsigned int [keyARROWRIGHT](#) = 74U
Keycode for the ARROWRIGHT key (architecture-dependent).
- const unsigned int [keyPAD0](#) = 75U
Keycode for the PAD0 key (architecture-dependent).
- const unsigned int [keyPAD1](#) = 76U
Keycode for the PAD1 key (architecture-dependent).
- const unsigned int [keyPAD2](#) = 77U
Keycode for the PAD2 key (architecture-dependent).
- const unsigned int [keyPAD3](#) = 78U
Keycode for the PAD3 key (architecture-dependent).
- const unsigned int [keyPAD4](#) = 79U
Keycode for the PAD4 key (architecture-dependent).
- const unsigned int [keyPAD5](#) = 80U
Keycode for the PAD5 key (architecture-dependent).
- const unsigned int [keyPAD6](#) = 81U
Keycode for the PAD6 key (architecture-dependent).
- const unsigned int [keyPAD7](#) = 82U
Keycode for the PAD7 key (architecture-dependent).
- const unsigned int [keyPAD8](#) = 83U
Keycode for the PAD8 key (architecture-dependent).
- const unsigned int [keyPAD9](#) = 84U
Keycode for the PAD9 key (architecture-dependent).
- const unsigned int [keyPADADD](#) = 85U
Keycode for the PADADD key (architecture-dependent).
- const unsigned int [keyPADSUB](#) = 86U

- Keycode for the `PADSUB` key (architecture-dependent).*
 - const unsigned int `keyPADMUL` = 87U
- Keycode for the `PADMUL` key (architecture-dependent).*
 - const unsigned int `keyPADDIV` = 88U
- Keycode for the `PADDDIV` key (architecture-dependent).*
 - const double `PI` = 3.14159265358979323846
- Value of the mathematical constant `PI`.*

7.2.1 Detailed Description

Contains *low-level* functions and variables of the `CImg` Library.

Most of the functions and variables within this namespace are used by the `CImg` library for low-level operations. You may use them to access specific const values or environment variables internally used by `CImg`.

Warning

Never write `using namespace cimg_library::cimg;` in your source code. Lot of functions in the `cimg:: namespace` have the same names as standard C functions that may be defined in the global namespace `::`.

7.2.2 Function Documentation

7.2.2.1 `std::FILE * output (std::FILE * file)`

Get/set default output stream for the `CImg` library messages.

Parameters

<i>file</i>	Desired output stream. Set to 0 to get the currently used output stream only.
-------------	---

Returns

Currently used output stream.

7.2.2.2 `void info ()`

Print information about `CImg` environnement variables.

Note

Output is done on the default output stream.

7.2.2.3 `unsigned int& cimg_library::cimg::exception_mode (const unsigned int mode)`

Set current `CImg` exception mode.

The way error messages are handled by `CImg` can be changed dynamically, using this function.

Parameters

<i>mode</i>	Desired exception mode. Possible values are: <ul style="list-style-type: none"> • 0: Hide library messages (quiet mode). • 1: Print library messages on the console. • 2: Display library messages on a dialog window (default behavior). • 3: Do as 1 + add extra debug warnings (slow down the code!). • 4: Do as 2 + add extra debug warnings (slow down the code!).
-------------	--

7.2.2.4 unsigned int& cimg_library::cimg::exception_mode ()

Return current CImg exception mode.

Note

By default, return the value of configuration macro `cimg_verbosity`

7.2.2.5 double eval (const char *const expression, const double x, const double y, const double z, const double c)

Evaluate math expression.

Parameters

<i>expression</i>	C-string describing the formula to evaluate.
<i>x</i>	Value of the pre-defined variable <code>x</code> .
<i>y</i>	Value of the pre-defined variable <code>y</code> .
<i>z</i>	Value of the pre-defined variable <code>z</code> .
<i>c</i>	Value of the pre-defined variable <code>c</code> .

Returns

Result of the formula evaluation.

Note

Set `expression` to 0 to keep evaluating the last specified `expression`.

Example

```
const double
res1 = cimg::eval("cos(x)^2 + sin(y)^2",2,2), // will return '1'.
res2 = cimg::eval(0,1,1);                  // will return '1' too.
```

7.2.2.6 void cimg_library::cimg::warn (const char *const format, ...)

Display a warning message on the default output stream.

Parameters

<i>format</i>	C-string containing the format of the message, as with <code>std::printf()</code> .
---------------	---

Note

If configuration macro `cimg_strict_warnings` is set, this function throws a `CImgWarningException` instead.

Warning

As the first argument is a format string, it is highly recommended to write

```
cimg::warn("%s", warning_message);
```

instead of

```
cimg::warn(warning_message);
```

if `warning_message` can be arbitrary, to prevent nasty memory access.

7.2.2.7 `int cimg_library::cimg::system (const char *const command, const char *const module_name = 0)`

Parameters

<i>command</i>	C-string containing the command line to execute.
<i>module_name</i>	Module name.

Returns

Status value of the executed command, whose meaning is OS-dependent.

Note

This function is similar to `std::system()` but it does not open an extra console windows on Windows-based systems.

7.2.2.8 `bool cimg_library::cimg::endianness ()`

Return the endianness of the current architecture.

Returns

`false` for *Little Endian* or `true` for *Big Endian*.

7.2.2.9 `void cimg_library::cimg::invert_endianness (T *const buffer, const unsigned long size)`

Reverse endianness of all elements in a memory buffer.

Parameters

<i>in, out</i>	<i>buffer</i>	Memory buffer whose endianness must be reversed.
	<i>size</i>	Number of buffer elements to reverse.

7.2.2.10 T& cimg_library::cimg::invert_endianness (T & a)

Reverse endianness of a single variable.

Parameters

<i>in, out</i>	<i>a</i>	Variable to reverse.
----------------	----------	----------------------

Returns

Reference to reversed variable.

7.2.2.11 unsigned long cimg_library::cimg::time ()

Return the value of a system timer, with a millisecond precision.

Note

The timer does not necessarily starts from 0.

7.2.2.12 unsigned long cimg_library::cimg::tic ()

Start tic/toc timer for time measurement between code instructions.

Returns

Current value of the timer (same value as [time\(\)](#)).

7.2.2.13 unsigned long cimg_library::cimg::toc ()

End tic/toc timer and displays elapsed time from last call to [tic\(\)](#).

Returns

Time elapsed (in ms) since last call to [tic\(\)](#).

7.2.2.14 void cimg_library::cimg::sleep (const unsigned int *milliseconds*)

Sleep for a given numbers of milliseconds.

Parameters

<i>milliseconds</i>	Number of milliseconds to wait for.
---------------------	-------------------------------------

Note

This function frees the CPU resources during the sleeping time. It can be used to temporize your program properly, without wasting CPU time.

7.2.2.15 `long cimg_library::cimg::wait (const unsigned int milliseconds)`

Wait for a given number of milliseconds since the last call to [wait\(\)](#).

Parameters

<i>milliseconds</i>	Number of milliseconds to wait for.
---------------------	-------------------------------------

Returns

Number of milliseconds elapsed since the last call to [wait\(\)](#).

Note

Same as [sleep\(\)](#) with a waiting time computed with regard to the last call of [wait\(\)](#). It may be used to temporize your program properly, without wasting CPU time.

7.2.2.16 `T cimg_library::cimg::mod (const T & x, const T & m)`

Return the modulo of a value.

Parameters

<i>x</i>	Input value.
<i>m</i>	Modulo value.

Note

This modulo function accepts negative and floating-points modulo numbers, as well as variables of any type.

7.2.2.17 `T cimg_library::cimg::minmod (const T & a, const T & b)`

Return the min-mod of two values.

Note

minmod(a,b) is defined to be:

- *minmod(a,b) = min(a,b)*, if *a* and *b* have the same sign.
- *minmod(a,b) = 0*, if *a* and *b* have different signs.

7.2.2.18 `T cimg_library::cimg::round (const T & x, const double y = 1, const int rounding_type = 0)`

Return rounded value.

Parameters

<i>x</i>	Value to be rounded.
<i>y</i>	Rounding precision.
<i>rounding_type</i>	Type of rounding operation (0 = nearest, -1 = backward, 1 = forward).

Returns

Rounded value, having the same type as input value *x*.

7.2.2.19 `double cimg_library::cimg::atof (const char *const str)`

Read value in a C-string.

Parameters

<i>str</i>	C-string containing the float value to read.
------------	--

Returns

Read value.

Note

Same as `std::atof()` extended to manage the retrieval of fractions from C-strings, as in "1/2".

7.2.2.20 `int cimg_library::cimg::strncasecmp (const char *const str1, const char *const str2, const int l)`

Compare the first *l* characters of two C-strings, ignoring the case.

Parameters

<i>str1</i>	C-string.
<i>str2</i>	C-string.
<i>l</i>	Number of characters to compare.

Returns

0 if the two strings are equal, something else otherwise.

Note

This function has to be defined since it is not provided by all C++-compilers (not ANSI).

7.2.2.21 `int cimg_library::cimg::strcasecmp (const char *const str1, const char *const str2)`

Compare two C-strings, ignoring the case.

Parameters

<i>str1</i>	C-string.
<i>str2</i>	C-string.

Returns

0 if the two strings are equal, something else otherwise.

Note

This function has to be defined since it is not provided by all C++-compilers (not ANSI).

7.2.2.22 `char* cimg_library::cimg::strellipse (char *const str, const unsigned int l = 64, const bool is_ending = true)`

Ellipsize a string.

Parameters

<i>str</i>	C-string.
<i>l</i>	Max number of characters.
<i>is_ending</i>	Tell if the dots are placed at the end or at the center of the ellipsized string.

7.2.2.23 `char* cimg_library::cimg::strellipse (const char *const str, char *const res, const unsigned int l = 64, const bool is_ending = true)`

Ellipsize a string.

Parameters

<i>str</i>	C-string.
<i>res</i>	output C-string.
<i>l</i>	Max number of characters.
<i>is_ending</i>	Tell if the dots are placed at the end or at the center of the ellipsized string.

7.2.2.24 `bool cimg_library::cimg::strpare (char *const str, const char delimiter, const bool is_symmetric, const bool is_iterative)`

Remove delimiters on the start and/or end of a C-string.

Parameters

<i>in, out</i>	<i>str</i>	C-string to work with (modified at output).
	<i>delimiter</i>	Delimiter character code to remove.
	<i>is_symmetric</i>	Tells if the removal is done only if delimiters are symmetric (both at the beginning and the end of <i>s</i>).
	<i>is_iterative</i>	Tells if the removal is done if several iterations are possible.

Returns

`true` if delimiters have been removed, `false` otherwise.

7.2.2.25 void cimg_library::cimg::strwindows_reserved (char *const *str*, const char *c* = ' _ ')

Replace reserved characters (for Windows filename) by another character.

Parameters

<i>in, out</i>	<i>str</i>	C-string to work with (modified at output).
<i>in</i>	<i>c</i>	Replacement character.

7.2.2.26 void cimg_library::cimg::strunescape (char *const *str*)

Replace escape sequences in C-strings by their binary ascii values.

Parameters

<i>in, out</i>	<i>str</i>	C-string to work with (modified at output).
----------------	------------	---

7.2.2.27 std::FILE* cimg_library::cimg::fopen (const char *const *path*, const char *const *mode*)

Open a file.

Parameters

<i>path</i>	Path of the filename to open.
<i>mode</i>	C-string describing the opening mode.

Returns

Opened file.

Note

Same as `std::fopen()` but throw a `CImgIOException` when the specified file cannot be opened, instead of returning 0.

7.2.2.28 `int cimg_library::cimg::fclose (std::FILE * file)`

Close a file.

Parameters

<i>file</i>	File to close.
-------------	----------------

Returns

0 if file has been closed properly, something else otherwise.

Note

Same as `std::fclose()` but display a warning message if the file has not been closed properly.

7.2.2.29 `bool cimg_library::cimg::is_directory (const char *const path)`

Check if a path is a directory.

Parameters

<i>path</i>	Specified path to test.
-------------	-------------------------

7.2.2.30 `bool cimg_library::cimg::is_file (const char *const path)`

Check if a path is a file.

Parameters

<i>path</i>	Specified path to test.
-------------	-------------------------

7.2.2.31 `int cimg_library::cimg::fdate (const char *const path, const unsigned int attr)`

Get last write time of a given file or directory.

Parameters

<i>path</i>	Specified path to get attributes from.
<i>attr</i>	Type of requested time attribute. Can be { 0=year 1=month 2=day 3=day of week 4=hour 5=minute 6=second }

Returns

-1 if requested attribute could not be read.

7.2.2.32 int cimg_library::cimg::date (const unsigned int *attr*)

Get current local time.

Parameters

<i>attr</i>	Type of requested time attribute. Can be { 0=year 1=month 2=day 3=day of week 4=hour 5=minute 6=second }
-------------	--

7.2.2.33 const char * temporary_path (const char *const *user_path*, const bool *reinit_path*)

Get/set path to store temporary files.

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path where temporary files can be saved.

7.2.2.34 const char * imagemagick_path (const char *const *user_path*, const bool *reinit_path*)

Get/set path to the *Program Files*/ directory (Windows only).

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path containing the program files. Get/set path to the ImageMagick's `convert` binary.

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path containing the `convert` binary.

7.2.2.35 `const char * graphicsmagick_path (const char *const user_path, const bool reinit_path)`

Get/set path to the GraphicsMagick's `gm` binary.

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path containing the `gm` binary.

7.2.2.36 `const char * medcon_path (const char *const user_path, const bool reinit_path)`

Get/set path to the XMedcon's `medcon` binary.

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path containing the `medcon` binary.

7.2.2.37 `const char * ffmpeg_path (const char *const user_path, const bool reinit_path)`

Get/set path to the FFMPEG's `ffmpeg` binary.

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path containing the `ffmpeg` binary.

7.2.2.38 `const char * gzip_path (const char *const user_path, const bool reinit_path)`

Get/set path to the `gzip` binary.

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path containing the `gzip` binary.

7.2.2.39 `const char * gzip_path (const char *const user_path, const bool reinit_path)`

Get/set path to the `gzip` binary.

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path containing the `gunzip` binary.

7.2.2.40 `const char * ddraw_path (const char *const user_path, const bool reinit_path)`

Get/set path to the `ddraw` binary.

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path containing the `ddraw` binary.

7.2.2.41 `const char * wget_path (const char *const user_path, const bool reinit_path)`

Get/set path to the `wget` binary.

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path containing the `wget` binary.

7.2.2.42 `const char * curl_path (const char *const user_path, const bool reinit_path)`

Get/set path to the `curl` binary.

Parameters

<i>user_path</i>	Specified path, or 0 to get the path currently used.
<i>reinit_path</i>	Force path to be recalculated (may take some time).

Returns

Path containing the `curl` binary.

7.2.2.43 `const char* cimg_library::cimg::split_filename (const char *const filename, char *const body = 0)`

Split filename into two C-strings `body` and `extension`.

filename and body must not overlap!

7.2.2.44 `size_t cimg_library::cimg::fread (T *const ptr, const size_t nmemb, std::FILE * stream)`

Read data from file.

Parameters

<i>out</i>	<i>ptr</i>	Pointer to memory buffer that will contain the binary data read from file.
	<i>nmemb</i>	Number of elements to read.
	<i>stream</i>	File to read data from.

Returns

Number of read elements.

Note

Same as `std::fread()` but may display warning message if all elements could not be read.

7.2.2.45 `size_t cimg_library::cimg::fwrite (const T * ptr, const size_t nmemb, std::FILE * stream)`

Write data to file.

Parameters

	<i>ptr</i>	Pointer to memory buffer containing the binary data to write on file.
	<i>nmemb</i>	Number of elements to write.
out	<i>stream</i>	File to write data on.

Returns

Number of written elements.

Note

Similar to `std::fwrite` but may display warning messages if all elements could not be written.

7.2.2.46 void cimg_library::cimg::fempty (std::FILE *const *file*, const char *const *filename*)

Create an empty file.

Parameters

<i>file</i>	Input file (can be 0 if <i>filename</i> is set).
<i>filename</i>	Filename, as a C-string (can be 0 if <i>file</i> is set).

7.2.2.47 const char * ftype (std::FILE *const *file*, const char *const *filename*)

Try to guess format from an image file.

Parameters

<i>file</i>	Input file (can be 0 if <i>filename</i> is set).
<i>filename</i>	Filename, as a C-string (can be 0 if <i>file</i> is set).

Returns

C-string containing the guessed file format, or 0 if nothing has been guessed.

7.2.2.48 char * load_network (const char *const *url*, char *const *filename_local*, const unsigned int *timeout*, const bool *try_fallback*, const char *const *referer*)

Load file from network as a local temporary file.

Parameters

	<i>filename</i>	Filename, as a C-string.
out	<i>filename_local</i>	C-string containing the path to a local copy of <i>filename</i> .
	<i>timeout</i>	Maximum time (in seconds) authorized for downloading the file from the URL.
Generated by Doxygen	<i>try_fallback</i>	When using libcurl, tells using system calls as fallbacks in case of libcurl failure.

Returns

Value of `filename_local`.

Note

Use the `libcurl` library, or the external binaries `wget` or `curl` to perform the download.

7.2.2.49 `CImgList<char> cimg_library::cimg::files (const char *const path, const bool is_pattern = false, const unsigned int mode = 2, const bool include_path = false)`

Return list of files/directories in specified directory.

Parameters

<i>path</i>	Path to the directory. Set to 0 for current directory.
<i>is_pattern</i>	Tell if specified path has a matching pattern in it.
<i>mode</i>	Output type, can be primary { 0=files only 1=folders only 2=files + folders }.
<i>include_path</i>	Tell if <i>path</i> must be included in resulting filenames.

Returns

A list of filenames.

7.2.2.50 `int cimg_library::cimg::dialog (const char *const title, const char *const msg, const char *const button1_label, const char *const button2_label, const char *const button3_label, const char *const button4_label, const char *const button5_label, const char *const button6_label, const CImg< t > & logo, const bool is_centered = false)`

Display a simple dialog box, and wait for the user's response.

Parameters

<i>title</i>	Title of the dialog window.
<i>msg</i>	Main message displayed inside the dialog window.
<i>button1_label</i>	Label of the 1st button.
<i>button2_label</i>	Label of the 2nd button (0 to hide button).
<i>button3_label</i>	Label of the 3rd button (0 to hide button).
<i>button4_label</i>	Label of the 4th button (0 to hide button).
<i>button5_label</i>	Label of the 5th button (0 to hide button).
<i>button6_label</i>	Label of the 6th button (0 to hide button).
<i>logo</i>	Image logo displayed at the left of the main message.
<i>is_centered</i>	Tells if the dialog window must be centered on the screen.

Returns

Indice of clicked button (from 0 to 5), or -1 if the dialog window has been closed by the user.

Note

- Up to 6 buttons can be defined in the dialog window.
- The function returns when a user clicked one of the button or closed the dialog window.
- If a button text is set to 0, the corresponding button (and the followings) will not appear in the dialog box. At least one button must be specified.

Chapter 8

Class Documentation

8.1 CImg< T > Struct Template Reference

Class representing an image (up to 4 dimensions wide), each pixel being of type `T`.

Public Types

- `typedef T * iterator`
Simple iterator type, to loop through each pixel value of an image instance.
- `typedef const T * const_iterator`
Simple const iterator type, to loop through each pixel value of a `const` image instance.
- `typedef T value_type`
Pixel value type.

Constructors / Destructor / Instance Management

- `~CImg ()`
Destroy image.
- `CImg ()`
Construct empty image.
- `CImg (const unsigned int size_x, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1)`
Construct image with specified size.
- `CImg (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const T &value)`
Construct image with specified size and initialize pixel values.
- `CImg (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const int value0, const int value1,...)`
Construct image with specified size and initialize pixel values from a sequence of integers.
- `CImg (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const double value0, const double value1,...)`
Construct image with specified size and initialize pixel values from a sequence of doubles.
- `CImg (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const char *const values, const bool repeat_values)`
Construct image with specified size and initialize pixel values from a value string.

- `template<typename t >`
`Clmg (const t *const values, const unsigned int size_x, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1, const bool is_shared=false)`
Construct image with specified size and initialize pixel values from a memory buffer.
- `Clmg (const T *const values, const unsigned int size_x, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1, const bool is_shared=false)`
*Construct image with specified size and initialize pixel values from a memory buffer **[specialization]**.*
- `Clmg (const char *const filename)`
Construct image from reading an image file.
- `template<typename t >`
`Clmg (const Clmg< t > &img)`
Construct image copy.
- `Clmg (const Clmg< T > &img)`
*Construct image copy **[specialization]**.*
- `template<typename t >`
`Clmg (const Clmg< t > &img, const bool is_shared)`
Advanced copy constructor.
- `Clmg (const Clmg< T > &img, const bool is_shared)`
*Advanced copy constructor **[specialization]**.*
- `template<typename t >`
`Clmg (const Clmg< t > &img, const char *const dimensions)`
Construct image with dimensions borrowed from another image.
- `template<typename t >`
`Clmg (const Clmg< t > &img, const char *const dimensions, const T &value)`
Construct image with dimensions borrowed from another image and initialize pixel values.
- `Clmg (const ClmgDisplay &disp)`
Construct image from a display window.
- `Clmg< T > & assign ()`
*Construct empty image **[in-place version]**.*
- `Clmg< T > & assign (const unsigned int size_x, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1)`
*Construct image with specified size **[in-place version]**.*
- `Clmg< T > & assign (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const T &value)`
*Construct image with specified size and initialize pixel values **[in-place version]**.*
- `Clmg< T > & assign (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const int value0, const int value1,...)`
*Construct image with specified size and initialize pixel values from a sequence of integers **[in-place version]**.*
- `Clmg< T > & assign (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const double value0, const double value1,...)`
*Construct image with specified size and initialize pixel values from a sequence of doubles **[in-place version]**.*
- `Clmg< T > & assign (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const char *const values, const bool repeat_values)`
*Construct image with specified size and initialize pixel values from a value string **[in-place version]**.*
- `template<typename t >`
`Clmg< T > & assign (const t *const values, const unsigned int size_x, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1)`
*Construct image with specified size and initialize pixel values from a memory buffer **[in-place version]**.*
- `Clmg< T > & assign (const T *const values, const unsigned int size_x, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1)`
*Construct image with specified size and initialize pixel values from a memory buffer **[specialization]**.*

- `template<typename t >`
`Clmg< T > & assign (const t *const values, const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const bool is_shared)`
Construct image with specified size and initialize pixel values from a memory buffer [overloading].
- `Clmg< T > & assign (const T *const values, const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const bool is_shared)`
Construct image with specified size and initialize pixel values from a memory buffer [overloading].
- `Clmg< T > & assign (const char *const filename)`
Construct image from reading an image file [in-place version].
- `template<typename t >`
`Clmg< T > & assign (const Clmg< t > &img)`
Construct image copy [in-place version].
- `template<typename t >`
`Clmg< T > & assign (const Clmg< t > &img, const bool is_shared)`
In-place version of the advanced copy constructor.
- `template<typename t >`
`Clmg< T > & assign (const Clmg< t > &img, const char *const dimensions)`
Construct image with dimensions borrowed from another image [in-place version].
- `template<typename t >`
`Clmg< T > & assign (const Clmg< t > &img, const char *const dimensions, const T &value)`
Construct image with dimensions borrowed from another image and initialize pixel values [in-place version].
- `Clmg< T > & assign (const ClmgDisplay &disp)`
Construct image from a display window [in-place version].
- `Clmg< T > & clear ()`
Construct empty image [in-place version].
- `template<typename t >`
`Clmg< t > & move_to (Clmg< t > &img)`
Transfer content of an image instance into another one.
- `Clmg< T > & move_to (Clmg< T > &img)`
Transfer content of an image instance into another one [specialization].
- `template<typename t >`
`ClmgList< t > & move_to (ClmgList< t > &list, const unsigned int pos=~0U)`
Transfer content of an image instance into a new image in an image list.
- `Clmg< T > & swap (Clmg< T > &img)`
Swap fields of two image instances.
- `static Clmg< T > & empty ()`
Return a reference to an empty image.
- `static const Clmg< T > & const_empty ()`
Return a reference to an empty image [const version].

Overloaded Operators

- `T & operator()` (const unsigned int x, const unsigned int y=0, const unsigned int z=0, const unsigned int c=0)
Access to a pixel value.
- `const T & operator()` (const unsigned int x, const unsigned int y=0, const unsigned int z=0, const unsigned int c=0) const
Access to a pixel value [const version].
- `T & operator()` (const unsigned int x, const unsigned int y, const unsigned int z, const unsigned int c, const unsigned longT wh, const unsigned longT whd=0)
Access to a pixel value.

- `const T & operator()` (const unsigned int x, const unsigned int y, const unsigned int z, const unsigned int c, const unsigned longT wh, const unsigned longT whd=0) const
Access to a pixel value [const version].
- `operator T * ()`
Implicitly cast an image into a T.*
- `operator const T * () const`
Implicitly cast an image into a T [const version].*
- `CImg< T > & operator=` (const T &value)
Assign a value to all image pixels.
- `CImg< T > & operator=` (const char *const expression)
Assign pixels values from a specified expression.
- `template<typename t >`
`CImg< T > & operator=` (const CImg< t > &img)
Copy an image into the current image instance.
- `CImg< T > & operator=` (const CImg< T > &img)
Copy an image into the current image instance [specialization].
- `CImg< T > & operator=` (const CImgDisplay &disp)
Copy the content of a display window to the current image instance.
- `template<typename t >`
`CImg< T > & operator+=` (const t value)
In-place addition operator.
- `CImg< T > & operator+=` (const char *const expression)
In-place addition operator.
- `template<typename t >`
`CImg< T > & operator+=` (const CImg< t > &img)
In-place addition operator.
- `CImg< T > & operator++ ()`
In-place increment operator (prefix).
- `CImg< T > operator++` (int)
In-place increment operator (postfix).
- `CImg< T > operator+ () const`
Return a non-shared copy of the image instance.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > operator+` (const t value) const
Addition operator.
- `CImg< Tfloat > operator+` (const char *const expression) const
Addition operator.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > operator+` (const CImg< t > &img) const
Addition operator.
- `template<typename t >`
`CImg< T > & operator-=` (const t value)
In-place subtraction operator.
- `CImg< T > & operator-=` (const char *const expression)
In-place subtraction operator.
- `template<typename t >`
`CImg< T > & operator-=` (const CImg< t > &img)
In-place subtraction operator.
- `CImg< T > & operator-- ()`
In-place decrement operator (prefix).
- `CImg< T > operator--` (int)
In-place decrement operator (postfix).

- `CImg< T > operator- () const`
Replace each pixel by its opposite value.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > operator- (const t value) const`
Subtraction operator.
- `CImg< Tfloat > operator- (const char *const expression) const`
Subtraction operator.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > operator- (const CImg< t > &img) const`
Subtraction operator.
- `template<typename t >`
`CImg< T > & operator*= (const t value)`
In-place multiplication operator.
- `CImg< T > & operator*= (const char *const expression)`
In-place multiplication operator.
- `template<typename t >`
`CImg< T > & operator*= (const CImg< t > &img)`
In-place multiplication operator.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > operator* (const t value) const`
Multiplication operator.
- `CImg< Tfloat > operator* (const char *const expression) const`
Multiplication operator.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > operator* (const CImg< t > &img) const`
Multiplication operator.
- `template<typename t >`
`CImg< T > & operator/= (const t value)`
In-place division operator.
- `CImg< T > & operator/= (const char *const expression)`
In-place division operator.
- `template<typename t >`
`CImg< T > & operator/= (const CImg< t > &img)`
In-place division operator.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > operator/ (const t value) const`
Division operator.
- `CImg< Tfloat > operator/ (const char *const expression) const`
Division operator.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > operator/ (const CImg< t > &img) const`
Division operator.
- `template<typename t >`
`CImg< T > & operator%= (const t value)`
In-place modulo operator.
- `CImg< T > & operator%= (const char *const expression)`
In-place modulo operator.
- `template<typename t >`
`CImg< T > & operator%= (const CImg< t > &img)`
In-place modulo operator.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > operator% (const t value) const`

- Modulo operator.*

 - `CImg< Tfloat > operator% (const char *const expression) const`

Modulo operator.

 - `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > operator% (const CImg< t > &img) const`

Modulo operator.

 - `template<typename t >`
`CImg< T > & operator&= (const t value)`

In-place bitwise AND operator.

 - `CImg< T > & operator&= (const char *const expression)`

In-place bitwise AND operator.

 - `template<typename t >`
`CImg< T > & operator&= (const CImg< t > &img)`

In-place bitwise AND operator.

 - `template<typename t >`
`CImg< T > operator& (const t value) const`

Bitwise AND operator.

 - `CImg< T > operator& (const char *const expression) const`

Bitwise AND operator.

 - `template<typename t >`
`CImg< T > operator& (const CImg< t > &img) const`

Bitwise AND operator.

 - `template<typename t >`
`CImg< T > & operator|= (const t value)`

In-place bitwise OR operator.

 - `CImg< T > & operator|= (const char *const expression)`

In-place bitwise OR operator.

 - `template<typename t >`
`CImg< T > & operator|= (const CImg< t > &img)`

In-place bitwise OR operator.

 - `template<typename t >`
`CImg< T > operator| (const t value) const`

Bitwise OR operator.

 - `CImg< T > operator| (const char *const expression) const`

Bitwise OR operator.

 - `template<typename t >`
`CImg< T > operator| (const CImg< t > &img) const`

Bitwise OR operator.

 - `template<typename t >`
`CImg< T > & operator^= (const t value)`

In-place bitwise XOR operator.

 - `CImg< T > & operator^= (const char *const expression)`

In-place bitwise XOR operator.

 - `template<typename t >`
`CImg< T > & operator^= (const CImg< t > &img)`

In-place bitwise XOR operator.

 - `template<typename t >`
`CImg< T > operator^ (const t value) const`

Bitwise XOR operator.

 - `CImg< T > operator^ (const char *const expression) const`

Bitwise XOR operator.

 - `template<typename t >`
`CImg< T > operator^ (const CImg< t > &img) const`

- Bitwise XOR operator.*

 - template<typename t >
CImg< T > & operator<<= (const t value)

In-place bitwise left shift operator.
- CImg< T > & operator<<= (const char *const expression)

In-place bitwise left shift operator.
- template<typename t >
CImg< T > & operator<<= (const CImg< t > &img)

In-place bitwise left shift operator.
- template<typename t >
CImg< T > operator<< (const t value) const

Bitwise left shift operator.
- CImg< T > operator<< (const char *const expression) const

Bitwise left shift operator.
- template<typename t >
CImg< T > operator<< (const CImg< t > &img) const

Bitwise left shift operator.
- template<typename t >
CImg< T > & operator>>= (const t value)

In-place bitwise right shift operator.
- CImg< T > & operator>>= (const char *const expression)

In-place bitwise right shift operator.
- template<typename t >
CImg< T > & operator>>= (const CImg< t > &img)

In-place bitwise right shift operator.
- template<typename t >
CImg< T > operator>> (const t value) const

Bitwise right shift operator.
- CImg< T > operator>> (const char *const expression) const

Bitwise right shift operator.
- template<typename t >
CImg< T > operator>> (const CImg< t > &img) const

Bitwise right shift operator.
- CImg< T > operator~ () const

Bitwise inversion operator.
- template<typename t >
bool operator== (const t value) const

Test if all pixels of an image have the same value.
- bool operator== (const char *const expression) const

Test if all pixel values of an image follow a specified expression.
- template<typename t >
bool operator== (const CImg< t > &img) const

Test if two images have the same size and values.
- template<typename t >
bool operator!= (const t value) const

Test if pixels of an image are all different from a value.
- bool operator!= (const char *const expression) const

Test if all pixel values of an image are different from a specified expression.
- template<typename t >
bool operator!= (const CImg< t > &img) const

Test if two images have different sizes or values.
- template<typename t >
CImgList< typename cimg::superset< T, t >::type > operator, (const CImg< t > &img) const

Construct an image list from two images.

- `template<typename t >`
`CImgList< typename cimg::superset< T, t >::type > operator, (const CImgList< t > &list) const`

Construct an image list from image instance and an input image list.

- `CImgList< T > operator< (const char axis) const`

Split image along specified axis.

Instance Characteristics

- `int width () const`
Return the number of image columns.
- `int height () const`
Return the number of image rows.
- `int depth () const`
Return the number of image slices.
- `int spectrum () const`
Return the number of image channels.
- `ulongT size () const`
Return the total number of pixel values.
- `T * data ()`
Return a pointer to the first pixel value.
- `const T * data () const`
*Return a pointer to the first pixel value **[const version]**.*
- `T * data (const unsigned int x, const unsigned int y=0, const unsigned int z=0, const unsigned int c=0)`
Return a pointer to a located pixel value.
- `const T * data (const unsigned int x, const unsigned int y=0, const unsigned int z=0, const unsigned int c=0) const`
*Return a pointer to a located pixel value **[const version]**.*
- `longT offset (const int x, const int y=0, const int z=0, const int c=0) const`
Return the offset to a located pixel value, with respect to the beginning of the pixel buffer.
- `iterator begin ()`
Return a `CImg<T>::iterator` pointing to the first pixel value.
- `const_iterator begin () const`
*Return a `CImg<T>::iterator` pointing to the first value of the pixel buffer **[const version]**.*
- `iterator end ()`
Return a `CImg<T>::iterator` pointing next to the last pixel value.
- `const_iterator end () const`
*Return a `CImg<T>::iterator` pointing next to the last pixel value **[const version]**.*
- `T & front ()`
Return a reference to the first pixel value.
- `const T & front () const`
*Return a reference to the first pixel value **[const version]**.*
- `T & back ()`
Return a reference to the last pixel value.
- `const T & back () const`
*Return a reference to the last pixel value **[const version]**.*
- `T & at (const int offset, const T &out_value)`
Access to a pixel value at a specified offset, using Dirichlet boundary conditions.
- `T at (const int offset, const T &out_value) const`
*Access to a pixel value at a specified offset, using Dirichlet boundary conditions **[const version]**.*

- T & [at](#) (const int [offset](#))
Access to a pixel value at a specified offset, using Neumann boundary conditions.
- const T & [at](#) (const int [offset](#)) const
*Access to a pixel value at a specified offset, using Neumann boundary conditions **[const version]**.*
- T & [atX](#) (const int x, const int y, const int z, const int c, const T &out_value)
Access to a pixel value, using Dirichlet boundary conditions for the X-coordinate.
- T [atX](#) (const int x, const int y, const int z, const int c, const T &out_value) const
*Access to a pixel value, using Dirichlet boundary conditions for the X-coordinate **[const version]**.*
- T & [atX](#) (const int x, const int y=0, const int z=0, const int c=0)
Access to a pixel value, using Neumann boundary conditions for the X-coordinate.
- const T & [atX](#) (const int x, const int y=0, const int z=0, const int c=0) const
*Access to a pixel value, using Neumann boundary conditions for the X-coordinate **[const version]**.*
- T & [atXY](#) (const int x, const int y, const int z, const int c, const T &out_value)
Access to a pixel value, using Dirichlet boundary conditions for the X and Y-coordinates.
- T [atXY](#) (const int x, const int y, const int z, const int c, const T &out_value) const
*Access to a pixel value, using Dirichlet boundary conditions for the X and Y coordinates **[const version]**.*
- T & [atXY](#) (const int x, const int y, const int z=0, const int c=0)
Access to a pixel value, using Neumann boundary conditions for the X and Y-coordinates.
- const T & [atXY](#) (const int x, const int y, const int z=0, const int c=0) const
*Access to a pixel value, using Neumann boundary conditions for the X and Y-coordinates **[const version]**.*
- T & [atXYZ](#) (const int x, const int y, const int z, const int c, const T &out_value)
Access to a pixel value, using Dirichlet boundary conditions for the X,Y and Z-coordinates.
- T [atXYZ](#) (const int x, const int y, const int z, const int c, const T &out_value) const
*Access to a pixel value, using Dirichlet boundary conditions for the X,Y and Z-coordinates **[const version]**.*
- T & [atXYZ](#) (const int x, const int y, const int z, const int c=0)
Access to a pixel value, using Neumann boundary conditions for the X,Y and Z-coordinates.
- const T & [atXYZ](#) (const int x, const int y, const int z, const int c=0) const
*Access to a pixel value, using Neumann boundary conditions for the X,Y and Z-coordinates **[const version]**.*
- T & [atXYZC](#) (const int x, const int y, const int z, const int c, const T &out_value)
Access to a pixel value, using Dirichlet boundary conditions.
- T [atXYZC](#) (const int x, const int y, const int z, const int c, const T &out_value) const
*Access to a pixel value, using Dirichlet boundary conditions **[const version]**.*
- T & [atXYZC](#) (const int x, const int y, const int z, const int c)
Access to a pixel value, using Neumann boundary conditions.
- const T & [atXYZC](#) (const int x, const int y, const int z, const int c) const
*Access to a pixel value, using Neumann boundary conditions **[const version]**.*
- Tfloat [linear_atX](#) (const float fx, const int y, const int z, const int c, const T &out_value) const
Return pixel value, using linear interpolation and Dirichlet boundary conditions for the X-coordinate.
- Tfloat [linear_atX](#) (const float fx, const int y=0, const int z=0, const int c=0) const
Return pixel value, using linear interpolation and Neumann boundary conditions for the X-coordinate.
- Tfloat [linear_atXY](#) (const float fx, const float fy, const int z, const int c, const T &out_value) const
Return pixel value, using linear interpolation and Dirichlet boundary conditions for the X and Y-coordinates.
- Tfloat [linear_atXY](#) (const float fx, const float fy, const int z=0, const int c=0) const
Return pixel value, using linear interpolation and Neumann boundary conditions for the X and Y-coordinates.
- Tfloat [linear_atXYZ](#) (const float fx, const float fy, const float fz, const int c, const T &out_value) const
Return pixel value, using linear interpolation and Dirichlet boundary conditions for the X,Y and Z-coordinates.
- Tfloat [linear_atXYZ](#) (const float fx, const float fy=0, const float fz=0, const int c=0) const
Return pixel value, using linear interpolation and Neumann boundary conditions for the X,Y and Z-coordinates.
- Tfloat [linear_atXYZC](#) (const float fx, const float fy, const float fz, const float fc, const T &out_value) const
Return pixel value, using linear interpolation and Dirichlet boundary conditions for all X,Y,Z,C-coordinates.
- Tfloat [linear_atXYZC](#) (const float fx, const float fy=0, const float fz=0, const float fc=0) const

- Return pixel value, using linear interpolation and Neumann boundary conditions for all X,Y,Z and C-coordinates.*
- Tfloat [cubic_atX](#) (const float fx, const int y, const int z, const int c, const T &out_value) const
- Return pixel value, using cubic interpolation and Dirichlet boundary conditions for the X-coordinate.*
- Tfloat [cubic_atX](#) (const float fx, const int y, const int z, const int c, const T &out_value, const Tfloat min_value, const Tfloat max_value) const
- Return damped pixel value, using cubic interpolation and Dirichlet boundary conditions for the X-coordinate.*
- Tfloat [cubic_atX](#) (const float fx, const int y=0, const int z=0, const int c=0) const
- Return pixel value, using cubic interpolation and Neumann boundary conditions for the X-coordinate.*
- Tfloat [cubic_atX](#) (const float fx, const int y, const int z, const int c, const Tfloat min_value, const Tfloat max_value) const
- Return damped pixel value, using cubic interpolation and Neumann boundary conditions for the X-coordinate.*
- Tfloat [cubic_atXY](#) (const float fx, const float fy, const int z, const int c, const T &out_value) const
- Return pixel value, using cubic interpolation and Dirichlet boundary conditions for the X and Y-coordinates.*
- Tfloat [cubic_atXY](#) (const float fx, const float fy, const int z, const int c, const T &out_value, const Tfloat min_value, const Tfloat max_value) const
- Return damped pixel value, using cubic interpolation and Dirichlet boundary conditions for the X,Y-coordinates.*
- Tfloat [cubic_atXY](#) (const float fx, const float fy, const int z=0, const int c=0) const
- Return pixel value, using cubic interpolation and Neumann boundary conditions for the X and Y-coordinates.*
- Tfloat [cubic_atXY](#) (const float fx, const float fy, const int z, const int c, const Tfloat min_value, const Tfloat max_value) const
- Return damped pixel value, using cubic interpolation and Neumann boundary conditions for the X,Y-coordinates.*
- Tfloat [cubic_atXYZ](#) (const float fx, const float fy, const float fz, const int c, const T &out_value) const
- Return pixel value, using cubic interpolation and Dirichlet boundary conditions for the X,Y and Z-coordinates.*
- Tfloat [cubic_atXYZ](#) (const float fx, const float fy, const float fz, const int c, const T &out_value, const Tfloat min_value, const Tfloat max_value) const
- Return damped pixel value, using cubic interpolation and Dirichlet boundary conditions for the XYZ-coordinates.*
- Tfloat [cubic_atXYZ](#) (const float fx, const float fy, const float fz, const int c=0) const
- Return pixel value, using cubic interpolation and Neumann boundary conditions for the X,Y and Z-coordinates.*
- Tfloat [cubic_atXYZ](#) (const float fx, const float fy, const float fz, const int c, const Tfloat min_value, const Tfloat max_value) const
- Return damped pixel value, using cubic interpolation and Neumann boundary conditions for the XYZ-coordinates.*
- CImg< T > & [set_linear_atX](#) (const T &value, const float fx, const int y=0, const int z=0, const int c=0, const bool is_added=false)
- Set pixel value, using linear interpolation for the X-coordinates.*
- CImg< T > & [set_linear_atXY](#) (const T &value, const float fx, const float fy=0, const int z=0, const int c=0, const bool is_added=false)
- Set pixel value, using linear interpolation for the X and Y-coordinates.*
- CImg< T > & [set_linear_atXYZ](#) (const T &value, const float fx, const float fy=0, const float fz=0, const int c=0, const bool is_added=false)
- Set pixel value, using linear interpolation for the X,Y and Z-coordinates.*
- CImg< charT > [value_string](#) (const char separator=',', const unsigned int max_size=0, const char *const format=0) const
- Return a C-string containing a list of all values of the image instance.*
- static const char * [pixel_type](#) ()
- Return the type of image pixel values as a C string.*

Instance Checking

- bool [is_shared](#) () const
- Test shared state of the pixel buffer.*
- bool [is_empty](#) () const

- Test if image instance is empty.*

 - bool `is_inf` () const

Test if image instance contains a 'inf' value.
- bool `is_nan` () const

Test if image instance contains a NaN value.
- bool `is_sameX` (const unsigned int size_x) const

Test if image width is equal to specified value.
- template<typename t >
bool `is_sameX` (const CImg< t > &img) const

Test if image width is equal to specified value.
- bool `is_sameX` (const CImgDisplay &disp) const

Test if image width is equal to specified value.
- bool `is_sameY` (const unsigned int size_y) const

Test if image height is equal to specified value.
- template<typename t >
bool `is_sameY` (const CImg< t > &img) const

Test if image height is equal to specified value.
- bool `is_sameY` (const CImgDisplay &disp) const

Test if image height is equal to specified value.
- bool `is_sameZ` (const unsigned int size_z) const

Test if image depth is equal to specified value.
- template<typename t >
bool `is_sameZ` (const CImg< t > &img) const

Test if image depth is equal to specified value.
- bool `is_sameC` (const unsigned int size_c) const

Test if image spectrum is equal to specified value.
- template<typename t >
bool `is_sameC` (const CImg< t > &img) const

Test if image spectrum is equal to specified value.
- bool `is_sameXY` (const unsigned int size_x, const unsigned int size_y) const

Test if image width and height are equal to specified values.
- template<typename t >
bool `is_sameXY` (const CImg< t > &img) const

Test if image width and height are the same as that of another image.
- bool `is_sameXY` (const CImgDisplay &disp) const

Test if image width and height are the same as that of an existing display window.
- bool `is_sameXZ` (const unsigned int size_x, const unsigned int size_z) const

Test if image width and depth are equal to specified values.
- template<typename t >
bool `is_sameXZ` (const CImg< t > &img) const

Test if image width and depth are the same as that of another image.
- bool `is_sameXC` (const unsigned int size_x, const unsigned int size_c) const

Test if image width and spectrum are equal to specified values.
- template<typename t >
bool `is_sameXC` (const CImg< t > &img) const

Test if image width and spectrum are the same as that of another image.
- bool `is_sameYZ` (const unsigned int size_y, const unsigned int size_z) const

Test if image height and depth are equal to specified values.
- template<typename t >
bool `is_sameYZ` (const CImg< t > &img) const

Test if image height and depth are the same as that of another image.
- bool `is_sameYC` (const unsigned int size_y, const unsigned int size_c) const

- Test if image height and spectrum are equal to specified values.*

 - `template<typename t >`
`bool is_sameYC (const CImg< t > &img) const`
Test if image height and spectrum are the same as that of another image.
 - `bool is_sameZC (const unsigned int size_z, const unsigned int size_c) const`
Test if image depth and spectrum are equal to specified values.
 - `template<typename t >`
`bool is_sameZC (const CImg< t > &img) const`
Test if image depth and spectrum are the same as that of another image.
 - `bool is_sameXYZ (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z) const`
Test if image width, height and depth are equal to specified values.
 - `template<typename t >`
`bool is_sameXYZ (const CImg< t > &img) const`
Test if image width, height and depth are the same as that of another image.
 - `bool is_sameXYC (const unsigned int size_x, const unsigned int size_y, const unsigned int size_c) const`
Test if image width, height and spectrum are equal to specified values.
 - `template<typename t >`
`bool is_sameXYC (const CImg< t > &img) const`
Test if image width, height and spectrum are the same as that of another image.
 - `bool is_sameXZC (const unsigned int size_x, const unsigned int size_z, const unsigned int size_c) const`
Test if image width, depth and spectrum are equal to specified values.
 - `template<typename t >`
`bool is_sameXZC (const CImg< t > &img) const`
Test if image width, depth and spectrum are the same as that of another image.
 - `bool is_sameYZC (const unsigned int size_y, const unsigned int size_z, const unsigned int size_c) const`
Test if image height, depth and spectrum are equal to specified values.
 - `template<typename t >`
`bool is_sameYZC (const CImg< t > &img) const`
Test if image height, depth and spectrum are the same as that of another image.
 - `bool is_sameXYZC (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c) const`
Test if image width, height, depth and spectrum are equal to specified values.
 - `template<typename t >`
`bool is_sameXYZC (const CImg< t > &img) const`
Test if image width, height, depth and spectrum are the same as that of another image.
 - `bool containsXYZC (const int x, const int y=0, const int z=0, const int c=0) const`
Test if specified coordinates are inside image bounds.
 - `template<typename t >`
`bool contains (const T &pixel, t &x, t &y, t &z, t &c) const`
Test if pixel value is inside image bounds and get its X,Y,Z and C-coordinates.
 - `template<typename t >`
`bool contains (const T &pixel, t &x, t &y, t &z) const`
Test if pixel value is inside image bounds and get its X,Y and Z-coordinates.
 - `template<typename t >`
`bool contains (const T &pixel, t &x, t &y) const`
Test if pixel value is inside image bounds and get its X and Y-coordinates.
 - `template<typename t >`
`bool contains (const T &pixel, t &x) const`
Test if pixel value is inside image bounds and get its X-coordinate.
 - `bool contains (const T &pixel) const`
Test if pixel value is inside image bounds.
 - `template<typename t >`
`bool is_overlapped (const CImg< t > &img) const`

Test if pixel buffers of instance and input images overlap.

- `template<typename tp , typename tc , typename to >`
`bool is_object3d (const ClmgList< tp > &primitives, const ClmgList< tc > &colors, const to &opacities, const`
`bool full_check=true, char *const error_message=0) const`

*Test if the set {*this,primitives,colors,opacities} defines a valid 3d object.*

- `bool is_Clmg3d (const bool full_check=true, char *const error_message=0) const`

Test if image instance represents a valid serialization of a 3d object.

Mathematical Functions

- `Clmg< T > & sqr ()`
Compute the square value of each pixel value.
- `Clmg< Tfloat > get_sqr () const`
Compute the square value of each pixel value [new-instance version].
- `Clmg< T > & sqrt ()`
Compute the square root of each pixel value.
- `Clmg< Tfloat > get_sqrt () const`
Compute the square root of each pixel value [new-instance version].
- `Clmg< T > & exp ()`
Compute the exponential of each pixel value.
- `Clmg< Tfloat > get_exp () const`
Compute the exponential of each pixel value [new-instance version].
- `Clmg< T > & log ()`
Compute the logarithm of each pixel value.
- `Clmg< Tfloat > get_log () const`
Compute the logarithm of each pixel value [new-instance version].
- `Clmg< T > & log2 ()`
Compute the base-2 logarithm of each pixel value.
- `Clmg< Tfloat > get_log2 () const`
Compute the base-10 logarithm of each pixel value [new-instance version].
- `Clmg< T > & log10 ()`
Compute the base-10 logarithm of each pixel value.
- `Clmg< Tfloat > get_log10 () const`
Compute the base-10 logarithm of each pixel value [new-instance version].
- `Clmg< T > & abs ()`
Compute the absolute value of each pixel value.
- `Clmg< Tfloat > get_abs () const`
Compute the absolute value of each pixel value [new-instance version].
- `Clmg< T > & sign ()`
Compute the sign of each pixel value.
- `Clmg< Tfloat > get_sign () const`
Compute the sign of each pixel value [new-instance version].
- `Clmg< T > & cos ()`
Compute the cosine of each pixel value.
- `Clmg< Tfloat > get_cos () const`
Compute the cosine of each pixel value [new-instance version].
- `Clmg< T > & sin ()`
Compute the sine of each pixel value.
- `Clmg< Tfloat > get_sin () const`
Compute the sine of each pixel value [new-instance version].

- `CImg< T > & sinc ()`
Compute the sinc of each pixel value.
- `CImg< Tfloat > get_sinc () const`
Compute the sinc of each pixel value **[new-instance version]**.
- `CImg< T > & tan ()`
Compute the tangent of each pixel value.
- `CImg< Tfloat > get_tan () const`
Compute the tangent of each pixel value **[new-instance version]**.
- `CImg< T > & cosh ()`
Compute the hyperbolic cosine of each pixel value.
- `CImg< Tfloat > get_cosh () const`
Compute the hyperbolic cosine of each pixel value **[new-instance version]**.
- `CImg< T > & sinh ()`
Compute the hyperbolic sine of each pixel value.
- `CImg< Tfloat > get_sinh () const`
Compute the hyperbolic sine of each pixel value **[new-instance version]**.
- `CImg< T > & tanh ()`
Compute the hyperbolic tangent of each pixel value.
- `CImg< Tfloat > get_tanh () const`
Compute the hyperbolic tangent of each pixel value **[new-instance version]**.
- `CImg< T > & acos ()`
Compute the arccosine of each pixel value.
- `CImg< Tfloat > get_acos () const`
Compute the arccosine of each pixel value **[new-instance version]**.
- `CImg< T > & asin ()`
Compute the arcsine of each pixel value.
- `CImg< Tfloat > get_asin () const`
Compute the arcsine of each pixel value **[new-instance version]**.
- `CImg< T > & atan ()`
Compute the arctangent of each pixel value.
- `CImg< Tfloat > get_atan () const`
Compute the arctangent of each pixel value **[new-instance version]**.
- `template<typename t >`
`CImg< T > & atan2 (const CImg< t > &img)`
Compute the arctangent2 of each pixel value.
- `template<typename t >`
`CImg< Tfloat > get_atan2 (const CImg< t > &img) const`
Compute the arctangent2 of each pixel value **[new-instance version]**.
- `template<typename t >`
`CImg< T > & mul (const CImg< t > &img)`
In-place pointwise multiplication.
- `template<typename t >`
`CImg< typename cimng::superset< T, t >::type > get_mul (const CImg< t > &img) const`
In-place pointwise multiplication **[new-instance version]**.
- `template<typename t >`
`CImg< T > & div (const CImg< t > &img)`
In-place pointwise division.
- `template<typename t >`
`CImg< typename cimng::superset< T, t >::type > get_div (const CImg< t > &img) const`
In-place pointwise division **[new-instance version]**.
- `CImg< T > & pow (const double p)`

- Raise each pixel value to a specified power.*

 - `Clmg< Tfloat > get_pow` (const double p) const

Raise each pixel value to a specified power [new-instance version].

 - `Clmg< T > & pow` (const char *const expression)

Raise each pixel value to a power, specified from an expression.

 - `Clmg< Tfloat > get_pow` (const char *const expression) const

Raise each pixel value to a power, specified from an expression [new-instance version].

 - `template<typename t >`
`Clmg< T > & pow` (const `Clmg< t > &img`)

Raise each pixel value to a power, pointwisely specified from another image.

 - `template<typename t >`
`Clmg< Tfloat > get_pow` (const `Clmg< t > &img`) const

Raise each pixel value to a power, pointwisely specified from another image [new-instance version].

 - `Clmg< T > & rol` (const unsigned int n=1)

Compute the bitwise left rotation of each pixel value.

 - `Clmg< T > get_rol` (const unsigned int n=1) const

Compute the bitwise left rotation of each pixel value [new-instance version].

 - `Clmg< T > & rol` (const char *const expression)

Compute the bitwise left rotation of each pixel value.

 - `Clmg< T > get_rol` (const char *const expression) const

Compute the bitwise left rotation of each pixel value [new-instance version].

 - `template<typename t >`
`Clmg< T > & rol` (const `Clmg< t > &img`)

Compute the bitwise left rotation of each pixel value.

 - `template<typename t >`
`Clmg< T > get_rol` (const `Clmg< t > &img`) const

Compute the bitwise left rotation of each pixel value [new-instance version].

 - `Clmg< T > & ror` (const unsigned int n=1)

Compute the bitwise right rotation of each pixel value.

 - `Clmg< T > get_ror` (const unsigned int n=1) const

Compute the bitwise right rotation of each pixel value [new-instance version].

 - `Clmg< T > & ror` (const char *const expression)

Compute the bitwise right rotation of each pixel value.

 - `Clmg< T > get_ror` (const char *const expression) const

Compute the bitwise right rotation of each pixel value [new-instance version].

 - `template<typename t >`
`Clmg< T > & ror` (const `Clmg< t > &img`)

Compute the bitwise right rotation of each pixel value.

 - `template<typename t >`
`Clmg< T > get_ror` (const `Clmg< t > &img`) const

Compute the bitwise right rotation of each pixel value [new-instance version].

 - `Clmg< T > & min` (const T &val)

Pointwise min operator between instance image and a value.

 - `Clmg< T > get_min` (const T &val) const

Pointwise min operator between instance image and a value [new-instance version].

 - `template<typename t >`
`Clmg< T > & min` (const `Clmg< t > &img`)

Pointwise min operator between two images.

 - `template<typename t >`
`Clmg< typename cimg::superset< T, t >::type > get_min` (const `Clmg< t > &img`) const

Pointwise min operator between two images [new-instance version].

 - `Clmg< T > & min` (const char *const expression)

- Pointwise min operator between an image and an expression.*

 - `CImg< Tfloat > get_min` (const char *const expression) const

Pointwise min operator between an image and an expression [new-instance version].

 - `CImg< T > & max` (const T &val)

Pointwise max operator between instance image and a value.

 - `CImg< T > get_max` (const T &val) const

Pointwise max operator between instance image and a value [new-instance version].

 - template<typename t >
`CImg< T > & max` (const `CImg< t > &img`)

Pointwise max operator between two images.

 - template<typename t >
`CImg< typename cimg::superset< T, t >::type > get_max` (const `CImg< t > &img`) const

Pointwise max operator between two images [new-instance version].

 - `CImg< T > & max` (const char *const expression)

Pointwise max operator between an image and an expression.

 - `CImg< Tfloat > get_max` (const char *const expression) const

Pointwise max operator between an image and an expression [new-instance version].

 - `T & min` ()

Return a reference to the minimum pixel value.

 - const T & `min` () const

Return a reference to the minimum pixel value [const version].

 - `T & max` ()

Return a reference to the maximum pixel value.

 - const T & `max` () const

Return a reference to the maximum pixel value [const version].

 - template<typename t >
`T & min_max` (t &max_val)

Return a reference to the minimum pixel value as well as the maximum pixel value.

 - template<typename t >
const T & `min_max` (t &max_val) const

Return a reference to the minimum pixel value as well as the maximum pixel value [const version].

 - template<typename t >
`T & max_min` (t &min_val)

Return a reference to the maximum pixel value as well as the minimum pixel value.

 - template<typename t >
const T & `max_min` (t &min_val) const

Return a reference to the maximum pixel value as well as the minimum pixel value [const version].

 - `T kth_smallest` (const unsigned int k) const

Return the kth smallest pixel value.

 - `T median` () const

Return the median pixel value.

 - double `product` () const

Return the product of all the pixel values.

 - double `sum` () const

Return the sum of all the pixel values.

 - double `mean` () const

Return the average pixel value.

 - double `variance` (const unsigned int variance_method=1) const

Return the variance of the pixel values.

 - template<typename t >
double `variance_mean` (const unsigned int variance_method, t &mean) const

Return the variance as well as the average of the pixel values.

- double [variance_noise](#) (const unsigned int variance_method=2) const
Return estimated variance of the noise.
- template<typename t >
double [MSE](#) (const [Clmg](#)< t > &img) const
Compute the MSE (Mean-Squared Error) between two images.
- template<typename t >
double [PSNR](#) (const [Clmg](#)< t > &img, const double max_value=255) const
Compute the PSNR (Peak Signal-to-Noise Ratio) between two images.
- double [eval](#) (const char *const expression, const double x=0, const double y=0, const double z=0, const double c=0, const [ClmgList](#)< T > *const list_inputs=0, [ClmgList](#)< T > *const list_outputs=0)
Evaluate math formula.
- double [eval](#) (const char *const expression, const double x=0, const double y=0, const double z=0, const double c=0, const [ClmgList](#)< T > *const list_inputs=0, [ClmgList](#)< T > *const list_outputs=0) const
Evaluate math formula [const version].
- template<typename t >
void [eval](#) ([Clmg](#)< t > &output, const char *const expression, const double x=0, const double y=0, const double z=0, const double c=0, const [ClmgList](#)< T > *const list_inputs=0, [ClmgList](#)< T > *const list_outputs=0)
Evaluate math formula.
- template<typename t >
void [eval](#) ([Clmg](#)< t > &output, const char *const expression, const double x=0, const double y=0, const double z=0, const double c=0, const [ClmgList](#)< T > *const list_inputs=0, [ClmgList](#)< T > *const list_outputs=0) const
Evaluate math formula [const version].
- template<typename t >
[Clmg](#)< doubleT > [eval](#) (const char *const expression, const [Clmg](#)< t > &xyzc, const [ClmgList](#)< T > *const list_inputs=0, [ClmgList](#)< T > *const list_outputs=0)
Evaluate math formula on a set of variables.
- template<typename t >
[Clmg](#)< doubleT > [eval](#) (const char *const expression, const [Clmg](#)< t > &xyzc, const [ClmgList](#)< T > *const list_inputs=0, [ClmgList](#)< T > *const list_outputs=0) const
Evaluate math formula on a set of variables [const version].
- [Clmg](#)< Tdouble > [get_stats](#) (const unsigned int variance_method=1) const
Compute statistics vector from the pixel values.
- [Clmg](#)< T > & [stats](#) (const unsigned int variance_method=1)
Compute statistics vector from the pixel values [in-place version].

Vector / Matrix Operations

- double [magnitude](#) (const int magnitude_type=2) const
Compute norm of the image, viewed as a matrix.
- double [trace](#) () const
Compute the trace of the image, viewed as a matrix.
- double [det](#) () const
Compute the determinant of the image, viewed as a matrix.
- template<typename t >
double [dot](#) (const [Clmg](#)< t > &img) const
Compute the dot product between instance and argument, viewed as matrices.
- [Clmg](#)< T > [get_vector_at](#) (const unsigned int x, const unsigned int y=0, const unsigned int z=0) const
Get vector-valued pixel located at specified position.
- [Clmg](#)< T > [get_matrix_at](#) (const unsigned int x=0, const unsigned int y=0, const unsigned int z=0) const
Get (square) matrix-valued pixel located at specified position.

- `Clmg< T > get_tensor_at` (const unsigned int x, const unsigned int y=0, const unsigned int z=0) const
Get tensor-valued pixel located at specified position.
- `template<typename t >`
`Clmg< T > & set_vector_at` (const `Clmg< t >` &vec, const unsigned int x, const unsigned int y=0, const unsigned int z=0)
Set vector-valued pixel at specified position.
- `template<typename t >`
`Clmg< T > & set_matrix_at` (const `Clmg< t >` &mat, const unsigned int x=0, const unsigned int y=0, const unsigned int z=0)
Set (square) matrix-valued pixel at specified position.
- `template<typename t >`
`Clmg< T > & set_tensor_at` (const `Clmg< t >` &ten, const unsigned int x=0, const unsigned int y=0, const unsigned int z=0)
Set tensor-valued pixel at specified position.
- `Clmg< T > & vector` ()
Unroll pixel values along axis y.
- `Clmg< T > get_vector` () const
Unroll pixel values along axis y [new-instance version].
- `Clmg< T > & matrix` ()
Resize image to become a scalar square matrix.
- `Clmg< T > get_matrix` () const
Resize image to become a scalar square matrix [new-instance version].
- `Clmg< T > & tensor` ()
Resize image to become a symmetric tensor.
- `Clmg< T > get_tensor` () const
Resize image to become a symmetric tensor [new-instance version].
- `Clmg< T > & diagonal` ()
Resize image to become a diagonal matrix.
- `Clmg< T > get_diagonal` () const
Resize image to become a diagonal matrix [new-instance version].
- `Clmg< T > & identity_matrix` ()
Replace the image by an identity matrix.
- `Clmg< T > get_identity_matrix` () const
Replace the image by an identity matrix [new-instance version].
- `Clmg< T > & sequence` (const T &a0, const T &a1)
Fill image with a linear sequence of values.
- `Clmg< T > get_sequence` (const T &a0, const T &a1) const
Fill image with a linear sequence of values [new-instance version].
- `Clmg< T > & transpose` ()
Transpose the image, viewed as a matrix.
- `Clmg< T > get_transpose` () const
Transpose the image, viewed as a matrix [new-instance version].
- `template<typename t >`
`Clmg< T > & cross` (const `Clmg< t >` &img)
Compute the cross product between two 1×3 images, viewed as 3d vectors.
- `template<typename t >`
`Clmg< typename cimg::superset< T, t >::type > get_cross` (const `Clmg< t >` &img) const
Compute the cross product between two 1×3 images, viewed as 3d vectors [new-instance version].
- `Clmg< T > & invert` (const bool use_LU=true)
Invert the instance image, viewed as a matrix.
- `Clmg< Tfloat > get_invert` (const bool use_LU=true) const
Invert the instance image, viewed as a matrix [new-instance version].

- `CImg< T > & pseudoinvert ()`
Compute the Moore-Penrose pseudo-inverse of the instance image, viewed as a matrix.
- `CImg< Tfloat > get_pseudoinvert () const`
*Compute the Moore-Penrose pseudo-inverse of the instance image, viewed as a matrix **[new-instance version]**.*
- `template<typename t >`
`CImg< T > & solve (const CImg< t > &A)`
Solve a system of linear equations.
- `template<typename t >`
`CImg< typename cimg::superset2< T, t, float >::type > get_solve (const CImg< t > &A) const`
*Solve a system of linear equations **[new-instance version]**.*
- `template<typename t >`
`CImg< T > & solve_tridiagonal (const CImg< t > &A)`
Solve a tridiagonal system of linear equations.
- `template<typename t >`
`CImg< typename cimg::superset2< T, t, float >::type > get_solve_tridiagonal (const CImg< t > &A) const`
*Solve a tridiagonal system of linear equations **[new-instance version]**.*
- `template<typename t >`
`const CImg< T > & eigen (CImg< t > &val, CImg< t > &vec) const`
Compute eigenvalues and eigenvectors of the instance image, viewed as a matrix.
- `CImgList< Tfloat > get_eigen () const`
Compute eigenvalues and eigenvectors of the instance image, viewed as a matrix.
- `template<typename t >`
`const CImg< T > & symmetric_eigen (CImg< t > &val, CImg< t > &vec) const`
Compute eigenvalues and eigenvectors of the instance image, viewed as a symmetric matrix.
- `CImgList< Tfloat > get_symmetric_eigen () const`
Compute eigenvalues and eigenvectors of the instance image, viewed as a symmetric matrix.
- `template<typename t >`
`CImg< T > & sort (CImg< t > &permutations, const bool is_increasing=true)`
Sort pixel values and get sorting permutations.
- `template<typename t >`
`CImg< T > get_sort (CImg< t > &permutations, const bool is_increasing=true) const`
*Sort pixel values and get sorting permutations **[new-instance version]**.*
- `CImg< T > & sort (const bool is_increasing=true, const char axis=0)`
Sort pixel values.
- `CImg< T > get_sort (const bool is_increasing=true, const char axis=0) const`
*Sort pixel values **[new-instance version]**.*
- `template<typename t >`
`const CImg< T > & SVD (CImg< t > &U, CImg< t > &S, CImg< t > &V, const bool sorting=true, const unsigned int max_iteration=40, const float lambda=0) const`
Compute the SVD of the instance image, viewed as a general matrix.
- `CImgList< Tfloat > get_SVD (const bool sorting=true, const unsigned int max_iteration=40, const float lambda=0) const`
Compute the SVD of the instance image, viewed as a general matrix.
- `template<typename t >`
`CImg< T > & dijkstra (const unsigned int starting_node, const unsigned int ending_node, CImg< t > &previous_node)`
Return minimal path in a graph, using the Dijkstra algorithm.
- `template<typename t >`
`CImg< T > get_dijkstra (const unsigned int starting_node, const unsigned int ending_node, CImg< t > &previous_node) const`
*Return minimal path in a graph, using the Dijkstra algorithm **[new-instance version]**.*
- `CImg< T > & dijkstra (const unsigned int starting_node, const unsigned int ending_node=~0U)`
Return minimal path in a graph, using the Dijkstra algorithm.

- `CImg< Tfloat > get_dijkstra` (const unsigned int starting_node, const unsigned int ending_node=~0U) const

Return minimal path in a graph, using the Dijkstra algorithm [new-instance version].

- template<typename tf, typename t >
static `CImg< T > dijkstra` (const tf &distance, const unsigned int nb_nodes, const unsigned int starting_node, const unsigned int ending_node, `CImg< t > &previous_node`)

Compute minimal path in a graph, using the Dijkstra algorithm.

- template<typename tf, typename t >
static `CImg< T > dijkstra` (const tf &distance, const unsigned int nb_nodes, const unsigned int starting_node, const unsigned int ending_node=~0U)

Return minimal path in a graph, using the Dijkstra algorithm.

- static `CImg< T > string` (const char *const str, const bool is_last_zero=true, const bool is_shared=false)

Return an image containing the ascii codes of the specified string.

- static `CImg< T > vector` (const T &a0)

Return a 1x1 image containing specified value.

- static `CImg< T > vector` (const T &a0, const T &a1)

Return a 1x2 image containing specified values.

- static `CImg< T > vector` (const T &a0, const T &a1, const T &a2)

Return a 1x3 image containing specified values.

- static `CImg< T > vector` (const T &a0, const T &a1, const T &a2, const T &a3)

Return a 1x4 image containing specified values.

- static `CImg< T > vector` (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4)

Return a 1x5 image containing specified values.

- static `CImg< T > vector` (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5)

Return a 1x6 image containing specified values.

- static `CImg< T > vector` (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6)

Return a 1x7 image containing specified values.

- static `CImg< T > vector` (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7)

Return a 1x8 image containing specified values.

- static `CImg< T > vector` (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8)

Return a 1x9 image containing specified values.

- static `CImg< T > vector` (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8, const T &a9)

Return a 1x10 image containing specified values.

- static `CImg< T > vector` (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8, const T &a9, const T &a10)

Return a 1x11 image containing specified values.

- static `CImg< T > vector` (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8, const T &a9, const T &a10, const T &a11)

Return a 1x12 image containing specified values.

- static `CImg< T > vector` (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8, const T &a9, const T &a10, const T &a11, const T &a12)

Return a 1x13 image containing specified values.

- static `CImg< T > vector` (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8, const T &a9, const T &a10, const T &a11, const T &a12, const T &a13)

Return a 1x14 image containing specified values.

- static `CImg< T > vector` (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8, const T &a9, const T &a10, const T &a11, const T &a12, const T &a13, const T &a14)

Return a 1x15 image containing specified values.

- static `CImg< T > vector` (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8, const T &a9, const T &a10, const T &a11, const T &a12, const T &a13, const T &a14, const T &a15)

Return a 1x16 image containing specified values.

- static `CImg< T > matrix` (const T &a0)

Return a 1x1 matrix containing specified coefficients.

- static `CImg< T > matrix` (const T &a0, const T &a1, const T &a2, const T &a3)

Return a 2x2 matrix containing specified coefficients.

- static `CImg< T > matrix` (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8)

Return a 3x3 matrix containing specified coefficients.

- static `CImg< T > matrix` (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8, const T &a9, const T &a10, const T &a11, const T &a12, const T &a13, const T &a14, const T &a15)

Return a 4x4 matrix containing specified coefficients.

- static `CImg< T > matrix` (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5, const T &a6, const T &a7, const T &a8, const T &a9, const T &a10, const T &a11, const T &a12, const T &a13, const T &a14, const T &a15, const T &a16, const T &a17, const T &a18, const T &a19, const T &a20, const T &a21, const T &a22, const T &a23, const T &a24)

Return a 5x5 matrix containing specified coefficients.

- static `CImg< T > tensor` (const T &a0)

Return a 1x1 symmetric matrix containing specified coefficients.

- static `CImg< T > tensor` (const T &a0, const T &a1, const T &a2)

Return a 2x2 symmetric matrix tensor containing specified coefficients.

- static `CImg< T > tensor` (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4, const T &a5)

Return a 3x3 symmetric matrix containing specified coefficients.

- static `CImg< T > diagonal` (const T &a0)

Return a 1x1 diagonal matrix containing specified coefficients.

- static `CImg< T > diagonal` (const T &a0, const T &a1)

Return a 2x2 diagonal matrix containing specified coefficients.

- static `CImg< T > diagonal` (const T &a0, const T &a1, const T &a2)

Return a 3x3 diagonal matrix containing specified coefficients.

- static `CImg< T > diagonal` (const T &a0, const T &a1, const T &a2, const T &a3)

Return a 4x4 diagonal matrix containing specified coefficients.

- static `CImg< T > diagonal` (const T &a0, const T &a1, const T &a2, const T &a3, const T &a4)

Return a 5x5 diagonal matrix containing specified coefficients.

- static `CImg< T > identity_matrix` (const unsigned int N)

Return a NxN identity matrix.

- static `CImg< T > sequence` (const unsigned int N, const T &a0, const T &a1)

Return a N-numbered sequence vector from a0 to a1.

- static `CImg< T > rotation_matrix` (const float x, const float y, const float z, const float w, const bool is_↵ quaternion=false)

Return a 3x3 rotation matrix along the (x,y,z)-axis with an angle w.

Value Manipulation

- `CImg< T > &fill` (const T &val)

Fill all pixel values with specified value.

- `CImg< T > get_fill` (const T &val) const

Fill all pixel values with specified value [new-instance version].

- `Clmg< T > & fill` (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9, const T &val10, const T &val11)
Fill sequentially all pixel values with specified values [overloading].
- `Clmg< T > get_fill` (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9, const T &val10, const T &val11) const
Fill sequentially all pixel values with specified values [new-instance version].
- `Clmg< T > & fill` (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9, const T &val10, const T &val11, const T &val12)
Fill sequentially all pixel values with specified values [overloading].
- `Clmg< T > get_fill` (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9, const T &val10, const T &val11, const T &val12) const
Fill sequentially all pixel values with specified values [new-instance version].
- `Clmg< T > & fill` (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9, const T &val10, const T &val11, const T &val12, const T &val13)
Fill sequentially all pixel values with specified values [overloading].
- `Clmg< T > get_fill` (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9, const T &val10, const T &val11, const T &val12, const T &val13) const
Fill sequentially all pixel values with specified values [new-instance version].
- `Clmg< T > & fill` (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9, const T &val10, const T &val11, const T &val12, const T &val13, const T &val14)
Fill sequentially all pixel values with specified values [overloading].
- `Clmg< T > get_fill` (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9, const T &val10, const T &val11, const T &val12, const T &val13, const T &val14) const
Fill sequentially all pixel values with specified values [new-instance version].
- `Clmg< T > & fill` (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9, const T &val10, const T &val11, const T &val12, const T &val13, const T &val14, const T &val15)
Fill sequentially all pixel values with specified values [overloading].
- `Clmg< T > get_fill` (const T &val0, const T &val1, const T &val2, const T &val3, const T &val4, const T &val5, const T &val6, const T &val7, const T &val8, const T &val9, const T &val10, const T &val11, const T &val12, const T &val13, const T &val14, const T &val15) const
Fill sequentially all pixel values with specified values [new-instance version].
- `Clmg< T > & fill` (const char *const expression, const bool repeat_values, const bool allow_formula=true, const `ClmgList< T >` *const list_inputs=0, `ClmgList< T >` *const list_outputs=0)
Fill sequentially pixel values according to a given expression.
- `Clmg< T > get_fill` (const char *const expression, const bool repeat_values, const bool allow_formula=true, const `ClmgList< T >` *const list_inputs=0, `ClmgList< T >` *const list_outputs=0) const
Fill sequentially pixel values according to a given expression [new-instance version].
- `template<typename t >`
`Clmg< T > & fill` (const `Clmg< t >` &values, const bool repeat_values=true)
Fill sequentially pixel values according to the values found in another image.
- `template<typename t >`
`Clmg< T > get_fill` (const `Clmg< t >` &values, const bool repeat_values=true) const
Fill sequentially pixel values according to the values found in another image [new-instance version].
- `Clmg< T > & fillX` (const unsigned int y, const unsigned int z, const unsigned int c, const int a0,...)
Fill pixel values along the X-axis at a specified pixel position.
- `Clmg< T > & fillX` (const unsigned int y, const unsigned int z, const unsigned int c, const double a0,...)
Fill pixel values along the X-axis at a specified pixel position [overloading].

- `CImg< T > & fillY` (const unsigned int x, const unsigned int z, const unsigned int c, const int a0,...)
Fill pixel values along the Y-axis at a specified pixel position.
- `CImg< T > & fillY` (const unsigned int x, const unsigned int z, const unsigned int c, const double a0,...)
*Fill pixel values along the Y-axis at a specified pixel position **[overloading]**.*
- `CImg< T > & fillZ` (const unsigned int x, const unsigned int y, const unsigned int c, const int a0,...)
Fill pixel values along the Z-axis at a specified pixel position.
- `CImg< T > & fillZ` (const unsigned int x, const unsigned int y, const unsigned int c, const double a0,...)
*Fill pixel values along the Z-axis at a specified pixel position **[overloading]**.*
- `CImg< T > & fillC` (const unsigned int x, const unsigned int y, const unsigned int z, const int a0,...)
Fill pixel values along the C-axis at a specified pixel position.
- `CImg< T > & fillC` (const unsigned int x, const unsigned int y, const unsigned int z, const double a0,...)
*Fill pixel values along the C-axis at a specified pixel position **[overloading]**.*
- `template<typename t >`
`CImg< T > & discard` (const `CImg< T >` &values, const char axis=0)
Discard specified sequence of values in the image buffer, along a specific axis.
- `template<typename t >`
`CImg< T > get_discard` (const `CImg< t >` &values, const char axis=0) const
- `CImg< T > & discard` (const char axis=0)
Discard neighboring duplicates in the image buffer, along the specified axis.
- `CImg< T > get_discard` (const char axis=0) const
*Discard neighboring duplicates in the image buffer, along the specified axis **[new-instance version]**.*
- `CImg< T > & invert_endianness` ()
Invert endianness of all pixel values.
- `CImg< T > get_invert_endianness` () const
*Invert endianness of all pixel values **[new-instance version]**.*
- `CImg< T > & rand` (const T &val_min, const T &val_max)
Fill image with random values in specified range.
- `CImg< T > get_rand` (const T &val_min, const T &val_max) const
*Fill image with random values in specified range **[new-instance version]**.*
- `CImg< T > & round` (const double y=1, const int rounding_type=0)
Round pixel values.
- `CImg< T > get_round` (const double y=1, const unsigned int rounding_type=0) const
*Round pixel values **[new-instance version]**.*
- `CImg< T > & noise` (const double sigma, const unsigned int noise_type=0)
Add random noise to pixel values.
- `CImg< T > get_noise` (const double sigma, const unsigned int noise_type=0) const
*Add random noise to pixel values **[new-instance version]**.*
- `CImg< T > & normalize` (const T &min_value, const T &max_value)
Linearly normalize pixel values.
- `CImg< Tfloat > get_normalize` (const T &min_value, const T &max_value) const
*Linearly normalize pixel values **[new-instance version]**.*
- `CImg< T > & normalize` ()
Normalize multi-valued pixels of the image instance, with respect to their L2-norm.
- `CImg< Tfloat > get_normalize` () const
*Normalize multi-valued pixels of the image instance, with respect to their L2-norm **[new-instance version]**.*
- `CImg< T > & norm` (const int norm_type=2)
Compute Lp-norm of each multi-valued pixel of the image instance.
- `CImg< Tfloat > get_norm` (const int norm_type=2) const
*Compute L2-norm of each multi-valued pixel of the image instance **[new-instance version]**.*
- `CImg< T > & cut` (const T &min_value, const T &max_value)
Cut pixel values in specified range.

- `CImg< T > get_cut` (const T &min_value, const T &max_value) const
Cut pixel values in specified range [new-instance version].
- `CImg< T > & quantize` (const unsigned int nb_levels, const bool keep_range=true)
Uniformly quantize pixel values.
- `CImg< T > get_quantize` (const unsigned int n, const bool keep_range=true) const
Uniformly quantize pixel values [new-instance version].
- `CImg< T > & threshold` (const T &value, const bool soft_threshold=false, const bool strict_threshold=false)
Threshold pixel values.
- `CImg< T > get_threshold` (const T &value, const bool soft_threshold=false, const bool strict_threshold=false) const
Threshold pixel values [new-instance version].
- `CImg< T > & histogram` (const unsigned int nb_levels, const T &min_value, const T &max_value)
Compute the histogram of pixel values.
- `CImg< T > & histogram` (const unsigned int nb_levels)
Compute the histogram of pixel values [overloading].
- `CImg< unsignedT > get_histogram` (const unsigned int nb_levels, const T &min_value, const T &max_value) const
Compute the histogram of pixel values [new-instance version].
- `CImg< unsignedT > get_histogram` (const unsigned int nb_levels) const
Compute the histogram of pixel values [new-instance version].
- `CImg< T > & equalize` (const unsigned int nb_levels, const T &min_value, const T &max_value)
Equalize histogram of pixel values.
- `CImg< T > & equalize` (const unsigned int nb_levels)
Equalize histogram of pixel values [overloading].
- `CImg< T > get_equalize` (const unsigned int nlevels, const T &val_min, const T &val_max) const
Equalize histogram of pixel values [new-instance version].
- `CImg< T > get_equalize` (const unsigned int nlevels) const
Equalize histogram of pixel values [new-instance version].
- `template<typename t > CImg< T > & index` (const CImg< t > &colormap, const float dithering=1, const bool map_indexes=false)
Index multi-valued pixels regarding to a specified colormap.
- `template<typename t > CImg< typename CImg< t >::Tuint > get_index` (const CImg< t > &colormap, const float dithering=1, const bool map_indexes=true) const
Index multi-valued pixels regarding to a specified colormap [new-instance version].
- `template<typename t > CImg< T > & map` (const CImg< t > &colormap, const unsigned int boundary_conditions=0)
Map predefined colormap on the scalar (indexed) image instance.
- `template<typename t > CImg< t > get_map` (const CImg< t > &colormap, const unsigned int boundary_conditions=0) const
Map predefined colormap on the scalar (indexed) image instance [new-instance version].
- `CImg< T > & label` (const bool is_high_connectivity=false, const Tfloat tolerance=0)
Label connected components.
- `CImg< unsignedT > get_label` (const bool is_high_connectivity=false, const Tfloat tolerance=0) const
Label connected components [new-instance version].
- `template<typename t > CImg< T > & label` (const CImg< t > &connectivity_mask, const Tfloat tolerance=0)
Label connected components [overloading].
- `template<typename t > CImg< unsignedT > get_label` (const CImg< t > &connectivity_mask, const Tfloat tolerance=0) const
Label connected components [new-instance version].

Color Base Management

- [CImg< T > & sRGBtoRGB \(\)](#)
Convert pixel values from sRGB to RGB color spaces.
- [CImg< Tfloat > get_sRGBtoRGB \(\) const](#)
Convert pixel values from sRGB to RGB color spaces **[new-instance version]**.
- [CImg< T > & RGBtosRGB \(\)](#)
Convert pixel values from RGB to sRGB color spaces.
- [CImg< Tfloat > get_RGBtosRGB \(\) const](#)
Convert pixel values from RGB to sRGB color spaces **[new-instance version]**.
- [CImg< T > & RGBtoHSV \(\)](#)
Convert pixel values from RGB to HSV color spaces.
- [CImg< Tfloat > get_RGBtoHSV \(\) const](#)
Convert pixel values from RGB to HSV color spaces **[new-instance version]**.
- [CImg< T > & HSVtoRGB \(\)](#)
Convert pixel values from HSV to RGB color spaces.
- [CImg< Tuchar > get_HSVtoRGB \(\) const](#)
Convert pixel values from HSV to RGB color spaces **[new-instance version]**.
- [CImg< T > & RGBtoHSL \(\)](#)
Convert pixel values from RGB to HSL color spaces.
- [CImg< Tfloat > get_RGBtoHSL \(\) const](#)
Convert pixel values from RGB to HSL color spaces **[new-instance version]**.
- [CImg< T > & HSLtoRGB \(\)](#)
Convert pixel values from HSL to RGB color spaces.
- [CImg< Tuchar > get_HSLtoRGB \(\) const](#)
Convert pixel values from HSL to RGB color spaces **[new-instance version]**.
- [CImg< T > & RGBtoHSI \(\)](#)
Convert pixel values from RGB to HSI color spaces.
- [CImg< Tfloat > get_RGBtoHSI \(\) const](#)
Convert pixel values from RGB to HSI color spaces **[new-instance version]**.
- [CImg< T > & HSItoRGB \(\)](#)
Convert pixel values from HSI to RGB color spaces.
- [CImg< Tfloat > get_HSItoRGB \(\) const](#)
Convert pixel values from HSI to RGB color spaces **[new-instance version]**.
- [CImg< T > & RGBtoYCbCr \(\)](#)
Convert pixel values from RGB to YCbCr color spaces.
- [CImg< Tuchar > get_RGBtoYCbCr \(\) const](#)
Convert pixel values from RGB to YCbCr color spaces **[new-instance version]**.
- [CImg< T > & YCbCrtoRGB \(\)](#)
Convert pixel values from RGB to YCbCr color spaces.
- [CImg< Tuchar > get_YCbCrtoRGB \(\) const](#)
Convert pixel values from RGB to YCbCr color spaces **[new-instance version]**.
- [CImg< T > & RGBtoYUV \(\)](#)
Convert pixel values from RGB to YUV color spaces.
- [CImg< Tfloat > get_RGBtoYUV \(\) const](#)
Convert pixel values from RGB to YUV color spaces **[new-instance version]**.
- [CImg< T > & YUVtoRGB \(\)](#)
Convert pixel values from YUV to RGB color spaces.
- [CImg< Tuchar > get_YUVtoRGB \(\) const](#)
Convert pixel values from YUV to RGB color spaces **[new-instance version]**.
- [CImg< T > & RGBtoCMY \(\)](#)

- Convert pixel values from RGB to CMY color spaces.*

 - `CImg< Tuchar > get_RGBtoCMY ()` const

Convert pixel values from RGB to CMY color spaces [new-instance version].
- `CImg< T > & CMYtoRGB ()`

Convert pixel values from CMY to RGB color spaces.

 - `CImg< Tuchar > get_CMYtoRGB ()` const

Convert pixel values from CMY to RGB color spaces [new-instance version].
- `CImg< T > & CMYtoCMYK ()`

Convert pixel values from CMY to CMYK color spaces.

 - `CImg< Tuchar > get_CMYtoCMYK ()` const

Convert pixel values from CMY to CMYK color spaces [new-instance version].
- `CImg< T > & CMYKtoCMY ()`

Convert pixel values from CMYK to CMY color spaces.

 - `CImg< Tfloat > get_CMYKtoCMY ()` const

Convert pixel values from CMYK to CMY color spaces [new-instance version].
- `CImg< T > & RGBtoXYZ ()`

Convert pixel values from RGB to XYZ_709 color spaces.

 - `CImg< Tfloat > get_RGBtoXYZ ()` const

Convert pixel values from RGB to XYZ_709 color spaces [new-instance version].
- `CImg< T > & XYZtoRGB ()`

Convert pixel values from XYZ_709 to RGB color spaces.

 - `CImg< Tuchar > get_XYZtoRGB ()` const

Convert pixel values from XYZ_709 to RGB color spaces [new-instance version].
- `CImg< T > & XYZtoLab ()`

Convert pixel values from XYZ_709 to Lab color spaces.

 - `CImg< Tfloat > get_XYZtoLab ()` const

Convert pixel values from XYZ_709 to Lab color spaces [new-instance version].
- `CImg< T > & LabtoXYZ ()`

Convert pixel values from Lab to XYZ_709 color spaces.

 - `CImg< Tfloat > get_LabtoXYZ ()` const

Convert pixel values from Lab to XYZ_709 color spaces [new-instance version].
- `CImg< T > & XYZtoxyY ()`

Convert pixel values from XYZ_709 to xyY color spaces.

 - `CImg< Tfloat > get_XYZtoxyY ()` const

Convert pixel values from XYZ_709 to xyY color spaces [new-instance version].
- `CImg< T > & xyYtoXYZ ()`

Convert pixel values from xyY pixels to XYZ_709 color spaces.

 - `CImg< Tfloat > get_xyYtoXYZ ()` const

Convert pixel values from xyY pixels to XYZ_709 color spaces [new-instance version].
- `CImg< T > & RGBtoLab ()`

Convert pixel values from RGB to Lab color spaces.

 - `CImg< Tfloat > get_RGBtoLab ()` const

Convert pixel values from RGB to Lab color spaces [new-instance version].
- `CImg< T > & LabtoRGB ()`

Convert pixel values from Lab to RGB color spaces.

 - `CImg< Tuchar > get_LabtoRGB ()` const

Convert pixel values from Lab to RGB color spaces [new-instance version].
- `CImg< T > & RGBtoxyY ()`

Convert pixel values from RGB to xyY color spaces.

 - `CImg< Tfloat > get_RGBtoxyY ()` const

Convert pixel values from RGB to xyY color spaces [new-instance version].

- `CImg< T > & xyYtoRGB ()`
Convert pixel values from xyY to RGB color spaces.
- `CImg< Tuchar > get_xyYtoRGB () const`
Convert pixel values from xyY to RGB color spaces **[new-instance version]**.
- `CImg< T > & RGBtoCMYK ()`
Convert pixel values from RGB to CMYK color spaces.
- `CImg< Tfloat > get_RGBtoCMYK () const`
Convert pixel values from RGB to CMYK color spaces **[new-instance version]**.
- `CImg< T > & CMYKtoRGB ()`
Convert pixel values from CMYK to RGB color spaces.
- `CImg< Tuchar > get_CMYKtoRGB () const`
Convert pixel values from CMYK to RGB color spaces **[new-instance version]**.
- `static const CImg< Tuchar > & default_LUT256 ()`
Return colormap "default", containing 256 colors entries in RGB.
- `static const CImg< Tuchar > & HSV_LUT256 ()`
Return colormap "HSV", containing 256 colors entries in RGB.
- `static const CImg< Tuchar > & lines_LUT256 ()`
Return colormap "lines", containing 256 colors entries in RGB.
- `static const CImg< Tuchar > & hot_LUT256 ()`
Return colormap "hot", containing 256 colors entries in RGB.
- `static const CImg< Tuchar > & cool_LUT256 ()`
Return colormap "cool", containing 256 colors entries in RGB.
- `static const CImg< Tuchar > & jet_LUT256 ()`
Return colormap "jet", containing 256 colors entries in RGB.
- `static const CImg< Tuchar > & flag_LUT256 ()`
Return colormap "flag", containing 256 colors entries in RGB.
- `static const CImg< Tuchar > & cube_LUT256 ()`
Return colormap "cube", containing 256 colors entries in RGB.

Geometric / Spatial Manipulation

- `CImg< T > & resize (const int size_x, const int size_y=-100, const int size_z=-100, const int size_c=-100, const int interpolation_type=1, const unsigned int boundary_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0)`
Resize image to new dimensions.
- `CImg< T > get_resize (const int size_x, const int size_y=-100, const int size_z=-100, const int size_c=-100, const int interpolation_type=1, const unsigned int boundary_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0) const`
Resize image to new dimensions **[new-instance version]**.
- `template<typename t >`
`CImg< T > & resize (const CImg< t > &src, const int interpolation_type=1, const unsigned int boundary_↵_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0)`
Resize image to dimensions of another image.
- `template<typename t >`
`CImg< T > get_resize (const CImg< t > &src, const int interpolation_type=1, const unsigned int boundary_↵_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0) const`
Resize image to dimensions of another image **[new-instance version]**.
- `CImg< T > & resize (const CImgDisplay &disp, const int interpolation_type=1, const unsigned int boundary_↵_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0)`

Resize image to dimensions of a display window.

- `CImg< T > get_resize` (const `CImgDisplay` &disp, const int interpolation_type=1, const unsigned int boundary_conditions=0, const float centering_x=0, const float centering_y=0, const float centering_z=0, const float centering_c=0) const

Resize image to dimensions of a display window [new-instance version].

- `CImg< T > &resize_halfXY` ()

Resize image to half-size along XY axes, using an optimized filter.

- `CImg< T > get_resize_halfXY` () const

Resize image to half-size along XY axes, using an optimized filter [new-instance version].

- `CImg< T > &resize_doubleXY` ()

Resize image to double-size, using the Scale2X algorithm.

- `CImg< T > get_resize_doubleXY` () const

Resize image to double-size, using the Scale2X algorithm [new-instance version].

- `CImg< T > &resize_tripleXY` ()

Resize image to triple-size, using the Scale3X algorithm.

- `CImg< T > get_resize_tripleXY` () const

Resize image to triple-size, using the Scale3X algorithm [new-instance version].

- `CImg< T > &mirror` (const char axis)

Mirror image content along specified axis.

- `CImg< T > get_mirror` (const char axis) const

Mirror image content along specified axis [new-instance version].

- `CImg< T > &mirror` (const char *const axes)

Mirror image content along specified axes.

- `CImg< T > get_mirror` (const char *const axes) const

Mirror image content along specified axes [new-instance version].

- `CImg< T > &shift` (const int delta_x, const int delta_y=0, const int delta_z=0, const int delta_c=0, const int boundary_conditions=0)

Shift image content.

- `CImg< T > get_shift` (const int delta_x, const int delta_y=0, const int delta_z=0, const int delta_c=0, const int boundary_conditions=0) const

Shift image content [new-instance version].

- `CImg< T > &permute_axes` (const char *const order)

Permute axes order.

- `CImg< T > get_permute_axes` (const char *const order) const

Permute axes order [new-instance version].

- `CImg< T > &unroll` (const char axis)

Unroll pixel values along specified axis.

- `CImg< T > get_unroll` (const char axis) const

Unroll pixel values along specified axis [new-instance version].

- `CImg< T > &rotate` (const float angle, const unsigned int interpolation=1, const unsigned int boundary_conditions=0)

Rotate image with arbitrary angle.

- `CImg< T > get_rotate` (const float angle, const unsigned int interpolation=1, const unsigned int boundary_conditions=0) const

Rotate image with arbitrary angle [new-instance version].

- `CImg< T > &rotate` (const float angle, const float cx, const float cy, const float zoom, const unsigned int interpolation=1, const unsigned int boundary_conditions=0)

Rotate image with arbitrary angle, around a center point.

- `CImg< T > get_rotate` (const float angle, const float cx, const float cy, const float zoom, const unsigned int interpolation=1, const unsigned int boundary_conditions=0) const

Rotate image with arbitrary angle, around a center point [new-instance version].

- `template<typename t >`
`Clmg< T > & warp` (const `Clmg< T >` &warp, const unsigned int mode=0, const unsigned int interpolation=1, const unsigned int boundary_conditions=0)
Warp image content by a warping field.
- `template<typename t >`
`Clmg< T > get_warp` (const `Clmg< t >` &warp, const unsigned int mode=0, const unsigned int interpolation=1, const unsigned int boundary_conditions=0) const
*Warp image content by a warping field **[new-instance version]***
- `Clmg< T > get_projections2d` (const unsigned int x0, const unsigned int y0, const unsigned int z0) const
Generate a 2d representation of a 3d image, with XY,XZ and YZ views.
- `Clmg< T > & projections2d` (const unsigned int x0, const unsigned int y0, const unsigned int z0)
*Construct a 2d representation of a 3d image, with XY,XZ and YZ views **[in-place version]**.*
- `Clmg< T > & crop` (const int x0, const int y0, const int z0, const int c0, const int x1, const int y1, const int z1, const int c1, const bool boundary_conditions=false)
Crop image region.
- `Clmg< T > get_crop` (const int x0, const int y0, const int z0, const int c0, const int x1, const int y1, const int z1, const int c1, const bool boundary_conditions=false) const
*Crop image region **[new-instance version]**.*
- `Clmg< T > & crop` (const int x0, const int y0, const int z0, const int x1, const int y1, const int z1, const bool boundary_conditions=false)
*Crop image region **[overloading]**.*
- `Clmg< T > get_crop` (const int x0, const int y0, const int z0, const int x1, const int y1, const int z1, const bool boundary_conditions=false) const
*Crop image region **[new-instance version]**.*
- `Clmg< T > & crop` (const int x0, const int y0, const int x1, const int y1, const bool boundary_conditions=false)
*Crop image region **[overloading]**.*
- `Clmg< T > get_crop` (const int x0, const int y0, const int x1, const int y1, const bool boundary_conditions=false) const
*Crop image region **[new-instance version]**.*
- `Clmg< T > & crop` (const int x0, const int x1, const bool boundary_conditions=false)
*Crop image region **[overloading]**.*
- `Clmg< T > get_crop` (const int x0, const int x1, const bool boundary_conditions=false) const
*Crop image region **[new-instance version]**.*
- `Clmg< T > & autocrop` (const T &value, const char *const axes="czyx")
Autocrop image region, regarding the specified background value.
- `Clmg< T > get_autocrop` (const T &value, const char *const axes="czyx") const
*Autocrop image region, regarding the specified background value **[new-instance version]**.*
- `Clmg< T > & autocrop` (const T *const color=0, const char *const axes="zyx")
Autocrop image region, regarding the specified background color.
- `Clmg< T > get_autocrop` (const T *const color=0, const char *const axes="zyx") const
*Autocrop image region, regarding the specified background color **[new-instance version]**.*
- `template<typename t >`
`Clmg< T > & autocrop` (const `Clmg< t >` &color, const char *const axes="zyx")
*Autocrop image region, regarding the specified background color **[overloading]**.*
- `template<typename t >`
`Clmg< T > get_autocrop` (const `Clmg< t >` &color, const char *const axes="zyx") const
*Autocrop image region, regarding the specified background color **[new-instance version]**.*
- `Clmg< T > get_column` (const int x0) const
Return specified image column.
- `Clmg< T > & column` (const int x0)
*Return specified image column **[in-place version]**.*
- `Clmg< T > & columns` (const int x0, const int x1)

- Return specified range of image columns.*

 - `CImg< T > get_columns` (const int x0, const int x1) const

Return specified range of image columns [in-place version].

 - `CImg< T > get_row` (const int y0) const

Return specified image row.

 - `CImg< T > & row` (const int y0)

Return specified image row [in-place version].

 - `CImg< T > get_rows` (const int y0, const int y1) const

Return specified range of image rows.

 - `CImg< T > & rows` (const int y0, const int y1)

Return specified range of image rows [in-place version].

 - `CImg< T > get_slice` (const int z0) const

Return specified image slice.

 - `CImg< T > & slice` (const int z0)

Return specified image slice [in-place version].

 - `CImg< T > get_slices` (const int z0, const int z1) const

Return specified range of image slices.

 - `CImg< T > & slices` (const int z0, const int z1)

Return specified range of image slices [in-place version].

 - `CImg< T > get_channel` (const int c0) const

Return specified image channel.

 - `CImg< T > & channel` (const int c0)

Return specified image channel [in-place version].

 - `CImg< T > get_channels` (const int c0, const int c1) const

Return specified range of image channels.

 - `CImg< T > & channels` (const int c0, const int c1)

Return specified range of image channels [in-place version].

 - `CImg< floatT > get_streamline` (const float x, const float y, const float z, const float L=256, const float dl=0.1f, const unsigned int interpolation_type=2, const bool is_backward_tracking=false, const bool is_oriented_↔ only=false) const

Return stream line of a 2d or 3d vector field.

 - `CImg< T > get_shared_points` (const unsigned int x0, const unsigned int x1, const unsigned int y0=0, const unsigned int z0=0, const unsigned int c0=0)

Return a shared-memory image referencing a range of pixels of the image instance.

 - const `CImg< T > get_shared_points` (const unsigned int x0, const unsigned int x1, const unsigned int y0=0, const unsigned int z0=0, const unsigned int c0=0) const

Return a shared-memory image referencing a range of pixels of the image instance [const version].

 - `CImg< T > get_shared_rows` (const unsigned int y0, const unsigned int y1, const unsigned int z0=0, const unsigned int c0=0)

Return a shared-memory image referencing a range of rows of the image instance.

 - const `CImg< T > get_shared_rows` (const unsigned int y0, const unsigned int y1, const unsigned int z0=0, const unsigned int c0=0) const

Return a shared-memory image referencing a range of rows of the image instance [const version].

 - `CImg< T > get_shared_row` (const unsigned int y0, const unsigned int z0=0, const unsigned int c0=0)

Return a shared-memory image referencing one row of the image instance.

 - const `CImg< T > get_shared_row` (const unsigned int y0, const unsigned int z0=0, const unsigned int c0=0) const

Return a shared-memory image referencing one row of the image instance [const version].

 - `CImg< T > get_shared_slices` (const unsigned int z0, const unsigned int z1, const unsigned int c0=0)

Return a shared memory image referencing a range of slices of the image instance.

 - const `CImg< T > get_shared_slices` (const unsigned int z0, const unsigned int z1, const unsigned int c0=0) const

- Return a shared memory image referencing a range of slices of the image instance [const version].*

 - `Clmg< T > get_shared_slice` (const unsigned int z0, const unsigned int c0=0)

Return a shared-memory image referencing one slice of the image instance.

 - const `Clmg< T > get_shared_slice` (const unsigned int z0, const unsigned int c0=0) const

Return a shared-memory image referencing one slice of the image instance [const version].

 - `Clmg< T > get_shared_channels` (const unsigned int c0, const unsigned int c1)

Return a shared-memory image referencing a range of channels of the image instance.

 - const `Clmg< T > get_shared_channels` (const unsigned int c0, const unsigned int c1) const

Return a shared-memory image referencing a range of channels of the image instance [const version].

 - `Clmg< T > get_shared_channel` (const unsigned int c0)

Return a shared-memory image referencing one channel of the image instance.

 - const `Clmg< T > get_shared_channel` (const unsigned int c0) const

Return a shared-memory image referencing one channel of the image instance [const version].

 - `Clmg< T > get_shared` ()

Return a shared-memory version of the image instance.

 - const `Clmg< T > get_shared` () const

Return a shared-memory version of the image instance [const version].

 - `ClmgList< T > get_split` (const char axis, const int nb=-1) const

Split image into a list along specified axis.

 - template<typename t >
`ClmgList< T > get_split` (const `Clmg< t >` &values, const char axis=0, const bool keep_values=true) const

Split image into a list of sub-images, according to a specified splitting value sequence and optionnally axis.

 - template<typename t >
`Clmg< T > & append` (const `Clmg< t >` &img, const char axis='x', const float align=0)

Append two images along specified axis.

 - `Clmg< T > & append` (const `Clmg< T >` &img, const char axis='x', const float align=0)

Append two images along specified axis [specialization].

 - template<typename t >
`Clmg< typename cimng::superset< T, t >::type > get_append` (const `Clmg< T >` &img, const char axis='x', const float align=0) const

Append two images along specified axis [const version].

 - `Clmg< T > get_append` (const `Clmg< T >` &img, const char axis='x', const float align=0) const

Append two images along specified axis [specialization].

 - template<typename tfunc >
static `Clmg< floatT > streamline` (const tfunc &func, const float x, const float y, const float z, const float L=256, const float dl=0.1f, const unsigned int interpolation_type=2, const bool is_backward_tracking=false, const bool is_oriented_only=false, const float x0=0, const float y0=0, const float z0=0, const float x1=0, const float y1=0, const float z1=0)

Return stream line of a 3d vector field.

 - static `Clmg< floatT > streamline` (const char *const expression, const float x, const float y, const float z, const float L=256, const float dl=0.1f, const unsigned int interpolation_type=2, const bool is_backward_tracking=true, const bool is_oriented_only=false, const float x0=0, const float y0=0, const float z0=0, const float x1=0, const float y1=0, const float z1=0)

Return stream line of a 3d vector field [overloading].

Filtering / Transforms

- template<typename t >
`Clmg< T > & correlate` (const `Clmg< t >` &mask, const unsigned int boundary_conditions=1, const bool is_normalized=false)
- Correlate image by a mask.*

- `template<typename t >`
`CImg< typename cimg::superset2< T, t, float >::type > get_correlate` (const `CImg< t >` &mask, const unsigned int boundary_conditions=1, const bool is_normalized=false) const
Correlate image by a mask [new-instance version].
- `template<typename t >`
`CImg< T > & convolve` (const `CImg< t >` &mask, const unsigned int boundary_conditions=1, const bool is_normalized=false)
Convolve image by a mask.
- `CImg< T > & cumulate` (const char axis=0)
Cumulate image values, optionally along specified axis.
- `CImg< Tlong > get_cumulate` (const char axis=0) const
Cumulate image values, optionally along specified axis [new-instance version].
- `CImg< T > & cumulate` (const char *const axes)
Cumulate image values, along specified axes.
- `CImg< Tlong > get_cumulate` (const char *const axes) const
Cumulate image values, along specified axes .
- `template<typename t >`
`CImg< typename cimg::superset2< T, t, float >::type > get_convolve` (const `CImg< t >` &mask, const unsigned int boundary_conditions=1, const bool is_normalized=false) const
Convolve image by a mask [new-instance version].
- `template<typename t >`
`CImg< T > & erode` (const `CImg< t >` &mask, const unsigned int boundary_conditions=1, const bool is_↵ normalized=false)
Erode image by a structuring element.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > get_erode` (const `CImg< t >` &mask, const unsigned int boundary_conditions=1, const bool is_normalized=false) const
Erode image by a structuring element [new-instance version].
- `CImg< T > & erode` (const unsigned int sx, const unsigned int sy, const unsigned int sz=1)
Erode image by a rectangular structuring element of specified size.
- `CImg< T > get_erode` (const unsigned int sx, const unsigned int sy, const unsigned int sz=1) const
Erode image by a rectangular structuring element of specified size [new-instance version].
- `CImg< T > & erode` (const unsigned int s)
Erode the image by a square structuring element of specified size.
- `CImg< T > get_erode` (const unsigned int s) const
Erode the image by a square structuring element of specified size [new-instance version].
- `template<typename t >`
`CImg< T > & dilate` (const `CImg< t >` &mask, const unsigned int boundary_conditions=1, const bool is_↵ normalized=false)
Dilate image by a structuring element.
- `template<typename t >`
`CImg< typename cimg::superset< T, t >::type > get_dilate` (const `CImg< t >` &mask, const unsigned int boundary_conditions=1, const bool is_normalized=false) const
Dilate image by a structuring element [new-instance version].
- `CImg< T > & dilate` (const unsigned int sx, const unsigned int sy, const unsigned int sz=1)
Dilate image by a rectangular structuring element of specified size.
- `CImg< T > get_dilate` (const unsigned int sx, const unsigned int sy, const unsigned int sz=1) const
Dilate image by a rectangular structuring element of specified size [new-instance version].
- `CImg< T > & dilate` (const unsigned int s)
Dilate image by a square structuring element of specified size.
- `CImg< T > get_dilate` (const unsigned int s) const
Dilate image by a square structuring element of specified size [new-instance version].

- `template<typename t >`
`Clmg< T > & watershed` (const `Clmg< t >` &priority, const bool is_high_connectivity=false)
Compute watershed transform.
- `template<typename t >`
`Clmg< T > get_watershed` (const `Clmg< t >` &priority, const bool is_high_connectivity=false) const
Compute watershed transform [new-instance version].
- `Clmg< T > & deriche` (const float sigma, const unsigned int order=0, const char axis='x', const bool boundary_conditions=true)
Apply recursive Deriche filter.
- `Clmg< Tfloat > get_deriche` (const float sigma, const unsigned int order=0, const char axis='x', const bool boundary_conditions=true) const
Apply recursive Deriche filter [new-instance version].
- `Clmg< T > & vanvliet` (const float sigma, const unsigned int order, const char axis='x', const bool boundary_↵_conditions=true)
Van Vliet recursive Gaussian filter.
- `Clmg< Tfloat > get_vanvliet` (const float sigma, const unsigned int order, const char axis='x', const bool boundary_conditions=true) const
Blur image using Van Vliet recursive Gaussian filter. [new-instance version].
- `Clmg< T > & blur` (const float sigma_x, const float sigma_y, const float sigma_z, const bool boundary_↵_conditions=true, const bool is_gaussian=false)
Blur image.
- `Clmg< Tfloat > get_blur` (const float sigma_x, const float sigma_y, const float sigma_z, const bool boundary_conditions=true, const bool is_gaussian=false) const
Blur image [new-instance version].
- `Clmg< T > & blur` (const float sigma, const bool boundary_conditions=true, const bool is_gaussian=false)
Blur image isotropically.
- `Clmg< Tfloat > get_blur` (const float sigma, const bool boundary_conditions=true, const bool is_↵_gaussian=false) const
Blur image isotropically [new-instance version].
- `template<typename t >`
`Clmg< T > & blur_anisotropic` (const `Clmg< t >` &G, const float amplitude=60, const float dl=0.8f, const float da=30, const float gauss_prec=2, const unsigned int interpolation_type=0, const bool is_fast_approx=1)
Blur image anisotropically, directed by a field of diffusion tensors.
- `template<typename t >`
`Clmg< Tfloat > get_blur_anisotropic` (const `Clmg< t >` &G, const float amplitude=60, const float dl=0.8f, const float da=30, const float gauss_prec=2, const unsigned int interpolation_type=0, const bool is_fast_↵_approx=true) const
Blur image anisotropically, directed by a field of diffusion tensors [new-instance version].
- `Clmg< T > & blur_anisotropic` (const float amplitude, const float sharpness=0.7f, const float anisotropy=0.6f, const float alpha=0.6f, const float sigma=1.1f, const float dl=0.8f, const float da=30, const float gauss_prec=2, const unsigned int interpolation_type=0, const bool is_fast_approx=true)
Blur image anisotropically, in an edge-preserving way.
- `Clmg< Tfloat > get_blur_anisotropic` (const float amplitude, const float sharpness=0.7f, const float anisotropy=0.6f, const float alpha=0.6f, const float sigma=1.1f, const float dl=0.8f, const float da=30, const float gauss_prec=2, const unsigned int interpolation_type=0, const bool is_fast_approx=true) const
Blur image anisotropically, in an edge-preserving way [new-instance version].
- `template<typename t >`
`Clmg< T > & blur_bilateral` (const `Clmg< t >` &guide, const float sigma_x, const float sigma_y, const float sigma_z, const float sigma_r, const float sampling_x, const float sampling_y, const float sampling_z, const float sampling_r)
Blur image, with the joint bilateral filter.
- `template<typename t >`
`Clmg< Tfloat > get_blur_bilateral` (const `Clmg< t >` &guide, const float sigma_x, const float sigma_y, const float sigma_z, const float sigma_r, const float sampling_x, const float sampling_y, const float sampling_z, const float sampling_r) const

Blur image, with the joint bilateral filter [new-instance version].

- `template<typename t >`
`CImg< T > & blur_bilateral (const CImg< t > &guide, const float sigma_s, const float sigma_r, const float sampling_s=0, const float sampling_r=0)`

Blur image using the joint bilateral filter.

- `template<typename t >`
`CImg< Tfloat > get_blur_bilateral (const CImg< t > &guide, const float sigma_s, const float sigma_r, const float sampling_s=0, const float sampling_r=0) const`

Blur image using the bilateral filter [new-instance version].

- `CImg< T > & boxfilter (const float boxsize, const int order, const char axis='x', const bool boundary_conditions=true)`
- `CImg< Tfloat > get_boxfilter (const float boxsize, const int order, const char axis='x', const bool boundary_conditions=true) const`
- `CImg< T > & blur_box (const float boxsize_x, const float boxsize_y, const float boxsize_z, const bool boundary_conditions=true)`

Blur image with a box filter.

- `CImg< Tfloat > get_blur_box (const float boxsize_x, const float boxsize_y, const float boxsize_z, const bool boundary_conditions=true) const`

Blur image with a box filter [new-instance version].

- `CImg< T > & blur_box (const float boxsize, const bool boundary_conditions=true)`

Blur image with a box filter.

- `CImg< Tfloat > get_blur_box (const float boxsize, const bool boundary_conditions=true) const`

Blur image with a box filter [new-instance version].

- `template<typename t >`
`CImg< T > & blur_guided (const CImg< t > &guide, const float radius, const float regularization)`

Blur image, with the image guided filter.

- `template<typename t >`
`CImg< Tfloat > get_blur_guided (const CImg< t > &guide, const float radius, const float regularization) const`

Blur image, with the image guided filter [new-instance version].

- `CImg< T > & blur_patch (const float sigma_s, const float sigma_p, const unsigned int patch_size=3, const unsigned int lookup_size=4, const float smoothness=0, const bool is_fast_approx=true)`

Blur image using patch-based space.

- `CImg< Tfloat > get_blur_patch (const float sigma_s, const float sigma_p, const unsigned int patch_size=3, const unsigned int lookup_size=4, const float smoothness=0, const bool is_fast_approx=true) const`

Blur image using patch-based space [new-instance version].

- `CImg< T > & blur_median (const unsigned int n, const float threshold=0)`

Blur image with the median filter.

- `CImg< T > get_blur_median (const unsigned int n, const float threshold=0) const`

Blur image with the median filter [new-instance version].

- `CImg< T > & sharpen (const float amplitude, const bool sharpen_type=false, const float edge=1, const float alpha=0, const float sigma=0)`

Sharpen image.

- `CImg< T > get_sharpen (const float amplitude, const bool sharpen_type=false, const float edge=1, const float alpha=0, const float sigma=0) const`

Sharpen image [new-instance version].

- `CImgList< Tfloat > get_gradient (const char *const axes=0, const int scheme=3) const`

Return image gradient.

- `CImgList< Tfloat > get_hessian (const char *const axes=0) const`

Return image hessian.

- `CImg< T > & laplacian ()`

Compute image laplacian.

- `CImg< Tfloat > get_laplacian () const`

- Compute image laplacian [new-instance version].*

 - `CImg< T > & structure_tensors` (const bool is_fwbw_scheme=false)

Compute the structure tensor field of an image.

 - `CImg< Tfloat > get_structure_tensors` (const bool is_fwbw_scheme=false) const

Compute the structure tensor field of an image [new-instance version].

 - `CImg< T > & diffusion_tensors` (const float sharpness=0.7f, const float anisotropy=0.6f, const float alpha=0.6f, const float sigma=1.1f, const bool is_sqrt=false)

Compute field of diffusion tensors for edge-preserving smoothing.

 - `CImg< Tfloat > get_diffusion_tensors` (const float sharpness=0.7f, const float anisotropy=0.6f, const float alpha=0.6f, const float sigma=1.1f, const bool is_sqrt=false) const

Compute field of diffusion tensors for edge-preserving smoothing [new-instance version].

 - `CImg< T > & displacement` (const `CImg< T > &source`, const float smoothness=0.1f, const float precision=5.0f, const unsigned int nb_scales=0, const unsigned int iteration_max=10000, const bool is_↵ backward=false, const `CImg< floatT > &guide=CImg< floatT >::const_empty()`)

Estimate displacement field between two images.

 - `CImg< floatT > get_displacement` (const `CImg< T > &source`, const float smoothness=0.1f, const float precision=5.0f, const unsigned int nb_scales=0, const unsigned int iteration_max=10000, const bool is_↵ backward=false, const `CImg< floatT > &guide=CImg< floatT >::const_empty()`) const

Estimate displacement field between two images [new-instance version].

 - `template<typename t1 , typename t2 >`
`CImg< T > & patchmatch` (const `CImg< T > &patch_image`, const unsigned int patch_width, const unsigned int patch_height, const unsigned int patch_depth, const unsigned int nb_iterations, const unsigned int nb_↵ randoms, const `CImg< t1 > &guide`, `CImg< t2 > &matching_score`)

Compute correspondence map between two images, using the patch-match algorithm.

 - `template<typename t1 , typename t2 >`
`CImg< intT > get_patchmatch` (const `CImg< T > &patch_image`, const unsigned int patch_width, const unsigned int patch_height, const unsigned int patch_depth, const unsigned int nb_iterations, const unsigned int nb_randoms, const `CImg< t1 > &guide`, `CImg< t2 > &matching_score`) const

Compute correspondence map between two images, using the patch-match algorithm [new-instance version].

 - `template<typename t >`
`CImg< T > & patchmatch` (const `CImg< T > &patch_image`, const unsigned int patch_width, const unsigned int patch_height, const unsigned int patch_depth, const unsigned int nb_iterations, const unsigned int nb_↵ randoms, const `CImg< t > &guide`)

Compute correspondence map between two images, using the patch-match algorithm [overloading].

 - `template<typename t >`
`CImg< intT > get_patchmatch` (const `CImg< T > &patch_image`, const unsigned int patch_width, const unsigned int patch_height, const unsigned int patch_depth, const unsigned int nb_iterations, const unsigned int nb_randoms, const `CImg< t > &guide`) const

Compute correspondence map between two images, using the patch-match algorithm [overloading].

 - `CImg< T > & patchmatch` (const `CImg< T > &patch_image`, const unsigned int patch_width, const unsigned int patch_height, const unsigned int patch_depth=1, const unsigned int nb_iterations=5, const unsigned int nb_randoms=5)

Compute correspondence map between two images, using the patch-match algorithm [overloading].

 - `CImg< intT > get_patchmatch` (const `CImg< T > &patch_image`, const unsigned int patch_width, const unsigned int patch_height, const unsigned int patch_depth=1, const unsigned int nb_iterations=5, const unsigned int nb_randoms=5) const

Compute correspondence map between two images, using the patch-match algorithm [overloading].

 - `CImg< T > & distance` (const T &value, const unsigned int metric=2)

Compute Euclidean distance function to a specified value.

 - `CImg< Tfloat > get_distance` (const T &value, const unsigned int metric=2) const

Compute distance to a specified value [new-instance version].

 - `template<typename t >`
`CImg< T > & distance` (const T &value, const `CImg< t > &metric_mask`)

Compute chamfer distance to a specified value, with a custom metric.

- `template<typename t >`
`CImg< Tfloat > get_distance` (const T &value, const CImg< t > &metric_mask) const
Compute chamfer distance to a specified value, with a custom metric [new-instance version].
- `template<typename t , typename to >`
`CImg< T > & distance_dijkstra` (const T &value, const CImg< t > &metric, const bool is_high_connectivity, CImg< to > &return_path)
Compute distance to a specified value, according to a custom metric (use dijkstra algorithm).
- `template<typename t , typename to >`
`CImg< typename cimg::superset< t, long >::type > get_distance_dijkstra` (const T &value, const CImg< t > &metric, const bool is_high_connectivity, CImg< to > &return_path) const
Compute distance map to a specified value, according to a custom metric (use dijkstra algorithm) [new-instance version].
- `template<typename t >`
`CImg< T > & distance_dijkstra` (const T &value, const CImg< t > &metric, const bool is_high_↵connectivity=false)
Compute distance map to a specified value, according to a custom metric (use dijkstra algorithm). [overloading].
- `template<typename t >`
`CImg< Tfloat > get_distance_dijkstra` (const T &value, const CImg< t > &metric, const bool is_high_↵connectivity=false) const
Compute distance map to a specified value, according to a custom metric (use dijkstra algorithm). [new-instance version].
- `template<typename t >`
`CImg< T > & distance_eikonal` (const T &value, const CImg< t > &metric)
Compute distance map to one source point, according to a custom metric (use fast marching algorithm).
- `template<typename t >`
`CImg< Tfloat > get_distance_eikonal` (const T &value, const CImg< t > &metric) const
Compute distance map to one source point, according to a custom metric (use fast marching algorithm).
- `CImg< T > & distance_eikonal` (const unsigned int nb_iterations, const float band_size=0, const float time_↵_step=0.5f)
Compute distance function to 0-valued isophotes, using the Eikonal PDE.
- `CImg< Tfloat > get_distance_eikonal` (const unsigned int nb_iterations, const float band_size=0, const float time_step=0.5f) const
Compute distance function to 0-valued isophotes, using the Eikonal PDE [new-instance version].
- `CImg< T > & haar` (const char axis, const bool invert=false, const unsigned int nb_scales=1)
Compute Haar multiscale wavelet transform.
- `CImg< Tfloat > get_haar` (const char axis, const bool invert=false, const unsigned int nb_scales=1) const
Compute Haar multiscale wavelet transform [new-instance version].
- `CImg< T > & haar` (const bool invert=false, const unsigned int nb_scales=1)
Compute Haar multiscale wavelet transform [overloading].
- `CImg< Tfloat > get_haar` (const bool invert=false, const unsigned int nb_scales=1) const
Compute Haar multiscale wavelet transform [new-instance version].
- `CImgList< Tfloat > get_FFT` (const char axis, const bool is_invert=false) const
Compute 1d Fast Fourier Transform, along a specified axis.
- `CImgList< Tfloat > get_FFT` (const bool is_invert=false) const
Compute n-d Fast Fourier Transform.
- static void `FFT` (CImg< T > &real, CImg< T > &imag, const char axis, const bool is_invert=false)
Compute 1d Fast Fourier Transform, along a specified axis.
- static void `FFT` (CImg< T > &real, CImg< T > &imag, const bool is_invert=false, const unsigned int nb_↵threads=0)
Compute n-d Fast Fourier Transform.

3d Objects Management

- `Clmg< T > & shift_object3d` (const float tx, const float ty=0, const float tz=0)
Shift 3d object's vertices.
- `Clmg< Tfloat > get_shift_object3d` (const float tx, const float ty=0, const float tz=0) const
*Shift 3d object's vertices **[new-instance version]**.*
- `Clmg< T > & shift_object3d` ()
Shift 3d object's vertices, so that it becomes centered.
- `Clmg< Tfloat > get_shift_object3d` () const
*Shift 3d object's vertices, so that it becomes centered **[new-instance version]**.*
- `Clmg< T > & resize_object3d` (const float sx, const float sy=-100, const float sz=-100)
Resize 3d object.
- `Clmg< Tfloat > get_resize_object3d` (const float sx, const float sy=-100, const float sz=-100) const
*Resize 3d object **[new-instance version]**.*
- `Clmg< T > resize_object3d` ()
Resize 3d object to unit size.
- `Clmg< Tfloat > get_resize_object3d` () const
*Resize 3d object to unit size **[new-instance version]**.*
- `template<typename tf, typename tp, typename tff >`
`Clmg< T > & append_object3d` (ClmgList< tf > &primitives, const Clmg< tp > &obj_vertices, const ClmgList< tff > &obj_primitives)
Merge two 3d objects together.
- `template<typename tp, typename tc, typename tt, typename tx >`
`const Clmg< T > & texturize_object3d` (ClmgList< tp > &primitives, ClmgList< tc > &colors, const Clmg< tt > &texture, const Clmg< tx > &coords=Clmg< tx >::const_empty()) const
Texturize primitives of a 3d object.
- `template<typename tf, typename tc, typename te >`
`Clmg< floatT > get_elevation3d` (ClmgList< tf > &primitives, ClmgList< tc > &colors, const Clmg< te > &elevation) const
Generate a 3d elevation of the image instance.
- `template<typename tf, typename tc >`
`Clmg< floatT > get_projections3d` (ClmgList< tf > &primitives, ClmgList< tc > &colors, const unsigned int x0, const unsigned int y0, const unsigned int z0, const bool normalize_colors=false) const
Generate the 3d projection planes of the image instance.
- `template<typename tf >`
`Clmg< floatT > get_isoline3d` (ClmgList< tf > &primitives, const float isoval, const int size_x=-100, const int size_y=-100) const
Generate a isoline of the image instance as a 3d object.
- `template<typename tf >`
`Clmg< floatT > get_isosurface3d` (ClmgList< tf > &primitives, const float isoval, const int size_x=-100, const int size_y=-100, const int size_z=-100) const
Generate an isosurface of the image instance as a 3d object.
- `template<typename tp, typename tc, typename to >`
`Clmg< T > & object3dtoClmg3d` (const ClmgList< tp > &primitives, const ClmgList< tc > &colors, const to &opacities, const bool full_check=true)
Convert 3d object into a Clmg3d representation.
- `template<typename tp, typename tc >`
`Clmg< T > & object3dtoClmg3d` (const ClmgList< tp > &primitives, const ClmgList< tc > &colors, const bool full_check=true)
*Convert 3d object into a Clmg3d representation **[overloading]**.*
- `template<typename tp >`
`Clmg< T > & object3dtoClmg3d` (const ClmgList< tp > &primitives, const bool full_check=true)
*Convert 3d object into a Clmg3d representation **[overloading]**.*

- `CImg< T > & object3dtoCImg3d` (const bool full_check=true)
Convert 3d object into a CImg3d representation **[overloading]**.
- `template<typename tp, typename tc, typename to >`
`CImg< floatT > get_object3dtoCImg3d` (const CImgList< tp > &primitives, const CImgList< tc > &colors, const to &opacities, const bool full_check=true) const
Convert 3d object into a CImg3d representation **[new-instance version]**.
- `template<typename tp, typename tc >`
`CImg< floatT > get_object3dtoCImg3d` (const CImgList< tp > &primitives, const CImgList< tc > &colors, const bool full_check=true) const
Convert 3d object into a CImg3d representation **[overloading]**.
- `template<typename tp >`
`CImg< floatT > get_object3dtoCImg3d` (const CImgList< tp > &primitives, const bool full_check=true) const
Convert 3d object into a CImg3d representation **[overloading]**.
- `CImg< floatT > get_object3dtoCImg3d` (const bool full_check=true) const
Convert 3d object into a CImg3d representation **[overloading]**.
- `template<typename tp, typename tc, typename to >`
`CImg< T > & CImg3dtoobject3d` (CImgList< tp > &primitives, CImgList< tc > &colors, CImgList< to > &opacities, const bool full_check=true)
Convert CImg3d representation into a 3d object.
- `template<typename tp, typename tc, typename to >`
`CImg< T > get_CImg3dtoobject3d` (CImgList< tp > &primitives, CImgList< tc > &colors, CImgList< to > &opacities, const bool full_check=true) const
Convert CImg3d representation into a 3d object **[new-instance version]**.
- `template<typename tf, typename tfunc >`
`static CImg< floatT > elevation3d` (CImgList< tf > &primitives, const tfunc &func, const float x0, const float y0, const float x1, const float y1, const int size_x=256, const int size_y=256)
Compute 3d elevation of a function as a 3d object.
- `template<typename tf >`
`static CImg< floatT > elevation3d` (CImgList< tf > &primitives, const char *const expression, const float x0, const float y0, const float x1, const float y1, const int size_x=256, const int size_y=256)
Compute 3d elevation of a function, as a 3d object **[overloading]**.
- `template<typename tf, typename tfunc >`
`static CImg< floatT > isoline3d` (CImgList< tf > &primitives, const tfunc &func, const float isoval, const float x0, const float y0, const float x1, const float y1, const int size_x=256, const int size_y=256)
Compute 0-isolines of a function, as a 3d object.
- `template<typename tf >`
`static CImg< floatT > isoline3d` (CImgList< tf > &primitives, const char *const expression, const float isoval, const float x0, const float y0, const float x1, const float y1, const int size_x=256, const int size_y=256)
Compute isolines of a function, as a 3d object **[overloading]**.
- `template<typename tf, typename tfunc >`
`static CImg< floatT > isosurface3d` (CImgList< tf > &primitives, const tfunc &func, const float isoval, const float x0, const float y0, const float z0, const float x1, const float y1, const float z1, const int size_x=32, const int size_y=32, const int size_z=32)
Compute isosurface of a function, as a 3d object.
- `template<typename tf >`
`static CImg< floatT > isosurface3d` (CImgList< tf > &primitives, const char *const expression, const float isoval, const float x0, const float y0, const float z0, const float x1, const float y1, const float z1, const int dx=32, const int dy=32, const int dz=32)
Compute isosurface of a function, as a 3d object **[overloading]**.
- `template<typename tf >`
`static CImg< floatT > box3d` (CImgList< tf > &primitives, const float size_x=200, const float size_y=100, const float size_z=100)
Generate a 3d box object.

- `template<typename tf >`
`static Clmg< floatT > cone3d (ClmgList< tf > &primitives, const float radius=50, const float size_z=100, const unsigned int subdivisions=24)`
Generate a 3d cone.
- `template<typename tf >`
`static Clmg< floatT > cylinder3d (ClmgList< tf > &primitives, const float radius=50, const float size_z=100, const unsigned int subdivisions=24)`
Generate a 3d cylinder.
- `template<typename tf >`
`static Clmg< floatT > torus3d (ClmgList< tf > &primitives, const float radius1=100, const float radius2=30, const unsigned int subdivisions1=24, const unsigned int subdivisions2=12)`
Generate a 3d torus.
- `template<typename tf >`
`static Clmg< floatT > plane3d (ClmgList< tf > &primitives, const float size_x=100, const float size_y=100, const unsigned int subdivisions_x=10, const unsigned int subdivisions_y=10)`
Generate a 3d XY-plane.
- `template<typename tf >`
`static Clmg< floatT > sphere3d (ClmgList< tf > &primitives, const float radius=50, const unsigned int subdivisions=3)`
Generate a 3d sphere.
- `template<typename tf, typename t >`
`static Clmg< floatT > ellipsoid3d (ClmgList< tf > &primitives, const Clmg< t > &tensor, const unsigned int subdivisions=3)`
Generate a 3d ellipsoid.

Drawing Functions

- `template<typename tc >`
`Clmg< T > & draw_point (const int x0, const int y0, const int z0, const tc *const color, const float opacity=1)`
Draw a 3d point.
- `template<typename tc >`
`Clmg< T > & draw_point (const int x0, const int y0, const tc *const color, const float opacity=1)`
Draw a 2d point [simplification].
- `template<typename t, typename tc >`
`Clmg< T > & draw_point (const Clmg< t > &points, const tc *const color, const float opacity=1)`
- `template<typename tc >`
`Clmg< T > & draw_line (const int x0, const int y0, const int x1, const int y1, const tc *const color, const float opacity=1, const unsigned int pattern=~0U, const bool init_hatch=true)`
Draw a 2d line.
- `template<typename tz, typename tc >`
`Clmg< T > & draw_line (Clmg< tz > &zbuffer, const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const tc *const color, const float opacity=1, const unsigned int pattern=~0U, const bool init_hatch=true)`
Draw a 2d line, with z-buffering.
- `template<typename tc >`
`Clmg< T > & draw_line (const int x0, const int y0, const int z0, const int x1, const int y1, const int z1, const tc *const color, const float opacity=1, const unsigned int pattern=~0U, const bool init_hatch=true)`
Draw a 3d line.
- `template<typename tc >`
`Clmg< T > & draw_line (const int x0, const int y0, const int x1, const int y1, const Clmg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const float opacity=1, const unsigned int pattern=~0U, const bool init_hatch=true)`
Draw a textured 2d line.

- `template<typename tc >`
`Clmg< T > & draw_line` (const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const `Clmg< tc >` &texture, const int tx0, const int ty0, const int tx1, const int ty1, const float opacity=1, const unsigned int pattern=~0U, const bool init_hatch=true)
Draw a textured 2d line, with perspective correction.
- `template<typename tz , typename tc >`
`Clmg< T > & draw_line` (`Clmg< tz >` &zbuffer, const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const `Clmg< tc >` &texture, const int tx0, const int ty0, const int tx1, const int ty1, const float opacity=1, const unsigned int pattern=~0U, const bool init_hatch=true)
Draw a textured 2d line, with perspective correction and z-buffering.
- `template<typename t , typename tc >`
`Clmg< T > & draw_line` (const `Clmg< t >` &points, const tc *const color, const float opacity=1, const unsigned int pattern=~0U, const bool init_hatch=true)
Draw a set of consecutive lines.
- `template<typename tc >`
`Clmg< T > & draw_arrow` (const int x0, const int y0, const int x1, const int y1, const tc *const color, const float opacity=1, const float angle=30, const float length=-10, const unsigned int pattern=~0U)
Draw a 2d arrow.
- `template<typename tc >`
`Clmg< T > & draw_spline` (const int x0, const int y0, const float u0, const float v0, const int x1, const int y1, const float u1, const float v1, const tc *const color, const float opacity=1, const float precision=0.25, const unsigned int pattern=~0U, const bool init_hatch=true)
Draw a 2d spline.
- `template<typename tc >`
`Clmg< T > & draw_spline` (const int x0, const int y0, const int z0, const float u0, const float v0, const float w0, const int x1, const int y1, const int z1, const float u1, const float v1, const float w1, const tc *const color, const float opacity=1, const float precision=4, const unsigned int pattern=~0U, const bool init_hatch=true)
Draw a 3d spline [overloading].
- `template<typename t >`
`Clmg< T > & draw_spline` (const int x0, const int y0, const float u0, const float v0, const int x1, const int y1, const float u1, const float v1, const `Clmg< t >` &texture, const int tx0, const int ty0, const int tx1, const int ty1, const float opacity=1, const float precision=4, const unsigned int pattern=~0U, const bool init_hatch=true)
Draw a textured 2d spline.
- `template<typename tp , typename tt , typename tc >`
`Clmg< T > & draw_spline` (const `Clmg< tp >` &points, const `Clmg< tt >` &tangents, const tc *const color, const float opacity=1, const bool is_closed_set=false, const float precision=4, const unsigned int pattern=~0U, const bool init_hatch=true)
Draw a set of consecutive splines.
- `template<typename tp , typename tc >`
`Clmg< T > & draw_spline` (const `Clmg< tp >` &points, const tc *const color, const float opacity=1, const bool is_closed_set=false, const float precision=4, const unsigned int pattern=~0U, const bool init_hatch=true)
Draw a set of consecutive splines [overloading].
- `template<typename tc >`
`Clmg< T > & draw_triangle` (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const tc *const color, const float opacity=1)
Draw a filled 2d triangle.
- `template<typename tc >`
`Clmg< T > & draw_triangle` (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const tc *const color, const float opacity, const unsigned int pattern)
Draw a outlined 2d triangle.
- `template<typename tz , typename tc >`
`Clmg< T > & draw_triangle` (`Clmg< tz >` &zbuffer, const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const int x2, const int y2, const float z2, const tc *const color, const float opacity=1, const float brightness=1)
Draw a filled 2d triangle, with z-buffering.

- `template<typename tc >`
`CImg< T > & draw_triangle` (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const tc *const color, const float brightness0, const float brightness1, const float brightness2, const float opacity=1)
Draw a Gouraud-shaded 2d triangle.
- `template<typename tz , typename tc >`
`CImg< T > & draw_triangle` (CImg< tz > &zbuffer, const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const int x2, const int y2, const float z2, const tc *const color, const float brightness0, const float brightness1, const float brightness2, const float opacity=1)
Draw a Gouraud-shaded 2d triangle, with z-buffering [overloading].
- `template<typename tc1 , typename tc2 , typename tc3 >`
`CImg< T > & draw_triangle` (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const tc1 *const color1, const tc2 *const color2, const tc3 *const color3, const float opacity=1)
Draw a color-interpolated 2d triangle.
- `template<typename tc >`
`CImg< T > & draw_triangle` (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const CImg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const float opacity=1, const float brightness=1)
Draw a textured 2d triangle.
- `template<typename tc >`
`CImg< T > & draw_triangle` (const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const int x2, const int y2, const float z2, const CImg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const float opacity=1, const float brightness=1)
Draw a 2d textured triangle, with perspective correction.
- `template<typename tz , typename tc >`
`CImg< T > & draw_triangle` (CImg< tz > &zbuffer, const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const int x2, const int y2, const float z2, const CImg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const float opacity=1, const float brightness=1)
Draw a textured 2d triangle, with perspective correction and z-buffering.
- `template<typename tc , typename tl >`
`CImg< T > & draw_triangle` (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const tc *const color, const CImg< tl > &light, const int lx0, const int ly0, const int lx1, const int ly1, const int lx2, const int ly2, const float opacity=1)
Draw a Phong-shaded 2d triangle.
- `template<typename tz , typename tc , typename tl >`
`CImg< T > & draw_triangle` (CImg< tz > &zbuffer, const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const int x2, const int y2, const float z2, const tc *const color, const CImg< tl > &light, const int lx0, const int ly0, const int lx1, const int ly1, const int lx2, const int ly2, const float opacity=1)
Draw a Phong-shaded 2d triangle, with z-buffering.
- `template<typename tc >`
`CImg< T > & draw_triangle` (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const CImg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const float brightness0, const float brightness1, const float brightness2, const float opacity=1)
Draw a textured Gouraud-shaded 2d triangle.
- `template<typename tc >`
`CImg< T > & draw_triangle` (const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const int x2, const int y2, const float z2, const CImg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const float brightness0, const float brightness1, const float brightness2, const float opacity=1)
Draw a textured Gouraud-shaded 2d triangle, with perspective correction [overloading].
- `template<typename tz , typename tc >`
`CImg< T > & draw_triangle` (CImg< tz > &zbuffer, const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const int x2, const int y2, const float z2, const CImg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const float brightness0, const float brightness1, const float brightness2, const float opacity=1)

Draw a textured Gouraud-shaded 2d triangle, with perspective correction and z-buffering [overloading].

- template<typename tc, typename tl >
CImg< T > & draw_triangle (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const CImg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const CImg< tl > &light, const int lx0, const int ly0, const int lx1, const int ly1, const int lx2, const int ly2, const float opacity=1)

Draw a textured Phong-shaded 2d triangle.

- template<typename tc, typename tl >
CImg< T > & draw_triangle (const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const int x2, const int y2, const float z2, const CImg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const CImg< tl > &light, const int lx0, const int ly0, const int lx1, const int ly1, const int lx2, const int ly2, const float opacity=1)

Draw a textured Phong-shaded 2d triangle, with perspective correction.

- template<typename tz, typename tc, typename tl >
CImg< T > & draw_triangle (CImg< tz > &zbuffer, const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const int x2, const int y2, const float z2, const CImg< tc > &texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const CImg< tl > &light, const int lx0, const int ly0, const int lx1, const int ly1, const int lx2, const int ly2, const float opacity=1)

Draw a textured Phong-shaded 2d triangle, with perspective correction and z-buffering.

- CImg< T > & draw_rectangle (const int x0, const int y0, const int z0, const int c0, const int x1, const int y1, const int z1, const int c1, const T val, const float opacity=1)

Draw a filled 4d rectangle.

- template<typename tc >
CImg< T > & draw_rectangle (const int x0, const int y0, const int z0, const int x1, const int y1, const int z1, const tc *const color, const float opacity=1)

Draw a filled 3d rectangle.

- template<typename tc >
CImg< T > & draw_rectangle (const int x0, const int y0, const int z0, const int x1, const int y1, const int z1, const tc *const color, const float opacity, const unsigned int pattern)

Draw an outlined 3d rectangle [overloading].

- template<typename tc >
CImg< T > & draw_rectangle (const int x0, const int y0, const int x1, const int y1, const tc *const color, const float opacity=1)

Draw a filled 2d rectangle.

- template<typename tc >
CImg< T > & draw_rectangle (const int x0, const int y0, const int x1, const int y1, const tc *const color, const float opacity, const unsigned int pattern)

Draw an outlined 2d rectangle [overloading].

- template<typename t, typename tc >
CImg< T > & draw_polygon (const CImg< t > &points, const tc *const color, const float opacity=1)

Draw a filled 2d polygon.

- template<typename t, typename tc >
CImg< T > & draw_polygon (const CImg< t > &points, const tc *const color, const float opacity, const unsigned int pattern)

Draw an outlined 2d polygon [overloading].

- template<typename tc >
CImg< T > & draw_ellipse (const int x0, const int y0, const float r1, const float r2, const float angle, const tc *const color, const float opacity=1)

Draw a filled 2d ellipse.

- template<typename t, typename tc >
CImg< T > & draw_ellipse (const int x0, const int y0, const CImg< t > &tensor, const tc *const color, const float opacity=1)

Draw a filled 2d ellipse [overloading].

- `template<typename tc >`
`Clmg< T > & draw_ellipse` (const int x0, const int y0, const float r1, const float r2, const float angle, const tc *const color, const float opacity, const unsigned int pattern)
Draw an outlined 2d ellipse.
- `template<typename t , typename tc >`
`Clmg< T > & draw_ellipse` (const int x0, const int y0, const `Clmg< t > &tensor`, const tc *const color, const float opacity, const unsigned int pattern)
*Draw an outlined 2d ellipse **[overloading]**.*
- `template<typename tc >`
`Clmg< T > & draw_circle` (const int x0, const int y0, int radius, const tc *const color, const float opacity=1)
Draw a filled 2d circle.
- `template<typename tc >`
`Clmg< T > & draw_circle` (const int x0, const int y0, int radius, const tc *const color, const float opacity, const unsigned int pattern)
Draw an outlined 2d circle.
- `template<typename t >`
`Clmg< T > & draw_image` (const int x0, const int y0, const int z0, const int c0, const `Clmg< t > &sprite`, const float opacity=1)
Draw an image.
- `Clmg< T > & draw_image` (const int x0, const int y0, const int z0, const int c0, const `Clmg< T > &sprite`, const float opacity=1)
*Draw an image **[specialization]**.*
- `template<typename t >`
`Clmg< T > & draw_image` (const int x0, const int y0, const int z0, const `Clmg< t > &sprite`, const float opacity=1)
*Draw an image **[overloading]**.*
- `template<typename t >`
`Clmg< T > & draw_image` (const int x0, const int y0, const `Clmg< t > &sprite`, const float opacity=1)
*Draw an image **[overloading]**.*
- `template<typename t >`
`Clmg< T > & draw_image` (const int x0, const `Clmg< t > &sprite`, const float opacity=1)
*Draw an image **[overloading]**.*
- `template<typename t >`
`Clmg< T > & draw_image` (const `Clmg< t > &sprite`, const float opacity=1)
*Draw an image **[overloading]**.*
- `template<typename ti , typename tm >`
`Clmg< T > & draw_image` (const int x0, const int y0, const int z0, const int c0, const `Clmg< ti > &sprite`, const `Clmg< tm > &mask`, const float opacity=1, const float mask_max_value=1)
Draw a masked image.
- `template<typename ti , typename tm >`
`Clmg< T > & draw_image` (const int x0, const int y0, const int z0, const `Clmg< ti > &sprite`, const `Clmg< tm > &mask`, const float opacity=1, const float mask_max_value=1)
*Draw a masked image **[overloading]**.*
- `template<typename ti , typename tm >`
`Clmg< T > & draw_image` (const int x0, const int y0, const `Clmg< ti > &sprite`, const `Clmg< tm > &mask`, const float opacity=1, const float mask_max_value=1)
*Draw a image **[overloading]**.*
- `template<typename ti , typename tm >`
`Clmg< T > & draw_image` (const int x0, const `Clmg< ti > &sprite`, const `Clmg< tm > &mask`, const float opacity=1, const float mask_max_value=1)
*Draw a image **[overloading]**.*
- `template<typename ti , typename tm >`
`Clmg< T > & draw_image` (const `Clmg< ti > &sprite`, const `Clmg< tm > &mask`, const float opacity=1, const float mask_max_value=1)
*Draw a image **[overloading]**.*

Draw an image.

- template<typename tc1 , typename tc2 , typename t >
Clmg< T > & **draw_text** (const int x0, const int y0, const char *const text, const tc1 *const foreground_color, const tc2 *const background_color, const float opacity, const **ClmgList**< t > &font,...)

Draw a text string.

- template<typename tc , typename t >
Clmg< T > & **draw_text** (const int x0, const int y0, const char *const text, const tc *const foreground_color, const int, const float opacity, const **ClmgList**< t > &font,...)

Draw a text string [overloading].

- template<typename tc , typename t >
Clmg< T > & **draw_text** (const int x0, const int y0, const char *const text, const int, const tc *const background_color, const float opacity, const **ClmgList**< t > &font,...)

Draw a text string [overloading].

- template<typename tc1 , typename tc2 >
Clmg< T > & **draw_text** (const int x0, const int y0, const char *const text, const tc1 *const foreground_color, const tc2 *const background_color, const float opacity=1, const unsigned int font_height=13,...)

Draw a text string [overloading].

- template<typename tc >
Clmg< T > & **draw_text** (const int x0, const int y0, const char *const text, const tc *const foreground_color, const int background_color=0, const float opacity=1, const unsigned int font_height=13,...)

Draw a text string [overloading].

- template<typename tc >
Clmg< T > & **draw_text** (const int x0, const int y0, const char *const text, const int, const tc *const background_color, const float opacity=1, const unsigned int font_height=13,...)

Draw a text string [overloading].

- template<typename t1 , typename t2 >
Clmg< T > & **draw_quiver** (const **Clmg**< t1 > &flow, const t2 *const color, const float opacity=1, const unsigned int sampling=25, const float factor=-20, const bool is_arrow=true, const unsigned int pattern=~0U)

Draw a 2d vector field.

- template<typename t1 , typename t2 >
Clmg< T > & **draw_quiver** (const **Clmg**< t1 > &flow, const **Clmg**< t2 > &color, const float opacity=1, const unsigned int sampling=25, const float factor=-20, const bool is_arrow=true, const unsigned int pattern=~0U)

Draw a 2d vector field, using a field of colors.

- template<typename t , typename tc >
Clmg< T > & **draw_axis** (const **Clmg**< t > &values_x, const int y, const tc *const color, const float opacity=1, const unsigned int pattern=~0U, const unsigned int font_height=13, const bool allow_zero=true)

Draw a labeled horizontal axis.

- template<typename t , typename tc >
Clmg< T > & **draw_axis** (const int x, const **Clmg**< t > &values_y, const tc *const color, const float opacity=1, const unsigned int pattern=~0U, const unsigned int font_height=13, const bool allow_zero=true)

Draw a labeled vertical axis.

- template<typename tx , typename ty , typename tc >
Clmg< T > & **draw_axes** (const **Clmg**< tx > &values_x, const **Clmg**< ty > &values_y, const tc *const color, const float opacity=1, const unsigned int pattern_x=~0U, const unsigned int pattern_y=~0U, const unsigned int font_height=13, const bool allow_zero=true)

Draw labeled horizontal and vertical axes.

- template<typename tc >
Clmg< T > & **draw_axes** (const float x0, const float x1, const float y0, const float y1, const tc *const color, const float opacity=1, const int subdivisionx=-60, const int subdivisiony=-60, const float precisionx=0, const float precisiony=0, const unsigned int pattern_x=~0U, const unsigned int pattern_y=~0U, const unsigned int font_height=13)

Draw labeled horizontal and vertical axes [overloading].

- template<typename tx , typename ty , typename tc >
Clmg< T > & **draw_grid** (const **Clmg**< tx > &values_x, const **Clmg**< ty > &values_y, const tc *const color, const float opacity=1, const unsigned int pattern_x=~0U, const unsigned int pattern_y=~0U)

Draw 2d grid.

- `template<typename tc >`
`Clmg< T > & draw_grid (const float delta_x, const float delta_y, const float offsetx, const float offsety, const bool invertx, const bool inverty, const tc *const color, const float opacity=1, const unsigned int pattern_↵
x=~0U, const unsigned int pattern_y=~0U)`

Draw 2d grid [simplification].

- `template<typename t, typename tc >`
`Clmg< T > & draw_graph (const Clmg< t > &data, const tc *const color, const float opacity=1, const unsigned int plot_type=1, const int vertex_type=1, const double ymin=0, const double ymax=0, const unsigned int pattern=~0U)`

Draw 1d graph.

- `template<typename tc, typename t >`
`Clmg< T > & draw_fill (const int x, const int y, const int z, const tc *const color, const float opacity, Clmg< t > ®ion, const float sigma=0, const bool is_high_connexity=false)`

Draw filled 3d region with the flood fill algorithm.

- `template<typename tc >`
`Clmg< T > & draw_fill (const int x, const int y, const int z, const tc *const color, const float opacity=1, const float sigma=0, const bool is_high_connexity=false)`

Draw filled 3d region with the flood fill algorithm [simplification].

- `template<typename tc >`
`Clmg< T > & draw_fill (const int x, const int y, const tc *const color, const float opacity=1, const float sigma=0, const bool is_high_connexity=false)`

Draw filled 2d region with the flood fill algorithm [simplification].

- `Clmg< T > & draw_plasma (const float alpha=1, const float beta=0, const unsigned int scale=8)`

Draw a random plasma texture.

- `template<typename tc >`
`Clmg< T > & draw_mandelbrot (const int x0, const int y0, const int x1, const int y1, const Clmg< tc > &colormap, const float opacity=1, const double z0r=-2, const double z0i=-2, const double z1r=2, const double z1i=2, const unsigned int iteration_max=255, const bool is_normalized_iteration=false, const bool is_julia_↵
set=false, const double param_r=0, const double param_i=0)`

Draw a quadratic Mandelbrot or Julia 2d fractal.

- `template<typename tc >`
`Clmg< T > & draw_mandelbrot (const Clmg< tc > &colormap, const float opacity=1, const double z0r=-2, const double z0i=-2, const double z1r=2, const double z1i=2, const unsigned int iteration_max=255, const bool is_normalized_iteration=false, const bool is_julia_set=false, const double param_r=0, const double param_i=0)`

Draw a quadratic Mandelbrot or Julia 2d fractal [overloading].

- `template<typename tc >`
`Clmg< T > & draw_gaussian (const float xc, const float sigma, const tc *const color, const float opacity=1)`

Draw a 1d gaussian function.

- `template<typename t, typename tc >`
`Clmg< T > & draw_gaussian (const float xc, const float yc, const Clmg< t > &tensor, const tc *const color, const float opacity=1)`

Draw a 2d gaussian function.

- `template<typename tc >`
`Clmg< T > & draw_gaussian (const int xc, const int yc, const float r1, const float r2, const float ru, const float rv, const tc *const color, const float opacity=1)`

Draw a 2d gaussian function [overloading].

- `template<typename tc >`
`Clmg< T > & draw_gaussian (const float xc, const float yc, const float sigma, const tc *const color, const float opacity=1)`

Draw a 2d gaussian function [overloading].

- `template<typename t, typename tc >`
`Clmg< T > & draw_gaussian (const float xc, const float yc, const float zc, const Clmg< t > &tensor, const tc *const color, const float opacity=1)`

Draw a 3d gaussian function [overloading].

- `template<typename tc >`
`Clmg< T > & draw_gaussian` (const float xc, const float yc, const float zc, const float sigma, const tc *const color, const float opacity=1)

Draw a 3d gaussian function [overloading].

- `template<typename tp , typename tf , typename tc , typename to >`
`Clmg< T > & draw_object3d` (const float x0, const float y0, const float z0, const `Clmg< tp >` &vertices, const `ClmgList< tf >` &primitives, const `ClmgList< tc >` &colors, const `Clmg< to >` &opacities, const unsigned int render_type=4, const bool is_double_sided=false, const float focale=700, const float lightx=0, const float lighty=0, const float lightz=-5e8, const float specular_lightness=0.2f, const float specular_shininess=0.1f)

Draw a 3d object.

- `template<typename tp , typename tf , typename tc , typename to , typename tz >`
`Clmg< T > & draw_object3d` (const float x0, const float y0, const float z0, const `Clmg< tp >` &vertices, const `ClmgList< tf >` &primitives, const `ClmgList< tc >` &colors, const `Clmg< to >` &opacities, const unsigned int render_type, const bool is_double_sided, const float focale, const float lightx, const float lighty, const float lightz, const float specular_lightness, const float specular_shininess, `Clmg< tz >` &zbuffer)

Draw a 3d object [simplification].

- `template<typename tp , typename tf , typename tc , typename to >`
`Clmg< T > & draw_object3d` (const float x0, const float y0, const float z0, const `Clmg< tp >` &vertices, const `ClmgList< tf >` &primitives, const `ClmgList< tc >` &colors, const `ClmgList< to >` &opacities, const unsigned int render_type=4, const bool is_double_sided=false, const float focale=700, const float lightx=0, const float lighty=0, const float lightz=-5e8, const float specular_lightness=0.2f, const float specular_shininess=0.1f)

Draw a 3d object [simplification].

- `template<typename tp , typename tf , typename tc , typename to , typename tz >`
`Clmg< T > & draw_object3d` (const float x0, const float y0, const float z0, const `Clmg< tp >` &vertices, const `ClmgList< tf >` &primitives, const `ClmgList< tc >` &colors, const `ClmgList< to >` &opacities, const unsigned int render_type, const bool is_double_sided, const float focale, const float lightx, const float lighty, const float lightz, const float specular_lightness, const float specular_shininess, `Clmg< tz >` &zbuffer)

Draw a 3d object [simplification].

- `template<typename tp , typename tf , typename tc >`
`Clmg< T > & draw_object3d` (const float x0, const float y0, const float z0, const `Clmg< tp >` &vertices, const `ClmgList< tf >` &primitives, const `ClmgList< tc >` &colors, const unsigned int render_type=4, const bool is_double_sided=false, const float focale=700, const float lightx=0, const float lighty=0, const float lightz=-5e8, const float specular_lightness=0.2f, const float specular_shininess=0.1f)

Draw a 3d object [simplification].

- `template<typename tp , typename tf , typename tc , typename tz >`
`Clmg< T > & draw_object3d` (const float x0, const float y0, const float z0, const `Clmg< tp >` &vertices, const `ClmgList< tf >` &primitives, const `ClmgList< tc >` &colors, const unsigned int render_type, const bool is_double_sided, const float focale, const float lightx, const float lighty, const float lightz, const float specular_lightness, const float specular_shininess, `Clmg< tz >` &zbuffer)

Draw a 3d object [simplification].

Data Input

- `Clmg< T > & select` (`ClmgDisplay` &disp, const unsigned int feature_type=2, unsigned int *const XYZ=0, const bool exit_on_anykey=false)

Launch simple interface to select a shape from an image.

- `Clmg< T > & select` (const char *const title, const unsigned int feature_type=2, unsigned int *const XYZ=0, const bool exit_on_anykey=false)

Simple interface to select a shape from an image [overloading].

- `Clmg< intT > get_select` (`ClmgDisplay` &disp, const unsigned int feature_type=2, unsigned int *const XYZ=0, const bool exit_on_anykey=false) const

Simple interface to select a shape from an image [new-instance version].

- `Clmg< intT > get_select` (const char *const title, const unsigned int feature_type=2, unsigned int *const XYZ=0, const bool exit_on_anykey=false) const
Simple interface to select a shape from an image [new-instance version].
- `Clmg< intT > get_select_graph` (ClmgDisplay &disp, const unsigned int plot_type=1, const unsigned int vertex_type=1, const char *const labelx=0, const double xmin=0, const double xmax=0, const char *const labely=0, const double ymin=0, const double ymax=0, const bool exit_on_anykey=false) const
Select sub-graph in a graph.
- `Clmg< T > & load` (const char *const filename)
Load image from a file.
- `Clmg< T > & load_ascii` (const char *const filename)
Load image from an ascii file.
- `Clmg< T > & load_ascii` (std::FILE *const file)
Load image from an ascii file [overloading].
- `Clmg< T > & load_dlm` (const char *const filename)
Load image from a DLM file.
- `Clmg< T > & load_dlm` (std::FILE *const file)
Load image from a DLM file [overloading].
- `Clmg< T > & load_bmp` (const char *const filename)
Load image from a BMP file.
- `Clmg< T > & load_bmp` (std::FILE *const file)
Load image from a BMP file [overloading].
- `Clmg< T > & load_jpeg` (const char *const filename)
Load image from a JPEG file.
- `Clmg< T > & load_jpeg` (std::FILE *const file)
Load image from a JPEG file [overloading].
- `Clmg< T > & load_magick` (const char *const filename)
Load image from a file, using Magick++ library.
- `Clmg< T > & load_png` (const char *const filename)
Load image from a PNG file.
- `Clmg< T > & load_png` (std::FILE *const file)
Load image from a PNG file [overloading].
- `Clmg< T > & load_pnm` (const char *const filename)
Load image from a PNM file.
- `Clmg< T > & load_pnm` (std::FILE *const file)
Load image from a PNM file [overloading].
- `Clmg< T > & load_pfm` (const char *const filename)
Load image from a PFM file.
- `Clmg< T > & load_pfm` (std::FILE *const file)
Load image from a PFM file [overloading].
- `Clmg< T > & load_rgb` (const char *const filename, const unsigned int dimw, const unsigned int dimh=1)
Load image from a RGB file.
- `Clmg< T > & load_rgb` (std::FILE *const file, const unsigned int dimw, const unsigned int dimh=1)
Load image from a RGB file [overloading].
- `Clmg< T > & load_rgba` (const char *const filename, const unsigned int dimw, const unsigned int dimh=1)
Load image from a RGBA file.
- `Clmg< T > & load_rgba` (std::FILE *const file, const unsigned int dimw, const unsigned int dimh=1)
Load image from a RGBA file [overloading].
- `Clmg< T > & load_tiff` (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, float *const voxel_size=0, Clmg< charT > *const description=0)
Load image from a TIFF file.

- `Clmg< T > & load_minc2` (const char *const filename)
Load image from a MINC2 file.
- `Clmg< T > & load_analyze` (const char *const filename, float *const voxel_size=0)
Load image from an ANALYZE7.5/NIFTI file.
- `Clmg< T > & load_analyze` (std::FILE *const file, float *const voxel_size=0)
Load image from an ANALYZE7.5/NIFTI file [overloading].
- `Clmg< T > & load_cimg` (const char *const filename, const char axis='z', const float align=0)
Load image from a .cimg[z] file.
- `Clmg< T > & load_cimg` (std::FILE *const file, const char axis='z', const float align=0)
Load image from a .cimg[z] file [overloading].
- `Clmg< T > & load_cimg` (const char *const filename, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1, const char axis='z', const float align=0)
Load sub-images of a .cimg file.
- `Clmg< T > & load_cimg` (std::FILE *const file, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1, const char axis='z', const float align=0)
Load sub-images of a .cimg file [overloading].
- `Clmg< T > & load_inr` (const char *const filename, float *const voxel_size=0)
Load image from an INRIMAGE-4 file.
- `Clmg< T > & load_inr` (std::FILE *const file, float *const voxel_size=0)
Load image from an INRIMAGE-4 file [overloading].
- `Clmg< T > & load_exr` (const char *const filename)
Load image from a EXR file.
- `Clmg< T > & load_pandore` (const char *const filename)
Load image from a PANDORE-5 file.
- `Clmg< T > & load_pandore` (std::FILE *const file)
Load image from a PANDORE-5 file [overloading].
- `Clmg< T > & load_parrec` (const char *const filename, const char axis='c', const float align=0)
Load image from a PAR-REC (Philips) file.
- `Clmg< T > & load_raw` (const char *const filename, const unsigned int size_x=0, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1, const bool is_multiplexed=false, const bool invert_endianness=false, const alongT offset=0)
Load image from a raw binary file.
- `Clmg< T > & load_raw` (std::FILE *const file, const unsigned int size_x=0, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1, const bool is_multiplexed=false, const bool invert_endianness=false, const alongT offset=0)
Load image from a raw binary file [overloading].
- `Clmg< T > & load_yuv` (const char *const filename, const unsigned int size_x, const unsigned int size_y=1, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true, const char axis='z')
Load image sequence from a YUV file.
- `Clmg< T > & load_yuv` (std::FILE *const file, const unsigned int size_x, const unsigned int size_y=1, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true, const char axis='z')
Load image sequence from a YUV file [overloading].
- `template<typename tf, typename tc>`
`Clmg< T > & load_off` (ClmgList< tf > &primitives, ClmgList< tc > &colors, const char *const filename)
Load 3d object from a .OFF file.
- `template<typename tf, typename tc>`
`Clmg< T > & load_off` (ClmgList< tf > &primitives, ClmgList< tc > &colors, std::FILE *const file)
Load 3d object from a .OFF file [overloading].

- [CImg< T > & load_video](#) (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const char axis='z', const float align=0)
Load image sequence from a video file, using OpenCV library.
- [CImg< T > & load_ffmpeg_external](#) (const char *const filename, const char axis='z', const float align=0)
Load image sequence using FFmpeg's external tool 'ffmpeg'.
- [CImg< T > & load_gif_external](#) (const char *const filename, const char axis='z', const float align=0)
Load gif file, using ImageMagick or GraphicsMagick's external tools.
- [CImg< T > & load_graphicmagick_external](#) (const char *const filename)
Load image using GraphicsMagick's external tool 'gm'.
- [CImg< T > & load_gzip_external](#) (const char *const filename)
Load gzipped image file, using external tool 'gunzip'.
- [CImg< T > & load_imagemagick_external](#) (const char *const filename)
Load image using ImageMagick's external tool 'convert'.
- [CImg< T > & load_medcon_external](#) (const char *const filename)
Load image from a DICOM file, using XMedcon's external tool 'medcon'.
- [CImg< T > & load_dcraw_external](#) (const char *const filename)
Load image from a RAW Color Camera file, using external tool 'dcraw'.
- [CImg< T > & load_camera](#) (const unsigned int camera_index=0, const unsigned int skip_frames=0, const bool release_camera=true, const unsigned int capture_width=0, const unsigned int capture_height=0)
Load image from a camera stream, using OpenCV.
- [CImg< T > & load_other](#) (const char *const filename)
Load image using various non-native ways.
- static [CImg< T > get_load](#) (const char *const filename)
*Load image from a file **[new-instance version]**.*
- static [CImg< T > get_load_ascii](#) (const char *const filename)
*Load image from an ascii file **[in-place version]**.*
- static [CImg< T > get_load_ascii](#) (std::FILE *const file)
*Load image from an ascii file **[new-instance version]**.*
- static [CImg< T > get_load_dlm](#) (const char *const filename)
*Load image from a DLM file **[new-instance version]**.*
- static [CImg< T > get_load_dlm](#) (std::FILE *const file)
*Load image from a DLM file **[new-instance version]**.*
- static [CImg< T > get_load_bmp](#) (const char *const filename)
*Load image from a BMP file **[new-instance version]**.*
- static [CImg< T > get_load_bmp](#) (std::FILE *const file)
*Load image from a BMP file **[new-instance version]**.*
- static [CImg< T > get_load_jpeg](#) (const char *const filename)
*Load image from a JPEG file **[new-instance version]**.*
- static [CImg< T > get_load_jpeg](#) (std::FILE *const file)
*Load image from a JPEG file **[new-instance version]**.*
- static [CImg< T > get_load_magick](#) (const char *const filename)
*Load image from a file, using Magick++ library **[new-instance version]**.*
- static [CImg< T > get_load_png](#) (const char *const filename)
*Load image from a PNG file **[new-instance version]**.*
- static [CImg< T > get_load_png](#) (std::FILE *const file)
*Load image from a PNG file **[new-instance version]**.*
- static [CImg< T > get_load_pnm](#) (const char *const filename)
*Load image from a PNM file **[new-instance version]**.*
- static [CImg< T > get_load_pnm](#) (std::FILE *const file)
*Load image from a PNM file **[new-instance version]**.*
- static [CImg< T > get_load_pfm](#) (const char *const filename)

- Load image from a PFM file [new-instance version].*
 - static CImg< T > [get_load_pfm](#) (std::FILE *const file)
- Load image from a PFM file [new-instance version].*
 - static CImg< T > [get_load_rgb](#) (const char *const filename, const unsigned int dimw, const unsigned int dimh=1)
- Load image from a RGB file [new-instance version].*
 - static CImg< T > [get_load_rgb](#) (std::FILE *const file, const unsigned int dimw, const unsigned int dimh=1)
- Load image from a RGB file [new-instance version].*
 - static CImg< T > [get_load_rgba](#) (const char *const filename, const unsigned int dimw, const unsigned int dimh=1)
- Load image from a RGBA file [new-instance version].*
 - static CImg< T > [get_load_rgba](#) (std::FILE *const file, const unsigned int dimw, const unsigned int dimh=1)
- Load image from a RGBA file [new-instance version].*
 - static CImg< T > [get_load_tiff](#) (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, float *const voxel_size=0, CImg< charT > *const description=0)
- Load image from a TIFF file [new-instance version].*
 - static CImg< T > [get_load_minc2](#) (const char *const filename)
- Load image from a MINC2 file [new-instance version].*
 - static CImg< T > [get_load_analyze](#) (const char *const filename, float *const voxel_size=0)
- Load image from an ANALYZE7.5/NIFTI file [new-instance version].*
 - static CImg< T > [get_load_analyze](#) (std::FILE *const file, float *const voxel_size=0)
- Load image from an ANALYZE7.5/NIFTI file [new-instance version].*
 - static CImg< T > [get_load_cimg](#) (const char *const filename, const char axis='z', const float align=0)
- Load image from a .cimg[z] file [new-instance version]*
 - static CImg< T > [get_load_cimg](#) (std::FILE *const file, const char axis='z', const float align=0)
- Load image from a .cimg[z] file [new-instance version]*
 - static CImg< T > [get_load_cimg](#) (const char *const filename, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1, const char axis='z', const float align=0)
- Load sub-images of a .cimg file [new-instance version].*
 - static CImg< T > [get_load_cimg](#) (std::FILE *const file, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1, const char axis='z', const float align=0)
- Load sub-images of a .cimg file [new-instance version].*
 - static CImg< T > [get_load_inr](#) (const char *const filename, float *const voxel_size=0)
- Load image from an INRIMAGE-4 file [new-instance version].*
 - static CImg< T > [get_load_inr](#) (std::FILE *const file, float *const voxel_size=0)
- Load image from an INRIMAGE-4 file [new-instance version].*
 - static CImg< T > [get_load_exr](#) (const char *const filename)
- Load image from a EXR file [new-instance version].*
 - static CImg< T > [get_load_pandore](#) (const char *const filename)
- Load image from a PANDORE-5 file [new-instance version].*
 - static CImg< T > [get_load_pandore](#) (std::FILE *const file)
- Load image from a PANDORE-5 file [new-instance version].*
 - static CImg< T > [get_load_parrec](#) (const char *const filename, const char axis='c', const float align=0)
- Load image from a PAR-REC (Philips) file [new-instance version].*
 - static CImg< T > [get_load_raw](#) (const char *const filename, const unsigned int size_x=0, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1, const bool is_multiplexed=false, const bool invert_endianness=false, const unsigned longT offset=0)
- Load image from a raw binary file [new-instance version].*

- static `Clmg< T > get_load_raw` (std::FILE *const file, const unsigned int size_x=0, const unsigned int size_y=1, const unsigned int size_z=1, const unsigned int size_c=1, const bool is_multiplexed=false, const bool invert_endianness=false, const ulongT offset=0)
Load image from a raw binary file [new-instance version].
- static `Clmg< T > get_load_yuv` (const char *const filename, const unsigned int size_x, const unsigned int size_y=1, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true, const char axis='z')
Load image sequence from a YUV file [new-instance version].
- static `Clmg< T > get_load_yuv` (std::FILE *const file, const unsigned int size_x, const unsigned int size_y=1, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true, const char axis='z')
Load image sequence from a YUV file [new-instance version].
- template<typename tf, typename tc >
static `Clmg< T > get_load_off` (ClmgList< tf > &primitives, ClmgList< tc > &colors, const char *const filename)
Load 3d object from a .OFF file [new-instance version].
- template<typename tf, typename tc >
static `Clmg< T > get_load_off` (ClmgList< tf > &primitives, ClmgList< tc > &colors, std::FILE *const file)
Load 3d object from a .OFF file [new-instance version].
- static `Clmg< T > get_load_video` (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const char axis='z', const float align=0)
Load image sequence from a video file, using OpenCV library [new-instance version].
- static `Clmg< T > get_load_ffmpeg_external` (const char *const filename, const char axis='z', const float align=0)
Load image sequence using FFmpeg's external tool 'ffmpeg' [new-instance version].
- static `Clmg< T > get_load_gif_external` (const char *const filename, const char axis='z', const float align=0)
Load gif file, using ImageMagick or GraphicsMagick's external tool 'convert' [new-instance version].
- static `Clmg< T > get_load_graphicmagick_external` (const char *const filename)
Load image using GraphicsMagick's external tool 'gm' [new-instance version].
- static `Clmg< T > get_load_gzip_external` (const char *const filename)
Load gzipped image file, using external tool 'gunzip' [new-instance version].
- static `Clmg< T > get_load_imagemagick_external` (const char *const filename)
Load image using ImageMagick's external tool 'convert' [new-instance version].
- static `Clmg< T > get_load_medcon_external` (const char *const filename)
Load image from a DICOM file, using XMedcon's external tool 'medcon' [new-instance version].
- static `Clmg< T > get_load_dcraw_external` (const char *const filename)
Load image from a RAW Color Camera file, using external tool 'dcraw' [new-instance version].
- static `Clmg< T > get_load_camera` (const unsigned int camera_index=0, const unsigned int skip_frames=0, const bool release_camera=true, const unsigned int capture_width=0, const unsigned int capture_height=0)
Load image from a camera stream, using OpenCV [new-instance version].
- static `Clmg< T > get_load_other` (const char *const filename)
Load image using various non-native ways [new-instance version].

Data Output

- const `Clmg< T > & print` (const char *const title=0, const bool display_stats=true) const
Display information about the image data.
- const `Clmg< T > & display` (ClmgDisplay &disp) const
Display image into a ClmgDisplay window.
- const `Clmg< T > & display` (ClmgDisplay &disp, const bool display_info, unsigned int *const XYZ=0, const bool exit_on_anykey=false) const
Display image into a ClmgDisplay window, in an interactive way.

- const CImg< T > & display (const char *const title=0, const bool display_info=true, unsigned int *const XYZ=0, const bool exit_on_anykey=false) const

Display image into an interactive window.

- template<typename tp, typename tf, typename tc, typename to >
const CImg< T > & display_object3d (CImgDisplay &disp, const CImg< tp > &vertices, const CImgList< tf > &primitives, const CImgList< tc > &colors, const to &opacities, const bool centering=true, const int render_static=4, const int render_motion=1, const bool is_double_sided=true, const float focale=700, const float light_x=0, const float light_y=0, const float light_z=-5e8f, const float specular_lightness=0.2f, const float specular_shininess=0.1f, const bool display_axes=true, float *const pose_matrix=0, const bool exit_on_↵ anykey=false) const

Display object 3d in an interactive window.

- template<typename tp, typename tf, typename tc, typename to >
const CImg< T > & display_object3d (const char *const title, const CImg< tp > &vertices, const CImgList< tf > &primitives, const CImgList< tc > &colors, const to &opacities, const bool centering=true, const int render_static=4, const int render_motion=1, const bool is_double_sided=true, const float focale=700, const float light_x=0, const float light_y=0, const float light_z=-5e8f, const float specular_lightness=0.2f, const float specular_shininess=0.1f, const bool display_axes=true, float *const pose_matrix=0, const bool exit_on_↵ anykey=false) const

Display object 3d in an interactive window [simplification].

- template<typename tp, typename tf, typename tc >
const CImg< T > & display_object3d (CImgDisplay &disp, const CImg< tp > &vertices, const CImgList< tf > &primitives, const CImgList< tc > &colors, const bool centering=true, const int render_static=4, const int render_motion=1, const bool is_double_sided=true, const float focale=700, const float light_x=0, const float light_y=0, const float light_z=-5e8f, const float specular_lightness=0.2f, const float specular_shininess=0.1f, const bool display_axes=true, float *const pose_matrix=0, const bool exit_on_anykey=false) const

Display object 3d in an interactive window [simplification].

- template<typename tp, typename tf, typename tc >
const CImg< T > & display_object3d (const char *const title, const CImg< tp > &vertices, const CImgList< tf > &primitives, const CImgList< tc > &colors, const bool centering=true, const int render_static=4, const int render_motion=1, const bool is_double_sided=true, const float focale=700, const float light_x=0, const float light_y=0, const float light_z=-5e8f, const float specular_lightness=0.2f, const float specular_shininess=0.1f, const bool display_axes=true, float *const pose_matrix=0, const bool exit_on_anykey=false) const

Display object 3d in an interactive window [simplification].

- template<typename tp, typename tf >
const CImg< T > & display_object3d (CImgDisplay &disp, const CImg< tp > &vertices, const CImgList< tf > &primitives, const bool centering=true, const int render_static=4, const int render_motion=1, const bool is_double_sided=true, const float focale=700, const float light_x=0, const float light_y=0, const float light_z=-5e8f, const float specular_lightness=0.2f, const float specular_shininess=0.1f, const bool display_axes=true, float *const pose_matrix=0, const bool exit_on_anykey=false) const

Display object 3d in an interactive window [simplification].

- template<typename tp, typename tf >
const CImg< T > & display_object3d (const char *const title, const CImg< tp > &vertices, const CImgList< tf > &primitives, const bool centering=true, const int render_static=4, const int render_motion=1, const bool is_double_sided=true, const float focale=700, const float light_x=0, const float light_y=0, const float light_z=-5e8f, const float specular_lightness=0.2f, const float specular_shininess=0.1f, const bool display_axes=true, float *const pose_matrix=0, const bool exit_on_anykey=false) const

Display object 3d in an interactive window [simplification].

- template<typename tp >
const CImg< T > & display_object3d (CImgDisplay &disp, const CImg< tp > &vertices, const bool centering=true, const int render_static=4, const int render_motion=1, const bool is_double_sided=true, const float focale=700, const float light_x=0, const float light_y=0, const float light_z=-5e8f, const float specular_↵_lightness=0.2f, const float specular_shininess=0.1f, const bool display_axes=true, float *const pose_↵_matrix=0, const bool exit_on_anykey=false) const

Display object 3d in an interactive window [simplification].

- `template<typename tp >`
`const Clmg< T > & display_object3d` (const char *const title, const Clmg< tp > &vertices, const bool centering=true, const int render_static=4, const int render_motion=1, const bool is_double_sided=true, const float focale=700, const float light_x=0, const float light_y=0, const float light_z=-5e8f, const float specular↵_lightness=0.2f, const float specular_shininess=0.1f, const bool display_axes=true, float *const pose↵_matrix=0, const bool exit_on_anykey=false) const
Display object 3d in an interactive window [simplification].
- `const Clmg< T > & display_graph` (ClmgDisplay &disp, const unsigned int plot_type=1, const unsigned int vertex_type=1, const char *const labelx=0, const double xmin=0, const double xmax=0, const char *const labely=0, const double ymin=0, const double ymax=0, const bool exit_on_anykey=false) const
Display 1d graph in an interactive window.
- `const Clmg< T > & display_graph` (const char *const title=0, const unsigned int plot_type=1, const unsigned int vertex_type=1, const char *const labelx=0, const double xmin=0, const double xmax=0, const char *const labely=0, const double ymin=0, const double ymax=0, const bool exit_on_anykey=false) const
Display 1d graph in an interactive window [overloading].
- `const Clmg< T > & save` (const char *const filename, const int number=-1, const unsigned int digits=6) const
Save image as a file.
- `const Clmg< T > & save_ascii` (const char *const filename) const
Save image as an ascii file.
- `const Clmg< T > & save_ascii` (std::FILE *const file) const
Save image as an ascii file [overloading].
- `const Clmg< T > & save_cpp` (const char *const filename) const
Save image as a .cpp source file.
- `const Clmg< T > & save_cpp` (std::FILE *const file) const
Save image as a .cpp source file [overloading].
- `const Clmg< T > & save_dlm` (const char *const filename) const
Save image as a DLM file.
- `const Clmg< T > & save_dlm` (std::FILE *const file) const
Save image as a DLM file [overloading].
- `const Clmg< T > & save_bmp` (const char *const filename) const
Save image as a BMP file.
- `const Clmg< T > & save_bmp` (std::FILE *const file) const
Save image as a BMP file [overloading].
- `const Clmg< T > & save_jpeg` (const char *const filename, const unsigned int quality=100) const
Save image as a JPEG file.
- `const Clmg< T > & save_jpeg` (std::FILE *const file, const unsigned int quality=100) const
Save image as a JPEG file [overloading].
- `const Clmg< T > & save_magick` (const char *const filename, const unsigned int bytes_per_pixel=0) const
Save image, using built-in ImageMagick++ library.
- `const Clmg< T > & save_png` (const char *const filename, const unsigned int bytes_per_pixel=0) const
Save image as a PNG file.
- `const Clmg< T > & save_png` (std::FILE *const file, const unsigned int bytes_per_pixel=0) const
Save image as a PNG file [overloading].
- `const Clmg< T > & save_pnm` (const char *const filename, const unsigned int bytes_per_pixel=0) const
Save image as a PNM file.
- `const Clmg< T > & save_pnm` (std::FILE *const file, const unsigned int bytes_per_pixel=0) const
Save image as a PNM file [overloading].
- `const Clmg< T > & save_pnk` (const char *const filename) const
Save image as a PNK file.
- `const Clmg< T > & save_pnk` (std::FILE *const file) const
Save image as a PNK file [overloading].
- `const Clmg< T > & save_pfm` (const char *const filename) const

- Save image as a PFM file.*

 - const CImg< T > & [save_pfm](#) (std::FILE *const file) const
- Save image as a PFM file **[overloading]**.*

 - const CImg< T > & [save_rgb](#) (const char *const filename) const
- Save image as a RGB file.*

 - const CImg< T > & [save_rgb](#) (std::FILE *const file) const
- Save image as a RGB file **[overloading]**.*

 - const CImg< T > & [save_rgba](#) (const char *const filename) const
- Save image as a RGBA file.*

 - const CImg< T > & [save_rgba](#) (std::FILE *const file) const
- Save image as a RGBA file **[overloading]**.*

 - const CImg< T > & [save_tiff](#) (const char *const filename, const unsigned int compression_type=0, const float *const voxel_size=0, const char *const description=0, const bool use_bigtiff=true) const
- Save image as a TIFF file.*

 - const CImg< T > & [save_minc2](#) (const char *const filename, const char *const imitate_file=0) const
- Save image as a MINC2 file.*

 - const CImg< T > & [save_analyze](#) (const char *const filename, const float *const voxel_size=0) const
- Save image as an ANALYZE7.5 or NIFTI file.*

 - const CImg< T > & [save_cimg](#) (const char *const filename, const bool is_compressed=false) const
- Save image as a .cimg file.*

 - const CImg< T > & [save_cimg](#) (std::FILE *const file, const bool is_compressed=false) const
- Save image as a .cimg file **[overloading]**.*

 - const CImg< T > & [save_cimg](#) (const char *const filename, const unsigned int n0, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0) const
- Save image as a sub-image into an existing .cimg file.*

 - const CImg< T > & [save_cimg](#) (std::FILE *const file, const unsigned int n0, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0) const
- Save image as a sub-image into an existing .cimg file **[overloading]**.*

 - const CImg< T > & [save_inr](#) (const char *const filename, const float *const voxel_size=0) const
- Save image as an INRIMAGE-4 file.*

 - const CImg< T > & [save_inr](#) (std::FILE *const file, const float *const voxel_size=0) const
- Save image as an INRIMAGE-4 file **[overloading]**.*

 - const CImg< T > & [save_exr](#) (const char *const filename) const
- Save image as an OpenEXR file.*

 - const CImg< T > & [save_pandore](#) (const char *const filename, const unsigned int colorspace=0) const
- Save image as a Pandore-5 file.*

 - const CImg< T > & [save_pandore](#) (std::FILE *const file, const unsigned int colorspace=0) const
- Save image as a Pandore-5 file **[overloading]**.*

 - const CImg< T > & [save_raw](#) (const char *const filename, const bool is_multiplexed=false) const
- Save image as a raw data file.*

 - const CImg< T > & [save_raw](#) (std::FILE *const file, const bool is_multiplexed=false) const
- Save image as a raw data file **[overloading]**.*

 - const CImg< T > & [save_yuv](#) (const char *const filename, const bool is_rgb=true) const
- Save image as a .yuv video file.*

 - const CImg< T > & [save_yuv](#) (std::FILE *const file, const bool is_rgb=true) const
- Save image as a .yuv video file **[overloading]**.*

 - template<typename tf, typename tc>
const CImg< T > & [save_off](#) (const CImgList< tf > &primitives, const CImgList< tc > &colors, const char *const filename) const
- Save 3d object as an Object File Format (.off) file.*

- `template<typename tf, typename tc >`
`const CImg< T > & save_off (const CImgList< tf > &primitives, const CImgList< tc > &colors, std::FILE *const file) const`
Save 3d object as an Object File Format (.off) file [overloading].
- `const CImg< T > & save_video (const char *const filename, const unsigned int fps=25, const char *codec=0, const bool keep_open=false) const`
Save volumetric image as a video, using the OpenCV library.
- `const CImg< T > & save_ffmpeg_external (const char *const filename, const unsigned int fps=25, const char *const codec=0, const unsigned int bitrate=2048) const`
Save volumetric image as a video, using ffmpeg external binary.
- `const CImg< T > & save_gzip_external (const char *const filename) const`
Save image using gzip external binary.
- `const CImg< T > & save_graphicmagick_external (const char *const filename, const unsigned int quality=100) const`
Save image using GraphicsMagick's external binary.
- `const CImg< T > & save_imagemagick_external (const char *const filename, const unsigned int quality=100) const`
Save image using ImageMagick's external binary.
- `const CImg< T > & save_medcon_external (const char *const filename) const`
Save image as a Dicom file.
- `const CImg< T > & save_other (const char *const filename, const unsigned int quality=100) const`
- `CImg< ucharT > get_serialize (const bool is_compressed=false) const`
Serialize a CImg<T> instance into a raw CImg<unsigned char> buffer.
- `static void save_empty_cimg (const char *const filename, const unsigned int dx, const unsigned int dy=1, const unsigned int dz=1, const unsigned int dc=1)`
Save blank image as a .cimg file.
- `static void save_empty_cimg (std::FILE *const file, const unsigned int dx, const unsigned int dy=1, const unsigned int dz=1, const unsigned int dc=1)`
Save blank image as a .cimg file [overloading].

8.1.1 Detailed Description

```
template<typename T>
struct cimg_library::CImg< T >
```

Class representing an image (up to 4 dimensions wide), each pixel being of type `T`.

This is the main class of the CImg Library. It declares and constructs an image, allows access to its pixel values, and is able to perform various image operations.

Image representation

A CImg image is defined as an instance of the container `CImg<T>`, which contains a regular grid of pixels, each pixel value being of type `T`. The image grid can have up to 4 dimensions: width, height, depth and number of channels. Usually, the three first dimensions are used to describe spatial coordinates (x, y, z) , while the number of channels is rather used as a vector-valued dimension (it may describe the R,G,B color channels for instance). If you need a fifth dimension, you can use image lists `CImgList<T>` rather than simple images `CImg<T>`.

Thus, the `CImg<T>` class is able to represent volumetric images of vector-valued pixels, as well as images with less dimensions (1d scalar signal, 2d color images, ...). Most member functions of the class `CImg<T>` are designed to handle this maximum case of (3+1) dimensions.

Concerning the pixel value type `T`: fully supported template types are the basic C++ types: `unsigned char`, `char`, `short`, `unsigned int`, `int`, `unsigned long`, `long`, `float`, `double`, Typically, fast image display can be done using `CImg<unsigned char>` images, while complex image processing algorithms may be rather coded using `CImg<float>` or `CImg<double>` images that have floating-point pixel values. The default value for the template `T` is `float`. Using your own template types may be possible. However, you will certainly have to define the complete set of arithmetic and logical operators for your class.

Image structure

The `CImg<T>` structure contains *six* fields:

- `_width` defines the number of *columns* of the image (size along the X-axis).
- `_height` defines the number of *rows* of the image (size along the Y-axis).
- `_depth` defines the number of *slices* of the image (size along the Z-axis).
- `_spectrum` defines the number of *channels* of the image (size along the C-axis).
- `_data` defines a *pointer* to the *pixel data* (of type `T`).
- `_is_shared` is a boolean that tells if the memory buffer `data` is shared with another image.

You can access these fields publicly although it is recommended to use the dedicated functions `width()`, `height()`, `depth()`, `spectrum()` and `ptr()` to do so. Image dimensions are not limited to a specific range (as long as you got enough available memory). A value of `1` usually means that the corresponding dimension is *flat*. If one of the dimensions is `0`, or if the data pointer is null, the image is considered as *empty*. Empty images should not contain any pixel data and thus, will not be processed by `CImg` member functions (a `CImgInstanceException` will be thrown instead). Pixel data are stored in memory, in a non interlaced mode (See [How pixel data are stored with CImg.](#)).

Image declaration and construction

Declaring an image can be done by using one of the several available constructors. Here is a list of the most used:

- Construct images from arbitrary dimensions:
 - `CImg<char> img;` declares an empty image.
 - `CImg<unsigned char> img(128,128);` declares a 128x128 greyscale image with unsigned char pixel values.
 - `CImg<double> img(3,3);` declares a 3x3 matrix with double coefficients.
 - `CImg<unsigned char> img(256,256,1,3);` declares a 256x256x1x3 (color) image (colors are stored as an image with three channels).
 - `CImg<double> img(128,128,128);` declares a 128x128x128 volumetric and greyscale image (with double pixel values).
 - `CImg<> img(128,128,128,3);` declares a 128x128x128 volumetric color image (with float pixels, which is the default value of the template parameter `T`).
 - **Note:** images pixels are **not automatically initialized to 0**. You may use the function `fill()` to do it, or use the specific constructor taking 5 parameters like this: `CImg<> img(128,128,128,3,0);` declares a 128x128x128 volumetric color image with all pixel values to 0.

- Construct images from filenames:
 - `CImg<unsigned char> img("image.jpg");` reads a JPEG color image from the file "image.jpg".
 - `CImg<float> img("analyze.hdr");` reads a volumetric image (ANALYZE7.5 format) from the file "analyze.hdr".
 - **Note:** You need to install [ImageMagick](#) to be able to read common compressed image formats (JPG,PNG, ...) (See [Files IO in CImg.](#)).
- Construct images from C-style arrays:
 - `CImg<int> img(data_buffer, 256, 256);` constructs a 256x256 greyscale image from a `int* buffer` data_buffer (of size 256x256=65536).
 - `CImg<unsigned char> img(data_buffer, 256, 256, 1, 3);` constructs a 256x256 color image from a `unsigned char* buffer` data_buffer (where R,G,B channels follow each others).

The complete list of constructors can be found [here](#).

Most useful functions

The `CImg<T>` class contains a lot of functions that operates on images. Some of the most useful are:

- [operator\(\)\(\)](#): Read or write pixel values.
- [display\(\)](#): displays the image in a new window.

8.1.2 Member Typedef Documentation

8.1.2.1 `typedef T* iterator`

Simple iterator type, to loop through each pixel value of an image instance.

Note

- The `CImg<T>::iterator` type is defined to be a `T*`.
- You will seldom have to use iterators in `CImg`, most classical operations being achieved (often in a faster way) using methods of `CImg<T>`.

Example

```
CImg<float> img("reference.jpg"); // Load image from file.
for (CImg<float>::iterator it = img.begin(), it<img.end(); ++it) *it = 0; // Set all
    pixels to '0', with a CImg iterator.
img.fill(0); // Do the same with a built-in
    method.
```

8.1.2.2 typedef const T* const_iterator

Simple const iterator type, to loop through each pixel value of a `const` image instance.

Note

- The `CImg<T>::const_iterator` type is defined to be a `const T*`.
- You will seldom have to use iterators in CImg, most classical operations being achieved (often in a faster way) using methods of `CImg<T>`.

Example

```
const CImg<float> img("reference.jpg");           // Load image from file.
float sum = 0;
for (CImg<float>::iterator it = img.begin(), it<img.end(); ++it) sum+=*it; // Compute
    sum of all pixel values, with a CImg iterator.
const float sum2 = img.sum();                     // Do the same with a built-in
    method.
```

8.1.2.3 typedef T value_type

Pixel value type.

Refer to the type of the pixel values of an image instance.

Note

- The `CImg<T>::value_type` type of a `CImg<T>` is defined to be a `T`.
- `CImg<T>::value_type` is actually not used in CImg methods. It has been mainly defined for compatibility with STL naming conventions.

8.1.3 Constructor & Destructor Documentation

8.1.3.1 ~CImg ()

Destroy image.

Note

- The pixel buffer `data()` is deallocated if necessary, e.g. for non-empty and non-shared image instances.
- Destroying an empty or shared image does nothing actually.

Warning

- When destroying a non-shared image, make sure that you will *not* operate on a remaining shared image that shares its buffer with the destroyed instance, in order to avoid further invalid memory access (to a deallocated buffer).

8.1.3.2 CImg ()

Construct empty image.

Note

- An empty image has no pixel data and all of its dimensions `width()`, `height()`, `depth()`, `spectrum()` are set to 0, as well as its pixel buffer pointer `data()`.
- An empty image may be re-assigned afterwards, e.g. with the family of `assign(unsigned int,unsigned int,unsigned int,unsigned int)` methods, or by `operator=(const CImg<t>&)`. In all cases, the type of pixels stays T.
- An empty image is never shared.

Example

```
CImg<float> img1, img2;           // Construct two empty images.
img1.assign(256,256,1,3);        // Re-assign 'img1' to be a 256x256x1x3 (color) image.
img2 = img1.get_rand(0,255);     // Re-assign 'img2' to be a random-valued version of 'img1'.
img2.assign();                   // Re-assign 'img2' to be an empty image again.
```

8.1.3.3 CImg (const unsigned int size_x, const unsigned int size_y = 1, const unsigned int size_z = 1, const unsigned int size_c = 1) [explicit]

Construct image with specified size.

Parameters

<code>size_x</code>	Image <code>width()</code> .
<code>size_y</code>	Image <code>height()</code> .
<code>size_z</code>	Image <code>depth()</code> .
<code>size_c</code>	Image <code>spectrum()</code> (number of channels).

Note

- It is able to create only *non-shared* images, and allocates thus a pixel buffer `data()` for each constructed image instance.
- Setting one dimension `size_x`, `size_y`, `size_z` or `size_c` to 0 leads to the construction of an *empty* image.
- A `CImgInstanceException` is thrown when the pixel buffer cannot be allocated (e.g. when requested size is too big for available memory).

Warning

- The allocated pixel buffer is *not* filled with a default value, and is likely to contain garbage values. In order to initialize pixel values during construction (e.g. with 0), use constructor `CImg(unsigned int,unsigned int,unsigned int,T)` instead.

Example

```
CImg<float> img1(256,256,1,3);    // Construct a 256x256x1x3 (color) image, filled with garbage values.
CImg<float> img2(256,256,1,3,0);  // Construct a 256x256x1x3 (color) image, filled with value '0'.
```

8.1.3.4 CImg (const unsigned int *size_x*, const unsigned int *size_y*, const unsigned int *size_z*, const unsigned int *size_c*, const T & *value*)

Construct image with specified size and initialize pixel values.

Parameters

<i>size_x</i>	Image width() .
<i>size_y</i>	Image height() .
<i>size_z</i>	Image depth() .
<i>size_c</i>	Image spectrum() (number of channels).
<i>value</i>	Initialization value.

Note

- Similar to [CImg\(unsigned int,unsigned int,unsigned int,unsigned int\)](#), but it also fills the pixel buffer with the specified *value*.

Warning

- It cannot be used to construct a vector-valued image and initialize it with *vector-valued* pixels (e.g. RGB vector, for color images). For this task, you may use [fillC\(\)](#) after construction.

8.1.3.5 CImg (const unsigned int *size_x*, const unsigned int *size_y*, const unsigned int *size_z*, const unsigned int *size_c*, const int *value0*, const int *value1*, ...)

Construct image with specified size and initialize pixel values from a sequence of integers.

Construct a new image instance of size *size_x* x *size_y* x *size_z* x *size_c*, with pixels of type T, and initialize pixel values from the specified sequence of integers *value0,value1,...*

Parameters

<i>size_x</i>	Image width() .
<i>size_y</i>	Image height() .
<i>size_z</i>	Image depth() .
<i>size_c</i>	Image spectrum() (number of channels).
<i>value0</i>	First value of the initialization sequence (must be an <i>integer</i>).
<i>value1</i>	Second value of the initialization sequence (must be an <i>integer</i>).
...	

Note

- Similar to `CImg(unsigned int,unsigned int,unsigned int,unsigned int)`, but it also fills the pixel buffer with a sequence of specified integer values.

Warning

- You must specify *exactly* `size_x*size_y*size_z*size_c` integers in the initialization sequence. Otherwise, the constructor may crash or fill your image pixels with garbage.

Example

```
const CImg<float> img(2,2,1,3,      // Construct a 2x2 color (RGB) image.
                    0,255,0,255,    // Set the 4 values for the red component.
                    0,0,255,255,    // Set the 4 values for the green component.
                    64,64,64,64);   // Set the 4 values for the blue component.
img.resize(150,150).display();
```

8.1.3.6 CImg (const unsigned int *size_x*, const unsigned int *size_y*, const unsigned int *size_z*, const unsigned int *size_c*, const double *value0*, const double *value1*, ...)

Construct image with specified size and initialize pixel values from a sequence of doubles.

Construct a new image instance of size `size_x x size_y x size_z x size_c`, with pixels of type `T`, and initialize pixel values from the specified sequence of doubles `value0,value1,...`

Parameters

<i>size_x</i>	Image width() .
<i>size_y</i>	Image height() .
<i>size_z</i>	Image depth() .
<i>size_c</i>	Image spectrum() (number of channels).
<i>value0</i>	First value of the initialization sequence (must be a <i>double</i>).
<i>value1</i>	Second value of the initialization sequence (must be a <i>double</i>).
...	

Note

- Similar to `CImg(unsigned int,unsigned int,unsigned int,unsigned int,int,int,...)`, but takes a sequence of double values instead of integers.

Warning

- You must specify *exactly* `dx*dy*dz*dc` doubles in the initialization sequence. Otherwise, the constructor may crash or fill your image with garbage. For instance, the code below will probably crash on most platforms:

```
const CImg<float> img(2,2,1,1, 0.5,0.5,255,255); // FAIL: The two last arguments are 'int', not 'double'!
```

8.1.3.7 CImg (const unsigned int *size_x*, const unsigned int *size_y*, const unsigned int *size_z*, const unsigned int *size_c*, const char *const *values*, const bool *repeat_values*)

Construct image with specified size and initialize pixel values from a value string.

Construct a new image instance of size *size_x* x *size_y* x *size_z* x *size_c*, with pixels of type *T*, and initializes pixel values from the specified string *values*.

Parameters

<i>size_x</i>	Image width() .
<i>size_y</i>	Image height() .
<i>size_z</i>	Image depth() .
<i>size_c</i>	Image spectrum() (number of channels).
<i>values</i>	Value string describing the way pixel values are set.
<i>repeat_values</i>	Tells if the value filling process is repeated over the image.

Note

- Similar to [CImg\(unsigned int,unsigned int,unsigned int,unsigned int\)](#), but it also fills the pixel buffer with values described in the value string *values*.
- Value string *values* may describe two different filling processes:
 - Either *values* is a sequences of values assigned to the image pixels, as in "1, 2, 3, 7, 8, 2". In this case, set *repeat_values* to *true* to periodically fill the image with the value sequence.
 - Either, *values* is a formula, as in " $\cos(x/10) * \sin(y/20)$ ". In this case, parameter *repeat_values* is pointless.
- For both cases, specifying *repeat_values* is mandatory. It disambiguates the possible overloading of constructor [CImg\(unsigned int,unsigned int,unsigned int,unsigned int,T\)](#) with *T* being a *const char**.
- A *CImgArgumentException* is thrown when an invalid value string *values* is specified.

Example

```
const CImg<float> img1(129,129,1,3,"0,64,128,192,255",true),           // Construct image filled
                    from a value sequence.
                    img2(129,129,1,3,"if(c==0,255*abs(cos(x/10)),1.8*y)",false); // Construct image filled
                    from a formula.
(img1,img2).display();
```

8.1.3.8 CImg (const t *const *values*, const unsigned int *size_x*, const unsigned int *size_y* = 1, const unsigned int *size_z* = 1, const unsigned int *size_c* = 1, const bool *is_shared* = false)

Construct image with specified size and initialize pixel values from a memory buffer.

Construct a new image instance of size *size_x* x *size_y* x *size_z* x *size_c*, with pixels of type *T*, and initializes pixel values from the specified *t** memory buffer.

Parameters

<i>values</i>	Pointer to the input memory buffer.
<i>size_x</i>	Image width() .
<i>size_y</i>	Image height() .
<i>size_z</i>	Image depth() .
<i>size_c</i>	Image spectrum() (number of channels).
<i>is_shared</i>	Tells if input memory buffer must be shared by the current instance.

Note

- If `is_shared` is `false`, the image instance allocates its own pixel buffer, and values from the specified input buffer are copied to the instance buffer. If buffer types `T` and `t` are different, a regular static cast is performed during buffer copy.
- Otherwise, the image instance does *not* allocate a new buffer, and uses the input memory buffer as its own pixel buffer. This case requires that types `T` and `t` are the same. Later, destroying such a shared image will not deallocate the pixel buffer, this task being obviously charged to the initial buffer allocator.
- A `CImgIOException` is thrown when the pixel buffer cannot be allocated (e.g. when requested size is too big for available memory).

Warning

- You must take care when operating on a shared image, since it may have an invalid pixel buffer pointer `data()` (e.g. already deallocated).

Example

```
unsigned char tab[256*256] = { 0 };
CImg<unsigned char> img1(tab,256,256,1,1,false), // Construct new non-shared image from buffer 'tab'.
                  img2(tab,256,256,1,1,true);  // Construct new shared-image from buffer 'tab'.
tab[1024] = 255;                               // Here, 'img2' is indirectly modified, but not 'img1'.
```

8.1.3.9 CImg (const char *const filename) [explicit]

Construct image from reading an image file.

Construct a new image instance with pixels of type `T`, and initialize pixel values with the data read from an image file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

Note

- Similar to `CImg(unsigned int,unsigned int,unsigned int,unsigned int)`, but it reads the image dimensions and pixel values from the specified image file.
- The recognition of the image file format by `CImg` highly depends on the tools installed on your system and on the external libraries you used to link your code against.
- Considered pixel type `T` should better fit the file format specification, or data loss may occur during file load (e.g. constructing a `CImg<unsigned char>` from a float-valued image file).
- A `CImgIOException` is thrown when the specified `filename` cannot be read, or if the file format is not recognized.

Example

```
const CImg<float> img("reference.jpg");
img.display();
```

8.1.3.10 CImg (const CImg< t > &img)

Construct image copy.

Construct a new image instance with pixels of type `T`, as a copy of an existing `CImg< t >` instance.

Parameters

<i>img</i>	Input image to copy.
------------	----------------------

Note

- Constructed copy has the same size `width()` x `height()` x `depth()` x `spectrum()` and pixel values as the input image `img`.
- If input image `img` is *shared* and if types `T` and `t` are the same, the constructed copy is also *shared*, and shares its pixel buffer with `img`. Modifying a pixel value in the constructed copy will thus also modifies it in the input image `img`. This behavior is needful to allow functions to return shared images.
- Otherwise, the constructed copy allocates its own pixel buffer, and copies pixel values from the input image `img` into its buffer. The copied pixel values may be eventually statically casted if types `T` and `t` are different.
- Constructing a copy from an image `img` when types `t` and `T` are the same is significantly faster than with different types.
- A `CImgInstanceException` is thrown when the pixel buffer cannot be allocated (e.g. not enough available memory).

8.1.3.11 CImg (const CImg< t > &img, const bool is_shared)

Advanced copy constructor.

Construct a new image instance with pixels of type `T`, as a copy of an existing `CImg<t>` instance, while forcing the shared state of the constructed copy.

Parameters

<i>img</i>	Input image to copy.
<i>is_shared</i>	Tells about the shared state of the constructed copy.

Note

- Similar to `CImg(const CImg<t>&)`, except that it allows to decide the shared state of the constructed image, which does not depend anymore on the shared state of the input image `img` :
 - If `is_shared` is `true`, the constructed copy will share its pixel buffer with the input image `img`. For that case, the pixel types `T` and `t` *must* be the same.
 - If `is_shared` is `false`, the constructed copy will allocate its own pixel buffer, whether the input image `img` is shared or not.
- A `CImgArgumentException` is thrown when a shared copy is requested with different pixel types `T` and `t`.

8.1.3.12 CImg (const CImg< t > &img, const char *const dimensions)

Construct image with dimensions borrowed from another image.

Construct a new image instance with pixels of type `T`, and size get from some dimensions of an existing `CImg<t>` instance.

Parameters

<i>img</i>	Input image from which dimensions are borrowed.
<i>dimensions</i>	C-string describing the image size along the X,Y,Z and C-dimensions.

Note

- Similar to [CImg\(unsigned int,unsigned int,unsigned int,unsigned int\)](#), but it takes the image dimensions (*not* its pixel values) from an existing `CImg<T>` instance.
- The allocated pixel buffer is *not* filled with a default value, and is likely to contain garbage values. In order to initialize pixel values (e.g. with 0), use constructor `CImg(const CImg<T>&,const char*,T)` instead.

Example

```
const CImg<float> img1(256,128,1,3),      // 'img1' is a 256x128x1x3 image.
                  img2(img1,"xyzc"),    // 'img2' is a 256x128x1x3 image.
                  img3(img1,"y,x,z,c"), // 'img3' is a 128x256x1x3 image.
                  img4(img1,"c,x,y,3",0), // 'img4' is a 3x128x256x3 image (with pixels initialized to
                  '0').
```

8.1.3.13 `CImg (const CImg< T > &img, const char *const dimensions, const T &value)`

Construct image with dimensions borrowed from another image and initialize pixel values.

Construct a new image instance with pixels of type `T`, and size get from the dimensions of an existing `CImg<T>` instance, and set all pixel values to specified `value`.

Parameters

<i>img</i>	Input image from which dimensions are borrowed.
<i>dimensions</i>	String describing the image size along the X,Y,Z and V-dimensions.
<i>value</i>	Value used for initialization.

Note

- Similar to [CImg\(const CImg<T>&,const char*\)](#), but it also fills the pixel buffer with the specified `value`.

8.1.3.14 `CImg (const CImgDisplay &disp) [explicit]`

Construct image from a display window.

Construct a new image instance with pixels of type `T`, as a snapshot of an existing `CImgDisplay` instance.

Parameters

<i>disp</i>	Input display window.
-------------	-----------------------

Note

- The `width()` and `height()` of the constructed image instance are the same as the specified `CImg↔Display`.
- The `depth()` and `spectrum()` of the constructed image instance are respectively set to 1 and 3 (i.e. a 2d color image).
- The image pixels are read as 8-bits RGB values.

8.1.4 Member Function Documentation

8.1.4.1 CImg<T>& assign ()

Construct empty image **[in-place version]**.

In-place version of the default constructor `CImg()`. It simply resets the instance to an empty image.

8.1.4.2 CImg<T>& assign (const unsigned int size_x, const unsigned int size_y = 1, const unsigned int size_z = 1, const unsigned int size_c = 1)

Construct image with specified size **[in-place version]**.

In-place version of the constructor `CImg(unsigned int,unsigned int,unsigned int,unsigned int)`.

8.1.4.3 CImg<T>& assign (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const T & value)

Construct image with specified size and initialize pixel values **[in-place version]**.

In-place version of the constructor `CImg(unsigned int,unsigned int,unsigned int,unsigned int,T)`.

8.1.4.4 CImg<T>& assign (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const int value0, const int value1, ...)

Construct image with specified size and initialize pixel values from a sequence of integers **[in-place version]**.

In-place version of the constructor `CImg(unsigned int,unsigned int,unsigned int,unsigned int,int,int,...)`.

8.1.4.5 CImg<T>& assign (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const double value0, const double value1, ...)

Construct image with specified size and initialize pixel values from a sequence of doubles **[in-place version]**.

In-place version of the constructor `CImg(unsigned int,unsigned int,unsigned int,unsigned int,double,double,...)`.

8.1.4.6 CImg<T>& assign (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c, const char *const values, const bool repeat_values)

Construct image with specified size and initialize pixel values from a value string **[in-place version]**.

In-place version of the constructor `CImg(unsigned int,unsigned int,unsigned int,unsigned int,const char*,bool)`.

8.1.4.7 CImg<T>& assign (const t *const values, const unsigned int size_x, const unsigned int size_y = 1, const unsigned int size_z = 1, const unsigned int size_c = 1)

Construct image with specified size and initialize pixel values from a memory buffer **[in-place version]**.

In-place version of the constructor CImg(const t*, unsigned int, unsigned int, unsigned int, unsigned int).

8.1.4.8 CImg<T>& assign (const char *const filename)

Construct image from reading an image file **[in-place version]**.

In-place version of the constructor CImg(const char*).

8.1.4.9 CImg<T>& assign (const CImg< t > & img)

Construct image copy **[in-place version]**.

In-place version of the constructor CImg(const CImg<t>&).

8.1.4.10 CImg<T>& assign (const CImg< t > & img, const bool is_shared)

In-place version of the advanced copy constructor.

In-place version of the constructor CImg(const CImg<t>&, bool).

8.1.4.11 CImg<T>& assign (const CImg< t > & img, const char *const dimensions)

Construct image with dimensions borrowed from another image **[in-place version]**.

In-place version of the constructor CImg(const CImg<t>&, const char*).

8.1.4.12 CImg<T>& assign (const CImg< t > & img, const char *const dimensions, const T & value)

Construct image with dimensions borrowed from another image and initialize pixel values **[in-place version]**.

In-place version of the constructor CImg(const CImg<t>&, const char*, T).

8.1.4.13 CImg<T>& assign (const CImgDisplay & disp)

Construct image from a display window **[in-place version]**.

In-place version of the constructor CImg(const CImgDisplay&).

8.1.4.14 CImg<T>& clear ()

Construct empty image **[in-place version]**.

Equivalent to [assign\(\)](#).

Note

- It has been defined for compatibility with STL naming conventions.

8.1.4.15 CImg<t>& move_to (CImg< t > & img)

Transfer content of an image instance into another one.

Transfer the dimensions and the pixel buffer content of an image instance into another one, and replace instance by an empty image. It avoids the copy of the pixel buffer when possible.

Parameters

<i>img</i>	Destination image.
------------	--------------------

Note

- Pixel types *T* and *t* of source and destination images can be different, though the process is designed to be instantaneous when *T* and *t* are the same.

Example

```
CImg<float> src(256,256,1,3,0), // Construct a 256x256x1x3 (color) image filled with value '0'.
             dest(16,16);      // Construct a 16x16x1x1 (scalar) image.
src.move_to(dest);             // Now, 'src' is empty and 'dest' is the 256x256x1x3 image.
```

8.1.4.16 CImgList<t>& move_to (CImgList< t > &list, const unsigned int pos = ~0U)

Transfer content of an image instance into a new image in an image list.

Transfer the dimensions and the pixel buffer content of an image instance into a newly inserted image at position *pos* in specified CImgList<t> instance.

Parameters

<i>list</i>	Destination list.
<i>pos</i>	Position of the newly inserted image in the list.

Note

- When optionnal parameter *pos* is ommited, the image instance is transfered as a new image at the end of the specified *list*.
- It is convenient to sequentially insert new images into image lists, with no additional copies of memory buffer.

Example

```
CImgList<float> list;           // Construct an empty image list.
CImg<float> img("reference.jpg"); // Read image from filename.
img.move_to(list);             // Transfer image content as a new item in the list (no buffer copy).
```

8.1.4.17 CImg<T>& swap (CImg< T > &img)

Swap fields of two image instances.

Parameters

<i>img</i>	Image to swap fields with.
------------	----------------------------

Note

- It can be used to interchange the content of two images in a very fast way. Can be convenient when dealing with algorithms requiring two swapping buffers.

Example

```
CImg<float> img1("lena.jpg"),
            img2("milla.jpg");
img1.swap(img2);           // Now, 'img1' is 'milla' and 'img2' is 'lena'.
```

8.1.4.18 static CImg<T>& empty () [static]

Return a reference to an empty image.

Note

This function is useful mainly to declare optional parameters having type `CImg<T>` in functions prototypes, e.g.

```
void f(const int x=0, const int y=0, const CImg<float>& img=CImg<float>::empty());
```

8.1.4.19 T& operator() (const unsigned int x, const unsigned int y = 0, const unsigned int z = 0, const unsigned int c = 0)

Access to a pixel value.

Return a reference to a located pixel value of the image instance, being possibly *const*, whether the image instance is *const* or not. This is the standard method to get/set pixel values in `CImg<T>` images.

Parameters

<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

- Range of pixel coordinates start from `(0, 0, 0, 0)` to `(width() - 1, height() - 1, depth() - 1, spectrum() - 1)`.
- Due to the particular arrangement of the pixel buffers defined in `CImg`, you can omit one coordinate if the corresponding dimension is equal to 1. For instance, pixels of a 2d image (`depth()` equal to 1) can be accessed by `img(x, y, c)` instead of `img(x, y, 0, c)`.

Warning

- There is *no* boundary checking done in this operator, to make it as fast as possible. You *must* take care of out-of-bounds access by yourself, if necessary. For debugging purposes, you may want to define macro `'cimg_verbosity'>=3` to enable additional boundary checking operations in this operator. In that case, warning messages will be printed on the error output when accessing out-of-bounds pixels.

Example

```

CImg<float> img(100,100,1,3,0);           // Construct a 100x100x1x3 (color) image with pixels set
    to '0'.
const float
    valR = img(10,10,0,0),                // Read red value at coordinates (10,10).
    valG = img(10,10,0,1),                // Read green value at coordinates (10,10)
    valB = img(10,10,2),                  // Read blue value at coordinates (10,10) (Z-coordinate
        can be omitted).
    avg = (valR + valG + valB)/3;          // Compute average pixel value.
img(10,10,0) = img(10,10,1) = img(10,10,2) = avg; // Replace the color pixel (10,10) by the average grey
    value.

```

8.1.4.20 T& operator() (const unsigned int x, const unsigned int y, const unsigned int z, const unsigned int c, const alongT wh, const alongT whd = 0)

Access to a pixel value.

Parameters

<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.
<i>wh</i>	Precomputed offset, must be equal to <code>width()*height()</code> .
<i>whd</i>	Precomputed offset, must be equal to <code>width()*height()*depth()</code> .

Note

- Similar to (but faster than) `operator()()`. It uses precomputed offsets to optimize memory access. You may use it to optimize the reading/writing of several pixel values in the same image (e.g. in a loop).

8.1.4.21 operator T* ()

Implicitly cast an image into a T*.

Implicitly cast a CImg<T> instance into a T* or const T* pointer, whether the image instance is *const* or not. The returned pointer points on the first value of the image pixel buffer.

Note

- It simply returns the pointer `data()` to the pixel buffer.
- This implicit conversion is convenient to test the empty state of images (`data()` being 0 in this case), e.g.

```

CImg<float> img1(100,100), img2; // 'img1' is a 100x100 image, 'img2' is an empty image.
if (img1) {                       // Test succeeds, 'img1' is not an empty image.
    if (!img2) {                   // Test succeeds, 'img2' is an empty image.
        std::printf("'img1' is not empty, 'img2' is empty.");
    }
}

```

- It also allows to use brackets to access pixel values, without need for a CImg<T>::operator[](), e.g.

```

CImg<float> img(100,100);
const float value = img[99]; // Access to value of the last pixel on the first row.
img[510] = 255;              // Set pixel value at (10,5).

```

8.1.4.22 CImg<T>& operator= (const T & *value*)

Assign a value to all image pixels.

Assign specified `value` to each pixel value of the image instance.

Parameters

<i>value</i>	Value that will be assigned to image pixels.
--------------	--

Note

- The image size is never modified.
- The `value` may be casted to pixel type `T` if necessary.

Example

```
CImg<char> img(100,100); // Declare image (with garbage values).
img = 0;                // Set all pixel values to '0'.
img = 1.2;              // Set all pixel values to '1' (cast of '1.2' as a 'char').
```

8.1.4.23 CImg<T>& operator= (const char *const expression)

Assign pixels values from a specified expression.

Initialize all pixel values from the specified string `expression`.

Parameters

<i>expression</i>	Value string describing the way pixel values are set.
-------------------	---

Note

- String parameter `expression` may describe different things:
 - If `expression` is a list of values (as in "1, 2, 3, 8, 3, 2"), or a formula (as in "(x*y) %255"), the pixel values are set from specified `expression` and the image size is not modified.
 - If `expression` is a filename (as in "reference.jpg"), the corresponding image file is loaded and replace the image instance. The image size is modified if necessary.

Example

```
CImg<float> img1(100,100), img2(img1), img3(img1); // Declare three 100x100 scalar images with uninitialized
pixel values.
img1 = "0,50,100,150,200,250,200,150,100,50";    // Set pixel values of 'img1' from a value sequence.
img2 = "10*((x*y)%25)";                          // Set pixel values of 'img2' from a formula.
img3 = "reference.jpg";                          // Set pixel values of 'img3' from a file (image size is
modified).
(img1,img2,img3).display();
```

8.1.4.24 CImg<T>& operator= (const CImg< t > & img)

Copy an image into the current image instance.

Similar to the in-place copy constructor `assign(const CImg<t>&)`.

8.1.4.25 CImg<T>& operator= (const CImgDisplay & disp)

Copy the content of a display window to the current image instance.

Similar to [assign\(const CImgDisplay&\)](#).

8.1.4.26 CImg<T>& operator+= (const t value)

In-place addition operator.

Add specified `value` to all pixels of an image instance.

Parameters

<i>value</i>	Value to add.
--------------	---------------

Note

- Resulting pixel values are casted to fit the pixel type `T`. For instance, adding `0.2` to a `CImg<char>` is possible but does nothing indeed.
- Overflow values are treated as with standard C++ numeric types. For instance,

```
CImg<unsigned char> img(100,100,1,1,255); // Construct a 100x100 image with pixel values '255'.
img+=1;                                // Add '1' to each pixels -> Overflow.
// here all pixels of image 'img' are equal to '0'.
```

- To prevent value overflow, you may want to consider pixel type `T` as `float` or `double`, and use [cut\(\)](#) after addition.

Example

```
CImg<unsigned char> img1("reference.jpg"); // Load a 8-bits RGB image (values in [0,255]).
CImg<float> img2(img1); // Construct a float-valued copy of 'img1'.
img2+=100; // Add '100' to pixel values -> goes out of [0,255] but
           // no problems with floats.
img2.cut(0,255); // Cut values in [0,255] to fit the 'unsigned char'
               // constraint.
img1 = img2; // Rewrite safe result in 'unsigned char' version
           // 'img1'.
const CImg<unsigned char> img3 = (img1 + 100).cut(0,255); // Do the same in a more simple and elegant
           // way.
(img1,img2,img3).display();
```

8.1.4.27 CImg<T>& operator+= (const char *const expression)

In-place addition operator.

Add values to image pixels, according to the specified string `expression`.

Parameters

<i>expression</i>	Value string describing the way pixel values are added.
-------------------	---

Note

- Similar to `operator=(const char*)`, except that it adds values to the pixels of the current image instance, instead of assigning them.

8.1.4.28 CImg<T>& operator+= (const CImg< t > & img)

In-place addition operator.

Add values to image pixels, according to the values of the input image `img`.

Parameters

<code>img</code>	Input image to add.
------------------	---------------------

Note

- The size of the image instance is never modified.
- It is not mandatory that input image `img` has the same size as the image instance. If less values are available in `img`, then the values are added periodically. For instance, adding one WxH scalar image (`spectrum()` equal to 1) to one WxH color image (`spectrum()` equal to 3) means each color channel will be incremented with the same values at the same locations.

Example

```
CImg<float> img1("reference.jpg"); // Load a RGB color image
    (img1.spectrum() == 3)
const CImg<float> img2(img1.width(),img.height(),1,1,"255*(x/w)^2"); // Construct a scalar shading
    (img2.spectrum() == 1) .
img1+=img2; // Add shading to each channel of
    'img1'.
img1.cut(0,255); // Prevent [0,255] overflow.
(img2,img1).display();
```

8.1.4.29 CImg<T>& operator++ ()

In-place increment operator (prefix).

Add 1 to all image pixels, and return a reference to the current incremented image instance.

Note

- Writing `++img` is equivalent to `img+=1`.

8.1.4.30 CImg<T> operator++ (int)

In-place increment operator (postfix).

Add 1 to all image pixels, and return a new copy of the initial (pre-incremented) image instance.

Note

- Use the prefixed version `operator++()` if you don't need a copy of the initial (pre-incremented) image instance, since a useless image copy may be expensive in terms of memory usage.

8.1.4.31 `CImg<T> operator+ () const`

Return a non-shared copy of the image instance.

Note

- Use this operator to ensure you get a non-shared copy of an image instance with same pixel type `T`. Indeed, the usual copy constructor `CImg<T>(const CImg<T>&)` returns a shared copy of a shared input image, and it may be not desirable to work on a regular copy (e.g. for a resize operation) if you have no information about the shared state of the input image.
- Writing `(+img)` is equivalent to `CImg<T>(img, false)`.

8.1.4.32 `CImg< typename cimg::superset<T,t>::type > operator+ (const t value) const`

Addition operator.

Similar to `operator+=(const t)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.33 `CImg<Tfloat> operator+ (const char *const expression) const`

Addition operator.

Similar to `operator+=(const char*)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.34 `CImg< typename cimg::superset<T,t>::type > operator+ (const CImg< t > & img) const`

Addition operator.

Similar to `operator+=(const CImg<t>&)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.35 `CImg<T>& operator-= (const t value)`

In-place subtraction operator.

Similar to `operator+=(const t)`, except that it performs a subtraction instead of an addition.

8.1.4.36 `CImg<T>& operator-= (const char *const expression)`

In-place subtraction operator.

Similar to `operator+=(const char*)`, except that it performs a subtraction instead of an addition.

8.1.4.37 CImg<T>& operator=(const CImg< t > & img)

In-place subtraction operator.

Similar to `operator+=(const CImg<t>&)`, except that it performs a subtraction instead of an addition.

8.1.4.38 CImg<T>& operator-- ()

In-place decrement operator (prefix).

Similar to `operator++()`, except that it performs a decrement instead of an increment.

8.1.4.39 CImg<T> operator-- (int)

In-place decrement operator (postfix).

Similar to `operator++(int)`, except that it performs a decrement instead of an increment.

8.1.4.40 CImg<T> operator- () const

Replace each pixel by its opposite value.

Note

- If the computed opposite values are out-of-range, they are treated as with standard C++ numeric types. For instance, the `unsigned char` opposite of 1 is 255.

Example

```
const CImg<unsigned char>
img1("reference.jpg"), // Load a RGB color image.
img2 = -img1;          // Compute its opposite (in 'unsigned char').
(img1, img2).display();
```

8.1.4.41 CImg< typename cimg::superset<T,t>::type > operator- (const t value) const

Subtraction operator.

Similar to `operator-=(const t)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.42 CImg<Tfloat> operator- (const char *const expression) const

Subtraction operator.

Similar to `operator-=(const char*)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type `T`, if necessary.

8.1.4.43 `CImg< typename cimg::superset<T,t>::type > operator- (const CImg< t > & img) const`

Substraction operator.

Similar to `operator-=(const CImg<t>&)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type T , if necessary.

8.1.4.44 `CImg<T>& operator*= (const t value)`

In-place multiplication operator.

Similar to `operator+=(const t)`, except that it performs a multiplication instead of an addition.

8.1.4.45 `CImg<T>& operator*= (const char *const expression)`

In-place multiplication operator.

Similar to `operator+=(const char*)`, except that it performs a multiplication instead of an addition.

8.1.4.46 `CImg<T>& operator*= (const CImg< t > & img)`

In-place multiplication operator.

Replace the image instance by the matrix multiplication between the image instance and the specified matrix `img`.

Parameters

<code>img</code>	Second operand of the matrix multiplication.
------------------	--

Note

- It does *not* compute a pointwise multiplication between two images. For this purpose, use `mul(const CImg<t>&)` instead.
- The size of the image instance can be modified by this operator.

Example

```
CImg<float> A(2,2,1,1, 1,2,3,4); // Construct 2x2 matrix A = [1,2;3,4].
const CImg<float> X(1,2,1,1, 1,2); // Construct 1x2 vector X = [1;2].
A*=X; // Assign matrix multiplication A*X to 'A'.
// 'A' is now a 1x2 vector whose values are [5;11].
```

8.1.4.47 `CImg< typename cimg::superset<T,t>::type > operator* (const t value) const`

Multiplication operator.

Similar to `operator*=(const t)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type T , if necessary.

8.1.4.48 CImg<Tfloat> operator* (const char *const *expression*) const

Multiplication operator.

Similar to `operator*=(const char*)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type T , if necessary.

8.1.4.49 CImg< typename cimg::superset<T,t>::type > operator* (const CImg< t > & *img*) const

Multiplication operator.

Similar to `operator*=(const CImg<t>&)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type T , if necessary.

8.1.4.50 CImg<T>& operator/= (const t *value*)

In-place division operator.

Similar to `operator+=(const t)`, except that it performs a division instead of an addition.

8.1.4.51 CImg<T>& operator/= (const char *const *expression*)

In-place division operator.

Similar to `operator+=(const char*)`, except that it performs a division instead of an addition.

8.1.4.52 CImg<T>& operator/= (const CImg< t > & *img*)

In-place division operator.

Replace the image instance by the (right) matrix division between the image instance and the specified matrix *img*.

Parameters

<i>img</i>	Second operand of the matrix division.
------------	--

Note

- It does *not* compute a pointwise division between two images. For this purpose, use `div(const CImg<t>&)` instead.
- It returns the matrix operation $A * \text{inverse}(img)$.
- The size of the image instance can be modified by this operator.

8.1.4.53 CImg< typename cimg::superset<T,t>::type > operator/ (const t *value*) const

Division operator.

Similar to `operator/=(const t)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type T , if necessary.

8.1.4.54 `CImg<Tfloat> operator/ (const char *const expression) const`

Division operator.

Similar to `operator/=(const char*)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type \mathbb{T} , if necessary.

8.1.4.55 `CImg< typename cimg::superset<T,t>::type > operator/ (const CImg< t > & img) const`

Division operator.

Similar to `operator/=(const CImg<t>&)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type \mathbb{T} , if necessary.

In-place modulo operator.

Similar to `operator+=(const t)`, except that it performs a modulo operation instead of an addition.

In-place modulo operator.

Similar to `operator+=(const char*)`, except that it performs a modulo operation instead of an addition.

In-place modulo operator.

Similar to `operator+=(const CImg<t>&)`, except that it performs a modulo operation instead of an addition.

Modulo operator.

Similar to `operator%=(const t)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type \mathbb{T} , if necessary.

Modulo operator.

Similar to `operator%=(const char*)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type \mathbb{T} , if necessary.

Modulo operator.

Similar to `operator%=(const CImg<t>&)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image may be a superset of the initial pixel type \mathbb{T} , if necessary.

8.1.4.56 `CImg<T>& operator&= (const t value)`

In-place bitwise AND operator.

Similar to `operator+=(const t)`, except that it performs a bitwise AND operation instead of an addition.

8.1.4.57 CImg<T>& operator&= (const char *const expression)

In-place bitwise AND operator.

Similar to `operator+=(const char*)`, except that it performs a bitwise AND operation instead of an addition.

8.1.4.58 CImg<T>& operator&= (const CImg< t > & img)

In-place bitwise AND operator.

Similar to `operator+=(const CImg<t>&)`, except that it performs a bitwise AND operation instead of an addition.

8.1.4.59 CImg<T> operator& (const t value) const

Bitwise AND operator.

Similar to `operator&=(const t)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is `T`.

8.1.4.60 CImg<T> operator& (const char *const expression) const

Bitwise AND operator.

Similar to `operator&=(const char*)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is `T`.

8.1.4.61 CImg<T> operator& (const CImg< t > & img) const

Bitwise AND operator.

Similar to `operator&=(const CImg<t>&)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is `T`.

8.1.4.62 CImg<T>& operator|= (const t value)

In-place bitwise OR operator.

Similar to `operator+=(const t)`, except that it performs a bitwise OR operation instead of an addition.

8.1.4.63 CImg<T>& operator|= (const char *const expression)

In-place bitwise OR operator.

Similar to `operator+=(const char*)`, except that it performs a bitwise OR operation instead of an addition.

8.1.4.64 CImg<T>& operator|= (const CImg< t > & img)

In-place bitwise OR operator.

Similar to `operator+=(const CImg<t>&)`, except that it performs a bitwise OR operation instead of an addition.

8.1.4.65 CImg<T> operator| (const t value) const

Bitwise OR operator.

Similar to `operator|=(const t)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is `T`.

8.1.4.66 CImg<T> operator| (const char *const expression) const

Bitwise OR operator.

Similar to `operator|=(const char*)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is `T`.

8.1.4.67 CImg<T> operator| (const CImg< t > & img) const

Bitwise OR operator.

Similar to `operator|=(const CImg<t>&)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is `T`.

8.1.4.68 CImg<T>& operator^= (const t value)

In-place bitwise XOR operator.

Similar to `operator+=(const t)`, except that it performs a bitwise XOR operation instead of an addition.

Warning

- It does *not* compute the *power* of pixel values. For this purpose, use `pow(const t)` instead.

8.1.4.69 CImg<T>& operator^= (const char *const expression)

In-place bitwise XOR operator.

Similar to `operator+=(const char*)`, except that it performs a bitwise XOR operation instead of an addition.

Warning

- It does *not* compute the *power* of pixel values. For this purpose, use `pow(const char*)` instead.

8.1.4.70 CImg<T>& operator^= (const CImg< t > & img)

In-place bitwise XOR operator.

Similar to `operator+=(const CImg<t>&)`, except that it performs a bitwise XOR operation instead of an addition.

Warning

- It does *not* compute the *power* of pixel values. For this purpose, use `pow(const CImg<t>&)` instead.

8.1.4.71 CImg<T> operator^ (const t value) const

Bitwise XOR operator.

Similar to `operator^=(const t)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is `T`.

8.1.4.72 CImg<T> operator^ (const char *const expression) const

Bitwise XOR operator.

Similar to `operator^=(const char*)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is `T`.

8.1.4.73 CImg<T> operator^ (const CImg< t > & img) const

Bitwise XOR operator.

Similar to `operator^=(const CImg<t>&)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is `T`.

8.1.4.74 CImg<T>& operator<<= (const t value)

In-place bitwise left shift operator.

Similar to `operator+=(const t)`, except that it performs a bitwise left shift instead of an addition.

8.1.4.75 CImg<T>& operator<<= (const char *const expression)

In-place bitwise left shift operator.

Similar to `operator+=(const char*)`, except that it performs a bitwise left shift instead of an addition.

8.1.4.76 CImg<T>& operator<<= (const CImg< t > & img)

In-place bitwise left shift operator.

Similar to `operator+=(const CImg<t>&)`, except that it performs a bitwise left shift instead of an addition.

8.1.4.77 CImg<T> operator<< (const t value) const

Bitwise left shift operator.

Similar to [operator<<=\(const t\)](#), except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is \mathbb{T} .

8.1.4.78 CImg<T> operator<< (const char *const expression) const

Bitwise left shift operator.

Similar to [operator<<=\(const char*\)](#), except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is \mathbb{T} .

8.1.4.79 CImg<T> operator<< (const CImg<t> &img) const

Bitwise left shift operator.

Similar to [operator<<=\(const CImg<t>&\)](#), except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is \mathbb{T} .

8.1.4.80 CImg<T>& operator>>= (const t value)

In-place bitwise right shift operator.

Similar to [operator+==\(const t\)](#), except that it performs a bitwise right shift instead of an addition.

8.1.4.81 CImg<T>& operator>>= (const char *const expression)

In-place bitwise right shift operator.

Similar to [operator+==\(const char*\)](#), except that it performs a bitwise right shift instead of an addition.

8.1.4.82 CImg<T>& operator>>= (const CImg<t> &img)

In-place bitwise right shift operator.

Similar to [operator+==\(const CImg<t>&\)](#), except that it performs a bitwise right shift instead of an addition.

8.1.4.83 CImg<T> operator>>> (const t value) const

Bitwise right shift operator.

Similar to [operator>>>=\(const t\)](#), except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is \mathbb{T} .

8.1.4.84 CImg<T> operator>> (const char *const *expression*) const

Bitwise right shift operator.

Similar to `operator>>=(const char*)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is `T`.

8.1.4.85 CImg<T> operator>> (const CImg< t > & *img*) const

Bitwise right shift operator.

Similar to `operator>>=(const CImg<t>&)`, except that it returns a new image instance instead of operating in-place. The pixel type of the returned image is `T`.

8.1.4.86 CImg<T> operator~ () const

Bitwise inversion operator.

Similar to `operator-()`, except that it compute the bitwise inverse instead of the opposite value.

8.1.4.87 bool operator==(const t *value*) const

Test if all pixels of an image have the same value.

Return `true` is all pixels of the image instance are equal to the specified `value`.

Parameters

<i>value</i>	Reference value to compare with.
--------------	----------------------------------

8.1.4.88 bool operator==(const char *const *expression*) const

Test if all pixel values of an image follow a specified expression.

Return `true` is all pixels of the image instance are equal to the specified `expression`.

Parameters

<i>expression</i>	Value string describing the way pixel values are compared.
-------------------	--

8.1.4.89 bool operator==(const CImg< t > & *img*) const

Test if two images have the same size and values.

Return `true` if the image instance and the input image `img` have the same dimensions and pixel values, and `false` otherwise.

Parameters

<i>img</i>	Input image to compare with.
------------	------------------------------

Note

- The pixel buffer pointers `data()` of the two compared images do not have to be the same for `operator==()` to return `true`. Only the dimensions and the pixel values matter. Thus, the comparison can be `true` even for different pixel types `T` and `t`.

Example

```
const CImg<float> img1(1,3,1,1, 0,1,2); // Construct a 1x3 vector [0;1;2] (with 'float' pixel values).
const CImg<char> img2(1,3,1,1, 0,1,2); // Construct a 1x3 vector [0;1;2] (with 'char' pixel values).
if (img1==img2) { // Test succeeds, image dimensions and values are the same.
    std::printf("'img1' and 'img2' have same dimensions and values.");
}
```

8.1.4.90 `bool operator!=(const t value) const`

Test if pixels of an image are all different from a value.

Return `true` is all pixels of the image instance are different than the specified `value`.

Parameters

<i>value</i>	Reference value to compare with.
--------------	----------------------------------

8.1.4.91 `bool operator!=(const char *const expression) const`

Test if all pixel values of an image are different from a specified expression.

Return `true` is all pixels of the image instance are different to the specified `expression`.

Parameters

<i>expression</i>	Value string describing the way pixel values are compared.
-------------------	--

8.1.4.92 `bool operator!=(const CImg< t > & img) const`

Test if two images have different sizes or values.

Return `true` if the image instance and the input image `img` have different dimensions or pixel values, and `false` otherwise.

Parameters

<i>img</i>	Input image to compare with.
------------	------------------------------

Note

- Writing `img1 != img2` is equivalent to `!(img1 == img2)`.

8.1.4.93 CImgList< typename cimg::superset<T,t>::type > operator, (const CImg< t > & img) const

Construct an image list from two images.

Return a new list of image (CImgList instance) containing exactly two elements:

- A copy of the image instance, at position [0].
- A copy of the specified image `img`, at position [1].

Parameters

<code>img</code>	Input image that will be the second image of the resulting list.
------------------	--

Note

- The family of `operator,()` is convenient to easily create list of images, but it is also *quite slow* in practice (see warning below).
- Constructed lists contain no shared images. If image instance or input image `img` are shared, they are inserted as new non-shared copies in the resulting list.
- The pixel type of the returned list may be a superset of the initial pixel type `T`, if necessary.

Warning

- Pipelining `operator,()` `N` times will perform `N` copies of the entire content of a (growing) image list. This may become very expensive in terms of speed and used memory. You should avoid using this technique to build a new CImgList instance from several images, if you are seeking for performance. Fast insertions of images in an image list are possible with `CImgList<T>::insert(const CImg<t>&,unsigned int,bool)` or `move_to(CImgList<t>&,unsigned int)`.

Example

```
const CImg<float>
    img1("reference.jpg"),
    img2 = img1.get_mirror('x'),
    img3 = img2.get_blur(5);
const CImgList<float> list = (img1,img2); // Create list of two elements from 'img1' and 'img2'.
(list,img3).display();                  // Display image list containing copies of 'img1','img2'
                                         and 'img3'.
```

8.1.4.94 CImgList< typename cimg::superset<T,t>::type > operator, (const CImgList< t > & list) const

Construct an image list from image instance and an input image list.

Return a new list of images (CImgList instance) containing exactly `list.size() + 1` elements:

- A copy of the image instance, at position [0].
- A copy of the specified image list `list`, from positions [1] to [`list.size()`].

Parameters

<i>list</i>	Input image list that will be appended to the image instance.
-------------	---

Note

- Similar to [operator,\(const CImg<t>&\) const](#), except that it takes an image list as an argument.

8.1.4.95 CImgList<T> operator< (const char axis) const

Split image along specified axis.

Return a new list of images ([CImgList](#) instance) containing the splitted components of the instance image along the specified axis.

Parameters

<i>axis</i>	Splitting axis (can be 'x','y','z' or 'c')
-------------	--

Note

- Similar to [get_split\(char,int\) const](#), with default second argument.

Example

```
const CImg<unsigned char> img("reference.jpg"); // Load a RGB color image.
const CImgList<unsigned char> list = (img<'c'>); // Get a list of its three R,G,B channels.
(img,list).display();
```

8.1.4.96 static const char* pixel_type () [static]

Return the type of image pixel values as a C string.

Return a `char*` string containing the usual type name of the image pixel values (i.e. a stringified version of the template parameter `T`).

Note

- The returned string may contain spaces (as in "unsigned char").
- If the pixel type `T` does not correspond to a registered type, the string "unknown" is returned.

8.1.4.97 int width () const

Return the number of image columns.

Return the image width, i.e. the image dimension along the X-axis.

Note

- The [width\(\)](#) of an empty image is equal to 0.
- [width\(\)](#) is typically equal to 1 when considering images as *vectors* for matrix calculations.
- [width\(\)](#) returns an `int`, although the image width is internally stored as an `unsigned int`. Using an `int` is safer and prevents arithmetic traps possibly encountered when doing calculations involving `unsigned int` variables. Access to the initial `unsigned int` variable is possible (though not recommended) by `(*this)._width`.

8.1.4.98 int height () const

Return the number of image rows.

Return the image height, i.e. the image dimension along the Y-axis.

Note

- The `height()` of an empty image is equal to 0.
- `height()` returns an `int`, although the image height is internally stored as an `unsigned int`. Using an `int` is safer and prevents arithmetic traps possibly encountered when doing calculations involving `unsigned int` variables. Access to the initial `unsigned int` variable is possible (though not recommended) by `(*this)._height`.

8.1.4.99 int depth () const

Return the number of image slices.

Return the image depth, i.e. the image dimension along the Z-axis.

Note

- The `depth()` of an empty image is equal to 0.
- `depth()` is typically equal to 1 when considering usual 2d images. When `depth()` > 1, the image is said to be *volumetric*.
- `depth()` returns an `int`, although the image depth is internally stored as an `unsigned int`. Using an `int` is safer and prevents arithmetic traps possibly encountered when doing calculations involving `unsigned int` variables. Access to the initial `unsigned int` variable is possible (though not recommended) by `(*this)._depth`.

8.1.4.100 int spectrum () const

Return the number of image channels.

Return the number of image channels, i.e. the image dimension along the C-axis.

Note

- The `spectrum()` of an empty image is equal to 0.
- `spectrum()` is typically equal to 1 when considering scalar-valued images, to 3 for RGB-coded color images, and to 4 for RGBA-coded color images (with alpha-channel). The number of channels of an image instance is not limited. The meaning of the pixel values is not linked up to the number of channels (e.g. a 4-channel image may indifferently stands for a RGBA or CMYK color image).
- `spectrum()` returns an `int`, although the image spectrum is internally stored as an `unsigned int`. Using an `int` is safer and prevents arithmetic traps possibly encountered when doing calculations involving `unsigned int` variables. Access to the initial `unsigned int` variable is possible (though not recommended) by `(*this)._spectrum`.

8.1.4.101 `ulongT size () const`

Return the total number of pixel values.

Return `width()*height()*depth()*spectrum()`, i.e. the total number of values of type `T` in the pixel buffer of the image instance.

Note

- The `size()` of an empty image is equal to 0.
- The allocated memory size for a pixel buffer of a non-shared `CImg<T>` instance is equal to `size()*sizeof(T)`.

Example

```
const CImg<float> img(100,100,1,3);           // Construct new 100x100 color image.
if (img.size()==30000)                       // Test succeeds.
    std::printf("Pixel buffer uses %lu bytes",
                img.size()*sizeof(float));
```

8.1.4.102 `T* data ()`

Return a pointer to the first pixel value.

Return a `T*`, or a `const T*` pointer to the first value in the pixel buffer of the image instance, whether the instance is `const` or not.

Note

- The `data()` of an empty image is equal to 0 (null pointer).
- The allocated pixel buffer for the image instance starts from `data()` and goes to `data()+size() - 1` (included).
- To get the pointer to one particular location of the pixel buffer, use `data(unsigned int,unsigned int,unsigned int,unsigned int)` instead.

8.1.4.103 `T* data (const unsigned int x, const unsigned int y = 0, const unsigned int z = 0, const unsigned int c = 0)`

Return a pointer to a located pixel value.

Return a `T*`, or a `const T*` pointer to the value located at `(x,y,z,c)` in the pixel buffer of the image instance, whether the instance is `const` or not.

Parameters

<code>x</code>	X-coordinate of the pixel value.
<code>y</code>	Y-coordinate of the pixel value.
<code>z</code>	Z-coordinate of the pixel value.
<code>c</code>	C-coordinate of the pixel value.

Note

- Writing `img.data(x,y,z,c)` is equivalent to `&(img(x,y,z,c))`. Thus, this method has the same properties as `operator()(unsigned int,unsigned int,unsigned int,unsigned int)`.

8.1.4.104 `longT offset (const int x, const int y = 0, const int z = 0, const int c = 0) const`

Return the offset to a located pixel value, with respect to the beginning of the pixel buffer.

Parameters

<code>x</code>	X-coordinate of the pixel value.
<code>y</code>	Y-coordinate of the pixel value.
<code>z</code>	Z-coordinate of the pixel value.
<code>c</code>	C-coordinate of the pixel value.

Note

- Writing `img.data(x,y,z,c)` is equivalent to `&(img(x,y,z,c)) - img.data()`. Thus, this method has the same properties as `operator()(unsigned int,unsigned int,unsigned int,unsigned int)`.

Example

```
const CImg<float> img(100,100,1,3);           // Define a 100x100 RGB-color image.
const long off = img.offset(10,10,0,2);      // Get the offset of the blue value of the pixel located at
(10,10).
const float val = img[off];                  // Get the blue value of this pixel.
```

8.1.4.105 `iterator begin ()`

Return a `CImg<T>::iterator` pointing to the first pixel value.

Note

- Equivalent to `data()`.
- It has been mainly defined for compatibility with STL naming conventions.

8.1.4.106 `iterator end ()`

Return a `CImg<T>::iterator` pointing next to the last pixel value.

Note

- Writing `img.end()` is equivalent to `img.data() + img.size()`.
- It has been mainly defined for compatibility with STL naming conventions.

Warning

- The returned iterator actually points to a value located *outside* the acceptable bounds of the pixel buffer. Trying to read or write the content of the returned iterator will probably result in a crash. Use it mainly as a strict upper bound for a `CImg<T>::iterator`.

Example

```
CImg<float> img(100,100,1,3);                // Define a 100x100 RGB color image.
for (CImg<float>::iterator it = img.begin(); it<img.end(); ++it) // 'img.end()' used
    here as an upper bound for the iterator.
    *it = 0;
```

8.1.4.107 T& front ()

Return a reference to the first pixel value.

Note

- Writing `img.front()` is equivalent to `img[0]`, or `img(0,0,0,0)`.
- It has been mainly defined for compatibility with STL naming conventions.

8.1.4.108 T& back ()

Return a reference to the last pixel value.

Note

- Writing `img.end()` is equivalent to `img[img.size() - 1]`, or `img(img.width() - 1, img.height() - 1, img.depth() - 1, img.spectrum() - 1)`.
- It has been mainly defined for compatibility with STL naming conventions.

8.1.4.109 T& at (const int *offset*, const T & *out_value*)

Access to a pixel value at a specified offset, using Dirichlet boundary conditions.

Return a reference to the pixel value of the image instance located at a specified `offset`, or to a specified default value in case of out-of-bounds access.

Parameters

<i>offset</i>	Offset to the desired pixel value.
<i>out_value</i>	Default value returned if <i>offset</i> is outside image bounds.

Note

- Writing `img.at(offset, out_value)` is similar to `img[offset]`, except that if `offset` is outside bounds (e.g. `offset < 0` or `offset >= img.size()`), a reference to a value `out_value` is safely returned instead.
- Due to the additional boundary checking operation, this method is slower than `operator()()`. Use it when you are *not* sure about the validity of the specified pixel offset.

8.1.4.110 T& at (const int *offset*)

Access to a pixel value at a specified offset, using Neumann boundary conditions.

Return a reference to the pixel value of the image instance located at a specified `offset`, or to the nearest pixel location in the image instance in case of out-of-bounds access.

Parameters

<code>offset</code>	Offset to the desired pixel value.
---------------------	------------------------------------

Note

- Similar to `at(int,const T)`, except that an out-of-bounds access returns the value of the nearest pixel in the image instance, regarding the specified offset, i.e.
 - If `offset < 0`, then `img[0]` is returned.
 - If `offset >= img.size()`, then `img[img.size() - 1]` is returned.
- Due to the additional boundary checking operation, this method is slower than `operator()()`. Use it when you are *not* sure about the validity of the specified pixel offset.
- If you know your image instance is *not* empty, you may rather use the slightly faster method `_at(int)`.

8.1.4.111 T& atX (const int x, const int y, const int z, const int c, const T & out_value)

Access to a pixel value, using Dirichlet boundary conditions for the X-coordinate.

Return a reference to the pixel value of the image instance located at (x,y,z,c), or to a specified default value in case of out-of-bounds access along the X-axis.

Parameters

<code>x</code>	X-coordinate of the pixel value.
<code>y</code>	Y-coordinate of the pixel value.
<code>z</code>	Z-coordinate of the pixel value.
<code>c</code>	C-coordinate of the pixel value.
<code>out_value</code>	Default value returned if (x,y,z,c) is outside image bounds.

Note

- Similar to `operator()()`, except that an out-of-bounds access along the X-axis returns the specified value `out_value`.
- Due to the additional boundary checking operation, this method is slower than `operator()()`. Use it when you are *not* sure about the validity of the specified pixel coordinates.

Warning

- There is *no* boundary checking performed for the Y,Z and C-coordinates, so they must be inside image bounds.

8.1.4.112 T& atX (const int x, const int y = 0, const int z = 0, const int c = 0)

Access to a pixel value, using Neumann boundary conditions for the X-coordinate.

Return a reference to the pixel value of the image instance located at (x,y,z,c), or to the nearest pixel location in the image instance in case of out-of-bounds access along the X-axis.

Parameters

<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

- Similar to `at(int,int,int,int,const T)`, except that an out-of-bounds access returns the value of the nearest pixel in the image instance, regarding the specified X-coordinate.
- Due to the additional boundary checking operation, this method is slower than `operator()()`. Use it when you are *not* sure about the validity of the specified pixel coordinates.
- If you know your image instance is *not* empty, you may rather use the slightly faster method `__atX↔at(int,int,int,int)`.

Warning

- There is *no* boundary checking performed for the Y,Z and C-coordinates, so they must be inside image bounds.

8.1.4.113 `T& atXY (const int x, const int y, const int z, const int c, const T & out_value)`

Access to a pixel value, using Dirichlet boundary conditions for the X and Y-coordinates.

Similar to `atX(int,int,int,int,const T)`, except that boundary checking is performed both on X and Y-coordinates.

8.1.4.114 `T& atXY (const int x, const int y, const int z = 0, const int c = 0)`

Access to a pixel value, using Neumann boundary conditions for the X and Y-coordinates.

Similar to `atX(int,int,int,int)`, except that boundary checking is performed both on X and Y-coordinates.

Note

- If you know your image instance is *not* empty, you may rather use the slightly faster method `__atX↔Y(int,int,int,int)`.

8.1.4.115 `T& atXYZ (const int x, const int y, const int z, const int c, const T & out_value)`

Access to a pixel value, using Dirichlet boundary conditions for the X,Y and Z-coordinates.

Similar to `atX(int,int,int,int,const T)`, except that boundary checking is performed both on X,Y and Z-coordinates.

8.1.4.116 T& atXYZ (const int x, const int y, const int z, const int c = 0)

Access to a pixel value, using Neumann boundary conditions for the X,Y and Z-coordinates.

Similar to [atX\(int,int,int,int\)](#), except that boundary checking is performed both on X,Y and Z-coordinates.

Note

- If you know your image instance is *not* empty, you may rather use the slightly faster method `_atXYZ↔Z(int,int,int,int)`.

8.1.4.117 T& atXYZC (const int x, const int y, const int z, const int c, const T & out_value)

Access to a pixel value, using Dirichlet boundary conditions.

Similar to `atX(int,int,int,int,const T)`, except that boundary checking is performed on all X,Y,Z and C-coordinates.

8.1.4.118 T& atXYZC (const int x, const int y, const int z, const int c)

Access to a pixel value, using Neumann boundary conditions.

Similar to [atX\(int,int,int,int\)](#), except that boundary checking is performed on all X,Y,Z and C-coordinates.

Note

- If you know your image instance is *not* empty, you may rather use the slightly faster method `_atXYZ↔C(int,int,int,int)`.

8.1.4.119 Tfloat linear_atX (const float fx, const int y, const int z, const int c, const T & out_value) const

Return pixel value, using linear interpolation and Dirichlet boundary conditions for the X-coordinate.

Return a linearly-interpolated pixel value of the image instance located at (fx,y,z,c) , or a specified default value in case of out-of-bounds access along the X-axis.

Parameters

<i>fx</i>	X-coordinate of the pixel value (float-valued).
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.
<i>out_value</i>	Default value returned if (fx,y,z,c) is outside image bounds.

Note

- Similar to `atX(int,int,int,int,const T)`, except that the returned pixel value is approximated by a linear interpolation along the X-axis, if corresponding coordinates are not integers.

- The type of the returned pixel value is extended to `float`, if the pixel type `T` is not float-valued.

Warning

- There is *no* boundary checking performed for the Y,Z and C-coordinates, so they must be inside image bounds.

8.1.4.120 `Tfloat linear_atX (const float fx, const int y = 0, const int z = 0, const int c = 0) const`

Return pixel value, using linear interpolation and Neumann boundary conditions for the X-coordinate.

Return a linearly-interpolated pixel value of the image instance located at (f_x, y, z, c) , or the value of the nearest pixel location in the image instance in case of out-of-bounds access along the X-axis.

Parameters

<code>fx</code>	X-coordinate of the pixel value (float-valued).
<code>y</code>	Y-coordinate of the pixel value.
<code>z</code>	Z-coordinate of the pixel value.
<code>c</code>	C-coordinate of the pixel value.

Note

- Similar to `linear_atX(float,int,int,int,const T) const`, except that an out-of-bounds access returns the value of the nearest pixel in the image instance, regarding the specified X-coordinate.
- If you know your image instance is *not* empty, you may rather use the slightly faster method `_linear_atX(float,int,int,int)`.

Warning

- There is *no* boundary checking performed for the Y,Z and C-coordinates, so they must be inside image bounds.

8.1.4.121 `Tfloat linear_atXY (const float fx, const float fy, const int z, const int c, const T & out_value) const`

Return pixel value, using linear interpolation and Dirichlet boundary conditions for the X and Y-coordinates.

Similar to `linear_atX(float,int,int,int,const T) const`, except that the linear interpolation and the boundary checking are achieved both for X and Y-coordinates.

8.1.4.122 `Tfloat linear_atXY (const float fx, const float fy, const int z = 0, const int c = 0) const`

Return pixel value, using linear interpolation and Neumann boundary conditions for the X and Y-coordinates.

Similar to `linear_atX(float,int,int,int) const`, except that the linear interpolation and the boundary checking are achieved both for X and Y-coordinates.

Note

- If you know your image instance is *not* empty, you may rather use the slightly faster method `_linear_atXY(float,float,int,int)`.

8.1.4.123 `Tfloat linear_atXYZ (const float fx, const float fy, const float fz, const int c, const T & out_value) const`

Return pixel value, using linear interpolation and Dirichlet boundary conditions for the X,Y and Z-coordinates.

Similar to `linear_atX(float,int,int,int,const T) const`, except that the linear interpolation and the boundary checking are achieved both for X,Y and Z-coordinates.

8.1.4.124 `Tfloat linear_atXYZ (const float fx, const float fy = 0, const float fz = 0, const int c = 0) const`

Return pixel value, using linear interpolation and Neumann boundary conditions for the X,Y and Z-coordinates.

Similar to `linear_atX(float,int,int,int) const`, except that the linear interpolation and the boundary checking are achieved both for X,Y and Z-coordinates.

Note

- If you know your image instance is *not* empty, you may rather use the slightly faster method `_linear_atXYZ(float, float, float, int)`.

8.1.4.125 `Tfloat linear_atXYZC (const float fx, const float fy, const float fz, const float fc, const T & out_value) const`

Return pixel value, using linear interpolation and Dirichlet boundary conditions for all X,Y,Z,C-coordinates.

Similar to `linear_atX(float,int,int,int,const T) const`, except that the linear interpolation and the boundary checking are achieved for all X,Y,Z and C-coordinates.

8.1.4.126 `Tfloat linear_atXYZC (const float fx, const float fy = 0, const float fz = 0, const float fc = 0) const`

Return pixel value, using linear interpolation and Neumann boundary conditions for all X,Y,Z and C-coordinates.

Similar to `linear_atX(float,int,int,int) const`, except that the linear interpolation and the boundary checking are achieved for all X,Y,Z and C-coordinates.

Note

- If you know your image instance is *not* empty, you may rather use the slightly faster method `_linear_atXYZC(float, float, float, float)`.

8.1.4.127 `Tfloat cubic_atX (const float fx, const int y, const int z, const int c, const T & out_value) const`

Return pixel value, using cubic interpolation and Dirichlet boundary conditions for the X-coordinate.

Return a cubically-interpolated pixel value of the image instance located at (fx,y,z,c) , or a specified default value in case of out-of-bounds access along the X-axis. The cubic interpolation uses Hermite splines.

Parameters

<code>fx</code>	d X-coordinate of the pixel value (float-valued).
<code>y</code>	Y-coordinate of the pixel value.
<code>z</code>	Z-coordinate of the pixel value.
<code>c</code>	C-coordinate of the pixel value.
<code>out_value</code>	Default value returned if (fx,y,z,c) is outside image bounds.

Note

- Similar to `linear_atX(float,int,int,int,const T) const`, except that the returned pixel value is approximated by a *cubic* interpolation along the X-axis.
- The type of the returned pixel value is extended to `float`, if the pixel type `T` is not float-valued.

Warning

- There is *no* boundary checking performed for the Y,Z and C-coordinates, so they must be inside image bounds.

8.1.4.128 `Tfloat cubic_atX (const float fx, const int y, const int z, const int c, const T & out_value, const Tfloat min_value, const Tfloat max_value) const`

Return damped pixel value, using cubic interpolation and Dirichlet boundary conditions for the X-coordinate.

Similar to `cubic_atX(float,int,int,int,const T) const`, except that you can specify the authorized minimum and maximum of the returned value.

8.1.4.129 `Tfloat cubic_atX (const float fx, const int y = 0, const int z = 0, const int c = 0) const`

Return pixel value, using cubic interpolation and Neumann boundary conditions for the X-coordinate.

Return a cubically-interpolated pixel value of the image instance located at (f_x, y, z, c) , or the value of the nearest pixel location in the image instance in case of out-of-bounds access along the X-axis. The cubic interpolation uses Hermite splines.

Parameters

<i>fx</i>	X-coordinate of the pixel value (float-valued).
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

- Similar to `cubic_atX(float,int,int,int,const T) const`, except that the returned pixel value is approximated by a cubic interpolation along the X-axis.
- If you know your image instance is *not* empty, you may rather use the slightly faster method `_cubic_atX(float, int, int, int)`.

Warning

- There is *no* boundary checking performed for the Y,Z and C-coordinates, so they must be inside image bounds.

8.1.4.130 Tfloat cubic_atX (const float *fx*, const int *y*, const int *z*, const int *c*, const Tfloat *min_value*, const Tfloat *max_value*) const

Return damped pixel value, using cubic interpolation and Neumann boundary conditions for the X-coordinate.

Similar to [cubic_atX\(float,int,int,int\) const](#), except that you can specify the authorized minimum and maximum of the returned value.

8.1.4.131 Tfloat cubic_atXY (const float *fx*, const float *fy*, const int *z*, const int *c*, const T & *out_value*) const

Return pixel value, using cubic interpolation and Dirichlet boundary conditions for the X and Y-coordinates.

Similar to [cubic_atX\(float,int,int,int,const T\) const](#), except that the cubic interpolation and boundary checking are achieved both for X and Y-coordinates.

8.1.4.132 Tfloat cubic_atXY (const float *fx*, const float *fy*, const int *z*, const int *c*, const T & *out_value*, const Tfloat *min_value*, const Tfloat *max_value*) const

Return damped pixel value, using cubic interpolation and Dirichlet boundary conditions for the X,Y-coordinates.

Similar to [cubic_atXY\(float,float,int,int,const T\) const](#), except that you can specify the authorized minimum and maximum of the returned value.

8.1.4.133 Tfloat cubic_atXY (const float *fx*, const float *fy*, const int *z* = 0, const int *c* = 0) const

Return pixel value, using cubic interpolation and Neumann boundary conditions for the X and Y-coordinates.

Similar to [cubic_atX\(float,int,int,int\) const](#), except that the cubic interpolation and boundary checking are achieved for both X and Y-coordinates.

Note

- If you know your image instance is *not* empty, you may rather use the slightly faster method `_cubic_atXY(float, float, int, int)`.

8.1.4.134 Tfloat cubic_atXY (const float *fx*, const float *fy*, const int *z*, const int *c*, const Tfloat *min_value*, const Tfloat *max_value*) const

Return damped pixel value, using cubic interpolation and Neumann boundary conditions for the X,Y-coordinates.

Similar to [cubic_atXY\(float,float,int,int\) const](#), except that you can specify the authorized minimum and maximum of the returned value.

8.1.4.135 Tfloat cubic_atXYZ (const float *fx*, const float *fy*, const float *fz*, const int *c*, const T & *out_value*) const

Return pixel value, using cubic interpolation and Dirichlet boundary conditions for the X,Y and Z-coordinates.

Similar to [cubic_atX\(float,int,int,int,const T\) const](#), except that the cubic interpolation and boundary checking are achieved both for X,Y and Z-coordinates.

8.1.4.136 `Tfloat cubic_atXYZ (const float fx, const float fy, const float fz, const int c, const T & out_value, const Tfloat min_value, const Tfloat max_value) const`

Return damped pixel value, using cubic interpolation and Dirichlet boundary conditions for the XYZ-coordinates.

Similar to `cubic_atXYZ(float,float,float,int,const T) const`, except that you can specify the authorized minimum and maximum of the returned value.

8.1.4.137 `Tfloat cubic_atXYZ (const float fx, const float fy, const float fz, const int c = 0) const`

Return pixel value, using cubic interpolation and Neumann boundary conditions for the X,Y and Z-coordinates.

Similar to `cubic_atX(float,int,int,int) const`, except that the cubic interpolation and boundary checking are achieved both for X,Y and Z-coordinates.

Note

- If you know your image instance is *not* empty, you may rather use the slightly faster method `_cubic_atXYZ(float, float, float, int)`.

8.1.4.138 `Tfloat cubic_atXYZ (const float fx, const float fy, const float fz, const int c, const Tfloat min_value, const Tfloat max_value) const`

Return damped pixel value, using cubic interpolation and Neumann boundary conditions for the XYZ-coordinates.

Similar to `cubic_atXYZ(float,float,float,int) const`, except that you can specify the authorized minimum and maximum of the returned value.

8.1.4.139 `Clmg<T>& set_linear_atX (const T & value, const float fx, const int y = 0, const int z = 0, const int c = 0, const bool is_added = false)`

Set pixel value, using linear interpolation for the X-coordinates.

Set pixel value at specified coordinates (fx,y,z,c) in the image instance, in a way that the value is spread amongst several neighbors if the pixel coordinates are float-valued.

Parameters

<i>value</i>	Pixel value to set.
<i>fx</i>	X-coordinate of the pixel value (float-valued).
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.
<i>is_added</i>	Tells if the pixel value is added to (<code>true</code>), or simply replace (<code>false</code>) the current image pixel(s).

Returns

A reference to the current image instance.

Note

- Calling this method with out-of-bounds coordinates does nothing.

8.1.4.140 `CImg<T>& set_linear_atXY (const T & value, const float fx, const float fy = 0, const int z = 0, const int c = 0, const bool is_added = false)`

Set pixel value, using linear interpolation for the X and Y-coordinates.

Similar to `set_linear_atX(const T&,float,int,int,int,bool)`, except that the linear interpolation is achieved both for X and Y-coordinates.

8.1.4.141 `CImg<T>& set_linear_atXYZ (const T & value, const float fx, const float fy = 0, const float fz = 0, const int c = 0, const bool is_added = false)`

Set pixel value, using linear interpolation for the X,Y and Z-coordinates.

Similar to `set_linear_atXY(const T&,float,float,int,int,bool)`, except that the linear interpolation is achieved both for X,Y and Z-coordinates.

8.1.4.142 `CImg<charT> value_string (const char separator = ' , ', const unsigned int max_size = 0, const char *const format = 0) const`

Return a C-string containing a list of all values of the image instance.

Return a new `CImg<char>` image whose buffer `data()` is a `char*` string describing the list of all pixel values of the image instance (written in base 10), separated by specified `separator` character.

Parameters

<i>separator</i>	A <code>char</code> character which specifies the separator between values in the returned C-string.
<i>max_size</i>	Maximum size of the returned image.
<i>format</i>	For float-values, tell the printf format used to generate the ascii representation of the numbers. (or 0 for default representation).

Note

- The returned image is never empty.
- For an empty image instance, the returned string is " ".
- If `max_size` is equal to 0, there are no limits on the size of the returned string.
- Otherwise, if the maximum number of string characters is exceeded, the value string is cut off and terminated by character `' \0 '`. In that case, the returned image size is `max_size + 1`.

8.1.4.143 `bool is_shared () const`

Test shared state of the pixel buffer.

Return `true` if image instance has a shared memory buffer, and `false` otherwise.

Note

- A shared image do not own his pixel buffer [data\(\)](#) and will not deallocate it on destruction.
- Most of the time, a `CImg<T>` image instance will *not* be shared.
- A shared image can only be obtained by a limited set of constructors and methods (see list below).

8.1.4.144 `bool is_empty () const`

Test if image instance is empty.

Return `true`, if image instance is empty, i.e. does *not* contain any pixel values, has dimensions `0 x 0 x 0 x 0` and a pixel buffer pointer set to `0` (null pointer), and `false` otherwise.

8.1.4.145 `bool is_inf () const`

Test if image instance contains a 'inf' value.

Return `true`, if image instance contains a 'inf' value, and `false` otherwise.

8.1.4.146 `bool is_nan () const`

Test if image instance contains a NaN value.

Return `true`, if image instance contains a NaN value, and `false` otherwise.

8.1.4.147 `bool is_sameXY (const unsigned int size_x, const unsigned int size_y) const`

Test if image width and height are equal to specified values.

Test if [is_sameX\(unsigned int\) const](#) and [is_sameY\(unsigned int\) const](#) are both verified.

8.1.4.148 `bool is_sameXY (const CImg< t > & img) const`

Test if image width and height are the same as that of another image.

Test if [is_sameX\(const CImg<t>&\) const](#) and [is_sameY\(const CImg<t>&\) const](#) are both verified.

8.1.4.149 `bool is_sameXY (const CImgDisplay & disp) const`

Test if image width and height are the same as that of an existing display window.

Test if [is_sameX\(const CImgDisplay&\) const](#) and [is_sameY\(const CImgDisplay&\) const](#) are both verified.

8.1.4.150 `bool is_sameXZ (const unsigned int size_x, const unsigned int size_z) const`

Test if image width and depth are equal to specified values.

Test if [is_sameX\(unsigned int\) const](#) and [is_sameZ\(unsigned int\) const](#) are both verified.

8.1.4.151 `bool is_sameXZ (const CImg< t > & img) const`

Test if image width and depth are the same as that of another image.

Test if [is_sameX\(const CImg<t>&\) const](#) and [is_sameZ\(const CImg<t>&\) const](#) are both verified.

8.1.4.152 `bool is_sameXC (const unsigned int size_x, const unsigned int size_c) const`

Test if image width and spectrum are equal to specified values.

Test if [is_sameX\(unsigned int\) const](#) and [is_sameC\(unsigned int\) const](#) are both verified.

8.1.4.153 `bool is_sameXC (const CImg< t > & img) const`

Test if image width and spectrum are the same as that of another image.

Test if [is_sameX\(const CImg<t>&\) const](#) and [is_sameC\(const CImg<t>&\) const](#) are both verified.

8.1.4.154 `bool is_sameYZ (const unsigned int size_y, const unsigned int size_z) const`

Test if image height and depth are equal to specified values.

Test if [is_sameY\(unsigned int\) const](#) and [is_sameZ\(unsigned int\) const](#) are both verified.

8.1.4.155 `bool is_sameYZ (const CImg< t > & img) const`

Test if image height and depth are the same as that of another image.

Test if [is_sameY\(const CImg<t>&\) const](#) and [is_sameZ\(const CImg<t>&\) const](#) are both verified.

8.1.4.156 `bool is_sameYC (const unsigned int size_y, const unsigned int size_c) const`

Test if image height and spectrum are equal to specified values.

Test if [is_sameY\(unsigned int\) const](#) and [is_sameC\(unsigned int\) const](#) are both verified.

8.1.4.157 `bool is_sameYC (const CImg< t > & img) const`

Test if image height and spectrum are the same as that of another image.

Test if [is_sameY\(const CImg<t>&\) const](#) and [is_sameC\(const CImg<t>&\) const](#) are both verified.

8.1.4.158 `bool is_sameZC (const unsigned int size_z, const unsigned int size_c) const`

Test if image depth and spectrum are equal to specified values.

Test if [is_sameZ\(unsigned int\) const](#) and [is_sameC\(unsigned int\) const](#) are both verified.

8.1.4.159 `bool is_sameZC (const CImg< t > & img) const`

Test if image depth and spectrum are the same as that of another image.

Test if [is_sameZ\(const CImg<t>&\) const](#) and [is_sameC\(const CImg<t>&\) const](#) are both verified.

8.1.4.160 `bool is_sameXYZ (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z) const`

Test if image width, height and depth are equal to specified values.

Test if [is_sameXY\(unsigned int,unsigned int\) const](#) and [is_sameZ\(unsigned int\) const](#) are both verified.

8.1.4.161 `bool is_sameXYZ (const CImg< t > & img) const`

Test if image width, height and depth are the same as that of another image.

Test if [is_sameXY\(const CImg<t>&\) const](#) and [is_sameZ\(const CImg<t>&\) const](#) are both verified.

8.1.4.162 `bool is_sameXYC (const unsigned int size_x, const unsigned int size_y, const unsigned int size_c) const`

Test if image width, height and spectrum are equal to specified values.

Test if [is_sameXY\(unsigned int,unsigned int\) const](#) and [is_sameC\(unsigned int\) const](#) are both verified.

8.1.4.163 `bool is_sameXYC (const CImg< t > & img) const`

Test if image width, height and spectrum are the same as that of another image.

Test if [is_sameXY\(const CImg<t>&\) const](#) and [is_sameC\(const CImg<t>&\) const](#) are both verified.

8.1.4.164 `bool is_sameXZC (const unsigned int size_x, const unsigned int size_z, const unsigned int size_c) const`

Test if image width, depth and spectrum are equal to specified values.

Test if [is_sameXZ\(unsigned int,unsigned int\) const](#) and [is_sameC\(unsigned int\) const](#) are both verified.

8.1.4.165 `bool is_sameXZC (const CImg< t > & img) const`

Test if image width, depth and spectrum are the same as that of another image.

Test if [is_sameXZ\(const CImg<t>&\) const](#) and [is_sameC\(const CImg<t>&\) const](#) are both verified.

8.1.4.166 `bool is_sameYZC (const unsigned int size_y, const unsigned int size_z, const unsigned int size_c) const`

Test if image height, depth and spectrum are equal to specified values.

Test if `is_sameYZ(unsigned int,unsigned int) const` and `is_sameC(unsigned int) const` are both verified.

8.1.4.167 `bool is_sameYZC (const Clmg< t > & img) const`

Test if image height, depth and spectrum are the same as that of another image.

Test if `is_sameYZ(const Clmg<t>&) const` and `is_sameC(const Clmg<t>&) const` are both verified.

8.1.4.168 `bool is_sameXYZC (const unsigned int size_x, const unsigned int size_y, const unsigned int size_z, const unsigned int size_c) const`

Test if image width, height, depth and spectrum are equal to specified values.

Test if `is_sameXYZ(unsigned int,unsigned int,unsigned int) const` and `is_sameC(unsigned int) const` are both verified.

8.1.4.169 `bool is_sameXYZC (const Clmg< t > & img) const`

Test if image width, height, depth and spectrum are the same as that of another image.

Test if `is_sameXYZ(const Clmg<t>&) const` and `is_sameC(const Clmg<t>&) const` are both verified.

8.1.4.170 `bool containsXYZC (const int x, const int y = 0, const int z = 0, const int c = 0) const`

Test if specified coordinates are inside image bounds.

Return `true` if pixel located at (x,y,z,c) is inside bounds of the image instance, and `false` otherwise.

Parameters

<code>x</code>	X-coordinate of the pixel value.
<code>y</code>	Y-coordinate of the pixel value.
<code>z</code>	Z-coordinate of the pixel value.
<code>c</code>	C-coordinate of the pixel value.

Note

- Return `true` only if all these conditions are verified:
 - The image instance is *not* empty.
 - `0 <= x <= width() - 1`.
 - `0 <= y <= height() - 1`.
 - `0 <= z <= depth() - 1`.
 - `0 <= c <= spectrum() - 1`.

8.1.4.171 `bool contains (const T & pixel, t & x, t & y, t & z, t & c) const`

Test if pixel value is inside image bounds and get its X,Y,Z and C-coordinates.

Return `true`, if specified reference refers to a pixel value inside bounds of the image instance, and `false` otherwise.

Parameters

	<i>pixel</i>	Reference to pixel value to test.
out	<i>x</i>	X-coordinate of the pixel value, if test succeeds.
out	<i>y</i>	Y-coordinate of the pixel value, if test succeeds.
out	<i>z</i>	Z-coordinate of the pixel value, if test succeeds.
out	<i>c</i>	C-coordinate of the pixel value, if test succeeds.

Note

- Useful to convert an offset to a buffer value into pixel value coordinates:

```
const CImg<float> img(100,100,1,3);      // Construct a 100x100 RGB color image.
const unsigned long offset = 1249;      // Offset to the pixel (49,12,0,0).
unsigned int x,y,z,c;
if (img.contains(img[offset],x,y,z,c)) { // Convert offset to (x,y,z,c) coordinates.
    std::printf("Offset %u refers to pixel located at (%u,%u,%u,%u).\n",
                offset,x,y,z,c);
}
```

8.1.4.172 `bool contains (const T & pixel, t & x, t & y, t & z) const`

Test if pixel value is inside image bounds and get its X,Y and Z-coordinates.

Similar to `contains(const T&,t&,t&,t&,t&) const`, except that only the X,Y and Z-coordinates are set.

8.1.4.173 `bool contains (const T & pixel, t & x, t & y) const`

Test if pixel value is inside image bounds and get its X and Y-coordinates.

Similar to `contains(const T&,t&,t&,t&,t&) const`, except that only the X and Y-coordinates are set.

8.1.4.174 `bool contains (const T & pixel, t & x) const`

Test if pixel value is inside image bounds and get its X-coordinate.

Similar to `contains(const T&,t&,t&,t&,t&) const`, except that only the X-coordinate is set.

8.1.4.175 `bool contains (const T & pixel) const`

Test if pixel value is inside image bounds.

Similar to `contains(const T&,t&,t&,t&,t&) const`, except that no pixel coordinates are set.

8.1.4.176 `bool is_overlapped (const CImg< t > & img) const`

Test if pixel buffers of instance and input images overlap.

Return `true`, if pixel buffers attached to image instance and input image `img` overlap, and `false` otherwise.

Parameters

<i>img</i>	Input image to compare with.
------------	------------------------------

Note

- Buffer overlapping may happen when manipulating *shared* images.
- If two image buffers overlap, operating on one of the image will probably modify the other one.
- Most of the time, CImg<T> instances are *non-shared* and do not overlap between each others.

Example

```
const CImg<float>
img1("reference.jpg"),           // Load RGB-color image.
img2 = img1.get_shared_channel(1); // Get shared version of the green channel.
if (img1.is_overlapped(img2)) {   // Test succeeds, 'img1' and 'img2' overlaps.
    std::printf("Buffers overlap!\n");
}
```

8.1.4.177 `bool is_object3d (const CImgList< tp > &primitives, const CImgList< tc > &colors, const to &opacities, const bool full_check=true, char *const error_message=0) const`

Test if the set {**this*,*primitives*,*colors*,*opacities*} defines a valid 3d object.

Return `true` is the 3d object represented by the set {**this*,*primitives*,*colors*,*opacities*} defines a valid 3d object, and `false` otherwise. The vertex coordinates are defined by the instance image.

Parameters

	<i>primitives</i>	List of primitives of the 3d object.
	<i>colors</i>	List of colors of the 3d object.
	<i>opacities</i>	List (or image) of opacities of the 3d object.
	<i>full_check</i>	Tells if full checking of the 3d object must be performed.
out	<i>error_message</i>	C-string to contain the error message, if the test does not succeed.

Note

- Set `full_checking` to `false` to speed-up the 3d object checking. In this case, only the size of each 3d object component is checked.
- Size of the string `error_message` should be at least 128-bytes long, to be able to contain the error message.

8.1.4.178 `bool is_CImg3d (const bool full_check=true, char *const error_message=0) const`

Test if image instance represents a valid serialization of a 3d object.

Return `true` if the image instance represents a valid serialization of a 3d object, and `false` otherwise.

Parameters

	<i>full_check</i>	Tells if full checking of the instance must be performed.
out	<i>error_message</i>	C-string to contain the error message, if the test does not succeed.

Note

- Set `full_check` to `false` to speed-up the 3d object checking. In this case, only the size of each 3d object component is checked.
- Size of the string `error_message` should be at least 128-bytes long, to be able to contain the error message.

8.1.4.179 CImg<T>& sqr ()

Compute the square value of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its square value $I_{(x,y,z,c)}^2$.

Note

- The **[in-place version]** of this method statically casts the computed values to the pixel type `T`.
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type `T` is *not* float-valued.

Example

```
const CImg<float> img("reference.jpg");
(img,img.get_sqr().normalize(0,255)).display();
```

8.1.4.180 CImg<T>& sqrt ()

Compute the square root of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its square root $\sqrt{I_{(x,y,z,c)}}$.

Note

- The **[in-place version]** of this method statically casts the computed values to the pixel type `T`.
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type `T` is *not* float-valued.

Example

```
const CImg<float> img("reference.jpg");
(img,img.get_sqrt().normalize(0,255)).display();
```

8.1.4.181 CImg<T>& exp ()

Compute the exponential of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its exponential $e^{I_{(x,y,z,c)}}$.

Note

- The **[in-place version]** of this method statically casts the computed values to the pixel type T .
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued.

8.1.4.182 CImg<T>& log ()

Compute the logarithm of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its logarithm $\log_e(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this method statically casts the computed values to the pixel type T .
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued.

8.1.4.183 CImg<T>& log2 ()

Compute the base-2 logarithm of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its base-2 logarithm $\log_2(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this method statically casts the computed values to the pixel type T .
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued.

8.1.4.184 CImg<T>& log10 ()

Compute the base-10 logarithm of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its base-10 logarithm $\log_{10}(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this method statically casts the computed values to the pixel type T .
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued.

8.1.4.185 CImg<T>& abs ()

Compute the absolute value of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its absolute value $|I_{(x,y,z,c)}|$.

Note

- The **[in-place version]** of this method statically casts the computed values to the pixel type T .
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued.

8.1.4.186 CImg<T>& sign ()

Compute the sign of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its sign $\text{sign}(I_{(x,y,z,c)})$.

Note

- The sign is set to:
 - 1 if pixel value is strictly positive.
 - -1 if pixel value is strictly negative.
 - 0 if pixel value is equal to 0.
- The **[in-place version]** of this method statically casts the computed values to the pixel type T .
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued.

8.1.4.187 CImg<T>& cos ()

Compute the cosine of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its cosine $\cos(I_{(x,y,z,c)})$.

Note

- Pixel values are regarded as being in *radian*.
- The **[in-place version]** of this method statically casts the computed values to the pixel type T .
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued.

8.1.4.188 CImg<T>& sin ()

Compute the sine of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its sine $\sin(I_{(x,y,z,c)})$.

Note

- Pixel values are regarded as being in *radian*.
- The **[in-place version]** of this method statically casts the computed values to the pixel type T .
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued.

8.1.4.189 CImg<T>& sinc ()

Compute the sinc of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its sinc $\text{sinc}(I_{(x,y,z,c)})$.

Note

- Pixel values are regarded as being exin *radian*.
- The **[in-place version]** of this method statically casts the computed values to the pixel type T.
- The **[new-instance version]** returns a CImg<float> image, if the pixel type T is *not* float-valued.

8.1.4.190 CImg<T>& tan ()

Compute the tangent of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its tangent $\tan(I_{(x,y,z,c)})$.

Note

- Pixel values are regarded as being exin *radian*.
- The **[in-place version]** of this method statically casts the computed values to the pixel type T.
- The **[new-instance version]** returns a CImg<float> image, if the pixel type T is *not* float-valued.

8.1.4.191 CImg<T>& cosh ()

Compute the hyperbolic cosine of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its hyperbolic cosine $\cosh(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this method statically casts the computed values to the pixel type T.
- The **[new-instance version]** returns a CImg<float> image, if the pixel type T is *not* float-valued.

8.1.4.192 CImg<T>& sinh ()

Compute the hyperbolic sine of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its hyperbolic sine $\sinh(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this method statically casts the computed values to the pixel type T.
- The **[new-instance version]** returns a CImg<float> image, if the pixel type T is *not* float-valued.

8.1.4.193 CImg<T>& tanh ()

Compute the hyperbolic tangent of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its hyperbolic tangent $\tanh(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this method statically casts the computed values to the pixel type T .
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued.

8.1.4.194 CImg<T>& acos ()

Compute the arccosine of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its arccosine $\text{acos}(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this method statically casts the computed values to the pixel type T .
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued.

8.1.4.195 CImg<T>& asin ()

Compute the arcsine of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its arcsine $\text{asin}(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this method statically casts the computed values to the pixel type T .
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued.

8.1.4.196 CImg<T>& atan ()

Compute the arctangent of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its arctangent $\text{atan}(I_{(x,y,z,c)})$.

Note

- The **[in-place version]** of this method statically casts the computed values to the pixel type T .
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued.

8.1.4.197 CImg<T>& atan2 (const CImg< t > & img)

Compute the arctangent2 of each pixel value.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its arctangent2 $\text{atan2}(I_{(x,y,z,c)})$.

Parameters

<i>img</i>	Image whose pixel values specify the second argument of the <code>atan2()</code> function.
------------	--

Note

- The **[in-place version]** of this method statically casts the computed values to the pixel type T.
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type T is *not* float-valued.

Example

```
const CImg<float>
img_x(100,100,1,1,"x-w/2",false), // Define an horizontal centered gradient, from '-width/2' to
'width/2'.
img_y(100,100,1,1,"y-h/2",false), // Define a vertical centered gradient, from '-height/2' to
'height/2'.
img_atan2 = img_y.get_atan2(img_x); // Compute atan2(y,x) for each pixel value.
(img_x,img_y,img_atan2).display();
```

8.1.4.198 CImg<T>& mul (const CImg< t > & img)

In-place pointwise multiplication.

Compute the pointwise multiplication between the image instance and the specified input image `img`.

Parameters

<i>img</i>	Input image, as the second operand of the multiplication.
------------	---

Note

- Similar to `operator+=(const CImg<t>&)`, except that it performs a pointwise multiplication instead of an addition.
- It does *not* perform a *matrix* multiplication. For this purpose, use `operator*=(const CImg<t>&)` instead.

Example

```
CImg<float>
img("reference.jpg"),
shade(img.width,img.height(),1,1,"-(x-w/2)^2-(y-h/2)^2",false);
shade.normalize(0,1);
(img,shade,img.get_mul(shade)).display();
```

8.1.4.199 CImg<T>& div (const CImg< t > & img)

In-place pointwise division.

Similar to `mul(const CImg<t>&)`, except that it performs a pointwise division instead of a multiplication.

8.1.4.200 CImg<T>& pow (const double p)

Raise each pixel value to a specified power.

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by its power $I_{(x,y,z,c)}^p$.

Parameters

p	Exponent value.
-----	-----------------

Note

- The **[in-place version]** of this method statically casts the computed values to the pixel type `T`.
- The **[new-instance version]** returns a `CImg<float>` image, if the pixel type `T` is *not* float-valued.

Example

```
const CImg<float>
img0("reference.jpg"),           // Load reference color image.
img1 = (img0/255).pow(1.8)*=255, // Compute gamma correction, with gamma = 1.8.
img2 = (img0/255).pow(0.5)*=255; // Compute gamma correction, with gamma = 0.5.
(img0, img1, img2).display();
```

8.1.4.201 `CImg<T>& pow (const char *const expression)`

Raise each pixel value to a power, specified from an expression.

Similar to `operator+=(const char*)`, except it performs a pointwise exponentiation instead of an addition.

8.1.4.202 `CImg<T>& pow (const CImg< t > & img)`

Raise each pixel value to a power, pointwisely specified from another image.

Similar to `operator+=(const CImg<t>& img)`, except that it performs an exponentiation instead of an addition.

8.1.4.203 `CImg<T>& rol (const unsigned int n = 1)`

Compute the bitwise left rotation of each pixel value.

Similar to `operator<<=(unsigned int)`, except that it performs a left rotation instead of a left shift.

8.1.4.204 `CImg<T>& rol (const char *const expression)`

Compute the bitwise left rotation of each pixel value.

Similar to `operator<<=(const char*)`, except that it performs a left rotation instead of a left shift.

8.1.4.205 `CImg<T>& rol (const CImg< t > & img)`

Compute the bitwise left rotation of each pixel value.

Similar to `operator<<=(const CImg<t>&)`, except that it performs a left rotation instead of a left shift.

8.1.4.206 CImg<T>&ror (const unsigned int $n = 1$)

Compute the bitwise right rotation of each pixel value.

Similar to `operator>>=(unsigned int)`, except that it performs a right rotation instead of a right shift.

8.1.4.207 CImg<T>&ror (const char *const expression)

Compute the bitwise right rotation of each pixel value.

Similar to `operator>>=(const char*)`, except that it performs a right rotation instead of a right shift.

8.1.4.208 CImg<T>&ror (const CImg< t > &img)

Compute the bitwise right rotation of each pixel value.

Similar to `operator>>=(const CImg<t>&)`, except that it performs a right rotation instead of a right shift.

8.1.4.209 CImg<T>&min (const T &val)

Pointwise min operator between instance image and a value.

Parameters

<i>val</i>	Value used as the reference argument of the min operator.
------------	---

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\min(I_{(x,y,z,c)}, \text{val})$.

8.1.4.210 CImg<T>&min (const CImg< t > &img)

Pointwise min operator between two images.

Parameters

<i>img</i>	Image used as the reference argument of the min operator.
------------	---

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\min(I_{(x,y,z,c)}, \text{img}_{(x,y,z,c)})$.

8.1.4.211 CImg<T>&min (const char *const expression)

Pointwise min operator between an image and an expression.

Parameters

<i>expression</i>	Math formula as a C-string.
-------------------	-----------------------------

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\min(I_{(x,y,z,c)}, \text{expr}_{(x,y,z,c)})$.

8.1.4.212 **CImg<T>& max (const T & val)**

Pointwise max operator between instance image and a value.

Parameters

<i>val</i>	Value used as the reference argument of the max operator.
------------	---

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\max(I_{(x,y,z,c)}, \text{val})$.

8.1.4.213 **CImg<T>& max (const CImg< t > & img)**

Pointwise max operator between two images.

Parameters

<i>img</i>	Image used as the reference argument of the max operator.
------------	---

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\max(I_{(x,y,z,c)}, \text{img}_{(x,y,z,c)})$.

8.1.4.214 **CImg<T>& max (const char *const expression)**

Pointwise max operator between an image and an expression.

Parameters

<i>expression</i>	Math formula as a C-string.
-------------------	-----------------------------

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\max(I_{(x,y,z,c)}, \text{expr}_{(x,y,z,c)})$.

8.1.4.215 T& min_max (t & max_val)

Return a reference to the minimum pixel value as well as the maximum pixel value.

Parameters

out	max_val	Maximum pixel value.
-----	---------	----------------------

8.1.4.216 T& max_min (t & min_val)

Return a reference to the maximum pixel value as well as the minimum pixel value.

Parameters

out	min_val	Minimum pixel value.
-----	---------	----------------------

8.1.4.217 T kth_smallest (const unsigned int k) const

Return the kth smallest pixel value.

Parameters

k	Rank of the search smallest element.
---	--------------------------------------

8.1.4.218 double variance (const unsigned int variance_method = 1) const

Return the variance of the pixel values.

Parameters

variance_method	<p>Method used to estimate the variance. Can be:</p> <ul style="list-style-type: none"> • 0: Second moment, computed as $1/N \sum_{k=1}^N (x_k - \bar{x})^2 = 1/N \left(\sum_{k=1}^N x_k^2 - \left(\sum_{k=1}^N x_k \right)^2 / N \right)$ with $\bar{x} = 1/N \sum_{k=1}^N x_k$. • 1: Best unbiased estimator, computed as $\frac{1}{N-1} \sum_{k=1}^N (x_k - \bar{x})^2$. • 2: Least median of squares. • 3: Least trimmed of squares.
-----------------	---

8.1.4.219 double variance_mean (const unsigned int variance_method, t & mean) const

Return the variance as well as the average of the pixel values.

Parameters

	<i>variance_method</i>	Method used to estimate the variance (see variance(const unsigned int) const).
out	<i>mean</i>	Average pixel value.

8.1.4.220 `double variance_noise (const unsigned int variance_method = 2) const`

Return estimated variance of the noise.

Parameters

<i>variance_method</i>	Method used to compute the variance (see variance(const unsigned int) const).
------------------------	--

Note

Because of structures such as edges in images it is recommended to use a robust variance estimation. The variance of the noise is estimated by computing the variance of the Laplacian $(\Delta I)^2$ scaled by a factor c insuring $cE[(\Delta I)^2] = \sigma^2$ where σ is the noise variance.

8.1.4.221 `double MSE (const CImg< t > & img) const`

Compute the MSE (Mean-Squared Error) between two images.

Parameters

<i>img</i>	Image used as the second argument of the MSE operator.
------------	--

8.1.4.222 `double PSNR (const CImg< t > & img, const double max_value = 255) const`

Compute the PSNR (Peak Signal-to-Noise Ratio) between two images.

Parameters

<i>img</i>	Image used as the second argument of the PSNR operator.
<i>max_value</i>	Maximum theoretical value of the signal.

8.1.4.223 `double eval (const char *const expression, const double x = 0, const double y = 0, const double z = 0, const double c = 0, const CImgList< T > *const list_inputs = 0, CImgList< T > *const list_outputs = 0)`

Evaluate math formula.

Parameters

<i>expression</i>	Math formula, as a C-string.
-------------------	------------------------------

Parameters

<i>x</i>	Value of the pre-defined variable <i>x</i> .
<i>y</i>	Value of the pre-defined variable <i>y</i> .
<i>z</i>	Value of the pre-defined variable <i>z</i> .
<i>c</i>	Value of the pre-defined variable <i>c</i> .
<i>list_inputs</i>	A list of input images attached to the specified math formula.
<i>list_outputs</i>	A pointer to a list of output images attached to the specified math formula.

8.1.4.224 `void eval (Clmg< t > & output, const char *const expression, const double x = 0, const double y = 0, const double z = 0, const double c = 0, const ClmgList< T > *const list_inputs = 0, ClmgList< T > *const list_outputs = 0)`

Evaluate math formula.

Parameters

out	<i>output</i>	Contains values of output vector returned by the evaluated expression (or is empty if the returned type is scalar).
	<i>expression</i>	Math formula, as a C-string.
	<i>x</i>	Value of the pre-defined variable <i>x</i> .
	<i>y</i>	Value of the pre-defined variable <i>y</i> .
	<i>z</i>	Value of the pre-defined variable <i>z</i> .
	<i>c</i>	Value of the pre-defined variable <i>c</i> .
	<i>list_inputs</i>	A list of input images attached to the specified math formula.
	<i>list_outputs</i>	A pointer to a list of output images attached to the specified math formula.

8.1.4.225 `Clmg<doubleT> eval (const char *const expression, const Clmg< t > & xyzc, const ClmgList< T > *const list_inputs = 0, ClmgList< T > *const list_outputs = 0)`

Evaluate math formula on a set of variables.

Parameters

<i>expression</i>	Math formula, as a C-string.
<i>xyzc</i>	Set of values (x,y,z,c) used for the evaluation.

8.1.4.226 `double magnitude (const int magnitude_type = 2) const`

Compute norm of the image, viewed as a matrix.

Parameters

<i>magnitude_type</i>	Norm type. Can be: <ul style="list-style-type: none"> • -1: L∞-norm • 0: L2-norm • 1: L1-norm
-----------------------	--

8.1.4.227 `double dot (const CImg< t > & img) const`

Compute the dot product between instance and argument, viewed as matrices.

Parameters

<i>img</i>	Image used as a second argument of the dot product.
------------	---

8.1.4.228 `CImg<T> get_vector_at (const unsigned int x, const unsigned int y = 0, const unsigned int z = 0) const`

Get vector-valued pixel located at specified position.

Parameters

<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.

8.1.4.229 `CImg<T> get_matrix_at (const unsigned int x = 0, const unsigned int y = 0, const unsigned int z = 0) const`

Get (square) matrix-valued pixel located at specified position.

Parameters

<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.

Note

- The [spectrum\(\)](#) of the image must be a square.

8.1.4.230 `CImg<T> get_tensor_at (const unsigned int x, const unsigned int y = 0, const unsigned int z = 0) const`

Get tensor-valued pixel located at specified position.

Parameters

<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.

8.1.4.231 CImg<T>& set_vector_at (const CImg< t > & *vec*, const unsigned int *x*, const unsigned int *y* = 0, const unsigned int *z* = 0)

Set vector-valued pixel at specified position.

Parameters

<i>vec</i>	Vector to put on the instance image.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.

8.1.4.232 CImg<T>& set_matrix_at (const CImg< t > & *mat*, const unsigned int *x* = 0, const unsigned int *y* = 0, const unsigned int *z* = 0)

Set (square) matrix-valued pixel at specified position.

Parameters

<i>mat</i>	Matrix to put on the instance image.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.

8.1.4.233 CImg<T>& set_tensor_at (const CImg< t > & *ten*, const unsigned int *x* = 0, const unsigned int *y* = 0, const unsigned int *z* = 0)

Set tensor-valued pixel at specified position.

Parameters

<i>ten</i>	Tensor to put on the instance image.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.

8.1.4.234 CImg<T>& vector ()

Unroll pixel values along axis *y*.

Note

Equivalent to

```
unroll('y');
```

.

8.1.4.235 CImg<T>& diagonal ()

Resize image to become a diagonal matrix.

Note

Transform the image as a diagonal matrix so that each of its initial value becomes a diagonal coefficient.

8.1.4.236 CImg<T>& identity_matrix ()

Replace the image by an identity matrix.

Note

If the instance image is not square, it is resized to a square matrix using its maximum dimension as a reference.

8.1.4.237 CImg<T>& sequence (const T & a0, const T & a1)

Fill image with a linear sequence of values.

Parameters

<i>a0</i>	Starting value of the sequence.
<i>a1</i>	Ending value of the sequence.

8.1.4.238 CImg<T>& transpose ()

Transpose the image, viewed as a matrix.

Note

Equivalent to

```
permute_axes("yxzc");
```

8.1.4.239 CImg<T>& cross (const CImg< t > & img)

Compute the cross product between two 1×3 images, viewed as 3d vectors.

Parameters

<i>img</i>	Image used as the second argument of the cross product.
------------	---

Note

The first argument of the cross product is **this*.

8.1.4.240 CImg<T>& invert (const bool use_LU = true)

Invert the instance image, viewed as a matrix.

Parameters

<i>use_LU</i>	Choose the inverting algorithm. Can be: <ul style="list-style-type: none"> • <i>true</i>: LU-based matrix inversion. • <i>false</i>: SVD-based matrix inversion.
---------------	--

8.1.4.241 CImg<T>& solve (const CImg< t > & A)

Solve a system of linear equations.

Parameters

<i>A</i>	Matrix of the linear system.
----------	------------------------------

Note

Solve $AX=B$ where $B=*this$.

8.1.4.242 CImg<T>& solve_tridiagonal (const CImg< t > & A)

Solve a tridiagonal system of linear equations.

Parameters

<i>A</i>	Coefficients of the tridiagonal system. A is a tridiagonal matrix $A = [b_0, c_0, 0, \dots; a_1, b_1, c_1, 0, \dots; \dots; \dots, 0, a_N, b_N]$, stored as a 3 columns matrix
----------	---

Note

Solve $AX=B$ where $B=*this$, using the Thomas algorithm.

8.1.4.243 `const CImg<T>& eigen (CImg< t > & val, CImg< t > & vec) const`

Compute eigenvalues and eigenvectors of the instance image, viewed as a matrix.

Parameters

out	<i>val</i>	Vector of the estimated eigenvalues, in decreasing order.
out	<i>vec</i>	Matrix of the estimated eigenvectors, sorted by columns.

8.1.4.244 `CImgList<Tfloat> get_eigen () const`

Compute eigenvalues and eigenvectors of the instance image, viewed as a matrix.

Returns

A list of two images [*val*; *vec*], whose meaning is similar as in [eigen\(CImg<t>&,CImg<t>&\) const](#).

8.1.4.245 `const CImg<T>& symmetric_eigen (CImg< t > & val, CImg< t > & vec) const`

Compute eigenvalues and eigenvectors of the instance image, viewed as a symmetric matrix.

Parameters

out	<i>val</i>	Vector of the estimated eigenvalues, in decreasing order.
out	<i>vec</i>	Matrix of the estimated eigenvectors, sorted by columns.

8.1.4.246 `CImgList<Tfloat> get_symmetric_eigen () const`

Compute eigenvalues and eigenvectors of the instance image, viewed as a symmetric matrix.

Returns

A list of two images [*val*; *vec*], whose meaning are similar as in [symmetric_eigen\(CImg<t>&,CImg<t>&\) const](#).

8.1.4.247 `CImg<T>& sort (CImg< t > & permutations, const bool is_increasing =true)`

Sort pixel values and get sorting permutations.

Parameters

out	<i>permutations</i>	Permutation map used for the sorting.
	<i>is_increasing</i>	Tells if pixel values are sorted in an increasing (<code>true</code>) or decreasing (<code>false</code>) way.

8.1.4.248 CImg<T>& sort (const bool *is_increasing* = `true`, const char *axis* = 0)

Sort pixel values.

Parameters

<i>is_increasing</i>	Tells if pixel values are sorted in an increasing (<code>true</code>) or decreasing (<code>false</code>) way.
<i>axis</i>	Tells if the value sorting must be done along a specific axis. Can be: <ul style="list-style-type: none"> • 0: All pixel values are sorted, independently on their initial position. • 'x': Image columns are sorted, according to the first value in each column. • 'y': Image rows are sorted, according to the first value in each row. • 'z': Image slices are sorted, according to the first value in each slice. • 'c': Image channels are sorted, according to the first value in each channel.

8.1.4.249 const CImg<T>& SVD (CImg< t > & U, CImg< t > & S, CImg< t > & V, const bool *sorting* = `true`, const unsigned int *max_iteration* = 40, const float *lambda* = 0) const

Compute the SVD of the instance image, viewed as a general matrix.

Compute the SVD decomposition `*this=U*S*V'` where `U` and `V` are orthogonal matrices and `S` is a diagonal matrix. `V'` denotes the matrix transpose of `V`.

Parameters

out	<i>U</i>	First matrix of the SVD product.
out	<i>S</i>	Coefficients of the second (diagonal) matrix of the SVD product. These coefficients are stored as a vector.
out	<i>V</i>	Third matrix of the SVD product.
	<i>sorting</i>	Tells if the diagonal coefficients are sorted (in decreasing order).
	<i>max_iteration</i>	Maximum number of iterations considered for the algorithm convergence.
	<i>lambda</i>	Epsilon used for the algorithm convergence.

Note

The instance matrix can be computed from `U`, `S` and `V` by

```
const CImg<> A; // Input matrix (assumed to contain some values).
CImg<> U, S, V;
A.SVD(U, S, V)
```

8.1.4.250 `CImgList<Tfloat> get_SVD (const bool sorting = true, const unsigned int max_iteration = 40, const float lambda = 0) const`

Compute the SVD of the instance image, viewed as a general matrix.

Returns

A list of three images [U; S; V], whose meaning is similar as in [SVD\(CImg<t>&,CImg<t>&,CImg<t>&,bool,unsigned int,float\) const](#).

8.1.4.251 `static CImg<T> dijkstra (const tf & distance, const unsigned int nb_nodes, const unsigned int starting_node, const unsigned int ending_node, CImg<t> & previous_node) [static]`

Compute minimal path in a graph, using the Dijkstra algorithm.

Parameters

<i>distance</i>	An object having operator()(unsigned int i, unsigned int j) which returns distance between two nodes (i,j).
<i>nb_nodes</i>	Number of graph nodes.
<i>starting_node</i>	Indice of the starting node.
<i>ending_node</i>	Indice of the ending node (set to ~0U to ignore ending node).
<i>previous_node</i>	Array that gives the previous node indice in the path to the starting node (optional parameter).

Returns

Array of distances of each node to the starting node.

8.1.4.252 `CImg<T>& dijkstra (const unsigned int starting_node, const unsigned int ending_node, CImg<t> & previous_node)`

Return minimal path in a graph, using the Dijkstra algorithm.

Parameters

<i>starting_node</i>	Indice of the starting node.
<i>ending_node</i>	Indice of the ending node.
<i>previous_node</i>	Array that gives the previous node indice in the path to the starting node (optional parameter).

Returns

Array of distances of each node to the starting node.

Note

image instance corresponds to the adjacency matrix of the graph.

8.1.4.253 `static CImg<T> string (const char *const str, const bool is_last_zero = true, const bool is_shared = false)`
`[static]`

Return an image containing the ascii codes of the specified string.

Parameters

<i>str</i>	input C-string to encode as an image.
<i>is_last_zero</i>	Tells if the ending '0' character appear in the resulting image.

8.1.4.254 `static CImg<T> vector (const T & a0)` `[static]`

Return a 1x1 image containing specified value.

Parameters

<i>a0</i>	First vector value.
-----------	---------------------

8.1.4.255 `static CImg<T> vector (const T & a0, const T & a1)` `[static]`

Return a 1x2 image containing specified values.

Parameters

<i>a0</i>	First vector value.
<i>a1</i>	Second vector value.

8.1.4.256 `static CImg<T> vector (const T & a0, const T & a1, const T & a2)` `[static]`

Return a 1x3 image containing specified values.

Parameters

<i>a0</i>	First vector value.
<i>a1</i>	Second vector value.
<i>a2</i>	Third vector value.

8.1.4.257 `static CImg<T> vector (const T & a0, const T & a1, const T & a2, const T & a3)` `[static]`

Return a 1x4 image containing specified values.

Parameters

<i>a0</i>	First vector value.
-----------	---------------------

Parameters

<i>a1</i>	Second vector value.
<i>a2</i>	Third vector value.
<i>a3</i>	Fourth vector value.

8.1.4.258 `static CImg<T> matrix (const T & a0) [static]`

Return a 1x1 matrix containing specified coefficients.

Parameters

<i>a0</i>	First matrix value.
-----------	---------------------

Note

Equivalent to `vector(const T&)`.

8.1.4.259 `static CImg<T> matrix (const T & a0, const T & a1, const T & a2, const T & a3) [static]`

Return a 2x2 matrix containing specified coefficients.

Parameters

<i>a0</i>	First matrix value.
<i>a1</i>	Second matrix value.
<i>a2</i>	Third matrix value.
<i>a3</i>	Fourth matrix value.

8.1.4.260 `static CImg<T> matrix (const T & a0, const T & a1, const T & a2, const T & a3, const T & a4, const T & a5, const T & a6, const T & a7, const T & a8) [static]`

Return a 3x3 matrix containing specified coefficients.

Parameters

<i>a0</i>	First matrix value.
<i>a1</i>	Second matrix value.
<i>a2</i>	Third matrix value.
<i>a3</i>	Fourth matrix value.
<i>a4</i>	Fifth matrix value.
<i>a5</i>	Sixth matrix value.
<i>a6</i>	Seventh matrix value.
<i>a7</i>	Eighth matrix value.
<i>a8</i>	Nineth matrix value.

8.1.4.261 `static CImg<T> tensor (const T & a0) [static]`

Return a 1x1 symmetric matrix containing specified coefficients.

Parameters

<i>a0</i>	First matrix value.
-----------	---------------------

Note

Equivalent to `vector(const T&)`.

8.1.4.262 `static CImg<T> identity_matrix (const unsigned int N) [static]`

Return a NxN identity matrix.

Parameters

<i>N</i>	Dimension of the matrix.
----------	--------------------------

8.1.4.263 `static CImg<T> sequence (const unsigned int N, const T & a0, const T & a1) [static]`

Return a N-numbered sequence vector from a0 to a1.

Parameters

<i>N</i>	Size of the resulting vector.
<i>a0</i>	Starting value of the sequence.
<i>a1</i>	Ending value of the sequence.

8.1.4.264 `static CImg<T> rotation_matrix (const float x, const float y, const float z, const float w, const bool is_quaternion = false) [static]`

Return a 3x3 rotation matrix along the (x,y,z)-axis with an angle w.

Parameters

<i>x</i>	X-coordinate of the rotation axis, or first quaternion coordinate.
<i>y</i>	Y-coordinate of the rotation axis, or second quaternion coordinate.
<i>z</i>	Z-coordinate of the rotation axis, or third quaternion coordinate.
<i>w</i>	Angle of the rotation axis, or fourth quaternion coordinate.
<i>is_quaternion</i>	Tell is the four arguments denotes a set { axis + angle } or a quaternion.

8.1.4.265 CImg<T>& fill (const T & val)

Fill all pixel values with specified value.

Parameters

<i>val</i>	Fill value.
------------	-------------

8.1.4.266 CImg<T>& fill (const T & val0, const T & val1)

Fill sequentially all pixel values with specified values.

Parameters

<i>val0</i>	First fill value.
<i>val1</i>	Second fill value.

8.1.4.267 CImg<T>& fill (const char *const expression, const bool repeat_values, const bool allow_formula = true, const CImgList< T > *const list_inputs = 0, CImgList< T > *const list_outputs = 0)

Fill sequentially pixel values according to a given expression.

Parameters

<i>expression</i>	C-string describing a math formula, or a list of values.
<i>repeat_values</i>	In case a list of values is provided, tells if this list must be repeated for the filling.
<i>allow_formula</i>	tells if a formula is allowed or only a list of values.
<i>list_inputs</i>	In case of a mathematical expression, attach a list of images to the specified expression.
<i>list_outputs</i>	In case of a mathematical expression, attach a list of images to the specified expression.

8.1.4.268 CImg<T>& fill (const CImg<t> & values, const bool repeat_values = true)

Fill sequentially pixel values according to the values found in another image.

Parameters

<i>values</i>	Image containing the values used for the filling.
<i>repeat_values</i>	In case there are less values than necessary in <i>values</i> , tells if these values must be repeated for the filling.

8.1.4.269 CImg<T>& fillX (const unsigned int y, const unsigned int z, const unsigned int c, const int a0, ...)

Fill pixel values along the X-axis at a specified pixel position.

Parameters

<i>y</i>	Y-coordinate of the filled column.
<i>z</i>	Z-coordinate of the filled column.
<i>c</i>	C-coordinate of the filled column.
<i>a0</i>	First fill value.

8.1.4.270 Clmg<T>& fillY (const unsigned int *x*, const unsigned int *z*, const unsigned int *c*, const int *a0*, ...)

Fill pixel values along the Y-axis at a specified pixel position.

Parameters

<i>x</i>	X-coordinate of the filled row.
<i>z</i>	Z-coordinate of the filled row.
<i>c</i>	C-coordinate of the filled row.
<i>a0</i>	First fill value.

8.1.4.271 Clmg<T>& fillZ (const unsigned int *x*, const unsigned int *y*, const unsigned int *c*, const int *a0*, ...)

Fill pixel values along the Z-axis at a specified pixel position.

Parameters

<i>x</i>	X-coordinate of the filled slice.
<i>y</i>	Y-coordinate of the filled slice.
<i>c</i>	C-coordinate of the filled slice.
<i>a0</i>	First fill value.

8.1.4.272 Clmg<T>& fillC (const unsigned int *x*, const unsigned int *y*, const unsigned int *z*, const int *a0*, ...)

Fill pixel values along the C-axis at a specified pixel position.

Parameters

<i>x</i>	X-coordinate of the filled channel.
<i>y</i>	Y-coordinate of the filled channel.
<i>z</i>	Z-coordinate of the filled channel.
<i>a0</i>	First filling value.

8.1.4.273 Clmg<T>& discard (const Clmg< t > & *values*, const char *axis* = 0)

Discard specified sequence of values in the image buffer, along a specific axis.

Parameters

<i>values</i>	Sequence of values to discard.
<i>axis</i>	Axis along which the values are discarded. If set to 0 (default value) the method does it for all the buffer values and returns a one-column vector.

Note

Discarded values will change the image geometry, so the resulting image is returned as a one-column vector.

8.1.4.274 `CImg<T>& rand (const T & val_min, const T & val_max)`

Fill image with random values in specified range.

Parameters

<i>val_min</i>	Minimal authorized random value.
<i>val_max</i>	Maximal authorized random value.

Note

Random variables are uniformly distributed in [val_min, val_max].

8.1.4.275 `CImg<T>& round (const double y = 1, const int rounding_type = 0)`

Round pixel values.

Parameters

<i>y</i>	Rounding precision.
<i>rounding_type</i>	Rounding type. Can be: <ul style="list-style-type: none"> • -1: Backward. • 0: Nearest. • 1: Forward.

8.1.4.276 `CImg<T>& noise (const double sigma, const unsigned int noise_type = 0)`

Add random noise to pixel values.

Parameters

<i>sigma</i>	Amplitude of the random additive noise. If <i>sigma</i> <0, it stands for a percentage of the global value range.
--------------	---

Parameters

<i>noise_type</i>	Type of additive noise (can be 0=gaussian, 1=uniform, 2=Salt and Pepper, 3=Poisson or 4=Rician).
-------------------	--


Returns

A reference to the modified image instance.

Note

- For Poisson noise (*noise_type*=3), parameter *sigma* is ignored, as Poisson noise only depends on the image value itself.
- Function `CImg<T>::get_noise()` is also defined. It returns a non-shared modified copy of the image instance.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_noise(40);
 (img, res.normalize(0, 255)). ();
```



8.1.4.277 CImg<T>& normalize (const T & min_value, const T & max_value)

Linearly normalize pixel values.

Parameters

<i>min_value</i>	Minimum desired value of the resulting image.
<i>max_value</i>	Maximum desired value of the resulting image.



Example

```
const CImg<float> img("reference.jpg"), res = img.get_normalize(160, 220);
 (img, res). ();
```

8.1.4.278 CImg<T>& normalize ()

Normalize multi-valued pixels of the image instance, with respect to their L2-norm.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_normalize();
 (img, res.normalize(0, 255)). ();
```


8.1.4.279 CImg<T>& norm (const int norm_type = 2)

Compute Lp-norm of each multi-valued pixel of the image instance.

Parameters

<i>norm_type</i>	Type of computed vector norm (can be $-1=L_{inf}$, or ≥ 0).
------------------	--

Example

```
const CImg<float> img("reference.jpg"), res = img.get_norm();
 (img, res.normalize(0, 255));
```


8.1.4.280 CImg<T>& cut (const T & min_value, const T & max_value)

Cut pixel values in specified range.

Parameters

<i>min_value</i>	Minimum desired value of the resulting image.
<i>max_value</i>	Maximum desired value of the resulting image.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_cut(160, 220);
 (img, res);
```

8.1.4.281 CImg<T>& quantize (const unsigned int nb_levels, const bool keep_range = true)

Uniformly quantize pixel values.

Parameters

<i>nb_levels</i>	Number of quantization levels.
<i>keep_range</i>	Tells if resulting values keep the same range as the original ones.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_quantize(4);
 (img, res);
```

8.1.4.282 CImg<T>& threshold (const T & value, const bool soft_threshold = false, const bool strict_threshold = false)

Threshold pixel values.

Parameters

<i>value</i>	Threshold value
<i>soft_threshold</i>	Tells if soft thresholding must be applied (instead of hard one).
<i>strict_threshold</i>	Tells if threshold value is strict.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_threshold(128);
(img, res.normalize(0, 255)).display();
```

8.1.4.283 CImg<T>& histogram (const unsigned int *nb_levels*, const T & *min_value*, const T & *max_value*)

Compute the histogram of pixel values.

Parameters

<i>nb_levels</i>	Number of desired histogram levels.
<i>min_value</i>	Minimum pixel value considered for the histogram computation. All pixel values lower than <i>min_value</i> will not be counted.
<i>max_value</i>	Maximum pixel value considered for the histogram computation. All pixel values higher than <i>max_value</i> will not be counted.

Note

- The histogram *H* of an image *I* is the 1d function where *H*(*x*) counts the number of occurrences of the value *x* in the image *I*.
- The resulting histogram is always defined in 1d. Histograms of multi-valued images are not multi-dimensional.

Example

```
const CImg<float> img = CImg<float>("reference.jpg").histogram(256);
img.display_graph(0, 3);
```

8.1.4.284 CImg<T>& equalize (const unsigned int *nb_levels*, const T & *min_value*, const T & *max_value*)

Equalize histogram of pixel values.

Parameters

<i>nb_levels</i>	Number of histogram levels used for the equalization.
<i>min_value</i>	Minimum pixel value considered for the histogram computation. All pixel values lower than <i>min_value</i> will not be counted.
<i>max_value</i>	Maximum pixel value considered for the histogram computation. All pixel values higher than <i>max_value</i> will not be counted.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_equalize(256);
(img, res).display();
```

8.1.4.285 CImg<T>& index (const CImg< t > & *colormap*, const float *dithering* = 1, const bool *map_indexes* = false)

Index multi-valued pixels regarding to a specified colormap.

Parameters

<i>colormap</i>	Multi-valued colormap used as the basis for multi-valued pixel indexing.
<i>dithering</i>	Level of dithering (0=disable, 1=standard level).
<i>map_indexes</i>	Tell if the values of the resulting image are the colormap indices or the colormap vectors.

Note

- `img.index(colormap, dithering, 1)` is equivalent to `img.index(colormap, dithering, 0).map(colormap)`

Example

```
const CImg<float> img("reference.jpg"), colormap(3,1,1,3, 0,128,255, 0,128,255, 0,128,255);
const CImg<float> res = img.get_index(colormap,1,true);
(img,res).display();
```

8.1.4.286 CImg<T>& map (const CImg< t > & colormap, const unsigned int boundary_conditions = 0)

Map predefined colormap on the scalar (indexed) image instance.

Parameters

<i>colormap</i>	Multi-valued colormap used for mapping the indexes.
<i>boundary_conditions</i>	The border condition type { 0=zero 1=dirichlet 2=periodic }.

Example

```
const CImg<float> img("reference.jpg"),
colormap1(3,1,1,3, 0,128,255, 0,128,255, 0,128,255),
colormap2(3,1,1,3, 255,0,0, 0,255,0, 0,0,255),
res = img.get_index(colormap1,0).map(colormap2);
(img,res).display();
```

8.1.4.287 CImg<T>& label (const bool is_high_connectivity = false, const Tfloat tolerance = 0)

Label connected components.

Parameters

<i>is_high_connectivity</i>	Boolean that choose between 4(false)- or 8(true)-connectivity in 2d case, and between 6(false)- or 26(true)-connectivity in 3d case.
<i>tolerance</i>	Tolerance used to determine if two neighboring pixels belong to the same region.

Note

The algorithm of connected components computation has been primarily done by A. Meijster, according to the publication: 'W.H. Hesselink, A. Meijster, C. Bron, "Concurrent Determination of Connected Components.", In: Science of Computer Programming 41 (2001), pp. 173–194'. The submitted code has then been modified to fit [CImg](#) coding style and constraints.

8.1.4.288 `CImg<T>& label (const CImg< t > & connectivity_mask, const Tfloat tolerance = 0)`

Label connected components **[overloading]**.

Parameters

<i>connectivity_mask</i>	Mask of the neighboring pixels.
<i>tolerance</i>	Tolerance used to determine if two neighboring pixels belong to the same region.

8.1.4.289 `static const CImg<Tuchar>& default_LUT256 () [static]`

Return colormap "*default*", containing 256 colors entries in RGB.

Returns

The following 256x1x1x3 colormap is returned:

8.1.4.290 `static const CImg<Tuchar>& HSV_LUT256 () [static]`

Return colormap "*HSV*", containing 256 colors entries in RGB.

Returns

The following 256x1x1x3 colormap is returned:

8.1.4.291 `static const CImg<Tuchar>& lines_LUT256 () [static]`

Return colormap "*lines*", containing 256 colors entries in RGB.

Returns

The following 256x1x1x3 colormap is returned:

8.1.4.292 `static const CImg<Tuchar>& hot_LUT256 () [static]`

Return colormap "*hot*", containing 256 colors entries in RGB.

Returns

The following 256x1x1x3 colormap is returned:

8.1.4.293 `static const CImg<Tuchar>& cool_LUT256 () [static]`

Return colormap "*cool*", containing 256 colors entries in RGB.

Returns

The following 256x1x1x3 colormap is returned:

8.1.4.294 `static const CImg<Tuchar>& jet_LUT256 () [static]`

Return colormap "*jet*", containing 256 colors entries in RGB.

Returns

The following 256x1x1x3 colormap is returned:

8.1.4.295 `static const CImg<Tuchar>& flag_LUT256 () [static]`

Return colormap "*flag*", containing 256 colors entries in RGB.

Returns

The following 256x1x1x3 colormap is returned:

8.1.4.296 `static const CImg<Tuchar>& cube_LUT256 () [static]`

Return colormap "*cube*", containing 256 colors entries in RGB.

Returns

The following 256x1x1x3 colormap is returned:

8.1.4.297 `CImg<T>& RGBtoXYZ ()`

Convert pixel values from RGB to XYZ_709 color spaces.

Note

Uses the standard D65 white point.

8.1.4.298 `CImg<T>& resize (const int size_x, const int size_y = -100, const int size_z = -100, const int size_c = -100, const int interpolation_type = 1, const unsigned int boundary_conditions = 0, const float centering_x = 0, const float centering_y = 0, const float centering_z = 0, const float centering_c = 0)`

Resize image to new dimensions.

Parameters

<i>size_x</i>	Number of columns (new size along the X-axis).
<i>size_y</i>	Number of rows (new size along the Y-axis).
<i>size_z</i>	Number of slices (new size along the Z-axis).
<i>size_c</i>	Number of vector-channels (new size along the C-axis).
<i>interpolation_type</i>	Method of interpolation: <ul style="list-style-type: none"> • -1 = no interpolation: raw memory resizing. • 0 = no interpolation: additional space is filled according to <i>boundary_conditions</i>. • 1 = nearest-neighbor interpolation. • 2 = moving average interpolation. • 3 = linear interpolation. • 4 = grid interpolation. • 5 = cubic interpolation. • 6 = lanczos interpolation.
<i>boundary_conditions</i>	Border condition type.
<i>centering_x</i>	Set centering type (only if <i>interpolation_type</i> =0).
<i>centering_y</i>	Set centering type (only if <i>interpolation_type</i> =0).
<i>centering_z</i>	Set centering type (only if <i>interpolation_type</i> =0).
<i>centering_c</i>	Set centering type (only if <i>interpolation_type</i> =0).

Note

If $\text{pd}[x,y,z,v] < 0$, it corresponds to a percentage of the original size (the default value is -100).

8.1.4.299 `Clmg<T>& resize (const Clmg< t > & src, const int interpolation_type = 1, const unsigned int boundary_conditions = 0, const float centering_x = 0, const float centering_y = 0, const float centering_z = 0, const float centering_c = 0)`

Resize image to dimensions of another image.

Parameters

<i>src</i>	Reference image used for dimensions.
<i>interpolation_type</i>	Interpolation method.
<i>boundary_conditions</i>	Boundary conditions.
<i>centering_x</i>	Set centering type (only if <i>interpolation_type</i> =0).
<i>centering_y</i>	Set centering type (only if <i>interpolation_type</i> =0).
<i>centering_z</i>	Set centering type (only if <i>interpolation_type</i> =0).
<i>centering_c</i>	Set centering type (only if <i>interpolation_type</i> =0).

8.1.4.300 `CImg<T>& resize (const CImgDisplay & disp, const int interpolation_type = 1, const unsigned int boundary_conditions = 0, const float centering_x = 0, const float centering_y = 0, const float centering_z = 0, const float centering_c = 0)`

Resize image to dimensions of a display window.

Parameters

<i>disp</i>	Reference display window used for dimensions.
<i>interpolation_type</i>	Interpolation method.
<i>boundary_conditions</i>	Boundary conditions.
<i>centering_x</i>	Set centering type (only if <code>interpolation_type=0</code>).
<i>centering_y</i>	Set centering type (only if <code>interpolation_type=0</code>).
<i>centering_z</i>	Set centering type (only if <code>interpolation_type=0</code>).
<i>centering_c</i>	Set centering type (only if <code>interpolation_type=0</code>).

8.1.4.301 `CImg<T>& resize_doubleXY ()`

Resize image to double-size, using the Scale2X algorithm.

Note

Use anisotropic upscaling algorithm [described here](#).

8.1.4.302 `CImg<T>& resize_tripleXY ()`

Resize image to triple-size, using the Scale3X algorithm.

Note

Use anisotropic upscaling algorithm [described here](#).

8.1.4.303 `CImg<T>& mirror (const char axis)`

Mirror image content along specified axis.

Parameters

<i>axis</i>	Mirror axis
-------------	-------------

8.1.4.304 `CImg<T>& mirror (const char *const axes)`

Mirror image content along specified axes.

Parameters

<i>axes</i>	Mirror axes, as a C-string.
-------------	-----------------------------

Note

axes may contains multiple characters, e.g. "xyz"

8.1.4.305 CImg<T>& shift (const int *delta_x*, const int *delta_y* = 0, const int *delta_z* = 0, const int *delta_c* = 0, const int *boundary_conditions* = 0)

Shift image content.

Parameters

<i>delta_x</i>	Amount of displacement along the X-axis.
<i>delta_y</i>	Amount of displacement along the Y-axis.
<i>delta_z</i>	Amount of displacement along the Z-axis.
<i>delta_c</i>	Amount of displacement along the C-axis.
<i>boundary_conditions</i>	Border condition.

- *boundary_conditions* can be:
 - 0: Zero border condition (Dirichlet).
 - 1: Nearest neighbors (Neumann).
 - 2: Repeat Pattern (Fourier style).

8.1.4.306 CImg<T>& permute_axes (const char *const *order*)

Permute axes order.

Parameters

<i>order</i>	Axes permutations, as a C-string of 4 characters. This function permutes image content regarding the specified axes permutation.
--------------	--

8.1.4.307 CImg<T>& unroll (const char *axis*)

Unroll pixel values along specified axis.

Parameters

<i>axis</i>	Unroll axis (can be 'x', 'y', 'z' or c'c').
-------------	---

8.1.4.308 `CImg<T>& rotate (const float angle, const unsigned int interpolation = 1, const unsigned int boundary_conditions = 0)`

Rotate image with arbitrary angle.

Parameters

<i>angle</i>	Rotation angle, in degrees.
<i>interpolation</i>	Type of interpolation. Can be { 0=nearest 1=linear 2=cubic }.
<i>boundary</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann 2=periodic }.

Note

Most of the time, size of the image is modified.

8.1.4.309 `CImg<T>& rotate (const float angle, const float cx, const float cy, const float zoom, const unsigned int interpolation = 1, const unsigned int boundary_conditions = 0)`

Rotate image with arbitrary angle, around a center point.

Parameters

<i>angle</i>	Rotation angle, in degrees.
<i>cx</i>	X-coordinate of the rotation center.
<i>cy</i>	Y-coordinate of the rotation center.
<i>zoom</i>	Zoom factor.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann 2=periodic }.
<i>interpolation_type</i>	Type of interpolation. Can be { 0=nearest 1=linear 2=cubic }.

8.1.4.310 `CImg<T>& warp (const CImg<t> & warp, const unsigned int mode = 0, const unsigned int interpolation = 1, const unsigned int boundary_conditions = 0)`

Warp image content by a warping field.

Parameters

<i>warp</i>	Warping field.
<i>mode</i>	Can be { 0=backward-absolute 1=backward-relative 2=forward-absolute 3=foward-relative }
<i>is_relative</i>	Tells if warping field gives absolute or relative warping coordinates.
<i>interpolation</i>	Can be { 0=nearest 1=linear 2=cubic }.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann 2=periodic }.

8.1.4.311 CImg<T> get_projections2d (const unsigned int x0, const unsigned int y0, const unsigned int z0) const

Generate a 2d representation of a 3d image, with XY,XZ and YZ views.

Parameters

x0	X-coordinate of the projection point.
y0	Y-coordinate of the projection point.
z0	Z-coordinate of the projection point.

8.1.4.312 CImg<T>& crop (const int x0, const int y0, const int z0, const int c0, const int x1, const int y1, const int z1, const int c1, const bool boundary_conditions = false)

Crop image region.

Parameters

x0	= X-coordinate of the upper-left crop rectangle corner.
y0	= Y-coordinate of the upper-left crop rectangle corner.
z0	= Z-coordinate of the upper-left crop rectangle corner.
c0	= C-coordinate of the upper-left crop rectangle corner.
x1	= X-coordinate of the lower-right crop rectangle corner.
y1	= Y-coordinate of the lower-right crop rectangle corner.
z1	= Z-coordinate of the lower-right crop rectangle corner.
c1	= C-coordinate of the lower-right crop rectangle corner.
boundary_conditions	= Dirichlet (false) or Neumann border conditions.

8.1.4.313 CImg<T>& autocrop (const T *const color = 0, const char *const axes = "zyx")

Autocrop image region, regarding the specified background color.

Parameters

color	Color used for the crop. If 0, color is guessed.
axes	Axes used for the crop.

8.1.4.314 CImg<T> get_column (const int x0) const

Return specified image column.

Parameters

x0	Image column.
----	---------------

8.1.4.315 `CImg<T>& columns (const int x0, const int x1)`

Return specified range of image columns.

Parameters

<code>x0</code>	Starting image column.
<code>x1</code>	Ending image column.

8.1.4.316 `CImg<T>& row (const int y0)`

Return specified image row **[in-place version]**.

Parameters

<code>y0</code>	Image row.
-----------------	------------

8.1.4.317 `CImg<T> get_rows (const int y0, const int y1) const`

Return specified range of image rows.

Parameters

<code>y0</code>	Starting image row.
<code>y1</code>	Ending image row.

8.1.4.318 `CImg<T> get_slice (const int z0) const`

Return specified image slice.

Parameters

<code>z0</code>	Image slice.
-----------------	--------------

8.1.4.319 `CImg<T> get_slices (const int z0, const int z1) const`

Return specified range of image slices.

Parameters

<code>z0</code>	Starting image slice.
<code>z1</code>	Ending image slice.

8.1.4.320 CImg<T> get_channel (const int *c0*) const

Return specified image channel.

Parameters

<i>c0</i>	Image channel.
-----------	----------------

8.1.4.321 CImg<T> get_channels (const int *c0*, const int *c1*) const

Return specified range of image channels.

Parameters

<i>c0</i>	Starting image channel.
<i>c1</i>	Ending image channel.

8.1.4.322 static CImg<floatT> streamline (const tfunc & *func*, const float *x*, const float *y*, const float *z*, const float *L* = 256, const float *dl* = 0.1f, const unsigned int *interpolation_type* = 2, const bool *is_backward_tracking* = false, const bool *is_oriented_only* = false, const float *x0* = 0, const float *y0* = 0, const float *z0* = 0, const float *x1* = 0, const float *y1* = 0, const float *z1* = 0) [static]

Return stream line of a 3d vector field.

Parameters

<i>func</i>	Vector field function.
<i>x</i>	X-coordinate of the starting point of the streamline.
<i>y</i>	Y-coordinate of the starting point of the streamline.
<i>z</i>	Z-coordinate of the starting point of the streamline.
<i>L</i>	Streamline length.
<i>dl</i>	Streamline length increment.
<i>interpolation_type</i>	Type of interpolation. Can be { 0=nearest int 1=linear 2=2nd-order RK 3=4th-order RK. }.
<i>is_backward_tracking</i>	Tells if the streamline is estimated forward or backward.
<i>is_oriented_only</i>	Tells if the direction of the vectors must be ignored.
<i>x0</i>	X-coordinate of the first bounding-box vertex.
<i>y0</i>	Y-coordinate of the first bounding-box vertex.
<i>z0</i>	Z-coordinate of the first bounding-box vertex.
<i>x1</i>	X-coordinate of the second bounding-box vertex.
<i>y1</i>	Y-coordinate of the second bounding-box vertex.
<i>z1</i>	Z-coordinate of the second bounding-box vertex.

8.1.4.323 CImg<T> get_shared_points (const unsigned int *x0*, const unsigned int *x1*, const unsigned int *y0* = 0, const unsigned int *z0* = 0, const unsigned int *c0* = 0)

Return a shared-memory image referencing a range of pixels of the image instance.

Parameters

<i>x0</i>	X-coordinate of the starting pixel.
<i>x1</i>	X-coordinate of the ending pixel.
<i>y0</i>	Y-coordinate.
<i>z0</i>	Z-coordinate.
<i>c0</i>	C-coordinate.

8.1.4.324 CImg<T> get_shared_rows (const unsigned int *y0*, const unsigned int *y1*, const unsigned int *z0* = 0, const unsigned int *c0* = 0)

Return a shared-memory image referencing a range of rows of the image instance.

Parameters

<i>y0</i>	Y-coordinate of the starting row.
<i>y1</i>	Y-coordinate of the ending row.
<i>z0</i>	Z-coordinate.
<i>c0</i>	C-coordinate.

8.1.4.325 CImg<T> get_shared_row (const unsigned int *y0*, const unsigned int *z0* = 0, const unsigned int *c0* = 0)

Return a shared-memory image referencing one row of the image instance.

Parameters

<i>y0</i>	Y-coordinate.
<i>z0</i>	Z-coordinate.
<i>c0</i>	C-coordinate.

8.1.4.326 CImg<T> get_shared_slices (const unsigned int *z0*, const unsigned int *z1*, const unsigned int *c0* = 0)

Return a shared memory image referencing a range of slices of the image instance.

Parameters

<i>z0</i>	Z-coordinate of the starting slice.
<i>z1</i>	Z-coordinate of the ending slice.
<i>c0</i>	C-coordinate.

8.1.4.327 CImg<T> get_shared_slice (const unsigned int *z0*, const unsigned int *c0* = 0)

Return a shared-memory image referencing one slice of the image instance.

Parameters

<i>z0</i>	Z-coordinate.
<i>c0</i>	C-coordinate.

8.1.4.328 CImg<T> get_shared_channels (const unsigned int *c0*, const unsigned int *c1*)

Return a shared-memory image referencing a range of channels of the image instance.

Parameters

<i>c0</i>	C-coordinate of the starting channel.
<i>c1</i>	C-coordinate of the ending channel.

8.1.4.329 CImg<T> get_shared_channel (const unsigned int *c0*)

Return a shared-memory image referencing one channel of the image instance.

Parameters

<i>c0</i>	C-coordinate.
-----------	---------------

8.1.4.330 CImgList<T> get_split (const char *axis*, const int *nb* = -1) const

Split image into a list along specified axis.

Parameters

<i>axis</i>	Splitting axis. Can be { 'x' 'y' 'z' 'c' }.
<i>nb</i>	Number of splitted parts.

Note

- If *nb*==0, instance image is splitted into blocs of egal values along the specified axis.
- If *nb*<=0, instance image is splitted into blocs of -*nb* pixel wide.
- If *nb*>0, instance image is splitted into *nb* blocs.

8.1.4.331 CImgList<T> get_split (const CImg< t > & *values*, const char *axis* = 0, const bool *keep_values* = true) const

Split image into a list of sub-images, according to a specified splitting value sequence and optionnally axis.

Parameters

<i>values</i>	Splitting value sequence.
<i>axis</i>	Axis along which the splitting is performed. Can be '0' to ignore axis.
<i>keep_values</i>	Tells if the splitting sequence must be kept in the splitted blocs.

8.1.4.332 `CImg<T>& append (const CImg< t > & img, const char axis = 'x', const float align = 0)`

Append two images along specified axis.

Parameters

<i>img</i>	Image to append with instance image.
<i>axis</i>	Appending axis. Can be { 'x' 'y' 'z' 'c' }.
<i>align</i>	Append alignment in [0,1].

8.1.4.333 `CImg<T>& correlate (const CImg< t > & mask, const unsigned int boundary_conditions = 1, const bool is_normalized = false)`

Correlate image by a mask.

Parameters

<i>mask</i>	= the correlation kernel.
<i>boundary_conditions</i>	= the border condition type (0=zero, 1=dirichlet)
<i>is_normalized</i>	= enable local normalization.

Note

- The correlation of the image instance `*this` by the mask `mask` is defined to be: $\text{res}(x,y,z) = \sum_{\{i,j,k\}} (*this)(x + i, y + j, z + k) * \text{mask}(i,j,k)$.

8.1.4.334 `CImg<T>& convolve (const CImg< t > & mask, const unsigned int boundary_conditions = 1, const bool is_normalized = false)`

Convolve image by a mask.

Parameters

<i>mask</i>	= the correlation kernel.
<i>boundary_conditions</i>	= the border condition type (0=zero, 1=dirichlet)
<i>is_normalized</i>	= enable local normalization.

Note

- The result `res` of the convolution of an image `img` by a mask `mask` is defined to be: $res(x,y,z) = \sum_{\{i,j,k\}} img(x-i,y-j,z-k) * mask(i,j,k)$

8.1.4.335 CImg<T>& cumulate (const char *axis* = 0)

Cumulate image values, optionally along specified axis.

Parameters

<i>axis</i>	Cumulation axis. Set it to 0 to cumulate all values globally without taking axes into account.
-------------	--

8.1.4.336 CImg<T>& cumulate (const char *const *axes*)

Cumulate image values, along specified axes.

Parameters

<i>axes</i>	Cumulation axes, as a C-string.
-------------	---------------------------------

Note

`axes` may contains multiple characters, e.g. "xyz"

8.1.4.337 CImg<T>& erode (const CImg< t > & *mask*, const unsigned int *boundary_conditions* = 1, const bool *is_normalized* = false)

Erode image by a structuring element.

Parameters

<i>mask</i>	Structuring element.
<i>boundary_conditions</i>	Boundary conditions.
<i>is_normalized</i>	Tells if the erosion is locally normalized.

8.1.4.338 CImg<T>& erode (const unsigned int *sx*, const unsigned int *sy*, const unsigned int *sz* = 1)

Erode image by a rectangular structuring element of specified size.

Parameters

<i>sx</i>	Width of the structuring element.
<i>sy</i>	Height of the structuring element.
<i>sz</i>	Depth of the structuring element.

8.1.4.339 CImg<T>& erode (const unsigned int s)

Erode the image by a square structuring element of specified size.

Parameters

<i>s</i>	Size of the structuring element.
----------	----------------------------------

8.1.4.340 CImg<T>& dilate (const CImg< t > & mask, const unsigned int boundary_conditions = 1, const bool is_normalized = false)

Dilate image by a structuring element.

Parameters

<i>mask</i>	Structuring element.
<i>boundary_conditions</i>	Boundary conditions.
<i>is_normalized</i>	Tells if the erosion is locally normalized.

8.1.4.341 CImg<T>& dilate (const unsigned int sx, const unsigned int sy, const unsigned int sz = 1)

Dilate image by a rectangular structuring element of specified size.

Parameters

<i>sx</i>	Width of the structuring element.
<i>sy</i>	Height of the structuring element.
<i>sz</i>	Depth of the structuring element.

8.1.4.342 CImg<T>& dilate (const unsigned int s)

Dilate image by a square structuring element of specified size.

Parameters

<i>s</i>	Size of the structuring element.
----------	----------------------------------

8.1.4.343 CImg<T>& watershed (const CImg< t > & priority, const bool is_high_connectivity = false)

Compute watershed transform.

Parameters

<i>priority</i>	Priority map.
-----------------	---------------

Parameters

<i>is_high_connectivity</i>	Boolean that choose between 4(false)- or 8(true)-connectivity in 2d case, and between 6(false)- or 26(true)-connectivity in 3d case.
-----------------------------	--

Note

Non-zero values of the instance instance are propagated to zero-valued ones according to specified the priority map.

8.1.4.344 `Clmg<T>& deriche (const float sigma, const unsigned int order = 0, const char axis = 'x', const bool boundary_conditions = true)`

Apply recursive Deriche filter.

Parameters

<i>sigma</i>	Standard deviation of the filter.
<i>order</i>	Order of the filter. Can be { 0=smooth-filter 1=1st-derivative 2=2nd-derivative }.
<i>axis</i>	Axis along which the filter is computed. Can be { 'x' 'y' 'z' 'c' }.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann }.

8.1.4.345 `Clmg<T>& vanvliet (const float sigma, const unsigned int order, const char axis = 'x', const bool boundary_conditions = true)`

Van Vliet recursive Gaussian filter.

Parameters

<i>sigma</i>	standard deviation of the Gaussian filter
<i>order</i>	the order of the filter 0,1,2,3
<i>axis</i>	Axis along which the filter is computed. Can be { 'x' 'y' 'z' 'c' }.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann }.

Note

dirichlet boundary condition has a strange behavior

I.T. Young, L.J. van Vliet, M. van Ginkel, Recursive Gabor filtering. IEEE Trans. Sig. Proc., vol. 50, pp. 2799-2805, 2002.

(this is an improvement over Young-Van Vliet, Sig. Proc. 44, 1995)

Boundary conditions (only for order 0) using Triggs matrix, from B. Triggs and M. Sdika. Boundary conditions for Young-van Vliet recursive filtering. IEEE Trans. Signal Processing, vol. 54, pp. 2365-2367, 2006.

8.1.4.346 `CImg<T>& blur (const float sigma_x, const float sigma_y, const float sigma_z, const bool boundary_conditions = true, const bool is_gaussian = false)`

Blur image.

Parameters

<i>sigma_x</i>	Standard deviation of the blur, along the X-axis.
<i>sigma_y</i>	Standard deviation of the blur, along the Y-axis.
<i>sigma_z</i>	Standard deviation of the blur, along the Z-axis.
<i>boundary_conditions</i>	Boundary conditions. Can be { false=dirichlet true=neumann }.
<i>is_gaussian</i>	Tells if the blur uses a gaussian (true) or quasi-gaussian (false) kernel.

Note

- The blur is computed as a 0-order Deriche filter. This is not a gaussian blur.
- This is a recursive algorithm, not depending on the values of the standard deviations.

See also

[deriche\(\)](#), [vanvliet\(\)](#).

8.1.4.347 `CImg<T>& blur (const float sigma, const bool boundary_conditions = true, const bool is_gaussian = false)`

Blur image isotropically.

Parameters

<i>sigma</i>	Standard deviation of the blur.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann }.a

See also

[deriche\(\)](#), [vanvliet\(\)](#).

8.1.4.348 `CImg<T>& blur_anisotropic (const CImg< t > & G, const float amplitude = 60, const float dl = 0.8f, const float da = 30, const float gauss_prec = 2, const unsigned int interpolation_type = 0, const bool is_fast_approx = 1)`

Blur image anisotropically, directed by a field of diffusion tensors.

Parameters

<i>G</i>	Field of square roots of diffusion tensors/vectors used to drive the smoothing.
<i>amplitude</i>	Amplitude of the smoothing.
<i>dl</i>	Spatial discretization.

Parameters

<i>da</i>	Angular discretization.
<i>gauss_prec</i>	Precision of the diffusion process.
<i>interpolation_type</i>	Interpolation scheme. Can be { 0=nearest-neighbor 1=linear 2=Runge-Kutta }.
<i>is_fast_approx</i>	Tells if a fast approximation of the gaussian function is used or not.

8.1.4.349 CImg<T>& blur_anisotropic (const float *amplitude*, const float *sharpness* = 0.7f, const float *anisotropy* = 0.6f, const float *alpha* = 0.6f, const float *sigma* = 1.1f, const float *dl* = 0.8f, const float *da* = 30, const float *gauss_prec* = 2, const unsigned int *interpolation_type* = 0, const bool *is_fast_approx* = true)

Blur image anisotropically, in an edge-preserving way.

Parameters

<i>amplitude</i>	Amplitude of the smoothing.
<i>sharpness</i>	Sharpness.
<i>anisotropy</i>	Anisotropy.
<i>alpha</i>	Standard deviation of the gradient blur.
<i>sigma</i>	Standard deviation of the structure tensor blur.
<i>dl</i>	Spatial discretization.
<i>da</i>	Angular discretization.
<i>gauss_prec</i>	Precision of the diffusion process.
<i>interpolation_type</i>	Interpolation scheme. Can be { 0=nearest-neighbor 1=linear 2=Runge-Kutta }.
<i>is_fast_approx</i>	Tells if a fast approximation of the gaussian function is used or not.

8.1.4.350 CImg<T>& blur_bilateral (const CImg< t > & *guide*, const float *sigma_x*, const float *sigma_y*, const float *sigma_z*, const float *sigma_r*, const float *sampling_x*, const float *sampling_y*, const float *sampling_z*, const float *sampling_r*)

Blur image, with the joint bilateral filter.

Parameters

<i>guide</i>	Image used to model the smoothing weights.
<i>sigma_x</i>	Amount of blur along the X-axis.
<i>sigma_y</i>	Amount of blur along the Y-axis.
<i>sigma_z</i>	Amount of blur along the Z-axis.
<i>sigma_r</i>	Amount of blur along the value axis.
<i>sampling_x</i>	Amount of downsampling along the X-axis used for the approximation. Defaults (0) to <i>sigma_x</i> .
<i>sampling_y</i>	Amount of downsampling along the Y-axis used for the approximation. Defaults (0) to <i>sigma_y</i> .
<i>sampling_z</i>	Amount of downsampling along the Z-axis used for the approximation. Defaults (0) to <i>sigma_z</i> .
<i>sampling_r</i>	Amount of downsampling along the value axis used for the approximation. Defaults (0) to <i>sigma_r</i> .

Note

This algorithm uses the optimisation technique proposed by S. Paris and F. Durand, in ECCV'2006 (extended for 3d volumetric images). It is based on the reference implementation <http://people.csail.mit.edu/jiawen/software/bilateralFilter.m>

8.1.4.351 `Clmg<T>& blur_bilateral (const Clmg< t > & guide, const float sigma_s, const float sigma_r, const float sampling_s = 0, const float sampling_r = 0)`

Blur image using the joint bilateral filter.

Parameters

<i>guide</i>	Image used to model the smoothing weights.
<i>sigma_s</i>	Amount of blur along the XYZ-axes.
<i>sigma_r</i>	Amount of blur along the value axis.
<i>sampling_s</i>	Amount of downsampling along the XYZ-axes used for the approximation. Defaults to sigma_s.
<i>sampling_r</i>	Amount of downsampling along the value axis used for the approximation. Defaults to sigma_r.

8.1.4.352 `Clmg<T>& boxfilter (const float boxsize, const int order, const char axis = 'x', const bool boundary_conditions = true)`

Parameters

<i>boxsize</i>	Size of the box window (can be subpixel)
<i>order</i>	the order of the filter 0,1 or 2.
<i>axis</i>	Axis along which the filter is computed. Can be { 'x' 'y' 'z' 'c' }.
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann }.

8.1.4.353 `Clmg<T>& blur_box (const float boxsize_x, const float boxsize_y, const float boxsize_z, const bool boundary_conditions = true)`

Blur image with a box filter.

Parameters

<i>boxsize_x</i>	Size of the box window, along the X-axis (can be subpixel).
<i>boxsize_y</i>	Size of the box window, along the Y-axis (can be subpixel).
<i>boxsize_z</i>	Size of the box window, along the Z-axis (can be subpixel).
<i>boundary_conditions</i>	Boundary conditions. Can be { false=dirichlet true=neumann }.

Note

- This is a recursive algorithm, not depending on the values of the box kernel size.

See also

[blur\(\)](#).

8.1.4.354 `CImg<T>& blur_box (const float boxsize, const bool boundary_conditions = true)`

Blur image with a box filter.

Parameters

<i>boxsize</i>	Size of the box window (can be subpixel).
<i>boundary_conditions</i>	Boundary conditions. Can be { 0=dirichlet 1=neumann }.a

See also

[deriche\(\)](#), [vanvliet\(\)](#).

8.1.4.355 `CImg<T>& blur_guided (const CImg< t > & guide, const float radius, const float regularization)`

Blur image, with the image guided filter.

Parameters

<i>guide</i>	Image used to guide the smoothing process.
<i>radius</i>	Spatial radius.
<i>regularization</i>	Regularization parameter.

Note

This method implements the filtering algorithm described in: He, Kaiming; Sun, Jian; Tang, Xiaoou, "↵ Guided Image Filtering," Pattern Analysis and Machine Intelligence, IEEE Transactions on , vol.35, no.6, pp.1397,1409, June 2013

8.1.4.356 `CImg<T>& blur_patch (const float sigma_s, const float sigma_p, const unsigned int patch_size = 3, const unsigned int lookup_size = 4, const float smoothness = 0, const bool is_fast_approx = true)`

Blur image using patch-based space.

Parameters

<i>sigma_s</i>	Amount of blur along the XYZ-axes.
<i>sigma_p</i>	Amount of blur along the value axis.
<i>patch_size</i>	Size of the patches.
<i>lookup_size</i>	Size of the window to search similar patches.
<i>smoothness</i>	Smoothness for the patch comparison.
<i>is_fast_approx</i>	Tells if a fast approximation of the gaussian function is used or not.

8.1.4.357 `CImg<T>& blur_median (const unsigned int n, const float threshold = 0)`

Blur image with the median filter.

Parameters

<i>n</i>	Size of the median filter.
<i>threshold</i>	Threshold used to discard pixels too far from the current pixel value in the median computation.

8.1.4.358 `CImg<T>& sharpen (const float amplitude, const bool sharpen_type = false, const float edge = 1, const float alpha = 0, const float sigma = 0)`

Sharpen image.

Parameters

<i>amplitude</i>	Sharpening amplitude
<i>sharpen_type</i>	Select sharpening method. Can be { false=inverse diffusion true=shock filters }.
<i>edge</i>	Edge threshold (shock filters only).
<i>alpha</i>	Gradient smoothness (shock filters only).
<i>sigma</i>	Tensor smoothness (shock filters only).

8.1.4.359 `CImgList<Tfloat> get_gradient (const char *const axes = 0, const int scheme = 3) const`

Return image gradient.

Parameters

<i>axes</i>	Axes considered for the gradient computation, as a C-string (e.g "xy").
<i>scheme</i>	= Numerical scheme used for the gradient computation: <ul style="list-style-type: none"> • -1 = Backward finite differences • 0 = Centered finite differences • 1 = Forward finite differences • 2 = Using Sobel masks • 3 = Using rotation invariant masks • 4 = Using Deriche recursive filter. • 5 = Using Van Vliet recursive filter.

8.1.4.360 `CImgList<Tfloat> get_hessian (const char *const axes = 0) const`

Return image hessian.

Parameters

<i>axes</i>	Axes considered for the hessian computation, as a C-string (e.g "xy").
-------------	--

8.1.4.361 **Clmg<T>& structure_tensors (const bool *is_fwbw_scheme* = false)**

Compute the structure tensor field of an image.

Parameters

<i>is_fwbw_scheme</i>	scheme. Can be { false=centered true=forward-backward }
-----------------------	---

8.1.4.362 **Clmg<T>& diffusion_tensors (const float *sharpness* = 0.7f, const float *anisotropy* = 0.6f, const float *alpha* = 0.6f, const float *sigma* = 1.1f, const bool *is_sqrt* = false)**

Compute field of diffusion tensors for edge-preserving smoothing.

Parameters

<i>sharpness</i>	Sharpness
<i>anisotropy</i>	Anisotropy
<i>alpha</i>	Standard deviation of the gradient blur.
<i>sigma</i>	Standard deviation of the structure tensor blur.
<i>is_sqrt</i>	Tells if the square root of the tensor field is computed instead.

8.1.4.363 **Clmg<T>& displacement (const Clmg< T > & *source*, const float *smoothness* = 0.1f, const float *precision* = 5.0f, const unsigned int *nb_scales* = 0, const unsigned int *iteration_max* = 10000, const bool *is_backward* = false, const Clmg< floatT > & *guide* = Clmg<floatT> : :const_empty())**

Estimate displacement field between two images.

Parameters

<i>source</i>	Reference image.
<i>smoothness</i>	Smoothness of estimated displacement field.
<i>precision</i>	Precision required for algorithm convergence.
<i>nb_scales</i>	Number of scales used to estimate the displacement field.
<i>iteration_max</i>	Maximum number of iterations allowed for one scale.
<i>is_backward</i>	If false, match $I_2(X + U(X)) = I_1(X)$, else match $I_2(X) = I_1(X - U(X))$.
<i>guide</i>	Image used as the initial correspondence estimate for the algorithm. 'guide' may have a last channel with boolean values (0=false other=true) that tells for each pixel if its correspondence vector is constrained to its initial value (constraint mask).

8.1.4.364 `CImg<T>& patchmatch (const CImg< T > & patch_image, const unsigned int patch_width, const unsigned int patch_height, const unsigned int patch_depth, const unsigned int nb_iterations, const unsigned int nb_randoms, const CImg< t1 > & guide, CImg< t2 > & matching_score)`

Compute correspondence map between two images, using the patch-match algorithm.

Parameters

	<i>patch_image</i>	The image containing the reference patches to match with the instance image.
	<i>patch_width</i>	Width of the patch used for matching.
	<i>patch_height</i>	Height of the patch used for matching.
	<i>patch_depth</i>	Depth of the patch used for matching.
	<i>nb_iterations</i>	Number of patch-match iterations.
	<i>nb_randoms</i>	Number of randomization attempts (per pixel).
	<i>guide</i>	Image used as the initial correspondence estimate for the algorithm. 'guide' may have a last channel with boolean values (0=false other=true) that tells for each pixel if its correspondence vector is constrained to its initial value (constraint mask).
out	<i>matching_score</i>	Returned as the image of matching scores.

Note

The patch-match algorithm is described in this paper: Connelly Barnes, Eli Shechtman, Adam Finkelstein, Dan B Goldman(2009), PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing

8.1.4.365 `CImg<T>& distance (const T & value, const unsigned int metric = 2)`

Compute Euclidean distance function to a specified value.

Parameters

<i>value</i>	Reference value.
<i>metric</i>	Type of metric. Can be { 0=Chebyshev 1=Manhattan 2=Euclidean 3=Squared-euclidean }.

Note

The distance transform implementation has been submitted by A. Meijster, and implements the article 'W.H. Hesselink, A. Meijster, J.B.T.M. Roerdink, "A general algorithm for computing distance transforms in linear time.", In: Mathematical Morphology and its Applications to Image and Signal Processing, J. Goutsias, L. Vincent, and D.S. Bloomberg (eds.), Kluwer, 2000, pp. 331-340.' The submitted code has then been modified to fit `CImg` coding style and constraints.

8.1.4.366 `CImg<T>& distance (const T & value, const CImg< t > & metric_mask)`

Compute chamfer distance to a specified value, with a custom metric.

Parameters

<i>value</i>	Reference value.
<i>metric_mask</i>	Metric mask.

Note

The algorithm code has been initially proposed by A. Meijster, and modified by D. Tschumperlé.

8.1.4.367 `Clmg<T>& distance_dijkstra (const T & value, const Clmg< t > & metric, const bool is_high_connectivity, Clmg< to > & return_path)`

Compute distance to a specified value, according to a custom metric (use dijkstra algorithm).

Parameters

<i>value</i>	Reference value.
<i>metric</i>	Field of distance potentials.
<i>is_high_connectivity</i>	Tells if the algorithm uses low or high connectivity.

8.1.4.368 `Clmg<T>& distance_eikonal (const T & value, const Clmg< t > & metric)`

Compute distance map to one source point, according to a custom metric (use fast marching algorithm).

Parameters

<i>value</i>	Reference value.
<i>metric</i>	Field of distance potentials.

8.1.4.369 `Clmg<T>& distance_eikonal (const unsigned int nb_iterations, const float band_size = 0, const float time_step = 0.5f)`

Compute distance function to 0-valued isophotes, using the Eikonal PDE.

Parameters

<i>nb_iterations</i>	Number of PDE iterations.
<i>band_size</i>	Size of the narrow band.
<i>time_step</i>	Time step of the PDE iterations.

8.1.4.370 `Clmg<T>& haar (const char axis, const bool invert = false, const unsigned int nb_scales = 1)`

Compute Haar multiscale wavelet transform.

Parameters

<i>axis</i>	Axis considered for the transform.
<i>invert</i>	Set inverse of direct transform.
<i>nb_scales</i>	Number of scales used for the transform.

8.1.4.371 **CImg<T>& haar** (*const bool invert = false*, *const unsigned int nb_scales = 1*)

Compute Haar multiscale wavelet transform **[overloading]**.

Parameters

<i>invert</i>	Set inverse of direct transform.
<i>nb_scales</i>	Number of scales used for the transform.

8.1.4.372 **CImgList<Tfloat> get_FFT** (*const char axis*, *const bool is_invert = false*) *const*

Compute 1d Fast Fourier Transform, along a specified axis.

Parameters

<i>axis</i>	Axis along which the FFT is computed.
<i>is_invert</i>	Tells if the forward (<i>false</i>) or inverse (<i>true</i>) FFT is computed.

8.1.4.373 **static void FFT** (**CImg< T > & real**, **CImg< T > & imag**, *const char axis*, *const bool is_invert = false*)
[static]

Compute 1d Fast Fourier Transform, along a specified axis.

Parameters

<i>in, out</i>	<i>real</i>	Real part of the pixel values.
<i>in, out</i>	<i>imag</i>	Imaginary part of the pixel values.
	<i>axis</i>	Axis along which the FFT is computed.
	<i>is_invert</i>	Tells if the forward (<i>false</i>) or inverse (<i>true</i>) FFT is computed.

8.1.4.374 **static void FFT** (**CImg< T > & real**, **CImg< T > & imag**, *const bool is_invert = false*, *const unsigned int nb_threads = 0*) [static]

Compute n-d Fast Fourier Transform.

Parameters

<i>in, out</i>	<i>real</i>	Real part of the pixel values.
----------------	-------------	--------------------------------

Parameters

<i>in, out</i>	<i>imag</i>	Imaginary part of the pixel values.
	<i>is_invert</i>	Tells if the forward (<i>false</i>) or inverse (<i>true</i>) FFT is computed.
	<i>nb_threads</i>	Number of parallel threads used for the computation. Use 0 to set this to the number of available cpus.

8.1.4.375 CImg<T>& shift_object3d (const float tx, const float ty = 0, const float tz = 0)

Shift 3d object's vertices.

Parameters

<i>tx</i>	X-coordinate of the 3d displacement vector.
<i>ty</i>	Y-coordinate of the 3d displacement vector.
<i>tz</i>	Z-coordinate of the 3d displacement vector.

8.1.4.376 CImg<T>& shift_object3d ()

Shift 3d object's vertices, so that it becomes centered.

Note

The object center is computed as its barycenter.

8.1.4.377 CImg<T>& resize_object3d (const float sx, const float sy = -100, const float sz = -100)

Resize 3d object.

Parameters

<i>sx</i>	Width of the 3d object's bounding box.
<i>sy</i>	Height of the 3d object's bounding box.
<i>sz</i>	Depth of the 3d object's bounding box.

8.1.4.378 CImg<T>& append_object3d (CImgList< tf > & primitives, const CImg< tp > & obj_vertices, const CImgList< tff > & obj_primitives)

Merge two 3d objects together.

Parameters

<i>in, out</i>	<i>primitives</i>	Primitives data of the current 3d object.
	<i>obj_vertices</i>	Vertices data of the additional 3d object.
	<i>obj_primitives</i>	Primitives data of the additional 3d object.

8.1.4.379 `const CImg<T>& texturize_object3d (CImgList< tp > & primitives, CImgList< tc > & colors, const CImg< tt > & texture, const CImg< tx > & coords = CImg<tx>::const_empty()) const`

Texturize primitives of a 3d object.

Parameters

in, out	<i>primitives</i>	Primitives data of the 3d object.
in, out	<i>colors</i>	Colors data of the 3d object.
	<i>texture</i>	Texture image to map to 3d object.
	<i>coords</i>	Texture-mapping coordinates.

8.1.4.380 `CImg<floatT> get_elevation3d (CImgList< tf > & primitives, CImgList< tc > & colors, const CImg< te > & elevation) const`

Generate a 3d elevation of the image instance.

Parameters

out	<i>primitives</i>	The returned list of the 3d object primitives (template type <i>tf</i> should be at least <i>unsigned int</i>).
out	<i>colors</i>	The returned list of the 3d object colors.
	<i>elevation</i>	The input elevation map.

Returns

The N vertices (xi,yi,zi) of the 3d object as a Nx3 CImg<float> image ($0 \leq i \leq N - 1$).

Example

```
const CImg<float> img("reference.jpg");
CImgList<unsigned int> faces3d;
CImgList<unsigned char> colors3d;
const CImg<float> points3d = img.get_elevation3d(faces3d, colors3d, img.get_norm()*0.2);
CImg<unsigned char>().display_object3d("Elevation3d", points3d, faces3d, colors3d);
```

8.1.4.381 `CImg<floatT> get_projections3d (CImgList< tf > & primitives, CImgList< tc > & colors, const unsigned int x0, const unsigned int y0, const unsigned int z0, const bool normalize_colors = false) const`

Generate the 3d projection planes of the image instance.

Parameters

out	<i>primitives</i>	Primitives data of the returned 3d object.
out	<i>colors</i>	Colors data of the returned 3d object.
	<i>x0</i>	X-coordinate of the projection point.
	<i>y0</i>	Y-coordinate of the projection point.
	<i>z0</i>	Z-coordinate of the projection point.
	<i>normalize_colors</i>	Tells if the created textures have normalized colors.

8.1.4.382 `CImg<floatT> get_isoline3d (CImgList< tf > &primitives, const float isovalue, const int size_x = -100, const int size_y = -100) const`

Generate a isoline of the image instance as a 3d object.

Parameters

out	<i>primitives</i>	The returned list of the 3d object primitives (template type <i>tf</i> should be at least <i>unsigned int</i>).
	<i>isovalue</i>	The returned list of the 3d object colors.
	<i>size_x</i>	The number of subdivisions along the X-axis.
	<i>size_y</i>	The number of subdivisions along the Y-axis.

Returns

The N vertices (xi,yi,zi) of the 3d object as a Nx3 CImg<float> image (0<=i<=N - 1).

Example

```
const CImg<float> img("reference.jpg");
CImgList<unsigned int> faces3d;
const CImg<float> points3d = img.get_isoline3d(faces3d,100);
CImg<unsigned char>().display_object3d("Isoline3d",points3d,faces3d,colors3d);
```

8.1.4.383 `CImg<floatT> get_isosurface3d (CImgList< tf > &primitives, const float isovalue, const int size_x = -100, const int size_y = -100, const int size_z = -100) const`

Generate an isosurface of the image instance as a 3d object.

Parameters

out	<i>primitives</i>	The returned list of the 3d object primitives (template type <i>tf</i> should be at least <i>unsigned int</i>).
	<i>isovalue</i>	The returned list of the 3d object colors.
	<i>size_x</i>	Number of subdivisions along the X-axis.
	<i>size_y</i>	Number of subdivisions along the Y-axis.
	<i>size_z</i>	Number of subdivisions along the Z-axis.

Returns

The N vertices (xi,yi,zi) of the 3d object as a Nx3 CImg<float> image (0<=i<=N - 1).

Example

```
const CImg<float> img = CImg<unsigned char>("reference.jpg").resize(-100,-100,20);
CImgList<unsigned int> faces3d;
const CImg<float> points3d = img.get_isosurface3d(faces3d,100);
CImg<unsigned char>().display_object3d("Isosurface3d",points3d,faces3d,colors3d);
```

8.1.4.384 `static CImg<floatT> elevation3d (CImgList< tf > &primitives, const tfunc &func, const float x0, const float y0, const float x1, const float y1, const int size_x = 256, const int size_y = 256) [static]`

Compute 3d elevation of a function as a 3d object.

Parameters

out	<i>primitives</i>	Primitives data of the resulting 3d object.
	<i>func</i>	Elevation function. Is of type <code>float (*func)(const float x, const float y)</code> .
	<i>x0</i>	X-coordinate of the starting point.
	<i>y0</i>	Y-coordinate of the starting point.
	<i>x1</i>	X-coordinate of the ending point.
	<i>y1</i>	Y-coordinate of the ending point.
	<i>size_x</i>	Resolution of the function along the X-axis.
	<i>size_y</i>	Resolution of the function along the Y-axis.

8.1.4.385 `static CImg<floatT> isoline3d (CImgList<tf> &primitives, const tfunc &func, const float isovalue, const float x0, const float y0, const float x1, const float y1, const int size_x = 256, const int size_y = 256) [static]`

Compute 0-isolines of a function, as a 3d object.

Parameters

out	<i>primitives</i>	Primitives data of the resulting 3d object.
	<i>func</i>	Elevation function. Is of type <code>float (*func)(const float x, const float y)</code> .
	<i>isovalue</i>	Isovalue to extract from function.
	<i>x0</i>	X-coordinate of the starting point.
	<i>y0</i>	Y-coordinate of the starting point.
	<i>x1</i>	X-coordinate of the ending point.
	<i>y1</i>	Y-coordinate of the ending point.
	<i>size_x</i>	Resolution of the function along the X-axis.
	<i>size_y</i>	Resolution of the function along the Y-axis.

Note

Use the marching squares algorithm for extracting the isolines.

8.1.4.386 `static CImg<floatT> isosurface3d (CImgList<tf> &primitives, const tfunc &func, const float isovalue, const float x0, const float y0, const float z0, const float x1, const float y1, const float z1, const int size_x = 32, const int size_y = 32, const int size_z = 32) [static]`

Compute isosurface of a function, as a 3d object.

Parameters

out	<i>primitives</i>	Primitives data of the resulting 3d object.
	<i>func</i>	Implicit function. Is of type <code>float (*func)(const float x, const float y, const float z)</code> .
	<i>isovalue</i>	Isovalue to extract.
	<i>x0</i>	X-coordinate of the starting point.
	<i>y0</i>	Y-coordinate of the starting point.
	<i>z0</i>	Z-coordinate of the starting point.

Parameters

	<i>x1</i>	X-coordinate of the ending point.
	<i>y1</i>	Y-coordinate of the ending point.
	<i>z1</i>	Z-coordinate of the ending point.
	<i>size_x</i>	Resolution of the elevation function along the X-axis.
	<i>size_y</i>	Resolution of the elevation function along the Y-axis.
	<i>size_z</i>	Resolution of the elevation function along the Z-axis.

Note

Use the marching cubes algorithm for extracting the isosurface.

8.1.4.387 `static CImg<floatT> box3d (CImgList< tf > & primitives, const float size_x = 200, const float size_y = 100, const float size_z = 100) [static]`

Generate a 3d box object.

Parameters

out	<i>primitives</i>	The returned list of the 3d object primitives (template type <i>tf</i> should be at least <i>unsigned int</i>).
	<i>size_x</i>	The width of the box (dimension along the X-axis).
	<i>size_y</i>	The height of the box (dimension along the Y-axis).
	<i>size_z</i>	The depth of the box (dimension along the Z-axis).

Returns

The N vertices (xi,yi,zi) of the 3d object as a Nx3 CImg<float> image (0<=i<=N - 1).

Example

```
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::box3d(faces3d,10,20,30);
CImg<unsigned char>().display_object3d("Box3d",points3d,faces3d);
```

8.1.4.388 `static CImg<floatT> cone3d (CImgList< tf > & primitives, const float radius = 50, const float size_z = 100, const unsigned int subdivisions = 24) [static]`

Generate a 3d cone.

Parameters

out	<i>primitives</i>	The returned list of the 3d object primitives (template type <i>tf</i> should be at least <i>unsigned int</i>).
	<i>radius</i>	The radius of the cone basis.
	<i>size_z</i>	The cone's height.
	<i>subdivisions</i>	The number of basis angular subdivisions.

Returns

The N vertices (xi,yi,zi) of the 3d object as a Nx3 CImg<float> image ($0 \leq i \leq N - 1$).

Example

```
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::cone3d(faces3d,50);
CImg<unsigned char>().display_object3d("Cone3d",points3d,faces3d);
```

8.1.4.389 static CImg<floatT> cylinder3d (CImgList< tf > &primitives, const float radius = 50, const float size_z = 100, const unsigned int subdivisions = 24) [static]

Generate a 3d cylinder.

Parameters

out	<i>primitives</i>	The returned list of the 3d object primitives (template type <i>tf</i> should be at least <i>unsigned int</i>).
	<i>radius</i>	The radius of the cylinder basis.
	<i>size_z</i>	The cylinder's height.
	<i>subdivisions</i>	The number of basis angular subdivisions.

Returns

The N vertices (xi,yi,zi) of the 3d object as a Nx3 CImg<float> image ($0 \leq i \leq N - 1$).

Example

```
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::cylinder3d(faces3d,50);
CImg<unsigned char>().display_object3d("Cylinder3d",points3d,faces3d);
```

8.1.4.390 static CImg<floatT> torus3d (CImgList< tf > &primitives, const float radius1 = 100, const float radius2 = 30, const unsigned int subdivisions1 = 24, const unsigned int subdivisions2 = 12) [static]

Generate a 3d torus.

Parameters

out	<i>primitives</i>	The returned list of the 3d object primitives (template type <i>tf</i> should be at least <i>unsigned int</i>).
	<i>radius1</i>	The large radius.
	<i>radius2</i>	The small radius.
	<i>subdivisions1</i>	The number of angular subdivisions for the large radius.
	<i>subdivisions2</i>	The number of angular subdivisions for the small radius.

Returns

The N vertices (xi,yi,zi) of the 3d object as a Nx3 CImg<float> image ($0 \leq i \leq N - 1$).

Example

```
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::torus3d(faces3d,20,4);
CImg<unsigned char>().display_object3d("Torus3d",points3d,faces3d);
```

8.1.4.391 `static CImg<floatT> plane3d (CImgList< tf > & primitives, const float size_x = 100, const float size_y = 100, const unsigned int subdivisions_x = 10, const unsigned int subdivisions_y = 10) [static]`

Generate a 3d XY-plane.

Parameters

out	<i>primitives</i>	The returned list of the 3d object primitives (template type <i>tf</i> should be at least <i>unsigned int</i>).
	<i>size_x</i>	The width of the plane (dimension along the X-axis).
	<i>size_y</i>	The height of the plane (dimensions along the Y-axis).
	<i>subdivisions_x</i>	The number of planar subdivisions along the X-axis.
	<i>subdivisions_y</i>	The number of planar subdivisions along the Y-axis.

Returns

The N vertices (xi,yi,zi) of the 3d object as a Nx3 CImg<float> image (0<=i<=N - 1).

Example

```
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::plane3d(faces3d,100,50);
CImg<unsigned char>().display_object3d("Plane3d",points3d,faces3d);
```

8.1.4.392 `static CImg<floatT> sphere3d (CImgList< tf > & primitives, const float radius = 50, const unsigned int subdivisions = 3) [static]`

Generate a 3d sphere.

Parameters

out	<i>primitives</i>	The returned list of the 3d object primitives (template type <i>tf</i> should be at least <i>unsigned int</i>).
	<i>radius</i>	The radius of the sphere (dimension along the X-axis).
	<i>subdivisions</i>	The number of recursive subdivisions from an initial icosahedron.

Returns

The N vertices (xi,yi,zi) of the 3d object as a Nx3 CImg<float> image (0<=i<=N - 1).

Example

```
CImgList<unsigned int> faces3d;
const CImg<float> points3d = CImg<float>::sphere3d(faces3d,100,4);
CImg<unsigned char>().display_object3d("Sphere3d",points3d,faces3d);
```

8.1.4.393 `static CImg<float> ellipsoid3d (CImgList< tf > &primitives, const CImg< t > &tensor, const unsigned int subdivisions = 3) [static]`

Generate a 3d ellipsoid.

Parameters

out	<i>primitives</i>	The returned list of the 3d object primitives (template type <i>tf</i> should be at least <i>unsigned int</i>).
	<i>tensor</i>	The tensor which gives the shape and size of the ellipsoid.
	<i>subdivisions</i>	The number of recursive subdivisions from an initial stretched icosahedron.

Returns

The N vertices (xi,yi,zi) of the 3d object as a Nx3 CImg<float> image (0<=i<=N - 1).

Example

```
CImgList<unsigned int> faces3d;
const CImg<float> tensor = CImg<float>::diagonal(10,7,3),
                      points3d = CImg<float>::ellipsoid3d(faces3d,tensor,4);
CImg<unsigned char>().display_object3d("Ellipsoid3d",points3d,faces3d);
```

8.1.4.394 `CImg<T>& object3dtoCImg3d (const CImgList< tp > &primitives, const CImgList< tc > &colors, const to & opacities, const bool full_check = true)`

Convert 3d object into a CImg3d representation.

Parameters

<i>primitives</i>	Primitives data of the 3d object.
<i>colors</i>	Colors data of the 3d object.
<i>opacities</i>	Opacities data of the 3d object.
<i>full_check</i>	Tells if full checking of the 3d object must be performed.

8.1.4.395 `CImg<T>& CImg3dtoobject3d (CImgList< tp > &primitives, CImgList< tc > &colors, CImgList< to > & opacities, const bool full_check = true)`

Convert CImg3d representation into a 3d object.

Parameters

out	<i>primitives</i>	Primitives data of the 3d object.
out	<i>colors</i>	Colors data of the 3d object.
out	<i>opacities</i>	Opacities data of the 3d object.
	<i>full_check</i>	Tells if full checking of the 3d object must be performed.

8.1.4.396 CImg<T>& draw_point (const int x0, const int y0, const int z0, const tc *const color, const float opacity = 1)

Draw a 3d point.

Parameters

<i>x0</i>	X-coordinate of the point.
<i>y0</i>	Y-coordinate of the point.
<i>z0</i>	Z-coordinate of the point.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.

Note

- To set pixel values without clipping needs, you should use the faster [CImg::operator\(\)\(\)](#) function.

Example:

```
CImg<unsigned char> img(100,100,1,3,0);
const unsigned char color[] = { 255,128,64 };
img.draw_point(50,50,color);
```

8.1.4.397 CImg<T>& draw_point (const CImg< t > & points, const tc *const color, const float opacity = 1)

Parameters

<i>points</i>	Image of vertices coordinates.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.

8.1.4.398 CImg<T>& draw_line (const int x0, const int y0, const int x1, const int y1, const tc *const color, const float opacity = 1, const unsigned int pattern = ~0U, const bool init_hatch = true)

Draw a 2d line.

Parameters

<i>x0</i>	X-coordinate of the starting line point.
<i>y0</i>	Y-coordinate of the starting line point.
<i>x1</i>	X-coordinate of the ending line point.
<i>y1</i>	Y-coordinate of the ending line point.
<i>color</i>	Pointer to spectrum() consecutive values of type T, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the line pattern.
<i>init_hatch</i>	Tells if a reinitialization of the hash state must be done.

Note

- Line routine uses Bresenham's algorithm.
- Set `init_hatch = false` to draw consecutive hatched segments without breaking the line pattern.

Example:

```
CImg<unsigned char> img(100,100,1,3,0);
const unsigned char color[] = { 255,128,64 };
img.draw_line(40,40,80,70,color);
```

8.1.4.399 `CImg<T>& draw_line (CImg< T > & zbuffer, const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const tc *const color, const float opacity = 1, const unsigned int pattern = ~0U, const bool init_hatch = true)`

Draw a 2d line, with z-buffering.

Parameters

<i>zbuffer</i>	Zbuffer image.
<i>x0</i>	X-coordinate of the starting point.
<i>y0</i>	Y-coordinate of the starting point.
<i>z0</i>	Z-coordinate of the starting point
<i>x1</i>	X-coordinate of the ending point.
<i>y1</i>	Y-coordinate of the ending point.
<i>z1</i>	Z-coordinate of the ending point.
<i>color</i>	Pointer to spectrum() consecutive values of type <code>T</code> , defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the line pattern.
<i>init_hatch</i>	Tells if a reinitialization of the hash state must be done.

8.1.4.400 `CImg<T>& draw_line (const int x0, const int y0, const int z0, const int x1, const int y1, const int z1, const tc *const color, const float opacity = 1, const unsigned int pattern = ~0U, const bool init_hatch = true)`

Draw a 3d line.

Parameters

<i>x0</i>	X-coordinate of the starting point.
<i>y0</i>	Y-coordinate of the starting point.
<i>z0</i>	Z-coordinate of the starting point
<i>x1</i>	X-coordinate of the ending point.
<i>y1</i>	Y-coordinate of the ending point.
<i>z1</i>	Z-coordinate of the ending point.
<i>color</i>	Pointer to spectrum() consecutive values of type <code>T</code> , defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the line pattern.
<i>init_hatch</i>	Tells if a reinitialization of the hash state must be done.

8.1.4.401 `CImg<T>& draw_line (const int x0, const int y0, const int x1, const int y1, const CImg< tc > & texture, const int tx0, const int ty0, const int tx1, const int ty1, const float opacity = 1, const unsigned int pattern = ~0U, const bool init_hatch = true)`

Draw a textured 2d line.

Parameters

<i>x0</i>	X-coordinate of the starting line point.
<i>y0</i>	Y-coordinate of the starting line point.
<i>x1</i>	X-coordinate of the ending line point.
<i>y1</i>	Y-coordinate of the ending line point.
<i>texture</i>	Texture image defining the pixel colors.
<i>tx0</i>	X-coordinate of the starting texture point.
<i>ty0</i>	Y-coordinate of the starting texture point.
<i>tx1</i>	X-coordinate of the ending texture point.
<i>ty1</i>	Y-coordinate of the ending texture point.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the line pattern.
<i>init_hatch</i>	Tells if the hash variable must be reinitialized.

Note

- Line routine uses the well known Bresenham's algorithm.

Example:

```
CImg<unsigned char> img(100,100,1,3,0), texture("texture256x256.ppm");
const unsigned char color[] = { 255,128,64 };
img.draw_line(40,40,80,70,texture,0,0,255,255);
```

8.1.4.402 `CImg<T>& draw_line (const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const CImg< tc > & texture, const int tx0, const int ty0, const int tx1, const int ty1, const float opacity = 1, const unsigned int pattern = ~0U, const bool init_hatch = true)`

Draw a textured 2d line, with perspective correction.

Parameters

<i>x0</i>	X-coordinate of the starting point.
<i>y0</i>	Y-coordinate of the starting point.
<i>z0</i>	Z-coordinate of the starting point
<i>x1</i>	X-coordinate of the ending point.
<i>y1</i>	Y-coordinate of the ending point.
<i>z1</i>	Z-coordinate of the ending point.
<i>texture</i>	Texture image defining the pixel colors.
<i>tx0</i>	X-coordinate of the starting texture point.
<i>ty0</i>	Y-coordinate of the starting texture point.
<i>tx1</i>	X-coordinate of the ending texture point.
<i>ty1</i>	Y-coordinate of the ending texture point.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the line pattern.
<i>init_hatch</i>	Tells if the hash variable must be reinitialized.

8.1.4.403 `Clmg<T>& draw_line (Clmg< tz > & zbuffer, const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const Clmg< tc > & texture, const int tx0, const int ty0, const int tx1, const int ty1, const float opacity = 1, const unsigned int pattern = ~0U, const bool init_hatch = true)`

Draw a textured 2d line, with perspective correction and z-buffering.

Parameters

<i>zbuffer</i>	Z-buffer image.
<i>x0</i>	X-coordinate of the starting point.
<i>y0</i>	Y-coordinate of the starting point.
<i>z0</i>	Z-coordinate of the starting point
<i>x1</i>	X-coordinate of the ending point.
<i>y1</i>	Y-coordinate of the ending point.
<i>z1</i>	Z-coordinate of the ending point.
<i>texture</i>	Texture image defining the pixel colors.
<i>tx0</i>	X-coordinate of the starting texture point.
<i>ty0</i>	Y-coordinate of the starting texture point.
<i>tx1</i>	X-coordinate of the ending texture point.
<i>ty1</i>	Y-coordinate of the ending texture point.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the line pattern.
<i>init_hatch</i>	Tells if the hash variable must be reinitialized.

8.1.4.404 `Clmg<T>& draw_line (const Clmg< t > & points, const tc *const color, const float opacity = 1, const unsigned int pattern = ~0U, const bool init_hatch = true)`

Draw a set of consecutive lines.

Parameters

<i>points</i>	Coordinates of vertices, stored as a list of vectors.
<i>color</i>	Pointer to spectrum() consecutive values of type <code>T</code> , defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the line pattern.
<i>init_hatch</i>	If set to true, init hatch motif.

Note

- This function uses several call to the single [Clmg::draw_line\(\)](#) procedure, depending on the vectors size in `points`.

8.1.4.405 `Clmg<T>& draw_arrow (const int x0, const int y0, const int x1, const int y1, const tc *const color, const float opacity = 1, const float angle = 30, const float length = -10, const unsigned int pattern = ~0U)`

Draw a 2d arrow.

Parameters

<i>x0</i>	X-coordinate of the starting arrow point (tail).
<i>y0</i>	Y-coordinate of the starting arrow point (tail).
<i>x1</i>	X-coordinate of the ending arrow point (head).
<i>y1</i>	Y-coordinate of the ending arrow point (head).
<i>color</i>	Pointer to <code>spectrum()</code> consecutive values of type <code>T</code> , defining the drawing color.
<i>angle</i>	Aperture angle of the arrow head.
<i>length</i>	Length of the arrow head. If negative, describes a percentage of the arrow length.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the line pattern.

8.1.4.406 `CImg<T>& draw_spline (const int x0, const int y0, const float u0, const float v0, const int x1, const int y1, const float u1, const float v1, const tc *const color, const float opacity = 1, const float precision = 0.25, const unsigned int pattern = ~0U, const bool init_hatch = true)`

Draw a 2d spline.

Parameters

<i>x0</i>	X-coordinate of the starting curve point
<i>y0</i>	Y-coordinate of the starting curve point
<i>u0</i>	X-coordinate of the starting velocity
<i>v0</i>	Y-coordinate of the starting velocity
<i>x1</i>	X-coordinate of the ending curve point
<i>y1</i>	Y-coordinate of the ending curve point
<i>u1</i>	X-coordinate of the ending velocity
<i>v1</i>	Y-coordinate of the ending velocity
<i>color</i>	Pointer to <code>spectrum()</code> consecutive values of type <code>T</code> , defining the drawing color.
<i>precision</i>	Curve drawing precision.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the line pattern.
<i>init_hatch</i>	If <code>true</code> , init hatch motif.

Note

- The curve is a 2d cubic Bezier spline, from the set of specified starting/ending points and corresponding velocity vectors.
- The spline is drawn as a serie of connected segments. The `precision` parameter sets the average number of pixels in each drawn segment.
- A cubic Bezier curve is sometimes defined by a set of 4 points $\{ (x_0, y_0), (x_a, y_a), (x_b, y_b), (x_1, y_1) \}$ where (x_0, y_0) is the starting point, (x_1, y_1) is the ending point and $(x_a, y_a), (x_b, y_b)$ are two *control* points. The starting and ending velocities (u_0, v_0) and (u_1, v_1) can be deduced easily from the control points as $u_0 = (x_a - x_0)$, $v_0 = (y_a - y_0)$, $u_1 = (x_1 - x_b)$ and $v_1 = (y_1 - y_b)$.

Example:

```
CImg<unsigned char> img(100,100,1,3,0);
const unsigned char color[] = { 255,255,255 };
img.draw_spline(30,30,0,100,90,40,0,-100,color);
```

8.1.4.407 `CImg<T>& draw_spline (const int x0, const int y0, const int z0, const float u0, const float v0, const float w0, const int x1, const int y1, const int z1, const float u1, const float v1, const float w1, const tc *const color, const float opacity = 1, const float precision = 4, const unsigned int pattern = ~0U, const bool init_hatch = true)`

Draw a 3d spline [**overloading**].

Note

- Similar to [CImg::draw_spline\(\)](#) for a 3d spline in a volumetric image.

8.1.4.408 `CImg<T>& draw_spline (const int x0, const int y0, const float u0, const float v0, const int x1, const int y1, const float u1, const float v1, const CImg< t > & texture, const int tx0, const int ty0, const int tx1, const int ty1, const float opacity = 1, const float precision = 4, const unsigned int pattern = ~0U, const bool init_hatch = true)`

Draw a textured 2d spline.

Parameters

<i>x0</i>	X-coordinate of the starting curve point
<i>y0</i>	Y-coordinate of the starting curve point
<i>u0</i>	X-coordinate of the starting velocity
<i>v0</i>	Y-coordinate of the starting velocity
<i>x1</i>	X-coordinate of the ending curve point
<i>y1</i>	Y-coordinate of the ending curve point
<i>u1</i>	X-coordinate of the ending velocity
<i>v1</i>	Y-coordinate of the ending velocity
<i>texture</i>	Texture image defining line pixel colors.
<i>tx0</i>	X-coordinate of the starting texture point.
<i>ty0</i>	Y-coordinate of the starting texture point.
<i>tx1</i>	X-coordinate of the ending texture point.
<i>ty1</i>	Y-coordinate of the ending texture point.
<i>precision</i>	Curve drawing precision.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the line pattern.
<i>init_hatch</i>	if <code>true</code> , reinit hatch motif.

8.1.4.409 `CImg<T>& draw_spline (const CImg< tp > & points, const CImg< tt > & tangents, const tc *const color, const float opacity = 1, const bool is_closed_set = false, const float precision = 4, const unsigned int pattern = ~0U, const bool init_hatch = true)`

Draw a set of consecutive splines.

Parameters

<i>points</i>	Vertices data.
<i>tangents</i>	Tangents data.
<i>color</i>	Pointer to spectrum() consecutive values of type <code>T</code> , defining the drawing color.
<i>opacity</i>	Drawing opacity.

Parameters

<i>is_closed_set</i>	Tells if the drawn spline set is closed.
<i>precision</i>	Precision of the drawing.
<i>pattern</i>	An integer whose bits describe the line pattern.
<i>init_hatch</i>	If <code>true</code> , init hatch motif.

8.1.4.410 **Clmg<T>& draw_spline** (`const Clmg<tp> &points`, `const tc *const color`, `const float opacity = 1`, `const bool is_closed_set = false`, `const float precision = 4`, `const unsigned int pattern = ~0U`, `const bool init_hatch = true`)

Draw a set of consecutive splines [**overloading**].

Similar to previous function, with the point tangents automatically estimated from the given points set.

8.1.4.411 **Clmg<T>& draw_triangle** (`const int x0`, `const int y0`, `const int x1`, `const int y1`, `const int x2`, `const int y2`, `const tc *const color`, `const float opacity = 1`)

Draw a filled 2d triangle.

Parameters

<i>x0</i>	X-coordinate of the first vertex.
<i>y0</i>	Y-coordinate of the first vertex.
<i>x1</i>	X-coordinate of the second vertex.
<i>y1</i>	Y-coordinate of the second vertex.
<i>x2</i>	X-coordinate of the third vertex.
<i>y2</i>	Y-coordinate of the third vertex.
<i>color</i>	Pointer to spectrum() consecutive values of type <code>T</code> , defining the drawing color.
<i>opacity</i>	Drawing opacity.

8.1.4.412 **Clmg<T>& draw_triangle** (`const int x0`, `const int y0`, `const int x1`, `const int y1`, `const int x2`, `const int y2`, `const tc *const color`, `const float opacity`, `const unsigned int pattern`)

Draw a outlined 2d triangle.

Parameters

<i>x0</i>	X-coordinate of the first vertex.
<i>y0</i>	Y-coordinate of the first vertex.
<i>x1</i>	X-coordinate of the second vertex.
<i>y1</i>	Y-coordinate of the second vertex.
<i>x2</i>	X-coordinate of the third vertex.
<i>y2</i>	Y-coordinate of the third vertex.
<i>color</i>	Pointer to spectrum() consecutive values of type <code>T</code> , defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the outline pattern.

8.1.4.413 `CImg<T>& draw_triangle (CImg< tz > & zbuffer, const int x0, const int y0, const float z0, const int x1, const int y1, const float z1, const int x2, const int y2, const float z2, const tc *const color, const float opacity = 1, const float brightness = 1)`

Draw a filled 2d triangle, with z-buffering.

Parameters

<i>zbuffer</i>	Z-buffer image.
<i>x0</i>	X-coordinate of the first vertex.
<i>y0</i>	Y-coordinate of the first vertex.
<i>z0</i>	Z-coordinate of the first vertex.
<i>x1</i>	X-coordinate of the second vertex.
<i>y1</i>	Y-coordinate of the second vertex.
<i>z1</i>	Z-coordinate of the second vertex.
<i>x2</i>	X-coordinate of the third vertex.
<i>y2</i>	Y-coordinate of the third vertex.
<i>z2</i>	Z-coordinate of the third vertex.
<i>color</i>	Pointer to <code>spectrum()</code> consecutive values of type <code>T</code> , defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>brightness</i>	Brightness factor.

8.1.4.414 `CImg<T>& draw_triangle (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const tc *const color, const float brightness0, const float brightness1, const float brightness2, const float opacity = 1)`

Draw a Gouraud-shaded 2d triangle.

Parameters

<i>x0</i>	X-coordinate of the first vertex in the image instance.
<i>y0</i>	Y-coordinate of the first vertex in the image instance.
<i>x1</i>	X-coordinate of the second vertex in the image instance.
<i>y1</i>	Y-coordinate of the second vertex in the image instance.
<i>x2</i>	X-coordinate of the third vertex in the image instance.
<i>y2</i>	Y-coordinate of the third vertex in the image instance.
<i>color</i>	Pointer to <code>spectrum()</code> consecutive values, defining the drawing color.
<i>brightness0</i>	Brightness factor of the first vertex (in [0,2]).
<i>brightness1</i>	brightness factor of the second vertex (in [0,2]).
<i>brightness2</i>	brightness factor of the third vertex (in [0,2]).
<i>opacity</i>	Drawing opacity.

8.1.4.415 `CImg<T>& draw_triangle (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const tc1 *const color1, const tc2 *const color2, const tc3 *const color3, const float opacity = 1)`

Draw a color-interpolated 2d triangle.

Parameters

<i>x0</i>	X-coordinate of the first vertex in the image instance.
<i>y0</i>	Y-coordinate of the first vertex in the image instance.
<i>x1</i>	X-coordinate of the second vertex in the image instance.
<i>y1</i>	Y-coordinate of the second vertex in the image instance.
<i>x2</i>	X-coordinate of the third vertex in the image instance.
<i>y2</i>	Y-coordinate of the third vertex in the image instance.
<i>color1</i>	Pointer to spectrum() consecutive values of type <i>T</i> , defining the color of the first vertex.
<i>color2</i>	Pointer to spectrum() consecutive values of type <i>T</i> , defining the color of the second vertex.
<i>color3</i>	Pointer to spectrum() consecutive values of type <i>T</i> , defining the color of the third vertex.
<i>opacity</i>	Drawing opacity.

8.1.4.416 `CImg<T>& draw_triangle (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const CImg<tc> & texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const float opacity = 1, const float brightness = 1)`

Draw a textured 2d triangle.

Parameters

<i>x0</i>	X-coordinate of the first vertex in the image instance.
<i>y0</i>	Y-coordinate of the first vertex in the image instance.
<i>x1</i>	X-coordinate of the second vertex in the image instance.
<i>y1</i>	Y-coordinate of the second vertex in the image instance.
<i>x2</i>	X-coordinate of the third vertex in the image instance.
<i>y2</i>	Y-coordinate of the third vertex in the image instance.
<i>texture</i>	Texture image used to fill the triangle.
<i>tx0</i>	X-coordinate of the first vertex in the texture image.
<i>ty0</i>	Y-coordinate of the first vertex in the texture image.
<i>tx1</i>	X-coordinate of the second vertex in the texture image.
<i>ty1</i>	Y-coordinate of the second vertex in the texture image.
<i>tx2</i>	X-coordinate of the third vertex in the texture image.
<i>ty2</i>	Y-coordinate of the third vertex in the texture image.
<i>opacity</i>	Drawing opacity.
<i>brightness</i>	Brightness factor of the drawing (in [0,2]).

8.1.4.417 `CImg<T>& draw_triangle (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const tc *const color, const CImg<tl> & light, const int lx0, const int ly0, const int lx1, const int ly1, const int lx2, const int ly2, const float opacity = 1)`

Draw a Phong-shaded 2d triangle.

Parameters

<i>x0</i>	X-coordinate of the first vertex in the image instance.
<i>y0</i>	Y-coordinate of the first vertex in the image instance.

Parameters

<i>x1</i>	X-coordinate of the second vertex in the image instance.
<i>y1</i>	Y-coordinate of the second vertex in the image instance.
<i>x2</i>	X-coordinate of the third vertex in the image instance.
<i>y2</i>	Y-coordinate of the third vertex in the image instance.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>light</i>	Light image.
<i>lx0</i>	X-coordinate of the first vertex in the light image.
<i>ly0</i>	Y-coordinate of the first vertex in the light image.
<i>lx1</i>	X-coordinate of the second vertex in the light image.
<i>ly1</i>	Y-coordinate of the second vertex in the light image.
<i>lx2</i>	X-coordinate of the third vertex in the light image.
<i>ly2</i>	Y-coordinate of the third vertex in the light image.
<i>opacity</i>	Drawing opacity.

8.1.4.418 `CImg<T>& draw_triangle (const int x0, const int y0, const int x1, const int y1, const int x2, const int y2, const CImg< tc > & texture, const int tx0, const int ty0, const int tx1, const int ty1, const int tx2, const int ty2, const float brightness0, const float brightness1, const float brightness2, const float opacity = 1)`

Draw a textured Gouraud-shaded 2d triangle.

Parameters

<i>x0</i>	X-coordinate of the first vertex in the image instance.
<i>y0</i>	Y-coordinate of the first vertex in the image instance.
<i>x1</i>	X-coordinate of the second vertex in the image instance.
<i>y1</i>	Y-coordinate of the second vertex in the image instance.
<i>x2</i>	X-coordinate of the third vertex in the image instance.
<i>y2</i>	Y-coordinate of the third vertex in the image instance.
<i>texture</i>	Texture image used to fill the triangle.
<i>tx0</i>	X-coordinate of the first vertex in the texture image.
<i>ty0</i>	Y-coordinate of the first vertex in the texture image.
<i>tx1</i>	X-coordinate of the second vertex in the texture image.
<i>ty1</i>	Y-coordinate of the second vertex in the texture image.
<i>tx2</i>	X-coordinate of the third vertex in the texture image.
<i>ty2</i>	Y-coordinate of the third vertex in the texture image.
<i>brightness0</i>	Brightness factor of the first vertex.
<i>brightness1</i>	Brightness factor of the second vertex.
<i>brightness2</i>	Brightness factor of the third vertex.
<i>opacity</i>	Drawing opacity.

8.1.4.419 **CImg<T>& draw_triangle** (const int *x0*, const int *y0*, const int *x1*, const int *y1*, const int *x2*, const int *y2*, const CImg< tc > & *texture*, const int *tx0*, const int *ty0*, const int *tx1*, const int *ty1*, const int *tx2*, const int *ty2*, const CImg< tl > & *light*, const int *lx0*, const int *ly0*, const int *lx1*, const int *ly1*, const int *lx2*, const int *ly2*, const float *opacity* = 1)

Draw a textured Phong-shaded 2d triangle.

Parameters

<i>x0</i>	X-coordinate of the first vertex in the image instance.
<i>y0</i>	Y-coordinate of the first vertex in the image instance.
<i>x1</i>	X-coordinate of the second vertex in the image instance.
<i>y1</i>	Y-coordinate of the second vertex in the image instance.
<i>x2</i>	X-coordinate of the third vertex in the image instance.
<i>y2</i>	Y-coordinate of the third vertex in the image instance.
<i>texture</i>	Texture image used to fill the triangle.
<i>tx0</i>	X-coordinate of the first vertex in the texture image.
<i>ty0</i>	Y-coordinate of the first vertex in the texture image.
<i>tx1</i>	X-coordinate of the second vertex in the texture image.
<i>ty1</i>	Y-coordinate of the second vertex in the texture image.
<i>tx2</i>	X-coordinate of the third vertex in the texture image.
<i>ty2</i>	Y-coordinate of the third vertex in the texture image.
<i>light</i>	Light image.
<i>lx0</i>	X-coordinate of the first vertex in the light image.
<i>ly0</i>	Y-coordinate of the first vertex in the light image.
<i>lx1</i>	X-coordinate of the second vertex in the light image.
<i>ly1</i>	Y-coordinate of the second vertex in the light image.
<i>lx2</i>	X-coordinate of the third vertex in the light image.
<i>ly2</i>	Y-coordinate of the third vertex in the light image.
<i>opacity</i>	Drawing opacity.

8.1.4.420 **CImg<T>& draw_rectangle** (const int *x0*, const int *y0*, const int *z0*, const int *c0*, const int *x1*, const int *y1*, const int *z1*, const int *c1*, const T *val*, const float *opacity* = 1)

Draw a filled 4d rectangle.

Parameters

<i>x0</i>	X-coordinate of the upper-left rectangle corner.
<i>y0</i>	Y-coordinate of the upper-left rectangle corner.
<i>z0</i>	Z-coordinate of the upper-left rectangle corner.
<i>c0</i>	C-coordinate of the upper-left rectangle corner.
<i>x1</i>	X-coordinate of the lower-right rectangle corner.
<i>y1</i>	Y-coordinate of the lower-right rectangle corner.
<i>z1</i>	Z-coordinate of the lower-right rectangle corner.
<i>c1</i>	C-coordinate of the lower-right rectangle corner.
<i>val</i>	Scalar value used to fill the rectangle area.
<i>opacity</i>	Drawing opacity.

8.1.4.421 `Clmg<T>& draw_rectangle (const int x0, const int y0, const int z0, const int x1, const int y1, const int z1, const tc *const color, const float opacity = 1)`

Draw a filled 3d rectangle.

Parameters

<i>x0</i>	X-coordinate of the upper-left rectangle corner.
<i>y0</i>	Y-coordinate of the upper-left rectangle corner.
<i>z0</i>	Z-coordinate of the upper-left rectangle corner.
<i>x1</i>	X-coordinate of the lower-right rectangle corner.
<i>y1</i>	Y-coordinate of the lower-right rectangle corner.
<i>z1</i>	Z-coordinate of the lower-right rectangle corner.
<i>color</i>	Pointer to spectrum() consecutive values of type <code>T</code> , defining the drawing color.
<i>opacity</i>	Drawing opacity.

8.1.4.422 `Clmg<T>& draw_rectangle (const int x0, const int y0, const int x1, const int y1, const tc *const color, const float opacity = 1)`

Draw a filled 2d rectangle.

Parameters

<i>x0</i>	X-coordinate of the upper-left rectangle corner.
<i>y0</i>	Y-coordinate of the upper-left rectangle corner.
<i>x1</i>	X-coordinate of the lower-right rectangle corner.
<i>y1</i>	Y-coordinate of the lower-right rectangle corner.
<i>color</i>	Pointer to spectrum() consecutive values of type <code>T</code> , defining the drawing color.
<i>opacity</i>	Drawing opacity.

8.1.4.423 `Clmg<T>& draw_polygon (const Clmg< t > & points, const tc *const color, const float opacity = 1)`

Draw a filled 2d polygon.

Parameters

<i>points</i>	Set of polygon vertices.
<i>color</i>	Pointer to spectrum() consecutive values of type <code>T</code> , defining the drawing color.
<i>opacity</i>	Drawing opacity.

8.1.4.424 `Clmg<T>& draw_ellipse (const int x0, const int y0, const float r1, const float r2, const float angle, const tc *const color, const float opacity = 1)`

Draw a filled 2d ellipse.

Parameters

<i>x0</i>	X-coordinate of the ellipse center.
<i>y0</i>	Y-coordinate of the ellipse center.
<i>r1</i>	First radius of the ellipse.
<i>r2</i>	Second radius of the ellipse.
<i>angle</i>	Angle of the first radius.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.

8.1.4.425 `CImg<T>& draw_ellipse (const int x0, const int y0, const CImg< t > & tensor, const tc *const color, const float opacity = 1)`

Draw a filled 2d ellipse **[overloading]**.

Parameters

<i>x0</i>	X-coordinate of the ellipse center.
<i>y0</i>	Y-coordinate of the ellipse center.
<i>tensor</i>	Diffusion tensor describing the ellipse.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.

8.1.4.426 `CImg<T>& draw_ellipse (const int x0, const int y0, const float r1, const float r2, const float angle, const tc *const color, const float opacity, const unsigned int pattern)`

Draw an outlined 2d ellipse.

Parameters

<i>x0</i>	X-coordinate of the ellipse center.
<i>y0</i>	Y-coordinate of the ellipse center.
<i>r1</i>	First radius of the ellipse.
<i>r2</i>	Second radius of the ellipse.
<i>angle</i>	Angle of the first radius.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the outline pattern.

8.1.4.427 `CImg<T>& draw_ellipse (const int x0, const int y0, const CImg< t > & tensor, const tc *const color, const float opacity, const unsigned int pattern)`

Draw an outlined 2d ellipse **[overloading]**.

Parameters

<i>x0</i>	X-coordinate of the ellipse center.
<i>y0</i>	Y-coordinate of the ellipse center.
<i>tensor</i>	Diffusion tensor describing the ellipse.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the outline pattern.

8.1.4.428 `Clmg<T>& draw_circle (const int x0, const int y0, int radius, const tc *const color, const float opacity = 1)`

Draw a filled 2d circle.

Parameters

<i>x0</i>	X-coordinate of the circle center.
<i>y0</i>	Y-coordinate of the circle center.
<i>radius</i>	Circle radius.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.

Note

- Circle version of the Bresenham's algorithm is used.

8.1.4.429 `Clmg<T>& draw_circle (const int x0, const int y0, int radius, const tc *const color, const float opacity, const unsigned int pattern)`

Draw an outlined 2d circle.

Parameters

<i>x0</i>	X-coordinate of the circle center.
<i>y0</i>	Y-coordinate of the circle center.
<i>radius</i>	Circle radius.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	An integer whose bits describe the outline pattern.

8.1.4.430 `Clmg<T>& draw_image (const int x0, const int y0, const int z0, const int c0, const Clmg< t > & sprite, const float opacity = 1)`

Draw an image.

Parameters

<i>sprite</i>	Sprite image.
<i>x0</i>	X-coordinate of the sprite position.
<i>y0</i>	Y-coordinate of the sprite position.
<i>z0</i>	Z-coordinate of the sprite position.
<i>c0</i>	C-coordinate of the sprite position.
<i>opacity</i>	Drawing opacity.

8.1.4.431 **CImg<T>& draw_image (const int *x0*, const int *y0*, const int *z0*, const int *c0*, const CImg<ti> & *sprite*, const CImg<tm> & *mask*, const float *opacity* = 1, const float *mask_max_value* = 1)**

Draw a masked image.

Parameters

<i>sprite</i>	Sprite image.
<i>mask</i>	Mask image.
<i>x0</i>	X-coordinate of the sprite position in the image instance.
<i>y0</i>	Y-coordinate of the sprite position in the image instance.
<i>z0</i>	Z-coordinate of the sprite position in the image instance.
<i>c0</i>	C-coordinate of the sprite position in the image instance.
<i>mask_max_value</i>	Maximum pixel value of the mask image <i>mask</i> .
<i>opacity</i>	Drawing opacity.

Note

- Pixel values of *mask* set the opacity of the corresponding pixels in *sprite*.
- Dimensions along x,y and z of *sprite* and *mask* must be the same.

8.1.4.432 **CImg<T>& draw_text (const int *x0*, const int *y0*, const char *const *text*, const tc1 *const *foreground_color*, const tc2 *const *background_color*, const float *opacity*, const CImgList<t> & *font*, ...)**

Draw a text string.

Parameters

<i>x0</i>	X-coordinate of the text in the image instance.
<i>y0</i>	Y-coordinate of the text in the image instance.
<i>text</i>	Format of the text ('printf'-style format string).
<i>foreground_color</i>	Pointer to spectrum() consecutive values, defining the foreground drawing color.
<i>background_color</i>	Pointer to spectrum() consecutive values, defining the background drawing color.
<i>opacity</i>	Drawing opacity.
<i>font</i>	Font used for drawing text.

8.1.4.433 `CImg<T>& draw_text (const int x0, const int y0, const char *const text, const tc *const foreground_color, const int , const float opacity, const CImgList< t > & font, ...)`

Draw a text string **[overloading]**.

Note

A transparent background is used for the text.

8.1.4.434 `CImg<T>& draw_text (const int x0, const int y0, const char *const text, const int , const tc *const background_color, const float opacity, const CImgList< t > & font, ...)`

Draw a text string **[overloading]**.

Note

A transparent foreground is used for the text.

8.1.4.435 `CImg<T>& draw_text (const int x0, const int y0, const char *const text, const tc1 *const foreground_color, const tc2 *const background_color, const float opacity = 1, const unsigned int font_height = 13, ...)`

Draw a text string **[overloading]**.

Parameters

<i>x0</i>	X-coordinate of the text in the image instance.
<i>y0</i>	Y-coordinate of the text in the image instance.
<i>text</i>	Format of the text ('printf'-style format string).
<i>foreground_color</i>	Array of spectrum() values of type T, defining the foreground color (0 means 'transparent').
<i>background_color</i>	Array of spectrum() values of type T, defining the background color (0 means 'transparent').
<i>opacity</i>	Drawing opacity.
<i>font_height</i>	Height of the text font (exact match for 13,23,53,103, interpolated otherwise).

8.1.4.436 `CImg<T>& draw_quiver (const CImg< t1 > & flow, const t2 *const color, const float opacity = 1, const unsigned int sampling = 25, const float factor = -20, const bool is_arrow = true, const unsigned int pattern = ~0U)`

Draw a 2d vector field.

Parameters

<i>flow</i>	Image of 2d vectors used as input data.
<i>color</i>	Image of spectrum() -D vectors corresponding to the color of each arrow.
<i>opacity</i>	Drawing opacity.
<i>sampling</i>	Length (in pixels) between each arrow.
<i>factor</i>	Length factor of each arrow (if <0, computed as a percentage of the maximum length).
<i>is_arrow</i>	Tells if arrows must be drawn, instead of oriented segments.
<i>pattern</i>	Used pattern to draw lines.

Note

Clipping is supported.

8.1.4.437 `CImg<T>& draw_quiver (const CImg< t1 > & flow, const CImg< t2 > & color, const float opacity = 1, const unsigned int sampling = 25, const float factor = -20, const bool is_arrow = true, const unsigned int pattern = ~0U)`

Draw a 2d vector field, using a field of colors.

Parameters

<i>flow</i>	Image of 2d vectors used as input data.
<i>color</i>	Image of spectrum() -D vectors corresponding to the color of each arrow.
<i>opacity</i>	Opacity of the drawing.
<i>sampling</i>	Length (in pixels) between each arrow.
<i>factor</i>	Length factor of each arrow (if <0, computed as a percentage of the maximum length).
<i>is_arrow</i>	Tells if arrows must be drawn, instead of oriented segments.
<i>pattern</i>	Used pattern to draw lines.

Note

Clipping is supported.

8.1.4.438 `CImg<T>& draw_axis (const CImg< t > & values_x, const int y, const tc *const color, const float opacity = 1, const unsigned int pattern = ~0U, const unsigned int font_height = 13, const bool allow_zero = true)`

Draw a labeled horizontal axis.

Parameters

<i>values_x</i>	Values along the horizontal axis.
<i>y</i>	Y-coordinate of the horizontal axis in the image instance.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	Drawing pattern.
<i>font_height</i>	Height of the labels (exact match for 13,23,53,103, interpolated otherwise).
<i>allow_zero</i>	Enable/disable the drawing of label '0' if found.

8.1.4.439 `CImg<T>& draw_axis (const int x, const CImg< t > & values_y, const tc *const color, const float opacity = 1, const unsigned int pattern = ~0U, const unsigned int font_height = 13, const bool allow_zero = true)`

Draw a labeled vertical axis.

Parameters

<i>x</i>	X-coordinate of the vertical axis in the image instance.
----------	--

Parameters

<i>values_y</i>	Values along the Y-axis.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern</i>	Drawing pattern.
<i>font_height</i>	Height of the labels (exact match for 13,23,53,103, interpolated otherwise).
<i>allow_zero</i>	Enable/disable the drawing of label '0' if found.

8.1.4.440 **CImg<T>& draw_axes (const CImg< tx > & values_x, const CImg< ty > & values_y, const tc *const color, const float opacity = 1, const unsigned int pattern_x = ~0U, const unsigned int pattern_y = ~0U, const unsigned int font_height = 13, const bool allow_zero = true)**

Draw labeled horizontal and vertical axes.

Parameters

<i>values_x</i>	Values along the X-axis.
<i>values_y</i>	Values along the Y-axis.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern_x</i>	Drawing pattern for the X-axis.
<i>pattern_y</i>	Drawing pattern for the Y-axis.
<i>font_height</i>	Height of the labels (exact match for 13,23,53,103, interpolated otherwise).
<i>allow_zero</i>	Enable/disable the drawing of label '0' if found.

8.1.4.441 **CImg<T>& draw_grid (const CImg< tx > & values_x, const CImg< ty > & values_y, const tc *const color, const float opacity = 1, const unsigned int pattern_x = ~0U, const unsigned int pattern_y = ~0U)**

Draw 2d grid.

Parameters

<i>values↔ _x</i>	X-coordinates of the vertical lines.
<i>values↔ _y</i>	Y-coordinates of the horizontal lines.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>pattern↔ _x</i>	Drawing pattern for vertical lines.
<i>pattern↔ _y</i>	Drawing pattern for horizontal lines.

8.1.4.442 `CImg<T>& draw_graph (const CImg< t > & data, const tc *const color, const float opacity = 1, const unsigned int plot_type = 1, const int vertex_type = 1, const double ymin = 0, const double ymax = 0, const unsigned int pattern = ~0U)`

Draw 1d graph.

Parameters

<i>data</i>	Image containing the graph values $I = f(x)$.
<i>color</i>	Pointer to <code>spectrum()</code> consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.
<i>plot_type</i>	Define the type of the plot: <ul style="list-style-type: none"> • 0 = No plot. • 1 = Plot using segments. • 2 = Plot using cubic splines. • 3 = Plot with bars.
<i>vertex_type</i>	Define the type of points: <ul style="list-style-type: none"> • 0 = No points. • 1 = Point. • 2 = Straight cross. • 3 = Diagonal cross. • 4 = Filled circle. • 5 = Outlined circle. • 6 = Square. • 7 = Diamond.
<i>ymin</i>	Lower bound of the y-range.
<i>ymax</i>	Upper bound of the y-range.
<i>pattern</i>	Drawing pattern.

Note

- if `ymin==ymax==0`, the y-range is computed automatically from the input samples.

8.1.4.443 `CImg<T>& draw_fill (const int x, const int y, const int z, const tc *const color, const float opacity, CImg< t > & region, const float sigma = 0, const bool is_high_connexity = false)`

Draw filled 3d region with the flood fill algorithm.

Parameters

	<i>x</i>	X-coordinate of the starting point of the region to fill.
	<i>y</i>	Y-coordinate of the starting point of the region to fill.
	<i>z</i>	Z-coordinate of the starting point of the region to fill.

Parameters

	<i>color</i>	Pointer to <code>spectrum()</code> consecutive values, defining the drawing color.
out	<i>region</i>	Image that will contain the mask of the filled region mask, as an output.
	<i>sigma</i>	Tolerance concerning neighborhood values.
	<i>opacity</i>	Opacity of the drawing.
	<i>is_high_connexity</i>	Tells if 8-connexity must be used (only for 2d images).

Returns

region is initialized with the binary mask of the filled region.

8.1.4.444 CImg<T>& draw_plasma (const float *alpha* = 1, const float *beta* = 0, const unsigned int *scale* = 8)

Draw a random plasma texture.

Parameters

<i>alpha</i>	Alpha-parameter.
<i>beta</i>	Beta-parameter.
<i>scale</i>	Scale-parameter.

Note

Use the mid-point algorithm to render.

8.1.4.445 CImg<T>& draw_mandelbrot (const int *x0*, const int *y0*, const int *x1*, const int *y1*, const CImg<tc> & *colormap*, const float *opacity* = 1, const double *z0r* = -2, const double *z0i* = -2, const double *z1r* = 2, const double *z1i* = 2, const unsigned int *iteration_max* = 255, const bool *is_normalized_iteration* = false, const bool *is_julia_set* = false, const double *param_r* = 0, const double *param_i* = 0)

Draw a quadratic Mandelbrot or Julia 2d fractal.

Parameters

<i>x0</i>	X-coordinate of the upper-left pixel.
<i>y0</i>	Y-coordinate of the upper-left pixel.
<i>x1</i>	X-coordinate of the lower-right pixel.
<i>y1</i>	Y-coordinate of the lower-right pixel.
<i>colormap</i>	Colormap.
<i>opacity</i>	Drawing opacity.
<i>z0r</i>	Real part of the upper-left fractal vertex.
<i>z0i</i>	Imaginary part of the upper-left fractal vertex.
<i>z1r</i>	Real part of the lower-right fractal vertex.
<i>z1i</i>	Imaginary part of the lower-right fractal vertex.
<i>iteration_max</i>	Maximum number of iterations for each estimated point.
<i>is_normalized_iteration</i>	Tells if iterations are normalized.

Parameters

<i>is_julia_set</i>	Tells if the Mandelbrot or Julia set is rendered.
<i>param_r</i>	Real part of the Julia set parameter.
<i>param_i</i>	Imaginary part of the Julia set parameter.

Note

Fractal rendering is done by the Escape Time Algorithm.

8.1.4.446 CImg<T>& draw_gaussian (const float xc, const float sigma, const tc *const color, const float opacity = 1)

Draw a 1d gaussian function.

Parameters

<i>xc</i>	X-coordinate of the gaussian center.
<i>sigma</i>	Standard variation of the gaussian distribution.
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.

8.1.4.447 CImg<T>& draw_gaussian (const float xc, const float yc, const CImg< t > & tensor, const tc *const color, const float opacity = 1)

Draw a 2d gaussian function.

Parameters

<i>xc</i>	X-coordinate of the gaussian center.
<i>yc</i>	Y-coordinate of the gaussian center.
<i>tensor</i>	Covariance matrix (must be 2x2).
<i>color</i>	Pointer to spectrum() consecutive values, defining the drawing color.
<i>opacity</i>	Drawing opacity.

8.1.4.448 CImg<T>& draw_object3d (const float x0, const float y0, const float z0, const CImg< tp > & vertices, const CImgList< tf > & primitives, const CImgList< tc > & colors, const CImg< to > & opacities, const unsigned int render_type = 4, const bool is_double_sided = false, const float focale = 700, const float lightx = 0, const float lighty = 0, const float lightz = -5e8, const float specular_lightness = 0.2f, const float specular_shininess = 0.1f)

Draw a 3d object.

Parameters

<i>x0</i>	X-coordinate of the 3d object position
<i>y0</i>	Y-coordinate of the 3d object position

Parameters

<i>z0</i>	Z-coordinate of the 3d object position
<i>vertices</i>	Image Nx3 describing 3d point coordinates
<i>primitives</i>	List of P primitives
<i>colors</i>	List of P color (or textures)
<i>opacities</i>	Image or list of P opacities
<i>render_type</i>	d Render type (0=Points, 1=Lines, 2=Faces (no light), 3=Faces (flat), 4=Faces(Gouraud)
<i>is_double_sided</i>	Tells if object faces have two sides or are oriented.
<i>focale</i>	length of the focale (0 for parallel projection)
<i>lightx</i>	X-coordinate of the light
<i>lighty</i>	Y-coordinate of the light
<i>lightz</i>	Z-coordinate of the light
<i>specular_lightness</i>	Amount of specular light.
<i>specular_shininess</i>	Shininess of the object

8.1.4.449 `CImg<T>& select (CImgDisplay & disp, const unsigned int feature_type = 2, unsigned int *const XYZ = 0, const bool exit_on_anykey = false)`

Launch simple interface to select a shape from an image.

Parameters

<i>disp</i>	Display window to use.
<i>feature_type</i>	Type of feature to select. Can be { 0=point 1=line 2=rectangle 3=ellipse }.
XYZ	Pointer to 3 values X,Y,Z which tells about the projection point coordinates, for volumetric images.

8.1.4.450 `CImg<T>& load (const char *const filename)`

Load image from a file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

Note

The extension of *filename* defines the file format. If no filename extension is provided, `CImg<T>::get_↵load()` will try to load the file as a .cimg or .cimgz file.

8.1.4.451 `CImg<T>& load_ascii (const char *const filename)`

Load image from an ascii file.

Parameters

<i>filename</i>	Filename, as a C -string.
-----------------	---------------------------

8.1.4.452 CImg<T>& load_dlm (const char *const *filename*)

Load image from a DLM file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.453 CImg<T>& load_bmp (const char *const *filename*)

Load image from a BMP file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.454 CImg<T>& load_jpeg (const char *const *filename*)

Load image from a JPEG file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.455 CImg<T>& load_magick (const char *const *filename*)

Load image from a file, using Magick++ library.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.456 CImg<T>& load_png (const char *const *filename*)

Load image from a PNG file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.457 **CImg<T>& load_pnm (const char *const filename)**

Load image from a PNM file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.458 **CImg<T>& load_pfm (const char *const filename)**

Load image from a PFM file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.459 **CImg<T>& load_rgb (const char *const filename, const unsigned int dimw, const unsigned int dimh = 1)**

Load image from a RGB file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>dimw</i>	Width of the image buffer.
<i>dimh</i>	Height of the image buffer.

8.1.4.460 **CImg<T>& load_rgba (const char *const filename, const unsigned int dimw, const unsigned int dimh = 1)**

Load image from a RGBA file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>dimw</i>	Width of the image buffer.
<i>dimh</i>	Height of the image buffer.

8.1.4.461 `CImg<T>& load_tiff (const char *const filename, const unsigned int first_frame = 0, const unsigned int last_frame = ~0U, const unsigned int step_frame = 1, float *const voxel_size = 0, CImg< charT > *const description = 0)`

Load image from a TIFF file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>first_frame</i>	First frame to read (for multi-pages tiff).
<i>last_frame</i>	Last frame to read (for multi-pages tiff).
<i>step_frame</i>	Step value of frame reading.

Note

- libtiff support is enabled by defining the precompilation directive `cimg_use_tif`.
- When libtiff is enabled, 2D and 3D (multipage) several channel per pixel are supported for `char`, `uchar`, `short`, `ushort`, `float` and `double` pixel types.
- If `cimg_use_tif` is not defined at compile time the function uses `CImg<T>& load_other(const char*)`.

8.1.4.462 `CImg<T>& load_minc2 (const char *const filename)`

Load image from a MINC2 file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.463 `CImg<T>& load_analyze (const char *const filename, float *const voxel_size = 0)`

Load image from an ANALYZE7.5/NIFTI file.

Parameters

	<i>filename</i>	Filename, as a C-string.
<i>out</i>	<i>voxel_size</i>	Pointer to the three voxel sizes read from the file.

8.1.4.464 `CImg<T>& load_cimg (const char *const filename, const char axis = 'z', const float align = 0)`

Load image from a .cimg[z] file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>axis</i>	Appending axis, if file contains multiple images. Can be { 'x' 'y' 'z' 'c' }.
<i>align</i>	Appending alignment.

8.1.4.465 `CImg<T>& load_cimg (const char *const filename, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1, const char axis = 'z', const float align = 0)`

Load sub-images of a .cimg file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>n0</i>	Starting frame.
<i>n1</i>	Ending frame (~0U for max).
<i>x0</i>	X-coordinate of the starting sub-image vertex.
<i>y0</i>	Y-coordinate of the starting sub-image vertex.
<i>z0</i>	Z-coordinate of the starting sub-image vertex.
<i>c0</i>	C-coordinate of the starting sub-image vertex.
<i>x1</i>	X-coordinate of the ending sub-image vertex (~0U for max).
<i>y1</i>	Y-coordinate of the ending sub-image vertex (~0U for max).
<i>z1</i>	Z-coordinate of the ending sub-image vertex (~0U for max).
<i>c1</i>	C-coordinate of the ending sub-image vertex (~0U for max).
<i>axis</i>	Appending axis, if file contains multiple images. Can be { 'x' 'y' 'z' 'c' }.
<i>align</i>	Appending alignment.

8.1.4.466 `CImg<T>& load_inr (const char *const filename, float *const voxel_size = 0)`

Load image from an INRIMAGE-4 file.

Parameters

	<i>filename</i>	Filename, as a C-string.
out	<i>voxel_size</i>	Pointer to the three voxel sizes read from the file.

8.1.4.467 `CImg<T>& load_exr (const char *const filename)`

Load image from a EXR file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.468 `CImg<T>& load_pandore (const char *const filename)`

Load image from a PANDORE-5 file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.469 `Clmg<T>& load_parrec (const char *const filename, const char axis = 'c', const float align = 0)`

Load image from a PAR-REC (Philips) file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>axis</i>	Appending axis, if file contains multiple images. Can be { 'x' 'y' 'z' 'c' }.
<i>align</i>	Appending alignment.

8.1.4.470 `Clmg<T>& load_raw (const char *const filename, const unsigned int size_x = 0, const unsigned int size_y = 1, const unsigned int size_z = 1, const unsigned int size_c = 1, const bool is_multiplexed = false, const bool invert_endianness = false, const alongT offset = 0)`

Load image from a raw binary file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>size_x</i>	Width of the image buffer.
<i>size_y</i>	Height of the image buffer.
<i>size_z</i>	Depth of the image buffer.
<i>size_c</i>	Spectrum of the image buffer.
<i>is_multiplexed</i>	Tells if the image values are multiplexed along the C-axis.
<i>invert_endianness</i>	Tells if the endianness of the image buffer must be inverted.
<i>offset</i>	Starting offset of the read in the specified file.

8.1.4.471 `Clmg<T>& load_yuv (const char *const filename, const unsigned int size_x, const unsigned int size_y = 1, const unsigned int first_frame = 0, const unsigned int last_frame = ~0U, const unsigned int step_frame = 1, const bool yuv2rgb = true, const char axis = 'z')`

Load image sequence from a YUV file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>size_x</i>	Width of the frames.
<i>size_y</i>	Height of the frames.
<i>first_frame</i>	Index of the first frame to read.
<i>last_frame</i>	Index of the last frame to read.
<i>step_frame</i>	Step value for frame reading.
<i>yuv2rgb</i>	Tells if the YUV to RGB transform must be applied.
<i>axis</i>	Appending axis, if file contains multiple images. Can be { 'x' 'y' 'z' 'c' }.

8.1.4.472 `Clmg<T>& load_off (ClmgList< tf > &primitives, ClmgList< tc > &colors, const char *const filename)`

Load 3d object from a .OFF file.

Parameters

out	<i>primitives</i>	Primitives data of the 3d object.
out	<i>colors</i>	Colors data of the 3d object.
	<i>filename</i>	Filename, as a C-string.

8.1.4.473 `Clmg<T>& load_video (const char *const filename, const unsigned int first_frame = 0, const unsigned int last_frame = ~0U, const unsigned int step_frame = 1, const char axis = 'z', const float align = 0)`

Load image sequence from a video file, using OpenCV library.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>first_frame</i>	Index of the first frame to read.
<i>last_frame</i>	Index of the last frame to read.
<i>step_frame</i>	Step value for frame reading.

8.1.4.474 `Clmg<T>& load_ffmpeg_external (const char *const filename, const char axis = 'z', const float align = 0)`

Load image sequence using FFMPEG's external tool 'ffmpeg'.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>axis</i>	Appending axis, if file contains multiple images. Can be { 'x' 'y' 'z' 'c' }.
<i>align</i>	Appending alignment.

8.1.4.475 `Clmg<T>& load_gif_external (const char *const filename, const char axis = 'z', const float align = 0)`

Load gif file, using Imagemagick or GraphicsMagicks's external tools.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>use_graphicsmagick</i>	Tells if GraphicsMagick's tool 'gm' is used instead of ImageMagick's tool 'convert'.
<i>axis</i>	Appending axis, if file contains multiple images. Can be { 'x' 'y' 'z' 'c' }.
<i>align</i>	Appending alignment.

8.1.4.476 CImg<T>& load_graphicsmagick_external (const char *const filename)

Load image using GraphicsMagick's external tool 'gm'.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.477 CImg<T>& load_gzip_external (const char *const filename)

Load gzipped image file, using external tool 'gunzip'.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.478 CImg<T>& load_imagemagick_external (const char *const filename)

Load image using ImageMagick's external tool 'convert'.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.479 CImg<T>& load_medcon_external (const char *const filename)

Load image from a DICOM file, using XMedcon's external tool 'medcon'.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.480 CImg<T>& load_dcraw_external (const char *const filename)

Load image from a RAW Color Camera file, using external tool 'dcraw'.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.481 `CImg<T>& load_camera (const unsigned int camera_index = 0, const unsigned int skip_frames = 0, const bool release_camera = true, const unsigned int capture_width = 0, const unsigned int capture_height = 0)`

Load image from a camera stream, using OpenCV.

Parameters

<i>camera_index</i>	Index of the camera to capture images from.
<i>skip_frames</i>	Number of frames to skip before the capture.
<i>release_camera</i>	Tells if the camera resource must be released at the end of the method.

8.1.4.482 `CImg<T>& load_other (const char *const filename)`

Load image using various non-native ways.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.483 `const CImg<T>& print (const char *const title = 0, const bool display_stats = true) const`

Display information about the image data.

Parameters

<i>title</i>	Name for the considered image.
<i>display_stats</i>	Tells to compute and display image statistics.

8.1.4.484 `const CImg<T>& display (CImgDisplay & disp) const`

Display image into a [CImgDisplay](#) window.

Parameters

<i>disp</i>	Display window.
-------------	-----------------

8.1.4.485 `const CImg<T>& display (CImgDisplay & disp, const bool display_info, unsigned int *const XYZ = 0, const bool exit_on_anykey = false) const`

Display image into a [CImgDisplay](#) window, in an interactive way.

Parameters

<i>disp</i>	Display window.
<i>display_info</i>	Tells if image information are displayed on the standard output.

8.1.4.486 `const CImg<T>& display (const char *const title = 0, const bool display_info = true, unsigned int *const XYZ = 0, const bool exit_on_anykey = false) const`

Display image into an interactive window.

Parameters

<i>title</i>	Window title
<i>display_info</i>	Tells if image information are displayed on the standard output.

8.1.4.487 `const CImg<T>& display_object3d (CImgDisplay & disp, const CImg<tp> & vertices, const CImgList<tf> & primitives, const CImgList<tc> & colors, const to & opacities, const bool centering = true, const int render_static = 4, const int render_motion = 1, const bool is_double_sided = true, const float focale = 700, const float light_x = 0, const float light_y = 0, const float light_z = -5e8f, const float specular_lightness = 0.2f, const float specular_shininess = 0.1f, const bool display_axes = true, float *const pose_matrix = 0, const bool exit_on_anykey = false) const`

Display object 3d in an interactive window.

Parameters

<i>disp</i>	Display window.
<i>vertices</i>	Vertices data of the 3d object.
<i>primitives</i>	Primitives data of the 3d object.
<i>colors</i>	Colors data of the 3d object.
<i>opacities</i>	Opacities data of the 3d object.
<i>centering</i>	Tells if the 3d object must be centered for the display.
<i>render_static</i>	Rendering mode.
<i>render_motion</i>	Rendering mode, when the 3d object is moved.
<i>is_double_sided</i>	Tells if the object primitives are double-sided.
<i>focale</i>	Focale
<i>light_x</i>	X-coordinate of the light source.
<i>light_y</i>	Y-coordinate of the light source.
<i>light_z</i>	Z-coordinate of the light source.
<i>specular_lightness</i>	Amount of specular light.
<i>specular_shininess</i>	Shininess of the object material.
<i>display_axes</i>	Tells if the 3d axes are displayed.
<i>pose_matrix</i>	Pointer to 12 values, defining a 3d pose (as a 4x3 matrix).

8.1.4.488 `const CImg<T>& display_graph (CImgDisplay & disp, const unsigned int plot_type = 1, const unsigned int vertex_type = 1, const char *const labelx = 0, const double xmin = 0, const double xmax = 0, const char *const labeledy = 0, const double ymin = 0, const double ymax = 0, const bool exit_on_anykey = false) const`

Display 1d graph in an interactive window.

Parameters

<i>disp</i>	Display window.
-------------	-----------------

Parameters

<i>plot_type</i>	Plot type. Can be { 0=points 1=segments 2=splines 3=bars }.
<i>vertex_type</i>	Vertex type.
<i>labelx</i>	Title for the horizontal axis, as a C-string.
<i>xmin</i>	Minimum value along the X-axis.
<i>xmax</i>	Maximum value along the X-axis.
<i>labeley</i>	Title for the vertical axis, as a C-string.
<i>ymin</i>	Minimum value along the X-axis.
<i>ymax</i>	Maximum value along the X-axis.

8.1.4.489 `const CImg<T>& save (const char *const filename, const int number = -1, const unsigned int digits = 6) const`

Save image as a file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>number</i>	When positive, represents an index added to the filename. Otherwise, no number is added.
<i>digits</i>	Number of digits used for adding the number to the filename.

Note

- The used file format is defined by the file extension in the filename *filename*.
- Parameter *number* can be used to add a 6-digit number to the filename before saving.

8.1.4.490 `const CImg<T>& save_ascii (const char *const filename) const`

Save image as an ascii file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.491 `const CImg<T>& save_cpp (const char *const filename) const`

Save image as a .cpp source file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.492 `const CImg<T>& save_dlm (const char *const filename) const`

Save image as a DLM file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.493 `const CImg<T>& save_bmp (const char *const filename) const`

Save image as a BMP file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.494 `const CImg<T>& save_jpeg (const char *const filename, const unsigned int quality = 100) const`

Save image as a JPEG file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>quality</i>	Image quality (in %)

8.1.4.495 `const CImg<T>& save_magick (const char *const filename, const unsigned int bytes_per_pixel = 0) const`

Save image, using built-in ImageMagick++ library.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>bytes_per_pixel</i>	Force the number of bytes per pixel for the saving, when possible.

8.1.4.496 `const CImg<T>& save_png (const char *const filename, const unsigned int bytes_per_pixel = 0) const`

Save image as a PNG file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>bytes_per_pixel</i>	Force the number of bytes per pixels for the saving, when possible.

8.1.4.497 `const CImg<T>& save_pnm (const char *const filename, const unsigned int bytes_per_pixel = 0) const`

Save image as a PNM file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>bytes_per_pixel</i>	Force the number of bytes per pixels for the saving.

8.1.4.498 `const CImg<T>& save_pnk (const char *const filename) const`

Save image as a PNK file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.499 `const CImg<T>& save_pfm (const char *const filename) const`

Save image as a PFM file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.500 `const CImg<T>& save_rgb (const char *const filename) const`

Save image as a RGB file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.501 `const CImg<T>& save_rgba (const char *const filename) const`

Save image as a RGBA file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.1.4.502 `const CImg<T>& save_tiff (const char *const filename, const unsigned int compression_type = 0, const float *const voxel_size = 0, const char *const description = 0, const bool use_bigtiff = true) const`

Save image as a TIFF file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>compression_type</i>	Type of data compression. Can be { 0=None 1=LZW 2=JPEG }.

Note

- libtiff support is enabled by defining the precompilation directive `cimg_use_tif`.
- When libtiff is enabled, 2D and 3D (multipage) several channel per pixel are supported for `char`, `uchar`, `short`, `ushort`, `float` and `double` pixel types.
- If `cimg_use_tif` is not defined at compile time the function uses `CImg<T>&save_other(const char*)`.

8.1.4.503 `const CImg<T>& save_minc2 (const char *const filename, const char *const imitate_file = 0) const`

Save image as a MINC2 file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>imitate_file</i>	If non-zero, reference filename, as a C-string, to borrow header from.

8.1.4.504 `const CImg<T>& save_analyze (const char *const filename, const float *const voxel_size = 0) const`

Save image as an ANALYZE7.5 or NIFTI file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>voxel_size</i>	Pointer to 3 consecutive values that tell about the voxel sizes along the X,Y and Z dimensions.

8.1.4.505 `const CImg<T>& save_cimg (const char *const filename, const bool is_compressed = false) const`

Save image as a .cimg file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>is_compressed</i>	Tells if the file contains compressed image data.

8.1.4.506 `const CImg<T>& save_cimg (const char *const filename, const unsigned int n0, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0) const`

Save image as a sub-image into an existing .cimg file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>n0</i>	Index of the image inside the file.
<i>x0</i>	X-coordinate of the sub-image location.
<i>y0</i>	Y-coordinate of the sub-image location.
<i>z0</i>	Z-coordinate of the sub-image location.
<i>c0</i>	C-coordinate of the sub-image location.

8.1.4.507 `static void save_empty_cimg (const char *const filename, const unsigned int dx, const unsigned int dy = 1, const unsigned int dz = 1, const unsigned int dc = 1) [static]`

Save blank image as a .cimg file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>dx</i>	Width of the image.
<i>dy</i>	Height of the image.
<i>dz</i>	Depth of the image.
<i>dc</i>	Number of channels of the image.

Note

- All pixel values of the saved image are set to 0.
- Use this method to save large images without having to instanciate and allocate them.

8.1.4.508 `static void save_empty_cimg (std::FILE *const file, const unsigned int dx, const unsigned int dy = 1, const unsigned int dz = 1, const unsigned int dc = 1) [static]`

Save blank image as a .cimg file **[overloading]**.

Same as [save_empty_cimg\(const char *,unsigned int,unsigned int,unsigned int,unsigned int\)](#) with a file stream argument instead of a filename string.

8.1.4.509 `const CImg<T>& save_inr (const char *const filename, const float *const voxel_size = 0) const`

Save image as an INRIMAGE-4 file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>voxel_size</i>	Pointer to 3 values specifying the voxel sizes along the X,Y and Z dimensions.

8.1.4.510 `const CImg<T>& save_exr (const char *const filename) const`

Save image as an OpenEXR file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

Note

The OpenEXR file format is [described here](#).

8.1.4.511 `const CImg<T>& save_pandore (const char *const filename, const unsigned int colorspace = 0) const`

Save image as a Pandore-5 file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>colorspace</i>	Colorspace data field in output file (see Pandore file specifications for more information).

8.1.4.512 `const CImg<T>& save_pandore (std::FILE *const file, const unsigned int colorspace = 0) const`

Save image as a Pandore-5 file **[overloading]**.

Same as [save_pandore\(const char *,unsigned int\) const](#) with a file stream argument instead of a filename string.

8.1.4.513 `const CImg<T>& save_raw (const char *const filename, const bool is_multiplexed = false) const`

Save image as a raw data file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>is_multiplexed</i>	Tells if the image channels are stored in a multiplexed way (<code>true</code>) or not (<code>false</code>).

Note

The .raw format does not store the image dimensions in the output file, so you have to keep track of them somewhere to be able to read the file correctly afterwards.

8.1.4.514 `const CImg<T>& save_raw (std::FILE *const file, const bool is_multiplexed = false) const`

Save image as a raw data file **[overloading]**.

Same as [save_raw\(const char *,bool\) const](#) with a file stream argument instead of a filename string.

8.1.4.515 `const CImg<T>& save_yuv (const char *const filename, const bool is_rgb = true) const`

Save image as a .yuv video file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>is_rgb</i>	Tells if pixel values of the instance image are RGB-coded (<code>true</code>) or YUV-coded (<code>false</code>).

Note

Each slice of the instance image is considered to be a single frame of the output video file.

8.1.4.516 `const CImg<T>& save_yuv (std::FILE *const file, const bool is_rgb = true) const`

Save image as a .yuv video file **[overloading]**.

Same as [save_yuv\(const char*,bool\) const](#) with a file stream argument instead of a filename string.

8.1.4.517 `const CImg<T>& save_off (const CImgList<tf> &primitives, const CImgList<tc> &colors, const char *const filename) const`

Save 3d object as an Object File Format (.off) file.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>primitives</i>	List of 3d object primitives.
<i>colors</i>	List of 3d object colors.

Note

- Instance image contains the vertices data of the 3d object.
- Textured, transparent or sphere-shaped primitives cannot be managed by the .off file format. Such primitives will be lost or simplified during file saving.

- The .off file format is [described here](#).

8.1.4.518 `const Clmg<T>& save_off (const ClmgList<tf> & primitives, const ClmgList<tc> & colors, std::FILE *const file) const`

Save 3d object as an Object File Format (.off) file **[overloading]**.

Same as `save_off(const ClmgList<tf>&,const ClmgList<tc>&,const char*) const` with a file stream argument instead of a filename string.

8.1.4.519 `const Clmg<T>& save_video (const char *const filename, const unsigned int fps = 25, const char * codec = 0, const bool keep_open = false) const`

Save volumetric image as a video, using the OpenCV library.

Parameters

<i>filename</i>	Filename to write data to.
<i>fps</i>	Number of frames per second.
<i>codec</i>	Type of compression (See http://www.fourcc.org/codecs.php to see available codecs).
<i>keep_open</i>	Tells if the video writer associated to the specified filename must be kept open or not (to allow frames to be added in the same file afterwards).

8.1.4.520 `const Clmg<T>& save_ffmpeg_external (const char *const filename, const unsigned int fps = 25, const char *const codec = 0, const unsigned int bitrate = 2048) const`

Save volumetric image as a video, using ffmpeg external binary.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>fps</i>	Video framerate.
<i>codec</i>	Video codec, as a C-string.
<i>bitrate</i>	Video bitrate.

Note

- Each slice of the instance image is considered to be a single frame of the output video file.
- This method uses `ffmpeg`, an external executable binary provided by [FFmpeg](#). It must be installed for the method to succeed.

8.1.4.521 `const Clmg<T>& save_gzip_external (const char *const filename) const`

Save image using gzip external binary.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

Note

This method uses `gzip`, an external executable binary provided by `gzip`. It must be installed for the method to succeed.

8.1.4.522 `const CImg<T>& save_graphicmagick_external (const char *const filename, const unsigned int quality = 100) const`

Save image using GraphicsMagick's external binary.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>quality</i>	Image quality (expressed in percent), when the file format supports it.

Note

This method uses `gm`, an external executable binary provided by `GraphicsMagick`. It must be installed for the method to succeed.

8.1.4.523 `const CImg<T>& save_imagemagick_external (const char *const filename, const unsigned int quality = 100) const`

Save image using ImageMagick's external binary.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>quality</i>	Image quality (expressed in percent), when the file format supports it.

Note

This method uses `convert`, an external executable binary provided by `ImageMagick`. It must be installed for the method to succeed.

8.1.4.524 `const CImg<T>& save_medcon_external (const char *const filename) const`

Save image as a Dicom file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

Note

This method uses `medcon`, an external executable binary provided by (X) `Medcon`. It must be installed for the method to succeed.

8.1.4.525 `const CImg<T> &save_other (const char *const filename, const unsigned int quality = 100) const`

Parameters

<i>filename</i>	Filename, as a C-string.
<i>quality</i>	Image quality (expressed in percent), when the file format supports it.

Note

- The filename extension tells about the desired file format.
- This method tries to save the instance image as a file, using external tools from `ImageMagick` or `GraphicsMagick`. At least one of these tool must be installed for the method to succeed.
- It is recommended to use the generic method `save(const char*, int) const` instead, as it can handle some file formats natively.

8.1.4.526 `CImg<ucharT> get_serialize (const bool is_compressed = false) const`

Serialize a `CImg<T>` instance into a raw `CImg<unsigned char>` buffer.

Parameters

<i>is_compressed</i>	tells if zlib compression must be used for serialization (this requires 'cimg_use_zlib' been enabled).
----------------------	--

8.2 CImgDisplay Struct Reference

Allow the creation of windows, display images on them and manage user events (keyboard, mouse and windows events).

Constructors / Destructor / Instance Management

- `~CImgDisplay ()`
Destructor.
- `CImgDisplay ()`
Construct an empty display.
- `CImgDisplay (const unsigned int width, const unsigned int height, const char *const title=0, const unsigned int normalization=3, const bool is_fullscreen=false, const bool is_closed=false)`
Construct a display with specified dimensions.
- `template<typename T > CImgDisplay (const CImg< T > &img, const char *const title=0, const unsigned int normalization=3, const bool is_fullscreen=false, const bool is_closed=false)`

- Construct a display from an image.*

 - `template<typename T >`
`CImgDisplay` (const `CImgList`< T > &list, const char *const title=0, const unsigned int normalization=3, const bool is_fullscreen=false, const bool is_closed=false)

Construct a display from an image list.

 - `CImgDisplay` (const `CImgDisplay` &disp)

Construct a display as a copy of an existing one.

 - `CImgDisplay` & `assign` ()

Destructor - Empty constructor [in-place version].

 - `CImgDisplay` & `assign` (const unsigned int width, const unsigned int height, const char *const title=0, const unsigned int normalization=3, const bool is_fullscreen=false, const bool is_closed=false)

Construct a display with specified dimensions [in-place version].

 - `template<typename T >`
`CImgDisplay` & `assign` (const `CImg`< T > &img, const char *const title=0, const unsigned int normalization=3, const bool is_fullscreen=false, const bool is_closed=false)

Construct a display from an image [in-place version].

 - `template<typename T >`
`CImgDisplay` & `assign` (const `CImgList`< T > &list, const char *const title=0, const unsigned int normalization=3, const bool is_fullscreen=false, const bool is_closed=false)

Construct a display from an image list [in-place version].

 - `CImgDisplay` & `assign` (const `CImgDisplay` &disp)

Construct a display as a copy of another one [in-place version].

 - static `CImgDisplay` & `empty` ()

Return a reference to an empty display.

 - static const `CImgDisplay` & `const_empty` ()

Return a reference to an empty display [const version].

Overloaded Operators

- `template<typename t >`
`CImgDisplay` & `operator=` (const `CImg`< t > &img)
Display image on associated window.
- `template<typename t >`
`CImgDisplay` & `operator=` (const `CImgList`< t > &list)
Display list of images on associated window.
- `CImgDisplay` & `operator=` (const `CImgDisplay` &disp)
Construct a display as a copy of another one [in-place version].
- `operator bool` () const
Return false if display is empty, true otherwise.

Instance Checking

- bool `is_empty` () const
Return true if display is empty, false otherwise.
- bool `is_closed` () const
Return true if display is closed (i.e. not visible on the screen), false otherwise.
- bool `is_resized` () const
Return true if associated window has been resized on the screen, false otherwise.
- bool `is_moved` () const
Return true if associated window has been moved on the screen, false otherwise.

- bool `is_event` () const
Return true if any event has occurred on the associated window, false otherwise.
- bool `is_fullscreen` () const
Return true if current display is in fullscreen mode, false otherwise.
- bool `is_key` () const
Return true if any key is being pressed on the associated window, false otherwise.
- bool `is_key` (const unsigned int `keycode`) const
Return true if key specified by given keycode is being pressed on the associated window, false otherwise.
- bool & `is_key` (const char *const `keycode`)
Return true if key specified by given keycode is being pressed on the associated window, false otherwise.
- bool `is_key_sequence` (const unsigned int *const `keycodes_sequence`, const unsigned int `length`, const bool `remove_sequence=false`)
Return true if specified key sequence has been typed on the associated window, false otherwise.
- bool `is_keyESC` () const
Return true if the ESC key is being pressed on the associated window, false otherwise.
- bool `is_keyF1` () const
- bool `is_keyF2` () const
- bool `is_keyF3` () const
- bool `is_keyF4` () const
- bool `is_keyF5` () const
- bool `is_keyF6` () const
- bool `is_keyF7` () const
- bool `is_keyF8` () const
- bool `is_keyF9` () const
- bool `is_keyF10` () const
- bool `is_keyF11` () const
- bool `is_keyF12` () const
- bool `is_keyPAUSE` () const
- bool `is_key1` () const
- bool `is_key2` () const
- bool `is_key3` () const
- bool `is_key4` () const
- bool `is_key5` () const
- bool `is_key6` () const
- bool `is_key7` () const
- bool `is_key8` () const
- bool `is_key9` () const
- bool `is_key0` () const
- bool `is_keyBACKSPACE` () const
- bool `is_keyINSERT` () const
- bool `is_keyHOME` () const
- bool `is_keyPAGEUP` () const
- bool `is_keyTAB` () const
- bool `is_keyQ` () const
- bool `is_keyW` () const
- bool `is_keyE` () const
- bool `is_keyR` () const
- bool `is_keyT` () const
- bool `is_keyY` () const
- bool `is_keyU` () const
- bool `is_keyI` () const
- bool `is_keyO` () const
- bool `is_keyP` () const

- `bool is_keyDELETE () const`
- `bool is_keyEND () const`
- `bool is_keyPAGEDOWN () const`
- `bool is_keyCAPSLOCK () const`
- `bool is_keyA () const`
- `bool is_keyS () const`
- `bool is_keyD () const`
- `bool is_keyF () const`
- `bool is_keyG () const`
- `bool is_keyH () const`
- `bool is_keyJ () const`
- `bool is_keyK () const`
- `bool is_keyL () const`
- `bool is_keyENTER () const`
- `bool is_keySHIFTLEFT () const`
- `bool is_keyZ () const`
- `bool is_keyX () const`
- `bool is_keyC () const`
- `bool is_keyV () const`
- `bool is_keyB () const`
- `bool is_keyN () const`
- `bool is_keyM () const`
- `bool is_keySHIFTRIGHT () const`
- `bool is_keyARROWUP () const`
- `bool is_keyCTRLLEFT () const`
- `bool is_keyAPPLEFT () const`
- `bool is_keyALT () const`
- `bool is_keySPACE () const`
- `bool is_keyALTGR () const`
- `bool is_keyAPPRIGHT () const`
- `bool is_keyMENU () const`
- `bool is_keyCTRLRIGHT () const`
- `bool is_keyARROWLEFT () const`
- `bool is_keyARROWDOWN () const`
- `bool is_keyARROWRIGHT () const`
- `bool is_keyPAD0 () const`
- `bool is_keyPAD1 () const`
- `bool is_keyPAD2 () const`
- `bool is_keyPAD3 () const`
- `bool is_keyPAD4 () const`
- `bool is_keyPAD5 () const`
- `bool is_keyPAD6 () const`
- `bool is_keyPAD7 () const`
- `bool is_keyPAD8 () const`
- `bool is_keyPAD9 () const`
- `bool is_keyPADADD () const`
- `bool is_keyPADSUB () const`
- `bool is_keyPADMUL () const`
- `bool is_keyPADDIV () const`

Instance Characteristics

- int [width](#) () const
Return display width.
- int [height](#) () const
Return display height.
- unsigned int [normalization](#) () const
Return normalization type of the display.
- const char * [title](#) () const
Return title of the associated window as a C-string.
- int [window_width](#) () const
Return width of the associated window.
- int [window_height](#) () const
Return height of the associated window.
- int [window_x](#) () const
Return X-coordinate of the associated window.
- int [window_y](#) () const
Return Y-coordinate of the associated window.
- int [mouse_x](#) () const
Return X-coordinate of the mouse pointer.
- int [mouse_y](#) () const
Return Y-coordinate of the mouse pointer.
- unsigned int [button](#) () const
Return current state of the mouse buttons.
- int [wheel](#) () const
Return current state of the mouse wheel.
- unsigned int [key](#) (const unsigned int pos=0) const
Return one entry from the pressed keys history.
- unsigned int [released_key](#) (const unsigned int pos=0) const
Return one entry from the released keys history.
- float [frames_per_second](#) ()
Return the current refresh rate, in frames per second.
- static int [screen_width](#) ()
Return width of the screen (current resolution along the X-axis).
- static int [screen_height](#) ()
Return height of the screen (current resolution along the Y-axis).
- static unsigned int [keycode](#) (const char *const keycode)
Return keycode corresponding to the specified string.

Window Manipulation

- template<typename T >
[CImgDisplay](#) & [display](#) (const [CImg](#)< T > &img)
Display image on associated window.
- template<typename T >
[CImgDisplay](#) & [display](#) (const [CImgList](#)< T > &list, const char axis='x', const float align=0)
Display list of images on associated window.
- [CImgDisplay](#) & [show](#) ()
Show (closed) associated window on the screen.
- [CImgDisplay](#) & [close](#) ()

- Close (visible) associated window and make it disappear from the screen.*

 - [CImgDisplay](#) & [move](#) (const int pos_x, const int pos_y)

Move associated window to a new location.
- [CImgDisplay](#) & [resize](#) (const bool force_redraw=true)

Resize display to the size of the associated window.
- [CImgDisplay](#) & [resize](#) (const int width, const int height, const bool force_redraw=true)

Resize display to the specified size.
- template<typename T >
[CImgDisplay](#) & [resize](#) (const [CImg](#)< T > &img, const bool force_redraw=true)

Resize display to the size of an input image.
- [CImgDisplay](#) & [resize](#) (const [CImgDisplay](#) &disp, const bool force_redraw=true)

Resize display to the size of another [CImgDisplay](#) instance.
- [CImgDisplay](#) & [set_normalization](#) (const unsigned int normalization)

Set normalization type.
- [CImgDisplay](#) & [set_title](#) (const char *const format,...)

Set title of the associated window.
- [CImgDisplay](#) & [set_fullscreen](#) (const bool is_fullscreen, const bool force_redraw=true)

Enable or disable fullscreen mode.
- [CImgDisplay](#) & [toggle_fullscreen](#) (const bool force_redraw=true)

Toggle fullscreen mode.
- [CImgDisplay](#) & [show_mouse](#) ()

Show mouse pointer.
- [CImgDisplay](#) & [hide_mouse](#) ()

Hide mouse pointer.
- [CImgDisplay](#) & [set_mouse](#) (const int pos_x, const int pos_y)

Move mouse pointer to a specified location.
- [CImgDisplay](#) & [set_button](#) ()

Simulate a mouse button release event.
- [CImgDisplay](#) & [set_button](#) (const unsigned int button, const bool is_pressed=true)

Simulate a mouse button press or release event.
- [CImgDisplay](#) & [set_wheel](#) ()

Flush all mouse wheel events.
- [CImgDisplay](#) & [set_wheel](#) (const int amplitude)

Simulate a wheel event.
- [CImgDisplay](#) & [set_key](#) ()

Flush all key events.
- [CImgDisplay](#) & [set_key](#) (const unsigned int keycode, const bool is_pressed=true)

Simulate a keyboard press/release event.
- [CImgDisplay](#) & [flush](#) ()

Flush all display events.
- [CImgDisplay](#) & [wait](#) ()

Wait for any user event occuring on the current display.
- [CImgDisplay](#) & [wait](#) (const unsigned int milliseconds)

Wait for a given number of milliseconds since the last call to [wait\(\)](#).
- template<typename T >
[CImgDisplay](#) & [render](#) (const [CImg](#)< T > &img)

Render image into internal display buffer.
- [CImgDisplay](#) & [paint](#) ()

Paint internal display buffer on associated window.
- template<typename T >
const [CImgDisplay](#) & [snapshot](#) ([CImg](#)< T > &img) const

Take a snapshot of the associated window content.

- static void [wait](#) ([CImgDisplay](#) &disp1)
Wait for any event occurring on the display `disp1`.
- static void [wait](#) ([CImgDisplay](#) &disp1, [CImgDisplay](#) &disp2)
Wait for any event occurring either on the display `disp1` or `disp2`.
- static void [wait](#) ([CImgDisplay](#) &disp1, [CImgDisplay](#) &disp2, [CImgDisplay](#) &disp3)
Wait for any event occurring either on the display `disp1`, `disp2` or `disp3`.
- static void [wait](#) ([CImgDisplay](#) &disp1, [CImgDisplay](#) &disp2, [CImgDisplay](#) &disp3, [CImgDisplay](#) &disp4)
Wait for any event occurring either on the display `disp1`, `disp2`, `disp3` or `disp4`.
- static void [wait](#) ([CImgDisplay](#) &disp1, [CImgDisplay](#) &disp2, [CImgDisplay](#) &disp3, [CImgDisplay](#) &disp4, [CImgDisplay](#) &disp5)
Wait for any event occurring either on the display `disp1`, `disp2`, `disp3`, `disp4` or `disp5`.
- static void [wait](#) ([CImgDisplay](#) &disp1, [CImgDisplay](#) &disp2, [CImgDisplay](#) &disp3, [CImgDisplay](#) &disp4, [CImgDisplay](#) &disp5, [CImgDisplay](#) &disp6)
Wait for any event occurring either on the display `disp1`, `disp2`, `disp3`, `disp4`, ... `disp6`.
- static void [wait](#) ([CImgDisplay](#) &disp1, [CImgDisplay](#) &disp2, [CImgDisplay](#) &disp3, [CImgDisplay](#) &disp4, [CImgDisplay](#) &disp5, [CImgDisplay](#) &disp6, [CImgDisplay](#) &disp7)
Wait for any event occurring either on the display `disp1`, `disp2`, `disp3`, `disp4`, ... `disp7`.
- static void [wait](#) ([CImgDisplay](#) &disp1, [CImgDisplay](#) &disp2, [CImgDisplay](#) &disp3, [CImgDisplay](#) &disp4, [CImgDisplay](#) &disp5, [CImgDisplay](#) &disp6, [CImgDisplay](#) &disp7, [CImgDisplay](#) &disp8)
Wait for any event occurring either on the display `disp1`, `disp2`, `disp3`, `disp4`, ... `disp8`.
- static void [wait](#) ([CImgDisplay](#) &disp1, [CImgDisplay](#) &disp2, [CImgDisplay](#) &disp3, [CImgDisplay](#) &disp4, [CImgDisplay](#) &disp5, [CImgDisplay](#) &disp6, [CImgDisplay](#) &disp7, [CImgDisplay](#) &disp8, [CImgDisplay](#) &disp9)
Wait for any event occurring either on the display `disp1`, `disp2`, `disp3`, `disp4`, ... `disp9`.
- static void [wait](#) ([CImgDisplay](#) &disp1, [CImgDisplay](#) &disp2, [CImgDisplay](#) &disp3, [CImgDisplay](#) &disp4, [CImgDisplay](#) &disp5, [CImgDisplay](#) &disp6, [CImgDisplay](#) &disp7, [CImgDisplay](#) &disp8, [CImgDisplay](#) &disp9, [CImgDisplay](#) &disp10)
Wait for any event occurring either on the display `disp1`, `disp2`, `disp3`, `disp4`, ... `disp10`.
- static void [wait_all](#) ()
Wait for any window event occurring in any opened [CImgDisplay](#).

8.2.1 Detailed Description

Allow the creation of windows, display images on them and manage user events (keyboard, mouse and windows events).

[CImgDisplay](#) methods rely on a low-level graphic library to perform: it can be either **X-Window** (X11, for Unix-based systems) or **GDI32** (for Windows-based systems). If both libraries are missing, [CImgDisplay](#) will not be able to display images on screen, and will enter a minimal mode where warning messages will be outputted each time the program is trying to call one of the [CImgDisplay](#) method.

The configuration variable `cimg_display` tells about the graphic library used. It is set automatically by [CImg](#) when one of these graphic libraries has been detected. But, you can override its value if necessary. Valid choices are:

- 0: Disable display capabilities.
- 1: Use **X-Window** (X11) library.
- 2: Use **GDI32** library.

Remember to link your program against **X11** or **GDI32** libraries if you use [CImgDisplay](#).

8.2.2 Constructor & Destructor Documentation

8.2.2.1 ~CImgDisplay ()

Destructor.

Note

If the associated window is visible on the screen, it is closed by the call to the destructor.

8.2.2.2 CImgDisplay ()

Construct an empty display.

Note

Constructing an empty [CImgDisplay](#) instance does not make a window appearing on the screen, until display of valid data is performed.

Example

```
CImgDisplay disp; // Does actually nothing.
...
disp.display(img); // Construct new window and display image in it.
```

8.2.2.3 CImgDisplay (const unsigned int *width*, const unsigned int *height*, const char *const *title* = 0, const unsigned int *normalization* = 3, const bool *is_fullscreen* = false, const bool *is_closed* = false)

Construct a display with specified dimensions.

Parameters

<i>width</i>	Window width.
<i>height</i>	Window height.
<i>title</i>	Window title.
<i>normalization</i>	Normalization type (0=none, 1=always, 2=once, 3=pixel type-dependent, see normalization()).
<i>is_fullscreen</i>	Tells if fullscreen mode is enabled.
<i>is_closed</i>	Tells if associated window is initially visible or not.

Note

A black background is initially displayed on the associated window.

8.2.2.4 CImgDisplay (const CImg< T > & *img*, const char *const *title* = 0, const unsigned int *normalization* = 3, const bool *is_fullscreen* = false, const bool *is_closed* = false) [explicit]

Construct a display from an image.

Parameters

<i>img</i>	Image used as a model to create the window.
<i>title</i>	Window title.
<i>normalization</i>	Normalization type (0=none, 1=always, 2=once, 3=pixel type-dependent, see normalization()).
<i>is_fullscreen</i>	Tells if fullscreen mode is enabled.
<i>is_closed</i>	Tells if associated window is initially visible or not.

Note

The pixels of the input image are initially displayed on the associated window.

8.2.2.5 CImgDisplay (const CImgList< T > & list, const char *const title = 0, const unsigned int normalization = 3, const bool is_fullscreen = false, const bool is_closed = false) [explicit]

Construct a display from an image list.

Parameters

<i>list</i>	The images list to display.
<i>title</i>	Window title.
<i>normalization</i>	Normalization type (0=none, 1=always, 2=once, 3=pixel type-dependent, see normalization()).
<i>is_fullscreen</i>	Tells if fullscreen mode is enabled.
<i>is_closed</i>	Tells if associated window is initially visible or not.

Note

All images of the list, appended along the X-axis, are initially displayed on the associated window.

8.2.2.6 CImgDisplay (const CImgDisplay & disp)

Construct a display as a copy of an existing one.

Parameters

<i>disp</i>	Display instance to copy.
-------------	---------------------------

Note

The pixel buffer of the input window is initially displayed on the associated window.

8.2.3 Member Function Documentation

8.2.3.1 CImgDisplay& assign ()

Destructor - Empty constructor **[in-place version]**.

Note

Replace the current instance by an empty display.

8.2.3.2 static CImgDisplay& empty () [static]

Return a reference to an empty display.

Note

Can be useful for writing function prototypes where one of the argument (of type [CImgDisplay&](#)) must have a default value.

Example

```
void foo(CImgDisplay& disp=CImgDisplay::empty());
```

8.2.3.3 CImgDisplay& operator= (const CImg< t > & img)

Display image on associated window.

Note

`disp = img` is equivalent to `disp.display(img)`.

8.2.3.4 CImgDisplay& operator= (const CImgList< t > & list)

Display list of images on associated window.

Note

`disp = list` is equivalent to `disp.display(list)`.

8.2.3.5 CImgDisplay& operator= (const CImgDisplay & disp)

Construct a display as a copy of another one **[in-place version]**.

Note

Equivalent to [assign\(const CImgDisplay&\)](#).

8.2.3.6 operator bool () const

Return `false` if display is empty, `true` otherwise.

Note

`if (disp) { ... }` is equivalent to `if (!disp.is_empty()) { ... }`.

8.2.3.7 bool is_closed () const

Return `true` if display is closed (i.e. not visible on the screen), `false` otherwise.

Note

- When a user physically closes the associated window, the display is set to closed.
- A closed display is not destroyed. Its associated window can be show again on the screen using [show\(\)](#).

8.2.3.8 bool is_key () const

Return `true` if any key is being pressed on the associated window, `false` otherwise.

Note

The methods below do the same only for specific keys.

8.2.3.9 bool is_key (const unsigned int *keycode*) const

Return `true` if key specified by given keycode is being pressed on the associated window, `false` otherwise.

Parameters

<i>keycode</i>	Keycode to test.
----------------	------------------

Note

Keycode constants are defined in the `cimg` namespace and are architecture-dependent. Use them to ensure your code stay portable (see [cimg::keyESC](#)).

Example

```
CImgDisplay disp(400,400);
while (!disp.is_closed()) {
    if (disp.key(cimg::keyTAB)) { ... } // Equivalent to 'if (disp.is_keyTAB())'.
    disp.wait();
}
```

8.2.3.10 bool& is_key (const char *const *keycode*)

Return `true` if key specified by given keycode is being pressed on the associated window, `false` otherwise.

Parameters

<i>keycode</i>	C-string containing the keycode label of the key to test.
----------------	---

Note

Use it when the key you want to test can be dynamically set by the user.

Example

```
CImgDisplay disp(400,400);
const char *const keycode = "TAB";
while (!disp.is_closed()) {
    if (disp.is_key(keycode)) { ... } // Equivalent to 'if (disp.is_keyTAB())'.
    disp.wait();
}
```

8.2.3.11 `bool is_key_sequence (const unsigned int *const keycodes_sequence, const unsigned int length, const bool remove_sequence = false)`

Return `true` if specified key sequence has been typed on the associated window, `false` otherwise.

Parameters

<i>keycodes_sequence</i>	Buffer of keycodes to test.
<i>length</i>	Number of keys in the <i>keycodes_sequence</i> buffer.
<i>remove_sequence</i>	Tells if the key sequence must be removed from the key history, if found.

Note

Keycode constants are defined in the `cimg` namespace and are architecture-dependent. Use them to ensure your code stay portable (see [cimg::keyESC](#)).

Example

```
CImgDisplay disp(400,400);
const unsigned int key_seq[] = { cimg::keyCTRLLEFT, cimg::keyD };
while (!disp.is_closed()) {
    if (disp.is_key_sequence(key_seq,2)) { ... } // Test for the 'CTRL+D' keyboard event.
    disp.wait();
}
```

8.2.3.12 `bool is_keyESC () const`

Return `true` if the `ESC` key is being pressed on the associated window, `false` otherwise.

Note

Similar methods exist for all keys managed by `CImg` (see [cimg::keyESC](#)).

8.2.3.13 `int width () const`

Return display width.

Note

The width of the display (i.e. the width of the pixel data buffer associated to the [CImgDisplay](#) instance) may be different from the actual width of the associated window.

8.2.3.14 int height () const

Return display height.

Note

The height of the display (i.e. the height of the pixel data buffer associated to the [CImgDisplay](#) instance) may be different from the actual height of the associated window.

8.2.3.15 unsigned int normalization () const

Return normalization type of the display.

The normalization type tells about how the values of an input image are normalized by the [CImgDisplay](#) to be correctly displayed. The range of values for pixels displayed on screen is $[0, 255]$. If the range of values of the data to display is different, a normalization may be required for displaying the data in a correct way. The normalization type can be one of:

- 0: Value normalization is disabled. It is then assumed that all input data to be displayed by the [CImgDisplay](#) instance have values in range $[0, 255]$.
- 1: Value normalization is always performed (this is the default behavior). Before displaying an input image, its values will be (virtually) stretched in range $[0, 255]$, so that the contrast of the displayed pixels will be maximum. Use this mode for images whose minimum and maximum values are not prescribed to known values (e.g. float-valued images). Note that when normalized versions of images are computed for display purposes, the actual values of these images are not modified.
- 2: Value normalization is performed once (on the first image display), then the same normalization coefficients are kept for next displayed frames.
- 3: Value normalization depends on the pixel type of the data to display. For integer pixel types, the normalization is done regarding the minimum/maximum values of the type (no normalization occurs then for `unsigned char`). For float-valued pixel types, the normalization is done regarding the minimum/maximum value of the image data instead.

8.2.3.16 const char* title () const

Return title of the associated window as a C-string.

Note

Window title may be not visible, depending on the used window manager or if the current display is in fullscreen mode.

8.2.3.17 int window_width () const

Return width of the associated window.

Note

The width of the display (i.e. the width of the pixel data buffer associated to the [CImgDisplay](#) instance) may be different from the actual width of the associated window.

8.2.3.18 `int window_height () const`

Return height of the associated window.

Note

The height of the display (i.e. the height of the pixel data buffer associated to the [CImgDisplay](#) instance) may be different from the actual height of the associated window.

8.2.3.19 `int window_x () const`

Return X-coordinate of the associated window.

Note

The returned coordinate corresponds to the location of the upper-left corner of the associated window.

8.2.3.20 `int window_y () const`

Return Y-coordinate of the associated window.

Note

The returned coordinate corresponds to the location of the upper-left corner of the associated window.

8.2.3.21 `int mouse_x () const`

Return X-coordinate of the mouse pointer.

Note

- If the mouse pointer is outside window area, `-1` is returned.
- Otherwise, the returned value is in the range `[0,width()-1]`.

8.2.3.22 `int mouse_y () const`

Return Y-coordinate of the mouse pointer.

Note

- If the mouse pointer is outside window area, `-1` is returned.
- Otherwise, the returned value is in the range `[0,height()-1]`.

8.2.3.23 unsigned int button () const

Return current state of the mouse buttons.

Note

Three mouse buttons can be managed. If one button is pressed, its corresponding bit in the returned value is set:

- bit 0 (value 0x1): State of the left mouse button.
- bit 1 (value 0x2): State of the right mouse button.
- bit 2 (value 0x4): State of the middle mouse button.

Several bits can be activated if more than one button are pressed at the same time.

Example

```
CImgDisplay disp(400,400);
while (!disp.is_closed()) {
    if (disp.button() & 1) { // Left button clicked.
        ...
    }
    if (disp.button() & 2) { // Right button clicked.
        ...
    }
    if (disp.button() & 4) { // Middle button clicked.
        ...
    }
    disp.wait();
}
```

8.2.3.24 int wheel () const

Return current state of the mouse wheel.

Note

- The returned value can be positive or negative depending on whether the mouse wheel has been scrolled forward or backward.
- Scrolling the wheel forward add 1 to the wheel value.
- Scrolling the wheel backward subtract 1 to the wheel value.
- The returned value cumulates the number of forward or backward scrolls since the creation of the display, or since the last reset of the wheel value (using [set_wheel\(\)](#)). It is strongly recommended to quickly reset the wheel counter when an action has been performed regarding the current wheel value. Otherwise, the returned wheel value may be for instance 0 despite the fact that many scrolls have been done (as many in forward as in backward directions).

Example

```
CImgDisplay disp(400,400);
while (!disp.is_closed()) {
    if (disp.wheel()) {
        int counter = disp.wheel(); // Read the state of the mouse wheel.
        ...                       // Do what you want with 'counter'.
        disp.set_wheel();          // Reset the wheel value to 0.
    }
    disp.wait();
}
```

8.2.3.25 unsigned int key (const unsigned int pos = 0) const

Return one entry from the pressed keys history.

Parameters

<i>pos</i>	Indice to read from the pressed keys history (indice 0 corresponds to latest entry).
------------	--

Returns

Keycode of a pressed key or 0 for a released key.

Note

- Each [CImgDisplay](#) stores a history of the pressed keys in a buffer of size 128. When a new key is pressed, its keycode is stored in the pressed keys history. When a key is released, 0 is put instead. This means that up to the 64 last pressed keys may be read from the pressed keys history. When a new value is stored, the pressed keys history is shifted so that the latest entry is always stored at position 0.
- Keycode constants are defined in the `cimg` namespace and are architecture-dependent. Use them to ensure your code stay portable (see [cimg::keyESC](#)).

8.2.3.26 `unsigned int released_key (const unsigned int pos = 0) const`

Return one entry from the released keys history.

Parameters

<i>pos</i>	Indice to read from the released keys history (indice 0 corresponds to latest entry).
------------	---

Returns

Keycode of a released key or 0 for a pressed key.

Note

- Each [CImgDisplay](#) stores a history of the released keys in a buffer of size 128. When a new key is released, its keycode is stored in the pressed keys history. When a key is pressed, 0 is put instead. This means that up to the 64 last released keys may be read from the released keys history. When a new value is stored, the released keys history is shifted so that the latest entry is always stored at position 0.
- Keycode constants are defined in the `cimg` namespace and are architecture-dependent. Use them to ensure your code stay portable (see [cimg::keyESC](#)).

8.2.3.27 `static unsigned int keycode (const char *const keycode) [static]`

Return keycode corresponding to the specified string.

Note

Keycode constants are defined in the `cimg` namespace and are architecture-dependent. Use them to ensure your code stay portable (see [cimg::keyESC](#)).

Example

```
const unsigned int keyTAB = CImgDisplay::keycode("TAB"); // Return cimg::keyTAB.
```


8.2.3.28 float frames_per_second ()

Return the current refresh rate, in frames per second.

Note

Returns a significant value when the current instance is used to display successive frames. It measures the delay between successive calls to [frames_per_second\(\)](#).

8.2.3.29 CImgDisplay& display (const CImg< T > & img)

Display image on associated window.

Parameters

<i>img</i>	Input image to display.
------------	-------------------------

Note

This method returns immediately.

8.2.3.30 CImgDisplay& display (const CImgList< T > & list, const char axis = 'x', const float align = 0)

Display list of images on associated window.

Parameters

<i>list</i>	List of images to display.
<i>axis</i>	Axis used to append the images along, for the visualization (can be x, y, z or c).
<i>align</i>	Relative position of aligned images when displaying lists with images of different sizes (0 for upper-left, 0.5 for centering and 1 for lower-right).

Note

This method returns immediately.

8.2.3.31 CImgDisplay& show ()

Show (closed) associated window on the screen.

Note

- Force the associated window of a display to be visible on the screen, even if it has been closed before.
- Using [show\(\)](#) on a visible display does nothing.

8.2.3.32 CImgDisplay& close ()

Close (visible) associated window and make it disappear from the screen.

Note

- A closed display only means the associated window is not visible anymore. This does not mean the display has been destroyed. Use [show\(\)](#) to make the associated window reappear.
- Using [close\(\)](#) on a closed display does nothing.

8.2.3.33 CImgDisplay& move (const int *pos_x*, const int *pos_y*)

Move associated window to a new location.

Parameters

<i>pos_x</i>	X-coordinate of the new window location.
<i>pos_y</i>	Y-coordinate of the new window location.

Note

Depending on the window manager behavior, this method may not succeed (no exceptions are thrown nevertheless).

8.2.3.34 CImgDisplay& resize (const bool *force_redraw* = true)

Resize display to the size of the associated window.

Parameters

<i>force_redraw</i>	Tells if the previous window content must be updated and refreshed as well.
---------------------	---

Note

- Calling this method ensures that [width\(\)](#) and [window_width\(\)](#) become equal, as well as [height\(\)](#) and [window_height\(\)](#).
- The associated window is also resized to specified dimensions.

8.2.3.35 CImgDisplay& resize (const int *width*, const int *height*, const bool *force_redraw* = true)

Resize display to the specified size.

Parameters

<i>width</i>	Requested display width.
<i>height</i>	Requested display height.
<i>force_redraw</i>	Tells if the previous window content must be updated and refreshed as well.

Note

The associated window is also resized to specified dimensions.

8.2.3.36 CImgDisplay&resize (const CImg< T > &img, const bool force_redraw = true)

Resize display to the size of an input image.

Parameters

<i>img</i>	Input image to take size from.
<i>force_redraw</i>	Tells if the previous window content must be resized and updated as well.

Note

- Calling this method ensures that `width()` and `img.width()` become equal, as well as `height()` and `img.height()`.
- The associated window is also resized to specified dimensions.

8.2.3.37 CImgDisplay&resize (const CImgDisplay &disp, const bool force_redraw = true)

Resize display to the size of another [CImgDisplay](#) instance.

Parameters

<i>disp</i>	Input display to take size from.
<i>force_redraw</i>	Tells if the previous window content must be resized and updated as well.

Note

- Calling this method ensures that `width()` and `disp.width()` become equal, as well as `height()` and `disp.height()`.
- The associated window is also resized to specified dimensions.

8.2.3.38 CImgDisplay&set_normalization (const unsigned int normalization)

Set normalization type.

Parameters

<i>normalization</i>	New normalization mode.
----------------------	-------------------------

8.2.3.39 CImgDisplay& set_title (const char *const *format*, ...)

Set title of the associated window.

Parameters

<i>format</i>	C-string containing the format of the title, as with <code>std::printf()</code> .
---------------	---

Warning

As the first argument is a format string, it is highly recommended to write

```
disp.set_title("%s", window_title);
```

instead of

```
disp.set_title(window_title);
```

if `window_title` can be arbitrary, to prevent nasty memory access.

8.2.3.40 CImgDisplay& set_fullscreen (const bool *is_fullscreen*, const bool *force_redraw* = true)

Enable or disable fullscreen mode.

Parameters

<i>is_fullscreen</i>	Tells is the fullscreen mode must be activated or not.
<i>force_redraw</i>	Tells if the previous window content must be displayed as well.

Note

- When the fullscreen mode is enabled, the associated window fills the entire screen but the size of the current display is not modified.
- The screen resolution may be switched to fit the associated window size and ensure it appears the largest as possible. For X-Window (X11) users, the configuration flag `cimg_use_xrandr` has to be set to allow the screen resolution change (requires the X11 extensions to be enabled).

8.2.3.41 CImgDisplay& toggle_fullscreen (const bool *force_redraw* = true)

Toggle fullscreen mode.

Parameters

<i>force_redraw</i>	Tells if the previous window content must be displayed as well.
---------------------	---

Note

Enable fullscreen mode if it was not enabled, and disable it otherwise.

8.2.3.42 CImgDisplay& show_mouse ()

Show mouse pointer.

Note

Depending on the window manager behavior, this method may not succeed (no exceptions are thrown nevertheless).

8.2.3.43 CImgDisplay& hide_mouse ()

Hide mouse pointer.

Note

Depending on the window manager behavior, this method may not succeed (no exceptions are thrown nevertheless).

8.2.3.44 CImgDisplay& set_mouse (const int *pos_x*, const int *pos_y*)

Move mouse pointer to a specified location.

Note

Depending on the window manager behavior, this method may not succeed (no exceptions are thrown nevertheless).

8.2.3.45 CImgDisplay& set_button ()

Simulate a mouse button release event.

Note

All mouse buttons are considered released at the same time.

8.2.3.46 CImgDisplay& set_button (const unsigned int *button*, const bool *is_pressed* = true)

Simulate a mouse button press or release event.

Parameters

<i>button</i>	Buttons event code, where each button is associated to a single bit.
<i>is_pressed</i>	Tells if the mouse button is considered as pressed or released.

8.2.3.47 CImgDisplay& set_wheel ()

Flush all mouse wheel events.

Note

Make [wheel\(\)](#) to return 0, if called afterwards.

8.2.3.48 CImgDisplay& set_wheel (const int *amplitude*)

Simulate a wheel event.

Parameters

<i>amplitude</i>	Amplitude of the wheel scrolling to simulate.
------------------	---

Note

Make [wheel\(\)](#) to return *amplitude*, if called afterwards.

8.2.3.49 CImgDisplay& set_key ()

Flush all key events.

Note

Make [key\(\)](#) to return 0, if called afterwards.

8.2.3.50 CImgDisplay& set_key (const unsigned int *keycode*, const bool *is_pressed* = true)

Simulate a keyboard press/release event.

Parameters

<i>keycode</i>	Keycode of the associated key.
<i>is_pressed</i>	Tells if the key is considered as pressed or released.

Note

Keycode constants are defined in the cimg namespace and are architecture-dependent. Use them to ensure your code stay portable (see [cimg::keyESC](#)).

8.2.3.51 CImgDisplay& flush ()

Flush all display events.

Note

Remove all passed events from the current display.

8.2.3.52 CImgDisplay& wait (const unsigned int *milliseconds*)

Wait for a given number of milliseconds since the last call to [wait\(\)](#).

Parameters

<i>milliseconds</i>	Number of milliseconds to wait for.
---------------------	-------------------------------------

Note

Similar to [cimg::wait\(\)](#).

8.2.3.53 CImgDisplay& render (const CImg< T > & *img*)

Render image into internal display buffer.

Parameters

<i>img</i>	Input image data to render.
------------	-----------------------------

Note

- Convert image data representation into the internal display buffer (architecture-dependent structure).
- The content of the associated window is not modified, until [paint\(\)](#) is called.
- Should not be used for common [CImgDisplay](#) uses, since [display\(\)](#) is more useful.

8.2.3.54 CImgDisplay& paint ()

Paint internal display buffer on associated window.

Note

- Update the content of the associated window with the internal display buffer, e.g. after a [render\(\)](#) call.
- Should not be used for common [CImgDisplay](#) uses, since [display\(\)](#) is more useful.

8.2.3.55 `const CImgDisplay& snapshot (CImg< T > & img) const`

Take a snapshot of the associated window content.

Parameters

out	img	Output snapshot. Can be empty on input.
-----	-----	---

8.3 CImgException Struct Reference

Instances of [CImgException](#) are thrown when errors are encountered in a `CImg` function call.

Inherits exception.

Inherited by `CImgArgumentException`, `CImgDisplayException`, `CImgInstanceException`, `CImgIOException`, and `CImgWarningException`.

Public Member Functions

- `const char * what () const throw ()`
Return a C-string containing the error message associated to the thrown exception.

8.3.1 Detailed Description

Instances of [CImgException](#) are thrown when errors are encountered in a `CImg` function call.

Overview

[CImgException](#) is the base class of all exceptions thrown by `CImg` (except `CImgAbortException`). [CImgException](#) is never thrown itself. Derived classes that specify the type of error are thrown instead. These classes can be:

- **CImgAbortException:** Thrown when a computationally-intensive function is aborted by an external signal. This is the only `non-derived` exception class.
- **CImgArgumentException:** Thrown when one argument of a called `CImg` function is invalid. This is probably one of the most thrown exception by `CImg`. For instance, the following example throws a `CImgArgumentException`:

```
CImg<float> img(100,100,1,3); // Define a 100x100 color image with float-valued pixels.
img.mirror('e');             // Try to mirror image along the (non-existing) 'e'-axis.
```

- **CImgDisplayException:** Thrown when something went wrong during the display of images in [CImgDisplay](#) instances.
- **CImgInstanceException:** Thrown when an instance associated to a called `CImg` method does not fit the function requirements. For instance, the following example throws a `CImgInstanceException`:

```
const CImg<float> img;          // Define an empty image.
const float value = img.at(0); // Try to read first pixel value (does not exist).
```


- **CImgIOException:** Thrown when an error occurred when trying to load or save image files. This happens when trying to read files that do not exist or with invalid formats. For instance, the following example throws a `CImgIOException`:

```
const CImg<float> img("missing_file.jpg"); // Try to load a file that does not exist.
```

- **CImgWarningException:** Thrown only if configuration macro `cimg_strict_warnings` is set, and when a `CImg` function has to display a warning message (see `cimg::warn()`).

It is not recommended to throw `CImgException` instances by yourself, since they are expected to be thrown only by `CImg`. When an error occurs in a library function call, `CImg` may display error messages on the screen or on the standard output, depending on the current `CImg` exception mode. The `CImg` exception mode can be get and set by functions `cimg::exception_mode()` and `cimg::exception_mode(unsigned int)`.

Exceptions handling

In all cases, when an error occurs in `CImg`, an instance of the corresponding exception class is thrown. This may lead the program to break (this is the default behavior), but you can bypass this behavior by handling the exceptions by yourself, using a usual `try { ... } catch () { ... }` bloc, as in the following example:

```
#define "CImg.h"
using namespace cimg_library;
int main() {
    cimg::exception_mode(0); // Enable quiet exception
    mode.
    try {
        ... // Here, do what you want to stress CImg.
    } catch (CImgException& e) { // You succeeded: something went
        wrong!
        std::fprintf(stderr, "CImg Library Error: %s", e.what()); // Display your custom error message.
        ... // Do what you want now to save the ship!
    }
}
```

8.4 CImgList< T > Struct Template Reference

Represent a list of images `CImg<T>`.

Public Types

- typedef `CImg< T > * iterator`
Simple iterator type, to loop through each image of a list.
- typedef const `CImg< T > * const_iterator`
Simple const iterator type, to loop through each image of a `const` list instance.
- typedef `T value_type`
Pixel value type.

Constructors / Destructor / Instance Management

- [~CImgList](#) ()
Destructor.
- [CImgList](#) ()
Default constructor.
- [CImgList](#) (const unsigned int n)
Construct list containing empty images.
- [CImgList](#) (const unsigned int n, const unsigned int [width](#), const unsigned int height=1, const unsigned int depth=1, const unsigned int spectrum=1)
Construct list containing images of specified size.
- [CImgList](#) (const unsigned int n, const unsigned int [width](#), const unsigned int height, const unsigned int depth, const unsigned int spectrum, const T &val)
Construct list containing images of specified size, and initialize pixel values.
- [CImgList](#) (const unsigned int n, const unsigned int [width](#), const unsigned int height, const unsigned int depth, const unsigned int spectrum, const int val0, const int val1,...)
Construct list containing images of specified size, and initialize pixel values from a sequence of integers.
- [CImgList](#) (const unsigned int n, const unsigned int [width](#), const unsigned int height, const unsigned int depth, const unsigned int spectrum, const double val0, const double val1,...)
Construct list containing images of specified size, and initialize pixel values from a sequence of doubles.
- template<typename t >
[CImgList](#) (const unsigned int n, const [CImg](#)< t > &img, const bool is_shared=false)
Construct list containing copies of an input image.
- template<typename t >
[CImgList](#) (const [CImg](#)< t > &img, const bool is_shared=false)
Construct list from one image.
- template<typename t1 , typename t2 >
[CImgList](#) (const [CImg](#)< t1 > &img1, const [CImg](#)< t2 > &img2, const bool is_shared=false)
Construct list from two images.
- template<typename t1 , typename t2 , typename t3 >
[CImgList](#) (const [CImg](#)< t1 > &img1, const [CImg](#)< t2 > &img2, const [CImg](#)< t3 > &img3, const bool is_↵
shared=false)
Construct list from three images.
- template<typename t1 , typename t2 , typename t3 , typename t4 >
[CImgList](#) (const [CImg](#)< t1 > &img1, const [CImg](#)< t2 > &img2, const [CImg](#)< t3 > &img3, const [CImg](#)< t4
> &img4, const bool is_shared=false)
Construct list from four images.
- template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 >
[CImgList](#) (const [CImg](#)< t1 > &img1, const [CImg](#)< t2 > &img2, const [CImg](#)< t3 > &img3, const [CImg](#)< t4
> &img4, const [CImg](#)< t5 > &img5, const bool is_shared=false)
Construct list from five images.
- template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 , typename t6 >
[CImgList](#) (const [CImg](#)< t1 > &img1, const [CImg](#)< t2 > &img2, const [CImg](#)< t3 > &img3, const [CImg](#)< t4
> &img4, const [CImg](#)< t5 > &img5, const [CImg](#)< t6 > &img6, const bool is_shared=false)
Construct list from six images.
- template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 , typename t6 , typename t7 >
[CImgList](#) (const [CImg](#)< t1 > &img1, const [CImg](#)< t2 > &img2, const [CImg](#)< t3 > &img3, const [CImg](#)< t4
> &img4, const [CImg](#)< t5 > &img5, const [CImg](#)< t6 > &img6, const [CImg](#)< t7 > &img7, const bool
is_shared=false)
Construct list from seven images.
- template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 , typename t6 , typename t7 , typename t8 >
[CImgList](#) (const [CImg](#)< t1 > &img1, const [CImg](#)< t2 > &img2, const [CImg](#)< t3 > &img3, const [CImg](#)< t4
> &img4, const [CImg](#)< t5 > &img5, const [CImg](#)< t6 > &img6, const [CImg](#)< t7 > &img7, const [CImg](#)< t8
> &img8, const bool is_shared=false)

- Construct list from eight images.*

 - `template<typename t >`
`ClmgList (const ClmgList< t > &list)`

Construct list copy.

 - `ClmgList (const ClmgList< T > &list)`

*Construct list copy **[specialization]**.*

 - `template<typename t >`
`ClmgList (const ClmgList< t > &list, const bool is_shared)`

Construct list copy, and force the shared state of the list elements.

 - `ClmgList (const char *const filename)`

Construct list by reading the content of a file.

 - `ClmgList (const ClmgDisplay &disp)`

Construct list from the content of a display window.

 - `ClmgList< T > get_shared ()`

Return a list with elements being shared copies of images in the list instance.

 - `const ClmgList< T > get_shared () const`

*Return a list with elements being shared copies of images in the list instance **[const version]**.*

 - `ClmgList< T > & assign ()`

*Destructor **[in-place version]**.*

 - `ClmgList< T > & clear ()`

*Destructor **[in-place version]**.*

 - `ClmgList< T > & assign (const unsigned int n)`

*Construct list containing empty images **[in-place version]**.*

 - `ClmgList< T > & assign (const unsigned int n, const unsigned int width, const unsigned int height=1, const unsigned int depth=1, const unsigned int spectrum=1)`

*Construct list containing images of specified size **[in-place version]**.*

 - `ClmgList< T > & assign (const unsigned int n, const unsigned int width, const unsigned int height, const unsigned int depth, const unsigned int spectrum, const T &val)`

*Construct list containing images of specified size, and initialize pixel values **[in-place version]**.*

 - `ClmgList< T > & assign (const unsigned int n, const unsigned int width, const unsigned int height, const unsigned int depth, const unsigned int spectrum, const int val0, const int val1,...)`

*Construct list with images of specified size, and initialize pixel values from a sequence of integers **[in-place version]**.*

 - `ClmgList< T > & assign (const unsigned int n, const unsigned int width, const unsigned int height, const unsigned int depth, const unsigned int spectrum, const double val0, const double val1,...)`

*Construct list with images of specified size, and initialize pixel values from a sequence of doubles **[in-place version]**.*

 - `template<typename t >`
`ClmgList< T > & assign (const unsigned int n, const Clmg< t > &img, const bool is_shared=false)`

*Construct list containing copies of an input image **[in-place version]**.*

 - `template<typename t >`
`ClmgList< T > & assign (const Clmg< t > &img, const bool is_shared=false)`

*Construct list from one image **[in-place version]**.*

 - `template<typename t1, typename t2 >`
`ClmgList< T > & assign (const Clmg< t1 > &img1, const Clmg< t2 > &img2, const bool is_shared=false)`

*Construct list from two images **[in-place version]**.*

 - `template<typename t1, typename t2, typename t3 >`
`ClmgList< T > & assign (const Clmg< t1 > &img1, const Clmg< t2 > &img2, const Clmg< t3 > &img3, const bool is_shared=false)`

*Construct list from three images **[in-place version]**.*

 - `template<typename t1, typename t2, typename t3, typename t4 >`
`ClmgList< T > & assign (const Clmg< t1 > &img1, const Clmg< t2 > &img2, const Clmg< t3 > &img3, const Clmg< t4 > &img4, const bool is_shared=false)`

*Construct list from four images **[in-place version]**.*

- `template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 >`
`CImgList< T > & assign (const CImg< t1 > &img1, const CImg< t2 > &img2, const CImg< t3 > &img3,`
`const CImg< t4 > &img4, const CImg< t5 > &img5, const bool is_shared=false)`
Construct list from five images [in-place version].
- `template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 , typename t6 >`
`CImgList< T > & assign (const CImg< t1 > &img1, const CImg< t2 > &img2, const CImg< t3 > &img3,`
`const CImg< t4 > &img4, const CImg< t5 > &img5, const CImg< t6 > &img6, const bool is_shared=false)`
Construct list from six images [in-place version].
- `template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 , typename t6 , typename t7 >`
`CImgList< T > & assign (const CImg< t1 > &img1, const CImg< t2 > &img2, const CImg< t3 > &img3,`
`const CImg< t4 > &img4, const CImg< t5 > &img5, const CImg< t6 > &img6, const CImg< t7 > &img7,`
`const bool is_shared=false)`
Construct list from seven images [in-place version].
- `template<typename t1 , typename t2 , typename t3 , typename t4 , typename t5 , typename t6 , typename t7 , typename t8 >`
`CImgList< T > & assign (const CImg< t1 > &img1, const CImg< t2 > &img2, const CImg< t3 > &img3,`
`const CImg< t4 > &img4, const CImg< t5 > &img5, const CImg< t6 > &img6, const CImg< t7 > &img7,`
`const CImg< t8 > &img8, const bool is_shared=false)`
Construct list from eight images [in-place version].
- `template<typename t >`
`CImgList< T > & assign (const CImgList< t > &list, const bool is_shared=false)`
Construct list as a copy of an existing list and force the shared state of the list elements [in-place version].
- `CImgList< T > & assign (const CImgList< T > &list, const bool is_shared=false)`
Construct list as a copy of an existing list and force shared state of elements [in-place version] [specialization].
- `CImgList< T > & assign (const char *const filename)`
Construct list by reading the content of a file [in-place version].
- `CImgList< T > & assign (const CImgDisplay &disp)`
Construct list from the content of a display window [in-place version].
- `template<typename t >`
`CImgList< t > & move_to (CImgList< t > &list)`
Transfer the content of the list instance to another list.
- `template<typename t >`
`CImgList< t > & move_to (CImgList< t > &list, const unsigned int pos)`
Transfer the content of the list instance at a specified position in another list.
- `CImgList< T > & swap (CImgList< T > &list)`
Swap all fields between two list instances.
- `static CImgList< T > & empty ()`
Return a reference to an empty list.
- `static const CImgList< T > & const_empty ()`
Return a reference to an empty list [const version].

Overloaded Operators

- `CImg< T > & operator() (const unsigned int pos)`
Return a reference to one image element of the list.
- `const CImg< T > & operator() (const unsigned int pos) const`
Return a reference to one image of the list.
- `T & operator() (const unsigned int pos, const unsigned int x, const unsigned int y=0, const unsigned int z=0,`
`const unsigned int c=0)`
Return a reference to one pixel value of one image of the list.
- `const T & operator() (const unsigned int pos, const unsigned int x, const unsigned int y=0, const unsigned int`
`z=0, const unsigned int c=0) const`
Return a reference to one pixel value of one image of the list [const version].

- `operator CImg< T > * ()`
Return pointer to the first image of the list.
- `operator const CImg< T > * () const`
Return pointer to the first image of the list **[const version]**.
- `template<typename t >`
`CImgList< T > & operator= (const CImg< t > &img)`
Construct list from one image **[in-place version]**.
- `template<typename t >`
`CImgList< T > & operator= (const CImgList< t > &list)`
Construct list from another list.
- `CImgList< T > & operator= (const CImgList< T > &list)`
Construct list from another list **[specialization]**.
- `CImgList< T > & operator= (const char *const filename)`
Construct list by reading the content of a file **[in-place version]**.
- `CImgList< T > & operator= (const CImgDisplay &disp)`
Construct list from the content of a display window **[in-place version]**.
- `CImgList< T > operator+ () const`
Return a non-shared copy of a list.
- `template<typename t >`
`CImgList< T > & operator, (const CImg< t > &img)`
Return a copy of the list instance, where image `img` has been inserted at the end.
- `template<typename t >`
`CImgList< T > operator, (const CImg< t > &img) const`
Return a copy of the list instance, where image `img` has been inserted at the end **[const version]**.
- `template<typename t >`
`CImgList< T > & operator, (const CImgList< t > &list)`
Return a copy of the list instance, where all elements of input list `list` have been inserted at the end.
- `template<typename t >`
`CImgList< T > & operator, (const CImgList< t > &list) const`
Return a copy of the list instance, where all elements of input `list` have been inserted at the end **[const version]**.
- `CImg< T > operator> (const char axis) const`
Return image corresponding to the appending of all images of the instance list along specified axis.
- `CImgList< T > operator< (const char axis) const`
Return list corresponding to the splitting of all images of the instance list along specified axis.

Instance Characteristics

- `int width () const`
Return the size of the list, i.e. the number of images contained in it.
- `unsigned int size () const`
Return the size of the list, i.e. the number of images contained in it.
- `CImg< T > * data ()`
Return pointer to the first image of the list.
- `const CImg< T > * data () const`
Return pointer to the first image of the list **[const version]**.
- `CImg< T > * data (const unsigned int pos)`
Return pointer to the `pos`-th image of the list.
- `const CImg< T > * data (const unsigned int l) const`
- `iterator begin ()`
Return iterator to the first image of the list.
- `const_iterator begin () const`

- Return iterator to the first image of the list [const version].*

 - `iterator end ()`

Return iterator to one position after the last image of the list.

 - `const_iterator end () const`

Return iterator to one position after the last image of the list [const version].

 - `CImg< T > & front ()`

Return reference to the first image of the list.

 - `const CImg< T > & front () const`

Return reference to the first image of the list [const version].

 - `const CImg< T > & back () const`

Return a reference to the last image of the list.

 - `CImg< T > & back ()`

Return a reference to the last image of the list [const version].

 - `CImg< T > & at (const int pos)`

Return pos-th image of the list.

 - `T & atXYZC (const int pos, const int x, const int y, const int z, const int c, const T &out_value)`

Access to pixel value with Dirichlet boundary conditions.

 - `T atXYZC (const int pos, const int x, const int y, const int z, const int c, const T &out_value) const`

Access to pixel value with Dirichlet boundary conditions [const version].

 - `T & atXYZC (const int pos, const int x, const int y, const int z, const int c)`

Access to pixel value with Neumann boundary conditions.

 - `T atXYZC (const int pos, const int x, const int y, const int z, const int c) const`

Access to pixel value with Neumann boundary conditions [const version].

 - `T & atXYZ (const int pos, const int x, const int y, const int z, const int c, const T &out_value)`

Access pixel value with Dirichlet boundary conditions for the 3 first coordinates (pos, x,y,z).

 - `T atXYZ (const int pos, const int x, const int y, const int z, const int c, const T &out_value) const`

Access pixel value with Dirichlet boundary conditions for the 3 first coordinates (pos, x,y,z) [const version].

 - `T & atXYZ (const int pos, const int x, const int y, const int z, const int c=0)`

Access to pixel value with Neumann boundary conditions for the 4 first coordinates (pos, x,y,z).

 - `T atXYZ (const int pos, const int x, const int y, const int z, const int c=0) const`

Access to pixel value with Neumann boundary conditions for the 4 first coordinates (pos, x,y,z) [const version].

 - `T & atXY (const int pos, const int x, const int y, const int z, const int c, const T &out_value)`

Access to pixel value with Dirichlet boundary conditions for the 3 first coordinates (pos, x,y).

 - `T atXY (const int pos, const int x, const int y, const int z, const int c, const T &out_value) const`

Access to pixel value with Dirichlet boundary conditions for the 3 first coordinates (pos, x,y) [const version].

 - `T & atXY (const int pos, const int x, const int y, const int z=0, const int c=0)`

Access to pixel value with Neumann boundary conditions for the 3 first coordinates (pos, x,y).

 - `T atXY (const int pos, const int x, const int y, const int z=0, const int c=0) const`

Access to pixel value with Neumann boundary conditions for the 3 first coordinates (pos, x,y) [const version].

 - `T & atX (const int pos, const int x, const int y, const int z, const int c, const T &out_value)`

Access to pixel value with Dirichlet boundary conditions for the 2 first coordinates (pos,x).

 - `T atX (const int pos, const int x, const int y, const int z, const int c, const T &out_value) const`

Access to pixel value with Dirichlet boundary conditions for the 2 first coordinates (pos,x) [const version].

 - `T & atX (const int pos, const int x, const int y=0, const int z=0, const int c=0)`

Access to pixel value with Neumann boundary conditions for the 2 first coordinates (pos, x).

 - `T atX (const int pos, const int x, const int y=0, const int z=0, const int c=0) const`

Access to pixel value with Neumann boundary conditions for the 2 first coordinates (pos, x) [const version].

 - `T & at (const int pos, const int x, const int y, const int z, const int c, const T &out_value)`

Access to pixel value with Dirichlet boundary conditions for the first coordinate (pos).

 - `T at (const int pos, const int x, const int y, const int z, const int c, const T &out_value) const`

Access to pixel value with Dirichlet boundary conditions for the first coordinate (pos) [const version].

- T & **atN** (const int pos, const int x=0, const int y=0, const int z=0, const int c=0)
*Return pixel value with Neumann boundary conditions for the first coordinate (*pos*).*
- T **atN** (const int pos, const int x=0, const int y=0, const int z=0, const int c=0) const
*Return pixel value with Neumann boundary conditions for the first coordinate (*pos*) [**const version**].*
- CImg< charT > **value_string** (const char separator=',', const unsigned int max_size=0) const
Return a C-string containing the values of all images in the instance list.
- static const char * **pixel_type** ()
Return the type of image pixel values as a C string.

Instance Checking

- bool **is_empty** () const
*Return *true* if list is empty.*
- bool **is_sameN** (const unsigned int size_n) const
Test if number of image elements is equal to specified value.
- template<typename t >
bool **is_sameN** (const CImgList< t > &list) const
Test if number of image elements is equal between two images lists.
- template<typename t >
bool **is_sameXY** (const CImg< t > &img) const
- template<typename t >
bool **is_sameXY** (const CImgList< t > &list) const
- template<typename t >
bool **is_sameNXY** (const unsigned int n, const CImg< t > &img) const
- template<typename t >
bool **is_sameNXY** (const CImgList< t > &list) const
- template<typename t >
bool **is_sameXZ** (const CImg< t > &img) const
- template<typename t >
bool **is_sameXZ** (const CImgList< t > &list) const
- template<typename t >
bool **is_sameNXZ** (const unsigned int n, const CImg< t > &img) const
- template<typename t >
bool **is_sameNXZ** (const CImgList< t > &list) const
- template<typename t >
bool **is_sameXC** (const CImg< t > &img) const
- template<typename t >
bool **is_sameXC** (const CImgList< t > &list) const
- template<typename t >
bool **is_sameNXC** (const unsigned int n, const CImg< t > &img) const
- template<typename t >
bool **is_sameNXC** (const CImgList< t > &list) const
- template<typename t >
bool **is_sameYZ** (const CImg< t > &img) const
- template<typename t >
bool **is_sameYZ** (const CImgList< t > &list) const
- template<typename t >
bool **is_sameNYZ** (const unsigned int n, const CImg< t > &img) const
- template<typename t >
bool **is_sameNYZ** (const CImgList< t > &list) const
- template<typename t >
bool **is_sameYC** (const CImg< t > &img) const
- template<typename t >
bool **is_sameYC** (const CImgList< t > &list) const

- `template<typename t >`
`bool is_sameNYC (const unsigned int n, const Clmg< t > &img) const`
- `template<typename t >`
`bool is_sameNYC (const ClmgList< t > &list) const`
- `template<typename t >`
`bool is_sameXYZ (const Clmg< t > &img) const`
- `template<typename t >`
`bool is_sameXYZ (const ClmgList< t > &list) const`
- `template<typename t >`
`bool is_sameNXYZ (const unsigned int n, const Clmg< t > &img) const`
- `template<typename t >`
`bool is_sameNXYZ (const ClmgList< t > &list) const`
- `template<typename t >`
`bool is_sameXYC (const Clmg< t > &img) const`
- `template<typename t >`
`bool is_sameXYC (const ClmgList< t > &list) const`
- `template<typename t >`
`bool is_sameNXYC (const unsigned int n, const Clmg< t > &img) const`
- `template<typename t >`
`bool is_sameNXYC (const ClmgList< t > &list) const`
- `template<typename t >`
`bool is_sameYZC (const Clmg< t > &img) const`
- `template<typename t >`
`bool is_sameYZC (const ClmgList< t > &list) const`
- `template<typename t >`
`bool is_sameNYZC (const unsigned int n, const Clmg< t > &img) const`
- `template<typename t >`
`bool is_sameNYZC (const ClmgList< t > &list) const`
- `template<typename t >`
`bool is_sameXYZC (const Clmg< t > &img) const`
- `template<typename t >`
`bool is_sameXYZC (const ClmgList< t > &list) const`
- `template<typename t >`
`bool is_sameNXYZC (const unsigned int n, const Clmg< t > &img) const`
- `template<typename t >`
`bool is_sameNXYZC (const ClmgList< t > &list) const`
- `bool is_sameX (const unsigned int val) const`
- `bool is_sameNX (const unsigned int n, const unsigned int val) const`
- `bool is_sameY (const unsigned int val) const`
- `bool is_sameNY (const unsigned int n, const unsigned int val) const`
- `bool is_sameZ (const unsigned int val) const`
- `bool is_sameNZ (const unsigned int n, const unsigned int val) const`
- `bool is_sameC (const unsigned int val) const`
- `bool is_sameNC (const unsigned int n, const unsigned int val) const`
- `bool is_sameXY (const unsigned int val1, const unsigned int val2) const`
- `bool is_sameNXY (const unsigned int n, const unsigned int val1, const unsigned int val2) const`
- `bool is_sameXZ (const unsigned int val1, const unsigned int val2) const`
- `bool is_sameNXZ (const unsigned int n, const unsigned int val1, const unsigned int val2) const`
- `bool is_sameXC (const unsigned int val1, const unsigned int val2) const`
- `bool is_sameNXC (const unsigned int n, const unsigned int val1, const unsigned int val2) const`
- `bool is_sameYZ (const unsigned int val1, const unsigned int val2) const`
- `bool is_sameNYZ (const unsigned int n, const unsigned int val1, const unsigned int val2) const`
- `bool is_sameYC (const unsigned int val1, const unsigned int val2) const`
- `bool is_sameNYC (const unsigned int n, const unsigned int val1, const unsigned int val2) const`
- `bool is_sameZC (const unsigned int val1, const unsigned int val2) const`
- `bool is_sameNZC (const unsigned int n, const unsigned int val1, const unsigned int val2) const`

- bool **is_sameXYZ** (const unsigned int val1, const unsigned int val2, const unsigned int val3) const
- bool **is_sameNXYZ** (const unsigned int n, const unsigned int val1, const unsigned int val2, const unsigned int val3) const
- bool **is_sameXYC** (const unsigned int val1, const unsigned int val2, const unsigned int val3) const
- bool **is_sameNXYC** (const unsigned int n, const unsigned int val1, const unsigned int val2, const unsigned int val3) const
- bool **is_sameXZC** (const unsigned int val1, const unsigned int val2, const unsigned int val3) const
- bool **is_sameNXZC** (const unsigned int n, const unsigned int val1, const unsigned int val2, const unsigned int val3) const
- bool **is_sameYZC** (const unsigned int val1, const unsigned int val2, const unsigned int val3) const
- bool **is_sameNYZC** (const unsigned int n, const unsigned int val1, const unsigned int val2, const unsigned int val3) const
- bool **is_sameXYZC** (const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dc) const
Test if dimensions of each image of the list match specified arguments.
- bool **is_sameNXYZC** (const unsigned int n, const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dc) const
Test if list dimensions match specified arguments.
- bool **containsNXYZC** (const int n, const int x=0, const int y=0, const int z=0, const int c=0) const
Test if list contains one particular pixel location.
- bool **containsN** (const int n) const
Test if list contains image with specified indice.
- template<typename t >
 bool **contains** (const T &pixel, t &n, t &x, t &y, t &z, t &c) const
Test if one image of the list contains the specified referenced value.
- template<typename t >
 bool **contains** (const T &pixel, t &n, t &x, t &y, t &z) const
Test if one of the image list contains the specified referenced value.
- template<typename t >
 bool **contains** (const T &pixel, t &n, t &x, t &y) const
Test if one of the image list contains the specified referenced value.
- template<typename t >
 bool **contains** (const T &pixel, t &n, t &x) const
Test if one of the image list contains the specified referenced value.
- template<typename t >
 bool **contains** (const T &pixel, t &n) const
Test if one of the image list contains the specified referenced value.
- bool **contains** (const T &pixel) const
Test if one of the image list contains the specified referenced value.
- template<typename t >
 bool **contains** (const CImg< T > &img, t &n) const
Test if the list contains the image 'img'.
- bool **contains** (const CImg< T > &img) const
Test if the list contains the image img.

Mathematical Functions

- T & **min** ()
Return a reference to the minimum pixel value of the instance list.
- const T & **min** () const
Return a reference to the minimum pixel value of the instance list [const version].
- T & **max** ()

- Return a reference to the maximum pixel value of the instance list.*

 - `const T & max () const`

*Return a reference to the maximum pixel value of the instance list **[const version]**.*
- `template<typename t >`
`T & min_max (t &max_val)`

Return a reference to the minimum pixel value of the instance list and return the maximum vvalue as well.
- `template<typename t >`
`const T & min_max (t &max_val) const`

*Return a reference to the minimum pixel value of the instance list and return the maximum vvalue as well **[const version]**.*
- `template<typename t >`
`T & max_min (t &min_val)`

Return a reference to the minimum pixel value of the instance list and return the minimum value as well.
- `template<typename t >`
`const T & max_min (t &min_val) const`

*Return a reference to the minimum pixel value of the instance list and return the minimum value as well **[const version]**.*

List Manipulation

- `template<typename t >`
`CImgList< T > & insert (const CImg< t > &img, const unsigned int pos=~0U, const bool is_shared=false)`

Insert a copy of the image `img` into the current image list, at position `pos`.
- `CImgList< T > & insert (const CImg< T > &img, const unsigned int pos=~0U, const bool is_shared=false)`

*Insert a copy of the image `img` into the current image list, at position `pos` **[specialization]**.*
- `template<typename t >`
`CImgList< T > get_insert (const CImg< t > &img, const unsigned int pos=~0U, const bool is_shared=false)`
`const`

*Insert a copy of the image `img` into the current image list, at position `pos` **[new-instance version]**.*
- `CImgList< T > & insert (const unsigned int n, const unsigned int pos=~0U)`

Insert `n` empty images `img` into the current image list, at position `pos`.
- `CImgList< T > get_insert (const unsigned int n, const unsigned int pos=~0U) const`

*Insert `n` empty images `img` into the current image list, at position `pos` **[new-instance version]**.*
- `template<typename t >`
`CImgList< T > & insert (const unsigned int n, const CImg< t > &img, const unsigned int pos=~0U, const bool is_shared=false)`

Insert `n` copies of the image `img` into the current image list, at position `pos`.
- `template<typename t >`
`CImgList< T > get_insert (const unsigned int n, const CImg< t > &img, const unsigned int pos=~0U, const bool is_shared=false) const`

*Insert `n` copies of the image `img` into the current image list, at position `pos` **[new-instance version]**.*
- `template<typename t >`
`CImgList< T > & insert (const CImgList< t > &list, const unsigned int pos=~0U, const bool is_shared=false)`

Insert a copy of the image list `list` into the current image list, starting from position `pos`.
- `template<typename t >`
`CImgList< T > get_insert (const CImgList< t > &list, const unsigned int pos=~0U, const bool is_↵
shared=false) const`

*Insert a copy of the image list `list` into the current image list, starting from position `pos` **[new-instance version]**.*
- `template<typename t >`
`CImgList< T > & insert (const unsigned int n, const CImgList< t > &list, const unsigned int pos=~0U, const bool is_shared=false)`

Insert `n` copies of the list `list` at position `pos` of the current list.

- `template<typename t >`
`ClmgList< T > get_insert` (const unsigned int n, const `ClmgList< t >` &list, const unsigned int pos=~0U, const bool is_shared=false) const
*Insert n copies of the list `list` at position `pos` of the current list **[new-instance version]**.*
- `ClmgList< T > & remove` (const unsigned int pos1, const unsigned int pos2)
Remove all images between from indexes.
- `ClmgList< T > get_remove` (const unsigned int pos1, const unsigned int pos2) const
*Remove all images between from indexes **[new-instance version]**.*
- `ClmgList< T > & remove` (const unsigned int pos)
Remove image at index `pos` from the image list.
- `ClmgList< T > get_remove` (const unsigned int pos) const
*Remove image at index `pos` from the image list **[new-instance version]**.*
- `ClmgList< T > & remove` ()
Remove last image.
- `ClmgList< T > get_remove` () const
*Remove last image **[new-instance version]**.*
- `ClmgList< T > & reverse` ()
Reverse list order.
- `ClmgList< T > get_reverse` () const
*Reverse list order **[new-instance version]**.*
- `ClmgList< T > & images` (const unsigned int pos0, const unsigned int pos1)
Return a sublist.
- `ClmgList< T > get_images` (const unsigned int pos0, const unsigned int pos1) const
*Return a sublist **[new-instance version]**.*
- `ClmgList< T > get_shared_images` (const unsigned int pos0, const unsigned int pos1)
Return a shared sublist.
- `const ClmgList< T > get_shared_images` (const unsigned int pos0, const unsigned int pos1) const
*Return a shared sublist **[new-instance version]**.*
- `Clmg< T > get_append` (const char axis, const float align=0) const
Return a single image which is the appending of all images of the current `ClmgList` instance.
- `ClmgList< T > & split` (const char axis, const int nb=-1)
Return a list where each image has been split along the specified axis.
- `ClmgList< T > get_split` (const char axis, const int nb=-1) const
*Return a list where each image has been split along the specified axis **[new-instance version]**.*
- `template<typename t >`
`ClmgList< T > & push_back` (const `Clmg< t >` &img)
Insert image at the end of the list.
- `template<typename t >`
`ClmgList< T > & push_front` (const `Clmg< t >` &img)
Insert image at the front of the list.
- `template<typename t >`
`ClmgList< T > & push_back` (const `ClmgList< t >` &list)
Insert list at the end of the current list.
- `template<typename t >`
`ClmgList< T > & push_front` (const `ClmgList< t >` &list)
Insert list at the front of the current list.
- `ClmgList< T > & pop_back` ()
Remove last image.
- `ClmgList< T > & pop_front` ()
Remove first image.
- `ClmgList< T > & erase` (const iterator iter)
Remove image pointed by iterator.

Data Input

- `Clmg< intT > get_select (ClmgDisplay &disp, const bool feature_type=true, const char axis='x', const float align=0, const bool exit_on_anykey=false) const`
Display a simple interactive interface to select images or sublists.
- `Clmg< intT > get_select (const char *const title, const bool feature_type=true, const char axis='x', const float align=0, const bool exit_on_anykey=false) const`
Display a simple interactive interface to select images or sublists.
- `ClmgList< T > & load (const char *const filename)`
Load a list from a file.
- `ClmgList< T > & load_cimg (const char *const filename)`
Load a list from a .cimg file.
- `ClmgList< T > & load_cimg (std::FILE *const file)`
Load a list from a .cimg file.
- `ClmgList< T > & load_cimg (const char *const filename, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1)`
Load a sublist list from a (non compressed) .cimg file.
- `ClmgList< T > & load_cimg (std::FILE *const file, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1)`
Load a sub-image list from a (non compressed) .cimg file [overloading].
- `ClmgList< T > & load_parrec (const char *const filename)`
Load a list from a PAR/REC (Philips) file.
- `ClmgList< T > & load_yuv (const char *const filename, const unsigned int size_x, const unsigned int size_y, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true)`
Load a list from a YUV image sequence file.
- `ClmgList< T > & load_yuv (std::FILE *const file, const unsigned int size_x, const unsigned int size_y, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true)`
Load a list from an image sequence YUV file [overloading].
- `ClmgList< T > & load_video (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1)`
Load an image from a video file, using OpenCV library.
- `ClmgList< T > & load_ffmpeg_external (const char *const filename)`
Load an image from a video file using the external tool 'ffmpeg'.
- `ClmgList< T > & load_gif_external (const char *const filename)`
Load gif file, using ImageMagick or GraphicsMagick's external tools.
- `ClmgList< T > & load_gzip_external (const char *const filename)`
Load a gzipped list, using external tool 'gunzip'.
- `template<typename tf, typename tc>
ClmgList< T > & load_off (const char *const filename, ClmgList< tf > &primitives, ClmgList< tc > &colors)`
Load a 3d object from a .OFF file.
- `ClmgList< T > & load_tiff (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, float *const voxel_size=0, Clmg< charT > *const description=0)`
Load images from a TIFF file.
- `static ClmgList< T > get_load (const char *const filename)`
Load a list from a file [new-instance version].
- `static ClmgList< T > get_load_cimg (const char *const filename)`
Load a list from a .cimg file [new-instance version].
- `static ClmgList< T > get_load_cimg (std::FILE *const file)`

Load a list from a .cimg file [new-instance version].

- static CImgList< T > [get_load_cimg](#) (const char *const filename, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1)

Load a sublist list from a (non compressed) .cimg file [new-instance version].

- static CImgList< T > [get_load_cimg](#) (std::FILE *const file, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1)

Load a sub-image list from a (non compressed) .cimg file [new-instance version].

- static CImgList< T > [get_load_parrec](#) (const char *const filename)

Load a list from a PAR/REC (Philips) file [new-instance version].

- static CImgList< T > [get_load_yuv](#) (const char *const filename, const unsigned int size_x, const unsigned int size_y=1, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true)

Load a list from a YUV image sequence file [new-instance version].

- static CImgList< T > [get_load_yuv](#) (std::FILE *const file, const unsigned int size_x, const unsigned int size_y=1, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool yuv2rgb=true)

Load a list from an image sequence YUV file [new-instance version].

- static CImgList< T > [get_load_video](#) (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1)

Load an image from a video file, using OpenCV library [new-instance version].

- static CImgList< T > [get_load_ffmpeg_external](#) (const char *const filename)

Load an image from a video file using the external tool 'ffmpeg' [new-instance version].

- static CImgList< T > [get_load_gif_external](#) (const char *const filename)

Load gif file, using ImageMagick or GraphicsMagick's external tools [new-instance version].

- static CImgList< T > [get_load_gzip_external](#) (const char *const filename)

Load a gzipped list, using external tool 'gunzip' [new-instance version].

- template<typename tf, typename tc >
static CImgList< T > [get_load_off](#) (const char *const filename, CImgList< tf > &primitives, CImgList< tc > &colors)

Load a 3d object from a .OFF file [new-instance version].

- static CImgList< T > [get_load_tiff](#) (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, float *const voxel_size=0, CImg< charT > *const description=0)

Load a multi-page TIFF file [new-instance version].

Data Output

- const CImgList< T > & [print](#) (const char *const title=0, const bool display_stats=true) const

Print information about the list on the standard output.

- const CImgList< T > & [display](#) (CImgDisplay &disp, const char axis='x', const float align=0) const

Display the current CImgList instance in an existing CImgDisplay window (by reference).

- const CImgList< T > & [display](#) (CImgDisplay &disp, const bool display_info, const char axis='x', const float align=0, unsigned int *const XYZ=0, const bool exit_on_anykey=false) const

Display the current CImgList instance in a new display window.

- const CImgList< T > & [display](#) (const char *const title=0, const bool display_info=true, const char axis='x', const float align=0, unsigned int *const XYZ=0, const bool exit_on_anykey=false) const

Display the current CImgList instance in a new display window.

- const CImgList< T > & [save](#) (const char *const filename, const int number=-1, const unsigned int digits=6) const

Save list into a file.

- `const CImgList< T > & save_gif_external` (const char *const filename, const float fps=25, const unsigned int nb_loops=0)
Save image sequence as a GIF animated file.
- `const CImgList< T > & save_yuv` (const char *const filename=0, const bool is_rgb=true) const
Save list as a YUV image sequence file.
- `const CImgList< T > & save_yuv` (std::FILE *const file, const bool is_rgb=true) const
Save image sequence into a YUV file.
- `const CImgList< T > & save_cimg` (const char *const filename, const bool is_compressed=false) const
Save list into a .cimg file.
- `const CImgList< T > & save_cimg` (std::FILE *file, const bool is_compressed=false) const
Save list into a .cimg file.
- `const CImgList< T > & save_cimg` (const char *const filename, const unsigned int n0, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0) const
Insert the image instance into into an existing .cimg file, at specified coordinates.
- `const CImgList< T > & save_cimg` (std::FILE *const file, const unsigned int n0, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0) const
Insert the image instance into into an existing .cimg file, at specified coordinates.
- `const CImgList< T > & save_tiff` (const char *const filename, const unsigned int compression_type=0, const float *const voxel_size=0, const char *const description=0, const bool use_bigtiff=true) const
Save list as a TIFF file.
- `const CImgList< T > & save_gzip_external` (const char *const filename) const
Save list as a gzipped file, using external tool 'gzip'.
- `const CImgList< T > & save_video` (const char *const filename, const unsigned int fps=25, const char *const codec=0, const bool keep_open=false) const
Save image sequence, using the OpenCV library.
- `const CImgList< T > & save_ffmpeg_external` (const char *const filename, const unsigned int fps=25, const char *const codec=0, const unsigned int bitrate=2048) const
Save image sequence, using the external tool 'ffmpeg'.
- `CImg< ucharT > get_serialize` (const bool is_compressed=false) const
Serialize a CImgList<T> instance into a raw CImg<unsigned char> buffer.
- `static bool is_saveable` (const char *const filename)
Tell if an image list can be saved as one single file.
- `static void save_empty_cimg` (const char *const filename, const unsigned int nb, const unsigned int dx, const unsigned int dy=1, const unsigned int dz=1, const unsigned int dc=1)
Save empty (non-compressed) .cimg file with specified dimensions.
- `static void save_empty_cimg` (std::FILE *const file, const unsigned int nb, const unsigned int dx, const unsigned int dy=1, const unsigned int dz=1, const unsigned int dc=1)
Save empty .cimg file with specified dimensions.
- `template<typename t >`
`static CImgList< T > get_unserialize` (const CImg< t > &buffer)
Unserialize a CImg<unsigned char> serialized buffer into a CImgList<T> list.

Others

- `CImgList< T > & crop_font` ()
Crop font along the X-axis.
- `CImgList< T > get_crop_font` () const
Crop font along the X-axis [new-instance version].
- `CImgList< T > & FFT` (const char axis, const bool invert=false)
Compute a 1d Fast Fourier Transform, along specified axis.
- `CImgList< Tfloat > get_FFT` (const char axis, const bool invert=false) const

- *Compute a 1-D Fast Fourier Transform, along specified axis [new-instance version].*
- `CImgList< T > & FFT` (const bool invert=false)
- *Compute a n-d Fast Fourier Transform.*
- `CImgList< Tfloat > get_FFT` (const bool invert=false) const
- *Compute a n-d Fast Fourier Transform [new-instance version].*
- `CImgList< T > & reverse_object3d` ()
- *Reverse primitives orientations of a 3d object.*
- `CImgList< T > get_reverse_object3d` () const
- *Reverse primitives orientations of a 3d object [new-instance version].*
- static const `CImgList< ucharT > & font` (const unsigned int font_height, const bool is_variable_width=true)
- *Return a CImg pre-defined font with desired size.*

8.4.1 Detailed Description

```
template<typename T>
struct cimg_library::CImgList< T >
```

Represent a list of images CImg<T>.

8.4.2 Member Typedef Documentation

8.4.2.1 typedef CImg<T>* iterator

Simple iterator type, to loop through each image of a list.

Note

- The `CImgList<T>::iterator` type is defined as a `CImg<T>*`.
- You may use it like this:

```
CImgList<> list; // Assuming this image list is not empty.
for (CImgList<>::iterator it = list.begin(); it<list.end(); ++it) (*it).mirror('x');
```
- Using the loop macro `cimglist_for` is another (more concise) alternative:

```
cimglist_for(list,l) list[l].mirror('x');
```

8.4.2.2 typedef const CImg<T>* const_iterator

Simple const iterator type, to loop through each image of a `const` list instance.

Note

- The `CImgList<T>::const_iterator` type is defined to be a `const CImg<T>*`.
- Similar to `CImgList<T>::iterator`, but for constant list instances.

8.4.2.3 typedef T value_type

Pixel value type.

Refer to the pixels value type of the images in the list.

Note

- The `CImgList<T>::value_type` type of a `CImgList<T>` is defined to be a `T`. It is then similar to `CImg<T>::value_type`.
- `CImgList<T>::value_type` is actually not used in `CImg` methods. It has been mainly defined for compatibility with STL naming conventions.

8.4.3 Constructor & Destructor Documentation

8.4.3.1 ~CImgList ()

Destructor.

Destroy current list instance.

Note

- Any allocated buffer is deallocated.
- Destroying an empty list does nothing actually.

8.4.3.2 CImgList ()

Default constructor.

Construct a new empty list instance.

Note

- An empty list has no pixel data and its dimension `width()` is set to 0, as well as its image buffer pointer `data()`.
- An empty list may be reassigned afterwards, with the family of the `assign()` methods. In all cases, the type of pixels stays `T`.

8.4.3.3 CImgList (const unsigned int n) [explicit]

Construct list containing empty images.

Parameters

<i>n</i>	Number of empty images.
----------	-------------------------

Note

Useful when you know by advance the number of images you want to manage, as it will allocate the right amount of memory for the list, without needs for reallocation (that may occur when starting from an empty list and inserting several images in it).

8.4.3.4 CImgList (const unsigned int *n*, const unsigned int *width*, const unsigned int *height* = 1, const unsigned int *depth* = 1, const unsigned int *spectrum* = 1)

Construct list containing images of specified size.

Parameters

<i>n</i>	Number of images.
<i>width</i>	Width of images.
<i>height</i>	Height of images.
<i>depth</i>	Depth of images.
<i>spectrum</i>	Number of channels of images.

Note

Pixel values are not initialized and may probably contain garbage.

8.4.3.5 CImgList (const unsigned int *n*, const unsigned int *width*, const unsigned int *height*, const unsigned int *depth*, const unsigned int *spectrum*, const T & *val*)

Construct list containing images of specified size, and initialize pixel values.

Parameters

<i>n</i>	Number of images.
<i>width</i>	Width of images.
<i>height</i>	Height of images.
<i>depth</i>	Depth of images.
<i>spectrum</i>	Number of channels of images.
<i>val</i>	Initialization value for images pixels.

8.4.3.6 CImgList (const unsigned int *n*, const unsigned int *width*, const unsigned int *height*, const unsigned int *depth*, const unsigned int *spectrum*, const int *val0*, const int *val1*, ...)

Construct list containing images of specified size, and initialize pixel values from a sequence of integers.

Parameters

<i>n</i>	Number of images.
<i>width</i>	Width of images.
<i>height</i>	Height of images.

Parameters

<i>depth</i>	Depth of images.
<i>spectrum</i>	Number of channels of images.
<i>val0</i>	First value of the initializing integers sequence.
<i>val1</i>	Second value of the initializing integers sequence.

Warning

You must specify at least `width*height*depth*spectrum` values in your argument list, or you will probably segfault.

8.4.3.7 `ClmgList` (`const unsigned int n`, `const unsigned int width`, `const unsigned int height`, `const unsigned int depth`, `const unsigned int spectrum`, `const double val0`, `const double val1`, ...)

Construct list containing images of specified size, and initialize pixel values from a sequence of doubles.

Parameters

<i>n</i>	Number of images.
<i>width</i>	Width of images.
<i>height</i>	Height of images.
<i>depth</i>	Depth of images.
<i>spectrum</i>	Number of channels of images.
<i>val0</i>	First value of the initializing doubles sequence.
<i>val1</i>	Second value of the initializing doubles sequence.

Warning

You must specify at least `width*height*depth*spectrum` values in your argument list, or you will probably segfault.

8.4.3.8 `ClmgList` (`const unsigned int n`, `const Clmg< t > &img`, `const bool is_shared = false`)

Construct list containing copies of an input image.

Parameters

<i>n</i>	Number of images.
<i>img</i>	Input image to copy in the constructed list.
<i>is_shared</i>	Tells if the elements of the list are shared or non-shared copies of <code>img</code> .

8.4.3.9 `ClmgList` (`const Clmg< t > &img`, `const bool is_shared = false`) `[explicit]`

Construct list from one image.

Parameters

<i>img</i>	Input image to copy in the constructed list.
<i>is_shared</i>	Tells if the element of the list is a shared or non-shared copy of <i>img</i> .

8.4.3.10 ClmgList (const Clmg< t1 > & *img1*, const Clmg< t2 > & *img2*, const bool *is_shared* = false)

Construct list from two images.

Parameters

<i>img1</i>	First input image to copy in the constructed list.
<i>img2</i>	Second input image to copy in the constructed list.
<i>is_shared</i>	Tells if the elements of the list are shared or non-shared copies of input images.

8.4.3.11 ClmgList (const Clmg< t1 > & *img1*, const Clmg< t2 > & *img2*, const Clmg< t3 > & *img3*, const bool *is_shared* = false)

Construct list from three images.

Parameters

<i>img1</i>	First input image to copy in the constructed list.
<i>img2</i>	Second input image to copy in the constructed list.
<i>img3</i>	Third input image to copy in the constructed list.
<i>is_shared</i>	Tells if the elements of the list are shared or non-shared copies of input images.

8.4.3.12 ClmgList (const Clmg< t1 > & *img1*, const Clmg< t2 > & *img2*, const Clmg< t3 > & *img3*, const Clmg< t4 > & *img4*, const bool *is_shared* = false)

Construct list from four images.

Parameters

<i>img1</i>	First input image to copy in the constructed list.
<i>img2</i>	Second input image to copy in the constructed list.
<i>img3</i>	Third input image to copy in the constructed list.
<i>img4</i>	Fourth input image to copy in the constructed list.
<i>is_shared</i>	Tells if the elements of the list are shared or non-shared copies of input images.

8.4.3.13 ClmgList (const Clmg< t1 > & *img1*, const Clmg< t2 > & *img2*, const Clmg< t3 > & *img3*, const Clmg< t4 > & *img4*, const Clmg< t5 > & *img5*, const bool *is_shared* = false)

Construct list from five images.

Parameters

<i>img1</i>	First input image to copy in the constructed list.
<i>img2</i>	Second input image to copy in the constructed list.
<i>img3</i>	Third input image to copy in the constructed list.
<i>img4</i>	Fourth input image to copy in the constructed list.
<i>img5</i>	Fifth input image to copy in the constructed list.
<i>is_shared</i>	Tells if the elements of the list are shared or non-shared copies of input images.

8.4.3.14 `CImgList (const CImg< t1 > &img1, const CImg< t2 > &img2, const CImg< t3 > &img3, const CImg< t4 > &img4, const CImg< t5 > &img5, const CImg< t6 > &img6, const bool is_shared = false)`

Construct list from six images.

Parameters

<i>img1</i>	First input image to copy in the constructed list.
<i>img2</i>	Second input image to copy in the constructed list.
<i>img3</i>	Third input image to copy in the constructed list.
<i>img4</i>	Fourth input image to copy in the constructed list.
<i>img5</i>	Fifth input image to copy in the constructed list.
<i>img6</i>	Sixth input image to copy in the constructed list.
<i>is_shared</i>	Tells if the elements of the list are shared or non-shared copies of input images.

8.4.3.15 `CImgList (const CImg< t1 > &img1, const CImg< t2 > &img2, const CImg< t3 > &img3, const CImg< t4 > &img4, const CImg< t5 > &img5, const CImg< t6 > &img6, const CImg< t7 > &img7, const bool is_shared = false)`

Construct list from seven images.

Parameters

<i>img1</i>	First input image to copy in the constructed list.
<i>img2</i>	Second input image to copy in the constructed list.
<i>img3</i>	Third input image to copy in the constructed list.
<i>img4</i>	Fourth input image to copy in the constructed list.
<i>img5</i>	Fifth input image to copy in the constructed list.
<i>img6</i>	Sixth input image to copy in the constructed list.
<i>img7</i>	Seventh input image to copy in the constructed list.
<i>is_shared</i>	Tells if the elements of the list are shared or non-shared copies of input images.

8.4.3.16 `CImgList (const CImg< t1 > &img1, const CImg< t2 > &img2, const CImg< t3 > &img3, const CImg< t4 > &img4, const CImg< t5 > &img5, const CImg< t6 > &img6, const CImg< t7 > &img7, const CImg< t8 > &img8, const bool is_shared = false)`

Construct list from eight images.

Parameters

<i>img1</i>	First input image to copy in the constructed list.
<i>img2</i>	Second input image to copy in the constructed list.
<i>img3</i>	Third input image to copy in the constructed list.
<i>img4</i>	Fourth input image to copy in the constructed list.
<i>img5</i>	Fifth input image to copy in the constructed list.
<i>img6</i>	Sixth input image to copy in the constructed list.
<i>img7</i>	Seventh input image to copy in the constructed list.
<i>img8</i>	Eighth input image to copy in the constructed list.
<i>is_shared</i>	Tells if the elements of the list are shared or non-shared copies of input images.

8.4.3.17 CImgList (const CImgList< t > & list)

Construct list copy.

Parameters

<i>list</i>	Input list to copy.
-------------	---------------------

Note

The shared state of each element of the constructed list is kept the same as in *list*.

8.4.3.18 CImgList (const CImgList< t > & list, const bool is_shared)

Construct list copy, and force the shared state of the list elements.

Parameters

<i>list</i>	Input list to copy.
<i>is_shared</i>	Tells if the elements of the list are shared or non-shared copies of input images.

8.4.3.19 CImgList (const char *const filename) [explicit]

Construct list by reading the content of a file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

8.4.3.20 CImgList (const CImgDisplay & disp) [explicit]

Construct list from the content of a display window.

Parameters

<i>disp</i>	Display window to get content from.
-------------	-------------------------------------

Note

Constructed list contains a single image only.

8.4.4 Member Function Documentation

8.4.4.1 `CImgList<T> get_shared ()`

Return a list with elements being shared copies of images in the list instance.

Note

`list2 = list1.get_shared()` is equivalent to `list2.assign(list1,true)`.

8.4.4.2 `CImgList<T>& assign ()`

Destructor **[in-place version]**.

See also

[CImgList\(\)](#).

8.4.4.3 `CImgList<T>& clear ()`

Destructor **[in-place version]**.

Equivalent to [assign\(\)](#).

Note

Only here for compatibility with STL naming conventions.

8.4.4.4 `CImgList<T>& assign (const unsigned int n)`

Construct list containing empty images **[in-place version]**.

See also

[CImgList\(unsigned int\)](#).

8.4.4.5 `CImgList<T>& assign (const unsigned int n, const unsigned int width, const unsigned int height = 1, const unsigned int depth = 1, const unsigned int spectrum = 1)`

Construct list containing images of specified size **[in-place version]**.

See also

[CImgList\(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int\)](#).

8.4.4.6 `CImgList<T>& assign (const unsigned int n, const unsigned int width, const unsigned int height, const unsigned int depth, const unsigned int spectrum, const T & val)`

Construct list containing images of specified size, and initialize pixel values **[in-place version]**.

See also

[CImgList\(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int, const T\)](#).

8.4.4.7 `CImgList<T>& assign (const unsigned int n, const unsigned int width, const unsigned int height, const unsigned int depth, const unsigned int spectrum, const int val0, const int val1, ...)`

Construct list with images of specified size, and initialize pixel values from a sequence of integers **[in-place version]**.

See also

[CImgList\(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int, const int, const int, ...\)](#).

8.4.4.8 `CImgList<T>& assign (const unsigned int n, const unsigned int width, const unsigned int height, const unsigned int depth, const unsigned int spectrum, const double val0, const double val1, ...)`

Construct list with images of specified size, and initialize pixel values from a sequence of doubles **[in-place version]**.

See also

[CImgList\(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int, const double, const double, ...\)](#).

8.4.4.9 `CImgList<T>& assign (const unsigned int n, const CImg< t > & img, const bool is_shared = false)`

Construct list containing copies of an input image **[in-place version]**.

See also

[CImgList\(unsigned int, const CImg<t>&, bool\)](#).

8.4.4.10 `ClmgList<T>& assign (const Clmg< t > & img, const bool is_shared = false)`

Construct list from one image **[in-place version]**.

See also

[ClmgList\(const Clmg<t>&, bool\).](#)

8.4.4.11 `ClmgList<T>& assign (const Clmg< t1 > & img1, const Clmg< t2 > & img2, const bool is_shared = false)`

Construct list from two images **[in-place version]**.

See also

[ClmgList\(const Clmg<t>&, const Clmg<t>&, bool\).](#)

8.4.4.12 `ClmgList<T>& assign (const Clmg< t1 > & img1, const Clmg< t2 > & img2, const Clmg< t3 > & img3, const bool is_shared = false)`

Construct list from three images **[in-place version]**.

See also

[ClmgList\(const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, bool\).](#)

8.4.4.13 `ClmgList<T>& assign (const Clmg< t1 > & img1, const Clmg< t2 > & img2, const Clmg< t3 > & img3, const Clmg< t4 > & img4, const bool is_shared = false)`

Construct list from four images **[in-place version]**.

See also

[ClmgList\(const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, bool\).](#)

8.4.4.14 `ClmgList<T>& assign (const Clmg< t1 > & img1, const Clmg< t2 > & img2, const Clmg< t3 > & img3, const Clmg< t4 > & img4, const Clmg< t5 > & img5, const bool is_shared = false)`

Construct list from five images **[in-place version]**.

See also

[ClmgList\(const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, bool\).](#)

8.4.4.15 **ClmgList<T>& assign (const Clmg< t1 > & *img1*, const Clmg< t2 > & *img2*, const Clmg< t3 > & *img3*, const Clmg< t4 > & *img4*, const Clmg< t5 > & *img5*, const Clmg< t6 > & *img6*, const bool *is_shared* = false)**

Construct list from six images **[in-place version]**.

See also

ClmgList(const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, bool).

8.4.4.16 **ClmgList<T>& assign (const Clmg< t1 > & *img1*, const Clmg< t2 > & *img2*, const Clmg< t3 > & *img3*, const Clmg< t4 > & *img4*, const Clmg< t5 > & *img5*, const Clmg< t6 > & *img6*, const Clmg< t7 > & *img7*, const bool *is_shared* = false)**

Construct list from seven images **[in-place version]**.

See also

ClmgList(const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, bool).

8.4.4.17 **ClmgList<T>& assign (const Clmg< t1 > & *img1*, const Clmg< t2 > & *img2*, const Clmg< t3 > & *img3*, const Clmg< t4 > & *img4*, const Clmg< t5 > & *img5*, const Clmg< t6 > & *img6*, const Clmg< t7 > & *img7*, const Clmg< t8 > & *img8*, const bool *is_shared* = false)**

Construct list from eight images **[in-place version]**.

See also

ClmgList(const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, const Clmg<t>&, bool).

8.4.4.18 **ClmgList<T>& assign (const ClmgList< t > & *list*, const bool *is_shared* = false)**

Construct list as a copy of an existing list and force the shared state of the list elements **[in-place version]**.

See also

[ClmgList\(const ClmgList<t>&, bool *is_shared*\).](#)

8.4.4.19 **ClmgList<T>& assign (const char *const *filename*)**

Construct list by reading the content of a file **[in-place version]**.

See also

[ClmgList\(const char *const\).](#)

8.4.4.20 `CImgList<T>& assign (const CImgDisplay & disp)`

Construct list from the content of a display window **[in-place version]**.

See also

[CImgList\(const CImgDisplay&\).](#)

8.4.4.21 `CImgList<t>& move_to (CImgList< t > & list)`

Transfer the content of the list instance to another list.

Parameters

<i>list</i>	Destination list.
-------------	-------------------

Note

When returning, the current list instance is empty and the initial content of `list` is destroyed.

8.4.4.22 `CImgList<t>& move_to (CImgList< t > & list, const unsigned int pos)`

Transfer the content of the list instance at a specified position in another list.

Parameters

<i>list</i>	Destination list.
<i>pos</i>	Index of the insertion in the list.

Note

When returning, the list instance is empty and the initial content of `list` is preserved (only images indexes may be modified).

8.4.4.23 `CImgList<T>& swap (CImgList< T > & list)`

Swap all fields between two list instances.

Parameters

<i>list</i>	List to swap fields with.
-------------	---------------------------

Note

Can be used to exchange the content of two lists in a fast way.

8.4.4.24 static CImgList<T>& empty () [static]

Return a reference to an empty list.

Note

Can be used to define default values in a function taking a CImgList<T> as an argument.

```
void f(const CImgList<char>& list=CImgList<char>::empty());
```

8.4.4.25 CImg<T>& operator() (const unsigned int *pos*)

Return a reference to one image element of the list.

Parameters

<i>pos</i>	Indice of the image element.
------------	------------------------------

8.4.4.26 const CImg<T>& operator() (const unsigned int *pos*) const

Return a reference to one image of the list.

Parameters

<i>pos</i>	Indice of the image element.
------------	------------------------------

8.4.4.27 T& operator() (const unsigned int *pos*, const unsigned int *x*, const unsigned int *y* = 0, const unsigned int *z* = 0, const unsigned int *c* = 0)

Return a reference to one pixel value of one image of the list.

Parameters

<i>pos</i>	Indice of the image element.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

`list(n, x, y, z, c)` is equivalent to `list[n](x, y, z, c)`.

8.4.4.28 operator CImg< T > * ()

Return pointer to the first image of the list.

Note

Images in a list are stored as a buffer of `CImg<T>`.

8.4.4.29 `CImgList<T>& operator= (const CImg< t > & img)`

Construct list from one image **[in-place version]**.

Parameters

<i>img</i>	Input image to copy in the constructed list.
------------	--

Note

`list = img;` is equivalent to `list.assign(img);`.

8.4.4.30 `CImgList<T>& operator= (const CImgList< t > & list)`

Construct list from another list.

Parameters

<i>list</i>	Input list to copy.
-------------	---------------------

Note

`list1 = list2` is equivalent to `list1.assign(list2);`.

8.4.4.31 `CImgList<T>& operator= (const char *const filename)`

Construct list by reading the content of a file **[in-place version]**.

See also

[CImgList\(const char *const\).](#)

8.4.4.32 `CImgList<T>& operator= (const CImgDisplay & disp)`

Construct list from the content of a display window **[in-place version]**.

See also

[CImgList\(const CImgDisplay&\).](#)

8.4.4.33 CImgList<T> operator+ () const

Return a non-shared copy of a list.

Note

`+list` is equivalent to `CImgList<T>(list, false)`. It forces the copy to have non-shared elements.

8.4.4.34 CImgList<T>& operator, (const CImg< t > &img)

Return a copy of the list instance, where image `img` has been inserted at the end.

Parameters

<i>img</i>	Image inserted at the end of the instance copy.
------------	---

Note

Define a convenient way to create temporary lists of images, as in the following code:

```
(img1,img2,img3,img4).display("My four images");
```

8.4.4.35 CImgList<T>& operator, (const CImgList< t > &list)

Return a copy of the list instance, where all elements of input list `list` have been inserted at the end.

Parameters

<i>list</i>	List inserted at the end of the instance copy.
-------------	--

8.4.4.36 CImg<T> operator> (const char axis) const

Return image corresponding to the appending of all images of the instance list along specified axis.

Parameters

<i>axis</i>	Appending axis. Can be { 'x' 'y' 'z' 'c' }.
-------------	---

Note

`list>'x'` is equivalent to `list.get_append('x')`.

8.4.4.37 CImgList<T> operator< (const char axis) const

Return list corresponding to the splitting of all images of the instance list along specified axis.

Parameters

<i>axis</i>	Axis used for image splitting.
-------------	--------------------------------

Note

`list<'x'>` is equivalent to `list.get_split('x')`.

8.4.4.38 static const char* pixel_type () [static]

Return the type of image pixel values as a C string.

Return a `char*` string containing the usual type name of the image pixel values (i.e. a stringified version of the template parameter `T`).

Note

- The returned string may contain spaces (as in `"unsigned char"`).
- If the pixel type `T` does not correspond to a registered type, the string `"unknown"` is returned.

8.4.4.39 int width () const

Return the size of the list, i.e. the number of images contained in it.

Note

Similar to [size\(\)](#) but returns result as a (signed) integer.

8.4.4.40 unsigned int size () const

Return the size of the list, i.e. the number of images contained in it.

Note

Similar to [width\(\)](#) but returns result as an unsigned integer.

8.4.4.41 CImg<T>* data ()

Return pointer to the first image of the list.

Note

Images in a list are stored as a buffer of `CImg<T>`.

8.4.4.42 CImg<T>* data (const unsigned int pos)

Return pointer to the pos-th image of the list.

Parameters

<i>pos</i>	Indice of the image element to access.
------------	--

Note

`list.data(n) ;` is equivalent to `list.data + n ;`.

8.4.4.43 Clmg<T>& at (const int *pos*)

Return pos-th image of the list.

Parameters

<i>pos</i>	Indice of the image element to access.
------------	--

8.4.4.44 T& atNXYZC (const int *pos*, const int *x*, const int *y*, const int *z*, const int *c*, const T & *out_value*)

Access to pixel value with Dirichlet boundary conditions.

Parameters

<i>pos</i>	Indice of the image element to access.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.
<i>out_value</i>	Default value returned if <i>offset</i> is outside image bounds.

Note

`list.atNXYZC(p,x,y,z,c) ;` is equivalent to `list[p].atXYZC(x,y,z,c) ;`.

8.4.4.45 T& atNXYZC (const int *pos*, const int *x*, const int *y*, const int *z*, const int *c*)

Access to pixel value with Neumann boundary conditions.

Parameters

<i>pos</i>	Indice of the image element to access.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

`list.atNXYZC(p, x, y, z, c) ;` is equivalent to `list[p].atXYZC(x, y, z, c) ;`.

8.4.4.46 T& atNXYZ (const int *pos*, const int *x*, const int *y*, const int *z*, const int *c*, const T & *out_value*)

Access pixel value with Dirichlet boundary conditions for the 3 first coordinates (*pos*, *x*,*y*,*z*).

Parameters

<i>pos</i>	Indice of the image element to access.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.
<i>out_value</i>	Default value returned if <i>offset</i> is outside image bounds.

Note

`list.atNXYZ(p, x, y, z, c) ;` is equivalent to `list[p].atXYZ(x, y, z, c) ;`.

8.4.4.47 T& atNXYZ (const int *pos*, const int *x*, const int *y*, const int *z*, const int *c* = 0)

Access to pixel value with Neumann boundary conditions for the 4 first coordinates (*pos*, *x*,*y*,*z*).

Parameters

<i>pos</i>	Indice of the image element to access.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

`list.atNXYZ(p, x, y, z, c) ;` is equivalent to `list[p].atXYZ(x, y, z, c) ;`.

8.4.4.48 T& atNXY (const int *pos*, const int *x*, const int *y*, const int *z*, const int *c*, const T & *out_value*)

Access to pixel value with Dirichlet boundary conditions for the 3 first coordinates (*pos*, *x*,*y*).

Parameters

<i>pos</i>	Indice of the image element to access.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.

Parameters

<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.
<i>out_value</i>	Default value returned if <i>offset</i> is outside image bounds.

Note

`list.atNXYZ(p, x, y, z, c) ;` is equivalent to `list[p].atXYZ(x, y, z, c) ;`.

8.4.4.49 T& atNXY (const int *pos*, const int *x*, const int *y*, const int *z* = 0, const int *c* = 0)

Access to pixel value with Neumann boundary conditions for the 3 first coordinates (*pos*, *x*, *y*).

Parameters

<i>pos</i>	Indice of the image element to access.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

`list.atNXYZ(p, x, y, z, c) ;` is equivalent to `list[p].atXYZ(x, y, z, c) ;`.

8.4.4.50 T& atNX (const int *pos*, const int *x*, const int *y*, const int *z*, const int *c*, const T & *out_value*)

Access to pixel value with Dirichlet boundary conditions for the 2 first coordinates (*pos*, *x*).

Parameters

<i>pos</i>	Indice of the image element to access.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.
<i>out_value</i>	Default value returned if <i>offset</i> is outside image bounds.

Note

`list.atNXYZ(p, x, y, z, c) ;` is equivalent to `list[p].atXYZ(x, y, z, c) ;`.

8.4.4.51 T& atNX (const int *pos*, const int *x*, const int *y* = 0, const int *z* = 0, const int *c* = 0)

Access to pixel value with Neumann boundary conditions for the 2 first coordinates (*pos*, *x*).

Parameters

<i>pos</i>	Indice of the image element to access.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

`list.atNXYZ(p, x, y, z, c) ;` is equivalent to `list[p].atXYZ(x, y, z, c) ;`.

8.4.4.52 `T& atN (const int pos, const int x, const int y, const int z, const int c, const T & out_value)`

Access to pixel value with Dirichlet boundary conditions for the first coordinate (`pos`).

Parameters

<i>pos</i>	Indice of the image element to access.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.
<i>out_value</i>	Default value returned if <code>offset</code> is outside image bounds.

Note

`list.atNXYZ(p, x, y, z, c) ;` is equivalent to `list[p].atXYZ(x, y, z, c) ;`.

8.4.4.53 `T& atN (const int pos, const int x = 0, const int y = 0, const int z = 0, const int c = 0)`

Return pixel value with Neumann boundary conditions for the first coordinate (`pos`).

Parameters

<i>pos</i>	Indice of the image element to access.
<i>x</i>	X-coordinate of the pixel value.
<i>y</i>	Y-coordinate of the pixel value.
<i>z</i>	Z-coordinate of the pixel value.
<i>c</i>	C-coordinate of the pixel value.

Note

`list.atNXYZ(p, x, y, z, c) ;` is equivalent to `list[p].atXYZ(x, y, z, c) ;`.

8.4.4.54 CImg<charT> value_string (const char *separator* = ' , ' , const unsigned int *max_size* = 0) const

Return a C-string containing the values of all images in the instance list.

Parameters

<i>separator</i>	Character separator set between consecutive pixel values.
<i>max_size</i>	Maximum size of the returned string.

Note

The result is returned as a CImg<char> image whose pixel buffer contains the desired C-string.

8.4.4.55 bool is_sameN (const unsigned int *size_n*) const

Test if number of image elements is equal to specified value.

Parameters

<i>size_n</i>	Number of image elements to test.
---------------	-----------------------------------

8.4.4.56 bool is_sameN (const CImgList< t > & *list*) const

Test if number of image elements is equal between two images lists.

Parameters

<i>list</i>	Input list to compare with.
-------------	-----------------------------

8.4.4.57 bool is_sameXYZC (const unsigned int *dx*, const unsigned int *dy*, const unsigned int *dz*, const unsigned int *dc*) const

Test if dimensions of each image of the list match specified arguments.

Parameters

<i>dx</i>	Checked image width.
<i>dy</i>	Checked image height.
<i>dz</i>	Checked image depth.
<i>dc</i>	Checked image spectrum.

8.4.4.58 `bool is_sameNXYZC (const unsigned int n, const unsigned int dx, const unsigned int dy, const unsigned int dz, const unsigned int dc) const`

Test if list dimensions match specified arguments.

Parameters

<i>n</i>	Number of images in the list.
<i>dx</i>	Checked image width.
<i>dy</i>	Checked image height.
<i>dz</i>	Checked image depth.
<i>dc</i>	Checked image spectrum.

8.4.4.59 `bool containsNXYZC (const int n, const int x = 0, const int y = 0, const int z = 0, const int c = 0) const`

Test if list contains one particular pixel location.

Parameters

<i>n</i>	Index of the image whom checked pixel value belong to.
<i>x</i>	X-coordinate of the checked pixel value.
<i>y</i>	Y-coordinate of the checked pixel value.
<i>z</i>	Z-coordinate of the checked pixel value.
<i>c</i>	C-coordinate of the checked pixel value.

8.4.4.60 `bool containsN (const int n) const`

Test if list contains image with specified indice.

Parameters

<i>n</i>	Index of the checked image.
----------	-----------------------------

8.4.4.61 `bool contains (const T & pixel, t & n, t & x, t & y, t & z, t & c) const`

Test if one image of the list contains the specified referenced value.

Parameters

	<i>pixel</i>	Reference to pixel value to test.
out	<i>n</i>	Index of image containing the pixel value, if test succeeds.
out	<i>x</i>	X-coordinate of the pixel value, if test succeeds.
out	<i>y</i>	Y-coordinate of the pixel value, if test succeeds.
out	<i>z</i>	Z-coordinate of the pixel value, if test succeeds.
out	<i>c</i>	C-coordinate of the pixel value, if test succeeds.

Note

If true, set coordinates (n,x,y,z,c).

8.4.4.62 bool contains (const T & *pixel*, t & *n*, t & *x*, t & *y*, t & *z*) const

Test if one of the image list contains the specified referenced value.

Parameters

	<i>pixel</i>	Reference to pixel value to test.
out	<i>n</i>	Index of image containing the pixel value, if test succeeds.
out	<i>x</i>	X-coordinate of the pixel value, if test succeeds.
out	<i>y</i>	Y-coordinate of the pixel value, if test succeeds.
out	<i>z</i>	Z-coordinate of the pixel value, if test succeeds.

Note

If true, set coordinates (n,x,y,z).

8.4.4.63 bool contains (const T & *pixel*, t & *n*, t & *x*, t & *y*) const

Test if one of the image list contains the specified referenced value.

Parameters

	<i>pixel</i>	Reference to pixel value to test.
out	<i>n</i>	Index of image containing the pixel value, if test succeeds.
out	<i>x</i>	X-coordinate of the pixel value, if test succeeds.
out	<i>y</i>	Y-coordinate of the pixel value, if test succeeds.

Note

If true, set coordinates (n,x,y).

8.4.4.64 bool contains (const T & *pixel*, t & *n*, t & *x*) const

Test if one of the image list contains the specified referenced value.

Parameters

	<i>pixel</i>	Reference to pixel value to test.
out	<i>n</i>	Index of image containing the pixel value, if test succeeds.
out	<i>x</i>	X-coordinate of the pixel value, if test succeeds.

Note

If true, set coordinates (n,x).

8.4.4.65 bool contains (const T & *pixel*, t & n) const

Test if one of the image list contains the specified referenced value.

Parameters

	<i>pixel</i>	Reference to pixel value to test.
out	<i>n</i>	Index of image containing the pixel value, if test succeeds.

Note

If true, set coordinates (n).

8.4.4.66 bool contains (const T & *pixel*) const

Test if one of the image list contains the specified referenced value.

Parameters

<i>pixel</i>	Reference to pixel value to test.
--------------	-----------------------------------

8.4.4.67 bool contains (const CImg< T > & *img*, t & n) const

Test if the list contains the image 'img'.

Parameters

	<i>img</i>	Reference to image to test.
out	<i>n</i>	Index of image in the list, if test succeeds.

Note

If true, returns the position (n) of the image in the list.

8.4.4.68 bool contains (const CImg< T > & *img*) const

Test if the list contains the image img.

Parameters

<i>img</i>	Reference to image to test.
------------	-----------------------------

8.4.4.69 T& min_max (t & max_val)

Return a reference to the minimum pixel value of the instance list and return the maximum vvalue as well.

Parameters

out	max_val	Value of the maximum value found.
-----	---------	-----------------------------------

8.4.4.70 const T& min_max (t & max_val) const

Return a reference to the minimum pixel value of the instance list and return the maximum vvalue as well [**const version**].

Parameters

out	max_val	Value of the maximum value found.
-----	---------	-----------------------------------

8.4.4.71 T& max_min (t & min_val)

Return a reference to the minimum pixel value of the instance list and return the minimum value as well.

Parameters

out	min_val	Value of the minimum value found.
-----	---------	-----------------------------------

8.4.4.72 CImgList<T>& insert (const CImg< t > &img, const unsigned int pos = ~0U, const bool is_shared = false)

Insert a copy of the image *img* into the current image list, at position *pos*.

Parameters

<i>img</i>	Image to insert a copy to the list.
<i>pos</i>	Index of the insertion.
<i>is_shared</i>	Tells if the inserted image is a shared copy of <i>img</i> or not.

8.4.4.73 CImgList<T>& insert (const unsigned int n, const unsigned int pos = ~0U)

Insert *n* empty images *img* into the current image list, at position *pos*.

Parameters

<i>n</i>	Number of empty images to insert.
<i>pos</i>	Index of the insertion.

8.4.4.74 `CImgList<T>& insert (const unsigned int n, const CImg< t > & img, const unsigned int pos = ~0U, const bool is_shared = false)`

Insert *n* copies of the image *img* into the current image list, at position *pos*.

Parameters

<i>n</i>	Number of image copies to insert.
<i>img</i>	Image to insert by copy.
<i>pos</i>	Index of the insertion.
<i>is_shared</i>	Tells if inserted images are shared copies of <i>img</i> or not.

8.4.4.75 `CImgList<T>& insert (const CImgList< t > & list, const unsigned int pos = ~0U, const bool is_shared = false)`

Insert a copy of the image list *list* into the current image list, starting from position *pos*.

Parameters

<i>list</i>	Image list to insert.
<i>pos</i>	Index of the insertion.
<i>is_shared</i>	Tells if inserted images are shared copies of images of <i>list</i> or not.

8.4.4.76 `CImgList<T>& insert (const unsigned int n, const CImgList< t > & list, const unsigned int pos = ~0U, const bool is_shared = false)`

Insert *n* copies of the list *list* at position *pos* of the current list.

Parameters

<i>n</i>	Number of list copies to insert.
<i>list</i>	Image list to insert.
<i>pos</i>	Index of the insertion.
<i>is_shared</i>	Tells if inserted images are shared copies of images of <i>list</i> or not.

8.4.4.77 `CImgList<T>& remove (const unsigned int pos1, const unsigned int pos2)`

Remove all images between from indexes.

Parameters

<i>pos1</i>	Starting index of the removal.
<i>pos2</i>	Ending index of the removal.

8.4.4.78 CImgList<T>& remove (const unsigned int *pos*)

Remove image at index *pos* from the image list.

Parameters

<i>pos</i>	Index of the image to remove.
------------	-------------------------------

8.4.4.79 CImgList<T>& images (const unsigned int *pos0*, const unsigned int *pos1*)

Return a sublist.

Parameters

<i>pos0</i>	Starting index of the sublist.
<i>pos1</i>	Ending index of the sublist.

8.4.4.80 CImgList<T> get_shared_images (const unsigned int *pos0*, const unsigned int *pos1*)

Return a shared sublist.

Parameters

<i>pos0</i>	Starting index of the sublist.
<i>pos1</i>	Ending index of the sublist.

8.4.4.81 CImg<T> get_append (const char *axis*, const float *align* = 0) const

Return a single image which is the appending of all images of the current [CImgList](#) instance.

Parameters

<i>axis</i>	Appending axis. Can be { 'x' 'y' 'z' 'c' }.
<i>align</i>	Appending alignment.

8.4.4.82 CImgList<T>& split (const char *axis*, const int *nb* = -1)

Return a list where each image has been split along the specified axis.

Parameters

<i>axis</i>	Axis to split images along.
<i>nb</i>	Number of splitting parts for each image.

8.4.4.83 CImgList<T>& push_back (const CImg< t > & *img*)

Insert image at the end of the list.

Parameters

<i>img</i>	Image to insert.
------------	------------------

8.4.4.84 CImgList<T>& push_front (const CImg< t > & *img*)

Insert image at the front of the list.

Parameters

<i>img</i>	Image to insert.
------------	------------------

8.4.4.85 CImgList<T>& push_back (const CImgList< t > & *list*)

Insert list at the end of the current list.

Parameters

<i>list</i>	List to insert.
-------------	-----------------

8.4.4.86 CImgList<T>& push_front (const CImgList< t > & *list*)

Insert list at the front of the current list.

Parameters

<i>list</i>	List to insert.
-------------	-----------------

8.4.4.87 CImgList<T>& erase (const iterator *iter*)

Remove image pointed by iterator.

Parameters

<i>iter</i>	Iterator pointing to the image to remove.
-------------	---

8.4.4.88 CImg<intT> get_select (CImgDisplay & disp, const bool *feature_type* = true, const char *axis* = 'x', const float *align* = 0, const bool *exit_on_anykey* = false) const

Display a simple interactive interface to select images or sublists.

Parameters

<i>disp</i>	Window instance to display selection and user interface.
<i>feature_type</i>	Can be <i>false</i> to select a single image, or <i>true</i> to select a sublist.
<i>axis</i>	Axis along whom images are appended for visualization.
<i>align</i>	Alignment setting when images have not all the same size.

Returns

A one-column vector containing the selected image indexes.

8.4.4.89 CImg<intT> get_select (const char *const *title*, const bool *feature_type* = true, const char *axis* = 'x', const float *align* = 0, const bool *exit_on_anykey* = false) const

Display a simple interactive interface to select images or sublists.

Parameters

<i>title</i>	Title of a new window used to display selection and user interface.
<i>feature_type</i>	Can be <i>false</i> to select a single image, or <i>true</i> to select a sublist.
<i>axis</i>	Axis along whom images are appended for visualization.
<i>align</i>	Alignment setting when images have not all the same size.

Returns

A one-column vector containing the selected image indexes.

8.4.4.90 CImgList<T>& load (const char *const *filename*)

Load a list from a file.

Parameters

<i>filename</i>	Filename to read data from.
-----------------	-----------------------------

8.4.4.91 CImgList<T>& load_cimg (const char *const *filename*)

Load a list from a .cimg file.

Parameters

<i>filename</i>	Filename to read data from.
-----------------	-----------------------------

8.4.4.92 CImgList<T>& load_cimg (std::FILE *const file)

Load a list from a .cimg file.

Parameters

<i>file</i>	File to read data from.
-------------	-------------------------

8.4.4.93 CImgList<T>& load_cimg (const char *const filename, const unsigned int n0, const unsigned int n1, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0, const unsigned int x1, const unsigned int y1, const unsigned int z1, const unsigned int c1)

Load a sublist list from a (non compressed) .cimg file.

Parameters

<i>filename</i>	Filename to read data from.
<i>n0</i>	Starting index of images to read (~0U for max).
<i>n1</i>	Ending index of images to read (~0U for max).
<i>x0</i>	Starting X-coordinates of image regions to read.
<i>y0</i>	Starting Y-coordinates of image regions to read.
<i>z0</i>	Starting Z-coordinates of image regions to read.
<i>c0</i>	Starting C-coordinates of image regions to read.
<i>x1</i>	Ending X-coordinates of image regions to read (~0U for max).
<i>y1</i>	Ending Y-coordinates of image regions to read (~0U for max).
<i>z1</i>	Ending Z-coordinates of image regions to read (~0U for max).
<i>c1</i>	Ending C-coordinates of image regions to read (~0U for max).

8.4.4.94 CImgList<T>& load_parrec (const char *const filename)

Load a list from a PAR/REC (Philips) file.

Parameters

<i>filename</i>	Filename to read data from.
-----------------	-----------------------------

8.4.4.95 CImgList<T>& load_yuv (const char *const filename, const unsigned int size_x, const unsigned int size_y, const unsigned int first_frame = 0, const unsigned int last_frame = ~0U, const unsigned int step_frame = 1, const bool yuv2rgb = true)

Load a list from a YUV image sequence file.

Parameters

<i>filename</i>	Filename to read data from.
<i>size_x</i>	Width of the images.
<i>size_y</i>	Height of the images.
<i>first_frame</i>	Index of first image frame to read.
<i>last_frame</i>	Index of last image frame to read.
<i>step_frame</i>	Step applied between each frame.
<i>yuv2rgb</i>	Apply YUV to RGB transformation during reading.

8.4.4.96 CImgList<T>& load_video (const char *const *filename*, const unsigned int *first_frame* = 0, const unsigned int *last_frame* = ~0U, const unsigned int *step_frame* = 1)

Load an image from a video file, using OpenCV library.

Parameters

<i>filename</i>	Filename, as a C-string.
<i>first_frame</i>	Index of the first frame to read.
<i>last_frame</i>	Index of the last frame to read.
<i>step_frame</i>	Step value for frame reading.

Note

If *step_frame*==0, the current video stream is forced to be released (without any frames read).

8.4.4.97 CImgList<T>& load_ffmpeg_external (const char *const *filename*)

Load an image from a video file using the external tool 'ffmpeg'.

Parameters

<i>filename</i>	Filename to read data from.
-----------------	-----------------------------

8.4.4.98 CImgList<T>& load_gif_external (const char *const *filename*)

Load gif file, using ImageMagick or GraphicsMagick's external tools.

Parameters

<i>filename</i>	Filename to read data from.
<i>use_graphicsmagick</i>	Tells if GraphicsMagick's tool 'gm' is used instead of ImageMagick's tool 'convert'.

8.4.4.99 CImgList<T>& load_gzip_external (const char *const filename)

Load a gzipped list, using external tool 'gunzip'.

Parameters

<i>filename</i>	Filename to read data from.
-----------------	-----------------------------

8.4.4.100 CImgList<T>& load_off (const char *const filename, CImgList<tf> & primitives, CImgList<tc> & colors)

Load a 3d object from a .OFF file.

Parameters

	<i>filename</i>	Filename to read data from.
out	<i>primitives</i>	At return, contains the list of 3d object primitives.
out	<i>colors</i>	At return, contains the list of 3d object colors.

Returns

List of 3d object vertices.

8.4.4.101 CImgList<T>& load_tiff (const char *const filename, const unsigned int first_frame = 0, const unsigned int last_frame = ~0U, const unsigned int step_frame = 1, float *const voxel_size = 0, CImg<charT> *const description = 0)

Load images from a TIFF file.

Parameters

<i>filename</i>	Filename to read data from.
<i>first_frame</i>	Index of first image frame to read.
<i>last_frame</i>	Index of last image frame to read.
<i>step_frame</i>	Step applied between each frame.

8.4.4.102 const CImgList<T>& print (const char *const title = 0, const bool display_stats = true) const

Print information about the list on the standard output.

Parameters

<i>title</i>	Label set to the information displayed.
<i>display_stats</i>	Tells if image statistics must be computed and displayed.

8.4.4.103 `const CImgList<T>& display (CImgDisplay & disp, const char axis = 'x', const float align = 0) const`

Display the current [CImgList](#) instance in an existing [CImgDisplay](#) window (by reference).

Parameters

<i>disp</i>	Reference to an existing CImgDisplay instance, where the current image list will be displayed.
<i>axis</i>	Appending axis. Can be { 'x' 'y' 'z' 'c' }.
<i>align</i>	Appending alignment.

Note

This function displays the list images of the current [CImgList](#) instance into an existing [CImgDisplay](#) window. Images of the list are appended in a single temporarily image for visualization purposes. The function returns immediately.

8.4.4.104 `const CImgList<T>& display (CImgDisplay & disp, const bool display_info, const char axis = 'x', const float align = 0, unsigned int *const XYZ = 0, const bool exit_on_anykey = false) const`

Display the current [CImgList](#) instance in a new display window.

Parameters

<i>disp</i>	Display window.
<i>display_info</i>	Tells if image information are displayed on the standard output.
<i>axis</i>	Alignment axis for images viewing.
<i>align</i>	Appending alignment.

Note

This function opens a new window with a specific title and displays the list images of the current [CImgList](#) instance into it. Images of the list are appended in a single temporarily image for visualization purposes. The function returns when a key is pressed or the display window is closed by the user.

8.4.4.105 `const CImgList<T>& display (const char *const title = 0, const bool display_info = true, const char axis = 'x', const float align = 0, unsigned int *const XYZ = 0, const bool exit_on_anykey = false) const`

Display the current [CImgList](#) instance in a new display window.

Parameters

<i>title</i>	Title of the opening display window.
<i>display_info</i>	Tells if list information must be written on standard output.
<i>axis</i>	Appending axis. Can be { 'x' 'y' 'z' 'c' }.
<i>align</i>	Appending alignment.

8.4.4.106 `const CImgList<T>& save (const char *const filename, const int number = -1, const unsigned int digits = 6) const`

Save list into a file.

Parameters

<i>filename</i>	Filename to write data to.
<i>number</i>	When positive, represents an index added to the filename. Otherwise, no number is added.
<i>digits</i>	Number of digits used for adding the number to the filename.

8.4.4.107 `static bool is_saveable (const char *const filename) [static]`

Tell if an image list can be saved as one single file.

Parameters

<i>filename</i>	Filename, as a C-string.
-----------------	--------------------------

Returns

`true` if the file format supports multiple images, `false` otherwise.

8.4.4.108 `const CImgList<T>& save_gif_external (const char *const filename, const float fps = 25, const unsigned int nb_loops = 0)`

Save image sequence as a GIF animated file.

Parameters

<i>filename</i>	Filename to write data to.
<i>fps</i>	Number of desired frames per second.
<i>nb_loops</i>	Number of loops (0 for infinite looping).

8.4.4.109 `const CImgList<T>& save_yuv (const char *const filename = 0, const bool is_rgb = true) const`

Save list as a YUV image sequence file.

Parameters

<i>filename</i>	Filename to write data to.
<i>is_rgb</i>	Tells if the RGB to YUV conversion must be done for saving.

8.4.4.110 `const CImgList<T>& save_yuv (std::FILE *const file, const bool is_rgb = true) const`

Save image sequence into a YUV file.

Parameters

<i>file</i>	File to write data to.
<i>is_rgb</i>	Tells if the RGB to YUV conversion must be done for saving.

8.4.4.111 `const CImgList<T>& save_cimg (const char *const filename, const bool is_compressed = false) const`

Save list into a .cimg file.

Parameters

<i>filename</i>	Filename to write data to.
<i>is_compressed</i>	Tells if data compression must be enabled.

8.4.4.112 `const CImgList<T>& save_cimg (std::FILE * file, const bool is_compressed = false) const`

Save list into a .cimg file.

Parameters

<i>file</i>	File to write data to.
<i>is_compressed</i>	Tells if data compression must be enabled.

8.4.4.113 `const CImgList<T>& save_cimg (const char *const filename, const unsigned int n0, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0) const`

Insert the image instance into into an existing .cimg file, at specified coordinates.

Parameters

<i>filename</i>	Filename to write data to.
<i>n0</i>	Starting index of images to write.
<i>x0</i>	Starting X-coordinates of image regions to write.
<i>y0</i>	Starting Y-coordinates of image regions to write.
<i>z0</i>	Starting Z-coordinates of image regions to write.
<i>c0</i>	Starting C-coordinates of image regions to write.

8.4.4.114 `const CImgList<T>& save_cimg (std::FILE *const file, const unsigned int n0, const unsigned int x0, const unsigned int y0, const unsigned int z0, const unsigned int c0) const`

Insert the image instance into into an existing .cimg file, at specified coordinates.

Parameters

<i>file</i>	File to write data to.
<i>n0</i>	Starting index of images to write.
<i>x0</i>	Starting X-coordinates of image regions to write.
<i>y0</i>	Starting Y-coordinates of image regions to write.
<i>z0</i>	Starting Z-coordinates of image regions to write.
<i>c0</i>	Starting C-coordinates of image regions to write.

8.4.4.115 `static void save_empty_cimg (const char *const filename, const unsigned int nb, const unsigned int dx, const unsigned int dy = 1, const unsigned int dz = 1, const unsigned int dc = 1) [static]`

Save empty (non-compressed) .cimg file with specified dimensions.

Parameters

<i>filename</i>	Filename to write data to.
<i>nb</i>	Number of images to write.
<i>dx</i>	Width of images in the written file.
<i>dy</i>	Height of images in the written file.
<i>dz</i>	Depth of images in the written file.
<i>dc</i>	Spectrum of images in the written file.

8.4.4.116 `static void save_empty_cimg (std::FILE *const file, const unsigned int nb, const unsigned int dx, const unsigned int dy = 1, const unsigned int dz = 1, const unsigned int dc = 1) [static]`

Save empty .cimg file with specified dimensions.

Parameters

<i>file</i>	File to write data to.
<i>nb</i>	Number of images to write.
<i>dx</i>	Width of images in the written file.
<i>dy</i>	Height of images in the written file.
<i>dz</i>	Depth of images in the written file.
<i>dc</i>	Spectrum of images in the written file.

8.4.4.117 `const CImgList<T>& save_tiff (const char *const filename, const unsigned int compression_type = 0, const float *const voxel_size = 0, const char *const description = 0, const bool use_big_tiff = true) const`

Save list as a TIFF file.

Parameters

<i>filename</i>	Filename to write data to.
<i>compression_type</i>	Compression mode used to write data.

8.4.4.118 `const ClmgList<T>& save_gzip_external (const char *const filename) const`

Save list as a gzipped file, using external tool 'gzip'.

Parameters

<i>filename</i>	Filename to write data to.
-----------------	----------------------------

8.4.4.119 `const ClmgList<T>& save_video (const char *const filename, const unsigned int fps = 25, const char * codec = 0, const bool keep_open = false) const`

Save image sequence, using the OpenCV library.

Parameters

<i>filename</i>	Filename to write data to.
<i>fps</i>	Number of frames per second.
<i>codec</i>	Type of compression (See http://www.fourcc.org/codecs.php to see available codecs).
<i>keep_open</i>	Tells if the video writer associated to the specified filename must be kept open or not (to allow frames to be added in the same file afterwards).

8.4.4.120 `const ClmgList<T>& save_ffmpeg_external (const char *const filename, const unsigned int fps = 25, const char *const codec = 0, const unsigned int bitrate = 2048) const`

Save image sequence, using the external tool 'ffmpeg'.

Parameters

<i>filename</i>	Filename to write data to.
<i>fps</i>	Number of frames per second.
<i>codec</i>	Type of compression.
<i>bitrate</i>	Output bitrate

8.4.4.121 `Clmg<ucharT> get_serialize (const bool is_compressed = false) const`

Serialize a ClmgList<T> instance into a raw Clmg<unsigned char> buffer.

Parameters

<i>is_compressed</i>	tells if zlib compression must be used for serialization (this requires 'cimg_use_zlib' been enabled).
----------------------	--

8.4.4.122 `static const CImgList<ucharT>& font (const unsigned int font_height, const bool is_variable_width = true)`
`[static]`

Return a [CImg](#) pre-defined font with desired size.

Parameters

<i>font_height</i>	Height of the desired font (exact match for 13,23,53,103).
<i>is_variable_width</i>	Decide if the font has a variable (<code>true</code>) or fixed (<code>false</code>) width.

8.4.4.123 `CImgList<T>& FFT (const char axis, const bool invert = false)`

Compute a 1d Fast Fourier Transform, along specified axis.

Parameters

<i>axis</i>	Axis along which the Fourier transform is computed.
<i>invert</i>	Tells if the direct (<code>false</code>) or inverse transform (<code>true</code>) is computed.

8.4.4.124 `CImgList<T>& FFT (const bool invert = false)`

Compute a n-d Fast Fourier Transform.

Parameters

<i>invert</i>	Tells if the direct (<code>false</code>) or inverse transform (<code>true</code>) is computed.
---------------	--