The CImg Library

1.5.8

Generated by Doxygen 1.8.4

Thu Feb 27 2014 14:25:02

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:

Chapter 3

Namespace Index

3.1 Namespace List

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

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Class Index

5.1 Class List

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

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 stuffs 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 stuffs.
- 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 catched with a bloc `try { ..} catch (CImgException) { .. }`. 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 a 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 provide 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 avoid to handle complex dependancies 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 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 CImg-based program with ICL :

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

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

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

- **g++ (Linux version)** : Use the following command to compile a CImg-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 CImg-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 CImg-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 CImg library package to see how it works.

If you are using another compilers and encounter problems, please `write me` since maintaining compatibility is one of the priority of the CImg Library. Nevertheless, old compilers that does not respect the C++ norm will not support the CImg Library.

6.1.4 What's next ?

If you are ready to get more, and to start writing more serious programs with CImg, you are invited to go to the Tutorial : Getting Started. section.

6.2 FAQ : Frequently Asked Questions.

6.2.1 FAQ Summary

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 stuffs 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 `Sourceforge servers`.

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 at Sourceforge, 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 undoubtly 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 `"<tt>CImg.h</tt>"` file. An include of the plugin file is performed in the CImg<T> class. If `cimg_plugin` if 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 CImg library with Visual C++ 2005 Express Edition ?.

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

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

- Download CImg 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\VCProject-Defaults\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 CImg 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 detailled 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 than 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 (first 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 than 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-Icpc);     // 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), neighbor(5,5);  // 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 is 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

usage	: 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 programm.

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

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

Returns

`cimg_option()` returns an object that has the *same type* than 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 an 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 nblevels  = 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(nblevels);
  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 writting 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 CImgList

  Represent a list of images CImg<T>.
- struct CImgException

  Instances of `CImgException` are thrown when errors are encountered in a `CImg` function call.
- struct CImgDisplay

  Allow to create windows, display images on them and manage user events (keyboard, mouse and windows events).

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 reasonnably 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 informations about `CImg` environement 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.
- 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.

- unsigned int wait (const unsigned int milliseconds)

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

- double rand ()

  Return a random variable between [0,1] with respect to an uniform distribution.

- double crand ()

  Return a random variable between [-1,1] with respect to an uniform distribution.

- 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.
- char uncase (const char x)

  Convert ascii character to lower case.
- void uncase (char *const str)

  Convert C-string to lower 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.
- bool strpare (char *const str, const char delimiter=' ', const bool is_symmetric=false, const bool is_-iterative=false)

  Remove delimiters on the start and/or end of a C-string.
- void strunescape (char *const str)

  Replace escape sequences in C-strings by their binary ascii values.
- const char * basename (const char *const str)

  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.
- const char * temporary_path (const char *const user_path=0, const bool reinit_path=false)

  Get/set path to store temporary files.
- const char * imagemagick_path (const char *const user_path=0, const bool reinit_path=false)

  Get/set path to the Program Files/ directory (Windows only).
- const char * graphicsmagick_path (const char *const user_path=0, const bool reinit_path=false)

Get/set path to the GraphicsMagick's `gm` binary.
- const char ∗ medcon_path (const char ∗const user_path=0, const bool reinit_path=false)

  Get/set path to the XMedcon's `medcon` binary.
- const char ∗ ffmpeg_path (const char ∗const user_path=0, const bool reinit_path=false)

  Get/set path to the FFMPEG's `ffmpeg` binary.
- const char ∗ gzip_path (const char ∗const user_path=0, const bool reinit_path=false)

  Get/set path to the `gzip` binary.
- const char ∗ gunzip_path (const char ∗const user_path=0, const bool reinit_path=false)

  Get/set path to the `gzip` binary.
- const char ∗ dcraw_path (const char ∗const user_path=0, const bool reinit_path=false)

  Get/set path to the `dcraw` binary.
- const char ∗ wget_path (const char ∗const user_path=0, const bool reinit_path=false)

  Get/set path to the `wget` binary.
- const char ∗ curl_path (const char ∗const user_path=0, const bool reinit_path=false)

  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.
- const char ∗ file_type (std::FILE ∗const file, const char ∗const filename)

  Try to guess format from an image file.
- template<typename T >
  int fread (T ∗const ptr, const unsigned long nmemb, std::FILE ∗stream)

  Read data from file.
- template<typename T >
  int fwrite (const T ∗ptr, const unsigned long nmemb, std::FILE ∗stream)

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

  Create an empty file.
- char ∗ load_network_external (const char ∗const filename, char ∗const filename_local)

  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 defaut, const char ∗const usage, const bool reset_static)

  Return options specified on the command line.
- 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 `PADDIV` 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

file	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 informations about `CImg` environement 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

mode	Desired exception mode. Possible values are: • `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

expression	C-string describing the formula to evaluate.
x	Value of the pre-defined variable `x`.
y	Value of the pre-defined variable `y`.
z	Value of the pre-defined variable `z`.
c	Value of the pre-defined variable `c`.

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

format	C-string containing the format of the message, as with `std::printf()`.

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

command	C-string containing the command line to execute.
module_name	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

`in,out`	*buffer*	Memory buffer whose endianness must be reversed.
	size	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

`in,out`	*a*	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

milliseconds	Number of milliseconds to wait for.

Note

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

7.2.2.15 unsigned int cimg_library::cimg::wait (const unsigned int *milliseconds*)

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

Parameters

milliseconds	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

x	Input value.
m	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

x	Value to be rounded.
y	Rounding precision.
rounding_type	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

str	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

str1	C-string.
str2	C-string.
l	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

str1	C-string.
str2	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 bool cimg_library::cimg::strpare (char ∗const *str*, const char *delimiter* = *' '* , const bool *is_symmetric* = `false`, const bool *is_iterative* = `false`)

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

Parameters

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

Returns

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

7.2.2.23 void cimg_library::cimg::strunescape (char ∗const *str*)

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

Parameters

`in,out`	*str*	C-string to work with (modified at output).

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

Open a file.

Parameters

path	Path of the filename to open.
mode	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`.

7.2.2.25 int cimg_library::cimg::fclose (std::FILE ∗ *file*)

Close a file.

Parameters

file	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.

7.2.2.26 const char∗ cimg_library::cimg::temporary_path (const char ∗const *user_path* = `0`, const bool *reinit_path* = `false`)

Get/set path to store temporary files.

Parameters

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

Returns

Path where temporary files can be saved.

7.2.2.27 const char* cimg_library::cimg::imagemagick_path (const char *const *user_path* = 0, const bool *reinit_path* = false)

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

Parameters

user_path	Specified path, or 0 to get the path currently used.
reinit_path	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

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

Returns

Path containing the `convert` binary.

7.2.2.28 const char* cimg_library::cimg::graphicsmagick_path (const char *const *user_path* = 0, const bool *reinit_path* = false)

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

Parameters

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

Returns

Path containing the `gm` binary.

7.2.2.29 const char* cimg_library::cimg::medcon_path (const char *const *user_path* = 0, const bool *reinit_path* = false)

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

Parameters

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

Returns

Path containing the `medcon` binary.

7.2.2.30 const char∗ cimg_library::cimg::ffmpeg_path (const char ∗const *user_path* = `0`, const bool *reinit_path* = `false`)

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

Parameters

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

Returns

Path containing the `ffmpeg` binary.

7.2.2.31 const char∗ cimg_library::cimg::gzip_path (const char ∗const *user_path* = `0`, const bool *reinit_path* = `false`)

Get/set path to the `gzip` binary.

Parameters

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

Returns

Path containing the `gzip` binary.

7.2.2.32 const char∗ cimg_library::cimg::gunzip_path (const char ∗const *user_path* = `0`, const bool *reinit_path* = `false`)

Get/set path to the `gzip` binary.

Parameters

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

Returns

Path containing the `gunzip` binary.

7.2.2.33 const char∗ cimg_library::cimg::dcraw_path (const char ∗const *user_path* = `0`, const bool *reinit_path* = `false`)

Get/set path to the `dcraw` binary.

Parameters

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

Returns

Path containing the `dcraw` binary.

7.2.2.34 const char* cimg_library::cimg::wget_path (const char *const *user_path* = 0, const bool *reinit_path* = false)

Get/set path to the `wget` binary.

Parameters

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

Returns

Path containing the `wget` binary.

7.2.2.35 const char* cimg_library::cimg::curl_path (const char *const *user_path* = 0, const bool *reinit_path* = false)

Get/set path to the `curl` binary.

Parameters

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

Returns

Path containing the `curl` binary.

7.2.2.36 const char* cimg_library::cimg::file_type (std::FILE *const *file*, const char *const *filename*)

Try to guess format from an image file.

Parameters

file	Input file (can be 0 if `filename` is set).
filename	Filename, as a C-string (can be 0 if `file` is set).

Returns

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

7.2.2.37 int cimg_library::cimg::fread (T *const *ptr*, const unsigned long *nmemb*, std::FILE * *stream*)

Read data from file.

Parameters

out	*ptr*	Pointer to memory buffer that will contain the binary data read from file.
	nmemb	Number of elements to read.
	stream	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.

7.2.2.38 int cimg_library::cimg::fwrite (const T * *ptr*, const unsigned long *nmemb*, std::FILE * *stream*)

Write data to file.

Parameters

	ptr	Pointer to memory buffer containing the binary data to write on file.
	nmemb	Number of elements to write.
out	*stream*	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.

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

Create an empty file.

Parameters

file	Input file (can be `0` if `filename` is set).
filename	Filename, as a C-string (can be `0` if `file` is set).

7.2.2.40 char* cimg_library::cimg::load_network_external (const char *const *filename*, char *const *filename_local*)

Load file from network as a local temporary file.

Parameters

	filename	Filename, as a C-string.
out	*filename_local*	C-string containing the path to a local copy of `filename`.

Returns

Value of `filename_local`.

Note

Use external binaries `wget` or `curl` to perform. You must have one of these tools installed to be able to use this function.

7.2.2.41 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

title	Title of the dialog window.
msg	Main message displayed inside the dialog window.
button1_label	Label of the 1st button.
button2_label	Label of the 2nd button (`0` to hide button).
button3_label	Label of the 3rd button (`0` to hide button).
button4_label	Label of the 4th button (`0` to hide button).
button5_label	Label of the 5th button (`0` to hide button).
button6_label	Label of the 6th button (`0` to hide button).
logo	Image logo displayed at the left of the main message.
is_centered	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 >
  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.

- 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 **[specialization]**.*
- CImg (const char *const filename)

  Construct image from reading an image file.
- template<typename t >
  CImg (const CImg< t > &img)

  Construct image copy.
- CImg (const CImg< T > &img)

  *Construct image copy **[specialization]**.*
- template<typename t >
  CImg (const CImg< t > &img, const bool is_shared)

  Advanced copy constructor.
- CImg (const CImg< T > &img, const bool is_shared)

  *Advanced copy constructor **[specialization]**.*
- template<typename t >
  CImg (const CImg< t > &img, const char *const dimensions)

  Construct image with dimensions borrowed from another image.
- template<typename t >
  CImg (const CImg< t > &img, const char *const dimensions, const T value)

  Construct image with dimensions borrowed from another image and initialize pixel values.
- CImg (const CImgDisplay &disp)

  Construct image from a display window.
- CImg< T > & assign ()

  *Construct empty image **[in-place version]**.*
- 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]**.*
- 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]**.*
- 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]**.*
- 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]**.*
- 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]**.*
- template<typename t >
  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]**.*
- 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 **[specialization]**.*
- template<typename t >
  CImg< 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]**.*

- CImg< 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]**.*
- CImg< T > & assign (const char *const filename)

  *Construct image from reading an image file **[in-place version]**.*
- template<typename t >
  CImg< T > & assign (const CImg< t > &img)

  *Construct image copy **[in-place version]**.*
- template<typename t >
  CImg< T > & assign (const CImg< t > &img, const bool is_shared)

  In-place version of the advanced copy constructor.
- template<typename t >
  CImg< T > & assign (const CImg< t > &img, const char *const dimensions)

  *Construct image with dimensions borrowed from another image **[in-place version]**.*
- template<typename t >
  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]**.*
- CImg< T > & assign (const CImgDisplay &disp)

  *Construct image from a display window **[in-place version]**.*
- CImg< T > & clear ()

  *Construct empty image **[in-place version]**.*
- template<typename t >
  CImg< t > & move_to (CImg< t > &img)

  Transfer content of an image instance into another one.
- CImg< T > & move_to (CImg< T > &img)

  *Transfer content of an image instance into another one **[specialization]**.*
- template<typename t >
  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.
- CImg< T > & swap (CImg< T > &img)

  Swap fields of two image instances.
- static CImg< T > & empty ()

  Return a reference to an empty image.

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 long wh, const unsigned long 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 long wh, const unsigned long whd=0) const

  *Access to a pixel value **[const version]**.*
- operator T * ()

  Implicitely cast an image into a `T*`.
- operator const T * () const

  *Implicitely 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 substraction operator.
- CImg< T > & operator-= (const char ∗const expression)

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

  In-place substraction 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

  Substraction operator.

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

  Substraction operator.

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

  Substraction 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.
- unsigned long 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]**.*
- long 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.
- 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.
- 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.
- 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.
- 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.
- 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 and 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 and 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 and 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 X,Y and Z-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 X,Y and Z-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

  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 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.
- 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.

Mathematical Functions

- CImg< T > & sqr ()

  Compute the square value of each pixel value.
- CImg< Tfloat > get_sqr () const

  *Compute the square value of each pixel value **[new-instance version]**.*
- CImg< T > & sqrt ()

  Compute the square root of each pixel value.
- CImg< Tfloat > get_sqrt () const

  *Compute the square root of each pixel value **[new-instance version]**.*
- CImg< T > & exp ()

  Compute the exponential of each pixel value.
- CImg< Tfloat > get_exp () const

  *Compute the exponential of each pixel value **[new-instance version]**.*
- CImg< T > & log ()

  Compute the logarithm of each pixel value.
- CImg< Tfloat > get_log () const

  *Compute the logarithm of each pixel value **[new-instance version]**.*
- CImg< T > & log2 ()

  Compute the base-2 logarithm of each pixel value.
- CImg< Tfloat > get_log2 () const

  *Compute the base-10 logarithm of each pixel value **[new-instance version]**.*
- CImg< T > & log10 ()

  Compute the base-10 logarithm of each pixel value.
- CImg< Tfloat > get_log10 () const

  *Compute the base-10 logarithm of each pixel value **[new-instance version]**.*
- CImg< T > & abs ()

  Compute the absolute value of each pixel value.
- CImg< Tfloat > get_abs () const

  *Compute the absolute value of each pixel value **[new-instance version]**.*
- CImg< T > & sign ()

  Compute the sign of each pixel value.
- CImg< Tfloat > get_sign () const

  *Compute the sign of each pixel value **[new-instance version]**.*
- CImg< T > & cos ()

  Compute the cosine of each pixel value.
- CImg< Tfloat > get_cos () const

  *Compute the cosine of each pixel value **[new-instance version]**.*

- CImg< T > & sin ()

  Compute the sine of each pixel value.
- CImg< 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 cimg::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 cimg::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.
- CImg< Tfloat > get_pow (const double p) const

  *Raise each pixel value to a specified power **[new-instance version]**.*
- CImg< T > & pow (const char ∗const expression)

  Raise each pixel value to a power, specified from an expression.
- CImg< 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 >
  CImg< T > & pow (const CImg< t > &img)

  Raise each pixel value to a power, pointwisely specified from another image.
- template<typename t >
  CImg< Tfloat > get_pow (const CImg< t > &img) const

  *Raise each pixel value to a power, pointwisely specified from another image **[new-instance version]**.*
- CImg< T > & rol (const unsigned int n=1)

  Compute the bitwise left rotation of each pixel value.
- CImg< T > get_rol (const unsigned int n=1) const

  *Compute the bitwise left rotation of each pixel value **[new-instance version]**.*
- CImg< T > & rol (const char ∗const expression)

  Compute the bitwise left rotation of each pixel value.
- CImg< T > get_rol (const char ∗const expression) const

  *Compute the bitwise left rotation of each pixel value **[new-instance version]**.*
- template<typename t >
  CImg< T > & rol (const CImg< t > &img)

  Compute the bitwise left rotation of each pixel value.
- template<typename t >
  CImg< T > get_rol (const CImg< t > &img) const

  *Compute the bitwise left rotation of each pixel value **[new-instance version]**.*
- CImg< T > & ror (const unsigned int n=1)

  Compute the bitwise right rotation of each pixel value.
- CImg< T > get_ror (const unsigned int n=1) const

  *Compute the bitwise right rotation of each pixel value **[new-instance version]**.*
- CImg< T > & ror (const char ∗const expression)

  Compute the bitwise right rotation of each pixel value.
- CImg< T > get_ror (const char ∗const expression) const

  *Compute the bitwise right rotation of each pixel value **[new-instance version]**.*
- template<typename t >
  CImg< T > & ror (const CImg< t > &img)

  Compute the bitwise right rotation of each pixel value.
- template<typename t >
  CImg< T > get_ror (const CImg< t > &img) const

  *Compute the bitwise right rotation of each pixel value **[new-instance version]**.*
- CImg< T > & min (const T val)

  Pointwise min operator between instance image and a value.
- CImg< T > get_min (const T val) const

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

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

  Pointwise min operator between two images.
- template< typename t >
  CImg< typename cimg::superset
  < T, t >::type > get_min (const CImg< t > &img) const

  *Pointwise min operator between two images **[new-instance version]**.*
- CImg< 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.
- Tdouble sum () const

  Return the sum of all the pixel values.

- Tdouble mean () const

  Return the average pixel value.
- Tdouble variance (const unsigned int variance_method=1) const

  Return the variance of the pixel values.
- template<typename t >
  Tdouble variance_mean (const unsigned int variance_method, t &mean) const

  Return the variance as well as the average of the pixel values.
- Tdouble variance_noise (const unsigned int variance_method=2) const

  Return estimated variance of the noise.
- template<typename t >
  Tdouble MSE (const CImg< t > &img) const

  Compute the MSE (Mean-Squared Error) between two images.
- template<typename t >
  Tdouble PSNR (const CImg< t > &img, const Tdouble 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

  Evaluate math formula.
- template<typename t >
  CImg< doubleT > eval (const char *const expression, const CImg< t > &xyzc) const

  Evaluate math formula on a set of variables.
- CImg< Tdouble > get_stats (const unsigned int variance_method=1) const

  Compute statistics vector from the pixel values.
- CImg< T > & stats (const unsigned int variance_method=1)

  Compute statistics vector from the pixel values ***[in-place version]****.*

Vector / Matrix Operations

- Tdouble magnitude (const int magnitude_type=2) const

  Compute norm of the image, viewed as a matrix.
- Tdouble trace () const

  Compute the trace of the image, viewed as a matrix.
- Tdouble det () const

  Compute the determinant of the image, viewed as a matrix.
- template<typename t >
  Tdouble dot (const CImg< t > &img) const

  Compute the dot product between instance and argument, viewed as matrices.
- 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.
- 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.
- 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.
- template<typename t >
  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.
- template<typename t >
  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.

- template< typename t >
  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.
- CImg< T > & vector ()

  Unroll pixel values along axis y.
- CImg< T > get_vector () const

  *Unroll pixel values along axis y **[new-instance version]**.*
- CImg< T > & matrix ()

  Resize image to become a scalar square matrix.
- CImg< T > get_matrix () const

  *Resize image to become a scalar square matrix **[new-instance version]**.*
- CImg< T > & tensor ()

  Resize image to become a symmetric tensor.
- CImg< T > get_tensor () const

  *Resize image to become a symmetric tensor **[new-instance version]**.*
- CImg< T > & diagonal ()

  Resize image to become a diagonal matrix.
- CImg< T > get_diagonal () const

  *Resize image to become a diagonal matrix **[new-instance version]**.*
- CImg< T > & identity_matrix ()

  Replace the image by an identity matrix.
- CImg< T > get_identity_matrix () const

  *Replace the image by an identity matrix **[new-instance version]**.*
- CImg< T > & sequence (const T a0, const T a1)

  Fill image with a linear sequence of values.
- CImg< T > get_sequence (const T a0, const T a1) const

  *Fill image with a linear sequence of values **[new-instance version]**.*
- CImg< T > & transpose ()

  Transpose the image, viewed as a matrix.
- CImg< T > get_transpose () const

  *Transpose the image, viewed as a matrix **[new-instance version]**.*
- template< typename t >
  CImg< T > & cross (const CImg< t > &img)

  Compute the cross product between two $1x3$ images, viewed as 3d vectors.
- template< typename t >
  CImg< typename cimg::superset
  < T, t >::type > get_cross (const CImg< t > &img) const

  *Compute the cross product between two $1x3$ images, viewed as 3d vectors **[new-instance version]**.*
- CImg< T > & invert (const bool use_LU=true)

  Invert the instance image, viewed as a matrix.
- CImg< 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)

  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]**.*

- CImg< T > & fill (const T val0, const T val1)

  Fill sequentially all pixel values with specified values.

- CImg< T > get_fill (const T val0, const T val1) const

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

- CImg< T > & fill (const T val0, const T val1, const T val2)

  *Fill sequentially all pixel values with specified values **[overloading]**.*

- CImg< T > get_fill (const T val0, const T val1, const T val2) const

  *Fill sequentially all pixel values with specified values **[new-instance version]**.*
- CImg< T > & fill (const T val0, const T val1, const T val2, const T val3)

  *Fill sequentially all pixel values with specified values **[overloading]**.*
- CImg< T > get_fill (const T val0, const T val1, const T val2, const T val3) const

  *Fill sequentially all pixel values with specified values **[new-instance version]**.*
- CImg< T > & fill (const T val0, const T val1, const T val2, const T val3, const T val4)

  *Fill sequentially all pixel values with specified values **[overloading]**.*
- CImg< T > get_fill (const T val0, const T val1, const T val2, const T val3, const T val4) const

  *Fill sequentially all pixel values with specified values **[new-instance version]**.*
- CImg< T > & fill (const T val0, const T val1, const T val2, const T val3, const T val4, const T val5)

  *Fill sequentially all pixel values with specified values **[overloading]**.*
- CImg< T > get_fill (const T val0, const T val1, const T val2, const T val3, const T val4, const T val5) const

  *Fill sequentially all pixel values with specified values **[new-instance version]**.*
- CImg< T > & fill (const T val0, const T val1, const T val2, const T val3, const T val4, const T val5, const T val6)

  *Fill sequentially all pixel values with specified values **[overloading]**.*
- CImg< 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

  *Fill sequentially all pixel values with specified values **[new-instance version]**.*
- CImg< 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)

  *Fill sequentially all pixel values with specified values **[overloading]**.*
- CImg< 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

  *Fill sequentially all pixel values with specified values **[new-instance version]**.*
- CImg< 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)

  *Fill sequentially all pixel values with specified values **[overloading]**.*
- CImg< 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

  *Fill sequentially all pixel values with specified values **[new-instance version]**.*
- CImg< 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)

  *Fill sequentially all pixel values with specified values **[overloading]**.*
- CImg< 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

  *Fill sequentially all pixel values with specified values **[new-instance version]**.*
- CImg< 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)

  *Fill sequentially all pixel values with specified values **[overloading]**.*
- CImg< 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

  *Fill sequentially all pixel values with specified values **[new-instance version]**.*
- CImg< 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]**.*
- CImg< 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]**.*
- CImg< 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]**.*

- CImg$<$ 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]**.*
- CImg$<$ 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]**.*
- CImg$<$ 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]**.*
- CImg$<$ 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]**.*
- CImg$<$ 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]**.*
- CImg$<$ 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]**.*
- CImg$<$ 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]**.*
- CImg$<$ T $>$ & fill (const char $*$const expression, const bool repeat_flag)

  Fill sequentially pixel values according to a given expression.
- CImg$<$ T $>$ get_fill (const char $*$const values, const bool repeat_values) const

  *Fill sequentially pixel values according to a given expression **[new-instance version]**.*
- template$<$typename t $>$
  CImg$<$ T $>$ & fill (const CImg$<$ t $>$ &values, const bool repeat_values=true)

  Fill sequentially pixel values according to the values found in another image.
- template$<$typename t $>$
  CImg$<$ T $>$ get_fill (const CImg$<$ t $>$ &values, const bool repeat_values=true) const

  *Fill sequentially pixel values according to the values found in another image **[new-instance version]**.*
- 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.
- CImg$<$ 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]**.*

- CImg< T > & discard (const T value)

  Discard specified value in the image buffer.

- CImg< T > get_discard (const T value) const

  *Discard specified value in the image buffer **[new-instance version]**.*

- template<typename t >
  CImg< T > & discard (const CImg< t > &values)

  Discard specified sequence of values in the image buffer.

- template<typename t >
  CImg< T > get_discard (const CImg< t > &values) const

  *Discard specified sequence of values in the image buffer **[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 L2-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=(T) 0, const T max_value=(T) 0)

  Compute the histogram of pixel values.
- CImg< ulongT > get_histogram (const unsigned int nb_levels, const T min_value=(T) 0, const T max_-value=(T) 0) const

  *Compute the histogram of pixel values **[new-instance version]**.*
- CImg< T > & equalize (const unsigned int nb_levels, const T min_value=(T) 0, const T max_value=(T) 0)

  Equalize histogram of pixel values.
- CImg< T > get_equalize (const unsigned int nblevels, const T val_min=(T) 0, const T val_max=(T) 0) 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)

  Map predefined colormap on the scalar (indexed) image instance.
- template< typename t >
  CImg< t > get_map (const CImg< t > &colormap) 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< unsigned long > 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< unsigned long > 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]**.*
- CImg< T > & RGBtoBayer ()

  Convert RGB color image to a Bayer-coded scalar image.
- CImg< T > get_RGBtoBayer () const

  *Convert RGB color image to a Bayer-coded scalar image **[new-instance version]**.*
- CImg< T > & BayertoRGB (const unsigned int interpolation_type=3)

Convert Bayer-coded scalar image to a RGB color image.

- CImg< Tuchar > get_BayertoRGB (const unsigned int interpolation_type=3) const

  *Convert Bayer-coded scalar image to a RGB color image **[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=0)

  Rotate image with arbitrary angle.
- CImg< T > get_rotate (const float angle, const unsigned int interpolation=1, const unsigned int boundary=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=3)

  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=3) const

  *Rotate image with arbitrary angle, around a center point **[new-instance version]**.*
- template<typename t >
  CImg< T > & warp (const CImg< t > &warp, const bool is_relative=false, const unsigned int interpolation=1, const unsigned int boundary_conditions=0)

  Warp image content by a warping field.
- template<typename t >
  CImg< T > get_warp (const CImg< t > &warp, const bool is_relative=false, const unsigned int interpolation=1, const unsigned int boundary_conditions=0) const

  *Warp image content by a warping field **[new-instance version]***
- 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.
- CImg< 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]**.*

- 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.

- CImg< 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]**.*

- CImg< 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]**.*

- CImg< 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]**.*

- CImg< T > & crop (const int x0, const int y0, const int x1, const int y1, const bool boundary_conditions=false)

  *Crop image region **[overloading]**.*

- CImg< 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]**.*

- CImg< T > & crop (const int x0, const int x1, const bool boundary_conditions=false)

  *Crop image region **[overloading]**.*

- CImg< T > get_crop (const int x0, const int x1, const bool boundary_conditions=false) const

  *Crop image region **[new-instance version]**.*

- CImg< T > & autocrop (const T value, const char *const axes="czyx")

  Autocrop image region, regarding the specified background value.

- CImg< T > get_autocrop (const T value, const char *const axes="czyx") const

  *Autocrop image region, regarding the specified background value **[new-instance version]**.*

- CImg< T > & autocrop (const T *const color=0, const char *const axes="zyx")

  Autocrop image region, regarding the specified background color.

- CImg< 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 >
  CImg< T > & autocrop (const CImg< t > &color, const char *const axes="zyx")

  *Autocrop image region, regarding the specified background color **[overloading]**.*

- template<typename t >
  CImg< T > get_autocrop (const CImg< t > &color, const char *const axes="zyx") const

  *Autocrop image region, regarding the specified background color **[new-instance version]**.*

- CImg< T > get_column (const int x0) const

  Return specified image column.

- CImg< T > & column (const int x0)

  *Return specified image column **[in-place version]**.*

- CImg< 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]**.*
- 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.
- const CImg< 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]**.*
- 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.
- const CImg< 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]**.*
- CImg< T > get_shared_channel (const unsigned int c0)
  Return a shared-memory image referencing one channel of the image instance.

- const CImg< T > get_shared_channel (const unsigned int c0) const

    *Return a shared-memory image referencing one channel of the image instance **[const version]**.*
- CImg< T > get_shared ()

    Return a shared-memory version of the image instance.
- const CImg< T > get_shared () const

    *Return a shared-memory version of the image instance **[const version]**.*
- CImgList< T > get_split (const char axis, const int nb=0) const

    Split image into a list along specified axis.
- CImgList< T > get_split (const T value, const bool keep_values, const bool is_shared) const

    Split image into a list of one-column vectors, according to a specified splitting value.
- template< typename t >
  CImgList< T > get_split (const CImg< t > &values, const bool keep_values, const bool is_shared) const

    Split image into a list of one-column vectors, according to a specified splitting value sequence.
- CImgList< T > get_split (const bool is_shared) const

    Split the image into a list of one-column vectors each having same values.
- template< typename t >
  CImg< T > & append (const CImg< t > &img, const char axis='x', const float align=0)

    Append two images along specified axis.
- CImg< T > & append (const CImg< T > &img, const char axis='x', const float align=0)

    *Append two images along specified axis **[specialization]**.*
- template< typename t >
  CImg< typename cimg::superset
  < T, t >::type > get_append (const CImg< T > &img, const char axis='x', const float align=0) const

    *Append two images along specified axis **[const version]**.*
- CImg< T > get_append (const CImg< T > &img, const char axis='x', const float align=0) const

    *Append two images along specified axis **[specialization]**.*
- template< typename tfunc >
  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)

    Return stream line of a 3d vector field.
- static CImg< 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 >
  CImg< T > & correlate (const CImg< 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.

- 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 >
  CImg< T > & watershed (const CImg< t > &priority, const bool fill_lines=true)

  Compute watershed transform.
- template<typename t >
  CImg< T > get_watershed (const CImg< t > &priority, const bool fill_lines=true) const

  *Compute watershed transform **[new-instance version]**.*
- CImg< T > & deriche (const float sigma, const int order=0, const char axis='x', const bool boundary_-conditions=true)

  Apply recursive Deriche filter.
- CImg< Tfloat > get_deriche (const float sigma, const int order=0, const char axis='x', const bool boundary-_conditions=true) const

  *Apply recursive Deriche filter **[new-instance version]**.*
- CImg< T > & vanvliet (const float sigma, const int order, const char axis='x', const bool boundary_-conditions=true)

Van Vliet recursive Gaussian filter.

- CImg< Tfloat > get_vanvliet (const float sigma, const int order, const char axis='x', const bool boundary_conditions=true) const

  Blur image using Van Vliet recursive Gaussian filter. ***[new-instance version]****.*

- 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.

- CImg< 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]****.*

- CImg< T > & blur (const float sigma, const bool boundary_conditions=true, const bool is_gaussian=false)

  Blur image isotropically.

- CImg< 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 >
  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.

- template<typename t >
  CImg< T > get_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=true) const

  Blur image anisotropically, directed by a field of diffusion tensors ***[new-instance version]****.*

- 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.

- CImg< T > 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 >
  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 int bgrid_x, const int bgrid_y, const int bgrid_z, const int bgrid_r, const bool interpolation_type=true)

  Blur image, with the joint bilateral filter.

- template<typename t >
  CImg< T > get_blur_bilateral (const CImg< t > &guide, const float sigma_x, const float sigma_y, const float sigma_z, const float sigma_r, const int bgrid_x, const int bgrid_y, const int bgrid_z, const int bgrid_r, const bool interpolation_type=true) 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 int bgrid_s=-33, const int bgrid_r=32, const bool interpolation_type=true)

  Blur image using the joint bilateral filter.

- template<typename t >
  CImg< T > get_blur_bilateral (const CImg< t > &guide, const float sigma_s, const float sigma_r, const int bgrid_s=-33, const int bgrid_r=32, const bool interpolation_type=true) const

  Blur image using the bilateral 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< T > 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)

  Blur image with the median filter.
- CImg< T > get_blur_median (const unsigned int n) 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 unsigned int scheme=2)

  Compute the structure tensor field of an image.
- CImg< Tfloat > get_structure_tensors (const unsigned int scheme=2) 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)

  Estimate displacement field between two images.
- CImg< Tfloat > 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

  *Estimate displacement field between two images **[new-instance version]**.*
- 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< Tfloat > 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 & 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

- CImg< T > & shift_object3d (const float tx, const float ty=0, const float tz=0)

  Shift 3d object's vertices.
- CImg< Tfloat > get_shift_object3d (const float tx, const float ty=0, const float tz=0) const

  Shift 3d object's vertices ***[new-instance version]***.
- CImg< T > & shift_object3d ()

Shift 3d object's vertices, so that it becomes centered.

- CImg< Tfloat > get_shift_object3d () const

  *Shift 3d object's vertices, so that it becomes centered **[new-instance version]**.*

- CImg< T > & resize_object3d (const float sx, const float sy=-100, const float sz=-100)

  Resize 3d object.

- CImg< Tfloat > get_resize_object3d (const float sx, const float sy=-100, const float sz=-100) const

  *Resize 3d object **[new-instance version]**.*

- CImg< T > resize_object3d ()

  Resize 3d object to unit size.

- CImg< Tfloat > get_resize_object3d () const

  *Resize 3d object to unit size **[new-instance version]**.*

- template<typename tf , typename tp , typename tff >
  CImg< T > & append_object3d (CImgList< tf > &primitives, const CImg< tp > &obj_vertices, const CImgList< tff > &obj_primitives)

  Merge two 3d objects together.

- template<typename tp , typename tc , typename tt , typename tx >
  const CImg< T > & texturize_object3d (CImgList< tp > &primitives, CImgList< tc > &colors, const CImg< tt > &texture, const CImg< tx > &coords=CImg< tx >::empty()) const

  Texturize primitives of a 3d object.

- template<typename tf , typename tc , typename te >
  CImg< floatT > get_elevation3d (CImgList< tf > &primitives, CImgList< tc > &colors, const CImg< te > &elevation) const

  Generate a 3d elevation of the image instance.

- template<typename tf , typename tc >
  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.

- template<typename tf >
  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.

- template<typename tf >
  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.

- template<typename tp , typename tc , typename to >
  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.

- template<typename tp , typename tc >
  CImg< T > & object3dtoCImg3d (const CImgList< tp > &primitives, const CImgList< tc > &colors, const bool full_check=true)

  *Convert 3d object into a CImg3d representation **[overloading]**.*

- template<typename tp >
  CImg< T > & object3dtoCImg3d (const CImgList< tp > &primitives, const bool full_check=true)

  *Convert 3d object into a CImg3d 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 isovalue, 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 isovalue, 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 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)

  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 isovalue, 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 CImg< floatT > cone3d (CImgList< tf > &primitives, const float radius=50, const float size_z=100, const unsigned int subdivisions=24)

  Generate a 3d cone.
- template<typename tf >
  static CImg< floatT > cylinder3d (CImgList< tf > &primitives, const float radius=50, const float size_z=100, const unsigned int subdivisions=24)

Generate a 3d cylinder.

- template<typename tf >
  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)

  Generate a 3d torus.
- template<typename tf >
  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)

  Generate a 3d XY-plane.
- template<typename tf >
  static CImg< floatT > sphere3d (CImgList< tf > &primitives, const float radius=50, const unsigned int subdivisions=3)

  Generate a 3d sphere.
- template<typename tf , typename t >
  static CImg< floatT > ellipsoid3d (CImgList< tf > &primitives, const CImg< t > &tensor, const unsigned int subdivisions=3)

  Generate a 3d ellipsoid.

Drawing Functions

- template<typename tc >
  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.
- template<typename tc >
  CImg< 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 >
  CImg< T > & draw_point (const CImg< t > &points, const tc *const color, const float opacity=1)
- template<typename tc >
  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.
- template<typename tz , typename tc >
  CImg< T > & draw_line (CImg< 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 >
  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.
- template<typename tc >
  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.
- template<typename tc >
  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.
- template<typename tz , typename tc >
  CImg< T > & draw_line (CImg< tz > &zbuffer, 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 and z-buffering.

- template<typename t , typename tc >
  CImg< T > & draw_line (const CImg< 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 >
  CImg< 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 >
  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.
- template<typename tc >
  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]**.*
- template<typename t >
  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.
- template<typename tp , typename tt , typename tc >
  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=~0-U, const bool init_hatch=true)

  Draw a set of consecutive splines.
- template<typename tp , typename tc >
  CImg< T > & draw_spline (const CImg< 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 >
  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 opacity=1)

  Draw a filled 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 tc *const color, const float opacity, const unsigned int pattern)

  Draw a outlined 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 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 a 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 a 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 >
  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.
- 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, const unsigned int pattern)

  *Draw an outlined 2d ellipse **[overloading]**.*

- template<typename tc >
  CImg< 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 >
  CImg< 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 >
  CImg< T > & draw_image (const int x0, const int y0, const int z0, const int c0, const CImg< t > &sprite, const float opacity=1)

  Draw an image.
- CImg< T > & draw_image (const int x0, const int y0, const int z0, const int c0, const CImg< T > &sprite, const float opacity=1)

  *Draw an image **[specialization]**.*
- template<typename t >
  CImg< T > & draw_image (const int x0, const int y0, const int z0, const CImg< t > &sprite, const float opacity=1)

  *Draw an image **[overloading]**.*
- template<typename t >
  CImg< T > & draw_image (const int x0, const int y0, const CImg< t > &sprite, const float opacity=1)

  *Draw an image **[overloading]**.*
- template<typename t >
  CImg< T > & draw_image (const int x0, const CImg< t > &sprite, const float opacity=1)

  *Draw an image **[overloading]**.*
- template<typename t >
  CImg< T > & draw_image (const CImg< t > &sprite, const float opacity=1)

  *Draw an image **[overloading]**.*
- template<typename ti , typename tm >
  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.
- template<typename ti , typename tm >
  CImg< T > & draw_image (const int x0, const int y0, const int z0, const CImg< ti > &sprite, const CImg< tm > &mask, const float opacity=1, const float mask_max_value=1)

  *Draw a masked image **[overloading]**.*
- template<typename ti , typename tm >
  CImg< T > & draw_image (const int x0, const int y0, const CImg< ti > &sprite, const CImg< tm > &mask, const float opacity=1, const float mask_max_value=1)

  *Draw a image **[overloading]**.*
- template<typename ti , typename tm >
  CImg< T > & draw_image (const int x0, const CImg< ti > &sprite, const CImg< tm > &mask, const float opacity=1, const float mask_max_value=1)

  *Draw a image **[overloading]**.*
- template<typename ti , typename tm >
  CImg< T > & draw_image (const CImg< ti > &sprite, const CImg< tm > &mask, const float opacity=1, const float mask_max_value=1)

  Draw an image.
- template<typename tc1 , typename tc2 , typename t >
  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.
- template<typename tc , typename t >
  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]**.*

- template< typename tc , typename t >
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]**.*

- template< typename tc1 , typename tc2 >
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]**.*

- template< typename tc >
CImg< 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 >
CImg< 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 >
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.

- template< typename t1 , typename t2 >
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.

- template< typename t , typename tc >
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.

- template< typename t , typename tc >
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.

- template< typename tx , typename ty , typename tc >
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.

- template< typename tc >
CImg< 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 >
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.

- template< typename tc >
CImg< 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=~0-U, const unsigned int pattern_y=~0U)

    *Draw 2d grid **[simplification]**.*

- template<typename t , typename tc >
  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.
- template<typename tc , typename t >
  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.
- template<typename tc >
  CImg< 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 >
  CImg< 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]****.*
- CImg< T > & draw_plasma (const float alpha=1, const float beta=0, const unsigned int scale=8)

  Draw a random plasma texture.
- template<typename tc >
  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.
- template<typename tc >
  CImg< T > & draw_mandelbrot (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 ***[overloading]****.*
- template<typename tc >
  CImg< 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 >
  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.
- template<typename tc >
  CImg< 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 >
  CImg< 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 >
  CImg< T > & draw_gaussian (const float xc, const float yc, const float zc, const CImg< t > &tensor, const tc ∗const color, const float opacity=1)

  Draw a 3d gaussian function ***[overloading]****.*
- template<typename tc >
  CImg< 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 >
  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.
- template<typename tp , typename tf , typename tc , typename to , typename tz >
  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, 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, CImg< tz > &zbuffer)

  *Draw a 3d object **[simplification]**.*
- template<typename tp , typename tf , typename tc , typename to >
  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 CImgList< 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 >
  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 CImgList< 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, CImg< tz > &zbuffer)

  *Draw a 3d object **[simplification]**.*
- template<typename tp , typename tf , typename tc >
  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 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 >
  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 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, CImg< tz > &zbuffer)

  *Draw a 3d object **[simplification]**.*

Data Input

- CImg< T > & select (CImgDisplay &disp, const unsigned int feature_type=2, unsigned int *const XYZ=0)

  Launch simple interface to select a shape from an image.
- CImg< T > & select (const char *const title, const unsigned int feature_type=2, unsigned int *const XYZ=0)

  *Simple interface to select a shape from an image **[overloading]**.*
- CImg< intT > get_select (CImgDisplay &disp, const unsigned int feature_type=2, unsigned int *const XYZ=0) const

  *Simple interface to select a shape from an image **[new-instance version]**.*
- CImg< intT > get_select (const char *const title, const unsigned int feature_type=2, unsigned int *const XYZ=0) const

  *Simple interface to select a shape from an image **[new-instance version]**.*
- CImg< intT > get_select_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 labely=0, const double ymin=0, const double ymax=0) const

  Select sub-graph in a graph.
- CImg< T > & load (const char *const filename)

Load image from a file.
- CImg< T > & load_ascii (const char *const filename)

  Load image from an ascii file.
- CImg< T > & load_ascii (std::FILE *const file)

  *Load image from an ascii file **[overloading]**.*
- CImg< T > & load_dlm (const char *const filename)

  Load image from a DLM file.
- CImg< T > & load_dlm (std::FILE *const file)

  *Load image from a DLM file **[overloading]**.*
- CImg< T > & load_bmp (const char *const filename)

  Load image from a BMP file.
- CImg< T > & load_bmp (std::FILE *const file)

  *Load image from a BMP file **[overloading]**.*
- CImg< T > & load_jpeg (const char *const filename)

  Load image from a JPEG file.
- CImg< T > & load_jpeg (std::FILE *const file)

  *Load image from a JPEG file **[overloading]**.*
- CImg< T > & load_magick (const char *const filename)

  Load image from a file, using Magick++ library.
- CImg< T > & load_png (const char *const filename)

  Load image from a PNG file.
- CImg< T > & load_png (std::FILE *const file)

  *Load image from a PNG file **[overloading]**.*
- CImg< T > & load_pnm (const char *const filename)

  Load image from a PNM file.
- CImg< T > & load_pnm (std::FILE *const file)

  *Load image from a PNM file **[overloading]**.*
- CImg< T > & load_pfm (const char *const filename)

  Load image from a PFM file.
- CImg< T > & load_pfm (std::FILE *const file)

  *Load image from a PFM file **[overloading]**.*
- CImg< T > & load_rgb (const char *const filename, const unsigned int dimw, const unsigned int dimh=1)

  Load image from a RGB file.
- CImg< T > & load_rgb (std::FILE *const file, const unsigned int dimw, const unsigned int dimh=1)

  *Load image from a RGB file **[overloading]**.*
- CImg< T > & load_rgba (const char *const filename, const unsigned int dimw, const unsigned int dimh=1)

  Load image from a RGBA file.
- CImg< T > & load_rgba (std::FILE *const file, const unsigned int dimw, const unsigned int dimh=1)

  *Load image from a RGBA file **[overloading]**.*
- 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)

  Load image from a TIFF file.
- CImg< T > & load_minc2 (const char *const filename)

  Load image from a MINC2 file.
- CImg< T > & load_analyze (const char *const filename, float *const voxel_size=0)

  Load image from an ANALYZE7.5/NIFTI file.
- CImg< T > & load_analyze (std::FILE *const file, float *const voxel_size=0)

  *Load image from an ANALYZE7.5/NIFTI file **[overloading]**.*
- CImg< T > & load_cimg (const char *const filename, const char axis='z', const float align=0)

  Load image from a .cimg[z] file.
- CImg< T > & load_cimg (std::FILE *const file, const char axis='z', const float align=0)

Load image from a .cimg[z] file ***[overloading]****.*

- 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.

- CImg< 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]****.*

- CImg< T > & load_inr (const char *const filename, float *const voxel_size=0)

    Load image from an INRIMAGE-4 file.

- CImg< T > & load_inr (std::FILE *const file, float *const voxel_size=0)

    Load image from an INRIMAGE-4 file ***[overloading]****.*

- CImg< T > & load_exr (const char *const filename)

    Load image from a EXR file.

- CImg< T > & load_pandore (const char *const filename)

    Load image from a PANDORE-5 file.

- CImg< T > & load_pandore (std::FILE *const file)

    Load image from a PANDORE-5 file ***[overloading]****.*

- CImg< T > & load_parrec (const char *const filename, const char axis='c', const float align=0)

    Load image from a PAR-REC (Philips) file.

- CImg< 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 unsigned long offset=0)

    Load image from a raw binary file.

- CImg< 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 unsigned long offset=0)

    Load image from a raw binary file ***[overloading]****.*

- CImg< T > & load_ffmpeg (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool pixel_format=true, const bool resume=false, const char axis='z', const float align=0)

    Load image sequence using FFMPEG av's libraries.

- CImg< 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.

- CImg< 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 >
  CImg< T > & load_off (CImgList< tf > &primitives, CImgList< tc > &colors, const char *const filename)

    Load 3d object from a .OFF file.

- template<typename tf , typename tc >
  CImg< T > & load_off (CImgList< tf > &primitives, CImgList< tc > &colors, std::FILE *const file)

    Load 3d object from a .OFF file ***[overloading]****.*

- 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 GraphicsMagicks's external tools.

- CImg< T > & load_graphicsmagick_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=false, 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)

  *Loadimage 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)

  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 *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 long offset=0)

  Load image from a raw binary file ***[new-instance version]***.
- static CImg< 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 unsigned long offset=0)

  Load image from a raw binary file ***[new-instance version]***.
- static CImg< T > get_load_ffmpeg (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=~0U, const unsigned int step_frame=1, const bool pixel_format=true, const bool resume=false, const char axis='z', const float align=0)

  Load image sequence using FFMPEG av's libraries ***[new-instance version]***.

- static CImg< 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 CImg< 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 CImg< T > get_load_off (CImgList< tf > &primitives, CImgList< tc > &colors, const char *const filename)

  *Load 3d object from a .OFF file **[new-instance version]**.*
- template<typename tf , typename tc >
  static CImg< T > get_load_off (CImgList< tf > &primitives, CImgList< tc > &colors, std::FILE *const file)

  *Load 3d object from a .OFF file **[new-instance version]**.*
- static CImg< 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 CImg< 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 CImg< T > get_load_graphicsmagick_external (const char *const filename)

  *Load image using GraphicsMagick's external tool 'gm' **[new-instance version]**.*
- static CImg< T > get_load_gzip_external (const char *const filename)

  *Load gzipped image file, using external tool 'gunzip' **[new-instance version]**.*
- static CImg< T > get_load_imagemagick_external (const char *const filename)

  *Load image using ImageMagick's external tool 'convert' **[new-instance version]**.*
- static CImg< 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 CImg< 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 CImg< T > get_load_camera (const unsigned int camera_index=0, const unsigned int skip_frames=0, const bool release_camera=false, const unsigned int capture_width=0, const unsigned int capture_height=0)

  *Load image from a camera stream, using OpenCV **[new-instance version]**.*
- static CImg< T > get_load_other (const char *const filename)

  *Load image using various non-native ways **[new-instance version]**.*

Data Output

- const CImg< T > & print (const char *const title=0, const bool display_stats=true) const

  Display informations about the image data.
- const CImg< T > & display (CImgDisplay &disp) const

  Display image into a CImgDisplay window.
- const CImg< T > & display (CImgDisplay &disp, const bool display_info, unsigned int *const XYZ=0) const

  Display image into a CImgDisplay 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

  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

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

  *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

  *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

  *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

  *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

  *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

  *Display object 3d in an interactive window **[simplification]**.*

- template<typename tp >
const CImg< T > & display_object3d (const char *const title, 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

  *Display object 3d in an interactive window **[simplification]**.*

- 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 labely=0, const double ymin=0, const double ymax=0) const

  Display 1d graph in an interactive window.

- const CImg< 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

*Display 1d graph in an interactive window **[overloading]**.*
- const CImg< T > & save (const char *const filename, const int number=-1, const unsigned int digits=6) const

  Save image as a file.
- const CImg< T > & save_ascii (const char *const filename) const

  Save image as an ascii file.
- const CImg< T > & save_ascii (std::FILE *const file) const

  *Save image as an ascii file **[overloading]**.*
- const CImg< T > & save_cpp (const char *const filename) const

  Save image as a .cpp source file.
- const CImg< T > & save_cpp (std::FILE *const file) const

  *Save image as a .cpp source file **[overloading]**.*
- const CImg< T > & save_dlm (const char *const filename) const

  Save image as a DLM file.
- const CImg< T > & save_dlm (std::FILE *const file) const

  *Save image as a DLM file **[overloading]**.*
- const CImg< T > & save_bmp (const char *const filename) const

  Save image as a BMP file.
- const CImg< T > & save_bmp (std::FILE *const file) const

  *Save image as a BMP file **[overloading]**.*
- const CImg< T > & save_jpeg (const char *const filename, const unsigned int quality=100) const

  Save image as a JPEG file.
- const CImg< T > & save_jpeg (std::FILE *const file, const unsigned int quality=100) const

  *Save image as a JPEG file **[overloading]**.*
- const CImg< T > & save_magick (const char *const filename, const unsigned int bytes_per_pixel=0) const

  Save image, using built-in ImageMagick++ library.
- const CImg< T > & save_png (const char *const filename, const unsigned int bytes_per_pixel=0) const

  Save image as a PNG file.
- const CImg< T > & save_png (std::FILE *const file, const unsigned int bytes_per_pixel=0) const

  *Save image as a PNG file **[overloading]**.*
- const CImg< T > & save_pnm (const char *const filename, const unsigned int bytes_per_pixel=0) const

  Save image as a PNM file.
- const CImg< T > & save_pnm (std::FILE *const file, const unsigned int bytes_per_pixel=0) const

  *Save image as a PNM file **[overloading]**.*
- const CImg< T > & save_pnk (const char *const filename) const

  Save image as a PNK file.
- const CImg< T > & save_pnk (std::FILE *const file) const

  *Save image as a PNK file **[overloading]**.*
- const CImg< 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

  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_ffmpeg (const char *const filename, const unsigned int fps=25, const unsigned int bitrate=2048) const

  Save image as a video file, using the FFmpeg library.
- 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_ffmpeg_external (const char *const filename, const char *const codec=0, const unsigned int fps=25, 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_graphicsmagick_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
- 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 = float>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,false);` constructs a 256x256 color image from a `unsigned char*` buffer `data_buffer` (where R,G,B channels follow each others).
  - `CImg<unsigned char> img(data_buffer,256,256,1,3,true);` constructs a 256x256 color image from a `unsigned char*` buffer `data_buffer` (where R,G,B channels are multiplexed).

  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()(): allows to access 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', through
       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, through 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

size_x	Image width().
size_y	Image height().
size_z	Image depth().
size_c	Image spectrum() (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,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

size_x	Image width().
size_y	Image height().
size_z	Image depth().
size_c	Image spectrum() (number of channels).
value	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

size_x	Image width().
size_y	Image height().
size_z	Image depth().
size_c	Image spectrum() (number of channels).
value0	First value of the initialization sequence (must be an *integer*).
value1	Second value of the initialization sequence (must be an *integer*).
...	

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

size_x	Image width().
size_y	Image height().
size_z	Image depth().
size_c	Image spectrum() (number of channels).
value0	First value of the initialization sequence (must be a *double*).
value1	Second value of the initialization sequence (must be a *double*).
...	

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

size_x	Image width().
size_y	Image height().

size_z	Image depth().
size_c	Image spectrum() (number of channels).
values	Value string describing the way pixel values are set.
repeat_values	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

values	Pointer to the input memory buffer.
size_x	Image width().
size_y	Image height().
size_z	Image depth().
size_c	Image spectrum() (number of channels).
is_shared	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 `CImgInstanceException` 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

filename	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 higly 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

img	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

img	Input image to copy.
is_shared	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

img	Input image from which dimensions are borrowed.
dimensions	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

img	Input image from which dimensions are borrowed.
dimensions	String describing the image size along the X,Y,Z and V-dimensions.
value	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

disp	Input display window.

Note

- The width() and height() of the constructed image instance are the same as the specified `CImgDisplay`.
- 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

img	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

list	Destination list.
pos	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

img	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

x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	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 debuging 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 unsigned long *wh*, const unsigned long *whd* = 0)

Access to a pixel value.

Parameters

x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.
wh	Precomputed offset, must be equal to `width()*height()`.
whd	Precomputed offset, must be equal to `width()*height()*depth()`.

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 * ( )

Implicitely cast an image into a `T*`.

Implicitely 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

value	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

expression	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 unitialized
      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

value	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

expression	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

img	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 cyclically. 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 informations 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 substraction operator.

Similar to operator+=(const t), except that it performs a substraction instead of an addition.

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

In-place substraction operator.

Similar to operator+=(const char*), except that it performs a substraction instead of an addition.

8.1.4.37 CImg<T>& operator-= (const CImg< t > & *img*)

In-place substraction operator.

Similar to operator+=(const CImg<t>&), except that it performs a substraction 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

Substraction 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

Substraction 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

img	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

img	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*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 `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 `T`, if necessary.

8.1.4.56 CImg<T>& operator%= (const t *value*)

In-place modulo operator.

Similar to operator+=(const t), except that it performs a modulo operation instead of an addition.

8.1.4.57 CImg<T>& operator%= (const char ∗const *expression*)

In-place modulo operator.

Similar to operator+=(const char∗), except that it performs a modulo operation instead of an addition.

8.1.4.58 CImg<T>& operator%= (const CImg< t > & *img*)

In-place modulo operator.

Similar to operator+=(const CImg<t>&), except that it performs a modulo operation instead of an addition.

8.1.4.59 CImg< typename cimg::superset<T,t>::type > operator% (const t *value*) const

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 `T`, if necessary.

8.1.4.60 CImg<Tfloat> operator% (const char ∗const *expression*) const

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 `T`, if necessary.

8.1.4.61 CImg< typename cimg::superset<T,t>::type > operator% (const CImg< t > & *img*) const

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 `T`, if necessary.

8.1.4.62 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.

8.1.4.63 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.

8.1.4.64 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.

8.1.4.65 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`.

8.1.4.66 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`.

8.1.4.67 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`.

8.1.4.68 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.

8.1.4.69 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.

8.1.4.70 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.

8.1.4.71 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`.

8.1.4.72 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`.

8.1.4.73 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`.

8.1.4.74 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.

8.1.4.75 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.

8.1.4.76 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.

8.1.4.77 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`.

8.1.4.78 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`.

8.1.4.79 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`.

8.1.4.80 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.

8.1.4.81 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.

8.1.4.82 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.

8.1.4.83 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 `T`.

8.1.4.84 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 `T`.

8.1.4.85 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 `T`.

8.1.4.86 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.

8.1.4.87 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.

8.1.4.88 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.

8.1.4.89 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 `T`.

8.1.4.90 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`.

8.1.4.91 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`.

8.1.4.92 CImg<T> operator∼ ( ) const

Bitwise inversion operator.

Similar to operator-(), except that it compute the bitwise inverse instead of the opposite value.

8.1.4.93 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

value	Reference value to compare with.

8.1.4.94 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

expression	Value string describing the way pixel values are compared.

8.1.4.95 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

img	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.");
}
```

8.1.4.96 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

value	Reference value to compare with.

8.1.4.97 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

expression	Value string describing the way pixel values are compared.

8.1.4.98 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

img	Input image to compare with.

Note

- Writing `img1!=img2` is equivalent to !(img1==img2).

8.1.4.99 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

img	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'.
```

8.1.4.100 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

list	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.

8.1.4.101 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

axis	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();
```

8.1.4.102 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.

8.1.4.103 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`.

8.1.4.104 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`.

8.1.4.105 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`.

8.1.4.106 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`.

8.1.4.107 unsigned long 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));
```

8.1.4.108 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.

8.1.4.109 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

x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	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).

8.1.4.110 long 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

x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	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.
```

8.1.4.111 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.

8.1.4.112 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 an 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;
```

8.1.4.113 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.

8.1.4.114 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.

8.1.4.115 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

offset	Offset to the desired pixel value.
out_value	Default value returned if `offset` 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.

8.1.4.116 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

offset	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)`.

8.1.4.117 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

x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.
out_value	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.

8.1.4.118 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

x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	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 `_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.

8.1.4.119 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.

8.1.4.120 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 `_atXY(int,int,int,int)`.

8.1.4.121 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.

8.1.4.122 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(int,int,int,int)`.

8.1.4.123 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.

8.1.4.124 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 `_atXYZC(int,int,int,int)`.

8.1.4.125 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

fx	X-coordinate of the pixel value (float-valued).
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.
out_value	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.

8.1.4.126 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 (fx,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

fx	X-coordinate of the pixel value (float-valued).
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	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.

8.1.4.127 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.

8.1.4.128 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)`.

8.1.4.129 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.

8.1.4.130 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)`.

8.1.4.131 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 and 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.

8.1.4.132 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)`.

8.1.4.133 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 cubicly-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

fx	d X-coordinate of the pixel value (float-valued).
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.
out_value	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.

8.1.4.134 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.

8.1.4.135 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 cubicly-interpolated pixel value of the image instance located at (`fx`,`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

fx	X-coordinate of the pixel value (float-valued).
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	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.

8.1.4.136 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.

8.1.4.137 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.

8.1.4.138 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 and 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.

8.1.4.139 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)`.

8.1.4.140 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 and 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.

8.1.4.141 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.

8.1.4.142 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 X,Y and Z-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.

8.1.4.143 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)`.

8.1.4.144 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 X,Y and Z-coordinates.

Similar to cubic_atXYZ(float,float,float,int) const, except that you can specify the authorized minimum and maximum of the returned value.

8.1.4.145 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.

Set pixel value at specified coordinates (`fx`,`fy`,`z`,`c`) in the image instance, in a way that the value is spread amongst several neighbors if the pixel coordinates are indeed float-valued.

Parameters

value	Pixel value to set.
fx	X-coordinate of the pixel value (float-valued).
fy	Y-coordinate of the pixel value (float-valued).
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.
is_added	Tells if the pixel value is added to (`true`), or simply replace (`false`) the current image pixel(s).

Returns

A reference to the current image instance.

Note

- If specified coordinates are outside image bounds, no operations are performed.

8.1.4.146 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.

8.1.4.147 CImg<charT> value_string (const char *separator* = ',', const unsigned int *max_size* = 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

separator	A `char` character which specifies the separator between values in the returned C-string.
max_size	Maximum size of the returned image.

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`.

8.1.4.148 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).

8.1.4.149 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.

8.1.4.150 bool is_inf ( ) const

Test if image instance contains a 'inf' value.

Return `true`, if image instance contains a 'inf' value, and `false` otherwise.

8.1.4.151 bool is_nan ( ) const

Test if image instance contains a 'nan' value.

Return `true`, if image instance contains a 'nan' value, and `false` otherwise.

8.1.4.152 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.

8.1.4.153 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.

8.1.4.154 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.

8.1.4.155 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.

8.1.4.156 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.

8.1.4.157 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.

8.1.4.158 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.

8.1.4.159 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.

8.1.4.160 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.

8.1.4.161 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.

8.1.4.162 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.

8.1.4.163 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.

8.1.4.164 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.

8.1.4.165 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.

8.1.4.166 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.

8.1.4.167 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.

8.1.4.168 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.

8.1.4.169 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.

8.1.4.170 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.

8.1.4.171 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.

8.1.4.172 bool is_sameYZC (const CImg< t > & *img*) const

Test if image height, depth and spectrum are the same as that of another image.

Test if is_sameYZ(const CImg<t>&) const and is_sameC(const CImg<t>&) const are both verified.

8.1.4.173 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.

8.1.4.174 bool is_sameXYZC (const CImg< t > & *img*) const

Test if image width, height, depth and spectrum are the same as that of another image.

Test if is_sameXYZ(const CImg<t>&) const and is_sameC(const CImg<t>&) const are both verified.

8.1.4.175 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

x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	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`.

8.1.4.176 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

	pixel	Reference to pixel value to test.
out	*x*	X-coordinate of the pixel value, if test succeeds.
out	*y*	Y-coordinate of the pixel value, if test succeeds.
out	*z*	Z-coordinate of the pixel value, if test succeeds.
out	*c*	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);
}
```

8.1.4.177 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.

8.1.4.178 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.

8.1.4.179 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.

8.1.4.180 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.

8.1.4.181 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

img	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");
}
```

8.1.4.182 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

	primitives	List of primitives of the 3d object.
	colors	List of colors of the 3d object.
	opacities	List (or image) of opacities of the 3d object.
	full_check	Tells if full checking of the 3d object must be performed.
`out`	*error_message*	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.

8.1.4.183 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

	full_check	Tells if full checking of the instance must be performed.
out	*error_message*	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.

8.1.4.184 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();
```

8.1.4.185 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();
```

8.1.4.186 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.

8.1.4.187 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.

8.1.4.188 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.

8.1.4.189 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.

8.1.4.190 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.

8.1.4.191 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 $\mathrm{sign}(I_{(x,y,z,c)})$.

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.

8.1.4.192 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.

8.1.4.193 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.

8.1.4.194 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 $\mathrm{sinc}(I_{(x,y,z,c)})$.

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.

8.1.4.195 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.

8.1.4.196 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.

8.1.4.197 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.

8.1.4.198 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.

8.1.4.199 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 $\mathrm{acos}(I_{(x,y,z,c)})$.

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.

8.1.4.200 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 $\mathrm{asin}(I_{(x,y,z,c)})$.

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.

8.1.4.201 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 $\mathrm{atan}(I_{(x,y,z,c)})$.

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.

8.1.4.202 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 $\mathrm{atan2}(I_{(x,y,z,c)})$.

Parameters

img	Image whose pixel values specify the second argument of the `atan2()` 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();
```

8.1.4.203 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

img	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();
```

8.1.4.204 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.

8.1.4.205 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();
```

8.1.4.206 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.

8.1.4.207 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.

8.1.4.208 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.

8.1.4.209 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.

8.1.4.210 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.

8.1.4.211 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.

8.1.4.212 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.

8.1.4.213 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.

8.1.4.214 CImg<T>& min (const T *val*)

Pointwise min operator between instance image and a value.

Parameters

val	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 $\min(I_{(x,y,z,c)}, \mathrm{val})$.

8.1.4.215 CImg<T>& min (const CImg< t > & *img*)

Pointwise min operator between two images.

Parameters

img	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 $\min(I_{(x,y,z,c)}, \mathrm{img}_{(x,y,z,c)})$.

8.1.4.216 CImg<T>& min (const char *const *expression*)

Pointwise min operator between an image and an expression.

Parameters

expression	Math formula as a C-string.

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\min(I_{(x,y,z,c)}, \mathrm{expr}_{(x,y,z,c)})$.

8.1.4.217 CImg<T>& max (const T *val*)

Pointwise max operator between instance image and a value.

Parameters

val	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 $\max(I_{(x,y,z,c)}, \mathrm{val})$.

8.1.4.218 CImg<T>& max (const CImg< t > & *img*)

Pointwise max operator between two images.

Parameters

img	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 $\max(I_{(x,y,z,c)}, \mathrm{img}_{(x,y,z,c)})$.

8.1.4.219 CImg<T>& max (const char ∗const *expression*)

Pointwise max operator between an image and an expression.

Parameters

expression	Math formula as a C-string.

Note

Replace each pixel value $I_{(x,y,z,c)}$ of the image instance by $\max(I_{(x,y,z,c)}, \mathrm{expr}_{(x,y,z,c)})$.

8.1.4.220 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.

8.1.4.221 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.

8.1.4.222 T kth_smallest (const unsigned int *k*) const

Return the kth smallest pixel value.

Parameters

k	Rank of the search smallest element.

8.1.4.223 Tdouble variance (const unsigned int *variance_method* = 1) const

Return the variance of the pixel values.

Parameters

variance_-method	Method used to estimate the variance. Can be: • 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.

8.1.4.224 Tdouble variance_mean (const unsigned int *variance_method,* t & *mean*) const

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

Parameters

	variance_-method	Method used to estimate the variance (see variance(const unsigned int) const).
out	*mean*	Average pixel value.

8.1.4.225 Tdouble variance_noise (const unsigned int *variance_method* = 2) const

Return estimated variance of the noise.

Parameters

variance_-method	Method used to compute the variance (see variance(const unsigned int) const).

Note

Because of structures such as edges in images it is recommanded 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.

8.1.4.226 Tdouble MSE (const CImg< t > & *img*) const

Compute the MSE (Mean-Squared Error) between two images.

Parameters

img	Image used as the second argument of the MSE operator.

8.1.4.227 Tdouble PSNR (const CImg< t > & *img*, const Tdouble *max_value* = 255) const

Compute the PSNR (Peak Signal-to-Noise Ratio) between two images.

Parameters

img	Image used as the second argument of the PSNR operator.
max_value	Maximum theoretical value of the signal.

8.1.4.228 double eval (const char *const *expression*, const double *x* = 0, const double *y* = 0, const double *z* = 0, const double *c* = 0) const

Evaluate math formula.

Parameters

expression	Math formula, as a C-string.
x	Value of the pre-defined variable `x`.
y	Value of the pre-defined variable `y`.
z	Value of the pre-defined variable `z`.
c	Value of the pre-defined variable `c`.

8.1.4.229 CImg<doubleT> eval (const char *const *expression*, const CImg< t > & *xyzc*) const

Evaluate math formula on a set of variables.

Parameters

expression	Math formula, as a C-string.
xyzc	Set of values (x,y,z,c) used for the evaluation.

8.1.4.230 Tdouble magnitude (const int *magnitude_type* = 2) const

Compute norm of the image, viewed as a matrix.

Parameters

magnitude_type	Norm type. Can be: • `-1`: Linf-norm • `0`: L2-norm • `1`: L1-norm

8.1.4.231 Tdouble dot (const CImg< t > & *img*) const

Compute the dot product between instance and argument, viewed as matrices.

Parameters

img	Image used as a second argument of the dot product.

8.1.4.232 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

x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.

8.1.4.233 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

x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.

Note

- The spectrum() of the image must be a square.

8.1.4.234 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

x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.

8.1.4.235 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

vec	Vector to put on the instance image.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.

8.1.4.236 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

mat	Matrix to put on the instance image.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.

8.1.4.237 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

ten	Tensor to put on the instance image.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.

8.1.4.238 CImg<T>& vector ( )

Unroll pixel values along axis `y`.

Note

Equivalent to

```
unroll('y');
```

.

8.1.4.239 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.

8.1.4.240 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.

8.1.4.241 CImg<T>& sequence (const T *a0*, const T *a1*)

Fill image with a linear sequence of values.

Parameters

a0	Starting value of the sequence.
a1	Ending value of the sequence.

8.1.4.242 CImg<T>& transpose ( )

Transpose the image, viewed as a matrix.

Note

Equivalent to

```
permute_axes("yxzc");
```

8.1.4.243 CImg<T>& cross (const CImg< t > & *img*)

Compute the cross product between two `1x3` images, viewed as 3d vectors.

Parameters

img	Image used as the second argument of the cross product.

Note

The first argument of the cross product is `*this`.

8.1.4.244 CImg<T>& invert (const bool *use_LU* = `true`)

Invert the instance image, viewed as a matrix.

Parameters

use_LU	Choose the inverting algorithm. Can be: • `true`: LU-based matrix inversion. • `false`: SVD-based matrix inversion.

8.1.4.245 CImg<T>& solve (const CImg< t > & *A*)

Solve a system of linear equations.

Parameters

A	Matrix of the linear system.

Note

Solve `AX=B` where `B=*this`.

8.1.4.246 CImg<T>& solve_tridiagonal (const CImg< t > & *A*)

Solve a tridiagonal system of linear equations.

Parameters

A	Coefficients of the tridiagonal system. A is a tridiagonal matrix A = [b0,c0,0,...; a1,b1,c1,0,... ; ... ; ...,0,aN,bN], stored as a 3 columns matrix

Note

Solve AX=B where `B=*this`, using the Thomas algorithm.

8.1.4.247 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	*val*	Vector of the estimated eigenvalues, in decreasing order.
out	*vec*	Matrix of the estimated eigenvalues, sorted by columns.

8.1.4.248 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.

8.1.4.249 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	*val*	Vector of the estimated eigenvalues, in decreasing order.
out	*vec*	Matrix of the estimated eigenvalues, sorted by columns.

8.1.4.250 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.

8.1.4.251 CImg<T>& sort (CImg< t > & *permutations*, const bool *is_increasing* = `true`)

Sort pixel values and get sorting permutations.

Parameters

out	*permutations*	Permutation map used for the sorting.
	is_increasing	Tells if pixel values are sorted in an increasing (`true`) or decreasing (`false`) way.

8.1.4.252 CImg<T>& sort (const bool *is_increasing* = `true`, const char *axis* = `0`)

Sort pixel values.

Parameters

is_increasing	Tells if pixel values are sorted in an increasing (`true`) or decreasing (`false`) way.
axis	Tells if the value sorting must be done along a specific axis. Can be: • `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.

8.1.4.253 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	*U*	First matrix of the SVD product.
out	*S*	Coefficients of the second (diagonal) matrix of the SVD product. These coefficients are stored as a vector.
out	*V*	Third matrix of the SVD product.
	sorting	Tells if the diagonal coefficients are sorted (in decreasing order).
	max_iteration	Maximum number of iterations considered for the algorithm convergence.
	lambda	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)
```

8.1.4.254 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.

8.1.4.255 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

distance	An object having operator()(unsigned int i, unsigned int j) which returns distance between two nodes (i,j).
nb_nodes	Number of graph nodes.
starting_node	Indice of the starting node.
ending_node	Indice of the ending node (set to ~0U to ignore ending node).
previous_node	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.

8.1.4.256 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

starting_node	Indice of the starting node.
ending_node	Indice of the ending node.
previous_node	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.

8.1.4.257 static CImg<T> string (const char *const *str*, const bool *is_last_zero* = `true`) `[static]`

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

Parameters

str	input C-string to encode as an image.
is_last_zero	Tells if the ending `'0'` character appear in the resulting image.

8.1.4.258 static CImg<T> vector (const T & *a0*) `[static]`

Return a `1x1` image containing specified value.

Parameters

a0	First vector value.

8.1.4.259 static CImg<T> vector (const T & *a0*, const T & *a1*) `[static]`

Return a `1x2` image containing specified values.

Parameters

a0	First vector value.
a1	Second vector value.

8.1.4.260 static CImg<T> vector (const T & *a0*, const T & *a1*, const T & *a2*) [static]

Return a `1x3` image containing specified values.

Parameters

a0	First vector value.
a1	Second vector value.
a2	Third vector value.

8.1.4.261 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

a0	First vector value.
a1	Second vector value.
a2	Third vector value.
a3	Fourth vector value.

8.1.4.262 static CImg<T> matrix (const T & *a0*) [static]

Return a 1x1 matrix containing specified coefficients.

Parameters

a0	First matrix value.

Note

Equivalent to vector(const T&).

8.1.4.263 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

a0	First matrix value.
a1	Second matrix value.
a2	Third matrix value.
a3	Fourth matrix value.

8.1.4.264 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

a0	First matrix value.
a1	Second matrix value.
a2	Third matrix value.
a3	Fourth matrix value.
a4	Fifth matrix value.
a5	Sixth matrix value.
a6	Seventh matrix value.
a7	Eighth matrix value.
a8	Nineth matrix value.

8.1.4.265 static CImg<T> tensor (const T & *a0*) [static]

Return a 1x1 symmetric matrix containing specified coefficients.

Parameters

a0	First matrix value.

Note

Equivalent to vector(const T&).

8.1.4.266 static CImg<T> identity_matrix (const unsigned int *N*) [static]

Return a NxN identity matrix.

Parameters

N	Dimension of the matrix.

8.1.4.267 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

N	Size of the resulting vector.
a0	Starting value of the sequence.
a1	Ending value of the sequence.

8.1.4.268 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

x	X-coordinate of the rotation axis, or first quaternion coordinate.
y	Y-coordinate of the rotation axis, or second quaternion coordinate.

z	Z-coordinate of the rotation axis, or third quaternion coordinate.
w	Angle of the rotation axis, or fourth quaternion coordinate.
is_quaternion	Tell is the four arguments denotes a set { axis + angle } or a quaternion.

8.1.4.269 CImg<T>& fill (const T *val*)

Fill all pixel values with specified value.

Parameters

val	Fill value.

8.1.4.270 CImg<T>& fill (const T *val0*, const T *val1*)

Fill sequentially all pixel values with specified values.

Parameters

val0	First fill value.
val1	Second fill value.

8.1.4.271 CImg<T>& fill (const char ∗const *expression*, const bool *repeat_flag*)

Fill sequentially pixel values according to a given expression.

Parameters

expression	C-string describing a math formula, or a list of values.
repeat_flag	In case a list of values is provided, tells if this list must be repeated for the filling.

8.1.4.272 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

values	Image containing the values used for the filling.
repeat_values	In case there are less values than necessary in `values`, tells if these values must be repeated for the filling.

8.1.4.273 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

y	Y-coordinate of the filled column.
z	Z-coordinate of the filled column.
c	C-coordinate of the filled column.
a0	First fill value.

8.1.4.274 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.

Parameters

x	X-coordinate of the filled row.
z	Z-coordinate of the filled row.
c	C-coordinate of the filled row.
a0	First fill value.

8.1.4.275 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.

Parameters

x	X-coordinate of the filled slice.
y	Y-coordinate of the filled slice.
c	C-coordinate of the filled slice.
a0	First fill value.

8.1.4.276 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.

Parameters

x	X-coordinate of the filled channel.
y	Y-coordinate of the filled channel.
z	Z-coordinate of the filled channel.
a0	First filling value.

8.1.4.277 CImg<T>& discard (const T *value*)

Discard specified value in the image buffer.

Parameters

value	Value to discard.

Note

Discarded values will change the image geometry, so the resulting image is returned as a one-column vector.

8.1.4.278 CImg<T>& discard (const CImg< t > & *values*)

Discard specified sequence of values in the image buffer.

Parameters

values	Sequence of values to discard.

Note

Discarded values will change the image geometry, so the resulting image is returned as a one-column vector.

8.1.4.279 CImg<T>& rand (const T *val_min*, const T *val_max*)

Fill image with random values in specified range.

Parameters

val_min	Minimal random value.
val_max	Maximal random value.

Note

Random samples are following a uniform distribution.

8.1.4.280 CImg<T>& round (const double *y* = 1, const int *rounding_type* = 0)

Round pixel values.

Parameters

y	Rounding precision.
rounding_type	Rounding type. Can be: • -1: Backward. • 0: Nearest. • 1: Forward.

8.1.4.281 CImg<T>& noise (const double *sigma*, const unsigned int *noise_type* = 0)

Add random noise to pixel values.

Parameters

sigma	Amplitude of the random additive noise. If `sigma<0`, it stands for a percentage of the global value range.
noise_type	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)).display();
```

8.1.4.282 CImg<T>& normalize (const T *min_value*, const T *max_value*)

Linearly normalize pixel values.

Parameters

min_value	Minimum desired value of the resulting image.
max_value	Maximum desired value of the resulting image.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_normalize(160,220);
(img,res).display();
```

8.1.4.283 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)).display();
```

8.1.4.284 CImg<T>& norm (const int *norm_type* = 2)

Compute L2-norm of each multi-valued pixel of the image instance.

Parameters

norm_type	Type of computed vector norm (can be `0=Linf`, `1=L1` or `2=L2`).

Example

```
const CImg<float> img("reference.jpg"), res = img.get_norm();
(img,res.normalize(0,255)).display();
```

8.1.4.285 CImg<T>& cut (const T *min_value*, const T *max_value*)

Cut pixel values in specified range.

Parameters

min_value	Minimum desired value of the resulting image.
max_value	Maximum desired value of the resulting image.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_cut(160,220);
(img,res).display();
```

8.1.4.286 CImg<T>& quantize (const unsigned int *nb_levels*, const bool *keep_range* = true)

Uniformly quantize pixel values.

Parameters

nb_levels	Number of quantization levels.

keep_range	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).display();
```

8.1.4.287 CImg<T>& threshold (const T *value*, const bool *soft_threshold* = `false`, const bool *strict_threshold* = `false`)

Threshold pixel values.

Parameters

value	Threshold value
soft_threshold	Tells if soft thresholding must be applied (instead of hard one).
strict_threshold	Tells if threshold value is strict.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_threshold(128);
(img,res.normalize(0,255)).display();
```

8.1.4.288 CImg<T>& histogram (const unsigned int *nb_levels*, const T *min_value* = `(T) 0`, const T *max_value* = `(T) 0`)

Compute the histogram of pixel values.

Parameters

nb_levels	Number of desired histogram levels.
min_value	Minimum pixel value considered for the histogram computation. All pixel values lower than `min_value` will not be counted.
max_value	Maximum pixel value considered for the histogram computation. All pixel values higher than `max_value` will not be counted.

Note

- The histogram H of an image I is the 1d function where H(x) counts the number of occurences of the value x in the image I.
- If `min_value==max_value==0` (default behavior), the function first estimates the whole range of pixel values then uses it to compute the histogram.
- 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);
```

8.1.4.289 CImg<T>& equalize (const unsigned int *nb_levels*, const T *min_value* = `(T) 0`, const T *max_value* = `(T) 0`)

Equalize histogram of pixel values.

Parameters

nb_levels	Number of histogram levels used for the equalization.
min_value	Minimum pixel value considered for the histogram computation. All pixel values lower than `min_value` will not be counted.
max_value	Maximum pixel value considered for the histogram computation. All pixel values higher than `max_value` will not be counted.

Note

- If `min_value==max_value==0` (default behavior), the function first estimates the whole range of pixel values then uses it to equalize the histogram.

Example

```
const CImg<float> img("reference.jpg"), res = img.get_equalize(256);
(img,res).display();
```

8.1.4.290 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

colormap	Multi-valued colormap used as the basis for multi-valued pixel indexing.
dithering	Level of dithering (0=disable, 1=standard level).
map_indexes	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(col`

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();
```

8.1.4.291 CImg<T>& map (const CImg< t > & *colormap*)

Map predefined colormap on the scalar (indexed) image instance.

Parameters

colormap	Multi-valued colormap used for mapping the indexes.

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();
```

8.1.4.292 CImg<T>& label (const bool *is_high_connectivity* = `false`, const Tfloat *tolerance* = `0`)

Label connected components.

Parameters

is_high_-connectivity	Boolean that choose between 4(false)- or 8(true)-connectivity in 2d case, and between 6(false)- or 26(true)-connectivity in 3d case.
tolerance	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.

8.1.4.293 CImg<T>& label (const CImg< t > & *connectivity_mask*, const Tfloat *tolerance* = `0`)

Label connected components **[overloading]**.

Parameters

connectivity_-mask	Mask of the neighboring pixels.
tolerance	Tolerance used to determine if two neighboring pixels belong to the same region.

8.1.4.294 static const CImg<Tuchar>& default_LUT256 ( ) `[static]`

Return colormap *"default"*, containing 256 colors entries in RGB.

Returns

The following `256x1x1x3` colormap is returned:

8.1.4.295 static const CImg<Tuchar>& HSV_LUT256 ( ) `[static]`

Return colormap *"HSV"*, containing 256 colors entries in RGB.

Returns

The following `256x1x1x3` colormap is returned:

8.1.4.296 static const CImg<Tuchar>& lines_LUT256 ( ) `[static]`

Return colormap *"lines"*, containing 256 colors entries in RGB.

Returns

The following `256x1x1x3` colormap is returned:

8.1.4.297 static const CImg<Tuchar>& hot_LUT256 ( ) `[static]`

Return colormap *"hot"*, containing 256 colors entries in RGB.

Returns

The following `256x1x1x3` colormap is returned:

8.1.4.298 static const CImg<Tuchar>& cool_LUT256 ( ) [static]

Return colormap *"cool"*, containing 256 colors entries in RGB.

Returns

The following `256x1x1x3` colormap is returned:

8.1.4.299 static const CImg<Tuchar>& jet_LUT256 ( ) [static]

Return colormap *"jet"*, containing 256 colors entries in RGB.

Returns

The following `256x1x1x3` colormap is returned:

8.1.4.300 static const CImg<Tuchar>& flag_LUT256 ( ) [static]

Return colormap *"flag"*, containing 256 colors entries in RGB.

Returns

The following `256x1x1x3` colormap is returned:

8.1.4.301 static const CImg<Tuchar>& cube_LUT256 ( ) [static]

Return colormap *"cube"*, containing 256 colors entries in RGB.

Returns

The following `256x1x1x3` colormap is returned:

8.1.4.302 CImg<T>& RGBtoXYZ ( )

Convert pixel values from RGB to XYZ_709 color spaces.

Note

Uses the standard D65 white point.

8.1.4.303 CImg<T>& RGBtoBayer ( )

Convert RGB color image to a Bayer-coded scalar image.

Note

First (upper-left) pixel if the red component of the pixel color.

8.1.4.304 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

size_x	Number of columns (new size along the X-axis).
size_y	Number of rows (new size along the Y-axis).
size_z	Number of slices (new size along the Z-axis).
size_c	Number of vector-channels (new size along the C-axis).
interpolation_-type	Method of interpolation: • -1 = no interpolation: raw memory resizing. • 0 = no interpolation: additional space is filled according to `boundary_-conditions`. • 1 = nearest-neighbor interpolation. • 2 = moving average interpolation. • 3 = linear interpolation. • 4 = grid interpolation. • 5 = cubic interpolation. • 6 = lanczos interpolation.
boundary_-conditions	Border condition type.
centering_x	Set centering type (only if `interpolation_type=0`).
centering_y	Set centering type (only if `interpolation_type=0`).
centering_z	Set centering type (only if `interpolation_type=0`).
centering_c	Set centering type (only if `interpolation_type=0`).

Note

If pd[x,y,z,v]<0, it corresponds to a percentage of the original size (the default value is -100).

8.1.4.305 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.

Parameters

src	Reference image used for dimensions.
interpolation_-type	Interpolation method.
boundary_-conditions	Boundary conditions.
centering_x	Set centering type (only if `interpolation_type=0`).
centering_y	Set centering type (only if `interpolation_type=0`).
centering_z	Set centering type (only if `interpolation_type=0`).
centering_c	Set centering type (only if `interpolation_type=0`).

8.1.4.306 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

disp	Reference display window used for dimensions.
interpolation_-type	Interpolation method.
boundary_-conditions	Boundary conditions.
centering_x	Set centering type (only if `interpolation_type=0`).
centering_y	Set centering type (only if `interpolation_type=0`).
centering_z	Set centering type (only if `interpolation_type=0`).
centering_c	Set centering type (only if `interpolation_type=0`).

8.1.4.307 CImg<T>& resize_doubleXY ( )

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

Note

Use anisotropic upscaling algorithm `described here`.

8.1.4.308 CImg<T>& resize_tripleXY ( )

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

Note

Use anisotropic upscaling algorithm `described here`.

8.1.4.309 CImg<T>& mirror (const char *axis*)

Mirror image content along specified axis.

Parameters

axis	Mirror axis

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

Mirror image content along specified axes.

Parameters

axes	Mirror axes, as a C-string.

Note

`axes` may contains multiple character, e.g. `"xyz"`

8.1.4.311 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

delta_x	Amount of displacement along the X-axis.
delta_y	Amount of displacement along the Y-axis.
delta_z	Amount of displacement along the Z-axis.
delta_c	Amount of displacement along the C-axis.
boundary_-conditions	Border condition. • `boundary_conditions` can be: – 0: Zero border condition (Dirichlet). – 1: Nearest neighbors (Neumann). – 2: Repeat Pattern (Fourier style).

8.1.4.312 CImg<T>& permute_axes (const char ∗const *order*)

Permute axes order.

Parameters

order	Axes permutations, as a C-string of 4 characters. This function permutes image content regarding the specified axes permutation.

8.1.4.313 CImg<T>& unroll (const char *axis*)

Unroll pixel values along specified axis.

Parameters

axis	Unroll axis (can be `'x'`, `'y'`, `'z'` or c 'c').

8.1.4.314 CImg<T>& rotate (const float *angle*, const unsigned int *interpolation* = `1`, const unsigned int *boundary* = `0`)

Rotate image with arbitrary angle.

Parameters

angle	Rotation angle, in degrees.
interpolation	Type of interpolation. Can be `{ 0=nearest | 1=linear | 2=cubic }`.
boundary	Boundary conditions. Can be `{ 0=dirichlet | 1=neumann | 2=cyclic }`.

Note

Most of the time, size of the image is modified.

8.1.4.315 CImg<T>& rotate (const float *angle*, const float *cx*, const float *cy*, const float *zoom*, const unsigned int *interpolation* = `1`, const unsigned int *boundary* = `3`)

Rotate image with arbitrary angle, around a center point.

Parameters

angle	Rotation angle, in degrees.
cx	X-coordinate of the rotation center.
cy	Y-coordinate of the rotation center.
zoom	Zoom factor.
boundary_-conditions	Boundary conditions. Can be `{ 0=dirichlet | 1=neumann | 2=cyclic }`.
interpolation_-type	Type of interpolation. Can be `{ 0=nearest | 1=linear | 2=cubic }`.

8.1.4.316 CImg<T>& warp (const CImg< t > & *warp*, const bool *is_relative* = `false`, const unsigned int *interpolation* = `1`, const unsigned int *boundary_conditions* = `0`)

Warp image content by a warping field.

Parameters

warp	Warping field.
is_relative	Tells if warping field gives absolute or relative warping coordinates.
interpolation	Can be `{ 0=nearest | 1=linear | 2=cubic }`.
boundary_-conditions	Boundary conditions. Can be `{ 0=dirichlet | 1=neumann | 2=cyclic }`.

8.1.4.317 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.

8.1.4.318 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.

8.1.4.319 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.

8.1.4.320 CImg<T> get_column (const int *x0*) const

Return specified image column.

Parameters

x0	Image column.

8.1.4.321 CImg<T>& columns (const int *x0,* const int *x1*)

Return specified range of image columns.

Parameters

x0	Starting image column.
x1	Ending image column.

8.1.4.322 CImg<T>& row (const int *y0*)

Return specified image row **[in-place version]**.

Parameters

y0	Image row.

8.1.4.323 CImg<T> get_rows (const int *y0,* const int *y1*) const

Return specified range of image rows.

Parameters

y0	Starting image row.
y1	Ending image row.

8.1.4.324 CImg<T> get_slice (const int *z0*) const

Return specified image slice.

Parameters

z0	Image slice.

8.1.4.325 CImg<T> get_slices (const int *z0,* const int *z1*) const

Return specified range of image slices.

Parameters

z0	Starting image slice.
z1	Ending image slice.

8.1.4.326 CImg<T> get_channel (const int *c0*) const

Return specified image channel.

Parameters

c0	Image channel.

8.1.4.327 CImg<T> get_channels (const int *c0*, const int *c1*) const

Return specified range of image channels.

Parameters

c0	Starting image channel.
c1	Ending image channel.

8.1.4.328 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

func	Vector field function.
x	X-coordinate of the starting point of the streamline.
y	Y-coordinate of the starting point of the streamline.
z	Z-coordinate of the starting point of the streamline.
L	Streamline length.
dl	Streamline length increment.
interpolation_type	Type of interpolation. Can be `{ 0=nearest int | 1=linear | 2=2nd-order RK | 3=4th-order RK. }`.
is_backward_tracking	Tells if the streamline is estimated forward or backward.
is_oriented_only	Tells if the direction of the vectors must be ignored.
x0	X-coordinate of the first bounding-box vertex.
y0	Y-coordinate of the first bounding-box vertex.
z0	Z-coordinate of the first bounding-box vertex.
x1	X-coordinate of the second bounding-box vertex.
y1	Y-coordinate of the second bounding-box vertex.
z1	Z-coordinate of the second bounding-box vertex.

8.1.4.329 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

x0	X-coordinate of the starting pixel.
x1	X-coordinate of the ending pixel.
y0	Y-coordinate.
z0	Z-coordinate.
c0	C-coordinate.

8.1.4.330 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

y0	Y-coordinate of the starting row.
y1	Y-coordinate of the ending row.
z0	Z-coordinate.
c0	C-coordinate.

8.1.4.331 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

y0	Y-coordinate.
z0	Z-coordinate.
c0	C-coordinate.

8.1.4.332 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

z0	Z-coordinate of the starting slice.
z1	Z-coordinate of the ending slice.
c0	C-coordinate.

8.1.4.333 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

z0	Z-coordinate.
c0	C-coordinate.

8.1.4.334 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

c0	C-coordinate of the starting channel.
c1	C-coordinate of the ending channel.

8.1.4.335 CImg<T> get_shared_channel (const unsigned int *c0*)

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

Parameters

c0	C-coordinate.

8.1.4.336 CImgList<T> get_split (const char *axis,* const int *nb* = 0) const

Split image into a list along specified axis.

Parameters

axis	Splitting axis. Can be `{ 'x'	'y'	'z'	'c' }`.
nb	Number of splitted parts.

Note

- If `nb==0`, there are as much splitted parts as the image size 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.

8.1.4.337 CImgList<T> get_split (const T *value,* const bool *keep_values,* const bool *is_shared*) const

Split image into a list of one-column vectors, according to a specified splitting value.

Parameters

value	Splitting value.
keep_values	Tells if the splitting value must be kept in the splitted blocs.
is_shared	Tells if the splitted blocs have shared memory buffers.

8.1.4.338 CImgList<T> get_split (const CImg< t > & *values,* const bool *keep_values,* const bool *is_shared*) const

Split image into a list of one-column vectors, according to a specified splitting value sequence.

Parameters

values	Splitting value sequence.
keep_values	Tells if the splitting sequence must be kept in the splitted blocs.
is_shared	Tells if the splitted blocs have shared memory buffers.

8.1.4.339 CImg<T>& append (const CImg< t > & *img,* const char *axis* = 'x', const float *align* = 0)

Append two images along specified axis.

Parameters

img	Image to append with instance image.
axis	Appending axis. Can be `{ 'x'	'y'	'z'	'c' }`.
align	Append alignment in [0,1].

8.1.4.340 CImg<T>& correlate (const CImg< t > & *mask*, const unsigned int *boundary_conditions* = `1`, const bool *is_normalized* = `false`)

Correlate image by a mask.

Parameters

mask	= the correlation kernel.
boundary_-conditions	= the border condition type (0=zero, 1=dirichlet)
is_normalized	= enable local normalization.

Note

- The correlation of the image instance `*this` by the mask `mask` is defined to be: res(x,y,z) = sum_{i,j,k} (*this)(x+i,y+j,z+k)*mask(i,j,k).

8.1.4.341 CImg<T>& convolve (const CImg< t > & *mask*, const unsigned int *boundary_conditions* = `1`, const bool *is_normalized* = `false`)

Convolve image by a mask.

Parameters

mask	= the correlation kernel.
boundary_-conditions	= the border condition type (0=zero, 1=dirichlet)
is_normalized	= 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)

8.1.4.342 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

mask	Structuring element.
boundary_-conditions	Boundary conditions.
is_normalized	Tells if the erosion is locally normalized.

8.1.4.343 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

sx	Width of the structuring element.
sy	Height of the structuring element.
sz	Depth of the structuring element.

8.1.4.344 CImg<T>& erode (const unsigned int *s*)

Erode the image by a square structuring element of specified size.

Parameters

s	Size of the structuring element.

8.1.4.345 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

mask	Structuring element.
boundary_-conditions	Boundary conditions.
is_normalized	Tells if the erosion is locally normalized.

8.1.4.346 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

sx	Width of the structuring element.
sy	Height of the structuring element.
sz	Depth of the structuring element.

8.1.4.347 CImg<T>& dilate (const unsigned int *s*)

Dilate image by a square structuring element of specified size.

Parameters

s	Size of the structuring element.

8.1.4.348 CImg<T>& watershed (const CImg< t > & *priority*, const bool *fill_lines* = `true`)

Compute watershed transform.

Parameters

priority	Priority map.

fill_lines	Tells if watershed lines must be filled or not.

Note

Non-zero values of the instance instance are propagated to zero-valued ones according to specified the priority map.

8.1.4.349 CImg<T>& deriche (const float *sigma*, const int *order* = `0`, const char *axis* = `'x'`, const bool *boundary_conditions* = `true`)

Apply recursive Deriche filter.

Parameters

sigma	Standard deviation of the filter.
order	Order of the filter. Can be `{ 0=smooth-filter | 1=1st-derivative | 2=2nd-derivative }`.
axis	Axis along which the filter is computed. Can be `{ 'x' | 'y' | 'z' | 'c' }`.
boundary_-conditions	Boundary conditions. Can be `{ 0=dirichlet | 1=neumann }`.

8.1.4.350 CImg<T>& vanvliet (const float *sigma*, const int *order*, const char *axis* = `'x'`, const bool *boundary_conditions* = `true`)

Van Vliet recursive Gaussian filter.

Parameters

sigma	standard deviation of the Gaussian filter
order	the order of the filter 0,1,2,3
axis	Axis along which the filter is computed. Can be `{ 'x' | 'y' | 'z' | 'c' }`.
boundary_-conditions	Boundary conditions. Can be `{ 0=dirichlet | 1=neumann }`.

Note

dirichlet boundary condition has a strange behavior

Ian T. Young, Lucas J. van Vliet, Recursive implementation of the Gaussian filter, Signal Processing, Volume 44, Issue 2, June 1995, Pages 139-151,

8.1.4.351 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

sigma_x	Standard deviation of the blur, along the X-axis.
sigma_y	Standard deviation of the blur, along the Y-axis.
sigma_z	Standard deviation of the blur, along the Z-axis.
boundary_-conditions	Boundary conditions. Can be `{ false=dirichlet | true=neumann }`.

is_gaussian	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().

8.1.4.352 CImg<T>& blur (const float *sigma*, const bool *boundary_conditions* = `true`, const bool *is_gaussian* = `false`)

Blur image isotropically.

Parameters

sigma	Standard deviation of the blur.
boundary_-conditions	Boundary conditions. Can be `{ 0=dirichlet	1=neumann }`.a

See Also

deriche(), vanvliet().

8.1.4.353 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

G	Field of square roots of diffusion tensors/vectors used to drive the smoothing.
amplitude	Amplitude of the smoothing.
dl	Spatial discretization.
da	Angular discretization.
gauss_prec	Precision of the diffusion process.
interpolation_-type	Interpolation scheme. Can be `{ 0=nearest-neighbor	1=linear	2=Runge-Kutta }`.
is_fast_approx	Tells if a fast approximation of the gaussian function is used or not.

8.1.4.354 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

amplitude	Amplitude of the smoothing.

sharpness	Sharpness.
anisotropy	Anisotropy.
alpha	Standard deviation of the gradient blur.
sigma	Standard deviation of the structure tensor blur.
dl	Spatial discretization.
da	Angular discretization.
gauss_prec	Precision of the diffusion process.
interpolation_-type	Interpolation scheme. Can be `{ 0=nearest-neighbor | 1=linear | 2=Runge-Kutta }`.
is_fast_approx	Tells if a fast approximation of the gaussian function is used or not.

8.1.4.355 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 int *bgrid_x*, const int *bgrid_y*, const int *bgrid_z*, const int *bgrid_r*, const bool *interpolation_type* = `true`)

Blur image, with the joint bilateral filter.

Parameters

guide	Image used to model the smoothing weights.
sigma_x	Amount of blur along the X-axis.
sigma_y	Amount of blur along the Y-axis.
sigma_z	Amount of blur along the Z-axis.
sigma_r	Amount of blur along the value axis.
bgrid_x	Size of the bilateral grid along the X-axis.
bgrid_y	Size of the bilateral grid along the Y-axis.
bgrid_z	Size of the bilateral grid along the Z-axis.
bgrid_r	Size of the bilateral grid along the value axis.
interpolation_-type	Use interpolation for image slicing.

Note

This algorithm uses the optimisation technique proposed by S. Paris and F. Durand, in ECCV'2006 (extended for 3d volumetric images).

8.1.4.356 CImg<T>& blur_bilateral (const CImg< t > & *guide*, const float *sigma_s*, const float *sigma_r*, const int *bgrid_s* = `-33`, const int *bgrid_r* = `32`, const bool *interpolation_type* = `true`)

Blur image using the joint bilateral filter.

Parameters

guide	Image used to model the smoothing weights.
sigma_s	Amount of blur along the XYZ-axes.
sigma_r	Amount of blur along the value axis.
bgrid_s	Size of the bilateral grid along the XYZ-axes.
bgrid_r	Size of the bilateral grid along the value axis.
interpolation_-type	Use interpolation for image slicing.

8.1.4.357 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

sigma_s	Amount of blur along the XYZ-axes.
sigma_p	Amount of blur along the value axis.
patch_size	Size of the patchs.
lookup_size	Size of the window to search similar patchs.
smoothness	Smoothness for the patch comparison.
is_fast_approx	Tells if a fast approximation of the gaussian function is used or not.

8.1.4.358 CImg<T>& blur_median (const unsigned int *n*)

Blur image with the median filter.

Parameters

n	Size of the median filter.

8.1.4.359 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

amplitude	Sharpening amplitude
sharpen_type	Select sharpening method. Can be `{ false=inverse diffusion | true=shock filters }`.
edge	Edge threshold (shock filters only).
alpha	Gradient smoothness (shock filters only).
sigma	Tensor smoothness (shock filters only).

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

Return image gradient.

Parameters

axes	Axes considered for the gradient computation, as a C-string (e.g "xy").
scheme	= Numerical scheme used for the gradient computation: • -1 = Backward finite differences • 0 = Centered finite differences • 1 = Forward finite differences • 2 = Using Sobel masks • 3 = Using rotation invariant masks • 4 = Using Deriche recusrsive filter. • 5 = Using Van Vliet recusrsive filter.

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

Return image hessian.

Parameters

axes	Axes considered for the hessian computation, as a C-string (e.g "xy").

8.1.4.362 CImg<T>& structure_tensors (const unsigned int *scheme* = `2`)

Compute the structure tensor field of an image.

Parameters

scheme	Numerical scheme. Can be `{ 0=central | 1=fwd/bwd1 | 2=fwd/bwd2 }`

8.1.4.363 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.

Parameters

sharpness	Sharpness
anisotropy	Anisotropy
alpha	Standard deviation of the gradient blur.
sigma	Standard deviation of the structure tensor blur.
is_sqrt	Tells if the square root of the tensor field is computed instead.

8.1.4.364 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`)

Estimate displacement field between two images.

Parameters

source	Reference image.
smoothness	Smoothness of estimated displacement field.
precision	Precision required for algorithm convergence.
nb_scales	Number of scales used to estimate the displacement field.
iteration_max	Maximum number of iterations allowed for one scale.
is_backward	If false, match I2(X+U(X)) = I1(X), else match I2(X) = I1(X-U(X)).

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

Compute Euclidean distance function to a specified value.

Parameters

value	Reference value.
metric	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

value	Reference value.
metric_mask	Metric mask.

Note

The algorithm code has been initially proposed by A. Meijster, and modified by D. Tschumperlé.

8.1.4.367 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).

Parameters

value	Reference value.
metric	Field of distance potentials.
is_high_-connectivity	Tells if the algorithm uses low or high connectivity.

8.1.4.368 CImg& 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).

Parameters

value	Reference value.
metric	Field of distance potentials.

8.1.4.369 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.

Parameters

nb_iterations	Number of PDE iterations.
band_size	Size of the narrow band.
time_step	Time step of the PDE iterations.

8.1.4.370 CImg<T>& haar (const char *axis*, const bool *invert* = `false`, const unsigned int *nb_scales* = `1`)

Compute Haar multiscale wavelet transform.

Parameters

axis	Axis considered for the transform.

invert	Set inverse of direct transform.
nb_scales	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

invert	Set inverse of direct transform.
nb_scales	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

axis	Axis along which the FFT is computed.
is_invert	Tells if the forward (`false`) or inverse (`true`) 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

`in,out`	*real*	Real part of the pixel values.
`in,out`	*imag*	Imaginary part of the pixel values.
	axis	Axis along which the FFT is computed.
	is_invert	Tells if the forward (`false`) or inverse (`true`) 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

`in,out`	*real*	Real part of the pixel values.
`in,out`	*imag*	Imaginary part of the pixel values.
	is_invert	Tells if the forward (`false`) or inverse (`true`) FFT is computed.
	nb_threads	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

tx	X-coordinate of the 3d displacement vector.
ty	Y-coordinate of the 3d displacement vector.
tz	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

sx	Width of the 3d object's bounding box.
sy	Height of the 3d object's bounding box.
sz	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

`in,out`	*primitives*	Primitives data of the current 3d object.
	obj_vertices	Vertices data of the additional 3d object.
	obj_primitives	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`>::empty`()`) const

Texturize primitives of a 3d object.

Parameters

`in,out`	*primitives*	Primitives data of the 3d object.
`in,out`	*colors*	Colors data of the 3d object.
	texture	Texture image to map to 3d object.
	coords	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	*primitives*	The returned list of the 3d object primitives (template type *tf* should be at least *unsigned int*).
out	*colors*	The returned list of the 3d object colors.
	elevation	The input elevation map.

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;
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	*primitives*	Primitives data of the returned 3d object.
out	*colors*	Colors data of the returned 3d object.
	x0	X-coordinate of the projection point.
	y0	Y-coordinate of the projection point.
	z0	Z-coordinate of the projection point.
	normalize_colors	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	*primitives*	The returned list of the 3d object primitives (template type *tf* should be at least *unsigned int*).
	isovalue	The returned list of the 3d object colors.
	size_x	The number of subdivisions along the X-axis.
	size_y	The number of subdisivions 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	*primitives*	The returned list of the 3d object primitives (template type *tf* should be at least *unsigned int*).
	isovalue	The returned list of the 3d object colors.
	size_x	Number of subdivisions along the X-axis.
	size_y	Number of subdisivions along the Y-axis.
	size_z	Number of subdisivions 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	*primitives*	Primitives data of the resulting 3d object.
	func	Elevation function. Is of type `float (*func)(const float x,const float y)`.
	x0	X-coordinate of the starting point.
	y0	Y-coordinate of the starting point.
	x1	X-coordinate of the ending point.
	y1	Y-coordinate of the ending point.
	size_x	Resolution of the function along the X-axis.
	size_y	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	*primitives*	Primitives data of the resulting 3d object.
	func	Elevation function. Is of type `float (*func)(const float x,const float y)`.
	isovalue	Isovalue to extract from function.
	x0	X-coordinate of the starting point.
	y0	Y-coordinate of the starting point.
	x1	X-coordinate of the ending point.
	y1	Y-coordinate of the ending point.
	size_x	Resolution of the function along the X-axis.

	size_y	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	*primitives*	Primitives data of the resulting 3d object.
	func	Implicit function. Is of type `float (*func)(const float x, const float y, const float z).`
	isovalue	Isovalue to extract.
	x0	X-coordinate of the starting point.
	y0	Y-coordinate of the starting point.
	z0	Z-coordinate of the starting point.
	x1	X-coordinate of the ending point.
	y1	Y-coordinate of the ending point.
	z1	Z-coordinate of the ending point.
	size_x	Resolution of the elevation function along the X-axis.
	size_y	Resolution of the elevation function along the Y-axis.
	size_z	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	*primitives*	The returned list of the 3d object primitives (template type *tf* should be at least *unsigned int*).
	size_x	The width of the box (dimension along the X-axis).
	size_y	The height of the box (dimension along the Y-axis).
	size_z	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	*primitives*	The returned list of the 3d object primitives (template type *tf* should be at least *unsigned int*).
	radius	The radius of the cone basis.
	size_z	The cone's height.
	subdivisions	The number of basis angular subdivisions.

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>::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	*primitives*	The returned list of the 3d object primitives (template type *tf* should be at least *unsigned int*).
	radius	The radius of the cylinder basis.
	size_z	The cylinder's height.
	subdivisions	The number of basis angular subdivisions.

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>::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	*primitives*	The returned list of the 3d object primitives (template type *tf* should be at least *unsigned int*).
	radius1	The large radius.
	radius2	The small radius.
	subdivisions1	The number of angular subdivisions for the large radius.
	subdivisions2	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<=i<=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	*primitives*	The returned list of the 3d object primitives (template type *tf* should be at least *unsigned int*).
	size_x	The width of the plane (dimension along the X-axis).
	size_y	The height of the plane (dimensions along the Y-axis).
	subdivisions_x	The number of planar subdivisions along the X-axis.
	subdivisions_y	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	*primitives*	The returned list of the 3d object primitives (template type *tf* should be at least *unsigned int*).
	radius	The radius of the sphere (dimension along the X-axis).
	subdivisions	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<floatT> ellipsoid3d (CImgList< tf > & *primitives*, const CImg< t > & *tensor*, const unsigned int *subdivisions* = 3) [static]

Generate a 3d ellipsoid.

Parameters

out	*primitives*	The returned list of the 3d object primitives (template type *tf* should be at least *unsigned int*).
	tensor	The tensor which gives the shape and size of the ellipsoid.
	subdivisions	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

primitives	Primitives data of the 3d object.
colors	Colors data of the 3d object.
opacities	Opacities data of the 3d object.
full_check	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	*primitives*	Primitives data of the 3d object.
out	*colors*	Colors data of the 3d object.
out	*opacities*	Opacities data of the 3d object.
	full_check	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

x0	X-coordinate of the point.
y0	Y-coordinate of the point.
z0	Z-coordinate of the point.
color	Pointer to spectrum() consecutive values, defining the drawing color.
opacity	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

points	Image of vertices coordinates.
color	Pointer to `spectrum()` consecutive values, defining the drawing color.
opacity	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

x0	X-coordinate of the starting line point.
y0	Y-coordinate of the starting line point.
x1	X-coordinate of the ending line point.
y1	Y-coordinate of the ending line point.
color	Pointer to `spectrum()` consecutive values of type `T`, defining the drawing color.
opacity	Drawing opacity.
pattern	An integer whose bits describe the line pattern.
init_hatch	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< 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.

Parameters

zbuffer	Zbuffer image.

x0	X-coordinate of the starting point.
y0	Y-coordinate of the starting point.
z0	Z-coordinate of the starting point
x1	X-coordinate of the ending point.
y1	Y-coordinate of the ending point.
z1	Z-coordinate of the ending point.
color	Pointer to `spectrum()` consecutive values of type `T`, defining the drawing color.
opacity	Drawing opacity.
pattern	An integer whose bits describe the line pattern.
init_hatch	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

x0	X-coordinate of the starting point.
y0	Y-coordinate of the starting point.
z0	Z-coordinate of the starting point
x1	X-coordinate of the ending point.
y1	Y-coordinate of the ending point.
z1	Z-coordinate of the ending point.
color	Pointer to `spectrum()` consecutive values of type `T`, defining the drawing color.
opacity	Drawing opacity.
pattern	An integer whose bits describe the line pattern.
init_hatch	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

x0	X-coordinate of the starting line point.
y0	Y-coordinate of the starting line point.
x1	X-coordinate of the ending line point.
y1	Y-coordinate of the ending line point.
texture	Texture image defining the pixel colors.
tx0	X-coordinate of the starting texture point.
ty0	Y-coordinate of the starting texture point.
tx1	X-coordinate of the ending texture point.
ty1	Y-coordinate of the ending texture point.
opacity	Drawing opacity.
pattern	An integer whose bits describe the line pattern.
init_hatch	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

x0	X-coordinate of the starting point.
y0	Y-coordinate of the starting point.
z0	Z-coordinate of the starting point
x1	X-coordinate of the ending point.
y1	Y-coordinate of the ending point.
z1	Z-coordinate of the ending point.
texture	Texture image defining the pixel colors.
tx0	X-coordinate of the starting texture point.
ty0	Y-coordinate of the starting texture point.
tx1	X-coordinate of the ending texture point.
ty1	Y-coordinate of the ending texture point.
opacity	Drawing opacity.
pattern	An integer whose bits describe the line pattern.
init_hatch	Tells if the hash variable must be reinitialized.

8.1.4.403 CImg<T>& draw_line (CImg< tz > & *zbuffer*, 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 and z-buffering.

Parameters

zbuffer	Z-buffer image.
x0	X-coordinate of the starting point.
y0	Y-coordinate of the starting point.
z0	Z-coordinate of the starting point
x1	X-coordinate of the ending point.
y1	Y-coordinate of the ending point.
z1	Z-coordinate of the ending point.
texture	Texture image defining the pixel colors.
tx0	X-coordinate of the starting texture point.
ty0	Y-coordinate of the starting texture point.
tx1	X-coordinate of the ending texture point.
ty1	Y-coordinate of the ending texture point.
opacity	Drawing opacity.
pattern	An integer whose bits describe the line pattern.
init_hatch	Tells if the hash variable must be reinitialized.

8.1.4.404 CImg<T>& draw_line (const CImg< 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

points	Coordinates of vertices, stored as a list of vectors.
color	Pointer to `spectrum()` consecutive values of type `T`, defining the drawing color.
opacity	Drawing opacity.
pattern	An integer whose bits describe the line pattern.
init_hatch	If set to true, init hatch motif.

Note

- This function uses several call to the single CImg::draw_line() procedure, depending on the vectors size in `points`.

8.1.4.405 CImg<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

x0	X-coordinate of the starting arrow point (tail).
y0	Y-coordinate of the starting arrow point (tail).
x1	X-coordinate of the ending arrow point (head).
y1	Y-coordinate of the ending arrow point (head).
color	Pointer to `spectrum()` consecutive values of type `T`, defining the drawing color.
angle	Aperture angle of the arrow head.
length	Length of the arrow head. If negative, describes a percentage of the arrow length.
opacity	Drawing opacity.
pattern	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

x0	X-coordinate of the starting curve point
y0	Y-coordinate of the starting curve point
u0	X-coordinate of the starting velocity
v0	Y-coordinate of the starting velocity
x1	X-coordinate of the ending curve point
y1	Y-coordinate of the ending curve point
u1	X-coordinate of the ending velocity
v1	Y-coordinate of the ending velocity
color	Pointer to `spectrum()` consecutive values of type `T`, defining the drawing color.
precision	Curve drawing precision.
opacity	Drawing opacity.
pattern	An integer whose bits describe the line pattern.
init_hatch	If `true`, 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 { (`x0`,`y0`), (`xa`,`ya`), (`xb`,`yb`), (`x1`,`y1`) } where (`x0`,`y0`) is the starting point, (`x1`,`y1`) is the ending point and (`xa`,`ya`), (`xb`,`yb`) are two *control* points. The starting and ending velocities (`u0`,`v0`) and (`u1`,`v1`) can be deduced easily from the control points as `u0` = (`xa` - `x0`), `v0` = (`ya` - `y0`), `u1` = (`x1` - `xb`) and `v1` = (`y1` - `yb`).

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

x0	X-coordinate of the starting curve point
y0	Y-coordinate of the starting curve point
u0	X-coordinate of the starting velocity
v0	Y-coordinate of the starting velocity
x1	X-coordinate of the ending curve point
y1	Y-coordinate of the ending curve point
u1	X-coordinate of the ending velocity
v1	Y-coordinate of the ending velocity
texture	Texture image defining line pixel colors.
tx0	X-coordinate of the starting texture point.
ty0	Y-coordinate of the starting texture point.
tx1	X-coordinate of the ending texture point.
ty1	Y-coordinate of the ending texture point.
precision	Curve drawing precision.
opacity	Drawing opacity.
pattern	An integer whose bits describe the line pattern.
init_hatch	if `true`, 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

points	Vertices data.
tangents	Tangents data.
color	Pointer to `spectrum()` consecutive values of type `T`, defining the drawing color.
opacity	Drawing opacity.
is_closed_set	Tells if the drawn spline set is closed.
precision	Precision of the drawing.
pattern	An integer whose bits describe the line pattern.
init_hatch	If `true`, init hatch motif.

8.1.4.410 CImg<T>& draw_spline (const CImg< 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 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 *opacity* = `1`)

Draw a filled 2d triangle.

Parameters

x0	X-coordinate of the first vertex.
y0	Y-coordinate of the first vertex.
x1	X-coordinate of the second vertex.
y1	Y-coordinate of the second vertex.
x2	X-coordinate of the third vertex.
y2	Y-coordinate of the third vertex.
color	Pointer to `spectrum()` consecutive values of type `T`, defining the drawing color.
opacity	Drawing opacity.

8.1.4.412 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 *opacity*, const unsigned int *pattern*)

Draw a outlined 2d triangle.

Parameters

x0	X-coordinate of the first vertex.
y0	Y-coordinate of the first vertex.
x1	X-coordinate of the second vertex.
y1	Y-coordinate of the second vertex.
x2	X-coordinate of the third vertex.
y2	Y-coordinate of the third vertex.
color	Pointer to `spectrum()` consecutive values of type `T`, defining the drawing color.
opacity	Drawing opacity.
pattern	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

zbuffer	Z-buffer image.
x0	X-coordinate of the first vertex.
y0	Y-coordinate of the first vertex.
z0	Z-coordinate of the first vertex.
x1	X-coordinate of the second vertex.
y1	Y-coordinate of the second vertex.
z1	Z-coordinate of the second vertex.
x2	X-coordinate of the third vertex.
y2	Y-coordinate of the third vertex.
z2	Z-coordinate of the third vertex.
color	Pointer to `spectrum()` consecutive values of type `T`, defining the drawing color.
opacity	Drawing opacity.
brightness	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

x0	X-coordinate of the first vertex in the image instance.
y0	Y-coordinate of the first vertex in the image instance.
x1	X-coordinate of the second vertex in the image instance.
y1	Y-coordinate of the second vertex in the image instance.
x2	X-coordinate of the third vertex in the image instance.
y2	Y-coordinate of the third vertex in the image instance.
color	Pointer to `spectrum()` consecutive values, defining the drawing color.
brightness0	Brightness factor of the first vertex (in [0,2]).
brightness1	brightness factor of the second vertex (in [0,2]).
brightness2	brightness factor of the third vertex (in [0,2]).
opacity	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

x0	X-coordinate of the first vertex in the image instance.
y0	Y-coordinate of the first vertex in the image instance.
x1	X-coordinate of the second vertex in the image instance.
y1	Y-coordinate of the second vertex in the image instance.
x2	X-coordinate of the third vertex in the image instance.
y2	Y-coordinate of the third vertex in the image instance.
color1	Pointer to `spectrum()` consecutive values of type `T`, defining the color of the first vertex.
color2	Pointer to `spectrum()` consecutive values of type `T`, defining the color of the seconf vertex.
color3	Pointer to `spectrum()` consecutive values of type `T`, defining the color of the third vertex.

opacity	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

x0	X-coordinate of the first vertex in the image instance.
y0	Y-coordinate of the first vertex in the image instance.
x1	X-coordinate of the second vertex in the image instance.
y1	Y-coordinate of the second vertex in the image instance.
x2	X-coordinate of the third vertex in the image instance.
y2	Y-coordinate of the third vertex in the image instance.
texture	Texture image used to fill the triangle.
tx0	X-coordinate of the first vertex in the texture image.
ty0	Y-coordinate of the first vertex in the texture image.
tx1	X-coordinate of the second vertex in the texture image.
ty1	Y-coordinate of the second vertex in the texture image.
tx2	X-coordinate of the third vertex in the texture image.
ty2	Y-coordinate of the third vertex in the texture image.
opacity	Drawing opacity.
brightness	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

x0	X-coordinate of the first vertex in the image instance.
y0	Y-coordinate of the first vertex in the image instance.
x1	X-coordinate of the second vertex in the image instance.
y1	Y-coordinate of the second vertex in the image instance.
x2	X-coordinate of the third vertex in the image instance.
y2	Y-coordinate of the third vertex in the image instance.
color	Pointer to `spectrum()` consecutive values, defining the drawing color.
light	Light image.
lx0	X-coordinate of the first vertex in the light image.
ly0	Y-coordinate of the first vertex in the light image.
lx1	X-coordinate of the second vertex in the light image.
ly1	Y-coordinate of the second vertex in the light image.
lx2	X-coordinate of the third vertex in the light image.
ly2	Y-coordinate of the third vertex in the light image.
opacity	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

x0	X-coordinate of the first vertex in the image instance.
y0	Y-coordinate of the first vertex in the image instance.
x1	X-coordinate of the second vertex in the image instance.
y1	Y-coordinate of the second vertex in the image instance.
x2	X-coordinate of the third vertex in the image instance.
y2	Y-coordinate of the third vertex in the image instance.
texture	Texture image used to fill the triangle.
tx0	X-coordinate of the first vertex in the texture image.
ty0	Y-coordinate of the first vertex in the texture image.
tx1	X-coordinate of the second vertex in the texture image.
ty1	Y-coordinate of the second vertex in the texture image.
tx2	X-coordinate of the third vertex in the texture image.
ty2	Y-coordinate of the third vertex in the texture image.
brightness0	Brightness factor of the first vertex.
brightness1	Brightness factor of the second vertex.
brightness2	Brightness factor of the third vertex.
opacity	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

x0	X-coordinate of the first vertex in the image instance.
y0	Y-coordinate of the first vertex in the image instance.
x1	X-coordinate of the second vertex in the image instance.
y1	Y-coordinate of the second vertex in the image instance.
x2	X-coordinate of the third vertex in the image instance.
y2	Y-coordinate of the third vertex in the image instance.
texture	Texture image used to fill the triangle.
tx0	X-coordinate of the first vertex in the texture image.
ty0	Y-coordinate of the first vertex in the texture image.
tx1	X-coordinate of the second vertex in the texture image.
ty1	Y-coordinate of the second vertex in the texture image.
tx2	X-coordinate of the third vertex in the texture image.
ty2	Y-coordinate of the third vertex in the texture image.
light	Light image.
lx0	X-coordinate of the first vertex in the light image.
ly0	Y-coordinate of the first vertex in the light image.
lx1	X-coordinate of the second vertex in the light image.
ly1	Y-coordinate of the second vertex in the light image.
lx2	X-coordinate of the third vertex in the light image.
ly2	Y-coordinate of the third vertex in the light image.
opacity	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

x0	X-coordinate of the upper-left rectangle corner.
y0	Y-coordinate of the upper-left rectangle corner.
z0	Z-coordinate of the upper-left rectangle corner.
c0	C-coordinate of the upper-left rectangle corner.
x1	X-coordinate of the lower-right rectangle corner.
y1	Y-coordinate of the lower-right rectangle corner.
z1	Z-coordinate of the lower-right rectangle corner.
c1	C-coordinate of the lower-right rectangle corner.
val	Scalar value used to fill the rectangle area.
opacity	Drawing opacity.

8.1.4.421 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.

Parameters

x0	X-coordinate of the upper-left rectangle corner.
y0	Y-coordinate of the upper-left rectangle corner.
z0	Z-coordinate of the upper-left rectangle corner.
x1	X-coordinate of the lower-right rectangle corner.
y1	Y-coordinate of the lower-right rectangle corner.
z1	Z-coordinate of the lower-right rectangle corner.
color	Pointer to `spectrum()` consecutive values of type `T`, defining the drawing color.
opacity	Drawing opacity.

8.1.4.422 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.

Parameters

x0	X-coordinate of the upper-left rectangle corner.
y0	Y-coordinate of the upper-left rectangle corner.
x1	X-coordinate of the lower-right rectangle corner.
y1	Y-coordinate of the lower-right rectangle corner.
color	Pointer to `spectrum()` consecutive values of type `T`, defining the drawing color.
opacity	Drawing opacity.

8.1.4.423 CImg<T>& draw_polygon (const CImg< t > & *points*, const tc *const *color*, const float *opacity* = `1`)

Draw a filled 2d polygon.

Parameters

points	Set of polygon vertices.
color	Pointer to `spectrum()` consecutive values of type `T`, defining the drawing color.
opacity	Drawing opacity.

8.1.4.424 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.

Parameters

x0	X-coordinate of the ellipse center.
y0	Y-coordinate of the ellipse center.
r1	First radius of the ellipse.
r2	Second radius of the ellipse.
angle	Angle of the first radius.
color	Pointer to `spectrum()` consecutive values, defining the drawing color.
opacity	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

x0	X-coordinate of the ellipse center.
y0	Y-coordinate of the ellipse center.
tensor	Diffusion tensor describing the ellipse.
color	Pointer to `spectrum()` consecutive values, defining the drawing color.
opacity	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

x0	X-coordinate of the ellipse center.
y0	Y-coordinate of the ellipse center.
r1	First radius of the ellipse.
r2	Second radius of the ellipse.
angle	Angle of the first radius.
color	Pointer to `spectrum()` consecutive values, defining the drawing color.
opacity	Drawing opacity.
pattern	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

x0	X-coordinate of the ellipse center.
y0	Y-coordinate of the ellipse center.
tensor	Diffusion tensor describing the ellipse.
color	Pointer to `spectrum()` consecutive values, defining the drawing color.
opacity	Drawing opacity.

pattern	An integer whose bits describe the outline pattern.

8.1.4.428 CImg<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

x0	X-coordinate of the circle center.
y0	Y-coordinate of the circle center.
radius	Circle radius.
color	Pointer to `spectrum()` consecutive values, defining the drawing color.
opacity	Drawing opacity.

Note

- Circle version of the Bresenham's algorithm is used.

8.1.4.429 CImg<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

x0	X-coordinate of the circle center.
y0	Y-coordinate of the circle center.
radius	Circle radius.
color	Pointer to `spectrum()` consecutive values, defining the drawing color.
opacity	Drawing opacity.
pattern	An integer whose bits describe the outline pattern.

8.1.4.430 CImg<T>& draw_image (const int *x0*, const int *y0*, const int *z0*, const int *c0*, const CImg< t > & *sprite*, const float *opacity* = 1)

Draw an image.

Parameters

sprite	Sprite image.
x0	X-coordinate of the sprite position.
y0	Y-coordinate of the sprite position.
z0	Z-coordinate of the sprite position.
c0	C-coordinate of the sprite position.
opacity	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

sprite	Sprite image.
mask	Mask image.
x0	X-coordinate of the sprite position in the image instance.
y0	Y-coordinate of the sprite position in the image instance.
z0	Z-coordinate of the sprite position in the image instance.
c0	C-coordinate of the sprite position in the image instance.
mask_max_-value	Maximum pixel value of the mask image `mask`.
opacity	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

x0	X-coordinate of the text in the image instance.
y0	Y-coordinate of the text in the image instance.
text	Format of the text ('printf'-style format string).
foreground_color	Pointer to `spectrum()` consecutive values, defining the foreground drawing color.
background_-color	Pointer to `spectrum()` consecutive values, defining the background drawing color.
opacity	Drawing opacity.
font	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

x0	X-coordinate of the text in the image instance.
y0	Y-coordinate of the text in the image instance.
text	Format of the text ('printf'-style format string).
foreground_color	Array of spectrum() values of type `T`, defining the foreground color (0 means 'transparent').
background_-color	Array of spectrum() values of type `T`, defining the background color (0 means 'transparent').
opacity	Drawing opacity.
font_height	Height of the text font (exact match for 13,24,32,57, 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

flow	Image of 2d vectors used as input data.
color	Image of spectrum()-D vectors corresponding to the color of each arrow.
opacity	Drawing opacity.
sampling	Length (in pixels) between each arrow.
factor	Length factor of each arrow (if <0, computed as a percentage of the maximum length).
is_arrow	Tells if arrows must be drawn, instead of oriented segments.
pattern	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

flow	Image of 2d vectors used as input data.
color	Image of spectrum()-D vectors corresponding to the color of each arrow.
opacity	Opacity of the drawing.
sampling	Length (in pixels) between each arrow.
factor	Length factor of each arrow (if <0, computed as a percentage of the maximum length).
is_arrow	Tells if arrows must be drawn, instead of oriented segments.
pattern	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

values_x	Values along the horizontal axis.
y	Y-coordinate of the horizontal axis in the image instance.
color	Pointer to `spectrum()` consecutive values, defining the drawing color.
opacity	Drawing opacity.
pattern	Drawing pattern.
font_height	Height of the labels (exact match for 13,24,32,57, interpolated otherwise).
allow_zero	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

x	X-coordinate of the vertical axis in the image instance.
values_y	Values along the Y-axis.
color	Pointer to `spectrum()` consecutive values, defining the drawing color.
opacity	Drawing opacity.
pattern	Drawing pattern.
font_height	Height of the labels (exact match for 13,24,32,57, interpolated otherwise).
allow_zero	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

values_x	Values along the X-axis.
values_y	Values along the Y-axis.
color	Pointer to `spectrum()` consecutive values, defining the drawing color.
opacity	Drawing opacity.
pattern_x	Drawing pattern for the X-axis.
pattern_y	Drawing pattern for the Y-axis.
font_height	Height of the labels (exact match for 13,24,32,57, interpolated otherwise).
allow_zero	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

values_x	X-coordinates of the vertical lines.
values_y	Y-coordinates of the horizontal lines.
color	Pointer to `spectrum()` consecutive values, defining the drawing color.
opacity	Drawing opacity.
pattern_x	Drawing pattern for vertical lines.
pattern_y	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

data	Image containing the graph values I = f(x).
color	Pointer to `spectrum()` consecutive values, defining the drawing color.
opacity	Drawing opacity.
plot_type	Define the type of the plot: • 0 = No plot. • 1 = Plot using segments. • 2 = Plot using cubic splines. • 3 = Plot with bars.
vertex_type	Define the type of points: • 0 = No points. • 1 = Point. • 2 = Straight cross. • 3 = Diagonal cross. • 4 = Filled circle. • 5 = Outlined circle. • 6 = Square. • 7 = Diamond.
ymin	Lower bound of the y-range.
ymax	Upper bound of the y-range.
pattern	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

	x	X-coordinate of the starting point of the region to fill.
	y	Y-coordinate of the starting point of the region to fill.
	z	Z-coordinate of the starting point of the region to fill.
	color	Pointer to `spectrum()` consecutive values, defining the drawing color.
`out`	*region*	Image that will contain the mask of the filled region mask, as an output.
	sigma	Tolerance concerning neighborhood values.
	opacity	Opacity of the drawing.
	is_high_-connexity	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

alpha	Alpha-parameter.
beta	Beta-parameter.
scale	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

x0	X-coordinate of the upper-left pixel.
y0	Y-coordinate of the upper-left pixel.
x1	X-coordinate of the lower-right pixel.
y1	Y-coordinate of the lower-right pixel.
colormap	Colormap.
opacity	Drawing opacity.
z0r	Real part of the upper-left fractal vertex.
z0i	Imaginary part of the upper-left fractal vertex.
z1r	Real part of the lower-right fractal vertex.
z1i	Imaginary part of the lower-right fractal vertex.
iteration_max	Maximum number of iterations for each estimated point.
is_normalized_-iteration	Tells if iterations are normalized.
is_julia_set	Tells if the Mandelbrot or Julia set is rendered.
param_r	Real part of the Julia set parameter.
param_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

xc	X-coordinate of the gaussian center.
sigma	Standard variation of the gaussian distribution.
color	Pointer to `spectrum()` consecutive values, defining the drawing color.
opacity	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

xc	X-coordinate of the gaussian center.
yc	Y-coordinate of the gaussian center.
tensor	Covariance matrix (must be 2x2).
color	Pointer to `spectrum()` consecutive values, defining the drawing color.
opacity	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

x0	X-coordinate of the 3d object position
y0	Y-coordinate of the 3d object position
z0	Z-coordinate of the 3d object position
vertices	Image Nx3 describing 3d point coordinates
primitives	List of P primitives
colors	List of P color (or textures)
opacities	Image or list of P opacities
render_type	d Render type (0=Points, 1=Lines, 2=Faces (no light), 3=Faces (flat), 4=Faces(Gouraud)
is_double_sided	Tells if object faces have two sides or are oriented.
focale	length of the focale (0 for parallel projection)
lightx	X-coordinate of the light
lighty	Y-coordinate of the light
lightz	Z-coordinate of the light
specular_-lightness	Amount of specular light.
specular_-shininess	Shininess of the object

8.1.4.449 CImg<T>& select (CImgDisplay & *disp*, const unsigned int *feature_type* = `2`, unsigned int *const *XYZ* = `0`)

Launch simple interface to select a shape from an image.

Parameters

disp	Display window to use.
feature_type	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

filename	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

filename	Filename, as a C -string.

8.1.4.452 CImg<T>& load_dlm (const char *const *filename*)

Load image from a DLM file.

Parameters

filename	Filename, as a C-string.

8.1.4.453 CImg<T>& load_bmp (const char *const *filename*)

Load image from a BMP file.

Parameters

filename	Filename, as a C-string.

8.1.4.454 CImg<T>& load_jpeg (const char *const *filename*)

Load image from a JPEG file.

Parameters

filename	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

filename	Filename, as a C-string.

8.1.4.456 CImg<T>& load_png (const char *const *filename*)

Load image from a PNG file.

Parameters

filename	Filename, as a C-string.

8.1.4.457 CImg<T>& load_pnm (const char *const *filename*)

Load image from a PNM file.

Parameters

filename	Filename, as a C-string.

8.1.4.458 CImg<T>& load_pfm (const char *const *filename*)

Load image from a PFM file.

Parameters

filename	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

filename	Filename, as a C-string.
dimw	Width of the image buffer.
dimh	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

filename	Filename, as a C-string.
dimw	Width of the image buffer.
dimh	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)

Load image from a TIFF file.

Parameters

filename	Filename, as a C-string.
first_frame	First frame to read (for multi-pages tiff).
last_frame	Last frame to read (for multi-pages tiff).
step_frame	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 compilation 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

filename	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

	filename	Filename, as a C-string.
out	*voxel_size*	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

filename	Filename, as a C-string.
axis	Appending axis, if file contains multiple images. Can be { 'x' | 'y' | 'z' | 'c' }.
align	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

filename	Filename, as a C-string.
n0	Starting frame.
n1	Ending frame (~0U for max).
x0	X-coordinate of the starting sub-image vertex.
y0	Y-coordinate of the starting sub-image vertex.
z0	Z-coordinate of the starting sub-image vertex.
c0	C-coordinate of the starting sub-image vertex.

x1	X-coordinate of the ending sub-image vertex (∼0U for max).
y1	Y-coordinate of the ending sub-image vertex (∼0U for max).
z1	Z-coordinate of the ending sub-image vertex (∼0U for max).
c1	C-coordinate of the ending sub-image vertex (∼0U for max).
axis	Appending axis, if file contains multiple images. Can be `{ 'x' | 'y' | 'z' | 'c' }`.
align	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

	filename	Filename, as a C-string.
`out`	*voxel_size*	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

filename	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

filename	Filename, as a C-string.

8.1.4.469 CImg<T>& load_parrec (const char ∗const *filename*, const char *axis* = `'c'`, const float *align* = `0`)

Load image from a PAR-REC (Philips) file.

Parameters

filename	Filename, as a C-string.
axis	Appending axis, if file contains multiple images. Can be `{ 'x' | 'y' | 'z' | 'c' }`.
align	Appending alignment.

8.1.4.470 CImg<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 unsigned long *offset* = `0`)

Load image from a raw binary file.

Parameters

filename	Filename, as a C-string.

size_x	Width of the image buffer.
size_y	Height of the image buffer.
size_z	Depth of the image buffer.
size_c	Spectrum of the image buffer.
is_multiplexed	Tells if the image values are multiplexed along the C-axis.
invert_-endianness	Tells if the endianness of the image buffer must be inverted.
offset	Starting offset of the read in the specified file.

8.1.4.471 CImg<T>& load_ffmpeg (const char *const *filename*, const unsigned int *first_frame* = `0`, const unsigned int *last_frame* = `~0U`, const unsigned int *step_frame* = `1`, const bool *pixel_format* = `true`, const bool *resume* = `false`, const char *axis* = `'z'`, const float *align* = `0`)

Load image sequence using FFMPEG av's libraries.

Parameters

filename	Filename, as a C-string.
first_frame	Index of the first frame to read.
last_frame	Index of the last frame to read.
step_frame	Step value for frame reading.
pixel_format	To be documented.
resume	To be documented.
axis	Appending axis, if file contains multiple images. Can be `{ 'x' | 'y' | 'z' | 'c' }`.
align	Appending alignment.

8.1.4.472 CImg<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

filename	Filename, as a C-string.
size_x	Width of the frames.
size_y	Height of the frames.
first_frame	Index of the first frame to read.
last_frame	Index of the last frame to read.
step_frame	Step value for frame reading.
yuv2rgb	Tells if the YUV to RGB transform must be applied.
axis	Appending axis, if file contains multiple images. Can be `{ 'x' | 'y' | 'z' | 'c' }`.

8.1.4.473 CImg<T>& load_off (CImgList< tf > & *primitives*, CImgList< tc > & *colors*, const char *const *filename*)

Load 3d object from a .OFF file.

Parameters

`out`	*primitives*	Primitives data of the 3d object.
`out`	*colors*	Colors data of the 3d object.

	filename	Filename, as a C-string.

8.1.4.474 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'.

Parameters

filename	Filename, as a C-string.
axis	Appending axis, if file contains multiple images. Can be { 'x' | 'y' | 'z' | 'c' }.
align	Appending alignment.

8.1.4.475 CImg<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

filename	Filename, as a C-string.
use_-graphicsmagick	Tells if GraphicsMagick's tool 'gm' is used instead of ImageMagick's tool 'convert'.
axis	Appending axis, if file contains multiple images. Can be { 'x' | 'y' | 'z' | 'c' }.
align	Appending alignment.

8.1.4.476 CImg<T>& load_graphicsmagick_external (const char *const *filename*)

Load image using GraphicsMagick's external tool 'gm'.

Parameters

filename	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

filename	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

filename	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

filename	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

filename	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* = false, const unsigned int *capture_width* = 0, const unsigned int *capture_height* = 0)

Load image from a camera stream, using OpenCV.

Parameters

camera_index	Index of the camera to capture images from.
skip_frames	Number of frames to skip before the capture.
release_camera	Tells if the camera ressource 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

filename	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 informations about the image data.

Parameters

title	Name for the considered image.
display_stats	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

disp	Display window.

8.1.4.485 const CImg<T>& display (CImgDisplay & *disp*, const bool *display_info*, unsigned int *const *XYZ* = 0) const

Display image into a CImgDisplay window, in an interactive way.

Parameters

disp	Display window.
display_info	Tells if image informations 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

Display image into an interactive window.

Parameters

title	Window title
display_info	Tells if image informations 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

Display object 3d in an interactive window.

Parameters

disp	Display window.
vertices	Vertices data of the 3d object.
primitives	Primitives data of the 3d object.
colors	Colors data of the 3d object.
opacities	Opacities data of the 3d object.
centering	Tells if the 3d object must be centered for the display.
render_static	Rendering mode.
render_motion	Rendering mode, when the 3d object is moved.
is_double_sided	Tells if the object primitives are double-sided.
focale	Focale
light_x	X-coordinate of the light source.
light_y	Y-coordinate of the light source.
light_z	Z-coordinate of the light source.
specular_-lightness	Amount of specular light.
specular_-shininess	Shininess of the object material.
display_axes	Tells if the 3d axes are displayed.
pose_matrix	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 *labely* = 0, const double *ymin* = 0, const double *ymax* = 0) const

Display 1d graph in an interactive window.

Parameters

disp	Display window.
plot_type	Plot type. Can be `{ 0=points | 1=segments | 2=splines | 3=bars }`.
vertex_type	Vertex type.
labelx	Title for the horizontal axis, as a C-string.
xmin	Minimum value along the X-axis.
xmax	Maximum value along the X-axis.
labely	Title for the vertical axis, as a C-string.
ymin	Minimum value along the X-axis.
ymax	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

filename	Filename, as a C-string.
number	When positive, represents an index added to the filename. Otherwise, no number is added.
digits	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

filename	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

filename	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

filename	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

filename	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

filename	Filename, as a C-string.
quality	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

filename	Filename, as a C-string.
bytes_per_pixel	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

filename	Filename, as a C-string.
bytes_per_pixel	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

filename	Filename, as a C-string.
bytes_per_pixel	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

filename	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

filename	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

filename	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

filename	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

Save image as a TIFF file.

Parameters

filename	Filename, as a C-string.
compression_type	Type of data compression. Can be `{ 1=None | 2=CCITTRLE | 3=CCITTFAX3 | 4=CCITTFAX4 | 5=LZW | 6=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 compilation 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

filename	Filename, as a C-string.
imitate_file	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

filename	Filename, as a C-string.
voxel_size	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

filename	Filename, as a C-string.
is_compressed	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

filename	Filename, as a C-string.
n0	Index of the image inside the file.
x0	X-coordinate of the sub-image location.
y0	Y-coordinate of the sub-image location.
z0	Z-coordinate of the sub-image location.
c0	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

filename	Filename, as a C-string.
dx	Width of the image.
dy	Height of the image.
dz	Depth of the image.
dc	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

filename	Filename, as a C-string.
voxel_size	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

filename	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

filename	Filename, as a C-string.
colorspace	Colorspace data field in output file (see `Pandore file specifications` for more informations).

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

filename	Filename, as a C-string.
is_multiplexed	Tells if the image channels are stored in a multiplexed way (`true`) or not (`false`).

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_ffmpeg (const char *const *filename*, const unsigned int *fps* = 25, const unsigned int *bitrate* = 2048) const

Save image as a video file, using the FFmpeg library.

Parameters

filename	Filename, as a C-string.
fps	Video framerate.
bitrate	Video bitrate.

Note

- Each slice of the instance image is considered to be a single frame of the output video file.
- This method uses functions provided by the `FFmpeg` library. Configuration macro `cimg_use_ffmpeg` must be set for the method to succeed natively. Otherwise, the method calls save_ffmpeg_external(const char∗,unsigned int,unsigned int,const char∗,unsigned int,unsigned int) const.

8.1.4.516 const CImg<T>& save_yuv (const char ∗const *filename*, const bool *is_rgb* = `true`) const

Save image as a .yuv video file.

Parameters

filename	Filename, as a C-string.
is_rgb	Tells if pixel values of the instance image are RGB-coded (`true`) or YUV-coded (`false`).

Note

Each slice of the instance image is considered to be a single frame of the output video file.

8.1.4.517 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.

8.1.4.518 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

filename	Filename, as a C-string.
primitives	List of 3d object primitives.
colors	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`.

8.1.4.519 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]**.

Same as save_off(const CImgList<tf>&,const CImgList<tc>&,const char∗) const with a file stream argument instead of a filename string.

8.1.4.520 const CImg<T>& save_ffmpeg_external (const char *const *filename*, const char *const *codec* = 0, const unsigned int *fps* = 25, const unsigned int *bitrate* = 2048) const

Save volumetric image as a video, using ffmpeg external binary.

Parameters

filename	Filename, as a C-string.
codec	Video codec, as a C-string.
fps	Video framerate.
bitrate	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 CImg<T>& save_gzip_external (const char *const *filename*) const

Save image using gzip external binary.

Parameters

filename	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_graphicsmagick_external (const char *const *filename*, const unsigned int *quality* = 100) const

Save image using GraphicsMagick's external binary.

Parameters

filename	Filename, as a C-string.
quality	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

filename	Filename, as a C-string.
quality	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

filename	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

filename	Filename, as a C-string.
quality	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.

8.2 CImgDisplay Struct Reference

Allow to create 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.

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 occured 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) const

  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 occuring on the display `disp1`.
- static void wait (CImgDisplay &disp1, CImgDisplay &disp2)

  Wait for any event occuring either on the display `disp1` *or* `disp2`.
- static void wait (CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3)

  Wait for any event occuring either on the display `disp1`, `disp2` *or* `disp3`.
- static void wait (CImgDisplay &disp1, CImgDisplay &disp2, CImgDisplay &disp3, CImgDisplay &disp4)

  Wait for any event occuring 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 occuring 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 occuring 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 occuring 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 occuring 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 occuring 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 occuring either on the display `disp1`, `disp2`, `disp3`, `disp4`, *...* `disp10`.
- static void wait_all ()

  Wait for any window event occuring in any opened CImgDisplay.

8.2.1 Detailed Description

Allow to create 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 outputed 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

width	Window width.
height	Window height.
title	Window title.
normalization	Normalization type (0=none, 1=always, 2=once, 3=pixel type-dependent, see normalization()).
is_fullscreen	Tells if fullscreen mode is enabled.
is_closed	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

img	Image used as a model to create the window.
title	Window title.
normalization	Normalization type (0=none, 1=always, 2=once, 3=pixel type-dependent, see normalization()).
is_fullscreen	Tells if fullscreen mode is enabled.
is_closed	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

list	The images list to display.
title	Window title.
normalization	Normalization type (0=none, 1=always, 2=once, 3=pixel type-dependent, see normalization()).
is_fullscreen	Tells if fullscreen mode is enabled.
is_closed	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

disp	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

keycode	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*) const

Return `true` if key specified by given keycode is being pressed on the associated window, `false` otherwise.

Parameters

keycode	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

keycodes_-sequence	Buffer of keycodes to test.
length	Number of keys in the `keycodes_sequence` buffer.
remove_-sequence	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 substract `1` to the wheel value.
- The returned value cumulates the number of forward of 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

pos	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

pos	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

img	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

list	List of images to display.
axis	Axis used to append the images along, for the visualization (can be x, y, z or c).
align	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

pos_x	X-coordinate of the new window location.
pos_y	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

force_redraw	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

width	Requested display width.
height	Requested display height.
force_redraw	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

img	Input image to take size from.
force_redraw	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

disp	Input display to take size from.
force_redraw	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

normalization	New normalization mode.

8.2.3.39 CImgDisplay& set_title (const char *const *format*, ...)

Set title of the associated window.

Parameters

format	C-string containing the format of the title, as with `std::printf()`.

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

is_fullscreen	Tells is the fullscreen mode must be activated or not.
force_redraw	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

force_redraw	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

button	Buttons event code, where each button is associated to a single bit.
is_pressed	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

amplitude	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

keycode	Keycode of the associated key.
is_pressed	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

milliseconds	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

img	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 C-ImgWarningException.

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`. CImgException is never thrown itself. Derived classes that specify the type of errord are thrown instead. These derived classes can be:

- **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 `CImgArgument-Exception:`

```
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 occured 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 the CImg
      library.
  } 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 >
  CImgList (const CImgList< t > &list)

  Construct list copy.
- CImgList (const CImgList< T > &list)

  *Construct list copy **[specialization]**.*
- template<typename t >
  CImgList (const CImgList< t > &list, const bool is_shared)

  Construct list copy, and force the shared state of the list elements.
- CImgList (const char *const filename)

  Construct list by reading the content of a file.
- CImgList (const CImgDisplay &disp)

  Construct list from the content of a display window.
- CImgList< T > get_shared ()

  Return a list with elements being shared copies of images in the list instance.
- const CImgList< T > get_shared () const

  *Return a list with elements being shared copies of images in the list instance **[const version]**.*
- CImgList< T > & assign ()

  *Destructor **[in-place version]**.*

- CImgList< T > & clear ()

  *Destructor **[in-place version]**.*

- CImgList< T > & assign (const unsigned int n)

  *Construct list containing empty images **[in-place version]**.*

- 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]**.*

- 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]**.*

- 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 containing images of specified size, and initialize pixel values from a sequence of integers **[in-place version]**.*

- 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 containing images of specified size, and initialize pixel values from a sequence of doubles **[in-place version]**.*

- template<typename t >
  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]**.*

- template<typename t >
  CImgList< T > & assign (const CImg< t > &img, const bool is_shared=false)

  *Construct list from one image **[in-place version]**.*

- template<typename t1 , typename t2 >
  CImgList< T > & assign (const CImg< t1 > &img1, const CImg< t2 > &img2, const bool is_shared=false)

  *Construct list from two images **[in-place version]**.*

- template<typename t1 , typename t2 , typename t3 >
  CImgList< T > & assign (const CImg< t1 > &img1, const CImg< t2 > &img2, const CImg< t3 > &img3, const bool is_shared=false)

  *Construct list from three images **[in-place version]**.*

- template<typename t1 , typename t2 , typename t3 , typename t4 >
  CImgList< T > & assign (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 **[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 the shared state of the list 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.

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 CImgList< t > &list)

  Return a copy of the list instance, where all elements of input list `list` have been inserted at the end.
- 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 & 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.
- T atNXYZC (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 & atNXYZC (const int pos, const int x, const int y, const int z, const int c)

  Access to pixel value with Neumann boundary conditions.
- T atNXYZC (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 & atNXYZ (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 three first coordinates (pos, x,y,z).
- T atNXYZ (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 three first coordinates (pos, x,y,z) **[const version]**.*
- 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 four first coordinates (pos, x,y,z).
- T atNXYZ (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 four first coordinates (pos, x,y,z) **[const version]**.*
- 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 three first coordinates (pos, x,y).
- T atNXY (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 three first coordinates (pos, x,y) **[const version]**.*
- 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 three first coordinates (pos, x,y).
- T atNXY (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 three first coordinates (pos, x,y) **[const version]**.*
- 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 two first coordinates (pos,x).
- T atNX (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 two first coordinates (pos,x) **[const version]**.*
- 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 two first coordinates (pos, x).
- T atNX (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 two first coordinates (pos, x) **[const version]**.*
- 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 coordinates (pos).
- T atN (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 coordinates (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 coordinates (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 coordinates (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 CImg< t > &img) const
- template< typename t >
  bool **is_sameNYC** (const CImgList< t > &list) const
- template< typename t >
  bool **is_sameXYZ** (const CImg< t > &img) const
- template< typename t >
  bool **is_sameXYZ** (const CImgList< t > &list) const
- template< typename t >
  bool **is_sameNXYZ** (const unsigned int n, const CImg< t > &img) const
- template< typename t >
  bool **is_sameNXYZ** (const CImgList< t > &list) const
- template< typename t >
  bool **is_sameXYC** (const CImg< t > &img) const
- template< typename t >
  bool **is_sameXYC** (const CImgList< t > &list) const
- template< typename t >
  bool **is_sameNXYC** (const unsigned int n, const CImg< t > &img) const
- template< typename t >
  bool **is_sameNXYC** (const CImgList< t > &list) const
- template< typename t >
  bool **is_sameYZC** (const CImg< t > &img) const
- template< typename t >
  bool **is_sameYZC** (const CImgList< t > &list) const
- template< typename t >
  bool **is_sameNYZC** (const unsigned int n, const CImg< t > &img) const

- template<typename t >
  bool **is_sameNYZC** (const CImgList< t > &list) const
- template<typename t >
  bool **is_sameXYZC** (const CImg< t > &img) const
- template<typename t >
  bool **is_sameXYZC** (const CImgList< t > &list) const
- template<typename t >
  bool **is_sameNXYZC** (const unsigned int n, const CImg< t > &img) const
- template<typename t >
  bool **is_sameNXYZC** (const CImgList< 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 >
  CImgList< T > get_insert (const unsigned int n, const CImgList< 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]****.*
- CImgList< T > & remove (const unsigned int pos1, const unsigned int pos2)

  Remove all images between from indexes.
- CImgList< T > get_remove (const unsigned int pos1, const unsigned int pos2) const

  Remove all images between from indexes ***[new-instance version]****.*
- CImgList< T > & remove (const unsigned int pos)

  Remove image at index `pos` *from the image list.*
- CImgList< T > get_remove (const unsigned int pos) const

  Remove image at index `pos` *from the image list* ***[new-instance version]****.*
- CImgList< T > & remove ()

  Remove last image.
- CImgList< T > get_remove () const

  Remove last image ***[new-instance version]****.*
- CImgList< T > & reverse ()

  Reverse list order.
- CImgList< T > get_reverse () const

  Reverse list order ***[new-instance version]****.*
- CImgList< T > & images (const unsigned int pos0, const unsigned int pos1)

  Return a sublist.
- CImgList< T > get_images (const unsigned int pos0, const unsigned int pos1) const

  Return a sublist ***[new-instance version]****.*

- CImgList< T > get_shared_images (const unsigned int pos0, const unsigned int pos1)

    Return a shared sublist.
- const CImgList< T > get_shared_images (const unsigned int pos0, const unsigned int pos1) const

    *Return a shared sublist **[new-instance version]**.*
- 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.
- CImgList< T > & split (const char axis, const int nb=0)

    Return a list where each image has been split along the specified axis.
- CImgList< T > get_split (const char axis, const int nb=0) const

    *Return a list where each image has been split along the specified axis **[new-instance version]**.*
- template< typename t >
  CImgList< T > & push_back (const CImg< t > &img)

    Insert image at the end of the list.
- template< typename t >
  CImgList< T > & push_front (const CImg< t > &img)

    Insert image at the front of the list.
- template< typename t >
  CImgList< T > & push_back (const CImgList< t > &list)

    Insert list at the end of the current list.
- template< typename t >
  CImgList< T > & push_front (const CImgList< t > &list)

    Insert list at the front of the current list.
- CImgList< T > & pop_back ()

    Remove last image.
- CImgList< T > & pop_front ()

    Remove first image.
- CImgList< T > & erase (const iterator iter)

    Remove image pointed by iterator.

Data Input

- CImg< intT > get_select (CImgDisplay &disp, const bool feature_type=true, const char axis='x', const float align=0) const

    Display a simple interactive interface to select images or sublists.
- CImg< intT > get_select (const char ∗const title, const bool feature_type=true, const char axis='x', const float align=0) const

    Display a simple interactive interface to select images or sublists.
- CImgList< T > & load (const char ∗const filename)

    Load a list from a file.
- CImgList< T > & load_cimg (const char ∗const filename)

    Load a list from a .cimg file.
- CImgList< T > & load_cimg (std::FILE ∗const file)

    Load a list from a .cimg file.
- 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.
- CImgList< 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]**.*
- CImgList< T > & load_parrec (const char ∗const filename)

Load a list from a PAR/REC (Philips) file.
- 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.
- CImgList< 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]**.*
- CImgList< T > & load_ffmpeg (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=∼0U, const unsigned int step_frame=1, const bool pixel_format=true, const bool resume=false)

  Load an image from a video file, using ffmpeg libraries.
- CImgList< T > & load_ffmpeg_external (const char *const filename)

  Load an image from a video file using the external tool 'ffmpeg'.
- CImgList< T > & load_gif_external (const char *const filename)

  Load gif file, using ImageMagick or GraphicsMagick's external tools.
- CImgList< T > & load_gzip_external (const char *const filename)

  Load a gzipped list, using external tool 'gunzip'.
- template<typename tf , typename tc >
  CImgList< T > & load_off (const char *const filename, CImgList< tf > &primitives, CImgList< tc > &colors)

  Load a 3d object from a .OFF file.
- 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)

  Load images from a TIFF file.
- static CImgList< T > get_load (const char *const filename)

  *Load a list from a file **[new-instance version]**.*
- static CImgList< T > get_load_cimg (const char *const filename)

  *Load a list from a .cimg file **[new-instance version]**.*
- static CImgList< 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_ffmpeg (const char *const filename, const unsigned int first_frame=0, const unsigned int last_frame=∼0U, const unsigned int step_frame=1, const bool pixel_format=true)

  *Load an image from a video file, using ffmpeg libraries **[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)

  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 informations 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

  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

  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 unsigned int fps=25, const unsigned int nb_loops=0)

  Save image sequence as a GIF animated file.
- const CImgList< T > & save_ffmpeg (const char *const filename, const unsigned int fps=25, const unsigned int bitrate=2048) const

  Save image sequence, using FFMPEG library.
- 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

  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_ffmpeg_external (const char *const filename, const char *const codec=0, const unsigned int fps=25, const unsigned int bitrate=2048) const

    Save image sequence, using the external tool 'ffmpeg'.
- 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.

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< T > & 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 = float>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

n	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

n	Number of images.
width	Width of images.
height	Height of images.
depth	Depth of images.
spectrum	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

n	Number of images.
width	Width of images.
height	Height of images.
depth	Depth of images.
spectrum	Number of channels of images.
val	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

n	Number of images.
width	Width of images.
height	Height of images.
depth	Depth of images.

spectrum	Number of channels of images.
val0	First value of the initializing integers sequence.
val1	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 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.

Parameters

n	Number of images.
width	Width of images.
height	Height of images.
depth	Depth of images.
spectrum	Number of channels of images.
val0	First value of the initializing doubles sequence.
val1	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 CImgList (const unsigned int *n*, const CImg< t > & *img*, const bool *is_shared* = `false`)

Construct list containing copies of an input image.

Parameters

n	Number of images.
img	Input image to copy in the constructed list.
is_shared	Tells if the elements of the list are shared or non-shared copies of `img`.

8.4.3.9 CImgList (const CImg< t > & *img*, const bool *is_shared* = `false`) `[explicit]`

Construct list from one image.

Parameters

img	Input image to copy in the constructed list.
is_shared	Tells if the element of the list is a shared or non-shared copy of `img`.

8.4.3.10 CImgList (const CImg< t1 > & *img1*, const CImg< t2 > & *img2*, const bool *is_shared* = `false`)

Construct list from two images.

Parameters

img1	First input image to copy in the constructed list.
img2	Second input image to copy in the constructed list.
is_shared	Tells if the elements of the list are shared or non-shared copies of input images.

8.4.3.11 CImgList (const CImg< t1 > & *img1*, const CImg< t2 > & *img2*, const CImg< t3 > & *img3*, const bool *is_shared* = `false`)

Construct list from three images.

Parameters

img1	First input image to copy in the constructed list.
img2	Second input image to copy in the constructed list.
img3	Third input image to copy in the constructed list.
is_shared	Tells if the elements of the list are shared or non-shared copies of input images.

8.4.3.12 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.

Parameters

img1	First input image to copy in the constructed list.
img2	Second input image to copy in the constructed list.
img3	Third input image to copy in the constructed list.
img4	Fourth input image to copy in the constructed list.
is_shared	Tells if the elements of the list are shared or non-shared copies of input images.

8.4.3.13 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.

Parameters

img1	First input image to copy in the constructed list.
img2	Second input image to copy in the constructed list.
img3	Third input image to copy in the constructed list.
img4	Fourth input image to copy in the constructed list.
img5	Fifth input image to copy in the constructed list.
is_shared	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

img1	First input image to copy in the constructed list.
img2	Second input image to copy in the constructed list.
img3	Third input image to copy in the constructed list.
img4	Fourth input image to copy in the constructed list.
img5	Fifth input image to copy in the constructed list.
img6	Sixth input image to copy in the constructed list.
is_shared	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

img1	First input image to copy in the constructed list.
img2	Second input image to copy in the constructed list.
img3	Third input image to copy in the constructed list.
img4	Fourth input image to copy in the constructed list.
img5	Fifth input image to copy in the constructed list.
img6	Sixth input image to copy in the constructed list.
img7	Seventh input image to copy in the constructed list.
is_shared	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

img1	First input image to copy in the constructed list.
img2	Second input image to copy in the constructed list.
img3	Third input image to copy in the constructed list.
img4	Fourth input image to copy in the constructed list.
img5	Fifth input image to copy in the constructed list.
img6	Sixth input image to copy in the constructed list.
img7	Seventh input image to copy in the constructed list.
img8	Eighth input image to copy in the constructed list.
is_shared	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

list	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

list	Input list to copy.
is_shared	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

filename	Filename, as a C-string.

8.4.3.20 CImgList (const CImgDisplay & *disp*) [explicit]

Construct list from the content of a display window.

Parameters

disp	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 containing 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 containing 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 CImgList<T>& assign (const CImg< t > & *img*, const bool *is_shared* = false)

Construct list from one image **[in-place version]**.

See Also

CImgList(const CImg<t>&, bool).

8.4.4.11 CImgList<T>& assign (const CImg< t1 > & *img1*, const CImg< t2 > & *img2*, const bool *is_shared* = false)

Construct list from two images **[in-place version]**.

See Also

CImgList(const CImg<t>&, const CImg<t>&, bool).

8.4.4.12 CImgList<T>& assign (const CImg< t1 > & *img1*, const CImg< t2 > & *img2*, const CImg< t3 > & *img3*, const bool *is_shared* = false)

Construct list from three images **[in-place version]**.

See Also

CImgList(const CImg<t>&, const CImg<t>&, const CImg<t>&, bool).

8.4.4.13 CImgList<T>& assign (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 **[in-place version]**.

See Also

CImgList(const CImg<t>&, const CImg<t>&, const CImg<t>&, const CImg<t>&, bool).

8.4.4.14 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]**.

See Also

CImgList(const CImg<t>&, const CImg<t>&, const CImg<t>&, const CImg<t>&, const CImg<t>&, bool).

8.4.4.15 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]**.

See Also

CImgList(const CImg<t>&, const CImg<t>&, const CImg<t>&, const CImg<t>&, const CImg<t>&, const CImg<t>&, bool).

8.4.4.16 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]**.

See Also

CImgList(const CImg<t>&, const CImg<t>&, const CImg<t>&, const CImg<t>&, const CImg<t>&, const CImg<t>&, const CImg<t>&, bool).

8.4.4.17 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]**.

See Also

CImgList(const CImg<t>&, const CImg<t>&, const CImg<t>&, const CImg<t>&, const CImg<t>&, const CImg<t>&, const CImg<t>&, const CImg<t>&, bool).

8.4.4.18 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]**.

See Also

CImgList(const CImgList<t>&, bool is_shared).

8.4.4.19 CImgList<T>& assign (const char ∗const *filename*)

Construct list by reading the content of a file **[in-place version]**.

See Also

CImgList(const char ∗const).

8.4.4.20 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

list	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

list	Destination list.
pos	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

list	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

pos	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

pos	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

pos	Indice of the image element.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	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

img	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

list	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

img	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

list	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

axis	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

axis	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

pos	Indice of the image element to access.

Note

`list.data(n);` is equivalent to `list.data + n;`.

8.4.4.43 CImg<T>& at (const int *pos*)

Return pos-th image of the list.

Parameters

pos	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

pos	Indice of the image element to access.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.
out_value	Default value returned if `offset` 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

pos	Indice of the image element to access.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	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 to pixel value with Dirichlet boundary conditions for the three first coordinates (`pos`, `x`,`y`,`z`).

Parameters

pos	Indice of the image element to access.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.
out_value	Default value returned if `offset` 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 four first coordinates (`pos`, `x`,`y`,`z`).

Parameters

pos	Indice of the image element to access.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	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 three first coordinates (`pos`, `x`,`y`).

Parameters

pos	Indice of the image element to access.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.
out_value	Default value returned if `offset` 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 three first coordinates (`pos`, `x`,`y`).

Parameters

pos	Indice of the image element to access.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	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 two first coordinates (`pos`,`x`).

Parameters

pos	Indice of the image element to access.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.
out_value	Default value returned if `offset` 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 two first coordinates (`pos`, `x`).

Parameters

pos	Indice of the image element to access.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	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 coordinates (`pos`).

Parameters

pos	Indice of the image element to access.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	C-coordinate of the pixel value.
out_value	Default value returned if `offset` 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 coordinates (`pos`).

Parameters

pos	Indice of the image element to access.
x	X-coordinate of the pixel value.
y	Y-coordinate of the pixel value.
z	Z-coordinate of the pixel value.
c	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

separator	Character separator set between consecutive pixel values.
max_size	Maximum size of the returned string.

Note

The result is returne 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

size_n	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

list	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

dx	Checked image width.
dy	Checked image height.
dz	Checked image depth.
dc	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

n	Number of images in the list.
dx	Checked image width.
dy	Checked image height.
dz	Checked image depth.
dc	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

n	Index of the image whom checked pixel value belong to.
x	X-coordinate of the checked pixel value.
y	Y-coordinate of the checked pixel value.
z	Z-coordinate of the checked pixel value.
c	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

n	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

	pixel	Reference to pixel value to test.
`out`	*n*	Index of image containing the pixel value, if test succeeds.
`out`	*x*	X-coordinate of the pixel value, if test succeeds.
`out`	*y*	Y-coordinate of the pixel value, if test succeeds.
`out`	*z*	Z-coordinate of the pixel value, if test succeeds.
`out`	*c*	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

	pixel	Reference to pixel value to test.
`out`	*n*	Index of image containing the pixel value, if test succeeds.
`out`	*x*	X-coordinate of the pixel value, if test succeeds.
`out`	*y*	Y-coordinate of the pixel value, if test succeeds.
`out`	*z*	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

	pixel	Reference to pixel value to test.
out	*n*	Index of image containing the pixel value, if test succeeds.
out	*x*	X-coordinate of the pixel value, if test succeeds.
out	*y*	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

	pixel	Reference to pixel value to test.
out	*n*	Index of image containing the pixel value, if test succeeds.
out	*x*	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

	pixel	Reference to pixel value to test.
out	*n*	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

pixel	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

	img	Reference to image to test.
out	*n*	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

img	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

img	Image to insert a copy to the list.
pos	Index of the insertion.
is_shared	Tells if the inserted image is a shared copy of `img` 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

n	Number of empty images to insert.
pos	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

n	Number of image copies to insert.
img	Image to insert by copy.
pos	Index of the insertion.
is_shared	Tells if inserted images are shared copies of `img` 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

list	Image list to insert.
pos	Index of the insertion.
is_shared	Tells if inserted images are shared copies of images of `list` 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

n	Number of list copies to insert.
list	Image list to insert.
pos	Index of the insertion.
is_shared	Tells if inserted images are shared copies of images of `list` or not.

8.4.4.77 CImgList<T>& remove (const unsigned int *pos1*, const unsigned int *pos2*)

Remove all images between from indexes.

Parameters

pos1	Starting index of the removal.
pos2	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

pos	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

pos0	Starting index of the sublist.
pos1	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

pos0	Starting index of the sublist.
pos1	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

axis	Appending axis. Can be { 'x' | 'y' | 'z' | 'c' }.
align	Appending alignment.

8.4.4.82 CImgList<T>& split (const char *axis*, const int *nb* = 0)

Return a list where each image has been split along the specified axis.

Parameters

axis	Axis to split images along.
nb	Number of spliting 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

img	Image to insert.

8.4.4.84 CImgList<T>& push_front (const CImg< t > & *img*)

Insert image at the front of the list.

Parameters

img	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

list	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

list	List to insert.

8.4.4.87 CImgList<T>& erase (const iterator *iter*)

Remove image pointed by iterator.

Parameters

iter	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

Display a simple interactive interface to select images or sublists.

Parameters

disp	Window instance to display selection and user interface.
feature_type	Can be `false` to select a single image, or `true` to select a sublist.
axis	Axis along whom images are appended for visualization.
align	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

Display a simple interactive interface to select images or sublists.

Parameters

title	Title of a new window used to display selection and user interface.
feature_type	Can be `false` to select a single image, or `true` to select a sublist.
axis	Axis along whom images are appended for visualization.
align	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

filename	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

filename	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

file	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

filename	Filename to read data from.
n0	Starting index of images to read (∼0U for max).
n1	Ending index of images to read (∼0U for max).
x0	Starting X-coordinates of image regions to read.
y0	Starting Y-coordinates of image regions to read.
z0	Starting Z-coordinates of image regions to read.
c0	Starting C-coordinates of image regions to read.
x1	Ending X-coordinates of image regions to read (∼0U for max).
y1	Ending Y-coordinates of image regions to read (∼0U for max).
z1	Ending Z-coordinates of image regions to read (∼0U for max).
c1	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

filename	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

filename	Filename to read data from.
size_x	Width of the images.
size_y	Height of the images.
first_frame	Index of first image frame to read.
last_frame	Index of last image frame to read.
step_frame	Step applied between each frame.
yuv2rgb	Apply YUV to RGB transformation during reading.

8.4.4.96 CImgList<T>& load_ffmpeg (const char *const *filename,* const unsigned int *first_frame =* `0`, const unsigned int *last_frame =* `~0U`, const unsigned int *step_frame =* `1`, const bool *pixel_format =* `true`, const bool *resume =* `false`)

Load an image from a video file, using ffmpeg libraries.

Parameters

filename	Filename, as a C-string.
first_frame	Index of the first frame to read.
last_frame	Index of the last frame to read.
step_frame	Step value for frame reading.
pixel_format	To be documented.
resume	To be documented.

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

filename	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

filename	Filename to read data from.
use_-graphicsmagick	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

filename	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

	filename	Filename to read data from.
`out`	*primitives*	At return, contains the list of 3d object primitives.
`out`	*colors*	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`)

Load images from a TIFF file.

Parameters

filename	Filename to read data from.
first_frame	Index of first image frame to read.
last_frame	Index of last image frame to read.
step_frame	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 informations about the list on the standard output.

Parameters

title	Label set to the informations displayed.
display_stats	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

disp	Reference to an existing CImgDisplay instance, where the current image list will be displayed.
axis	Appending axis. Can be `{ 'x' | 'y' | 'z' | 'c' }`.
align	Appending alignmenet.

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 temporarly 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

Display the current CImgList instance in a new display window.

Parameters

disp	Display window.
display_info	Tells if image informations are displayed on the standard output.
axis	Alignment axis for images viewing.
align	Apending 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 temporarly 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

Display the current CImgList instance in a new display window.

Parameters

title	Title of the opening display window.
display_info	Tells if list informations must be written on standard output.
axis	Appending axis. Can be `{ 'x' | 'y' | 'z' | 'c' }`.
align	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

filename	Filename to write data to.
number	When positive, represents an index added to the filename. Otherwise, no number is added.
digits	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

filename	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 unsigned int *fps* = 25, const unsigned int *nb_loops* = 0)

Save image sequence as a GIF animated file.

Parameters

filename	Filename to write data to.
fps	Number of desired frames per second.
nb_loops	Number of loops (`0` for infinite looping).

8.4.4.109 const CImgList<T>& save_ffmpeg (const char *const *filename*, const unsigned int *fps* = `25`, const unsigned int *bitrate* = `2048`) const

Save image sequence, using FFMPEG library.

Parameters

filename	Filename to write data to.
fps	Desired framerate (in frames per seconds) if chosen format supports it.
bitrate	Desired bitrate (in bits per seconds) if chosen format supports it.

8.4.4.110 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

filename	Filename to write data to.
is_rgb	Tells if the RGB to YUV conversion must be done for saving.

8.4.4.111 const CImgList<T>& save_yuv (std::FILE *const *file*, const bool *is_rgb* = `true`) const

Save image sequence into a YUV file.

Parameters

file	File to write data to.
is_rgb	Tells if the RGB to YUV conversion must be done for saving.

8.4.4.112 const CImgList<T>& save_cimg (const char *const *filename*, const bool *is_compressed* = `false`) const

Save list into a .cimg file.

Parameters

filename	Filename to write data to.
is_compressed	Tells if data compression must be enabled.

8.4.4.113 const CImgList<T>& save_cimg (std::FILE * *file*, const bool *is_compressed* = `false`) const

Save list into a .cimg file.

Parameters

file	File to write data to.

is_compressed	Tells if data compression must be enabled.

8.4.4.114 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

filename	Filename to write data to.
n0	Starting index of images to write.
x0	Starting X-coordinates of image regions to write.
y0	Starting Y-coordinates of image regions to write.
z0	Starting Z-coordinates of image regions to write.
c0	Starting C-coordinates of image regions to write.

8.4.4.115 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

file	File to write data to.
n0	Starting index of images to write.
x0	Starting X-coordinates of image regions to write.
y0	Starting Y-coordinates of image regions to write.
z0	Starting Z-coordinates of image regions to write.
c0	Starting C-coordinates of image regions to write.

8.4.4.116 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

filename	Filename to write data to.
nb	Number of images to write.
dx	Width of images in the written file.
dy	Height of images in the written file.
dz	Depth of images in the written file.
dc	Spectrum of images in the written file.

8.4.4.117 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

file	File to write data to.
nb	Number of images to write.
dx	Width of images in the written file.
dy	Height of images in the written file.
dz	Depth of images in the written file.
dc	Spectrum of images in the written file.

8.4.4.118 const CImgList<T>& save_tiff (const char ∗const *filename*, const unsigned int *compression_type* = 0) const

Save list as a TIFF file.

Parameters

filename	Filename to write data to.
compression_-type	Compression mode used to write data.

8.4.4.119 const CImgList<T>& save_gzip_external (const char ∗const *filename*) const

Save list as a gzipped file, using external tool 'gzip'.

Parameters

filename	Filename to write data to.

8.4.4.120 const CImgList<T>& save_ffmpeg_external (const char ∗const *filename*, const char ∗const *codec* = 0, const unsigned int *fps* = 25, const unsigned int *bitrate* = 2048) const

Save image sequence, using the external tool 'ffmpeg'.

Parameters

filename	Filename to write data to.
codec	Type of compression.
fps	Number of frames per second.
bitrate	Output bitrate

8.4.4.121 static const CImgList<T>& font (const unsigned int *font_height*, const bool *is_variable_width* = true) [static]

Return a CImg pre-defined font with desired size.

Parameters

font_height	Height of the desired font (exact match for 11,13,17,19,24,32,38,57)
is_variable_-width	Decide if the font has a variable (`true`) or fixed (`false`) width.

8.4.4.122 CImgList<T>& FFT (const char *axis*, const bool *invert* = false)

Compute a 1d Fast Fourier Transform, along specified axis.

Parameters

axis	Axis along which the Fourier transform is computed.
invert	Tells if the direct (`false`) or inverse transform (`true`) is computed.

8.4.4.123 CImgList<T>& FFT (const bool *invert* = `false`)

Compute a n-d Fast Fourier Transform.

Parameters

invert	Tells if the direct (`false`) or inverse transform (`true`) is computed.

Index